Task generality in relation extraction

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Sean Papay

aus Santa Clara, Kalifornien, Vereinigte Staaten

Hauptberichter:	Prof. Dr. Sebastian Padó
1. Mitberichter:	PD Dr. Roman Klinger
2. Mitberichter:	Dr. Vlad Niculae

Tag der mündlichen Prüfung: 2. Oktober, 2023

Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart

2024

Abstract

Relation extraction involves the identification of relations between entities in text. Many distinct tasks in natural language processing, including semantic role labeling, quotation analysis, and event extraction, can be categorized as instances of relation extraction, and share similar structures. However, despite the similarities between these tasks, modeling approaches tend to show little overlap, and model architectures designed for one type of relation extraction task can rarely be applied to others. This situation stands in contrast to other task paradigms common in natural language processing, such as text classification and text generation, wherein existing architectures tend to be highly generalizable to many distinct tasks within their paradigms.

This dissertation investigates *task generality* for relation extraction, that is, the ability or inability of relation extraction model architectures to be successfully applied to diverse relation extraction tasks. To this end, we make a number of concrete contributions: First, we present a formal description language for specifying the properties of different relation extraction tasks, and introduce a

software framework for developing model architectures which can automatically account for these properties. By delineating task-specific frontends from task-general backends, this framework enables task-general architectures to be easily adapted to the specifics of particular tasks. Next, we investigate task generality for span extraction, an important subtask of relation extraction. We identify architecture design choices which facilitate task-generality, and go on to statistically analyze how different types of architectures generalize to different types of tasks, gleaning insights into which task properties, model properties, and interactions therebetween are important for generalization. Finally, we present a method for enforcing regular-language constraints on the outputs of a class of sequence labeling models. We show how constraints can be constructed which capture the specific structures of relation extraction tasks, such that label sequences can be interpreted as relations. Overall, this dissertation works towards making relation extraction more task-general, and we hope our contributions can spur further work in this direction.

Zusammenfassung

Relationsextraktion beinhaltet die Identifizierung von Relationen zwischen Entitäten in Texten. Viele verschiedene Aufgaben der natürlichen Sprachverarbeitung, einschließlich semantischer Rollenzuweisung, Zitatanalyse und Ereignisextraktion, können als Instanzen von Relationsextraktion kategorisiert werden und weisen ähnliche Strukturen auf. Trotz der Ähnlichkeiten zwischen diesen Aufgaben zeigen Modellansätze tendenziell wenig Überschneidungen, und Modellarchitekturen, die für einen bestimmten Typ von Relationsextraktionsaufgabe entwickelt wurden, können selten auf andere angewendet werden. Diese Situation steht im Gegensatz zu anderen Aufgabenparadigmen in der natürlichen Sprachverarbeitung, wie Textklassifikation und Textgenerierung, bei denen bestehende Architekturen in der Regel für viele verschiedene Aufgaben innerhalb ihrer Paradigmen hochgradig verallgemeinerbar sind.

Diese Dissertation untersucht die *Aufgabengeneralität* für Relationsextraktion, d.h. die Fähigkeit oder Unfähigkeit von Modellarchitekturen für Relationsextraktion, erfolgreich auf verschiedene Relationsextraktionsaufgaben

angewendet zu werden. Zu diesem Zweck leisten wir eine Reihe konkreter Beiträge: Zunächst präsentieren wir eine formale Beschreibungssprache zur Spezifizierung der Eigenschaften verschiedener Relationsextraktionaufgaben, und stellen ein Software-Framework vor, mit dem Modellarchitekturen entwickelt werden können, die automatisch diese Eigenschaften berücksichtigen können. Durch die Trennung von aufgabenspezifischen Frontends und aufgabengenerellen Backends ermöglicht dieses Framework, dass aufgabengenerelle Architekturen leicht an die Spezifika bestimmter Aufgaben angepasst werden können. Anschließend untersuchen wir die Aufgabengeneralität für die Extraktion von Textabschnitten, eine wichtige Teilaufgabe der Relationsextraktion. Wir identifizieren Architekturentwurfsentscheidungen, die die Aufgabengeneralität fördern, und analysieren statistisch, wie verschiedene Arten von Architekturen auf verschiedene Arten von Aufgaben verallgemeinerbar sind, um Erkenntnisse darüber zu gewinnen, welche Aufgabeneigenschaften, Modelleigenschaften und Wechselwirkungen zwischen ihnen für die Verallgemeinerung wichtig sind. Schließlich präsentieren wir eine Methode zur Durchsetzung von Einschränkungen von regulären Sprachen

für die Ausgaben einer Klasse von Sequenzbeschriftungsmodellen. Wir zeigen, wie Einschränkungen konstruiert werden können, die die spezifischen Strukturen von Relationsextraktionsaufgaben erfassen, so dass Beschriftungssequenzen als Relationen interpretiert werden können. Insgesamt trägt diese Dissertation dazu bei, Relationsextraktion auf eine aufgabengenerelle Art und Weise zu gestalten, und wir hoffen, dass unsere Beiträge weitere Arbeiten in diese Richtung anregen können.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Relation extraction is a central step in obtaining structured information from unstructured text. At a conceptual level, the task involves the identification of *relations* between *entities* in a text. These relations generally correspond to some sort of semantic relation conveyed by the text, such as the relation between an organization and its founder, or that between a quotation and its speaker, although the specific interpretations of relations can be quite varied. As complex structures can often be built from individual relations between entities, relation extraction forms the first step in pipelines for many tasks with structured outputs. For instance, extracting relations from text is a first and central step for building knowledge graphs, as in Luan et al. (2018), or for analyzing political discourse networks, as in Padó et al. (2019).

Structurally, relation extraction tasks can vary quite significantly. We can examine this variety by exploring a few concrete examples. In the case of coreference resolution tasks, such as the CoNLL-2012 shared task (Pradhan et al., 2012), the relation we are interested in, coreference, might be formalized as a binary symmetric relation between two mentions (in this case, the construction of longer coreference chains would be formalized as a second step in a pipeline). Meanwhile, tasks like TACRED (Zhang et al., 2017) might be interested in semantic relations between entities, such as determining if a particular person lives in a particular city. Such a `lives-in` relation would also be binary, but it would not be symmetric (since a city cannot live in a person). Quotation analysis tasks such as RiQuA (Papay and Padó, 2020) might concern themselves with relations between more than two entities, such as "*who* said *what* to *whom*". When some arguments are optional (e.g. the *whom* might not be mentioned in the text), relations might be variadic, taking a variable number of entities. Quite commonly, a single relation extraction task will have many different types of relations, and these relation types might differ in their structures.

As for the entities, these are usually spans of text, which may either be explicitly marked in the input or unmarked (or, not uncommonly, partially marked). As with relations, it is common for a relation extraction task to distinguish between many types of entities. These entity types often restrict which types of relations an entity can take part in – relations usually require a certain combination of entity types. For instance, the binary `lives-in` relation might expect one entity of type `person` and one of type `city`, while the `coreferent` relation might expect two entities of type `reference`.

1.1 Task generality

Relation extraction models are tasked with identifying entities and relations between them from a text. A large variety of modeling approaches exist, ranging from rule- or knowledge-based systems (e.g. Ravikumar et al., 2017; Mirza and Tonelli, 2016) to models based upon deep neural networks (e.g. Liu et al., 2013; Sui et al., 2023). While not all models employ learning – rule-based systems, for instance, often rely exclusively on hand-crafted rules – this dissertation will focus specifically on machine learn-

ing models for relation extraction, i.e. models which learn to extract relations by generalizing from training data.

1.1.1 Task generality in machine learning

Let us briefly examine the process by which a machine learning model is deployed for a given task. In the most conceptually simple case, first a *model architecture* is selected, then *hyperparameters* are given values, then *model parameters* are learned from training data, and finally the trained model is used to make predictions for unseen data. In this setting, predictions are the result of factors which depend on the training data (model parameters) as well as factors which are specified a priori (model architecture and hyperparameters).

For machine-learning models, we will define task generality to be a model architecture's ability to be trained and applied to many distinct but structurally-similar tasks. For instance, a task-general model architecture applicable for hate speech detection might be expected to work "out-of-the-box" on the task of sentiment analysis – these two tasks, though distinct, share the same basic structure, in that they accept text as input and predict a sentence-level,

categorical output.

This notion of task generality is related to, but distinct from, the concept of model transferability in transfer learning. While model transferability concerns the ability of a model's learned parameters to be adapted for a new task (Ben-Akiva and Bolduc, 1987; Zamir et al., 2018; Bao et al., 2019), task generality is only a property of a model architecture, independent of any trained parameters.

While there is no reason that we should expect our model architectures to be task-general a priori,[1] for many families of tasks, the common neural architectures generalize so consistently that this task generality goes assumed and unstated. This task generality allows the research community to make rapid progress in a large number of structurally-similar tasks – for instance, a novel sequence labeling architecture might push forward the state-of-the-art for named-entity recognition, shallow parsing, part-of-speech tagging, and countless other sequence labeling tasks.

Unfortunately, common models for relation extraction

[1] In fact, there is good reason to expect that they should *not* be – the "no free lunch theorems" in optimization (Wolpert and Macready, 1997) tell us that a learning algorithm's high performance at one task must be offset by low performance on some other task.

do *not* display the extent of task generality seen for many other categories of task (Adel et al., 2018). Model architectures for a given relation extraction task generally cannot be applied to other relation extraction tasks as-is, and often resist attempts at being adapted to new tasks. At a mechanical level, this lack of adaptability is usually the result of differing assumptions made by model architectures, either about the structure of their input, the structure of their output, or the patterns of independencies and independencies between their inputs and outputs.

Seeing the benefits that powerful, task-general architectures bring to other types of NLP tasks, it would be desirable to pursue this type of task generality for relation extraction architectures. As we have identified restrictive task-specific assumptions as the major hindrance to task generality, it seems that we might find what we are looking for by investigating models which make as few assumptions as possible.

1.1.2 Task general relation extraction

Unfortunately, we find that the space of possible relation structures explodes combinatorially as we dispense with assumptions. For instance, if we take entities to be contiguous spans of tokens, we find a document of length n has quadratically many potential entities, as each entity is defined by its starting and ending index. If we then consider candidate relations between these entities, treating relations as simple k-tuples of entities, we must consider $O((n^2)^k) = O(n^{2k})$ potential relations. Finally, for our model to account for mutual exclusivities, interdependencies, and interactions between relation candidates, it cannot predict relations independently, but instead must predict the whole relation structure for a document, i.e. the set of all relations present within the document. This leaves our model with an output space of $O(2^{n^{2k}})$ possible relation structures. The very assumptions that got in the way of task generality are often responsible for limiting this search space to a reasonable size.

1.2 Contributions

If task-specific assumptions limit architecture generalizability, but are nonetheless necessary for tractable, effective relation extraction models, what can we do to improve architectures' task generalizability? This is, in fact, the central research question we will investigate in this dissertation. We will argue that there is a richly populated and under-explored "middle-ground" between the two extremes of practical task-specific architectures and task-general-yet-intractable models. This dissertation identifies a number of directions in which this middle-ground can be explored, and reports on what we have found while exploring in these directions.

Firstly, a promising area for compromise is to not entirely dispense with task-specific assumptions, but rather to formalize them. By formally specifying tasks' structural properties and models' assumptions, we can better understand when architectures might or might not be applicable to new tasks. Additionally, if we can specify tasks and their structures in a machine-interpretable way, we open the door to a class of meta-architectures which can automatically instantiate an appropriate architecture

incorporating the appropriate assumptions based solely on the formal description of the task. Chapter 3 discusses experiments to this end: We define a task specification language, implement a software framework for designing such meta-architectures, and propose a concrete baseline system.

As the extraction of entity spans from text is a prominent subtask of relation extraction, we decide to investigate task generality for entity extraction in isolation. This work is presented in Chapter 4 of this dissertation. We first investigate corpus generality for quotation extraction, a specific span extraction task. We observe that corpus generality shares many of the same challenges as task generality, and develop a model architecture capable of generalizing across corpora of varying formalisms, modalities, and languages, identifying a number of strategies for developing successful corpus-general models along the way. We follow this up by investigating full task generality for span extraction. Here, we collect a large "zoo" of span extraction tasks and model architectures, and use performance prediction to carry out a large-scale quantitative analysis. This not only yields a performance prediction model, which can predict how well a given model ar-

chitecture will perform on a given task, but also gleans insight into the properties of models and tasks which interact to affect generalizability.

For full relation extraction, if we want to work with formally specified task structures, it becomes desirable to find model architectures capable of handling a large family of such structures. In Chapter 5, we discuss such an architecture, based upon conditional random field models and capable (with some caveats) of modeling those task structures representable as regular languages.

This dissertation encompases work from the following publications: The work presented in Chapter 3 is based on work presented in (Adel et al., 2018), a demo paper presented at EMNLP 2018. I was one of five authors for this work – I participated in formalizing the structure of task- and model-specifications, was responsible for implementing the decoding procedure for the baseline model, and contributed to writing the manuscript for publication. Chapter 4 describes work from two separate publications: Papay and Padó (2019), presented at RANLP 2019, and Papay et al. (2020), presented at EMNLP 2020. I was the lead author for both of these works, performing all implementation and experiments, and doing a majority of the

writing. Chapter 5 is based on work performed for Papay et al. (2022), which was presented at ICLR 2021. I was likewise the lead author for this work, and performed all implementation, experiments, and a majority of the writing.

Chapter 2

Background

In order to investigate task generality as it applies to relation extraction, we would like to establish a solid groundwork of background knowledge, and a consistent scheme of notation and terminology for us to use later. This chapter will seek to do exactly this. We start by discussing machine learning in general in Section 2.1, introducing useful notation and conventions, and briefly summarizing common machine learning techniques which we use persistently in this dissertation. The remaining sections of this chapter will discuss relation extraction. Section 2.2 establishes definitions for relation extraction tasks which we will utilize throughout this dissertation Section 2.3 will discuss common patterns in machine learning models for relation extraction, and finally, Section 2.4 will discus a small selection of specific relation

extraction tasks, and existing model architectures which are commonly applied to those tasks.

2.1 Machine learning and neural networks

This section will provide a brief overview of machine learning and neural networks, focusing on the formalisms and techniques which will be relevant to relation extraction.

2.1.1 Distributions and datasets

We will formalize machine learning in terms of probability distributions. In all cases, we start with a joint distribution $P(v^1, v^2, \cdots)$ over a set of random variables $\{v^1, v^2, \cdots\}$. We will call this distribution our *data distribution*. For notational brevity, we will take $\boldsymbol{v}$ to be the sequence $\langle v^1, v^2, \cdots \rangle$, and may write the data distribution as $P(\boldsymbol{v})$. We normally have access to one or more finite sets of samples from the data distribution – we call such

a set of samples a *dataset*.[1] A common setting is to have three such datasets: a training set, a development set, and a validation set, although such a split is not universal. In any case, for a dataset D, we can define $\tilde{P}_D(\boldsymbol{\nu})$ to be the probability distribution obtained by uniformly randomly selecting one datapoint from D. We might refer to this as the *dataset distribution* of D.

For a specific machine learning task, we will partition the set of random variables as a disjoint union of *input variables* $\{\mathrm{x}^1, \mathrm{x}^2, \cdots\}$ and *output variables* $\{\mathrm{y}^1, \mathrm{y}^2, \cdots\}$. Our goal is to computationally model the conditional probability distribution $P(\mathbf{y} \mid \mathbf{x}) = P(\mathrm{y}^1, \mathrm{y}^2, \cdots \mid \mathrm{x}^1, \mathrm{x}^2, \cdots)$, which we will call the *target distribution*.

2.1.2 Models and model architectures

We will use the word *model* to refer to a family of probability distributions comparable to the target distribution, i.e. defined over and conditioned on the same output and

[1] Technically, as multiplicity is generally allowed in datasets, it would be more accurate to call these datamultisets, or perhaps databags. However, for most tasks and dataset sizes, it is uncommon to have any sample with multiplicity > 1, making the distinction largely irrelevant, and so we will stick to the established, if misleading, terminology.

input variables. Such models are parameterized by a parameter vector $\boldsymbol{\theta}$ in some vector space Θ, such that each value of $\boldsymbol{\theta} \in \Theta$ corresponds to an individual distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$. We will term such distributions *model distributions*. It is important to note that the target distribution itself need not be one of these model distributions. In this dissertation, we will always assume that $\Theta = \mathbb{R}^k$ for some (finite) k.

While any parameterized family of distributions can be called a model, this dissertation will focus on those families of distributions which can be efficiently represented and approximated algorithmically. Different types of models might have different algorithmic representations, and some algorithms might only represent their models implicitly. In general, though, for any particular model, we will usually have a particular algorithmic representation in mind, to the point where we may at times informally identify models with their associated algorithms.

We will often need to work with a family of related models. We will refer to such a family as a *model architecture*, or often just an *architecture* for short. Just as models are parameterized by a parameterization $\boldsymbol{\theta}$, we parame-

terize architectures by a hyperparameterization $\boldsymbol{\lambda} \in \Lambda$ – we notate the model yielded by hyperparameterization $\boldsymbol{\lambda}$ as $\hat{P}^{\boldsymbol{\lambda}}$, and we notate the individual model distribution of that model for parameterization $\boldsymbol{\theta}$ as $\hat{P}^{\boldsymbol{\lambda}}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$. In the case that the values of either $\boldsymbol{\theta}$ or $\boldsymbol{\lambda}$ are clear from context, or if their specific values are irrelevant, we may omit subscripts or superscripts respectively.

While we stipulated that $\boldsymbol{\theta}$ was a real-valued vector, we will make no such assumption for $\boldsymbol{\lambda}$, allowing Λ to be any well-defined set. In practice, a hyperparameterization is usually envisioned as a collection of individual hyperparameters, with $\boldsymbol{\lambda}$ being a tuple of values for these hyperparameters and Λ being the Cartesian product of those hyperparameters' configuration spaces.

Our definitions leave us some freedom with what we call a parameter and what we call a hyperparameter. We can always pick out and remove some dimension of our parameter space Θ, and insert a real-valued hyperparameter in its place. In the case where our hyperparameterization is a tuple containing some real number, we can do exactly the opposite, promoting that hyperparameter to a parameter. While we will always keep a strict dichotomy between model parameters and hyperparam-

eters, we may use this freedom to convert parameters to hyperparameters and and vice versa.

2.1.3 Parameters and optimization

Optimization is the task of selecting the best value $\boldsymbol{\theta}^*$ for a model's parameter vector, according to some well-defined sense of quality. As this is usually quite difficult, it is normal to dispense with perfectionism, and use the word optimization to refer to the task of merely finding a good parameter value $\boldsymbol{\theta}^*$. Usually, this is done with the goal of making $\hat{P}_{\boldsymbol{\theta}^*}(\mathbf{y} \mid \mathbf{x})$ approximate the target distribution as closely as possible. While this problem is widely studied, with many different approaches, this dissertation consistently relies on gradient-based optimization. This section will therefore focus on parameter optimization using gradient-based approaches.

Of course, if optimization involves looking for the best parameterization $\boldsymbol{\theta}^*$, it is natural to also consider the task of selecting the best hyperparameterization $\boldsymbol{\lambda}^*$, i.e. the hyperparameterization for which the best $\boldsymbol{\theta}^*$ exists. This task is termed hyperparameter optimization. As hyperparameter optimization notoriously relies on architecture-

specific intuitions and a good measure of "black magic" (Anand et al., 2022), we will only discuss hyperparameter optimization in the context of specific architectures.

Gradient-based parameter optimization

Here we will briefly describe gradient descent and related parameter optimization algorithms. We formalize optimization in terms of a *loss function* $L(\boldsymbol{\theta})$, a differentiable real-valued function of $\boldsymbol{\theta}$ that quantifies how "poorly" the model distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$ approximates the target distribution $P(\mathbf{y} \mid \mathbf{x})$. A common choice of loss function is to approximate the cross entropy between the model distribution and the target distribution by summing over samples from a dataset distribution, but all that we will require is that the loss function has a global minimum, and that low values of loss correspond in some ill-defined sense to "good models" of the target distribution.

The gradient descent algorithm (Cauchy et al., 1847; Ruder, 2016) starts with an initial parameter value $\boldsymbol{\theta}^0$, and iteratively updates the parameter value based on the gradient of the loss function. $\boldsymbol{\theta}^0$ is selected randomly from some distribution over the parameter space, usually a zero-mean normal or uniform distribution. We then

define our sequence of updated parameters with the rule

$$\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i - \alpha \cdot \nabla L(\boldsymbol{\theta}^i)$$

where $\alpha \in \mathbb{R}^+$, the *learning rate*, is an optimization metaparameter – that is, a free variable whose value affects the optimization procedure.[2] As long as L is locally linear at the length scale of the gradient update, each subsequent parameter update should yield a lower loss value, and the process should converge to some local minimum of L:

$$\boldsymbol{\theta}^* = \lim_{i \to \infty} \boldsymbol{\theta}^i$$

In practice, such limits aren't computable, and in fact often don't exist depending on the loss function and our choice of learning rate. Instead, we often take

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}^k$$

for some $k \gg 0$.

[2]While learning rate and similar optimization metaparameters are often labeled as hyperparameters in existing literature, such a characterization would be inappropriate by our definitions, as optimization metaparameters only affect the behavior of the optimizer, and have no bearing on the model.

There exist a number of variations on vanilla gradient descent. One common approach, stochastic gradient descent (SGD), uses small subsets (batches) of the full dataset to approximate the loss function during each update step. This drastically improves the efficiency of computing the loss function, at the cost of accuracy – small batches only provide an approximation of the true loss function. Nonetheless, this turns out to be a blessing in disguise, as these noisy approximations of the true loss function can help optimizers escape suboptimal local minima and saddle points during optimization (Keskar et al., 2016; Kleinberg et al., 2018). For modern gradient-based models, SGD is ubiquitous.

Other approaches modulate gradient descent's update rule. In gradient descent with momentum (Polyak, 1964; Sutskever et al., 2013), the optimizer keeps track of a velocity vector. During each update step, the gradient updates this velocity vector, and that velocity vector in turn updates the parameter value – this approach is argued to converge more quickly and to better local minima.

Finally, some techniques replace the static learning rate with more sophisticated machinery to account for different length-scales in different regions of the parameter

space, and along different directions. One well-known algorithm, Adam (Kingma and Ba, 2015), keeps track of dynamic estimates for the variance of each parameter's gradient, and uses these estimates to set parameter-wise learning rates. Similar approaches in this category include AdaGrad (Duchi et al., 2011) and RMSProp (Tieleman and Hinton, 2012).

2.1.4 Artificial neural networks

Artificial neural networks (ANNs) have become a central component of many prominent machine learning model architectures. While it is hard to find agreement on their exact definition, ANNs are a method for representing and computing diverse families of vector-valued functions of a vector argument, with each such family parameterized by a vector $\boldsymbol{\theta}$. In this section, we shall use $f_{\boldsymbol{\theta}}$ to notate the function represented by some neural network for parameter $\boldsymbol{\theta}$, with $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$. Importantly for gradient-based optimization, $\boldsymbol{y}$ is generally differentiable with respect to $\boldsymbol{\theta}$.

While most models we will investigate have ANNs at their core, it is not always straightforward to represent a

probability distribution using a neural network. Usually, values of the random variable $\mathbf{x}$ are encoded as vectors and identified with the vector $\boldsymbol{x}$. In the case when the target distribution has small sample space, the network's output vector $\boldsymbol{y}$ can be interpreted directly as a probability distribution over the possible values of the random variable $\mathbf{y}$, so long as care is taken to ensure the components are non-negative and sum to unity. In other cases, more sophisticated machinery will be needed – we will discuss such details as they arise.

Artificial neural networks generally represent $f_{\boldsymbol{\theta}}$ as a composition of many *layers* $l^i_{\boldsymbol{\theta}}$:

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = \left(l^k_{\boldsymbol{\theta}} \circ l^{k-1}_{\boldsymbol{\theta}} \circ \cdots \circ l^1_{\boldsymbol{\theta}}\right)(\boldsymbol{x})$$

These layers generally represent conceptually simple and easy-to-compute functions – we will abuse notation and also use $\boldsymbol{x}$ and $\boldsymbol{y}$ to refer to the inputs and outputs for a single layer, respectively. Two of the most common layer types are *linear layers*, which represent bilinear functions of $\boldsymbol{x}$ and $\boldsymbol{\theta}$ [3], and *activation functions*, which are usually

[3] In practice, this usually means building a matrix out of some components of $\boldsymbol{\theta}$, and multiplying the input vector $\boldsymbol{x}$ by that matrix.

simple non-linear functions applied componentwise to the input vector. The composition of a linear layer with an activation function is termed a *feedforward layer*. The simplest category of ANNs, multi-layer perceptrons (MLPs), are pure compositions of multiple feedforward layers, but other, more complicated kinds of networks might be appropriate for certain task settings.

A case of particular interest for natural language processing is where $\boldsymbol{x}$ and $\boldsymbol{y}$ represent sequences of values. Of course, when the sequence length is a constant k, we can interpret these both to be concatenations of subvectors, maintaining our vector-to-vector formalism:

$$\boldsymbol{x} = \boldsymbol{x}^1 \oplus \boldsymbol{x}^2 \oplus \cdots \oplus \boldsymbol{x}^k, \qquad \boldsymbol{y} = \boldsymbol{y}^1 \oplus \boldsymbol{y}^2 \oplus \cdots \oplus \boldsymbol{y}^k$$

However, special care must be taken in designing neural networks which can learn to be sensitive to the sequence structure of their inputs and outputs. Such networks will often have the additional benefit of being well-defined for variable sequence lengths k. In the following sections, we will discuss common types of network for this setting.

Recurrent neural networks

One approach to processing sequences with neural networks involves repeated application of the same layer $l^r_{\boldsymbol{\theta}}$. For instance, let's start by defining $\boldsymbol{y}^0 = 0$. We can then define

$$\boldsymbol{y}^i = l^r_{\boldsymbol{\theta}}(\boldsymbol{y}^{i-1} \oplus \boldsymbol{x}^i)$$

for $i \in [\![1 \cdot\cdot k]\!]$. In this way, the value of each $\boldsymbol{y}^i$ depends on the values of all input vectors $\boldsymbol{x}^j$ where $j \leq i$, and the final output vector $\boldsymbol{y}^k$ depends on the entirety of $\boldsymbol{x}$. We term such layers *recurrent layers*, and networks which utilize these are termed *recurrent neural networks* (RNNs).

It is important to note that RNNs are inherently directional – each output vector can depend only on previous input vectors, and not on subsequent ones. While this property may be desired in some tasks (for instance, in autoregressive language modeling), other circumstances may call call for $\boldsymbol{y}^i$ vectors which depend on both preceding and succeeding $\boldsymbol{x}^j$ vectors. In these cases, *bidirectional RNNs* can be applied. In a bidirectional recurrent layer, two separate recurrent layers are employed: one applied to the sequence $\langle \boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^k \rangle$, and another applied to its reversal $\langle \boldsymbol{x}^k, \boldsymbol{x}^{k-1}, \cdots, \boldsymbol{x}^1 \rangle$. The outputs of these two recur-

rent layers can be combined (e.g. by vector concatenation) for use as input to further layers in the neural network.

As an RNN's prediction for $\boldsymbol{y}^i$ can only depend on distant $\boldsymbol{x}^j$s "by proxy" of all of the intervening output vectors $\boldsymbol{y}^{[\![j+1..i-1]\!]}$, modeling long range interactions often degrades into a game of Chinese whispers over a noisy channel – when viewed in terms of the update signal provided by gradient descent, this is often termed the *vanishing gradient problem*. Many proposals have been made to partially alleviate this problem in RNNs, leading to a number of popular RNN layer types. The most well-known among these is the LSTM (Hochreiter and Schmidhuber, 1997), or long short-term memory network, which utilizes a notion of gates in order to allow information to flow more directly between time steps. The GRU (Cho et al., 2014), or gated recurrent unit, is a similar variation on the same theme.

Self-attention and transformers

Transformers (Vaswani et al., 2017) are an alternate approach to sequence processing, based not on RNNs but on *self-attention layers*. While we will relegate a complete technical description of transformers to Vaswani et al.'s

paper, in this section we will discuss them at an informal level and contrast them with RNNs-based networks.

Macroscopically, transformer networks consist of alternating self-attention layers and token-wise feedforward layers. As the self-attention layers are responsible for all sequence processing, we will focus our discussion on these.

In a self-attention layer, each output vector $\boldsymbol{y}^i$ depends directly on all input vectors $\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^k$. This differs significantly from RNNs, wherein each output vector can only depend on distant input vectors indirectly. This is achieved by means of an *attention mechanism* – for each pair $(\boldsymbol{x}^i, \boldsymbol{x}^j)$ an attention score is calculated, and $\boldsymbol{y}^i$ is represented as a weighted sum of contributions from each $\boldsymbol{x}^j$, with the weights of this sum deriving from the attention scores.

This approach to sequence processing has a number of consequences that make transformers vary significantly from RNNs. Firstly, transformers are intrinsically bidirectional – attention scores *can* be defined in such a way to ensure that each $\boldsymbol{y}^i$ can only attend to $\boldsymbol{x}^j$ where $j \leq i$, but this requires special effort. In fact, self-attention layers not only aren't directional, but aren't sensitive to the sequence

structure of $\boldsymbol{x}$ at all – as weighted sums are commutative, self-attention layers are permutation invariant. This property is actually detrimental in most NLP tasks, where we would like models to be sensitive to the order of tokens – for this reason, transformer networks utilize *positional embeddings* to explicitly encode within each $\boldsymbol{x}^i$ information about the value of i. While this allows transformers to be sensitive to the order of its inputs, it is important to note that transformers must "learn" to utilize this sequence information – while RNNs are intrinsically more sensitive to short-term interactions than to long-term ones, transformers have no such inductive bias.

2.1.5 Pretrained embedding networks

A recent trend in NLP is increasing reliance on large, pretrained neural networks to acquire *word embeddings*, vector representations of words to be used as inputs to further ANN layers. This section will discuss these pretrained models in roughly chronological order, outlining their history and examining their applications in state-of-the-art models.

Distributional semantics and and word embeddings

The concept of word embeddings grew out of the field of distributional semantics. This field is interested in formalizing and representing the meanings of words in terms of their statistical properties in a corpus of text. With this goal, vectors turn out to be a natural way of summarizing these statistical properties – the earliest approaches to generating word embeddings generally consist of a fixed pipeline of transformations applied to corpus frequencies in order to obtain vectors.

Distributional semantics came to be linked to neural networks in two ways. First, it was realized that neural networks could be used to obtain embedding vectors – Bengio et al. (2000) describe the training of a neural language model, and observe that some of the model's learned parameters can be interpreted as semantic representations. Secondly, during their work on multitask learning, Collobert and Weston (2008) note that embeddings learned from one task can be re-used to initialize a new neural network to improve performance.

This second discovery in particular led to the proliferation of general-purpose word embeddings – embeddings could be generated from a large corpus of unlabeled text,

and then published to be incorporated into neural models for any number of NLP tasks. Word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) are two widely used frameworks for the generation of such pretrained embeddings.

Contextualized embeddings

One limitation of traditional word embeddings is in dealing with polysemy: as a single vector is assigned to each unique word, words with multiple senses receive only a single vector which must encompass all of these senses. Contextualized embeddings emerged as a technique for dealing with this problem. In the framework of contextualized embeddings, instead of assigning an embedding vector to each word, a sequence of embedding vectors is assigned to each sequence of words. In this way, the value of a specific embedding can depend on the context of its corresponding word within a larger text. While traditional embeddings could be disseminated as a serialized list of vectors for some finite vocabulary of words, contextualized embeddings must be calculated from scratch for each sequence of words. Therefore, contextualized embedding systems are usually distributed in the form

of a pretrained neural network which can be evaluated for a user-provided input sequence.

While some early experiments with contextualized word embeddings were carried out in Lee et al. (2017) and McCann et al. (2017), the technique exploded in popularity with the release of ELMo (Peters et al., 2018). ELMo is an LSTM-based neural network that was trained for language modeling, with the outputs of the network's internal layers then used as word embeddings. As language modeling is an inherently directional task, ELMo largely consists of two independent networks, one which does left-to-right language modeling and one which does right-to-left language modeling. Each network produces embedding vectors, and these are concatenated to form ELMo's contextualized word representations.

From embeddings to transfer learning

When a neural network is used to generate contextualized embeddings, and these embeddings are then used as inputs for another neural network, it is possible to consider the composition of these two ANNs as one large network. From this perspective, it becomes natural to consider optimizing the parameters of this composed net-

work jointly, updating the parameters in the embedding network and learning parameters for the downstream network concurrently. This places us squarely in the field of transfer learning, wherein our embedding network was first *pretrained* for one task, and subsequently *fine-tuned* for a different, but related task. This transfer learning approach has proven quite popular with contextualized embedding networks.

While transfer learning is possible with ELMo, it is not commonly employed. In their original publication, Peters et al. recommend fine-tuning ELMo on its language modeling pretraining task for new datasets, but do not explore updating weights during training of the main task. Peters et al. (2019) note inconsistent effects of fine-tuning ELMo, with minor improvements on some tasks but minor degradations on others. On the other hand, shortly after the release of ELMo, a new family of transformer-based embedding models appeared which seemed much more amenable to fine-tuning.

BERT (Devlin et al., 2019a) was the first of these to make significant waves. Apart from using a transformer instead of LSTMs, BERT differs from ELMo in its pretraining task – while ELMo is pretrained as a pair of unidirec-

tional language models, BERT is pretrained primarily on the cloze task (Taylor, 1953), or *masked language modeling*. In this setting, some percentage of words are "masked" out of a text, and BERT must predict which words were masked. This allows for a fully non-directional embedding model, as BERT can depend the left context, the right context, and interactions between them when computing the embedding for a particular word.

BERT ushered in a plethora of similar embedding models based on masked language modeling and transformers. Some of the most salient of these include RoBERTa (Liu et al., 2019), which improves upon the training procedure of BERT, DistilBERT (Sanh et al., 2019), a distilled version of BERT which aims to preserve its performance with lower computational requirements, and SpanBERT (Joshi et al., 2020), with a focus on representing spans of text.

2.2 Relation extraction

In order to discuss relation extraction in detail, it is desirable to establish an exact vocabulary of definitions and formalisms. This is especially crucial given the conflicting

assumptions commonly made when working on different tasks or datasets. We will therefore establish a vocabulary as general as possible, with the hopes of being able to accommodate as much of the existing literature as possible.

2.2.1 Documents

We assume that each dataset comprises a set of documents. In general, we will use the word *document* to refer to the smallest self-contained unit of a dataset. Each document has its own input text, and all relations "live" in exactly one document – we disallow relations between different documents. It should be possible to work with individual documents separately – any relation extraction model should be able to make predictions for a single document of a larger dataset, and separate predictions made for separate documents should always be made independently from one another.

Concretely, the level of textual subdivision represented by a document can vary considerably depending on the task. Some tasks might treat individual sentences as documents, while others might treat an entire novel as a single document. When defining a document for a specific task,

we always choose the smallest level of subdivision which has the independence properties we are interested in.

2.2.2 Text

All relation extraction tasks which we consider will take text as their input. For our purposes, we consider a text to be a finite-length sequence of *tokens*. Datasets are free to define their own notion of tokens – some might treat words or subwords as tokens, and come with a predefined vocabulary, while datasets which use raw text as input can be interpreted as treating each character as a token. *Retokenization* might be necessary when a dataset and a model disagree about the meaning of a token – for instance, it might be the case that a task defines tokens to be individual characters, while a model uses subwords as tokens. In these cases, such retokenization should be regarded as an internal detail of the model, and the model's final output should be interpreted in terms of the data set's original tokenization.

Many datasets also make extra-textual input available to models: for instance, datasets based on the Penn Treebank (Marcinkiewicz, 1994) might also provide part-of-

speech tags or constituency parsings of the input text. When this is the case, we will discuss specifically how that input is treated.

2.2.3 Entities

We define an *entity* to be anything which can participate in a relation. Specific relation extraction tasks are free to define these entities however they would like, but we will discuss a few common choices here.

We will stipulate that each entity has an *entity type*, which comes from some task-defined finite set of possible entity types. While individual tasks can define the exact semantics of these types, entities of the same type will generally be capable of engaging in the same types of relations. Of course, for tasks with no useful notion of different types of entities, we can simply assign all entities the same singleton type.

Textual references and spans

By far the most common kind of entity (and in fact the only choice that this dissertation will explore in much detail), textual references are fragments of the input text

which act as entities. Usually, these fragments will be textual realizations of some semantic objects – for example, they might be noun phrases which refer to specific physical objects. In the case that these fragments are contiguous spans of text, we will refer to these entities as *spans*.

For some tasks, some or all textual references may be specified a priori. For instance, a task for extracting relations between named entities might come with named entity mentions pre-identified. While these cases can have major consequences for task difficulty and modeling approaches, for the sake of consistent nomenclature, we will still consider these to be textual references that models must "identify," although this identification task might be trivial (i.e. models must only copy these entities from their input to their output).

Extratextual entities

Some tasks involve relations between entities which are not textual references. While our novel contributions in this dissertation do not address these cases thoroughly, we will list here a few possibilities for extratextual entities.

A common circumstance where extratextual entities

arise are in cases where we know some ontology of objects a-priori. In these cases, we might be interested in finding relations involving these entities themselves, rather than textual references to them. This sort of structure arises commonly across task domains: For instance, in literature analysis, we might be interested in relations between characters known a priori, as in Kim and Klinger (2019) or Wiedmer et al. (2020).

In practice, such cases are often, but not always, modeled as a pipeline, where the first step involves finding relations between textual references, while the second step involves aligning these textual references with extra-textual entities.

Relations as entities

Some tasks allow for relations themselves to be entities in higher-order relations. This sort of structure adds considerable complexity to relation extraction tasks, allowing for many difficult-to-model possibilities such as reference cycles and self-referential relations, and admitting an infinitude of possible relation structures for finite texts. When relations are allowed as entities for a task, the task will usually make other strict assumptions about which

structures are legal in order to limit the task complexity.

2.2.4 Relations

We define a *relation* to be a collection of entities, along with additional structure to define how these entities are related to one another. We represent this structure in terms of *roles*, wherein each entity has a particular role in each relation it participates in.

As with entities, each relation is assigned a *relation type* from some finite task-defined set of possible relation types.

Roles

We organize the entities which participate in a relation in terms of roles. Each entity participating in a relation has a role within that relation, where these roles come from a finite set of possible roles, defined for each task a priori. Of course, tasks with no natural notion of roles can be formalized with one singleton role. It is possible for multiple entities to participate in the same relation with the same role, and it is likewise possible for the same entity to participate in a relation multiple times

under different roles. It is therefore natural to treat the participating entities for a relation as a set of (entity, role) pairs.

2.2.5 Relation structures, constraints, and evaluation

Periodically, we will refer to the *relation structure* of a document – by this, we mean the set of all entities and relations in a document taken together. Such a notion is useful to formalize the task of relation extraction in the notation we have established for machine learning – a relation extraction task has a target distribution of the form $P(\mathbf{y} \mid \mathbf{x})$, where $\mathbf{x}$ varies over all possible documents and $\mathbf{y}$ varies over all possible relation structures.

Constraints

Each task is free to define constraints on which sorts of relation structures are allowed and which are disallowed. These can correspond to local properties of particular relations (e.g. "Each relation must have exactly two entities") as well as global properties of relation structures (e.g. "No entity can occur in more than two relations"). For

the sake of generality, we will formalize such constraints as a task-defined legality predicate on relation structures. However, we should be careful to define these predicates in such a way so as to be sensitive to structure rather than content – for instance, if we are modeling `parent_of` relations, our predicate might ensure that each such relation has exactly two entities of type `person`, but it should *not* be responsible for enforcing that the first entity is the parent of the second – ensuring this would be the job of a model for this task.

2.2.6 Evaluation

Evaluation is the process of quantitatively comparing a model's predictions to the true labels, yielding a numeric *evaluation score*. Higher scores usually correspond to higher quality predictions, but this is not always the case (e.g. when scores represent error rates). Evaluation takes place primarily at a per-document level, with per-document evaluation scores aggregated across a dataset to yield a final evaluation score for a model. While this can in principle depend directly on the model distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$, such a setting is rare in relation extraction.

Rather, for each $(\boldsymbol{x}, \boldsymbol{y})$ pair in the dataset, evaluation usually proceeds by comparing the relation structure $\boldsymbol{y}$ to a model's predicted structure $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}'} \hat{P}(\boldsymbol{y}' \mid \boldsymbol{x})$, and assigning a numerical score based on how well these structures align.

The intended evaluation metric is usually specified along with a particular task, and some metrics may only be well-defined for certain tasks. We will discuss a few common cases here, but leave specifics to our discussions of particular tasks.

The simplest-to-define metric is document-wise accuracy, where we assign a score of one if $\hat{\boldsymbol{y}} = \boldsymbol{y}$, assign a score of zero otherwise, and aggregate scores across the dataset by simple averaging. This approach is not too common in practice, as it can not assign "partial credit" to distinguish between mostly-correct predictions and catastrophic failures.

Relation-level metrics address this by counting each relation separately, allowing for more fine-grained distinctions than document- level evaluation. One approach here is relation-wise F_1-score. Even this is often too coarse-grained for many tasks, and metrics are often defined which can account for partial matches between re-

lations and entities. This becomes especially important when relations are non-binary. For n-ary relations, the space of relation candidates grows quite large, and it might be desirable to assign partial credit for predictions which match in some, but not all, of the true entities. Approaches to accounting for partial relation matches often reduce to representing n-ary relations as a collection of binary relations, and evaluating all such binary relations together. For example, the CoNLL 2005 shared task for semantic role labeling (Carreras and Màrquez, 2005a), which involves variadic relations between verbs and a large number of other roles, evaluates by decomposing these large relations into collections of verb-role binary relations, and calculating F_1-scores for these binary relations.

2.3 Architectural motifs for relation extraction

While architectures for relation extraction are hugely varied, and are often tuned to the specific task they target, a number of motifs tend to recur across many architectures.

This section will briefly define and discuss a few of these patterns. We will try to keep all discussions here abstract and high-level, relegating more concrete discussions of applications of these motifs to discussions of particular architectures.

2.3.1 Pipeline models

Pipelines are a common motif wherein the full task is decomposed into a sequence of subtasks, and these subtasks are then carried out by a sequence of submodels in order to obtain predictions, with the output of each submodel being used as the input for subsequent submodels. It should be noted that pipeline models are not specific to relation extraction, and are in fact quite ubiquitous across many areas in machine learning.

Formally, we can view pipelines as a way of factorizing our target distribution: if we have a target distribution over k output variables $\mathbf{y} = \langle \mathrm{y}^1, \cdots, \mathrm{y}^k \rangle$, a pipeline model uses the chain rule to represent this distribution in terms of k submodels, where each submodel is responsible for only a single output variable:

$$\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathrm{x}) = \hat{P}_{\boldsymbol{\theta}}(\mathrm{y}^1 \mid \mathrm{x}) \cdot \hat{P}_{\boldsymbol{\theta}}(\mathrm{y}^2 \mid \mathrm{x}, \mathrm{y}^1) \cdots \hat{P}_{\boldsymbol{\theta}}(\mathrm{y}^k \mid \mathrm{x}, \mathrm{y}^1, \cdots, \mathrm{y}^{k-1})$$

In principle, search algorithms such as beam search can be used to find high-likelihood assignments to all of the output variables during prediction, but often times simple greedy decoding is used, selecting the most likely prediction for each submodel.

In relation extraction, pipeline models often factorize the full task into two subtasks: *entity extraction*, which is the task of identifying and type-labeling all entities present given a text, and *relation building*, which involves predicting a set of relations given the text and the set of all entities present. Not uncommonly, these subtasks might themselves be modeled as pipelines: for example, entity extraction might be done by first extracting unlabeled entities, and subsequently assigning entity types to them.

2.3.2 Sequence labeling for span extraction

When only text spans are considered as entities during entity extraction, we may term this step as *span extraction*. As many relation extraction tasks only consider relations between spans, span extraction is a ubiquitous first step in pipeline models. While many approaches exist, a common choice is to treat span extraction as a se-

quence labeling task, encoding the spans of a document in terms of token-wise labels and tasking a model with predicting these labels.

BIO labeling (Ramshaw and Marcus, 1999) is one common method of encoding a set of non-overlapping spans in a document as a label sequence. Under this framework, the first token of each span is assigned a begin-label (`B`), all subsequent tokens in a span are assigned an inside-label (`I`), and all tokens which are not part of any span are assigned an outside-label (`O`). In this way, any set of non-overlapping spans in a document can be uniquely represented by a label sequence, and sequence labeling models can be used for span extraction. In order to account for span types, `B`- and `I`-labels can be marked by span type.

Other sequence-labeling approaches to span extraction tend to be minor variations on BIO labeling. The most common variant, BILOU labeling (Ratinov and Roth, 2009), extends BIO labeling by marking the last token of each span with an `L`-label. This necessitates a new label, `U`, to mark single-token spans. In general, sequence-labeling based approaches are all limited in their ability to represent nested or overlapping spans, and generally assume

that no spans can share tokens.

2.4 Specific relation extraction tasks

We will now investigate a number of existing relation extraction tasks from the literature in terms of the definitions introduced in Section 2.2, and briefly discuss modeling approaches used for these tasks. This will serve three major purposes: First, this will provide concrete examples of how existing tasks can be formalized in terms of the framework we have established. Secondly, by purposely choosing a diverse set of tasks, this will act as a review of the breadth of relation extraction tasks in existing literature. Finally, we will get a first concrete glimpse at how existing models depend on task-specific features.

2.4.1 TACRED

TACRED (Zhang et al., 2017) is a corpus for relation extraction over newswire text. TACRED defines 41 types of binary relations between 23 types of entities; examples include a `city_of_death` relation between a `PERSON` entity and a `CITY` entity and a `founded_by` relation between

a ORGANIZATION entity and a PERSON entity.

TACRED makes a number of simplifying assumptions in its task statement. Each document is a single sentence, and is guaranteed to contain at most one relation between two entity spans. These two spans, labeled by entity types, are provided as model input. One of these is labeled as the subject of the relation, and the other is labeled as the object. As we will soon discuss, these assumptions have significant implications for what kinds of model architectures are successful for this task.

It is quite straightforward to describe TACRED in terms of the definitions we have established. TACRED's notion of entity and relation types coincides exactly with our definitions, and TACRED's subjects and objects are naturally interpreted in terms of our notion of roles. The corpus comes tokenized at the word level. Evaluation is done by micro-averaged relation F_1-score.

TACRED's constraints, formalized as a legality predicate, would only admit relation structures meeting the following criteria:

- The only entities present are the two spans present in the input.

- There is at most one relation present.

- If there is a relation, it has exactly two entities – those present in the input.

- If there is a relation, each entity's role within that relation is identical to that specified in the input.

When we consider modeling approaches within these constraints, it becomes clear that there isn't too much left for a model to do that isn't already provided in the input. Since we already know the two entities and their roles, all that is left for a model is to a) decide if there is a relation present, and b) if so, decide that relation's relation type. In fact, almost all models for this task are ultimately just classification models (e.g. Zhang et al., 2017; Huang et al., 2022; Baek and Choi, 2022), where the set of classes is the set of relation types along with a separate "no-relation" label. Most modeling complexity revolves not around the output structure, but rather in representing the input text and graceful handling of minority labels.

2.4.2 OntoNotes semantic role labeling

Semantic role labeling (SRL) can be informally described as answering the question "Who does what to whom?" in a sentence. This task is not usually characterized as an instance of relation extraction, but it is easily accommodated by our formalisms. As the task is defined differently for different datasets, we will focus on the task as it is described for the CoNLL 2005 shared task (Carreras and Màrquez, 2005b) for use with the OntoNotes corpus (Weischedel et al., 2011).

In the CoNLL 2005 shared task, all predicates are marked in the input, making SRL more a question of "Who does p to whom?" for some known predicate p. Given each p and its context in a sentence, models must identify all arguments to that predicate, each fulfilling a "role". In general, each of these is a span, but a built-in notion of continuation spans does allow for discontiguous arguments. In our framework, it is natural to have one relation for each predicate encompassing all of that predicate's arguments. As might be expected, the shared task's notion of roles coincides exactly with ours.

The CoNLL 2005 shared task specifies a system of linguistically-motivated constraints based upon roles. Roles

are characterized as being either core roles or non-core roles. For each predicate, each core role can only occur up to one time, while non-core roles can occur any number of times. Additionally, the requirement that each relation coincide one-to-one with a predicate could be taken as a constraint.

The shared task published a script to use for evaluation. The metric used is a per-argument F_1-score, allowing partial credit for relations based on the number of correct arguments, but not assigning partial credit for partially-overlapping argument spans.

Models for this dataset tend to be span extraction models at their core. Since predicates are known a priori, and since each predicate corresponds with one relation, a common approach is to use a sentence with a single marked predicate as input to a span extraction model, putting all identified spans together with the predicate into one relation. On top of this premise, individual architectures distinguish themselves largely by how they handle the task's constraints. For example, He et al. (2017a) use A* search during decoding time to find span sets which conform with the constraints, while Papay et al. (2022) use a CRF variant to enforce constraints during both training

and decoding.

2.4.3 GENIA event extraction

GENIA (Kim et al., 2003) is a corpus of biomedical text, annotated with a number of structures. In 2009, a shared task for event extraction was announced using this corpus (Kim et al., 2009). This task involves identifying descriptions of biomedical events involving proteins, such as instances of gene expression or protein phosphorylation. Each event involves at least one protein, but may involve additional proteins, mentions of binding sites, or other events. All protein names are marked in the input, but everything else must be predicted by models.

Despite the differing terminology, GENIA's events can be mapped directly to our notion of relations. Proteins and binding sites are all realized as text spans, and the various types of events can be represented by multiple relation types. GENIA defines different types of arguments, a notion which matches our notion of roles.

Similarly to SRL with OntoNotes, GENIA enforces constraints based on how often different roles can occur in different events. Some roles are required exactly once in

an event, some are optional, and others may occur any number of times in a single event. Interestingly, all events are "triggered" by mentions of their name: in our terminology, each relation type has a corresponding entity type, and there must be a one-to-one mapping between those entities and relations.

Perhaps the most structurally interesting aspect of GENIA is the presence of higher-order relations – relations can participate as entities in other relations. Such structures are allowed by our formalisms, but are somewhat rare "in the wild." Nonetheless, due to GENIA's one-to-one mapping between relations and trigger spans, the presence of these higher-order relations does not lead to too many complications, as any higher-order relation involving another relation as an entity can be re-interpreted as a first-order relation involving that other relation's trigger span.

The 2009 event extraction shared task defines a number of evaluation metrics for the task. All of these are relation-level F_1-scores, but differ in their assignment of partial credit. While a strict evaluation mode requires relations that match exactly, recursively checking equality of sub-relations, other modes relax this recursive checking and

assign partial credit for partial span overlaps.

Due to the complexity of GENIA's possible relation structures, most approaches to this task tend to be pipelines, which can decompose the complex structured prediction task into manageable subtasks. For example, Buyko et al. (2009) use a pipeline involving dependency parsing, trigger span identification, and a number of graph pruning steps to arrive at the final relations. Trieu et al. (2020) is a more modern approach, using deep neural networks for individual subtasks, and even training these components jointly. However, the inference procedure still largely resembles a pipeline model, with the predictions of early subtasks being used as input for latter subtasks.

Chapter 3

Formal specification of relation extraction tasks

In our quest towards task-general relation extraction models, it might seem desirable to seek model architectures which are "agnostic" to the specifics of an individual task. After all, we might expect a model which has no knowledge of any task-specific properties to generalize well across tasks which differ in those properties. This hypothesis, though intuitive, does not work well in practice. From experience, we find that the best models for different relation extraction tasks often achieve their high performance by leveraging a priori knowledge about their tasks. The lack of generalizability in such models is precisely a consequence of this reliance on task-specific properties.

Thus, if models can't simply ignore task-specific properties, but at the same time a reliance on these properties leads to a lack of generality, a fertile ground for compromise becomes apparent in formal description of tasks' properties and models' assumptions. If we can formally describe the structural properties of many diverse relation extraction tasks using the same descriptive language, we can formalize what exactly makes tasks different, and when exactly these differences influence the applicability of model architectures. Furthermore, such formal descriptions can allow architectures to adapt themselves to a particular task, automatically selecting hyperparameters to match the properties of the task at hand.

This chapter will study the formal specification of relation extraction tasks, and how such specification can facilitate the development of task-general relation extraction architectures. In order to investigate in this direction, and as a concrete contribution in its own right, we develop DeRE (Declarative Relation Extraction), a software framework for specifying the structures of relation extraction tasks, and developing architectures which can account for tasks' structures. With DeRE, we hope to encourage further research into task-general architectures

by providing a framework which separates task-specific *frontends* from task-general *backends*.

DeRE enables users to

- specify (novel or established) relation extraction tasks in terms of their structural properties and assumptions in the frontend
- develop general architectures for relation extraction, capable of instantiating models for specific tasks based on the specification of those tasks' structures in the backend, and
- instantiate, train, and evaluate models for specific relation extraction tasks.

DeRE achieves this by providing (a) a general mechanism to declaratively specify relation extraction tasks, and (b) a shared processing framework that decouples frontend and backend. For the cases where architectures do generalize well, this allows users to easily select architectures and tasks independently, disentangling the questions of "What task do I have?" and "which model should I use?". Furthermore, when the generality of architectures isn't known, DeRE provides an attractive testbed for investigating the generality of architectures, enabling users to

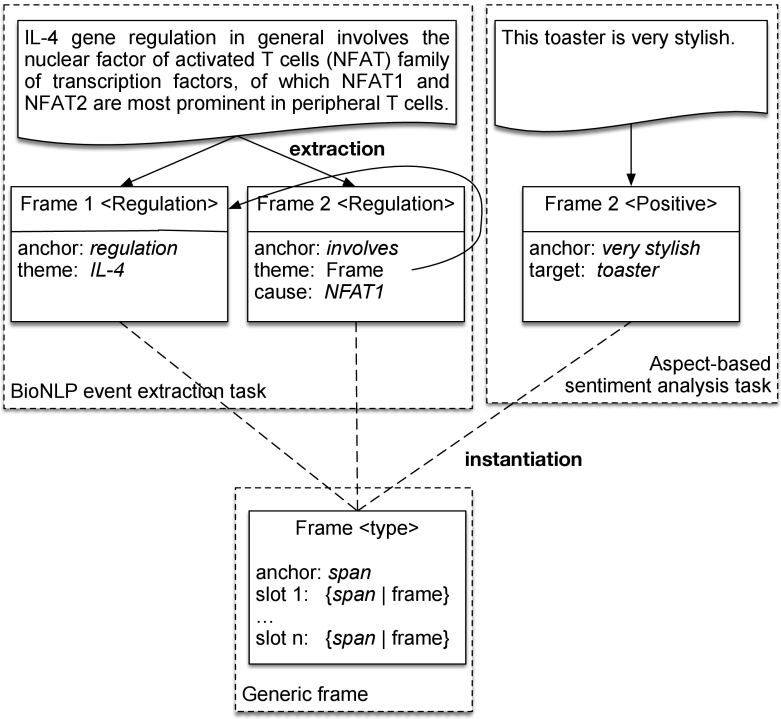


Figure 3.1: Example formalizations of two different tasks in terms of frames, slots, and spans.

painlessly experiment with novel tasks, novel architectures, and novel combinations therebetween.

The declarative specification of a task (which we call a *schema*) describes a task's structure in terms of *frames* and *spans*, concepts which map almost directly to the notions of relations and spans as introduced in Section 2.2. Figure 3.1 shows the general structure of frames, and two concrete instantiations for specific relation extraction tasks, BioNLP event extraction and aspect-based sentiment analysis. Each frame is anchored (triggered) by a span, e.g. a BioNLP event anchor, such as "regulation" or "involves", or a subjective evaluating phrase like "very stylish."

Frames hold a task-specific number of typed slots, filled by relation arguments. The frames for ABSA have a slot filled by the target (aspect) of the sentiment while the frames for the BioNLP regulation event hold a Theme slot and an optional Cause slot. While anchors are always textual spans, slots can be filled by either spans or frames, depending on the task specification. We argue that this simple setup can model an interesting subset of relation extraction tasks. Note that the framework poses no theoretical restrictions to the window from which frames are

extracted. Thus, it can model sentence-level, document-level as well as multi-document tasks.

3.1 Related Work

The framework we present here is quite general, and accommodates many common relation extraction tasks, such as the BioNLP shared task (Kim et al., 2009), semantic role labeling (Das et al., 2014), and (temporal) slot filling (Surdeanu, 2013). However, as is the trend for relation extraction tasks, all systems we are aware of for solving these tasks are tailored to specific scenarios (Angeli et al., 2016; Adel et al., 2016, i.a.). As a result, it is not straightforward to apply them to other use cases. In contrast, our framework is designed to enable the creation of task- and domain-general architectures.

Clarke et al. (2012) develop an NLP component manager which combines several existing NLP tools in a pipeline. Similarly, Curran (2003) aims at a general NLP infrastructure but only reports implementations of non-relational sequence-tagging tasks. Examples of the few available toolkits which are intended to provide users with the possibility of automatically extracting informa-

tion from text data are Jet (Java Extraction Toolkit), GATE (General Architecture for Text Engineering, Cunningham et al., 2013), UIMA (Unstructured Information Management Architecture, Ferrucci and Lally, 2004), FACTORIE McCallum et al. (2009) and Stanbol, which integrates other NLP frameworks, e.g. OpenNLP Morton et al. (2005).

Stanbol and OpenNLP, however, focus on tagging tasks and do not provide tools for relation extraction. FACTORIE is a general approach to formulate factor graphs for arbitrary tasks. Our framework allows arbitrary model paradigms to be used as backends, and is focused on relation extraction, which enables the abstraction layers introduced earlier. Jet, on the other hand, is an information extraction engine developed specifically for the ACE task specification.

GATE is most similar to our framework in scope. It offers both a framework for programmers and an environment for language engineers and computational linguists. However, it is a very general framework, and working with it requires both domain and machine learning knowledge. In contrast, our framework provides end users with an interface for training models on new tasks

without requiring any specific knowledge.

3.2 Framework design

As a software framework, DeRE is a useful tool both for those wishing to develop new architectures, and to those wishing to use existing architectures for a specific task. For researchers and model developers, DeRE acts as a convenient library for quickly developing task-general relation extraction models. It provides a convenient API for loading and saving corpora, representing and manipulating frames and spans, and evaluating model predictions. Furthermore, we provide tools for reading and parsing task schemata, so that developers can easily write architectures which automatically adapt to tasks' structures. For end-users, DeRE makes it straightforward to apply architectures to new or existing tasks. As long as a user can specify the task's structure as a DeRE task schema, they can apply any backend architecture they would like to the task with no additional programming, and without any knowledge of model internals.

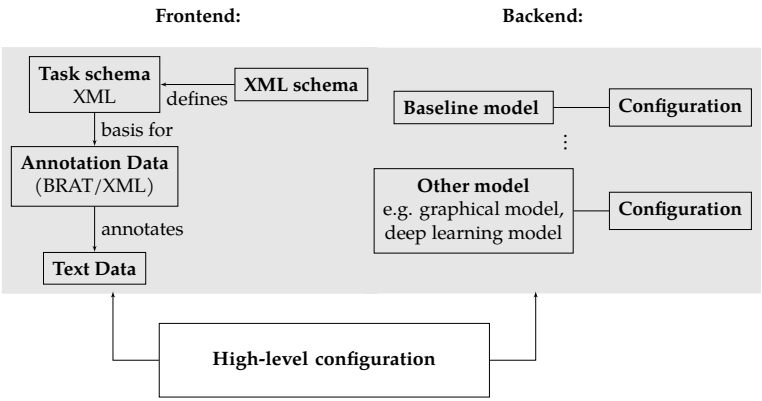


Figure 3.2: Structure of the DeRE framework.

3.2.1 Framework structure

Figure 3.2 illustrates the structure of the DeRE framework. It is composed of two main components: The *frontend* comprises the user specification of the task ("task schema"), including the types of spans and entities to be identified, and the possible relations that can exist between them. It manages reading corpora and annotation files and provides an interface for users. The *backend* hosts the models that make actual predictions for spans, frames, and slots, given the task schema, and their configurations. DeRE backends follow a modular design,

wherein different backends, using different methods for prediction, can be used interchangeably with minimal changes to the frontend.

3.2.2 Task schemata

A *task schema* is DeRE's formal specification of the structural properties of a task. One goal of DeRE's design is to concentrate all specifics about particular tasks into these schemata – instead of incorporating task-specific assumptions directly into model architecture code, task-general architectures can be given a task schema as input, and adapt themselves dynamically to the structure of a specific task.

A task schema specifies possible relation structures in terms of *frames*, *spans*, and *slots*. These concepts map almost directly to the notions of *relations*, *spans*, and *roles* respectively, as introduced in 2.2. Each frame has some number of slots, as defined by its frame type, and each of these slots can be filled by zero or more spans or other

```
<deREschema name="BioNLP-ST 2009" ver="0.01"
   ↪ auth="Klinger">
  <spantypes>
    <span name="Protein" predict="False"/>
    <span name="Gene_expression" anchors=
       ↪ "Gene_expression" predict="True"/>
    <span name="Binding" anchors="Binding"
       ↪ predict="True"/>
  </spantypes>
  <frames>
    <frame name="Gene_expression">
      <slot name="Theme" types="Protein"
         ↪ cardinality="1"/>
    </frame>
    <frame name="Binding">
      <slot name="Theme" types="Protein"
         ↪ mincardinality="0"/>
    </frame>
  </frames>
</deREschema>
```

Figure 3.3: A small but complete task schema for part of the BioNLP shared task. Three span types are specified: `Protein`, `Gene_expression`, and `Binding`. The latter two anchor frames of the same name. Both frames possess a single slot `Theme` which can be filled by `Protein` spans. `Gene_expression` frames always have exactly one `Theme`, while `Binding` frames may have zero or more `Themes`.

frames. A task schema specifies:

- What frame types are present in the task
- What span types are present in the task
- Which spans and frames are present in the input, and which must be predicted
- What slots each frame type has
- For each slot, what types of spans of frames can fill it
- For each slot, how many spans or frames can fill it

Concretely, task schemata are written as XML files. Figure 3.3 gives an example task schema file, for a subset of the BioNLP shared task (Kim et al., 2009).

As task schemata allow users to specify which types of relation structures are permitted or forbidden for a particular relation extraction task, they can be viewed as a specification language for task-specific constraints. However, while we define constraints very broadly in Section 2.2.5, task schemata are limited in their expressivity to type agreement and cardinality constraints within individual relations. This leaves unrepresentable many constraints for real tasks, such as the constraint that each

TACRED document contain at most one relation [1], or constraints which recquire some roles to co-occur with other roles within OnotoNotes relations. Limiting expressibility was a conscious design decision – the more expressive we make task schemata, the more difficult it becomes to write a task-general architecture capable of respecting those task schemata. In the end, we chose type and cardinality constraints as these are particularly common constraint types across tasks, and are simple enough that many approaches to modeling should be able to enforce them.

3.2.3 Data files

Annotated data, needed for training models, are provided to DeRE as annotation files. We currently support annotations in the BRAT (Stenetorp et al., 2012) format, which represents spans and relations in separate files from the text in terms of character offsets. BRAT's formalisms match quite closely with those of DeRE, allowing the represen-

[1] In fact, under the default known-entities setting for TACRED, this constraint can be enforced by anchoring the relation frames to one of the given arguments, but such an approach would not work when entities must also be predicted.

tation of typed spans, and anchored relations between arbitrarily many spans. cf. Figure 3.4 shows an example of such a BRAT annotation file.

```
T1      Protein 1650 1655       IP-10
T2      Protein 951 955 PU.1
T3      Protein  1665 1670      ISG54
T4      Protein 978 992 CSF receptor
T5      Binding 932 937 binds
T6      Gene_expression 1634 1644 expression
E1      Binding:T5 Theme:T2 Theme2:T4
E2      Gene_expression:T6 Theme:T1
E3      Gene_expression:T6 Theme:T3
```

Figure 3.4: An example annotation in BRAT format, following the task specification from Figure 3.3. The text-bound annotations `T` are the span annotations, the event annotations `E` define our frames.

3.3 Proof-of-concept system

As a proof of concept, we implement a simple backend architecture comprising a pipeline of traditional NLP

formalizations: First, spans relevant for the task are extracted. Then, a classifier decides for each pair of relevant spans which slots of which frame they are likely to fill. Finally, a heuristic decoding step compiles the results into frames. Figure 3.5 illustrates this pipeline. The proof-of-concept system only supports non-recursive structures: slots of frames cannot be filled by other frames, but must be filled by spans – i.e., the right-hand BioNLP frame from Figure 3.1 could not be predicted in this implementation. Note that this is only a proof-of-concept baseline, and that the framework is not limited to pipeline models. Models which can cope with recursive structures, as well as those which predict entities and relations jointly, can also be implemented as DeRE backends.

3.3.1 Span extraction

We cast the span extraction problem as a BIO-style sequence-labeling task that predicts span boundaries. To model overlapping spans, we train one model per span type which outputs all spans of that type. Our proof-of-concept system uses non-neural conditional random fields (Lafferty et al., 2001). The feature set consists of the lower-

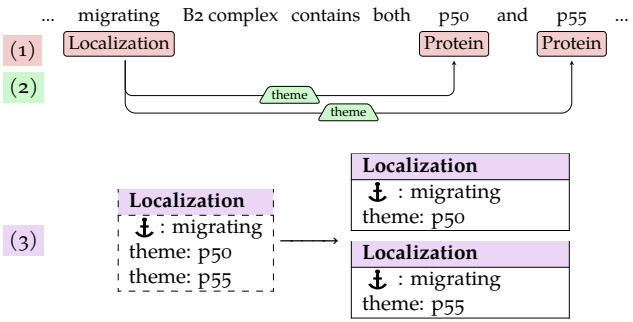


Figure 3.5: Proof-of-concept pipeline: span identification (1), slot classification (2), and decoding into frames (3). ⚓ : frame anchors (triggers)

cased words, their stems, their shape (orthographic case, digits, punctuation), and a flag indicating whether the word is included in a task-specific gazetteer. All features (except the last one) are applicable to any NLP task. The gazetteer feature is based on a simple lexicon of label-specific words (e.g. positive words for detecting positive spans for sentiment analysis) and can be instantiated without any technical knowledge.

3.3.2 Slot classification

Once the spans are identified, the slot classifier is used to predict which slots of which frame they are likely to fill. We break this question down to a classification task at the level of span pairs – one anchor span representing a frame, and another span representing a potential argument. The search space is restricted to those pairs with compatible types according to the schema.

Formally, the classifier takes as input the set S of all spans identified previously, along with a task schema. For each pair $(s_i, s_j) \in S^2$ of spans following the task schema, our classifier produces as output either a single relation label r_{ij}, or NR (no relation)[2] if the two spans are unrelated. Conceptually, two spans s_i and s_j are related iff s_i anchors a frame, and s_j fills a slot in that same frame. Relation labels r_{ij} are pairs (f_i, l_j), where f_i is the frame type anchored by s_i and l_j is the slot type in f_i that s_j fills. This enables us to model, e.g. in the task schema in Figure 3.3, Binding.Theme and Gene_expression.Theme as separate relations. A linear support vector machine is used to predict the most likely relation label (or NR).

[2]We generate negative examples automatically.

Users can enable subsampling of negative examples.

As outlined in the introduction, the features we take into account are included with the aim of being task-agnostic. Intra-span features are types of identified spans and the bag of words in both spans. Inter-span features take into account context. We use the bag of words of tokens between the spans, and of the tokens on the shortest path connecting the spans in a parsed dependency tree, which we assume to accurately capture the relationship expressed by the slot that links the two spans. Since spans can contain multiple tokens, there can be several shortest paths between tokens from the two spans. Under the assumption that tokens in a span are closely related to each other, we select the shortest of these paths. In addition, we also use a bag of bigrams of alternating label-token sequence on that same path. Finally, we measure the length of the shortest path and the token distance.

3.3.3 Decoding

Once the slot classifier identifies all related span pairs, the decoding step generates frames. Pairs of spans (s_i, s_j) that stand in a relation r are first partitioned into equiv-

alence classes C_h according to their anchor span (i.e., $(s_i, s_j) \in C_i$). It would be possible to produce one frame for each equivalence class C_h, anchored by the common anchoring span s_h, and with slots filled according to each span pair's relation label r. However, as equivalence classes can be arbitrarily large, this would allow for each slot to be filled by arbitrarily many spans (as illustrated in the bottom-left of Figure 3.5). As the task schema might impose cardinality constraints, further processing is required to ensure that all produced frames are consistent with the task schema. For each equivalence class C_h, we consider all possible *legal frames* – i.e., all frames that are consistent with the task schema and whose slots are filled according to some subset of C_h. Of these legal frames, we retain all *maximally-filled legal frames* (see bottom-right of Figure 3.5).

3.3.4 Evaluation and results

To demonstrate the feasibility of our proof of concept, we report results with this configuration on the 2009 BioNLP shared task, for which we re-use the original evaluation machinery. The evaluation calculates the F_1 scores for

the individual frames (events in the BioNLP task) using a soft matching for anchor boundaries and approximate recursive matching. Table 3.1 provides the results of our simple system on that task. Due to the restriction of our proof of concept to non-recursive structures (cf. Section 3.3), we only report on the BioNLP event types where all slots are filled by spans. In comparison to the second-ranked system, which also reports results on dev (Buyko et al., 2009), our performance is slightly lower (1 percentage point less for protein catabolism, 13pp less for gene expression and phosphorylation, but 11pp more for localization). This confirms the general usability of our general method.

Due to DeRE's separation of frontend and backend, this same architecture can be directly applied to other tasks – table 3.2 provides the results of applying it to the USAGE corpus for aspect based sentiment analysis (Klinger and Cimiano, 2014), with 10-fold cross validation on the English subset. In comparison to previous results, our numbers are very low. Previous work showed that joint inference had a large positive effect on performance (Klinger and Cimiano, 2013; Yang and Cardie, 2013), an approach not taken by our proof-of-concept system. However, this

Event Class	Precision	Recall	F1
Gene_expression	68.12	57.30	62.25
Transcription	70.59	14.63	24.24
Protein_catabolism	64.00	76.19	69.57
Phosphorylation	65.85	57.45	61.36
Localization	78.57	41.51	54.32
SVT-TOTAL	68.46	50.27	57.97

Table 3.1: Performance of the proof-of-concept system for biomedical relation extraction (BioNLP'09 dev set).

proof-of-concept implementation of the same model already shows the reusability of our framework by only changing the task schema specification. It motivates and enables further research on reusable models across tasks with different needs.

3.3.5 Technical details and availability

The framework is implemented in Python, following an object-oriented design for frontend and backends to support easy interchangeability of components. The choice of Python will also help with future integration of neural network models. For the proof-of-concept backend,

Sentiment Class	Precision	Recall	F1
Positive	41.07	24.19	28.57
Negative	26.68	7.15	11.00
Neutral	5.83	4.50	5.08

Table 3.2: Performance of the proof-of-concept system for aspect based sentiment analysis (10-fold cross-validation on USAGE corpus).

we use scikit-learn for feature extraction and training Pedregosa et al. (2011) with crfsuite and liblinear. Tokenization and stemming is done with NLTK Loper and Bird (2002), dependency features are extracted with spacy Honnibal and Johnson (2015) and dependency graphs are stored and processed using NetworkX Schult (2008). The code is available under the Apache 2.0 License.[3]

3.4 Conclusion

This chapter introduced DeRE, a general framework for doing task-general relation extraction, built on a notion of formal specification of tasks. DeRE facilitates the de-

[3] http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/DeRE.en.html

velopment of task-general models, and makes it easy to apply existing models to new tasks.

From a more theoretical perspective, DeRE acts as demonstration of the utility of formal task specifications for task generality. By drawing a clear dividing line between a task-general backend and a task-specific frontend, DeRE allows architectures to rely on task-specific priors in a way that does not hinder those architectures' generalizability. By declaring task structures in a machine-readable way, we can concentrate all of our task-specific assumptions into our formal task specifications, leaving our model code task-general. In fact, we use a similar pattern, albeit with a different specification language, for defining task-general architectures in Chapter 5.

While formal specification of tasks allows for more general architectures which can adapt themselves to the task at hand, DeRE-like approaches only solve some of the obstacles in the way of fully task-general architectures. Many architectures which work well for some tasks are fundamentally incompatible with the structures of others, and no amount of formal specification will fix that. While meta-architectures can be devised to pick the best applicable architecture for a task at hand, the vast space

of potential task structures makes it difficult for such a meta-architecture to cover a significant subset of tasks. Furthermore, the approach to task-generality presented in this chapter only accounts for those task properties which are formally specifiable and specified. In the case where an architecture's applicability is dictated by properties not accounted for in a task schema, we are forced to fall back to good old fashioned informal reasoning and empirical trial and error. Nonetheless, DeRE provides an elegant framework for developing architectures which can generalize to some extent, and applying those architectures to tasks within their range or applicability.

Chapter 4

Task-generality for span extraction

As span extraction models form critical components of most relation extraction models, identifying task-general architectures for span extraction is an important step towards creating task-general relation extraction architectures. Interestingly, while task-generality for full relation extraction is complicated by the vast space of task structures and corresponding structure-specific assumptions made by common architectures, the situation is quite a bit simpler for span extraction. While there is still a rich variety of task-specific properties in span extraction tasks, most of these properties don't directly conflict with the hard constraints and assumptions employed by common span extraction architectures, and therefore don't

limit the fundamental applicability of architectures to tasks. In fact, when deciding if an architecture is applicable for a task, the only two questions that usually need to be answered are "Are nesting or overlapping spans allowed?" and "Can spans be predicted independently of one another?"

This chapter will focus on the cases where the answers to these questions are "No." and "Not really, but we might try to anyway." [1] This limitation is general enough to admit a wide range of tasks and datasets, while still specific enough to allow us to make direct comparisons between different architectures and between different tasks. Within this space, while all of our architectures will be *applicable* to all tasks, we certainly don't expect them to be equally *performant*: Certain architectures might be better-suited to certain tasks, not due to any hard-constraints or task-specific assumptions, but rather to subtler interactions between the properties of the data distribution and

[1] More concretely, it is usually never ideal to treat spans as independent of one another, but we will be focusing on tasks where doing so doesn't lose us too much. In fact, we will investigate architectures which do enforce some types of dependencies between spans in a document, but the types of dependencies we will be able to represent are rather limited (in general, only between directly abutting spans).

the inductive biases of the architecture.

This chapter will comprise two sections. In Section 4.1, we investigate corpus-generality for quotation extraction. We find that designing architectures capable of generalizing well across different corpora faces many of the same challenges as are seen in task-generality. We present an architecture capable of generalizing well across different corpora, languages, and modalities, and evaluate its performance across four corpora. In Section 4.2, we discuss a more systematic study of different architectures' performance across distinct tasks. We use performance prediction to analyze which architecture components work well for what kinds of tasks, and which generalize well across tasks.

4.1 Corpus- and language-general quotation detection

Quotation is a general notion that covers different kinds of direct and indirect speech, thought, and writing in text (Semino and Short, 2004). Quotations are a prominent linguistic device used to express claims, assessments, or

attitudes attributed to speakers. Consequently, the analysis of quotations is important in many areas of computational linguistics and digital humanities, providing evidence for speaker relationships (Elson et al., 2010; Agarwal et al., 2012), inter-speaker sentiment (Nalisnick and Baird, 2013), politeness (Faruqui and Pado, 2012), and narrative structure (Jannidis et al., 2018).

As is often the case with semantic phenomena, manual annotation of quotations has shown to be slow and resource-intensive, in particular when undertaken in conjunction with the annotation of speakers and information quality (Brunner, 2013; Pareti, 2015). This provides the rationale for automatic *quotation extraction* methods. As quotations are generally contiguous spans of text, or are at least made up of such spans, it is natural to conceptualize quotation extraction as a span extraction task.

Not surprisingly, existing corpora differ substantially across a number of relevant dimensions, including text genre, annotation scheme, and theoretical assumptions. For example, Pareti et al. (2013) focus exclusively on newspaper text and focus on developing a *uniform* annotation schema that captures the shared properties of all kinds of annotations. Thus, even though this corpus

contains direct, indirect, and mixed quotations, these are not marked as instances of their specific subtypes. In addition, each quote is assumed to be introduced by a *cue*:

(1) Hillary Clinton on Saturday cue[acknowledged] quote[the state of the economy is good].

This assumption is generally true for newspaper text, and simplifies the task of quotation detection.

The situation is rather different in the literary texts considered by Semino and Short (2004). Cues are much more varied, and are sometimes omitted entirely, such as in this exchange from Dickens' *Christmas Carol*:

(2) quote[“Much!”] – Marley's voice, no doubt about it.
quote[“Who are you?”]
quote[“Ask me who I was.”]

The study follows a generally more *differentiating* approach. It develops and annotates a rich typology of different subtypes of quotations to distinguish, e.g., direct from indirect quotations, and speech from thoughts from writing.

For the most part, existing models for quotation detection were developed for one specific corpus. This leads to two problems:

1. The models inherit the corpora's *structural and theoretical assumptions*, such as the presence of a cue assumed by models for the Pareti et al. (2013) corpus.

2. The models typically include *domain-specific* features and knowledge sources that happened to be available from the corpus, such as lists of likely cue verbs or syntactic realizations of quotations.

In this section, we approach the question of *corpus-generality* for quotation detection in much the same way that we deal with task-generality elsewhere this dissertation. We find that corpus generality acts as a microcosm for the greater problem of task generality, and that designing a corpus-general architecture for a specific task faces many of the same challenges as are faced when developing task-general models. We present a neural model architecture for automatic quotation detection that makes as few assumptions as possible about the corpus to be

modeled, but is still expressive enough to deal with the challenges inherent in quotation detection.

4.1.1 Related work: datasets and models

We now review the state of the art in automatic quotation annotation, describing the three major quotation corpora for English and German and the corresponding models. We exclude corpora that focus on one specific quotation subtype such as the Columbia Speech Attribution corpus (Elson and McKeown, 2010) which only covers direct speech.

PARC Dataset

Dataset. The Penn Attribution Relation Corpus (Pareti, 2015), version 3 (PARC3) is a subset of the Penn Treebank, annotated with quotations and attribution relations. It consists of English newswire text from the Wall Street Journal. Each attribution relation consists of a cue, optionally a source (speaker), and content (quotation span), all marked as text spans. As part of the Penn Treebank, PARC3 provides manually annotated tokenization, POS tags, lemmas, and constituency parses.

Quotation spans are not labeled with more specific types, but PARC3 distinguishes informally (based on the surface form) between direct quotations (starting and ending with quotation marks), indirect quotations (without any quotation marks), and mixed quotations (everything else).

Pareti model. Pareti (2015), an extension of Pareti et al. (2013), presents a pipeline architecture for quotation annotation. It first applies a *k*-NN classifier to identify quotation cues within the corpus. Then, a linear-chain conditional random field (CRF) is used to identify quotation spans in the vicinity of each cue. Both components of the Pareti model rely on a corpus-specific feature set, including a list of known roles, organizations, and titles, and handcrafted features sensitive to punctuation conventions in English newswire text.

Scheible model. Scheible et al. (2016) retain the pipeline architecture of Pareti (2015) and its feature set, but replace the components. Cue annotation is performed with an averaged perceptron. More importantly, they replace quotation annotation proper with a sampling-based procedure: a perceptron samples tokens as likely span boundaries, which are then combined into complete quotation

spans, using a semi-Markov model.

STOP Dataset

Semino and Short (2004) presents a corpus-based ontology of quotations in English text. It introduces two dimensions: (a), speech vs. thought vs. writing; and (b), direct vs. indirect vs. free indirect vs. reported, yielding a Cartesian product of twelve quotation subclasses. These are used to annotate the Speech, Thought, and Writing Presentation corpus (STOP). It comprises 120 sections, split evenly across three genres (fiction, newspaper, and biographies), of about 2,000 words each (Total size: 250,000 tokens; 8,000 quotations). The corpus has no linguistic annotation: the only features available are words' surface forms. To our knowledge, there are no other published models for this dataset.

Redewiedergabe Dataset

Dataset The Redewiedergabe ('reported speech') corpus (RWG) (Brunner, 2013) is a corpus of German narrative text, comprising thirteen public-domain German narratives from between 1787 and 1913. The quotation anno-

tations in RWG adopt the scheme by Semino and Short (2004) and distinguish direct, indirect, free indirect, and reported variants of speech, thought, and writing. The total size of the corpus is 57,000 tokens, and 17,000 quotation spans.

Unlike STOP, RWG contains some linguistic information, namely POS tags, lemmas, and morphological features (case, number, gender). However, this information is not manually annotated, but is obtained automatically using various NLP systems.

Models. Brunner (2013) proposes two models for quotation annotation on RWG. Both models work at the sentence level and predict only the presence of absence of quotations in sentences, and not the quotations' exact spans (even though this information is annotated). The first model is rule-based (**Brunner RB**). It uses a set of handcrafted rules to identify direct, indirect, reported, and free indirect quotations. The second model (**Brunner ML**) is a simple classification model based on random forests.

4.1.2 Neural Quotation Detection (NQD)

We now define a neural architecture, NQD, with the goal of modeling the quotations in all three corpora described in Section 4.1.1. We design our model to leverage the commonalities across datasets, while not depending on the features of any dataset in particular. As all datasets involve long quotation spans with long-distance dependencies, an LSTM-based approach was natural, given such models' ability to capture very long-distance dependencies of up to 200 tokens (Khandelwal et al., 2018). Conversely, given the structural differences between corpora, we found we could not benefit from a pipeline model like those employed by Pareti (2015) and Scheible et al. (2016), and thus predict quotations directly without first looking for cues.

NQD frames quotation prediction as token classification, classifying each token as either beginning a quotation (BEGIN), ending a quotation (END), or neither (NEITHER). Quotation spans then consist of all tokens starting with a BEGIN tag, up to (but not including) the next END or BEGIN tag, or the end of sequence. This model is not limited to the sentence level: it is able to make predictions for a whole document and in this manner can capture very

long quotation spans.

The input texts are represented as a sequence of tokens, where each token is a bag of features. Each feature value is represented as an n-dimensional continuous vector, and each token is represented as the sum of these vectors. This approach to feature representation allows our model to work with corpora with arbitrary types of token-level features. In the simplest case, when only raw text is present in the corpus, each token is given a single feature for that token's surface form. If other token-level features are present, such as POS-tags, lemmas, or even parse tree information, these can be incorporated as additional feature vectors, without requiring any changes to the model architecture. Feature vectors can also be initialized to pre-trained representations (e.g. word embeddings) when these are available, or initialized randomly and learned when they are not. Section 4.1.1 describes in detail which features are used for the corpora we consider.

NQD uses sequence-to-sequence models to classify each token as either beginning a quotation span (BEGIN), ending a quotation span (END), or neither of the two (NEITHER). This sequence-to-sequence model comprises a 2-layer bi-LSTM network, with the outputs of the sec-

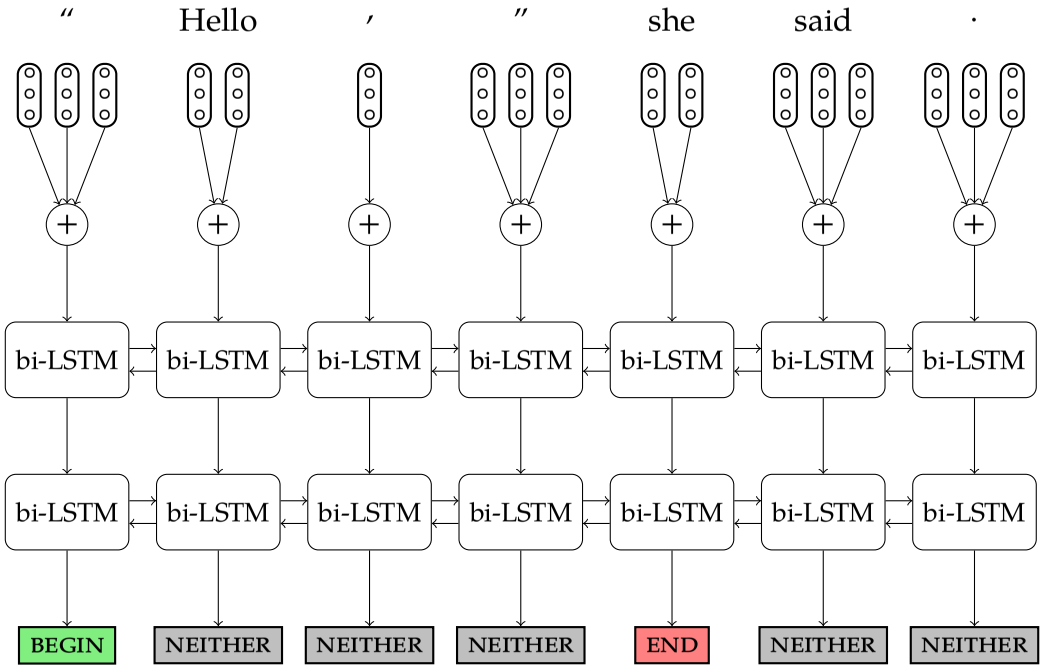


Figure 4.1: The NQD architecture. Tokens are represented as a bag of feature embeddings, and each token sequence is processed by a 2-layer bi-LSTM network, before a max-entropy classifier labels each token.

ond bi-LSTM feeding into a 3-class softmax classifier. Token sequences, represented as described above, are the inputs to these sequence-to-sequence models, and the outputs are a sequence of token labels. Figure 4.1 shows a schematic diagram of the NQD architecture. For datasets with multiple quotation types, NQD uses a separate sequence-to-sequence model for each span type, connecting them by weight sharing.

All code for NQD is available online at `https://www.ims.uni-stuttgart.de/en/research/resources/tools/quote-detection/`.

4.1.3 Experimental evaluation

We now train and test NQD on the three corpora and compare against the state-of-the-art.

Experimental Setup

PARC3. For PARC3, we train a single classifier on the quote content spans and ignore the cue and source spans. As features, we use token surface forms, lemmas, POS tags, as well as, for each token, the bags of constituents that start with, end with, and contain it. These features are

a subset of the features used by Scheible et al. (2016) and Pareti (2015), and like these studies, we use gold standard annotation. We initialize the features for word surface forms with the default GloVe Wikipedia word embeddings (Pennington et al., 2014). Our model makes predictions on entire documents at a time. We use performance on the corpus's development set to guide early stopping during training, and we evaluate on the corpus's test set.

STOP. For STOP, we train four classifiers for the four quote types (direct, indirect, free indirect, reported). We train and evaluate our model on a per-document basis as for PARC3. We use word surface forms (and their GloVe embeddings) as features, we used no other features in this model. As the corpus contains no held-out development or test sets, we used 10-fold cross validation to evaluate our model, using 8 folds for training, 1 for development, as 1 as for testing in each iteration.

RWG. For RWG, we adopt the same four-classifier setting as for STOP, using the word, lemma, POS, and morphological features available. For the sake of comparability with Brunner (2013), we train and evaluate on individual sentences, as opposed to entire documents. We use 10-fold

Model	Features	Overall			Direct			Indirect			Mixed		
		Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Pareti et al. (2013)	W S D	63	**80**	71	88	**94**	91	56	**78**	65	60	67	63
Scheible et al. (2016)	W S D	**71**	79	**75**	93	**94**	**94**	**64**	73	**69**	68	**81**	**74**
NQD	W S	**71**	67	69	**94**	82	88	**64**	64	64	**70**	59	64
NQD	W	61	61	61	90	84	87	53	56	54	60	54	57

Table 4.1: Results on PARC3 (exact span match evaluation)

cross validation again, randomly partitioning all corpus sentences into 10 subsets. We use GloVe embeddings pre-trained on the German Wikipedia.[2]

Evaluation. Previous studies on PARC3 adopted an exact span match setting, i.e., only those predicted spans that exactly match a gold standard span count as true positives. We report precision, recall, and F_1 in this setting for PARC3 and STOP. For RWG, we report the sentence-level accuracy used by Brunner (2013). In this mode, we train and predict with our model as before, but for evaluation we just record whether the model predicts the presence of a quotation type in a sentence.

Model	Features	Overall			Direct			Indirect			Free Indirect			Reported		
		Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
NQD	W	51	66	57	78	83	80	33	49	40	01	04	01	46	58	51

Table 4.2: Results on STOP (exact span match evaluation)

Model	Features	Overall			Direct			Indirect			Free Indirect			Reported		
		Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Brunner (2013) RB	W S	**71**	67	**69**	**87**	81	84	**62**	**81**	**71**	**44**	24	31	**64**	51	**57**
Brunner (2013) ML	W S	63	77	**69**	85	**88**	**87**	47	62	53	29	63	40	45	**56**	50
NQD	W S	60	**78**	68	77	86	82	52	69	60	31	**68**	**42**	34	**56**	43
NQD	W	59	73	65	77	83	80	40	69	50	14	62	23	41	50	45

Table 4.3: Results on RWG (sentence-level accuracy evaluation)

Results

PARC3. The results in Table 4.1 show that NQD cannot beat the performance of Scheible et al. (2016), but does almost as well as Pareti et al. (2013). Given that our model is substantially simpler than either of these two (both include a special cue classifier, dictionaries, etc.), we see this as a success. Our model is competitive with the Scheible et al. model with regard to recall, but shows subpar precision for all quotation types, indicating a remaining weakness in the input encoding: for direct quota-

[2] Available from deepset at `https://deepset.ai/german-word-embeddings`

tions, quote characters should provide strong indicators for quotation boundaries.

Note that these results, as well as the earlier studies (Pareti et al., 2013; Scheible et al., 2016), use unrealistic gold standard features. Therefore, we ran a second version of NQD using only word features, but no tags or structural information. The model is clearly worse, but still surprisingly good at 61% F_1. Not surprisingly, we see the highest drop for indirect quotations, which are most sensitive to syntactic structure.

STOP. To our knowledge, the results in Table 4.2 are the first modeling results on STOP. In comparison to PARC3, the results are noticeably lower. It is still the case that direct quotations are easiest to find, but their F_1 is somewhat lower than in PARC3. Indirect quotations are much more difficult, and free indirect quotations essentially impossible. This involves multiple factors: (a) STOP is significantly smaller, but more varied, than PARC3, providing sparser training data; (b) STOP covers a wider variety of quotation types, some of these are intrinsically difficult to model – in particular free indirect quotations (McHale, 2009).

RWG. The results in Table 4.3 show a picture that is

overall similar to PARC3:[3] NQD does not outperform the state-of-the-art, but approximates it closely despite the lack of corpus-specific tuning. As in STOP, we see the lowest results for free indirect quotations, showing that this class is generally hard to classify. In general, even though this resource's size and annotation are similar to STOP, we see significantly higher numbers. This is mostly due to the different evaluation we use for RWG to compare to previous work: detecting the presence of quotes is easier than identifying their spans.

On RWG, we also run a basic NQD with only word form information. With this corpus and evaluation, this results in a drop of merely 3 points F_1 – due to losses in the indirect and free indirect categories – which bolsters the potential of this configuration.

4.1.4 Error analysis

To gain some insights into the failure modes of NQD, we perform a brief qualitative analysis of the cases where our model gave false predictions.

[3] Brunner (2013) does not report overall results. We compute them as micro-averages over reported per-type results.

These errors can broadly be divided into three categories: cases where the model predicts the presence of extraneous quotations (false positives), cases where the model fails to identify existing quotations (false negatives), and cases where the model correctly identified the presence of a quote, but did not correctly determine its boundaries (boundary mismatch, leading both false positives and false negatives in our exact span evaluation). We focus our error analysis on PARC, the previously best-explored of our three corpora.

False Positives

Among the false positives produced by our model was a surprising number of quotations that are correct according to PARC's guidelines, but which are not annotated in the corpus. As an example, our model correctly identifies the presence of an unannotated quotation in the following sentence:

(3) predicted
Britain's retail price index rose 0.7% in September from August and was up 7.6% for the year, the Central Statistical Office said.

Outside of these cases, proper false positives seem to be rare. Many of the false positives we found were boundary mismatches, discussed separately below.

False Negatives

Among the false negatives we analyzed, we found that the model is most likely to miss "tricky" quotations that are unusual in their grammatical structure. In particular, it tends to miss a class of quotations that are expressed as short noun phrases or adjectival phrases embedded within a non-quotation sentence such as

(4) Mandela, considered gold[the most prominent leader of the ANC] remains in prison. But gold[his release within the next few months] is widely expected.

According to the PARC guidelines, these are cases of quotations since they are attributable statements, but they are difficult for the model to retrieve since they are hard to distinguish from "non-quotation" nominal phrases – in particular in cases like this one, where there are not even overly realized speakers. In STOP and RWG, these cases might arguably not even be annotated as quotations.

Boundary Mismatches

A large proportion of the errors of NQD are boundary errors, where the model identifies the presence of a quotation, but fails to identify its exact boundaries. This can happen when our model correctly predicts one quotation boundary, but not the other.

For example, in the following sentence, our classifier identified the first quotation's beginning, but not its end, causing the classifier to label the remainder of the document as being part of the quote (it also failed to identify the second quotation entirely – a false negative):

(5) He reiterated gold[his opposition to such funding], but expressed gold[hope of a compromise].

This type of error occurs both for noun phrases and verb phrases and embedded sentences, but for different reasons: noun phrases are difficult to recognize since they are not marked by punctuation as are almost all other cases of quotation spans; verb phrases, on the other hand, can become arbitrarily complex. In the case above, the segmentation problems are exacerbated by the fact that the noun phrase quotation span occurs in a complex syn-

tactic environment involving coordination.

4.1.5 Conclusion

Here we have argued that existing models for automatic quotation detection suffer from the tight relation between corpus annotation and model properties, suffering from a corpus specificity similar to the task specificity discussed elsewhere in this dissertation. As an alternative, we have presented a general neural architecture, NQD, which can be trained "as is" and which generalizes well to various corpora that differ in terms of genre, structure, and language. While the architecture does not reach the state of the art on any particular corpus, it performs close to it on all of them. Notably, the architecture is also able to deal relatively graciously with the absence of linguistic information.

As NQD makes independent predictions for each token, it cannot model correlations and mutual exclusions between labels, and there is no guarantee for well-formed output class sequences. In the next section, we will investigate models incorporating linear-chain CRF layers, and show that these can lead to improvements, particularly

when spans are long as is typical in quotation detection.

The overall greatest challenge that NQD faces is data scarcity — all existing corpora with manual annotation are small, and our results show consistently bad performance for infrequent quotation types. In this situation, transfer learning seems like a natural proposition, and our model makes it possible for the first time to apply straightforward transfer learning to quotation annotation.

4.2 Investigating task-generalizability with performance prediction

Due to the rapid development of deep learning, an abundance of model architectures is available for the implementation of span extraction tasks. These include isolated token classification models (Berger et al., 1996; Chieu and Ng, 2003), probabilistic models such as hidden Markov models (Rabiner, 1989), maximum entropy Markov models (McCallum et al., 2000), conditional random fields (Lafferty et al., 2001), recurrent neural networks such as LSTMs (Hochreiter and Schmidhuber, 1997), and trans-

formers such as BERT (Devlin et al., 2019b).

Though we have some understanding what each of these models can and cannot learn, there is, to our knowledge, little work on systematically understanding the task-generality and task-specificity of different architectures: Are there model architectures that work well and generalize well across tasks? When complete task-generality is not possible, can we identify properties of span extraction tasks that can help us select suitable model architectures on a task-by-task basis? Answers to these questions could narrow the scope of architecture search for these tasks, and could help with comparisons between existing methods and more recent developments.

In this section, we address these questions by applying *meta-learning* to span extraction (Vilalta and Drissi, 2002; Vanschoren, 2018). Meta-learning means "systematically observing how different machine learning approaches perform [...] to learn new tasks much faster" (Vanschoren, 2018), with examples such as architecture search (Elsken et al., 2019) and hyperparameter optimization (Bergstra and Bengio, 2012). Our specific approach is to apply *performance prediction* to span extraction tasks, using both task properties and model architectures as

features, in order to obtain a better understanding of the differences among span extraction tasks and architectures.

Our specific contributions are as follows: First, we collect a set of English span extraction tasks, quantify key properties of the tasks (such as how distinct the spans are from their context, and how clearly their boundaries are marked) and formulate hypotheses linking properties to performance (Section 4.2.1). Next, we describe relevant neural model architectures for span extraction (Section 4.2.3). We then train a linear regression model as a meta-model to predict span extraction performance based on model features and task metrics in an unseen-task setting (Section 4.2.4). We find the best of these architectures perform at or close to the state of the art, and their success can be relatively well predicted by the meta-model (Section 4.2.5). Finally, we carry out a detailed analysis of the regression model's parameters (Section 4.2.7), gaining insight into the relationship between span extraction tasks and different neural model architectures. For example, we establish that spans that are not very distinct from their context are consistently difficult to identify, but that CRFs are specifically helpful for this

class of span extraction tasks.

4.2.1 Tasks and datasets

We work with five widely used English span extraction datasets. All of them have non-overlapping spans from a closed set of span types, and are thus amenable to BIO labeling (which we will use for all architectures). In the following, we discuss the datasets chosen, the span types present in those datasets, and the properties of those span types. Since the different span types within a dataset may vary significantly in their distributional properties, the remainder of this work will treat each span type as its own span extraction task, allowing for a finer-grained analysis of span types individually.

Quotation detection: PARC and RiQuA.

We use two quotation detection datasets: The Penn Attribution Relation Corpus (PARC) version 3.0 (Pareti, 2016), as we used in Section 4.1, and the Rich Quotation Attribution Corpus (RiQuA, Papay and Padó, 2020). Both datasets contain a mix of direct and indirect quotation spans in text. The corpora cover articles from the Penn

Treebank (PARC) and 19th century English novels (RIQUA), respectively. Within each text, quotations are identified, along with each quotation's speaker (or source), and its cue (an introducing word, usually a verb like "said"). We model detection of quotations as well as cues. As speaker and addressee identification are primarily relation extraction tasks, relying much more heavily on these spans' relations to quotations and cues rather than their own intrinsic properties, we exclude these span types.

Chunking: CoNLL'00.

Chunking (shallow parsing) is an important preprocessing step in a number of NLP applications. We use the corpus from the 2000 CoNLL shared task on chunking (CoNLL'00) (Tjong Kim Sang and Buchholz, 2000). Like PARC, this corpus consists of a subset of the PTB. This dataset is labeled with non-overlapping chunks of eleven phrase types. In this work, we will consider the seven phrase types with more than 100 instances in the training partition: `ADJP`, `ADVP`, `NP`, `PP`, `PRT`, `SBAR`, and `VP`.

NER: OntoNotes and ChemDNer.

For recognition and classification of proper names, we use the NER layer of OntoNotes Corpus v5.0 (Weischedel et al., 2011) and Biocreative's ChemDNer corpus v1.0 (Krallinger et al., 2015). OntoNotes, a general language NER corpus, is our largest dataset, with over 2.2 million tokens. The NER layer comprises 18 span types, both typical entity types such as `Person` and `Organization` as well as numerical value types such as `Date` and `Quantity`. We use all span types. ChemDNer is a NER corpus specific to chemical and drug names, comprising titles and abstracts from 10,000 PubMed articles. It labels names of chemicals and drugs and assigns them to eight classes, corresponding to chemical name nomenclatures. We use seven span types: `Abbreviation`, `Family`, `Formula`, `Identifier`, `Systematic`, `Trivial`, and `Multiple`. We exclude the class `No class` as infrequent (fewer than 100 instances).

4.2.2 Span type properties and hypotheses

While quotation detection, chunking, and named entity recognition are all span extraction tasks, they vary quite

widely in their properties. As mentioned in the introduction, we know of little work on quantifying the similarities and differences of span types, and thus, span extraction tasks.

We now present four metrics which we propose should capture the relevant characteristics of span types, and make concrete our hypotheses regarding their effect on model performance. Table 4.4 reports the value of each metric for each span type across all datasets.

Frequency is the number of spans for a span type in the dataset's training corpus. It is well established that the performance of a machine learning model benefits from higher amounts of training data (Halevy et al., 2009). Thus, we expect this property to be positively correlated with performance. However, some architectural choices, such as the use of transfer learning, are purported to reduce the data requirements of machine learning models (Pan and Yang, 2009), so we expect a smaller correlation for architectures which incorporate transfer learning.

Span length is the geometric mean of spans' lengths, in tokens. Scheible et al. (2016) note that traditional CRF models perform poorly at the identification of long spans due to the strict Markov assumption they make Lafferty

Dataset	Span type	Freq.	Length	Span dist.	Boundary dist.
PARC	content	17416	13.86	0.15	1.73
	cue	15424	1.16	2.69	1.09
RiQuA	cue	2325	1.05	4.04	1.37
	quotation	4843	14.06	0.22	1.67
CoNLL'00	ADJP	2060	1.22	3.13	1.22
	ADVP	4227	1.07	3.02	0.74
	NP	55048	1.89	0.48	0.65
	PP	21281	1.01	2.08	0.59
	PRT	556	1.00	4.59	2.20
	SBAR	2207	1.02	3.68	1.26
	VP	21467	1.39	1.60	0.50
OntoNotes	cardinal	10901	1.20	3.45	0.90
	date	18791	1.87	2.62	0.88
	event	1009	2.65	3.15	1.32
	facility	1158	2.33	3.54	1.22
	GPE	21938	1.16	3.66	0.81
	language	355	1.03	7.26	1.99
	law	459	2.92	3.16	1.69
	location	2160	1.69	4.14	1.10
	money	5217	2.61	3.87	1.41
	NORP	9341	1.04	4.85	0.98
	ordinal	2195	1.00	5.99	1.39
	organization	24163	1.93	2.22	0.74
	percent	3802	2.30	4.35	1.50
	person	22035	1.51	3.54	1.24
	product	992	1.51	4.58	1.65
	quantity	1240	2.25	3.79	1.35
	time	1703	1.95	3.50	1.24
	work_of_art	1279	2.77	2.15	1.67
ChemDNer	abbreviation	4536	1.17	3.85	0.94
	family	4089	1.44	3.15	0.99
	formula	4445	1.98	2.50	0.99
	identifier	672	2.59	3.61	1.43
	multiple	202	6.49	2.10	1.60
	systematic	6654	2.17	2.14	0.98
	trivial	8832	1.15	3.64	0.86

Table 4.4: All span types considered for each dataset, with frequency, mean span length, span distinctiveness, and boundary distinctiveness.

et al. (2001). Thus, we expect architectures which rely on such assumptions and which have no way to model long distance dependencies to perform poorly on span types with a high average span length, while LSTMs or transformers should do better on long spans (Khandelwal et al., 2018; Vaswani et al., 2017).

Span distinctiveness is a measure of how distinctive the text that comprises spans is compared to the overall text of the corpus. Formally, we define it as the KL divergence $D_{\mathrm{KL}}(\tilde{P}_{\mathrm{span}} \parallel \tilde{P})$, where $\tilde{P}$ is the unigram word distribution of the corpus, and $\tilde{P}_{\mathrm{span}}$ is the unigram distribution of tokens within a span. A high span distinctiveness indicates that different words are used inside spans compared to the rest of the text, while a low span distinctiveness indicates that the word distribution is similar inside and outside of spans.

We expect this property to be positively correlated with model performance. Furthermore, we hypothesize that span types with a high span distinctiveness should be able to rely more heavily on local features, as each token carries strong information about span membership, while low span distinctiveness calls for sequence information. Consequently, we expect that architectures incorporating

sequence models such as CRFs, LSTMs, and transformers should perform better at low-distinctive span types.

Boundary distinctiveness is a measure of how distinctive the starts and ends of spans are. We formalize this in terms of a KL-divergence as well, namely as $D_{\mathrm{KL}}(\tilde{P}_{\text{boundary}} \parallel \tilde{P})$ between the unigram word distribution ($\tilde{P}$) and the distribution of boundary tokens ($\tilde{P}_{\text{boundary}}$), where boundary tokens are those which occur immediately before the start of a span, or immediately after the end of a span. A high boundary distinctiveness indicates that the start and end points of spans are easy to spot, while low distinctiveness indicates smooth transitions.

We expect boundary distinctiveness to be positively correlated with model performance, based on studies that obtained improvements from specifically modeling the transition between span and context Todorovic et al. (2008); Scheible et al. (2016). As sequence information is required to utilize boundary information, high boundary distinctiveness should improve performance more for LSTMs, CRFs, or transformers.

Task profiles

As Table 4.4 shows, the metrics we propose appear to capture the task structure of the span types well: quotations have long spans with low span distinctiveness (anything can be said) but high boundary distinctiveness (punctuation, cues). In chunking, the more common chunk types have notably lower span distinctiveness, as these chunks comprise a significant portion of the entire corpus, and NER spans show high distinctiveness (semantic classes) but are short and have somewhat indistinct boundaries as well.

4.2.3 Model architectures

For span extraction, we use the BIO framework (Ramshaw and Marcus, 1999), framing span extraction as a sequence labeling task. As each span type has its own B and I labels, and there is one O label, a dataset with n span types leads to a $(2n + 1)$-label classification problem for each token.

We investigate a set of sequence labeling models, ranging from baselines to state-of-the-art architectures. We group our models by common components, and build

complex models through combination of simpler models. Except for the models using BERT, all architectures assume one 300-dimensional GloVe embedding (Pennington et al., 2014) per token as input.

Baseline

As a baseline model, we use a simple token-level classifier. This architecture labels each token using a softmax classifier without access to sequence information (neither at the label level nor at the feature level).

CRF

This model uses a linear-chain conditional random field (CRF) to predict token label sequences (Lafferty et al., 2001). It can access neighboring labels in the sequence of predictions.

LSTM and LSTM+CRF

These architectures incorporate Bi-directional LSTMs (biLSTMs) (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997) as components. The simplest architecture,

LSTM, passes the inputs through a 2-layer biLSTM network, and then predicts token labels using a softmax layer. The LSTM+CRF architecture combines the biLSTM network with a CRF layer, training all weights simultaneously. These models can learn to combine sequential input and labeling information.

BERT and BERT+CRF

These architectures include the pre-trained BERT language model (Devlin et al., 2019b) as a component. The simplest architecture in this category, BERT, comprises a pre-trained BERT encoder and a softmax output layer, which is trained while the BERT encoder is fine-tuned. BERT+CRF combines a BERT encoder with a linear-chain CRF output layer, which directly uses BERT's output embeddings as inputs. In this architecture, the CRF layer is first trained to convergence while BERT's weights are held constant, and then both models are jointly fine-tuned to convergence. As BERT uses WordPiece tokenization (Wu et al., 2016), the input must be re-tokenized for BERT architectures.

BERT+LSTM+CRF

This architecture combines all components previously mentioned. It first uses a pre-trained BERT encoder to generate a sequence of contextualized embeddings. These embeddings are projected to 300 dimensions using a linear layer, yielding a sequence of vectors, which are then used as input for a LSTM+CRF network. As with BERT+CRF, we first train the non-BERT parameters to convergence while holding BERT's parameters fixed, and subsequently fine-tune all parameters jointly.

Handcrafted features

Some studies have shown marked increases in performance by adding hand-crafted features (e.g. Shimaoka et al., 2017). We develop such features for our tasks and treat these to be an additional architecture component. For architectures with this component, a bag of features is extracted for each token (the exact features used for each dataset are enumerated in Table A.1 in the Appendix). For each feature, we learn a 300-dimensional feature embedding which is averaged with the GloVe or BERT embedding to obtain a token embedding. Hand-

crafted features can be used with the Baseline, LSTM, LSTM+CRF, and BERT+LSTM+CRF architectures. BERT and BERT+CRF cannot utilize manual features, as they have no way of accepting token embeddings as input.

4.2.4 Meta-learning model

Recall that our meta-learning model is a model for predicting the performance of the model architectures from Section 4.2.3 when applied to span extraction tasks from Section 4.2.1. We model this task of performance prediction as linear regression, a well established framework for the statistical analysis of language data (Baayen, 2008). The predictors are task properties, model architecture properties, and their interactions, and the dependent variable is (scaled) F_1 score.

While a linear model is not powerful enough to capture the full range of interactions, its weights are immediately interpretable, it can be trained on limited amounts of data, and it does not overfit easily (see Section 4.2.6). All three properties make it a reasonable choice for meta-learning.

Predictors and interactions

As predictors for our performance prediction task, we use the span type properties described above, and a number of binary model properties. For the span type properties [freq] and [span length], we use the logarithms of these values as predictors. The two distinctiveness properties are already logarithms, and so we used them as-is. For model properties, we used four binary predicates: The presence of handcrafted features, of a CRF output layer, of a bi-LSTM layer, and of a BERT layer.

In addition to main effects of properties of models and corpora on performance (does a CRF layer help?), we are also interested in interactions of these properties (does a CRF layer help in particular for longer spans?). As such interactions are not captured automatically in a linear regression model, we encode them as predictors. We include interactions between span type and model properties, as well as among model properties.

All predictors (including interactions) are standardized so as to have a mean of zero and standard deviation of one.

Scaling the predicted performance

Instead of directly predicting the F_1 score, we instead make our predictions in a logarithmic space, which eases the linearity requirements of linear regression. We cannot directly use the logit function to transform F_1 scores into

$$F' = \operatorname{logit}\left(\frac{F_1}{100}\right)$$

since the presence of zeros in our F_1 scores makes this process ill-defined. Instead, we opted for a "padded" logit transformation

$$F' = \operatorname{logit}\left((1-\alpha)\cdot\frac{F_1}{100} + \alpha\cdot\frac{100-F_1}{100}\right)$$

with a hyperparameter $\alpha \in [0, 0.5)$. This rescales the argument of the logit function from $[0, 1]$ to the smaller interval $[\alpha, 1-\alpha]$, avoiding the zero problem of a bare logit. Through cross-validation (cf. Section 4.2.6), we set $\alpha = 0.2$. We use the inverse of this transformation to scale the output of our prediction as an F_1 score, clamping the result to $[0, 100]$.

4.2.5 Experiment

4.2.6 Experimental procedure

Our meta-learning experiment comprises two steps: span extraction model training, and meta-model training.

Span extraction model training

We begin by training and subsequently evaluating each model architecture on each dataset five times, using different random initializations. With 12 model architectures and 5 datasets under consideration, this procedure yields $12 \times 5 \times 5 = 300$ individual experiments.

For each dataset, we use the established train/test partition. Since RiQuA does not come with such a partition, we use cross-validation, partitioning the dataset by its six authors and holding out one author per cross-validation step.

We use early stopping for regularization, stopping training once the (micro-averaged) performance on a validation set reaches its maximum. To prevent overfitting, all models utilize feature dropout – during training, each feature in a token's bag of input features is dropped with

a probability of 50%. At evaluation time, all features are used.

Meta-learning model training

This step involves training our performance prediction model on the F_1 scores obtained from the first step. For each (architecture, span type) pair of the 12 model architectures and 36 span types, we already obtained 5 F_1 scores. This yields a total of $12 \times 36 \times 5 = 2160$ input-output pairs to train our performance prediction model.

We investigate both L^1 and L^2 regularization in an elastic net setting (Zou and Hastie, 2005) but consistently find best cross-validation performance with no regularization whatsoever. Thus, we use ordinary least squares regression.

To ensure that our performance predictions generalize, we use a cross-validation setup when generating model predictions. To generate performance predictions for a particular span type, we train our meta-model on data from all other span types, holding out the span type for which we want a prediction. We repeat this for all 36 span types, holding out a different span type each time, in order to collect performance predictions for each span

type.

Span extraction results

After training and evaluating our span extraction models, we have 5 evaluation F_1 scores for each (architecture, span type) pair. This section summarizes the main findings. Table 4.5 lists the performance of each model architecture on each dataset. Unsurprisingly, BERT+Feat+LSTM+CRF, the model with the most components, performs best on a plurality of 16 span types. This provides strong evidence that this maximal architecture can generalize well across many tasks. However, this dominance is not absolute: BERT+CRF and Feat+LSTM+CRF each perform best on 7 span types, and BERT+LSTM+CRF on 6 span types. Thus, 'bespoke' modeling of span types can evidently improve results.

Even though our architectures are task-agnostic, and not tuned to particular tasks or datasets, our best architectures still perform quite competitively. For instance, on CoNLL'00, our BERT+Feat+LSTM+CRF model comes within 0.12 F_1 points of the best published model's F_1 score of 97.62 (Akbik et al., 2018). For PARC, existing literature focuses only on F_1 scores for content span detection.

		Baseline	Feat+Baseline	CRF	Feat+CRF	LSTM	Feat+LSTM	LSTM+CRF	Feat+LSTM+CRF	BERT	BERT+CRF	BERT+LSTM+CRF	BERT+Feat+LSTM+CRF
PARC	content	0.0	1.6	15.5	40.0	50.3	70.4	69.1	77.2	71.7	76.6	77.8	**78.1**
	cue	68.0	64.9	69.1	68.5	77.1	83.3	82.2	86.3	84.4	85.8	**86.7**	86.4
RQUA	cue	64.6	49.0	58.4	47.4	73.9	74.5	81.9	80.8	78.8	83.1	83.9	**84.3**
	quotation	0.0	0.0	5.6	6.0	76.5	90.2	89.0	**92.3**	90.3	90.6	90.0	90.2
CoNLL'00	ADJP	29.1	22.8	47.0	61.9	56.6	72.8	66.3	77.2	82.4	**84.2**	83.6	83.5
	ADVP	51.8	58.0	66.9	74.0	70.4	76.0	76.8	81.2	85.3	86.2	**86.4**	86.3
	NP	59.5	64.3	79.7	86.0	88.8	92.8	91.4	94.9	97.0	97.1	97.2	**97.3**
	PP	57.5	56.6	90.4	94.5	94.4	96.5	96.0	97.4	98.5	98.6	**98.6**	98.6
	PRT	40.9	41.0	64.3	63.0	66.4	68.7	73.8	75.1	84.3	83.3	**84.6**	84.6
	SBAR	33.2	63.3	67.1	73.8	81.4	86.1	67.1	65.1	94.2	94.3	94.2	**94.5**
	VP	49.3	56.6	74.8	89.2	84.8	92.7	88.6	94.4	96.6	96.6	96.5	**96.7**
OntoNotes	cardinal	25.8	19.0	57.1	55.3	53.1	60.9	72.8	**81.0**	80.8	80.9	79.9	79.2
	date	38.6	29.0	65.6	69.0	63.1	68.3	79.5	84.3	85.5	**85.9**	85.9	85.7
	event	0.0	0.0	29.4	40.7	0.9	0.0	39.0	46.4	63.2	**65.0**	60.5	64.9
	facility	0.0	0.0	7.1	17.8	0.0	0.0	30.6	45.6	62.0	64.7	64.7	**72.8**
	GPE	60.8	44.5	75.0	76.7	69.5	72.0	85.2	91.6	93.4	94.1	94.0	**94.7**
	language	0.0	0.0	29.0	33.2	0.0	0.0	47.5	40.5	72.0	**76.5**	71.6	70.8
	law	0.0	0.0	19.1	17.9	0.0	0.0	44.7	52.1	61.5	64.8	55.7	**67.2**
	location	11.4	0.0	38.6	42.1	19.4	9.0	54.2	67.1	65.8	67.1	66.8	**70.8**
	money	9.8	32.5	67.9	79.2	64.5	76.1	82.6	**90.1**	87.8	88.7	89.4	88.9
	NORP	66.6	48.0	78.9	80.0	66.4	73.9	81.4	91.0	89.1	89.8	90.3	**91.5**
	ordinal	55.6	0.4	50.9	56.0	33.1	46.8	69.3	**83.3**	79.2	79.8	78.3	76.8
	organization	27.3	19.1	49.1	60.6	46.3	58.6	72.5	84.0	83.8	85.4	85.6	**88.6**
	percent	30.1	18.8	80.0	85.8	73.8	81.3	83.3	**88.6**	88.3	87.6	88.5	88.1
	person	25.9	15.3	52.6	71.8	64.7	70.1	79.4	87.6	92.8	**93.7**	93.3	93.6
	product	0.0	0.0	43.0	35.1	11.4	4.6	47.3	50.8	59.6	64.1	62.4	**64.9**
	quantity	0.0	0.0	38.0	49.8	25.9	0.0	64.5	**68.0**	67.0	66.7	59.2	60.6
	time	2.9	0.0	40.4	33.6	26.7	13.0	49.1	**63.7**	61.5	61.3	60.7	61.8
	work_of_art	0.0	0.0	6.0	7.3	0.5	17.3	29.3	57.2	55.2	59.0	57.0	**62.4**
ChemDNer	abbreviation	50.0	12.0	54.7	54.5	51.6	48.3	62.7	71.3	78.2	**79.1**	78.2	77.1
	family	47.4	3.8	57.9	56.6	53.7	13.8	64.5	68.8	77.6	78.6	78.9	**79.1**
	formula	31.4	10.8	46.3	53.8	48.2	47.4	72.4	76.3	76.9	80.2	81.3	**81.8**
	identifier	0.0	0.0	44.1	37.7	38.1	0.7	69.1	66.2	79.5	**83.1**	82.5	81.8
	multiple	0.0	0.0	5.1	3.0	0.0	0.0	35.5	53.4	57.8	64.6	65.6	**69.0**
	systematic	50.5	25.4	54.6	59.7	60.5	49.7	76.3	79.1	86.2	87.4	**87.9**	87.8
	trivial	61.3	30.2	66.6	65.0	65.0	52.6	75.0	77.9	90.2	91.0	**91.2**	91.1

Table 4.5: F_1 scores for each model architecture on each span type. Each entry is averaged over five runs.

	MAE	r^2
Full model	11.38	0.73
No interactions	14.00	0.61
Only architecture predictors	18.88	0.37
Only task predictors	20.87	0.22
Empty model	23.78	N/A

Table 4.6: Evaluation of performance prediction models.

In this case, we find that our BERT+Feat+LSTM+CRF model beats the existing state of the art on this span type, achieving an F_1 score of 78.1, compared to the score of 75 reported in Scheible et al. (2016).

Meta-learning Results

The result of Step 2 is our performance prediction model. Table 4.6 shows both mean absolute error (MAE), which is directly interpretable as the mean difference between predicted and actual F_1 score for each data point, and r^2, which provides the amount of variance accounted for by the model. The full performance prediction model, including both span type and model architecture features, accounts for 73% of the variance, with an MAE of about 11. We see this as an acceptable model fit. To validate the

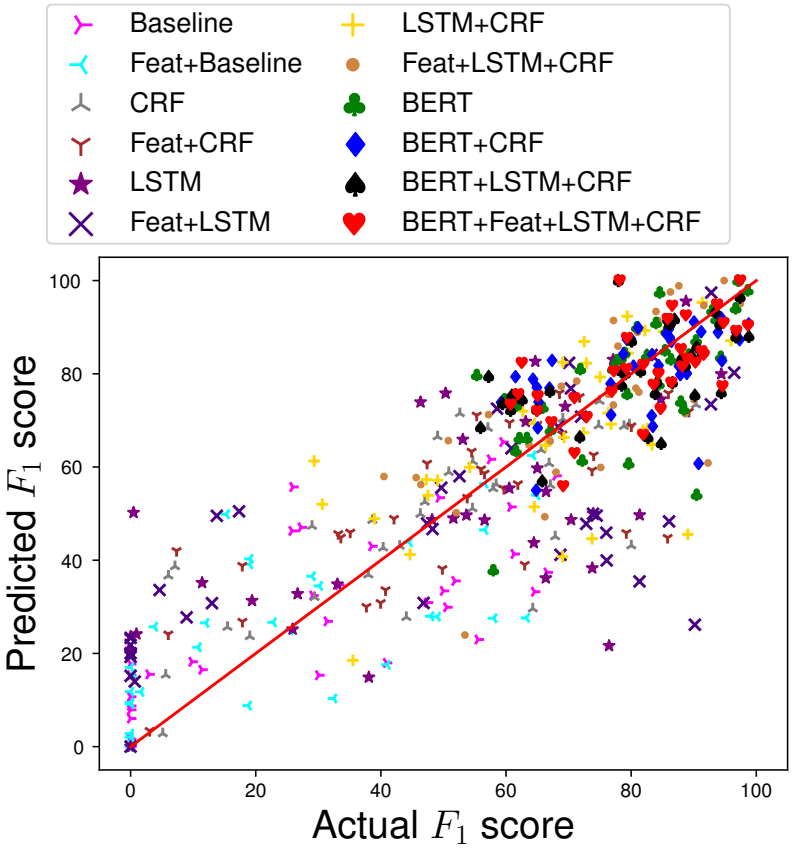


Figure 4.2: Scatter plot of actual vs. predicted F_1 scores for all 36 span types × 12 model architectures

usefulness of the predictor groups and interaction terms, we carry out ablation experiments wherein these are excluded, including a model with no interaction terms, a model with only span type-predictors, a model with only architecture predictors, and an empty model, which only predicts the average of all F_1 scores. The reduced models do better than the empty model,[4] but show marked increases in MAE and corresponding drops in r^2 compared to the full model. While the usefulness of the architecture predictors is expected, this also constitutes strong evidence for the usefulness of the span type predictors we have proposed in Section 4.2.1.

Figure 4.2 shows a scatter plot of predicted and actual F_1 scores. Our meta-learning model generally predicts high performances better than low performances. The largest cluster of errors occurs for experiments with an actual F_1 score of exactly zero, arguably an uninteresting case. Thus, we believe that the overall MAE underestimates rather than overestimates the quality of the performance prediction for practical purposes.

[4]For the empty model, r^2 is undefined because the variance of the predictions is zero.

4.2.7 Analysis

We now investigate the linear regression coefficients of our performance prediction model to assess our hypotheses from Section 4.2.1. To obtain a single model to analyze, we retrain our regression model on all data points, with no cross-validation.

Table 4.7 shows the resulting coefficients. Using Bonferroni correction at $\alpha = 0.05$, we consider a coefficient significant if p<0.002. Non-significant coefficients are shown in parentheses. Due to the scaling of F_1 scores performed as described in section 4.2.4, the coefficients cannot be directly interpreted in terms of linear change on the F_1 scale. However, as we standardized all predictors, we can compare coefficients with one another. Coefficients with a greater magnitude have larger effects on F_1 score, with positive values indicating an increase, and negative values a decrease.

When analyzing these coefficients, one must consider main effects and interactions together. E.g., the main effect coefficient for LSTMs is negative, which seems to imply that adding an LSTM will hurt performance. However, the LSTM × [freq] and LSTM × [boundary distinctness] interactions are both strongly positive, so LSTMs

should help on frequent span types with high boundary distinctiveness. Our main observations are the following:

Frequency helps, length hurts. The main effects of our span type predictors show mostly an expected pattern. Frequency has a strong positive effect (frequent span types are easier to learn), while length has an even stronger negative effect (long span types are difficult). More distinct boundaries help performance as well. More surprising is the negative sign of the span distinctiveness predictor, which would mean that more distinct spans are more difficult to recognize. However, this might be due to the negative correlation between span distinctiveness and frequency ($r = -0.46$ in standardized predictors): Less frequent spans are, by virtue of their rarity, more distinctive.

BERT is good for performance, especially with few examples. The presence of a BERT component is the highest-impact positive predictor for model performance, with a positive coefficient of 1. This finding is not entirely surprising, given the recent popularity of BERT-based

Model predictors		
Handcrafted		(−0.11)
CRF		0.50
LSTM		−0.35
BERT		1.00
Span type predictors		
freq		0.40
length		−0.49
span distinct.		−0.22
boundary distinct.		0.16
Model–span type interactions		
Handcrafted ×	freq	(0.05)
	length	(−0.04)
	span distinct.	(−0.09)
	boundary distinct.	(0.09)
CRF ×	freq	−0.33
	length	0.19
	span distinct.	0.34
	boundary distinct.	−0.30
LSTM ×	freq	0.47
	length	0.08
	span distinct.	(−0.09)
	boundary distinct.	0.22
BERT ×	freq	−0.43
	length	0.13
	span distinct.	(0.04)
	boundary distinct.	(−0.05)
Model–model interactions		
Handcrafted ×	CRF	0.10
	LSTM	0.05
	BERT	−0.05
CRF ×	LSTM	(−0.05)
	BERT	−0.24
LSTM ×	BERT	−0.17

Table 4.7: Regression coefficients from performance prediction model. Coefficients not statistically significant at $p < 0.002$ (as per Bonferroni correction) in parentheses.

models for span extraction problems Li et al. (2020); Hu et al. (2019). Furthermore, the strong negative value of the (BERT × [freq]) predictor shows that BERT's benefits are strongest when there are few training examples, validating our hypothesis about transfer learning. BERT is also robust: largely independent of span or boundary distinctiveness effects.

LSTMs require a lot of data. While the main effect of LSTMs is negative, this effect is again modulated by the high positive coefficient of the (LSTM × [freq]) interaction. This means that their performance is highly dependent on the amount of training data. Also, LSTMs lead to improvements for long span types and those with distinct boundaries – properties that LSTMs arguably can pick up well but that other models struggle with.

CRFs help. After BERT, the presence of a CRF shows the second-most positive main effect on model performance. Given the strong correlation between adjacent tags in a BIO label sequence, it makes sense that a model capable of enforcing correlations in its output sequence would perform well. CRFs can also exploit span dis-

tinctiveness well, presumably by the same mechanism. Surprisingly, CRFs show reduced effectiveness for highly frequent spans with distinct boundaries. We believe that this pattern is best considered as a relative statement: for frequent, well-separated span types CRFs gain less than other model types.

Handcrafted features are complicated. Looking purely at our regression coefficients, handcrafted features don't seem to matter much: we find neither a significant main effect of handcrafted features, nor any significant interactions with span type predictors. Interactions with model predictors are significant, but rather small. However, it is easy to find specific cases where handcrafted features seem to help quite a bit: For instance, the Feat+LSTM+CRF architecture outperforms the LSTM+CRF architecture on all but four span types. We suspect that, while handcrafted features can be important, their effect on F_1-score may be subject to a number of nonlinear interactions which our meta-learning model is unable to capture.

Combining model components shows diminishing returns. All interactions between LSTM, CRF, and BERT

are negative. This demonstrates an overlap in these components' utility. In fact, a simple "maximal" combination does not always perform best, as Table 4.5 confirms.

4.2.8 Related Work

Meta-learning and performance prediction are umbrella terms which comprise a variety of approaches and formalisms in the literature. We focus on the literature most relevant to our work and discuss the relationship.

Performance Prediction for Trained Models. In NLP, a number of studies investigate predicting the performance of models that have been trained previously on novel input. An example is Chen (2009) which develops a general method to predict the performance of a family of language models. Similar ideas have been applied more recently to machine translation (Bojar et al., 2017), and automatic speech recognition Elloumi et al. (2018), among others. While these approaches share our goal of performance prediction, they predict performance for the same task and model on new data, while we generalize across tasks and architectures. Thus, these approaches

are better suited to estimating confidence at prediction time, while our meta-learning approach can predict a model's performance before it is trained.

AutoML. Automated machine learning, or AutoML, aims at automating various aspects of machine learning model creation, including hyperparameter selection, architecture search, and feature engineering (Yao et al., 2018; He et al., 2021) While the task of performance prediction does not directly fall within this research area, a model for predicting performance is directly applicable to architecture search. Within AutoML, the auto-sklearn system (Feurer et al., 2015) takes an approach rather similar to ours, wherein they identify meta-features of datasets, and select appropriate model architectures based on those meta-features. However, auto-sklearn does not predict absolute performance as we do, but instead simply selects good candidate architectures via a k-nearest-neighbors approach in meta-feature space. Other related approaches in AutoML use Bayesian optimization, including the combined model selection and hyperparameter optimization of Auto-WEKA (Thornton et al., 2013) and the neural architecture search of Auto-keras (Jin et al., 2019).

Model Interpretability. A number of works have investigated how to analyze and explain the decisions made by machine learning models. LIME (Ribeiro et al., 2016) and Anchors (Ribeiro et al., 2018) are examples of systems for explaining a model's decisions for specific training instances. Other works seek to explain and summarize how models perform across an entire dataset. This can be achieved e.g. through comparison of architecture performances, as in Nguyen and Guo (2007), or through meta-modeling of trained models, as was done in Weiss et al. (2018). Our present work falls into this category, including both a comparison of architectures across datasets and a meta-learning task of model performance.

Meta-learning for One- and Few-shot Learning. A recent trend is the application of meta-learning to models for one- or few-shot learning. In this setting, a meta-learning approach is used to train models on many distinct tasks, such that they can subsequently be rapidly fine-tuned to a particular task Finn et al. (2017); Santoro et al. (2016). While such approaches use the same meta-learning framework as we do, their task and methodology are substantially different. They focus on learning with

very few training examples, while we focus on optimizing performance with normally sized corpora. Additionally, these models selectively train preselected model architectures, while we are concerned with comparisons between architectures.

Model and Corpus Comparisons in Survey Papers. In a broad sense, our goal of comparison between existing corpora and modeling approaches is shared with many existing survey papers. Surveys include quantitative comparisons of existing systems' performances on common tasks, producing a results matrix very similar to ours (Li et al., 2020; Yadav and Bethard, 2018; Bostan and Klinger, 2018, i.a.). However, most of these surveys limit themselves to collecting results across models and datasets without performing a detailed quantitative analysis of these results to identify recurring patterns, as we do with our performance prediction approach.

4.2.9 Conclusion

Here we systematically investigated task generality for span extraction tasks. By collecting a large set of span ex-

traction tasks and a large set of compatible architectures, and by training and evaluating on the full Cartesian product of these sets, we were able to obtain large amounts of data on architecture performance for specific tasks. We then used meta-learning to analyze these data, and were able to glean specific, interpretable insights into when we might expect an architecture to generalize well to a new task, and when we might not. In the process of doing so, we identified four easily-quantified properties of span extraction tasks which seem to be highly informative for determining when a particular architecture might be applicable.

Given the success of this approach to span extraction, it seems desirable to apply similar techniques to full relation extraction. While this isn't a bad idea, there are some caveats that must be considered. For span extraction, we were able to start with minimal limiting assumptions (no overlapping spans, and no arbitrary dependencies between spans), and then collect a diverse variety of tasks and architectures compatible with these assumptions. The same is not possible for relation extraction – for example, any reasonable architecture for TACRED (Zhang et al., 2017) should assume that only one relation can

occur in each input sentence, and that it must relate the two entities provided, but an architecture making these assumptions would be inapplicable to relation extraction tasks like GENIA event extraction (Kim et al., 2013), where the total number of relations is unknown, and a large portion of the task's difficulty involves determining how exactly to group the identified entities together into cohesive relations.

Chapter 5

Task-general joint modeling with regular-constrained CRFs

As we discussed in Chapter 3, one promising direction towards task generality for relation extraction lies in formal specification of tasks and their constraints. However, this approach depends upon the availability of architectures which can accept and utilize task specifications as hyperparameters. It thus becomes desirable to search for architectures capable of accommodating a wide range of possible task structures. This chapter presents one specific architecture, based upon linear-chain conditional random fields, which is capable (with caveats)

of capturing any task structure specifiable as a regular language.

5.1 Task properties as sequence labeling constraints

Suppose we would like to extract a single relation from a text. While we have explored pipeline-based approaches wherein we first identify entity spans, and then predict relations between those entities we found, we could also try to do everything "all in one go" – we ask a span extraction model to produce a set of labeled spans, and take the whole output to be a single relation. Individual span labels can specify both the entity type and the role each entity span plays in the relation. Of course, to extract multiple relations, we can simply ask the model to produce output multiple times, as long as we somehow notate in the input which relation we are interested in for each prediction.

One problem with this approach is that, with a standard span extraction model, we have no way of ensuring that the model predicts the right number of entities

with the correct combination of types for the relation. While span extraction models don't usually pick spans entirely independently from one another, the dependencies are almost always local effects, such as BIO-based approaches requiring that no spans overlap, or CRF-based approaches modeling agreement between directly adjacent tokens. On the other hand, constraints like type agreement for relation extraction are global properties that entire relations must satisfy, properties which cannot be learned as local dependencies in standard span extraction models.

This chapter will formalize relation extraction in terms of sequence labeling with global constraints. We introduce a novel method for enforcing regular-language constraints on CRF sequence labeling models – the resulting regular-constrained CRF (RegCCRF) functions identically to a standard model, except that it is guaranteed to only produce outputs in an a-priori-specified regular language $\mathcal{L}$. In conjunction with BIO-labeling, this provides us exactly the machinery we need for relation extraction, allowing us to extract sets of spans subject to user-specified global constraints.

We investigate the resulting RegCCRF architecture from

both a theoretical and empirical perspective. Theoretically, we will investigate some mathematical properties of the distributions a RegCCRF can learn, and will use synthetic data to contrast models' performance under different training and inference settings. Empirically, we apply our a RegCCRF model to Semantic Role Labeling, a relation extraction task with variadic relations and a highly nontrivial set of constraints. We show modest yet significant improvements on this task when using a RegCCRF model, validating our theoretical arguments and attesting to the practical utility of the architecture.

5.2 Preliminaries and notation

As our construction involves finite-state automata and conditional random fields, we define these here and specify the notation we will use in the remainder of this section.

Finite-state automata. All automata are taken to be nondeterministic finite-state automata (NFAs) without epsilon transitions. Let such an NFA be defined as a 5-tuple (Σ, Q, q^1, F, E), where

- $\Sigma = \{a^1, a^2, ..., a^{|\Sigma|}\}$ is a finite alphabet of symbols,
- $Q = \{q^1, q^2, ..., q^{|Q|}\}$ is a finite set of states,
- $q^1 \in Q$ is the sole starting state,
- $F \subseteq Q$ is a set of accepting states,
- and $E \subseteq Q \times \Sigma \times Q$ is a set of directed, symbol-labeled edges between states. The edges define the NFA's transition function $\Delta : Q \times \Sigma \rightarrow 2^Q$, with $r \in \Delta(q, a) \leftrightarrow (q, a, r) \in E$.

An automaton is said to accept a string $\boldsymbol{s} \in \Sigma^*$ iff there exists a contiguous path of edges from q^1 to some accepting state whose edge labels are exactly the symbols of $\boldsymbol{s}$. The *language* defined by an automaton is the set of all such accepted strings. A language is *regular* if and only if it is the language of some NFA.

Linear-chain conditional random fields. Linear-chain conditional random fields (CRFs) Lafferty et al. (2001) are an architecture for sequence labeling models. Parameterized by $\boldsymbol{\theta}$, CRFs represent their model distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$, a distribution over label sequences $\boldsymbol{y} =$

$\langle y^1, \cdots, y^t \rangle$ conditioned on input sequences $\boldsymbol{x} = \langle x^1, \cdots, x^t \rangle$, as

$$\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \propto \exp \sum_j f^j_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{y}) \,,$$

with individual observations x^i coming from some observation space X, and outputs y^i coming from some finite alphabet Y. In this work, we use CRFs for sequence labeling problems, but the dataset labels do not correspond directly to the CRF's outputs y^i. In order to avoid ambiguity, and since the term "state" already has a meaning for NFAs, we call $\boldsymbol{y}$ the CRF's *tag sequence*, and each y^i a *tag*. The terms *label sequence* and *label* will thus unambiguously refer to the original dataset labels.

Each $f^j_{\boldsymbol{\theta}}$ is a potential function of $\boldsymbol{x}$ and $\boldsymbol{y}$, parameterized by $\boldsymbol{\theta}$. Importantly, in a linear-chain CRF, these potential functions are limited to two kinds: The *transition function* $g_{\boldsymbol{\theta}}(y^i, y^{i+1})$ assigns a potential to each pair (y^i, y^{i+1}) of adjacent tags in $\boldsymbol{y}$, and the *emission function* $h_{\boldsymbol{\theta}}(y^i, \boldsymbol{x}, i)$ assigns a potential to each possible output tag y^i given the observation sequence $\boldsymbol{x}$ and its position i. With these,

the distribution defined by a CRF is

$$\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \propto \exp\left(\sum_{i=1}^{t-1} g_{\boldsymbol{\theta}}(y^i, y^{i+1}) + \sum_{i=1}^{t} h_{\boldsymbol{\theta}}(\boldsymbol{x}, y^i, i)\right). \tag{5.1}$$

Limiting our potential functions in this way imposes a Markov assumption on our model, as potential functions can only depend on a single tag or a single pair of adjacent tags. This makes learning and inference tractable: the forward algorithm (Jurafsky and Martin, 2009) can calculate negative log-likelihood (NLL) loss during training, and the Viterbi algorithm (Viterbi, 1967; Jurafsky and Martin, 2009) can be used for inference. Both are linear in t, and quadratic in $|Y|$ in both time and space.

In practice, the transition function $g_{\boldsymbol{\theta}}$ is usually represented explicitly as a $|Y| \times |Y|$ parameter matrix, and the emission function $h_{\boldsymbol{\theta}}$ can be an arbitrary learnable function – increasingly commonly represented as a deep neural network.

5.3 Related work

We identify three modeling approaches in the field of structured prediction that are relevant to the current work: constrained decoding, which can enforce output constraints at decoding time, techniques for weakening the Markov assumption of CRFs to learn long-distance dependencies, and weight-learning in finite-state transducers.

Constrained decoding. A common approach to enforcing constraints in model output is *constrained decoding*: Models are trained in a standard fashion, and decoding ensures that the model output satisfies the constraints. This almost always corresponds to finding or approximating a version of the model's distribution conditionalized on the output obeying the specified constraints. This approach is useful if constraints are not available at training time, such as in the interactive information extraction task of Kristjansson et al. (2004). They present *constrained conditional random fields*, which can enforce that particular tokens are or are not assigned particular labels (positive and negative constraints, respectively). Formally,

our work is a strict generalization of this approach, as position-wise constraints can be formulated as a regular language, but regular languages go beyond position-wise constraints. Other studies treat decoding with constraints as a search problem, searching for the most probable decoding path which satisfies all constraints. For example, He et al. (2017b) train a neural network to predict token-wise output probabilities for semantic role labeling following the BIO label-alphabet (Ramshaw and Marcus, 1999), and then use exact A* search to ensure that the output forms a valid BIO sequence and that particular task-specific constraints are satisfied. For autoregressive models, constrained beam search (Hokamp and Liu, 2017; Anderson et al., 2017; Hasler et al., 2018) can enforce regular-language constraints during search. We further discuss constrained decoding as it relates to RegCCRFs in Section 5.5.

Markov relaxations. While our approach can relax the Markov assumption of CRFs through nonlocal hard constraints, another strand of work has developed models which can directly *learn* nonlocal dependencies in CRFs: *Semi-Markov CRFs* (Sarawagi and Cohen, 2004) relax the

Markov property to the semi-Markov property. In this setting, CRFs are tasked with segmentation, where individual segments may depend only on their immediate neighbors, but model behavior within a particular segment need not be Markovian. As such, semi-Markov CRFs are capable of capturing nonlocal dependencies between output variables, but only to a range of one segment and inside of a segment. *Skip-chain CRFs* (Sutton and McCallum, 2004) change the expressiveness of CRFs by relaxing the assumption that only the linear structure of the input matters: they add explicit dependencies between distant nodes in an otherwise linear-chain CRF. These dependencies are picked based on particular properties, e.g., input variables of the same value or which share other properties. In doing so, they add loops to the model's factor graph, which makes exact training and inference intractable, and leads to the use of approximation techniques such as loopy belief propagation and Gibbs sampling.

Weight learning for finite-state transducers. While our approach focuses on the task of constraining the CRF distribution to a known regular language, a related task is

that of learning a weighted regular language from data. This task is usually formalized as learning the weights of a weighted finite-state transducer (FST), as in e.g. Eisner (2002) with directly parameterized weights and Rastogi et al. (2016) with weights parameterized by a neural network. Despite the difference in task-setting, this task is quite similar to ours in the formal sense, and in fact our proposal can be viewed as a particularly well-behaved special case of FST weight learning for an appropriately chosen transducer architecture and parameterization. We discuss this connection further in Section 5.4.3.

5.4 Regular-constrained CRFs

Here we will present our approach to enforcing regular-language constrains on a CRF. Given a regular language $\mathcal{L}$, we would like to constrain a CRF to $\mathcal{L}$. We formalize this notion of constraint with conditional probabilities – a CRF constrained to $\mathcal{L}$ is described by a (further) conditionalized version of that CRF's distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$, conditioned on the event that the tag sequence $\boldsymbol{y}$ is in $\mathcal{L}$ – that is, the distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} = \boldsymbol{y} \mid \mathbf{x} = \boldsymbol{x}; \mathbf{y} \in \mathcal{L})$. We will refer to this distribution as $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ for short,

noting that

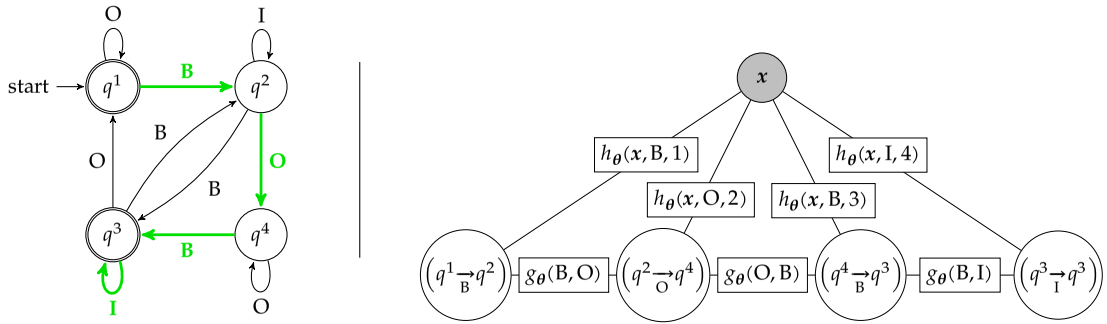
$$\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L}) = \begin{cases} \alpha \cdot \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) & \text{if } \boldsymbol{y} \in \mathcal{L} \\ 0 & \text{otherwise}, \end{cases} \tag{5.2}$$

with $\alpha \geq 1$ defined as $\alpha^{-1} = \sum_{\boldsymbol{y} \in \mathcal{L}} \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$.

In order to utilize this distribution for machine learning, we need to be able to compute NLL losses and perform MAP inference. As discussed in Section 5.2, both of these are efficiently computable for CRFs. Thus, if we can construct a separate CRF whose output distribution can be interpreted as $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$, both of these operations will be available. We do this in the next section.

5.4.1 Construction

Let $M = (\Sigma, Q, q^1, F, E)$ be an NFA that describes $\mathcal{L}$. We assume that M is *unambiguous* – i.e., every string in $\mathcal{L}$ is accepted by exactly one path through M. As every NFA can be transformed into an equivalent unambiguous NFA (Mohri, 2012), this assumption involves no loss of generality. Our plan is to represent $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ by constructing a separate CRF with a distinct tag set, whose



$$Y' = \left\{ \left(q^1 \underset{\mathrm{O}}{\rightarrow} q^1\right), \left(q^1 \underset{\mathrm{B}}{\rightarrow} q^2\right), \left(q^2 \underset{\mathrm{I}}{\rightarrow} q^2\right), \left(q^2 \underset{\mathrm{B}}{\rightarrow} q^3\right), \left(q^2 \underset{\mathrm{O}}{\rightarrow} q^4\right), \left(q^3 \underset{\mathrm{O}}{\rightarrow} q^1\right), \left(q^3 \underset{\mathrm{B}}{\rightarrow} q^2\right), \left(q^3 \underset{\mathrm{I}}{\rightarrow} q^3\right), \left(q^4 \underset{\mathrm{B}}{\rightarrow} q^3\right), \left(q^4 \underset{\mathrm{O}}{\rightarrow} q^4\right) \right\}$$

Figure 5.1: Example for a RegCCRF, showing NFA and unrolled factor graph. $\mathcal{L}$ describes the language $(\mathrm{O} \mid \mathrm{BI}^*\mathrm{O}^*\mathrm{BI}^*)^*$, the language of valid BIO sequences for an even number of spans. We would like to calculate $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ for $\boldsymbol{y} = \langle \mathrm{B}, \mathrm{O}, \mathrm{B}, \mathrm{I} \rangle$. We show an unambiguous automaton M for $\mathcal{L}$ (left), and a factor graph (right) for the auxiliary CRF computing $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y}' \mid \boldsymbol{x})$, where $y' \in Y'^*$ corresponds to the sole accepting path of $\boldsymbol{y}$ through M (marked).

output sequences can be interpreted directly as paths through M. As M is unambiguous, each label sequence in $\mathcal{L}$ corresponds to exactly one such path. We parameterize this auxiliary CRF identically to our original CRF – that is, with label-wise (not tag-wise) transition and emission functions. Thus, for all parameterizations $\boldsymbol{\theta}$, both distributions $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$ and $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ are well defined.

There are many ways to construct such a CRF. As CRF training and inference are quadratic in the size of the tag set Y, we would prefer a construction which minimizes $|Y|$. However, for clarity, we will first present a conceptually simple construction, and discuss approaches to reduce $|Y|$ in Section 5.4.2. We start with our original CRF, parameterized by $\boldsymbol{\theta}$, with tag set $Y = \Sigma$, transition function $g_{\boldsymbol{\theta}}$, and emission function $h_{\boldsymbol{\theta}}$, describing the probability distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x})$, where $\mathbf{y}$ varies over sequences in Σ^*. From this, we construct a new CRF, also parameterized by the same $\boldsymbol{\theta}$, but with tag set Y', transition function $g'_{\boldsymbol{\theta}}$, and emission function $h'_{\boldsymbol{\theta}}$. This auxiliary CRF describes the distribution $\hat{P}'_{\boldsymbol{\theta}}(\mathbf{y}' \mid \mathbf{x})$ (with $\mathbf{y}'$ varying over Y'^*), which we will be able to interpret as $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$. The

construction is as follows:

$$Y' = E \tag{5.3}$$

$$g'_{\boldsymbol{\theta}}\left((q,a,r),(q',a',r')\right) = \begin{cases} g_{\boldsymbol{\theta}}(a,a') & \text{if } r = q' \\ -\infty & \text{otherwise} \end{cases} \tag{5.4}$$

$$h'_{\boldsymbol{\theta}}\left(\boldsymbol{x},(q,a,r),i\right) = \begin{cases} -\infty & \text{if } i = 1, q \neq q^1 \\ -\infty & \text{if } i = t, r \notin F \\ h_{\theta}(\boldsymbol{x},a,i) & \text{otherwise} \end{cases} \tag{5.5}$$

This means that the tags of our new CRF are the edges of M, the transition function assigns zero probability to transitions between edges which do not pass through a shared NFA state, and the emission function assigns zero probability to tag sequences which do not begin at the starting state or end at an accepting state. Apart from these constraints, the transition and emission functions depend only on edge labels, and not on the edges themselves, and agree with the standard CRF's $g_{\boldsymbol{\theta}}$ and $h_{\boldsymbol{\theta}}$ when no constraints are violated.

As M is unambiguous, every tag sequence $\boldsymbol{y}$ corresponds to a single path through M, representable as an edge sequence $\boldsymbol{\pi} = \langle \pi^1, \pi^2, ..., \pi^t \rangle, \pi^i \in E$. Since this

path is a tag sequence for our auxiliary CRF, we can directly calculate the auxiliary CRF's $\hat{P}'_{\boldsymbol{\theta}}(\boldsymbol{\pi} \mid \boldsymbol{x})$. From the construction of $g'_{\boldsymbol{\theta}}$ and $h'_{\boldsymbol{\theta}}$, this must be equal to $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$, as it is proportional to $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$ for $\boldsymbol{y} \in \mathcal{L}$, and zero (or, more correctly, undefined) otherwise. Figure 5.1 illustrates this construction with a concrete example.

5.4.2 Time and space efficiency

As the Viterbi and forward algorithms are quadratic in $|Y|$, very large tag sets can lead to performance issues, possibly making training or inference intractable in extreme cases. Thus, we would like to characterize under which conditions a RegCCRF can be used tractably, and identify techniques for improving performance. As Y corresponds to the edges of M, we would like to select our unambiguous automaton M to have as few edges as possible. For arbitrary languages, this problem is NP-complete (Jiang and Ravikumar, 1991), and, assuming P $\neq$ NP, is not even efficiently approximable (Gruber and Holzer, 2007). Nonetheless, for many common classes of languages, there exist approaches to obtain a tractably small automaton.

One straightforward method is to construct M directly from a short unambiguous regular expression. Brüggemann-Klein and Wood (1992) present a simple algorithm for constructing an unambiguous automaton from an unambiguous regular expression, with $|Q|$ linear in the length of the expression. Using this method to construct M, the time- and space-complexity of Viterbi are polynomial in the length of our regular expression, with a worst-case of quartic complexity when the connectivity graph of M is dense.

For many other tasks, a reasonable approach is to leverage a priori knowledge about the constraints to manually construct a small unambiguous automaton. For example, if the constraints require that a particular label occurs exactly n times in the output sequence, an automaton could be constructed manually to count occurrences of that label. Multiple constraints of this type can then be composed via automaton union and intersection.

Without making changes to M, we can also reduce the size of $|Y|$ by adjusting our construction. Instead of making each edge of M a tag, we can adopt equivalence classes of edges. Reminiscent of standard NFA minimization, we define $(q, a, r) \sim (q', a', r') \leftrightarrow (q, a) = (q', a')$. When

constructing our CRF, whenever a transition would have been allowed between two edges, we allow a transition between their corresponding equivalence classes. We do the same to determine which classes are allowed as initial or final tags. As each equivalence class corresponds (non-uniquely) to a single symbol a, we can translate between tag sequences and strings of $\mathcal{L}$ just as before.

Finally, when other approaches for reducing the size of M are not enough, it is always possible to simplify the constraint language $\mathcal{L}$ to a language which only approximates the desired constraints, but which is expressible with a smaller automaton. Of note, this doesn't necessarily involve making the constraints more permissive – stricter constraints might lead to a smaller automaton, at the expense of excluding some valid predictions. In practice, such simplifications should be carried out with a good understanding of the dataset and domain, such that the simplified language will only lead to degraded performance on a small minority of predictions. In Section 5.7.2, we will describe the use of these sorts of simplifications in practice.

5.4.3 Interpretation as a weighted finite-state transducer

While we present our model as a variation of a standard CRF which enforces regular-language constraints, an alternate characterization is as a weighted finite-state transducer with the transducer topology and weight parameterization chosen so as to yield the distribution $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$. In order to accommodate CRF transition weights, such an approach involves weight-learning in an auxiliary automaton whose edges correspond to edge-pairs in M – we give a full construction in Appendix B.2.

This interpretation enables direct comparison to studies on weight learning in finite-state transducers, such as Rastogi et al. (2016). While RegCCRFs can be viewed as special cases of neural-weighted FSTs, they inherit a number of useful properties from CRFs not possessed by neural-weighted automata in general. Firstly, as $|\boldsymbol{y}|$ is necessarily equal to $|\boldsymbol{x}|$, the partition function $\sum_{y \in \mathcal{L}} \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$ is guaranteed to be finite, and $\hat{P}_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ is a well-defined probability distribution for all $\boldsymbol{\theta}$, which is not true for weighted transducers in general, which may admit paths with unbounded lengths and weights. Secondly,

as M is assumed to be unambiguous, string probabilities correspond exactly to path probabilities, allowing for exact MAP inference with the Viterbi algorithm. In contrast, finding the most probable string in the highly ambiguous automata used when learning edge weights for an unknown language is NP-Hard (Casacuberta and de la Higuera, 1999), necessitating approximation methods such as crunching (May and Knight, 2006). Finally, as each RegCCRF can be expressed as a CRF with a particular parameterization, the convexity guarantees of standard CRFs carry over, in that the loss is convex with respect to emission and transition scores. In contrast, training losses in general weighted finite-state transducers are usually nonconvex (Rastogi et al., 2016).

5.5 Comparing constrained training to constrained decoding

Our construction suggests two possible use cases for a RegCCRF: *constrained decoding*, where a CRF is trained without constraints, and the learned weights are then used in a RegCCRF at decoding time, and *constrained*

training, where a RegCCRF is both trained and decoded with constraints. In this section, we compare between these two approaches and a standard, *unconstrained CRF*. We assume we have a data distribution $P(\boldsymbol{x}, \boldsymbol{y})$ over $(\boldsymbol{x}, \boldsymbol{y})$ pairs with each $\boldsymbol{x} \in X^*$, and each $\boldsymbol{y}$ of matching length in some regular language $\mathcal{L} \subseteq \Sigma^*$. We wish to model our target distribution $P(\mathbf{y} \mid \mathbf{x})$ with either a CRF or a RegCCRF, by way of maximizing the model's (log) likelihood given the data distribution. This section will investigate the best-case performance of CRFs and RegCCRFs, and will assume perfect optimization. Latter sections will compare the two architectures empirically using more realistic optimization settings.

The unconstrained CRF corresponds to a CRF that has been trained, without constraints, on samples from P, and is used directly for inference: It makes no use of the language $\mathcal{L}$. The model's output distribution is $\hat{P}_{\boldsymbol{\theta}^*_{\mathrm{u}}}(\mathbf{y} \mid \mathbf{x})$, with parameter vector $\boldsymbol{\theta}^*_{\mathrm{u}}$ minimizing the NLL objective:

$$\boldsymbol{\theta}^*_{\mathrm{u}} = \arg\min_{\boldsymbol{\theta}} E_{\boldsymbol{x},\boldsymbol{y} \sim P} \left[-\ln \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \right] \tag{5.6}$$

In constrained decoding, a CRF is trained unconstrained, but its weights are used in a RegCCRF at decoding time.

The output distribution of such a model is $\hat{P}_{\boldsymbol{\theta}^*_{\mathrm{u}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$. It is parameterized by the same parameter vector $\boldsymbol{\theta}^*_{\mathrm{u}}$ as the unconstrained CRF, as the training procedure is identical, but the output distribution is conditioned on the event $\boldsymbol{y} \in \mathcal{L}$.

Constrained training involves directly optimizing a RegCCRF's output distribution, avoiding any asymmetry between training and decoding time. In this case, the output distribution of the model is $\hat{P}_{\boldsymbol{\theta}^*_{\mathrm{c}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$, where

$$\boldsymbol{\theta}^*_{\mathrm{c}} = \arg\min_{\boldsymbol{\theta}} E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})\right] \tag{5.7}$$

is the parameter vector which minimizes the NLL of the RegCCRF's constrained distribution.

We will demonstrate that these three approaches form a hierarchy in terms of their ability to match the data distribution:

$$L_{\text{unconstrained}} \geq L_{\text{constrained decoding}} \geq L_{\text{constrained training}}, \tag{5.8}$$

with each L corresponding to the negative log-likelihood assigned by each model to the data. This suggests that, all other factors notwithstanding, we should prefer the

constrained training regimen to best model the target distribution. Here we will prove this inequality.

Theorem 1. *For arbitrary* $\boldsymbol{\theta}$*:*
$E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})\right] \geq E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})\right]$

Here we compare the distributions $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$ and $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$. We wish to demonstrate that $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$ can never achieve lower NLL than $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$, and that the two distributions achieve identical NLL only when $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) = \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$ i.e. when constraints have no effect. Of note, this proof is valid for *all* parameterizations $\boldsymbol{\theta}$, and not just for $\boldsymbol{\theta}^*_{\mathrm{u}}$.

Proof. Since every $\boldsymbol{y}$ in P is in $\mathcal{L}$,

$$\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L}) = \alpha \cdot \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}), \tag{5.9}$$

with $\alpha \geq 1$. Thus, the NLL of the regular-constrained CRF is

$$E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})\right] = E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})\right] - \ln \alpha. \tag{5.10}$$

This differs from the NLL of the unconstrained CRF only by the term $-\ln \alpha$. As $\alpha \geq 1$, the regular-constrained

CRF's NLL is less than or equal to that of the unconstrained CRF, with equality only when $\alpha = 1$ and therefore $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) = \hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$. □

Theorem 2.
$E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})\right] \geq E_{\boldsymbol{x},\boldsymbol{y}\sim P}\left[-\ln \hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})\right]$

In this case, we compare the distributions $\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$ and $\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$. We will demonstrate that the former cannot achieve a lower NLL against the data distribution than the latter.

Proof. This follows directly from our definitions, as we define $\boldsymbol{\theta}^*_{\text{c}}$ to minimize the NLL of $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$ against the data distribution. Thus, $\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$ could never yield a lower NLL than $\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$, as that would contradict our definition of $\boldsymbol{\theta}^*_{\text{c}}$. □

5.6 Synthetic data experiments

While constrained training cannot underperform constrained decoding, the conditions where it is strictly better depend on exactly how the transition and emission functions are parameterized, and are not easily stated

in general terms. We now empirically show two simple experiments on synthetic data where the two are not equivalent.

The procedure is similar for both experiments. We specify a regular language $\mathcal{L}$, an observation alphabet X, and a data distribution $P(\boldsymbol{x}, \boldsymbol{y})$ over observation sequences in X^* and label sequences in $\mathcal{L}$. We then train two models, one with a RegCCRF, parameterized by $\boldsymbol{\theta}_{\text{c}}^*$, and one with an unconstrained CRF, parameterized by $\boldsymbol{\theta}_{\text{u}}^*$. For each model, we initialize parameters randomly, then use stochastic gradient descent to minimize the NLL objective. We directly generate samples from P to use as training data. After optimizing $\boldsymbol{\theta}_{\text{c}}^*$ and $\boldsymbol{\theta}_{\text{u}}^*$, we construct a RegCCRF with $\boldsymbol{\theta}_{\text{u}}^*$ for use as a constrained-decoding model, and we compare the constrained-training distribution $\hat{P}_{\boldsymbol{\theta}_{\text{c}}^*}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ with the constrained-decoding distribution $\hat{P}_{\boldsymbol{\theta}_{\text{u}}^*}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$.

We use a simple architecture for our models, with both the transition functions $g_{\boldsymbol{\theta}}$ and emission functions $h_{\boldsymbol{\theta}}$ represented as parameter matrices. We list training hyperparameters in Appendix B.1.

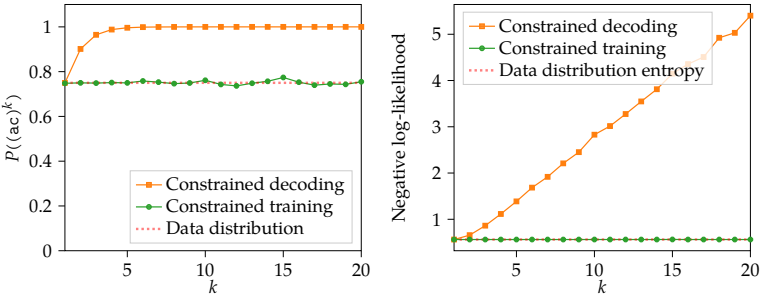


Figure 5.2: Model output probabilities, and NLL losses, plotted against sequence length k. As k increases, constrained decoding becomes a progressively worse approximation for the data distribution, while constrained training is consistently able to match the data distribution.

5.6.1 Arbitrarily large differences in likelihood

We would like to demonstrate that, when comparing constrained training to constrained decoding in terms of likelihood, constrained training can outperform constrained decoding by an arbitrary margin. We choose $\mathcal{L} = (\texttt{ac})^* \mid (\texttt{bc})^*$ to make conditional independence particularly relevant – as every even-indexed label is `c`, an unconstrained CRF must model odd-indexed labels independently, while a constrained CRF can use its constraints

to account for nonlocal dependencies. For simplicity, we hold the input sequence constant, with $X = \{\texttt{o}\}$.

Our approach is to construct sequences of various lengths. For $k \in \mathbb{N}$, we let our data distribution be

$$\begin{aligned} P(\texttt{o}^{2k}, (\texttt{ac})^k) &= \frac{3}{4}; \\ P(\texttt{o}^{2k}, (\texttt{bc})^k) &= \frac{1}{4}. \end{aligned} \tag{5.11}$$

As the marginal distributions for odd-indexed characters are not independent, an unconstrained CRF cannot exactly represent the target distribution $P(\mathbf{y} \mid \mathbf{x})$. We train and evaluate individual models for each sequence length k. Figure 5.2 plots model probabilities and NLL losses for various k. We see that, regardless of k, $\hat{P}_{\theta_c}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ is able to match $P(\mathbf{y} \mid \mathbf{x})$ almost exactly, with only small deviations due to sampling noise in SGD. On the other hand, as sequence length increases, $\hat{P}_{\theta_u}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ becomes progressively "lop-sided", assigning almost all of its probability mass to the string $(\texttt{ac})^k$. This behavior is reflected in the models' likelihoods – constrained training stays at near-constant likelihood for all k, while the negative log-likelihood of constrained decoding grows linearly with k.

5.6.2 Differences in MAP inference

We show here that constrained training and constrained decoding can disagree about which label sequence they deem most likely. Furthermore, in this case, MAP inference agrees with the data distribution's mode for constrained training, but not for constrained decoding. To do this, we construct a fixed-length output language $\mathcal{L} = \texttt{acd} \mid \texttt{bcd} \mid \texttt{bce}$, where an unconstrained CRF is limited by the Markov property to predict $\boldsymbol{y}$'s prefix and suffix independently, and choose a data distribution which violates this independence assumption. We select our data distribution,

$$\begin{aligned} P(\texttt{ooo}, \texttt{acd}) &= 0.4; \\ P(\texttt{ooo}, \texttt{bcd}) &= 0.3; \\ P(\texttt{ooo}, \texttt{bce}) &= 0.3\,, \end{aligned} \tag{5.12}$$

to be close to uniform, but with one label sequence holding the slight majority, and we ensure that the majority label sequence is *not* the label sequence with both the majority prefix and the majority suffix (i.e. `bcd`). As before, we hold the observation sequence as a constant (`ooo`). We train a constrained and an unconstrained CRF to con-

$\boldsymbol{y}$	$P(\boldsymbol{y} \mid \boldsymbol{x})$	$\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$	$\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$
`acd`	**0.4**	0.32	**0.40**
`bcd`	0.3	**0.48**	0.30
`bce`	0.3	0.20	0.30

Table 5.1: Output distributions for constrained decoding ($\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$) and constrained training ($\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$), compared to the target distribution $P(\boldsymbol{y} \mid \boldsymbol{x})$. Constrained decoding cannot learn the target distribution exactly, and yields a mode which disagrees with that of the target distribution.

vergence, and compare $\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ to $\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$.

Table 5.1 shows $\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ and $\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ as they compare to $P(\mathbf{y} \mid \mathbf{x})$. We find that, while the mode of $P(\mathbf{y} \mid \mathbf{x})$ is `acd`, with probability of 0.4, the mode of constrained decoding distribution $\hat{P}_{\boldsymbol{\theta}^*_{\text{u}}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ is `bcd`, the string with the majority prefix and the majority suffix, to which the model assigns a probability of 0.48. Conversely, the constrained training distribution $\hat{P}_{\boldsymbol{\theta}^*_{\text{c}}}(\mathbf{y} \mid \mathbf{x}; \mathcal{L})$ matches the data distribution almost exactly, and predicts the correct mode.

5.7 Real-world data experiment: semantic role labeling

As a final experiment, we apply our RegCCRF to the relation extraction task of semantic role labeling (SRL) in the popular PropBank framework (Palmer et al., 2005). We use our architecture to extract relations corresponding to events, with each relation containing a predicate and all of its arguments as spans. In line with previous work, we adopt the *known-predicate setting*, where event predicates are given and the task is to mark token spans as event participants, labeled by their (semantic) roles. PropBank assumes 7 semantic *core roles* (`ARG0` through `ARG5` plus `ARGA`) plus 21 *non-core roles* for modifiers such as times or locations. For example, in

the argument labels inform us who does the seeing (`ARG0`), who is seen (`ARG1`), and when the event took place (`ARGM-TMP`). In addition, role spans may be labeled as *continuations* of previous role spans, or as *references* to another role span in the sentence. SRL can be framed naturally as a sequence labeling task (He et al., 2017b). However,

the task comes with a number of hard constraints that are not automatically satisfied by standard CRFs:

1. Each core role may occur at most once per event.

2. For continuations, the corresponding span type must occur previously in the sentence.

3. For references, the corresponding span type must occur elsewhere in the sentence (before or after).

5.7.1 Data

In line with previous work (Ouchi et al., 2018), we work with the OntoNotes corpus as used in the CoNLL 2012 shared task[1] (Weischedel et al., 2011; Pradhan et al., 2012), whose training set comprises 66 roles (7 core roles, 21 non-core roles, 19 continuation types, and 19 reference types).

[1] As downloaded from `https://catalog.ldc.upenn.edu/LDC2013T19`, and preprocessed according to `https://cemantix.org/data/ontonotes.html`

5.7.2 RegCCRF Models

To encode the three constraints listed above in a RegCCRF, we define a regular language describing valid BIO sequences (Ramshaw and Marcus, 1999) over the 66 roles. A minimal unambiguous NFA for this language has more than $2^2 \cdot 3^{19}$ states, which is too large to run the Viterbi algorithm on our hardware. However, as many labels are very rare, we can shrink our automaton by discarding them at the cost of imperfect recall. We achieve further reductions in size by ignoring constraints on reference roles, treating them identically to non-core roles. Our final automaton recognizes 5 core role types (`ARG0` through `ARG4`), 17 non-core / reference roles, and one continuation role type (for `ARG1`). This automaton has 672 states, yielding a RegCCRF with 2592 tags. A description of our procedure for constructing this automaton can be found in Appendix B.3.

Our model is given by this RegCCRF, with emission scores provided by a linear projection of the output of a pretrained RoBERTa network (Liu et al., 2019). In order to provide the model with event information, the given predicates are prefixed by a special reserved token in the input sequence. RoBERTa parameters are fine-tuned

during the learning of transition scores and projection weights. We perform experiments with both constrained training and constrained decoding settings – we will refer to these as *ConstrTr* and *ConstrDec* respectively. A full description of the training procedure, including training times, is provided in Appendix B.1.

As RegCCRF loss is only finite for label sequences in $\mathcal{L}$, we must ensure that our training data do not violate our constraints. We discard some roles, as described above, by simply removing the offending labels from the training data. In six cases, training instances directly conflict with the constraints specified — all cases involve continuation roles missing a valid preceding role. We discard these instances for *ConstrTr*.

5.7.3 CRF baselines

As baseline models, we use the same architecture, but with a standard CRF replacing the RegCCRF. Since we are not limited by GPU memory for CRFs, we are optionally able to include all role types present in the training set, using the complete training set. We present two CRF baseline models: *CRF-full*, which is trained on all

	Model	Precision$^{(\text{rank})}$	Recall$^{(\text{rank})}$	$F_1{}^{(\text{rank})}$
Our models	CRF (*CRF-full*)	$86.89^{(2)}$	$87.81^{(1)}$	$87.35^{(2)}$
	CRF (*CRF-reduced*)	$86.92^{(2)}$	$87.35^{(2)}$	$87.13^{(3)}$
	RegCCRF (*ConstrDec*)	$87.05^{(1.5)}$	$87.38^{(2)}$	$87.21^{(2.5)}$
	RegCCRF (*ConstrTr*)	$87.31^{(1)}$	$87.76^{(1)}$	$87.53^{(1)}$
Results from literature	He et al. (2017b)	—	—	85.5
	Ouchi et al. (2018)	87.1	85.3	86.2
	Ouchi et al. (2018) Ensemble	88.5	85.5	87.0
	Li et al. (2019)	85.7	86.3	86.0

Table 5.2: Results from our experiments (averaged over twelve trials), along with selected reported results from recent literature. We rank of our models by precision, recall, and F_1 score – there exists a significant difference between two comparable values if and only if their rankings differ by one or more. Statistical significance is reported at $p < 0.05$ (two-tailed), as measured by a permutation test.

role-types from the training set, and *CRF-reduced*, which includes the same subset of roles as the RegCCRF models. For *CRF-reduced*, we use the same learned weights as for *ConstrDec*, but we decode without constraints.

5.7.4 Results and analysis

We evaluate our models on the evaluation partition, and measure performance using F_1 score for exact span matches.

For comparability with prior work, we use the evaluation script[2] for the CoNLL-2005 shared task (Carreras and Màrquez, 2005b). These results, averaged over twelve trials, are presented in Table 5.2.

All of our models outperformed the ensemble model of (Ouchi et al., 2018), which represented the state of the art for this task at the time of original publication of this work. We ascribe this improvement over the existing literature to our use of RoBERTa – prior work in SRL relies on ELMo (Peters et al., 2018), which tends to underperform transformer-based models on downstream tasks (Devlin et al., 2019a).

Of our models, *ConstrTr* significantly[3] outperforms the others in F_1 score and yields a new SOTA for SRL on OntoNotes, in line with expectations from theoretical analysis and on synthetic data. For our unconstrained models, *CRF-full* and *CRF-reduced*, the constraints specified in our automaton are violated in 0.81% and 0.84% of all output sequences respectively.[4]

[2] As available from `https://www.cs.upc.edu/~srlconll/soft.html`.

[3] All significance results are at the $p < 0.05$ level (two-tailed), as measured by a permutation test over the twelve trials of each model.

[4] For *CRF-full*, we only count violations of constraints for those roles

Among our four models, we see interesting trade-offs between precision and recall. For precision, while not all comparisons reach statistical significance, models with constraints seem to outperform those without. This is not surprising at all: the purpose of constraints is to prevent models from making predictions we know to be false a priori, and so we should expect constrained models to be more precise overall.

For recall, we observe two clusters: *CRF-full* and *ConstrTr* both perform a bit better, while *CRF-reduced* and *ConstrDec* both perform a bit worse. As *CRF-full* is the only model using the full tag set, and thus the only one capable of predicting rare role types, its high recall is to be expected. Our other three models show an interesting pattern with regards to recall: *CRF-reduced* and *ConstrDec* perform about at parity, with *ConstrTr* showing significant improvements. This behavior turns out to be quite intuitive: *CRF-reduced* and *ConstrDec* were trained identically, and only differ in their decoding procedure. The decoding-time constraints of *ConstrDec* largely work to prevent spurious predictions, and so we shouldn't expect these models to differ much in the number of true roles

that our automaton accounts for.

	Model	Precision$^{(\text{rank})}$	Recall$^{(\text{rank})}$	$F_1{}^{(\text{rank})}$
Core roles	CRF (*CRF-full*)	$89.81^{(1.5)}$	$90.62^{(2)}$	$90.21^{(2)}$
	CRF (*CRF-reduced*)	$89.79^{(2)}$	$90.59^{(2)}$	$90.19^{(2)}$
	RegCCRF (*ConstrDec*)	$90.01^{(1.5)}$	$90.56^{(2)}$	$90.28^{(2)}$
	RegCCRF (*ConstrTr*)	$90.11^{(1)}$	$90.92^{(1)}$	$90.51^{(1)}$
Non-core roles	CRF (*CRF-full*)	$80.70^{(2)}$	$81.84^{(1)}$	$81.27^{(1)}$
	CRF (*CRF-reduced*)	$80.75^{(1.5)}$	$80.44^{(3)}$	$80.59^{(2)}$
	RegCCRF (*ConstrDec*)	$80.71^{(1.5)}$	$80.59^{(3)}$	$80.65^{(2)}$
	RegCCRF (*ConstrTr*)	$81.29^{(1)}$	$81.03^{(2)}$	$81.16^{(1)}$

Table 5.3: Results for our models, broken down for core and noncore roles. Rankings and significance tests are calculated identically as in Table 5.2.

they predict. On the other hand, since *ConstrTr* is trained with constraints, it can learn to be less "cautious," with its predictions, as its constraints will automatically prevent many false positives. Thus, at evaluation time, *ConstrTr* finds more roles that were avoided by *CRF-reduced* and *ConstrDec*.

Looking at individual roles, we find that *ConstrTr* improves upon both *ConstrDec* and *CRF-reduced* on both core- and noncore roles, as is shown in Table 5.3. This is somewhat surprising, as noncore roles are not themselves subject to any constraints, and so we might expect that model constraints shouldn't affect performance on this class. Nonetheless, as we demonstrated in Section 5.6.2,

constraints during training can help models select the correct alternative during inference even when none of those alternatives violate the constraints. We suspect a similar effect is at play here.

5.8 Conclusion and future work

In this chapter, we presented a method for joint entity and relation extraction based on enforcing regular-language constraints on a CRF sequence labeling model. We investigated the resulting RegCCRF architecture both theoretically and empirically. From the theoretical perspective, we examined how constraints interact with training – while existing approaches allow constraints to be enforced at prediction time, our construction also permits constraints to be used during training. We demonstrate conclusively that training-time constraints can better capture the target distribution, and should be preferred. Empirically, we use our proposed model for semantic role labeling, showing that our proposal is a practical and effective approach to joint entity and relation extraction.

Considering task-general relation extraction as a whole, RegCCRFs represent a promising approach to jointly

modeling entities and relations, and provide an elegant formalism for encoding task constraints. However, despite their success at SRL, RegCCRFs posses a number of limitations that prevent their use as stand-alone task-general architectures. We propose that they are best seen not as an all-in-one architecture, but as a useful building block for use within task-general architectures. Here, we will briefly discuss how RegCCRFs could be incorporated into architectures which are applicable for wider classes of tasks, framing this discussion in terms of their limitations as stand-alone architectures.

The most immediately apparent limitation of RegCCRFs is their inability to enforce non-regular constraints. Somewhat surprisingly, this turns out to not limit us very much, as the vast majority of relation extraction tasks seem to have constraints that are expressible as a regular language. Nonetheless, there are cases when a more expressive language for constraints is desired – for instance, in tasks that allow relations to be entities in super-relations, relations may take on a tree structure, and might necessitate context-free constraints. Nonetheless, even here, RegCCRFs could be used along with regular-language approximations of the true constraints – in the context

free case, for instance, such approximations are easily computable and well-studied (Nederhof, 2000; Mohri and Nederhof, 2001; Egecioglu, 2009). An architecture for such tasks could incorporate a RegCCRF to enforce depth-limited constraints, with some post-processing to ensure that predictions abide by the full constraints.

A more subtle, yet much more relevant shortcoming involves how RegCCRFs model long-distance dependencies. While the CRF can learn arbitrary local interactions between adjacent labels, the long-distance dependencies introduced by constraints can only be specified a priori, not learned. Thus, a pure RegCCRF-based model for relation extraction cannot learn arbitrary interactions between the different spans of a relation. In practice, in a document containing multiple relations, a RegCCRF has no way of partitioning the spans it finds into multiple cohesive relations. For example, if a document contains two distinct binary relations of the same type (involving four distinct spans), a pure RegCCRF would have no mechanism for preferring the two true relations over the two false relations by taking one entity from each true relation. In the case of SRL, since predicates are marked in the input, the RegCCRF is capable of learning

interactions between all spans and the predicate – these interactions prove sufficient for this task.

One promising approach to tackling this shortcoming involves using RegCCRFs as the second stage of a pipeline architecture based around *anchor spans*, similar to those discussed in Chapter 3. For many tasks, there is a single span type that exists in a one-to-one relation with relations – each occurrence of such an *anchor span* corresponds to a single occurrence of a relation containing that span as an entity. For SRL, predicates act as anchor spans, albeit ones that are given in the input. For any task that has anchor spans, we could devise a pipeline architecture wherein the first step involves identifying these anchor spans, and the second step uses a RegCCRF to build relations for each anchor span. As the anchor spans will be marked in the input of the RegCCRF, it will be able to learn arbitrary interactions between these anchor spans and all other spans.

Overall, we hope that RegCCRFs can be a valuable tool in building further task-general relation extraction architectures.

Chapter 6

Conclusion

This dissertation has focused on task generality as it applies to relation extraction tasks and architectures. Task generality, as we define it, is an oft-overlooked property of model architectures which describes how well those architectures will generalize when separately trained and evaluated on diverse tasks. If a model architecture is highly task-general, we would expect to be able to instantiate specific models for many diverse tasks, such that each model can be successfully trained to yield good performance. Conversely, an architecture which is less task-general may only allow us to instantiate models for a narrow category of tasks, or at least the models of that architecture might only show empirical success on such a narrow subset. Unlike model transferability, a concept

for describing the generalization of a specific model's *already-learned parameters* to other tasks, task generality deals with models' ability to learn successful parameters when they are trained from scratch on tasks.

We note that task generality is of particular difficulty for relation extraction. While this is partially due to conflicting definitions for different relation extraction tasks, such conceptual mismatches can be addressed. Indeed, in Section 2.2, we present a general framework of definitions for relation extraction tasks which can accommodate most existing relation extraction tasks. However, while we can *describe* most existing relation extraction tasks using a general framework, *modeling* such diverse tasks using a general architecture proves much less straightforward. This can be viewed as a consequence of the large space of possible relation structures, and a lack of of universally applicable limiting independence assumptions: Our general definitions lead to a combinatorial explosion in possible relation structures for models to predict, and there is no obvious, task-general way of breaking things up into smaller, tractable subtasks. Instead, specific limiting assumptions are applicable for specific tasks, and models which incorporate such task-specific assumptions

are not capable of being applied to tasks for which those assumptions don't hold.

This dissertation has investigated how we can work towards task generality for relation extraction in light of these challenges. To this end, we have identified a number of directions for progress in which we have made contributions:

In Chapter 3, we explored the use of formal specification in order to facilitate task generality in architectures. We developed a specification language capable of capturing much of the variety in relation extraction tasks, and incorporated this as part of DeRE, a software framework for developing and using task-general relation extraction architectures.

In Chapter 4, we focused only on the subtask of span extraction. Under this context, we investigated corpus generality in Section 4.1, where we identified many of the same difficulties as are found with task generality, and proposed a model architecture capable of generalizing to the different corpora's assumptions about input features, language, and output structure. Section 4.2 then investigated the use of performance prediction as a tool for investigating task generality. We found that

the task of predicting an architecture's performance on a task, based on the properties of the task, the properties of the architecture, and their interactions, is a powerful tool for understanding the circumstances under which architectures might be expected to generalize well to new tasks.

Finally, Chapter5 investigated relation extraction as span extraction with global constraints. We propose RegCCRFs, a method of enforcing global regular-language constraints on the output of linear-chain CRFs. After investigating the architecture's theoretical properties, we demonstrate that such models can be used directly for relation extraction by applying a RegCCRF to semantic role labeling, using the constraint language to encode information about the task structure for the model.

6.1 Persisting limitations

While we were able to make progress towards more task-general architectures, there is still an enormous amount of work to be done before we might start approaching the levels of task generality found in task paradigms such as text classification and text generation. In this section,

we will highlight a few outstanding issues which we consider to be the main stumbling blocks towards more task-general architectures, and discuss how these limitations might be addressed in the future.

6.1.1 Extrinsic incompatibilities

While there are many deep, theoretical challenges to task-generality in relation extraction, it is important not to understate the more surface-level hurdles. Perhaps due to a lack of established consensus about relation extraction formalisms, many task pairs for which architectures *should* be mutually compatible have subtle differences in formalisms which make task-generality not impossible, but simply annoying. Such persistent annoyances can discourage research into the task-generality of architectures before it begins. More nefariously, some mismatches in assumptions, while conceptually trivial, leave no straightforward way to "bridge the gap," and preclude fair comparisons between architectures which disagree about those assumptions. This section will outline some concrete examples of this sort of extrinsic incompatibility, and discuss how such problems may be avoided in the

future.

Tokenization mismatches

As we note in Section 2.2.2, different tasks may make different decisions regarding if and how the input text should be broken up into tokens. While many tasks, such as GENIA Event Extraction (Kim et al., 2013) and RiQuA Quotation Analysis (Papay and Padó, 2020), treat their input text as a pure character sequence, with no notion of tokenization, others, such as PARC quotation attribution (Pareti, 2016) and OntoNotes semantic role labeling (Weischedel et al., 2011), come with their input text pre-tokenized, according to some tokenization scheme. Model architectures usually require their text to be tokenized in some way, and those based on pre-trained language models require text be tokenized in the same way as it was during pre-training. This is usually inconsistent with corpus-provided tokenizations, requiring such models to retokenize their input text.

While tokenization woes are ubiquitous in natural language processing, the central role of text spans in most relation extraction tasks makes tokenization mismatches particularly painful. In tasks which define tokenizations,

all spans will align with the task's token boundaries, and most common models can only predict spans which align with the model's tokenization. When a model has a token boundary where none is present in the corpus, the model misses out on a useful hint that it should not predict a span boundary there. When a corpus has a token boundary where none is present for the model, the model is entirely incapable of predicting spans beginning or ending at that boundary.

An idealistic solution to these problems might be to hope for some universally applicable tokenization scheme, according to which all datasets can be tokenized and which all models can accommodate. This is impractical – datasets of different languages, and even different modalities, have drastically different requirements for their tokenizations in order to ensure the representability of their spans, and even the optimal tokenization scheme for models can depend on the language of the text (Domingo et al., 2023). Instead, we would propose that relation extraction tasks should not specify *any* tokenization, and architectures should be responsible for tokenizing text "at their own risk." Such a proposal wouldn't directly lead to task-general architectures, as many tokenization

schemes don't generalize well to many tasks. For instance, a model tokenizer which splits on whitespace may work well for RiQuA, where quotation boundaries generally coincide with spaces, but not for GENIA, where protein names might be embedded in longer chemical formulæ. Nonetheless, this convention would clearly delineate the responsibilities of datasets and model architectures, and could pave the road to future research into how architectures can perform tokenization in a task-general manner. It would furthermore allow for fairer comparisons between architectures with differing tokenization schemes – at present, architectures capable of using a dataset's tokenization as-is have an unfair advantage over those which must retokenize the text.

Differing input and feature assumptions

While relation extraction is primarily conceptualized as the extraction of relations from text, it is not uncommon for particular relation extraction datasets to come with non-textual input. The nature of such input can vary from simple textual annotations, such as the part-of-speech and constituency information present for input texts in PARC (Pareti, 2016), to entirely non-textual modalities, such as

in the MNRE multimodal relation extraction task (Zheng et al., 2021), wherein relations are illustrated with accompanying images which models can use to help make their decisions. When two tasks make different assumptions about what extratextual information is provided as input for models, task-general architectures must either anticipate all possible variations on model input, or simply ignore some task-specific inputs, hindering performance on those tasks.

In many cases, this problem can not, and perhaps should not, be solved. For instance, multi-modal relation extraction was introduced as a task specifically to investigate how models can combine textual and visual cues – model architectures for this task are designed to incorporate visual information as a central component, and we would not expect models designed for non-visual tasks to generalize well here. On the other hand, for extra-textual information such as part-of-speech tags, we might expect architectures to make use of these as features when present, while still working well when they are absent.

In this case, we feel that the path towards more task-generality looks very different from the perspective of architectures and from the perspective of tasks. For ar-

chitectures, we suggest designs which can easily incorporate optional information, while working fine when such information is absent. For example, the bag-of-feature approach to token representation we use for NQD in Section 4.1 allows us to elegantly account for the variety in tasks' features. While it is impossible for architecture designers to anticipate all types of features that tasks might make available, simply allowing for arbitrary per-token and per-document features is a good start. On the other hand, for new tasks, we recommend that no extra-textual information be provided unless it is central to the task. Not only is this formalism generally more realistic (deployed relation extraction models will rarely have access to gold-standard parse trees, for instance), it also greatly eases model comparisons, and precludes unfair comparisons between feature-aware and feature-agnostic architectures.

6.1.2 Challenges with relation building

Apart from these extrinsic hurdles to task-generality, there exist more fundamental, theoretical challenges that limit the task-generality of model architectures for relation

extraction. Throughout the works presented in this dissertation, a common theme has been that task-general entity extraction seems attainable, while fully task-general architectures for end-to-end relation extraction remain out of grasp. For instance, in Section 4.2, while we do observe significant differences in how each span extraction architecture performs on different tasks, our higher-performing architectures tend to perform adequately across all tasks, and seem to satisfy our practical needs for task-general span extraction systems. On the other hand, the two end-to-end relation extraction systems we present, the DyGIE baseline model from Chapter 3 and the RegCCRF architecture presented in Chapter 5, are both quite limited not even in their performance across tasks, but in their applicability to diverse tasks: the DyGIE baseline can only be applied to tasks where relations are anchored by a particular span type, and the RegCCRF architecture can only be applied to tasks whose relation structures can be represented as a regular language. A large scale comparison between full relation extraction models across tasks, similar to that presented in Section 4.2 for span extraction, would not be possible, as we never identified *any* relation extraction models which could be broadly

applied across tasks.

We argue that this is largely a consequence of two factors: differences in the combinatorics of span extraction and relation building, and the availability of broadly-relevant simplifying assumptions for the former but not for the latter. Combinatorically, it is easily observable that the number of possible span structures for a document of length n, $O\left(2^{n^2}\right)$, is dwarfed by the number of possible relation structures (assuming one k-ary relation type between spans), $O\left(2^{n^{2k}}\right)$, making end-to-end relation extraction an intrinsically harder task than simple span extraction. However, span extraction is further facilitated by broad acceptability of two critical simplifying assumptions: a) Spans cannot overlap, and b) Distant span candidates can be considered independently from one another. While these two assumptions are not true across all possible span extraction tasks, they are applicable across a sizeable core of many diverse tasks. Importantly, these allow span extraction to be reduced to sequence labeling, a task paradigm for which an abundance of powerful model architectures exist. On the other hand, we were unable to identify such a broadly applicable and helpful suite of simplifying assumptions which

can be used for relation building. Relation candidates commonly compete with one another, precluding broad independence assumptions between different candidates, and often share entities between them, preventing us from disallowing overlaps. Even if we *could* identify a set of such assumptions for relation building, there are no obvious choices for model architectures that could take advantage of such assumptions equivalent to sequence labeling architectures for span extraction. While the situation for task-general relation building may never be as simple as that for span extraction, one way forward involves a better understanding of the simplifying assumptions that are applicable to different classes of relation extraction tasks, and of the model architecutures that are available which can leverage these assumptions.

6.1.3 A poor understanding of interdependencies

An formal investigation of this sort of simplifying assumption is best understood in terms of dependency relations between random variables in the data and model distributions. We propose that much of the "mystery"

surrounding the generalizability or architectures comes down to the independence assumptions of these architectures, which can either hold, approximately hold, or not hold for different tasks. Given this, we feel that a better understanding and consistent treatment of the independencies between variables of different tasks is essential to furthering our understanding of architecture generalizability.

(In)dependencies known a priori

In some cases, we will know a priori for a certain task that certain output variables are to be treated as independent, or conversely that they depend on one another in a certain well-defined way. As a trivial example, almost all tasks assume independence across documents, such that predictions for each document can be made independently from other documents. As for dependencies, an example might be the core role types in Semantic role labeling for OntoNotes (Weischedel et al., 2011), wherein at most one of each may occur for a given predicate – models should ideally respect the dependency relations between the candidates for each core role in order to avoid predicting more than one of them.

This dependency information can be interpreted as "promises" particular task descriptions make about their data distributions, wherein we might either be promised that two variables are independent, or conversely that a well-defined relationship exists between the values of some variables. [1] When we are given such a promise, there are benefits in selecting an architecture which makes similar promises about its model distributions. As modeling variables independently is generally much easier than modeling dependencies between them, a priori knowledge of independencies can allow us to make "safe" simplifying independence assumptions in models. Conversely, if we can ensure an architecture will respect a known relation between output variables, we can reduce the search space of our task "for free."

In order to make better use of a priori dependency information, it would be advantageous to create and catalogue more model architectures which can directly account known dependencies and independencies via hyperparameters. This dissertation has proposed two such

[1] Of course, these promises might not be true – for instance, data collection procedures for many datasets introduce dependencies across documents. Nonetheless, we will consider the setting where we would like to accept these promises without question

architectures: The baseline architecture for DeRE, presented in Chapter 3, takes task specifications, including cardinality constraints for spans, as a hyperparameter, and the RegCCRF architecture, presented in Chapter 5, accepts as a hyperparameter a regular language encoding a task's hard constraints. However, both of these are quite limited in their scope of applicability, and for most relation extraction tasks, prior dependency knowledge can only be utilized by bespoke architectures. We hope that future research can develop further architectures capable of being adapted to the known dependencies of tasks, such that a broader range of tasks can be modeled by adapting existing architectures.

Learnable dependencies and independence assumptions

In contrast to the dependency knowledge discussed above, many tasks have interdependencies between output variables which are not directly specified in task descriptions, but rather arise as a consequence of the particulars of the data. For instance, the RiQuA quotation detection task does not constrain how distant a cue can be from a quotation span, but empirically, a vast majority of cues

occur within ten tokens of their corresponding quotation (Papay and Padó, 2020). This represents a dependency between the textual position of these two entities within a relation – models which attempt to predict quotations and cues independently would be unable to to model this preference for nearby spans.

This particular example was chosen to be as explicit as possible, but also vastly understates the importance of this sort of dependency. In fact, capturing these dependencies is arguably the core of relation extraction – models *need* to account for dependencies between entities in order to ensure that those entities are actually in a relation with one another. However, as discussed in Section 1.1.2, if models try to account for *all* interdependencies that might be important, the space of model outputs explodes combinatorially, making modeling intractable.

Many of the task-specific assumptions seen in contemporary model architectures can be viewed as *independence assumptions*, wherein models assume that two variables can be predicted independently of one another. For example, in the CRF-based entity extractors seen in Chapters 3 and 4, the Markov assumption assumes independence of distant tags (when conditioned on an intervening tag).

For relation building, we have discussed two models with quite similar independence assumptions: In Chapter 3, our baseline model assumes independence between pairs of non-anchor spans in each relation, and in Chapter 5, our RegCCRF model for semantic role labeling assumes (Markov) independence between the different roles of each predicate. [2]

A major limitation we face in the development of more task-general models is a lack of nuanced understanding of when such independence assumptions might be appropriate. Very rarely are these independence assumptions fully justified. For instance, in the GENIA event extraction task, there is no reason we should expect non-anchor spans *not* to interact. Nonetheless, as architecture developers, we need to make *some* independence assumptions in order to produce a tractable model architecture, and assuming independence between non-anchor spans in each relation seems to work "well enough." However, as we had no deep, theoretical reason for making this

[2] In actuality, both of these cases are complicated by other factors: for the DyNE baseline model, the heuristic decoding step introduces dependencies between competing role candidates, as do the constraints of our RegCCRF. Nonetheless, these dependencies are all hard constraints specified a priori, and these models are incapable of *learning* dependencies between these variables.

assumption for this task, we have no insight into how well this assumption might generalize to other tasks.

In the future, we might hope for a more powerful theoretical framework for justifying such independence assumptions. This might involve the identification of specific dataset statistics which can adequately "summarize" the variety in tasks' dependency structures, such that the values of these statistics can provide information about which simplifying model assumptions might be appropriate. Such an approach would be similar to our work for span extraction in Section 4, but we suspect that we would need to account for many more task properties, as relation extraction tasks seem much more varied than span extraction tasks.

6.2 Future approaches to relation extraction

Apart from addressing these shortcomings to existing approaches, future work for relation extraction might progress in entirely new directions. This section will briefly discuss some ways the field might move forward

which were not already explored in the work presented in this dissertation.

6.2.1 Large language models

During the completion of this dissertation, the state of the art of language modeling advanced considerably, disrupting many disparate areas of machine learning in a similar manner that BERT and deep contextual embedding systems did earlier. While this disruption arguably began with the publication of GPT-2 in 2018 (Radford et al., 2019), subsequent language models such as GPT-3 (Brown et al., 2020), PaLM (Chowdhery et al., 2022), and LLaMA (Touvron et al., 2023) have continued to push the limits of what is deemed possible for autoregressive language modeling. Such language models, often termed *large language models* (LLMs), don't represent a fundamental departure from existing paradigms of language modeling; rather these models take advantage of the scalability of transformer-based architectures, training billions of parameters on billions of tokens of text. By scaling to such extremes in parameter count and corpus size, LLMS are capable of generating high-quality, coher-

ent text, often indistinguishable from human-generated text (Brown et al., 2020).

It has been demonstrated that LLMs can be adapted for use for a wide range of tasks in NLP. One surprisingly effective approach, termed prompt-based learning, involves reframing tasks in terms of natural-language completion, encoding input as a *prompt* and polling a large language model for a completion, which is interpreted as the output. This approach takes advantage of LLMs' text-completion abilities as-is, without any fine-tuning. Prompt-based learning has been applied quite successfully to a number of NLP tasks which would traditionally require task-specific supervised training. For instance, in their initial publication on the GPT-3 model, Brown et al. (2020) report competitive results in tasks such as question answering, machine translation, and natural language inference from a prompt-based approach. A large number of variations on this approach exist; Liu et al. (2023) provides a detailed overview of many common techniques.

Given our focus on task-generality in this dissertation, LLMs' amenability to being adapted to so many disparate tasks certainly warrants our attention. In fact, this behav-

ior does not quite coincide with task-generality as we define it: as LLMs can be adapted to new tasks with only a change of prompt, and without any retraining, the behavior exhibited is closer to transfer learning. On the other hand, as fine tuning of LLM weights is often infeasible for practitioners, an alternative perspective is to treat the pretrained LLM weights as hyperparameters, and to vary the choice of prompt as if that were the model parameterization.[3] From this perspective, the behavior exhibited by LLMs matches our notion of task generality quite well.

An obvious direction for future research lies in applying LLMs to relation extraction tasks, and investigating their task-generality. As LLMs seem capable of generalizing across entirely distinct tasks in NLP, it is natural to hope that they might also be capable of generalizing across different types of relation extraction. Some research has already been done in this direction: Wan et al.

[3] As prompts are usually treated to be token sequences, and not proper parameter vectors, this should be treated only as a fruitful metaphor, and not a literal equivalence. However, under formalisms involving vector-valued prompts, such as the continuous prompts discussed in Liu et al. (2023), it is entirely correct, according to our formalisms, to treat LLM weights as hyperparameters and the prompt as the parameter.

(2023) use a GPT-3-based model to achieve competitive results across four relation extraction tasks. However, as all four of these tasks take a known-entity setting, and involve predicting a single relation type for a known entity pair, this is arguably an "uninteresting" setting from our perspective, as supervised classification models can already generalize well across such tasks. Wadhwa et al. (2023) present another promising approach, using GPT-3 for four distinct relation extraction tasks, all of which require the identification of both entities and relations. While they report very good qualitative performance, they note that it is difficult to get models to generate the exact gold-standard answer, and rely on human annotation of model predictions for evaluation. As such, this approach cannot be directly compared to any existing architectures, and it is unclear exactly how well this model is generalizing across tasks.

While LLMs do seem to be promising candidates for a new class of task-general relation extraction architectures, we see good reason to temper our expectations. As generative models, LLMs do a superb job at approximating the task of *sampling* from the target distribution – that is, generating texts with probabilities which align well to the

data. However, for relation extraction, most evaluation settings are interested in models' ability to exactly match gold-standard labels. Even if these gold standard labels correspond to an LLM's highest-probability string, the task of producing such strings, termed MAP inference, is intractable for autoregressive language models, and must be approximated with techniques such as beam search (Meister et al., 2020). Thus, even with powerful language models, the underlying difficulty of selecting the single most probable relation structure remains, and must be addressed at decoding time.

6.2.2 AutoML

Despite the desirability of a single, simple architecture compatible with all relation extraction tasks, no such silver bullet has yet shown itself. On the contrary, our research into task generality has driven us to making finer distinctions between the peculiarities of different tasks, and has begun to shed light on the sensitivities of different architectures to these particulars. It seems plausible that research into task-general relation extraction will continue this way, not with the discovery of highly

task-general monolithic architectures, but rather with a furthering understanding of the circumstances different architectures need to work well. In this case, automated machine learning, or AutoML, represents a promising path towards practical task-generality.

AutoML, as it is generally conceived, involves automated construction of a machine learning pipeline for a task (He et al., 2021). While such pipelines involve everything from data processing to feature engineering, we are particularly interested in the automated selection of model architectures and hyperparameters. In principle, if we were to gain a good understanding of how different architectures depend on different data properties, and if we collected a high-coverage library of architectures, an AutoML system could automatically analyze a new relation extraction task and select an appropriate architecture based on that task's properties. This approach is so far underexplored for relation extraction – to our knowledge, there is no existing work investigating AutoML specifically for relation extraction.

A concrete scheme for automatic architecture selection could proceed quite straightforwardly from a performance prediction paradigm as was proposed for span

extraction in Section 4. When choosing an architecture for a new task, structural and statistical properties of that task would first be determined, either automatically from data, or by user specification. These properties could then be used to identify a set of plausible candidate architectures, ones whose assumptions aren't fundamentally violated. Finally, performance prediction, based on the known task properties, could be used to estimate which architecture is expected to work best for the particular task at hand. This entire process could be envisioned as a single, task-general "meta-architecture," with all architecture search being considered an internal detail. From the perspective of an end user, such a meta-architecture would obey the same interface as any other architecture.

As we come to address more of the limitations discussed in Section 6.1, AutoML will become a more viable approach to task-general relation extraction architectures. A better understanding of the interactions between task peculiarities and model assumptions, and a broader collection of models with differing assumptions, are crucial components to the success of this approach. Through AutoML, improvements in these areas can directly lead to more task-general (meta-)architectures.

Bibliography

Heike Adel, Benjamin Roth, and Hinrich Schütze. Comparing convolutional neural networks to traditional models for slot filling. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 828–838, San Diego, California, June 2016. Association for Computational Linguistics.

Heike Adel, Laura Ana Maria Bostan, Sean Papay, Sebastian Padó, and Roman Klinger. DERE: A task and domain-independent slot filling framework for declarative relation extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Brussels, Belgium, 2018.

Apoorv Agarwal, Augusto Corvalan, Jacob Jensen, and Owen Rambow. Social network analysis of Alice in Wonderland. In *Proceedings of the NAACL-HLT 2012 Workshop on Computational Linguistics for Literature*,

pages 88–96, Montréal, Canada, June 2012. Association for Computational Linguistics.

Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

Kanav Anand, Ziqi Wang, Marco Loog12, and Jan van Gemert. Black magic in deep learning: How human skill impacts network training. In *31st British Machine Vision Conference*, 2022.

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

G Angeli, V Zhong, D Chen, A Chaganty, J Bolton, MJ Premkumar, P Pasupat, S Gupta, and CD Manning.

Bootstrapped self training for knowledge base population. In *TAC*, 2016.

R. H. Baayen. *Analyzing Linguistic Data: A Practical Introduction to Statistics using R*. Cambridge University Press, 2008.

Hyeong-Ryeol Baek and Yong-Suk Choi. Enhancing targeted minority class prediction in sentence-level relation extraction. *Sensors*, 22(13):4911, 2022.

Yajie Bao, Yang Li, Shao-Lun Huang, Lin Zhang, Lizhong Zheng, Amir Zamir, and Leonidas Guibas. An information-theoretic approach to transferability in task transfer learning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 2309–2313, 2019.

Moshe Ben-Akiva and Denis Bolduc. *Approaches to model transferability and updating: the combined transfer estimator*. Département d'économique, Université Laval, 1987.

Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1): 39–71, 1996.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13: 281–305, February 2012.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. Findings of the 2017 conference on machine translation (WMT17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Laura-Ana-Maria Bostan and Roman Klinger. An analysis of annotated corpora for emotion classification in text. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2104–2119, Santa

Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

Anne Brüggemann-Klein and Derick Wood. Deterministic regular languages. In Alain Finkel and Matthias Jantzen, editors, *STACS 92*, pages 173–184, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

Annelen Brunner. Automatic recognition of speech, thought, and writing representation in German nar-

rative texts. *Literary and Linguistic Computing*, 28(4): 563–575, 2013.

Ekaterina Buyko, Erik Faessler, Joachim Wermter, and Udo Hahn. Event extraction from trimmed dependency graphs. In *BioNLP'09 Shared Task on Event Extraction*, 2009.

X Carreras and L Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *CoNLL*, 2005a.

Xavier Carreras and Lluís Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Ninth Conference on Computational Natural Language Learning (CoNLL-2005)*, pages 152–164, Ann Arbor, Michigan, June 2005b. Association for Computational Linguistics.

F. Casacuberta and C. de la Higuera. Optimal linguistic decoding is a difficult computational problem. *Pattern Recognition Letters*, 20(8):813–821, January 1999.

Augustin Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Stanley Chen. Performance prediction for exponential language models. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 450–458, Boulder, Colorado, June 2009. Association for Computational Linguistics.

Hai Leong Chieu and Hwee Tou Ng. Named entity recognition with a maximum entropy approach. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 160–163, 2003.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. PaLM: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

J Clarke, V Srikumar, M Sammons, and D Roth. An NLP curator (or: How i learned to stop worrying and love NLP pipelines). In *LREC*, 2012.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

H Cunningham, V Tablan, A Roberts, and K Bontcheva. Getting more out of biomedical documents with GATE's full lifecycle open source text analytics. *PLOS Computational Biology*, 9(2), 2013.

JR Curran. Blueprint for a high performance NLP infrastructure. In *Workshop on software engineering and architecture of language technology systems*, 2003.

D Das, D Chen, A Martins, N Schneider, and NA Smith. Frame-semantic parsing. *Computational Linguistics*, 40: 9–56, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional

transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019a. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, Minnesota, June 2019b. Association for Computational Linguistics.

Miguel Domingo, Mercedes García-Martínez, Alexandre Helle, Francisco Casacuberta, and Manuel Herranz. How much does tokenization affect neural machine translation? In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, pages 545–554, Cham, 2023. Springer Nature Switzerland.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic

optimization. *J. Mach. Learn. Res.*, 12(null):2121–2159, jul 2011.

Ömer Egecioglu. Strongly regular grammars and regular approximation of context-free languages. In *Developments in Language Theory*, pages 207–220. Springer, 2009.

Jason Eisner. Parameter estimation for probabilistic finite-state transducers. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 1–8, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

Zied Elloumi, Laurent Besacier, Olivier Galibert, Juliette Kahn, and Benjamin Lecouteux. ASR performance prediction on unseen broadcast programs using convolutional neural networks. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5894–5898. IEEE, 2018.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21, 2019.

David Elson and Kathleen McKeown. Automatic attribution of quoted speech in literary narrative. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

David Elson, Nicholas Dames, and Kathleen McKeown. Extracting social networks from literary fiction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 138–147, Uppsala, Sweden, 2010.

Manaal Faruqui and Sebastian Pado. Towards a model of formal and informal address in English. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 623–633, Avignon, France, 2012.

D Ferrucci and A Lally. UIMA: An architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3–4):327–348, 2004.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In

Advances in neural information processing systems, pages 2962–2970, 2015.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135, 2017.

Hermann Gruber and Markus Holzer. Inapproximability of nondeterministic state and transition complexity assuming P≠ NP. In *International Conference on Developments in Language Theory*, pages 205–216. Springer, 2007.

Alon Halevy, Peter Norvig, and Fernando Pereira. The unreasonable effectiveness of data. *IEEE Intelligent Systems*, 24(2):8–12, 2009.

Eva Hasler, Adrià de Gispert, Gonzalo Iglesias, and Bill Byrne. Neural machine translation decoding with terminology constraints. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 506–512, New Orleans,

Louisiana, June 2018. Association for Computational Linguistics.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), pages 473–483, Vancouver, Canada, July 2017a. Association for Computational Linguistics.

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. Deep semantic role labeling: What works and what's next. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), pages 473–483, Vancouver, Canada, July 2017b. Association for Computational Linguistics.

Xin He, Kaiyong Zhao, and Xiaowen Chu. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Chris Hokamp and Qun Liu. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (*Volume 1: Long Papers*), pages 1535–1546, Vancouver, Canada, July 2017. Association for Computational Linguistics.

M Honnibal and M Johnson. An improved non-monotonic transition system for dependency parsing. In *EMNLP*, September 2015.

Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing, 2017.

Minghao Hu, Yuxing Peng, Zhen Huang, Dongsheng Li, and Yiwei Lv. Open-domain targeted sentiment analysis via span-based extraction and classification. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 537–546, Florence, Italy, July 2019. Association for Computational Linguistics.

James Y. Huang, Bangzheng Li, Jiashu Xu, and Muhao

Chen. Unified semantic typing with meaningful label inference. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2642–2654, Seattle, United States, July 2022. Association for Computational Linguistics.

Fotis Jannidis, Albin Zehe, Leonard Konle, Andreas Hotho, and Markus Krug. Analysing direct speech in German novels. In *Proceedings of DhD*, Cologne, Germany, 2018.

Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. In *International Colloquium on Automata, Languages, and Programming*, pages 629–640. Springer, 1991.

Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S Weld, Luke Zettlemoyer, and Omer Levy. SpanBERT: Improving pre-training by representing and predicting

spans. *Transactions of the Association for Computational Linguistics*, 8:64–77, 2020.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc., USA, 2009.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836, 2016.

Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. Sharp nearby, fuzzy far away: How neural language models use context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 284–294, Melbourne, Australia, July 2018. Association for Computational Linguistics.

Evgeny Kim and Roman Klinger. Frowning Frodo, wincing Leia, and a seriously great friendship: Learning to classify emotional relationships of fictional characters. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics:*

Human Language Technologies, Volume 1 (Long and Short Papers), pages 647–653, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

J-D Kim, Tomoko Ohta, Yuka Tateisi, and Jun'ichi Tsujii. GENIA corpus—a semantically annotated corpus for bio-textmining. *Bioinformatics*, 19(suppl_1):i180–i182, 2003.

Jin-Dong Kim, Tomoko Ohta, Sampo Pyysalo, Yoshinobu Kano, and Jun'ichi Tsujii. Overview of BioNLP'09 shared task on event extraction. In *BioNLP*, 2009.

Jin-Dong Kim, Yue Wang, and Yamamoto Yasunori. The Genia event extraction shared task, 2013 edition-overview. In *Proceedings of the BioNLP Shared Task 2013 Workshop*, pages 8–15, 2013.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

Bobby Kleinberg, Yuanzhi Li, and Yang Yuan. An alternative view: When does SGD escape local minima?

In *International Conference on Machine Learning*, pages 2698–2707. PMLR, 2018.

R Klinger and P Cimiano. Joint and pipeline probabilistic models for fine-grained sentiment analysis: Extracting aspects, subjective phrases and their relations. In *ICDMW*, 2013.

R Klinger and P Cimiano. The USAGE review corpus for fine grained multilingual opinion analysis. In *LREC*, 2014.

Martin Krallinger, Obdulia Rabal, Florian Leitner, Miguel Vazquez, David Salgado, Zhiyong Lu, Robert Leaman, Yanan Lu, Donghong Ji, Daniel M Lowe, et al. The CHEMDNER corpus of chemicals and drugs and its annotation principles. *Journal of Cheminformatics*, 7(1): 1–17, 2015.

Trausti Kristjansson, Aron Culotta, Paul Viola, and Andrew McCallum. Interactive information extraction with constrained conditional random fields. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 412–418, 2004.

John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001)*, pages 282–289, 2001.

Kenton Lee, Luheng He, Mike Lewis, and Luke Zettlemoyer. End-to-end neural coreference resolution. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 188–197, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. A survey on deep learning for named entity recognition. *IEEE Transactions on Knowledge and Data Engineering*, 2020.

Zuchao Li, Shexia He, Hai Zhao, Yiqing Zhang, Zhuosheng Zhang, Xi Zhou, and Xiang Zhou. Dependency or span, end-to-end uniform semantic role labeling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):6730–6737, Jul. 2019.

ChunYang Liu, WenBo Sun, WenHan Chao, and Wanx-

iang Che. Convolution neural network for relation extraction. In *Advanced Data Mining and Applications: 9th International Conference, ADMA 2013, Hangzhou, China, December 14-16, 2013, Proceedings, Part II 9*, pages 231–242. Springer, 2013.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9), jan 2023.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

E Loper and S Bird. Nltk: The natural language toolkit. In *Workshop on Effective Tools and Methodologies for Teaching NLP and CL*, 2002.

Yi Luan, Luheng He, Mari Ostendorf, and Hannaneh Hajishirzi. Multi-task identification of entities, relations, and coreference for scientific knowledge graph

construction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3219–3232, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Using Large Corpora*, 273, 1994.

Jonathan May and Kevin Knight. A better n-best list: Practical determinization of weighted finite tree automata. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, pages 351–358, New York City, USA, June 2006. Association for Computational Linguistics.

A McCallum, K Schultz, and S Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *NIPS*, 2009.

Andrew McCallum, Dayne Freitag, and Fernando C. N. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learn-*

ing, ICML '00, page 591–598, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. *Advances in neural information processing systems*, 30, 2017.

Brian McHale. Speech representation. In Peter Hühn, John Pier, Wolf Schmid, and Jörg Schönert, editors, *Handbook of Narratology*. De Gruyter, 2009.

Clara Meister, Ryan Cotterell, and Tim Vieira. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2173–2185, Online, November 2020. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

Paramita Mirza and Sara Tonelli. Catena: Causal and temporal relation extraction from natural language texts.

In *The 26th international conference on computational linguistics*, pages 64–75. ACL, 2016.

Mehryar Mohri. A disambiguation algorithm for finite automata and functional transducers. In *International Conference on Implementation and Application of Automata*, pages 265–277. Springer, 2012.

Mehryar Mohri and Mark-Jan Nederhof. Regular approximation of context-free grammars through transformation. In *Robustness in language and speech technology*, pages 153–163. Springer, 2001.

T Morton, J Kottmann, J Baldridge, and G Bierner. OpenNLP: A Java-based NLP toolkit. `http://opennlp.sourceforge.net`, 2005.

Eric T. Nalisnick and Henry S. Baird. Character-to-character sentiment analysis in Shakespeare's plays. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics* (*Volume 2: Short Papers*), pages 479–483, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Mark-Jan Nederhof. Practical experiments with regular

approximation of context-free languages. *Computational Linguistics*, 26(1):17–44, 2000.

Nam Nguyen and Yunsong Guo. Comparisons of sequence labeling algorithms and extensions. In *Proceedings of the 24th International Conference on Machine learning*, pages 681–688, 2007.

Hiroki Ouchi, Hiroyuki Shindo, and Yuji Matsumoto. A span selection model for semantic role labeling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1630–1642, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

Sebastian Padó, Andre Blessing, Nico Blokker, Erenay Dayanik, Sebastian Haunss, and Jonas Kuhn. Who sides with whom? Towards computational construction of discourse networks for political debates. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2841–2847, Florence, Italy, July 2019. Association for Computational Linguistics.

Martha Palmer, Daniel Gildea, and Paul Kingsbury. The

Proposition Bank: An annotated corpus of semantic roles. *Computational Linguistics*, 31(1):71–106, 2005.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

Sean Papay and Sebastian Padó. Quotation detection and classification with a corpus-agnostic model. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pages 888–894, Varna, Bulgaria, September 2019. INCOMA Ltd.

Sean Papay and Sebastian Padó. Riqua: A corpus of rich quotation annotation for english literary text. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 835–841, Marseille, France, May 2020. European Language Resources Association.

Sean Papay, Roman Klinger, and Sebastian Padó. Dissecting span identification tasks with performance prediction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*,

pages 4881–4895, Online, November 2020. Association for Computational Linguistics.

Sean Papay, Roman Klinger, and Sebastian Pado. Constraining linear-chain CRFs to regular languages. In *International Conference on Learning Representations*, 2022.

Silvia Pareti. *Attribution: A Computational Approach*. PhD thesis, University of Edinburgh, 2015.

Silvia Pareti. PARC 3.0: A corpus of attribution relations. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 3914–3920, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA).

Silvia Pareti, Tim O'Keefe, Ioannis Konstas, James R. Curran, and Irena Koprinska. Automatically detecting and attributing indirect quotations. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 989–999, Seattle, WA, 2013.

F Pedregosa et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825–2830, 2011.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

Matthew E. Peters, Sebastian Ruder, and Noah A. Smith. To tune or not to tune? adapting pretrained representations to diverse tasks. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 7–14, Florence, Italy, August 2019. Association for Computational Linguistics.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina, and Yuchen Zhang. CoNLL-2012 shared task: Modeling multilingual unrestricted coreference in OntoNotes. In *Joint Conference on EMNLP and CoNLL - Shared Task*, pages 1–40, Jeju Island, Korea, July 2012. Association for Computational Linguistics.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Lance Ramshaw and Mitchell Marcus. Text chunking using transformation-based learning. In *Natural Language Processing Using Very Large Corpora*, pages 157–176. Springer Netherlands, Dordrecht, 1999.

Pushpendre Rastogi, Ryan Cotterell, and Jason Eisner. Weighting finite-state transductions with neural context. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 623–633,

San Diego, California, June 2016. Association for Computational Linguistics.

Lev Ratinov and Dan Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, 6 2009.

KE Ravikumar, Majid Rastegar-Mojarad, and Hongfang Liu. Belminer: adapting a rule-based relation extraction system to extract biological expression language statements from bio-medical literature evidence sentences. *Database*, 2017, 2017.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 1135–1144, 2016.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS EMC² Workshop*, 2019.

Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850, 2016.

Sunita Sarawagi and William W Cohen. Semi-Markov conditional random fields for information extraction. *Advances in neural information processing systems*, 17: 1185–1192, 2004.

Christian Scheible, Roman Klinger, and Sebastian Padó. Model architectures for quotation detection. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages

1736–1745, Berlin, Germany, August 2016. Association for Computational Linguistics.

DA Schult. Exploring network structure, dynamics, and function using networkx. In *Python in Science Conference*, 2008.

Mike Schuster and Kuldip Paliwal. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, 45:2673 – 2681, 12 1997.

Elena Semino and Michael Short. *Corpus Stylistics: Speech, Writing And Thought Presentation In A Corpus Of English Writing*. Routledge Advances In Corpus Linguistics. Routledge, London, 2004.

Sonse Shimaoka, Pontus Stenetorp, Kentaro Inui, and Sebastian Riedel. Neural architectures for fine-grained entity type classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1271–1280, Valencia, Spain, April 2017. Association for Computational Linguistics.

P Stenetorp, S Pyysalo, G Topić, T Ohta, S Ananiadou,

and J Tsujii. brat: a web-based tool for NLP-assisted text annotation. In *EACL*, 2012.

Dianbo Sui, Xiangrong Zeng, Yubo Chen, Kang Liu, and Jun Zhao. Joint entity and relation extraction with set prediction networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

M Surdeanu. Overview of the TAC2013 knowledge base population evaluation: English slot filling and temporal slot filling. In *TAC*, 2013.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

Charles Sutton and Andrew McCallum. Collective segmentation and labeling of distant entities in information extraction. In *Proceedings of the ICML 2004 Workshop on Statistical Relational Learning*, 2004.

Wilson L Taylor. "Cloze procedure": A new tool for measuring readability. *Journalism quarterly*, 30(4):415–433, 1953.

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 847–855, New York, NY, USA, 2013. Association for Computing Machinery.

Tijmen Tieleman and Geoffrey Hinton. Neural networks for machine learning, lecture 6.5. Lecture notes, 2012.

Erik F. Tjong Kim Sang and Sabine Buchholz. Introduction to the CoNLL-2000 Shared Task: Chunking. In *Proceedings of the 2nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning*, page 127–132, Lisbon, Portugal, 2000. Association for Computational Linguistics.

Branimir T Todorovic, Svetozar R Rancic, Ivica M Markovic, Edin H Mulalic, and Velimir M Ilic. Named entity recognition and classification using context hidden Markov model. In *2008 9th Symposium on Neural Network Applications in Electrical Engineering*, pages 43–46. IEEE, 2008.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Hai-Long Trieu, Thy Thy Tran, Khoa N A Duong, Anh Nguyen, Makoto Miwa, and Sophia Ananiadou. DeepEventMine: end-to-end neural nested event extraction from biomedical texts. *Bioinformatics*, 36(19): 4910–4917, 06 2020.

Joaquin Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18(2):77–95, 2002.

Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.

IEEE Transactions on Information Theory, 13(2):260–269, 1967.

Somin Wadhwa, Silvio Amir, and Byron C Wallace. Revisiting relation extraction in the era of large language models. *arXiv preprint arXiv:2305.05003*, 2023.

Zhen Wan, Fei Cheng, Zhuoyuan Mao, Qianying Liu, Haiyue Song, Jiwei Li, and Sadao Kurohashi. Gpt-re: In-context learning for relation extraction using large language models. *arXiv preprint arXiv:2305.02105*, 2023.

Ralph Weischedel, Eduard Hovy, Mitchell Marcus, Martha Palmer, Robert Belvin, Sameer Pradhan, Lance Ramshaw, and Nianwen Xue. OntoNotes: A large training corpus for enhanced processing. In *Handbook of Natural Language Processing and Machine Translation: DARPA Global Autonomous Language Exploitation*. Springer, 2011.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Pro-*

ceedings of Machine Learning Research, pages 5247–5256, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Nathalie Wiedmer, Janis Pagel, and Nils Reiter. Romeo, Freund des Mercutio: Semi-Automatische Extraktion von Beziehungen zwischen dramatischen Figuren. In *Proceedings of the 7th Conference of the Organization "Digital Humanities im deutschsprachigen Raum"* (*DHd 2020*). Zenodo, February 2020.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine

Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Vikas Yadav and Steven Bethard. A survey on recent advances in named entity recognition from deep learning models. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2145–2158, Santa Fe, New Mexico, USA, August 2018. Association for Computational Linguistics.

Bishan Yang and Claire Cardie. Joint inference for fine-grained opinion extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1640–1649, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Hu Yi-Qi, Li Yu-Feng, Tu Wei-Wei, Yang Qiang, and Yu Yang. Taking the human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.

Amir R. Zamir, Alexander Sax, William B. Shen, Leonidas J. Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2018.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP 2017)*, pages 35–45, 2017.

Changmeng Zheng, Junhao Feng, Ze Fu, Yi Cai, Qing Li, and Tao Wang. Multimodal relation extraction with efficient graph alignment. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 5298–5306, 2021.

Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal statistical society: series B (Statistical methodology)*, 67(2):301–320, 2005.

Appendix A

Investigating task-generalizability with performance prediction

---

This appendix provides additional details to the content presented in Section 4.

A.1 Training Models

All code used for training span extraction and performance prediction models is available for download at our project website: `https://www.ims.uni-stuttgart.de/data/span-id-meta-learning`. All text logs generated during training of span extraction models are included.

A.1.1 Hardware

All span extraction models were trained using GeForce GTX 1080 Ti GPUs. Training time varied considerably across architectures – exact training times for individual experiments are found in the corresponding training logs.

The performance prediction model was trained on a CPU in a few seconds.

A.1.2 Tokenization

For PARC, OntoNotes, and CoNLL'00, which include tokenization information, and we use the datasets' tokenizations directly For RiQuA, we use spaCy (Honnibal and Montani, 2017) to word-tokenize the text. We found that spaCy's tokenization performed particularly poorly for ChemDNer, and so for this corpus we treated all sequences of alphabetic characters as a token, all sequences of numbers as a token, and all other characters as single-character tokens. For ChemDNer, we found that some spans within the corpus still did not align with token boundaries. In these cases, we excluded the spans entirely from the training data, and treated them as an automatic false-negative for evaluation purposes.

For models including a BERT component, tokens were sub-tokenized using word-piece tokenization (Wu et al., 2016) so as to be compatible with BERT. The same bag of token features was given to each word piece. Models predicted BIO sequences for these sub-tokens, and spans were only evaluated as correct when their boundaries matched exactly with the originally-tokenized corpus.

A.1.3 Hyperparameters

Due to the large number of experiments run, it was infeasible to do a full grid-search for hyperparameters for each architecture-dataset combination. As such, we tried to pick reasonable values for hyperparameters, motivated by existing literature, prior research, and implementation defaults of existing libraries. For BERT-based models, our choice of pre-trained model – '`bert-base-uncased`' as provided by the HuggingFace Transformers library Wolf et al. (2019) – fixed some of these hyperparameters for us. Table A.2 enumerates the hyperparameter values used for our architectures.

A.1.4 Optimizer and Training

All models were trained with the Adam optimizer (Kingma and Ba, 2015). For BERT+CRF and BERT+LSTM+CRF, we train the non-BERT parameters as a first training phase, and then fine-tune all parameters jointly as a second training phase. In these cases, Adam was re-initialized between two training phases. For training all non-BERT architectures, and for first training phase in the BERT+CRF and BERT+LSTM+CRF architectures, an initial learning rate of 0.001 was used. For BERT, and for the second training phase in the BERT+CRF and BERT+LSTM+CRF architectures, an initial learning rate of 2×10^{-5} was used.

A.1.5 Early Stopping

To guide early stopping, micro-averaged F_1 scores on the development set were computed after every epoch. These were computed for all span types, including those which were subsequently excluded from our meta-model. For datasets which had no dedicated development partition, a portion of the training set was held out for this purpose. After each epoch, model parameters were saved to disk if the development-set F_1 score exceeded the best

seen so far. An exponential moving average of these F_1 scores was kept, and training terminated when an epoch's F_1 score fell below this average. For BERT+CRF and BERT+LSTM+CRF, this same early stopping procedure was used for both training phases. The training logs list development set performance at each epoch for each experiment.

A.1.6 Features

Table A.1 lists all manual features that were used in models with the "Feat" component.

PARC	Token POS tag
	Token lemma
	Constituents containing token
	Constituents starting at token
	Constituents ending at token
RIQUA	Token POS tag †
	Token lemma †
	Is token a quotation mark?
	Is token a quotation mark?
	Is token capitalized?
	Is token all caps?
CoNLL'00	Token POS tag †
OntoNotes	Token POS tag
	Is token capitalized?
	Is token all caps?
	Character bi- and trigrams
	Constituents containing token
	Constituents starting at token
	Constituents ending at token
ChemDNer	Token POS tag †
	Token lemma †
	Is token capitalized?
	Is token all caps?
	Is token purely alphabetic?
	Is token all digits?

Table A.1: Hand-crafted features used. Entries marked with a dagger (†) were predicted using spaCy (Honnibal and Montani, 2017) – others were either manually annotated, or were exactly specified by the tokens' surface forms

Hyperparameter	Value
Input dimensionality	300
LSTM units	300
Softmax output layer units	300
CRF units	300
LSTM layers	2
LSTM dropout probability	0.5
Learning rate (non-BERT)	1×10^{-3}
Learning rate (BERT)	2×10^{-5}

Table A.2: Hyperparameter choices

Appendix B

Task-general joint modeling with regular-constrained CRFs

This appendix provides additional details to the content presented in Chapter 5.

B.1 Experimental Design

Here we detail the training procedures and hyperparameter choices for our experiments. These are summarized in Table B.1. Full code for all experiments, along with training logs, are also included in the supplementary materials.

CRFs	Transition score initialization	$\mathcal{N}(0, 0.1)$
Synthetic data experiments	Emission score initialization	PyTorch default
	Optimizer	SGD
	Training iterations	5000
	Batch size	50
	Initial learning rate	1.0
	Learning rate decay	10% every 100 steps
SRL experiments	RoBERTa weights	`roberta-base`
	Projection weight and bias initialization	PyTorch default
	Optimizer	Adam
	Learning rate	10^{-5}
	Batch size	2
	Gradient accumulation	4 batches

Table B.1: Summary of hyperparameters for our models and experiments.

B.1.1 CRFs

For all CRFs and RegCCRFs, transition potentials were initialized randomly from a normal distribution with mean zero and standard deviation 0.1. No CRFs or RegCCRFs employed special start- or end-transitions – that is, we did not insert any additional beginning-of-sequence or end-of-sequence tags for the Viterbi or forward algorithms.

B.1.2 Synthetic data experiments – training procedure

For both synthetic data experiments, the emission potentials were represented explicitly for each position as trainable parameters – since the observation sequence was constant in all experiments, these did not depend on x.

Parameters were initialized randomly using PyTorch default initialization, and optimized using stochastic gradient descent. To ensure fast convergence to a stable distribution, we employed learning rate decay – learning rate was initially set to 1.0, and reduced by 10% every 100 training steps.

We trained all models for a total of 5000 steps with a batch size of 50. All models were trained on CPUs. For the experiment described in Section 6.1, we trained separate models for each k – the total training time for this experiment was approximately 35 minutes. The experiment described in Section 6.2 completed training in approximately 30 seconds.

B.1.3 Semantic role labeling – training procedure

In the semantic role labeling (SRL) experiments, we incorporated a pretrained RoBERTa network (Liu et al., 2019) – the implementation and weights for this model were obtained using the `roberta-base` model from the Hugging Face transformers library (Wolf et al., 2020). RoBERTa embeddings were projected down to transmission scores using a linear layer with a bias – projection weights and biases were initialized using the PyTorch default initialization.

Input tokens were sub-tokenized using RoBERTa's tokenizer. The marked predicate in each sentence was prefixed by a special `<unk>` token. During training, for efficiency reasons, we excluded all sentences with 120 or more subtokens – this amounted to 0.23% of all training instances. We nonetheless predicted on all instances, regardless of length.

We optimized models using the Adam optimizer (Kingma and Ba, 2015) with a learning rate of 10^{-5}. We fine-tune RoBERTa parameters and learn the projection and RegCCRF weights jointly. For performance reasons, batch

size was set to 2, but we utilized gradient accumulation over groups of 4 batches to simulate a batch size of 8.

We utilized early stopping to avoid overfitting. Every 5000 training steps, we approximated our model's F_1 score against a subset of the provided development partition, using a simplified reimplementation of the official evaluation script. Each time we exceeded the previous best F_1 score for a model, we saved all model weights to disk. After 50 such evaluations with no improvement, we terminated training, and used the last saved copy of model weights for final evaluation.

We performed all SRL experiments on GeForce GTX 1080 Ti GPUs. Each experiment used a single GPU. Training took an average of 88 hours for RegCCRF models with constrained training, 23 hours for RegCCRF with constrained decoding, and 24 hours for CRF baseline models. All training logs with timestamps are included in the supplementary materials.

B.2 Construction as weighted FST

Here we present a construction of the RegCCRF as a weighted finite-state transducer with weight sharing. We

do this by first specifying the transducer topology used, and then specifying how edge weights are parameterized in terms of $\boldsymbol{\theta}$. The resulting transducer yields an identical distribution to that of the CRF-based construction, $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$.

B.2.1 Transducer topology

Starting from $\mathcal{L}$, we define $\ddot{\mathcal{L}}$ to be the regular language of bigram sequences for the strings in $\mathcal{L}$, i.e.,

$$\ddot{\mathcal{L}} = \left\{ \left\langle (s^1, s^2), (s^2, s^3), ..., (s^{|s-1|}, s^{|s|}), (s^{|s|}, \$) \right\rangle \mid s \in \mathcal{L} \right\}, \tag{B.1}$$

with \$ acting as a end-of-string symbol. We let $\ddot{M}$ be an unambiguous FSA for the language $\ddot{\mathcal{L}}$, and choose to interpret this automaton as a finite-state transducer by stipulating that all edges should accept any symbol in the input language (but only one symbol per transition, and without allowing epsilon transitions). This unweighted transducer will be used as the topology for our weighted finite-state transducer.

B.2.2 Edge weights

In line with Rastogi et al. (2016), we would like to assign weights to the edges of our transducer $\ddot{M}$ with a neural network. In order to obtain the same distribution as from our CRF-based construction, these weights must be parameterized in terms of our transition function $g_{\boldsymbol{\theta}}$ and emission function $h_{\boldsymbol{\theta}}$. For each edge in $\ddot{M}$, the weight depends only on the emitted bigram, the input sequence, and the index of the current input symbol – the weight does not depend on the FST states. For a symbol bigram (a, b), input sequence $\boldsymbol{x}$, and index i, the edge weight is equal to

$$W_{a,b} = \begin{cases} \exp\left(g_{\boldsymbol{\theta}}(a, b) + h_{\boldsymbol{\theta}}(\boldsymbol{x}, a, i)\right) & b \neq \$ \\ \exp\left(h_{\boldsymbol{\theta}}(\boldsymbol{x}, a, i)\right) & \text{otherwise} \end{cases} \tag{B.2}$$

Each string in $\mathcal{L}$ corresponds bijectively to exactly one bigram sequence in $\ddot{\mathcal{L}}$, which corresponds bijectively to exactly one accepting path in $\ddot{M}$ – this path's weight is equal to the unscaled probability produced by our CRF construction, and so the weighted FST, interpreted as a probability distribution, yields the distribution $\hat{P}_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}; \mathcal{L})$.

B.3 Automaton construction for semantic role labeling

Here we describe how we generate the automaton architecture for our semantic role labeling experiments. While our experiments used 5 core-roles, 17 non-core roles, and one continuation role, we discuss here a generalized setting with arbitrary sets of core, noncore, and continuations of core roles.

Algorithm 1 provides pseudocode for our construction. The core idea is to use subsets of core roles as NFA states, so that we can keep track of which core roles have already ocurred. Additional states are used in order to ensure all strings are valid BIO sequences.

$\Sigma \leftarrow \{\textsc{Outside}\} \cup (\{\textsc{Begin}, \textsc{Inside}\} \times (\mathcal{R}_{\text{core}} \cup \mathcal{R}_{\text{noncore}} \cup \mathcal{R}_{\text{continuation}}))$;
$Q \leftarrow \emptyset$;
$q_1 \leftarrow \emptyset$;
$F \leftarrow \emptyset$;
$E \leftarrow \emptyset$;
for $p \in 2^{\mathcal{R}_{\text{core}}}$ **do**
    $Q \leftarrow Q \cup \{p\}$;
    $F \leftarrow F \cup \{p\}$;
    $E \leftarrow E \cup \{(p, \textsc{Outside}, p)\}$;
    for $r \in \mathcal{R}_{noncore}$ **do**
        $s \leftarrow (r, p)$;
        $Q \leftarrow Q \cup \{s\}$;
        $E \leftarrow E \cup \{(p, (\textsc{Begin}, r), p), (p, (\textsc{Begin}, r), s)\}$;
        $E \leftarrow E \cup \{(r, (\textsc{Inside}, r), s), (r, (\textsc{Inside}, r), p)\}$;
    end
    for $r \in \mathcal{R}_{continuation}$ **do**
        if *The core role corresponding to r is in p* **then**
            $s \leftarrow (r, p)$;
            $Q \leftarrow Q \cup \{s\}$;
            $E \leftarrow$
             $E \cup \{(p, (\textsc{Begin}, r), p), (p, (\textsc{Begin}, r), s)\}$;
            $E \leftarrow$
             $E \cup \{(r, (\textsc{Inside}, r), s), (r, (\textsc{Inside}, r), p)\}$;
        end
    end

for $r \in (\mathcal{R}_{core} \setminus p)$ **do**
$s \leftarrow (r, p)$;
$t \leftarrow p \cup \{r\}$;
$Q \leftarrow Q \cup \{s\}$;
$E \leftarrow E \cup \{(p, (\textsc{Begin}, r), s), (p, (\textsc{Begin}, r), t)\}$;
$E \leftarrow E \cup \{(s, (\textsc{Inside}, r), s), (s, (\textsc{Inside}, r), t)\}$;
end
end
return (Σ, Q, q_1, F, E)

Algorithm 1: Construction of an FSA from given sets of core, noncore, and continuation roles. To represent BIO labels, we use tuples of the form (Begin, `<roleType>`) for B labels, tuples of the form (Inside, `<roleType>`) for I labels, and the symbol Outside for the sole O label.