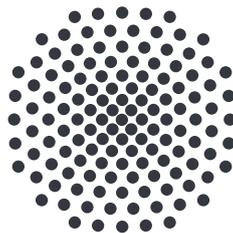


Predicting Instantaneous Decay Rates at the Transition State Using Neural Networks and Investigating Their Usefulness for the Reaction Rates of Arbitrary Ensembles

Master's Thesis of
Pooja Mishra

4. März 2021

First Examiner: Apl. Prof. Dr. Jörg Main
Second Examiner: Prof. Dr. Hans Peter Büchler



Institut für Theoretische Physik I
Universität Stuttgart
Pfaffenwaldring 57, 70550 Stuttgart

Contents

1	Introduction	5
1.1	Motivation	5
1.2	Outline of the Thesis	6
2	Transition State Theory	7
2.1	Normally Hyperbolic Invariant Manifold	8
2.2	Binary Contraction Method	10
2.3	Local Manifold Analysis	11
3	Neural Networks	15
3.1	Feedforward Neural Network	17
3.2	Layers	21
3.3	Training of a Neural Network	22
3.3.1	Optimization of a Neural Network	22
3.3.2	Loss functions for a Neural Network	23
3.4	TensorFlow	24
4	Results	25
4.1	Prediction of Instantaneous Rates by Neural Networks	25
4.2	Usefulness for Reaction Rates of Arbitrary Ensembles	30
5	Summary and Outlook	35
	Bibliography	37
	Acknowledgements	43

1 Introduction

One of the most researched topic, which is of interest for more than one science field including physics, chemistry, and mathematics, is transition state theory (TST). In a typical reaction, one or more reactants combine together to form either a single product or multiple products. The interesting point is how fast or slow the reaction takes place and what are its resultant products.

The reaction dynamics can be described by the potential energy surface (PES). The combined state of all involved particles is encoded as an effective particle on the PES. The reactant and product states correspond to different regions on the PES. This potential energy surface is, normally, divided by a saddle point into two basins, the reactant and product basin. Reactants are converted to products when the reactant overpasses this saddle. Investigating whole ensembles of reactants, some will overpass the saddle. Some will not overpass the saddle and, thus, will not be converted to products. How many of the reactants will react (overpass the saddle point) and how fast they do is encoded in the reaction rate. On the other hand, decay rates explain the crossing of the dividing surface by the reactants. For a one-dimensional reaction (reaction with one degree of freedom) the decay rate can be determined by counting the number of reactants that surpass the maximum of the saddle in a given time. For this to yield the decay rate, the ensemble has to be initialized close to the saddle point. However, with higher degrees of freedom, the reaction rate cannot be determined by only knowing the maximum of the saddle. Then the whole structure of the saddle needs to be taken into consideration. Using TST, one can find a dividing surface (DS) that clearly separates reactants and products. The reaction rate then is calculated by the flux of reactants through this DS.

1.1 Motivation

The world is trying to utilize the machines for every small and big problem. As we look around ourselves we can see most of the things are now machine operated. Everything is getting simpler. And it is always good to simplify any obstacle. It makes our life easier. As we hear stories from our parents and grandparents, we realize that life is much smoother and easier today compared to what it was earlier.

Machine learning (ML) applications have been used in a wide range of fields including medical diagnosis, stock trading, robot control, law, scientific discovery, video games, toys etc. ML is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention. We would also want to employ the ML to predict the decay rates of a trajectory near the normally hyperbolic invariant manifold (NHIM) and understand the behavior of TST. An important object in TST is the NHIM. It is a geometric structure and the region of the phase space which forms the basis for the definition of the dividing surface (DS) separating reactants from products. To calculate decay rates, we do lots of calculations involving binary contraction method (BCM) to compute the NHIM. But the BCM turns out to be a bit expensive to compute the NHIM.

1.2 Outline of the Thesis

In this thesis we are going to discuss the TST (chapter 2 will discuss it). Briefly explaining the NHIM and the instantaneous decay rates. The main topic of discussion is the neural networks, as discussed in chapter 3 and their emerging usage in the market. We will build our own neural network to predict the time-dependent instantaneous decay rates of a chemical reaction from random samples at the position and velocity coordinates on the NHIM. Also we are going to discuss the prediction of neural networks taking coordinates of a trajectory as the initial conditions. And eventually we will investigate on the usefulness for the reaction rates of arbitrary ensembles by predicting the time-dependent instantaneous rates of trajectories near the NHIM.

2 Transition State Theory

Transition state theory (TST) demonstrates the reaction rates of elementary chemical reactions. This theory explains about the decay of the reactants into the final products. According to this theory, a complex is formed as the intermediate step between the reactants and the products. The theory assumes a special type of chemical equilibrium (quasi-equilibrium) between reactants and activated transition state complexes [2].

TST as derived in the 1930s approaches the problem of calculating a reaction rate for a rarely occurring elementary reaction by separating space into two regions called the reactant region and the product region. The reactant region defines the general region in which the system can be found before reacting, and the product region defines what is thought of as a product of the elementary reaction in question. The border between the two regions is referred to as the transition state (TS). The lowest energy configurations in the reactant and product regions are often referred to as the initial state and final state, respectively [3].

Transition state theory can also be explained in other terms: atoms and molecules of the reactants collide and combine to form an unstable, high energy complex. When the molecules (high energy complex) fall out of this high energy state, they may do so as new and different molecules, or in their original states. The energy required to reach the activated state must be available if the molecules are to change into something new. Fig. 2.1 shows the activation energy required to reach the product side for an endothermic and exothermic reaction.

The framework of TST [4] provides a powerful tool for the qualitative and quantitative description of these reactions from initial state (reactants) to final state (products). Usually, these states appear as (local) minima in the corresponding potential energy surface and they are separated by an energy barrier or saddle. Typically, such a saddle is characterized by a rank-1 saddle point, which has one unstable direction, the reaction coordinate and an arbitrary number of stable degrees of freedom, called orthogonal modes.

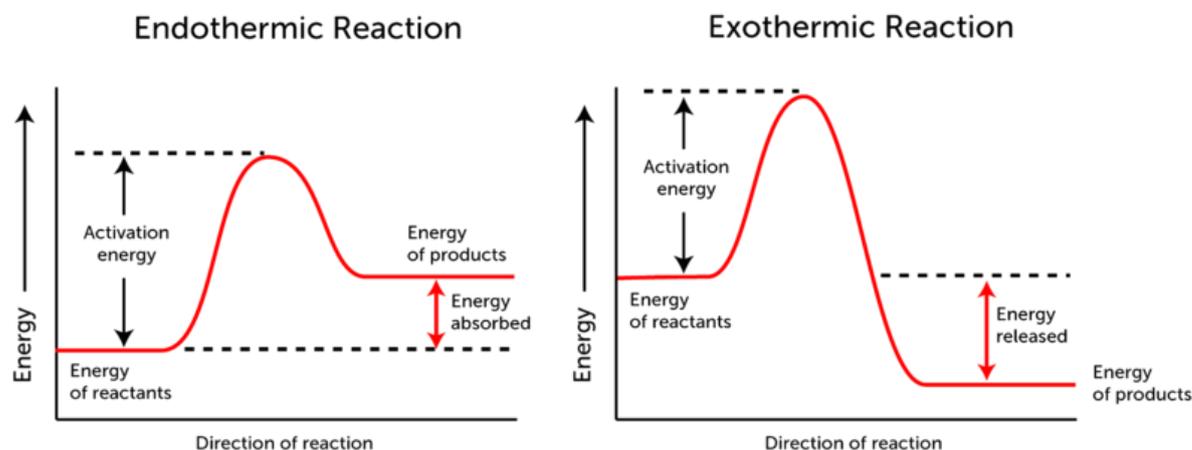


Figure 2.1: Reaction coordinate diagram for endothermic and exothermic reactions. An energy barrier with a saddle (transition state) dividing the reactants and products. Activation energy (ΔG) is the standard Gibbs energy required by the reactants to convert to products [1].

2.1 Normally Hyperbolic Invariant Manifold

In classical systems reactive trajectories must have at least enough energy to overcome a barrier close to its saddle point on their way from a reactant state to a product state. In other words, the localized area of a saddle point in the phase space creates a hinderance for most of the reactive trajectories. Since their energy is minimum here, the corresponding reactive flux is lowest, and, hence, measuring it is important for obtaining reaction rates. Therefore, good knowledge of the dynamics of reactive and non-reactive trajectories in the barrier region is very important, which can be known by learning about another localized area of the phase space which is called as normally hyperbolic invariant manifold (NHIM) [6]. On the NHIM, trajectories are unstably bound to the vicinity of the barrier for infinite time at the boundary between reactants and products. Hence, this intermediate state is often referred to as the TS.

To obtain the reactive flux and the associated rate one has to know exactly when and where a reactive trajectory reacts, or more precisely, crosses the boundary between reactants and products in the phase space, called dividing surface (DS), see Fig. 2.2. In case of a rank-1 saddle a simple but intuitive DS to separate reactants from products in the phase space would be a planar surface which is attached to the saddle point of an energy barrier and spanned perpendicular to its unstable degree of freedom in direction of the orthogonal modes. However, this is in general a rather strong approximation since such a planar DS is usually recrossed by many reactive trajectories on their way from reactants to products [7, 8]. These recrossings lead to an over estimation of both the flux

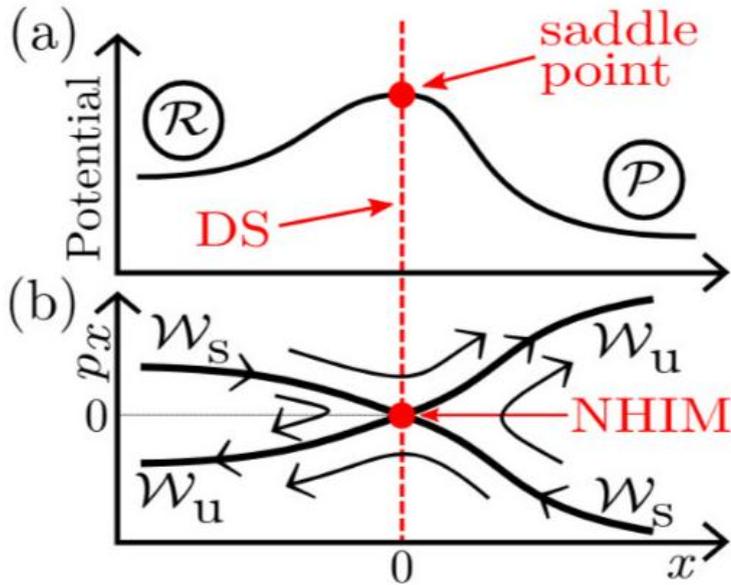


Figure 2.2: (a) Sketch of an energy barrier with reaction coordinate x . The origin of the coordinates is set to $x = 0$ at the maximum of the barrier. At the saddle point, a DS segregates the system into reactant and product. (b) Sketch of the phase space x, p_x showing the stable W_s and the unstable W_u manifolds. The interaction of these manifolds, situated at $x = 0$ as the saddle point, known as NHIM [5].

of reactive trajectories through such a planar DS as well as of the associated reaction rate and, consequently, these rates are in that case always an upper bound to the true reaction rate. To obtain accurate rates a DS has to be recrossing-free meaning that it should be crossed once and only once by each reactive trajectory. Thus, these recrossing-free DSs are usually multidimensional, non-trivially curved and their construction is a challenging task.

To characterize a reaction in the context of TST, the phase space geometry in the barrier region is important. Based on the dynamics of trajectories close to a saddle this region can be divided into different reactive and non-reactive areas. The boundaries of these areas, called the stable and the unstable manifold intersect on the NHIM. To this intersection, a recrossing-free DS can be attached.

Finding the NHIM can be done numerically. For a given initial reaction coordinate x_0 , the corresponding velocities v_x of the stable and the unstable manifolds can be obtained using a standard iterative routine to find the maxima of, e.g., the modified Lagrangian descriptor (LD) or the time descriptor (TD). The NHIM is located at the intersection of the stable and the unstable manifold. At the corresponding position x^{NHIM} , the v_x

distance between these manifolds is zero. Consequently, x^{NHIM} can be found iteratively using a root search for the distance of the stable and the unstable manifold at various positions x_i . This procedure, however, is numerically very expensive since it nests the iterative routine to find the maxima of, e.g., the modified LD or the TD into the root search of the v_x -difference between the two manifolds. The fact that obtaining any of the many LD or TD values requires the propagation of a full trajectory makes the situation even worse. Let's discuss a different approach to numerically obtain the position of the NHIM in the phase space, the binary contraction method (BCM) which is much faster and more elegant than just using nested iterations.

2.2 Binary Contraction Method

By introducing a barrier region $x \in [x_R, x_P]$ for an appropriately chosen reaction coordinate x , various trajectories can be classified to different reactive areas. Thereby, the basic idea is to determine at which side a trajectory entered the barrier region in the past and to which side it leaves the barrier region when propagated forward in time. The classification of reactive and non-reactive trajectories offers an alternative way to find the NHIM, the BCM [6], which avoids the nested iterations needed to find the stable and the unstable manifolds. The binary contraction method (BCM) is explained best by means of Fig. 2.3. It starts with a quadrangle in an (x, v_x) section of the phase space. Each vertex of the quadrangle has to be located in a different reactive or non-reactive area which are separated by the stable and the unstable manifold. This initialization is non-trivial since the specific location of the manifolds is a priori unknown. An initial quadrangle fulfilling this requirement can be either obtained by choosing it large enough or making an educated guess.

To begin the BCM, four trajectories, initialized at each vertex of the quadrangle, have to be propagated both forward in and backward in time until they leave the barrier region. This procedure allows to assign each trajectory to one of the reactive and non-reactive areas and therefore is a self-consistency check if the BCM is initialized correctly. During the following iterations of the BCM a candidate vertex is successively chosen at the middle of each side of the quadrangle. At each candidate vertex, another trajectory is propagated both forward and backward in time and classified to one of the reactive areas. Now, the area corresponding vertex of the current quadrangle is replaced by the candidate vertex. This procedure gradually shrinks the area of the quadrangle, as visualized in Fig. 2.3 for the first four iterative steps. Finally, its center converges to the intersection of the manifolds yielding x^{NHIM} and v_x^{NHIM} . Depending on the desired accuracy, the BCM can, e.g., be stopped if the relative error of this center point is small enough. For multidimensional systems the position $(x^{\text{NHIM}}, v_x^{\text{NHIM}})(y, v_y, t)$ of the NHIM

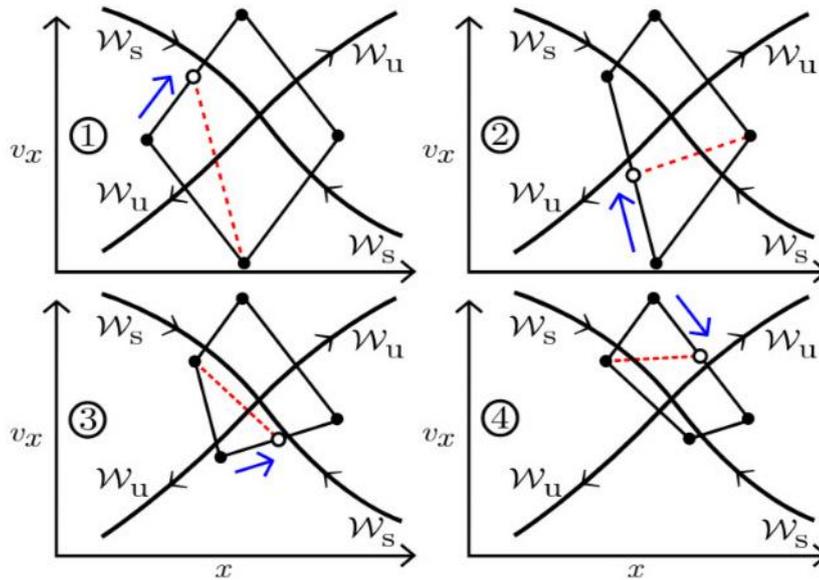


Figure 2.3: Sketch of the binary contraction method. BCM is an iterative method that starts with a quadrangle having its four vertices in each of the four different areas of the phase space, which are separated by the stable W_s and the unstable W_u manifold. Shrinking the quadrangle leads to the intersection of the manifolds [6].

for given orthogonal modes (y, v_y) at a time t may be found pointwise using the BCM. Here, for any point sought on the NHIM, a new quadrangle has to be initialized in an appropriate section of the full phase space.

2.3 Local Manifold Analysis

The geometry of the phase space can be utilized in a regime very close to the NHIM. Here, the dynamics is linear to a good approximation and an analysis of the slopes of its stable and unstable manifolds – referred to as the local manifold analysis (LMA) [9] – provides a considerably faster approach to obtain decay rates of the TS. The basic idea behind LMA is linear dynamics in a close neighbourhood of the NHIM. Instantaneous decay rates can be written as

$$k(t) = \frac{-\dot{N}_{react}(t)}{N_{react}(t)} \quad (2.1)$$

LMA calculates the instantaneous decay rates k along saddle bound trajectories near the activated complex by utilizing the local properties of the stable and the unstable

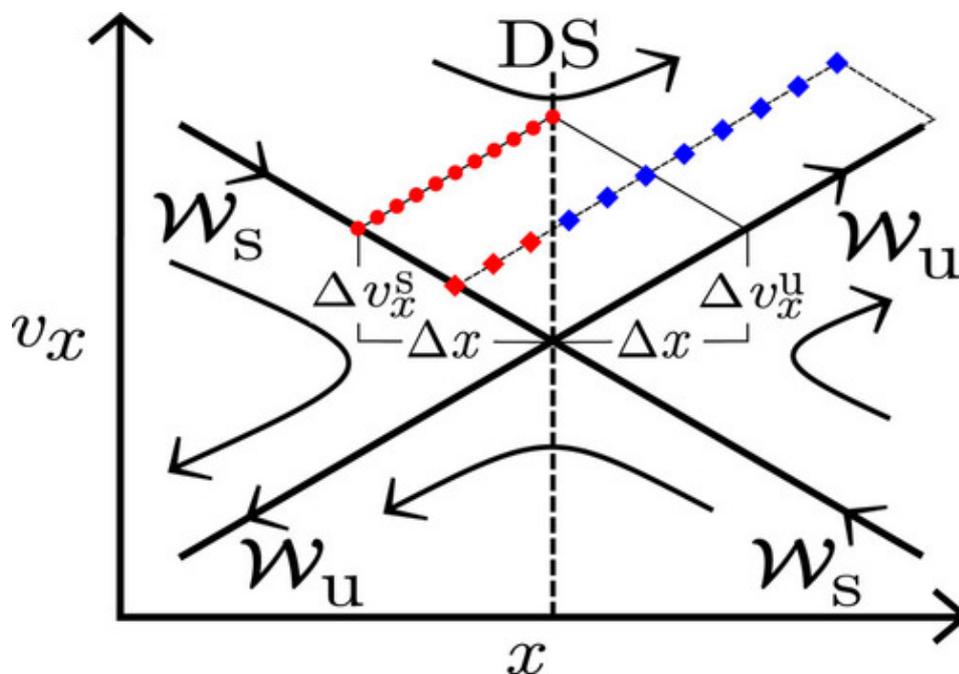


Figure 2.4: Sketch illustrating the local manifold analysis [9].

manifold related to the NHIM.

LMA can be best explained with the help of Fig. 2.4. Consider an initial ensemble of trajectories in the local vicinity of the stable and unstable manifolds of an arbitrary point on the NHIM sketched by the red circles in Fig. 2.4. As already mentioned the basic thought is to utilize the linearized dynamics in the local vicinity of the stable and unstable manifold to propagate the whole line segment. When moving from t to $t + \Delta t$, the linearized dynamics can be separated in two parts. A compression toward the NHIM in the direction of the stable manifold by the factor $\Delta x^s(t + \Delta t)/\Delta x^s(t)$ pulls the line closer to the unstable manifold and a stretching in the direction of the unstable manifold by the factor $\Delta x^u(t + \Delta t)/\Delta x^u(t)$ lengthens the line segment. The result of this motion is shown by the red and blue diamonds in Fig. 2.4. With simple geometry, we can now determine the ratio between the length of the line segment left of the DS (red diamonds in Fig. 2.4) and the length of the entire line. Due to the properties of the linear mapping that occurred, we know that the density of reactive trajectories along the line is constant. Thus, the obtained ratio of line segments is proportional to the number of reactants $N_{react}(t)$. By choosing $\Delta x^s(t) = \Delta x^u(t) \equiv \Delta x$, the number of reactants at time $t + \Delta t$ is given by

$$N_{react}(t + \Delta t) = \frac{\Delta x^s(t + \Delta t)}{\Delta x^u(t + \Delta t)} N_{react}(t) \quad (2.2)$$

It is important to note that the stable and unstable manifolds shown in Fig. 2.4 can

independently rotate around their intersection point as functions of time. However, Eq. (2.2) stays valid in these cases. One can always separate the linear mapping that occurs into two mappings. First, one that transforms the parallelogram as shown in Fig. 2.4, such that these rotations are accounted for, all the while keeping $\Delta x^u(t)$ and $\Delta x^s(t)$ constant, which in turn ensures that the line segment does not cross further into the DS before the next step. Subsequently, we can perform the stretching and compression of the appropriate lengths, as discussed before, to complete the mapping of the linearized dynamics.

Using the equations of motion, and also writing the dependencies on the bath coordinates y and velocities v_y , Eq. (2.2) can be written as

$$k(y, v_y, t) = \frac{\Delta v_x^u}{\Delta x^u}(y, v_y, t) - \frac{\Delta v_x^s}{\Delta x^s}(y, v_y, t) \quad (2.3)$$

This shows that for any point on the NHIM parameterized by the bath coordinates y and velocities v_y , the instantaneous rate $k(y, v_y, t)$ is simply given by the difference of the slopes of the stable and unstable manifolds in the (x, v_x) diagram shown in Fig. 2.4.

3 Neural Networks

Neural networks are a set of algorithms, which are based on the human brain, that are designed to remember the patterns. The fact that neural Networks can be trained to learn any arbitrary nonlinear input/output relationships from corresponding data and the acquired knowledge has resulted in their use in several of areas. The NN is a computational tool inspired by the network of neurons in the biological nervous system. It is a network consisting of arrays of artificial neurons linked together with different weights of connection. The states of the neurons as well as the weights of connections among them evolve according to certain learning rules. Practically speaking, neural networks are nonlinear statistical modeling tools that can be used to find the relationship between input and output or to find patterns in vast database. NN has been applied in statistical model development, adaptive control system, pattern recognition in data mining, and decision making under uncertainty [10].

The proposal of neural networks began foreseeably as an imitation of how neurons function in the brain, termed “connectionism” and connected circuits were used to simulate intelligent behaviour. In 1943, neurophysiologist Warren McCulloch and mathematician Walter Pitts demonstrated with a simple electrical circuit [11] a Boolean logic to demonstrate neural behaviour. The McCulloch–Pitts concept of a logical neural network has been described as a landmark event in the history of cybernetics. Based on the “all-or-none” character of nervous activity, they constructed a Boolean logic to describe neural events and their relations. The “all-or-none” concept led McCulloch and Pitts to idealize neurons as on off devices, they either fired or they did not. Connecting this to the “true–false” nature of propositions in logic, McCulloch and Pitts constructed hypothetical networks of excitatory and inhibitory neurons, with varying patterns of connection, and demonstrated an isomorphism with these hypothetical arrangements of neurons and the logic of propositions [11]. Donald Hebb expanded the idea on another level in his book, *The Organization of Behaviour* (1949), proposing that neural pathways strengthen over each successive use, especially between neurons that tend to fire at the same time thus beginning the long journey towards quantifying the complex processes of the brain [12].

Neural networks can be hardware- (neurons are represented by physical components) or software-based (computer models), and can use a variety of topologies and learning algorithms. Hardware-based NN, also known as physical neural network in which an

electrically adjustable material is used to emulate the function of a neural synapse. “Physical neural network” is used to emphasize the reliance on physical hardware used to emulate neurons as opposed to software-based approaches which simulate neural networks [13].

We will discuss briefly three important types of neural networks that form the basis for most pre-trained models in deep learning:

- Artificial Neural Networks (ANN)
- Convolution Neural Networks (CNN)
- Recurrent Neural Networks (RNN)

Let’s introduce each neural network in briefly. A single perceptron (or neuron) can be imagined as a Logistic Regression. Artificial Neural Network, or ANN, is a group of multiple perceptrons/neurons at each layer.

ANN is also known as a feed forward NN because inputs are processed only in the forward direction. In a feed forward network information always moves in one direction, it never goes backwards. It is the simplest form of NN. We are going to use a feed forward NN for the prediction of k .

A CNN is a type of NN most commonly applied to analyzing visual imagery. CNN architecture is influenced by the organization and functionality of the visual cortex and designed to immitate the connectivity pattern of neurons within the human/animal brain. The neurons within a CNN are split into a three-dimensional structure, with each set of neurons analyzing a small region or feature of the image. CNNs use the predictions from the layers to produce a final output that presents a vector of probability scores to represent the likelihood that a specific feature belongs to a certain class [14]. Applications of convolution neural networks include: image recognition [15], video analysis [16], drug discovery [17].

RNN has a recurrent connection on the hidden (internal) states. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. RNN can mathematically be described as

$$h_t = f_w(h_{t-1}, x_t) \tag{3.1}$$

where h_t is the new state, f_w is some function with parameters w , h_{t-1} is the old state, x_t is input vector at some time step. The state consists of a single ‘hidden’ vector h . The new state can also be written as

$$h_t = \tanh(w_{hh}h_{t-1} + w_{xh}x_t) \quad (3.2)$$

where w_{hh} is the weight matrix.

Applications of recurrent neural networks include: Machine Translation [18], robot control [19], time series prediction [20, 21], speech recognition [22], speech synthesis [23], time series anomaly detection [24], rhythm learning [25], music composition [26], grammar learning [27], handwriting recognition [28], human action recognition [29], protein Homology Detection [30], predicting subcellular localization of proteins [31], several prediction tasks in the area of business process management [32], and prediction in medical care pathways [33].

3.1 Feedforward Neural Network

Taking reference from [34] [35], let us discuss a little more about the feedforward neural network, as this is going to be useful in developing our model. As we can see Fig. 3.1, we can give any number of inputs and any number of outputs can be obtained from a NN. The accuracy of the results depends upon the hidden layers.

The most fundamental component of a NN is the neuron which is responsible for the information processing. An artificial neuron (also referred to as a perceptron) is a mathematical function. It takes one or more inputs that are multiplied by values called weights and added together. This value is then passed to a non-linear function, known as an activation function, to become the neuron's output. This is illustrated in Fig. 3.2. The input channels are denoted as x_j and they have a different relevance in respect to each other for the training a neuron. This is expressed by weighting the inputs with weights w_{ij} . Input information for a neuron i can be expressed as

$$\sum_{j=1}^N w_{ij}x_j, \quad (3.3)$$

where N is the number of all input channels and w_{ij} is the weight of the connection between channel j and neuron i . The input channels may be output values of other neurons. The value of the weights are a measure of how important the incoming information is. In respect to biological neurons which need to reach a threshold before being active, one introduces an offset b_i which is called bias and determines the area of

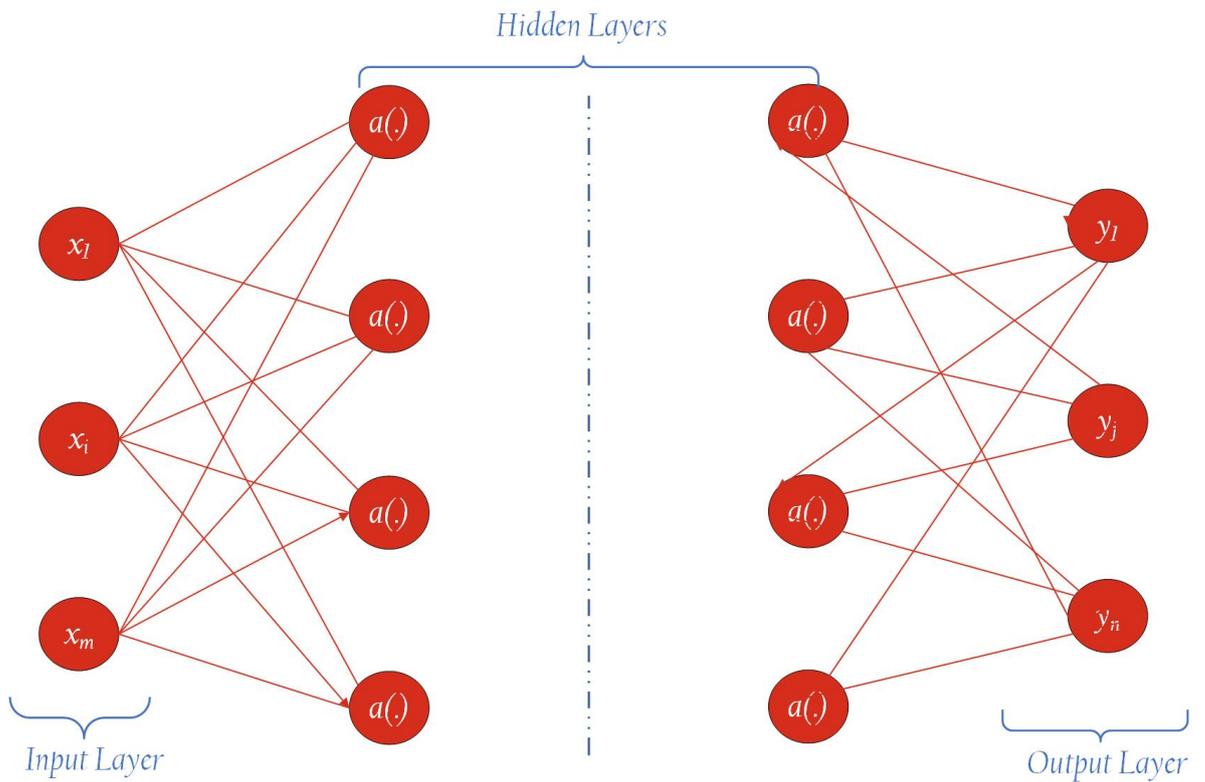


Figure 3.1: Sketch of a feedforward neural network. There can be any number of inputs, outputs and hidden layers. Depending upon the inputs and outputs, a forwardfeed network is developed. We can assign different number of hidden layers and activation factors.

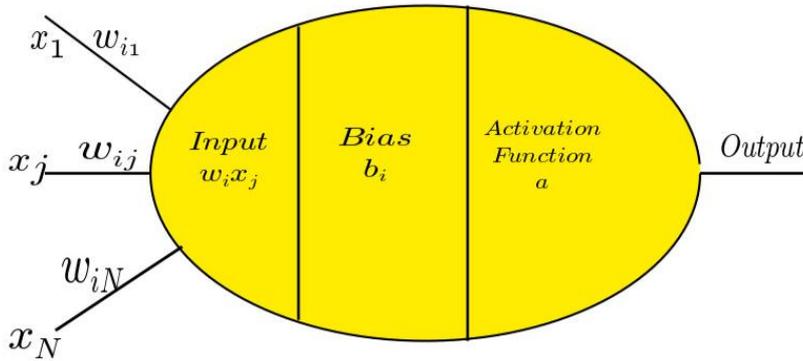


Figure 3.2: Sketch of a neuron. The x_j represent the input channels, could be an output of a previous neuron. w_{ij} is weighted connection. The bias b_i is summed with the weighted inputs to form net inputs. The output (input for the next neuron) of a neuron can range from $-\infty$ to $+\infty$. The neuron doesn't know the boundary. So we need a mapping mechanism between the input and output of the neuron. This mechanism of mapping inputs to output is known as activation function, a . The output is the sum of the weighted inputs and biases, with an influence of the activation function.

activation. A bias can also be seen as an individual neuron or input channel which has a constant value. This results to the weighted input

$$\Sigma_i = \left(\sum_{j=1}^N w_{ij} x_j \right) + b_i. \quad (3.4)$$

To this point, this would only allow a linear mapping between input and output of a neuron. Hence, an activation function, $a(\cdot)$, is introduced that is of great importance for the functionality and learning behavior of the neural network. An activation function is a non-linear function applied by a neuron to introduce non-linear properties in the network. The activation function processes the weighted input to the output y_j

$$y_i = a(\Sigma_i) = a \left(\sum_{j=1}^N w_{ij} x_j + b_i \right). \quad (3.5)$$

The choice of the activation function is crucial and depends on the task of the neural network. The activation function is, in general, nonlinear. However, sometimes a linear activation is favorable, for example in the input and output layer.

Let us quickly have a look at the different types of activation functions.

1. Linear activation function: It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. In one sense, a

linear function is better than a step function because it allows multiple outputs, not just yes and no. It can be expressed as

$$f(z) = az. \tag{3.6}$$

Output values ranges from $-\infty$ to ∞ . However, a linear activation function has a major problem. It is not possible to do backpropagation. So it's not possible to go back and understand which weights in the input neurons can provide a better prediction.

2. Sigmoid Activation Function: It has a smooth gradient which prevents jumps in output values. Output values bound between 0 and 1, normalizing the output of each neuron. It can be expressed as

$$\text{sigmoid}(z) = \frac{1}{1 + e^{-z}}. \tag{3.7}$$

3. TanH/Hyperbolic Tangent: It is like a sigmoid function, but zero centered, making it easier to model inputs that have strongly negative, neutral, and strongly positive values. It can be expressed as

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{3.8}$$

4. Rectified linear unit activation function: Also known as ReLU. It is computationally efficient as it allows the network to converge very quickly. Although it looks like a linear function, ReLU has a derivative function and allows for backpropagation, and is non-linear. Output values bound between 0 and ∞ . It can be expressed as

$$\text{relu}(z) = \max(0, z). \tag{3.9}$$

When inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

5. Leaky ReLU: It prevents dying ReLU problem. This variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values. It can be expressed as

$$\text{leakyrelu}(z) = \begin{cases} 0.001z & \text{for } z < 0 \\ z & \text{for } z \geq 0. \end{cases} \tag{3.10}$$

Output values ranges between $-\infty$ and ∞ . The disadvantage of leakyrelu is that it does not provide consistent predictions for negative input values.

6. Softmax activation function: This activation function is able to handle multiple classes. It normalizes the outputs for each class between 0 and 1, and divides by their sum, giving the probability of the input value being in a specific class. Typically Softmax is used only for the output layer, for neural networks that need to classify inputs into multiple categories. It can be expressed as

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \quad (3.11)$$

3.2 Layers

Considering to solve the task, it is not sufficient for a single neuron to give the output with high accuracy. Thus, a neural network consists of many neurons. In feedforward neural networks, the type of neural network we will be using, several neurons are gathered in a layer and the information passes from one layer to the next layer. Therefore, each neuron is connected to all neurons of the former and next layer. These yield the input arguments and the neural network output. The input layer represents the input of the user or task-dependent variables. Hence, in most cases the input layer has an activation function as

$$a(x) = x. \quad (3.12)$$

Combining all the neurons of a layer, Eq. (3.4) can be written with tensors

$$\sum^{(l)} = w^{(l)} a^{(l-1)} + b^{(l)}. \quad (3.13)$$

Where $\sum^{(l)}$, $a^{(l-1)}$ and $b^{(l)}$ are vectors and $w^{(l)}$ is a matrix. l refers to the current layer. $l - 1$ refer to the previous layer. Recollecting the activation function from Eq. (3.5), we can write it as

$$(a^{(l-1)})_i = a_i^{(l-1)} = y_i^{(l-1)}. \quad (3.14)$$

And the overall output of the neural network can be expressed as

$$a^{(k)} a^{(k-1)} \dots a^{(1)}, \quad (3.15)$$

where k is the output layer and 1 refers to the input layer. Every layer which is not an input or output layer is called a hidden layer. Further, the phrase deep learning refers among other things to neural networks with several hidden layers. The number of neurons in a specific layer and also the number of layers are hyper-parameters of the neural network. The choice of these numbers is an educated guess.

3.3 Training of a Neural Network

To achieve the expected behavior of a neural network, one has to adjust its parameters, i.e. weights and biases, in the correct way. Therefore, one defines a cost function that satisfies the relation

$$\lim_{(y, \tilde{y})} C(y, \tilde{y}) \rightarrow 0, \quad (3.16)$$

where y is the expected value, \tilde{y} is the actual/expected output of the NN and C is the cost function, which tells the accuracy of a NN.

To train a NN optimally, one should try to reduce this cost function, which means that the difference in y and \tilde{y} should be minimal. Therefore, the cost function has to be minimized too. For the minimization one can use the gradient descend method. A possible way to define a cost function is

$$C(y, \tilde{y}) = \frac{1}{2n} \sum_{i=1}^n |y_i - \tilde{y}x_i|^2, \quad (3.17)$$

where x_i is an input vector of one point in the training data in respect to the expected output y_i and n is the total number of training points. The function $\tilde{y}(x_i)$ is the output of the neural network for the given input x_i .

3.3.1 Optimization of a Neural Network

The learning of the NN is basically to aim $C = 0$ or minimize C as small as possible, by influencing the free parameters of the NN, i.e the weights and biases, see Eq. (3.4). To have a better understanding we will rewrite this equation as

$$C = C(w, b). \quad (3.18)$$

Where w is all weights and b is all biases. To train a NN, the weights and biases are varied to obtain maximum accuracy. This method of changing and manipulating the weights and the biases is known as “Optimization”. Optimizers are algorithms or methods used to change the attributes of a NN such as weights and learning rate in order to reduce the losses.

There are a number of different optimizers available. But we will be discussing only the Adam optimizer [36] briefly. It might be good to read further about the other optimizers like gradient descent [37], Adagrad [38], Adadelta [39], RMSprop [40] etc.

Adaptive Moment Estimation, also known as *Adam optimizer* works with momentums of first and second order. The intuition behind the Adam is that we don't want to roll so

fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients, it also keeps an exponentially decaying average of past gradients $M(t)$.

$m(t)$ and $v(t)$ are values of the first moment which is the mean and the second moment which is the uncentered variance of the gradients respectively

$$\dot{m}_t = \frac{m_t}{1 - \beta_1^t}; \quad \dot{v}_t = \frac{v_t}{1 - \beta_2^t},$$

β_1 and β_2 are the new introduced hyper-parameters of the algorithm. They have really good default values of 0.9 and 0.999 respectively. Almost no one ever changes these values.

The advantages of Adam optimizer is that it is quite fast and converges rapidly. Adam was designed to combine the advantages of Adagrad, which works well with sparse gradients, and RMSprop, which works well in on-line settings. Having both of these enables us to use Adam for broader range of tasks. Adam can also be looked at as the combination of RMSprop and SGD with momentum.

3.3.2 Loss functions for a Neural Network

The loss function is one of the important components of the NN. Loss is nothing but a prediction error of NN. And the method to calculate the loss is called loss function. In simple words, the Loss is used to calculate the gradients. And gradients are used to update the weights of the NN. This is how a NN is trained.

Keras and Tensorflow (APIs for machine learning) have various inbuilt loss functions for different objectives. There are a number of loss functions. Let's discuss some of the loss functions used extensively.

Mean Squared Error (MSE) loss is used for regression tasks. As the name suggests, this loss is calculated by taking the mean of squared differences between actual (target) and predicted values.

Binary Crossentropy (BCE) loss is used for the binary classification tasks. If we are using BCE loss function, we just need one output node to classify the data into two classes. The output value should be passed through a sigmoid activation function and the range of output is between 0 and 1.

Categorical Crossentropy (CCE) It is majorly used in multi-class classification task. While using CCE loss function, there must be the same number of output nodes as the classes. And the final layer output should be passed through a softmax activation so that each node output has a probability value between 0 and 1.

Further reading of the other types of loss functions is recommended from Refs. [41], [42] and [43].

3.4 TensorFlow

For the computations we resort to TensorFlow, which is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks. TensorFlow is a symbolic math library based on dataflow and differentiable programming. It was developed by the Google Brain team for internal Google use. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices [44].

TensorFlow enables you to build dataflow graphs and structures to define how data moves through a graph by taking inputs as a multi-dimensional array called Tensor. It allows you to construct a flowchart of operations that can be performed on these inputs, which goes at one end and comes at the other end as output.

Tensorflow's architecture works in three parts: 1. Preprocessing the data, 2. Build the model and 3. Train and estimate the model.

It is called TensorFlow because it takes input as a multi-dimensional array, also known as tensors. You can construct a sort of flowchart of operations that you want to perform on that input. The input goes in and flows through this system of multiple operations and comes out as output. This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out on as results.

TensorFlow software package [45] used in our thesis for the prediction of k is TensorFlow 2. There are so many possibilities to apply it in various problems involving deep learning NN. Further reading is recommended from Refs. [41, 46, 47].

4 Results

4.1 Prediction of Instantaneous Rates by Neural Networks

As we have already discussed in chapter 2, the instantaneous decay rates of a chemical reaction is quite useful in different applications and researches. We have been calculating k using the binary contraction method, which involves a lot a time and effort. Since science has always strived for easier solution for any issue, therefore, we are also aiming to develop a simple model which can predict the decay rates with an easy input. As also discussed in chapter 3, a neural network can predict a desired output by providing suitable inputs. It has already been shown that the NN can be used to approximate the NHIM [48]. We want to investigate if the NN can predict k with good precision. We want decay rates as output, but we should also be aware of the inputs to be given. When we are computing k manually, we should already know the numerical position of the NHIM. We are aiming to obtain the numerical value of time-dependent k , we must know the bath coordinates (y, v_y) at a given time t . Therefore, we would be taking y and v_y as the two inputs to the NN, and since we are examining time-dependent instantaneous decay rates, the third input must be time t .

As discussed in section 3, we have to develop a simple regression model of neural network to predict the instantaneous rates k . The NN model takes three inputs t , y and v_y and gives one output k as already mentioned in the above discussion, see Fig. 4.1.

A key requirement for a neural network is its training data. We already know that the neural network requires a good amount of data for training so that it can predict the result accurately. Another important ingredient of a neural network is its validation data. Because, for large datasets the ideal ratio between training set and validation set should be 80:20 to 90:10. We took 100000 random samples of a Gaussian distribution to be as the training set. For the validation set, we took 10000 samples of the Gaussian distribution.

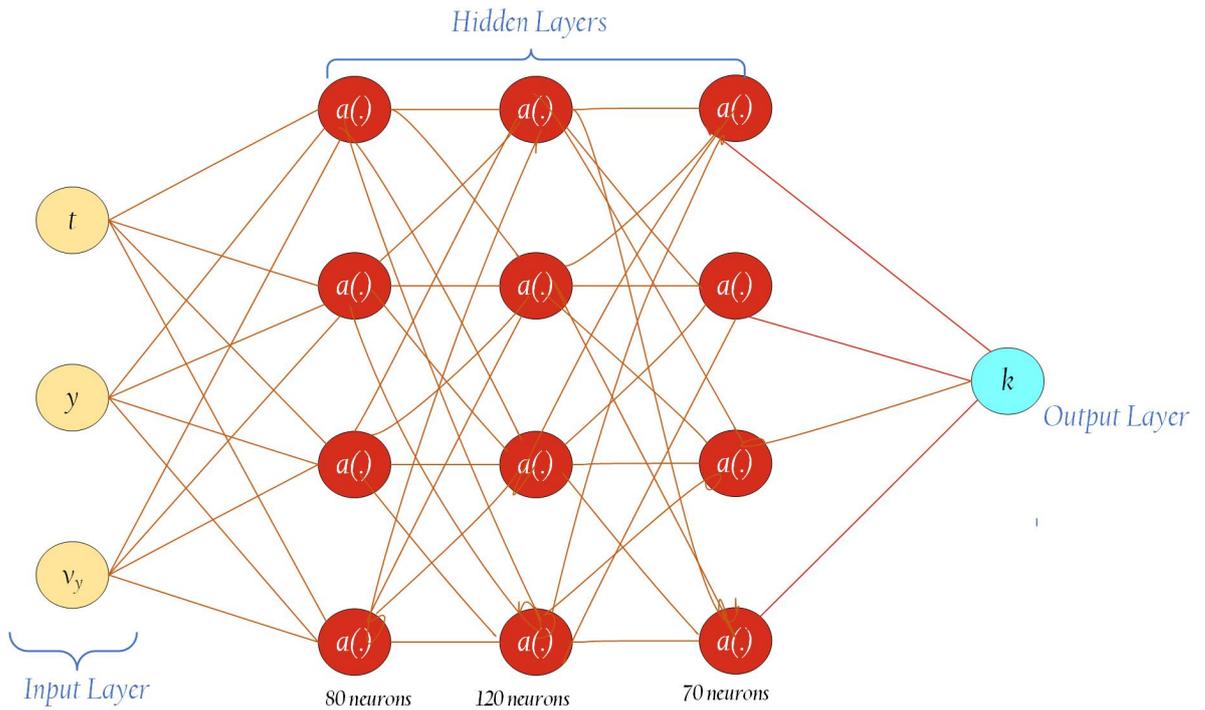


Figure 4.1: Sketch of the forwardfeed neural network used in our investigation. The NN is designed to continuously approximate k for the time-dependent NHIM. The network consists of three hidden layers, each containing various neurons, 80, 120, and 70 respectively. Here, the input to the first layer, given by the orthogonal mode y , the associated velocity v_y , and time t , is processed from layer to layer towards the output layer. Each neuron is connected to all neurons of a previous layer.

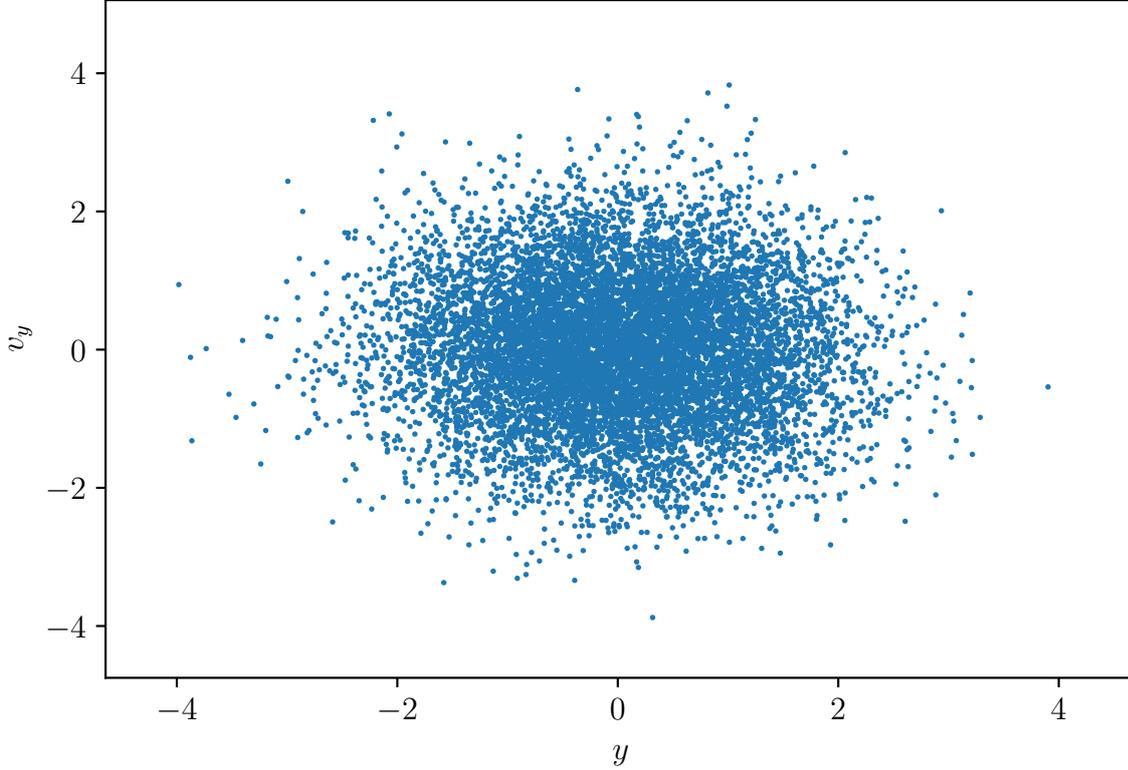


Figure 4.2: Subset of 100000 random samples of y and v_y , generated using the Gaussian distribution. The points are taken in the range from -5 to 5 to cover the whole range of the trajectory. The values of y and v_y along with time t are fed in the neural network as inputs for the training of the neural networks.

In probability theory, a Gaussian distribution is a type of continuous probability distribution for a real-valued random variable. The general form of its probability density function is

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}, \quad (4.1)$$

where μ is the mean of the distribution and σ is the standard deviation.

The Gaussian distribution is used because we are most interested in low-energy trajectories, i.e. those near the origin; the Gaussian distribution ensures that they can be predicted with a relatively high accuracy while not wasting too much resources on the outer regions. We have used $\mu = 0$ and $\sigma = 1$. We have limited the values of y and v_y as $-5 \leq y, v_y \leq 5$, as can be seen in Fig. 4.2, because samples too far out can lead to numerical problems, e.g., the BCM might not converge anymore without a human tweaking the BCM's

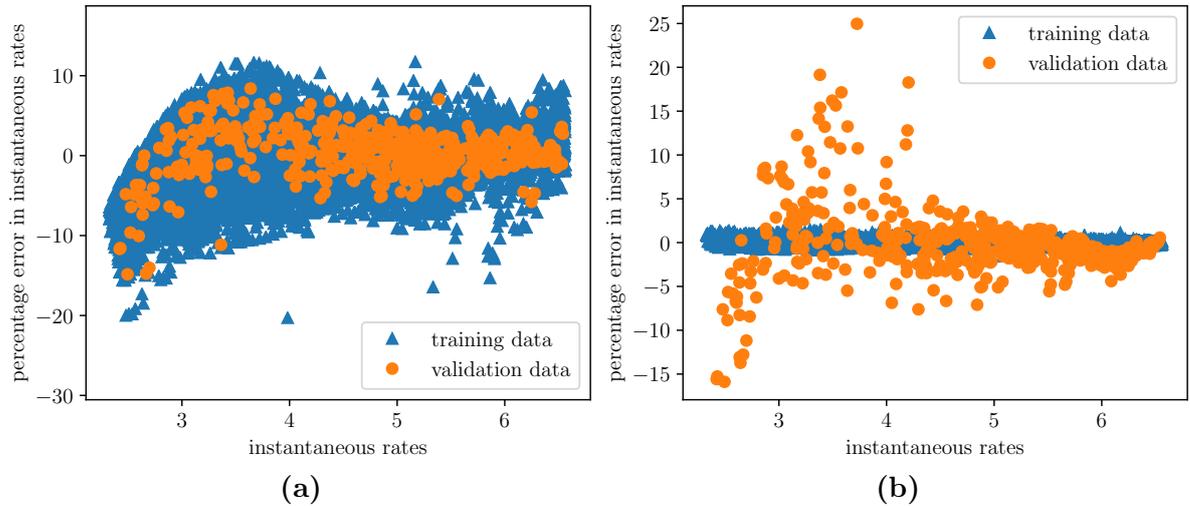


Figure 4.3: (a) Error in the prediction of k when the number of hidden layer is only 1. This depicts the underfitting of the NN model with very high error because of the neurons not optimally able to gather and learn from the training data. (b) Percentage error in the prediction of k when the NN model has 22 layers. The error in prediction of training data is very small, but the error in prediction of validation data is very large. This shows that the model is overfitting and learning even the noise along with the training data, thus effecting the result negatively.

parameters. We first chose a number of time points t per period and then generated a separate random sample of (y, v_y) tuples per time point t .

We tried to see the effect of number of layers on the results and found out that after reaching the optimal number of layers, which in this case is 3, the error tends to increase for the overfitting or overtraining. Similarly, reducing the layers also meant that the network is not training enough to predict the desired results. As can be seen in Fig. 4.3a, the NN can neither model the training data nor generalize the new data and therefore gives really bad results. The number of epochs remains the same as 10 but we have reduced the number of hidden layers to 1 with 80 neurons. This erroneous result is because of the NN model is not training enough and this is known as underfitting. Similarly, we can observe overfitting in Fig. 4.3b. We ran the model for 22 epochs and observed that the training data is learnt very well with error in prediction to be very minimal (within 2%) but the error in prediction of k for validation data is very large (more than 20%). The NN models the training data too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

As one of the factor on which the accuracy depends is on the number of layers of a

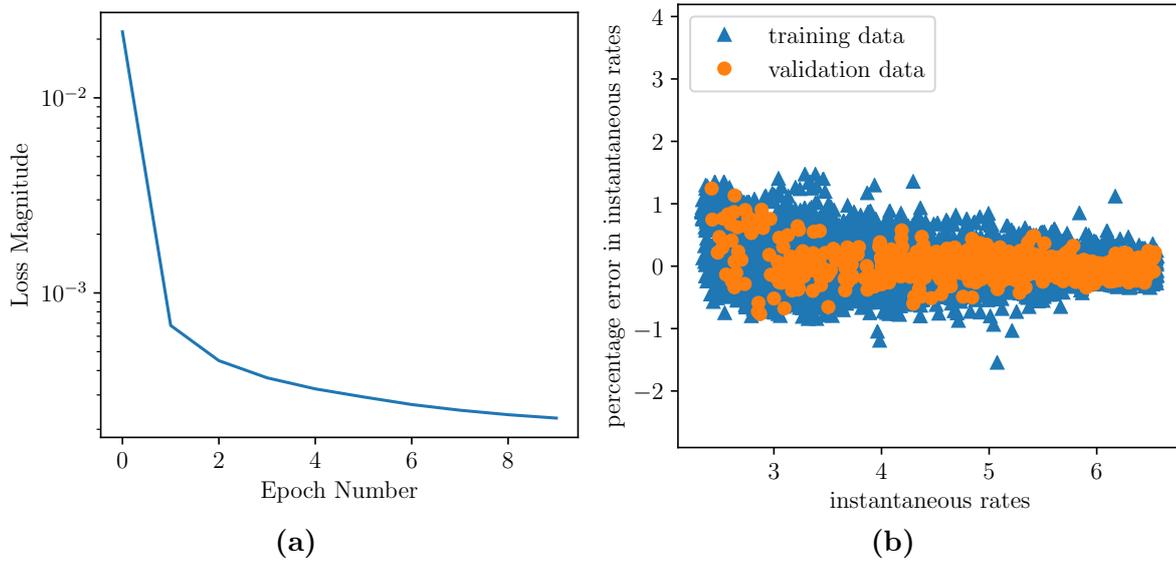


Figure 4.4: (a) Loss of data per epoch for predicting the instantaneous rates k for random samples. (b) Percentage error in prediction of decay rates, k by neural networks for 10 epochs for random samples as the validation data. The error in the prediction of training data is more than the error in the prediction of validation data because the number of samples for the training data is way larger than the validation samples.

NN, another significant factor is the number of neurons per layer. How increasing and decreasing the number of neurons per layer can alter the prediction of the neural network can also be a different topic of discuss. For our NN model we have taken 3 hidden layers with 80, 120 and 70 neurons respectively for the consequent three layers, see Fig. 4.1.

We have already mentioned different optimizers in section 3.3.1. The optimizer we are using in our NN model is Adam optimizer [36] because it nearly always works better (faster and more reliably reaching a global minimum) when minimising the cost function in training neural nets. Let's discuss the adam configuration parameters used in our model in breif. α : also referred to as the learning rate or step size. We have used 0.002 as the learning rate. β_1 : the exponential decay rate for the first moment estimates is 0.9. β_2 : the exponential decay rate for the second-moment estimates is 0.999. We have used the mean squared error for the loss in our model. As the name suggests, it computes the mean of squares of errors between expected results and predictions. See section 3.3.2 for more details on types of losses in NN.

As we can see in the Fig. 4.4b, the neural network can predict good values with very small error (within 2%). The neural network has been trained optimally to produce such results. To develop our model, we have first tried to get the prediction results for random samples with the same parameters as that of the training data of Fig. 4.2. The

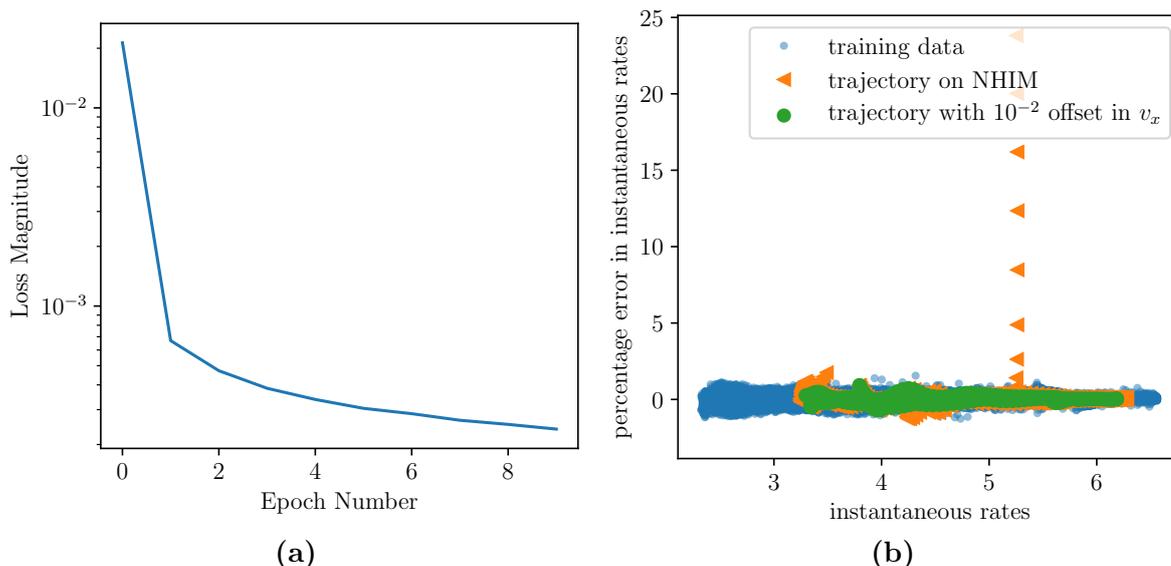


Figure 4.5: (a) Loss of data per epoch for predicting the instantaneous rates k for trajectories as validation data for 10 epochs. (b) Error in percentage in the prediction of decay rates, k by the NN for 10 epochs for the two trajectories. There is some quite large error for a little part of the trajectory on the NHIM. This is due to overfitting or overtraining of the data. The NN is learning even the noise present in the training data leading to the large error for a part of the trajectory on the NHIM.

initial conditions and parameters for both the training data and the test data are taken to be the same to obtain desired results. We can see in Fig: 4.4a, to reach these desired results, we ran the NN for 10 epochs.

4.2 Usefulness for Reaction Rates of Arbitrary Ensembles

Since, in the recent past, there has been some research involving decay rates in transition state theory [49–51], it is much useful that the model developed in the previous section 4.1 is tested for the prediction of k of trajectories.

The same regression model of the NN, see Fig. 4.1, as described in the above section 4.1, where the orthogonal mode y , its respective velocity v_y at a given time point t are the three inputs, has been used for the prediction of the instantaneous decay rates k for trajectories. As we can see in Fig. 4.5b, for the same number of epochs i.e. 10 (Fig. 4.5a), our model tends to overfit when the inputs are trajectories. The error in prediction

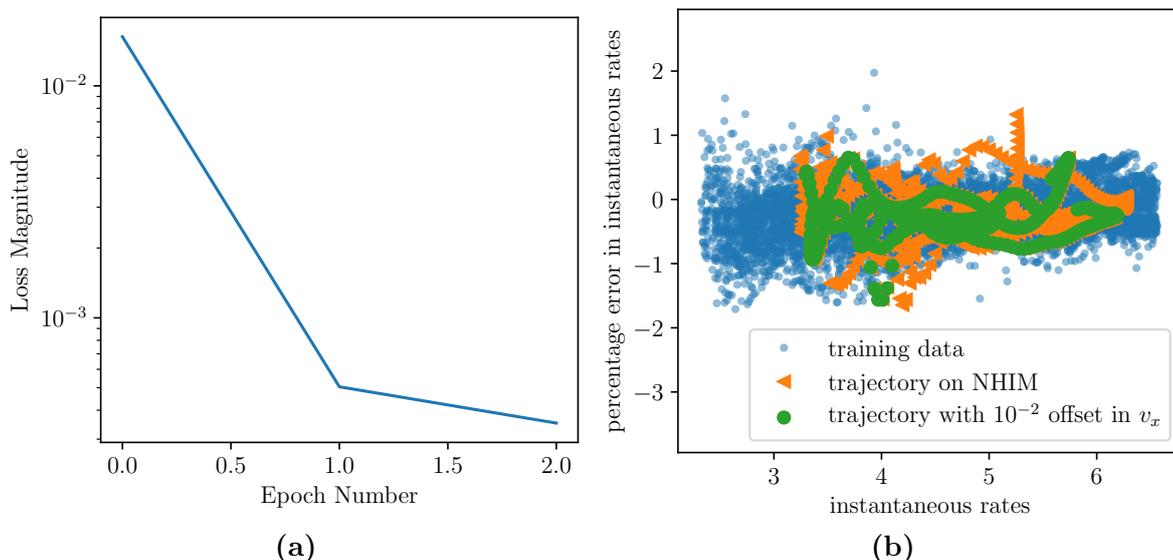


Figure 4.6: (a) Loss of data per epoch for predicting the instantaneous rates k for trajectories as validation data for 3 epochs. (b) Percentage error in prediction of decay rates, k by neural networks for 3 epochs for two trajectories, one on the NHIM and other with a slight offset of 10^{-2} in v_x . Error in prediction is quite small (within 2%) exhibiting that the NN designed is sufficient for the approximation of k .

for the training data is very small, but for some part of the trajectory on the NHIM, the error in prediction of k is very large. As can be seen in Fig. 4.6a, the number of epochs required to get more accurate results is by reducing the epochs to 3. And the result obtained is with a good amount of accuracy or with very little 1% to 2% error in prediction, as shown in Fig. 4.6b.

To evaluate the usefulness of this model of neural network, we have two trajectories as the input for the prediction of k . One trajectory is on the NHIM and the other with a slight offset of 10^{-2} in v_x to the same trajectory which remains close to the NHIM for a short time and then moves away. The two trajectories can be seen in Fig. 4.7a. In this scenario, we are looking at reactive particles. They start on the reactant side, react over the saddle, and end up on the product side. They are closest to the NHIM when they cross the dividing surface. Only then can they be influenced significantly by the dynamics on the NHIM. The decay rates that we calculated (e.g. using the LMA) are derived from these dynamics on the NHIM. The purpose of the calculation in this section is to get an effective decay rate for trajectories close to the NHIM by averaging the decay rate corresponding to the dynamics on the NHIM. The hard cutoff at $\Delta x > 0.1$ is just a simple model and can be improved in the future. And hence, we have sliced the trajectory, as shown in Fig. 4.7b.

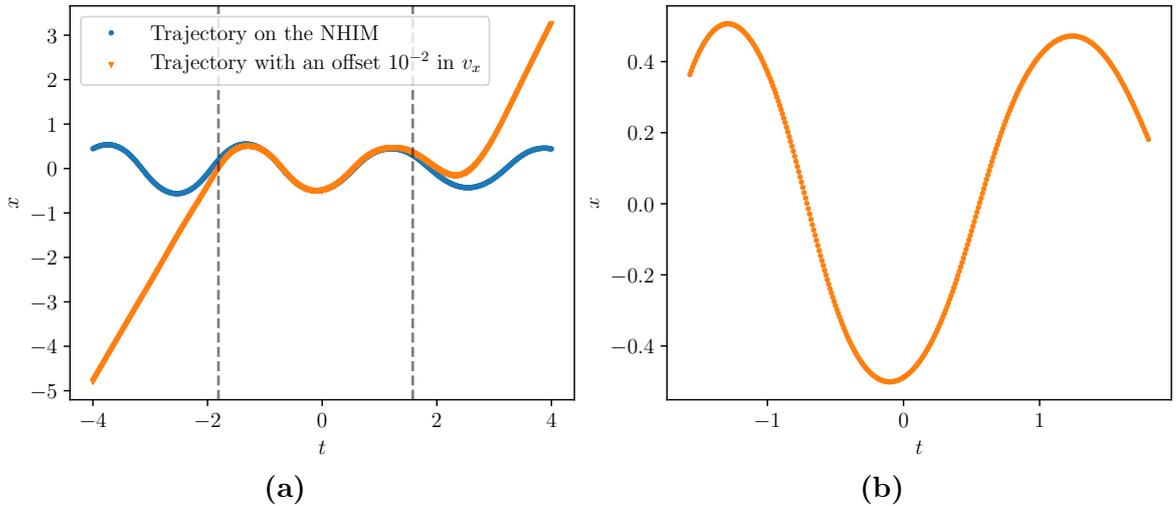


Figure 4.7: (a) Comparing the two trajectories, one on the NHIM and the other with an offset of 10^{-2} in v_x . We are interested in the trajectories not very far away from the NHIM with a maximum difference of 0.1 in x , which has been marked by the dashed lines on both the sides. Therefore we will be using only the sliced trajectory for predicting k . (b) Sliced trajectory with an offset of 10^{-2} in v_x . The purpose of slicing the trajectory is to get an effective decay rate for trajectories close to the NHIM by averaging the decay rates corresponding to the dynamics on the NHIM.

The error in the prediction of k has been determined by projecting the trajectory onto the NHIM via the BCM by using our classical method LMA. As can be seen in Fig. 4.6b, the time-dependent instantaneous rates for both the trajectories have been predicted with very small error, confirming the efficiency of the NN.

The quantitative relation of decay rates with time can be seen in Fig. 4.8. The average instantaneous rate for the trajectory on the NHIM $k_{\text{avg}}^{\text{NHIM}}$ as calculated by averaging the predicted values $k_{\text{prediction}}^{\text{NHIM}}$ is 4.78 see Fig. 4.8a. Also the average instantaneous rate for the trajectory with 10^{-2} offset for v_x , $k_{\text{avg}}^{\text{offset}}$ is 4.79 see Fig. 4.8b. The predicted values of k within the vicinity of the NHIM are useful for averaging an ensemble as well.

However, the result (prediction of k for trajectories near the NHIM) may change for a different system. We need to develop different NN for different systems.

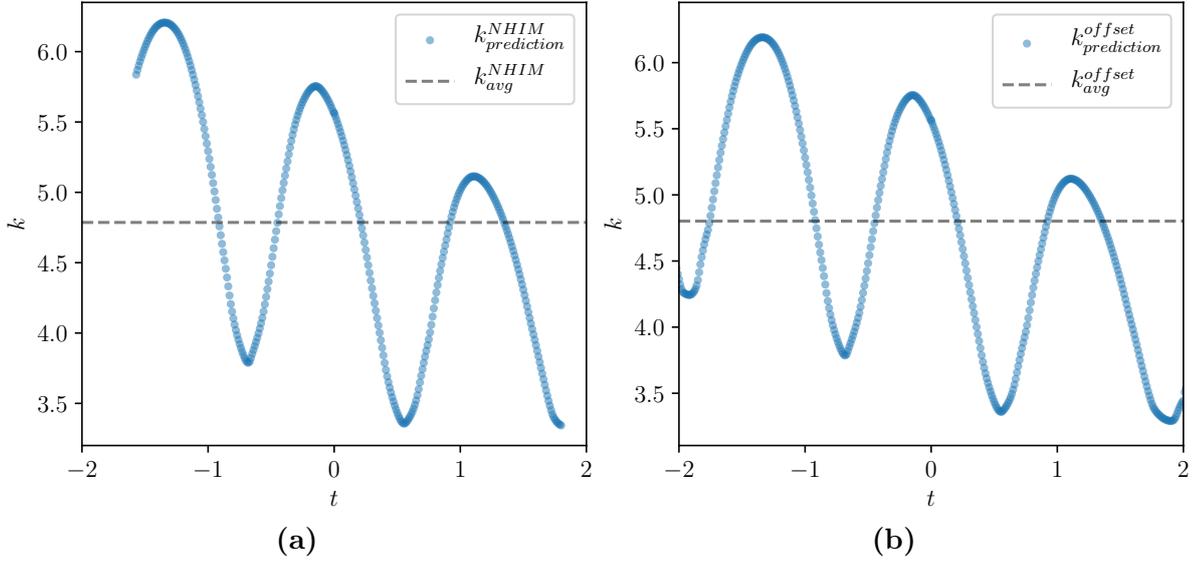


Figure 4.8: (a) Relation of decay rates $k_{prediction}^{NHIM}$ predicted by the NN with time t for a trajectory on the NHIM within the range from -2 to 2. The dashed line is the average of the predicted decay rates for a trajectory on the NHIM k_{avg}^{NHIM} is 4.78. (b) Relation of decay rates $k_{prediction}^{offset}$ predicted by the NN with time t for a trajectory with an offset of 10^{-2} in v_x within the range from -2 to 2. The dashed line is the average of the predicted decay rates for a trajectory with an offset of 10^{-2} in v_x k_{avg}^{offset} at 4.79.

5 Summary and Outlook

In this thesis, a simple feedforward neural network is designed and investigated for its usefulness in the prediction of decay rates.

The NN model developed is not too complex, rather a simple one with only 3 hidden layers and only 3 inputs, which are orthogonal mode y , the respective velocity v_y at a given time point t . The decay rates k predicted by the NN have a very small error (within 2%).

The dependence of the results on increasing and decreasing the number of layers influencing the result has also been investigated in this thesis. One more parameter which has been examined is the change in the number of epochs, letting us identify overfitting and underfitting.

The NN predicts decay rates with large accuracy for both a trajectory whether on the NHIM or with a slight offset in v_x , and also for random samples.

It was shown that neural networks are suitable for the prediction of decay rates in reaction dynamics. With much less effort to calculate the NHIM using the BCM and k using the classical LMA, the NN was able to predict the result with very high accuracy. In future applications the method may be applied for an efficient computation of average rates of large trajectory ensembles.

Bibliography

- [1] Toppr. *Effect of Temperature on Rate of Reactions*. URL: <https://www.toppr.com/ask/en-in/content/concept/effect-of-temperature-on-rate-of-reactions-203361/> (visited on 02/09/2021).
- [2] F. Revuelta, Thomas Bartsch, P. L. Garcia-Muller, Rigoberto Hernandez, R. M. Benito, and F. Borondo. “Transition state theory for solvated reactions beyond recrossing-free dividing surfaces”. In: *Phys. Rev. E* 93 (6 June 2016), p. 062304. DOI: [10.1103/PhysRevE.93.062304](https://doi.org/10.1103/PhysRevE.93.062304). URL: <https://link.aps.org/doi/10.1103/PhysRevE.93.062304>.
- [3] Baron Peters. “Chapter 18 - Diffusion over barriers”. In: *Reaction Rate Theory and Rare Events Simulations*. Ed. by Baron Peters. Amsterdam: Elsevier, 2017, pp. 473–506. ISBN: 978-0-444-56349-1. DOI: <https://doi.org/10.1016/B978-0-44-456349-1.00018-0>. URL: <http://www.sciencedirect.com/science/article/pii/B9780444563491000180>.
- [4] Philip Pechukas. “Transition state theory”. In: *Annual Review of Physical Chemistry* 32.1 (1981), pp. 159–177.
- [5] Matthias Feldmaier. “Phase-space resolved decay rates of driven systems near the transition state”. PhD thesis. Universität Stuttgart, 1. Institut für Theoretische Physik, 2020.
- [6] Matthias Feldmaier, Philippe Schraft, Robin Bardakcioglu, Johannes Reiff, Melissa Lober, Martin Tschöpe, Andrej Junginger, Jörg Main, Thomas Bartsch, and Rigoberto Hernandez. “Invariant Manifolds and Rate Constants in Driven Chemical Reactions”. In: *The Journal of Physical Chemistry B* 123.9 (Feb. 2019), pp. 2070–2086. ISSN: 1520-5207. DOI: [10.1021/acs.jpcc.8b10541](https://doi.org/10.1021/acs.jpcc.8b10541). URL: <http://dx.doi.org/10.1021/acs.jpcc.8b10541>.
- [7] Nandor Balazs. “Wigner on Physical Chemistry”. In: *Part I: Physical Chemistry. Part II: Solid State Physics*. Ed. by Arthur S. Wightman. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 3–20. ISBN: 978-3-642-59033-7. DOI: [10.1007/978-3-642-59033-7_1](https://doi.org/10.1007/978-3-642-59033-7_1). URL: https://doi.org/10.1007/978-3-642-59033-7_1.

- [8] Baron Peters. In: *Reaction Rate Theory and Rare Events Simulations*. Ed. by Baron Peters. Amsterdam: Elsevier, 2017, pp. 79–128. ISBN: 978-0-444-56349-1. DOI: <https://doi.org/10.1016/B978-0-44-456349-1.00004-0>. URL: <http://www.sciencedirect.com/science/article/pii/B9780444563491000040>.
- [9] Matthias Feldmaier, Robin Bardakcioglu, Johannes Reiff, Jörg Main, and Rigoberto Hernandez. “Phase-space resolved rates in driven multidimensional chemical reactions”. In: *The Journal of chemical physics* 151.24 (2019), p. 244108.
- [10] W. T. Illingworth. “Beginner’s guide to neural networks”. In: *IEEE Aerospace and Electronic Systems Magazine* 4.9 (1989), pp. 44–49.
- [11] Tara Abraham. “(Physio)logical circuits: The intellectual origins of the McCulloch-Pitts neural networks”. In: *Journal of the history of the behavioral sciences* 38 (Feb. 2002), pp. 3–25. DOI: [10.1002/jhbs.1094](https://doi.org/10.1002/jhbs.1094).
- [12] Richard Brown. “Donald O. Hebb and the Organization of Behavior: 17 years in the writing”. In: *Molecular Brain* 13 (Dec. 2020). DOI: [10.1186/s13041-020-00567-8](https://doi.org/10.1186/s13041-020-00567-8).
- [13] F. Caravelli, F. L. Traversa, and M. Di Ventra. “Complex dynamics of memristive circuits: Analytical results and universal slow relaxation”. In: *Physical Review E* 95.2 (Feb. 2017). ISSN: 2470-0053. DOI: [10.1103/physreve.95.022140](https://doi.org/10.1103/physreve.95.022140). URL: <http://dx.doi.org/10.1103/PhysRevE.95.022140>.
- [14] S. Albawi, T. A. Mohammed, and S. Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: [10.1109/ICEngTechnol.2017.8308186](https://doi.org/10.1109/ICEngTechnol.2017.8308186).
- [15] Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. *High-Performance Neural Networks for Visual Object Classification*. 2011. arXiv: [1102.0183](https://arxiv.org/abs/1102.0183) [cs.AI].
- [16] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. *A.: Sequential deep learning for human action recognition*. 2011.
- [17] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. *Understanding Neural Networks Through Deep Visualization*. 2015. arXiv: [1506.06579](https://arxiv.org/abs/1506.06579) [cs.CV].
- [18] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*. 2014. arXiv: [1409.3215](https://arxiv.org/abs/1409.3215) [cs.CL].
- [19] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber. “A System for Robotic Heart Surgery that Learns to Tie Knots Using Recurrent Neural Networks”. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 543–548. DOI: [10.1109/IR0S.2006.282190](https://doi.org/10.1109/IR0S.2006.282190).

-
- [20] Gábor Petneházi. *Recurrent Neural Networks for Time Series Forecasting*. 2019. arXiv: [1901.00069](https://arxiv.org/abs/1901.00069) [cs.LG].
- [21] Hansika Hewamalage, Christoph Bergmeir, and Kasun Bandara. “Recurrent Neural Networks for Time Series Forecasting: Current status and future directions”. In: *International Journal of Forecasting* 37.1 (Jan. 2021), pp. 388–427. ISSN: 0169-2070. DOI: [10.1016/j.ijforecast.2020.06.008](https://doi.org/10.1016/j.ijforecast.2020.06.008). URL: <http://dx.doi.org/10.1016/j.ijforecast.2020.06.008>.
- [22] Alex Graves and Jürgen Schmidhuber. “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In: *NEURAL NETWORKS* (2005), pp. 5–6.
- [23] G. K. Anumanchipalli, Josh Chartier, and E. Chang. “Speech synthesis from neural decoding of spoken sentences”. In: *Nature* 568 (2019), pp. 493–498.
- [24] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. “Long Short Term Memory Networks for Anomaly Detection in Time Series”. In: Apr. 2015.
- [25] Felix Gers, Nicol Schraudolph, and Jürgen Schmidhuber. “Learning Precise Timing with LSTM Recurrent Networks”. In: *Journal of Machine Learning Research* 3 (Jan. 2002), pp. 115–143. DOI: [10.1162/153244303768966139](https://doi.org/10.1162/153244303768966139).
- [26] Douglas Eck and Jürgen Schmidhuber. “Learning the long-term structure of the blues”. In: *IN PROC. INTL. CONF.* 2002, pp. 284–289.
- [27] J. Schmidhuber, F. Gers, and D. Eck. “Learning Nonregular Languages: A Comparison of Simple Recurrent Networks and LSTM”. In: *NEURAL COMPUTATION* 14 (2002), p. 2002.
- [28] Alex Graves, Santiago Fernández, Marcus Liwicki, Horst Bunke, and Jürgen Schmidhuber. “Unconstrained Online Handwriting Recognition with Recurrent Neural Networks”. In: *Proceedings of the 20th International Conference on Neural Information Processing Systems*. NIPS’07. Vancouver, British Columbia, Canada: Curran Associates Inc., 2007, pp. 577–584. ISBN: 9781605603520.
- [29] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atila Baskurt. “Lecture Notes in Computer Science”. In: Jan. 2011, pp. 29–39. ISBN: 978-3-642-25445-1. DOI: [10.1007/978-3-642-25446-8_4](https://doi.org/10.1007/978-3-642-25446-8_4).
- [30] Sepp Hochreiter, Martin Heusel, and Klaus Obermayer. “Fast model-based protein homology detection without alignment”. In: *Bioinformatics* 23.14 (May 2007), pp. 1728–1736. ISSN: 1367-4803. DOI: [10.1093/bioinformatics/btm247](https://doi.org/10.1093/bioinformatics/btm247). eprint: <https://academic.oup.com/bioinformatics/article-pdf/23/14/1728/466668/btm247.pdf>. URL: <https://doi.org/10.1093/bioinformatics/btm247>.

- [31] T. Thireou and M. Reczko. “Bidirectional Long Short-Term Memory Networks for Predicting the Subcellular Localization of Eukaryotic Proteins”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 4.3 (2007), pp. 441–446. DOI: [10.1109/tcbb.2007.1015](https://doi.org/10.1109/tcbb.2007.1015).
- [32] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. “Predictive Business Process Monitoring with LSTM Neural Networks”. In: *Lecture Notes in Computer Science* (2017), pp. 477–492. ISSN: 1611-3349. DOI: [10.1007/978-3-319-59536-8_30](https://doi.org/10.1007/978-3-319-59536-8_30). URL: http://dx.doi.org/10.1007/978-3-319-59536-8_30.
- [33] Edward Choi, Mohammad Taha Bahadori, Andy Schuetz, Walter F. Stewart, and Jimeng Sun. *Doctor AI: Predicting Clinical Events via Recurrent Neural Networks*. 2016. arXiv: [1511.05942](https://arxiv.org/abs/1511.05942) [cs.LG].
- [34] George Bebis and Michael Georgiopoulos. “Feed-forward neural networks”. In: *IEEE Potentials* 13.4 (1994), pp. 27–31.
- [35] Terrence L Fine. *Feedforward neural network methodology*. Springer Science & Business Media, 2006.
- [36] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: [1412.6980](https://arxiv.org/abs/1412.6980) [cs.LG].
- [37] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2017. arXiv: [1609.04747](https://arxiv.org/abs/1609.04747) [cs.LG].
- [38] Agnes Lydia and Sagayaraj Francis. “Adagrad—An optimizer for stochastic gradient descent”. In: *Int. J. Inf. Comput. Sci.* 6.5 (2019).
- [39] Matthew D Zeiler. “Adadelata: an adaptive learning rate method”. In: *arXiv preprint arXiv:1212.5701* (2012).
- [40] Tijmen Tieleman and Geoffrey Hinton. “Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude”. In: *COURSERA: Neural networks for machine learning* 4.2 (2012), pp. 26–31.
- [41] Nishant Shukla and Kenneth Fricklas. *Machine learning with TensorFlow*. Manning Greenwich, 2018.
- [42] URL: <https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/> (visited on 02/16/2021).
- [43] URL: https://www.tensorflow.org/api_docs/python/tf/keras/losses (visited on 02/16/2021).
- [44] Peter Goldsborough. “A tour of tensorflow”. In: *arXiv preprint arXiv:1610.01178* (2016).

-
- [45] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. Software available from tensorflow.org. 2015. URL: <http://tensorflow.org>.
- [46] Giancarlo Zaccone. *Getting started with TensorFlow*. Packt Publishing Ltd, 2016.
- [47] Antonio Gulli, Amita Kapoor, and Sujit Pal. *Deep learning with TensorFlow 2 and Keras: regression, ConvNets, GANs, RNNs, NLP, and more with TensorFlow 2 and the Keras API*. Packt Publishing Ltd, 2019.
- [48] Martin Tschöpe, Matthias Feldmaier, Jörg Main, and Rigoberto Hernandez. “Neural network approach for the dynamics on the normally hyperbolic invariant manifold of periodically driven systems”. In: *Phys. Rev. E* 101 (2 Feb. 2020), p. 022219. DOI: [10.1103/PhysRevE.101.022219](https://doi.org/10.1103/PhysRevE.101.022219). URL: <https://link.aps.org/doi/10.1103/PhysRevE.101.022219>.
- [49] Matthias Feldmaier, Robin Bardakcioglu, Johannes Reiff, Jörg Main, and Rigoberto Hernandez. “Phase-space resolved rates in driven multidimensional chemical reactions”. In: *The Journal of Chemical Physics* 151.24 (2019), p. 244108. DOI: [10.1063/1.5127539](https://doi.org/10.1063/1.5127539). eprint: <https://doi.org/10.1063/1.5127539>. URL: <https://doi.org/10.1063/1.5127539>.
- [50] Matthias Feldmaier, Johannes Reiff, Rosa M. Benito, Florentino Borondo, Jörg Main, and Rigoberto Hernandez. “Influence of external driving on decays in the geometry of the LiCN isomerization”. In: *The Journal of Chemical Physics* 153.8 (2020), p. 084115. DOI: [10.1063/5.0015509](https://doi.org/10.1063/5.0015509). eprint: <https://doi.org/10.1063/5.0015509>. URL: <https://doi.org/10.1063/5.0015509>.
- [51] Robin Bardakcioglu, Johannes Reiff, Matthias Feldmaier, Jörg Main, and Rigoberto Hernandez. “Thermal decay rates of an activated complex in a driven model chemical reaction”. In: *Phys. Rev. E* 102 (6 Dec. 2020), p. 062204. DOI: [10.1103/PhysRevE.102.062204](https://doi.org/10.1103/PhysRevE.102.062204). URL: <https://link.aps.org/doi/10.1103/PhysRevE.102.062204>.

Acknowledgements

I would like to mention a few names without whom this thesis would not have been written.

Apl. Prof. Dr. Jörg Main: for guiding and giving me this opportunity to explore this wonderful topic.

Prof. Dr. Hans Peter Büchler: for guiding me throughout this master's course.

Dr. Mia Kumric: for helping me out with every little thing concerning this master's program.

Special thanks to Johannes Reiff, for his patience in guiding and helping me throughout the thesis.

Sachin Tewari, my husband: for believing in me and motivating me always.

Siddharth & Kiara, my kids: for being such lovely kids letting mama complete her thesis.

Thanks to all the colleagues and members of ITP1, Uni Stuttgart, and MSc Physics (International), Uni Stuttgart.

Erklärung

Ich versichere,

- dass ich diese Masterarbeit selbstständig verfasst habe,
- dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
- dass ich die Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe
- dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
- und dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

Stuttgart, 4. März 2021

Pooja Mishra