Institut für Parallele und Verteilte Systeme

Machine Learning & Robotics Lab

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Bachelorarbeit Nr. 2484

Regularizing Gradient Properties On Deep Neural Networks

Sebastian Wallkötter

Studiengang:	Technische Kybernetik
Prüfer:	Prof. Dr. rer. nat. Toussaint, Marc
Betreuer:	Prof. Dr. rer. nat. Toussaint, Marc
begonnen am:	07.10.2015
beendet am:	05.03.2016
CR-Klassifikation:	F.1.1 , G.1.6 , I.2.6 , I.5.1

Abstract

Diese Bachelorarbeit präsentiert einen neuen Ansatz zum Trainieren von tiefen neuronalen Netzen. Während des Rückwärtspropagierens in solchen tiefen Architekturen wird häufig beobachtet, dass der Gradient verschwindet. Weiterhin wird beobachtet, dass Schichten mit einer logistischen Aktivierungsfunktion von oben nach unten saturieren, was die Konvergenz verlangsamt, da der Gradient nur schlecht hinter diesen Schichten propagiert. Beide Beobachtungen erzeugen den Wunsch Eigenschaften des Gradienten direkt beeinflussen zu können und seine Eigenschaften direkt festlegen zu können. Diese Arbeit ermöglicht ein solches Vorgehen, indem sie die Kostenfunktion des Netzwerks modifiziert. Hierdurch werden die klassischen "back propagation"-Gleichungen modifiziert, weshalb neue "extended back propagation"-Gleichungen hergeleitet werden. Abschließend werden zwei Methoden und deren Kombination zur Regulierung des Gradienten vorgestellt und an einem zwei-Klassen, sowie einem mehr-Klassen (MNIST) Klassifikationsproblem getestet, um die Vorteile des Trainierens mit dieser Methode herauszustellen. Das Ergebnis ist, dass diese Methode das Trainieren von tiefen neuronalen Netzen mit logistischer Aktivierungsfunktion stark verbessert und einerseits die sonst unmögliche Klassifikation im mehr-Klassen-Fall ermöglicht, während andererseits eine Verbesserung der Konvergenzgeschwindigkeit im ein-Klassen Fall erreicht wird.

Abstract

This bachelor thesis presents a novel approach to training deep neural networks. While back propagating on these deep architectures, it is often found that the gradient vanishes. Further, layers with logistic activation functions will saturate from top to bottom, which is slowing down convergence as the gradient can't propagate well past these saturated layers. Both observations awaken the wish to have the ability to regularize the gradient and directly force its properties. This thesis enables such regularization by modifying the network's cost function. Such changes modify the classic back propagation equations and therefore, the new extended back propagation equations are computed. Finally, two methods of regularization and their combination are presented and tested on a binary and a multi-class (MNIST) classification problem to show the benefits of training with these methods. A result of this thesis is the finding that this setup massively improves training on logistic networks, on the one hand enabling otherwise impossible classification in the multi-class case, while on the other speeding up training on a single class.

Contents

1 Introduction

1.1 Motivation

In recent years, training of deep architectures, especially neural networks, has drawn increased attention from the machine learning community. Research in this area has been made possible by the collection of large data sets, advancements in parallel computing (GPU-computing) and proposals of new algorithms to tackle training of several stacked hidden layers.

A problem encountered, while exploring these architectures, is the error's gradient, which 'fades' as the network depth increases. This causes high layers to receive big updates, while low levels barely experience change in weights resulting in slow or impossible convergence. Observations indicate that the choices of used non-linearity and method of initialization greatly influences this behavior.

In literature a common choice to circumvent this issue is the usage of ReLUs (rectifier linear units) [6] or similar looking functions like $softmax$. Recently proposed highway networks [15] are another choice. These, similar to LSTM-networks, use identity neurons to ensure good back propagation of the gradient. An alternative approach is the usage of *sigmoid* or *tanh* transfer functions paired with careful initialization [5] and pre-training to find good starting points for gradient descent. Above approaches alter the gradient in two ways. The first two modify the network's structure and thereby ensure good behavior of the gradient. Latter intelligently chooses the region from which gradient descent training should start. These initially chosen weights allow for a good propagation.

Yet there is another tuning parameter that can be modified in order to ensure good gradients: the objective function. This would be done by adding a regularization term to the loss function, similar to L_1 or L_2 norms sometimes applied to the weights. Unfortunately it is difficult to see how a function based upon the network layers' activation or blunt weights will alter the gradient. It would be a lot simpler if the gradient could be used directly within the error function to express desired properties. Hence, the first effort of this thesis is to derive update equations that enable direct use of the loss function's gradient in the error function. Next a selection of possible regularization functions will be discussed each

forcing the network to learn different desirable weight compositions with good propagation properties. In the experiment section it will be shown how this addition to the loss can beneficially influence optimization and even succeed in training deep networks that use logistic ($y = 1/(1 - e^{-x})$) units.

1.2 Related Work

1998 LeCunn et al. [10] published a paper displaying implementation tricks on neural networks. The paper shows that, in order to properly train neural networks with back propagation, the following points should be concidered:

- input data should be centered, normalized and decorrelated
- the desired output value should be inside the output functions reachable space
- weights should be initialized, so that the input distribution to a neuron is $\mathcal{N}(0|1)$
- learning rates should be individualized for each neuron, proportional to $\sqrt{\#\text{inputs}}$ and biased depending on the depth of the neuron; The deeper the node, the higher the learning rate.
- the transfer function should be symmetric to 0 and adding a small linear term can help to avoid plateaus encountered during training

It is further mentioned, that classical 2nd order gradient descent methods are impractical for current training tasks as they are either to computational expensive or don't work with stochastic descent used in conjunction with big data sets. Further the logistic function is not recommended due to its non-zero mean, which introduces a bias for each layer, massively slowing down convergence.

Above speaks against the usage of the logistic function as a transfer function in neural networks. To avoid the well known plateauing effect, adding a linear term is suggested, with the purpose of propagating the gradient well in regions where the non linearity is unable to. The approach differs from this thesis in that the work presented here aims to make the network organize its 'good gradient' property by itself and not artificially change the structure to circumvent it as suggested by the authors.

2010 A stud, made by Glorot et al. [5] investigates the issue of why plain gradient descent preforms so poor on deep architectures. Experiments were made using neural networks with $1 - 5$ hidden layers. These nets were trained to test the *sigmoid*, *tanh* and the $softsign$ as activation functions. The goal was to observe how the gradient propagates through these networks over time. It was shown that, when using the $softsign$, all layers will saturate with the same speed while, when using *tanh*, the layers saturate starting with the lowest one and moving up.

Odd performance was found when testing the *sigmoid* as in this case the top hidden layer starts to saturate first. This is exceptionally bad as saturated *sigmoid* units have

a low derivative, depleting the back propagated gradient even further. The hypothesized reason for this is that random initialization causes the network to start in a poor state. Fortunately the layer moves out of the saturation regime if trained long enough allowing deeper layers of the network to converge. Hence, the first conclusion of the paper is that *sigmoid* transfer functions are unsuited for deep architectures and further that this behavior might explain an effect described as plateaus during training.

In account for the decay of gradient, a new initialization scheme is purposed. Weights are to be initialized using

$$w = U[-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}] \ . \tag{1.1}$$

This is done to approximately satisfy the two objectives of 1) maintaining activation variances and 2) preserve the back propagated gradients.

The idea of monitoring the saturation and propagation of gradients presented in the paper is further advanced in this thesis. The goal of doing this is to understand how a network can be regularized to avoid the saturation regime mentioned in et al.'s publication.

2010 Asking 'Why does unsupervised pre-training help deep learning?', Erhan et al. [4] presented an in-depth study showing the influence of pre-training.

It is argued that gradient descent is difficult, as deep and shallow layers have to be trained at the same time. Further, the top two layers often suffice to already (over-)fit the training data causing poor generalization. The authors conclude that pre-training helps because it initializes the weights in basins with good generalization allowing better training on the deeper layers. This is achieved by having the network learn an unsupervised objective function, learning the input sample distribution $P(X)$. Afterwards the network is trained on a supervised goal, learning $P(Y|X)$ similar to classical gradient descent.

In a sense this regularizes the network, as pre-training chooses the starting point and thus the region which is reachable in parameter space. A possible explanation why this works is that small disturbances in the parameters allow 'skipping' into another basin of attraction in the early stages of training. Later this doesn't seem possible anymore. Erhan et al. test above theory on a set of neural networks, stacked auto encoders [16] and restricted Boltzmann machines [14] with varying depth finding that pre-training significantly improves the results, as network depth increases.

This idea will also be used in the present work. Training a network only to have good gradient properties, regardless of the resulting loss value, is a kind of pre-training. However, the gradient indirectly depends on the chosen loss, not an unsupervised one. It is shown that this has beneficial influence on the network such as allowing deep logistic networks to perform prediction, which can't be achieved with standard back propagation.

2011 One way of circumventing the problem of poor gradient propagation is the usage of deep sparse rectifier networks, presented by Glorot et al. [6]. These networks use a different transfer function, so called rectifier units $y = \max(0, x)$. The paper argues that this is a

(biologically) more plausible choice then *sigmoid* or *tanh* as it is closer to the activation model commonly used by biologists but additionally has good numeric properties desired by the machine learning community. Rectifier units have the interesting behavior to create sparse networks in which only a fraction of the units have a nonzero value. This combined with the observation, that they yield better results if no prior pre-training stage is used make rectifier units an interesting choice for training of neural networks. It is further stated, that the hard 0, compared to the numeric 0 or -1 in other non linear functions, is beneficial for training the activation.

This presents a different approach to the problem this thesis is solving. As a rectifier unit does not saturate slowly like usual activation functions, a unit always propagates well until it receives negative input and completely turns off, circumventing the issue of very low gradients (and thus very slow convergence) in saturated regions of other activation functions.

2011 Bekir Karlik and A Vehbi Olgac preformed a performance analysis comparing different transfer functions [8]. The studies were made on small multi-layer-perceptrons testing different transfer functions for speed of convergence and quality of generalization. Similar to the findings by Glorot et al. [6], the logistic function converged slowest and lacks generalization power compared to commonly used *tanh* or symmetrical sigmoid.

2012 A new training theme for deep networks has been proposed by Weston et al. [17], showing how a network's generalization power can be increased by learning simultaneously from an unsupervised task and a supervised one using unlabeled pairs of examples. This is done by constructing a weight matrix W_{ij}, which measures the similarity of two samples $x_i,\ x_j$. Using this matrix and the prediction of both examples, a loss can be created for each pair:

$$\sum_{ij} L(f(x_i, \alpha), f(x_j, \alpha), W_{ij}) \ . \tag{1.2}$$

The authors purpose that above loss can be used in three ways: First it can be added directly to the supervised training's loss. Second it can be applied to each hidden layer, regularizing these. And finally, an auxiliary layer can be added that branches from any arbitrary hidden layer and serves as input for the unsupervised loss. This layer is discarded after training. This approach can be generalized for multi-layer Neural Networks and is easy to optimize using standard or stochastic gradient descent. Although it is easy to implement and cheap to compute as there is no need for a decoder compared to stacked auto-encoders, a drawback of this approach may be the construction of the weight matrix W, which may be time consuming for big data sets.

Tests were performed on several data sets including MNIST. Further a semantic role labeling task was trained and the approach was used in an object recognition task on unlabeled video data. Training on MNIST was done by comparing a convolutional neural network to a shallow neural network. Afterwards experiments on very deep architectures were made with scaling depth of up to 15 layers. While the discussed approach always outperforms classic back propagation, it additionally starts acting as a regularization for increasing depth

preventing the network from overfitting.

Above is similar to the procedure chosen in this thesis. Additional training of the hidden layers on a specific task enables learning deep representations and fulfills a regularizing property. This thesis' approach builds upon the work done by Weston et al. as it adds new choices to the design of the unsupervised function. A difference is that this thesis only looks into unsupervised losses that origin from a single sample instead of considering similarity between two samples.

2015 Very recently an altered architecture [7], called batch normalization, has been proposed by Ioffe and Szegedy, massively improving the speed of convergence. It evolves around removing the internal covariate shift from the network by whitening each layer individually. This is beneficial as regular training requires each network's layer to learn optimal weights for a distribution, which changes on each training step due to stochastic training. This constant shift in the layer's input distribution forces low learning rates if the network is to converge. As it is known to be beneficial to whiten the input of a network, the authors argue that the same procedure is beneficial for each individual layer. It is proposed that this has positive effects not only on the normalized layer but also on previous and prior layers.

In order to achieve this effect an additional layer is added between the non-linear transfer function and the calculation of the neuron's activations. This layer centers and decorrelates each neuron individually for each mini-batch. It is better to decorrelate each unit individually then the entire input data jointly as it is significantly less computationally expensive. However this may reduce the network's representation capability. To compensate for this reduced capability, two new hyper parameters are introduced removing the need of a bias layer in the process. Because this procedure directly modifies the network's structure, the authors compute the new, resulting back propagation equations. Experiments displayed in the paper suggest that, when doing batch normalization, the need for dropout is removed. Further it allows for larger learning rates compared to classic gradient descent and even succeeds in training networks with bounded non-linearities.

This framework is another example of how changing the networks structure can overcome problems in training deep architectures. It differs from other papers in a way that it regularizes the non-linearity's input to assure well designed gradients, rather then choosing an activation function with a nicer gradient. It stands in contrast to this thesis, as the good propagation is again achieved by a modification of structure rather then clever design and avoidance of saturated region.

2 | Standard Back Propagation

2.1 Neural Network Notation

For all experiments and calculations this thesis will examine feed forward neural networks using a logistic transfer function. The first and last layer are special as they will be using linear transfer functions. Such networks contain L layers, having N neurons each, again with exceptions for the first and last layer. Each layer n is fully connected to its predecessor $n-1$ and ancestor $n+1$, meaning that a neuron in one layer has a connection to all neurons in the other.

A single layer can be described by the equation

$$\begin{aligned} z_{l+1} &= W_l x_l + \lambda_l \\ x_l &= \sigma(z_l) \ , \end{aligned} \tag{2.1}$$

in which W_l is a matrix where each row contains the magnitude of influences from the previous layers' neurons on a neuron in the current layer. λ_l is a linear bias and σ is the logistic function applied component wise to the vector z_l. The final output of the network in the last layer is given as

$$z_{L+1} = W_L x_L + \lambda_L \tag{2.2}$$

and the input data is denoted as x_1. Iterating (2.1) for all $l = 1, \ldots, L$ computes the prediction of the network and is usually called forward propagation. The predicted class label is the index of the largest entry in z_{L+1}, which is computed by

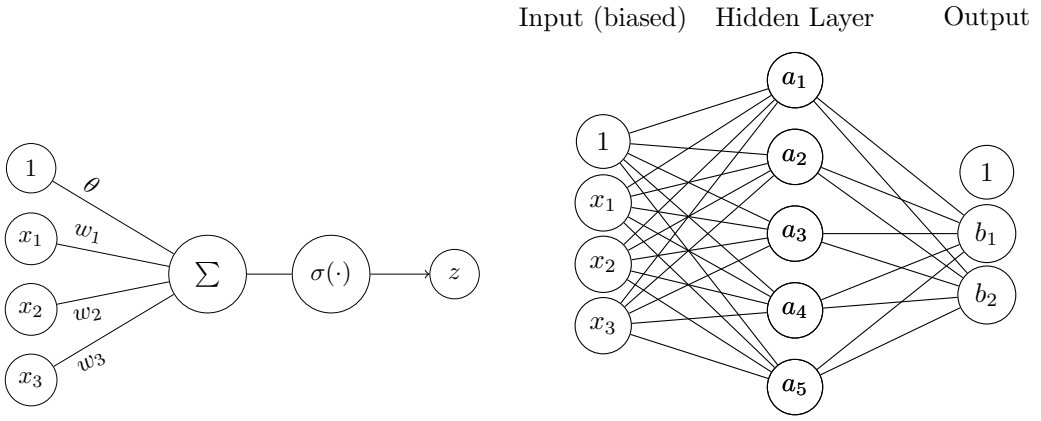
$$\underset{i}{\operatorname{argmax}}(z_{L+1,i}) \ . \tag{2.3}$$

Therefore, a neural network in this thesis is a chain of functions f_l given in (2.1)

$$z_{L+1} = (f_L \circ f_{L-1} \circ \cdots \circ f_1)(x) \tag{2.4}$$

Sigmoid Function A specialty, not that popular in research anymore, is the use of the a logistic transfer function

$$\sigma(x) = \frac{1}{1+e^{-x}} \ . \tag{2.5}$$



(a) A preceptron (b) A small sample network build of preceptrions.

Figure 2.1: Left: A preceptron, which is the simplest element of a neural network. This one has three inputs x_i, weighted by w_i and a bias weight θ which are summed up and fed into a non-linear activation function σ, resulting in the output of the unit.
Right: A sample neural network with 3 inputs x_i, 5 preceptrons a_i and 2 outputs b_i. Each a_i represents a preceptron, with their inputs drawn as edges between the nodes. Each edge has its own weight, which is omitted for readability. The ones represent a constant input for the bias weight. Replacing the output layer b with an arbitrary amount of layers filled with preceptrons a followed by the output layer creates deep networks.

This function has the convenient mathematical derivative

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{2.6}$$

and only reaches 0 and 1 in its limits. Further the logistic function satisfies the conditions made by Cybenko [3] and thus enables the network to approximate any function on a compact subset of $\mathbb{R}^n$ provided the layers are big enough. A disadvantage of this particular activation function is that it has non-zero mean, which slows down convergence [10]. To compensate for this effect and to center the data, a bias λ_l is trained for each layer. This bias may be considered as an extra input neuron that has a variable weight together with a fixed activation of 1, which integrates it smoothly into the networks, as sown in Figure 2.1. Although it may be viewed as an attempt to 'shift' the logistic function and make it symmetric around 0, the range of it would remain half of *tanh*'s range ($[-0.5; 0.5]$ compared to $[-1; 1]$). This suggests that, despite adding the bias, a logistic activation function may still be inferior to a scaled *sigmoid* or the *tanh*.

2.2 Back-Propagation

Partial and Total Derivative Back-propagation was first introduced by Rumelhart et al. [13] in 1988. This thesis uses the same method, however, it is done in a vectorized way.

In order to do this, a distinction between

$$\frac{\partial f}{\partial x} \text{ and } \frac{df}{dx} \tag{2.7}$$

has to be made. $\partial f/\partial x$ refers to the partial derivative, that only accounts for a direct dependency from f on x. df/dx is the total derivative of f with respect to x, computing not just direct dependencies, but also latent ones. More precise, let f depend on $h_1, \ldots, h_n$ then

$$\frac{df}{dx} = \frac{\partial f}{\partial x} + \sum_{i=1}^{n} \frac{\partial f}{\partial h_i} \frac{dh_i}{dx} \ . \tag{2.8}$$

Concider $f(g(x)) = (f \circ g)(x)$ as an easy example. Then

$$\frac{df}{dx} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x} \text{ , but } \frac{\partial f}{\partial x} = 0 \ . \tag{2.9}$$

Hinge-Loss in Multi-Class Classification In order to do back propagation a loss function has to be designed first. Here the hinge loss is chosen, which has to be extended in order to do multi-class classification. This was done by Crammer and Singer [2], reading as

$$L(z_{L+1}, y) = \max(0, 1 + \max_{i \neq y}(z_{L+1,i}) - z_{L+1,y}) \ , \tag{2.10}$$

where $z_{L+1,y}$ is the networks 'score' for the true class, while y is the desired label. The idea behind this is that the network's output for the true label should be 1 value bigger than all other labels. A behavior of this loss is that the gradient is $\neq 0$ for two classes at most, which is undesirable in this case. The reason for this is that $\max_{i \neq y}(z_{L+1,i})$ will only return the currently highest-valued unwanted label. It is better to additionally update all other undesired labels that, from the loss function's view, don't have enough distance from the desired label's score. Hence, a slightly altered loss function

$$L(z_{L+1}, y) = \sum_{i \neq y} \max(0, 1 + z_{L+1,i} - z_{L+1,y}) \tag{2.11}$$

is used, which accounts for the terms filtered by the maximum operator. Deriving (2.11) for $i \neq y$ leads to

$$\frac{\partial L}{\partial z_i} = [i \neq y \text{ and } 1 + z_i - z_y > 0] \ , \tag{2.12}$$

where $[\cdot] = 1$ if the expression is true and 0 if it isn't. For $i = y$

$$\frac{\partial L}{\partial z_y} = \sum_{k} -[1 + z_k - z_y > 0] \tag{2.13}$$

is derived.

The Gradient is an Outer Product Now that a loss function has been chosen,

$$W_{l,next} = W_l - \alpha \frac{dL}{dW_l} \tag{2.14}$$

can be computed and iterated for each layer l. dL/dW_l can be rewritten using [12]:eq40, which reads

$$\frac{\partial a^T W b}{\partial W} = ab^T \ , \tag{2.15}$$

where ab^T denotes the outer product of two vectors. Above has to be extended to allow a function instead of a. Let $f : \mathbb{R}^m \to \mathbb{R}$, $x \in \mathbb{R}^n$ and $W \in \mathbb{R}^{m \times n}$. Then

$$\begin{aligned} \frac{df(Wx)}{dW} &= \begin{bmatrix} \frac{df(Wx)}{dW_{1,1}} & \cdots & \frac{df(Wx)}{dW_{1,n}} \\ & \vdots & \\ \frac{df(Wx)}{dW_{m,1}} & \cdots & \frac{df(Wx)}{dW_{m,n}} \end{bmatrix} = \begin{bmatrix} \frac{df_1}{dWx} x_1 & \cdots & \frac{df_1}{dWx} x_n \\ & \vdots & \\ \frac{df_m}{dWx} x_1 & \cdots & \frac{df_m}{dWx} x_n \end{bmatrix} \\ &= \left(\frac{df}{d(Wx)} \right)^T x^T \ . \end{aligned} \tag{2.16}$$

where $df/d(Wx)$ is the derivate of f, also known as its Jacobian.

Back-Propagation Equations Using (2.4) the loss can be reformulated

$$L = (L(\cdot, y) \circ f_L \circ \cdots \circ f_{l+1} \circ f_l \circ f_{l-1} \circ \cdots \circ f_1)(x) \tag{2.17}$$

$$L = \underbrace{(L(\cdot, y) \circ f_L \circ \cdots \circ f_{l+1})}_{h_{l+1}} \underbrace{(W_l x_l + \lambda_l)}_{z_{l+1}} \tag{2.18}$$

and with (2.16) derived as

$$\frac{dL}{dW_l} = \left(\frac{\partial h_{l+1}}{\partial z_{l+1}} \right)^T x_l^T = \left(\frac{dL}{dz_{l+1}} \right)^T x_l^T \ . \tag{2.19}$$

It can be seen easily, that $h_l = h_{l+1} \circ f_l$, which allows to recursively compute $\partial h_{l+1}/\partial z_{l+1}$ starting with h_{L+1}.

$$\delta_{L+1} = \frac{\partial h_{L+1}}{\partial z_{L+1}} = \frac{\partial L}{\partial z_{L+1}} \ , \tag{2.20}$$

which is the loss gradient and has already been computed above. For h_l, where $0 < l < L+1$ and

$$\delta_l = \frac{\partial h_l}{\partial z_l} = \frac{\partial h_{l+1}}{\partial f_l} \frac{\partial f_l}{\partial z_l} = \left(\frac{\partial h_{l+1}}{\partial f_l} W_l \right) \cdot x_l^T \cdot (1 - x_l)^T \ . \tag{2.21}$$

Here $a \cdot b$ denotes an element wise multiplication of two vectors. The trick used next is $f_l = z_{l+1}$ and therefore $\partial h_{l+1}/\partial f_l = \delta_{l+1}$, which has already been computed. Hence, the (vectorized) back propagation equations are

$$\delta_{L+1} = \partial L / \partial z_{L+1} \tag{2.22}$$

$$\delta_l = (\delta_{l+1} W_l) \cdot x_l^T \cdot (1 - x_l)^T \tag{2.23}$$

$$\frac{dL}{dW_l} = \delta_{l+1}^T x_l^T \tag{2.24}$$

By using these equations and modifying the training set by adding perturbed images, state-of-the-art performance can be achieved [1]. Thus, it makes sense to compare against back propagation in terms of quality and speed of convergence.

Regularizing Hidden Layers Currently a network can only be trained on the loss L which has only a single direct dependency to the network, z_{L+1}. If one may want to regularize hidden layers, for example to be sparse $\|z_i\|_1 \rightsquigarrow \min$, above equations have to be modified. Suppose

$$O(z_1, \ldots, z_{L+1}, y) = L(z_{L+1}, y) + R(z_1, \ldots, z_{L+1}) \ , \tag{2.25}$$

minimizing O will minimize L while accounting for any regularization R that may be desirable for the hidden layers. As above is a sum, both terms can be differentiated independently resulting in the same update as normal back propagation for L. For R it is

$$\frac{dR}{dz_l} = \frac{\partial R}{\partial z_l} + \underbrace{\frac{dR}{dz_{l+1}}}_{\rho_{l+1}} \frac{\partial z_{l+1}}{\partial z_l} \tag{2.26}$$

which again is a recursion similar to previous back propagation algorithm. $\partial z_{l+1}/\partial z_l = \partial f_l/\partial z_l$ has already been calculated above. Further we can use (2.16) one more time to compute, that the update part from R is

$$\frac{dR}{dW} = \left(\frac{dR}{dz_{l+1}}\right)^T x_l^T \ , \tag{2.27}$$

hence both iterations can be combined to

$$\hat{\delta}_{L+1} = \delta_{L+1} + \cancelto{0}{\rho_{L+1}} \tag{2.28}$$

$$\hat{\delta}_l = (\hat{\delta}_{l+1} W_l) \cdot x_l^T \cdot (1 - x_l)^T + \frac{\partial R}{\partial z_l} \tag{2.29}$$

$$\frac{dO}{dW_l} = \hat{\delta}_{l+1}^T x_l^T \tag{2.30}$$

which is the 'old' back propagation algorithm but with an added term to regularize hidden layers. An important thing to note is that, although the objective function has changed and got more dependent on the network, gradients still pass through the same structure, causing update equations very similar to classic back propagation. This will be encountered again in the next chapter.

3 Regularizing Gradient Properties

3.1 Gradient Regularization (Extended Backprop)

Extending the objective function to contain not just z_{L+1}, but any term depending on z_l, $l \in [1, L+1]$, described in section 2.2, enables many kinds of regularization. Nevertheless, it is not possible to regularize on δ_l, which is the gradient of L with respect to z_l. To solve above issue, the first goal of this thesis is to derive update equations to enable such regularization. Hence, consider in the following equation

$$O(y, z_k, \delta_k) = L(y, z_{L+1}) + R(z_k, \delta_k) \ , \tag{3.1}$$

where z_k and δ_k are short terms for $z_1, \ldots, z_{L+1}$ and $\delta_1, \ldots, \delta_{L+1}$ respectively. This again leads to

$$\frac{dO}{dW_l} = \frac{dL}{dW_l} + \frac{dR}{dW_l} \tag{3.2}$$

with an unknown expression for dR/dW_l. To further break this down, it will be usefull to introduce a utility variable $\kappa_l = \delta_{l+1} W_l$, so that the derivative can be written as

$$\frac{dR}{dW_l} = \cancelto{0}{\frac{\partial R}{\partial W}} + \left(\frac{dR}{d\kappa_l}\right)^T \delta_{l+1} + \left(\frac{dR}{dz_l}\right)^T x_l^T \ . \tag{3.3}$$

Again this can be computed iteratively using a back propagation like syntax. Practically speaking, that means the gradients are propagated forward and then backward through the network an additional time using values from all previous passes on each propagation.

Another possibility of looking at this is to view the calculation of z_k and δ_k as one extended forward propagation step, as both are needed to compute O. Afterwards an extended back propagation step is being performed, computing an element based on its predecessor and the data from forward propagation.

Starting with $dR/d\kappa_l$ the gradient can be computed using

$$\frac{dR}{d\kappa_l} = \frac{dR}{d\delta_l}\frac{\partial \delta_l}{\partial \kappa_l} = \underbrace{\frac{dR}{d\delta_l}}_{\eta_l} \cdot x_l^T \cdot (1 - x_l)^T \ . \tag{3.4}$$

Note that instead of looking at $dR/d\delta_{l+1}$ depending on κ_k, it can be viewed as dependent on the prior δ_k, where $k < l + 1$. Therefore, $dR/d\delta_k$ can be computed iteratively and from this the update $dR/d\kappa_l$ can be calculated. δ_1 does not have any δ_k prior to it (with a smaller index), thus its respective derivative is only a partial one

$$\frac{dR}{d\delta_1} = \frac{\partial R}{\partial \delta_1} \ . \tag{3.5}$$

All other values are computed as

$$\eta_l = \frac{\partial R}{\partial \delta_l} + \eta_{l-1}\frac{\partial \delta_{l-1}}{\partial \delta_l} \tag{3.6}$$

$$\eta_l = \frac{\partial R}{\partial \delta_l} + \left(\eta_{l-1} \cdot x_{l-1}^T \cdot (1 - x_{l-1})^T\right) W_{l-1}^T \ , \tag{3.7}$$

which is the forward pass through the network mentioned earlier. Next on the list are gradients with respect to z_{L+1}. If the loss derivative L still depends on z_{L+1}, the activation may influence δ_{L+1}. The hinge-loss however has a constant derivative, so

$$\frac{\partial \delta_{L+1}}{\partial z_{L+1}} = \frac{\partial^2 L}{(\partial z_{L+1})^2} = 0 \tag{3.8}$$

and thus

$$\frac{dR}{dz_{L+1}} = \frac{\partial R}{\partial z_{L+1}} + \frac{dR}{d\delta_{L+1}}\cancelto{0}{\frac{\partial \delta_{L+1}}{\partial z_{L+1}}} \ . \tag{3.9}$$

Meaning that above computed η_{L+1} does not influence the starting value of the now following backward propagation. From here derivations are similar to above classic back propagation with added possibility for regularization on hidden layers. A twist is that $\partial \delta_l / \partial z_l \neq 0$ for $l \neq L + 1$, which means that there is another term added in each step of the iteration. In more detail

$$\frac{dR}{dz_l} = \frac{\partial R}{\partial z_l} + \frac{dR}{d\delta_l}\frac{\partial \delta_l}{z_l} + \frac{dR}{dz_{l+1}}\frac{\partial z_{l+1}}{\partial z_l} \tag{3.10}$$

and

$$\xi_l = \frac{dR}{d\delta_l}\frac{\partial \delta_l}{\partial z_l} = \eta_l \cdot (\delta_{l+1} W_l) \cdot (1 - 2x_l)^T \cdot (1 - x_l)^T \cdot x_l^T \ . \tag{3.11}$$

The other derivatives are known from the previous section and computed in the same way. Hence, the entire update equation reads as follows

$$\xi_l = \frac{\partial R}{\partial z_l} + (\xi_{l+1} W_l) \cdot (1 - x_l)^T \cdot x_l^T + \eta_l \cdot (\delta_{l+1} W_l) \cdot (1 - 2x_l)^T \cdot (1 - x_l)^T \cdot x_l^T \tag{3.12}$$

To compute above equation, ξ_{l+1} and η_l have to be known. Therefore, the resulting iteration

equations read as follows

$$\eta_1 = \frac{\partial R}{\partial \delta_1} \tag{3.13}$$

$$\eta_{l+1} = \frac{\partial R}{\partial \eta_{l+1}} + \left(\eta_l \cdot (1 - \sigma(x_l))^T \cdot \sigma(x_l)^T\right) W_l^T \tag{3.14}$$

$$\xi_{L+1} = \frac{\partial R}{\partial z_{l+1}} + \frac{\partial^2 L}{(\partial z_{L+1})^2} \tag{3.15}$$

$$\xi_l = \frac{\partial R}{\partial z_l} + \left((\xi_{l+1} W_l) + \eta_l \cdot (\delta_{l+1} W_l) \cdot (1 - 2x_l)^T\right) \cdot (1 - x_l)^T \cdot x_l \ . \tag{3.16}$$

and the gradient with respect to the weights is

$$\frac{dO}{dW_l} = \left(\eta_l \cdot (1 - x_l)^T \cdot x_l^T\right)^T \delta_{l+1} + \xi_l^T x_l^T + \delta_{l+1}^T x_l^T \ . \tag{3.17}$$

Like above, ξ_l and δ_l can be factored together resulting in

$$\xi_l + \delta_{l+1} = \frac{\partial R}{\partial z_l} + \left((\xi_{l+1} + \delta_{l+2}) W_l + \eta_l \cdot (\delta_{l+1} W_l) \cdot (1 - 2x_l)^T\right) \cdot (1 - x_l)^T \cdot x_l \ . \tag{3.18}$$

This propagates all the way to $\xi_L + \delta_{L+1}$, which is

$$\xi_L + \delta_{L+1} = \frac{\partial R}{\partial z_l} + \frac{\partial L}{\partial z_{L+1}} + \left((\xi_{l+1} W_l) + \eta_l \cdot (\delta_{l+1} W_l) \cdot (1 - 2x_l)^T\right) \cdot (1 - x_l)^T \cdot x_l \tag{3.19}$$

and thus can be computed efficiently.

Again the equations have a similar structure as back propagation, as the only change was the objective function O becoming more dependent on the networks parameters. Similar to above section, each update term has strong similarity to classical back propagation, however, this time one can think of it as doing 4 back propagation iterations.

An interesting fact that originates from this is that, despite the gradient of R requiring elements of the loss Hessian, the updates still have similar computational complexity as back propagation has, meaning that both extended and classic back propagation have linear runtime in the number of layers.

3.2 Zero-Gradient Training

Above equations of chapter 3.1 enable to train the network on arbitrary functions dependent on δ_i. A very interesting, yet more simple way of using this is to demand

$$\left\| \frac{dL}{dW_i} \right\|^2 = 0 \quad \forall \ i \ , \tag{3.20}$$

where $\|\cdot\|$ is the spectral norm. This is driving the network towards a critical point as $\dot{f}(x) = 0$ is the necessary condition for this. Two interesting implications can be drawn from this. For the pure convex case, above equation would find the same minimum as classic back propagation, however, as high values in the gradient produce a quadratic error,

those weights will be reduced faster compared to the classic loss. Hence, both will converge to the same result, but use different paths on their way. The second implication is active in regions close to a saddle point. The regularization part of the loss is becoming zero in such points, which allows the optimization task get close to it. As the loss is being minimized at the same time, the optimization may skip basins of attraction to a minimum unreachable when only minimizing the loss.

This regularization uses the Hessian to compute an update as can be motivated with a simple example. Assume that $f(x) = x^T Ax$ where $A \in \mathbb{R}^{2\times 2}$ and thus

$$\delta = J_f = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} . \tag{3.21}$$

Minimizing $\|\delta\|^2$, we can compute its derivative

$$\frac{d\delta^T \delta}{dx} = 2 \begin{bmatrix} \frac{\partial f}{\partial x_1}\frac{\partial^2 f}{\partial (x_1)^2} + \frac{\partial f}{\partial x_2}\frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial f}{\partial x_2}\frac{\partial^2 f}{\partial (x_2)^2} + \frac{\partial f}{\partial x_1}\frac{\partial^2 f}{\partial x_1 x_2} \end{bmatrix}^T = 2 \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial^2 f}{\partial (x_1)^2} & \frac{\partial^2 f}{\partial x_1 x_2} \\ \frac{\partial^2 f}{\partial x_1 x_2} & \frac{\partial^2 f}{\partial (x_2)^2} \end{bmatrix} = 2 J_f H_f . \tag{3.22}$$

Here J_f is the Jacobean of f and H_f is its Hessian. This is not the Gauss-Newton update, which is computed using $J_f H_f^{-1}$. Instead, $J_f H_f$ expresses how the gradient will change under its own update. While the example is by no means a mathematical proof, it suggests that performing an update based on this term for each layer of a neural network means taking control of the saturation speed, as the only two ways to have a low gradient in the next iteration are to either classify better or to saturate the units within a layer.

In order to implement the regularization proposed above, (3.20) can be simplified using

$$\left\| \frac{dL}{dW_i} \right\|^2 = \left\| \delta_{i+1}^T x_i^T \right\|^2 = \left\| \delta_{i+1} \right\|^2 \left\| x_i \right\|^2 , \tag{3.23}$$

as the spectral norm, induced by the euclidean norm, of a matrix A is

$$\|A\| = \sqrt{\lambda_{\max}(A^* A)} \tag{3.24}$$

with $\lambda_{max}(\cdot)$ returning the biggest eigenvalue of the passed matrix. This expression can easily be derived with respect to δ_i and x_i and updates can be computed using the extended back propagation equations introduced in section 3.1.

3.3 Good Gradient Regularization

Using above insights the regularization presented next trains a neural network to have 'good gradients', meaning that every layer of the network should change at the same magnitude during each update. This implies that the network has to set its weights in a way that they are large enough to allow deep and shallow layers to affect the loss equally, while at the same

time setting them low enough to not saturate its activation functions, which would prohibit learning and gradient propagation.

This is done in order to address the issue that deeper layers are reported to converge more slowly than shallow ones. A common way to compensate for this is to choose higher learning rates for these deep layers [10, 5]. The effect may origin from the gradient being closer to 0 for those layers, meaning that they don't affect the loss in same magnitude as shallow layers. Yet it should be possible to formulate a regularization that accounts for this effect without a human needed to fine tune each layers learning rate.

Such a regularization would not only allow to train using the same learning rate on all layers, but additionally have the network choose a good initialization similar to unsupervised pre-training. This is expected to increase the speed and quality of training as the loss can sufficiently influence even deep layers right from the beginning.

With such a promising expectation, how could a regularization that is able to enforce 'good gradients' look like? As the top layer has direct influence on the loss, an idea is to compare every other layer with it. Because two layers don't need to have the same amount of neurons, their respective norms are matched, instead of comparing individual units with each other. A regularization that can accomplish this is

$$R = \sum_{l=1}^{L} |\|\delta_l\|^2 - \|\delta_{L+1}\|^2|^2 \ . \tag{3.25}$$

Similar to (3.20), this regularization has the property of not modifying loss function's optimal solutions as all δ_i will be 0 in the optimal case and thus the regularization will be 0 as well.

What can not be expected is that doing above will cause activation functions to escape saturation and become unsaturated if, it is 'stuck' already. Instead, the network should avoid saturation until the loss can not be decreased otherwise.

Therefore, two different experiments were performed. First networks with varying size and depth were pre-trained using this regularization while not considering the loss, to identify if a network can be set up with 'good gradients' prior to training representations. The tested networks had $[1000, 100, 100, 100]$ units and $[3, 5, 10, 19]$ hidden layers and were run for $[1000, 1000, 1000, 5000]$ updates with a fixed step size of $\lambda = 10^{-4}$. All networks were trained on the MNIST data set [11] using stochastic gradient descent and a mini-batch size of 100 random samples.

An additional test was made with the same set up on a network with 6 hidden layers for $10,000$ episodes. The purpose of this test was to evaluate any long-term effects that may origin from training on an objective function that does not directly minimize the loss.

In the second test a pre-trained, deep network was optimized to classify MNIST in order to investigate how a network with 'good gradients' behaves differently from other networks. Th is experiment will be introduced in more details during the next section.

To observe the effectiveness of this regularization and to investigate if a network can train 'good gradients', the $\|\delta_i\|$ values are monitored during training. In the experiment section they will be shown for a selection of the trained networks.

Another plot shown in this context is a plot of each layer's average saturation. It is done to compare with [5] and to see if networks saturate differently when using this regularization. The plot is generated by folding each layers activation at $\sigma(x) = 0.5$. At this point the logistic function is considered most 'active' or unsaturated. $\sigma(x) = 1 \vee \sigma(x) = 0$ are said to be saturated. Folding at 0.5 maps both saturation areas onto a single side allowing for clear visualization of the average saturation.

3.4 Training A Deep Network With Good Gradients

As mentioned earlier, both regularization terms don't change local minimums of the loss, meaning that a minimum of L will also be a minimum of O, however, additional minimums may be created. This fact makes it possible to train a network on the two regularization terms and on the hinge loss at the same time. The combined training is necessary to preserve the 'good gradients' pre-trained with the second regularization, as not accounting for them would likely cause the network to unlearn this property.

To test above theses, the second experiment done for this thesis is to train such a network for many iterations. Using 100 neurons per layer and 8 hidden layers, the network was trained for about 3 weeks computing a total of $258,000$ updates.

The first 500 episodes were used for pre-training on the 'good gradient' regularization. For all following episodes training was done using

$$O(x_i, y, \delta_i) = L(x_{L+1}, y) + \|\delta_{L+1}\|^2 + \frac{1}{100}\sum_{l=1}^{L} |\|\delta_l\|^2 - \|\delta_{L+1}\|^2|^2 \ . \tag{3.26}$$

As computing the entire data set's update is to expensive, stochastic gradient descent on mini-batches was used. A mini-batch was created for each iteration by combining 100 random samples each, having a loss > 0. For descent, a fixed step size of $\lambda = 10^{-4}$ has been used.

The thought process behind the chosen O is to first obtain the 'good gradient' property and then bring down the top layer's error. Focusing on minimizing this norm should also minimize the other layer's norms as each one is constantly adapting to the top layer's norm.

4 Experiments

4.1 Preliminary Experiments

Prior to training networks on the MNIST data set, preliminary tests on a binary classification task have been made. The data was generated by drawing 200 samples from a 2-dimensional Gaussian distribution, with different mean and covariance for each class. The resulting data set with 400 elements was used to train two networks with 3 hidden layers and 100 units per layer, using gradient descent on the entire batch with a step size of $\lambda = 2.5 \cdot 10^{-5}$. One network was trained using classic back propagation on the hinge loss, while the other one was trained on the combination of hinge loss and zero-gradient regularization.

As shown in Figure 4.1 the false classification rate on the training set descents faster if using the regularization, while still resulting in the same quality.

Doing similar tests on MNIST [11], no improvement could be observed, as the tested networks have all failed to converge and continued predicting the same class regardless of the provided input. For pure back propagation the same problem, as described by Glorot et al. [5] was encountered, leading to no new insights.

4.2 Pre-Training Good Gradients

Moving on to the first big experiment on MNIST, training networks showed that for a random initialization $\|\delta_i\|$ decays exponentially from output to input layer. This can be seen in Figures 4.2 and 4.3 shown by the lowest blue line, which represents the first episode.

From here all tests with exception for the long-term test showed similar results, hence only two plots are shown.

Figure 4.2 shows the typical behavior, when pre-training on 'good gradients'. What makes this plot interesting is that despite the 19 hidden layer network not converging to its final solution within 5000 episodes, it is showing nicely how the regularization proceeds, matching norms to the top layer's one after the other. If training had continued until convergence, the green line representing the last episode would be strictly horizontal as it could be observed for the other networks.

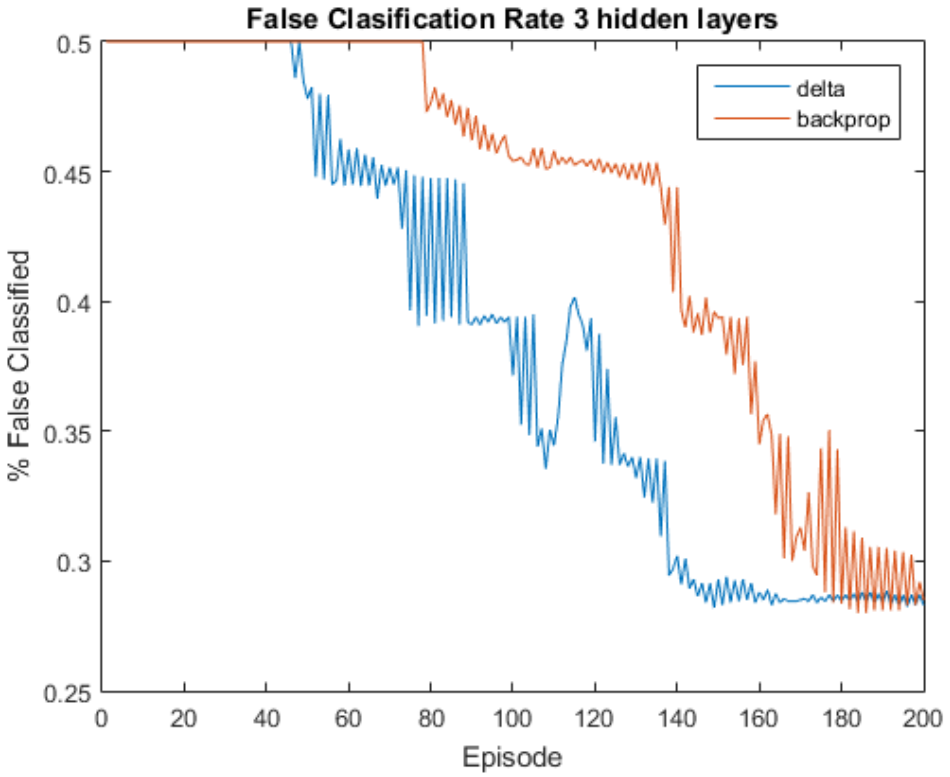


Figure 4.1: The classification error on the training set for both zero-gradient training and standard back propagation, when training on the binary classification task.

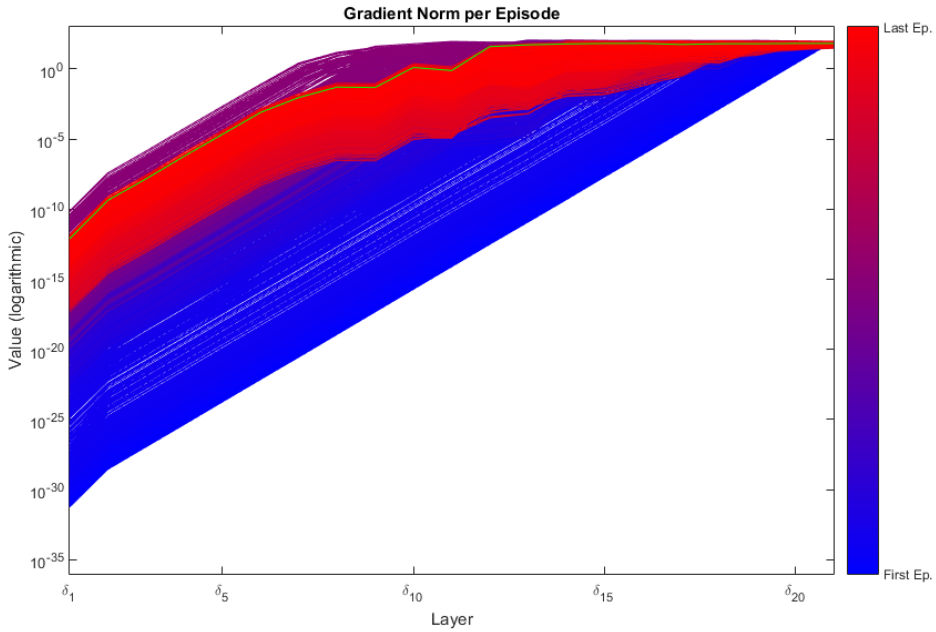


Figure 4.2: Norm of each layers gradient for a 20 layer neural network. It is pre-trained for $5,000$ episodes on the 'good gradient' regularization.

A drawback discovered during training of the network with $10,000$ episodes is that pure optimization on the regularization without minimizing the loss explicitly can lead to saturation of a single hidden layer. This was observed long after the regularization converged to the expected optimum and might be caused by the optimization trying to account for the loss, which is indirectly accessible through δ_i. Saturation causes all layers deeper then the saturated one to influence the loss much less, which is shown in figure 4.3. As assumed in section 3.3, the network did not seem able to escape this saturation as training continued.

After $1,000$ episodes the desired result had been reached which is indicated by the violet lines in Figure 4.3 (b) and is pointed out in (a). Continued training led to instability at about $4,800$ episodes, saturating a hidden layer. This caused other layers to saturate as well, what can be seen in the average activation plot. Before saturation the network behaved exactly as the other tested ones showing both, the straight line in the $\|\delta_i\|$ plot and the about 50% saturated average activation level for each layer.

4.3 Training A Deep Network On MNIST

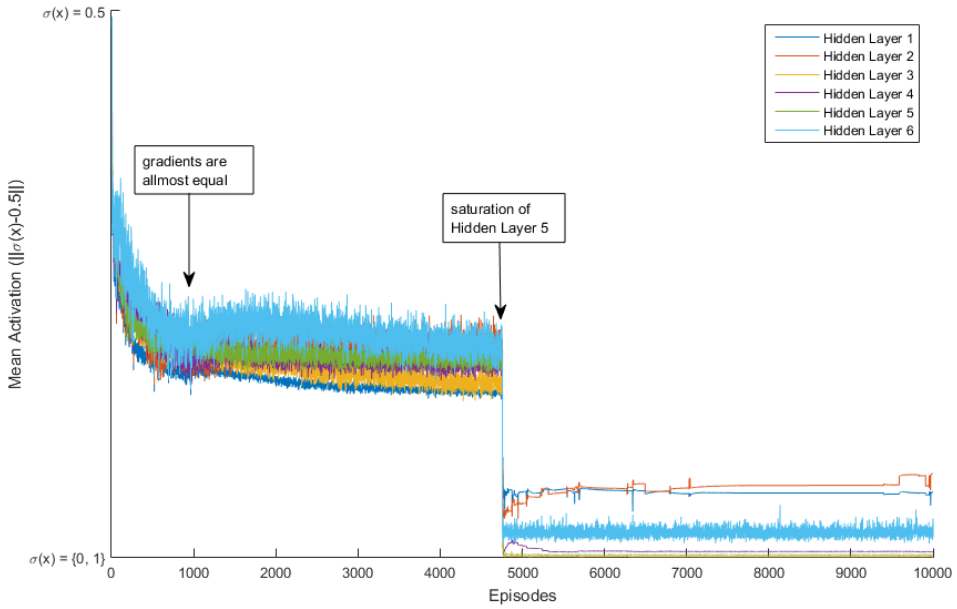
The results of training a neural network described in section 3.4 are presented in figure 4.4 and figure 4.5.

The first one shows the average activation per layer and the loss gradient's norm during training. As implied by the pre-training experiments, layers neither saturate quickly, nor do individual ones go into saturation. This is a direct improvement to the reported performance [5] as the layers now saturate slowly and jointly. The other part of the figure shows how the 'good gradient' property is acquired during the first episodes and then kept, while reducing each layers gradient at the same speed as training proceeds.

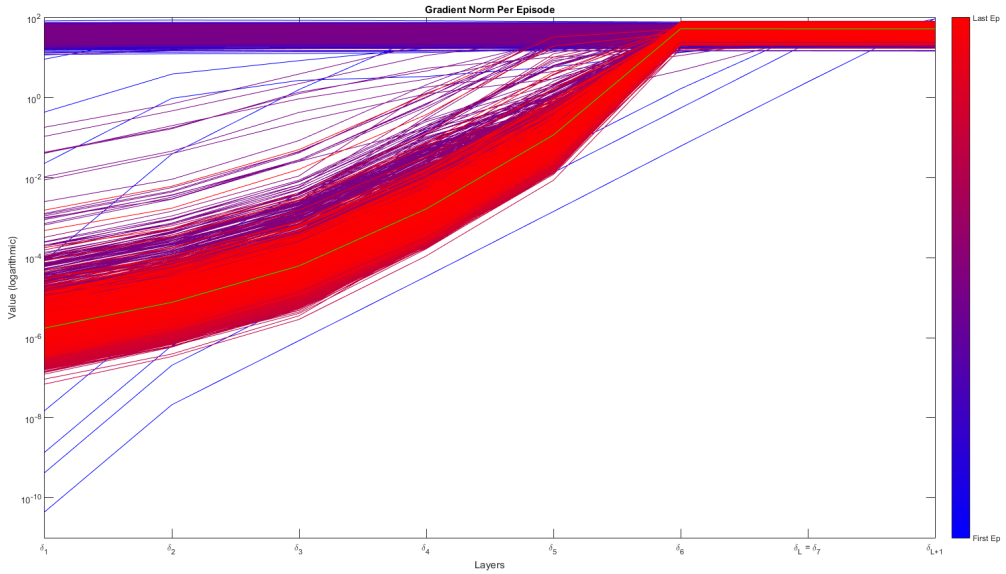
The second figure (4.5) show the false classification rate for both training and test set. This was measured every 1000 iterations having the best performance after around $156,000$ updates with a false classification rate of 28.52% on the test set. Surprisingly the false classification rate on the test set is always within 1% of the training set's rate, indicating that the trained architecture is not overfitting.

The increasing test and training error from $150,000$ episodes onward may indicate that the optimization has not converged yet, as the network's loss showed consistent monotonic behavior. However, as training is really slow on these big tasks, it was canceled prior due to time constraints.

In contrast to the preliminary experiments done using classic back propagation or zero-gradient regularization, the network started to predict different classes for varying inputs, even during early stages of training. This is a significant improvement, given that only logistic activation functions are used, which appeared to be not trainable using classic back propagation when stacked to deep networks.

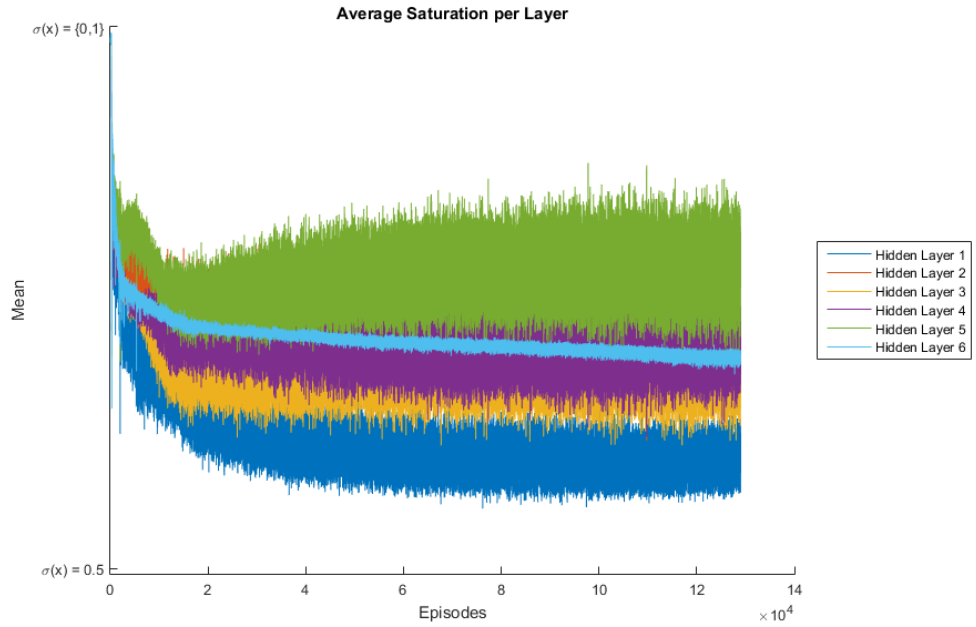


(a) Folded average of activation of each layer.

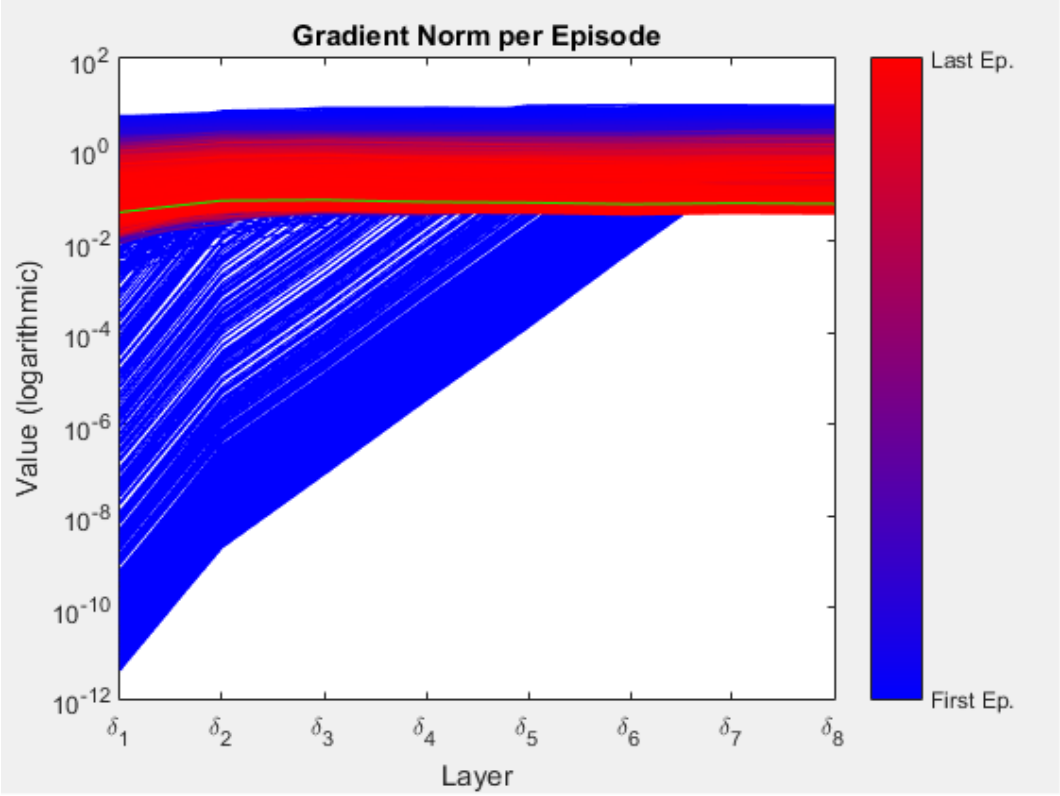


(b) Resulting norm of the gradient in each episode

Figure 4.3: The two plots show the problem when purely training on a good gradient without actively considering the loss. At around 5000 episodes, after training the regularization has long converged (about 1000 episodes), a hidden layer saturates causing bad gradients for all layers deeper down in the achitecture.
TOP: Average activations for each layer. They are generated, by folding the activation values at $\sigma(x) = 0.5$ and averaging the result for each episode.
BOTTOM: The norm of gradients in each layer during training. One line represents one episode moving from blue as the first episode to red in the later stages of training. Green is highlighting the last episode.

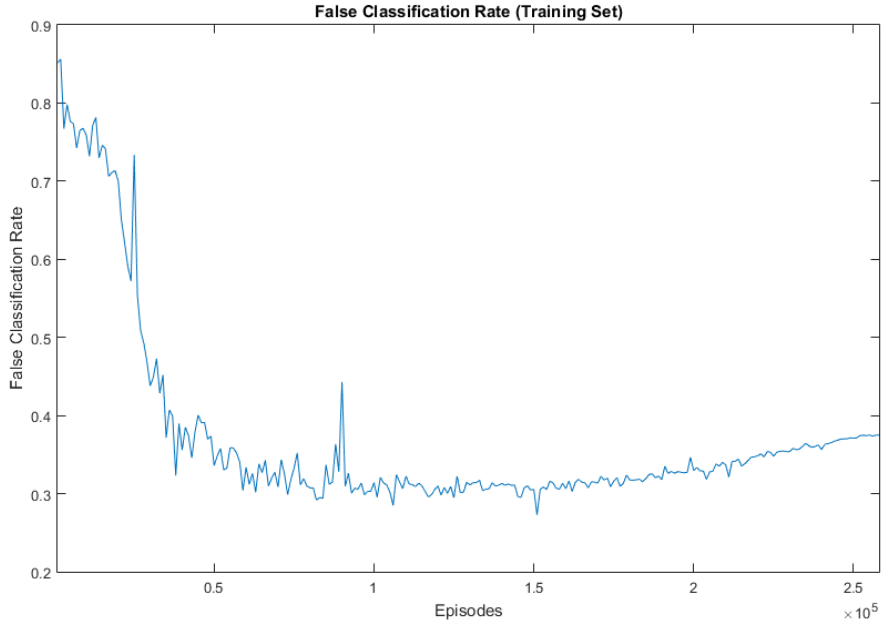


(a) Average Activation per layer (color)

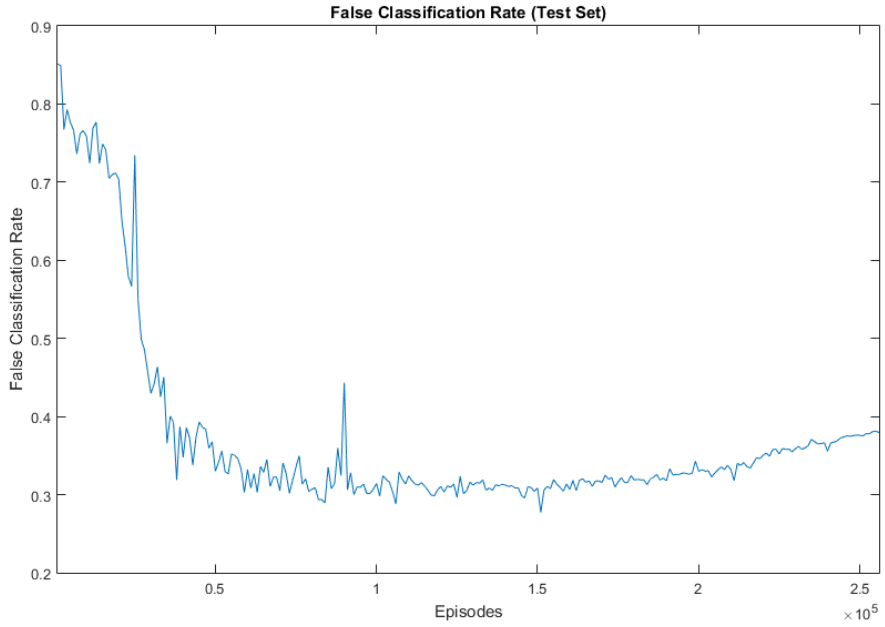


(b) The norm for each layer during time (color)

Figure 4.4: The networks response to training on both the loss and 'good gradients'.
TOP: The Units in each layer learn a representation, while not saturating to much, allowing the gradient to affect deeper layers at the same rate.
BOTTOM: The gradient's norm of each layer during training. Initially the network trains for 'good gradients' then starts to minimize the error preserving the gradients.



(a) False classification rate on the training data (in %)



(b) false classification rate on the training data (in %)

Figure 4.5: The false classification rate for training and test set. Train and test error rate are always within 1% of each other.

5 | Conclusion and Discussion

In this thesis a novel approach on regularizing deep neural networks has been presented. New update equations have been derived to allow regularizing the loss's gradient. Further two regularization methods based on the loss gradient have been presented and discussed. The first, zero-gradient regularization, offers a different path to minimize the loss and to pull the network towards critical points. Latter, introduced as good gradient regularization, trains networks to have good propagation properties. Experiments have been performed, testing the benefits of using these regularizations. Although no new benchmarks could be achieved, significant improvements in training logistic networks were found. The experiments indicate, that regularizing gradient properties is beneficial for pre-training and helps during actual training.

While implementing and testing above equations, it was found that randomly initialized logistic neural networks 'forget' the input values. That means that despite having differences in the beginning, forward propagation assimilates distinct inputs resulting in a similar output and prediction of the same class.

It was shown that this behavior changes, when regularization for 'good gradients' is applied, using above method, which allows these networks to learn more efficiently.

While the focus of this thesis was to examine neural networks with logistic activation functions, it was identified that the choice of non-linearity is the most constraining property in respect to performance. Being both asymmetric around 0 and bounded, logistic activation functions are often found to be slow, poor preforming and not suited for deep neural networks.

Changing the activation function to another one such as the $tanh$ or a symmetric $sigmoid$, may already be enough to overcome this issue resulting in a large performance gain in prediction. Papers like [1, 5, 6, 17] back up this idea recording better results with different activation functions.

Another approach would have been the choice of convolutional neural networks, which have been applied very successfully by LeCunn et al. [9] on the MNIST database, outperforming many other proposed networks. They embed a folding of the input image and have their focus on extracting data from local regions in the image. There is no reason why

these networks can't be combined with the regularizations provided in this thesis potentially achieving even better results.

As a conclusion it can be said that, while adding a constant factor to the complexity of the algorithm, using extended back propagation shows promising results in overcoming the issue of bad propagating gradients, especially in the context of a logistic activation function, where classification is massively increased. It was found that gradient regularization allows for otherwise impossible classification on these deep architectures.

Bibliography

[1] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Juergen Schmidhuber. Deep big simple neural nets excel on handwritten digit recognition. *arXiv preprint arXiv:1003.0358*, 2010.

[2] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[4] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[6] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[8] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *Int. J. Artificial Intell. Expert Syst*, 1(4):111–122, 2011.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[10] Yann LeCun, Leon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.

[11] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[12] Kaare Brandt Petersen, Michael Syskind Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7:15, 2008.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5:3, 1988.

[14] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM, 2007.

[15] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.

[16] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[17] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart,

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart,