
**BRNO UNIVERSITY OF TECHNOLOGY**
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

**FACULTY OF INFORMATION TECHNOLOGY**
**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

**DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA**
**ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ**


# ADVANCED VISUALIZATION OF NEURAL NETWORK TRAINING

**POKROČILÁ VIZUALIZACE TRÉNOVÁNÍ NEURONOVÝCH SÍTÍ**

**BACHELOR'S THESIS**
**BAKALÁŘSKÁ PRÁCE**


**AUTHOR** **SAMUEL KUCHTA**
**AUTOR PRÁCE**

**SUPERVISOR** **Ing. KAREL BENEŠ**
**VEDOUCÍ PRÁCE**



**BRNO 2023**

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

# Bachelor's Thesis Assignment

147671

| | |
|---|---|
| Institut: | Department of Computer Graphics and Multimedia (UPGM) |
| Student: | **Kuchta Samuel** |
| Programme: | Information Technology |
| Specialization: | Information Technology |
| Title: | **Advanced Visualization of Neural Network Training** |
| Category: | Artificial Intelligence |
| Academic year: | 2022/23 |

Assignment:

1. Get acquinted with training of modern neural networks
2. Get acquinted with methods for visualization of training progress
3. Propose inspection-worthy aspects of the training
4. Visualize them using suitable data and model architectures
5. Evaluate the results

Literature:

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep Learning. MIT Press, 2016.
- Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe: Qualitatively characterizing neural network optimization problems. 2015.

Requirements for the semestral defence:
1, 2 and partial work on 3 and 4

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

| | |
|---|---|
| Supervisor: | **Beneš Karel, Ing.** |
| Head of Department: | Černocký Jan, prof. Dr. Ing. |
| Beginning of work: | 1.11.2022 |
| Submission deadline: | 17.5.2023 |
| Approval date: | 31.10.2022 |

# Abstract

This work aims to propose visualization methods and analyze with them the phenomena arising during the training of neural networks, based on which new knowledge regarding deep learning could be discovered. In this work, a program was created that tests the impact of training using different techniques and visualizes the training results using different methods. This work presents two methods of visualization of the training process. The first method displays the area around the path of the trained model by averaging the path's points weighted by their distance from the displayed point. The second method is to display step sizes during learning. The result of the work is represented by graphs and a discussion of the phenomena captured by the visualizations.



# Abstrakt

Cílem této práce je navrhnout metody vizualizace, a analyzovat nimi jevy vznikající během trénování neuronových sítí, na základě kterých by mohly být zjištěny nové poznatky ohledně hlubokého učení. V této práci byl vytvořen program, který testuje dopad na trénování za použití různých technik, a vizualizuje výsledky trénování pomocí různých metod. Tato práce představuje dvě metody vizualizace tréninkového procesu. První metoda je zobrazení plochy okolo cesty trénovaného modelu pomocí průměrování bodů cesty váhovanými jejich vzdáleností od zobrazovaného bodu. Druhá metoda je zobrazení velikosti kroků během učení. Výsledek práce je znázorněn grafy, a diskuzí nad jevy zachycenými vizualizacemi.



# Keywords

Neural networks, gradient size, machine learning, loss function.

# Klíčová slova

Neuronové sítě, velikost gradientu, strojové učení, účelová funkce.


# Reference

KUCHTA, Samuel. *Advanced Visualization of Neural Network Training.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Karel Beneš

# Rozšířený abstrakt

Se vzrůstajícím výpočetním výkonem, objevováním efektivnějších algoritmů, a nasazením do aplikací, je strojové učení populárnější než kdy předtím. I přes oblibu, se v oblasti trénování vyskytuje mnoho nejasností.

Tato práce se zabývá trénováním neuronových sítí a vizualizací tréninkového procesu z různých úhlů pohledu a pomocí různých metod. Tyto metody se poté porovnají a vybere se vhodnoá metoda pro zobrazování průběhu trénování za účelem hlubšího zkoumání trénování vybraných neuronových sítí.

Teoretická část začíná úvodem do neuronových sítí, včetně diskuze o algoritmech v strojovém učení. Pozornost je pak věnována vizualizaci neuronových sítí. Práce představuje různé metody pro vizualizaci průběhu trénování, a vizualizace konvolučních vrstev a jejich výstupů, a navrhuje nové metody pro vizualizaci trénování neuronových sítí. Tyto vizualizace zahrnují analýzu účelové funkce kolem trénovací cesty, a zobrazení velikosti kroků provedených během učení.

Experimentální část práce se zaměřuje na modely konvolučních neuronových sítí, trénování pomocí stochastic gradient descent (SGD), a adaptive moment estimation (ADAM) algoritmů, a využití regularizační techniky dropout.

Na základě tří-rozměrné metody vizualizace okolí kolem trénovací cesty zobrazené pomocí analýzy hlavních komponent (PCA), je navržena metoda pro zobrazení okolí cesty, která co nejvěrněji kopíruje jak cestu trénování, tak dvou-dimenzionální vizualizaci lineární interpolace parametrů mezi počátečním a koncovým bodem trénování. Tato metoda byla úspěšně implementována s využitím průměrování bodů na cestě, kterou ušel trénovací algoritmus, váhovanými jejich vzdáleností od vykreslovaného bodu, pro výpočet jeho parametrů, nad kterými se počíta výsledek účelové funkce. Experiment potvrzuje nečekané chování odvozené z menší míry vysvětlitelnosti pomocí PCA u ADAM algoritmu v porovnání s SGD tím, že se porovná míra rozdílu vizualizace užitím dvou-dimenzionálních a mnoho-dimenzionálních vzdáleností. Výsledky tohoto experimentu korespondují s výsledky vysvětlitelnosti variance pomocí PCA.

Další zkoumanou oblastí v průběhu trénování neuronové sítě je zkoumání velikosti kroků během trénování. Běžné zobrazení ztráty (loss) nemusí vždy poskytovat dostatečný obraz o problémech během trénování. Pomocí zobrazení velikosti kroků, případně její porovnání se ztrátou, se přišlo na jevy, které umožňuji zefektivnit průběh trénování.

Experimenty ukázali korelaci mezi ztrátou (loss) a velikostí kroků, a možnost předběžně předpovědět, jestli má model potenciál se dobře natrénovat. Taktéž se podařilo zefektivnit trénování využitím znalosti velikosti kroků.

# Advanced Visualization of Neural Network Training

## Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Karel Beneš. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Samuel Kuchta
May 16, 2023


## Acknowledgements

My sincere gratitude goes to my supervisor, Ing. Karel Beneš, who has always been more than willing to provide me with guidance, feedback and materials crucial to writing this bachelor thesis. I also wish to thank my family for their continuous support.

# Contents

# Chapter 1

# Introduction

Visualization techniques are becoming increasingly popular in machine learning to improve the understanding of these complex models by visually representing the model's performance over time, usually using plots and graphs. Neural network visualization has various benefits, including assisting in understanding the network's architecture, identifying and diagnosing problems, presenting the network's complexity, pinpointing areas for improvement, and providing opportunities for experimentation and exploration of network behaviour and architectures. The history of neural networks dates back to the 1940s when Warren McCulloch and Walter Pitts proposed a model of artificial neurons which used thresholding logic, meaning that the network's output was binary 0 or 1 [16]. In the 1950s, researchers began developing computational machines to simulate neural networks. In 1954, Farley and Clark first used computational machines to simulate a Hebbian network [4]. It is based on the idea that when two neurons are activated together, the connection between them is strengthened. This is known as Hebb's rule. Donald Hebb first proposed the theory in 1949 [8].

In 1958, Rosenblatt created the perceptron, an algorithm for pattern recognition [22]. In the 1960s and 1970s, researchers developed more complex neural network models, such as the multi-layer perceptron. The backpropagation algorithm was applied in 1982 as a new way to train these more complex models [27]. However, neural networks fell out of favour in the 1980s due to their inability to solve complex problems. In the 1990s, researchers developed new neural network architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), that could solve more complex problems.

The field of connectionism emerged in the 1980s [17], emphasizing the study of neural networks and their ability to simulate human cognition. Researchers started exploring ways to visualize the activation patterns of neurons and their connections to gain insights into how the networks were functioning. As neural networks became more complex, visualization techniques started to focus on understanding the internal representations learned by the networks. In the 2010s [30, 29, 31, 20], deep learning experienced a resurgence in popularity due to advancements in hardware and large-scale datasets. As deep neural networks with numerous layers became more prevalent, there was an increased demand for visualization techniques to interpret and understand their behaviour.

This study examines various methods for visualizing the neural network training process. These techniques include plotting the training process using linear path interpolation, displaying the region surrounding the path of the trained model through principal component analysis, and presenting the step size during learning. The results show that visualization of neural network training is valuable for understanding model behavior, identifying phenomena during training, and improving neural network performance.

# Chapter 2

# Neural Networks

Neural networks are used in various applications, including image and speech recognition, natural language processing, and autonomous vehicles. They are especially useful for tasks that involve complex patterns and relationships in data, where traditional machine-learning models may struggle to find meaningful insights.

Neural networks are a machine learning model inspired by the structure and function of the human brain. They consist of layers of interconnected nodes or "neurons" that process information in a way that allows the network to learn and make predictions based on the input data.

Each neuron in a neural network receives input from neurons in the previous layer, processes the input using a set of weights and biases, and then passes the output to the next layer. This process continues until the last layer produces an output or prediction.

During training, the weights and biases of the neurons are adjusted based on the error between the predicted output and the target output. This process allows the network to learn and improve its predictions over time.

## 2.1 Learning Techniques

Learning is essential to machine learning because it allows machines to learn from data and improve their performance over time. Neural Networks are used for tasks like regression and classification. For these tasks, functions such as linear regression, logistic regression, and softmax function are used. Three main learning techniques in Machine Learning are supervised, unsupervised, and reinforcement learning.

### Supervised Learning

In Supervised Learning, the machine is trained on labelled data, meaning the input data is paired with a corresponding output or label. The goal is to learn the mapping between input and output data so that the machine can accurately predict new, unseen data. Supervised learning is often used for tasks such as image and speech recognition.

*Linear regression* is a well-known supervised learning algorithm that predicts a scalar value $y$ based on an input vector $\mathbf{x}$. The algorithm computes the output as an approximation of $y$ through linear regression of the input. The formula for calculating the output is given by

$$y = \mathbf{w}^T\mathbf{x} + b, \tag{2.1}$$

where $b$ is the bias term, $\mathbf{w}$ is the vector of weights, and $\mathbf{x}$ is the input vector. The weights in the vector of parameters determine the contribution of each feature to the prediction.

Logistic regression serves as a classifier in machine learning, even though it is classified as a statistical regression technique. Logistic regression enables the interpretation of feature importance within a given dataset, with the number of inputs in the model equaling the number of features in the dataset. Logistic regression is a generalized linear model for predicting the probability of an instance belonging to a particular class in a classification problem. The logistic function, also known as the sigmoid function, is defined as:

$$P(y = 1|x) = \frac{1}{1 + e^{-(z)}}, \tag{2.2}$$

where $z = \mathbf{w}^T\mathbf{x} + b$.

The softmax function is used in supervised learning for multi-class classification. It is used to convert the output of a linear function into probabilities that can be used to classify the input into one of the classes. Softmax function converts a vector of $K$ real numbers into a probability distribution of $K$ possible outcomes. It is often used as the last activation function of a neural network to convert the network output into a probability distribution over predicted output classes. The softmax function is defined as follows:

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}, \tag{2.3}$$

where $\mathbf{z}$ is a vector of K real numbers, and $\text{softmax}(\mathbf{z})_i$ is the i-th element of the resulting probability distribution.

## Unsupervised Learning

In unsupervised learning, the machine is trained on unlabeled data, meaning there are no predefined labels for the input data. The goal is to find patterns or structures in the data, such as clusters or groups of similar data points [6]. Unsupervised learning is used for anomaly detection and data clustering.

Principal component analysis (PCA) is an unsupervised learning algorithm for multivariate data analysis. In such data, the observations are typically described by interdependent quantitative variables. The main goal of PCA is to extract critical information from a set of input data and represent it as a set of new orthogonal variables known as principal components, thus creating a new coordinate system where the principal components are derived as linear combinations of the original variables. This way, patterns of similarity between observations and variables can be identified. The PCA method relies on two mathematical techniques: eigendecomposition of positive semidefinite matrices and singular decomposition of rectangular matrices.

PCA has several purposes, including simplifying and reducing the size of a dataset by removing redundancy and keeping only the most relevant information while preserving as much of the original variance in the data as possible, analyzing the structure of variables and observations, and building predictive models. Neural networks can be initialized using PCA [23].

**function** PCA($X$, $k$)
  $\bar{\mathbf{X}} \leftarrow X - \frac{1}{n}\mathbf{1}_n(\mathbf{1}_n)^\top X$
  $C \leftarrow \frac{1}{n}\bar{\mathbf{X}}^\top\bar{\mathbf{X}}$
  $Y \leftarrow \bar{\mathbf{X}}\mathbf{W}$
  **return** $Y$
**end function**

Where $X$ is the input dataset, $k$ is the number of principal components to be obtained, $\bar{\mathbf{X}}$ is the mean-centered data matrix, $\mathbf{1}_n$ is a vector of length $n$ with all elements equal to 1, $C$ is the covariance matrix of $\bar{\mathbf{X}}$, $\mathbf{W}$ is the matrix of eigenvectors corresponding to the $k$ largest eigenvalues, $Y$ is the projected data matrix representing the original data in a lower-dimensional space spanned by the principal components.

PCA is a valuable data visualization tool because it can reduce high-dimensional data to a lower dimension that can be plotted on a graph.

## Reinforcement Learning

In reinforcement learning, the machine learns by trial and error. A machine is given a goal or objective and takes action in the environment to achieve that goal. It receives feedback in the form of rewards or penalties based on its actions, and the goal is to learn a policy that maximizes cumulative reward over time. Reinforcement learning can be used with deep neural networks to solve complex problems like game playing and robotics.

## 2.2 Neural Networks

The computational power of neural networks comes from the density and complexity of their interconnections.

The input to a neuron is the dot product of two vectors: the inputs $x$ and the weights $w$. The output of the neuron is the value of its activation function:

$$y = f(\mathbf{w}^T\mathbf{x} + b), \tag{2.4}$$

where $f$ is the activation function, $w$ is the vector of weights, $x$ is the input data vector, and $b$ is the bias. Notice that the only difference between neural network and *linear regression* is the usage of the activation function.

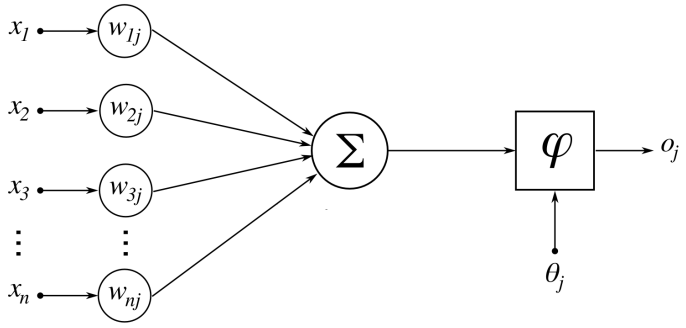



Figure 2.1: Model of an artificial neuron, where $\mathbf{x}$ is a set of inputs, $\mathbf{w}$ is a set of weights. The input of the neuron is the dot product of these two vectors. $\varphi$ is an activation function, $\theta$ is threshold and $o$ is the output [3].

Neurons are grouped into layers in a neural network. Activation functions are used to propagate the output of one layer forward to the next layer. They take a scalar output of a neuron and yield another scalar as an activation of the neuron. Activation functions introduce nonlinearity into the model and define its behaviour.

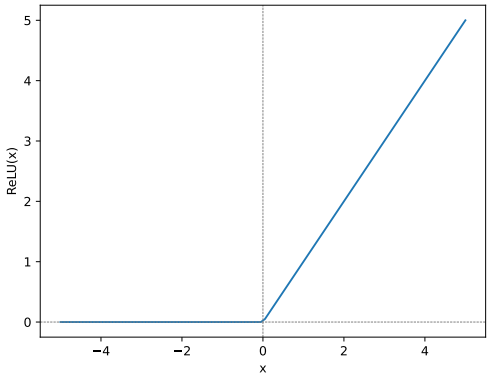



Figure 2.2: ReLU activation function

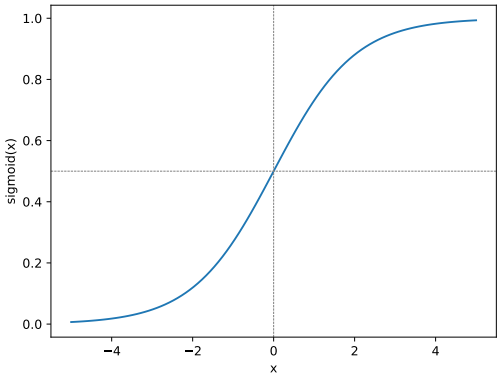



Figure 2.3: Sigmoid activation function

### Feed-forward Neural Networks

A feed-forward neural network (FNN) is an artificial neural network where node connections do not form a cycle. In this network, information moves forward from the input nodes, through any hidden nodes, and to the output nodes.

A basic form of Feed-forward Neural Network (FNN) is a linear network that comprises only one layer of output nodes. The inputs are directly connected to the outputs through a set of weights.

FNNs can have multiple layers, with each layer consisting of multiple neurons. The output of one layer serves as the input to the next layer. The last layer of neurons produces the final output.

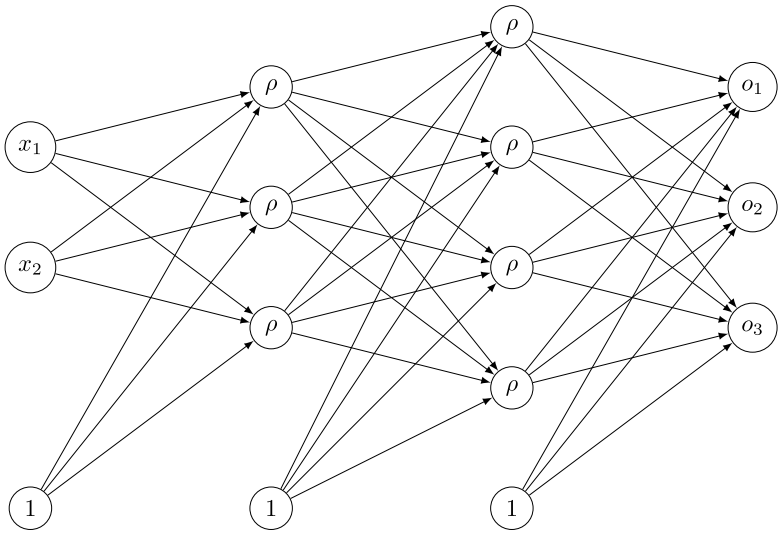



Figure 2.4: Feed forward neural network with two hidden layers. Where **x** represents input data, $\rho$ represents hidden neurons, **o** represents the output. The arrows indicate the direction of data flow inside the network.

matrix representation of the neural networks in Figure 2.4 might look as follows:

$$\begin{pmatrix} x_0 & x_1 & 1 \end{pmatrix}^\top \begin{pmatrix} \Theta_{00}^{(1)} & \Theta_{01}^{(1)} & \Theta_{02}^{(1)} \\ \Theta_{10}^{(1)} & \Theta_{11}^{(1)} & \Theta_{12}^{(1)} \\ \Theta_{20}^{(1)} & \Theta_{21}^{(1)} & \Theta_{22}^{(1)} \end{pmatrix} \xrightarrow{\varphi} \begin{pmatrix} \rho_0^{(2)} & \rho_1^{(2)} & \rho_2^{(2)} & 1 \end{pmatrix}^\top \begin{pmatrix} \Theta_{00}^{(2)} & \Theta_{01}^{(2)} & \Theta_{02}^{(2)} & \Theta_{03}^{(2)} \\ \Theta_{10}^{(2)} & \Theta_{11}^{(2)} & \Theta_{12}^{(2)} & \Theta_{13}^{(2)} \\ \Theta_{20}^{(2)} & \Theta_{21}^{(2)} & \Theta_{22}^{(2)} & \Theta_{23}^{(2)} \\ \Theta_{30}^{(2)} & \Theta_{31}^{(2)} & \Theta_{32}^{(2)} & \Theta_{33}^{(2)} \end{pmatrix} \xrightarrow{\varphi}$$

$$\xrightarrow{\varphi} \begin{pmatrix} \rho_0^{(3)} & \rho_1^{(3)} & \rho_2^{(3)} & \rho_3^{(3)} & \rho_4^{(3)} & 1 \end{pmatrix}^\top \begin{pmatrix} \Theta_{00}^{(3)} & \Theta_{10}^{(3)} & \Theta_{20}^{(3)} & \Theta_{30}^{(3)} & \Theta_{40}^{(3)} \\ \Theta_{01}^{(3)} & \Theta_{11}^{(3)} & \Theta_{21}^{(3)} & \Theta_{31}^{(3)} & \Theta_{41}^{(3)} \\ \Theta_{02}^{(3)} & \Theta_{12}^{(3)} & \Theta_{22}^{(3)} & \Theta_{32}^{(3)} & \Theta_{42}^{(3)} \end{pmatrix} \xrightarrow{\varphi} \begin{pmatrix} o_0 & o_1 & o_2 \end{pmatrix}^\top$$

### Recurrent Neural Network

A recurrent neural network (RNN) is an artificial neural network where node connections can form a cycle, which allows output from some nodes to affect subsequent input to the same nodes, allowing the network to exhibit temporal dynamic behaviour.

RNNs can use their internal state (memory) to process variable-length sequences of inputs. This makes them applicable to unsegmented, connected handwriting or speech recognition tasks [14].

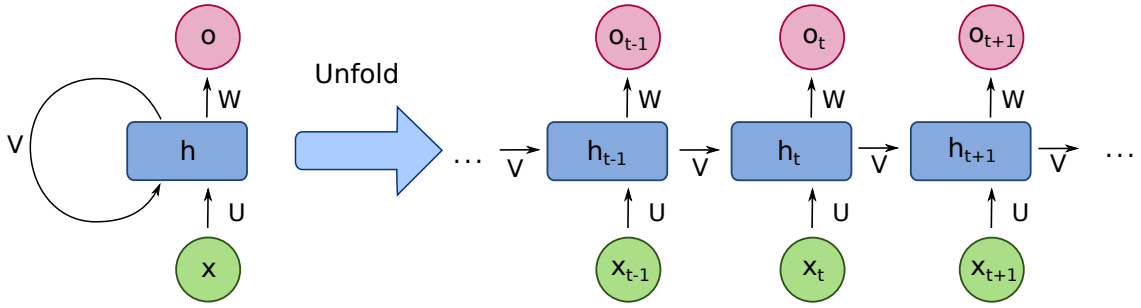



Figure 2.5: Recurrent neural network, and its representation unfolded in time [5].

$$\mathbf{h_t} = f(\mathbf{U} \cdot \mathbf{x_t} + \mathbf{V} \cdot \mathbf{h_{t-1}} + \mathbf{b}), \tag{2.5}$$

$$\mathbf{o_t} = f(\mathbf{W} \cdot \mathbf{h_t}), \tag{2.6}$$

where for each time step t, $\mathbf{h}$ is the hidden internal state vector, $\mathbf{x}$ is the input vector, $\mathbf{U}$, $\mathbf{W}$ and $\mathbf{V}$ are weight matrices, $\mathbf{b}$ is the bias vector, $f$ is the activation function.

### Convolutional Neural Network

A convolutional Neural Network (CNN) is an artificial neural network commonly applied to analyze visual imagery. CNNs use a mathematical operation called cross-correlation instead of general matrix multiplication in at least one of their layers. They are specifically designed to process pixel data and are used in image recognition and processing. CNNs have applications in image and video recognition, recommender systems, image classification, image segmentation, and medical image analysis.

Convolutional Neural Networks consist of convolutional, pooling, and fully connected layers. The convolutional layers are responsible for extracting features from the input images. In contrast, the pooling layers reduce the dimensionality of the feature maps, and the fully connected layers perform the classification.

2D cross-correlation between two matrices is represented in equation 2.7.

$$f[x,y] * g[x,y] = \sum_i \sum_j f[i,j] \cdot g[x-i, y-j], \tag{2.7}$$

where the $f[x,y]$ is original data sample, $g[x,y]$ is filter, the indices $i$ and $j$ represent the coordinates within the matrices $f$ and $g$, respectively.

In the context of the convolutional layer, as depicted in Figure 2.6, an essential step involves performing a multiplication operation between the input data values and a convolutional kernel or filter. This multiplication yields a single value called an activation map or feature map, which serves as the layer's output. The network weights, represented by the numbers within the filter, are updated after each optimizer step.

To construct the convolutional layer's output, each filter's feature maps are stacked together along the depth dimension. Pooling layers are employed to reduce the dimensionality of the representation. These layers operate on each feature map individually, scaling down its dimensionality. Typically, the pooling operation involves using the max function to select the maximum value from the pooled vector. The size of the pooled vector is determined by the kernel size, which is typically the same as the size of the kernel used in the convolutional layer and the subsequent pooling layer.

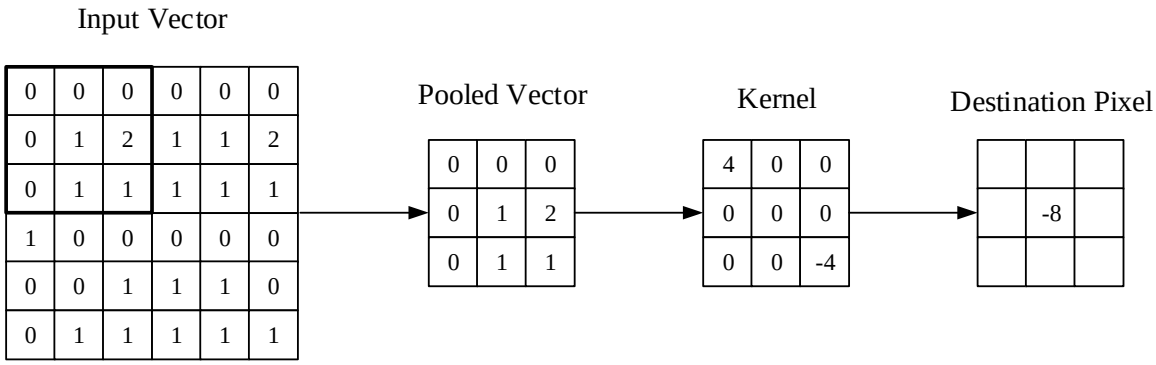



Figure 2.6: Representation of a convolution operation in a convolutional layer [21].

## 2.3 Neural Network Training

Neural network training is the process of finding values of parameters in a neural network that minimizes the error between the predicted output of the network and the target output. The loss function measures the discrepancy between predicted and target outputs, while the cost function evaluates the overall performance by averaging the loss functions. The gradient, calculated through backpropagation, guides weight and bias updates to minimize the loss function and improve the neural network's performance.

### Loss and Cost Functions

A *loss function* is a mathematical function that measures the difference between the predicted output of a neural network and the target output. The goal of training a neural network is to minimize the loss function by adjusting the parameters of the network. Many different types of loss functions can be used depending on the problem being solved.

The *cost function* averages all loss functions for all training examples. It is used to evaluate the performance of a model over a training dataset by comparing its predictions with the target values.

### Gradient

The gradient is a vector of partial derivatives of a function with respect to its inputs. In the context of neural networks, the gradient is used to update the weights and biases of the network during training.

$$\nabla f(\mathbf{w}) = (\frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, ..., \frac{\partial f}{\partial w_n}), \tag{2.8}$$

where $f$ is a function of $n$ variables and $\nabla f(x)$ is the gradient of $f$ at point $\mathbf{w}$.

### Backpropagation

*Backpropagation* is a method used to calculate the gradient of the loss function with respect to each weight in the neural network, pointing in the direction of the value that maximizes the loss function. Backpropagation relies on the chain rule of calculus to calculate the gradient backwards through the layers of a neural network. The gradient is then used to update every weight individually to gradually reduce the loss function over many training iterations.

The algorithm involves first calculating the derivatives at layer $L$, which is the last layer. These derivatives are an ingredient in the chain rule formula for layer $L-1$, so they can be saved and reused for the second-to-last layer. Thus, backpropagation works its way backwards through the network from the last layer to the first layer:

$$z^{(L)} = \mathbf{w}^{(L)} a^{(L-1)} + b^{(L)}, \tag{2.9}$$

$$a^{(L)} = f(z^{(L)}), \tag{2.10}$$

$$\frac{\partial C}{\partial \mathbf{w}^{(L)}} = \mathbf{w}^{(L)} \cdot \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C}{\partial a^{(L)}}, \tag{2.11}$$

$$\frac{\partial C}{\partial b^{(L)}} = \frac{\partial a^{(L)}}{\partial z^{(L)}} \cdot \frac{\partial C}{\partial a^{(L)}} \tag{2.12}$$

where $\mathbf{w}$ represents the weights, $b$ represents the bias, f represents the activation function, $a$ represents the output of the activation function of a neuron, and $L$ represents the layer, $C$ represents the cost[1].

[1] https://www.3blue1brown.com/lessons/backpropagation-calculus

# Chapter 3

# Neural Network Training Techniques

Choosing an appropriate training technique for a given neural network can significantly impact the model's convergence rate and final performance. Different techniques may better suit different data types, model architectures, and optimisation goals. It may be necessary to experiment with multiple techniques to find the one that works best for a particular problem. Overall, neural network training techniques are a crucial part of the machine learning process. They play a vital role in enabling neural networks to learn from data and improve their performance over time.

## 3.1 Neural Network Optimization Algorithms

The purpose of neural network training optimization algorithms such as SGD (Stochastic Gradient Descent) or Adam is to find the optimal set of weights and biases that minimize the loss function and improve the network's performance. These algorithms iteratively update the network's parameters based on the gradient of the loss function, allowing the network to converge toward a better solution during the training process.

### Stochastic Gradient Descent

A commonly used optimisation algorithm is stochastic gradient descent (SGD). In each iteration, SGD uses a randomly selected batch of data (also known as a "mini-batch") obtained from the dataset to compute the gradient of the loss function. It then updates the parameters in the direction of a negative gradient that reduces the loss. One of the main advantages of SGD is that it can be used to train large machine learning models on massive datasets because it processes the data in small mini-batches rather than the entire dataset at once, making it computationally efficient, although sensitive to the choice of learning rate. Therefore, it may require careful tuning to achieve good performance.

Pseudo-code of SGD algorithm.:

**function** SGD($\theta_0$, $N$, D)
  $t_0 \leftarrow 0$
  **while** t != N **do**
    $L \leftarrow feedforward\_pass(\theta_t, D_t)$
    $\nabla f(\theta_t) \leftarrow backpropagation(\theta_t, L)$
    $\theta_{t+1} \leftarrow \theta_t - \alpha \nabla f(\theta_t)$
    $t \leftarrow t + 1$
  **end while**
  **return** $\theta_{t+1}$
**end function**

where $N$ is the number of iterations, $D$ is a dataset, $D_t$ is a batch of the dataset, $L$ is loss, $\nabla f(\theta_t)$ is the gradient of the loss function concerning parameters $\theta$, and $\alpha$ is the learning rate.

### Adaptive Learning Rate Methods

Adaptive learning rate methods adjust the learning rate of the optimiser based on training data using strategies such as momentum or learning rate decay.

*Momentum* accelerates the training process. It involves adding a momentum term to the gradient descent update rule, which helps the optimisation algorithm to move more efficiently through the weight space.

*Learning rate decay* improves the convergence of the optimisation algorithm. It involves decreasing the learning rate over time, which can help the algorithm more reliably find a good solution.

Several adaptive learning rate strategies are commonly used, and ADAM is one of the most commonly used optimization algorithms using these strategies.

### Adam

Adam (Adaptive Moment Estimation) is an optimisation algorithm used in machine learning to update the parameters of a model. It is an extension of stochastic gradient descent that uses moving averages of the parameters to provide a running estimate of the second raw moments of the gradients; the term adaptive in the name refers to the fact that the algorithm "adapts" the learning rates of each parameter based on the historical gradient information.

Adam was introduced in a 2014 [10]. It has become one of the most popular optimisation algorithms for training deep learning models due to its efficiency and good performance.

Adam works by maintaining an exponentially decaying average of past gradient information and an exponentially decaying average of past squared gradient information. The algorithm uses these averages to adjust each parameter's learning rate and helps prevent oscillations and divergence of the model's parameters. Adam also includes a bias correction term to help the algorithm converge more quickly when the data is sparse.

Pseudo-code of ADAM algorithm [26].:

**function** ADAM($\theta_0$, $N$, D)
  $m_0 \leftarrow 0$, $v_0 \leftarrow 0$, $\hat{m}_0 \leftarrow 0$, $\hat{v}_0 \leftarrow 0$, $t_0 \leftarrow 0$
  **while** t != N **do**
    $L \leftarrow feedforward\_pass(\theta_t, D_t)$
    $\nabla f(\theta_t) \leftarrow backpropagation(\theta_t, L)$
    $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1)\nabla f(\theta_t)$
    $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2)(\nabla f(\theta_t))^2$
    $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$
    $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$
    $\theta_{t+1} \leftarrow \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$
    $t \leftarrow t + 1$
  **end while**
  **return** $\theta_{t+1}$
**end function**

where $N$ is the number of iterations, $D$ is a dataset, $D_t$ is a batch of the dataset, $L$ is loss, $m$ and $v$ are the first and second moments of the gradient, respectively, $\beta_1$ and $\beta_2$ are hyperparameters that control the decay rates of these moments, $\epsilon$ is a small constant to avoid division by zero, $\nabla f(\theta_t)$ is the gradient of the loss function concerning parameters $\theta$, $\hat{m}$ and $\hat{v}$ are bias-corrected estimates of $m$ and $v$, and $\alpha$ is the learning rate.

## 3.2 Neural Network Regularization

*Regularisation* is a technique used to improve the generalisation of a machine-learning model by reducing the complexity of the model and preventing overfitting, thus making it perform well not only on the training data but also on new, unseen data. Several types of regularisation techniques can be used to prevent overfitting, including L1 and L2 regularisation, which add a penalty to the model's loss function based on the absolute and squared values of the model's parameters, respectively, or data augmentation, which artificially increases the size of the training dataset by generating new, synthesised data points from the existing data. A commonly used regularisation technique is a dropout. Regularisation techniques can be used alone or in combination, depending on the problem's specific requirements and the dataset's characteristics.

### Dropout

Dropout prevents co-adaptation[1] of the units [25] by randomly dropping out (setting to zero) a specified percentage of the model's units (neurons) during each update of the training. This means that the units are used neither in the forward nor backward passes of the model, and their weights are not updated during training. Dropout is being applied to the network's values; therefore, it can be applied to inputs, and most layers, such as fully connected, convolutional, and recurrent layers.

[1] Interdependencies, where the activation of one neuron becomes highly correlated with the activation of another.

It is vital to carefully tune the dropout rate, as a too-high dropout rate can lead to underfitting, while a too-low dropout rate may not provide sufficient regularisation. Dropout is typically used during training, although a technique called MC-dropout (Monte Carlo Dropout) also allows dropout to be used during inference. It involves performing multiple forwards passes through the network with dropout active and stochastically sampling predictions at each pass. By averaging the predictions obtained from multiple passes, MC-Dropout provides a way to estimate model uncertainty and obtain more robust predictions.

## 3.3 Neural Network Normalization

Normalization is a valuable preprocessing technique that helps ensure efficient and effective training of neural networks, improves generalization and enhances the network's ability to handle varying input conditions by scaling and shifting data values to a standard range without distorting differences in ranges or relationships among data. Normalisation methods can be classified into two main groups: data normalisation methods and activation normalisation methods.

Activation normalisation techniques normalise activations of hidden units within the model by scaling and shifting the values based on the mean and standard deviation. The most well-known activation normalisation is Batch normalisation.

### Batch Normalization

*Batch normalisation* is a technique used to normalise the activations of hidden units in a deep learning model. It is often used as a regularisation method to improve model stability and performance [9]. In batch normalisation, hidden unit activations are normalised for each mini-batch of training data. This is done by scaling and shifting the activations based on the mean and standard deviation of the activations in each mini-batch. The scaling and shifting parameters are learned during training and are usually implemented as additional parameters in the model. Using batch normalisation may have several benefits, including [1]:

- Improved model stability: One of the main benefits of batch normalisation is that it can help to stabilise the training of deep neural networks. Deep neural networks are often susceptible to the initialisation of the weights and can be difficult to train. Batch normalisation can help reduce this sensitivity and stabilise the training process.
- Improved model performance: Batch normalisation can help improve model generalisation and reduce overfitting. It can also allow the model to use a higher learning rate, which can speed up the training process.
- improve the training of Generative Adversarial Networks (GANs). GANs are difficult to train, suffering from problems such as mode collapse and disturbing visual artefacts [28]. Batch normalisation can help to stabilise the training of GANs and improve their performance.

However, like any technique, it has its limitations and potential drawbacks:

- Increased computational cost: Batch normalisation requires the model to calculate the mean and standard deviation of the activations in each mini-batch, which can add additional computational cost to the model.
- Sometimes, the use of batch normalisation in GANs can also have some unintended consequences. For example, batch normalisation can sometimes negatively impact the quality of the trained model .

Batch normalisation is typically applied to the inputs of a layer, either before or after the activation function. The general use case is batch normalisation between a network's linear and non-linear layers because it normalises the input to the activation function [9]. The equation for Batch Normalization can be represented as follows:

$$\text{Batch Normalization}(\mathbf{x}) = \gamma \cdot \frac{\mathbf{x} - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{3.1}$$

where $x$ represents the input tensor or activations of a layer, $\mu$ is the mean of the batch, calculated as the average of $x$ over the batch dimension, $\sigma^2$ is the variance of the batch, calculated as the average of $(x - \mu)^2$ over the batch dimension, $\epsilon$ is a small constant added for numerical stability to avoid division by zero, $\gamma$ and $\beta$ are learnable parameters called the scale and shift parameters respectively.

# Chapter 4

# Neural Network Training Visualization

Visualisations can help understand how well the neural network is learning and identify any issues or problems that may occur during training. Many techniques for visualizing network behavior include heat maps, saliency maps, feature importance maps, and low-dimensional projections. In machine learning, accuracy and loss are two standard metrics used to evaluate model performance. Accuracy measures the model's ability to make correct predictions on the test set, while loss quantifies the discrepancy between the model's predictions and the target labels.

Various visualizations can be employed to comprehend the training process of a neural network. The loss function plot displays the loss function value over time with the number of training iterations or epochs on the x-axis, where a decreasing loss indicates efficient learning. The accuracy plot shows the network's accuracy on the training or validation set over time, where increasing accuracy signifies effective learning. Plotting training and validation loss, or training and validation accuracy, can show how well the model fits the training data and generalizes to new data.

A confusion matrix is another helpful visualization to understand which classes the model struggles with and where it is making mistakes. Lastly, the activation histogram is a graph that illustrates the distribution of activations for a specific layer in the network. These activations refer to the output of a particular neuron in the network and can help understand the network's behavior at different points during the training process.

### Visual Representation of Accuracy and Loss During Training

A plot can be created to visualise the accuracy and loss of a model during the training process, showing how the values of these metrics change over time as the model is trained. Plotting the accuracy and loss through the training process shows how the model improves over time, illustrated in Figure 4.1.

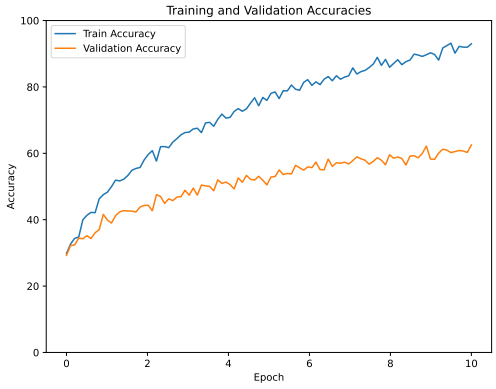



Figure 4.1: Training accuracies


Figure 4.2: Training losses

## 4.1 Convolutional Neural Network Training Visualization

Most deep learning methods need more interpretability and explainability. Despite their potential good performance on test and validation data, the poor understanding of the internal processes leads to a lack of confidence in these methods. For the 2D image classification, there are methods to highlight areas in the image that are of special interest for the decision-making process of a convolutional neural network. These methods can generally be categorised into gradient-based, perturbation-based, and other underlying mechanics [2].

We use Filter visualisation, feature maps, and saliency maps to visualise and analyse the internals of a convolutional neural network (CNN). Each has slightly different purposes and focuses on different aspects of the model.

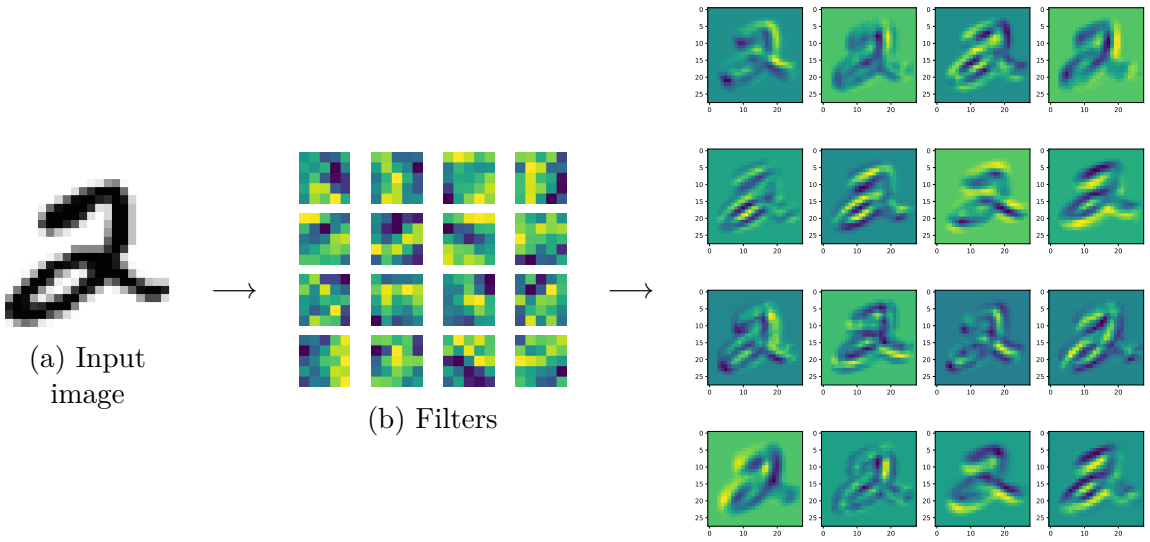
### Filter Visualisation and Feature Map

*Filter visualisation* is a technique used to visualise the learned filters in the convolutional layers of a CNN. It can be used to understand how the model extracts and learns different features from the input data.

Feature map, also known as activation map, represents each filter's activations at each position of the input image and can be used to visualise how the model is analysing and interpreting the features in the image. Feature maps can be visualised by plotting the activations of the neurons in the convolutional layers as images or heatmaps.

### Saliency Map

Saliency maps are visualisations of the importance or relevance of different parts of an input image to the model's prediction. They are typically generated by computing the gradient of the output of the model concerning the input to highlight the parts of the image that have the greatest impact on the prediction by back-propagating a gradient from the end of the network and projecting it onto an image plane [18]. Saliency maps can be used to understand which parts of the image the model uses to predict and identify potential issues with the model's interpretation of the input data.



(a) Input image (b) Filters (c) Feature maps

Figure 4.3: Visualisation of feature maps by propagating the input through filters.

(a) Original image (b) Transformed image as input (c) Saliency map

Figure 4.4: Visualisation of saliency map compared to the input image.

## 4.2 Linear Path Interpolation

Linear Path Interpolation [7] generates a slice of the loss function along the shortest path possible represented by values on a line. By employing linear interpolation of the parameters, it is possible to assess the contribution of each layer to the overall performance of the network.

A neural network has to be already trained to perform Linear Path Interpolation. A line is defined in the parameter space of the network, representing the path along which interpolation will be performed. Two points on the line, typically the starting and ending points, are chosen. Linear interpolation is performed between the network parameters at these two points by linearly combining them to create new parameter values. The loss function is evaluated using the interpolated parameters, providing the loss values along the line. Finally, the loss values obtained from the interpolation are plotted, representing a slice of the loss function along the shortest path possible.

## 4.3 Loss Surface Around Training Path

Loss surface [13] analysis around the training path of neural networks might help us:

- gain insights into the optimisation process and understand why specific algorithms may perform better or worse on a particular neural network architecture or task.
- Understand the geometry of the space of the network's parameters and how the optimisation algorithm navigates that space during training.
- Identify and diagnose problems with the optimisation process. For example, if the loss surface has many local minima or saddle points, the optimisation algorithm may get stuck and have difficulty finding the global minimum. We can identify these problem areas by analysing the loss surface and modifying the optimisation algorithm to navigate the space better.
- with the design and selection of appropriate hyperparameters. For instance, if the loss surface has a high degree of curvature or contains many sharp minima, the optimiser might require a smaller or more aggressive learning rate scheduling to converge appropriately. Conversely, suppose the loss surface is relatively smooth and has few minima. In that case, the optimiser can use a higher learning rate or less aggressive learning rate scheduling to converge more quickly.
- Understand the dynamics of the optimisation process and how different optimisation algorithms or techniques affect the optimisation path. For example, by analysing the loss surface and the gradients along the optimisation path, we can gain insights into how momentum, weight decay, or other regularisation techniques affect the optimisation process and how different optimisation algorithms might be able to take advantage of the geometry of the loss surface.
- Understand the trade-offs between different network architectures and hyperparameters. For instance, we can identify regions of the space that are associated with lower loss but higher complexity or vice versa. This information can guide the design of more efficient and effective neural network architectures. For example, if the loss function around the minimum is rough, we can add residual layers.
- design more efficient optimisation algorithms. For example, if the loss surface has specific desirable properties, such as being convex or having a low condition[1] number, we can design optimisation algorithms that take advantage of these properties and converge faster or more reliably. Inversely, we can gain insights into the sources of non-convexity and ill-conditioning[2] in the optimisation problem and develop optimisation algorithms that can more effectively navigate these challenges. This can lead to the development of more efficient and robust optimisation algorithms for training deep neural networks.

Although loss surface analysis along the training path of neural networks can provide valuable insights into the optimization process and the geometry of parameter space, its computational complexity makes it impractical for practical usage.

[1] Small changes in the input or parameters lead to small changes in the output.
[2] Small changes in the input or parameters can lead to large changes in the output.

## 4.4 Size of the Steps

The step size or step length refers to the size of the change made to the neural network parameters during each iteration of the optimisation algorithm. The step size is usually determined by the learning rate, a hyperparameter that must be set before the training begins. The learning rate determines the size of the step taken in the gradient direction during each iteration of the optimisation algorithm.

The gradient refers to the vector of partial derivatives of the loss function concerning the model parameters. The gradient indicates the direction of the steepest ascent of the loss function and is used to update the model parameters during each iteration of the optimisation algorithm.

If the learning rate is too high, the model parameters may overshoot the optimal solution and diverge, whereas if the learning rate is too low, the convergence may be slow.

There are different strategies for choosing the learning rate during training, such as fixed learning rate, adaptive learning rate, and annealing learning rate. In the fixed learning rate strategy, the learning rate is set to a fixed value throughout the training process. In the adaptive learning rate strategy, the learning rate is adjusted dynamically based on the behaviour of the optimisation algorithm. In the annealing learning rate strategy, the learning rate is decreased gradually over time to ensure convergence to the optimal solution [15].

The size of the steps analysis allows us to monitor the size of the steps taken by the optimization algorithm during training. This information can be used to adjust the learning rate to ensure that the algorithm is converging to the optimal solution at an appropriate rate. Additionally, the size of the steps analysis can provide insight into the behaviour of the optimization algorithm, such as whether it oscillates or converges smoothly.

Furthermore, the size of the steps analysis can also help identify a problem with the training process. For example, if the size of the steps becomes very small and the loss function does not improve significantly, it could indicate that the optimization algorithm is stuck in a local minimum or that the model is not powerful enough to fit the data. Similarly, if the size of the steps becomes huge, it could indicate that the model is overshooting the minimum and may not converge to the optimal solution. To perform the size of the steps analysis, I plotted the average magnitudes of the steps during a training session.

# Chapter 5

# Implementation

This chapter presents a program's design to visualise the neural network training process. Subsequently, the technologies used and the outputs of selected program parts will be discussed. The program's main purposes are training and visualisation.

The training part of the program allows the user to choose from a set of Neural Networks, datasets, optimisation algorithms, and regularisation techniques. Based on the parameters used, the program will create a directory named after the parameters where the training steps will be saved.

In the visualisation part, the user can choose from three visualisations: linear interpolation of the network parameters, step size during training, and surface around the training path. The user can choose to execute any of these visualisations.

The program is divided into multiple modules where every substantial part has its file containing its functionality. Individual networks are represented as classes with the same interface but different properties.

The program saves the progress of its components' outputs so that when the user stops executing the program, at the next execution, already computed parts will not be computed again.

The implementation language chosen is Python because it provides high readability and has many useful libraries for machine learning and data visualisation.

Neural networks have been implemented and trained using an open-source machine learning library called `PyTorch`[1], which provides optimised work with tensors, NVIDIA[2] GPU utilisation using the `torch.cuda` module, and many more helpful modules.

Array computations and output saving and loading are primarily done using the *Numpy*[3] library. Plotting the results is done using *Matplotlib*[4] library.

The program layout and implementation are inspired by the code provided in a public repository[5] as supplementary material to the bachelor thesis authored by Silvie Němcová [19]. The program can be executed on both Windows and Linux systems. The only system-specific constraint is file system path, where the user has to write the workspace location of the program in the `__main__.py` file or open the project using IDE in the parent directory of `„src"`.

The program source code is available on Github[6].

[1] https://pytorch.org
[2] https://developer.nvidia.com/cuda-zone
[3] https://numpy.org
[4] https://matplotlib.org
[5] https://github.com/suzrz/vis_net_loss_landscape
[6] https://github.com/Samuel-2000/BP-NN_Visualisation

# Chapter 6

# Examining the Training Progress of Neural Networks

This thesis focuses mainly on inventing a new method of visualising a two-dimensional loss surface around training path examination. It should find better results than analysis using PCA directions from chosen point and its comparison with the one-dimensional linear interpolation variant results.

Another goal is to provide insights into the consequences of step size change during training various models.

These goals are achieved by examining the training progress of various networks and datasets.

## 6.1 Architectures Used in Visualization Experiments

Experiments were performed on five different models to identify and confirm new phenomena that may arise during visualisation experiments. The models are described in upcoming subsections. The networks' first and last layers might differ based on the dataset.

Lenet was used as a basepoint to model a compact variant called Modified-LeNet, which was used in all visualisations.

Step-size visualisations were also performed on simpler models called TinyNN and Tiny-CNN and a more robust model called VGG.

### LeNet-5 and Modified-LeNet

LeNet-5 was initially designed for handwritten digit recognition, and it has been widely used as a benchmark for other image classification tasks, such as object recognition and face detection. The architecture is based on LeCun's LeNet-5 architecture [12]. It has also served as a starting point for developing more advanced CNN architectures, such as AlexNet [11] and VGGNet [24]. Figure 6.1 illustrates the architecture of LeNet-5, featuring convolutional layers with channel dimensions @ resolution, fully connected layers with a specified number of neurons, no padding during convolution, and filters of size 3x3.

Modified-LeNet has three convolutional layers and only one fully-connected layer. Modified-LeNet architecture is illustrated in Figure 6.2.

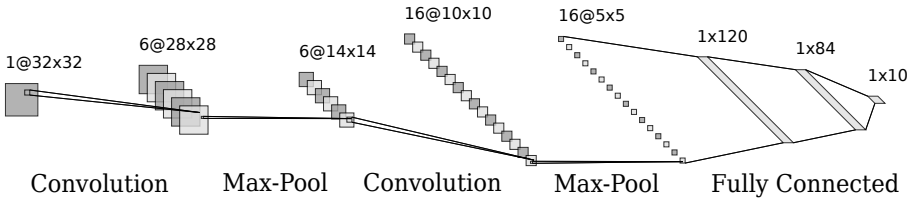



Figure 6.1: **LeNet-5 Architecture.** Kernel size is 5x5, without padding.

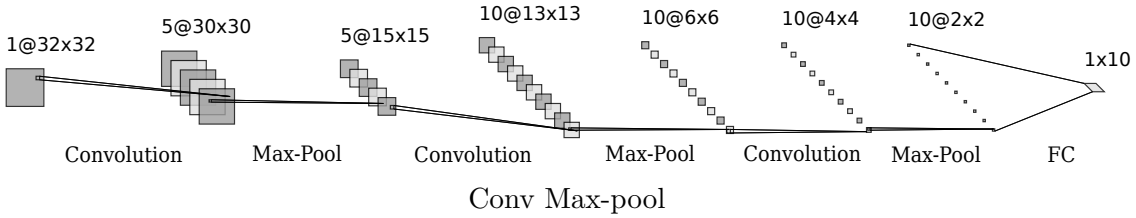



Figure 6.2: **Modified-LeNet Architecture.** Kernel size is 3x3, without padding.

### Tiny Networks and VGG

Two smaller networks, TinyNN and TinyCNN, and a larger VGG network were modeled to verify the experiments.

TinyCNN adopts a similar structure to Modified-LeNet but utilizes only three channels in its hidden layers. On the other hand, TinyNN is a shallow network that directly maps the input to the output. The architecture of TinyNN is depicted in figure 6.3.

A more intricate network is constructed based on the Visual Geometry Group (VGG) architecture, which incorporates multiple sequential convolutions followed by pooling operations. The VGG architecture, as illustrated in Figure 6.4, demonstrates this configuration.

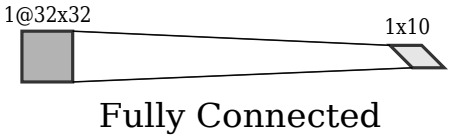



Figure 6.3: **TinyNN Architecture.**

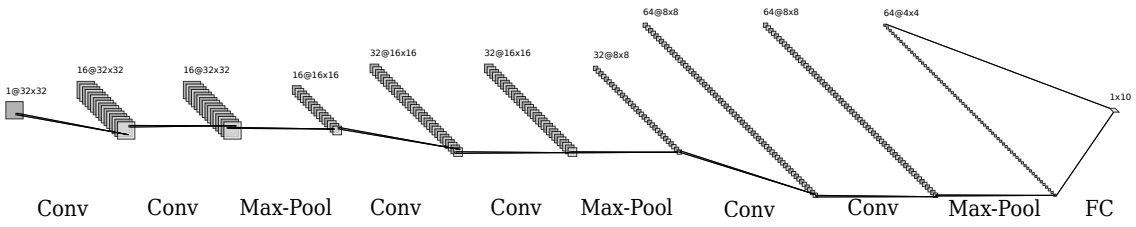



Figure 6.4: **VGG Architecture.** Kernel size is 3x3, with the “same” padding.

## 6.2 Datasets

Datasets are essential to machine learning because they provide the necessary information for training and evaluating machine learning models. Datasets used in experiments are MNIST and CIFAR-10.

### MNIST

The MNIST (Modified National Institute of Standards and Technology) dataset[1] is an extensive database of handwritten digits that is commonly used for training various image processing systems. It has a training set of 60,000 examples and a test set of 10,000 examples. The images in the MNIST dataset are grayscale and have a resolution of 28x28 pixels, which were resized to 32x32 to train the models mentioned above.

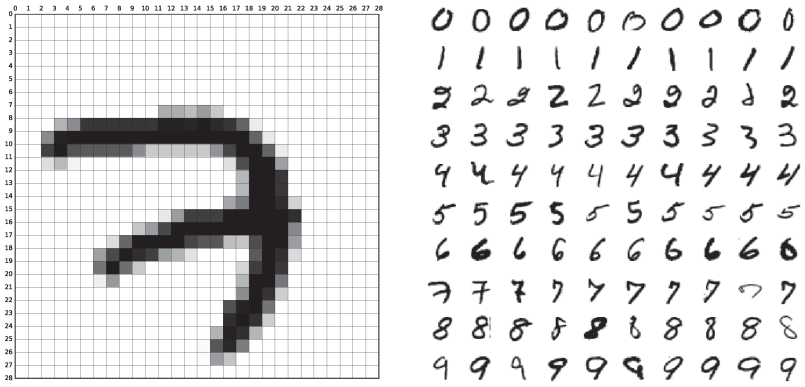


Figure 6.5: MNIST dataset sample

### Cifar-10 and Cifar-100

The CIFAR-10[2] contains 60,000 32x32 colour images in 10 different classes. The dataset comprises five training subsets and one test subset, each with 10,000 images.

The CIFAR-100 dataset is a subset of the Tiny Images dataset and consists of 60,000 32x32 colour images in 100 classes. The 100 classes in the CIFAR-100 are grouped into 20 superclasses. There are 600 images per class. Each image comes with a "fine" label (the class to which it belongs) and a "coarse" label (the superclass to which it belongs).


Figure 6.6: Cifar-10 dataset sample

[1] http://yann.lecun.com/exdb/mnist
[2] https://www.cs.toronto.edu/~kriz/cifar.html

## 6.3 Preliminary Experiments

Training process experiments to reach optimal training accuracy while minimising training time have been done in the bachelor thesis by Silvie Němcová [19]. Therefore, I was experimenting with visualisation optimisation.

There were problems with the immense size of networks since saving every network parameter value for every step, with different networks and parameters during training, can take up much space.

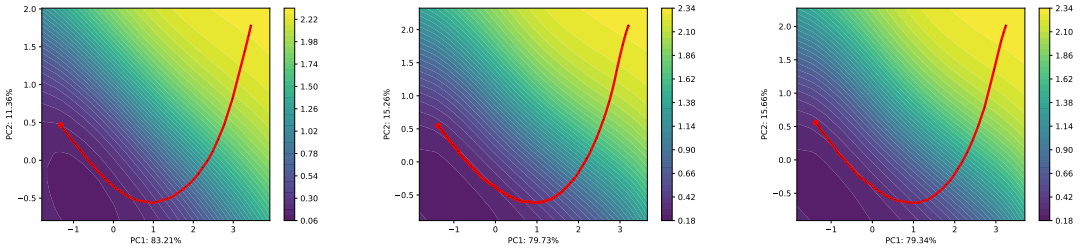
The two proposed ideas to reduce the size were to minimise the network size or sample the training path.

### Minimising the Network Size

I have done experiments to find a more efficient version of LeNet-5. The final version of the network is aforementioned *Modified-LeNet*, which uses about 30x fewer parameters, yet accuracy change concerning visualisation experiments is hardly recognizable 6.7. LeNet-5 has an accuracy of 98.6 %, while Modified-LeNet has an accuracy of 97.4 % on the MNIST dataset using SGD during 13 training epochs with the mini-batches of size 64.

### Sampling the Training Path

Another experiment visually compared the sampled and whole paths on the loss surface visualisation. Using only 123 samples of the path yielded almost identical visual results as rendering the path using all 12,195 samples 6.7. Even though this optimisation provided 100x data savings, it had to be abandoned, as the step-size experiments would provide incorrect results since if we measure the distance from the starting point to the endpoint of a curve, the resulting value would be smaller compared to a straight line of the same size.




(a) LeNet-5 (b) Modified-LeNet (c) Sampled Modified-LeNet

Figure 6.7: Comparison of LeNet-5 vs. Modified Lenet vs. Sampled Modified Lenet

### Loss Surface Visualisation Speed Optimisation

Loss Surface computation is a very computationally demanding and time-consuming task since the network loss must be computed for every point on the two-dimensional grid. To achieve the most accurate results, the loss has to be computed over the whole training dataset and with a big resolution, which is computationally demanding. Therefore an optimal resolution and dataset chunk size had to be found to make the experiments feasible.

After experimenting with the resolution, dataset chunk size parameters, and visual comparisons of the experiments, I have found the optimal settings where increasing resolution

or data samples would yield almost the same visual results. The resolution has been set to a grid of 19x19, and the loss is computed on 10000 training data samples instead of the whole dataset of 60000 samples.

The dataset optimization results in a linear six-fold increase in speed, while the resolution optimization leads to a quadratic improvement in speed.

The purpose of the visualization is to provide a general overview of the loss landscape rather than precise numerical accuracy. Using a representative subset of the data, the main characteristics of the loss surface can still be captured without compromising its validity. Lower resolution can still capture the essential features, and trends of the loss surface can still be observed and analyzed effectively, allowing for meaningful insights and comparisons.

## 6.4 Loss Surface Around Training Path

The experiments described in the subsequent text aim to explore different methods of visualizing and analyzing the loss surface around the training path and use it to find phenomena in the training process.

### Making of Grid and Path

2D path and interpolation directions were made using the sklearn package.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
path_2d = pca.fit_transform(path)  # path on 2D graph
x_dir, y_dir = pca.components_  # direction of multidimensional vectors
```

2D Grid coordinates were created around a centre point of the training path.
Moving through x and y coordinates is equivalent to moving through `x_dir` and `y_dir`.
for each grid point, we need to find the corresponding model weights state. Methods of finding and interpolating weights are discussed in the subsections below. Loss is computed using model weights for each point.

### PCA

This method approximates parameters (state of the network in each grid point) by calculating the distance from the centre point to the interpolated point (2D) and multiplying the distance by multidimensional vectors in both directions.

$$\theta_{\mathbf{pca}} = \theta_{\mathbf{c}} + \mathbf{d} \cdot \mathbf{M}, \tag{6.1}$$

where $\theta_c$ represents the parameters at the centre point, the vector $d$ represents the two-dimensional distance from the centre point to the point on the grid, $M$ is the matrix with `x_dir` and `y_dir` as its columns and $\cdot$ denotes the dot product operation.

Based on the "PCA" image 6.8, it is evident that this method lacks sufficient accuracy.

### Sampled

Considering that PCA directions from the centre point typically do not intersect with the training path points, I have suggested a method that involves selecting the closest point

from the path to the grid point. This selected point will then be interpolated using PCA directions.

The Sampled method involves selecting the closest point from the training path to the point on the grid and then interpolating with PCA directions. Let the closest point on the path be represented by the vector $q$, and let the vector from this point to the point on the grid be represented by $d$. The parameters for this point on the grid can then be approximated as follows:

$$\theta_{\mathbf{sampled}} = \theta_{\mathbf{q}} + \mathbf{d} \cdot \mathbf{M}, \tag{6.2}$$

where $\theta_q$ represents the parameters at the closest point on the path, $M$ is the matrix with `x_dir` and `y_dir` as its columns, and $\cdot$ denotes the dot product operation.

Based on the "Sampled" image 6.8, it is evident that this method encounters difficulties when selecting points from the training path in regions where the loss surface is steep and the steps are large. Consequently, this leads to a jagged area filled with local extremes.

I suggest two methods to enhance the visualization. The first method involves interpolating the closest path point with the centre point. The second method entails averaging all the path points instead of selecting just one.

### Sampled and PCA Parameters L2 Distance

In order to calculate and visualize the influence of each component on the final image during interpolation between Sampled and PCA values, a way of calculating and visualising how much each component affects the final image needs to be established.

Firstly, it is necessary to determine the grid points for which interpolation is required by determining if it falls within a convex area of the path. The center point should not be considered if the grid point is located behind the path.

To interpolate the centre point $C$ with a path point $P$, weights based on the distance from the grid point need to be computed, where a smaller distance corresponds to a larger weight.

The conditional statement in the code can be represented mathematically as follows:

$$w_C = \begin{cases} 0 & \text{if } f < 0 \\ \frac{d_P}{d_C + d_P} & \text{if } f \geq 0 \end{cases} \qquad w_P = \begin{cases} 1 & \text{if } f < 0 \\ \frac{d_C}{d_C + d_P} & \text{if } f \geq 0 \end{cases}, \tag{6.3}$$

where $f$ represents whether the interpolated grid point is inside the convex part of the path, $d_P$ is the Euclidean distance between interpolated grid point $G$ and a path point $P$, $d_C$ is the Euclidean distance between interpolated grid point $G$ and a centre point $C$.

The computed weights of distances are utilized in the equation for interpolating the parameters of the grid point.

$$\theta_{\mathbf{sampled\_pca}} = w_P \cdot \theta_{\mathbf{sampled}} + w_C \cdot \theta_{\mathbf{pca}} \tag{6.4}$$

Before computing the loss, it is beneficial to visualize the L2 distances between the parameters for each point in space. This allows for an understanding of the interpolation process. The image 6.8 titled "Sampled and Sampled with PCA parameters L2 distance" effectively showcases the successful interpolation.

### Sampled with PCA

Now that an understanding of how each method affects the image have been gained, the loss can be computed. From the "Sampled with PCA" image 6.8, it is noticeable that the middle of the contour exhibits a larger loss. This observation explains the path around it and addresses the issue of rapidly changing path points. However, it is important to note that the loss between the starting point and endpoint does not correspond to the output of the one-dimensional interpolation (where the loss should strictly decrease). Based on these observations, it can be concluded that this method is not explanatory since it does not accurately explain the loss function around the centre.

### Averaged

To address the problem of rapidly changing path points on nearby grid points, an alternative method is proposed, which involves averaging all the path points. Let $\theta_{avg}$ denote the interpolated grid point, and $P$ represents the set of path points. The L2 distances from $\theta_{avg}$ to each path point are calculated and stored in $D$.

The softmax function is applied to the inverted distances to assign weights to each path point. This ensures that closer points have higher weights. The weights are computed as follows:

$$\mathbf{w_i} = \frac{e^{-d_i}}{\sum_{j=1}^{n} e^{-d_j}}, \tag{6.5}$$

where $d_i$ represents the L2 distance between the $i$-th path point and the current grid point, and $n$ is the total number of path points in $P$.

Finally, the weighted average is computed using the weights:

$$\theta_{\mathbf{avg}} = \sum_{i=1}^{n} w_i \theta_{\mathbf{i}}, \tag{6.6}$$

where $\theta_i$ represents the parameters of the $i$-th path point.

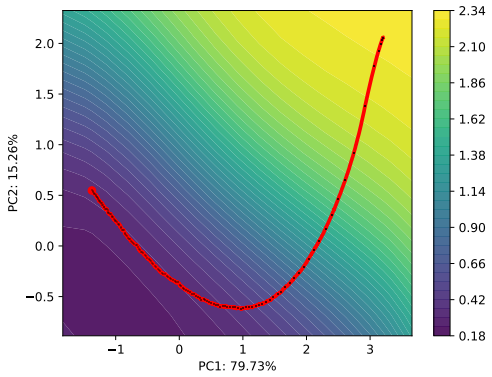
The image 6.8 titled "Averaged" demonstrates significantly improved results compared to "Sampled with PCA." It accurately corresponds to both the actual path and the one-dimensional interpolation path. As a result, the averaging method has been chosen as the most suitable candidate for subsequent experiments.

### Averaged with Multidimensional Distances

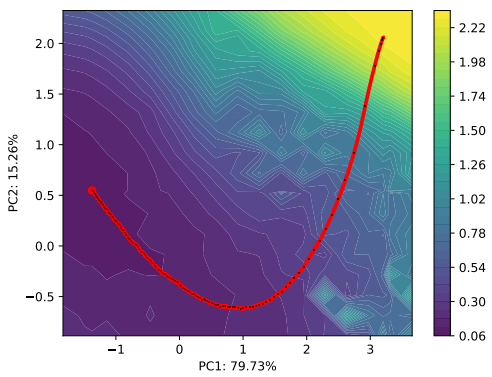
When calculating point distances from two-dimensional coordinates, there is a potential for evaluation errors that could result in slightly incorrect distances compared to the correct distances in higher dimensions. These errors may not be readily apparent when using stochastic gradient descent (SGD), as PCA explains it with 95 % accuracy. However, it might improve visualisation using ADAM, which PCA explains with only 65 % accuracy.

The visualizations 6.9 labeled as "Multidimensional" indicate that the ADAM algorithm's difference in two-dimensional and multidimensional distances is visually more different than that of the SGD algorithm, which correlates to the lower PCA explainability.

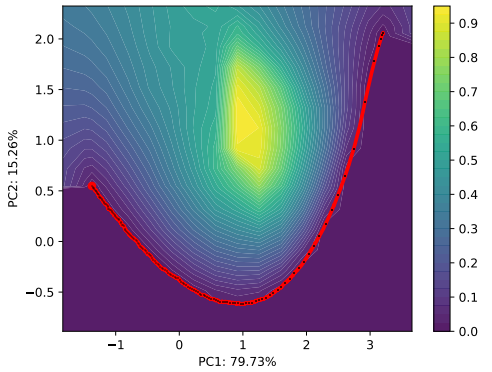
Considering that ADAM incorporates momentum, which should theoretically result in a smoother and more straightforward path, it is perplexing that the experiment demonstrates the opposite effect.



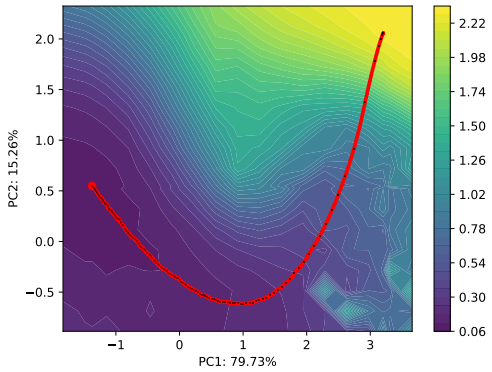

(a) PCA



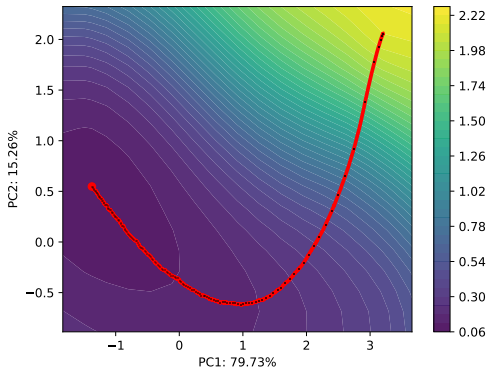

(b) Sampled



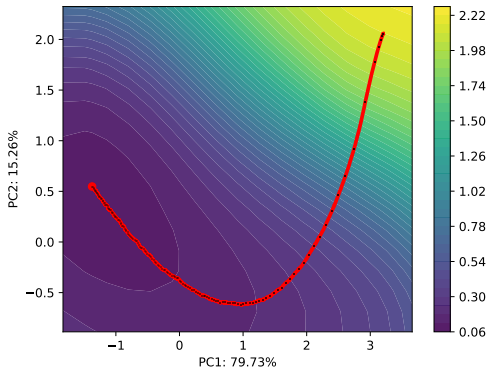

(c) L2 parameters interpolation distance




(d) Sampled with PCA




(e) Averaged



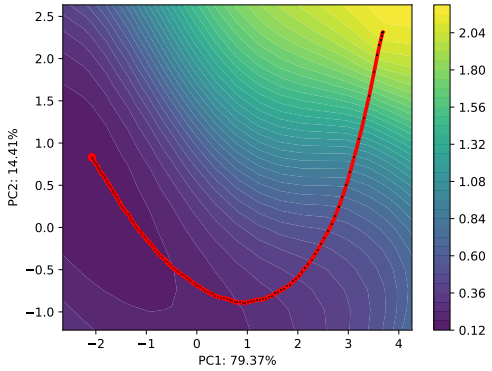

(f) Multidimensional Average

Figure 6.8: Comparison of methods to compute the loss surface around the training path.

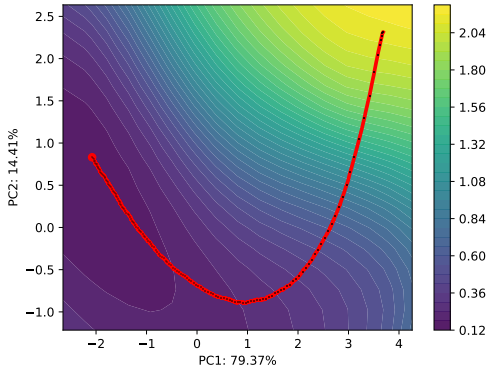
### Loss Comparison using Linear Interpolation Visualisation

It is challenging to determine whether the new contours accurately represent the reality of the loss surface far away from the path. However, one way to assess this is by drawing a straight line connecting the beginning and end points and comparing the resulting loss with that obtained from the Linear Path Interpolation experiment. By examining the curvature

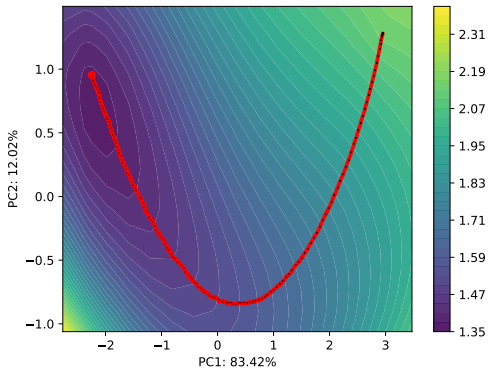
of the loss in both cases, we can gain insights into the similarity or dissimilarity between them. This comparison can provide valuable information regarding the accuracy of the new contours in capturing the characteristics of the loss surface.



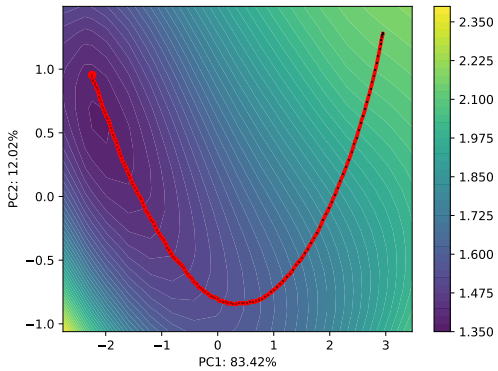

(a) SGD Dropout MNIST



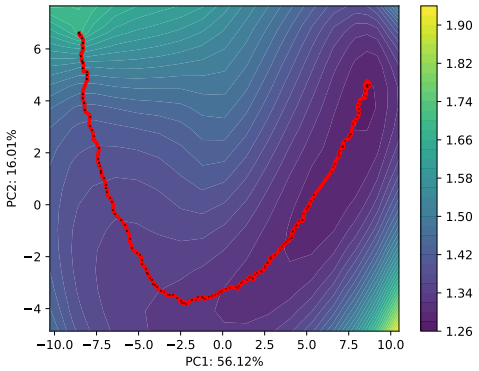

(b) Multidimensional SGD Dropout MNIST



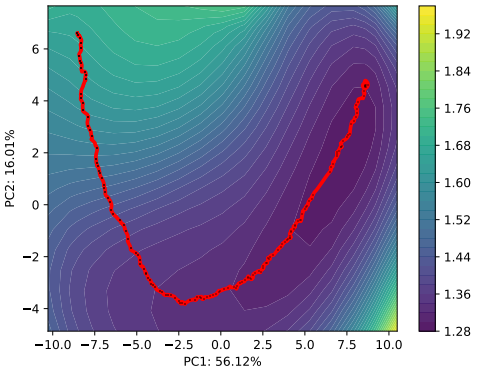

(c) SGD Cifar-10




(d) Multidimensional SGD Cifar-10




(e) Adam Cifar-10




(f) Multidimensional Adam Cifar-10

Figure 6.9: Comparison of averaged ann multi-dimensional averaged over many experiments to verify ADAM's behaviour.

## 6.5 Size of the Steps

This experiment aims to investigate the effects of optimization algorithms on the training dynamics of neural networks. Specifically, I will analyze the step sizes during training to gain insights into the behavior and performance of various optimization methods.

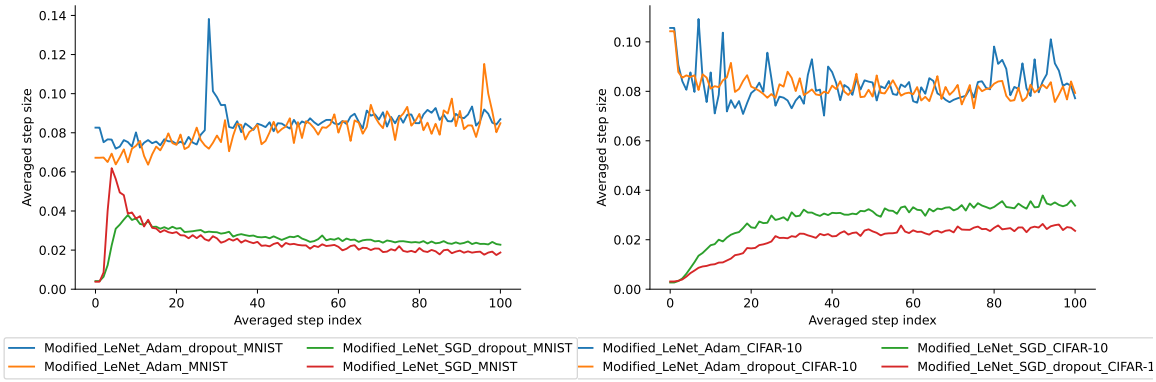
I aim to compare the step sizes during training using Adam, SGD, and dropout regularization. I will analyze the patterns and dynamics of step sizes across different optimization algorithms. Additionally, I will examine how optimization algorithms affect training effectiveness on different datasets, namely MNIST and CIFAR.

To ensure compatibility with the neural network architecture, I will apply standard preprocessing steps to the data, including normalization and reshaping. I will individually train the neural network on the MNIST and CIFAR datasets for each optimization algorithm while monitoring and recording the step sizes during training.

To analyze and compare the step sizes obtained from different optimization algorithms. Furthermore, I will assess the training effectiveness by measuring loss curves for each optimization algorithm and dataset combination.

I anticipate observing distinct patterns in step sizes during training with different optimization algorithms. Results are visualised in figure 6.10.

### Derivation and Confirmation of Phenomena From the Graphs




(a) MNIST (b) CIFAR

Figure 6.10: Step size differences of Modified-LeNet using different training techniques on two different datasets.

When comparing the Modified-LeNet architecture, the visualized step sizes during neural network training reveal interesting patterns for different optimization methods.

The Adam optimizer exhibits larger step sizes compared to SGD. In the case of effective training on the MNIST dataset, the step sizes show a continuous climb. However, during ineffective training on CIFAR, the step sizes tend to remain almost constant.

On the other hand, SGD demonstrates a hill-like pattern at the beginning of training. As training progresses effectively on MNIST, the step sizes continuously decline. Conversely, during ineffective training on CIFAR, the step sizes continuously climb.

When incorporating dropout regularization, SGD initially results in a smaller hill-shaped pattern. Furthermore, when ADAM is utilized, it reduces the speed of the climb

during training. Similar to SGD, during ineffective training on CIFAR, the step sizes still exhibit a continuous climb.

These observations highlight the differences in step size dynamics among the optimization methods and provide insights into their behavior during neural network training.

### Verification of the Results

This experiment aims to verify and extend the results obtained from the previous analysis by comparing the step size dynamics of different optimization algorithms on neural networks with varying sizes. Specifically, I will investigate the behavior and performance of optimization methods on three network architectures: TinyNN, TinyCNN, and VGG. Results are visualised in figure 6.11. Results show that the more complex the network is, the more prominent the bump using SGD.

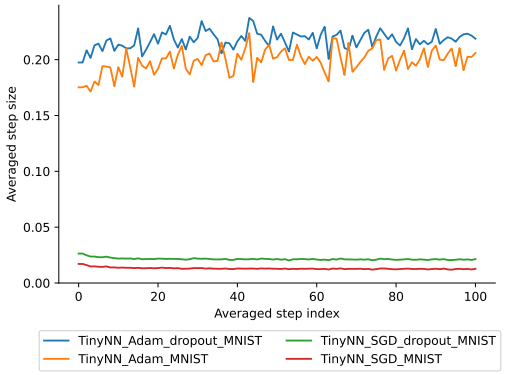
### Loss and Gradients Comparison Experiment

From the results above, if the SGD gradients decrease, the network will train poorly (have low accuracy) and vice versa. Therefore I will try a comparison with loss. Results are shown in figure 6.12, confirming the hypothesis.

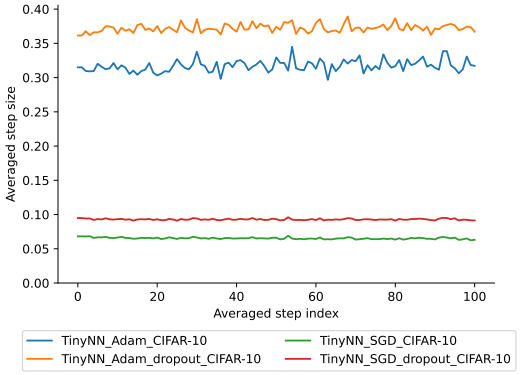
### Learning Rate Experiment

Given the presence of a bump in the SGD training process, I will attempt to set the learning rate in accordance with the observed graph to assess whether this adjustment leads to improvements in the training performance.

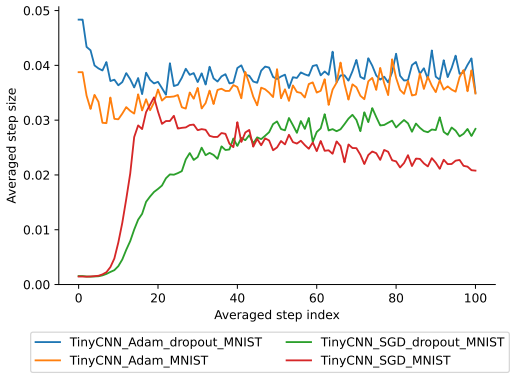
Figure 6.13 demonstrates that leveraging the awareness of the bump's presence resulted in more efficient training of the network.



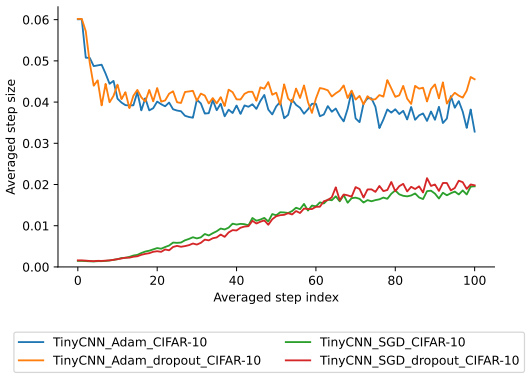

(a) TinyNN MNIST



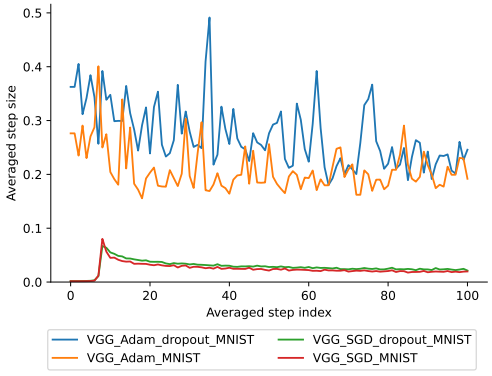

(b) TinyNN CIFAR



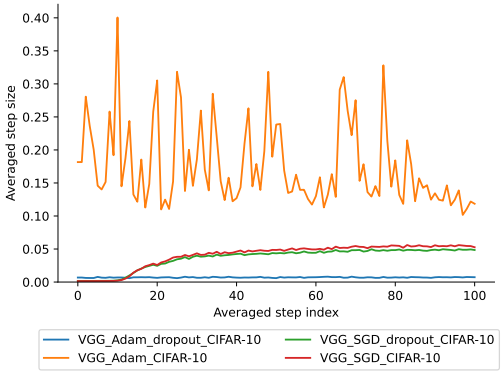

(c) TinyCNN MNIST




(d) TinyCNN CIFAR

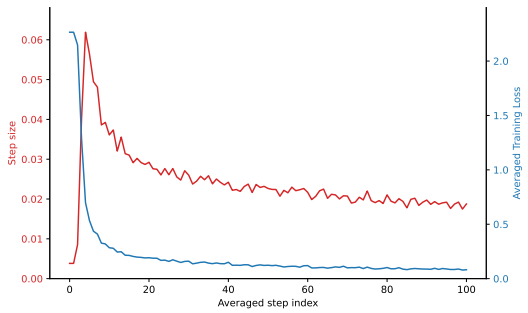



(e) VGG MNIST

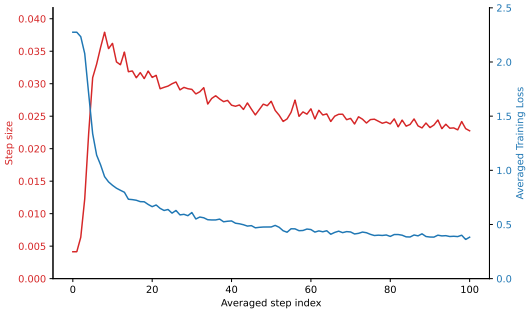



(f) VGG CIFAR

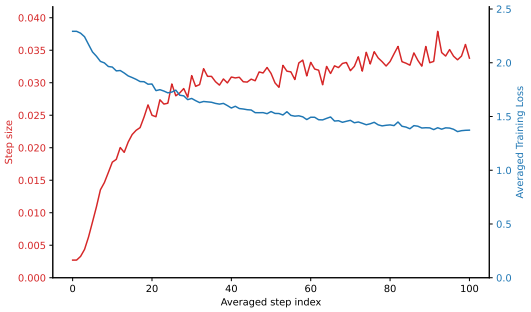
Figure 6.11: Verification of the step sizes by utilizing other networks.



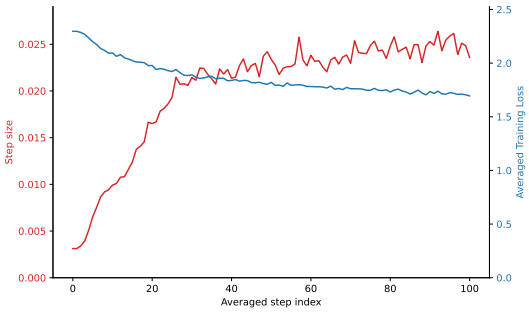

(a) MNIST




(b) Dropout MNIST

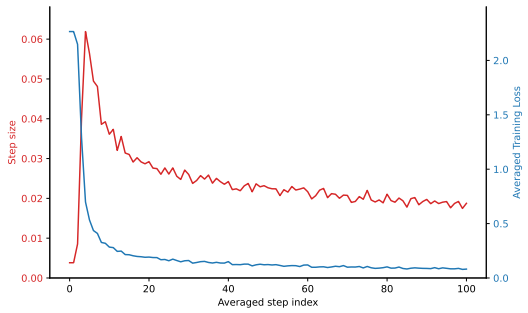



(c) TinyCNN CIFAR

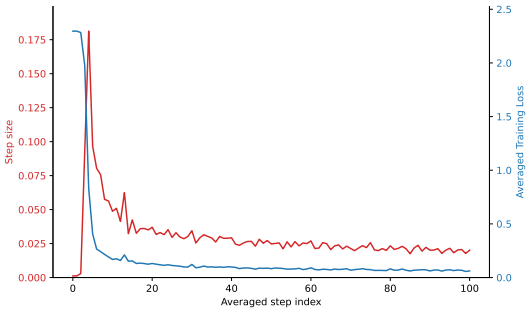



(d) Dropout CIFAR

Figure 6.12: Comparison of Modified_LeNet step size development using SGD, and its consequences on minimum loss prediction.




(a) Fixed learning rate.




(b) Learning rate equal to original step sizes.

Figure 6.13: Modified-LeNet using SGD on MNIST without dropout. Training efficiency comparison.

# Chapter 7

# Conclusion

This thesis aimed to visualise and examine the training progress of neural networks. The neural network training progress is computationally demanding, with many unexplained phenomena. The implemented program provides various methods to visualise the training progress.

The loss function surface visualisation using PCA directions was successfully replicated. The PCA directions projection allows visualisation of the path taken by the algorithm during the training. A new method of surface visualisation around the training path using multiple path points was implemented to evaluate the phenomena of the ADAM algorithm, previously hypothesised by its PCA variance explainability.

The exploration of alternative distance metrics, such as Manhattan distance or Mahalanobis distance, could be valuable in improving the accuracy of the interpolation, considering that the current approach relies on Euclidean distance as the primary measure for determining the nearest path point.

To enhance the visualization of the loss surface, it would be beneficial to investigate adaptive resolution techniques that dynamically adjust the grid resolution based on the complexity or curvature of the loss surface. This adaptive approach could provide more precise representations in areas of interest while reducing computational requirements in less critical regions.

One aspect that the thesis needs to address is uncertainty estimation. Incorporating techniques such as bootstrapping, Bayesian inference, or dropout-based uncertainty estimation would enable quantifying confidence or variability in the visualization results. This additional information would facilitate more informed interpretations of the loss surface by researchers and practitioners.

Expanding the experiments to include other model architectures and datasets beyond Modified LeNet, MNIST, and CIFAR-10 would contribute to assessing the generalizability and robustness of the visualization techniques. This broader evaluation would help to comprehensively understand how the loss surface behaves across different neural network architectures and datasets.

In addition to visual comparisons, incorporating quantitative metrics to assess the interpolated surfaces' accuracy, fidelity, or smoothness would result in a more comprehensive evaluation. Measures like the structural similarity index could be employed to compare the quality of the interpolated surfaces.

While contour plots serve as the primary visualization method in the thesis, exploring alternative visualization techniques, such as interactive visualizations, would offer diverse

perspectives and enable more profound insights into the characteristics of the loss landscapes.

To enhance the credibility of the results, empirical experiments could be conducted to validate the findings presented in the thesis, complementing the visual comparisons and interpretations of the loss surface with empirical evidence.

The step size was visualised to find new ways of training optimisation. Observations were made and used in further experiments. Utilizing the bump at the start helped train the network more effectively, while comparing loss to gradients provided a way to tell whether the network could train to give good results.

The implemented program is published under an MIT license on GitHub.

# Bibliography


[1] Balestriero, R. and Baraniuk, R. G. *Batch Normalization Explained*. 2022.

[2] Bayer, J., Münch, D. and Arens, M. *A Comparison of Deep Saliency Map Generators on Multispectral Data in Object Detection*. 2021. Available at: https://arxiv.org/abs/2108.11767.

[3] Chrislb. *Artificial Neuron Model*. 2005 [cit. 2023-03-17]. Available at: https://commons.wikimedia.org/wiki/File:ArtificialNeuronModel_english.png.

[4] Farley, B. and Clark, W. Simulation of self-organizing systems by digital computer. *Transactions of the IRE Professional Group on Information Theory*. 1st ed. 1954, vol. 4, no. 4, p. 76–84. DOI: 10.1109/TIT.1954.1057468.

[5] fdeloche. *Recurrent Neural Network*. 2017 [cit. 2023-04-27]. Available at: https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg.

[6] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. 1st ed. MIT Press, 2016. ISBN 978-0-262-33743. Available at: http://www.deeplearningbook.org.

[7] Goodfellow, I. J., Vinyals, O. and Saxe, A. M. *Qualitatively characterizing neural network optimization problems*. 2015.

[8] Hebb, D. The organization of behavior: a neuropsychological theory. *Psychology Press*. 1stth ed. J. Wiley; Chapman & Hall. 1949. ISBN 978-0805843002.

[9] Ioffe, S. and Szegedy, C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR*. 2015, abs/1502.03167. Available at: http://arxiv.org/abs/1502.03167.

[10] Kingma, D. P. and Ba, J. *Adam: A Method for Stochastic Optimization*. 2017.

[11] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C., Bottou, L. and Weinberger, K., ed. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2012, vol. 25. Available at: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[12] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, vol. 86, no. 11, p. 2278–2324. Available at: https://ieeexplore.ieee.org/document/726791.

[13] Li, H., Xu, Z., Taylor, G., Studer, C. and Goldstein, T. *Visualizing the Loss Landscape of Neural Nets*. 2018. Available at: https://proceedings.neurips.cc/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf.

[14] Li, X. and Wu, X. Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition. *CoRR*. 2014, abs/1410.4281. Available at: http://arxiv.org/abs/1410.4281.

[15] Li, Y., Wei, C. and Ma, T. Towards Explaining the Regularization Effect of Initial Large Learning Rate in Training Neural Networks. *CoRR*. 2019, abs/1907.04595. Available at: http://arxiv.org/abs/1907.04595.

[16] McCulloch, W. S. and Pitts, W. A Logical Calculus of the Ideas Immanent in Nervous Activity. *The Bulletin of Mathematical Biophysics*. 1943, vol. 5, no. 4, p. 115–133. ISBN 9780262267137.

[17] Medler, D. A. A brief history of connectionism. *Neural Computing Surveys*. 1998, vol. 1.

[18] Mundhenk, T. N., Chen, B. Y. and Friedland, G. Efficient Saliency Maps for Explainable AI. *CoRR*. 2019, abs/1911.11293. Available at: http://arxiv.org/abs/1911.11293.

[19] Němcová, S. *Neural Network Training Progress Visualization*. Brno, CZ, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Available at: https://www.fit.vut.cz/study/thesis/23446/.

[20] Olah, C., Mordvintsev, A. and Schubert, L. Feature Visualization. *Distill*. 2017. DOI: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

[21] O'Shea, K. and Nash, R. An Introduction to Convolutional Neural Networks. *CoRR*. 2015, abs/1511.08458. Available at: http://arxiv.org/abs/1511.08458.

[22] Rosenblatt, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*. 1958, vol. 65, no. 6, p. 386–408. DOI: 10.1037/h0042519.

[23] Seuret, M., Alberti, M., Ingold, R. and Liwicki, M. PCA-Initialized Deep Neural Networks Applied To Document Image Analysis. *CoRR*. 2017, abs/1702.00177. Available at: http://arxiv.org/abs/1702.00177.

[24] Simonyan, K. and Zisserman, A. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015.

[25] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, vol. 15, no. 56, p. 1929–1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html.

[26] Textbook, C. U. C. O. O. *Adam* [online]. 2021 [cit. 2022-12-16]. Available at: https://optimization.cbe.cornell.edu/index.php?title=Adam.

[27] Werbos, P. J. Applications of advances in nonlinear sensitivity analysis. In: Drenick, R. F. and Kozin, F., ed. *System Modeling and Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1982, p. 762–770. ISBN 978-3-540-39459-4.

[28] Xiang, S. and Li, H. *On the Effects of Batch and Weight Normalization in Generative Adversarial Networks*. 2017.

[29] Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T. J. and Lipson, H. Understanding Neural Networks Through Deep Visualization. *CoRR*. 2015, abs/1506.06579. Available at: http://arxiv.org/abs/1506.06579.

[30] Zeiler, M. D. and Fergus, R. Visualizing and Understanding Convolutional Networks. *CoRR*. 2013, abs/1311.2901. Available at: http://arxiv.org/abs/1311.2901.

[31] Zintgraf, L. M., Cohen, T. S., Adel, T. and Welling, M. Visualizing Deep Neural Network Decisions: Prediction Difference Analysis. *CoRR*. 2017, abs/1702.04595. Available at: http://arxiv.org/abs/1702.04595.