Summary of the Deep Learning Lecture at the University of Bonn

Tim Nogga

July 4, 2024

Contents

1 Introduction

1.1 Task T

Machine Learning Tasks describes how a machine learning system learns from data. Such data can be images(pixels) among many. Typical Tasks include Klassification, Regression, and Density Function Estimation.

1.2 Experience E

Specifies the Information an Algorithm can use during the Learning Process.

1.2.1 Supervised Learning

The Algorithm is given a dataset with the correct answers. The Algorithm then tries to learn the mapping from the input to the output. This can be described with an Estimation $p_data(y|x)$.

1.2.2 Unsupervised Learning

The Algorithm is given a dataset without the correct answers. The Algorithm then tries to learn the underlying structure of the data. This can be described with an Estimation $p_data(x)$.

1.3 Performance Measure P

Learning Algorithms can be evaluated based on their Performance. In Classification Problems, the Accuracy is a common measure. The Perfomance Measure is calculated on a Test Set, which is not used during the Training Process. So there a are two sets a Test Set and a Training Set.

1.4 Training Error, Test Error and Generalization

Generalization is the ability of an algorithm to perform well on previously unseen data. During Training the Training Error should be reduced, but the Test Error should be reduced as well. If the Training Error is reduced but the Test Error is not, the algorithm is overfitting. Meaning the algorithm is learning the noise in the data instead of the underlying structure. But if the Test Error is not even reduced the algorithm is underfitting.

1.5 Capacity

Overfitting and underfitting are also dependent on the capacity of the model. Say you take a model with with a low capacity, it will underfit the data, as it is not able to learn the underlying structure. But if a model with high Capacity is used, it will just memorize the data and thus overfit, as it is not able to generalize.

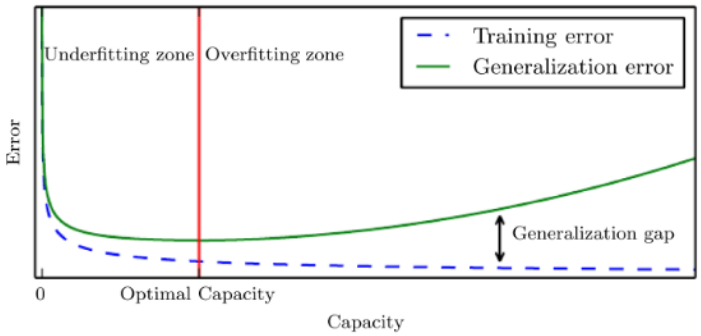

Figure 1:

1.6 Capacity vs Error

It becomes apparent that the Training Error decreases with increasing Capacity. But the Test Error decreases first and then increases again. This is due to the fact that the model is overfitting the data, thus not able to generalize anymore. This is also why it is called Generalization Gap.

1.7 Hyperparameters

Leraning Algorithms usually have Hyperparameters which are paramters not learned during Training, but set before. These could be things such as Model Capacity and Learning Rate. This poses the Question wether Capacity Hyperparameters can be derived from the Trainingset. The answer is no, since the Learning Algorithm would always choose the highest Capacity, as it would always reduce the Training Error.

1.8 Cross Validation

Cross Validation can be used to determine the best Hyperparameters. The Training Set is split into k parts. Then the Algorithm is trained on k-1 parts and tested on the remaining part. This is done k times, so that every part is used as a Test Set once. The average Test Error is then used to determine the best Hyperparameters.

2 Recap Statistics

2.1 Terms and Definitions

- **Sample Space** Ω: Set of all possible outcomes of an experiment. For example a dice roll has a Sample Space of $\Omega = \{1, 2, 3, 4, 5, 6\}$
- **Event** A: Subset of the Sample Space. For example the Event of rolling an even number is $A = \{2, 4, 6\}$
- **Sigma Algebra** Collection of Events. For example the Sigma Algebra of the dice roll is $\sigma(\Omega) = \{\emptyset, \{1, 2, 3, 4, 5, 6\}, \{1, 3, 5\}, \{2, 4, 6\}\}$

- **Measurable space** $(\Omega, \mathcal{A})$: Is a Measurable Space if Ω is a Sample Space and $\mathcal{A}$ is a Sigma Algebra.
- **Probability Density Function if Ω is Discrete** x $p : \Omega \to [0, 1]$ with $\sum_{\omega \in \Omega} p(\omega) = 1$.
- **Probability Density Function if Ω is Continuous** x $p : \Omega \to [0, \infty)$ with $\int_{\omega \in \Omega} p(\omega) d\omega = 1$.

2.2 Multivariate Statistics

It just means that there are more than one Random Variable. For example if we have two Random Variables X and Y we can define a Joint Probability Density Function $p(x, y)$.

2.3 Marginalization in a Discrete Case

We can get the PDF of one Random Variable by summing over the other. For example $p(x) = \sum_y p(x, y)$. or $p(y) = \sum_x p(x, y)$. In the Case of getting $p(x)$ this is just Summing over all possible values of y, which is rather Intuitive.

2.4 Marginalization in a Continuous Case

In the Continuous Case we have to integrate over the other Random Variable. For example $p(x) = \int p(x, y) dy$. or $p(y) = \int p(x, y) dx$.

2.5 Marginalization in a Mixed Case

In a Mixed case if $p(x)$ is Continuous and $p(y)$ is Discrete we have to sum over y and integrate over x. For example $p(x) = \int p(x, y) dy$ or $p(y) = \sum_x p(x, y)$.

2.6 Marginalization in Multiple Dimensions

If we for example want to know (p(x,y) given 4 parameters x,y,z,w) we have to marginalize over z and w. If w is Discrete we sum over w and if z is Continuous we integrate over z. This results in the following $p(x, y) = \sum_w \int p(x, y, z, w) dz dw$.

2.7 Conditional Probability

The Conditional Probability is defined as $p(x|y = y_1)$ and is the Probability of x given that y is y_1. $p(x|y = y^*) = \frac{p(x, y=y^*)}{p(y=y^*)}$ This is rather intuitiv aswell if you imagine that you are only interested in this one case where y is y^*. And then you will need to devide by the Probability of this case happening, independent of x.

This can be rewritten as $p(x|y) = \frac{p(x,y)}{p(y)}$.
Which in turn can be rewritten as $p(x, y) = p(x|y)p(y)$ and $p(x, y) = p(y|x)p(x)$.

2.8 Independence

Two Random Variables are independent if $p(x, y) = p(x)p(y)$. This means that the Probability of x and y happening is the same as the Probability of x happening times the Probability of y happening.

2.9 independent and Identically Distributed

As an example one could think of a Dice Roll. The Dice Roll is independent and identically distributed, as the Probability of rolling a 1 is the same for every roll and the Probability of rolling a 1 and a 2 is the same as the Probability of rolling a 1 times the Probability of rolling a 2. Formally this can be written $p(x_1, x_2, ..., x_n) = p(x_1)p(x_2)...p(x_n)$.

2.10 Bayes Rule

As previously discussed we have $p(x, y) = p(x|y)p(y) and p(x, y) = p(y|x)p(x)$ By setting these two equations equal we get $p(y|x)p(x) = p(x|y)p(y)$ This can be rewritten as $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$ which is the Bayes Rule. With the background knowledge that $p(x) = \sum_y p(x|y)p(y)$ for discrete and $p(x) = \int p(x|y)p(y)$ for Continuous cases we can rewrite the Bayes Rule as $p(y|x) = \frac{p(x|y)p(y)}{\sum_y p(x|y)p(y)}$ or $p(y|x) = \frac{p(x|y)p(y)}{\int p(x|y)p(y)dy}$. Here the Likelihood is the $p(x|y)$, the Prior is the $p(y)$ and the Posterior is the $p(y|x)$ and the Evidence is the $p(x)$. The Likelihood is the the Probability to observe a certain x given a certain y. The Prior is the Probability of y happening. The Posterior is the Probability of y happening given x. The Evidence is the Probability of x happening.

2.11 Expectation

The Expectation is the average value of a Random Variable. It is defined as $E[x] = \sum_x xp(x)$ for Discrete Random Variables and $E[x] = \int xp(x)dx$ for Continuous Random Variables.

2.12 Variance and Standard Deviation

The Variance is a measure of how much the values of a Random Variable differ from the mean. It is defined as $Var[x] = E[(x - E[x])^2] = E[x^2] - E[x]^2$. The Standard Deviation is the square root of the Variance. It is defined as $\sigma = \sqrt{Var[x]}$.

2.13 Shannon Entropy

The Shannon Entropy gives us an answer to the Question how many bits are needed to encode Samples from a Probability Distribution. It is defined as $H(x) = -\sum_x p(x) \log p(x) = E[-\log(p(X))]$ for Discrete Random Variables. This can be used to measure the Uncertainty of a Random Variable. If the Entropy is high, the Random Variable is uncertain. If the Entropy is low, the Random Variable is more certain. So for a Coin Flip the Entropy would be $H(x) = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1$. This means that it takes 1 Bit to encode the Coin Flip.

2.14 Cross Entropy

The Cross Entropy is the same as the Shannon Entropy, but for two Probability Distributions. It is defined as $H(p, q) = -\sum_x p(x) \log q(x) = E_p[-\log(q(X))]$

for Discrete Random Variables. This can be used to measure the difference between two Probability Distributions. If the Cross Entropy is high, the Distributions are different. If the Cross Entropy is low, the Distributions are similar.

2.15 Kullback-Leibler Divergence

The Kullback-Leibler Divergence is a measure of how much one Probability Distribution differs from another. It is defined as $D_{KL}(p||q) = H(p,q) - H(p) = \sum_x p(x) \log \frac{p(x)}{q(x)} = E_p[\log \frac{p(X)}{q(X)}]$ for Discrete Random Variables. This can be used to measure the difference between two Probability Distributions. If the Kullback-Leibler Divergence is high, the Distributions are different. If the Kullback-Leibler Divergence is low, the Distributions are similar, if the Divergence is 0 the Distributions are the same.

3 Estimators

3.1 Point Estimation

A Point Estimator is a function of the data that is used to estimate an unknown parameter $\hat{\theta}$ This can also be interpreted as an Learning Algorithm, where the Parameter is supposed to be learned. The Point Estimator is denoted as $\hat{\theta}$.

3.2 Bias

The Bias of an Estimator is the difference between the expected value of the Estimator and the true value of the parameter. It is defined as $Bias(\hat{\theta}) = E[\hat{\theta}] - \theta$. If the Bias is 0, the Estimator is unbiased.

3.3 Mean Squared Error

The Mean Squared Error is a measure of how well an Estimator performs. This was also discussed on one of the Tasks handed out. Here we had to Prove that the Mean Squared Error can be decomposed into the Variance of the Estimator and the Bias of the Estimator.

$$\begin{aligned}
\text{MSE}(\hat{\theta}_m) &= \text{Bias}(\hat{\theta}_m)^2 + \text{Var}(\hat{\theta}_m) \\
\Leftrightarrow \mathbb{E}[(\hat{\theta}_m - \theta)^2] &= (\mathbb{E}[\hat{\theta}_m] - \theta)^2 + \mathbb{E}[\hat{\theta}_m^2] - \mathbb{E}[\hat{\theta}_m]^2 \\
\Leftrightarrow \mathbb{E}[\hat{\theta}_m^2 - 2\hat{\theta}_m\theta + \theta^2] &= \mathbb{E}[\hat{\theta}_m]^2 - 2\mathbb{E}[\hat{\theta}_m]\theta + \theta^2 + \mathbb{E}[\hat{\theta}_m^2] - \mathbb{E}[\hat{\theta}_m]^2 \\
\Leftrightarrow \mathbb{E}[\hat{\theta}_m^2] - 2\mathbb{E}[\hat{\theta}_m]\theta + \theta^2 &= \mathbb{E}[\hat{\theta}_m^2] - 2\mathbb{E}[\hat{\theta}_m]\theta + \theta^2
\end{aligned}$$

The Mean Squared Error can be decomposed into the Variance of the Estimator and the Bias of the Estimator. If the Mean Squared Error is low, the Estimator is good. The Goal of an Estimator is to minimize the Mean Squared Error.

3.4 Bias-Variance Tradeoff

As Depicted in the Figure, the Bias and the Variance are inversely proportional. If the Bias is high, the Variance is low and vice versa. The Goal is to find the

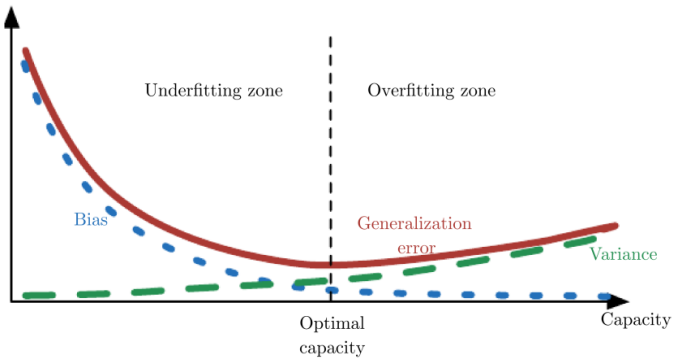

Figure 2:

sweet spot where the Bias and the Variance are both low. This is called the Bias-Variance Tradeoff.

3.5 Point Estamation and the Consinstency of the Estimator

A Point Estimator is consistent if it converges in Probability to the true value of the parameter. This means that the Estimator is getting better and better with more data. Formally this is written as $\lim_{m\to\infty} P(|\hat{\theta}_m - \theta| > \epsilon) = 0$.

3.6 Maximum Likelihood Estimation

The Maximum Likelihood Estimation is a method to estimate the parameter previously discussed. It is defined as $\hat{\theta}_{ML} = \arg\max_\theta p_{model}(X|\theta) = \arg\max_\theta \prod_{i=1}^m p_{model}(x_i|\theta)$. This means that the Maximum Likelihood Estimation maximizes $\hat{\theta}_{ML}$ given the data X. With no poof what so ever we will assume, that the Maximum Likelihood Estimation is consistent, when the true distrubution p_{data} is in the model class and p_{data} is a Value of θ

3.7 (Negative) Log Likelihood

Since it is numerically Problematic to take the Product, due to overflow/underflow we can take the Logarithm of the Likelihood. This is called the Log Likelihood. It is defined as

$$\begin{aligned}
\arg\max_\theta \log p_{model}(X|\theta) &= \arg\max_\theta \log \prod_{i=1}^m p_{model}(x_i|\theta) \\
&= \arg\max_\theta \sum_{i=1}^m \log p_{model}(x_i|\theta) \\
\arg\min_\theta (-\log p_{model}(X|\theta)) &= \arg\min_\theta \left(-\sum_{i=1}^m \log p_{model}(x_i|\theta) \right)
\end{aligned}$$

The last Equation is the Negative Log Likelihood. It is the same as the Log Likelihood, but with a minus sign in front. This is done to make it a minimization Problem, as most Optimization Algorithms are designed to minimize a Function.

3.8 Negative Log-Likelihood and Cross Entropy

Negative Log-Likelihood and Cross Entropy are equivalent when m approaches infinity. This is due to the fact that the Negative Log-Likelihood is the average Cross Entropy. This can be shown as follows: Given that the samples x_i are drawn from the true data-generating distribution $p_{data}(x)$, and the negative log-likelihood (NLL) is divided by the number of samples m, the expression for $m \to \infty$ approximates the cross-entropy between the modeled distribution $p_{model}(x|\theta)$ and the true data-generating distribution $p_{data}(x)$:

$$\begin{aligned}\frac{1}{m}\sum_{i=1}^{m} -\log p_{model}(x_i|\theta) &\approx \mathbb{E}_{x\sim p_{data}}[-\log p_{model}(x|\theta)] \\ &= -\int p_{data}(x)\log p_{model}(x|\theta)\,dx \\ &\approx H(p_{data}, p_{model})\end{aligned}$$

As $m \to \infty$, the average NLL converges to the cross-entropy between p_{data} and p_{model}.

3.9 Conditional Negative Log-Likelihood

We can generalize the Negative Log-Likelihood to the Conditional Negative Log-Likelihood. This is done by conditioning the Negative Log-Likelihood on the input. It is defined as $\arg\min_\theta \sum_{i=1}^{m} -\log p_{model}(y_i|x_i,\theta)$. This is particularly useful, because it can be used in Classification, Segmentation, among many other Tasks, where we want to Intepret the conditional Probability of the output given the input.

3.10 Cross Entropy for Classification/Segmentation

In Classification and Segmentation Problems a Neural Network usually learns Probability Function right away lets call it $p_\theta(y|x)$ This leads to the following Negative Log Likelihood: $-\log p(Y|X,\theta) = \sum_i -\log p(y_i|x_i,\theta)$. Thus follows the Cross Entropy Loss: $H = \frac{1}{m}\sum_i -\log p_\theta(y_i|x_i)$

3.11 Segmentation and Pixel Wise Classification

In Segmentation and Pixel Wise Classification the Cross Entropy Loss is calculated for every Pixel. This is done by calculating the Cross Entropy Loss for every Pixel and then averaging over all Pixels. This is done to get a single Value for the Loss. The Figure is a visualization of the Cross Entropy Loss for a Pixel Wise Classification Task. Every Layer here represents a Class. So here the average for all Classes is calculated and then compared to get a Classification Result.

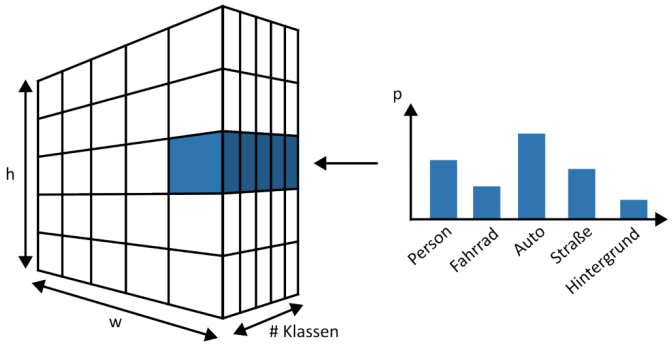

Figure 3:

3.12 Mean Squared Error for Regression

If we consider the Normal Distribution as the Model Distribution, the mean of that is learned by the Neural Networks, considering most Regression Problems. This leads to the Likelihood being defined as $p(Y|X,\theta,\sigma) = \prod_i \mathcal{N}(y_i, \mu_\theta(x_i), \sigma)$ Thus follows the Negative Log Likelihood: $-\log p(Y|X,\theta,\sigma) = \sum_i \frac{1}{2} 2(\pi\sigma^2) + \frac{1}{2}\frac{(y_i-\mu_i(x_i))^2}{\sigma^2}$. If we assume that the Variance is constant, we can drop the first term, also the Denominator can be dropped, as it is constant aswell. This leads to the Mean Squared Error: $MSE = \frac{1}{m}\sum_i (y_i - \mu_i(x_i))^2$ The minimum of the Mean Squared Error is if $\mu(x_i$ is the mean of the samples.)

3.13 Mean Absolute Error

For the MAE or $L1$ a Laplace Distribution is assumed. This leads to the Likelihood being defined as $p(Y|X,\theta,\sigma) = \prod_i \mathcal{L}(y_i, \mu_\theta(x_i), \sigma)$ With the NLL being defined as $-\log p(Y|X,\theta,\sigma) = \sum_i log(2\sigma) + \frac{|y_i-\mu_i(x_i)|}{\sigma}$. This leads to the Mean Absolute Error: $MAE = \frac{1}{m}\sum_i |y_i - \mu_i(x_i)|$ The Major difference is the missing squared making it more robust to outliers. But also the minimum is not the mean of the samples, but the median.

3.14 Maximum A Posteriori Estimation

An alternative to the Maximum Likelihood Estimation is the Maximum A Posteriori Estimation. It is defined as $\hat{Theta}_{MAP} = \arg\max_\theta p(\theta|X)$ Assuming θ is a random variable, we can use Bayes Rule to rewrite this as $\hat{\theta}_{MAP} = \arg\max_\theta p(X|\theta)p(\theta)$ To further simplify the operations, we can take the log as per usual. This leads to $\hat{\theta}_{MAP} = \arg\max_\theta \log p(X|\theta) + \log p(\theta)$ Here we can incoperate a sum again thus follows $\hat{\theta}_{MAP} = \arg\max_\theta \sum_i \log p(x_i|\theta) + \log p(\theta)$ The is as the Name already hints, the Maximum Likelihood Estimation with a Prior. This is useful if we have some prior knowledge about the parameter. With backround knowledge we can push θ to a realistic value. Say we have knowledge X we can have $P(Y|X,\theta)$ here the same with x follows $\hat{\theta}_{MAP} = \arg\max_\theta \sum_i \log p(x_i|x_i\theta) + \log p(\theta)$ This is a key Concept later on for Regularization of Weights for Neural Networks.

4 Error functions

The NLL $\hat{\theta}_{ML} = \arg\min_\theta(-\sum_{i=1}^{m} \log p(x_i|\theta)) = \arg\min_\theta f(\theta)$ the Maximum Likelihood Estimator will now be viewed as the minimum of a Function. One might notice that this does not look too different from minimizing or maximizing a function during Training in Machine-Learning. This would in that Context be reffered to as Objective Function, cost function, error function or loss function. The Variable θ that is to be Optimized would then be equivalent to the Parameters of the Model. Thus the Maximum Likelihood Point Estimation is equivalent to a Machine Learning Algorithm. The obvious question now is, how do we minimize this function?

4.1 Local and Global Minima

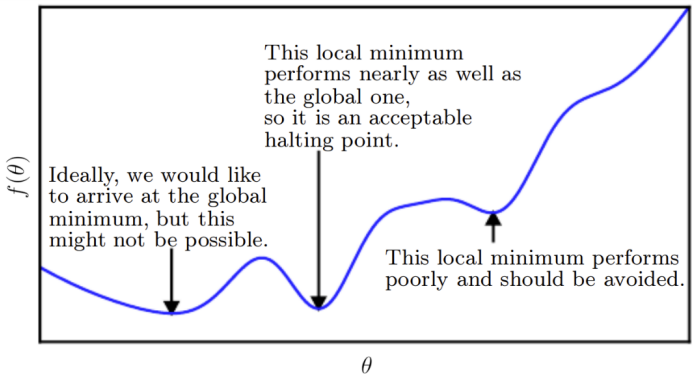

Figure 4:

As explained before the Local Minima is the one found by Gradient Descent. This might not be too bad if it is approximitely the same as the Global Minima. But if it is not, the Algorithm will get stuck in the Local Minima. A good rule of thumb is to compare the Local Minima to low values of the cost function, if they are not far off, the Local Minima is good enough.

4.2 Convex Functions

Convex function if $\forall x, y \in D \text{and} t \in [0,1] \rightarrow f(tx+(1-t)y) \leq tf(x)+(1-t)f(y)$ hold. This basically just means that every point is below the average of the two points. Convex Functions have the nice property that they have only one minimum. However this is rather rare in the context of Machine Learning.

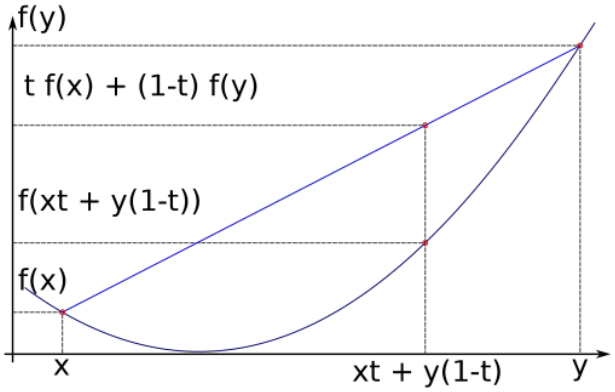

Figure 5:

4.3 Different approaches to find Minima

4.3.1 Least Squares

Least Squares is a method to find $\hat{\theta}$ for a linear Model this is done by taking $\arg\min_\theta ||A\theta - y||^2 = (A^*A)^{-1}A^*y$ This only works for least squares though and Machine Learning Models are usually not linear.

4.3.2 Grid Search

Grid Search is a method to find the minimum of a function. It is done by evaluating the function at every point on a grid. This is rather slow, as the number of points grows exponentially with the number of dimensions.

4.3.3 Evoluationary Algorithms

Evolutionary Algorithms are a class of algorithms that are inspired by the process of natural selection. They work by creating a population of solutions and then using genetic operators such as mutation and crossover to evolve the population over time. This is rather slow aswell, as it is a brute force method. It works fine for 10-1000 Parameters but becomes inneficient for more. It can be usefull for HyperParameters though.

4.3.4 Gradient Based Optimization

If the Model and the Error Function are differentiable, Gradient Based Optimization can be used. This is done by calculating the Gradient of the Error Function and then following the Gradient to the minimum.

5 Gradient Descent

5.1 Error Surface

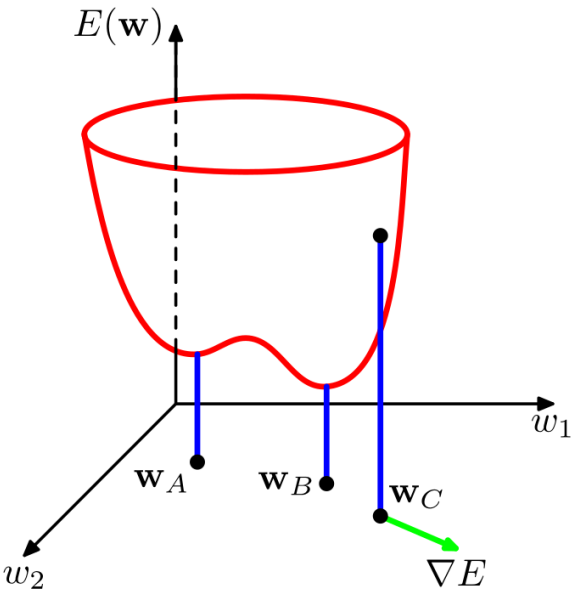

Figure 6:

This is a Geometric Representation of the Errror Function. w_A, w_B is a local and a Global minimum. Here Gradient Descent comes in handy. If we start at a random point, here w_C we can follow the Gradient to the minimum. So basically we just find the derivative of the function, follow the Gradient to the opposite direction and repeat this until we reach the minimum. The opposite direction is taken, as the Gradient points to the steepest ascent, and for minimization we want to go in the opposite direction.

5.2 Formal Definition of Gradient Descent

We are looking for θ that minimizes the Error Function $f(\theta)$ Since we can not do this analytically we will use a numerical method. For that we choose a starting value θ_0 and then update it iteratively. The update rule is defined as $\theta_{i+1} = \theta_i + \Delta\theta_i$ The usual way to do this is to introduce a learning rate ϵ and then update the parameter as follows $\theta_{i+1} = \theta_i - \epsilon\nabla f(\theta_i)$

5.3 Gradient - Taking the Derivative in Multiple Dimensions

The Error Function of a Neural Net is usually dependent on multiple Parameters. This is why it becomes necessary to talk about partial derivatives, thus the concept of Gradients is introduced. Let $f(\theta)$ be a skalar Function. Meaning it mapps from $\mathbb{R}^n \to \mathbb{R}$. The Gradient of f is defined as $\nabla f(\theta)_i = \frac{\partial f(\theta)}{\partial \theta_i} f(\theta)$ This is just the partial derivative of f with respect to θ_i. If u write this in vector form it becomes $\nabla f(\theta) = \begin{pmatrix} \frac{\partial f(\theta)}{\partial \theta_1} \\ \frac{\partial f(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\theta)}{\partial \theta_n} \end{pmatrix}$ So every entry of the Gradient is the partial derivative of f with respect to the corresponding parameter in our cases usually the weight. A stationary point is given if every entry of the Gradient is 0.

5.4 Direcional Derivative

The Directional Derivative describes the rate in which a function changes in a certain direction. It is defined as $\nabla_u f(\theta) = \lim_{\alpha \to 0} \frac{f(\theta + \alpha u) - f(\theta)}{h}$ This is just the derivative of f in the direction of u. This can be rewritten as $\nabla_u f(\theta) = \nabla f(\theta) u^T$ This is just the dot product of the Gradient and the direction. To minimize f, we must find the direction u, where f decreases the fastest, this means $min_{u,u^T u=1}\ u^T \nabla f(\theta) = min_{u,u^T u=1} ||u||_2 \cdot ||\nabla f(\theta)||_2 \cdot \cos(\phi)$. where ϕ is the angle between the Gradient and the direction. Since u is a unit vector, the minimum can be simplified to $min_u \cos(\phi)$, since the Gradient is independent of u. This is due to the fact that cos oxcilates between -1 and 1. Thus the minimun is -1 and $\cos(\pi) = -1$

5.5 Learning Rate ϵ

For now we will have just take ϵ as a small constant. The danger when ϵ is too large is that may not converge to the minimum. If ϵ is too small, the convergence will be slow. This is why it is important to choose ϵ wisely.

5.6 Jacobi Matrix

If we now have vectors as inputs and outputs it is handy to define a Matrix called Jacobi-Matrix. This matrix conatins the partial derivative of f at the position x. It is defined as

$$(J_f(\theta))_{i,j} = \frac{\partial}{\partial \theta_j} f(\theta)_i$$

This can be written in the following Matrix Form

$$J_f(\theta) = \begin{pmatrix} \frac{\partial f_1(\theta)}{\partial \theta_1} & \frac{\partial f_1(\theta)}{\partial \theta_2} & \cdots & \frac{\partial f_1(\theta)}{\partial \theta_n} \\ \frac{\partial f_2(\theta)}{\partial \theta_1} & \frac{\partial f_2(\theta)}{\partial \theta_2} & \cdots & \frac{\partial f_2(\theta)}{\partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(\theta)}{\partial \theta_1} & \frac{\partial f_m(\theta)}{\partial \theta_2} & \cdots & \frac{\partial f_m(\theta)}{\partial \theta_n} \end{pmatrix}$$

5.7 The Derivative of second order

The derivative of second order is important for us as well since it can imply the improvement that is expected of one iteration of the Gradient Descent.

5.8 Hessian Matrix

The Hessian Matrix is the Matrix of second order partial derivatives of a function. It is defined as

$$H_f(\theta)_{i,j} = \frac{\partial^2}{\partial\theta_i \partial\theta_j} f(\theta)$$

The directional Derivative of second order to a Directionvector u is given by $u^T H_f(\theta) u$. If u is an Eigenvektor of H, then the derivative of second order is equivalent to the corresponding Eigenvalue. For other directions, the derivative of second order is the weighted average of the Eigenvalues. Where Directions which correspond better with the Eigenvalues are more important. This also implies that the second directional derivative is beween the smallest and the largest Eigenvalue of H, because it is a weighted sum, it must be contrained by the smallest and the largest Eigenvalue.

This is also equivalent to the Jacobi Matrix of the Gradient.

5.9 Approximate Second-Order Methods

Let's have a look at the Newton Method. The Newton Method is used to find the minimum of a function. It is defined as $\theta_{i+1} = \theta_i - H_f(\theta_i)^{-1} \nabla f(\theta_i)$ This is just the Gradient Descent with the Hessian Matrix. I mean it makes sense though, since we basically exchanged our shitty ϵ to a more dynamic way of choosing the learning rate. It brings some disatvantages though, as the Hessian Matrix calculation as well as the inversion can be rather expensive for large disatvantages.

6 Stochastic Gradient Descent

If the calculation of the Gradient is done for the entire Dataset, it might be slow, due to the immense size. Also the data during an iteration is saved in RAM or Graphicsmemory, this can be become impossible to handle for large Datasets. This is why we introduce Stochastic Gradient Descent. Which takes Batches of the Data and calculates the Gradient for these Batches. Lets assume that the Trainingsample is i.i.d. then the log likelihood can be written as $L_\theta(X) = \frac{1}{n} \sum_{i=1}^{n} L(f_\theta(x_i), y_i)$ which implicates that the Gradient can be written as $\nabla_\theta L_\theta(X) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta L(f_\theta(x_i), y_i)$ This is just the average of the Gradients of the Batches.

6.1 Mini-Batch Gradient as an unbiased Estimator

The Approximation of the Gradient with the help of Mini-Batches can be seen as an unbiased Estimator of the Gradient. In math language we can write

$$\begin{aligned}
\mathbb{E}\left[\nabla_\theta L_\theta(\hat{X})\right] &= \mathbb{E}\left[\frac{1}{k}\sum_{x\in\hat{X}} \nabla_\theta L_\theta(x)\right] \\
&= \frac{1}{|\mathcal{X}_k|}\sum_{\hat{X}\in\mathcal{X}_k} \frac{1}{k}\sum_{x\in\hat{X}} \nabla_\theta L_\theta(x) \\
&= \frac{1}{k}\sum_{x\in X} \frac{\binom{n-1}{k-1}}{\binom{n}{k}} \nabla_\theta L_\theta(x) \\
&= \frac{1}{k}\sum_{x\in X} \frac{k}{n} \nabla_\theta L_\theta(x) \\
&= \frac{1}{n}\sum_{x\in X} \nabla_\theta L_\theta(x) \\
&= \nabla_\theta L_\theta(X)
\end{aligned}$$

The given equations demonstrate that the expected gradient of the loss function, when computed over a randomly sampled subset of the data, equals the gradient of the loss function over the entire dataset. This result shows that using mini-batches in stochastic gradient descent provides an unbiased estimate of the full gradient. As a result, SGD can efficiently and effectively optimize the model parameters by using only small subsets of data at each iteration, making it scalable and practical for large datasets. (rewritten by chatgpt so it becomes more coherent lol)

6.2 Batch Size

The Batch size is as per usual a trade off, either we choose large Batches. This would lead to a more accurate Gradient, but needs to more computation power. If we choose small Batches, the Gradient is less accurate, but the computation is faster. This makes it necessary to choose a smaller learning rate so the training remains stable. For the Varianz of the Gradient $Var(\frac{1}{|I|}\sum_{i\in I} g_i) = \frac{1}{|I|^2}\sum_{i\in I} Var(g_i) = \frac{\sigma^2}{|I|}$ I is a set of indices of the Patch and the Gradients are g_i This makes sense as it exactly depicts the relation we established before the equation. The choice of the Batch size, since it is a Tradeoff depends on several Factors

1. If the Batches are processed in parallel, the Batch size should be chosen to fit the memory of the GPU.
2. Multiprozessor Systems can't be used to their full potential if the Batch size is too small. Since the Batches have fixed cost for the computation at a certain size.
3. Hardware is optimized for certain Batch sizes. For GPUs this is usually 2^n.
4. Small Batches have a Regularization effect, as the Gradient is more noisy.

6.3 Algorithm for Stochastic Gradient Descent

Algorithm 1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k
Require: Initial parameter θ
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$
Compute gradient estimate: $\hat{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
Apply update: $\theta \leftarrow \theta - \epsilon \hat{g}$
end while

Lets go through the Algorithm step by step. First we give a learning rate and an initial parameter. Then we sample a minibatch of m examples from the training set. We compute the gradient estimate by taking the average of the gradients of the loss function over the minibatch. We then apply the update rule to the parameter by subtracting the learning rate times the gradient estimate. We repeat this process until a stopping criterion is met, which could be a maximum number of iterations, a convergence criterion, or another condition.

6.4 Learning Rate ϵ

The first Parameter was the Learning Rate ϵ. The Learning Rate needs to go to Infinity, so $\sum_{k=1}^{\infty} \epsilon_k = \infty$ and $\sum_{k=1}^{\infty} \epsilon = \infty$ Usually the learning rate is being reduced over time. This is done by a learning rate schedule. For example a linear decay, where the learning rate is reduced by a contant factor until it reaches a certain iteration, where is remains constant.

6.5 Momentum

6.6 Algorithm for Stochastic Gradient Descent with Momentum

Momentum is a method to accelerate the convergence of Gradient Descent. Momentum aggregates the Gradients of previous Iterations.

$$v \leftarrow \alpha v + \epsilon \nabla_\theta (\frac{1}{m} \sum_{i=1}^{m} L(f(\theta^{(i)}; \theta), y^{(i)}))$$

This is just the Gradient Descent with the addition of the Momentum. The Momentum is a Hyperparameter and is usually set to 0.9, 0.99, or 0.5. The Momentum is then used to update the parameter as follows $\theta \leftarrow \theta + v$ This is just the Gradient Descent with the addition of the Momentum. The Momentum is a Hyperparameter and is usually set to 0.9, 0.99, or 0.5. The Momentum is then used to update the parameter as follows $\theta \leftarrow \theta - v$

7 Adaptive Learning Rates

This basically just means that we can adapt the learning rate to the Parameters. There are several methods to do this.

Algorithm 2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α
Require: Initial parameter θ, initial velocity v
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$
Compute gradient estimate: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
Compute velocity update: $v \leftarrow \alpha v - \epsilon g$
Apply update: $\theta \leftarrow \theta + v$
end while

1. AdaGrad scales a learning rate by the inverse of the square root of the sum of all historical squared gradients. This means that the learning rate is reduced for parameters that have received large gradients in the past.

2. RMSProp is similar to AdaGrad, but uses a moving average of squared gradients instead of the sum of squared gradients. This allows the learning rate to adapt more quickly to changes in the gradients.

3. Adam combines the ideas of momentum and RMSProp. It uses a moving average of gradients and a moving average of squared gradients to adapt the learning rate for each parameter.

7.1 AdaGrad Algorithm

Algorithm 3 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter θ
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $r = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{x^{(1)}, \ldots, x^{(m)}\}$ with corresponding targets $y^{(i)}$
Compute gradient: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$
Accumulate squared gradient: $r \leftarrow r + g \odot g$
Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot g$ (Division and square root applied element-wise)
Apply update: $\theta \leftarrow \theta + \Delta\theta$
end while

So the steps added here are the Accumulation of the squared Gradient, and the update. As mentioned in the short description before, since the Multiplikation and the Division are applied element wise, the update is done for every parameter and so every parameter that was previosuly large will have a reduced learning rate. The other algorithms for other adaptive learning rates are similar to this one, but with the steps I descibed before.

8 Feed-Forward Networks

8.1 Single-Neuron

The output of single Neuron with D input values x_i, weights w_i and Bias b can be calculated as follows:

$$a = \sigma(\sum_{i=1}^{D} w_i x_i + b)$$

where σ is the activation function. The activation function is usually a non-linear function, such as the sigmoid function, the tanh function, or the ReLU function. The output of the neuron is then passed through the activation function to produce a non linear output. This is done, so that the Model can learn non-linear relationships between the input and the output, otherwise the Model would just be a linear regression.

8.2 One Layer and multiple Neurons

Often a layer is defined by Multiple Neruons, those are fed with the same input and combined in one layer. The output of such a layer can be calculated by

$$a = \sigma(\sum_{i=1}^{D} w_{ji} \cdot x_i + b_j)$$

The Compact form of this would be $a = \sigma(Wx + b)$ where W is the Matrix of the weights and b is the Bias.

8.3 Multiple Layers

Well, now that we have the defenition of one Layer down, why not do Multiple and thus create a neural Net. The output of a Neural Net with L layers can be calculated as follows:

$$\begin{aligned} a^{(1)} &= \sigma(W^{(1)}x + b^{(1)}) \\ a^{(2)} &= \sigma(W^{(2)}a^{(1)} + b^{(2)}) \\ &\vdots \\ a^{(L)} &= f^{out}(W^{(L)}a^{(L-1)} + b^{(L)}) \end{aligned}$$

Now we can consider writing this as a parameterized function

$$\mathrm{nn}_\theta(x) = f_{\text{out}}\left(W^{(N)} \cdot f^{(N-1)}\left(\dots W^{(2)} \cdot f^{(1)}\left(W^{(1)} \cdot x + b^{(1)}\right) + b^{(2)} \dots\right) + b^{(N)}\right)$$

With the parameters:

$$\theta = \left((W^{(1)}, b^{(1)}), (W^{(2)}, b^{(2)}), \dots, (W^{(N)}, b^{(N)})\right)$$

The f I use is just a placeholder for the activation function.

How many layers exist is something we will define with the amount of layers being the amount of weight matrices used for Multiplication.

8.4 Fun Facts for Linear Activation

Lets say we decide to use a linear function, lets show by its properties that it is not a good choice.

$$\text{nn}_\theta(x) = f_{\text{out}}\left(W^{(N)} \cdot f^{(N-1)}\left(\cdots f^{(2)}\left(W^{(2)} \cdot f^{(1)}\left(W^{(1)} \cdot x\right)\right)\right)\right)$$

This can be simplified by grouping the weight matrices and activation functions:

$$\text{nn}_\theta(x) = f_{\text{out}}\left(W^{(N)} \cdot f^{(N-1)}\left(\cdots f^{(2)}\left(W^{(2)} \cdot W^{(1)} \cdot f^{(1)}(x)\right)\right)\right)$$

Combining weights into a single matrix W':

$$\text{nn}_\theta(x) = f_{\text{out}}\left(W^{(N)} \cdots W^{(2)} \cdot W^{(1)} \cdot f^{(N-1)} \circ f^{(2)} \circ f^{(1)}(x)\right)$$

Finally, we get:

$$\text{nn}_\theta(x) = f_{\text{out}}\left(W' g(x)\right)$$