Inhalt

[1. Introductory 4](#_Toc40884469)

[What is Machine Learning + Deep Learning 4](#_Toc40884470)

[AI: 4](#_Toc40884471)

[ML: 4](#_Toc40884472)

[DL: 5](#_Toc40884473)

[Neural Network Basics 5](#_Toc40884474)

[Artificial Neurons und Activation Funktion 5](#_Toc40884475)

[Gewichte 6](#_Toc40884476)

[Lernen in DNN 6](#_Toc40884477)

[CNN 6](#_Toc40884478)

[RNN 7](#_Toc40884479)

[End2End Learning 7](#_Toc40884480)

[2. Gradient Descent 8](#_Toc40884481)

[Supervised Learning 8](#_Toc40884482)

[Supervised Parametric Learning 8](#_Toc40884483)

[Predict 8](#_Toc40884484)

[Compare 9](#_Toc40884485)

[Learn (Weight adjustment) 9](#_Toc40884486)

[Basic Gradient Descent Algorithm 9](#_Toc40884487)

[Error/weight Relationship 10](#_Toc40884488)

[- Das einzige was wir ändern können, um error zu nullen, ist weight. Weil input, goal\_pred und die Formel gleichbleiben müssen bzw. Änderung keinen Sinn macht 10](#_Toc40884489)

[Divergence 10](#_Toc40884490)

[Learning Rate 11](#_Toc40884491)

[3. Maths Refresher 12](#_Toc40884492)

[Linear Algebra 12](#_Toc40884493)

[Probability 13](#_Toc40884494)

[Differential Calculus 13](#_Toc40884495)

[Gradient Descent 13](#_Toc40884496)

[4. Feed Forward Neural Networks 14](#_Toc40884497)

[Gradient Descent 14](#_Toc40884498)

[Networks with multiple inputs 14](#_Toc40884499)

[Networks with multiple outputs 14](#_Toc40884500)

[Networks with multiple inputs and outputs 14](#_Toc40884501)

[Correlation 15](#_Toc40884502)

[Learning Correlation 15](#_Toc40884503)

[Creating Correlation 16](#_Toc40884504)

[ACTIVATION FUNCTION 17](#_Toc40884505)

[Backpropagation 19](#_Toc40884506)

[Update Weights 19](#_Toc40884507)

[Weights/Model 19](#_Toc40884508)

[Bias 19](#_Toc40884509)

[Loss Function 20](#_Toc40884510)

[Loss & Activationfunction relationship 21](#_Toc40884511)

[Grundregeln der Backpropagation 22](#_Toc40884512)

[Optimierungsalgorithmen 24](#_Toc40884513)

[CNN 24](#_Toc40884514)

[Convolution 24](#_Toc40884515)

[Convolutional Layers 24](#_Toc40884516)

[Max Pooling 24](#_Toc40884517)

[RNN 24](#_Toc40884518)

[Motivation 24](#_Toc40884519)

[Network Structures 24](#_Toc40884520)

[Backpropagation 24](#_Toc40884521)

[Gated RNN 24](#_Toc40884522)

[LSTM 24](#_Toc40884523)

[Gated Recurrent Unit 24](#_Toc40884524)

[Backpropagation & Regularisation 25](#_Toc40884525)

[Backpropagation 25](#_Toc40884526)

[Generalisation 25](#_Toc40884527)

[Generalisation of a supervised model 25](#_Toc40884528)

[Data Partitioning 25](#_Toc40884529)

[Generalisation Errors 25](#_Toc40884530)

[Regularisation 25](#_Toc40884531)

[Mini-Batch Learning 25](#_Toc40884532)

[Weight Penalties 25](#_Toc40884533)

[Data Augmentation 25](#_Toc40884534)

[Training with Noise 25](#_Toc40884535)

[Dropout 25](#_Toc40884536)

[Early Stopping 25](#_Toc40884537)

[Sequence To Sequence Learning 25](#_Toc40884538)

[Natural Language Processing 25](#_Toc40884539)

[Language Models 25](#_Toc40884540)

[Deep Learning 25](#_Toc40884541)

[Word Embedding 25](#_Toc40884542)

[Neural Language Models 25](#_Toc40884543)

[Sequence to Sequence Learning 25](#_Toc40884544)

[Attention Mechanisms 25](#_Toc40884545)

[Connectionist Temporal Classification 25](#_Toc40884546)

# Introductory

## What is Machine Learning + Deep Learning

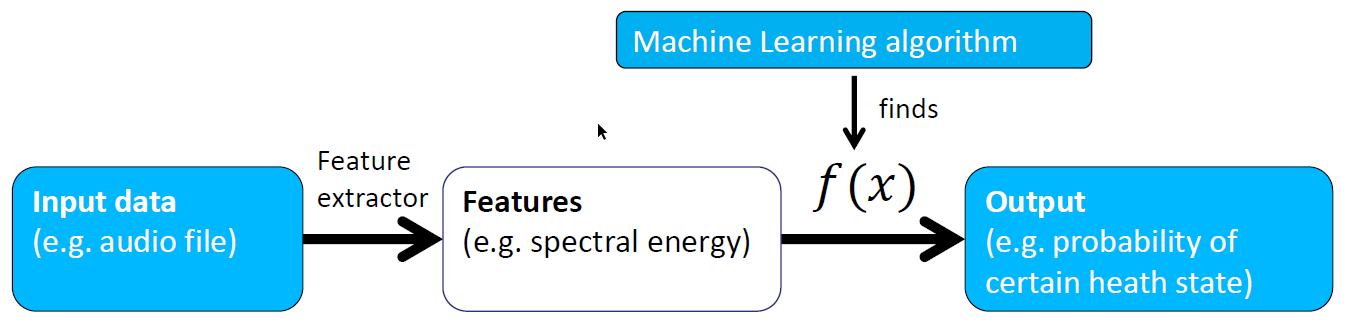
(AI(ML(DL)))

### AI:

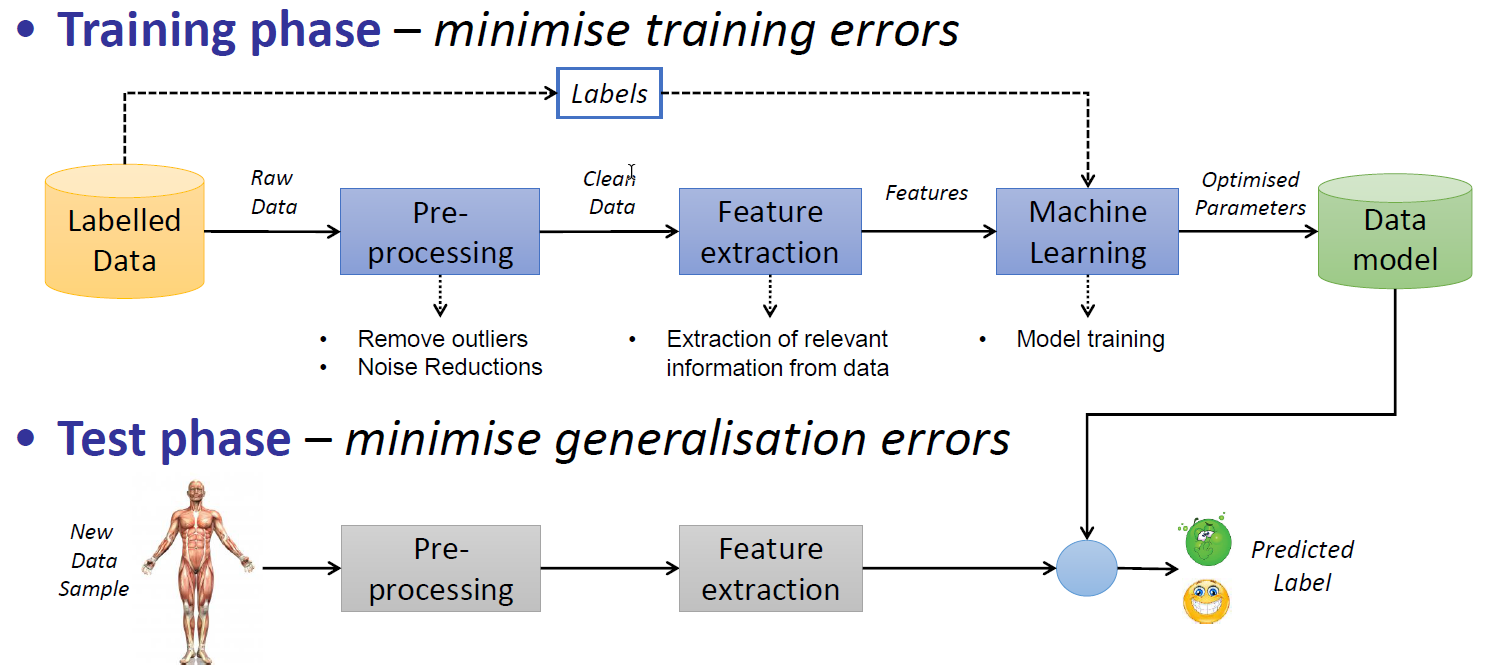
* Maschinen denken wie Menschen

### ML:

* Daten werden benutzt, um automatisch Tasks zu verbessern
* Daten und Antworten einspeisen, um Regeln zu erhalten
* Wird eher trainiert als programmiert, identifiziert strategische Struktur in Tasks, die Regelbildung erlaubt um diese auf neue Daten anzuwenden
* 3 Sachen für ML: Input Data, Expected Output, Maßstab ob der Algorithmus gut ist um current und expected Output zu vergleichen, anzupassen, zu lernen

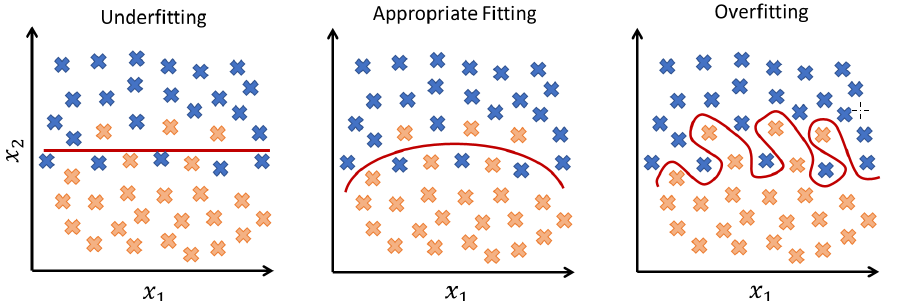


* Features ist Repräsentation der Daten, die dem ML-Algorithmus präsentiert werden, jedes Feature ist Single Information. Hunderttausende bilden Featurevektor. Alg identifiziert Patterns aus Vektorsammlung in iterativem Prozess.
* Ziel: Robuste, vorhersagende Funktion f zu erlernen, Mapping von Featurespace X zu Labelspace Y. Y\* = f(x\*) (\* ist gleich Datensatz)



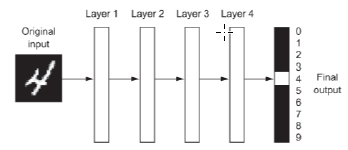
#### Generalisierungsfehler:

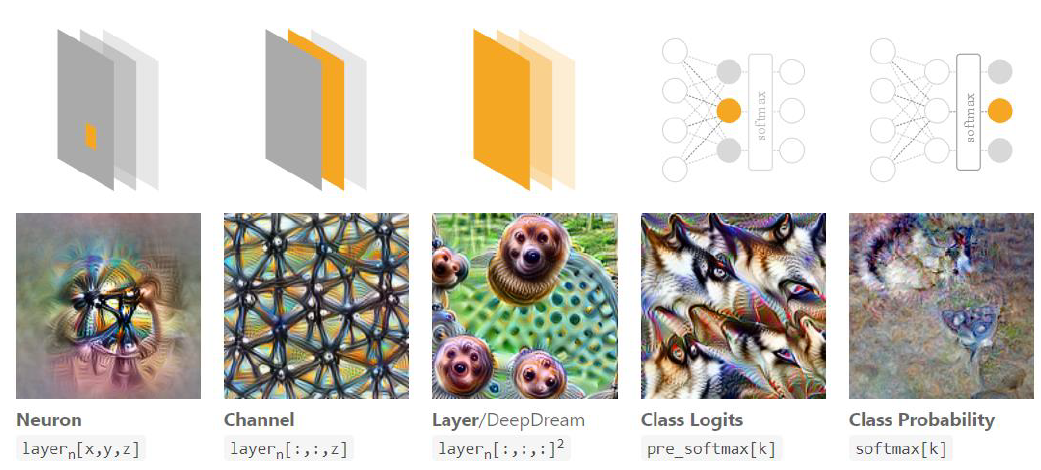
* Underfitting: Model ist zu einfach, Sensitivität fehlt
* Overfitting: Model ist zu komplex, versucht für alle Variationen Abgrenzungen



### DL:

* Daten durch multilayered Netzwerk Neural Network wie Human Brain
* Spezielle Form von ML Algorithmen
* Lernen aufeinanderfolgender Darstellungsebenen („Deep“)

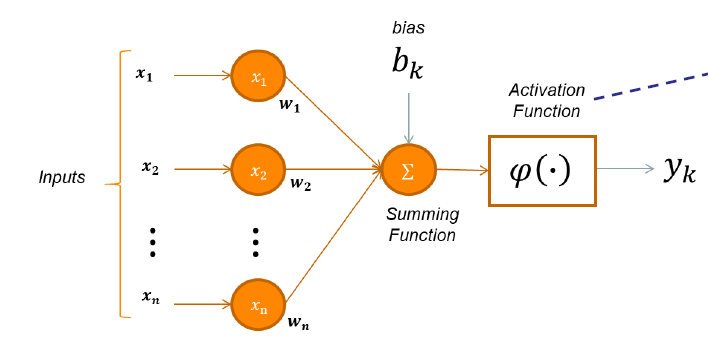




## Neural Network Basics

### Artificial Neurons und Activation Funktion

* Kombiniert verschiedene Inputs zu einzelnem Output
* Bilden Inputlayer, jedes Neuron nimmt einzelne Information auf
* Weight und Bias gehen in Activation Function, so wird bestimmt ob oder wie stark Neuron aktiviert wird



### Gewichte

* Bestimmen Einflussgrad einzelner Neuronen auf Berechnung der Aktivierung
* Werden mithilfe von Gradient Descent gelernt
* Ein einzelnes Gewicht zu ändern hat auf alle anderen Einfluss
* Repräsentiert Wissen

### Lernen in DNN

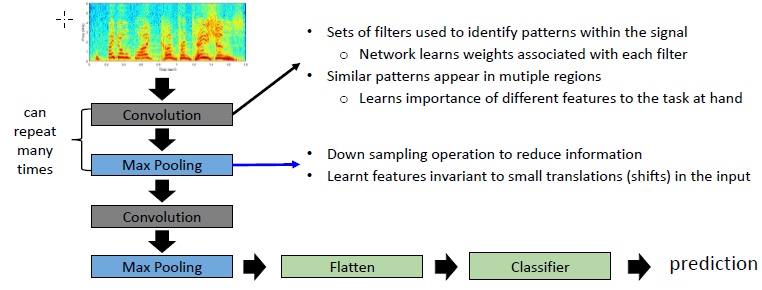
* Loss function wird benutzt, um Gewichtupdates zu kontrollieren: Wie krass unterscheidet sich der Output vom Erwarteten

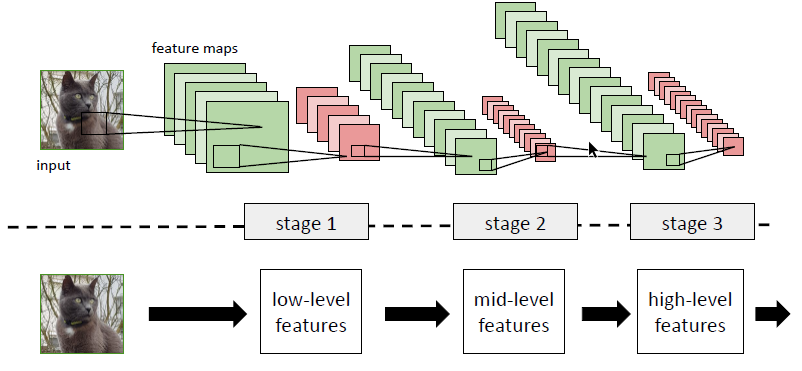


* DL gut bei riesigen Datasets (besonders bei Imageklassifizierung Natural Language und Speech)
* Universell einsetzbar: Netzwerk mit einzigem, großem verstecktem Layer ist für meiste Funktionen gut einsetzbar
* Im Gegensatz zum normalen ML brauch man keine Experten, weil Features selbst rausgefunden werden

## CNN

* Besseres Featurelearning
* Erkennt mit Filtern Strukturen in Inoput Daten, je höher Filterebene desto abstrakter





## RNN

* Im Gegensatz zu Feed Forward können hier Neuronen mit vorangegangenen oder gleichebigen Neuronen verbunden sein
* Hiddenoutput wird in Memory gestored und im nächsten Timestep wiederverwendet

## End2End Learning

# Gradient Descent

## Supervised Learning

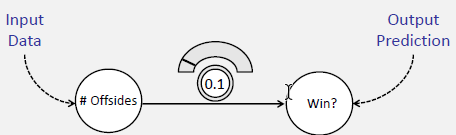
* Datensatz transferieren
* Input: Was wissen wir Output: Was wollen wir wissen
* 1. Welche Art von Daten soll Datensatz enthalten?
* 2. Datenerhebung entsprechend durchführen
* 3. Struktur von Funktion und Alg bestimmen
* 4. Lernalg auf Datensatz anwenden

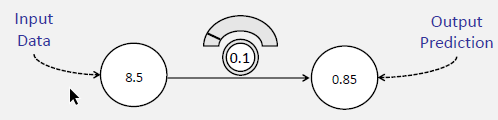
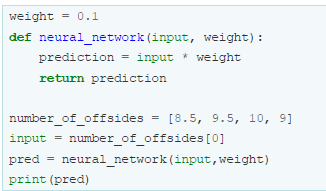
### Supervised Parametric Learning

* Modell das Daten mit Parametern fester Größe zusammenfast
* Zuerst Form für Funktion auswähen und dann Koeffizienten für die Funktion von Trainingsdaten lernen zB 𝑏𝑜+𝑏1𝑥1+𝑏2𝑥2=𝑦
* Analogie: Maschine mit fester Anzahl an Knöpfen, Knopfstellung sagt wie Daten processed werden sollen. Jeder Knopf repräsentiert die Sensitivität zu verschiedenen Arten an Input
* Prozessing transformiert input Daten zu output Vorhersage
* Learning dreht an den Knöpfen
* Keysteps sind: Predict, Compare with truth, Learn the pattern (dreh an Knöpfen)

### Predict

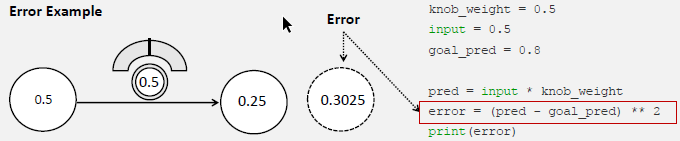
**Simple Prediction**

* Ein Datapoint input, einer Output
* Ein einzelner Knopf (Gewicht)
* ****
* 1. Netzwerk definieren 2. Input feeden 3. Value mit Weight multiplizieren 4.Output

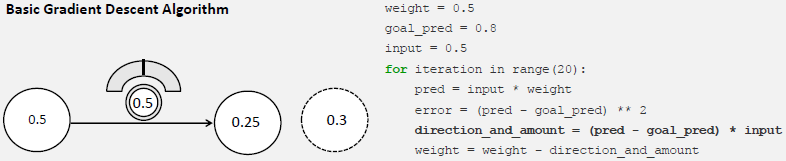
****

* Prediction zu hoch -> weights lower Prediction zu low -> weights higher

### Compare

* Wie richtig/falsch war die Prediction?
* 
* Error² falls negativ -> Große Fehler werden größer, kleine kleiner
* Mean Square Error Equation: 
* Weights so tunen, dass Error null ergibt (leichter als tunen um Target vorherzusagen)
* Wir wollen in riesigen Netzwerken den Durchschnitterror auf 0 bringen. Deshalb brauchen wir positive Errors, damit sich diese nicht mit den negativen ausnullen.

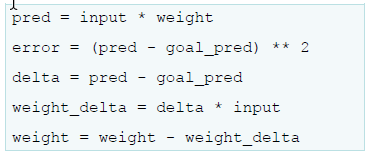
### Learn (Weight adjustment)



* Direction und Amount: Wieviel muss ich das Gewicht ändern, um Error zu reduzieren
* Pure Error ist pred – goalpred, falls positiv ist Pred zu hoch, falls negativ zu niedrig
* Scaling: Weightchange ist proportional zu Input
* Negative Reversal: Sichergehen, dass Gewicht in richtige Richtung angepasst wird selbst wenn Input negativ
* Stopping: Wenn Input zero, wird nichts angepasst

## Basic Gradient Descent Algorithm

* Goldene Methode für Neurales Lernen

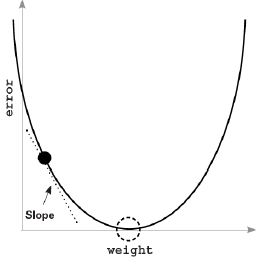


* Regelt jedes Gewicht in die korrekte Richtung mit dem korrekten Betrag, sodass Error auf null geht

### Error/weight Relationship

* Für jedes input/pred Paar kann Relationship zwischen Error und Gewicht definiert werden, wenn man prediction und error Formeln kombiniert



Wenn der schwarze Punkt der momentane Punkt ist, zeigt Slope immer auf Minimum der Funktion

### Das einzige was wir ändern können, um error zu nullen, ist weight. Weil input, goal\_pred und die Formel gleichbleiben müssen bzw. Änderung keinen Sinn macht

Zu lernen bedeutet, das weight zu beeinflussen, um error zu nullen. Ziel: Kenne die Richtung und den Wert wie sich der Error verhält, wenn du das Gewicht anpasst.

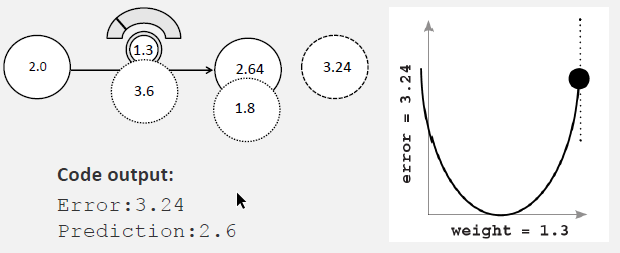
Mit Derivativen kann man in jeder Formel zwei Variablen auswählen und schauen wie sie zueinanderstehen. Das können wir nutzen, um das Weight so anzupassen, damit der Error auf 0 geht (das ist unser weighted\_delta).

* Für jeden Punkt auf dem Graphen sagt es uns, um wieviel der Error sich verändert, wenn wir weight anpassen. Ist immer in die gegenteilige Richtung von minimum gerichtet. Deshalb moved man immer in die entgegengesetzte Richtung, wenn man „weight = weight – weight-delta“ berechnet

Beim Lernen wird solange iteriert bis Error 0, immer wieder weight anpassen bis schwarzer Punkt den Graphtiefpunkt erreicht

### Divergence

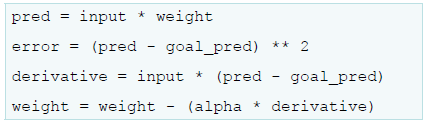


* Wenn der Input groß ist wird das Weightupdate automatisch auch groß, selbst wenn der Error klein ist. Dies kann dazu führen, dass das Update overshootet/overcorrectet und sich vom Wunschpunkt entfernt. Bei großem Error umso mehr
* 
* Deshalb führt man zusätzliche variable „alpha“ ein, um das Gewichtupdate zu skalen. Üblicherweiße mit irgendwaszwischen 0 und 1.
* 

### Learning Rate

* Wie bestimmt man Alpha? Mit Erfahrung!
* Wenn Divergenz eintritt ist Alpha zu hoch, wenn man zu langsam lernt ist es zu niedrig

Final Basic Gradient Descent for Neural Learning:



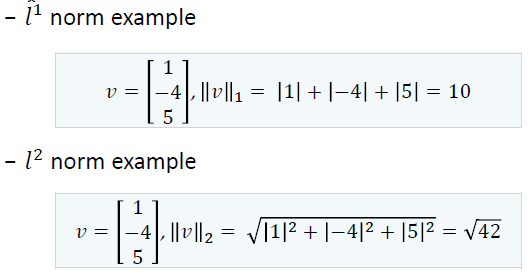
Derivative sagt Richtung und Menge für Weightadjustment vorher

Alpha hilft, Divergenzeffekte zu minimieren, wenn Input groß

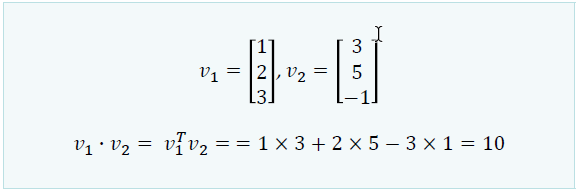
# Maths Refresher

## Linear Algebra

VEKTOR NORM

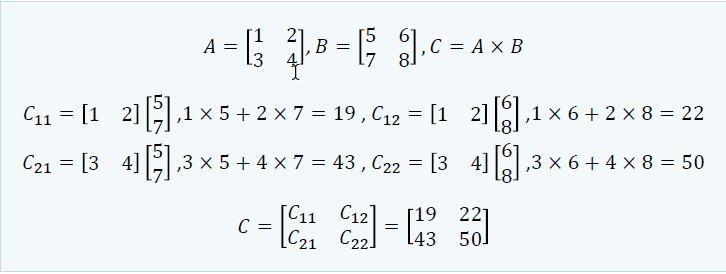


DOT PRODUCT



* Winkel zwischen 2 Vektoren, wie ähnlich sind sich 2 Vektoren?

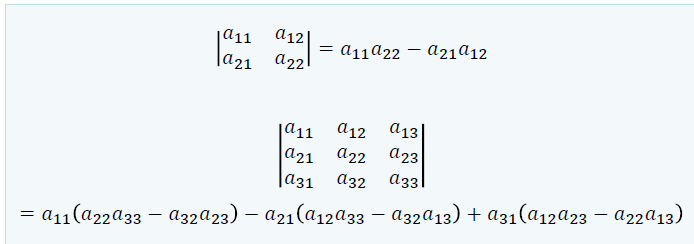
MATRIX MULTIPLIKATION



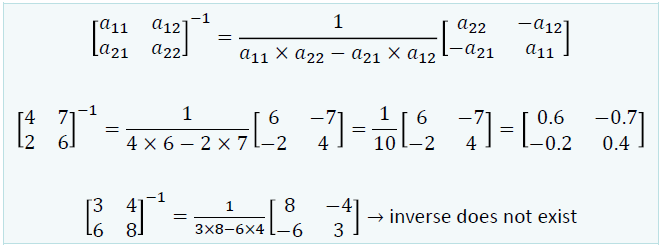
MATRIX TRANSPOSE

* Die Spalten als Reihen Transposen: Wichtig, weil [a\*b] = a^T b

DETERMINANT



MATRIX INVERSE A^-1



## Probability

* ML hat immer Unsicherheiten
* Wahrscheinlichkeitsgesetze bestimmen, wie ML Algorithmus schließen sollte

## Differential Calculus

## Gradient Descent

# Feed Forward Neural Networks

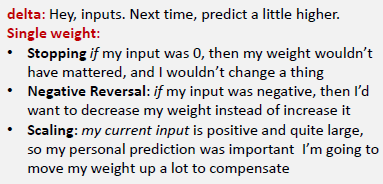
## Gradient Descent

### Networks with multiple inputs

* Jeder Input hat eigenes Gewicht
* Output ist Summe aus Inputs \* Weights

Alg:

1. Output = Input[0] \* Weight[0] + Input[1] \* Weight[1] …
2. Error = (Output – Tatsächlich)²
3. Delta = Output – Tatsächlich
4. Weight\_delta[i] = input[i] \* Delta
5. New\_Weight[i] = Old\_Weight[i] – alpha \* Weight\_delta[i]

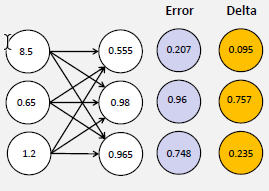


Dann wird iteriert solange bis Error nahe 0 (selber Input)

### Networks with multiple outputs

* Netzwerk verhält sich wie 3 unabhängige Komponenten mit selbem Input
* Alle Variablen individuell voneinander berechnen, außer eben Input

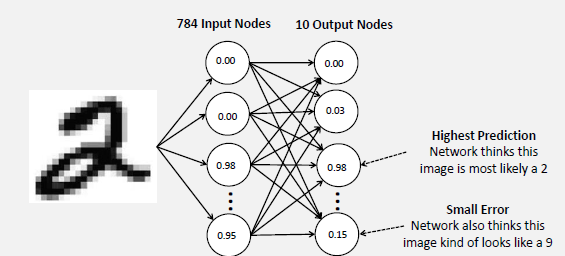
### Networks with multiple inputs and outputs

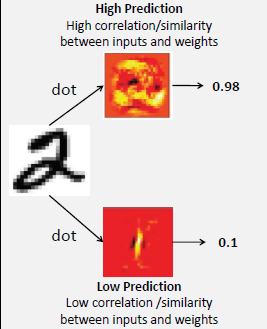
* Straightforward, alle Inputs werden in allen Outputs verarbeitet
* Bei 3 In und 3 Outputs gibt es 9 Weights
* 
* Aus jedem Error und Delta werden die neuen Weights berechnet

## Correlation

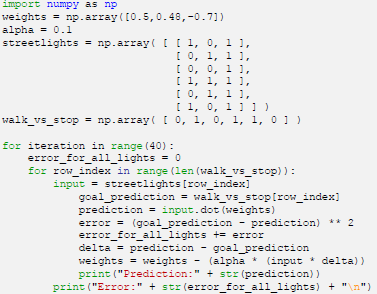
### Learning Correlation

* Was lernen die Gewichte im Ganzen?



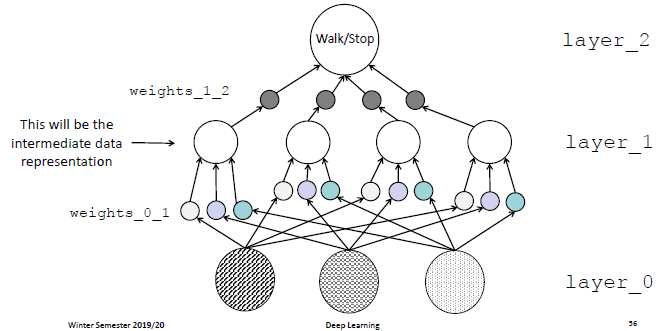
* Wenn ein Gewicht hoch ist, denkt das Model, dass es viel Korrelation zwischen diesem Input und der Prediction gibt
* Gewichte werden via Dot Products gefunden und Dot Products kodieren Ähnlichkeit
* 

Wie lernt man mit kompletten Datensätzen?

* Alle Daten iterieren und dann erst neuen Trainingszyklus starten
* 
* Manche Inputs bekommen mehr upward und manche mehr downwards pressure, das heißt nach einigen Iterationen wir sich ein Input als wertvoller als der andere herausstellen
* Jede Node versucht individuell den Output zu predicten, einziger Share ist der gemeinsame Error
* Lernen belohnt Korrelation mit größeren Weights, bestraft disccorr mit niedrigen

### Creating Correlation

* Was wenn es keine Korrelation zwischen In- und Output gibt? Error wird nie 0 erreichen!
* Da ein Netzwerk Korrelation zwischen In- und Outputlayern sucht, nutzen wir einfach mehr Netzwerke bzw. mehr Layer: Layer 1 hat limitierte Korrelation zu Output, Hidden Layer nutzt diese um den Output zu predicten „Intermediate Datasets“



Weights der oberen Hälfte werden wie bisher geupdatet, Rest neu:

* Wenn man‘s so wie bisher (lineare Transformation) macht bringt das Hidden Layer nichts, weil mehrere Multiplikationen prinzipiell auch in einer gemacht werden könnten
* In linearem Netz haben Hidden keine eigene Korrelation, weil sie mit den Inputnodes korrelieren. Idealerweise sollen sie manchmal und machmal nicht mit Input korrelieren („Conditional Correlation“)
* Hierzu braucht man nicht-lineares:

Beim Deep Learning geht’s darum, Zwischenlayer zu kreieren. Jede Node darin repräsentiert das Dasein oder Nichtdasein einer anderen Inputkonfiguration

* Kein individueller Input muss direkt mit Target korrelieren

Es geht also darum, eine Konfiguration von Inputs zu finden, die den Output kreieren, nicht einzelne Inputs sollen vom Output abhängig sein

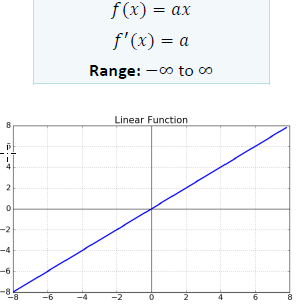
### ACTIVATION FUNCTION

* Benutzt den gewichteten Inputwert um das Level der Outputactivation zu bestimmen
* Wird auf Neuronen im Layer angewandt

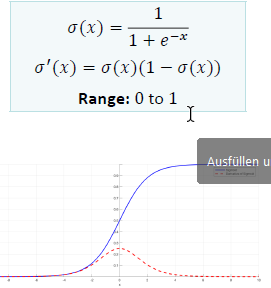
#### Grundeigenschaften

* Funktion hat unendlichen Wirkungsbereich
* Ändert nie die Richtung
* Sind nichtlinear
* Effizient zu berechnen

#### Linear

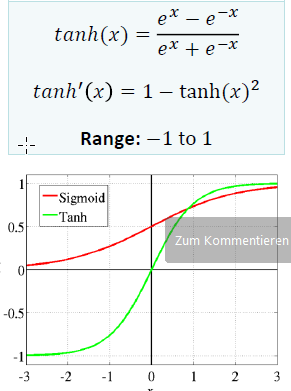


#### Sigmoid

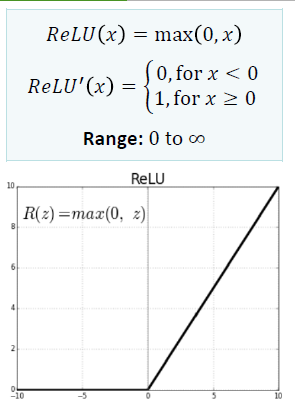
Komplexer und wird nicht allzu groß, aber bei großem x wird der derivative fast 0

* - Vanishing Gradient möglich

#### Hyperbolic Tangent

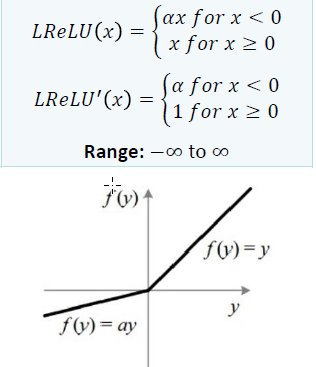
* Komplexere Patterns möglich
* Nicht allzu große Values
* Negativer Input gibt negativen Output
* 0 gibt 0
* Vanishing Gradient möglich

#### Rectified Linear Unit (ReLu)



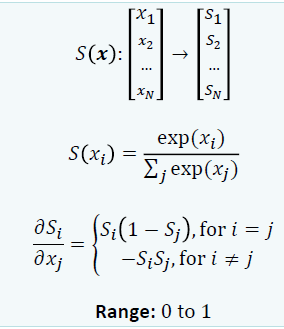
* Komplexe Patterns
* Kann sehr großen Output geben
* Leichte Ableitung
* Keine Negativen: Manche Strukturen koennten verpasst werden
* Dying RELU: gradient wird null und Gewicht wird nicht geupdatet

#### Leaky Rectified Linear Uni (LRELU)



* Komplex, großer Output
* Leichte Ableitung
* Soll RELU Probleme lösen

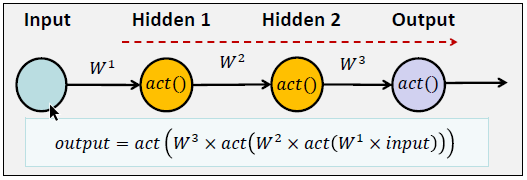
#### Softmax



* Modelliert Wahrscheinlichkeitsverteilung
* Values zwischen 0 und 1, Summe gibt 1
* Typischerweise nur in Outputlayer benutzt

## Backpropagation

* Bei Feed Forward geht’s von links nach rechts



* Backpropagation ist Tool für mehrere Layer, Gewichte werden berechnet mit Respekt zu jedem Weight

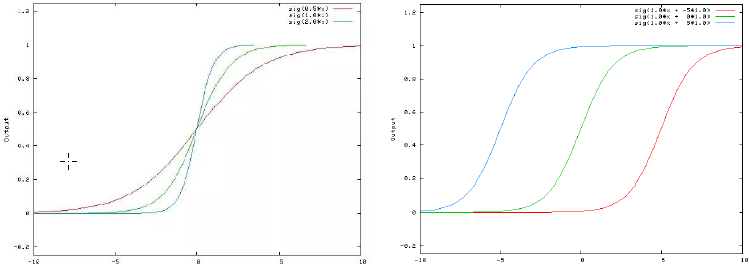
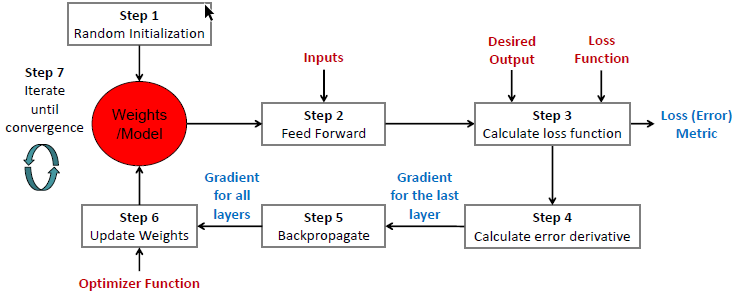
## Update Weights

## Weights/Model

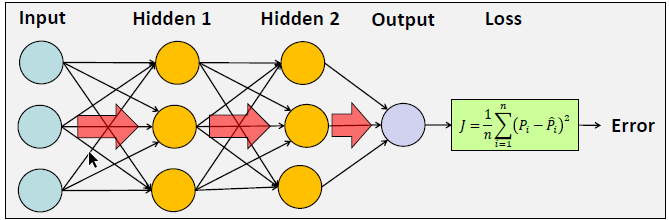
### Bias

* Konstanter Vektor, das zu jedem Produkt Input\*Weights hinzugefügt wird



* Activationfunktion kann nach link und rechts angepasst werden, zur besseren Anpassung an die Daten
* Starten gewöhnlich mit 0 und werden vom Algorithmus angepasst, sind nicht abhängig vom Output früherer Layer
* 
* Gewichte beeinflussen Steigung(links), Bias den links/rechts Shift (bei Act. Function)
* 

### Loss Function



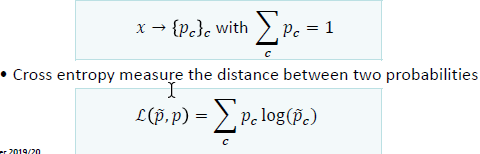
* Misst Differenz zwischen Vorhersage ^y und was es vorhersagen hätte sollen
* Mit dieser Info kann das Model geupdatet werden
* Loss ist Prediction error
* Loss Function berechnet den Loss
* Loss wird benutzt, um Gradient zu berechnen
* Gradient wird benutzt, um Weights upzudaten

#### Mean Squared Error (MSE)

Üblicherweise für Regression genutzt

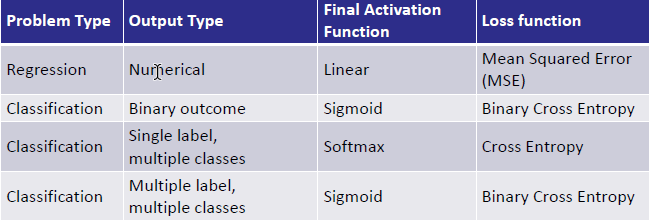
#### Cross Entropy

Für Klassifizierungen, das Model sagt die Warhscheinlichkeit voraus, das eine Beobachtung ein bestimmtes Label hat



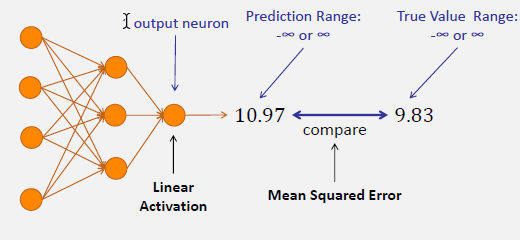
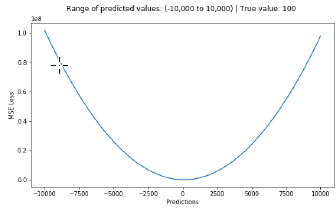
### Loss & Activationfunction relationship

Je nach Output sollte man eine andere Costfunction wählen

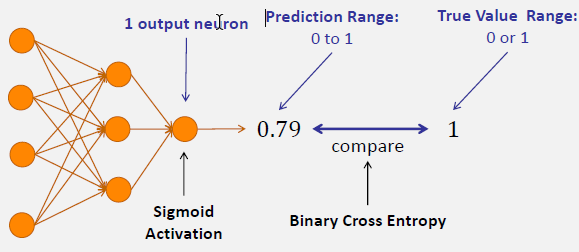


* Was versucht man zu lösen/vorherzusagen?
* Binary: Data ist (k)eine Klasse?

#### Regression

#### Binary Outcome



* Sigmund führt zu Wert zwischen 0 und 1, aus dem man Klassenzuordnung schließen kann
* Binary Cross Entropy:

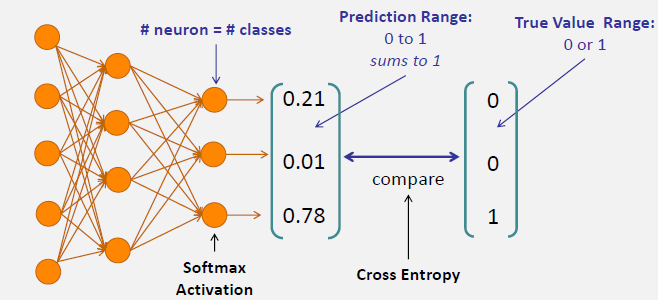
Model sagt Verteilung {p, 1-p} vorraus

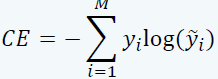
Wir vergleichen dies mit echter

Verteilung {y,1-y}

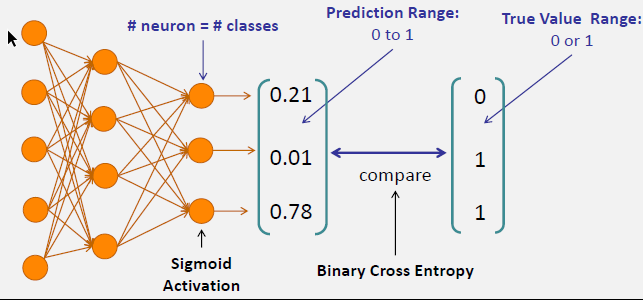


#### Single Label – Multiple Classes

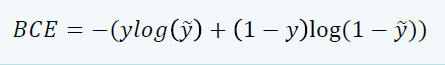


* Softmax gibt Werte zwischen 0 und 1 die addiert 1 ergeben, Wahrscheinlichkeitsverteilung
* Cross Entropy für Differenz
* 

#### Multiple Layers – Multiple Classes



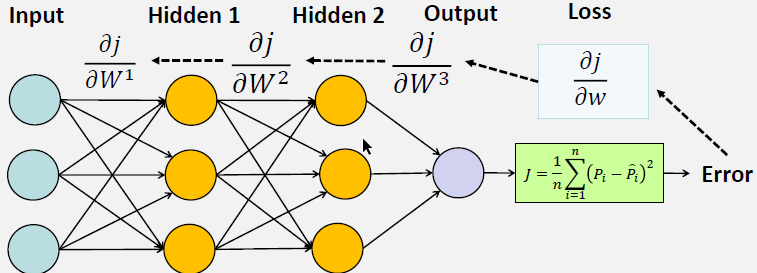
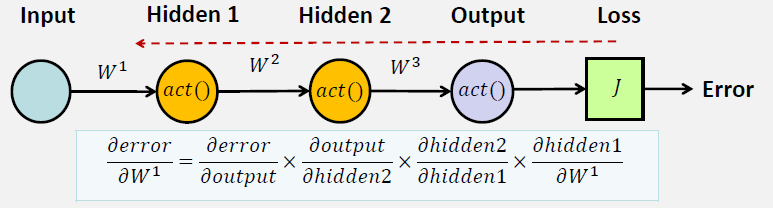
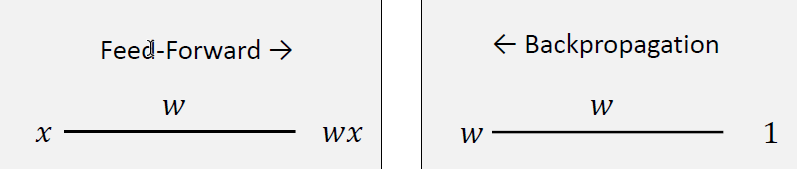
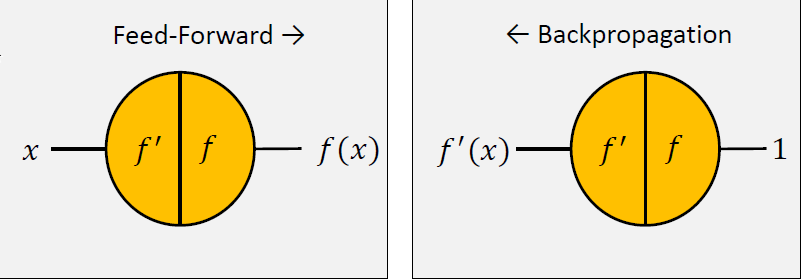
* Das letzte Layer hat ein Neuron für jede Klasse, welches Wert zw. 0 und 1 zurückgibt
* Mit BCE vergleichen

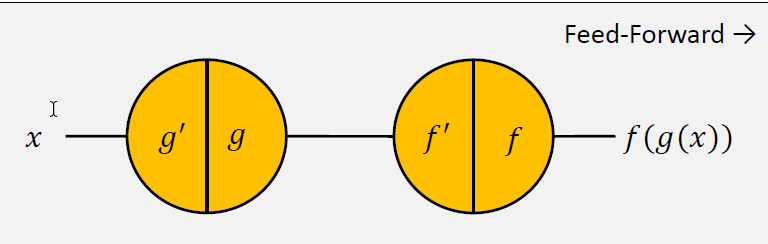


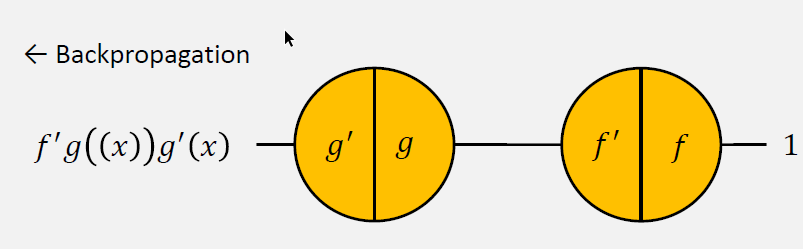
## Grundregeln der Backpropagation

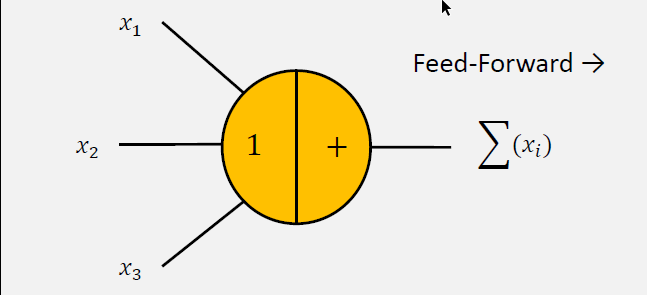
* Gewichte Updaten um Lossfunction zu minimieren
* Gradient der Lossfunction wird genommen, hierbei werden alle vorangegangenen Gewichte respektiert

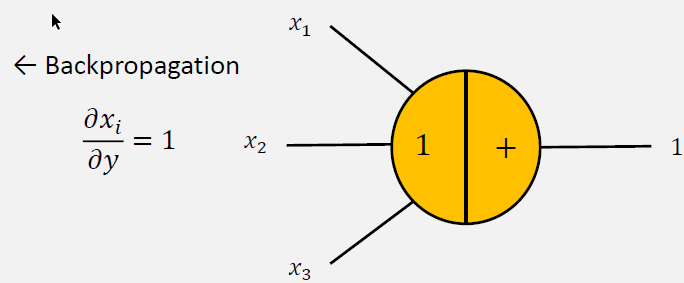


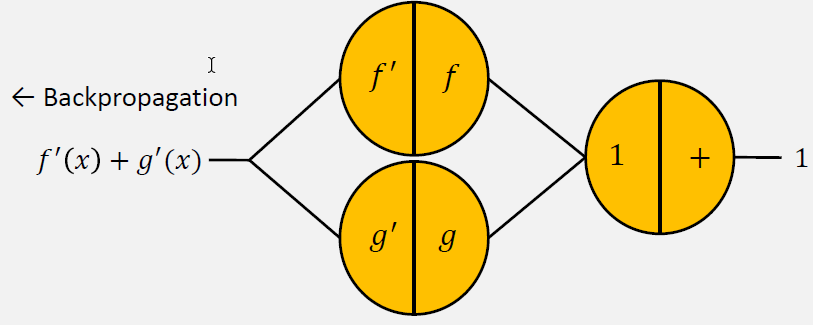
* Von hinten nach vorne werden Gewichte geupdatet
* Gradient der Lossfunction wird berechnet
* 
* 
* Weights
* Act Function



Composite Function

Summe



Addition of Functions

## Optimierungsalgorithmen

### Gradient Descent

* Gradient gibt Richtung des Increase an, deshalb verwendet man gegenteilige Richtung um das Minimum zu finden

#### Stochastic Gradient Descent

* Prediction und Weight für jedes Trainingsbeispiel selber geupdatet
* 
* Vorteile: Schneller als Batch, Gewichte leicht upgradebar wenn neue Beispiele benötigt
* Nachteile: Beim ständigen Updaten schwankt die Kostfunktion stark

#### Full Gradient Descent

* Berechnet durchsch. Weight-delta über komplettes Datenset, Updatet nur sobald kompletter Schnitt berechnet wurde

#### Batch Gradient Descent



* Batchsize zwischen 8 und 256 aussuchen um weight\_delta zu berechnen
* Respektiert beim Gradient berechnen das ganze Trainingsset
* Leicht zu verstehen
* Redundante Computation
* Langsam

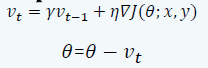
#### MiniBatch



* Minibatch für alle n Trainingsbeispiele
* Stabiler, schneller, gute Approximation
* Aber gesamter Loss nicht aufaddiert

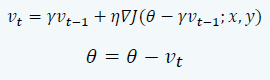
### Optimisierungstypen

#### Momentum

* Beschleunigt annäherung des GD
* Nimmt jetzigen GD und den vorheriger Steps
* Hilft bei Kostfunktion mit Schlucht
* 

#### Nesterov Accelerated Gradient

* Momentum nicht immer gut, manchmal muss man langsam machen
* Man nimmt den künftigen Step zum Kalkulieren



#### Adagrad – Adaptive Gradient Algorithm

CCNs & RNNs

## CNN

### Convolution

### Convolutional Layers

#### Backpropagation

### Max Pooling

#### Backpropagation

## RNN

### Motivation

### Network Structures

### Backpropagation

### Gated RNN

### LSTM

### Gated Recurrent Unit

# Backpropagation & Regularisation

## Backpropagation

## Generalisation

### Generalisation of a supervised model

### Data Partitioning

### Generalisation Errors

## Regularisation

### Mini-Batch Learning

### Weight Penalties

### Data Augmentation

### Training with Noise

### Dropout

### Early Stopping

# Sequence To Sequence Learning

## Natural Language Processing

### Language Models

## Deep Learning

### Word Embedding

### Neural Language Models

## Sequence to Sequence Learning

## Attention Mechanisms

## Connectionist Temporal Classification