hochschule mannheim

EMB-Lab, Fak. N

Lineare Klassifikation 2

DLM – Deep Learning Methoden

Benjamin Kraus

Kevin Höfle, Prof. Dr. Marcus Vetter

Mannheim, 15.10.2018

Inhalt

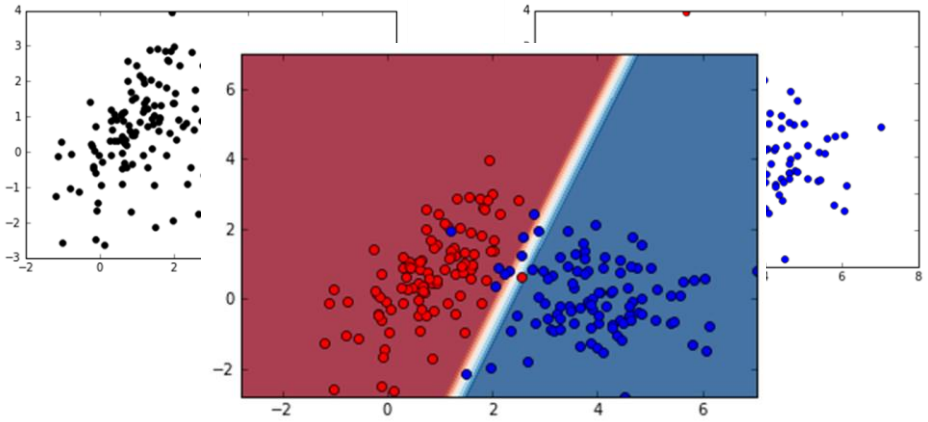
- Organisatorisches
- Recap der letzten Wochen
- Loss function
  - SVM
  - Softmax
  - Regularisierung
- Optimierung anhand der Loss function
- Gradientenabstieg
- Lernraten
- Ausblick

Übersicht der Themen und Termine

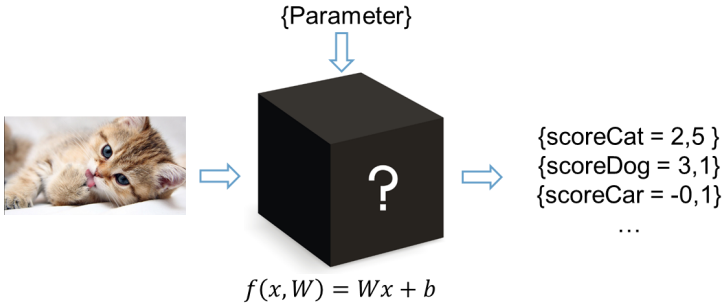
Datum	Nr.	Thema der Vorlesung	Themen der Praktika
01. Okt	1	Einführung in Deep Learning	Einführung in Python
08. Okt	2	Lineare Klassifikatoren	Einführung in Numpy
15. Okt	3	Loss Function Optimization & Fully Connected Layer	Übung zu Linearen Klassifikatoren
22. Okt	4	Backpropagation and neural Networks	Übung zu Backpropagation mit Numpy
29. Okt	5	Architektutren & Layer I (Conv, Drop usw.)	Keras Sequential am Beispiel von MNIST
05. Nov	6	Klassifikation, Regression, Segmentierung, Generation & Regularisierungsmethoden	Keras mit CIFAR selbst machen
12. Nov	7	Architekturen & Layer II (Unet, Recurrent, Skips usw.)	Keras Segmentieren Übung
19. Nov	8	Training von Convolutional Neural Network(+Transfer Learning)	LSTM mit Keras
26. Nov	9	Recap & Ausgabe Semesteraufgaben	Semesteraufgabe
03. Dez	10	Betreuung der Semesteraufgabe	
10. Dez	11	Betreuung der Semesteraufgabe	
17. Dez	12	Betreuung der Semesteraufgabe	
07. Jan	13	Präsentation der Ergebnisse	
14. Jan	14	Prüfung	

Recap

- Klassifikation –
  - Was ist das?
  - Welche Probleme gibt es?
- Data-driven Approach
- Lineare Klassifikation
- Score Function

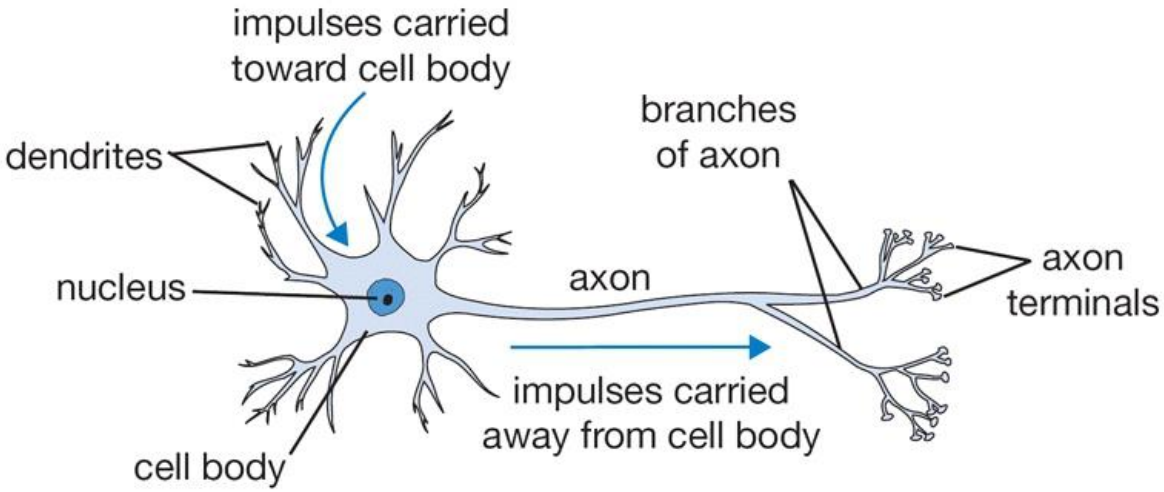


$$f(x, W, b) = Wx + b$$



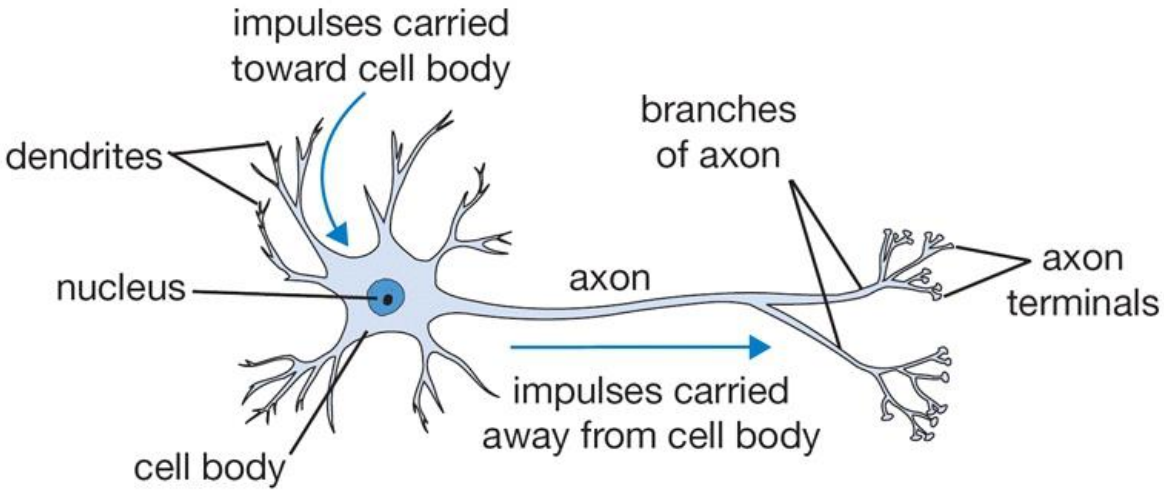
Recap

- Nochmal die Biologie:



Recap

- Nochmal die Biologie:

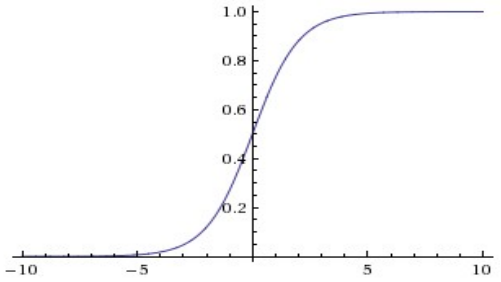


Summe

$$\sum_i w_i x_i + b$$

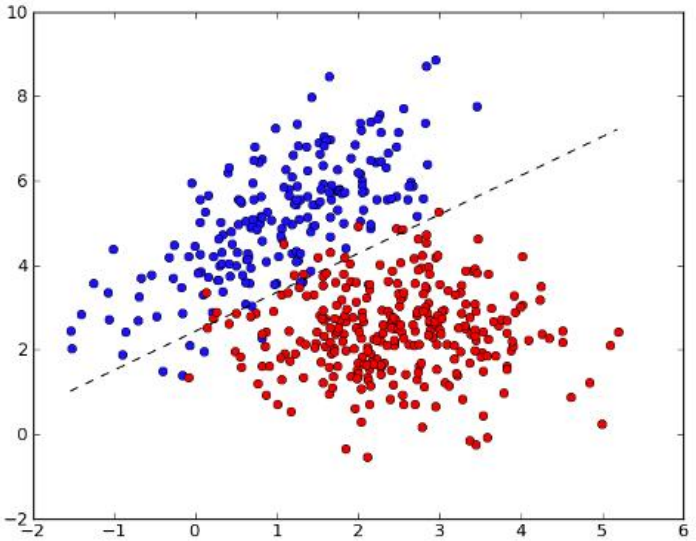
Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



Recap

- Nochmal die Biologie:
- Damit lässt sich ein lineares Klassifikationsproblem lösen!



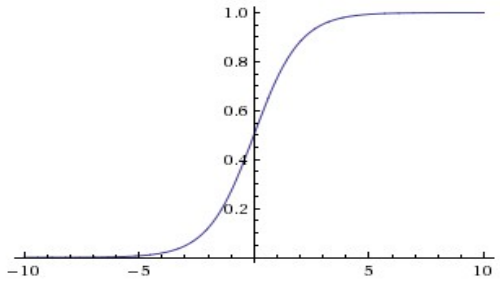
- …und kommt uns bekannt vor

Summe

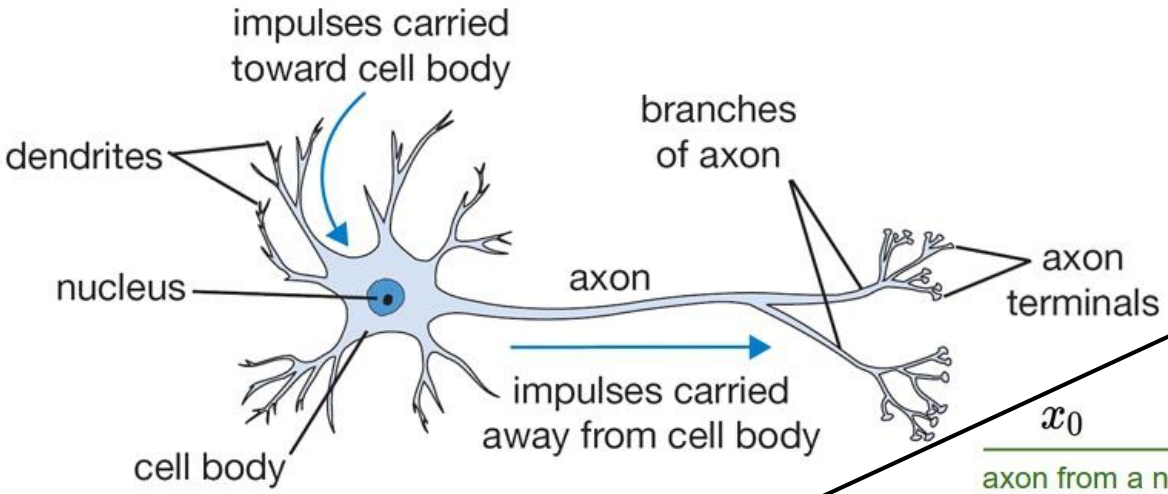
$$\sum_i w_i x_i + b$$

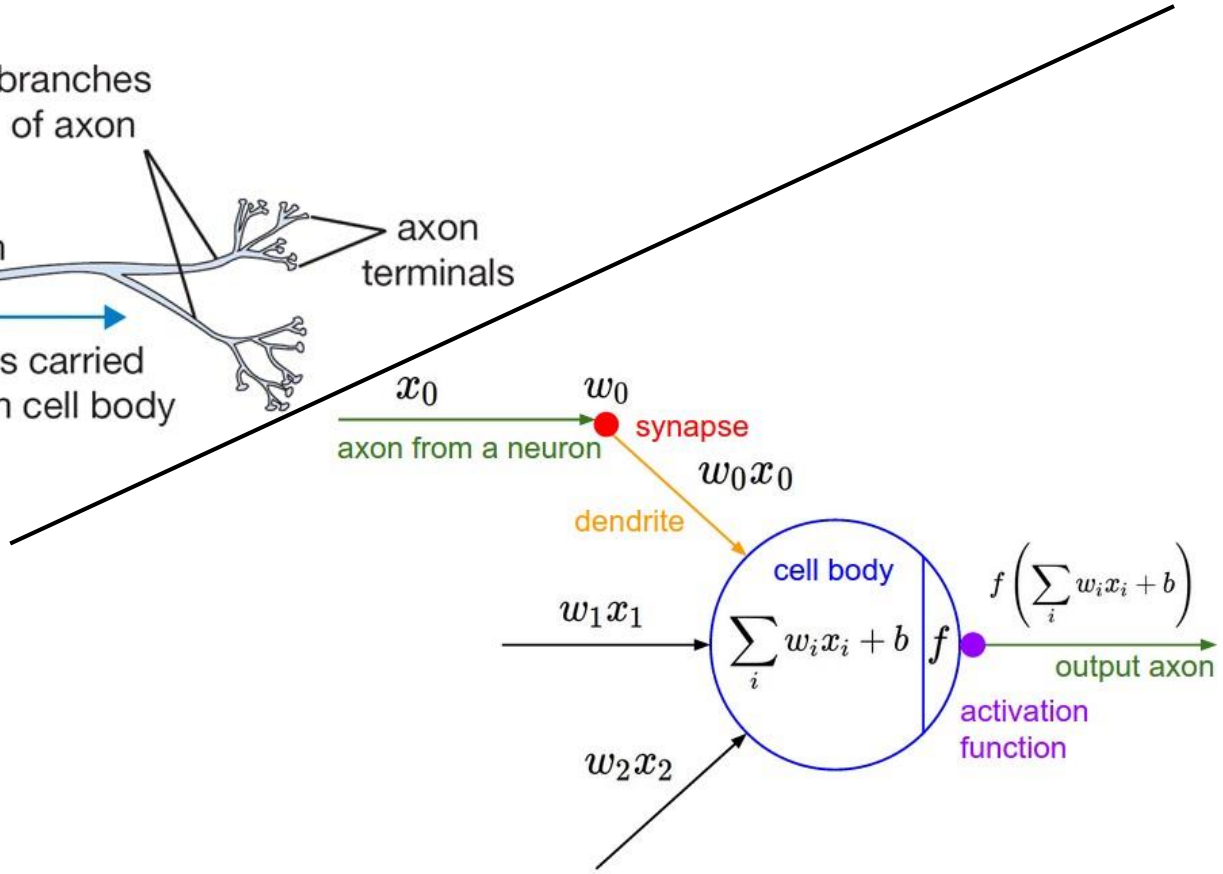
Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$



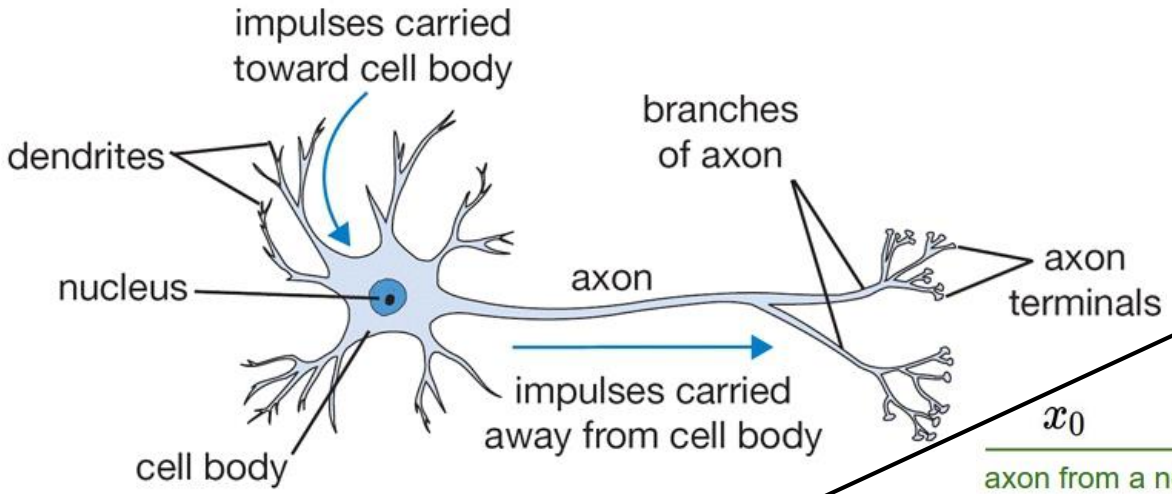
Recap

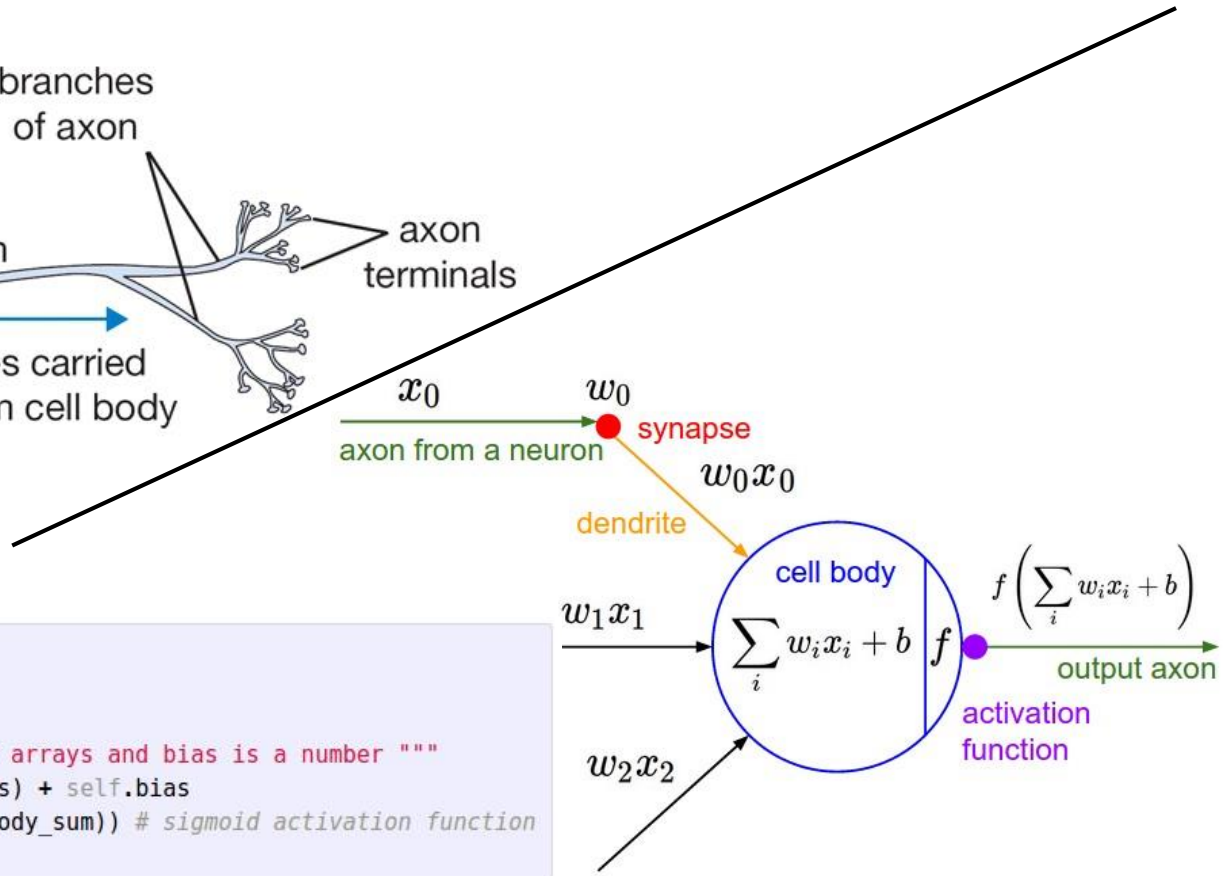




[Andrej Karpathy, Stanford Universitiy]

Recap

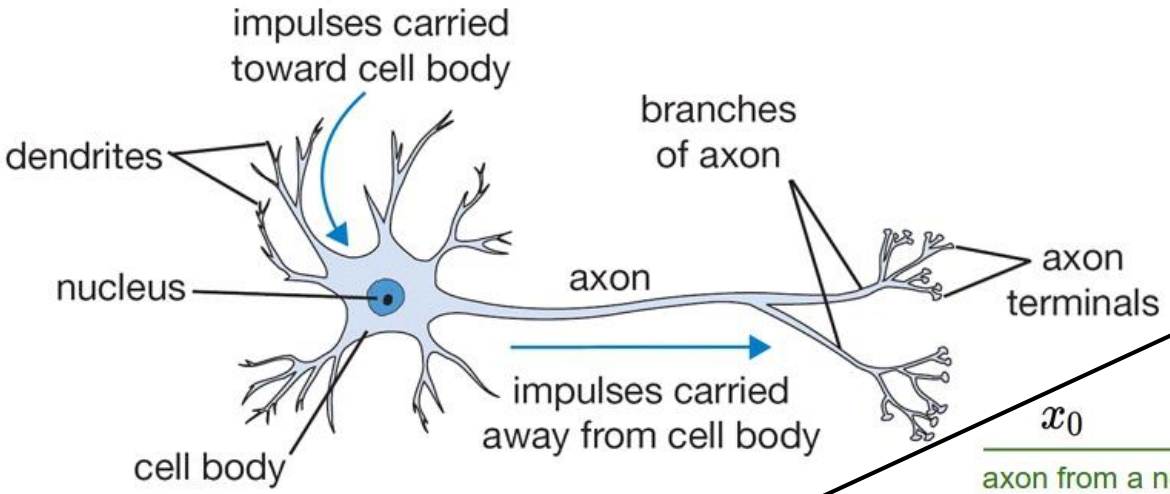


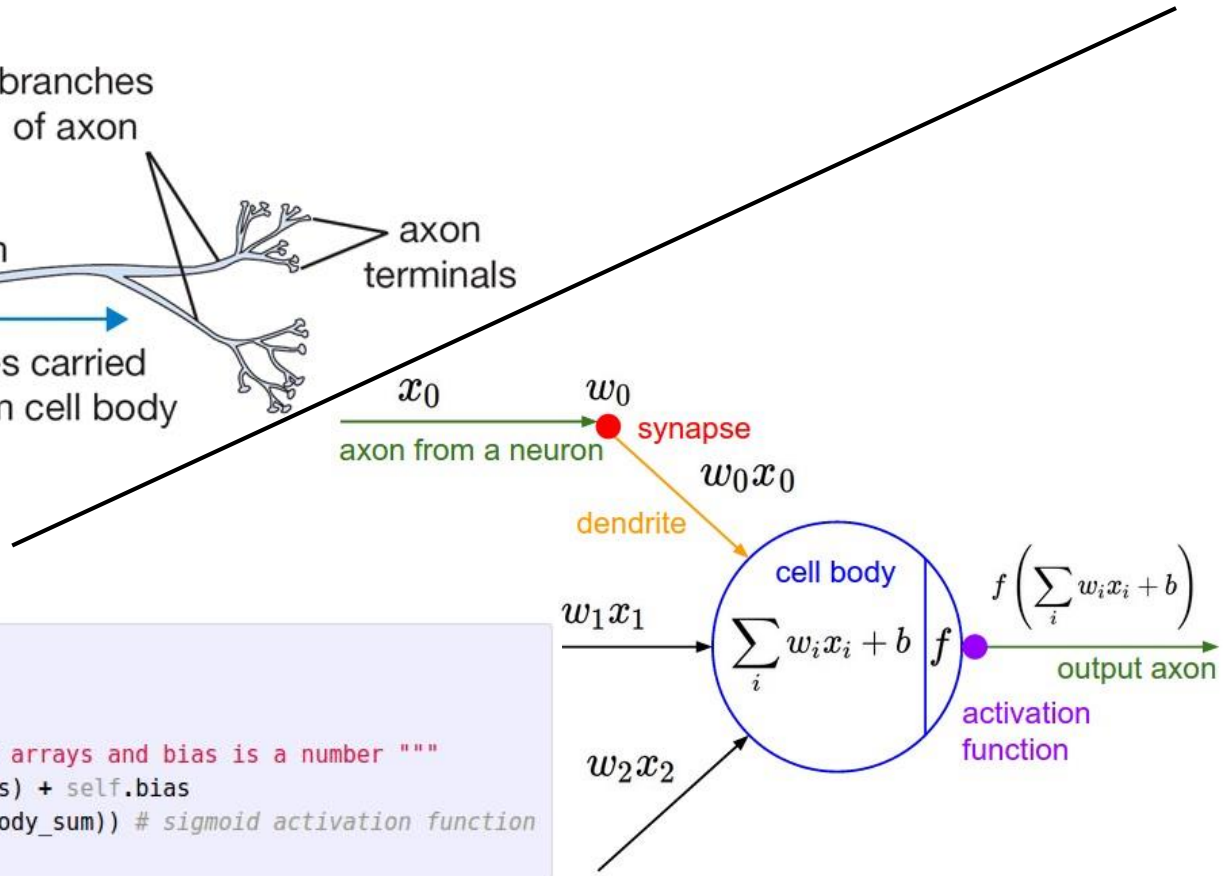



```
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

[Andrej Karpathy, Stanford Universitiy]

Recap






```python
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

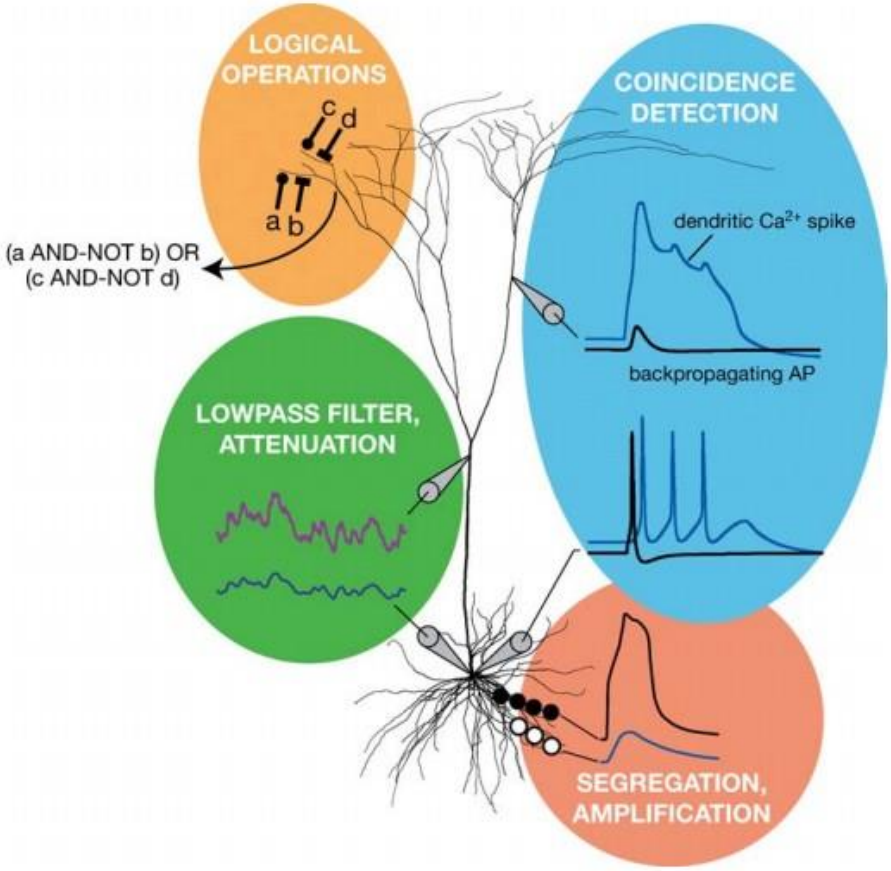
[Andrej Karpathy, Stanford Universitiy]

Recap

Vorsicht mit Gehirn-Analogien:

Biologische Neuronen sind vielseitiger:

- Viele verschiedene Typen
- Dendriten können komplexe non-lineare Berechnungen realisieren
- Synapsen sind nicht ein einzelnes Gewicht sondern ein nicht-lineares dynamisches System



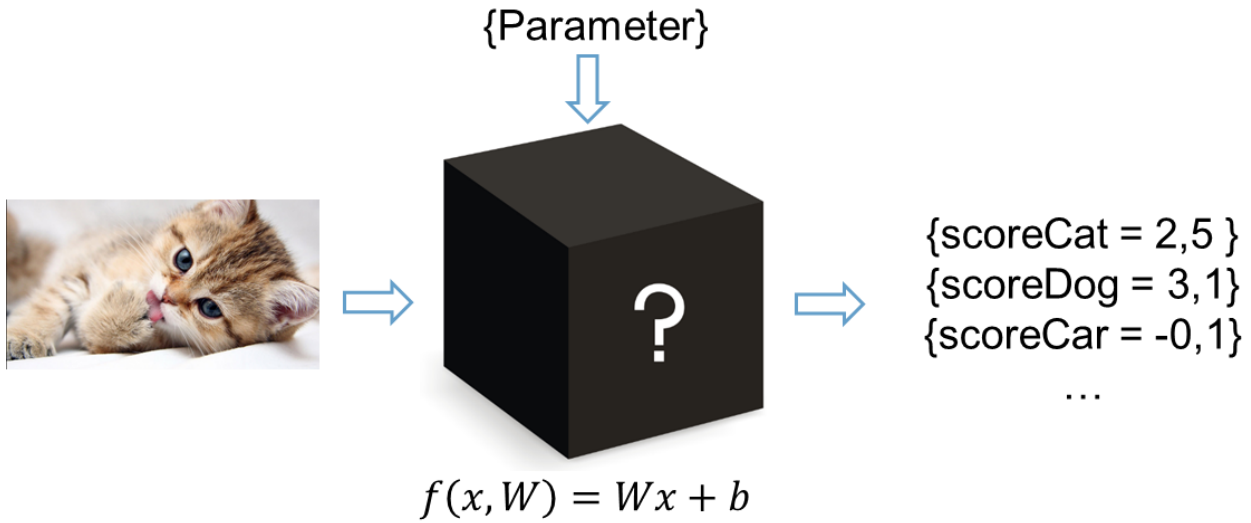
[Dendritic Computation. London and Hausser]

Recap

- Score Function:

$$f(x, W, b) = Wx + b$$

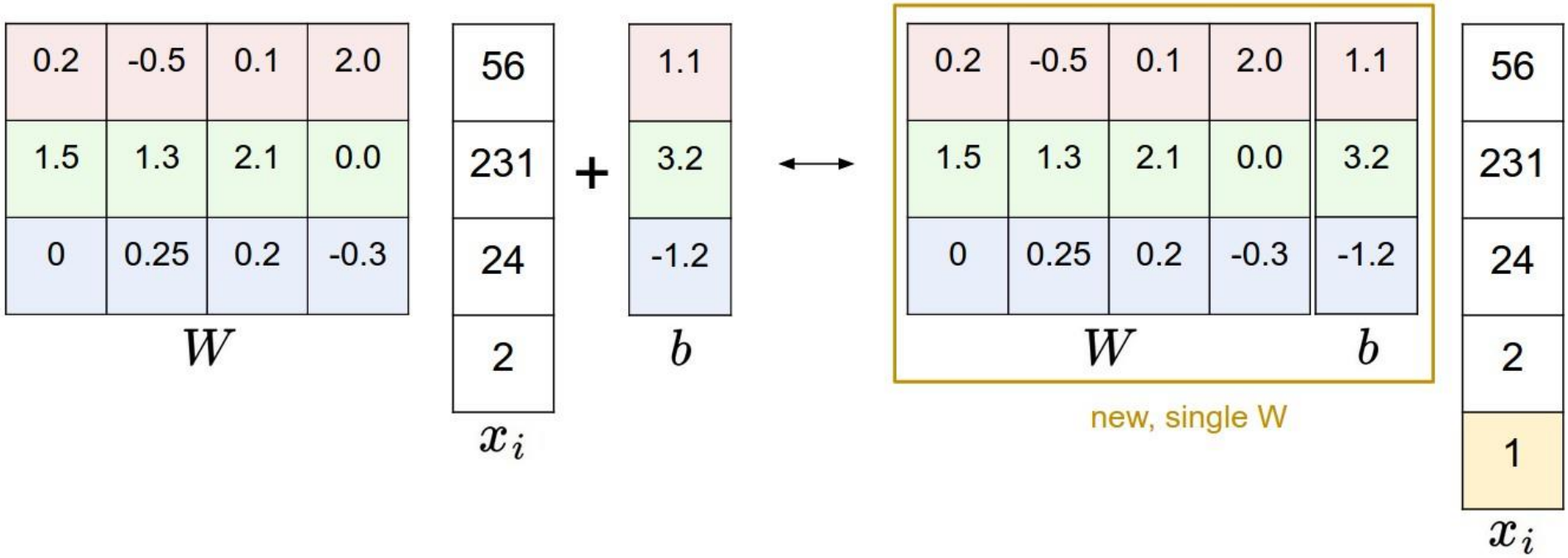
- Funktion die Pixelwerte auf Klassen abbildet
  - Wir erhalten einen Wert pro Klasse
  - Wir fordern, dass der Wert der richtigen Klasse der Größte sein soll



Bias-Trick

- Score Function:

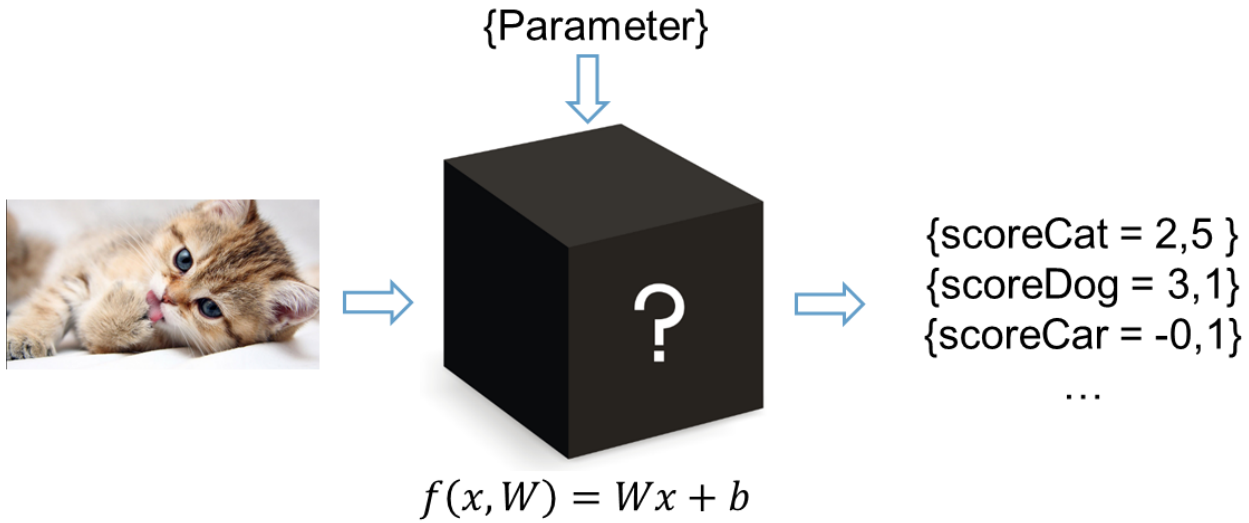
$$f(x, W, b) = Wx + b \quad \longrightarrow \quad f(x, W, b) = Wx$$



[Andrej Karpathy, Stanford Universitiy]

Loss Function

- Dies ist aber nicht automatisch der Fall
  - Ein zufällig initialisiertes *W* liefert wahrscheinlich keine “guten Ergebnisse”
  - Wir haben noch nicht definiert was ein “gutes Ergebnis” genau ist



Loss Function

- Was ist ein “gutes Ergebnis”?
  - Wir brauchen ein Maß unserer (Un-) Zufriedenheit mit der aktuellen Klassifikation um weitere Schritte davon ableiten zu können

$$f(x, W) = \begin{matrix} \{\text{scoreCat} = 1{,}0\} \\ \{\text{scoreDog} = 0{,}2\} \\ \{\text{scoreCar} = -0{,}5\} \end{matrix}$$

$$f(x, W) = \begin{matrix} \{\text{scoreCat} = 2{,}5\} \\ \{\text{scoreDog} = 3{,}1\} \\ \{\text{scoreCar} = -0{,}1\} \end{matrix}$$

Loss Function

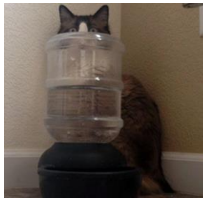
- Was ist ein “gutes Ergebnis”?

  - Wir brauchen ein Maß unserer (Un-)Zufrienheit mit der aktuellen Klassifikation um weitere Schritte davon ableiten zu können

  - Wir müssen eine Loss Function (Cost Function / Objective) definieren
    - Der Loss soll hoch sein, wenn der Klassifikator schlecht arbeitet

Gut:

$f(x, W) =$ {scoreCat = 1,0}
{scoreDog = 0,2}
{scoreCar = -0,5}



Schlecht:

$f(x, W) =$ {scoreCat = 2,5}
{scoreDog = 3,1}
{scoreCar = -0,1}

Loss Function

- Es gibt viele Varianten solch eine Funktion zu definieren
- Zu den Gebräuchlichsten zählen:

  - Multiclass SVM Loss (Hinge Loss)
  - Softmax (Multinomial Logistic Regression)

Hinge Loss / Multiclass SVM Loss

Ein Bild x_i sei dem Klassen-Label y_i zugeordnet

Der Funktionswert von $f(x_i, W)$ sei s

wobei s_j der Score von x_i für die Klasse j ist,

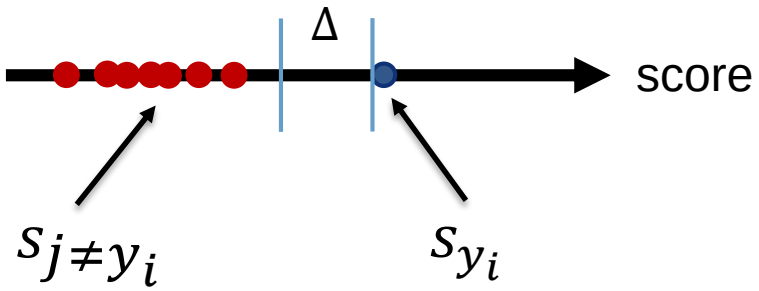
und s_{y_i} der Score von x_i für die Klasse y_i ist.

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + \Delta)$$

Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:		

Hinge Loss / Multiclass SVM Loss

- Wie kann man sich das vorstellen?
  - Hinge Loss ist dann Ideal wenn zwischen der richtigen Klasse und allen anderen Klassen mindestens der Abstand Δ liegt



$$L_i = \sum_{j \neq y_i} \max(0; \quad s_j - s_{y_i} + \Delta)$$

Hinge Loss / Multiclass SVM Loss

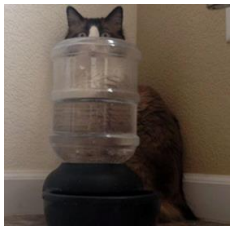
Ein Bild x_i sei dem Klassen-Label y_i zugeordnet

Der Funktionswert von $f(x_i, W)$ sei s

wobei s_j der score von x_i für die Klasse j ist,

und s_{y_i} der score von x_i für die Klasse y_i ist.

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + \Delta)$$



Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:		

Hyperparameter,
$\Delta = 1$ funktioniert fast immer

Hinge Loss / Multiclass SVM Loss

Ein Bild x_i sei dem Klassen-Label y_i zugeordnet

Der Funktionswert von $f(x_i, W)$ sei s

wobei s_j der score von x_i für die Klasse j ist,

und s_{y_i} der score von x_i für die Klasse y_i ist.

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:		

Hinge Loss / Multiclass SVM Loss

Ein Bild x_i sei dem Klassen-Label y_i zugeordnet

Der Funktionswert von $f(x_i, W)$ sei s

wobei s_j der score von x_i für die Klasse j ist,

und s_{y_i} der score von x_i für die Klasse y_i ist.

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	

Hinge Loss / Multiclass SVM Loss

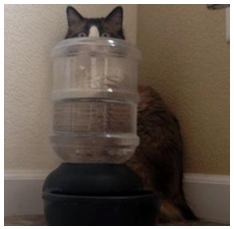
Ein Bild x_i sei dem Klassen-Lable y_i zugeordnet

Der Funktionswert von $f(x_i, W)$ sei s

wobei s_j der score von x_i für die Klasse j ist,

und s_{y_i} der score von x_i für die Klasse y_i ist.

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$



Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	1,6

Hinge Loss / Multiclass SVM Loss

Ein Bild x_i sei dem Klassen-Lable y_i zugeordnet

Der Funktionswert von $f(x_i, W)$ sei s

wobei s_j der score von x_i für die Klasse j ist,

und s_{y_i} der score von x_i für die Klasse y_i ist.

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

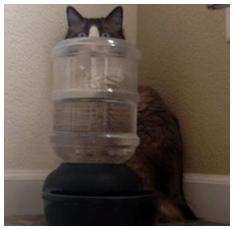
Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	1,6

Loss über alle Trainingsdaten:
(0,2+1,6) /2 = 0,9

Hinge Loss / Multiclass SVM Loss

Fragen:

Was ist der minimale Loss und wann wird dieser erreicht?



Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	1,6

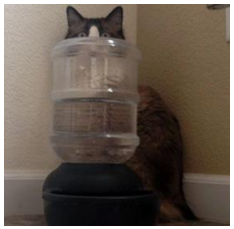
$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Hinge Loss / Multiclass SVM Loss

Fragen:

Was ist der minimale Loss und wann wird dieser erreicht?



0 wird erreicht, wenn alle Scores den Margin Δ einhalten.

Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	1,6

$$L_i = \sum_{j \neq y_i} \max(0; \quad s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Hinge Loss / Multiclass SVM Loss

Fragen:

Was bedeutet das für das Beispiel s1?

Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	1,6

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Hinge Loss / Multiclass SVM Loss

Fragen:

Was bedeutet das für das Beispiel s1?

Obwohl die Klassifikation richtig war, wird weiter optimiert.

Classes	s1	s2
Cat (i=1)	1,0	2,5
Dog (i=2)	0,2	3,1
Car (i=3)	-0,5	-0,1
Losses:	0,2	1,6

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Hinge Loss / Multiclass SVM Loss

- Code:

$$f(x, W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

```
def L_i_vectorized(x, y, W):
  scores = W.dot(x)
  margins = np.maximum(0, scores - scores[y] + 1)
  margins[y] = 0
  loss_i = np.sum(margins)
  return loss_i
```

[Andrej Karpathy, Stanford Universitiy]

Softmax

- Multinomial Logistic Regression
- Normalisierte Exponentiale
- Cross-entropy loss

- Ziel:
  - Die Loss Function soll besser interpretierbar sein.

  - Ist ein hinge loss von 3,5 gut oder schlecht?

Softmax

- Ansatz :
  - Wir möchten, dass unser Klassifikator Wahrscheinlichkeiten ausgibt.
  - Wir wollen also die Scores zu normierten Wahrscheinlichkeiten zuordnen

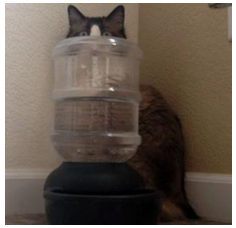
$$\sum_{i=1}^{N} P(y_i) = 1$$

  - Wir möchten die log likelihood für die richtige Klasse optimieren
  - Die richtige Klasse soll ideal eine log Wahrscheinlichkeit von

$$P(y_1|x_1; W) \approx 1$$ haben

Softmax

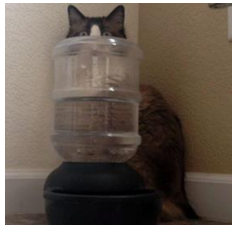
- Vorgehen:
  - Die Scores wie gehabt berechnen $s = f(x, W)$



Classes	s		
Cat (i=1)	2,5		
Dog (i=2)	3,1		
Car (i=3)	-0,1		

Softmax

- Vorgehen:
  - Die Scores als nicht normaliserte log Wahrscheinlichkeiten der Klassen betrachten:

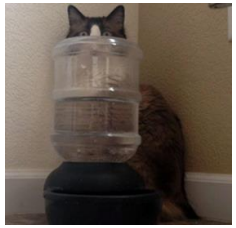


$e^{s_{y_i}}$

Classes	s		
Cat (i=1)	2,5		
Dog (i=2)	3,1		
Car (i=3)	-0,1		

Softmax

- Vorgehen:
  - Die Scores als nicht normaliserte **log** Wahrscheinlichkeiten der Klassen betrachten:

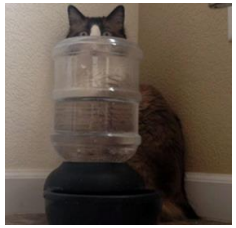


$e^{s_{y_i}}$

Classes	s	exp(s)	
Cat (i=1)	2,5	12,2	
Dog (i=2)	3,1	22,2	
Car (i=3)	-0,1	0,9	

Softmax

- Vorgehen:
  - Die Scores als **nicht normaliserte** log Wahrscheinlichkeiten der Klassen betrachten:



$$\sum_{i=1}^{N} P(y_i) = 1$$

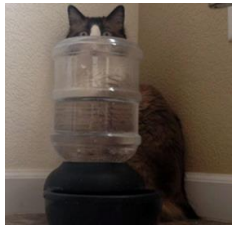
$e^{s_{y_i}}$ $\qquad$ $\frac{1}{\sum_i e^{s_{y_i}}}$

Classes	s	exp(s)	P
Cat (i=1)	2,5	12,2	
Dog (i=2)	3,1	22,2	
Car (i=3)	-0,1	0,9	

Softmax

- Vorgehen:
  - Die Scores als **nicht normaliserte** log Wahrscheinlichkeiten der Klassen betrachten:

$$\sum_{i=1}^{N} P(y_i) = 1$$



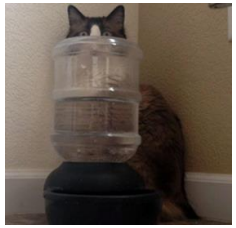
$e^{s_{y_i}}$ $\qquad$ $\frac{1}{\sum_i e^{s_{y_i}}}$

Classes	s	exp(s)	P
Cat (i=1)	2,5	12,2	0,3453
Dog (i=2)	3,1	22,2	0,6291
Car (i=3)	-0,1	0,9	0,0172

1 / (exp(3,1) + exp(2,5)+ exp(-0,1)) = 0,0283

Softmax

- Vorgehen:
  - Die Scores als **nicht normaliserte** log Wahrscheinlichkeiten der Klassen betrachten:



$e^{s_{y_i}}$ $\quad$ $\frac{1}{\sum_i e^{s_{y_i}}}$

Classes	s	exp(s)	P
Cat (i=1)	2,5	12,2	0,3453
Dog (i=2)	3,1	22,2	0,6291
Car (i=3)	-0,1	0,9	0,0172

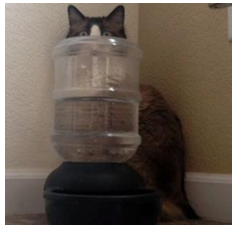
$$\sum_{i=1}^{N} P(y_i) = 1$$

0,35
+0,63
+0,02
1,00

1 / (exp(3,1) + exp(2,5)+ exp(-0,1)) = 0,0283

Softmax

- Vorgehen:
  - Die Scores als **nicht normaliserte** log Wahrscheinlichkeiten der Klassen betrachten:



$e^{s_{y_i}}$ $\quad$ $\frac{1}{\sum_i e^{s_{y_i}}}$

Classes	s	exp(s)	P
Cat (i=1)	2,5	12,2	0,3453
Dog (i=2)	3,1	22,2	0,6291
Car (i=3)	-0,1	0,9	0,0172

$$\sum_{i=1}^{N} P(y_i) = 1$$

0,35
+0,63
+0,02
1,00

1 / (exp(3,1) + exp(2,5)+ exp(-0,1)) = 0,0283

Softmax

- Wir wissen nun wie man von den Scores zu Wahrscheinlichkeiten kommt

$$P = \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

Softmax

- Wir wissen nun wie man von den Scores zu Wahrscheinlichkeiten kommt

$$P = \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

- Eine hohe Warscheinlichkeit für die richtige Klasse entspricht also einem geringen Loss.
  - Wir müssen unsere Formel nur noch leicht anpassen um dies zu erreichen:

$$L_i = -\log \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

Softmax

- Wir wissen nun wie man von den Scores zu Wahrscheinlichkeiten kommt

$$P = \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

- Eine hohe Warscheinlichkeit für die richtige Klasse entspricht also einem geringen Loss.
  - Wir müssen unsere Formel nur noch leicht anpassen um dies zu erreichen:

$$L_i = -\log \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

Welchen Wert nimmt $\log(x)$ für 1 und 0 an?

Softmax

- Wir wissen nun wie man von den Scores zu Wahrscheinlichkeiten kommt

$$P = \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

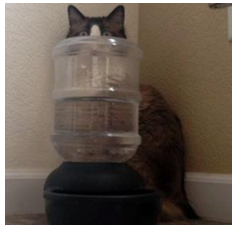
- Eine hohe Warscheinlichkeit für die richtige Klasse entspricht also einem geringen Loss.
  - Wir müssen unsere Formel nur noch leicht anpassen um dies zu erreichen:

$$L_i = -\log \left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}} \right)$$

Welchen Werte nimmt $\log(x)$ für 1 und 0 an?
$\log(1) = 0$ und $\log(0) \to -\infty$

Softmax

- Vorgehen:
  - Die Scores als **nicht normaliserte** log Wahrscheinlichkeiten der Klassen betrachten:



$e^{s_{y_i}}$ $\quad$ $\frac{1}{\sum_i e^{s_{y_i}}}$

Classes	s	exp(s)	P
Cat (i=1)	**2,5**	**12,2**	**0,3453**
Dog (i=2)	3,1	22,2	0,6291
Car (i=3)	-0,1	0,9	0,0172

$$L_i = -\log(0{,}3453) = 1{,}0635$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + R(W)$$

Gradientenabstieg

- Score Function:
  - Wie teilt der Klassifikator Features zu Klassen ein

$$f(x, W) = Wx$$

- Loss Function:

$$L_i = \sum_{j \neq y_i} \max(0;\ \ s_j - s_{y_i} + 1)$$

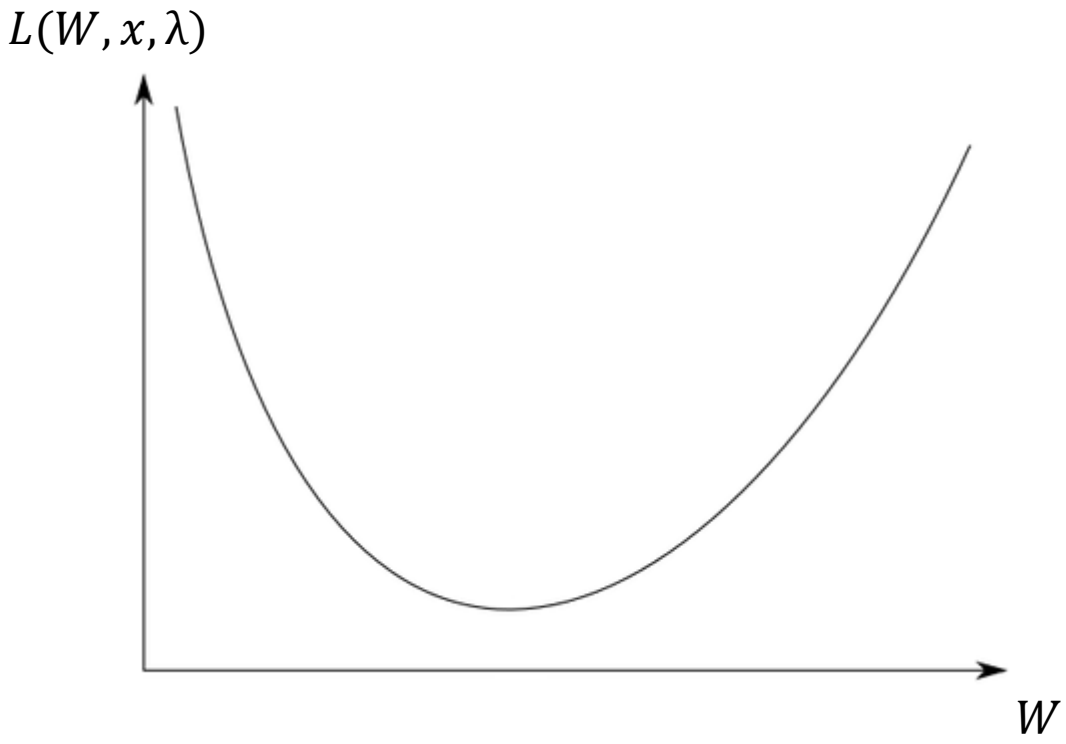
$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_i e^{s_{y_i}}}\right)$$

- Was fehlt?
  - Eine Strategie wie W zu verbessern ist

Gradientenabstieg

- Gute Weights W erzeugen ein Minimum in der Loss Function.
- Also ein Optimierungsproblem!

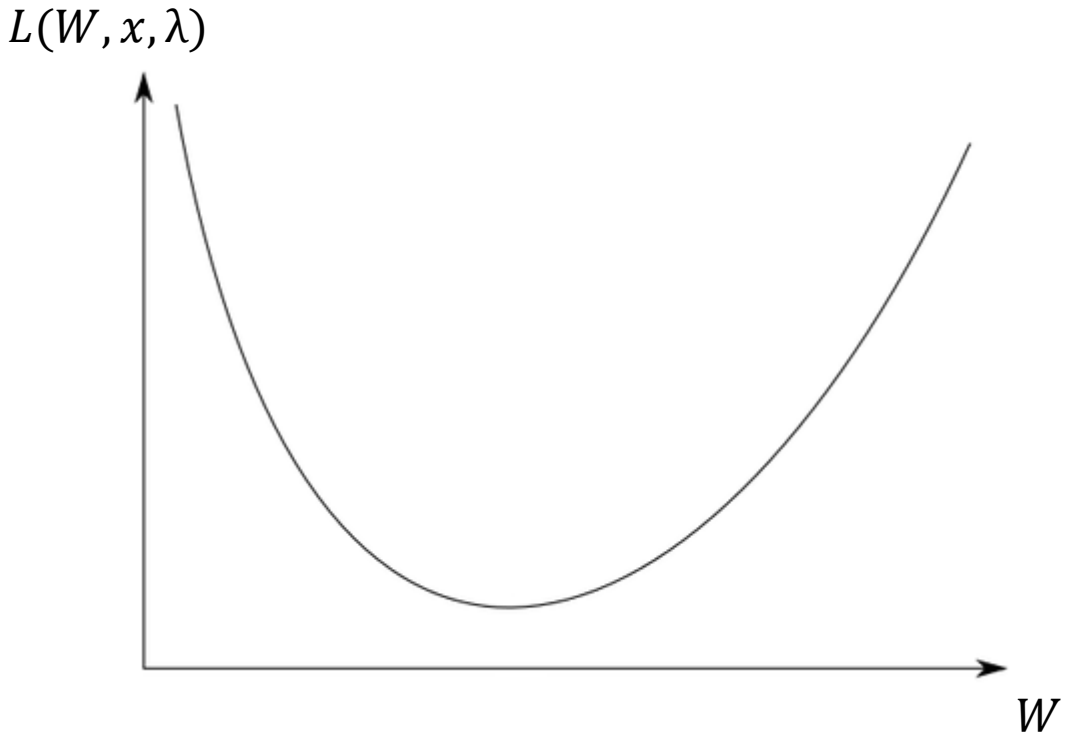
  W sei 1-Dim, L ist eine Funktion von W , den Daten x und den Hyperparametern λ



Gradientenabstieg

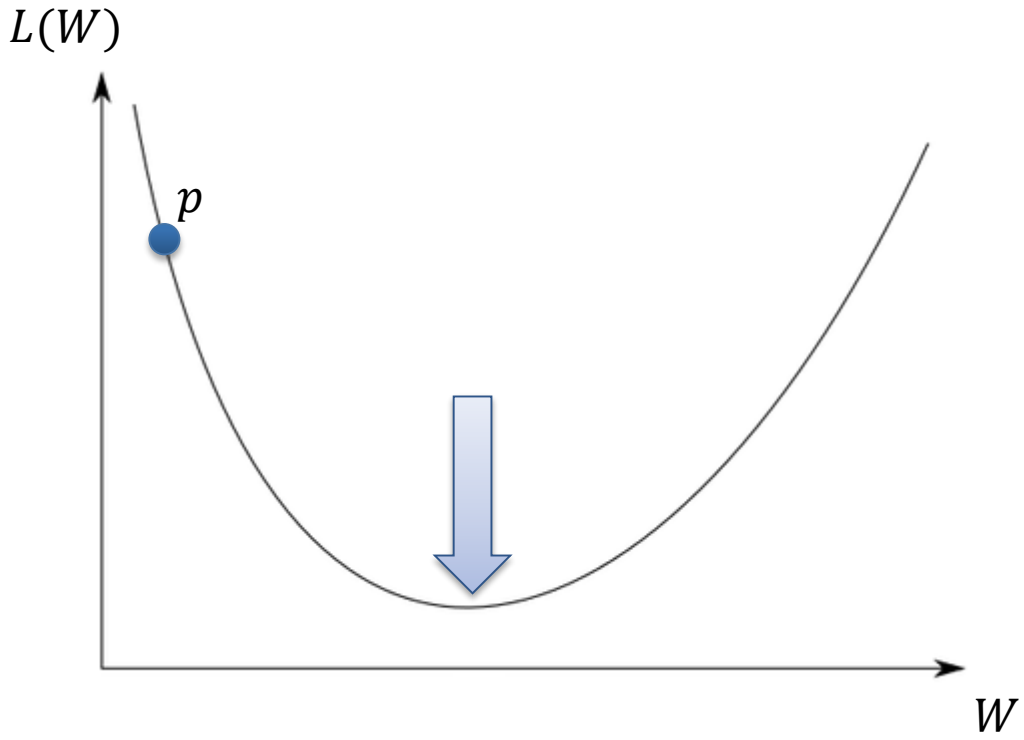
- Gute Weights W erzeugen ein Minimum in der Loss Function.
- Also ein Optimierungsproblem!

  W sei 1-Dim, L ist eine Funktion von W , den Daten x und den Hyperparametern λ ➔ Hiervon ist nur W eine Variable im Sinne der Optimierung



Gradientenabstieg

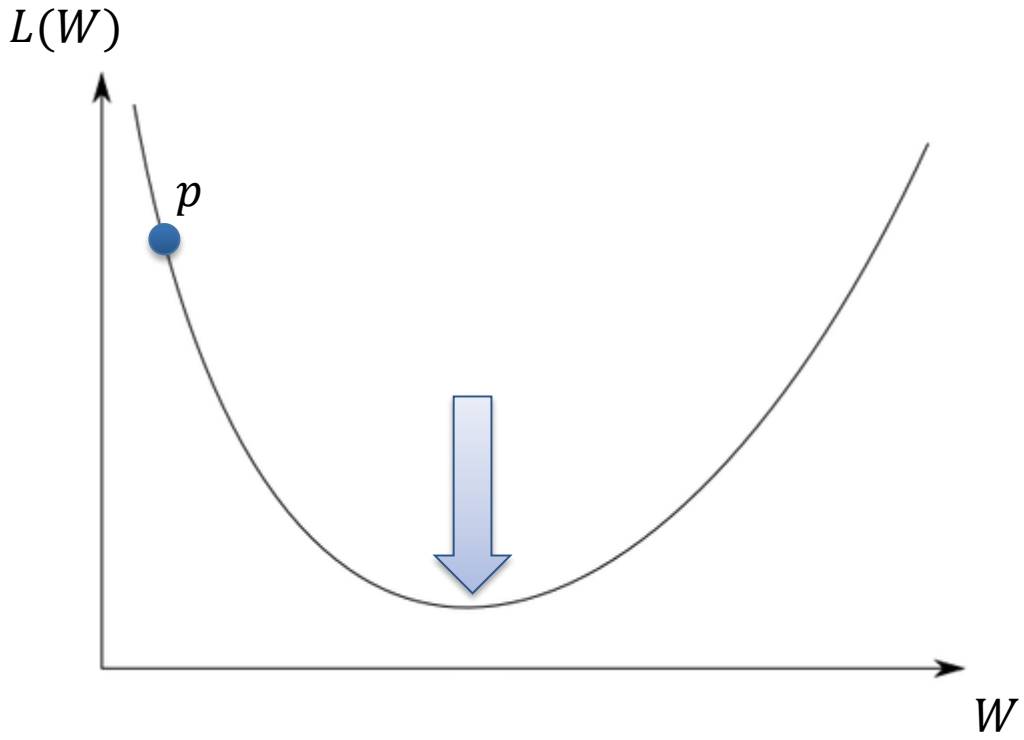
- Wir sind am Punkt p, wir wollen zum Minimum.
- In welche Richtung müssen wir?



Gradientenabstieg

- Wir sind am Punkt p, wir wollen zum Minimum.
- In welche Richtung müssen wir?
- Ableitung: Die Steigung der Funktion berechnen

  W in Richtung der negativen Steigung anpassen

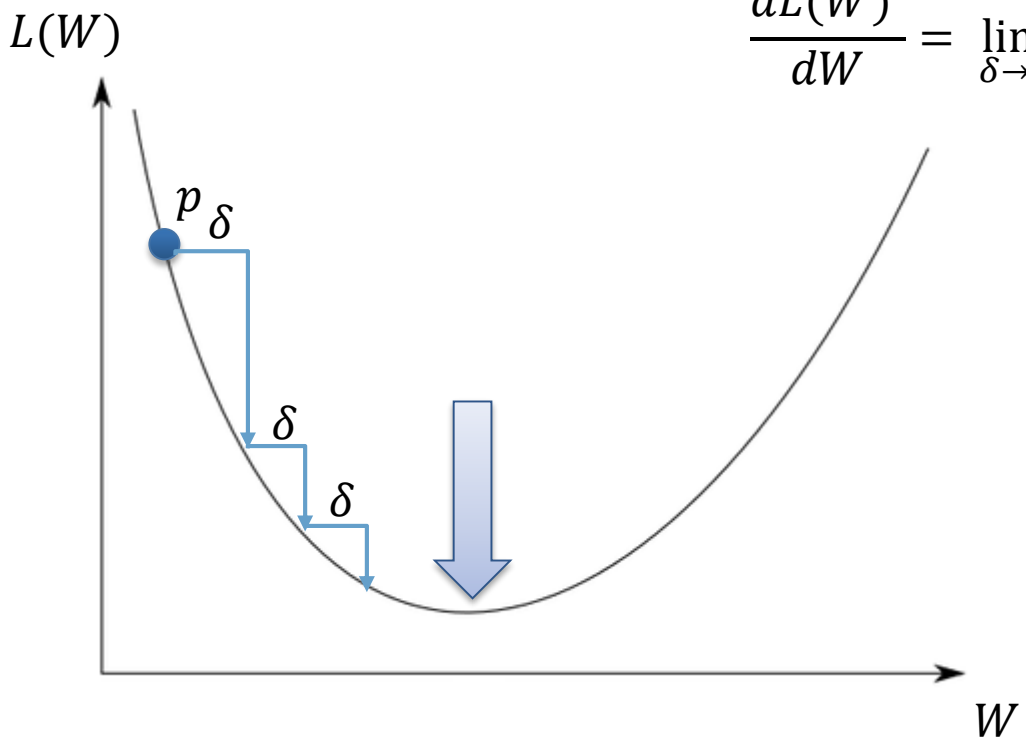


Gradientenabstieg

- Wir sind am Punkt p, wir wollen zum Minimum.
- In welche Richtung müssen wir?
- Ableitung: Die Steigung der Funktion berechnen

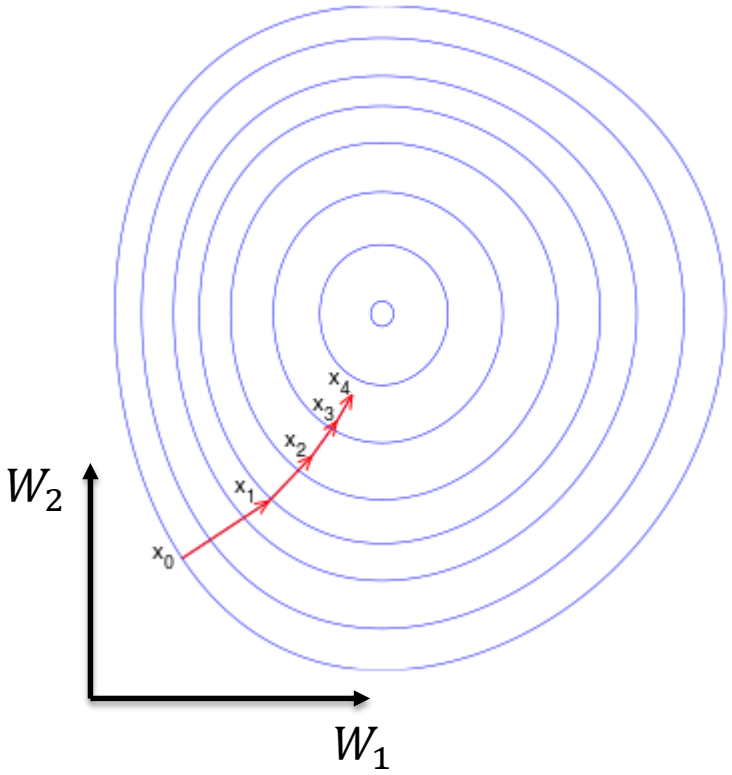
W in Richtung der negativen Steigung anpassen

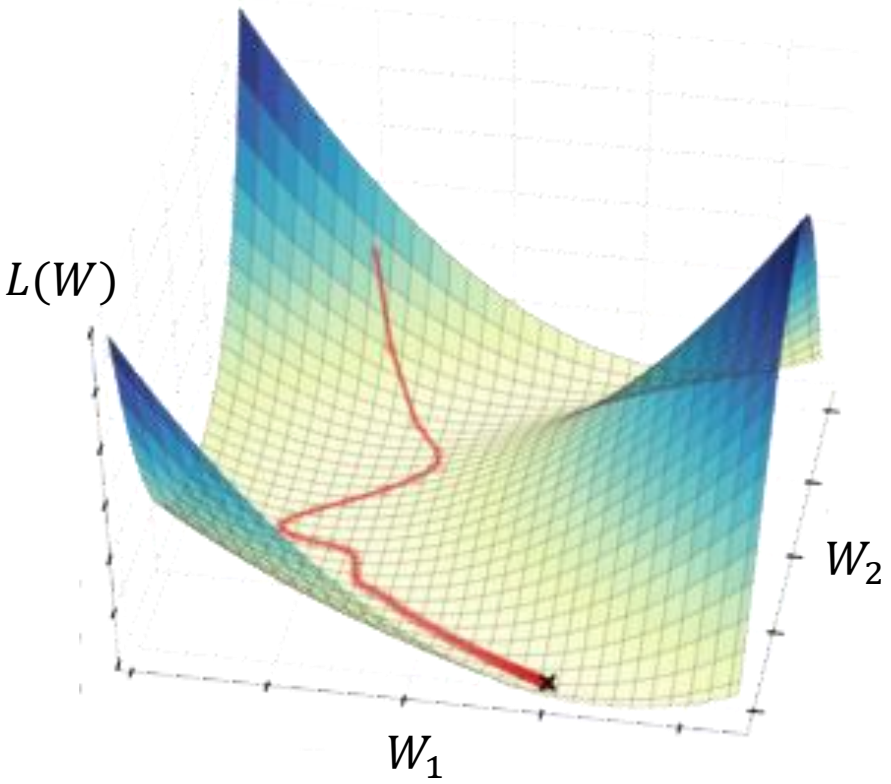
$$\frac{dL(W)}{dW} = \lim_{\delta \to 0} \frac{L(W + \delta) - L(W)}{\delta}$$



Gradientenabstieg

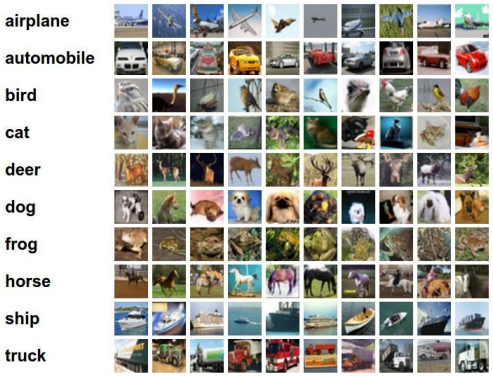
- In höheren Dimensionen funktioniert das ebenso





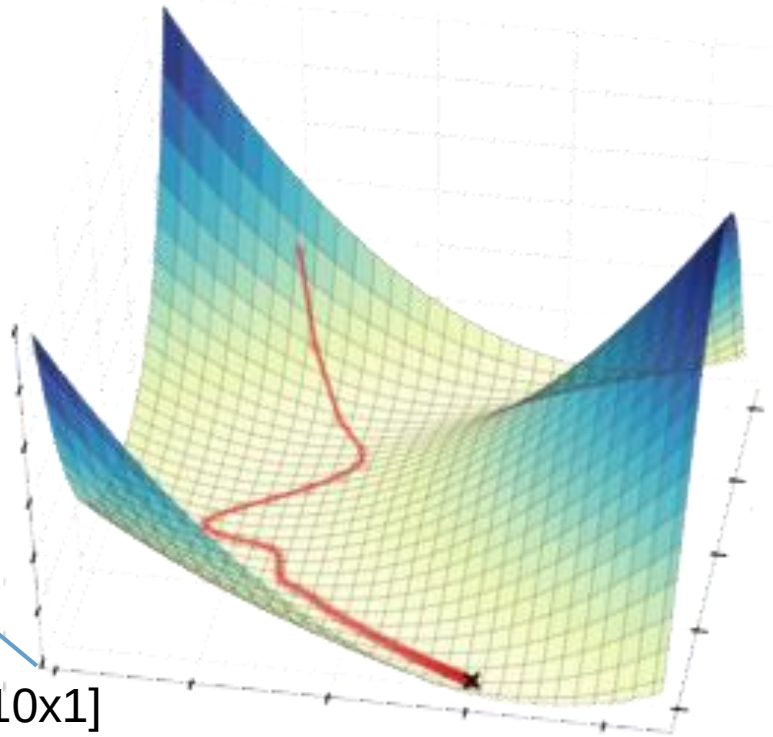
Gradientenabstieg

- In höheren Dimensionen funktioniert das ebenso
  - … wird aber schnell unübersichtlich



$$f(x, W, b) = Wx + b$$

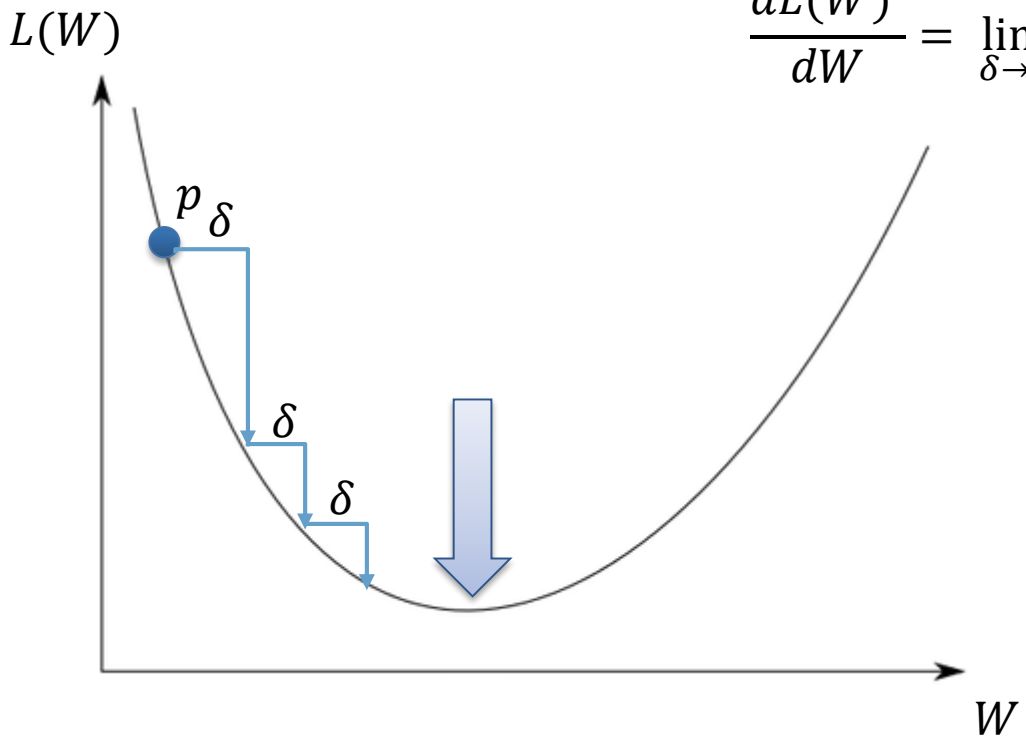
[10x1] [10x3072] [3072x1] [10x1]



Gradientenabstieg

- Es gibt zwei Möglichkeiten dies zu berechnen:
  - Numerisch
  - Analytisch

$$\frac{dL(W)}{dW} = \lim_{\delta \to 0} \frac{L(W + \delta) - L(W)}{\delta}$$



Gradientenabstieg

- Numerischer Gradientenabstieg:
- Für jeden Trainingsschritt
  - Wähle einen kleinen Wert δ
  - Modifiziere eine Komponente von W_{alt} mit δ
  - Berechne $L(W_{neu}) - L(W_{alt})$
  - Modifiziere W mit dem negativen Gradienten, so dass L im nächsten Trainingsschritt kleiner wird.

$$\frac{dL(W)}{dW} = \lim_{\delta \to 0} \frac{L(W + \delta) - L(W)}{\delta}$$

Gradientenabstieg

- Einfach zu programmieren
- Ist nur approximativ
- Langsam

$$\frac{dL(W)}{dW} = \lim_{\delta \to 0} \frac{L(W + \delta) - L(W)}{\delta}$$

```
W = np.arange(1,7).reshape(2,3)

Lvalue = L(W)                          # L berechen
gradient = np.zeros_like(W)
delta = 0.00001                        # Delta festlegen

w_iterator = np.nditer(W, flags=['multi_index'], op_flags=['readwrite'])
while not w_iterator.finished:
    index =  w_iterator.multi_index
    W_at_index = W[index]              # Den aktuellen Wert von W speichern
    W[index] = W_at_index + delta      # Delta anwenden
    newL = L(W)                        # Funktionswert von L(W+delta) bestimmen
    gradient[index] = (newL - Lvalue)/delta  # Komponente berechen

    W[index] = W_at_index              # Ursprünglichen Wert in W wiederherstellen
    w_iterator.iternext()              # In der nächsten Dimension weitermachen


```

Gradientenabstieg

- Tatsächlich ist die Loss Funktion nur eine Funktion von W

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

- Man Kann davon eine Ableitung analytisch bestimmen

Gradientenabstieg

- Tatsächlich ist die Loss Funktion nur eine Funktion von W

$$L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

- Man Kann davon eine Ableitung analytisch bestimmen
  - Sehr Schnell
  - Keine Approximation, echte Werte da $\lim_{\delta \to 0}$
  - Fehleranfällig, wenn man es selbst berechnen muss

Gradientenabstieg

- Numerisch:
  - Einfach zu programmieren
  - Ist nur approximativ
  - Langsam

- Analytisch:
  - Schnell
  - Keine Approximation
  - Fehleranfällig

Gradientenabstieg

- Numerisch:
  - Einfach zu programmieren
  - Ist nur approximativ
  - Langsam
- Analytisch:
  - Schnell
  - Keine Approximation
  - Fehleranfällig

- Praktisch:
- Immer den analytische Ansatz benutzen, aber mit dem numerischen Ansatz überprüfen ➔ Gradient Check
- Caffe und andere Tool Kits haben uns das für ihre Layer bereits abgenommen!

Lernraten

- Wir haben den Gradienten unserer Gewichte

- Jetzt die Gewichte updaten!

```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += grad_w
```

Lernraten

- Wir haben den Gradienten unserer Gewichte

- Jetzt die Gewichte updaten!

```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += grad_W
```

Lernraten

- Wir haben den Gradienten unserer Gewichte

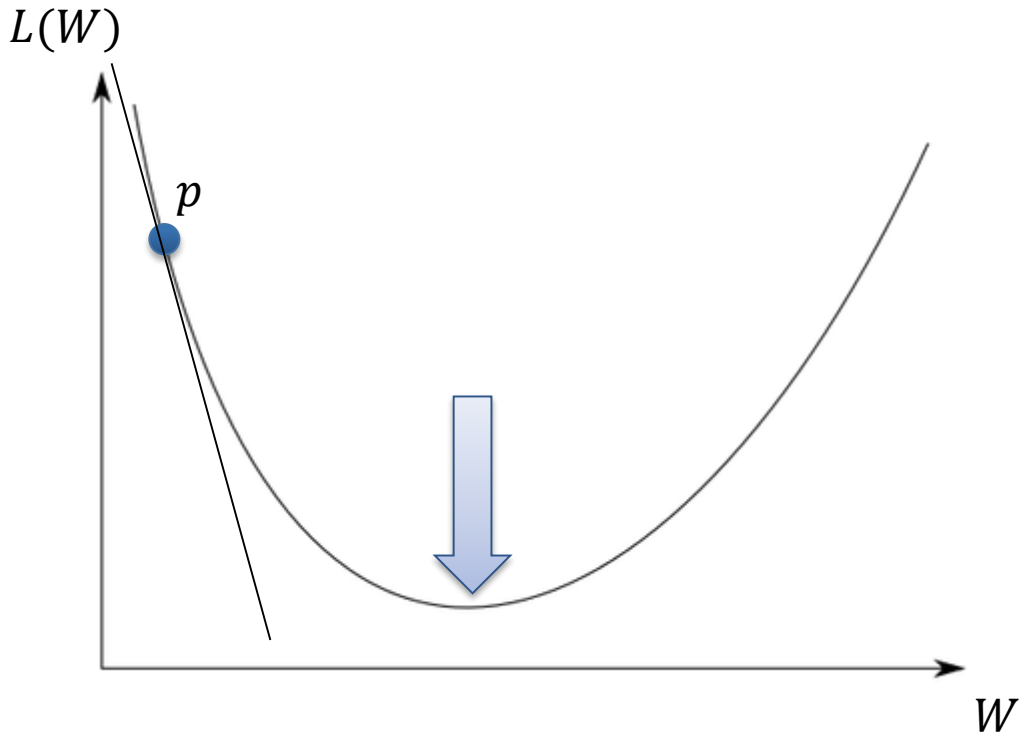
- Jetzt die Gewichte updaten!

```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += -grad_w
```

- Der Gradientenabstieg sagt uns in welche “Richtung” wir laufen müssen
  Aber wie weit?

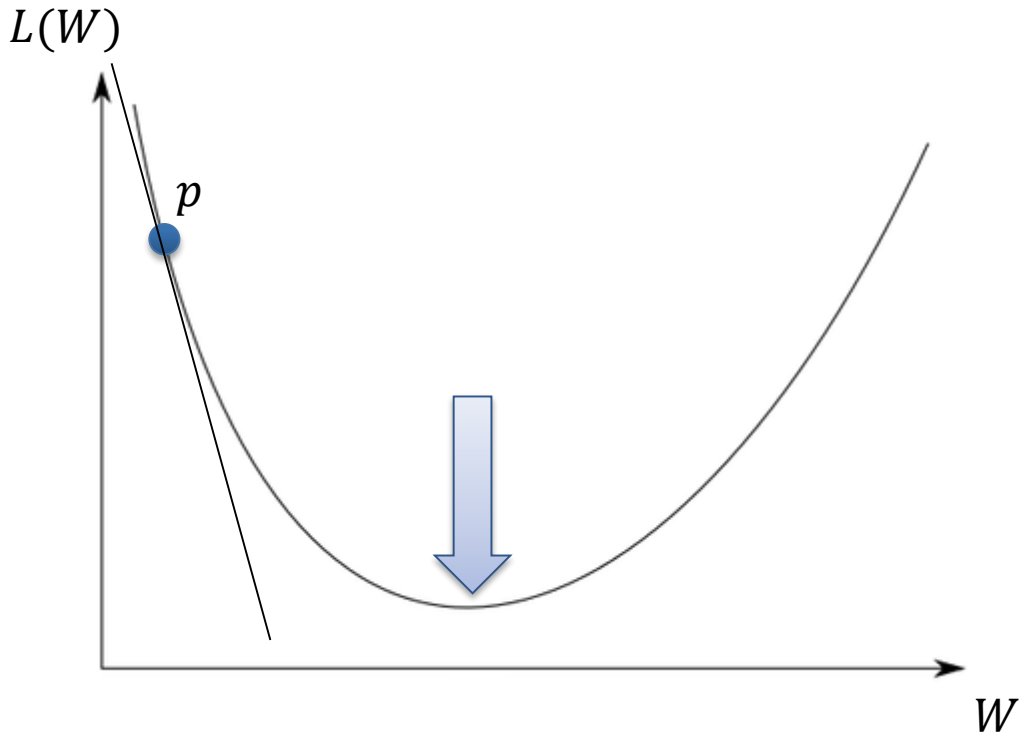
Lernraten

- Wir berechen einen Gradienten an p
  - Die Steigung ist stark negativ!



Lernraten

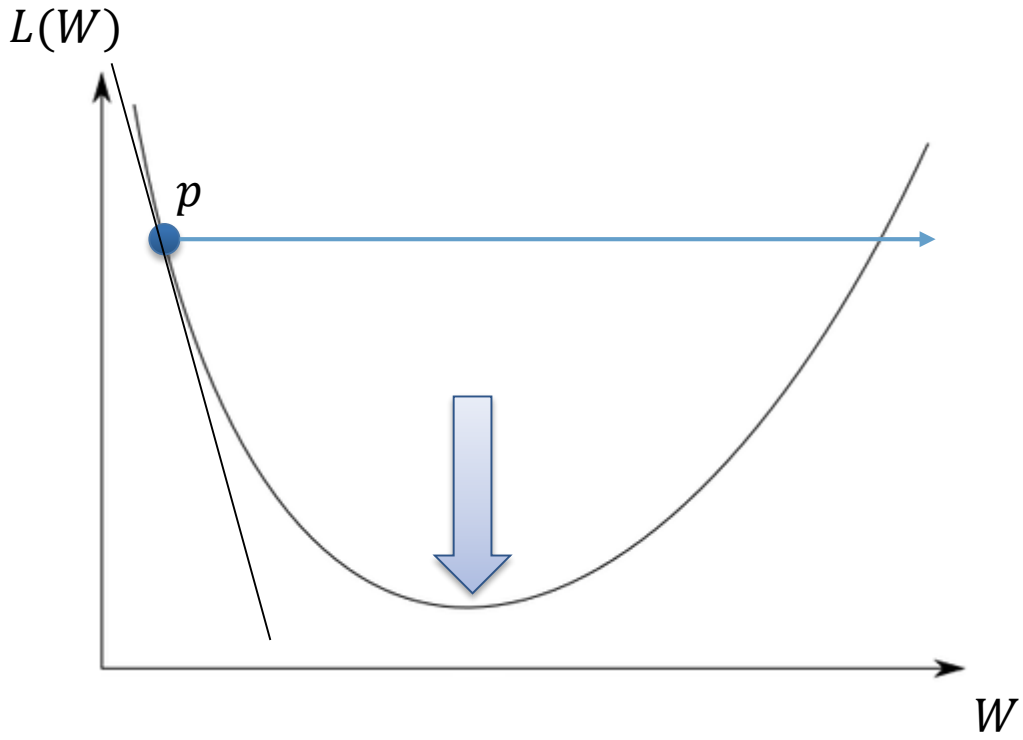
- Wir möchten die Gewichte in Richtung des negativen Gradienten anpassen



Lernraten

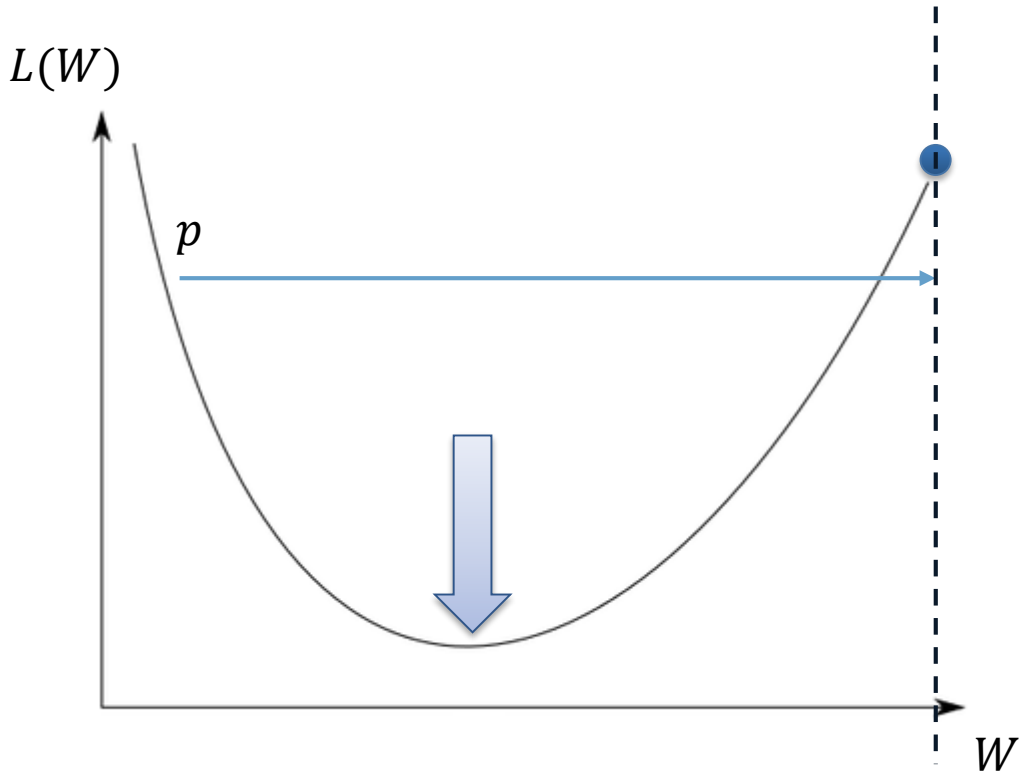
- Der Gradientenabstieg sagt uns in welche “Richtung” wir laufen müssen
  Aber wie weit?

```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += -grad_w
```



Lernraten

- Wir sind weit in Richtung des negativen Gradienten gegangen
  - Der Loss Wert steigt!



Lernraten

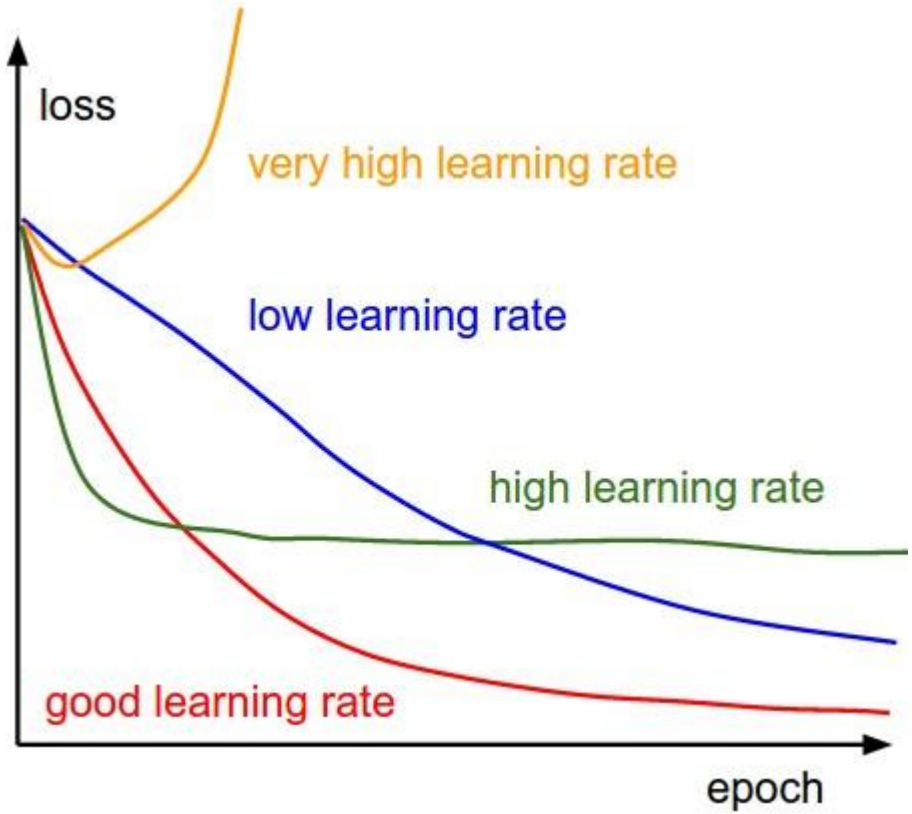
- Der Gradientenabstieg sagt uns in welche Richtung wir laufen müssen
  - Aber wie weit?
- Wir führen einen Skalierungsfaktor ein
- Die Learning Rate

```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += - learning_rate * grad_W
```

Lernraten

- Der Gradientenabstieg sagt uns in welche Richtung wir laufen müssen
  - Aber wie weit?
- Wir führen einen Skalierungsfaktor ein
- Die Learning Rate

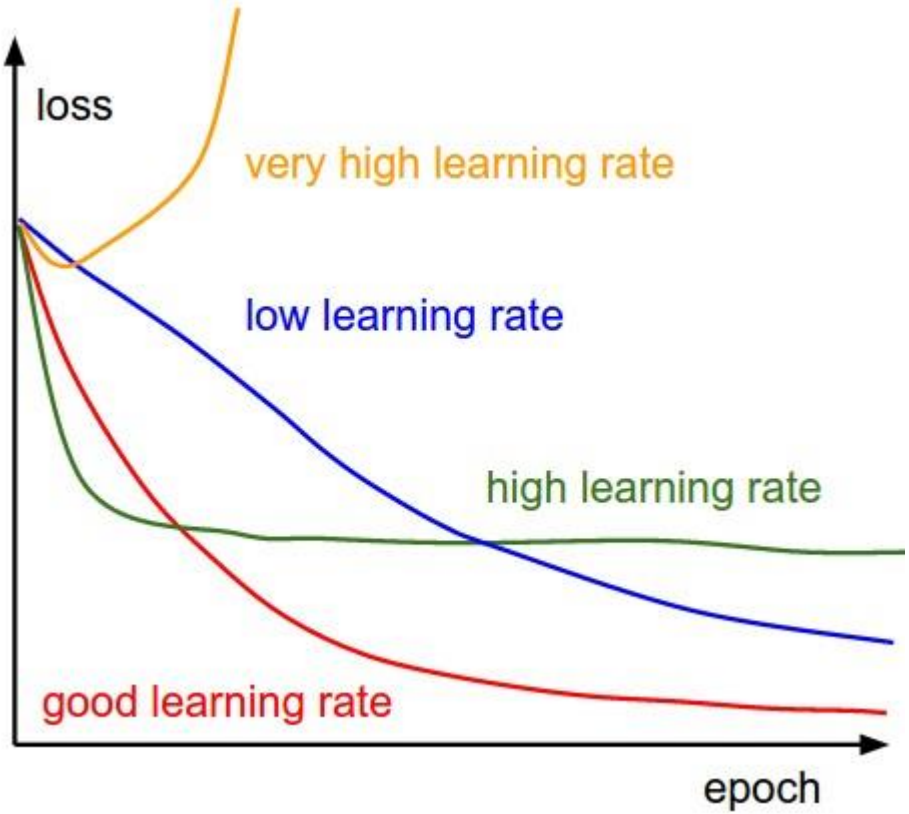
```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += - learning_rate * grad_W
```



Lernraten

- Die Learning Rate ist einer der wichtigsten Hyperparameter
- Es gibt sehr schlaue Verfahren die Lerning anzupassen
  - Dazu mehr in späteren Vorlesungen!

```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += - learning_rate * grad_W
```



Ausblick

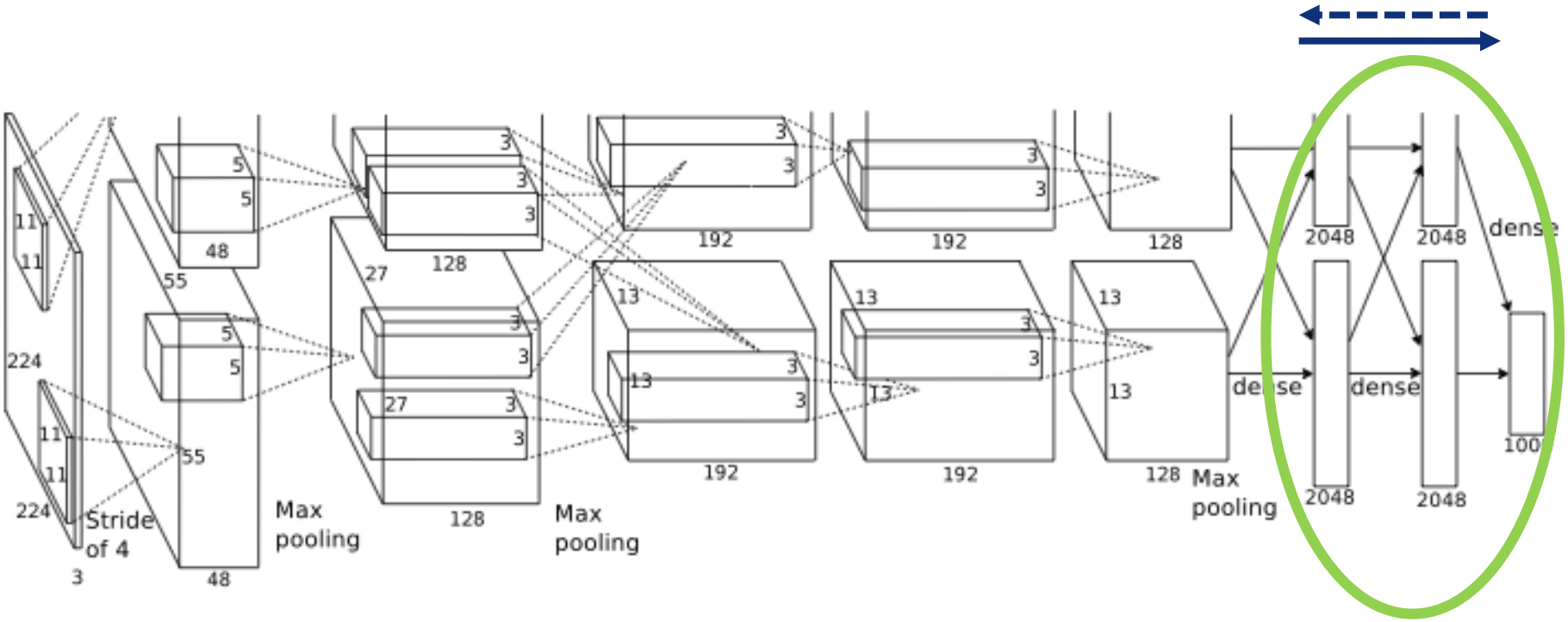
- Loss functions
  - Ist *W* gut?

- Optimierung
  - Wie wird *W* besser?

- Backpropagation
  - Wie verbessere ich W über mehrere Schichten hinweg?
- CNN
  - Gibt es bessere Score Functions $f(x, W)$ für Bilder
- DNN
  - Mehr Layer!

Ausblick

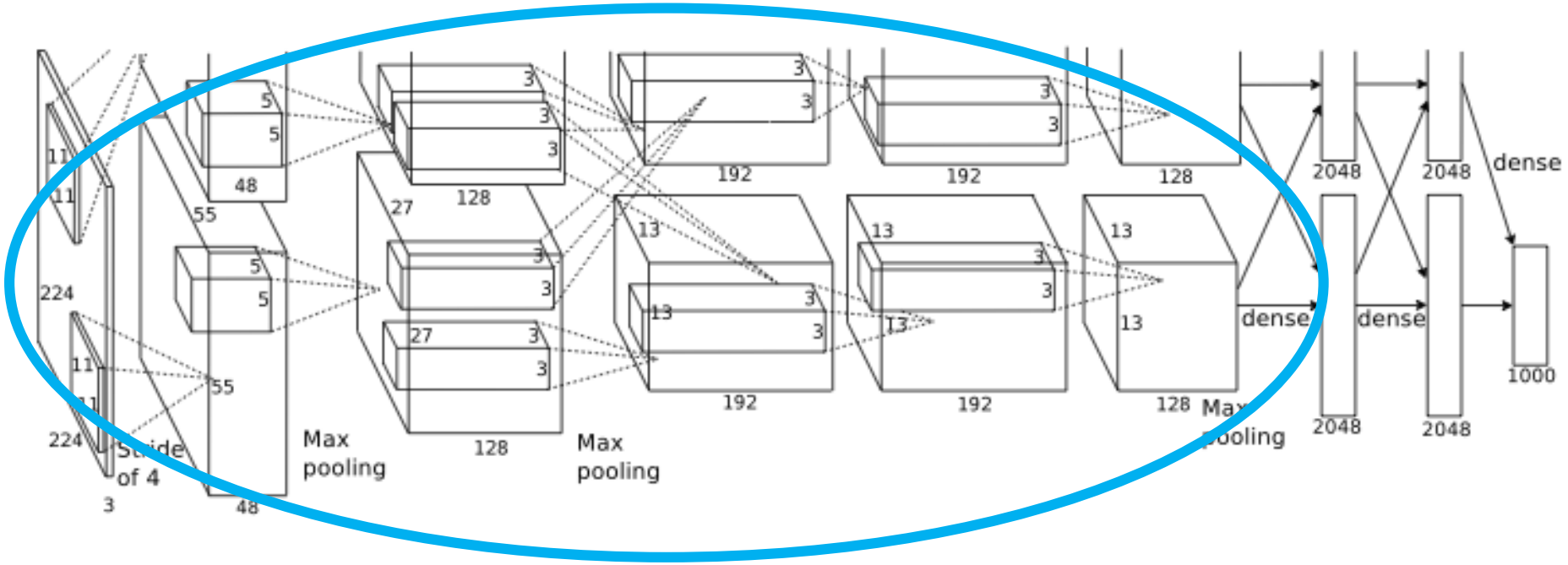
- Fully Connected Layer (Forward Path)
- Grobe Idee wie der Lernvorgang abläuft
  - Nächste Vorlesung: Backpropagation und Neuronale Netze



[convnet from Krizhevsky et al.'s NIPS 2012 ImageNet classification paper]

Ausblick

- Convolutional Layer (Faltungsschichten)
  - Die Funktion $f(x, W)$ ändert sich, den Rest beherrschen sie bereits!
- Pooling Layer
  - Sind sehr einfach!



[convnet from Krizhevsky et al.'s NIPS 2012 ImageNet classification paper]

hochschule mannheim

Extras

Hinge Loss / Multiclass SVM Loss

- Fragen:

  Wenn ein W gefunden wird für das $L = 0$ gilt, ist diese Lösung einzigartig?

  Nein! Wenn W einen loss $L = 0$ erzeugt dann tut das auch $2 * W$ und beliebig viele andere Lösungen.

  In der Praxis wird hier eine Regularisierung durchgeführt.

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

[Andrej Karpathy, Stanford Universitiy]

Hinge Loss / Multiclass SVM Loss

- Regularisierung:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

λ ➔ Hyperparameter

[Andrej Karpathy, Stanford Universitiy]

Hinge Loss / Multiclass SVM Loss

- Regularisierung:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

- L2 Regularisierung:

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

Von Lösungen mit gleichem SVM-Loss wird die mit dem W bevorzugt, deren L2 Distanz zu 0 am kleinsten ist.

[Andrej Karpathy, Stanford Universitiy]

Hinge Loss / Multiclass SVM Loss

- Regularisierung:

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$

[Andrej Karpathy, Stanford Universitiy]

Hinge Loss / Multiclass SVM Loss

- Regularisierung:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$$

Data Loss
Abhängig von *W*

Regularisierungs Loss
Abhängig von *W*

[Andrej Karpathy, Stanford Universitiy]