hochschule mannheim

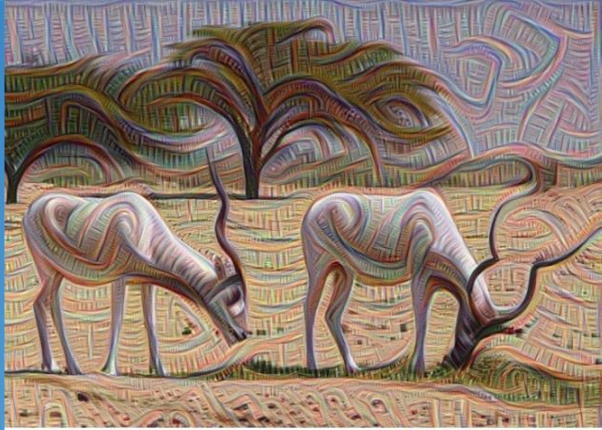


Image: Google Research Blog

Informationstechnik
Backpropagation

Prof. Dr. Marcus Vetter,
Benjamin Kraus, Kevin Höfle

Inhalt

1. Wiederholung
- der Loss-Funktion
- Gradientenabstieg
- Optimierung /

2. Backpropagation

3. Beispiel eines einfachen Neuronalen Netztes

Was haben wir bisher

1. Score Function:
   - Abbildung von Features auf Zielgröße

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

2. Loss Function:
   - Definiert was “gut” und “schlecht”

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

Softmax, Cross-entropy

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

erzeugt eine Art Wahrscheinlichkeit

SVM-Loss, hinge loss

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

fordert einen Mindestabstand der Klassen

Optimierung

Zur Optimierung von W benötigen wir die Ableitung von L nach W

Hier ist x eine Konstante

Wir suchen: $\nabla_w L$

L2-Regularisierung

$$\nabla_w L = \nabla_w \left(\frac{1}{N} \sum_{i=1}^{N} L_i(x, W) + \lambda \sum_k W_k^2 \right)$$

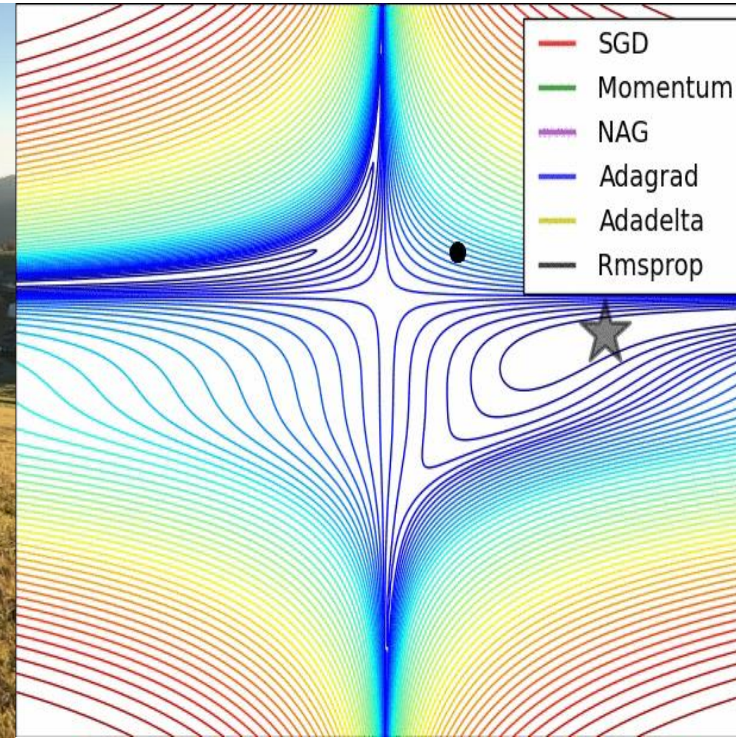
Wir suchen also den Gradienten von L nach W

Beispiel:

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

$$\nabla_w L = \frac{\delta L(s(x, W))}{\delta w} = \begin{cases} -sx \; if \; sxW < 1 \\ 0 \end{cases}$$

Optimierung

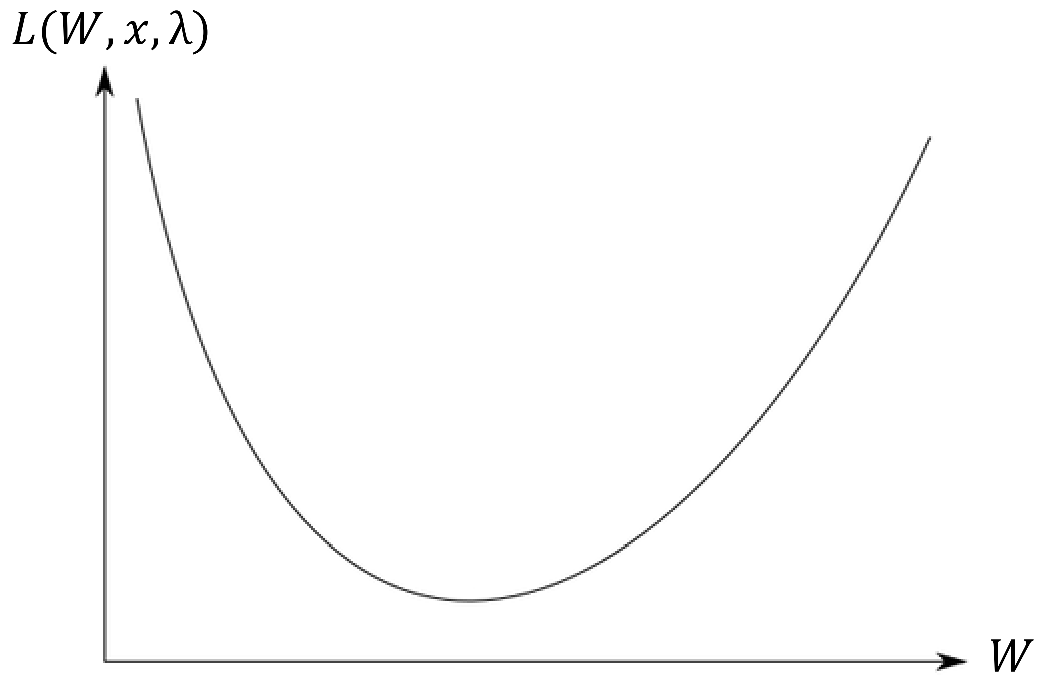



```
while True:
        gewichtsGradient = gradient(lossFun, data, weights)
        weights -= learning_rate * gewichtsGradient
```

Gradientenabstieg

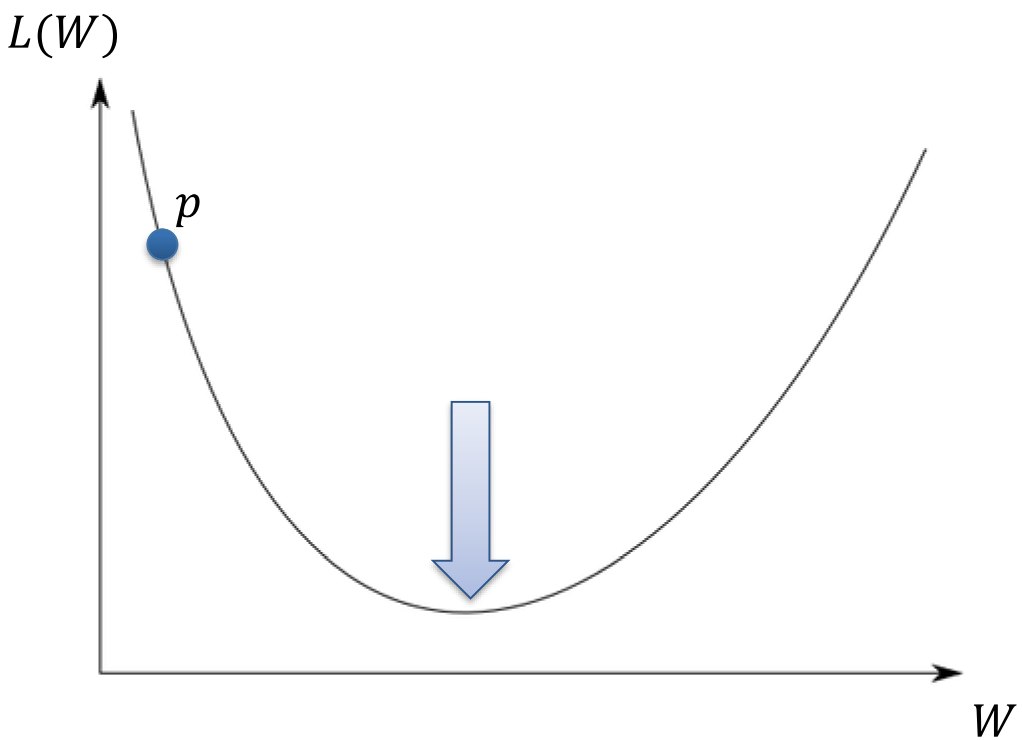
1. Gute Weights W erzeugen ein Minimum in der Loss Function.
2. Also ein Optimierungsproblem!

- Hiervon ist nur W eine Variable im Sinne der Optimierung
- W sei 1-Dim,
- L ist eine Funktion von W
  - x sind die Daten
  - λ sind Hyperparameter



Gradientenabstieg

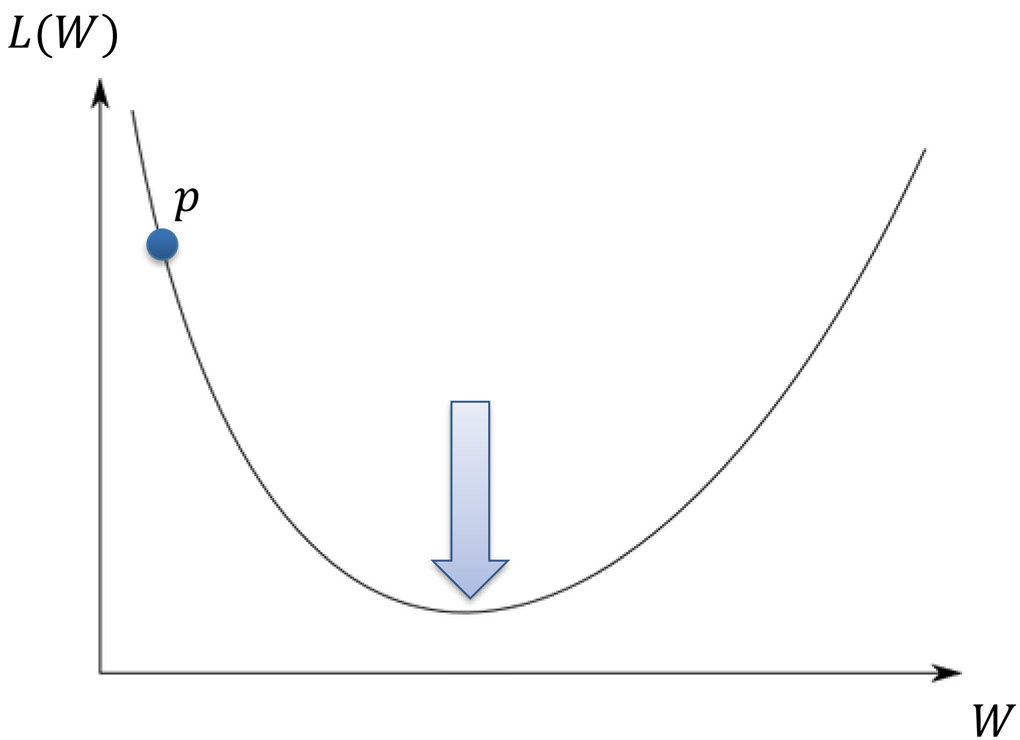
1. Wir sind am Punkt p, wir wollen zum Minimum.
2. In welche Richtung müssen wir?



Gradientenabstieg

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

W in Richtung der negativen Steigung anpassen

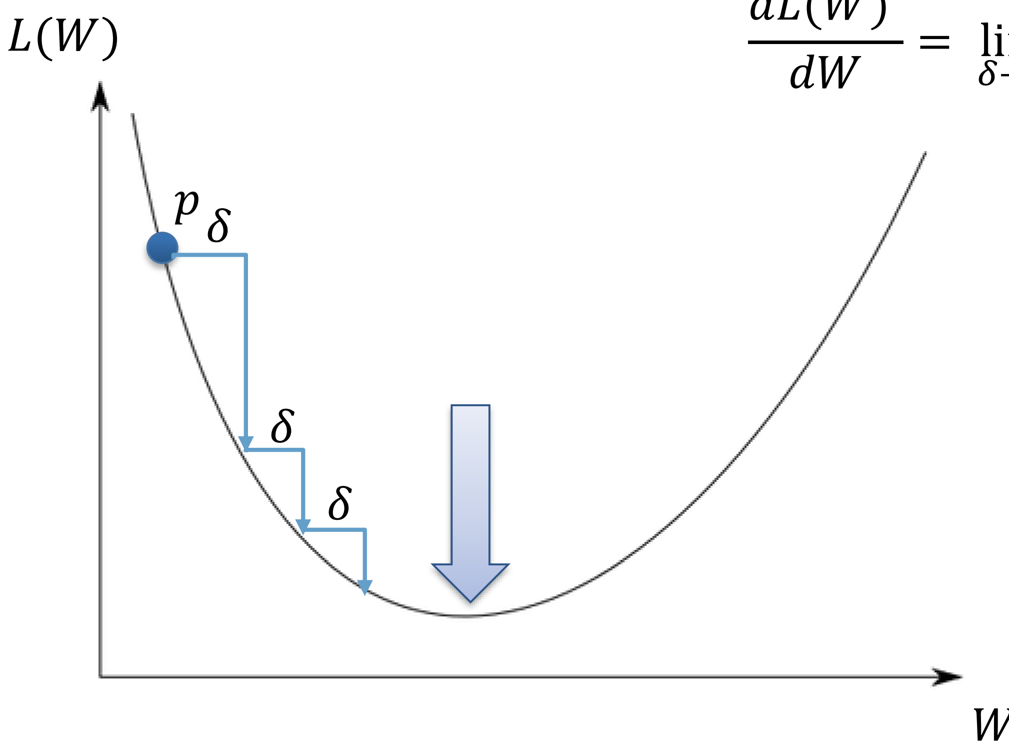


Gradientenabstieg

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

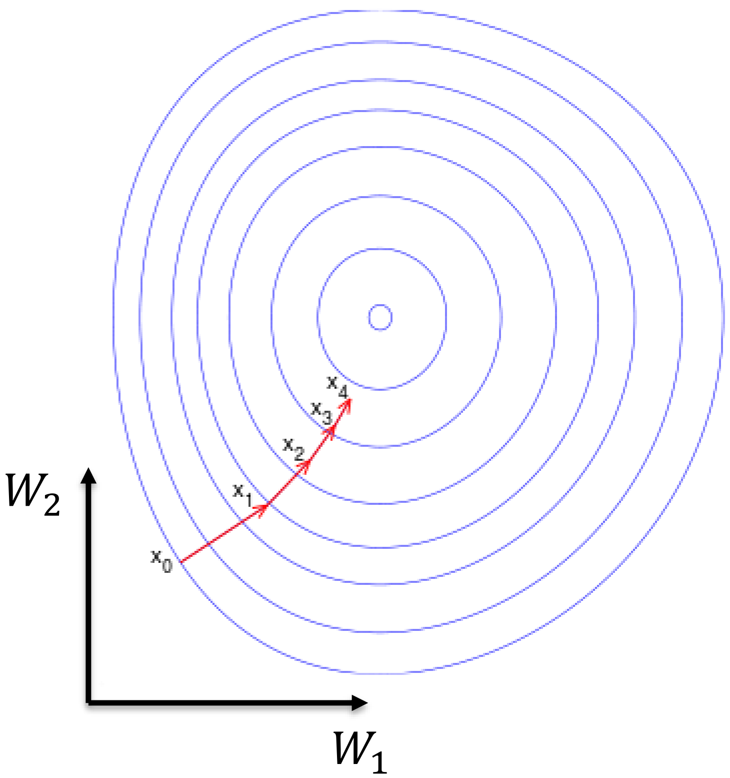
W in Richtung der negativen Steigung anpassen

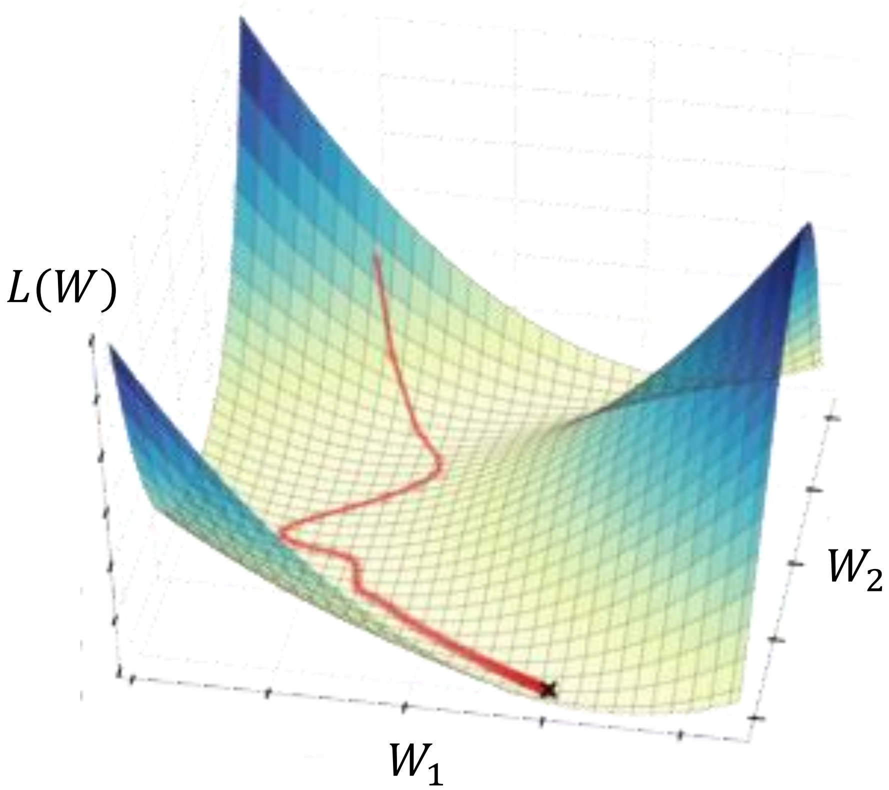
$$\frac{dL(W)}{dW} = \lim_{\delta \to 0} \frac{L(W + \delta) - L(W)}{\delta}$$



Gradientenabstieg

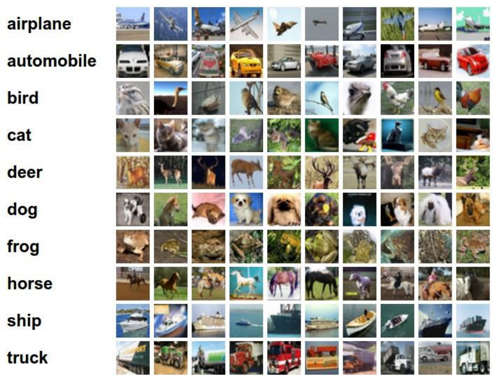
1. In höheren Dimensionen funktioniert das ebenso





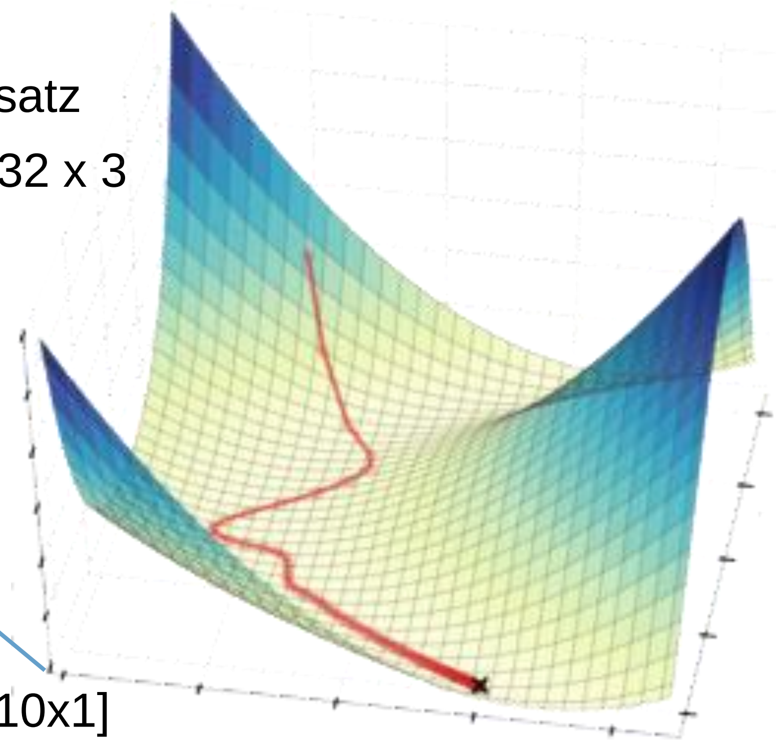
Gradientenabstieg

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



CIFAR-10 Datensatz

Dimension: 32 x 32 x 3



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

[10x1]

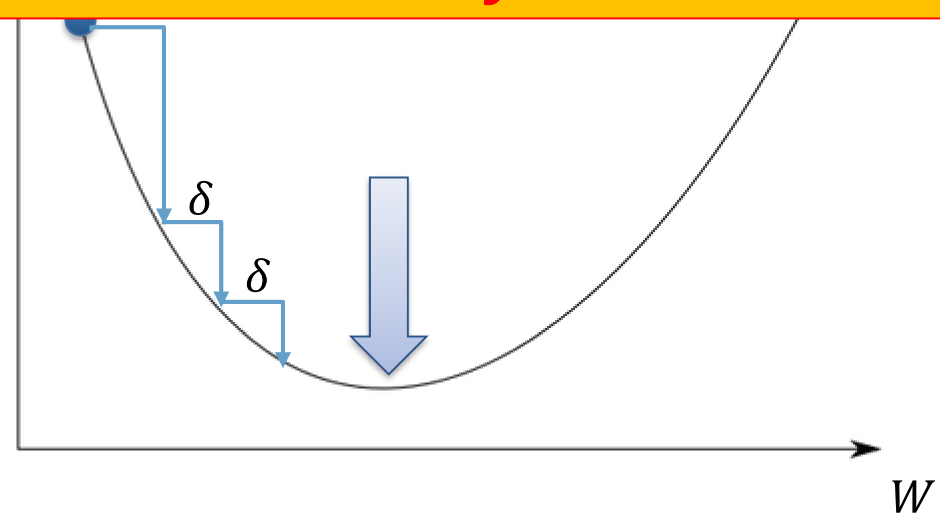
[10x3072] [3072x1] [10x1]

→ 30730 Dimensionen zu optimieren

Gradientenabstieg

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

Was sind Vor- und Nachteile der Numerischen bzw. Analytischen Berechnung?



Gradientenabstieg

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Numerische Lösung ist:

langsam ☹

Näherung ☹

aber leicht zu programmieren ☺

Analytische Lösung:

schnell ☺

exakt ☺

fehleranfällig ☹

In der Praxis:

analytische Lösung,
Überprüfung der Implementierung
mit numerischer Steigung

hochschule mannheim

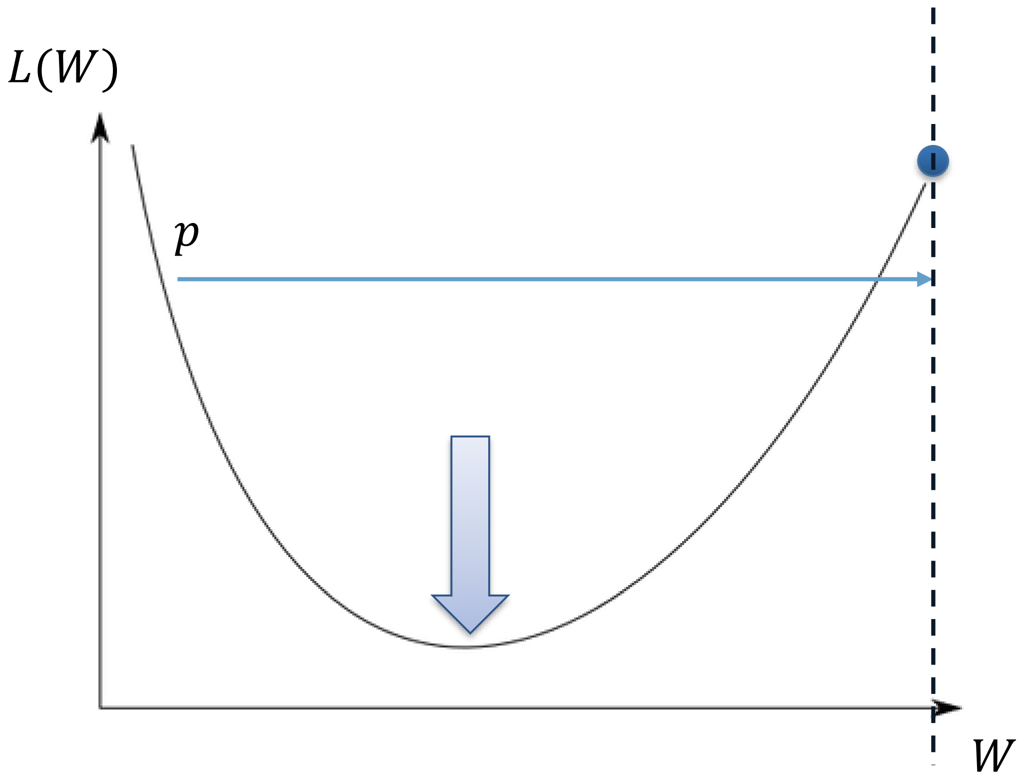
Wiederholung der Lernraten

1. Der Gradientenabstieg sagt uns in welche Richtung wir laufen müssen
2. Wir führen einen Skalierungsfaktor ein: Die Learning Rate

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

Wiederholung der Lernraten

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



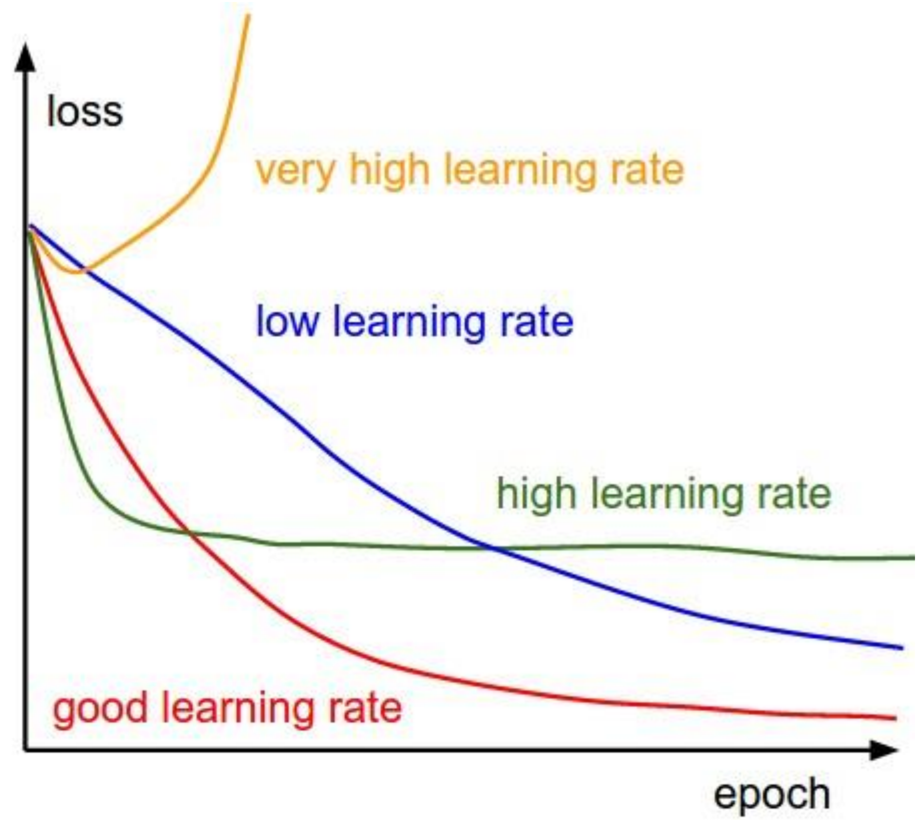
- Konvergiert nicht!
- Der Loss oszilliert

Wiederholung der Lernraten

Wir führen einen Skalierungsfaktor ein:

Die Learning Rate

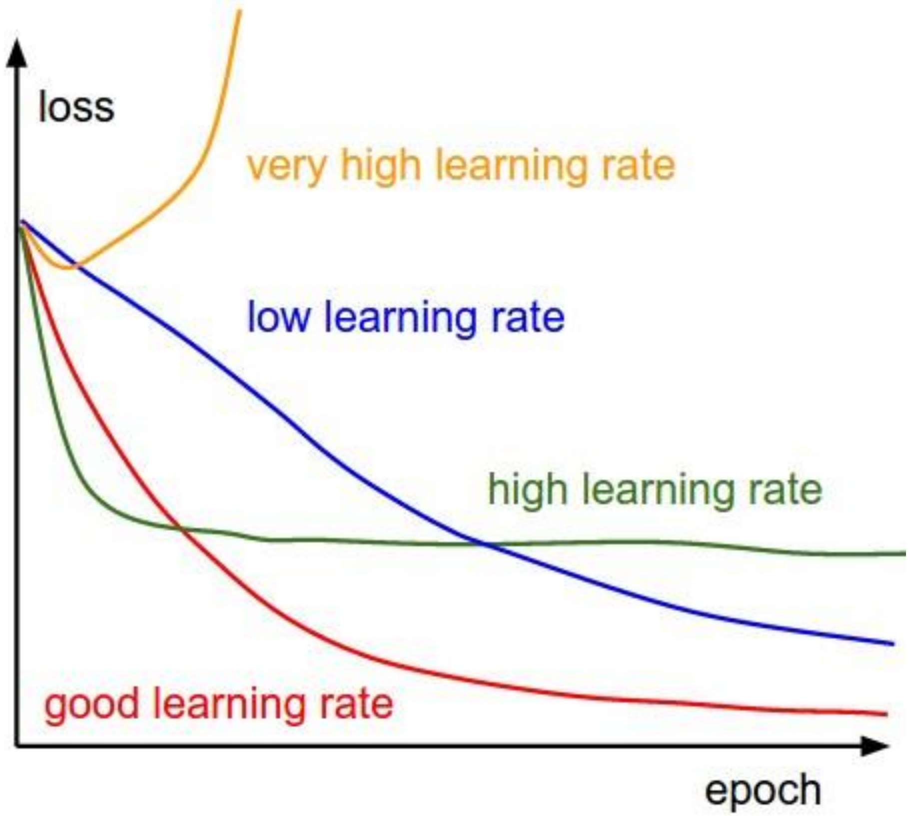
```
while True:
    grad_W =  eval_gradient(X, y, W)
    W += - learning_rate * grad_W
```



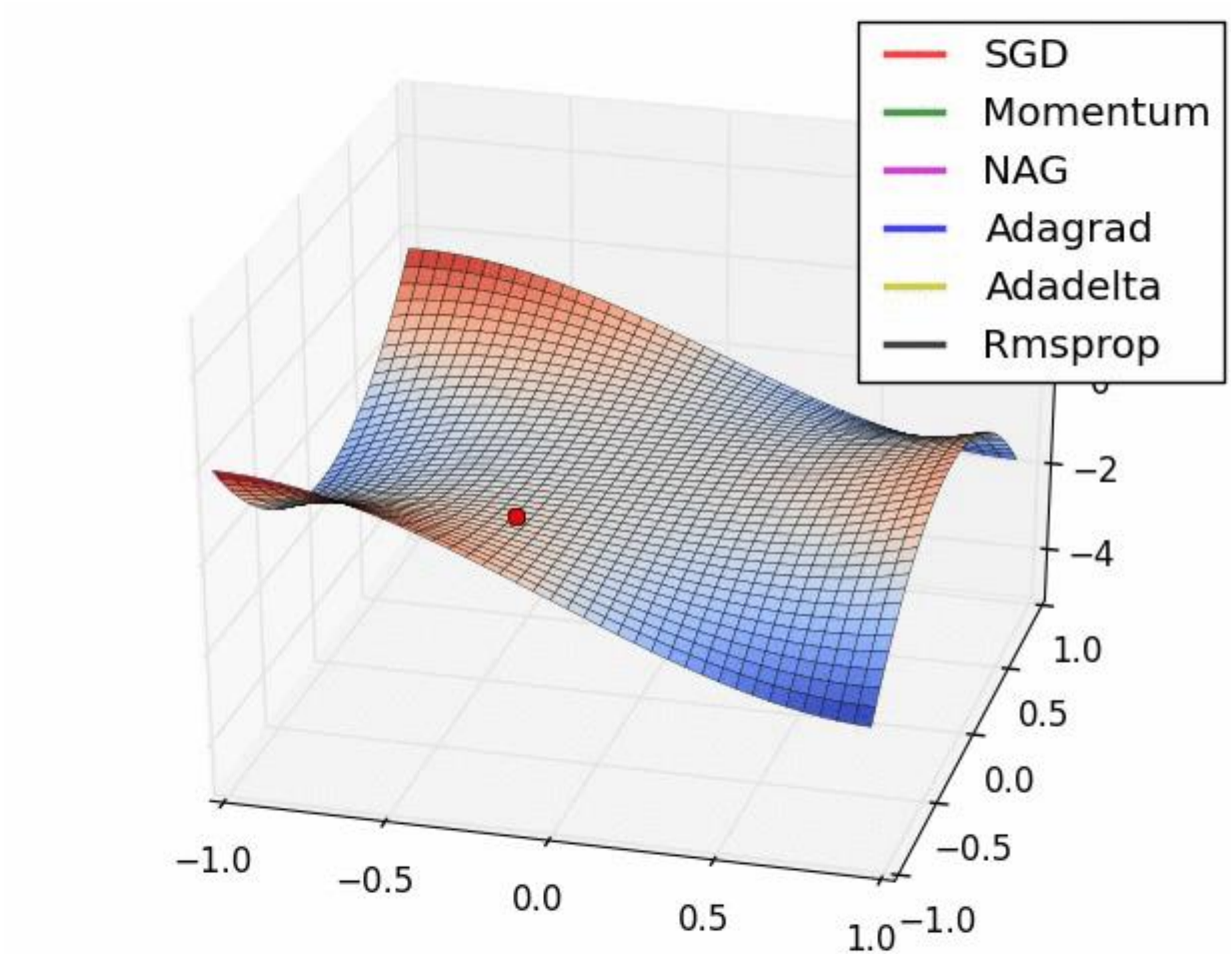
Lernraten

- 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
```



Gradienten Abstieg



[Graphs: Alec radford]

Backpropagation

1. Wie kann der Gradient über mehrere Layer analytisch bestimmt werden?

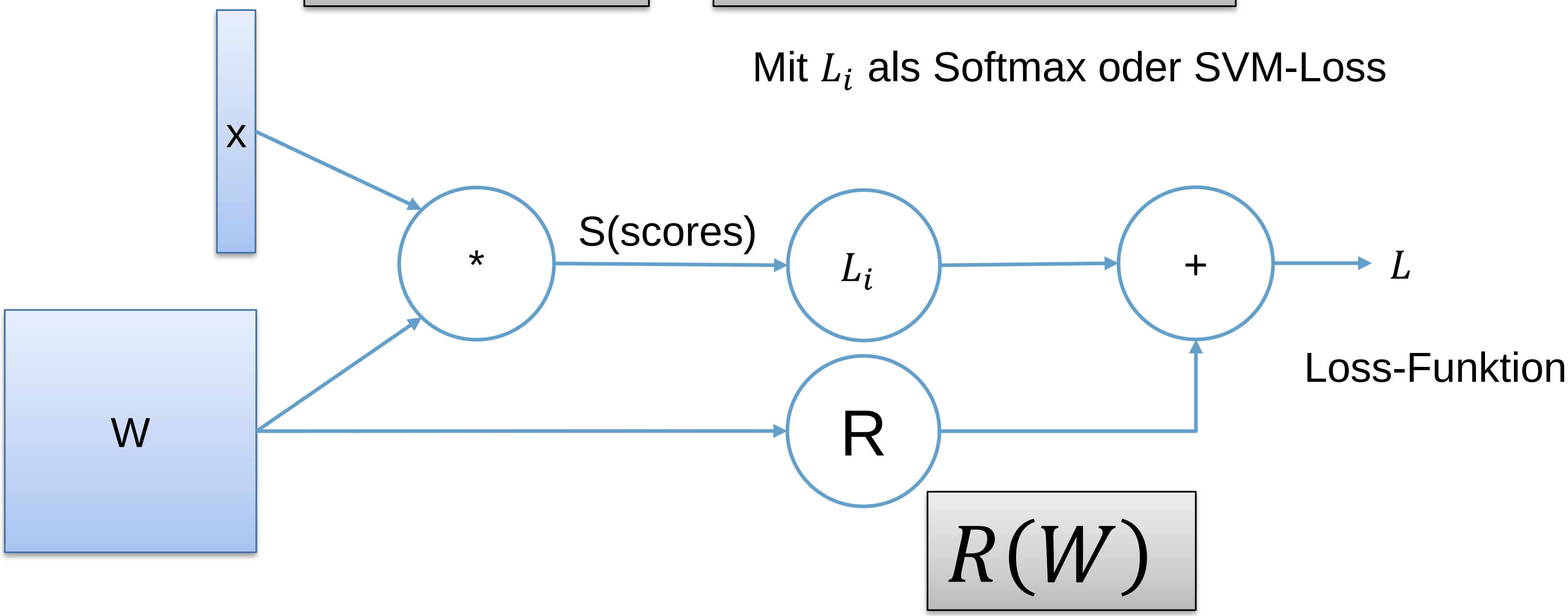
→überlegen wir uns dies Zunächst mal für ein Layer

Berechnungsgraph

$$f = Wx$$

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \lambda \sum_{k} W_k^2$$

Mit L_i als Softmax oder SVM-Loss



Convolutional Network (AlexNet)

Bild

Convolutional Network (AlexNet)

Gewichte

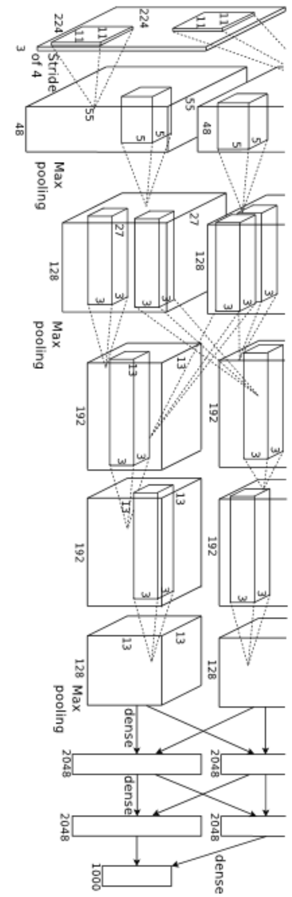
Gewichte

Gewichte

Gewichte

Gewichte

Gewichte

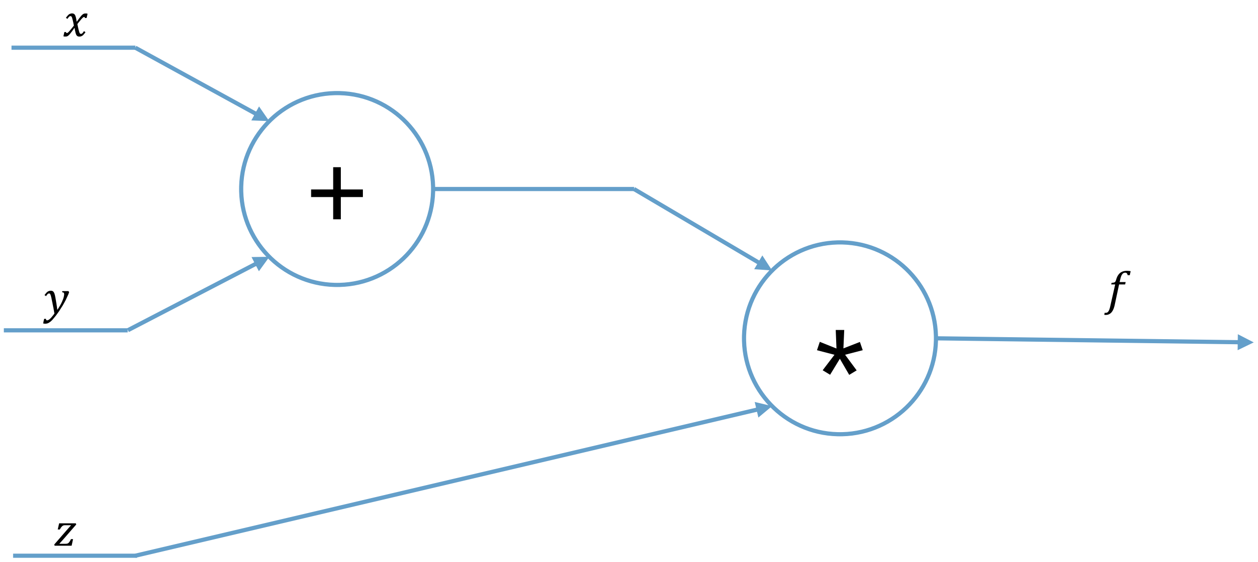


Backpropagation

loss

Backpropagation Beispiel einer Funktion

$$f(x, y, z) = (x + y)z$$



Forward-Pass

Backpropagation Beispiel

Forward-Pass   Backward-Pass

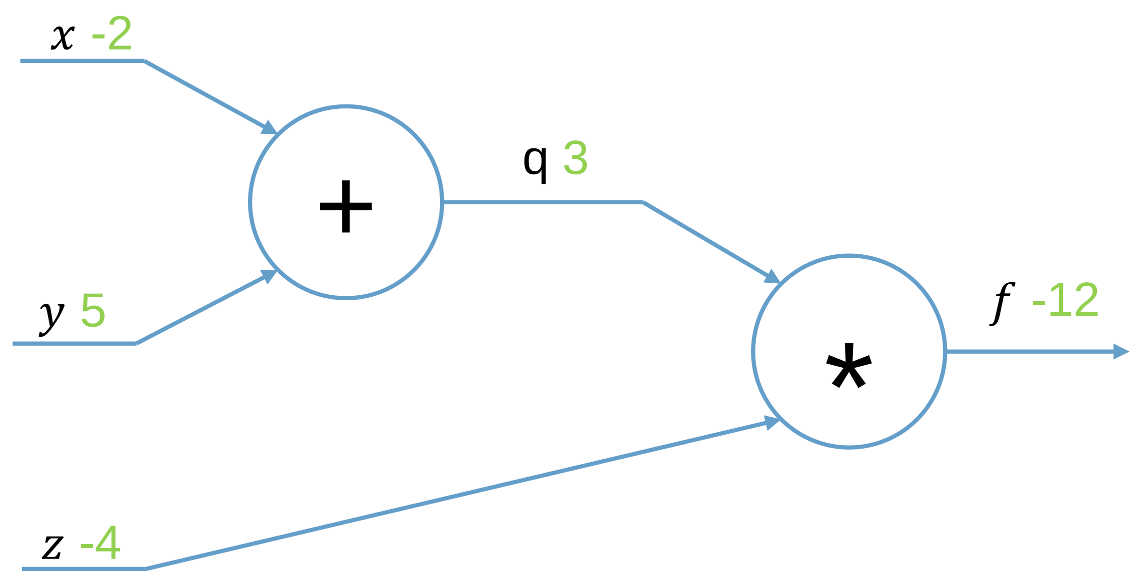
$$f(x,y,z) = (x+y)z \qquad Beispiel: x = -2, y = 5, z = -4$$

$$\mathrm{f} = qz;\ \frac{\partial f}{\partial q} = z;\ \frac{\partial f}{\partial z} = q$$

$$\mathrm{q} = x + y;\ \frac{\partial q}{\partial x} = 1;\ \frac{\partial q}{\partial y} = 1$$

Wir suchen:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



Backpropagation Beispiel

Backward-Pass

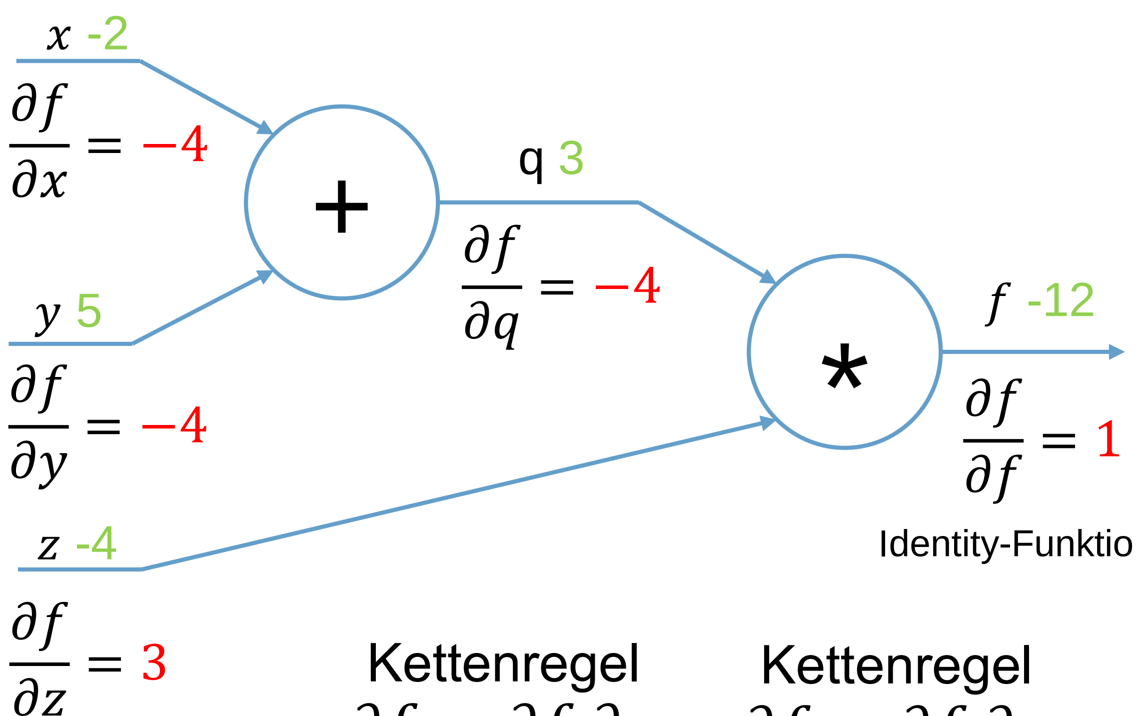
$$f(x, y, z) = (x + y)z \qquad Beispiel: x = -2, y = 5, z = -4$$

$$\text{f} = qz; \; \frac{\partial f}{\partial q} = z; \; \frac{\partial f}{\partial z} = q$$

$$\text{q} = x + y; \; \frac{\partial q}{\partial x} = 1; \; \frac{\partial q}{\partial y} = 1$$

Wir suchen:

$$\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$$



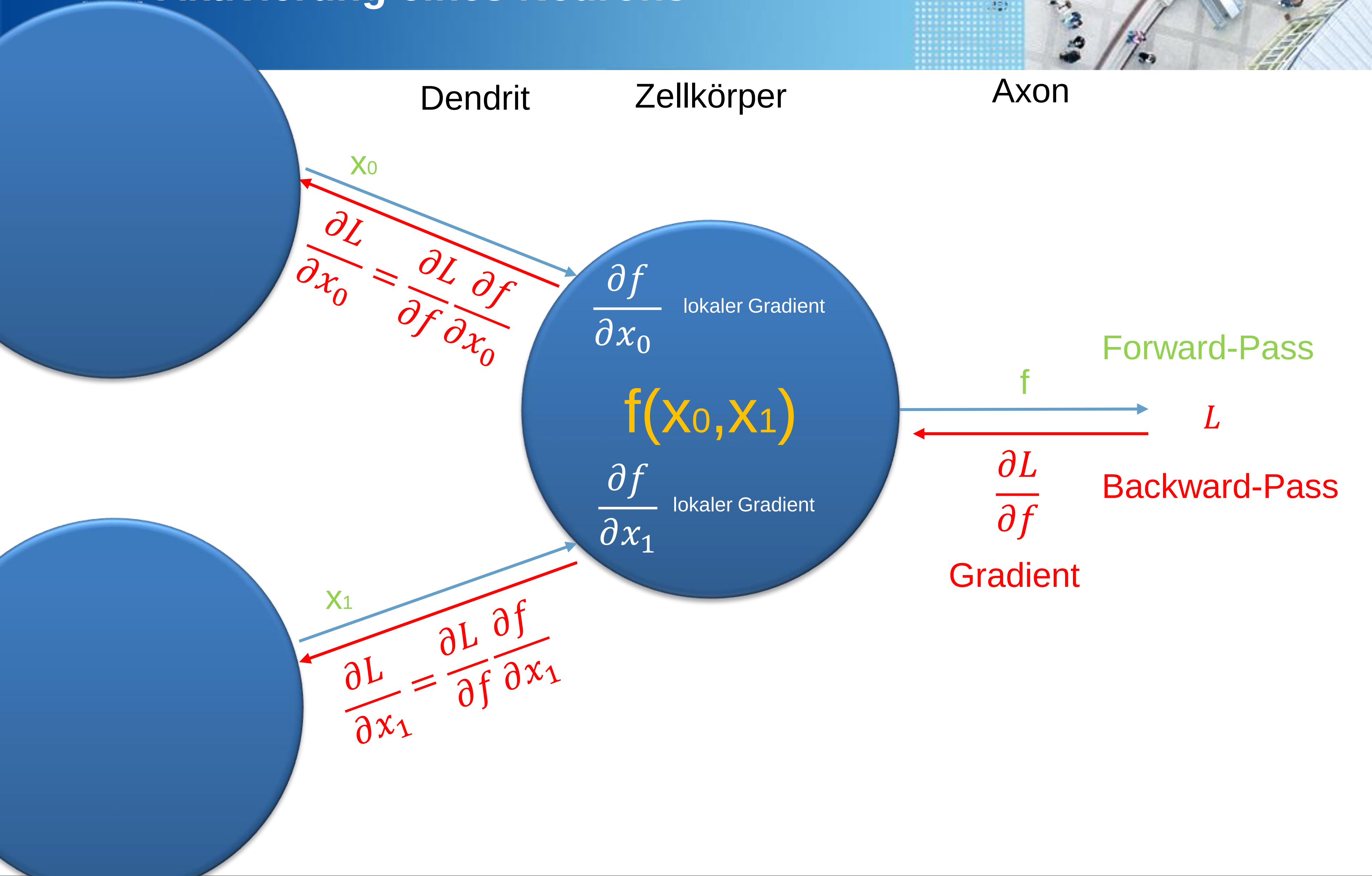
Kettenregel

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Kettenregel

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial y}$$

Aktivierung eines Neurons

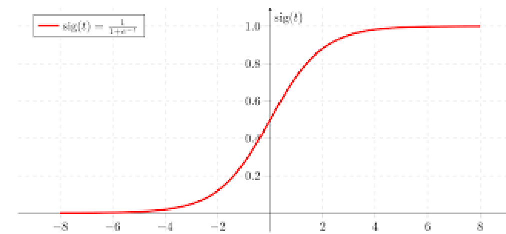


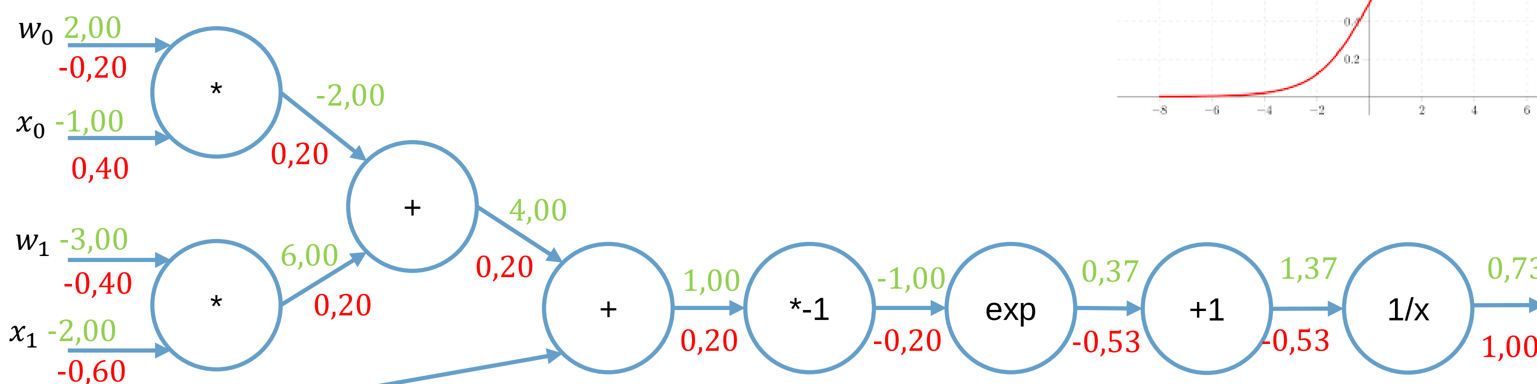
Backpropagation eines Neurons mit einer Sigmoid-Aktivierungsfunktion

Forward-Pass Backward-Pass

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

Sigmoid Gate





$f(x) = ax$	$f(x) = e^x$	$f(x) = c + x$	$f(x) = \frac{1}{x}$
$\frac{df}{dx} = a$	$\frac{df}{dx} = e^x$	$\frac{df}{dx} = 1$	$\frac{df}{dx} = -\frac{1}{x^2}$
-1*(-0,19)	$e^{-1}(-0{,}53)$	$(1)(-0{,}53)$	$\left(\frac{-1}{1{,}37^2}\right)(1{,}0)$
= 0,20	= -0,20	$= -0{,}53$	$= -0{,}53$

Sigmoid Gate

$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$

$$\sigma = \frac{1}{1+e^{-x}} \quad \text{Sigmoid Gate}$$

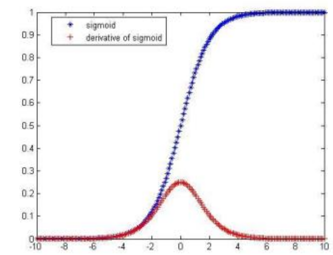
$$\frac{d\sigma}{dx} = \frac{e^{-x}}{(1+e^{-x})^2}$$

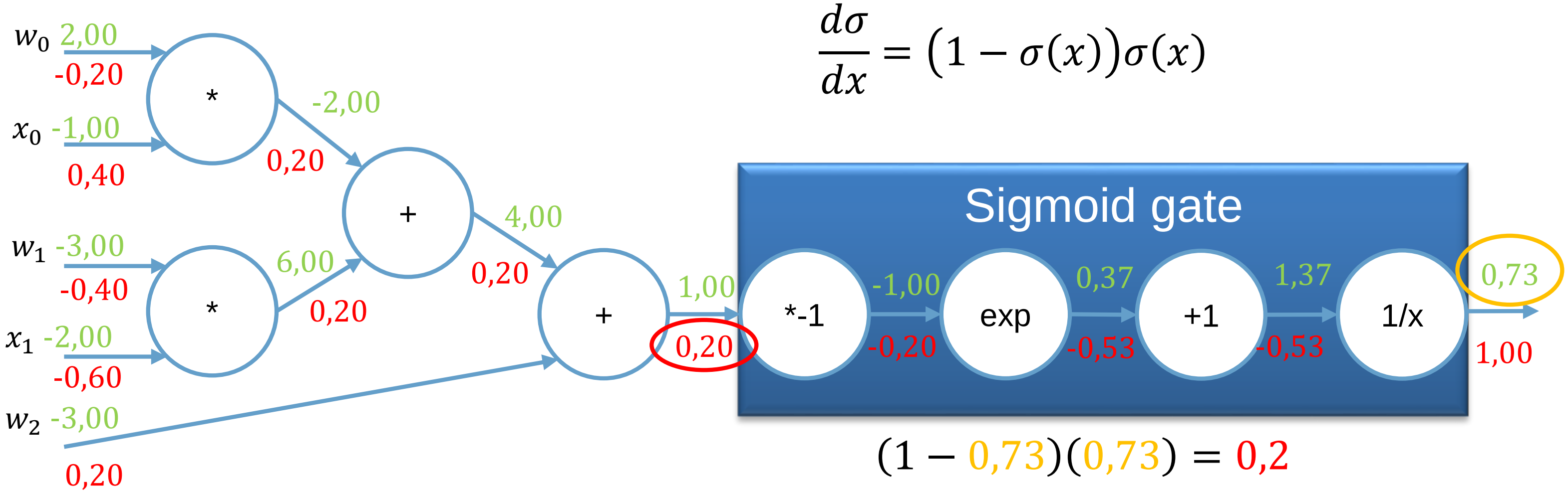
$$\frac{d\sigma}{dx} = \left(\frac{e^{-x}}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$$

$$\frac{d\sigma}{dx} = \left(\frac{1+\ e^{-x} - 1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$$

$$\frac{d\sigma}{dx} = \left(\frac{1+\ e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right)$$

$$\frac{d\sigma}{dx} = \big(1-\sigma(x)\big)\sigma(x)$$





$$(1 - 0{,}73)(0{,}73) = 0{,}2$$

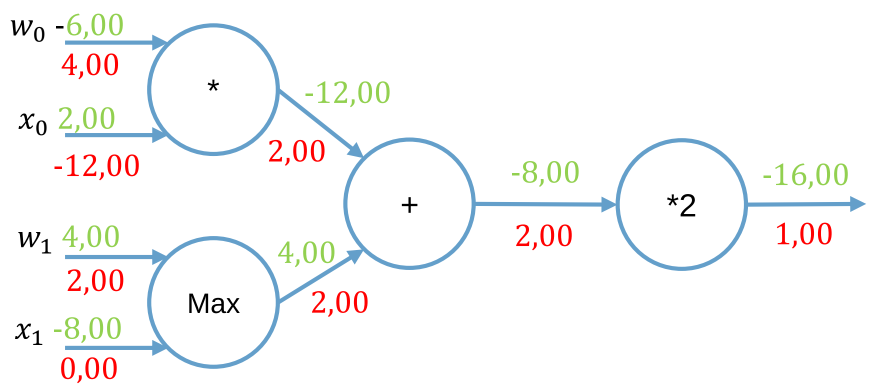
Muster in Backpropagation

Forward-Pass Backward-Pass

ADD-Gate: Gradient weiterleiten

MULT-Gate: Gradient Switcher

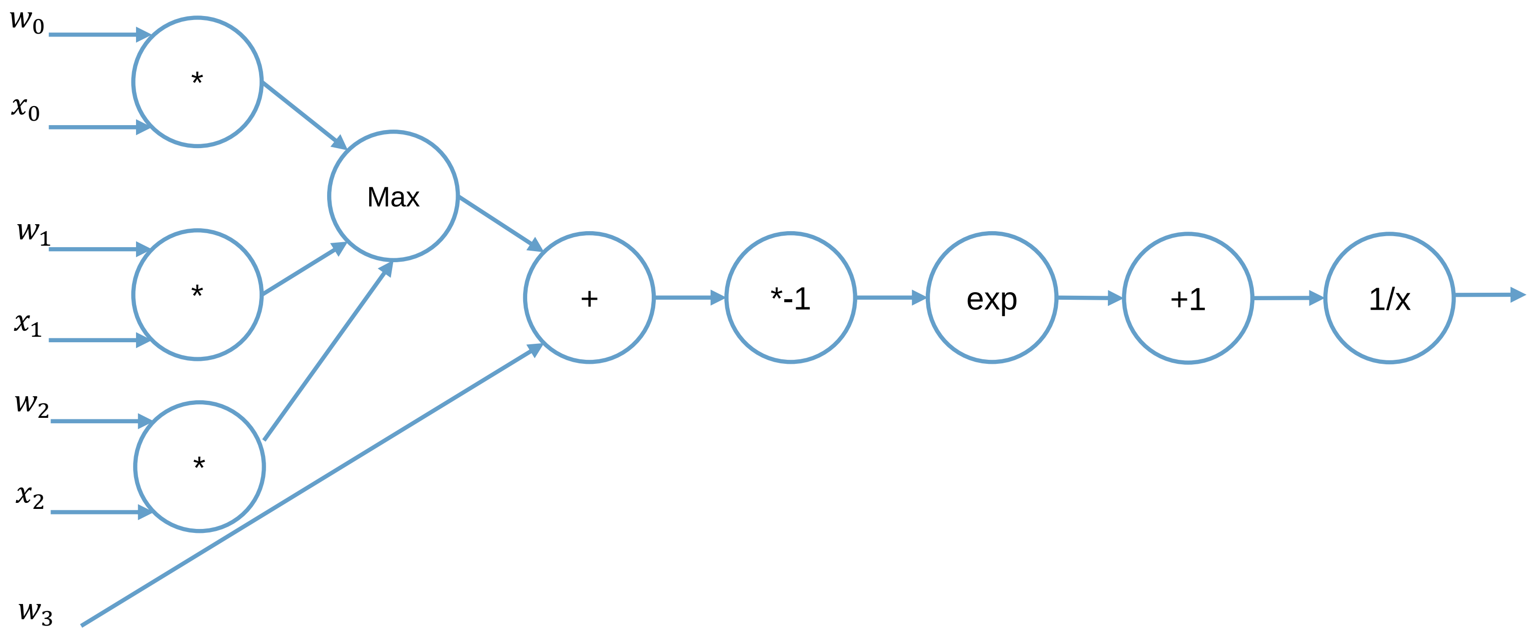
MAX-Gate: Gradient Router



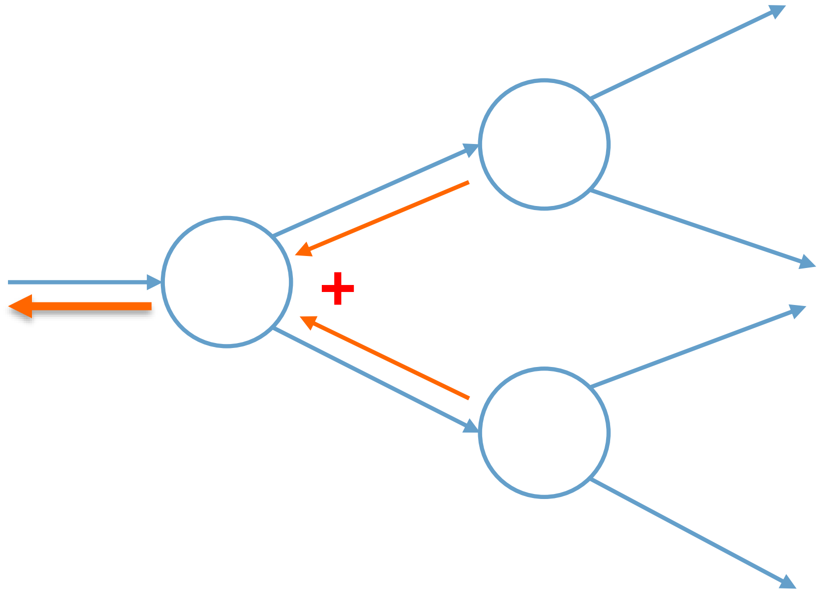
Aufgabe

$w_0 = -1; w_1 = 2; w_2 = -2 ; w_3 = 4$

$x_0 = 2; x_1 = 4; x_2 = 3$



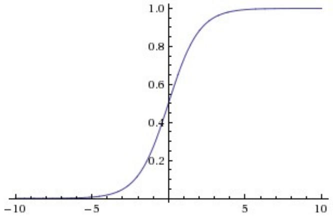
Gradient an Verzweigungen

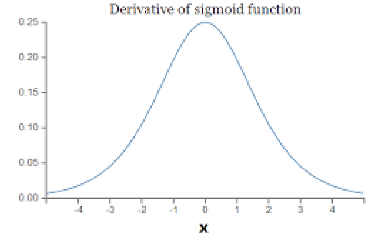


Zurückfliesende Gradienten in einem Netz werden addiert

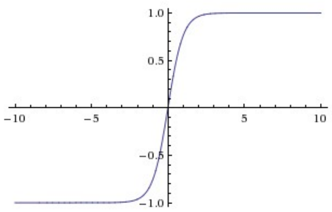
Activation Functions

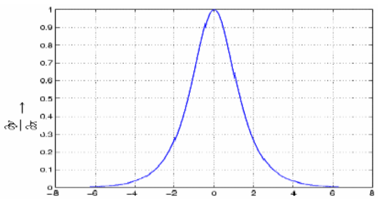
Sigmoid $\sigma(x) = 1/(1 + e^{-x})$



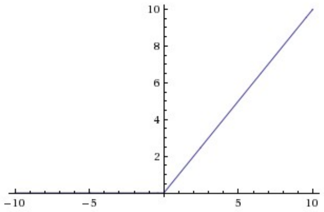


tanh tanh(x)



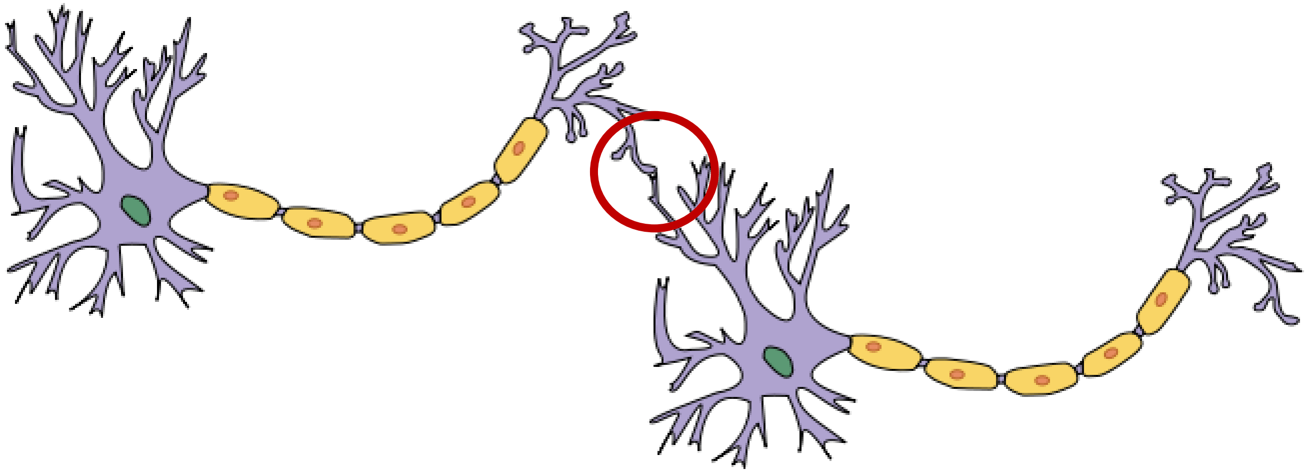


ReLU max(0,x)



?

Neuronales Netz mit zwei Layer



$f_1(x, W) = Wx$ $f_2(x, W) = Wx$

- L(f) = Softmax(f)
- $f(x, W_1, W_2) = W_2 * \tanh(W_1 x)$
- L(f(x, W_1, W_2)) = Softmax($W_2 * \tanh(W_1 x)$)

Neuronale Netze

1. Lineare Score Funtion:
   - $f(x, W) = Wx$
2. Zwei Layer NN:
   - $f(x, W_1, W_2) = W_2 * \text{sign}(W_1 x)$

Netz

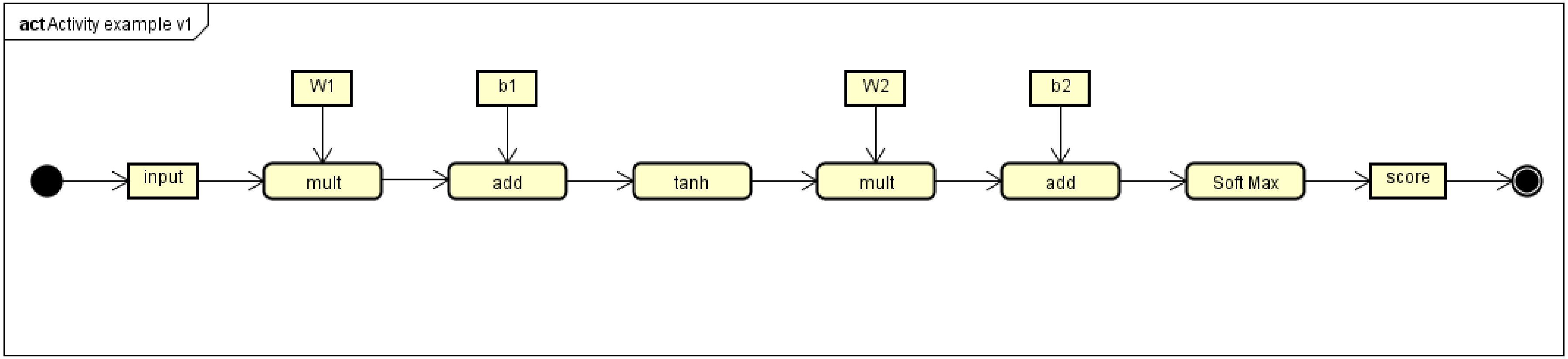
Gewichte Layer 2

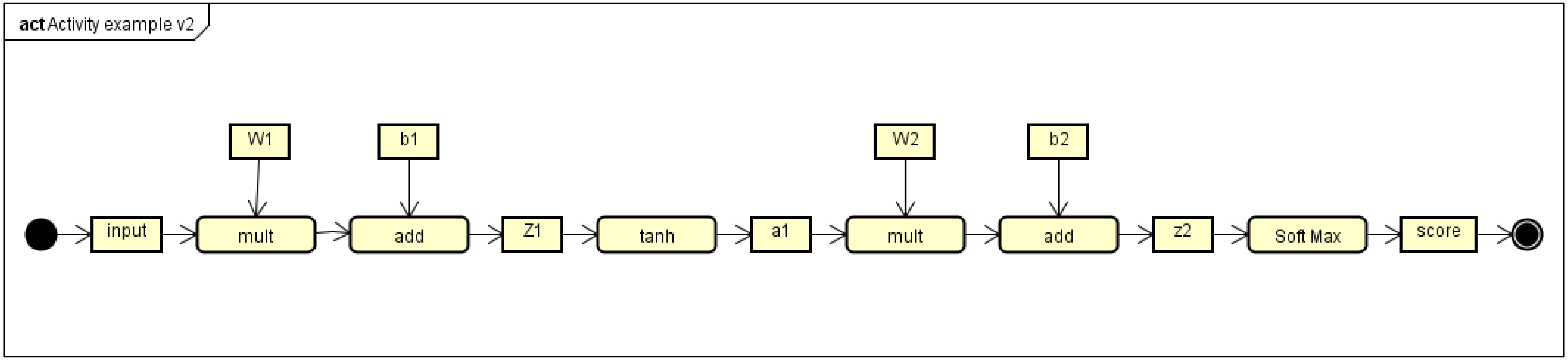
Übertragungs-funktion Layer 1

Gewichte Layer 2

Input

Beispiel von



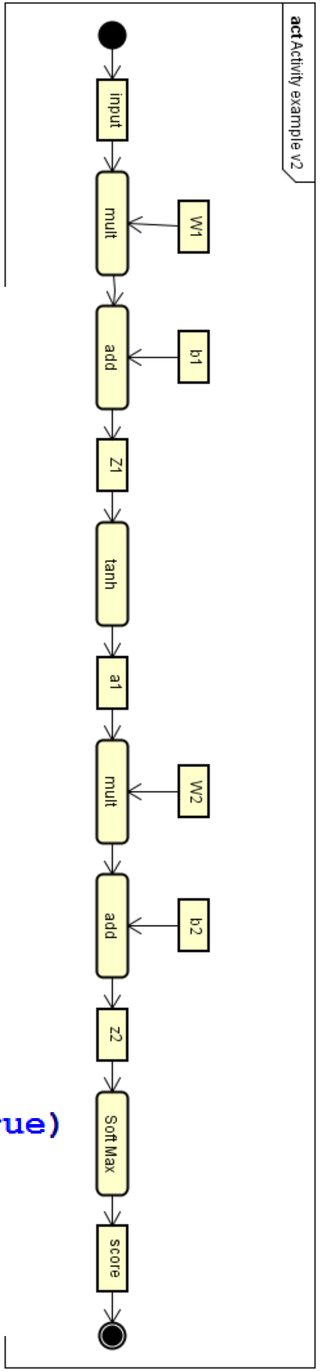


hochschule mannheim

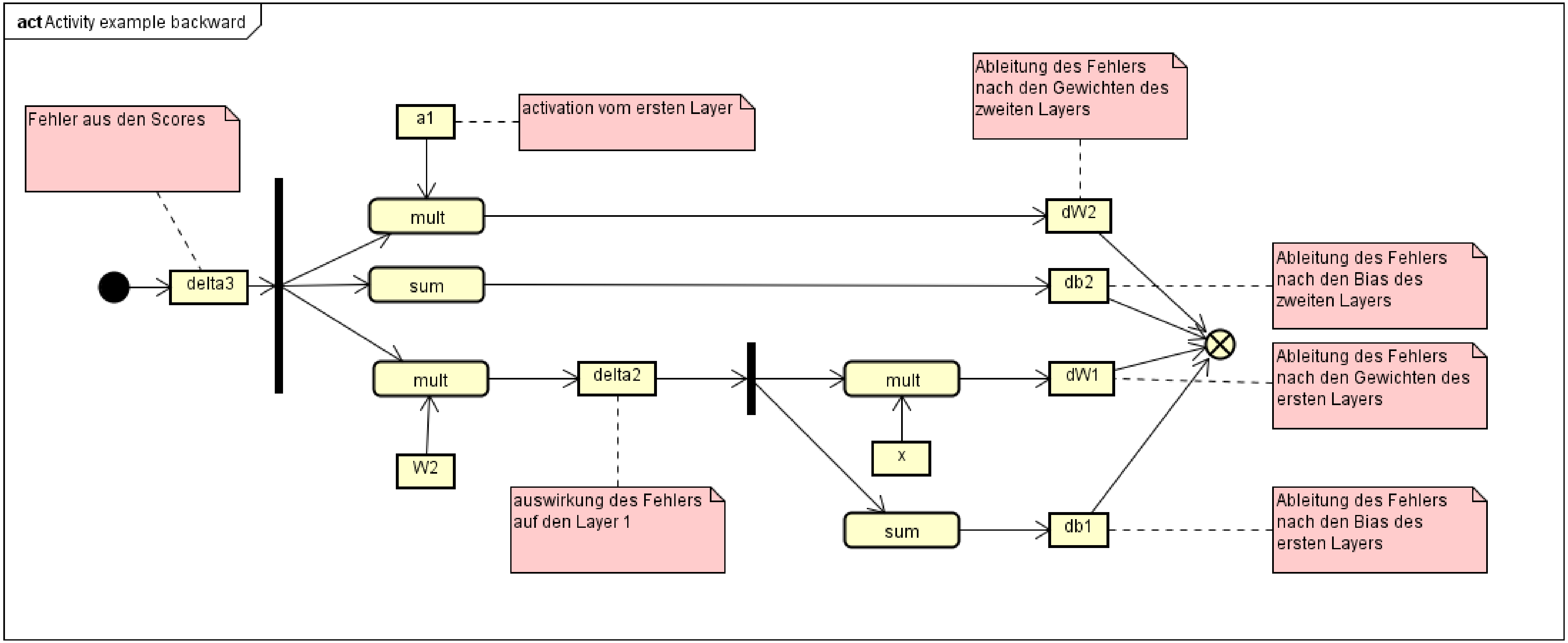
Feedforward

```
#Feedfoward pass
def feedforward(X, model):
    W1 = model['W1']
    b1 = model['b1']
    W2 = model['W2']
    b2 = model['b2']

    #x.W+b
    z1 = X.dot(W1) + b1

    #Activation function
    a1 = np.tanh(z1)

    #a1.W+b
    z2 = a1.dot(W2) + b2

    #Softmax
    exp_scores = np.exp(z2)
    probs = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)

    model['a1'] = a1

    return probs, model
```

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

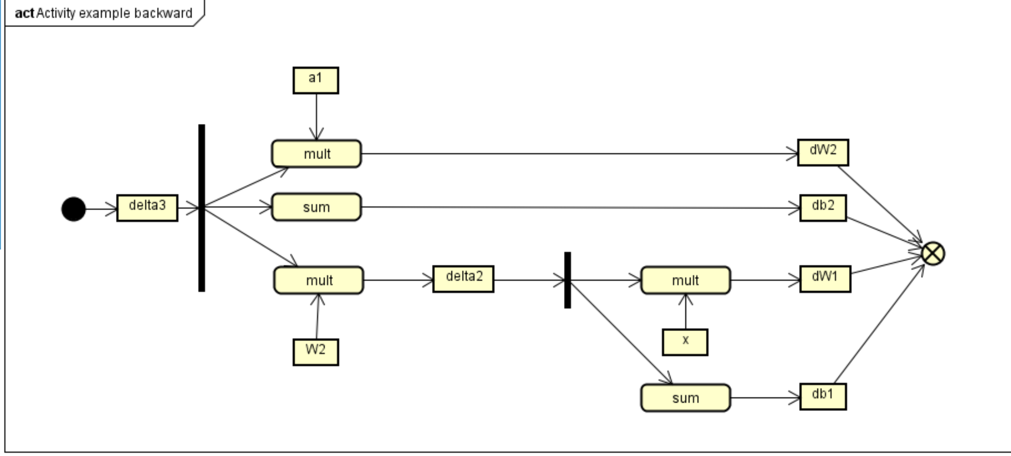


Feed backward



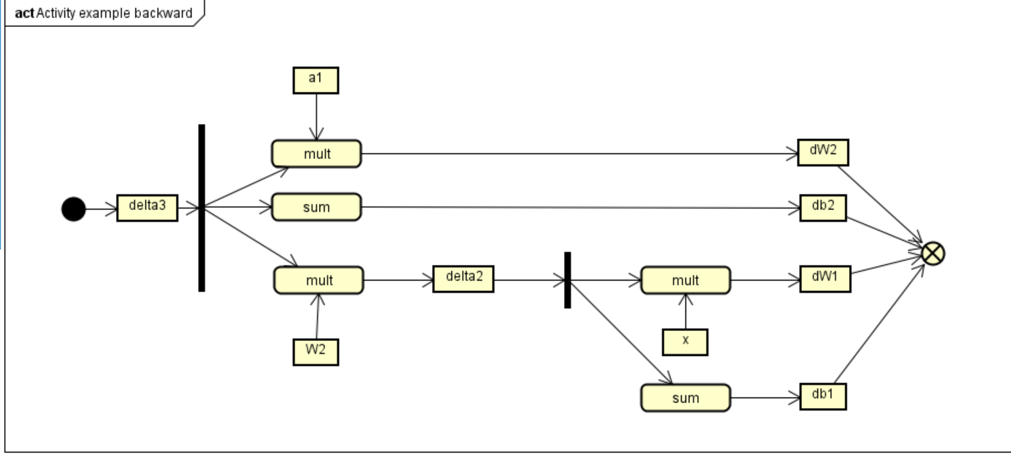
hochschule mannheim

Feed backward




```
#Backward pass
def backprop(X, y, probs, model):
    W1 = model['W1']
    W2 = model['W2']
    a1 = model['a1']

    delta3 = np.array(probs)
    delta3[range(len(X)), y] -= 1
    dW2 = (a1.T).dot(delta3)
    db2 = np.sum(delta3, axis=0, keepdims=True)
    delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
    dW1 = np.dot(X.T, delta2)
    db1 = np.sum(delta2, axis=0)

    #Ableitung der Regularisierung
    dW2 += reg_lambda * W2
    dW1 += reg_lambda * W1

    deltas = { 'dW1' : dW1, 'db1' : db1, 'dW2' : dW2, 'db2' : db2}

    return deltas
```

Feed backward




```
#Backward pass
def backprop(X, y, probs, model):
    W1 = model['W1']
    W2 = model['W2']
    a1 = model['a1']

    delta3 = np.array(probs)
    delta3[range(len(X)), y] -= 1
    dW2 = (a1.T).dot(delta3)
    db2 = np.sum(delta3, axis=0, keepdims=True)
    delta2 = delta3.dot(W2.T) * (1 - np.power(a1, 2))
    dW1 = np.dot(X.T, delta2)
    db1 = np.sum(delta2, axis=0)

    #Ableitung der Regularisierung
    dW2 += reg_lambda * W2
    dW1 += reg_lambda * W1

    deltas = { 'dW1' : dW1, 'db1' : db1, 'dW2' : dW2, 'db2' : db2}

    return deltas
```

Wahrscheinlichkeiten (Softmax)

Der Fehler aus den Scores

Auswirkung der Gewichte auf den Fehler

Auswirkung des Bias auf den Feh

Ableitung der Aktivierungsfunktion

tanh(z1)

$$\lambda \sum_k W_k$$

Parameter Update

```
def parameter_update(model, deltas, l_r):

    learning_rate = l_r

    dW1 = deltas['dW1']
    db1 = deltas['db1']
    dW2 = deltas['dW2']
    db2 = deltas['db2']

    model['W1'] += -learning_rate * dW1
    model['b1'] += -learning_rate * db1
    model['W2'] += -learning_rate * dW2
    model['b2'] += -learning_rate * db2

    return model
```

Beispiel Netz

```python
def main():
    #Generate Data
    X, y = create_data(300)

    #Split data
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2)

    #Train model
    model = train_model(X_train, y_train, 2, 200)

    #Valdiate model
    validate(X_val, y_val, model)

    #Show results
    plot_results(X_val, y_val, model)
```

Train

```
# This function learns parameters for the neural network and returns the r
# - neurons: Number of neurons in the hidden layer
# - epochs: Number of passes through the training data for gradient descer
def train_model(X, y, nn_hdim, epochs=200):

    # Initialize the parameters to random values. We need to learn these.
    np.random.seed(0)
    W1 = np.random.randn(nn_input_dim, nn_hdim) / np.sqrt(nn_input_dim)
    b1 = np.zeros((1, nn_hdim))
    W2 = np.random.randn(nn_hdim, nn_output_dim) / np.sqrt(nn_hdim)
    b2 = np.zeros((1, nn_output_dim))

    # This is what we return at the end
    model = { 'W1': W1, 'b1': b1, 'W2': W2, 'b2': b2}

    # Gradient descent. For the complete training data...
    for i in xrange(0, epochs):

        # Forward propagation
        probs, model = feedforward(X, model)

        # Backpropagation
        deltas = backprop(X, y, probs, model)

        # Gradient descent parameter update
        model = parameter_update(model, deltas, learning_rate/len(X))

    #Accuracy
    y_pred = np.argmax(probs, axis=1)
    accuracy = np.mean(np.array(y_pred == y, dtype=np.uint))
    print 'Training accuracy for epoch %d : %f ' %(i, accuracy)

    return model
```

Xavier-Initialisierung

Backpropagation

Keras und andere Tool-Kits haben uns die bestimmung der analytischen Lösung für die existiereden Layer bereits abgenommen!

Ausblick

1. Loss functions
   - Ist W gut?

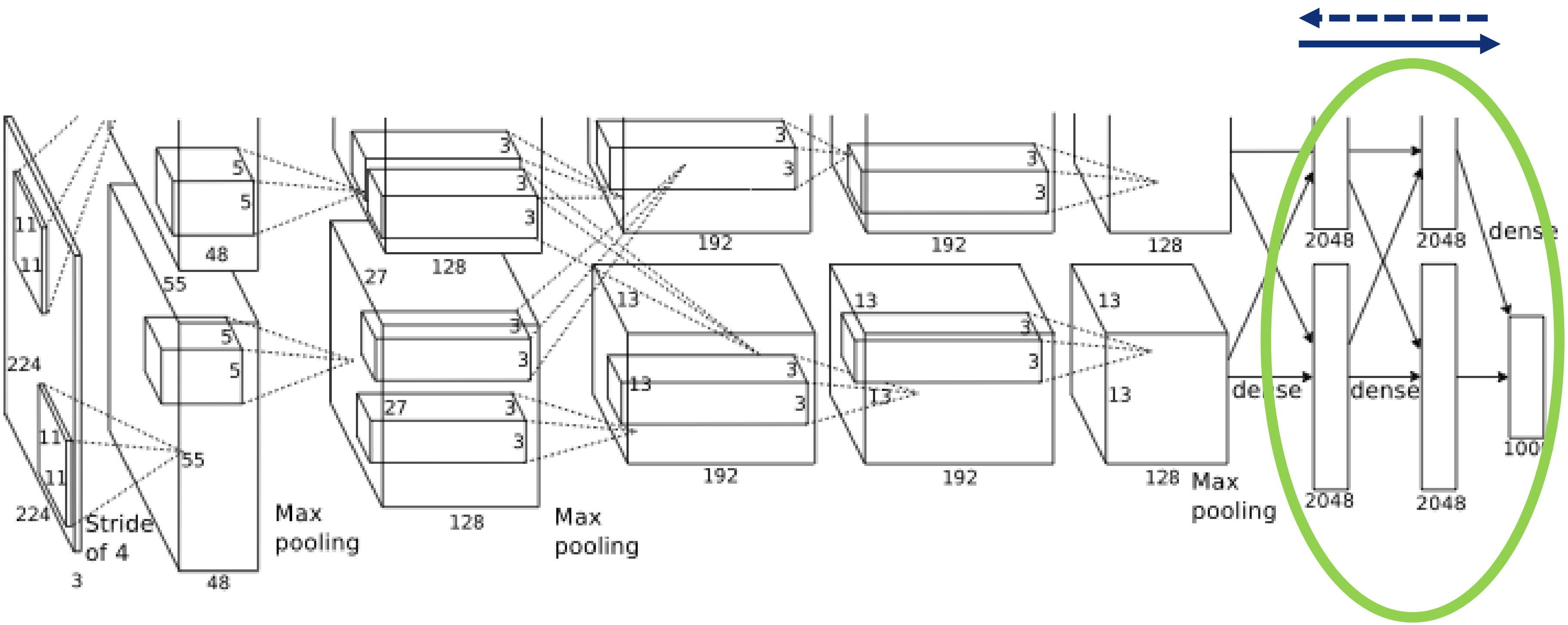
1. Optimierung
   - Wie wird W besser?

- Backpropagation
  - Wie verbessere ich W über mehrere Schichten hinweg?

2. CNN
   - Gibt es bessere Score Functions $f(x, W)$ für Bilder

3. DNN
   - Mehr Layer!

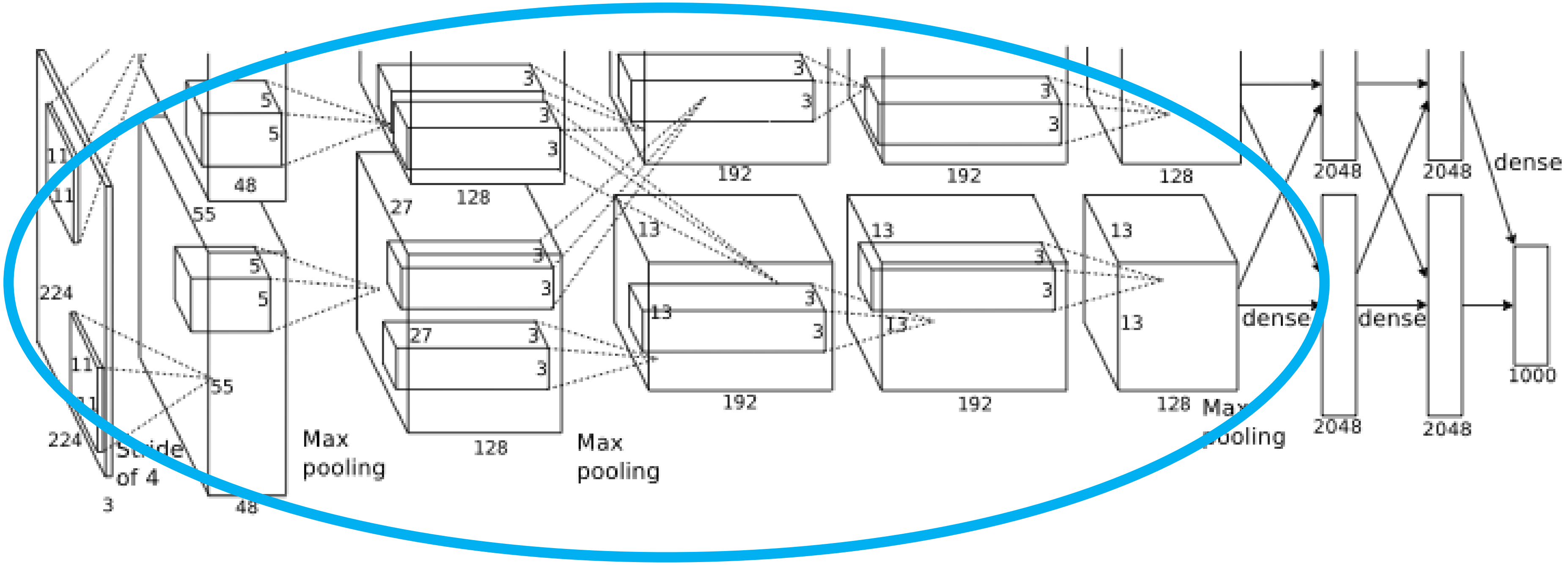
Ausblick



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

Ausblick

1. Convolutional Layer (Faltungsschichten)
   - Die Funktion $f(x, W)$ ändert sich, den Rest beherrschen sie bereits!
2. Pooling Layer



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