

# Deep Learning

Eine Einführung

# Überblick

- Das Grundmodell
  - Forward propagation
    - Layer 1:
    - Layer 2: Aktivierungsfunktion, Nichtlinearität
  - Loss function
  - Optimierungsfunktion
  - Backward propagation
  - Regularisierung
    - Dropout
- Architekturen
  - Simple Feedforward Network
  - Convolutional Network
  - Recurrent Network
    - LSTM
- Ausblick:
  - Autoencoder / -decoder
  - Generative Adversarial Networks
  - Reinforcement Learning

# Beispiel: Klassifikation

MNIST:  
Erkennung von  
handschriftlich  
notierten Zahlen

70.000 28x28  
Graustufen-Bilder

label = 5



label = 0



label = 4



label = 1



label = 9



label = 2



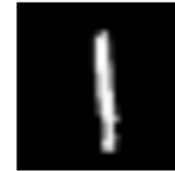
label = 1



label = 3



label = 1



label = 4



label = 3



label = 5



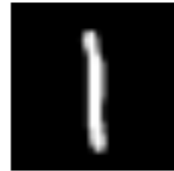
label = 3



label = 6



label = 1



label = 7



label = 2



label = 8



label = 6




label = 9




# Klassifikation (überwachtes Lernen)


## Training

x = 


y = 5

x = 

y = 2

x = 

y = 3


x = 

y = 2



Modell

## Anwendung

x = 

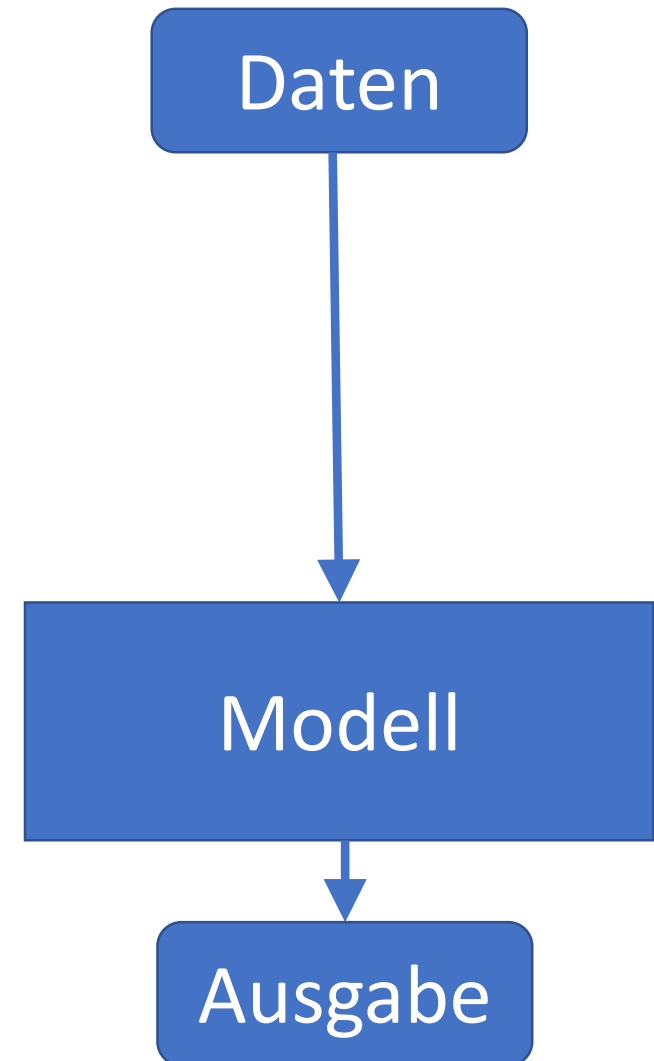
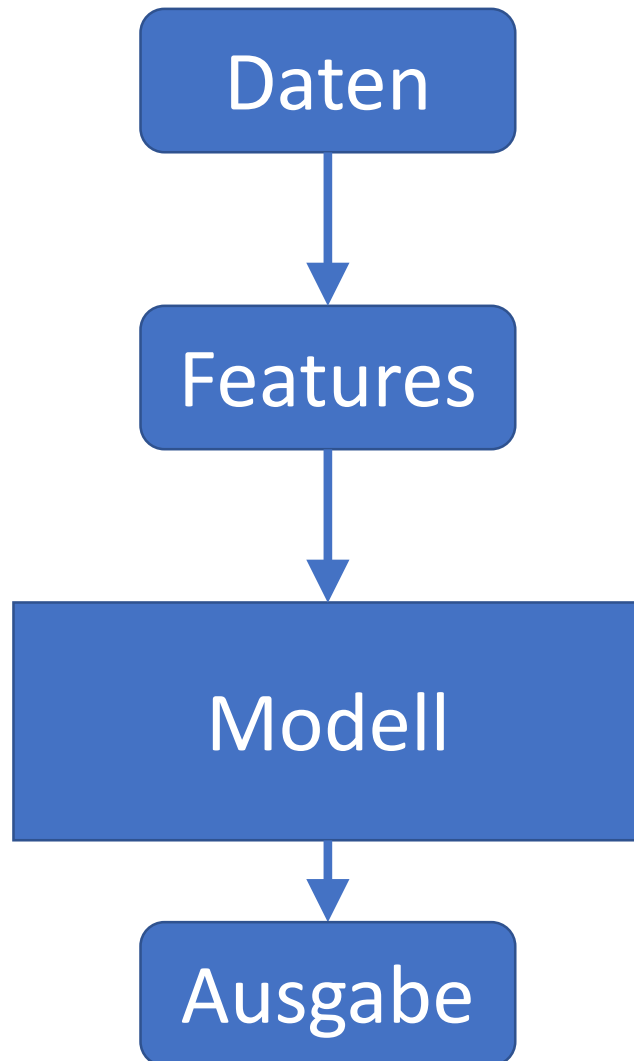
Modell

y = 5

# Überwachtes Maschinelles Lernen: Klassifikation

- Ziel ist es eine Funktion zu finden, die die Eingabe  $x$  (Daten) auf die Ausgabe  $y$  (Klasse) abbildet:
- $y = f(x)$
- Diese Funktion wird das ‚Modell‘ genannt.

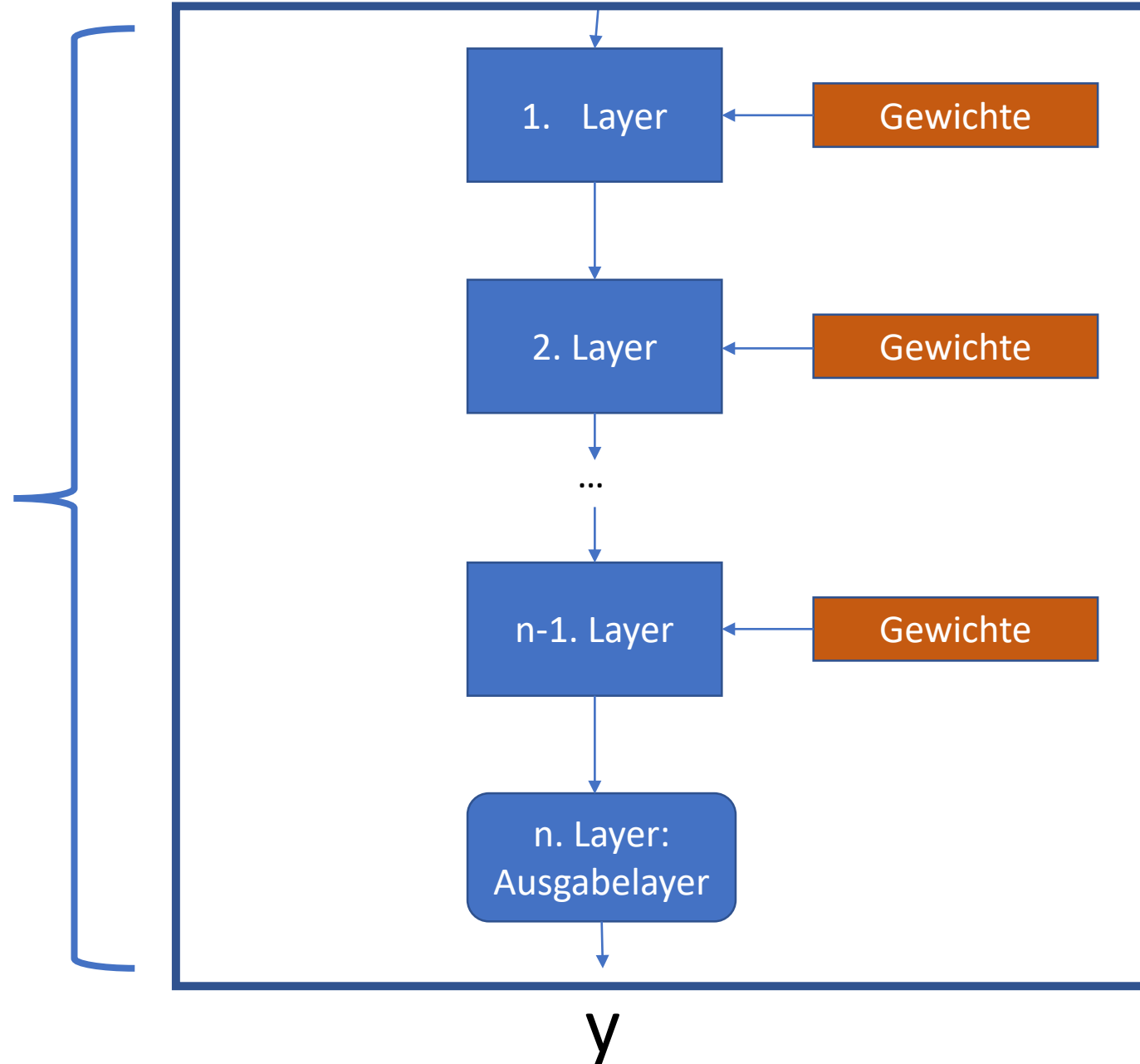
# Maschinelles Lernen vs. Deep Learning



Modell

Modell

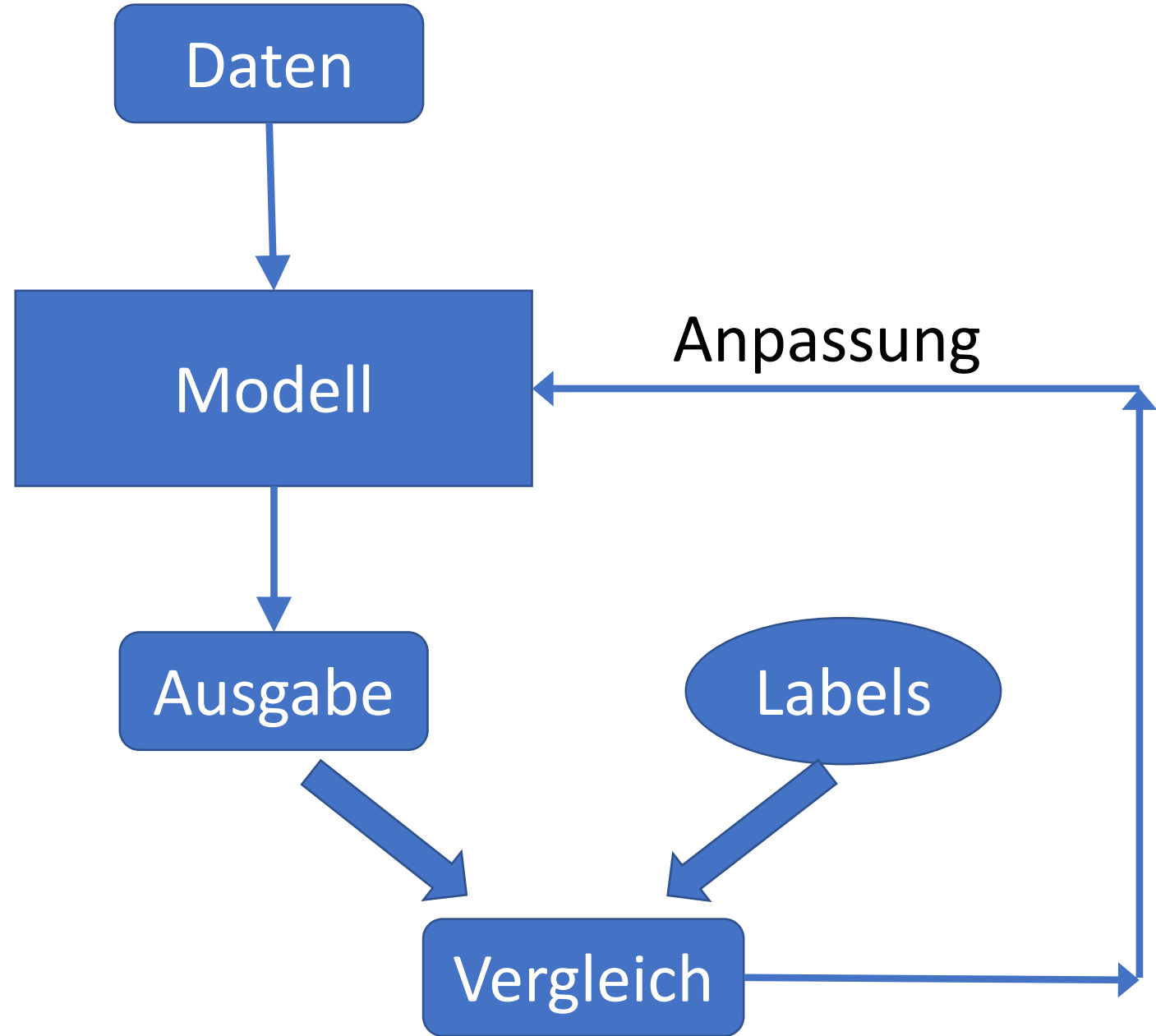
Input x



# Training

Beispiel  
Trainingsdaten:

label = 5





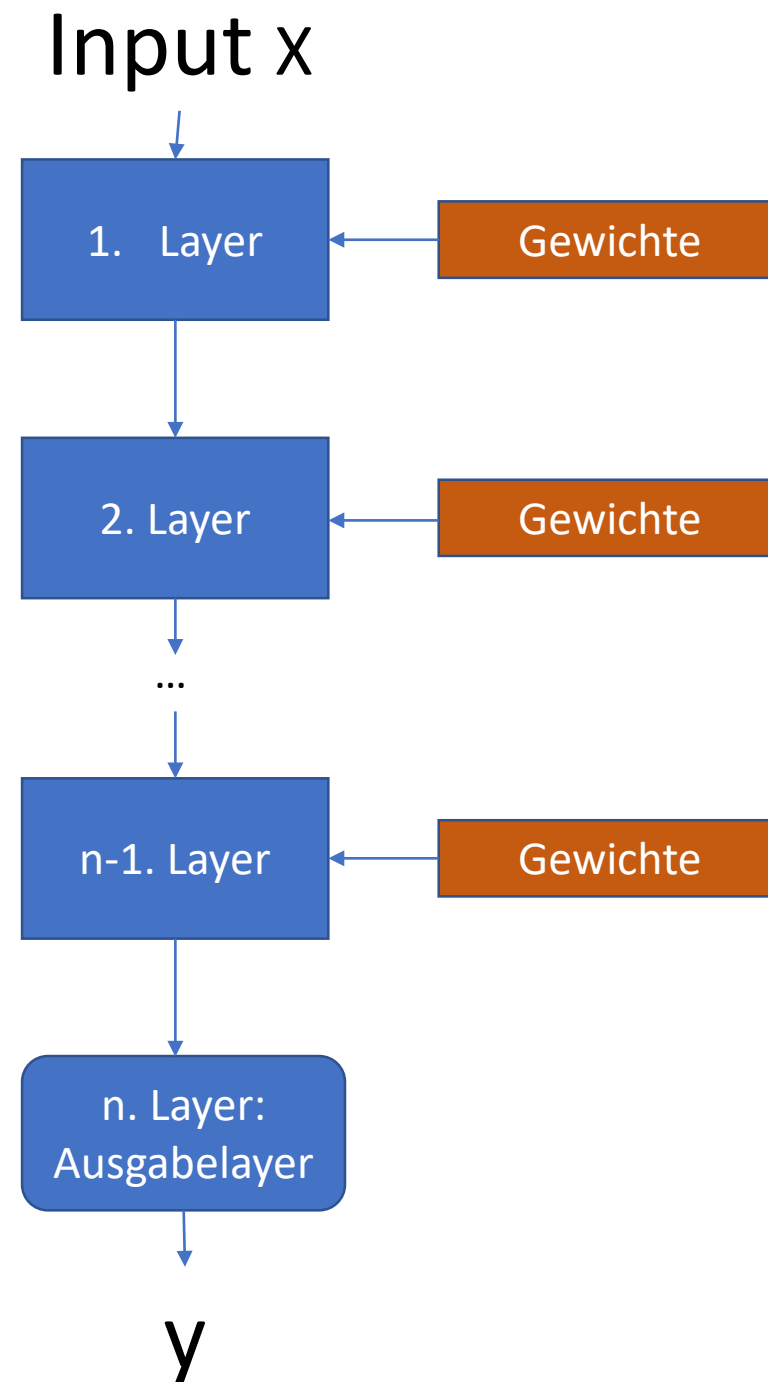
# Deep Learning: Training

Klassifikation

# DL im Überblick

## Forward propagation

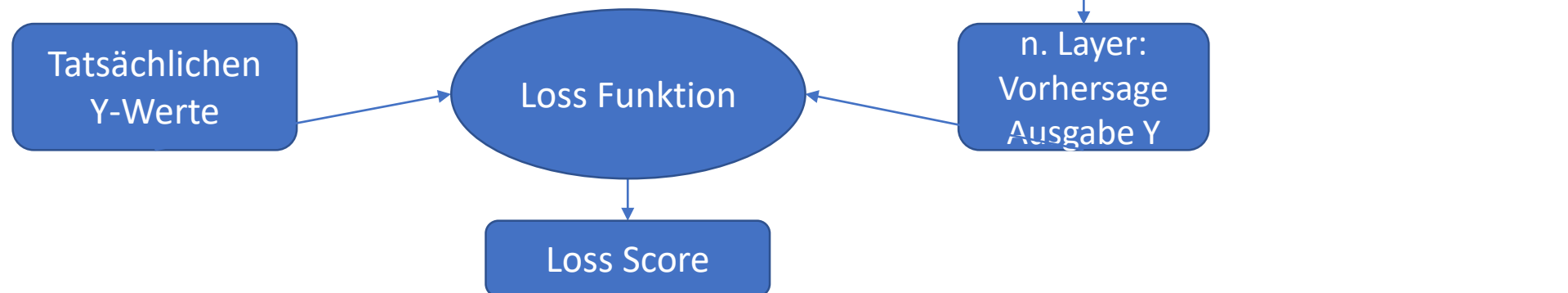
Ziel des Trainings beim Deep Learning ist es, die besten Gewichte zu finden.



# DL im Überblick

## Loss Function

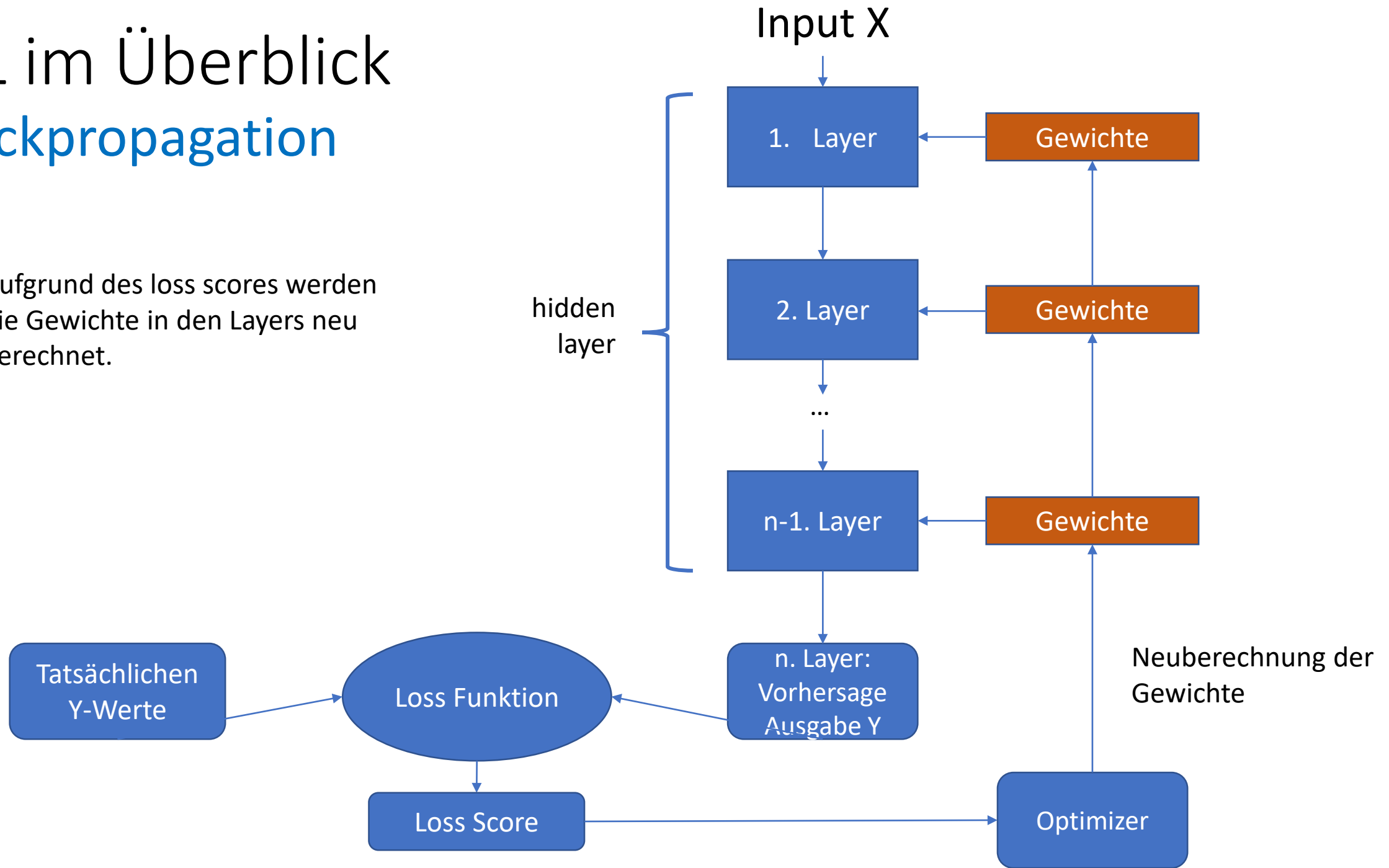
Die Loss Function gibt Auskunft über die Qualität der Vorhersage des Netzwerks. Das Maß dafür ist der Loss Score



# DL im Überblick

## Backpropagation

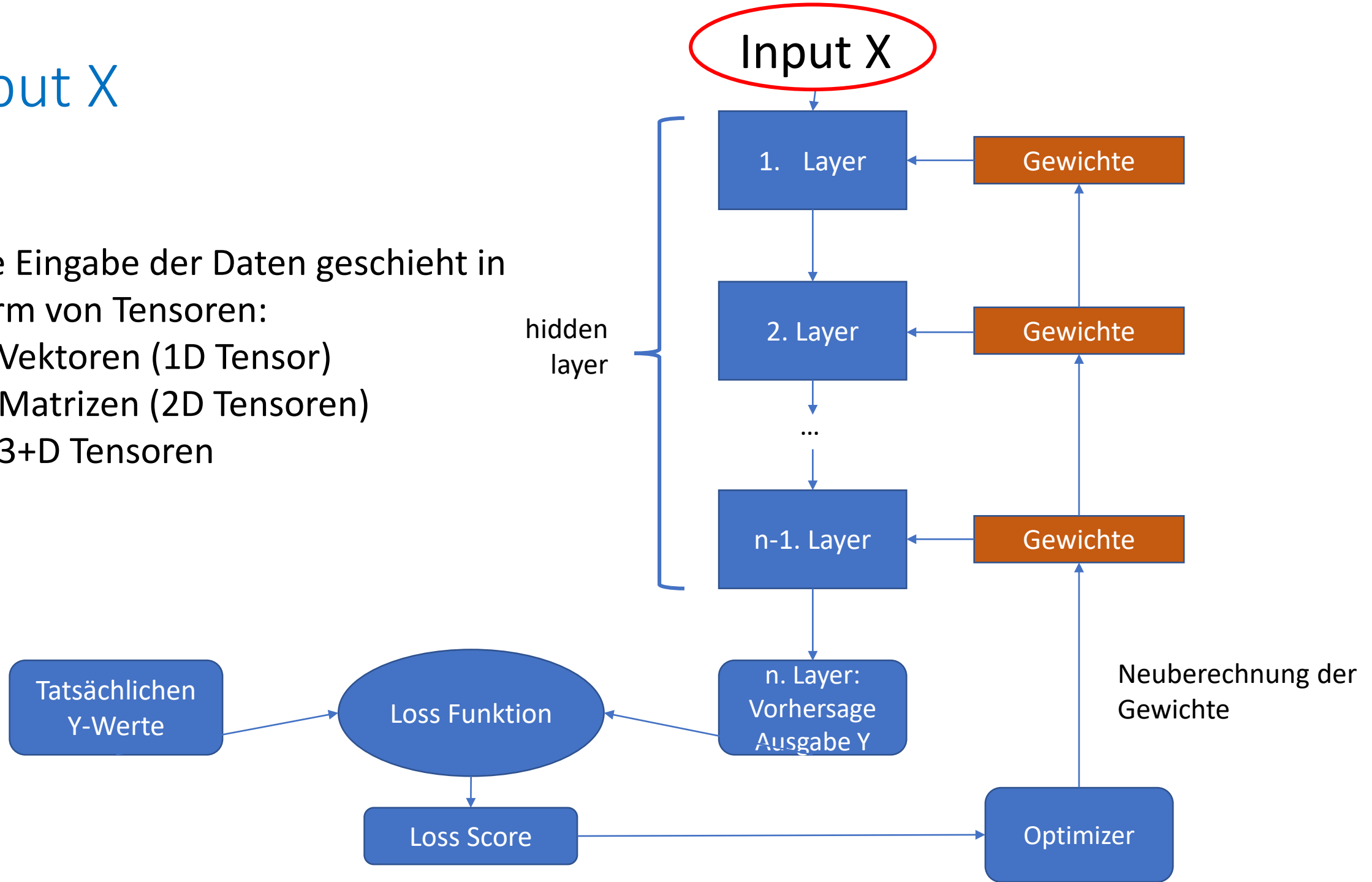
Aufgrund des loss scores werden die Gewichte in den Layers neu berechnet.



# Input X

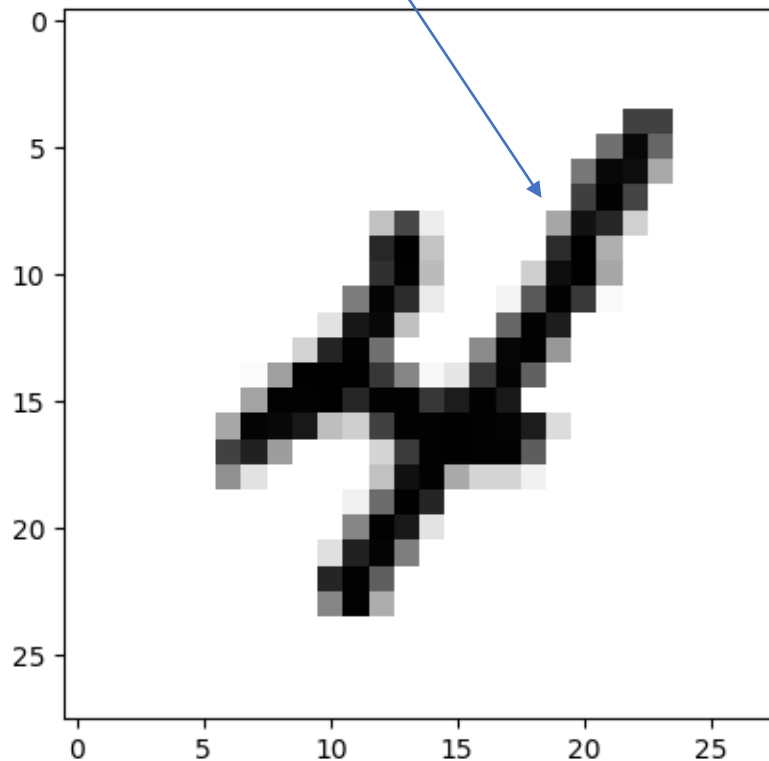
Die Eingabe der Daten geschieht in Form von Tensoren:

- Vektoren (1D Tensor)
- Matrizen (2D Tensoren)
- 3+D Tensoren



# Beispiel Eingabe Bilddaten: ein Bild

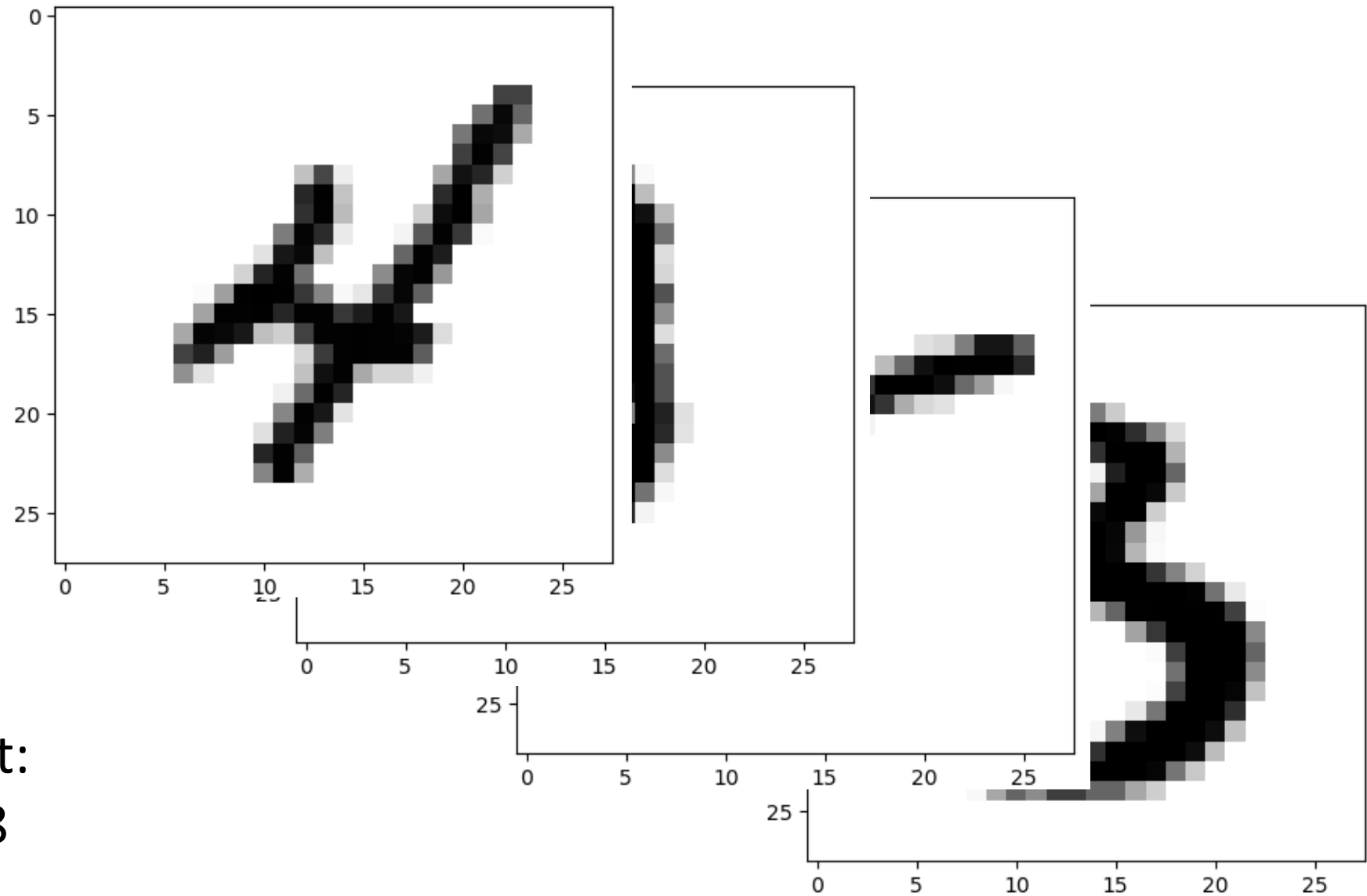
28x28 Grauwerte,  
8-bit kodiert



Matrix 28x28 mit Integers

```
[ [ 0  0  0  0  0  ...  0  0  0  0  0  0  0 ]  
  [ 0  0  0  0  0  ...  0  0  0  0  0  0  0 ]  
  [ 0  0  0  0  0  ...  0  0  0  0  0  0  0 ]  
  [ 0  0  0  0  0  ...  0  0  0  0  0  0  0 ]  
  [ 0  0  0  0  0  ...  0  0 189 190  0  0  0 ]  
  [ 0  0  0  0  0  ...  0  0 143 247 153  0  0 ]  
  [ 0  0  0  0  0  ...  0  0 136 247 242  0  0 ]  
  ...  
]
```

Input X

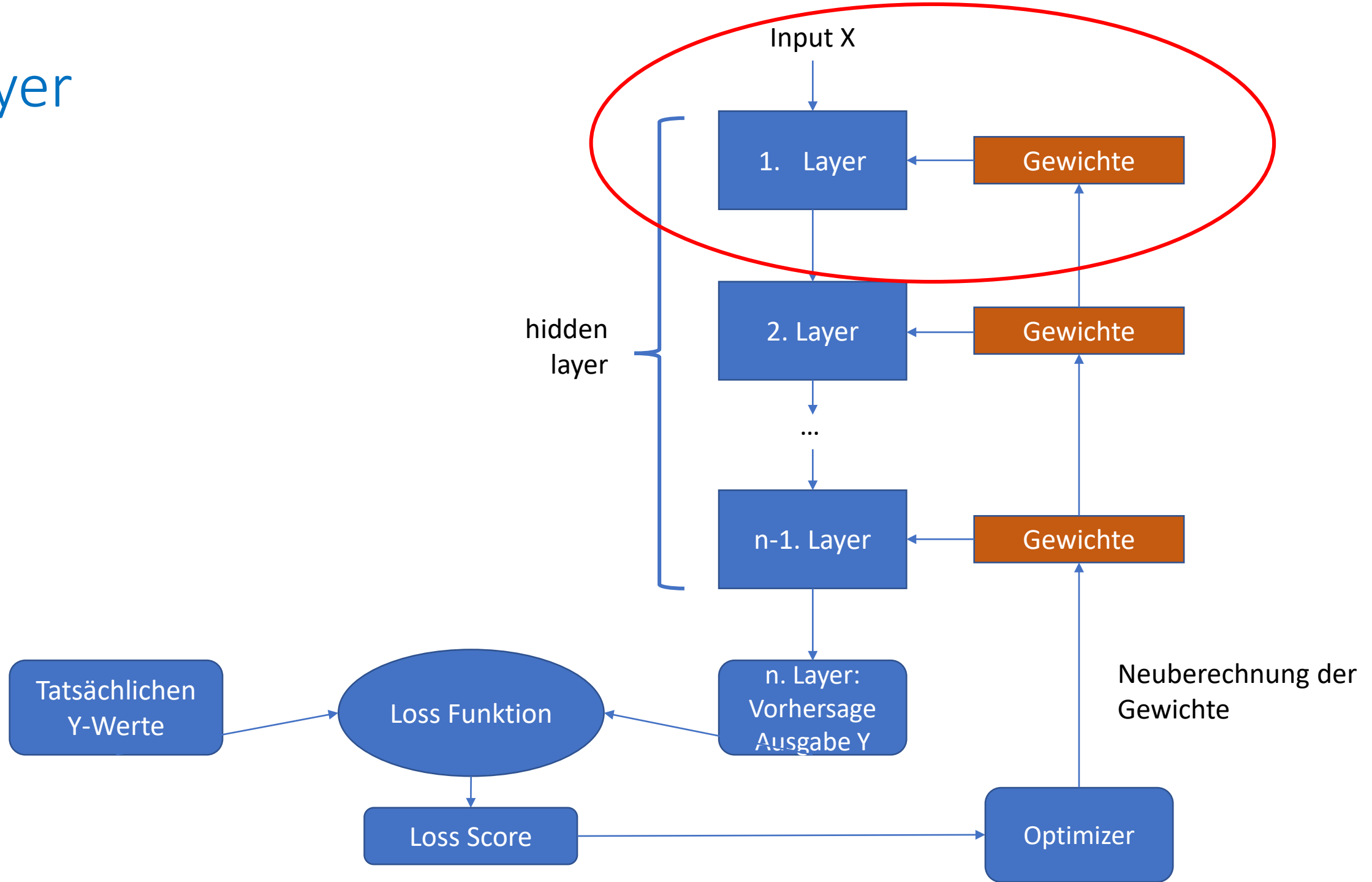


3D Tensor mit:  
60000, 28, 28

60.000 Bilder

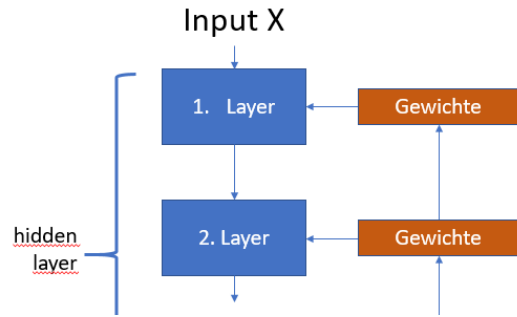


# Layer

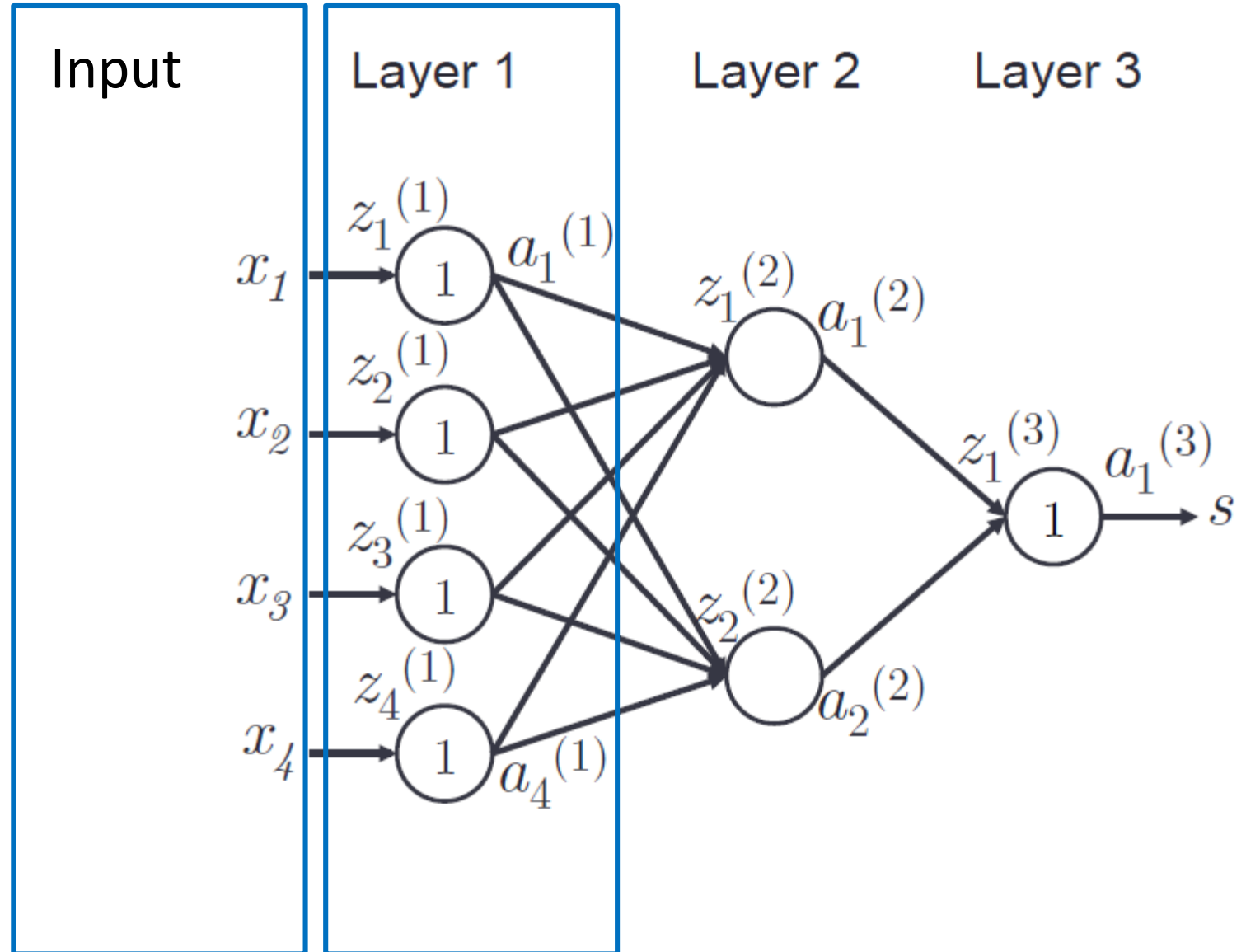




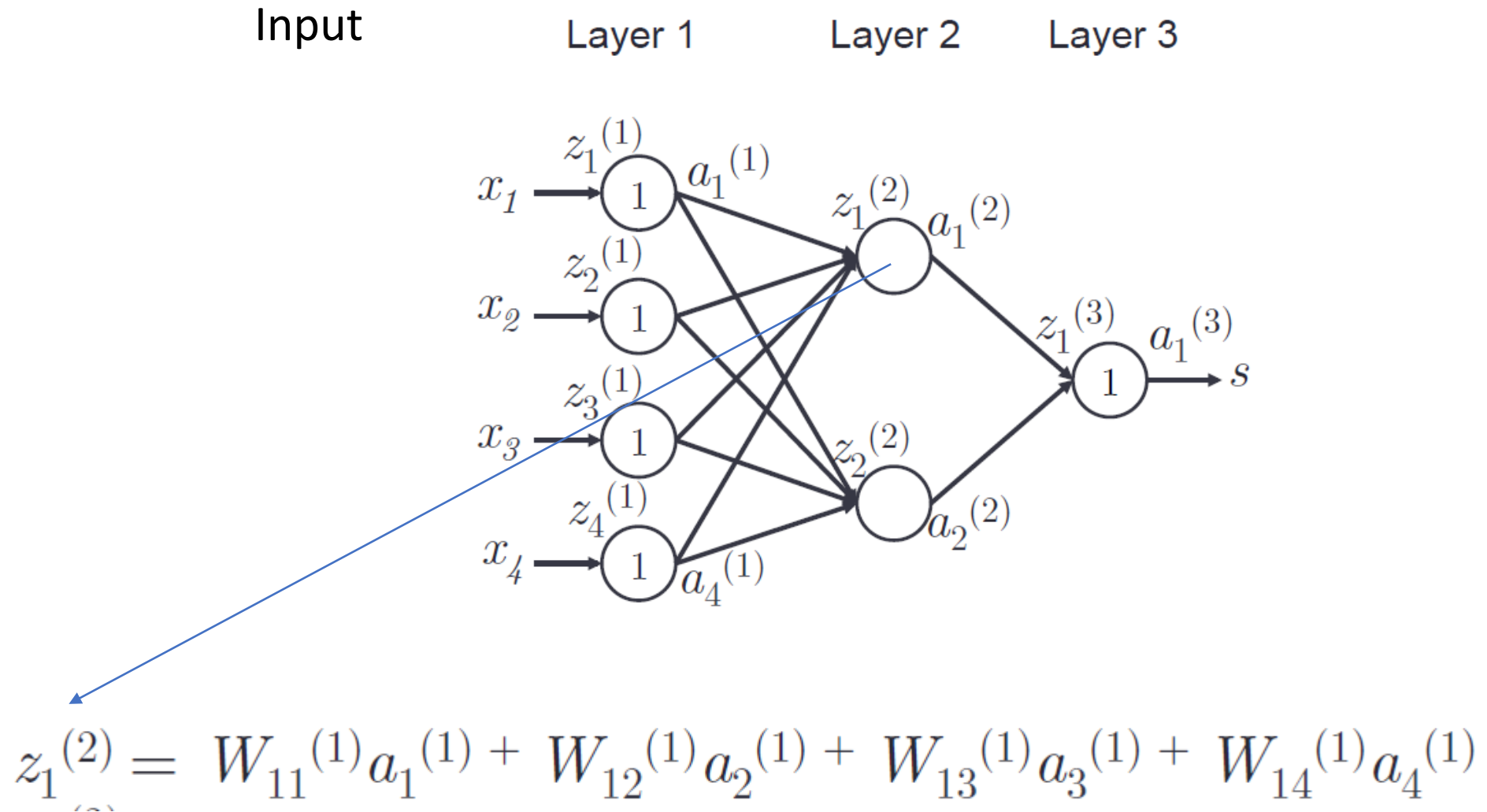
# Layer



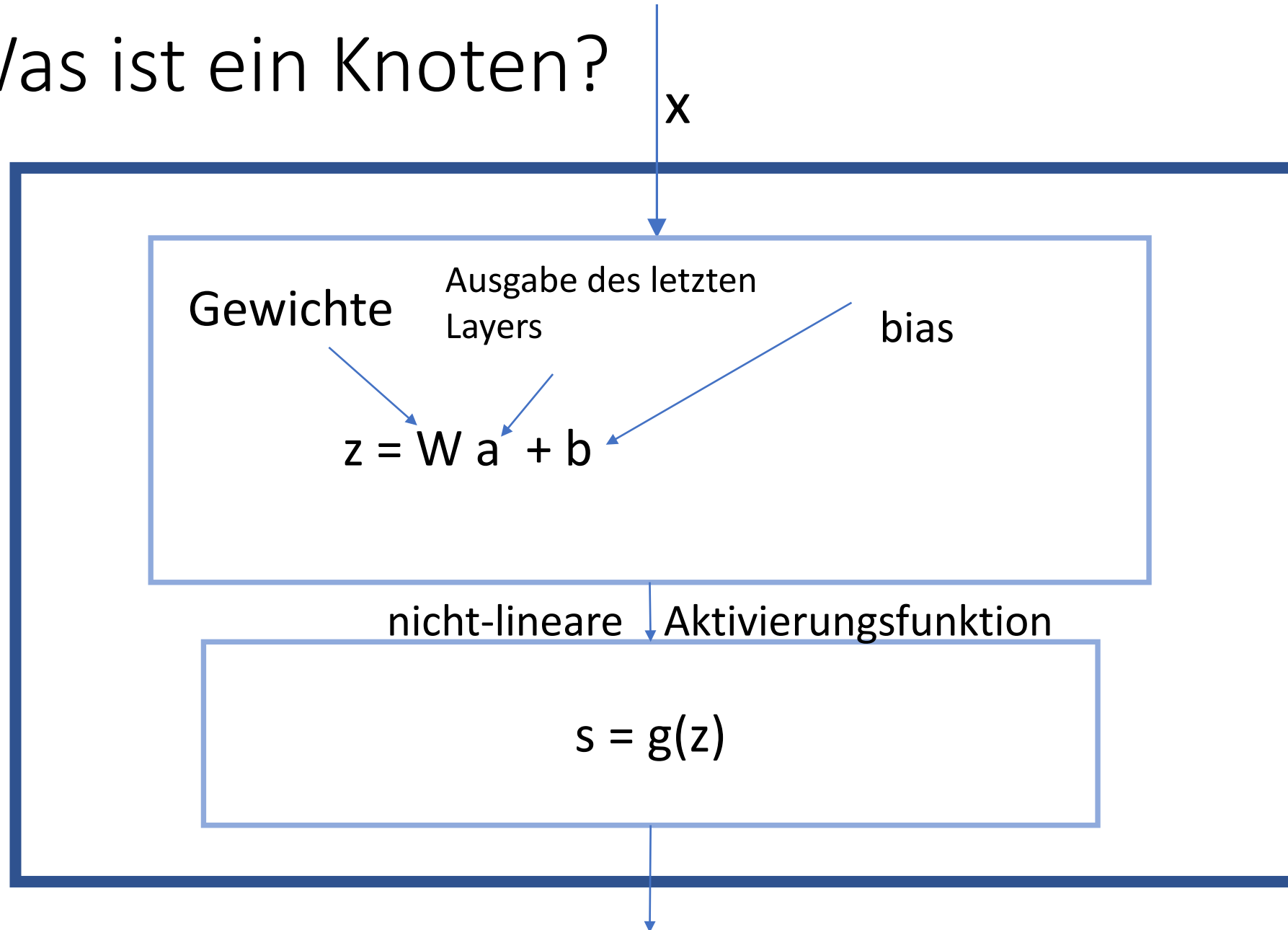
In Keras: Dense Layer



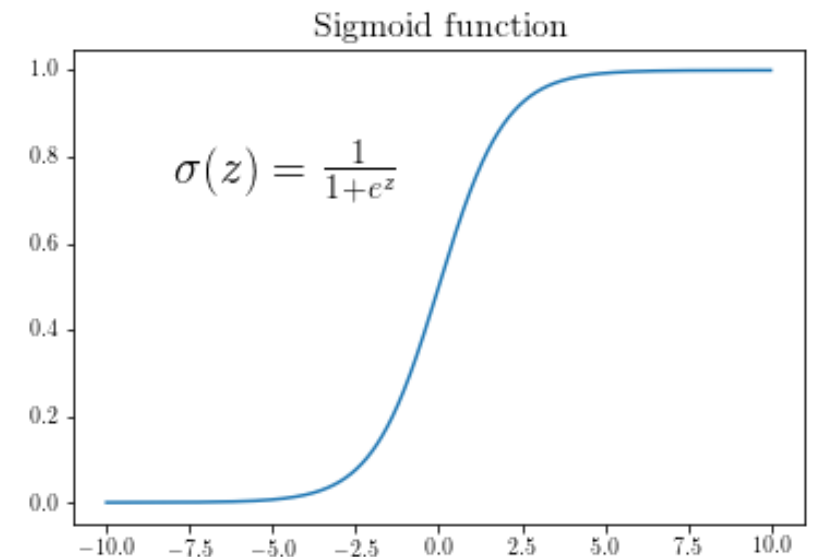
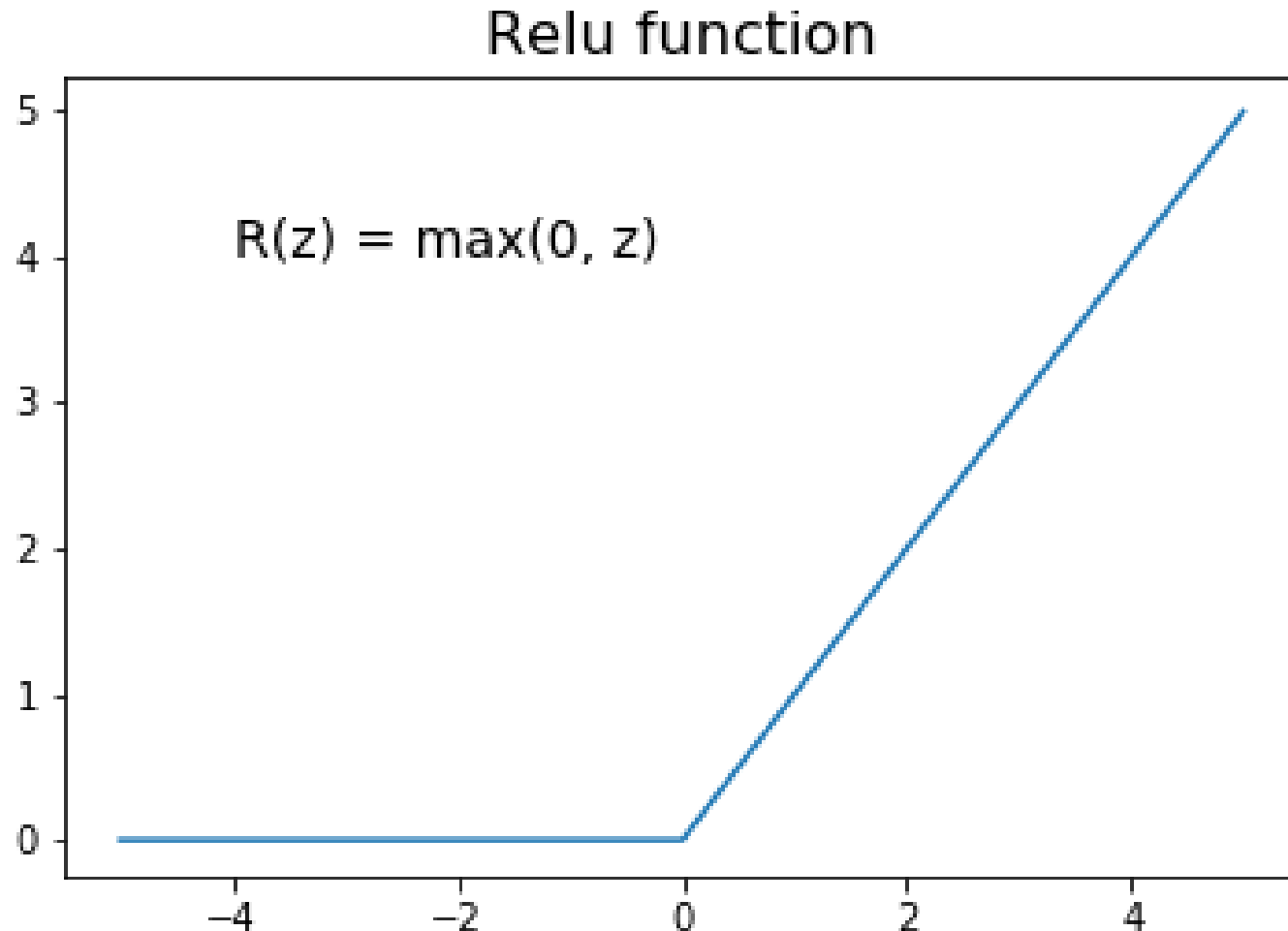
# Layer, Knoten und Verbindungen



# Was ist ein Knoten?



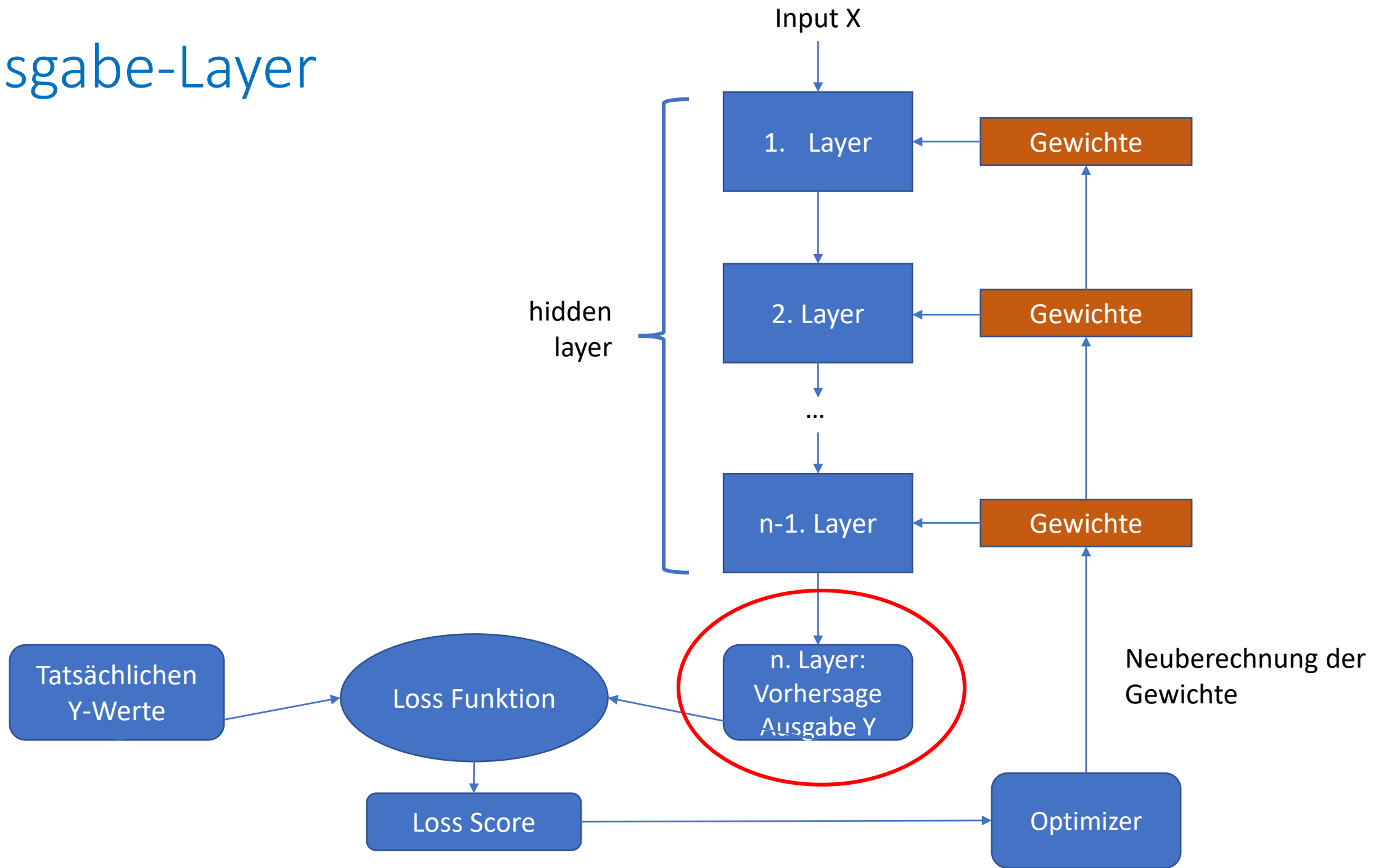
# Nicht-lineare Aktivierungsfunktion: Rectified linear unit



# Warum ‚nicht-linear‘

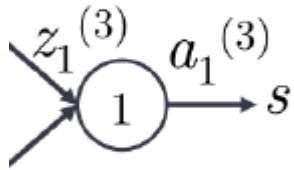
- Die erste ‚Schicht‘ eines Knoten besteht aus der Funktion  
 $z = W a + b$
- Sowohl die Multiplikation als auch die Addition ergeben nur eine affine (lineare) Transformation der Eingabewerte.
- Selbst wenn man mehrere solche Layer hintereinander schaltet, würde das nur auf die gleiche Form der Transformation ergeben.
- Eine Folge wäre, dass der Hypothesenraum zu eingeschränkt wäre (Hypothesenraum: Menge der möglichen Funktionen, die die Eingabe  $x$  auf die Ausgabe  $y$  abbilden).
- Funktion der nicht-linearen Aktivierungsfunktion: Den Hypothesenraum zu erweitern.

# Ausgabe-Layer



# Ausgabe. Beispiel Klassifikation

## Softmax



Nimmt als Eingabe einen Vektor mit  $n$  Elementen mit beliebigen Werten und erzeugt als Ausgabe einen Vektor mit  $n$  Elementen zwischen 0 und 1, deren Summe 1 ergibt. Wird verwendet, um einen Vektor mit beliebigen Werten in eine Wahrscheinlichkeitsverteilung zu verwandeln und dabei den höchsten Wert hervorzuheben.

Beispiel:

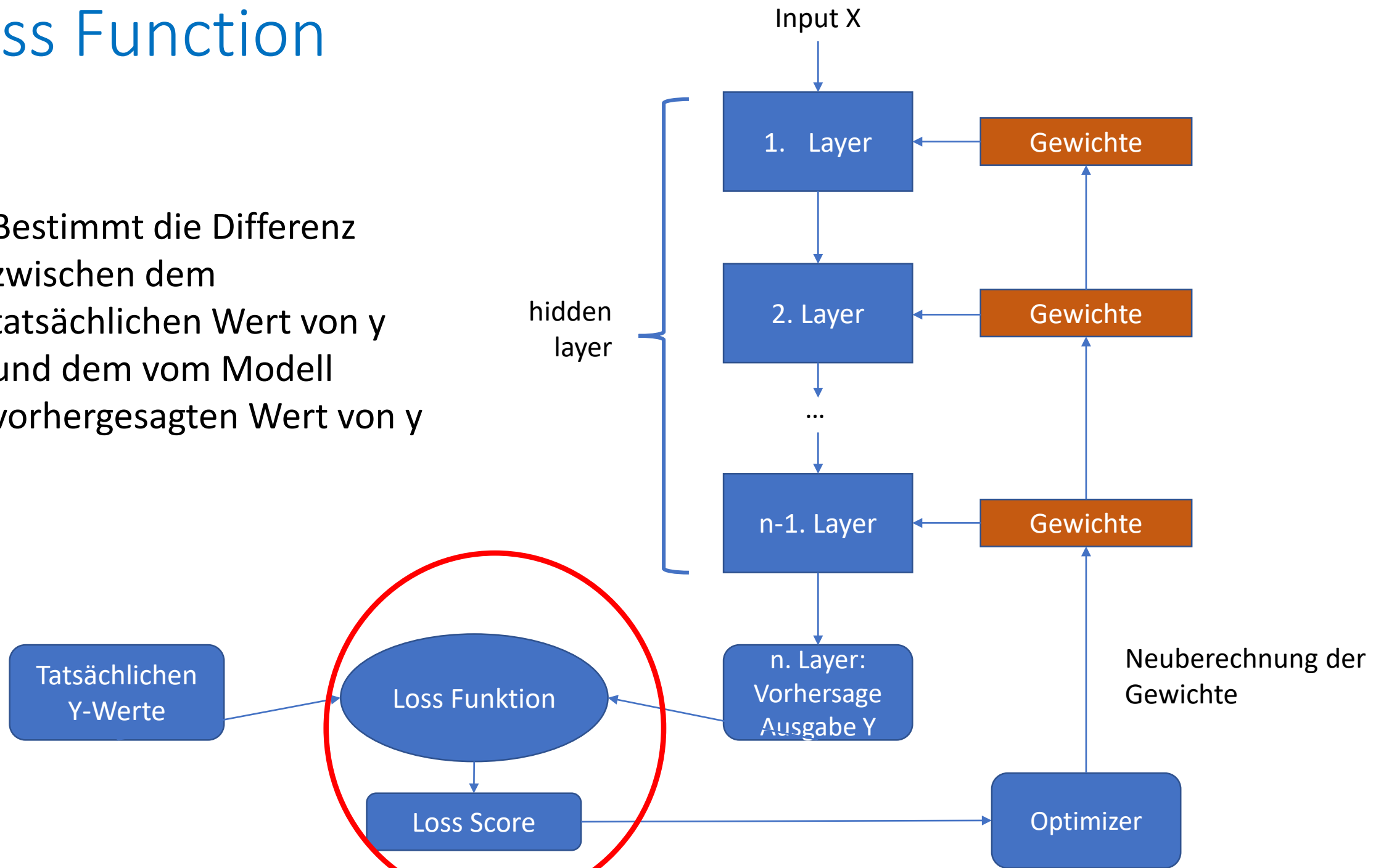
$$z = [1, 2, 3, 4, 1, 2, 3]$$

$$s(z) = [0.024, 0.064, , 0.175, 0.475, 0.024, 0.064, 0.175]$$

wird verstanden als  $p(y=1 \mid x)$

# Loss Function

Bestimmt die Differenz zwischen dem tatsächlichen Wert von  $y$  und dem vom Modell vorhergesagten Wert von  $y$





# Loss Function (aka ,Cost Function', ,Object Function')

$$C(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2.$$

Beispiel für eine Loss Function:  
Mean Squared Error (MSE)

w: Gewichte des Netzwerks

b: bias

x: Eingabewert

a: Ausgabe des Netzwerks

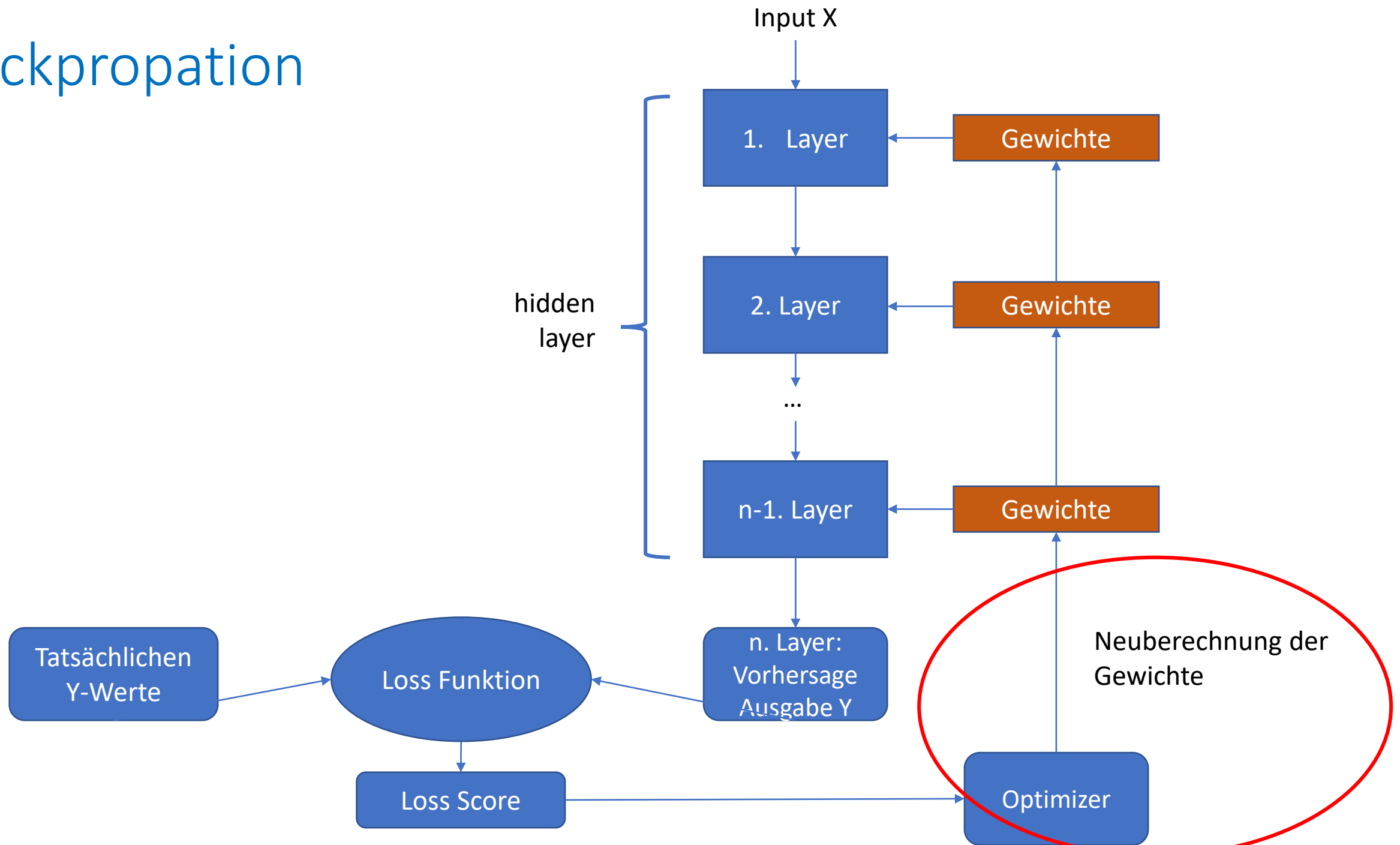
n: Anzahl der Trainingswerte, also Anzahl von (x, y) Paaren

C geht gegen 0, wenn die Vorhersage des Netzwerks den wahren Werten sehr nah sind

Wenn C groß ist, dann ist die Vorhersage weit entfernt von den wahren Werten.

**Ziel des Trainings ist es also Werte für w und b zu finden, so dass C (für die Trainingsdaten) möglichst klein wird.**

# Backpropagation

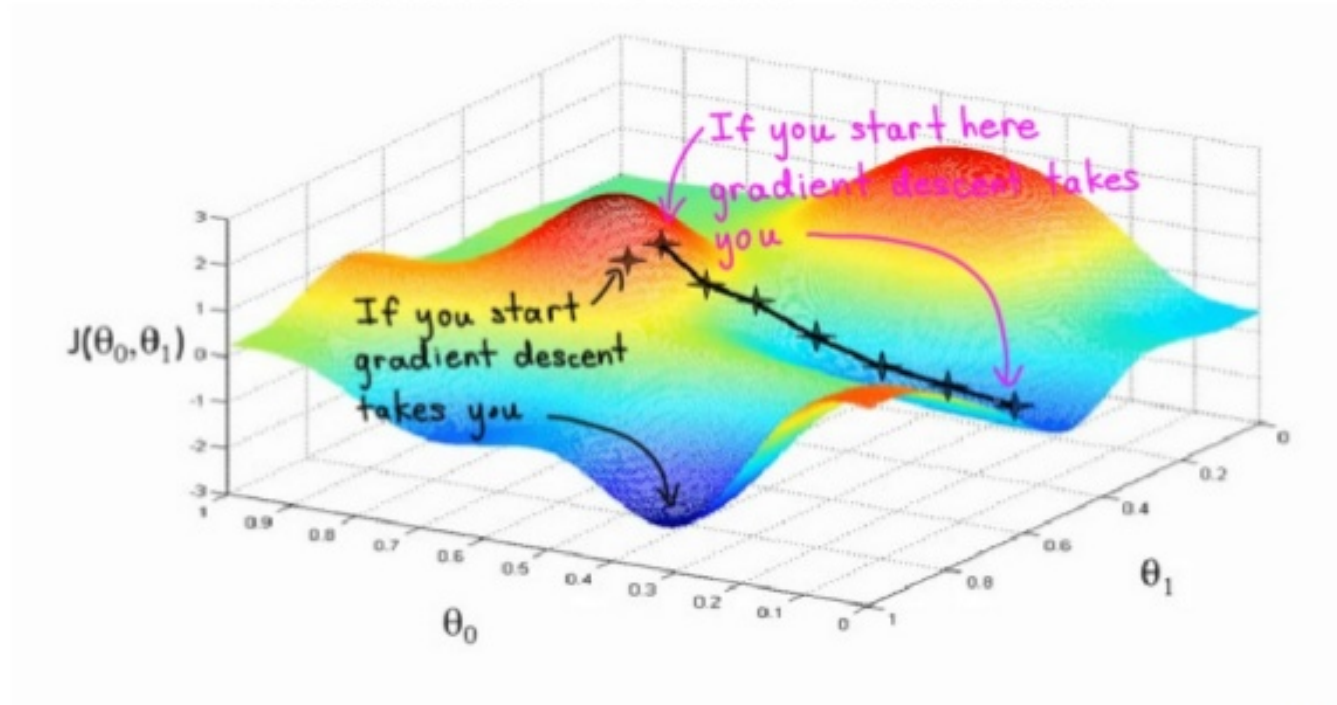


# Backpropation

- Aufgabe: Minimum der Loss-Funktion finden
- Finden eines Minimums: Optimierungsfunktion
- Beispiel Gradient descent

# Gradient Descent

## Gradient Descent - Intuition



Backpropagation employs Local Gradient Descent for its optimization technique.

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$
$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

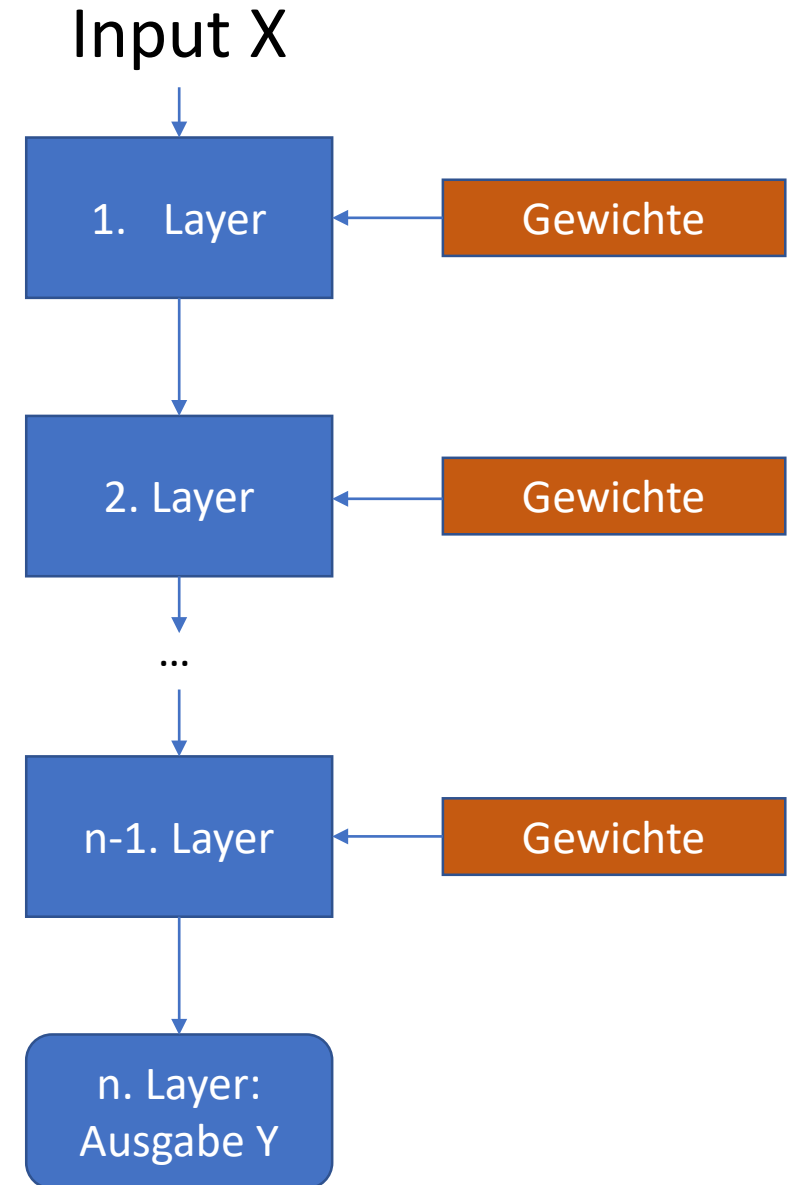
# Backpropagation

- Im Zuge der Backpropagation werden die Gewichte aktualisiert.
- Grundlage dafür ist das Signal über die Differenz von vorhergesagtem  $\hat{y}$  und realem  $y$ . Die partiellen Ableitung berechnen von hinten nach vorne die Steigung an den jeweiligen Knoten und ,machen jeweils einen kleinen Schritt Richtung Minimum‘.
- Problem sind lokale Minima, da dort das Lernen stoppt.

# Architekturen

mit Beispielen in Keras

# Simple Feedforward Network





## Home

Keras: The Python Deep Learning library

You have just found Keras.

Guiding principles

Getting started: 30 seconds to Keras

Installation

Switching from TensorFlow to CNTK or Theano

Support

Why this name, Keras?

Why use Keras

Getting started

Guide to the Sequential model

Guide to the Functional API

FAQ

Models

About Keras models

Sequential

Model (functional API)

Layers

About Keras layers

Core Layers

Convolutional Layers

Pooling Layers

## Keras: The Python Deep Learning library



### You have just found Keras.

Keras is a high-level neural networks API, written in Python and capable of running on top of **TensorFlow**, **CNTK**, or **Theano**. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*

Use Keras if you need a deep learning library that:

- Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).
- Supports both convolutional networks and recurrent networks, as well as combinations of the two.
- Runs seamlessly on CPU and GPU.

Read the documentation at **Keras.io**.

Keras is compatible with: **Python 2.7-3.6**.

### Guiding principles

- **User friendliness.** Keras is an API designed for human beings, not machines. It puts user experience front and center. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon



# An open-source software library for Machine Intelligence

[GET STARTED](#)

## Eager Execution

We're announcing eager execution, an imperative, define-by-run interface to TensorFlow. Check out the [README](#) to get started today.

[LEARN MORE](#)

## TensorFlow Dev Summit 2018

Save the date for the 2018 TensorFlow Dev Summit! Space is limited – sign up now for further updates.

[LEARN MORE](#)

## Announcing TensorFlow Lite

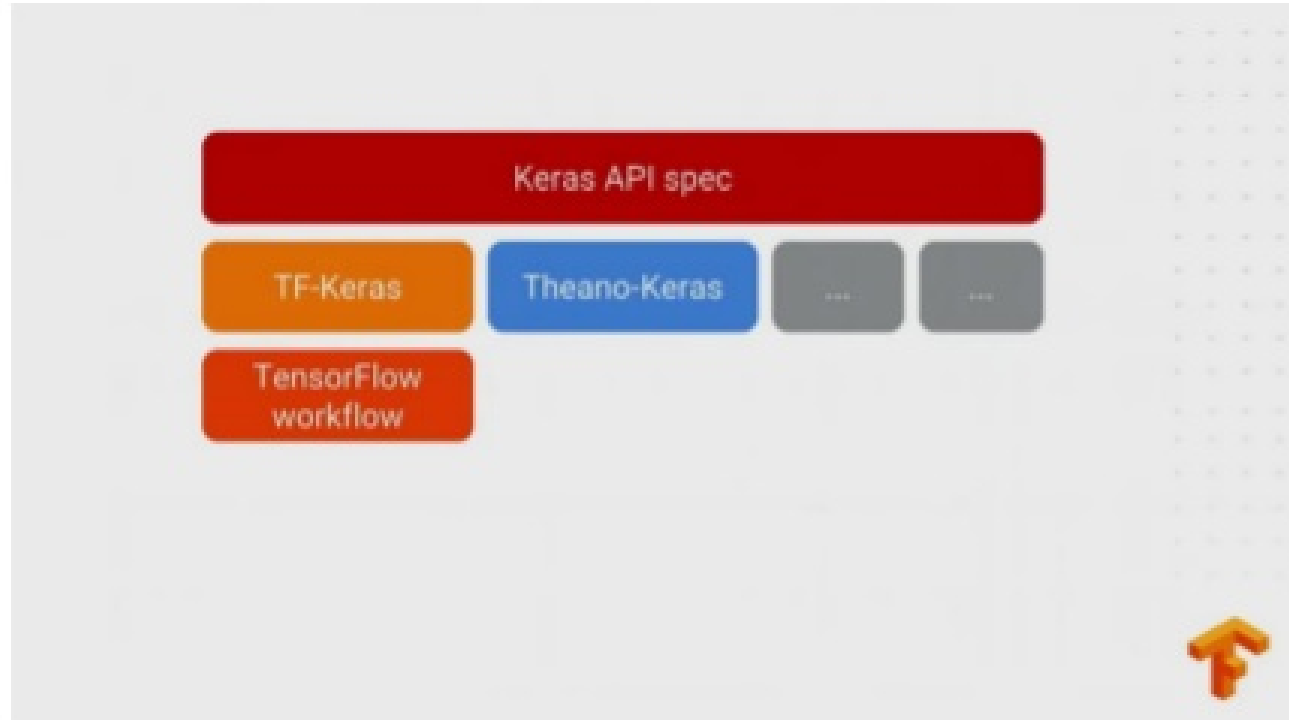
Learn more about TensorFlow's lightweight solution for mobile and embedded devices.

[LEARN MORE](#)

## About TensorFlow

# Keras

- An API spec for building deep learning models across many platforms



```
# load data
```

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
# reshaping image data
```

```
train_images = train_images.reshape((60000, 28*28))
```

```
train_images = train_images.astype('float32') / 255
```

```
test_images = test_images.reshape((10000, 28*28))
```

```
test_images = test_images.astype('float32') / 255
```

```
# preparing the labels
```

```
train_labels = to_categorical(train_labels)
```

```
test_labels = to_categorical(test_labels)
```

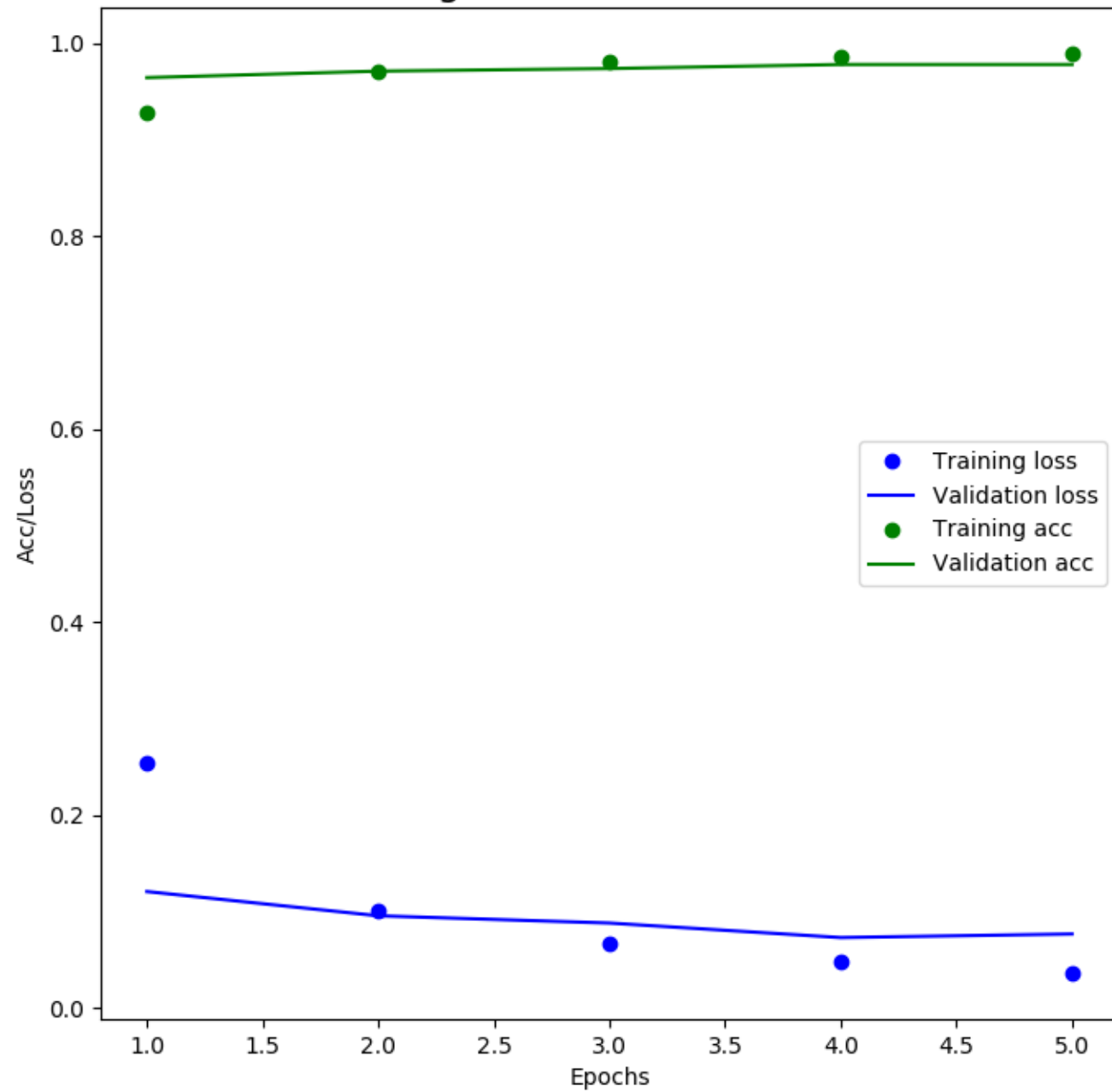
```
# defining the layers
model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))

# defining loss function, optimizer, and metric.
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Running the network
history = model.fit(train_images, train_labels,
                    epochs=5, batch_size=128,
                    validation_data=(test_images,
test_labels))
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
print(loss_and_metrics)
```

```
(dl) D:\mydata\Dropbox\uni\Vorträge\2018 Würzburg - Deep Learning Workshop>python mnist.py
Using TensorFlow backend.
training images shape: (60000, 28, 28)
Number of training samples: 60000
test images shape: (10000, 28, 28)
Number of test samples: 10000
Epoch 1/5
2018-01-10 00:02:38.669347: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\35\tensorflow\tensorflow\ops\compiled_to_use: AVX AVX2
2018-01-10 00:02:38.947273: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\35\tensorflow\tensorflow\ops\compiled_to_use: AVX AVX2
name: GeForce GTX 1080 major: 6 minor: 1 memoryClockRate(GHz): 1.8225
pciBusID: 0000:01:00.0
totalMemory: 8.00GiB freeMemory: 6.66GiB
2018-01-10 00:02:38.947408: I C:\tf_jenkins\home\workspace\rel-win\M\windows-gpu\PY\35\tensorflow\tensorflow\ops\compiled_to_use: AVX AVX2
GeForce GTX 1080, pci bus id: 0000:01:00.0, compute capability: 6.1)
60000/60000 [=====] - 6s 96us/step - loss: 0.2593 - acc: 0.9250
Epoch 2/5
60000/60000 [=====] - 5s 75us/step - loss: 0.1053 - acc: 0.9692
Epoch 3/5
60000/60000 [=====] - 4s 74us/step - loss: 0.0693 - acc: 0.9790
Epoch 4/5
60000/60000 [=====] - 4s 71us/step - loss: 0.0503 - acc: 0.9847
Epoch 5/5
60000/60000 [=====] - 4s 71us/step - loss: 0.0377 - acc: 0.9887
10000/10000 [=====] - 1s 127us/step
test accuracy: 0.9804
```

MNIST with a simple Feedforward Network  
Training and validation acc/loss

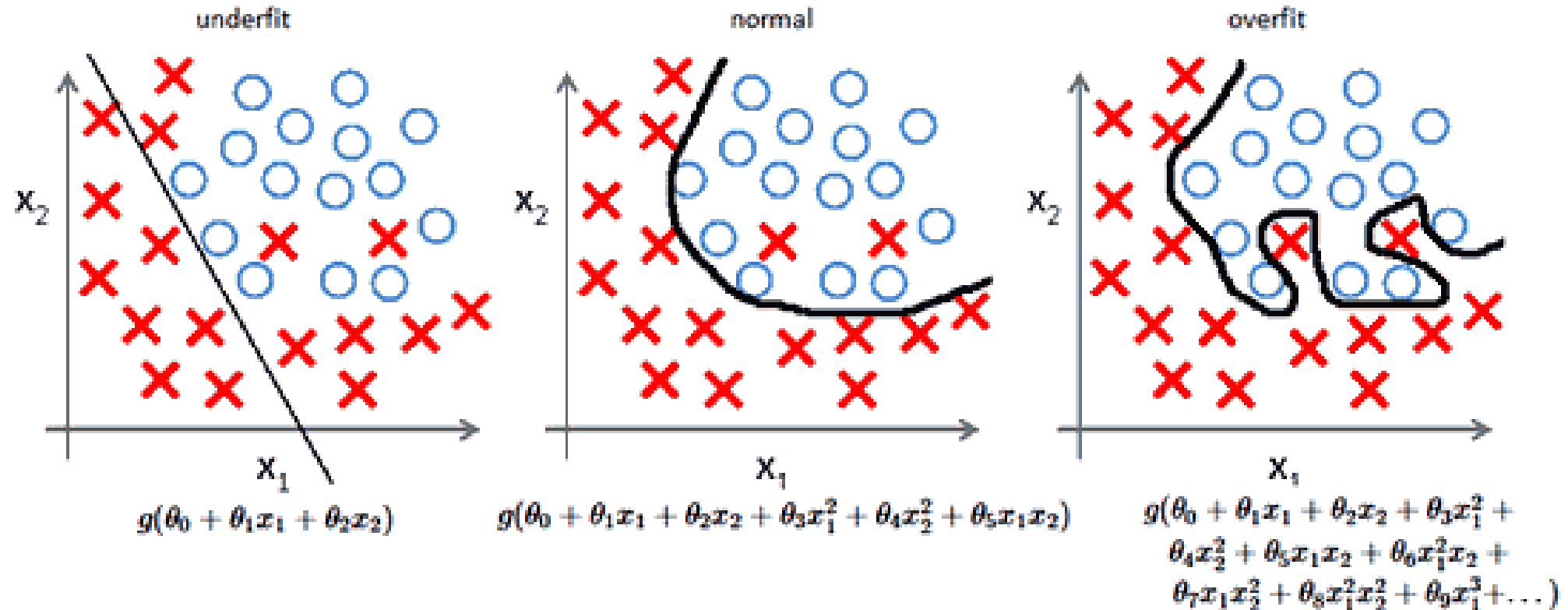


# Accuracy: Trainings- und Testwerte auswerten

- Wenn das Trainingsset sehr viel bessere Werte hat als das Testset:  
Wahrscheinlich overfitting -> Regularisierung
- Wenn das Trainingsset und das Testset schlechte Werte hat:  
Wahrscheinlich underfitting -> Komplexeres Netz versuchen, länger trainieren

# Overfitting

Overfitting: Das Modell gibt die Eigenschaften der Trainingsdaten zu gut wieder. Problem: Übertragbarkeit auf neue Daten nicht gut.



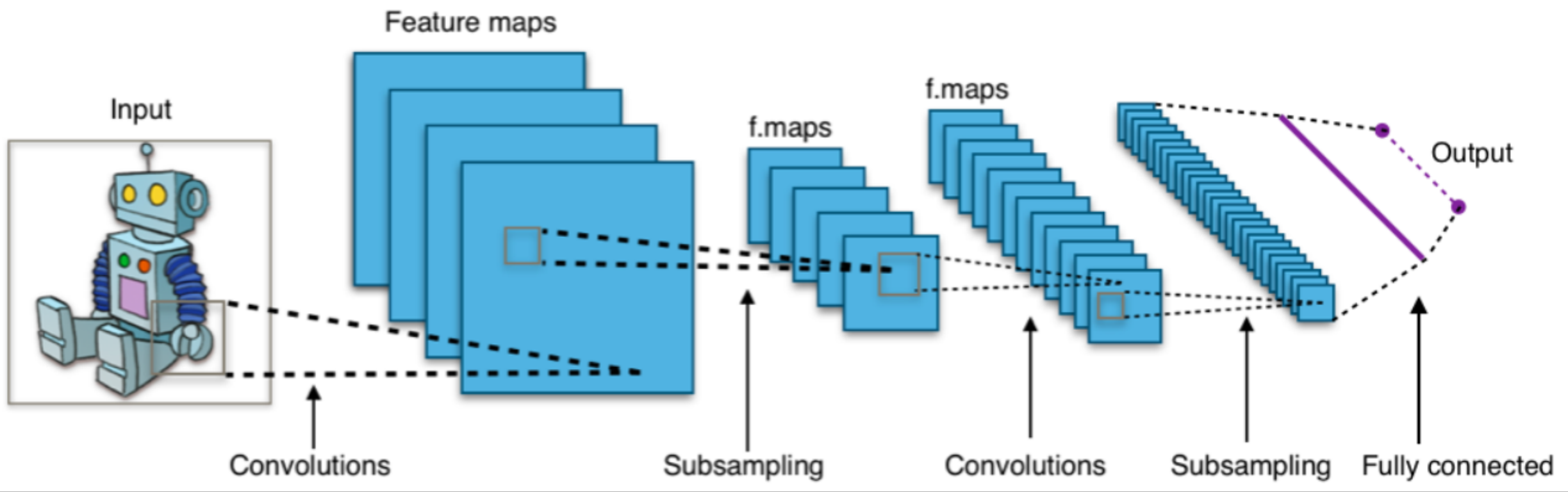


# Regularisierung

- Regularisierung der Gewichte der Loss Function: Große Gewichte werden ‚bestraft‘ (In Keras z.B. L1 und L2: Regularisierung)  
`model.add(layers.Dense(512, kernel_regularizer=regularizers.l2(0.001),  
activation='relu', input_shape=(28*28,))`
- Dropout: Während des Trainings lässt man einige Output-Features zufällig eines Layers wegfallen (dropout), d.h. sie werden auf 0 gesetzt.  
`model.add(layers.Dropout(0.5))`

# Convolutional Network

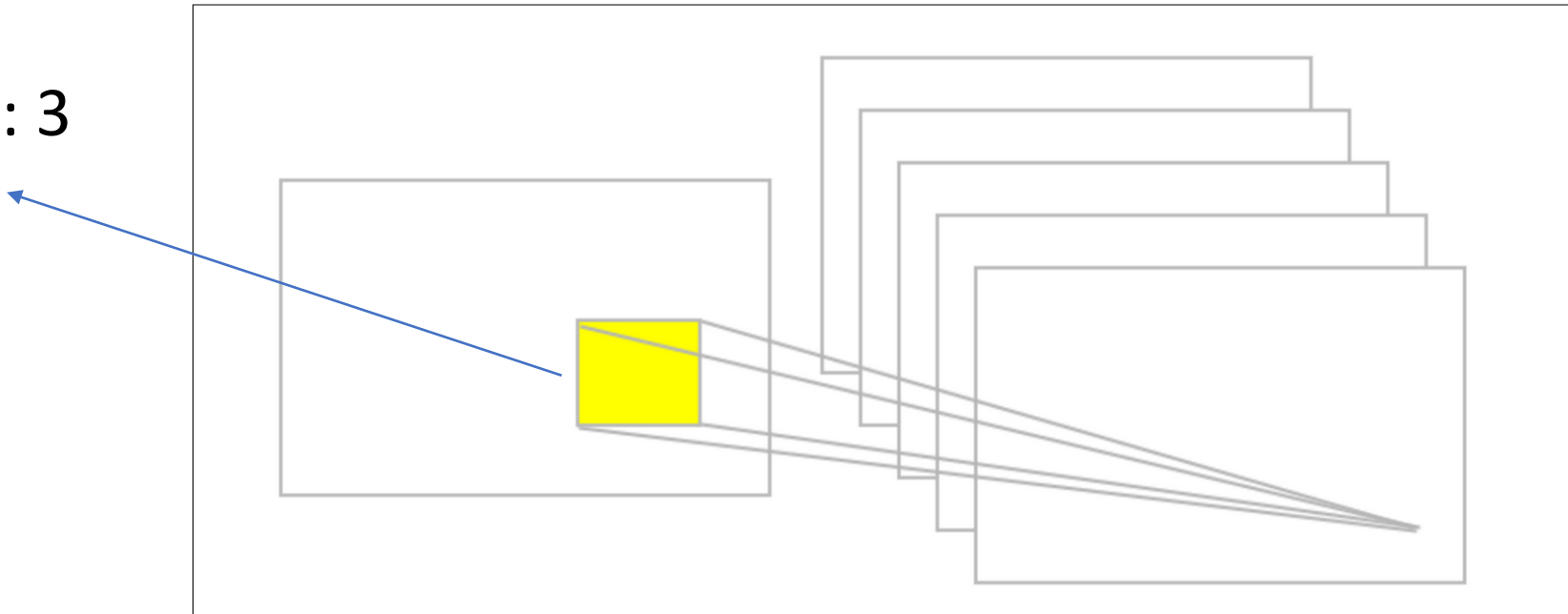
- Werden vor allem für die Signal-Verarbeitung, insbesondere Bilder verwendet.
- Sehr erfolgreich.
- Vergleichsweise schnell.



# Conv nets in Keras

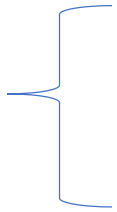
```
model.add(Conv2D(32, kernel_size=3, input_shape=(256, 256, 3)))
```

Seitenlänge: 3



# Maxpooling

poolsize

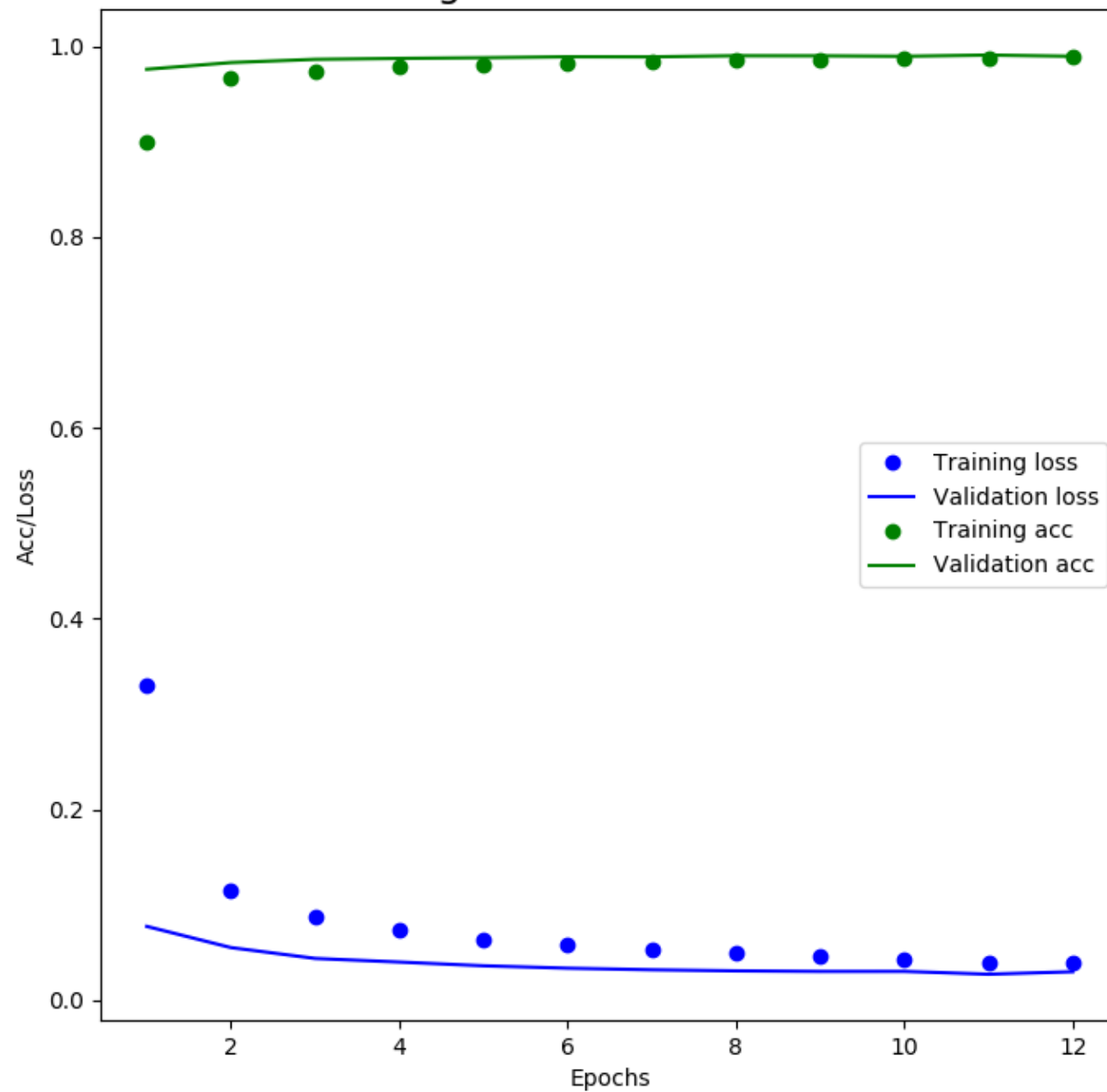


1	0	3	6
2	4	5	2
2	6	2	0
3	4	1	7

4	6
6	7

```
model = Sequential()  
model.add(Conv2D(32, kernel_size=(3, 3),  
                activation='relu',  
                input_shape=input_shape))  
model.add(Conv2D(64, (3, 3), activation='relu'))  
model.add(MaxPooling2D(pool_size=(2, 2)))  
model.add(Dropout(0.25))  
model.add(Flatten())  
model.add(Dense(128, activation='relu'))  
model.add(Dropout(0.5))  
model.add(Dense(num_classes, activation='softmax'))
```

MNIST with a Convolutional Network  
Training and validation acc/loss

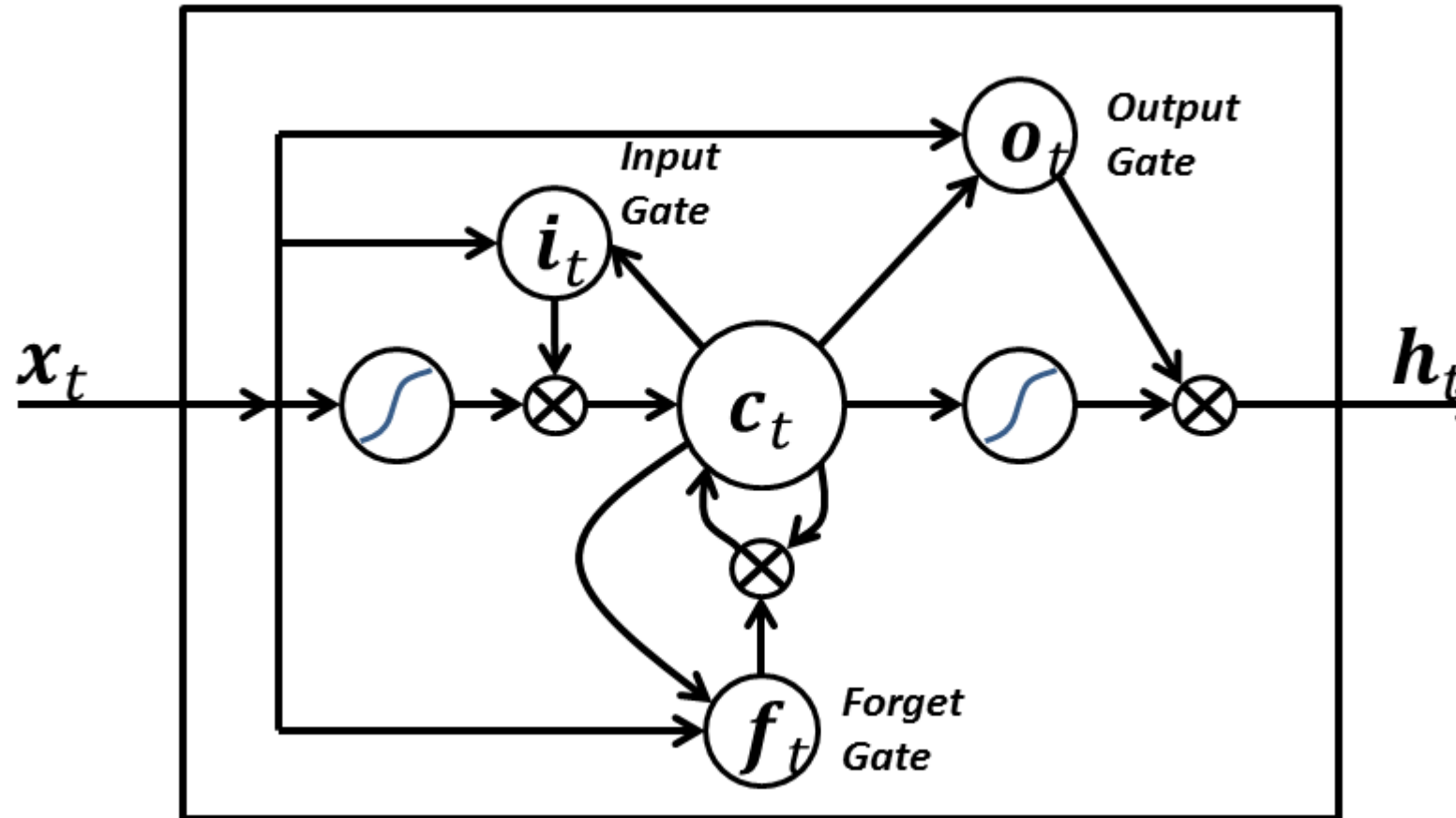


# Recurrent Network: LSTM

- Dienen der Verarbeitung von sequentiellen Daten
- LSTM: Long short term memory
- Werden z.B. sehr erfolgreich im NLP-Bereich eingesetzt



- a **cell**: erinnert Werte über längere Zeit
- an **input gate**, kontrolliert Einfluß neuer Werte
- an **output gate**: kontrolliert Ausgabewert
- a **forget gate**: kontrolliert, wie Werte behalten werden



Grober Aufbau eines LSTM-Moduls mit der inneren Zelle im Zentrum. Die  $\otimes$ -Symbole repräsentieren hier den Faltungsoperator. Die großen Kreise mit S-artiger Kurve sind die Sigmoidfunktionen. Die Pfeile, die von der Zelle jeweils zu den Gates zeigen, sind die Gucklochinformationen vom letzten Durchlauf.

```
max_features = 20000
maxlen = 80 # cut texts after this number of words (among top
max_features most common words)
batch_size = 32

print('Loading data...')
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)
print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')

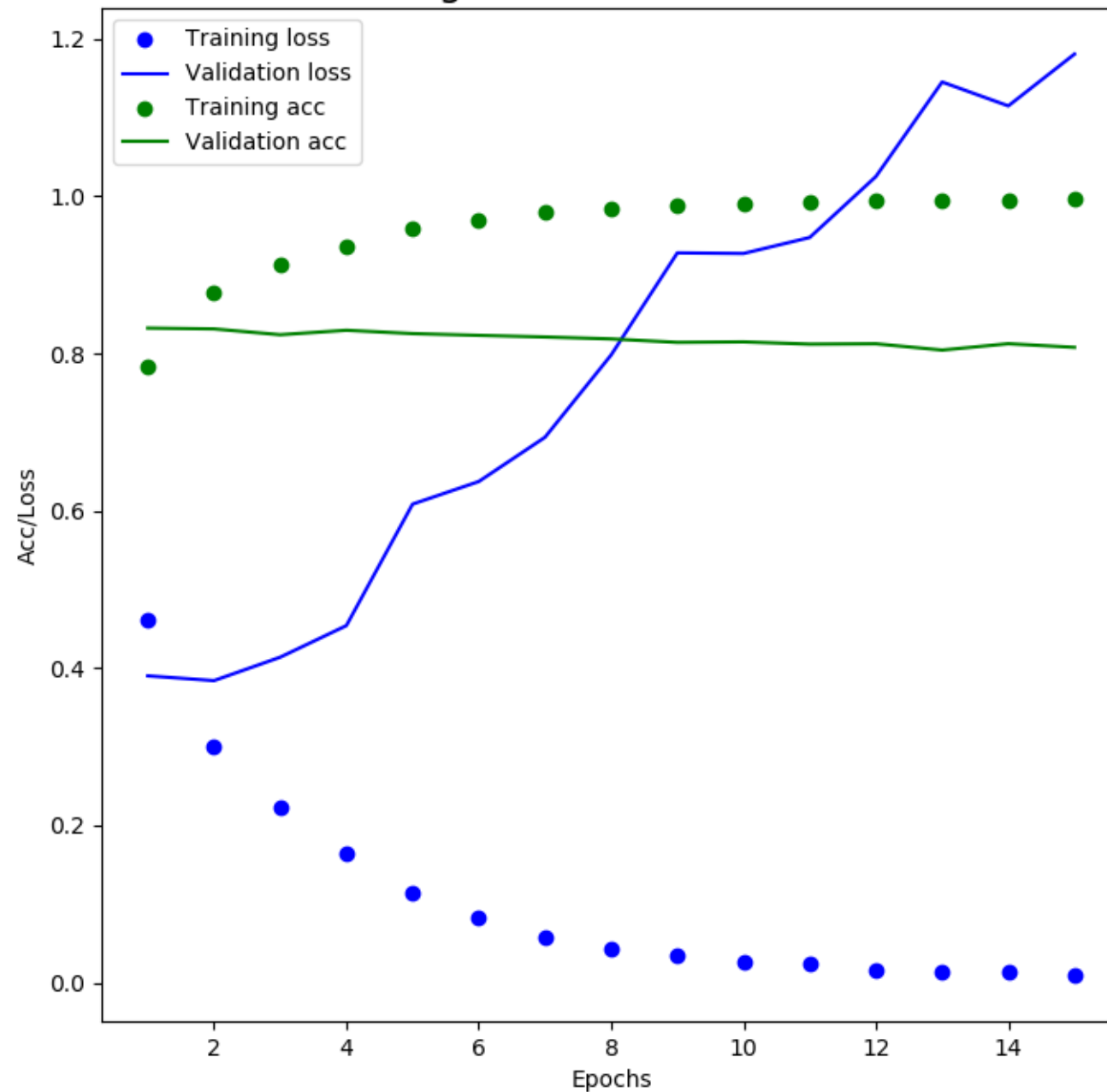
print('Pad sequences (samples x time)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)
```

```
model = Sequential()
model.add(Embedding(max_features, 128))
model.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='sigmoid'))

# try using different optimizers and different optimizer configs
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

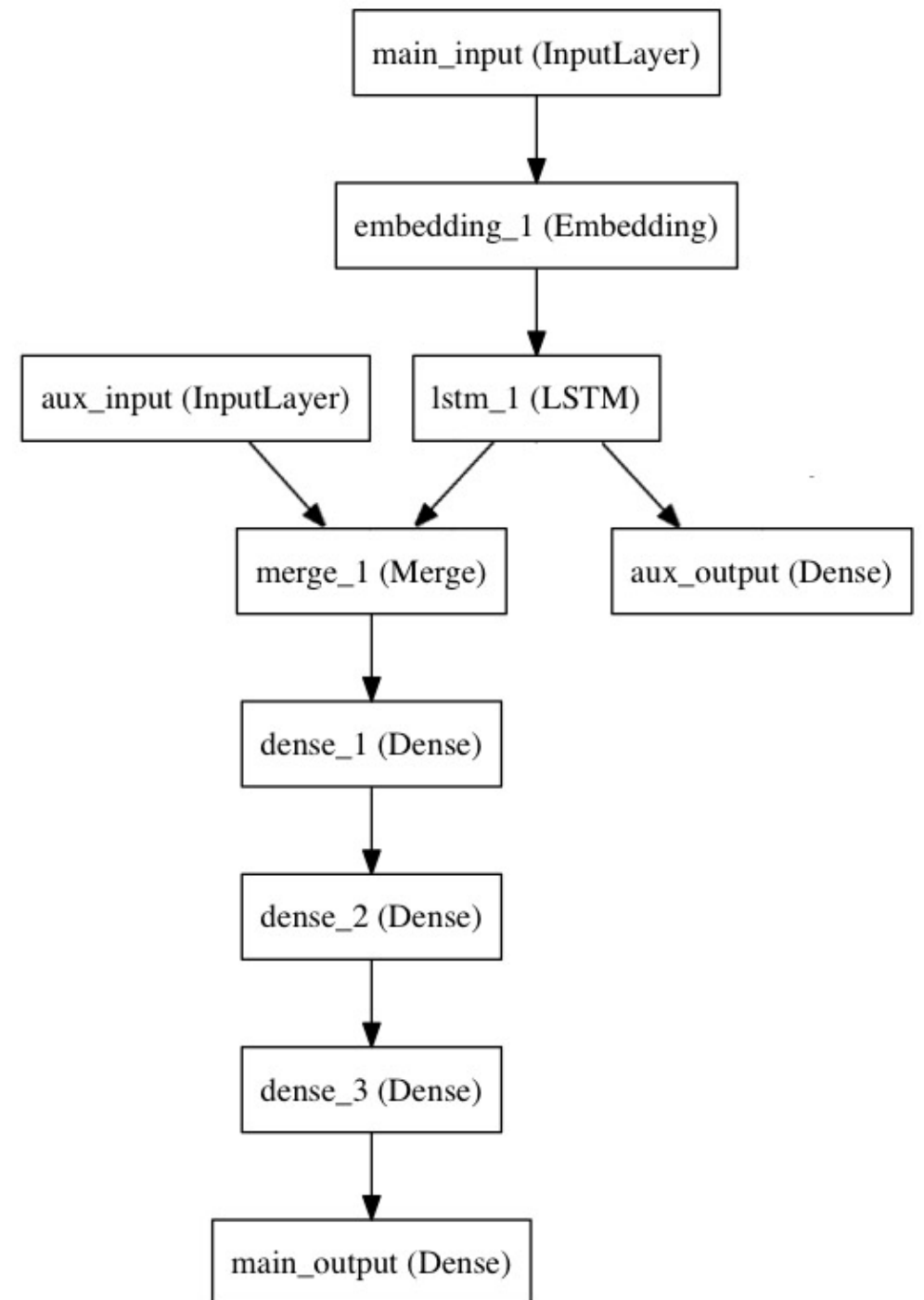
print('Train...')
csv_logger = CSVLogger('log.csv', append=True, separator=';')
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=15,
                    validation_data=(x_test, y_test),
                    callbacks=[csv_logger])
score, acc = model.evaluate(x_test, y_test,
                             batch_size=batch_size)
```

IMDB sentiment classification with a LSTM network  
Training and validation acc/loss



# Keras: Functional API

- Erlaubt die Beschreibung komplexerer Modelle
  - multi-input models
  - multi-output models,
  - directed acyclic graphs,
  - models with shared layers.



# Unser einfaches Feedforward Modell

```
# defining the layers
inputs = Input(shape=(28 * 28,))
x = Dense(512, activation='relu')(inputs)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

# Keras erweitern

- Eigene Layer definieren (siehe: *Writing your own Keras layers*)
- Tensorflow API und Keras API Anweisungen können gemeinsam genutzt werden

# Ausblick 1

- Autoencoder / -decoder
- Generative Adversarial Networks
- Reinforcement Learning



# Ausblick 2: Argumente gegen DL

- Deep learning thus far is data hungry
- Deep learning thus far is shallow and has limited capacity for transfer
- Deep learning thus far has no natural way to deal with hierarchical structure
- Deep learning thus far has struggled with open-ended inference
- Deep learning thus far is not sufficiently transparent
- Deep learning thus far has not been well integrated with prior knowledge
- Deep learning thus far cannot inherently distinguish causation from correlation
- Deep learning presumes a largely stable world, in ways that may be problematic
- Deep learning thus far works well as an approximation, but its answers often cannot be fully trusted
- Deep learning thus far is difficult to engineer with