Einführung in Neuronale Netzwerke

TensorFlow Kurs

Neuronale Netzwerke

In dieser Vortragsreihe werden wichtige theoretische Aspekte behandelt:

- Neuronen und Aktivierungsfunktionen (**Neurons and Activation Functions**)
- Kostenfunktionen (**Cost Functions**)
- Gradientenverfahren (**Gradient Descent**)
- Rückführung (**Backpropagation**)

Neuronale Netzwerke

- Sobald wir ein allgemeines Verständnis aufgebaut haben, werden wir all diese Themen selbst mit Python implementieren, ohne die Hilfe einer unterstützenden Deep Learning Bibliothek.
- Danach werden wir TensorFlow einsetzten!

Neuronale Netzwerke

- Wenn du einen groben Überblick über diese Schlüsselelemente hast, dann wird es viel einfacher zu verstehen, was passiert, wenn wir mit TensorFlow beginnen.
- Tensorflow hat in seiner Syntax direkte Verbindungen zu diesen Konzepten.

Lass uns loslegen!

Einführung in das Perceptron

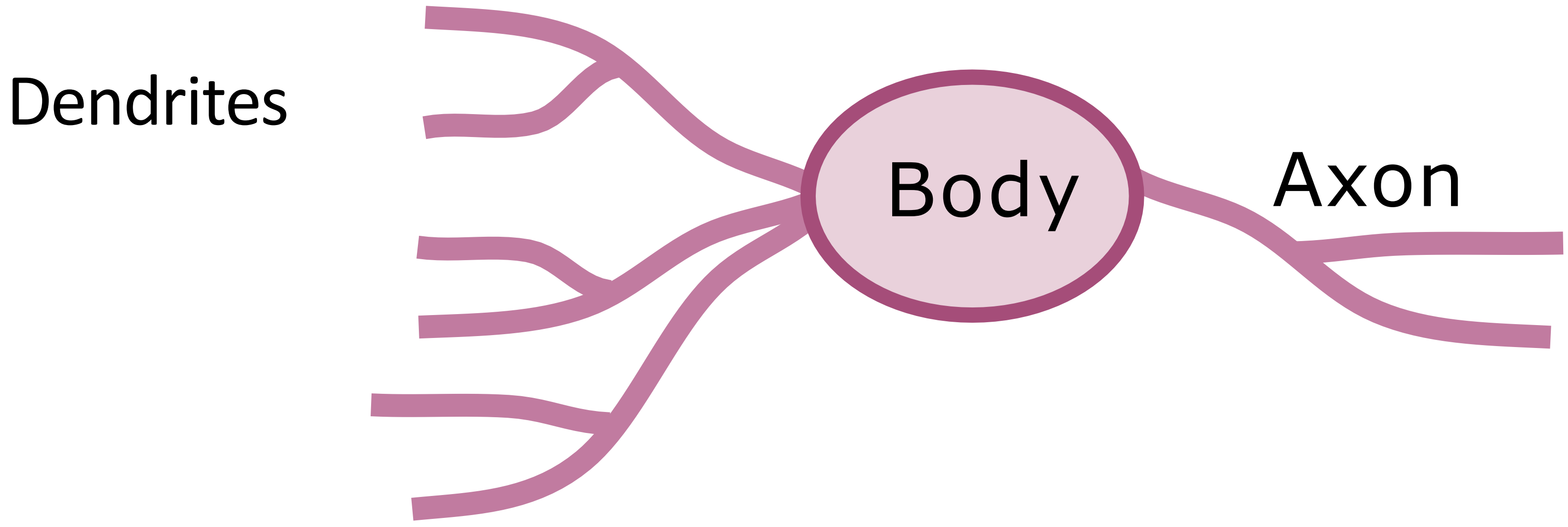
Neuronale Netzwerke

- Bevor wir direkt in neuronale Netze einsteigen, müssen wir zunächst die einzelnen Komponenten, wie z.B. ein einzelnes "Neuron", verstehen.

Neuronale Netzwerke

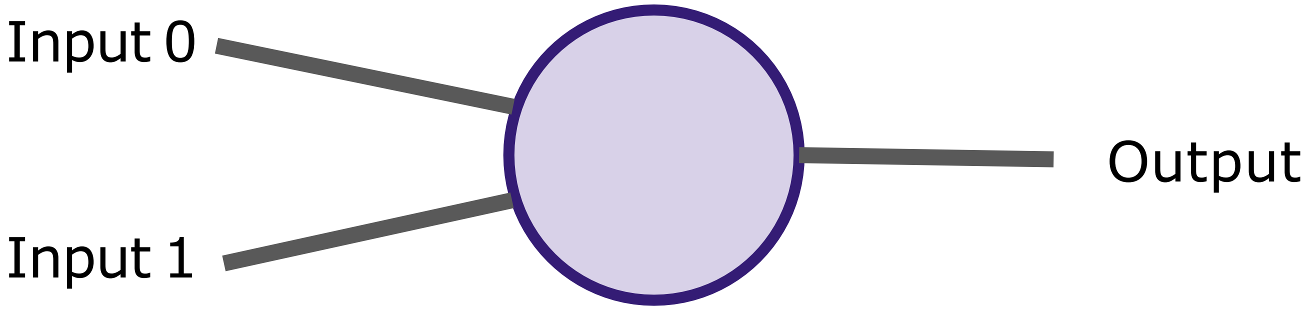
- Künstliche Neuronale Netzwerke (eng. Artificial Neural Networks (ANN)) haben tatsächlich eine Basis in der Biologie
- Mal sehen, wie wir biologische Neuronen mit einem künstlichen Neuron, dem so genannten Perzeptron, nachbilden können .

Das biologische Neuron



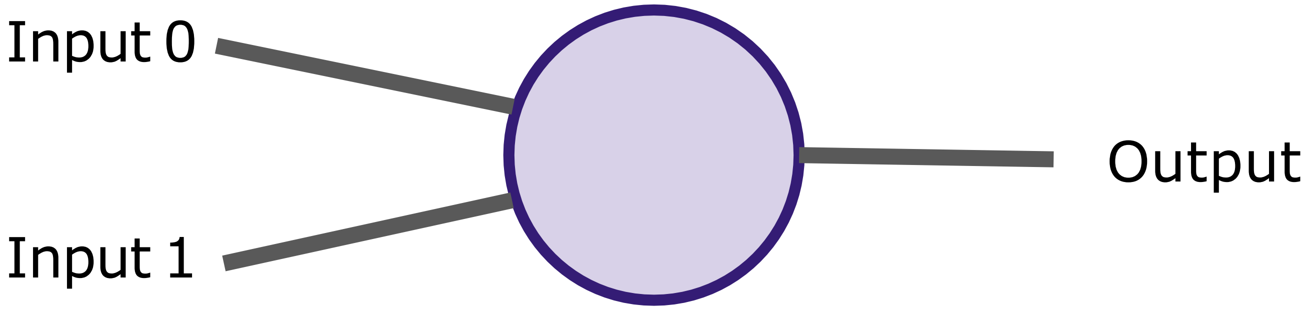
Neuronale Netzwerke

- Das künstliche Neuron hat auch Ein- und Ausgänge.



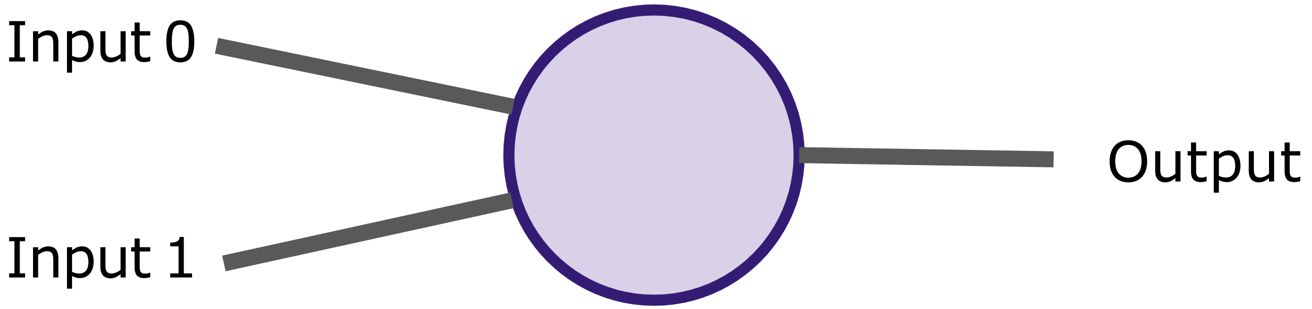
Neuronale Netzwerke

- Dieses einfache Modell wird als Perzeptron bezeichnet.



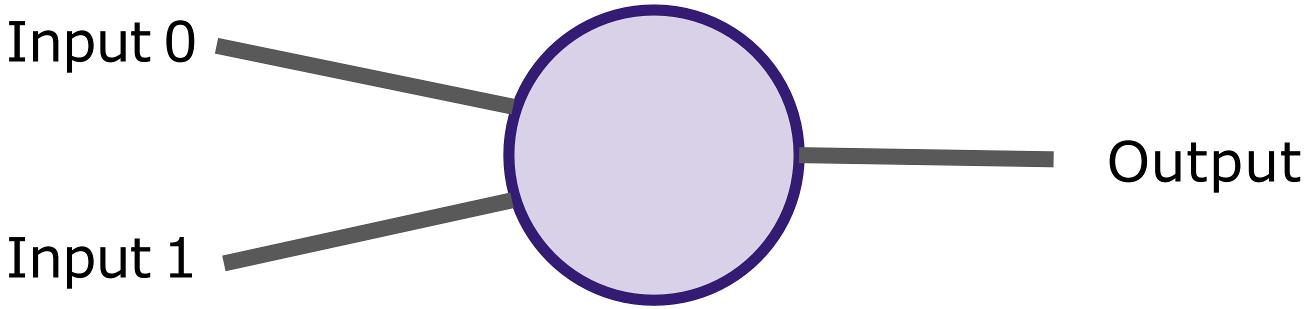
Neuronale Netzwerke

- Einfaches Beispiel, wie es funktioniert.



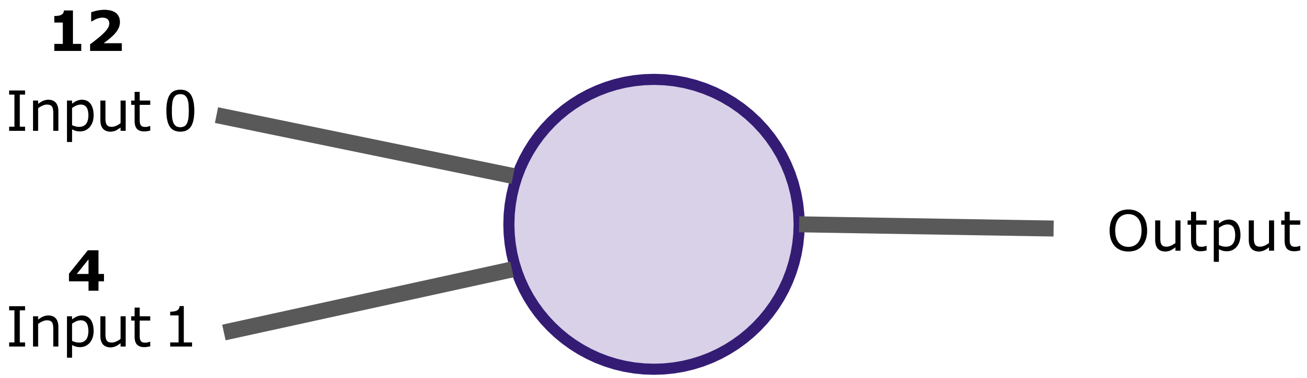
Neuronale Netzwerke

- Wir haben zwei Eingänge und einen Ausgang



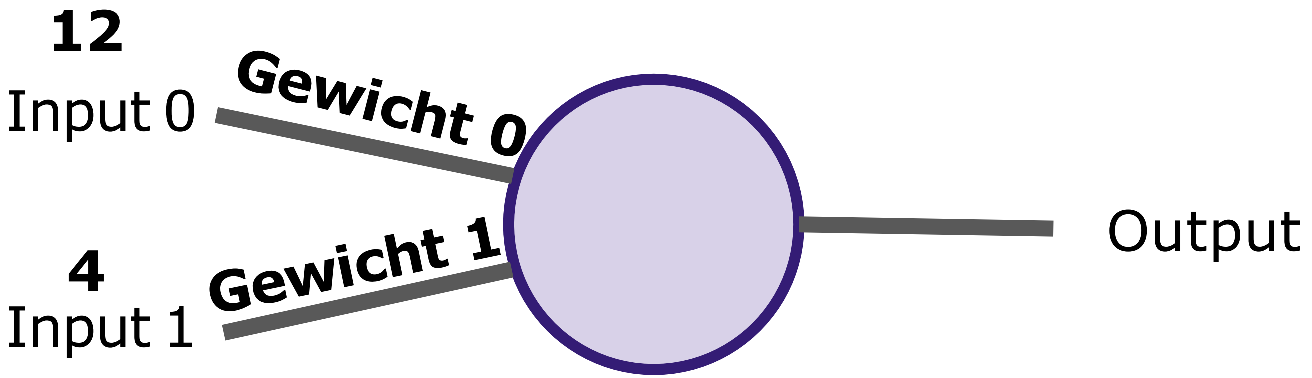
Neuronale Netzwerke

- Die Eingaben sind Werte von Features



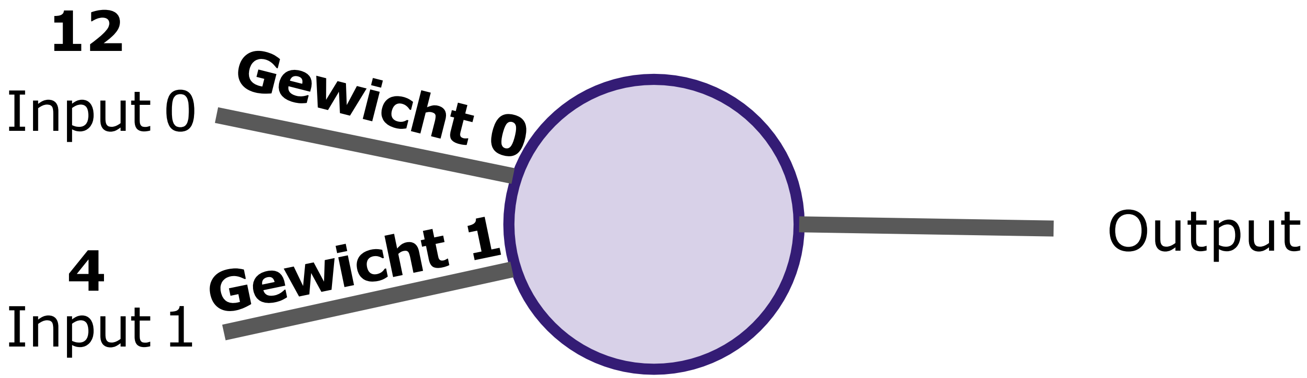
Neuronale Netzwerke

- Eingaben werden mit einem Gewicht multipliziert.



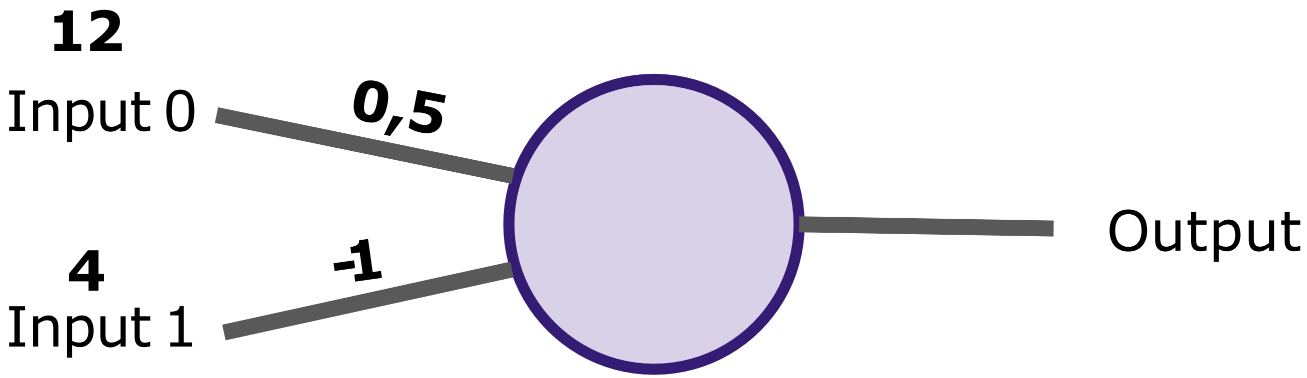
Neuronale Netzwerke

- Gewichte beginnen zunächst als Zufallsgewichte



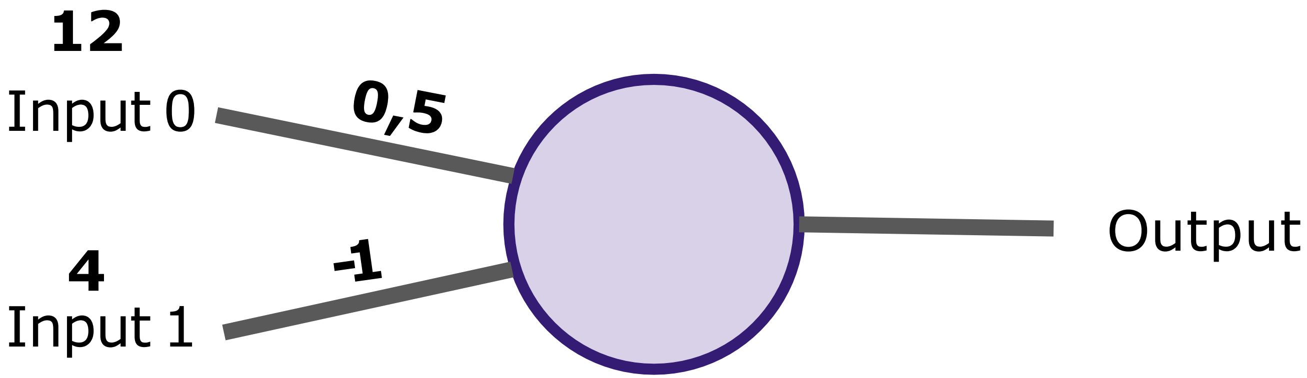
Neuronale Netzwerke

- Gewichte beginnen zunächst als Zufallsgewichte



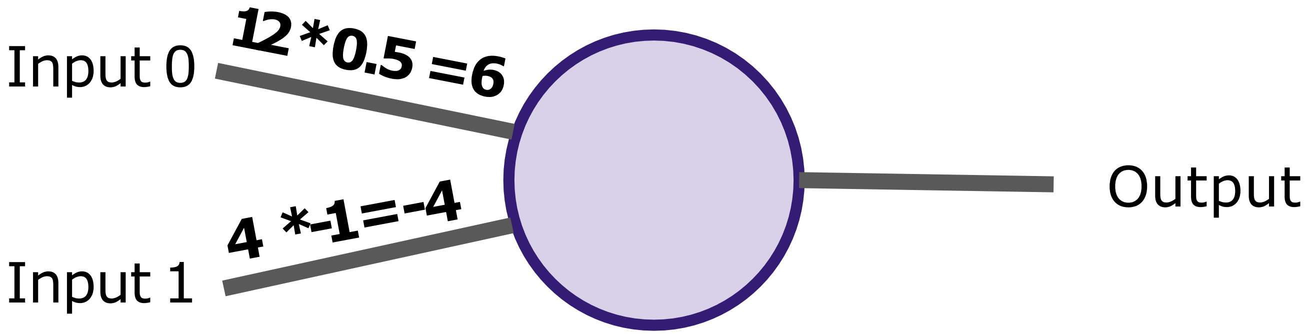
Neuronale Netzwerke

- Eingaben werden nun mit Gewichten multipliziert.



Neuronale Netzwerke

- Eingaben werden nun mit Gewichten multipliziert.



Neuronale Netzwerke

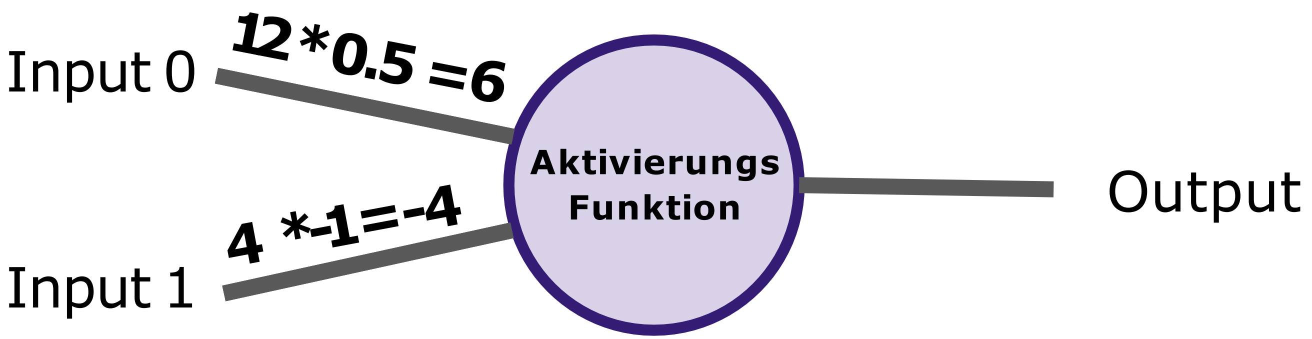
- Diese Ergebnisse werden dann an eine Aktivierungsfunktion übergeben.

Neuronale Netzwerke

- Hier gibt es viele Aktivierungsfunktionen zur Auswahl, auf die wir später noch näher eingehen werden.

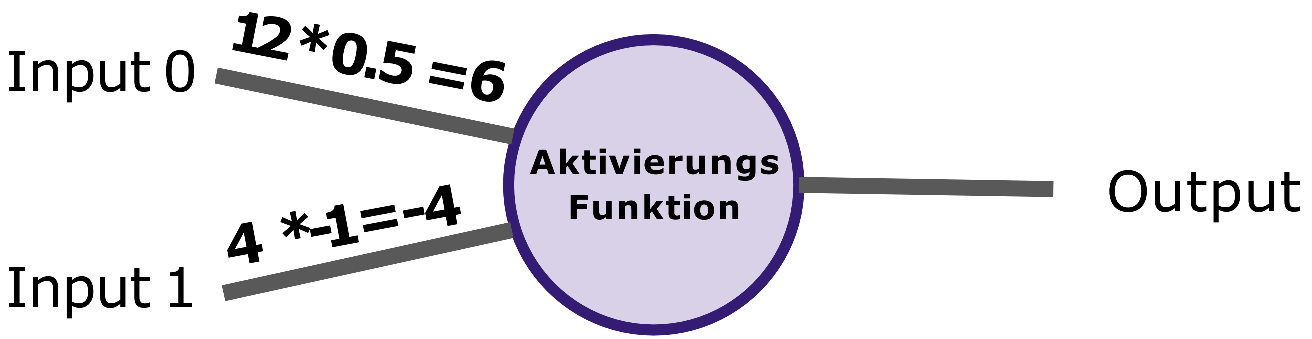
Neuronale Netzwerke

- Im Moment wird unsere Aktivierungsfunktion sehr einfach sein....



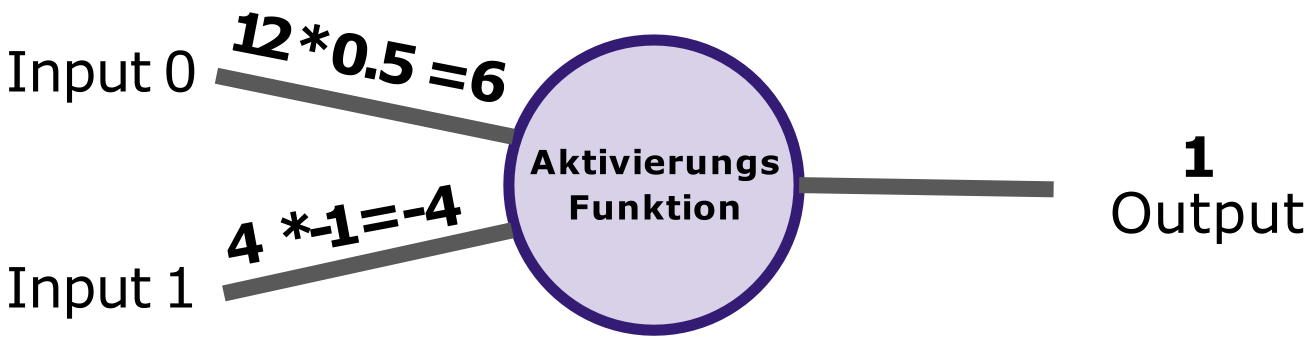
Neuronale Netzwerke

- Wenn die Summe der Eingänge positiv ist, wird 1 zurückgegeben, wenn die Summe negativ ist, wird 0 ausgegeben.



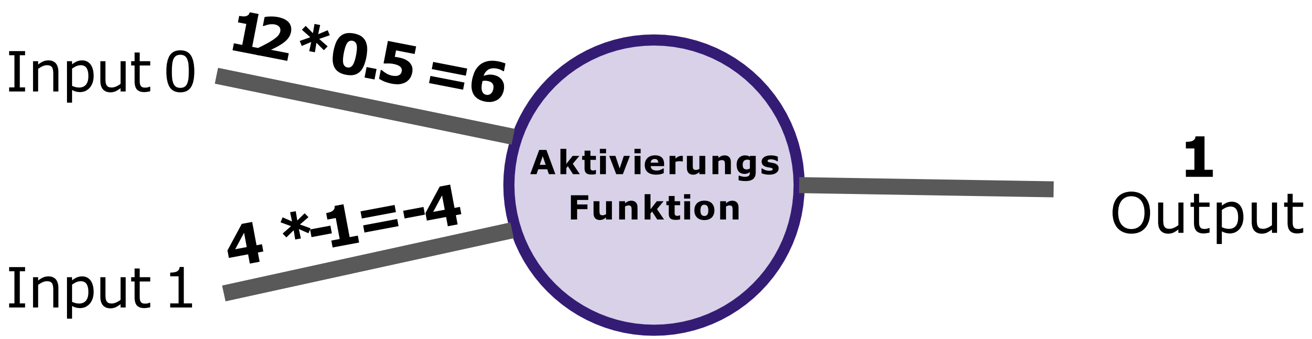
Neuronale Netzwerke

- In diesem Fall 6-4=2, also gibt die Aktivierungsfunktion 1 zurück.



Neuronale Netzwerke

- Es gibt ein mögliches Problem. Was ist, wenn die ursprünglichen Eingaben mit Null beginnen?

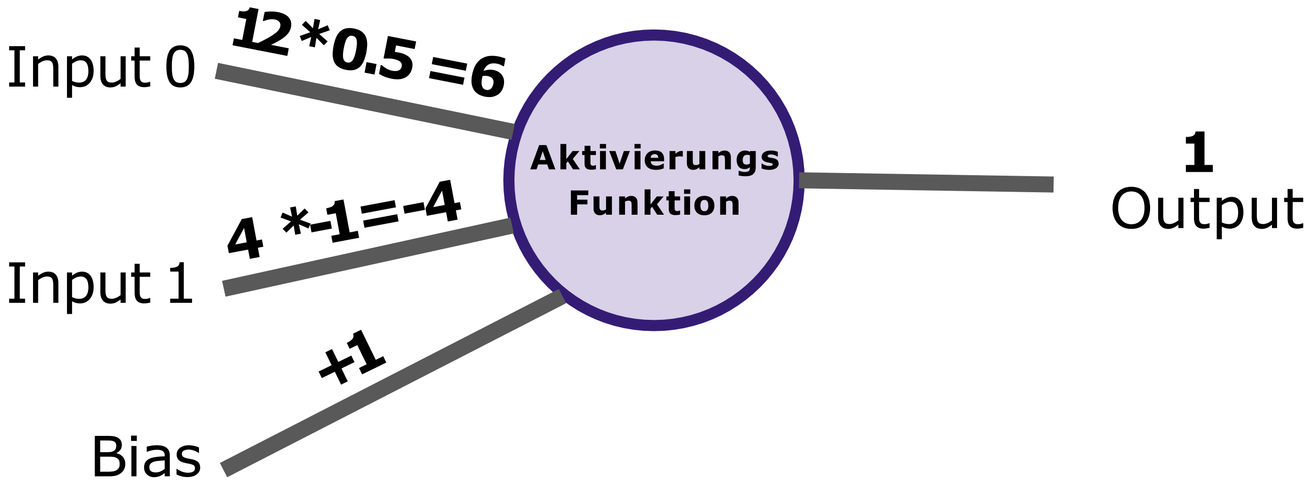


Neuronale Netzwerke

- Dann würde jedes Gewicht, multipliziert mit der Eingabe, immer noch Null ergeben.

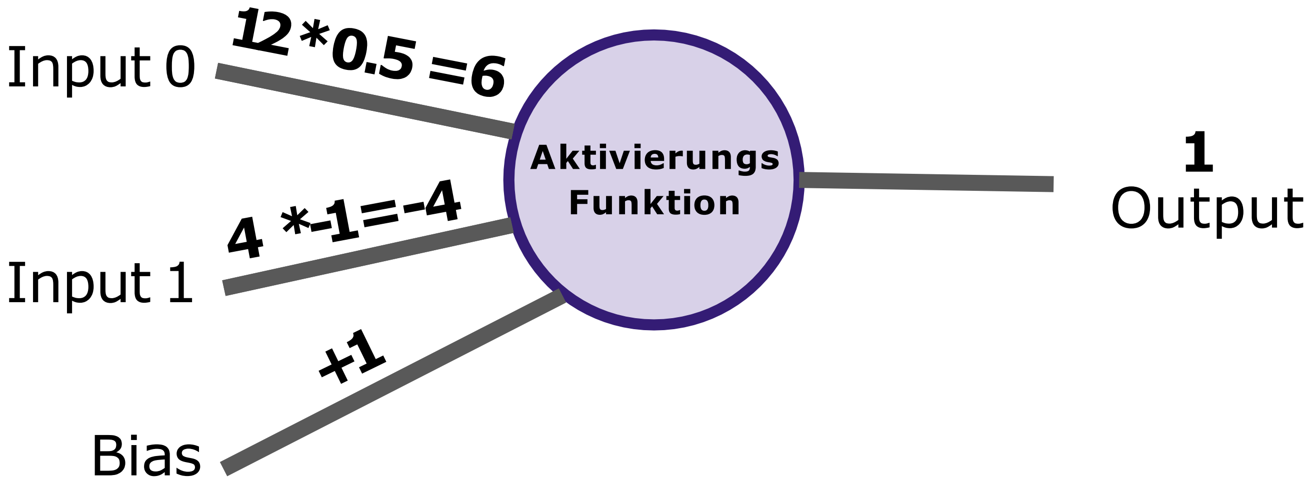
Neuronale Netzwerke

- Wir beheben dies, indem wir einen Bias-Begriff (Verzerrung) hinzufügen, in diesem Fall wählen wir 1.



Neuronale Netzwerke

- Wie sieht das jetzt mathematisch aus?



Neuronale Netzwerke

- Lass uns schnell darüber nachdenken, wie wir dieses Perzeptronmodell mathematisch darstellen können:

$$\sum_{i=0}^{n} w_i x_i + b$$

Neuronale Netzwerke

- Sobald wir viele Perzeptrone in einem Netzwerk haben, werden wir sehen, wie wir diese leicht zu einer Matrixform erweitern können!

$$\sum_{i=0}^{n} w_i x_i + b$$

Wiederholung

- Biologisches Neuron
- Perzeptron-Modell
- Mathematische Darstellung

Einführung in Neuronale Netzwerke

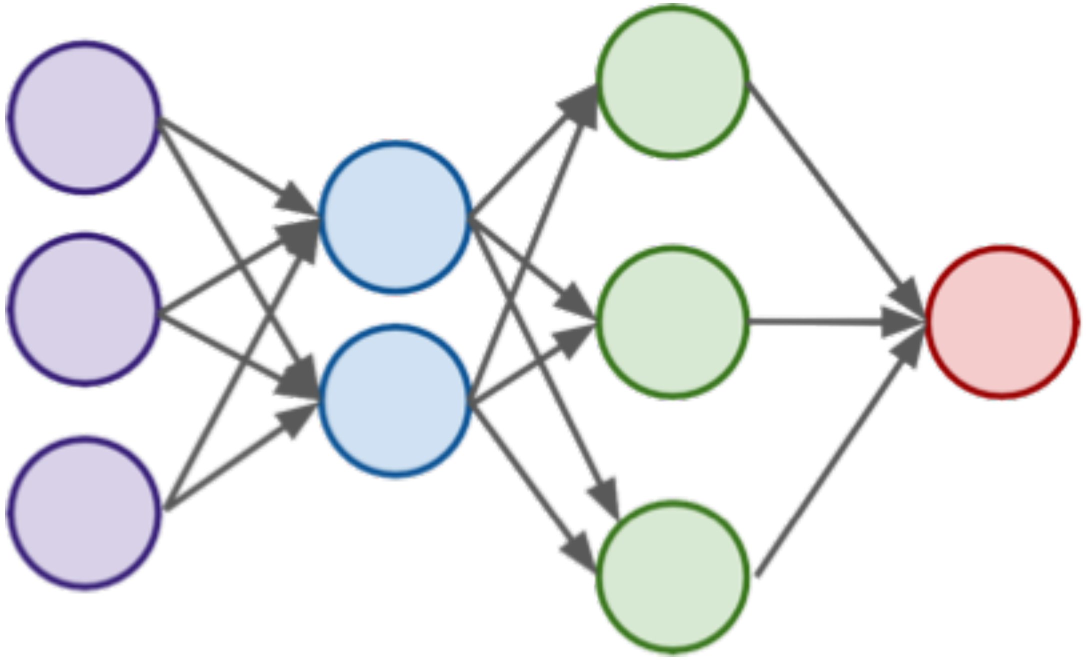
Neuronale Netzwerke

- Wir haben gesehen, wie sich ein einzelnes Perzeptron verhält, nun wollen wir dieses Konzept auf die Idee eines neuronalen Netzwerkes erweitern!

- Lass uns nun anschauen, wie man viele Perzeptrone miteinander verbindet und diese dann mathematisch darstellt.

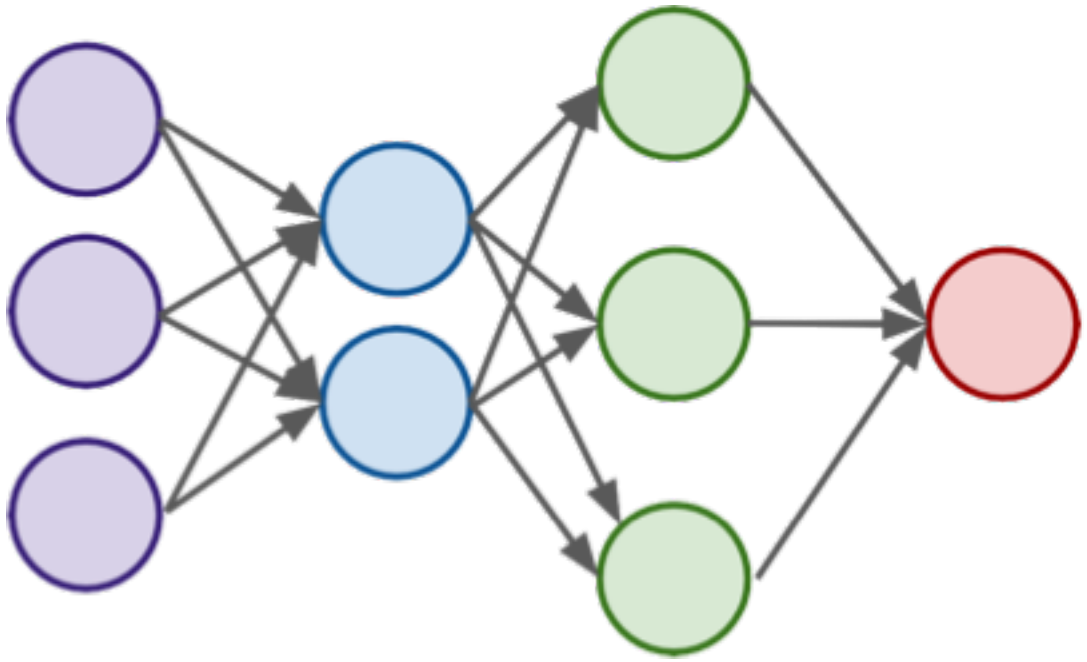
Neuronale Netzwerke

- Mehrlagiges Perzeptronen Netzwerk (Eng.: Multiple Perceptrons Network)



Neuronale Netzwerke

- Input Layer. 2 versteckte (Hidden) Layer. Ausgabe (Output) Layer



Neuronale Netzwerke

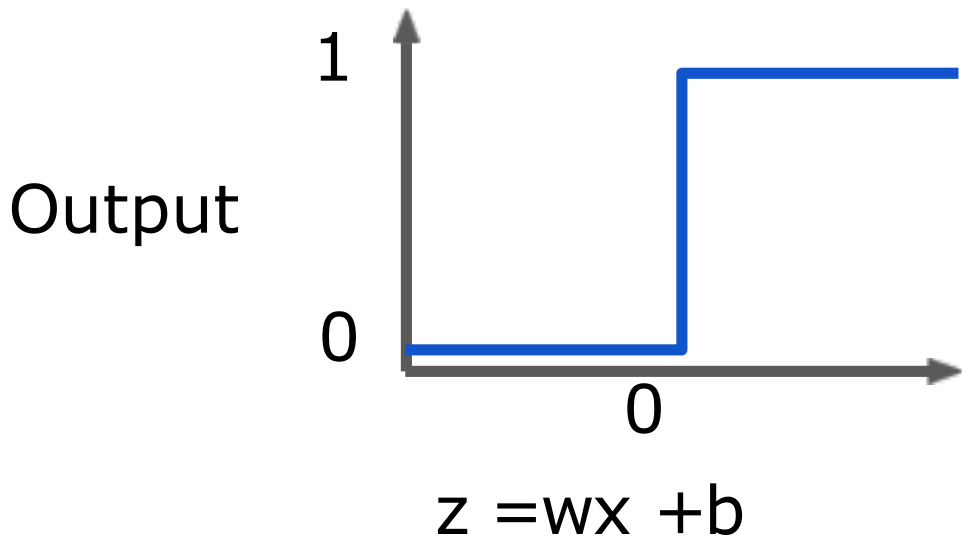
- Input Layers
  - Reale Werte aus den Daten
- Hidden Layers
  - Ebenen zwischen Eingang und Ausgang
  - 3 oder mehr Schichten sind “Deep Network“
- Output Layer
  - Endgültige Schätzung der Ausgabe

Neuronale Netzwerke

- Wenn du durch mehrere Ebenen vorwärts gehst, steigt der Grad der Abstraktion.
- Lass uns nun etwas ausführlicher auf die Aktivierungsfunktion eingehen.

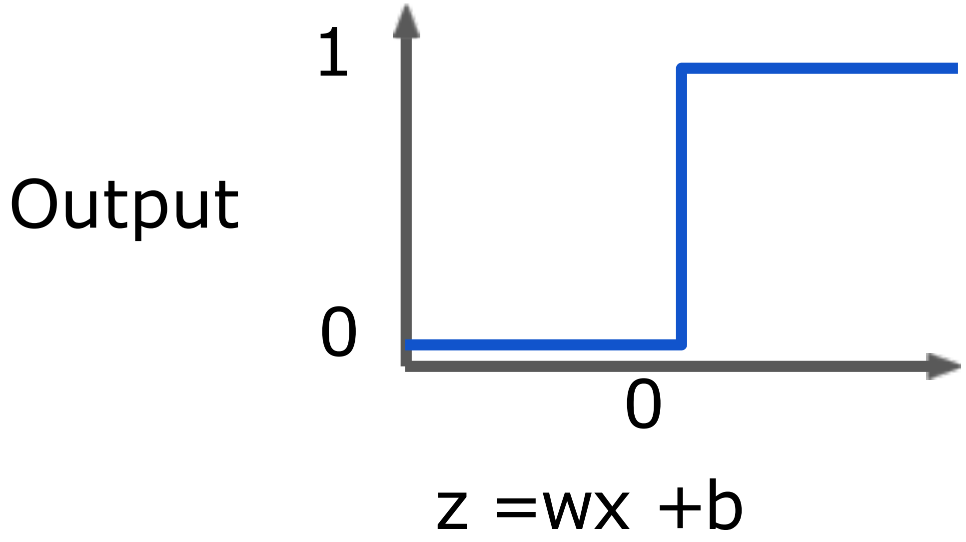
Aktivierungsfunktion

- Bisher war unsere Aktivierungsfunktion nur eine einfache Funktion, die 0 oder 1 ausgibt.



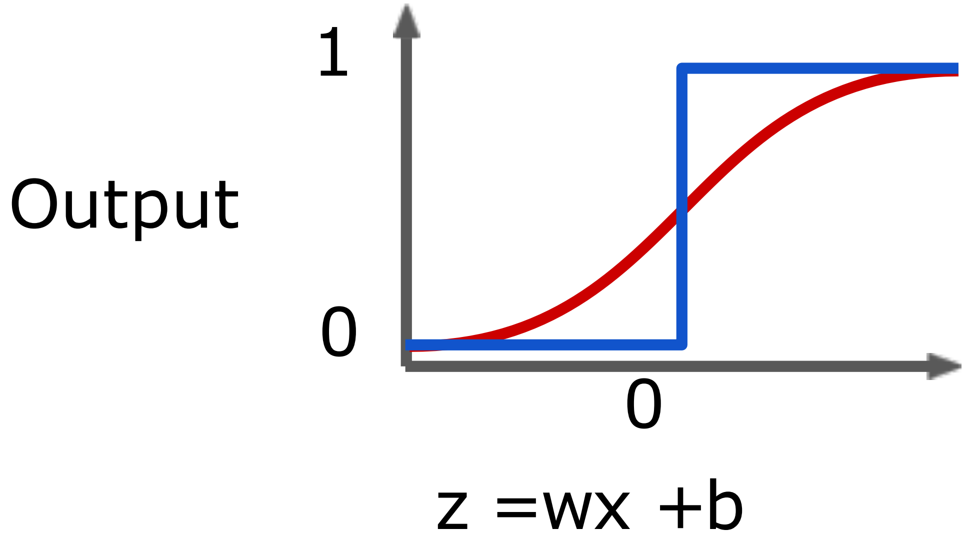
Aktivierungsfunktion

- Dies ist eine ziemlich dramatische Funktion, da kleine Änderungen nicht berücksichtigt werden.



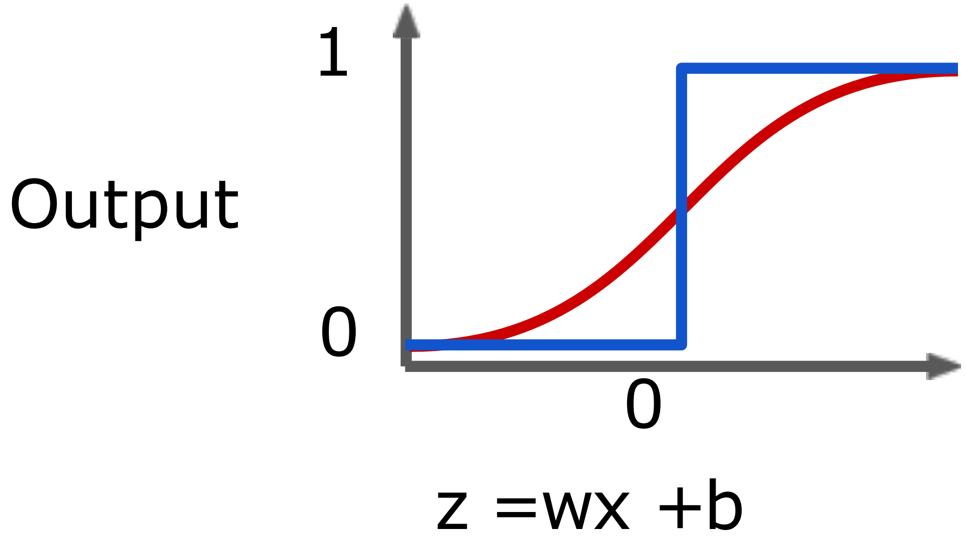
Aktivierungsfunktion

- Es wäre schön, wenn wir eine dynamischere Funktion hätten, zum Beispiel die rote Linie.



Aktivierungsfunktion

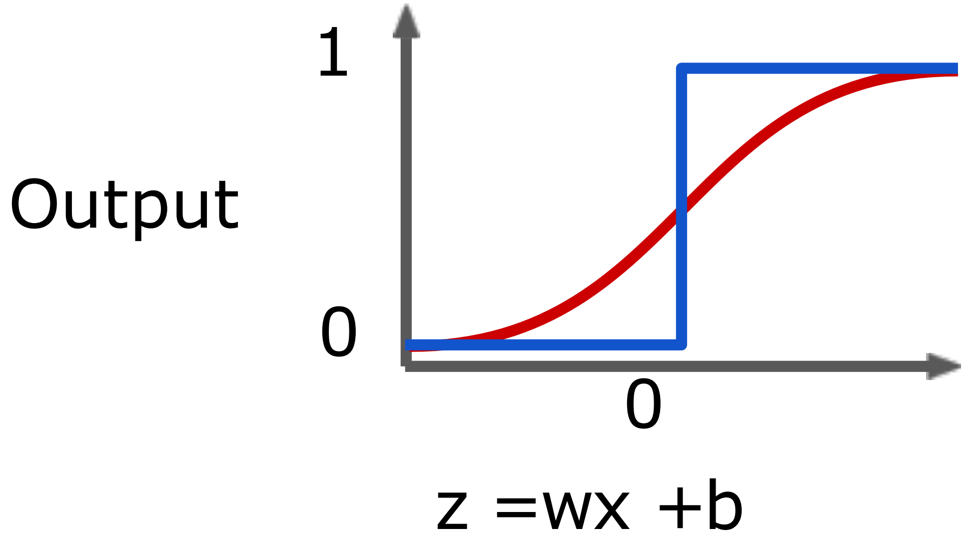
- Wir haben Glück, das entspricht der Sigmoid-Funktion.



$$f(x) = \frac{1}{1+e^{-(x)}}$$

Aktivierungsfunktion

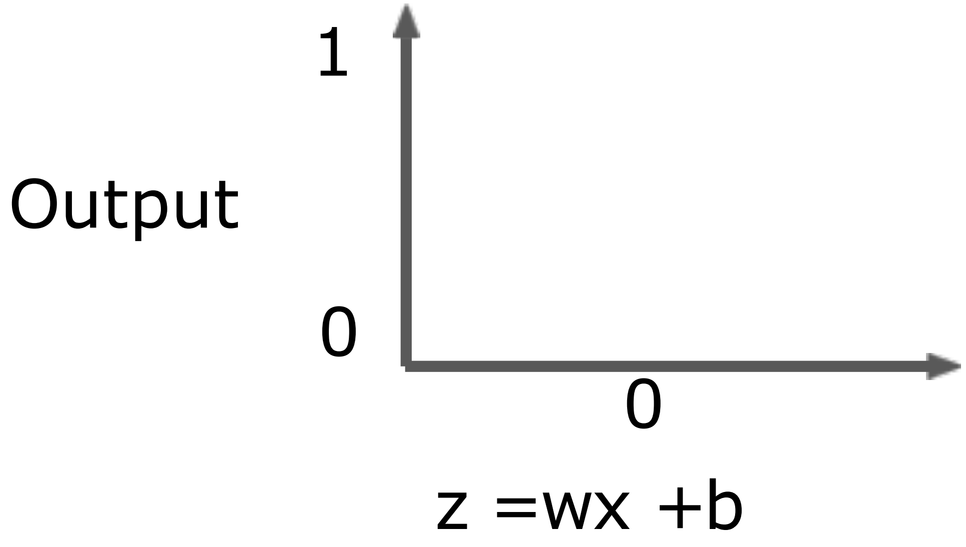
- Eine Änderung der verwendeten Aktivierungsfunktion kann je nach Aufgabe sinnvoll sein.



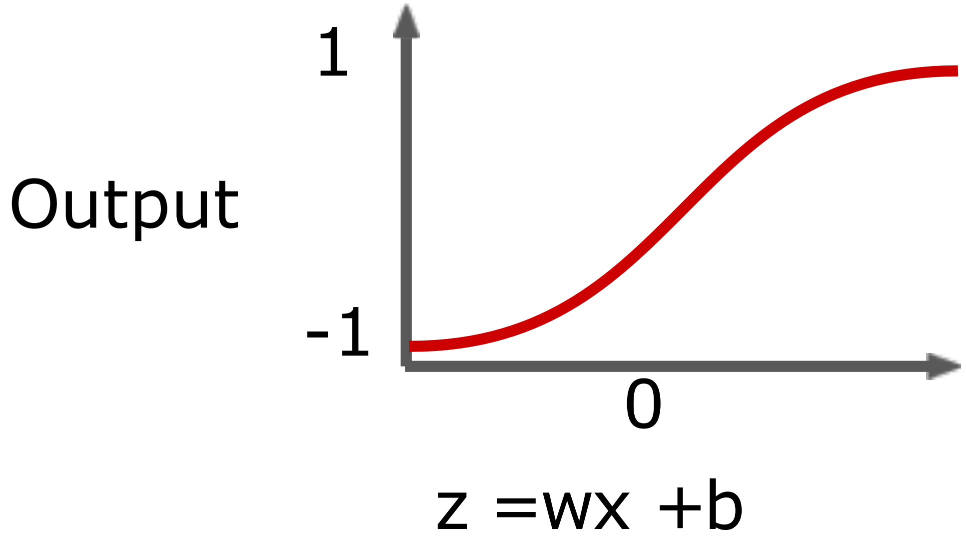
$$f(x) = \frac{1}{1+e^{-(x)}}$$

Aktivierungsfunktion

- Lass uns noch ein paar weitere Aktivierungsfunktionen besprechen, die uns begegnen werden.



Hyperbolischer Tangens: tanh(z)



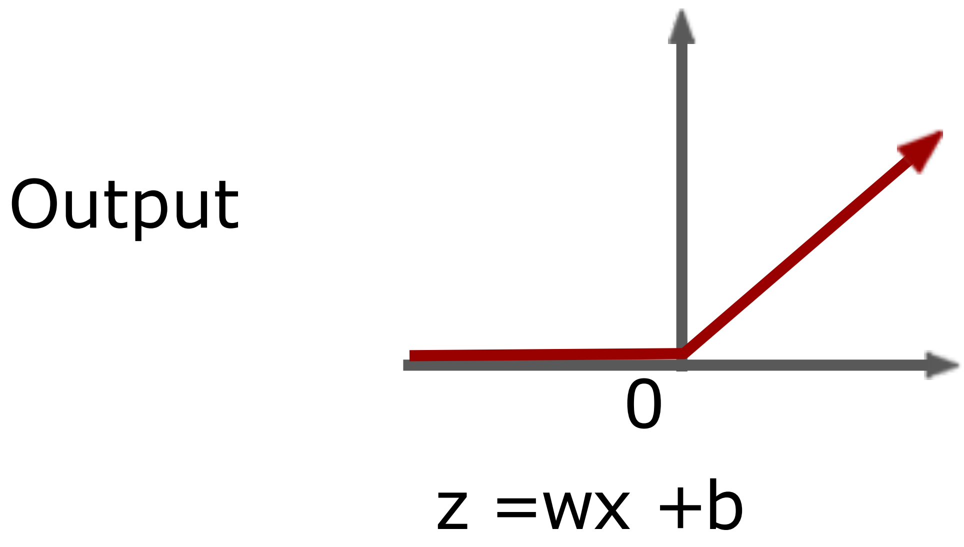
$$\cosh x = \frac{e^x + e^{-x}}{2}$$

$$\sinh x = \frac{e^x - e^{-x}}{2}$$

$$\tanh x = \frac{\sinh x}{\cosh x}$$

Rectified Linear Unit, (ReLU)

- Rectified Linear Unit (Gleichgerichtete Lineareinheit):
  Dies ist eigentlich eine relativ einfache Funktion: max(0,z)



Aktivierungsfunktion

- ReLu und tanh haben in der Regel die beste Leistung, daher werden wir uns auf diese beiden konzentrieren.
- Deep-Learning-Libraries haben diese für uns eingebaut, so dass wir uns keine Sorgen machen müssen, sie manuell zu implementieren.

Neuronale Netzwerke

- Im weiteren Kurs werden wir noch über einige weitere hochmoderne Aktivierungsfunktionen sprechen.

- Als nächstes werden wir die Kostenfunktionen besprechen, mit denen wir messen können, wie gut diese Neuronen funktionieren.

Kostenfunktionen
(Cost Funktions)

Kostenfunktion

- Lass uns nun untersuchen, wie wir die Leistung eines Neurons bewerten können.

- Mit einer Kostenfunktion können wir messen, wie weit wir vom erwarteten Wert entfernt sind.

Kostenfunktion

Wir verwenden die folgenden Variablen:

- *y*, um den wahren Wert darzustellen
- *a* zur Darstellung der Vorhersage des Neurons

In Bezug auf Gewichte und Bias:

- w*x + b = z
- Übergebe *z* in die Aktivierungsfunktion $\sigma(z) = a$

Quadratische Kosten

- $C = \Sigma(y-a)^2 / n$
- Wir können sehen, dass größere Fehler durch die Quadratur deutlicher sichtbar werden.
- Leider kann diese Berechnung zu einer Verlangsamung der Lerngeschwindigkeit führen.

Kreuz-Entropie

- $C = (-1/n)\ \Sigma\ (y \cdot \ln(a) + (1-y) \cdot \ln(1-a)$
- Diese Kostenfunktion ermöglicht ein schnelleres Lernen.
- Je größer der Unterschied, desto schneller kann das Neuron lernen.

Neuronale Netze

- Wir haben jetzt 2 Schlüsselaspekte des Lernens mit neuronalen Netzen, die Neuronen mit ihrer Aktivierungsfunktion und die Kostenfunktion.

- Uns fehlt noch ein wichtiger Schritt, nämlich das "Lernen"!

Neuronale Netzwerke

- Wir müssen herausfinden, wie wir unsere Neuronen und die Messung des Fehlers (unsere Kostenfunktion) nutzen können und dann versuchen, unsere Vorhersage zu korrigieren, mit anderen Worten, "lernen"!

Gradientenverfahren

- In der nächsten Vorlesung werden wir kurz darauf eingehen, wie wir dies mit dem Gradientenverfahren (Eng.:Gradient Descent) erreichen können.

Gradientenverfahren und Backpropagation

(Gradient Descent and Backpropagation)

Gradientenverfahren

- Wenn du dich schon einmal mit maschinellem Lernen beschäftigt hast, hast du vielleicht schon vom Gradientenverfahren (Gradient Descent) gehört!

- Lass uns das noch mal schnell durchgehen um uns einen Überblick zu verschaffen

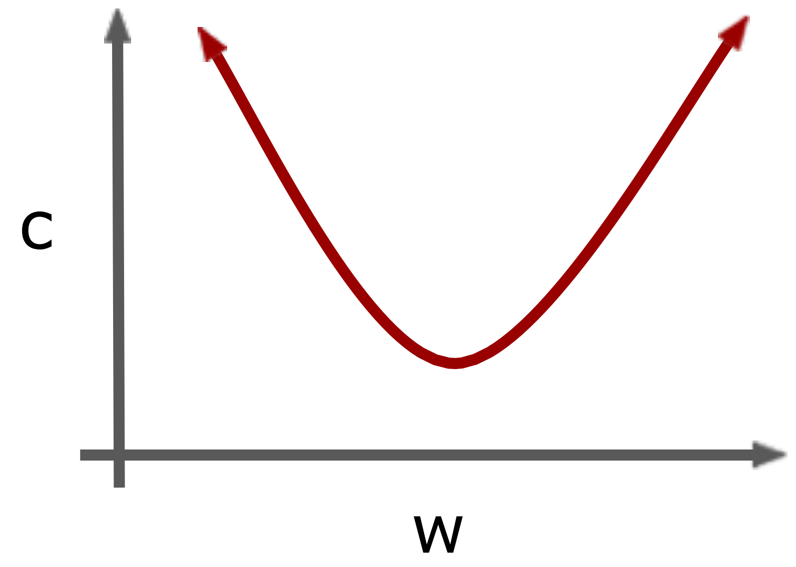
Gradientenverfahren

- Gradientenverfahren ist ein Optimierungsalgorithmus, um das Minimum einer Funktion zu finden.

- Um ein lokales Minimum zu finden, unternehmen wir Schritte proportional zum Negativ des Gradienten.

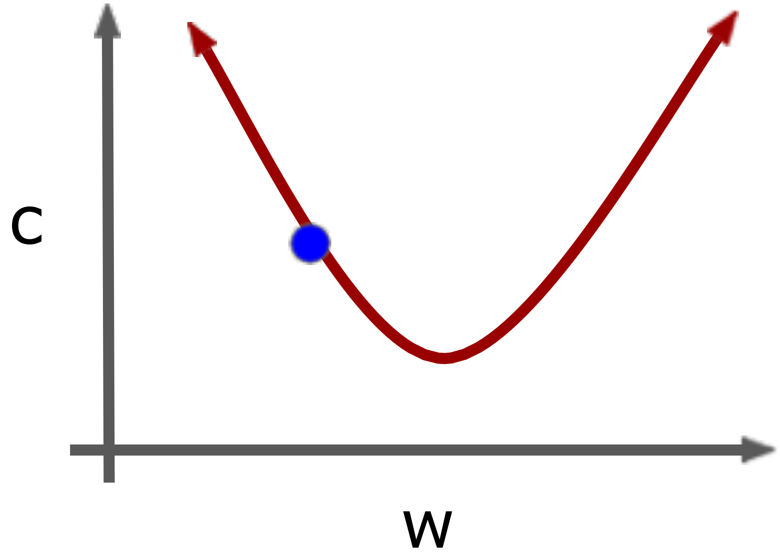
Gradientenverfahren

- In einer Dimension



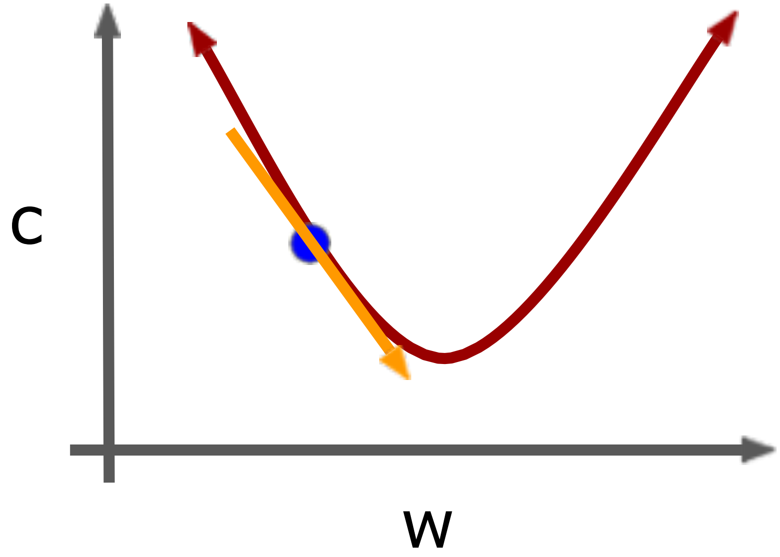
Gradientenverfahren

- In einer Dimension



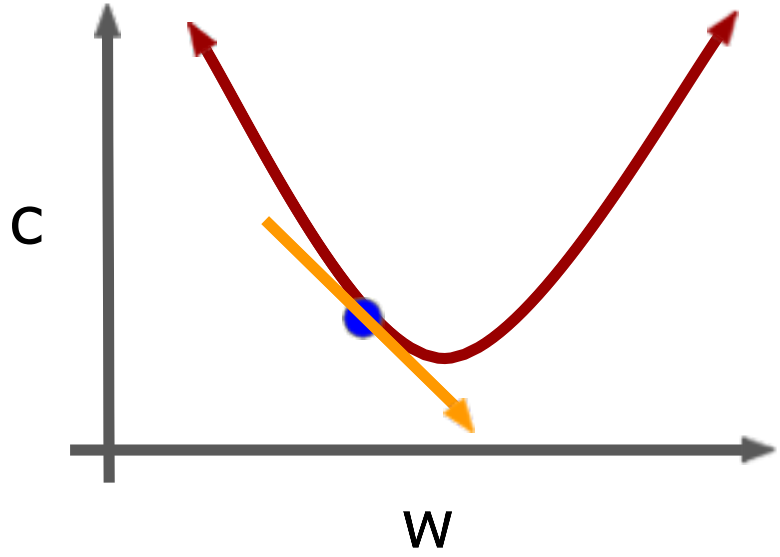
Gradientenverfahren

- In einer Dimension



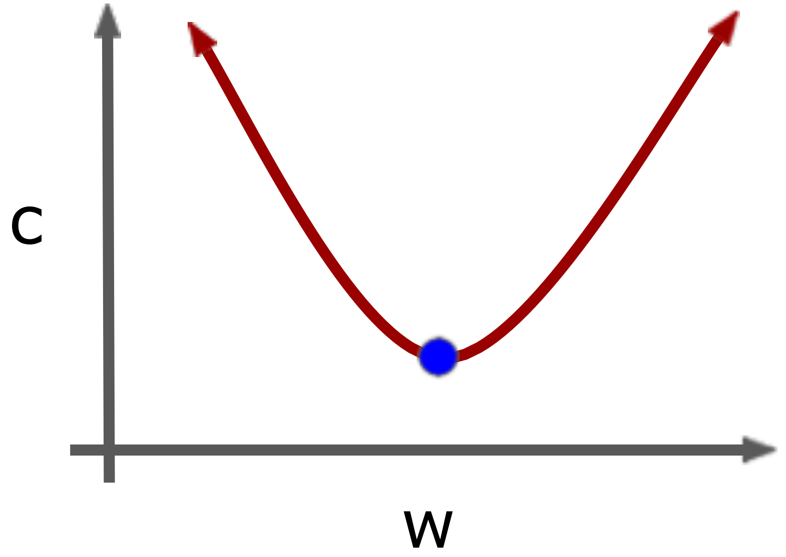
Gradientenverfahren

- In einer Dimension



Gradientenverfahren

- Visuell können wir sehen, welchen Parameterwert wir wählen müssen, um unsere Kosten zu minimieren.



Gradientenverfahren

- Dieses Minimum zu finden ist für 1 Dimension einfach. Allerdings werden unsere Fälle viel mehr Parameter haben. Das bedeutet, dass wir die eingebaute lineare Algebra verwenden müssen, die unsere Deep Learning Bibliothek bieten wird.

Gradientenverfahren

- Mit Hilfe der Gradientenabsenkung können wir die besten Parameter zur Minimierung unserer Kosten ermitteln, z.B. die besten Werte für die Gewichte der Neuroneneingänge.

Backpropagation

- Wir haben jetzt nur noch ein Problem zu lösen. Wie können wir schnell die optimalen Parameter oder Gewichte in unserem gesamten Netzwerk einstellen?

- Hier kommt die Backpropagation (Rückführung) ins Spiel!

Backpropagation

- Backpropagation wird verwendet, um den Fehlerbeitrag jedes Neurons nach der Verarbeitung eines Datenstapels zu berechnen.
- Es stützt sich stark auf die Kettenregel, um durch das Netzwerk zurückzugehen und diese Fehler zu berechnen.

Backpropagation

- Backpropagation arbeitet mit der Berechnung des Fehlers am Ausgang und verteilt sich dann wieder über die Netzwerkschichten.
- Du benötigst für jeden Eingabewert einen bekannten Sollwert (überwachten Lernen / Supervised Learning).

Backpropagation

- Die Umsetzung der Backpropagation wird weiter verdeutlicht, wenn wir in das Mathematikbeispiel eintauchen.
- Lass uns unsere komplexe Diskussion mit dem „Playground“ von TensorFlow beenden.

TensorFlow Playground

TensorFlow Playground

Gehe zu:

playground.tensorflow.org

Manuelle Neuronale Netzwerke

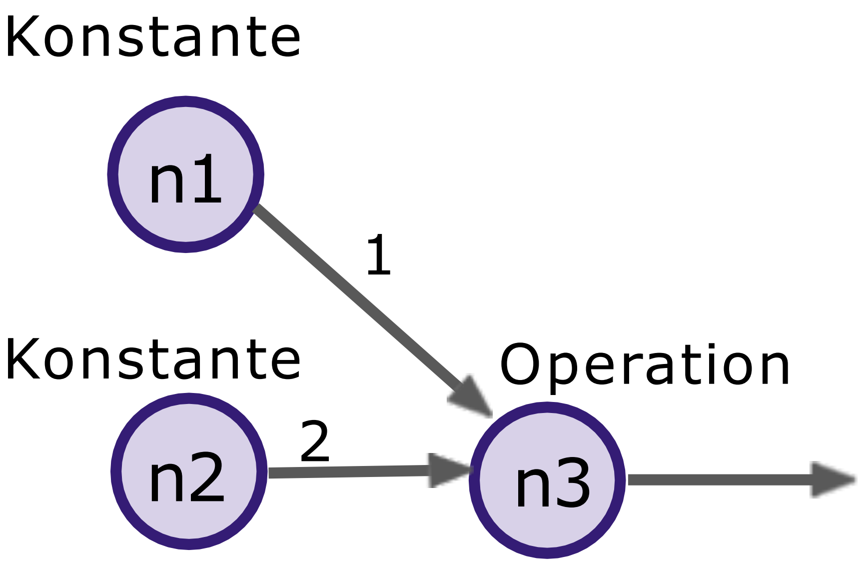
Teil 2: Operatoren

Operation-Klasse

- Eingabeknoten (Input nodes)
- Ausgabeknoten (Output nodes)
- Globale Standard-Grafikvariable (Global Default Graph Variable)
- Berechnungsmethode (Compute)
  - Überschrieben durch erweiterte Klassen

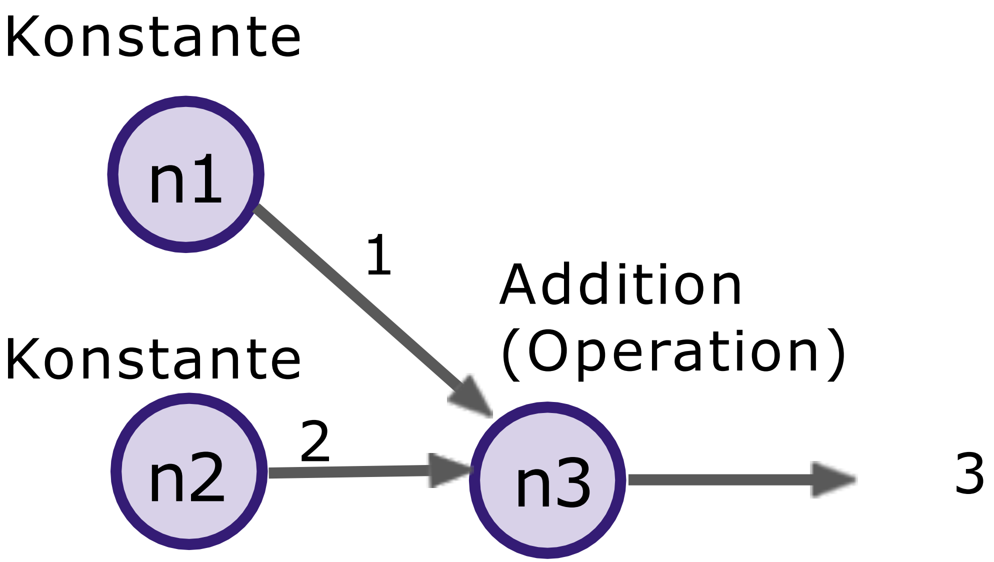
Operationen

- Graph – eine Globale Variable



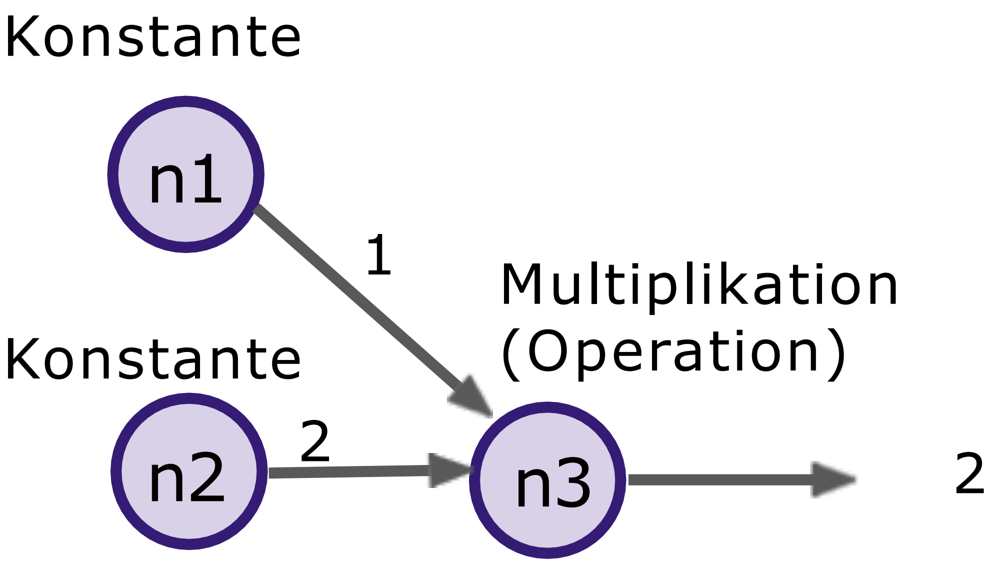
Operationen

- Graph



Operationen

- Graph



Platzhalter, Variablen und Graphen

- Platzhalter - Ein "leerer" Knoten, der einen Wert zur Berechnung der Ausgabe benötigt.

- Variablen - Änderbare Parameter des Graphen

- Graph - Globale Variable, die Variablen und Platzhalter mit Operationen verbindet.

Lasst uns loslegen!

Manuelle Neuronale Netzwerke

Teil 4: Session

PostOrder Tree Traversal

- Jetzt, da der Graph alle Knoten hat, müssen wir alle Operationen innerhalb einer Sitzung ausführen.

- Wir verwenden einen PostOrder Tree Traversal, um sicherzustellen, dass wir die Knoten in der richtigen Reihenfolge ausführen.

Manuelle Neuronale Netzwerke

Teil 5: Klassifikation

Manuelle Neuronale Netzwerke

Teil 6: Perzeptron

Definieren des Perzeptrons

- y = mx + b
- y = -1x + 5
- Denke daran, dass sowohl y als auch x Features sind!
- Feat2 = -1*Feat1 + 5
- Feat2 + Feat1 - 5 = 0
- FeatMatrix [1, 1] - 5 = 0

Exklusive Gutscheine

Verwende den Gutschein „**SLIDESHARE2018**“ auf *Udemy* oder die Shortlinks und erhalte unsere Kurse für nur 10,99€ (95% Rabatt).

Deep Learning Grundlagen mit TensorFlow und Python	https://goo.gl/FqNoAe
Python für Data Science und Machine Learning:	https://goo.gl/cE7TQ3
Original Python Bootcamp - Von 0 auf 100:	https://goo.gl/gin7pX
R für Data Science und Machine Learning:	https://goo.gl/8h5tH7