From Perceptron to Transformer: The Evolution of Neural Networks (1)

1. Perceptron and Gradient Descent

Ein Perceptron ist das einfachste Modell eines künstlichen Neurons und bildet die Grundlage für neuronale Netzwerke. Der Algorithmus wurde von Frank Rosenblatt entwickelt und wurde in dem 1962 veröffentlichten Papier “Principles of Neuro-dynamics: Perceptrons and the Theory of Brain Mechanisms” zusammengefasst [1]. Perceptron kann für binäre Klassifikation verwendet werden, um lineare Entscheidungsgrenzen zu lernen und Daten in zwei Klassen zu trennen.

Frank Rosenblatt wurde bei der Entwicklung des Perceptrons von der Struktur und Funktionsweise biologischer Neuronen inspiriert . Er wollte ein Modell schaffen, das die Lernfähigkeit des menschlichen Gehirns nachahmt [Fig.1].

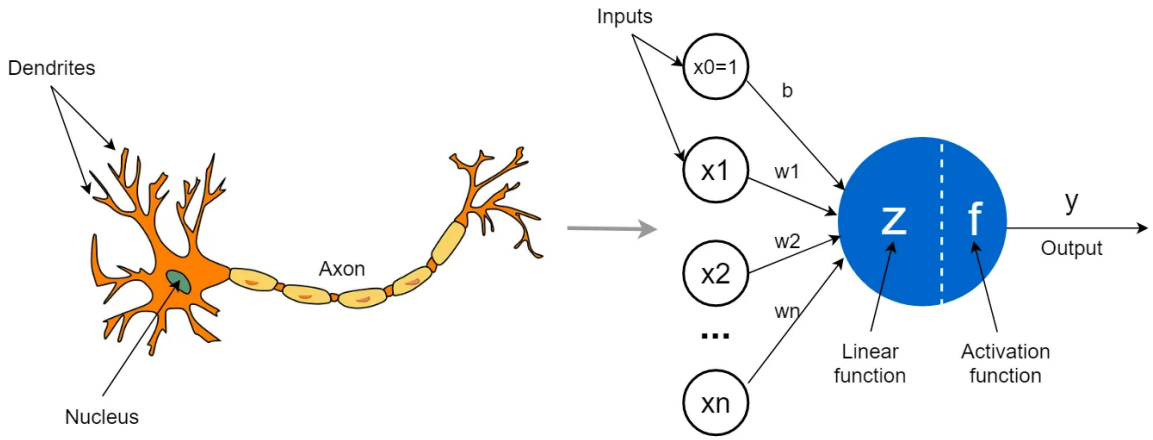


Fig.1 repräsentier eine Analogie zwischen einem biologischen Neuron und einem künstlichen Neuron (Perceptron) in einem neuronalen Netzwerk. [2]

- **Inputs:** repräsentieren die Signale benachbarter Nervenzellen, die über Synapsen zu den Dendriten gelangen.
- **Weights:** in der linearen Funktion repräsentieren, wie das Neuron auf die Signale reagieren wird. Wenn das Signal stark genug ist, führt es zu einem Aktionspotenzial.
- **Activation Function:** repräsentiert dieses Aktionspotenzial [3].
  - **Depolarisation**: wenn ein Neuron ein ausreichend starkes Eingangssignal erhält, wird die elektrische Ladung einer Zelle positiver. Dies geschieht, wenn positiv geladene Ionen wie Natrium (Na+) in die Zelle strömen oder wenn negativ geladene Ionen wie Chlorid (Cl-) aus der Zelle herausströmen.
  - **Repolarisation**: normalisiert die elektrische Ladung eines Neurons durch Ausströmung von positiven Ionen Kalium (K+) aus der Zelle oder die Einstrom von negative Ionen Chlorid (Cl-) in die Zelle.
  - **Hyperpolarisation**: Dies bezieht sich auf einen Zustand, in dem die elektrische Ladung innerhalb einer Zelle negativer wird als im Ruhezustand.

1.1 Perceptron und boolesche Operationen

Perceptron benutzt die Threshold-Funktion (**Heaviside Step Function**), um zu entscheiden, welcher Klasse ein Eingangsvektor zugeordnet wird. Diese Funktion gibt 1 zurück, wenn die Summer der Eingaben (**Inputs**) $\begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$, die durch Gewichte (**Weights**) $\begin{pmatrix} w_1 \\ \dots \\ w_n \end{pmatrix}$ skaliert → $(w_1, \dots, w_n) \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix}$ und mit **Bias** w_0 summiert werden, größer oder gleich null ist.

$$y = \begin{cases} 1 & \text{wenn } w_0 + \sum_{i=1}^{N} w_i * x_i \geq 0 \\ 0 & \text{sonts} \end{cases}$$

So ein Perceptron kann grundlegende boolesche Operationen wie **AND**, **OR, NOT** und **XOR** darstellen. Jede dieser Operationen kann durch ein einzelnes Perceptron modelliert werden, indem geeignete Gewichte und Bias-Werte gewählt werden.

- AND → $(w_0 = -1.5, w_1 = 1, w_2 = 1)$
- OR → $(w_0 = -0.5, w_1 = 1, w_2 = 1)$
- NOT → $(w_0 = 0.5, w_1 = -1)$
- XOR → AND(OR, NOT(AND)) (mit einem einzelnen Perceptron kann die XOR-Operation nicht dargestellt werden, da sie eine nicht-lineare Trennung benötigt)

Beispiele:

- AND: $x = (1,1) \rightarrow y = -1.5 + (1 \cdot 1 + 1 \cdot 1) = 0.5 \geq 0 = 1$
- OR: $x = (0,0) \rightarrow y = -0.5 + (1 \cdot 0 + 1 \cdot 0) = -0.5 < 0 = 0$
- NOT: $x = (0) \rightarrow y = 0.5 + (-1 \cdot 0) = 0.5 \geq 0 = 1$
- XOR: $x = (1,1) \rightarrow y = \text{AND}(\text{OR}, \text{NOT}(\text{AND})) = \text{AND}(1, 0) = 0$

1.2 Perceptron-Lernregel

Die Perceptron-Lernregel ist ein Algorithmus, der verwendet wird, um die Gewichte w und den Bias b eines Perceptrons so anzupassen, dass es eine gegebene Menge von Trainingsdaten korrekt klassifiziert [4].

Algorithm:

1. $y^{(i)} = \begin{cases} 1 & \text{wenn } \sum_{i=1}^{n} w_i x_i^{(i)} + b > 0 \\ 0 & \text{sonst} \end{cases}$ → Vorhersage
2. Wenn $\overline{y}^{(i)} \neq y^{(i)}$
   a. $w_j = w_j + \alpha(y^{(i)} - y^{(i)}) \cdot x^{(i)}$
   b. $b = b + \alpha(y^{(i)} - y^{(i)})$

1.3 Gradient Descent

Gradient Descent ist ein Optimierungsalgorithmus, der in der Mathematik und im maschinellen Lernen verwendet wird, um die Parameter eines Modells zu optimieren und die Kostenfunktion (oder den Fehler) zu minimieren [5, S. 482-492].

$$J := \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - f(x^{(i)}))^2$$

Algorithm (z.B. für Linear regression mit einem Gewicht $f(x^{(i)}) = \theta_0 + \theta_1 x^{(i)}$ **)**:

Für jede **Epoche**

1. $J := \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - f(x^{(i)}))^2$ → Cost function
2. $\frac{\partial J}{\partial \theta_0} = -\frac{2}{N}\sum_{i=1}^{N}(y^{(i)} - f(x^{(i)}))$ → Partial derivation for θ_0
3. $\frac{\partial J}{\partial \theta_1} = -\frac{2}{N}\sum_{i=1}^{N} x^{(i)} \cdot (y^{(i)} - f(x^{(i)}))$ → Partial derivation for θ_1

Jetzt können die Werte aus den Ableitungen genutzt werden, um die Gewichte zu optimieren.

4. $\theta_0 = \theta_0 + \alpha * \frac{\partial J}{\partial \theta_0} \mid \theta_1 = \theta_1 + \alpha * \frac{\partial J}{\partial \theta_1}$ → Updating bias and weight θ_0, θ_1
    - α → learning rate

Beispiel:

Datensatz besteht aus zwei Variablen: x = "Years Experience" und y = "Salary". Die unabhängige Variable x misst die Berufserfahrung in Jahren, während die abhängige Variable x das Jahresgehalt in US-Dollar darstellt.

$y = [1.1, 1.3, \ldots, 10.3, 10.5]$

$x = [39343, 46205, \ldots 122391, 121872]$

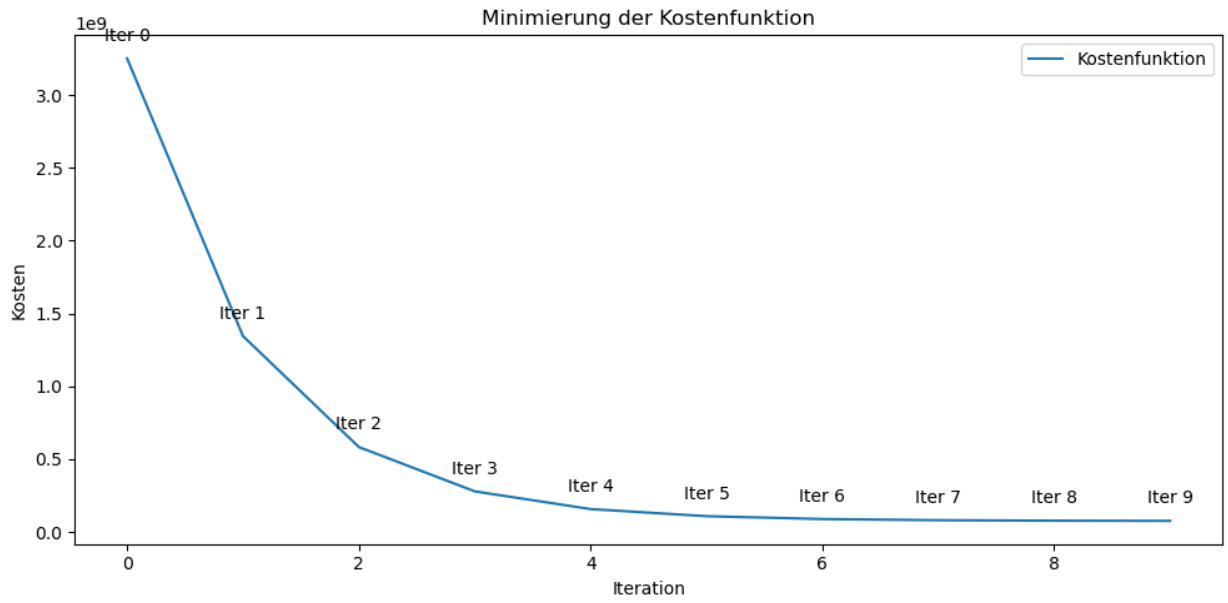
$w_1 = 0, w_0 = 0, a = 0.01$

Ablauf:

1. $\frac{\partial J}{\partial \theta_1} = -\frac{2 \cdot ((0-39343) \cdot 1.1 + \cdots + (0-121872) \cdot 10.5)}{30} = -477398$
2. $\frac{\partial J}{\partial \theta_1} = -\frac{2 \cdot ((0-39343) + \cdots + (0-121872))}{30} = -76003$
3. $w_1 = 0 - 0.01 \cdot (-477398) = 4773$
4. $w_0 = 0 - 0.01 \cdot (-76003) = 760$

    $\ldots$

39. $w_1 = 12785$
40. $w_0 = 2408$



Es gibt mehrere Ähnlichkeiten zwischen der **Perceptron-Lernregel** und **Gradient Descent.**

- **Beide verwenden eine Kostenfunktion:** Sowohl die Perceptron-Lernregel als auch Verfahren wie Gradient Descent verwenden eine Kostenfunktion, um den Fehler zu quantifizieren und die Richtung der Gewichtsänderungen zu bestimmen.
- **Iterative Optimierung:** Beide Methoden passen die Gewichte iterativ an, um den Fehler zu minimieren. Beim Perceptron wird dies durch Hinzufügen der Eingaben zu den Gewichten bei Fehlklassifikationen erreicht, während Gradient Descent die Gewichte basierend auf dem **Gradienten** $(w_i = w_i - \alpha \frac{\partial J}{\partial \theta_i})$ der Kostenfunktion aktualisiert.
- **Konvergenz zu einem lokalen Minimum:** Sowohl die Perceptron-Lernregel als auch Gradient Descent haben das Ziel, zu einem Minimum der Kostenfunktion zu konvergieren, auch wenn dies unter Umständen nur ein lokales Minimum sein kann.

References

1. Frank Rosenblatt. Principles of Neurodynamics. Perceptrons and the theory of brain mechanisms (15 March, 1961)
2. The Concept of Artificial Neurons (Perceptrons) in Neural Networks
3. Grider MH, Jessu R, Kabir R. Physiology, Action Potential. 2023 May 8.
4. Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works
5. Aston Zhang, Zachary C. Lipton, Mu Li, And Alexander J. Smola. Dive into Deep Learning