Matrizen in Neuronalen Netzen

Tim Zollner

17.02.2025

1 Einleitung

„Neuronale Netze sind die Kerntechnologie für **Large Language Models (LLM)**, Anwendungen wie ChatGPT, Technologien aus der **Bilderkennung** wie Gesichtserkennung und Objekt-Tracking oder **generative KI** für Bild und Film.“(Michael Kipp. Neuronale Netze und Deep Learning. TH Augsburg. https://michaelkipp.de/deeplearning)

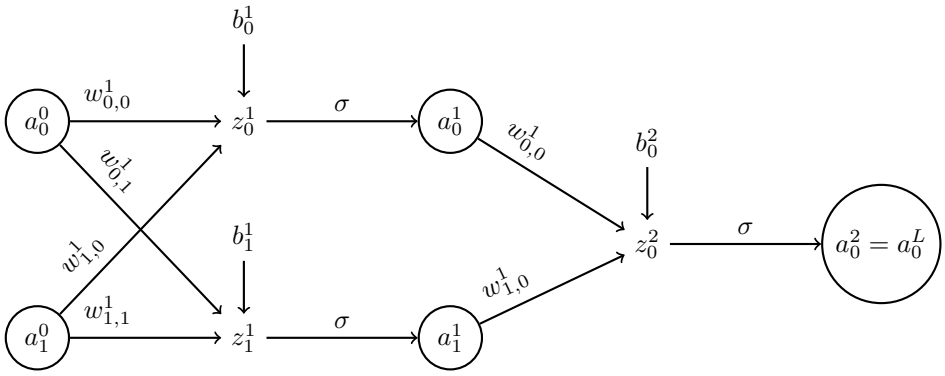
Mithilfe von neuronalen Netzen kann aus einer beliebigen Anzahl an numerischen Eingabewerten (Inputs) ein Ergebnis, das ebenfalls aus beliebig vielen numerischen Werten besteht berechnet werden. Diese Berechnung wird durch Parameter, die sogenannten Gewichte und Biases, bestimmt. Ein Neuronales Netz ist also einfach eine Funktion mit sehr vielen Parametern Die Berechnung eines Ergebnisses nennt man Feed-Forward, das bestimmen geeigneter Parameter heisst Backpropagation.

2 Feed-Forward

2.1 Einführung

Bei dem Feed-Forward Prozess wird ausgehend von dem Input a^0, also der ersten Schicht, immer die nächste Schicht berechnet. Dieser Vorgang wird bis zur Ausgabeschicht a^L wiederholt. Alle Schichten zwischen der Input- und der Ausgabeschicht werden als „Hidden Layer“ bezeichnet. Im folgenden Beispiel wäre die Schicht $l = 0$ die Eingabeschicht, die Schicht $l = 1$ ein „Hidden Layer“ und die Schicht $l = 2$ ($l = L$) die Ausgabeschicht.

2.2 Grafik



2.3 Berechnung

Dabei ergibt sich der Wert einer Roheingabe aus allen neuronen der vorherigen Schicht. die Gewichte bestimmen wie viel Einfluss jede vorherige Aktivierung auf den Roheingabe hat. zuletzt wird noch der Bias addiert.

$$z_0^1 = a_0^0 \cdot w_{0,0} + a_1^0 \cdot w_{1,0} + b^1$$

allgemein:

$$z_i^l = b_i^l + \sum_{j=0}^{n^l} a_j^{l-1} \cdot w_{j,i}^l$$

Die Roheingabe wir in die Aktivierungsfunktion $\sigma(x)$ eingesetzt um die Aktivierung zu erhalten. Diese transformiert die Roheingabe zu einem Wert zwischen 0 und 1.

$$a_j^l = \sigma(z_j^l) \qquad \sigma(x) = \frac{1}{1+e^{-x}}$$

2.4 Berechnung als Vektor

Um die Berechnung der Schichten zu vereinfachen und zu beschleunigen, wird Matrixmultiplikation verwendet. Dazu werden die gewichte als Matrix, und die Roheingaben, Aktivierungen und Biases einer Schicht jeweils als Vektor behandelt:

$$\vec{z}^l = \begin{pmatrix} z_0^l \\ z_1^l \\ ... \\ z_j^l \end{pmatrix} \qquad \vec{a}^l = \begin{pmatrix} a_0^l \\ a_1^l \\ ... \\ a_j^l \end{pmatrix} \qquad \vec{b}^l = \begin{pmatrix} b_0^l \\ b_1^l \\ ... \\ b_j^l \end{pmatrix} \qquad W^l = \begin{pmatrix} w_{0,0}^l & w_{1,0}^l & ... & w_{j,0}^l \\ w_{0,1}^l & w_{1,1}^l & ... & w_{j,1}^l \\ ... & ... & ... & ... \\ w_{0,i}^l & w_{1,i}^l & ... & w_{j,i}^l \end{pmatrix}$$

Durch die Matrixmultiplikation fällt das Summenzeichen weg. Außerdem kann eine gesamte Schicht in einem Vorgang berechnet werden

$$\vec{z}^l = W^l \cdot \vec{a}^{l-1} + \vec{b}^l$$

$$\begin{pmatrix} z_0^l \\ z_1^l \\ ... \\ z_i^l \end{pmatrix} = \begin{pmatrix} w_{0,0}^l & w_{1,0}^l & ... & w_{j,0}^l \\ w_{0,1}^l & w_{1,1}^l & ... & w_{j,1}^l \\ ... & ... & ... & ... \\ w_{0,i}^l & w_{1,i}^l & ... & w_{j,i}^l \end{pmatrix} \cdot \begin{pmatrix} a_0^{l-1} \\ a_1^{l-1} \\ ... \\ a_j^{l-1} \end{pmatrix} + \begin{pmatrix} b_0^l \\ b_1^l \\ ... \\ b_i^l \end{pmatrix}$$

$$\begin{pmatrix} z_0^l \\ z_1^l \\ ... \\ z_i^l \end{pmatrix} = \begin{pmatrix} w_{0,0}^l \cdot a_0^{l-1} + w_{1,0}^l \cdot a_1^{l-1} + ... + w_{j,0}^l \cdot a_j^{l-1} \\ w_{0,1}^l \cdot a_0^{l-1} + w_{1,1}^l \cdot a_1^{l-1} + ... + w_{j,1}^l \cdot a_j^{l-1} \\ ... \\ w_{0,i}^l \cdot a_0^{l-1} + w_{1,i}^l \cdot a_1^{l-1} + ... + w_{j,i}^l \cdot a_j^{l-1} \end{pmatrix} + \begin{pmatrix} b_0^l \\ b_1^l \\ ... \\ b_i^l \end{pmatrix}$$

3 Backpropagation

3.1 Einführung

Um die Backpropagation durchführen zu können braucht man zunächst geeignete Trainingsdaten. Diese bestehen aus Inputdaten und einer gewünschten Ausgabe. Diese gewünschte Ausgabe wird mit der tatsächlichen Ausgabe der Feed-Forward Berechnung verglichen. Dann wird jedes Parameter leicht angepasst um das Ergebnis für dieses Beipiel näher an die gewünschte Ausgabe zu bringen. Durch das Wiederholen mit vielen verschiedenen Trainingsdaten kann eine Parameterbelegung gefunden werden, die für die meisten Inputs ein Ergebnis erzeugt, das dem gewünschten zumindest nahe ist.

3.2 Verlustfunktion

Die Verlustfunktion gibt einen numerischen Wert (E) für den Unterschied zwischen gewünschter und tatsächlicher Ausgabe aus. Durch die Potenz werden größere Abweichungen besonders stark gewichtet, während kleinere irrelevant werden.

$$E_j = \frac{1}{2}(y_j - a_j^L)^2$$

Bei der Berechnung der Verlustfunktion der gesamten Ausgabeschicht werden die einzelnen Verluste einfach addiert:

$$E = \frac{1}{2}\sum_{j}^{n^L}(y_j - a_j^L)^2$$

Die Verlustfunktion ist so gewählt, dass die Ableitung möglichst simpel ist:

$$\frac{dE_j}{da_j^L} = (a_j^L - y_j)$$

3.3 Lernvorgang

Bei dem Lernvorgang wird ein Minimum der Verlustfunktion gesucht. Da die Verlustfunktion genauso viele Dimensionen wie Parameter hat, ist es unwahrscheinlich, dass man in einem hohen lokalen Minimum „steckenbleibt“ Dazu wird die Verlustfunktion nach den einzelnen Parametern abgeleitet um die Steigung der Verlustfunktion zu bestimmen. Dies wird für mehrere Trainingsbeispiel wiederholt. Der für die Steigung über mehrere Trainingsbeispiele wird dann von dem anfänglichen Wert des Parameters abgezogen, um die Verlustfunktion zu minimieren. Dadurch werden Parameter die einen größeren Einfluss auf den finalen Wert der Verlustfunktion haben stärker verändert als andere. Die Lernrate bestimmt wie start die Parameter bei jedem Durchlauf verändert werden.

$$w_{j,i}^l \to w_{j,i}^l - \frac{\eta}{N} \cdot \sum_{k=0}^{N} \frac{\partial E^k}{\partial w_{j,i}^l}$$

$$b_j^l \to b_j^l - \frac{\eta}{N} \cdot \sum_{k=0}^{N} \frac{\partial E^k}{\partial b_j^l}$$

η = Lernrate N = Anzahl der Trainingsbeispiele k = Trainingsbeispiel

3.4 Ableitung der sigmoid Funktion

Um nach den Parametern ableiten zu können muss auch die Aktivierungsfunktion abgeleitet werden.

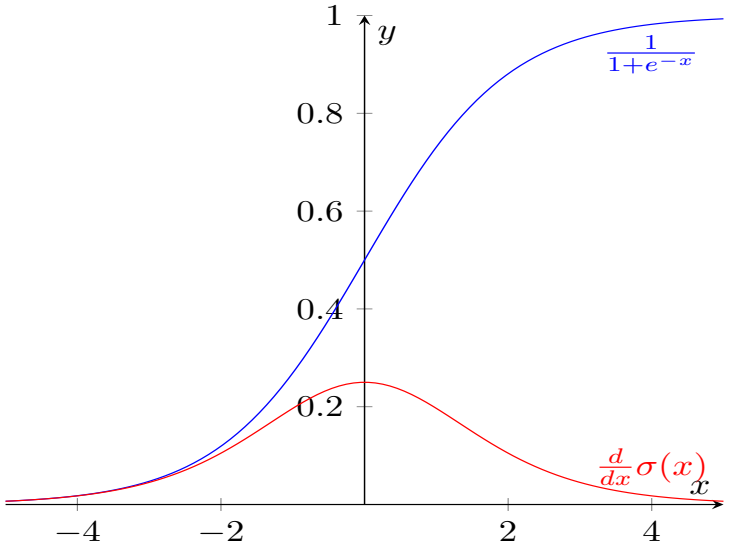
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

$$\frac{d}{dx}\sigma(x) = \frac{0(\cdot 1 + e^{-x}) - 1 \cdot e^{-x} \cdot -1}{(1+e^{-x})^2} = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1+e^{-x}-1}{(1+e^{-x})^2}$$

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

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

Da die Ableitung durch Werte berechnet wird, die bereits im Feed-Forward Prozess bestimmt wurden, können diese wiederverwendet werden.



3.5 Fehler(δ)

Der sogenannte Fehler ist ein Zwischenwert der zur berechnung der Änderungsraten der Gewichte und Biases benutzt wird. Der Fehlerterm berechnet jeweils die partielle Ableitung der Kostenfunktion E nach einer Roheingabe z. Es gibt deshalb für jede Roheingabe einen Fehlerwert.

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} \qquad \frac{\partial E}{\partial w_{j,i}^l} = \delta_i^l \cdot a_j^{l-1} \qquad \frac{\partial E}{\partial b_j^l} = \delta_j^l$$

Bei der Berechnung der Fehler wird zunächst die letzte Schicht berechnet und dann rekursiv die anderen Schichten. Für die Fehler der letzen Schicht ergibt sich durch die Kettenregel die folgende Formel:

$$\delta_j^L = \frac{\partial E}{\partial z_j^L} = \frac{\partial E}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L}$$

Nach dem Einsetzen der Ableitungen:

$$= (a_j^L - y_j) \cdot \sigma(z_j^L) \cdot (1 - \sigma(z_j^L))$$

Die Formel für den Fehler der restlichen Schichten kann auf zwei Teile aufgeteilt werden:

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l}$$

Die Ableitung der Aktivierung nach der Roheingabe entspricht einfach der Ableitung der Aktivierungsfunktion (hier sigmoid):

$$\frac{\partial a_j^l}{\partial z_j^l} = \sigma'(z_j^l)$$

Bei der Berechnung der Ableitung muss der jeweilige Einfluss auf die nachfolgenden Neuronen addiert werden:

$$\frac{\partial E}{\partial a_j^l} = \sum_{i=0}^{n^{l+1}} \frac{\partial E}{\partial z_i^{l+1}} \cdot \frac{\partial z_i^{l+1}}{\partial a_j^l} = \sum_{i=0}^{n^{l+1}} \delta_i^{l+1} \cdot \frac{\partial z_i^{l+1}}{\partial a_j^l}$$

$$\frac{\partial z_i^{l+1}}{\partial a_j^l} = \frac{\partial}{\partial a_j^l}\left(\sum_{p=0}^{n^{l+1}} a_p^l \cdot w_{p,i}^{l+1} + b_p^{l+1}\right) = w_{j,i}^{l+1}$$

Die Ableitung der Summe besteht hier aus einem einzigen Gewicht, da die Ableitung 0 ergibt wenn p $\neq$ j

$$\frac{\partial E}{\partial a_j^l} = \sum_{i=0}^{n^{l+1}} \delta_i^{l+1} \cdot w_{j,i}^{l+1}$$

Nach dem Zusammenführen der Einzelergebnisse ergibt sich die folgende Formel für die Berechnung der Fehlerterme der restlichen Schichten:

$$\delta_j^l = \left[\sum_{i=0}^{n^{l+1}} \delta_i^{l+1} \cdot w_{j,i}^{l+1}\right] \cdot \sigma'(z_j^l)$$

3.6 Berechnung des Fehlers als vektor

Wie bei dem Feed-Forward Prozess, kann auch bei der Backpropagation Matrixmultiplikation eingesetzt werden, um die Berechnungen zu vereinfachen und zu beschleunigen

3.6.1 Fehler des letzten Layers

Die Berechnung des letzten Layers benutzt noch keine Matrixmultiplikation, da die Parameter die Verlustfunktion nur auf einem direkten weg beeinflussen.

$$\delta_j^L = (a_j^L - y_j) \cdot \sigma'(z_j^L)$$

$$\vec{\delta^L} = \begin{pmatrix} \delta_0^L \\ \delta_1^L \\ ... \\ \delta_j^L \end{pmatrix} \qquad \vec{y} = \begin{pmatrix} y_0 \\ y_1 \\ ... \\ y_j \end{pmatrix} \qquad \vec{a^L} = \begin{pmatrix} a_0^L \\ a_1^L \\ ... \\ a_j^L \end{pmatrix} \qquad \vec{z^L} = \begin{pmatrix} z_0^L \\ z_1^L \\ ... \\ z_j^L \end{pmatrix}$$

$$\vec{\delta^L} = (\vec{a^L} - \vec{y^L}) \odot \sigma'(\vec{z^L})$$

3.6.2 Definition $\odot$

Das sogenannte Hadamart Produkt beschreibt die elementweise Multiplikation zweier Vektoren oder Matrizen mit den selben Dimensionen:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \odot \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_0 \cdot b_0 \\ a_1 \cdot b_1 \\ a_2 \cdot b_2 \\ a_3 \cdot b_3 \end{pmatrix}$$

3.6.3 Fehler eines beliebigen Layers

Wie beim Feed-Forward Prozess bleibt die Formel, bis auf das wegfallende Summenzeichen, fast gleich. Nur die Gewichtsmatrix muss transponiert werden. So kann wieder eine gesamte Schicht in einem Vorgang berechnet werden.

$$\delta_j^l = [\sum_{i=0}^{n^{l+1}} w_{j,i}^{l+1} \cdot \delta_i^{l+1}] \cdot \sigma'(z_j^l)$$

$$\delta_0^l = (\delta_0^{l+1} \cdot w_{0,0}^{l+1} + \delta_1^{l+1} \cdot w_{0,1}^{l+1} + ... + \delta_i^{l+1} \cdot w_{0,i}^{l+1}) \cdot \sigma'(z_0^l)$$

$$\delta_1^l = (\delta_0^{l+1} \cdot w_{1,0}^{l+1} + \delta_1^{l+1} \cdot w_{1,1}^{l+1} + ... + \delta_i^{l+1} \cdot w_{1,i}^{l+1}) \cdot \sigma'(z_1^l)$$

$$W^{l+1} = \begin{pmatrix} w_{0,0}^{l+1} & w_{1,0}^{l+1} & ... & w_{j,0}^{l+1} \\ w_{0,1}^{l+1} & w_{1,1}^{l+1} & ... & w_{j,1}^{l+1} \\ ... & ... & ... & ... \\ w_{0,i}^{l+1} & w_{1,i}^{l+1} & ... & w_{j,i}^{l+1} \end{pmatrix} \qquad (W^{l+1})^T = \begin{pmatrix} w_{0,0}^{l+1} & w_{0,1}^{l+1} & ... & w_{0,i}^{l+1} \\ w_{1,0}^{l+1} & w_{1,1}^{l+1} & ... & w_{1,i}^{l+1} \\ ... & ... & ... & ... \\ w_{j,0}^{l+1} & w_{j,1}^{l+1} & ... & w_{j,i}^{l+1} \end{pmatrix} \qquad \vec{\delta^{l+1}} = \begin{pmatrix} \delta_0^{l+1} \\ \delta_1^{l+1} \\ ... \\ \delta_j^{l+1} \end{pmatrix}$$

$$\Rightarrow \vec{\delta^l} = (W^{l+1})^T \cdot \vec{\delta^{l+1}} \odot \sigma'(\vec{z^l})$$

3.7 Feed-Forward Berechnung als Matriz

In der Vektorberechnung kann der Inputvektor auch durch eine Matrix aus den Inputs mehrerer Trainingbeispiele ersetzt werden. Jede Spalte der Matrix enthält die Input-Werte eines Trainingbeispiels. Der Bias vektor muss außerdem wie folgt zu einer Matrix umgewandelt werden. So wird der Feed-Forward prozess weiter vereinfacht und durch optimierte Matrixmultiplikation beschleunigt.

$$A^l = \begin{pmatrix} a_0^{l,0} & a_0^{l,1} & ... & a_0^{l,k} \\ a_1^{l,0} & a_1^{l,1} & ... & a_1^{l,k} \\ ... & ... & ... & ... \\ a_j^{l,0} & a_j^{l,1} & ... & a_j^{l,k} \end{pmatrix} \qquad B^l = \begin{pmatrix} b_0^l & b_0^l & ... & b_0^l \\ b_1^l & b_1^l & ... & b_1^l \\ ... & ... & ... & ... \\ b_j^l & b_j^l & ... & b_j^l \end{pmatrix}$$

$$A^l = \sigma(W^l \cdot A^{[l-1]} + \vec{b^l})$$

3.8 Backpropagation als Matrix

Auch bei der Matrixmultiplikation können die Vektoren nach dem selben Prinzip durch Matrizen ersetzt werden.

3.8.1 Fehlerberechnung

$$[\delta^l] = \begin{pmatrix} \delta_0^{l,0} & \delta_0^{l,1} & ... & \delta_0^{l,k} \\ \delta_1^{l,0} & \delta_1^{l,1} & ... & \delta_1^{l,k} \\ ... & ... & ... & ... \\ \delta_j^{l,0} & \delta_j^{l,1} & ... & \delta_j^{l,k} \end{pmatrix}$$

$$[\delta^L] = (Z^L - Y^L) \odot \sigma'(A^L)$$

$$[\delta^l] = (W^{l+1})^T \cdot [\delta^{l+1}] \odot \sigma'(A^l)$$

$$\Rightarrow \frac{\partial E}{B^l} = [\delta^l] \qquad \Rightarrow \frac{\partial E}{b_j^l, k} = \delta_j^l, k$$

Um die durchschnittliche Steigung eines Biases zu berechnen, kann man die Bias-Matrix mit einem Einsvektor multiplizieren. Auch dieser Vorgang wird also durch Matrizen vereinfacht.

$$\varnothing \frac{E}{b^l} = [\delta^l] \cdot \begin{pmatrix} 1 \\ 1 \\ ... \\ 1 \end{pmatrix}$$

3.8.2 Berechnung der Gradienten der Gewichte

$$[\delta^l] = \begin{pmatrix} \delta_0^{l,0} & \delta_0^{l,1} & ... & \delta_0^{l,k} \\ \delta_1^{l,0} & \delta_1^{l,1} & ... & \delta_1^{l,k} \\ ... & ... & ... & ... \\ \delta_j^{l,0} & \delta_j^{l,1} & ... & \delta_j^{l,k} \end{pmatrix} \qquad A^{l-1} = \begin{pmatrix} a_0^{l-1,0} & a_0^{l-1,1} & ... & a_0^{l-1,k} \\ a_1^{l-1,0} & a_1^{l-1,1} & ... & a_1^{l-1,k} \\ ... & ... & ... & ... \\ a_j^{l-1,0} & a_j^{l-1,1} & ... & a_j^{l-1,k} \end{pmatrix}$$

$$W^l = \begin{pmatrix} w_{0,0}^l & w_{1,0}^l & ... & w_{j,0}^l \\ w_{0,1}^l & w_{1,1}^l & ... & w_{j,1}^l \\ ... & ... & ... & ... \\ w_{0,i}^l & w_{1,i}^l & ... & w_{j,i}^l \end{pmatrix}$$

$$\frac{\partial E^k}{\partial w_{j,i}^l} = \delta_i^{l,k} \cdot z_j^{l-1,k}$$

$$\frac{\partial E}{\partial w_{0,0}^l} = \sum_{k=0}^{N} \delta_0^{l,k} \cdot a_0^{l-1,k}$$

$$\frac{\partial E}{\partial w_{1,0}^l} = \sum_{k=0}^{N} \delta_0^{l,k} \cdot a_1^{l-1,k}$$

$$\Rightarrow \frac{\partial E}{W^l} = [\delta^l] \cdot (A^{l-1})^T$$

4 Quellen

Michael Nielsen - Neural Networks and Deep Learning(Algorithmus)
Michael Kipp - Neurale Netze und Deep Learning (Algorithmus)
Sudeep Raja - A Derivation of Backpropagation in Matrix Form (Prüfen der Batch-weisen Berechnung)