Dokumentation des Neuronalen Netzes zur Erkennung Handgeschriebener Zahlen

Jacek Langer Jacek.Langer@Tu-Darmstadt.de

2023-03-09

Aufgabenstellung

Die Aufgabe war es eine Anwendung zur Erkennung von Handgeschriebenen Zahlen zu erstellen. Für die Erkennung der Zahlen soll ein neuronales Netz erstellt und trainiert werden. Als Trainingssatz kommt der MNIST Datensatz zu tragen der 60.000 handgeschriebene Zahlen mit Label zugehörigem beinhaltet, sowie 10.000 weitere Zahlen zur Kontrolle des Netzes.

Datensatz

Der Datensatz liegt in Serialisierter Form vor, dabei ist zwischen den Bildern und den Labels zu unterscheiden. Der Bilder Datensatz setzt sich aus 16 Byte Metainformationen sowie 60.000 Bildern a 28x28 Byte zusammen. Dies hat zufolge, dass Bilder aus einem eindimensionalen Array über eine Index-spanne geladen werden müssen. Bei den Labels besteht der Datensatz aus 8 Byte Metainformationen und 60.000 Byte.

Neuronales Netz

Ein neuronales Netz besteht in seiner einfachsten Form aus, aus einem Eingangs-Layer, einem Ausgangs-Layer und einer Gewichtungs-Matrix. Dabei werden Eingangswerte verschieden gewichtet und an Ausgänge weitergeleitet. Die Summe aller Eingänge gewichtet für den jeweiligen Ausgang wird an eine Aktivierungsfunktion übergeben, welche wiederum einen Wert zwischen 0 und 1 für jeden Ausgang zurückgibt.

Zum Trainieren werden die Ausgänge überprüft und die Fehler zurückgeführt, dabei werden Fehler für jedes Gewicht in der Gewichtungs-Matrix ermittelt und diese Korrigiert. Für die Fehlerrückführung kommt das Gradienten verfahren zu tragen, mit dem man im besten Fall, ein globales Minimum finden kann.

Hidden Layer

Um die Genauigkeit des Netzes zu verbessern, lassen sich Hidden Layer zwischen die Eingänge und Ausgänge schalten. Je mehr Layer man zwischen Schaltet, desto feiner können gewichte angepasst werden, da die einzelnen Gewichte keinen so großen Einfluss mehr besitzen. Jedoch ist zu bedenken das mit jedem Hidden Layer sich die Berechnungszeit erhöht. So hat eine hohe Anzahl an Hidden-Layern zur Folge das, dass neuronale Netz sehr langsam wird.

Werden Hidden Layer eingesetzt, müssen die Eingänge für jeden Layer ebenfalls an die Aktivierungsfunktion übergeben werden bevor sie an den nächsten Layer weitergegeben werden können.

Hidden Layer erstellen

Bei der Erstellung weiterer Schichten stellt sich die Frage wie groß diese denn sein sollen. Für die Größe lassen sich mehrere Ansätze finden. Man kann eine Lineare Reduktion der enthaltenen Ausgänge anstreben und in jeder Schicht n weniger Ausgänge erstellen als in der vorherigen. Dies lässt sich so oft wiederholen, bis man die gewünschte Anzahl an Ausgängen erreicht hat. Dies hat zu Folge das es unter Umständen zu sehr vielen Versteckte Schichten kommt, wenn die Schritte zu klein gewählt werden.

In diesem Netzwerk bestimmen wir jedoch die Anzahl der gewünschten Schichten im Vorfeld und ermitteln die Anzahl der Ausgänge über die Formel $\sqrt{I \cdot A}$ wobei I die Anzahl der Ausgänge der vorherigen Schicht(oder Eingänge in die aktuelle Schicht) ist und A die Anzahl an gewünschten Ausgängen. Der Wert Konvergiert gegen 0 und kann, wenn viele Schichten gewünscht sind um A gehoben werden ($\sqrt{I \cdot A} + A$). Dabei werden die ersten Schichten stark in ihrer Größe reduziert, während spätere Schichten nur noch wenig reduziert werden. Dies hat zur Folge, dass in den letzten schichten eine feinere Abstimmung der Gewichte stattfinden kann während der Informationen innerhalb der ersten paar schichten stark komprimiert werden.

Die Aktivierungsfunktion

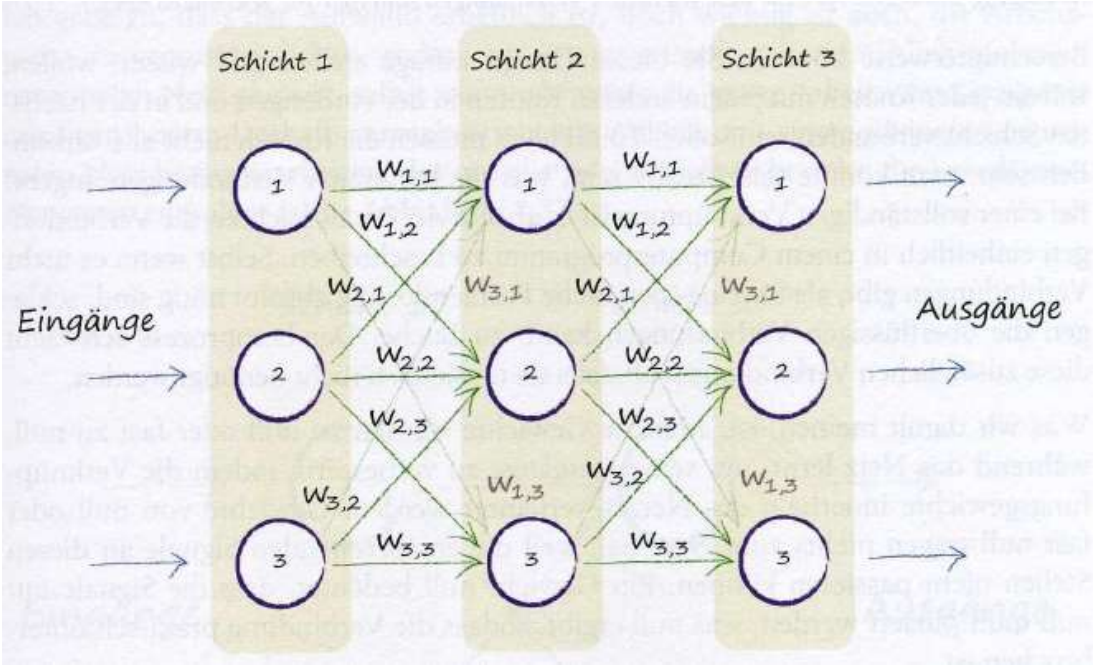
Als Aktivierungsfunktion wird die Sigmoidfunktion verwendet $\sigma = \frac{1}{1+e^{-x}}$. Dies hat den Vorteil, dass das Ergebnis stets im Bereich [-1,1] liegt.

Double Fehler führt zu falschen Ergebnissen

Es ist zu beachten, dass die Sigmoidfunktion zwar gegen 1 bzw. -1 konvergiert dieser Sachverhalt jedoch zu Problemen mit dem Arbeiten mit double führen kann. Double welche nach IEEE 754 Standard implementiert sind haben eine Genauigkeit von 15 bis 17 Nachkommastellen. Bewegt man sich in einem Bereich nahe der 1 oder 0 kann es vorkommen, dass fälschlicherweise ein Wert von 1 bzw. 0 angenommen wird. Für die Sigmoidfunktion würde das bedeuten, dass bei ausreichend großen x Werte 1 zurückgegeben wird. Dies kann bei der Fehler Rückführung zu Problemen führen, daher ist es Ratsam, eingangs die Gewichte sehr klein zu wählen.

Berechnen der Ausgangswerte

Für jede Schicht bis auf die Eingang Schicht müssen werte aktiviert werden bevor sie weitergereicht werden.



Schichten und Gewichte, Aufbau eines Neuralen Netzes

Dies geschieht, indem man die Summe aller Ausgänge der vorherigen Schicht multipliziert mit dem jeweiligen Gewicht für den Ausgang, and die Sigmoid Funktion übergibt.

D.h.:

i = Index der Schicht

j = Index des aktuellen Ausgangs in der Schicht i

$$\sigma_{ij} = \frac{1}{1 + e^{-\left(\sum_{k=0}^{n} x_k \cdot w_{kj}\right)}}$$

Betrachtet man die Summe der Gewichte multipliziert mit den Ausgängen der vorherigen Schicht stellt man fest, dass es sich hier um den Scalar zweier Vektoren handelt. $\sum_{k=0}^{n} x_k \cdot w_{kj} = x_0 \cdot w_{oj} + x_1 \cdot w_{1j} + \cdots + x_n \cdot w_{nj}$. Daraus lässt sich folgern, dass wenn jeder Ausgang einer Schicht der Skalar aus dem Ausgangsvektor der Vorherigen Schicht mit dem Gewichtungsvektor für den aktuellen Ausgang, dann ist die Aktuelle Schicht auch das Resultat der Matrizenmultiplikation der Gewichtungsmatrix mit dem Vektor der vorherigen Schicht $w_i \cdot A_i = A_{i+1}$.

Somit ist die Ausgabe des Netzwerks auch die Summe aller Schichten multipliziert mit ihrem Gewichts Vector gewichten. $\sigma_a = \sum_{i=0}^{n} \frac{1}{1+e^{-\sum_{j=0}^{k} w_{j,k} \cdot \frac{1}{1+e^{-\sum \cdots}}}}$

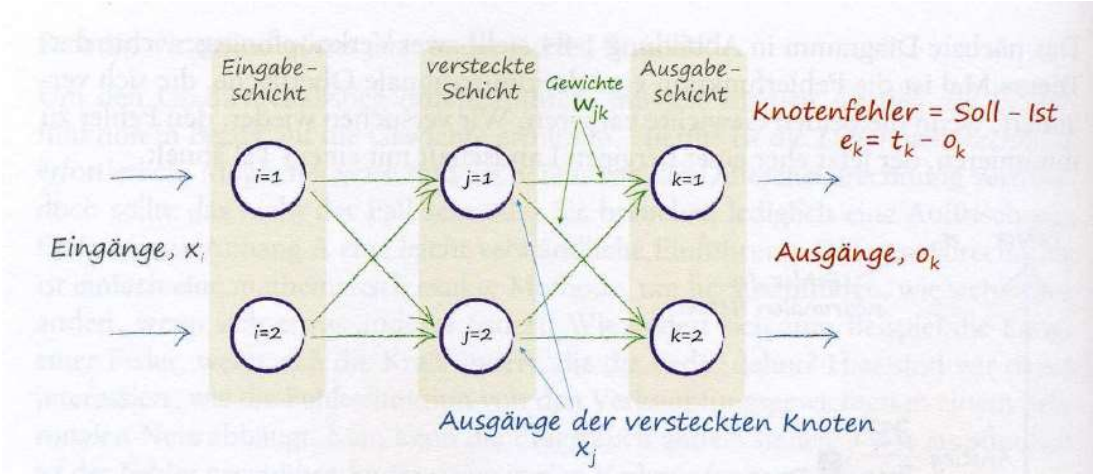
Ermitteln des Fehlers

Für das Trainieren des Netzwerks ist es essential den Fehler zu ermitteln und anhand des Fehlers, die Gewichte zu korrigieren. Dieser wird ermittelt in dem man jeden Ausgang des Ausgangsschicht mit dem Erwartungswert vergleicht. Die Differenz zwischen erwartetem Ergebnis und tatsächlicher Ausgabe bildet den Fehler in der Aktuellen Schicht. Dieser wird Quadriert um ein stets einen Positiven Fehler zu erhalten.

$$E = (l_i - a_i)^2, l = 0 \vee 1, a_i \in [0,1]$$

Rückführung des Fehlers und Korrektur der Gewichte

Wie bereits erwähnt wird für die Fehler Rückführung das Gradientenverfahren angewendet. Dies bedeutet im Allgemeinen, dass die Gewichte als eine "hügelige Landschaft" betrachtet werden und man stets in die Richtung des größten Abstieges geht um im besten Fall ein globales Minimum zu finden. Mehr dazu im Kapitel Gradientenverfahren. Um einen Gradienten abstieg durchzuführen, muss jedoch erst der Anstieg der Fehlerfunktion in Bezug zu den Gewichten ermittelt werden. Differenzieren wir diesen Bezug, erhalten wir Erkenntnis darüber wie der Fehler auf eine Veränderung der Gewichte reagiert. Um das Verhältnis des jeweiligen Fehlers zu allen Gewichten zu betrachten, betrachten wir $\frac{\partial E}{\partial W_{jk}}$.



Gewichte zwischen den versteckten Schichten und Ausgangsschicht

E ist die Summe aller Fehler der Ausgabeschicht. $\sum_{k=0}^{n}(t_k - o_k)^2$.

Bei genauerer Betrachtung stellt man fest, dass die Ausgabe eines Knoten k, nur von den Verknüpften Eingängen abhängig ist. Das bedeutet, dass die Ausgabe o_k nur von den eingängen w_{jk} abhängt und nicht von w_{jb} und $b \neq k$ ist. Dies bedeutet das man immer nur o_k betrachtet und die Summe nicht weiter berücksichtigt werden muss.

Die Ableitung der Fehlerfunktion

sei t_k =Konstant so ist $[(t_k - o_k)^2]' = u' \cdot v' = 2(t_k - o_k) \cdot -1 = -2(t_k - o_k)$

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

$$f'(x) = \frac{u'v - uv'}{v^2} = \frac{e^x(1 + e^x) - (e^x)^2}{(1 + e^x)^2} = \frac{e^x + e^{2x} - e^{2x}}{(1 + e^x)^2} = \frac{e^x}{(1 + e^x)^2} = \frac{e^x}{(1 + e^x)(1 + e^x)}$$

$$= \frac{e^x}{e^x + 1} \cdot \frac{1}{e^x + 1} = \frac{e^x}{e^x + 1} \cdot \left(\frac{e^x}{e^x} - \frac{e^x}{e^x + 1}\right) = \frac{e^x}{e^x + 1} \cdot \left(1 - \frac{e^x}{e^x + 1}\right) = f(x) \cdot \big(1 - f(x)\big)$$

$$\frac{\partial E}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}}(t_k - o_k)^2$$

$$\Leftrightarrow \frac{\partial E}{\partial W_{jk}} = \frac{\partial E}{\partial o_k} \cdot \frac{\partial o_k}{\partial W_{jk}}$$

$$\Leftrightarrow \frac{\partial E}{\partial W_{jk}} = -2(t_k - o_k) \cdot \frac{\partial o_k}{\partial W_{jk}}$$

$$\Leftrightarrow \frac{\partial E}{\partial W_{jk}} = -2(t_k - o_k) \cdot sigmoid(x) \cdot \big(1 - sigmoid(x)\big)$$

$$x = \sum_j W_{jk} \cdot o_j$$

$$\Rightarrow \frac{\partial E}{\partial W_{jk}} = -2(t_k - o_k) \cdot sigmoid\left(\sum_j W_{jk} \cdot o_j\right) \cdot \left(1 - sigmoid\left(\sum_j W_{jk} \cdot o_j\right)\right) \cdot o_j$$

Da unsere Ausgänge bereits die Sigmoidfunktion angewendet auf die Summe der Verknüpfungen zurückgeben können wir weiter vereinfachen und erhalten:

$$\Delta W_{jk} = \alpha \cdot E_k \cdot o_k \cdot (1 - o_k) \cdot o_j^T$$

Wobei α unsere Lernrate darstellt.

Aktualisieren der Gewichte

Um die Gewichte zu aktualisieren, muss zunächst der Fehler in jeder Schicht ermittelt werden. Dazu wird der Fehler einer Schicht an die vorhergehende Schicht weitergereicht Somit wird ein Fehlervektor $\vec{E_j}$ für jede Schicht erstellt. D.h. $\vec{E_j} = W_j \cdot \vec{E_k}$

Der ermittelte Fehler jeder Schicht wird nun an die nach W_{jk} differenzierte Gleichung übergeben und auf die Gewichtungsmatrix addiert.

$$W_{k_{neu}} = W_{jk} + \Delta W_{jk} = W_k + \alpha \cdot \vec{E}_k \cdot \vec{O}_k \cdot \left(1 - \vec{O}_k\right) \cdot \vec{O}_{k-1}^T$$

Gradientenverfahren

> Im Allgemeinen bezeichnet Gradientenverfahren eine Optimierungsmethode, bei der die Abstiegsrichtung durch Gradienteninformation gewonnen wird, also nicht notwendigerweise auf den negativen Gradienten beschränkt ist.
>
> — Dimitri P. Bertsekas: Nonlinear programming. 3. Auflage. Athena Scientific 2016

Beim Gradientenverfahren startet man i.d.R. ab einem zufälligen Punkt und schreitet in kleinen Schritten in Richtung des Gradienten mit der größten Steigung ($|\nabla f(x,y)| = \max$). Dies führt zwangsläufig zu einem Lokalen Minimum. Gilt für dieses Minimum $f(x_0, y_0) < f(x,y) \forall x \in \mathbb{R}$ so ist dieses Lokale minimum auch ein globales Minimum.

Anwendung

```
-h                                      opens the help
-c                                      clean run, ignores existing weights
and creates new weights matrix.
-g <int>                                Number of generations to run for.
-l | --learning-rate <double>           Learning rate in percent.
-i | --increase-learning-rate           Increases the learning rate with each
Generation until a learning rate of 100 %.
--hidden                                The number of hidden layers
generated, should be used with option [-c]
```

Anwendungsbeispiel

```
# Startet das Netzwerk im Trainings Modus
# für 5 Generationen mit einer Lernrate von 3%
# die Lernrate steigert sich um 7.5% je Generation.

    java -jar neuronalesJavaNetz.jar -g 5 -c --lr 3 -i 7.5
```