

Machine Learning 3: Neuronale Netze IV

Ergänzungsfach Informatik, 2021/2022, pro@kswe.ch

22. November 2021

1 Das KNN trainieren¹

Zu Beginn haben wir einen linearen Klassifizierer verfeinert, indem wir als Parameter den Anstieg w_1 der linearen Funktion schrittweise angepasst haben. Gesteuert haben wir diese Verfeinerung anhand des Fehlers. Der Fehler berechnete sich als Sollwert minus berechneter Wert. In der Trainingsphase des KNN müssen wir die Gewichte nun ebenfalls anpassen - denn darin besteht das "Lernen" bei einem KNN. Wie aktualisieren wir die Gewichte, wenn mehr als ein Knoten zu einem Ausgang und seinem Fehler beiträgt? Abbildung 1 zeigt die Situation.

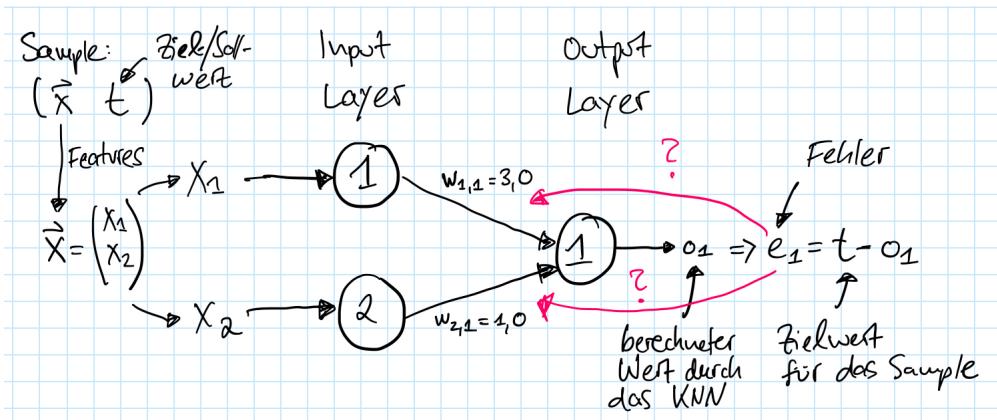


Abbildung 1: Der Knoten in der Ausgabeschicht hat zwei eingehende Verbindungen.

Die Lage wäre viel einfacher, wenn nur ein Knoten zu einem Ausgabeknoten führte. Wie nutzen wir aber bei zwei Knoten diesen Ausgabefehler?

1.1 Gewichte von mehr als einem Knoten lernen

Es ist nicht sinnvoll, mit dem gesamten Fehler nur ein Gewicht zu aktualisieren, weil dann die andere Verknüpfung und ihr Gewicht ignoriert werden. Immerhin sind für diesen Fehler mehrere Verbindungen verantwortlich. Man könnte deshalb beispielsweise den Fehler unter allen beitragenden Knoten gleichmäßig aufteilen. Abbildung 2 zeigt ein Beispiel.

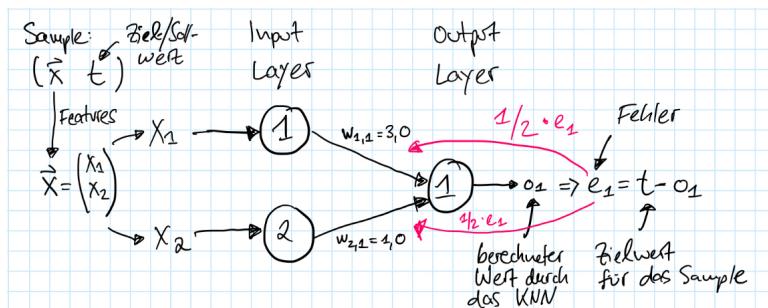


Abbildung 2: Der Fehler wird 50 : 50 auf die Gewichte verteilt.

¹Auszug aus dem Buch "Neuronale Netze selbst programmieren", Tariq Rashid"

Eine andere Idee ist, den Fehler zwar aufzuteilen, aber nicht gleichmässig. Stattdessen ordnen wir den Verbindungen, die grössere Gewichte haben, einen grösseren Anteil des Fehlers zu. Warum? Weil sie **mehr zum Fehler** beitragen. Abbildung 3 zeigt die Idee.

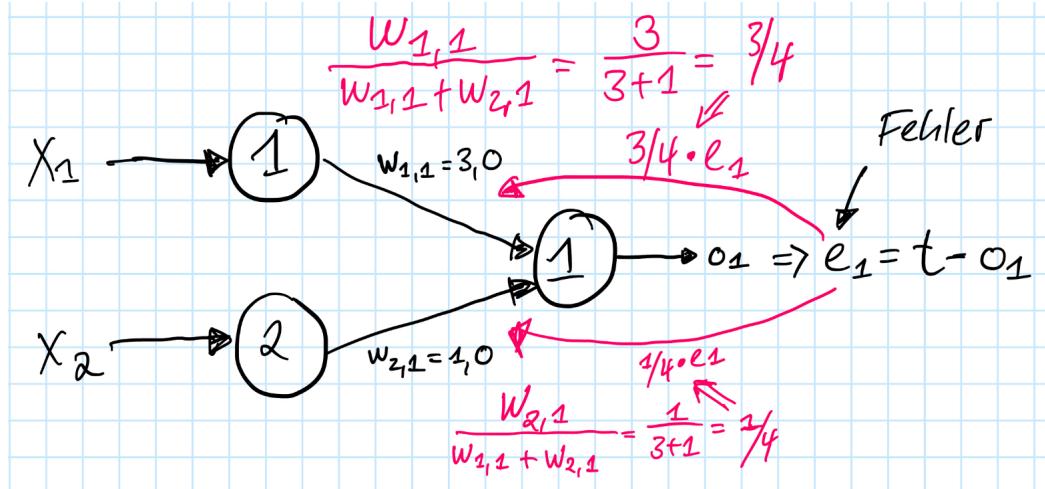


Abbildung 3: Der Fehler wird proportional zu den Gewichten aufgeteilt.

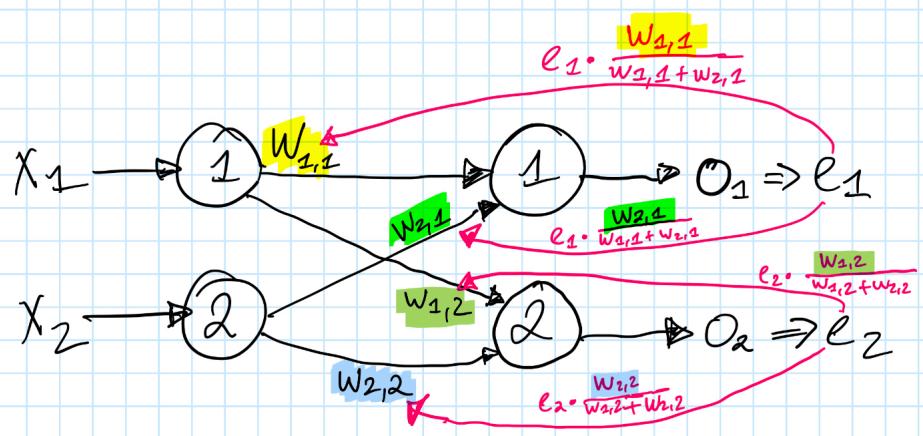
Wir erhalten eine proportionale Aufteilung des Fehlers, in dem wir den Anteil des Gewichts zum Gesamtgewicht der Verbindungen berechnen. Für das Beispiel aus Abbildung 3 ergeben sich dadurch folgende Berechnungen: $\frac{w_{1,1}}{w_{1,1}+w_{2,1}} = \frac{3,0}{3,0+1,0} = \frac{3}{4}$ und $\frac{w_{2,1}}{w_{1,1}+w_{2,1}} = \frac{1,0}{3,0+1,0} = \frac{1}{4}$. Diese Idee kann man auf beliebig viele Verbindungen zu einem Ausgabeknoten anwenden. Wir verwenden also die Gewichte, um den Fehler in der Gegenrichtung von der Ausgabe zurück ins KNN zu leiten. Diese Methode wird **Fehlerrückführung** (eng. **Backpropagation**) genannt.

Wichtig!. Wir haben bisher "nur" geklärt, wie der Fehler im Netz zurückgeführt wird. Die Aktualisierung der Gewichte ist noch ausstehend. Wir brauchen aber den Fehler, damit wir die Gewichte aktualisieren können!

1.2 Fehlerrückführung von mehreren Ausgabeknoten

Wir schauen uns nun an, wie man den Fehler bei mehr als einem Ausgabeknoten zurückführen kann. Abbildung 4 erweitert das Beispiel.

Abbildung 4: KNN mit zwei Schichten und je zwei Knoten.



Beide Ausgabeknoten können einen Fehler haben - das ist sogar äusserst wahrscheinlich, wenn das Netz noch nicht trainiert wurde. Wir können aber den gleichen Ansatz wie zuvor wählen, da die Verbindungen zu o_1 unabhängig von den Verbindungen zu o_2 sind. Wir führen die Fehlerrückführung somit einfach pro Ausgabeknoten aus und berechnen für jeden Ausgabeknoten die proportionale Verteilung des Fehlers mit den Gewichten zum Ausgabeknoten. Wir zeigen die Berechnungen pro Knoten am Beispiel aus Abbildung 4.

1.2.1 Fehler e_1 aufteilen

Der Fehler e_1 wird proportional auf die Gewichte $w_{1,1}$ und $w_{2,1}$ aufgeteilt. Somit berechnet sich der Anteil von e_1 zur Aktualisierung von $w_{1,1}$ wie folgt:

$$e_1 \cdot \frac{w_{1,1}}{w_{1,1} + w_{2,1}}$$

Der Anteil von e_1 zur Aktualisierung von $w_{2,1}$ berechnet sich wie folgt:

$$e_1 \cdot \frac{w_{2,1}}{w_{1,1} + w_{2,1}}$$

1.2.2 Fehler e_2 aufteilen

Der Fehler e_2 wird proportional auf die Gewichte $w_{1,2}$ und $w_{2,2}$ aufgeteilt. Somit berechnet sich der Anteil von e_2 zur Aktualisierung von $w_{1,2}$ wie folgt:

$$e_2 \cdot \frac{w_{1,2}}{w_{1,2} + w_{2,2}}$$

Der Anteil von e_2 zur Aktualisierung von $w_{2,2}$ berechnet sich wie folgt:

$$e_2 \cdot \frac{w_{2,2}}{w_{1,2} + w_{2,2}}$$

1.2.3 Zusammenfassung

Wir teilen den Fehler eines Ausgabeknotens so auf, dass grössere Gewichte einen grösseren Anteil des Fehlers erhalten und kleinere Gewichte einen kleineren Fehler. Bei mehreren Ausgabeknoten wird es nicht komplizierter. Wir tun einfach das Gleiche für jeden Ausgabeknoten.

1.3 Fehler auf mehrere Schichten zurückführen

Abbildung 5 zeigt ein KNN mit drei Schichten. Wir haben bisher gesehen, wie man den Fehler e_1 bzw. e_2 proportional auf die Gewichte der vorhergehenden Schicht ($w_{1,1}, w_{1,2}, w_{2,1}$ und $w_{2,2}$) aufteilen kann. Wir haben also eine Möglichkeit den Fehler auf die Gewichte zwischen dem Output Layer und dem Hidden Layer zu verteilen.

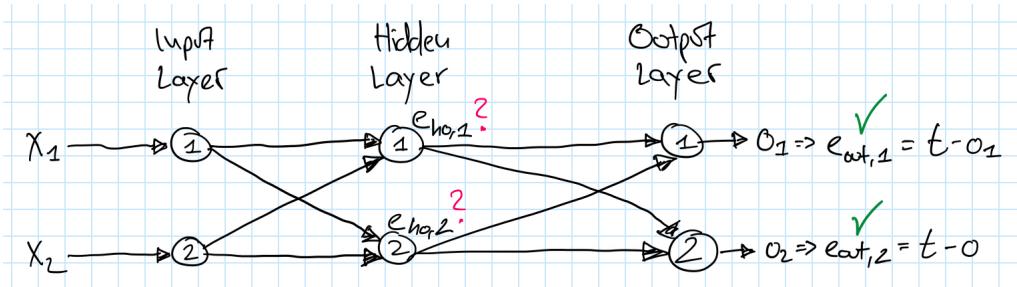


Abbildung 5: KNN mit drei Schichten, den Gewichten zwischen den letzten beiden Schichten und den Fehlern. Die Fehler $e_{out,1}$ und $e_{out,2}$ können einfach bestimmt werden, da Sollwerte in den Trainingsdaten vorliegen.

Nun müssen wir eine Möglichkeit finden, die Gewichte zwischen dem Input Layer und dem Hidden Layer anzupassen. Auch hier möchten wir die Gewichte proportional zum Fehler anpassen. Nun ist jedoch nicht klar, wie man zum Beispiel den Fehler $e_{ho,1}$ erhält, da ein Knoten in einer versteckten Schicht keinen offensichtlichen Fehler hat. Wir haben für die versteckten Knoten keine Solldaten aus den Trainingsdaten, denn lediglich für die Ausgabe im Output Layer ist bekannt, was pro Eingabe berechnet werden soll. Wir brauchen aber einen Fehler, um mit ihm die Gewichte in der vorhergehenden

Schicht aktualisieren zu können. Man behilft sich nun damit, dass man die Fehler aus der vorhergehenden Schicht verwendet. Jeder Knoten betrachtet nun die **ausgehenden** Verbindungen in die nächste Schicht. Dort gibt es für jede Verbindung bereits einen Fehler. Man bildet nun von diesen Fehlern eine gewichtete Summe und ordnet dem Knoten diesen Fehler zu. Abbildung 6 zeigt die Verteilung und Berechnung des Fehlers.

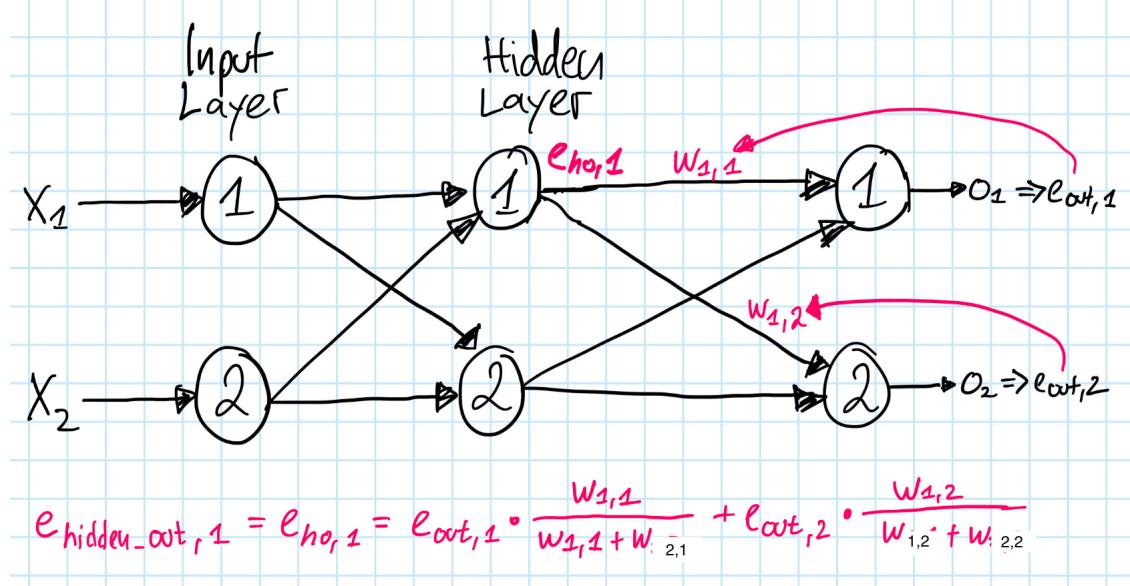


Abbildung 6: Der Fehler $e_{\text{hidden_out},1}$ berechnet sich als gewichtete Summe.

Wir können diese Berechnung für jeden Knoten im Output Layer durchführen und so für alle Knoten im Hidden Layer einen Fehler berechnen. Abbildung 7 erweitert das Beispiel.

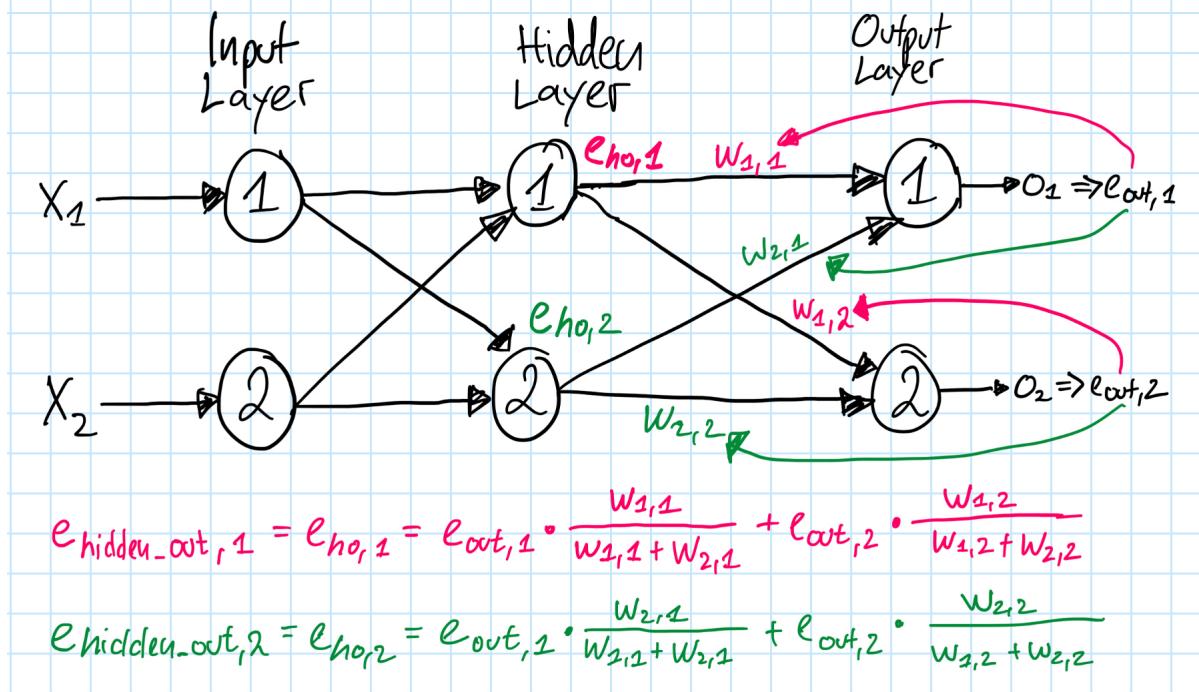


Abbildung 7: Die Fehler $e_{\text{hidden_out},1}$ und $e_{\text{hidden_out},2}$ berechnen sich auf die gleiche Weise.

Bei den Berechnungen ist darauf zu achten, dass die korrekten Gewichte miteinander verrechnet werden. Insbesondere im Nenner in den Brüchen kann schnell ein Fehler passieren. Im Nenner werden jeweils die **eingehenden** Gewichte für den **Fehler im Output Layer** verwendet. Der Zähler ist das **ausgehende** Gewicht vom Knoten der verdeckten Schicht. Auch bei mehreren verdeckten Schichten können wir dieses Prinzip anwenden. Wir propagieren dann einfach die Fehler der **vorhergehenden verdeckten Schicht** (**nicht** immer die Fehler im Output Layer!) weiter in Richtung Input Layer.

1.3.1 Kurzaufgabe

Berechnen Sie die Fehler $e_{\text{hidden_out},1}$ und $e_{\text{hidden_out},2}$ mit folgenden Angaben: $e_{\text{out},1} = 0,8$, $e_{\text{out},2} = 0,5$ und der Gewichtsmatrix $W_{\text{hidden_output}} = \begin{pmatrix} 2,0 & 3,0 \\ 1,0 & 4,0 \end{pmatrix}$

Lösungsvorschlag:

$$\begin{aligned} e_{\text{hidden_out},1} &= e_{\text{out},1} \cdot \frac{w_{1,1}}{w_{1,1} + w_{2,1}} + e_{\text{out},2} \cdot \frac{w_{1,2}}{w_{1,2} + w_{2,2}} = 0,8 \cdot \frac{2,0}{2,0 + 3,0} + 0,5 \cdot \frac{1,0}{1,0 + 4,0} \\ &= 0,32 + 0,1 = 0,42 \\ e_{\text{hidden_out},2} &= e_{\text{out},1} \cdot \frac{w_{2,1}}{w_{1,1} + w_{2,1}} + e_{\text{out},2} \cdot \frac{w_{2,2}}{w_{1,2} + w_{2,2}} = 0,8 \cdot \frac{3,0}{2,0 + 3,0} + 0,5 \cdot \frac{4,0}{1,0 + 4,0} \\ &= 0,48 + 0,4 = 0,88 \end{aligned}$$

1.4 Zusammenfassung

Neuronale Netze lernen, indem ihre Gewichte angepasst werden. Dies wird durch den Fehler gesteuert - die Differenz zwischen der richtigen Antwort, die durch die Trainingsdaten vorgegeben ist, und der tatsächlichen Ausgabe. Der Fehler an den Ausgabeknoten ist einfach die Differenz zwischen der gewünschten und der tatsächlichen Ausgabe. Der den Knoten in den Hidden Layer zugeordnete Fehler ist aber nicht offensichtlich. Eine Möglichkeit ist, die Fehler im Output Layer proportional zur Grösse der verbundenen Gewichte aufzuteilen und dann die jeweiligen Anteile an jedem Konten des Hidden Layers zu addieren.

1.5 Backpropagation von Fehlern mit der Vektor-Matrixmultiplikation

Wir versuchen nun die aufwendigen Berechnungen wieder durch Vektoren und Matrizen darzustellen. Man sagt, dass man den Prozess vektorisiert. Dazu notieren wir zunächst die Fehler im Output Layer und die Fehler im Hidden Layer als Vektoren:

$$\vec{e}_{\text{out}} = \begin{pmatrix} e_{\text{out},1} \\ e_{\text{out},2} \end{pmatrix} \quad \vec{e}_{\text{ho}} = \begin{pmatrix} e_{\text{ho},1} \\ e_{\text{ho},2} \end{pmatrix}$$

Wir müssen nun die Berechnungen für die Fehler aus dem Hidden Layer aus Abbildung 7 nochmals betrachten. Wir können die vier Brüche als Matrix notieren:

$$W_{\text{hidden_output}}^* = W_{\text{ho}}^* = \begin{pmatrix} \frac{w_{1,1}}{w_{1,1}+w_{2,1}} & \frac{w_{1,2}}{w_{1,2}+w_{2,2}} \\ \frac{w_{2,1}}{w_{1,1}+w_{2,1}} & \frac{w_{2,2}}{w_{1,2}+w_{2,2}} \end{pmatrix}$$

Wir können nun das Matrix-Vektor-Produkt anwenden, um die Berechnungen darzustellen:

$$\begin{pmatrix} e_{\text{ho},1} \\ e_{\text{ho},2} \end{pmatrix} = \begin{pmatrix} \frac{w_{1,1}}{w_{1,1}+w_{2,1}} & \frac{w_{1,2}}{w_{1,2}+w_{2,2}} \\ \frac{w_{2,1}}{w_{1,1}+w_{2,1}} & \frac{w_{2,2}}{w_{1,2}+w_{2,2}} \end{pmatrix} \cdot \begin{pmatrix} e_{\text{out},1} \\ e_{\text{out},2} \end{pmatrix} \Leftrightarrow \vec{e}_{\text{out}} = W_{\text{ho}}^* \cdot \vec{e}_{\text{out}}$$

Wir könnten dies überprüfen, indem wir das Matrix-Vektorprodukt berechnen. Wir erhalten dann die Berechnungen aus Abbildung 7. Man kann nun noch eine Vereinfachung vornehmen, um die Matrix W_{ho}^* kompakter zu gestalten. Dazu kann man folgende Überlegung anstellen: Ein grösseres Gewicht bewirkt, dass der Fehler sich stärker auf den Fehler im Hidden Layer auswirkt. Die wesentliche Grössen in der Matrix sind somit die Zähler in den Brüchen. Die Nenner sind "nur" eine Art Normalisierungsfaktor. Wenn wir den Nenner jeweils weglassen, dann verlieren wir nur eine Skalierung der Gewichte. Wir gewinnen jedoch eine vereinfachte Darstellung der Matrix. Neu ist also die Matrix W_{ho}^* wie folgt zu notieren:

$$W_{\text{hidden_output}}^* = W_{\text{ho}}^* = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix}$$

Diese Matrix ist sehr ähnlich zur Matrix der Gewichte zwischen dem Hidden Layer und dem Output Layer:

$$W_{\text{hidden_output}} = W_{\text{ho}} = \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix}$$

Es ist gerade so, dass die Matrix an der Diagonalen gespiegelt ist. Wir erhalten die Matrix $W_{\text{hidden_output}}^*$ durch das Spiegeln der Matrix $W_{\text{hidden_output}}$ an der Diagonalen. Diese Operation nennt sich **Transponieren** und wird wie folgt notiert:

$$W_{\text{hidden_output}}^* = W_{\text{ho}}^* = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix}^T = W_{\text{hidden_output}}^T = W_{\text{ho}}^T$$

Wir können die Fehlerrückführung somit einfach durch folgende Matrix-Vektormultiplikation ausdrücken:

$$\vec{e}_{\text{ho}} = W_{\text{ho}}^T \cdot \vec{e}_{\text{out}}$$

Wir haben nun eine einfache Formel für die Fehlerrückführung entwickelt. Man könnte sich nun noch fragen, ob die Vereinfachung der Matrix durch das Weglassen der Nenner akzeptabel ist oder nicht. Erfahrungen haben gezeigt, dass die vereinfachte Fehlerrückführung genauso gut arbeitet wie die kompliziertere. Das KNN wird sich selbst bei übermäßig grossen oder kleinen zurückgeführten Fehlern während der nächsten Lerniteration selbst korrigieren. Wichtig ist vor allem, dass die zurückgeführten Fehler die Stärke der Gewichte beachten, weil sich hieraus am besten ableiten lässt, wie der Fehler geteilt werden soll.

1.6 Zusammenfassung

Damit wir die Gewichte zwischen zwei Schichten anpassen können, müssen wir einen Fehler berechnen. Für den Output Layer lässt sich der Fehler einfach berechnen. Diese Fehler können benutzt werden, um die Gewichte zwischen dem Hidden Layer und dem Output Layer anzupassen. Für die Gewichte zwischen dem Input Layer und dem Hidden Layer braucht es jedoch auch einen Fehler. Wir können eine Fehlerrückführung durchführen und somit die Fehler anteilmässig aufteilen. Die Fehlerrückführung (eng. error backpropagation) lässt sich als Matrix-Vektormultiplikation ausdrücken. Dadurch können wir ungeachtet der Grösse des KNN prägnanter formulieren und effizienter die Berechnungen durchführen, da Programmiermodule (z.B. NumPy) auf die Berechnung von Matrixmultiplikationen optimiert sind. Nun müssen wir noch klären wie man die Gewichte eigentlich genau aktualisiert. Dies erledigen wir zu einem späteren Zeitpunkt. Nun ist es Zeit für etwas Praxis!

2 Trainingsdaten vorbereiten

In diesem Abschnitt beschäftigen wir uns damit, wie man die Trainingsdaten bestmöglich vorbereitet, die zufälligen Anfangsgewichte festlegt und sogar die Ausgaben konzipiert, damit der Trainingsprozess gute Aussicht auf erfolgreiches Klassifizieren hat.

2.1 Eingaben

Sehen Sie sich nochmals den Graphen der Sigmoid-Aktivierungsfunktion in Abbildung 8 an.

Optimal ist es, wenn die Werte im Bereich der Sigmoidfunktion liegen. Wir sollten die Trainingsdaten also in einen Bereich zwischen 0 und 1 oder noch besser in einen Bereich zwischen 0,01 und 0,99 bringen, da 0 und 1 nie erreicht werden.

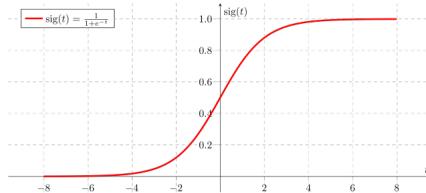


Abbildung 8: Graph der Sigmoidfunktion.

2.2 Ausgaben

Wir müssen die Labels ebenfalls in einen Bereich von 0,01 und 0,99 bringen, da im Output Layer nur diese Ergebnisse entstehen. Hier ist eine Codierung von Klassen zu Ausgabewerten notwendig.

2.3 Zufällige Anfangsgewichte

Die Gewichte sollten zufällig ausgewählt sein und erfahrungsgemäß klein sein. Der Wert 0 ist zu vermeiden. Eine zufällige Wahl im Bereich -1 und $+1$ kann z.B. eingesetzt werden.

2.4 Neuronales Netz mit Python

1. Erstellen Sie eine Python-Datei `neural_network_mnist.py` mit folgendem Codegerüst:

```

1 class NeuralNetwork:
2     def __init__(self):
3         pass
4
5     def train(self, sample_vektor, target_vektor):
6         pass
7
8     def predict(self, sample_vektor):
9         pass

```

2. Es soll ein KNN mit drei Schichten entstehen. Fügen Sie in der `init`-Methode vier Parameter hinzu. Die Anzahl der Knoten für den Input Layer, die Anzahl der Knoten für den Hidden Layer und die Anzahl der Knoten für den Output Layer. Außerdem soll ein Parameter die Lernrate speichern.
3. Initialisieren Sie die beiden Gewichtsmatrizen W_{ih} und W_{hi} mit zufälligen Werten zwischen -0.5 und 0.5 . Verwenden Sie `numpy.rand` und denken Sie an die korrekte Dimension der Matrizen. Verwenden Sie die Attribute aus 2.
4. Fügen Sie den Code für die Vorhersage hinzu in der Methode `predict` hinzu. Programmieren Sie auch eine Ausgabe. Sie können von der letzten Woche profitieren.
5. Programmieren Sie die Trainingsphase in der Methode `train`. Wir müssen uns die Aktualisierung der Gewichte noch anschauen, wir verwenden hier einfach die "fix-fertige-Lösung".
 - (a) Berechnen Sie den Vorrücksschritt wie bei der Vorhersage.
 - (b) Berechnen Sie den Fehler für den Output Layer.
 - (c) Berechnen Sie den Fehler für den Hidden Layer.
 - (d) Aktualisieren Sie die Gewichtsmatrizen wie folgt:

$$\vec{W}_{\text{hidden_output}} = \vec{W}_{\text{hidden_output}} + L \cdot \overrightarrow{e_{\text{out}}} \star \vec{o} \star (1 - \vec{o}) \cdot \vec{o}_{\text{hidden}}^T$$

$$\vec{W}_{\text{input_hidden}} = \vec{W}_{\text{input_hidden}} + L \cdot \overrightarrow{e_{\text{hidden}}} \star \vec{o}_{\text{hidden}} \star (1 - \vec{o}_{\text{hidden}}) \cdot \vec{x}^T$$

Wobei \vec{x} der Eingabevektor mit den Samplewerten darstellt und \star die komponentenweise Multiplikation. Dies übernimmt NumPy automatisch.

6. Laden Sie die MNIST Daten. Trainieren Sie das Netzwerk zunächst mit wenigen Daten. Testen Sie dann das Netzwerk mit weiteren Daten. Verwenden Sie dazu eine **weitere** Python-Datei und erstellen Sie daran das Objekt und laden Sie die Datensätze. Sie müssen darin auch die Daten transformieren und skalieren. Sie können für die Skalierung der Eingabedaten jeden Pixelwert durch 255 teilen und mit 0.99 multiplizieren. Addieren Sie dann noch 0.01.