Fehler auf mehrere Schichten zurückführen

Abbildung 1-67 zeigt ein einfaches neuronales Netz mit drei Schichten – einer Eingabeschicht, einer versteckten Schicht und der Ausgabeschicht.

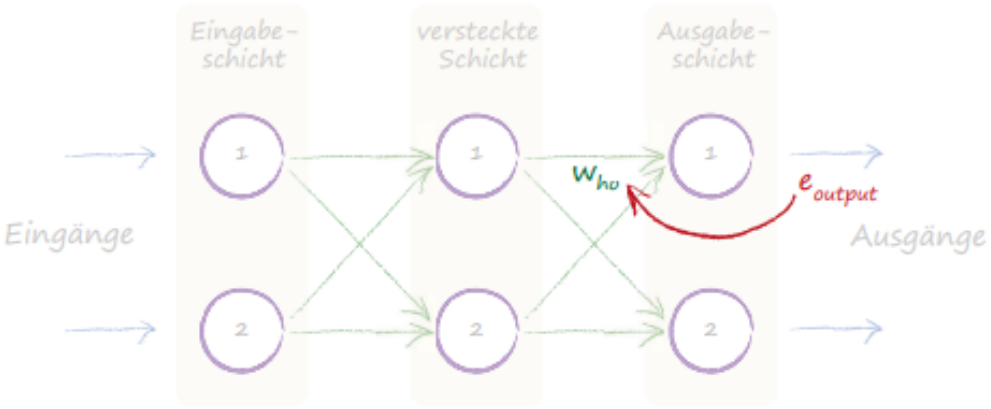

Abbildung 1-67: Beispielnetz mit drei Schichten

Wenn wir von der Ausgabeschicht auf der rechten Seite rückwärtsgehen, sehen wir, dass die Fehler in dieser Ausgabeschicht die Verfeinerung der Gewichte der Verknüpfungen steuern, die in die letzte Schicht eingehen. Wir haben die Ausgabefehler generisch als e_{output} bezeichnet und die Gewichte der Verknüpfungen zwischen der versteckten und der Ausgabeschicht als w_{ho}. Außerdem haben wir die konkreten Fehleranteile ermittelt, die jeder Verknüpfung zugeordnet werden, indem wir die Fehler proportional zur Größe der Gewichte selbst aufgeteilt haben. Abbildung 1-68 veranschaulicht, was wir für die neu hinzugekommene Schicht

unternehmen müssen. Wir nehmen einfach diejenigen Fehler, die den Ausgabeknoten der versteckten Schicht e_{hidden} zugeordnet sind, und teilen diese erneut proportional über die vorhergehenden Verknüpfungen zwischen der Eingabeschicht und der versteckten Schicht w_{ih} auf.

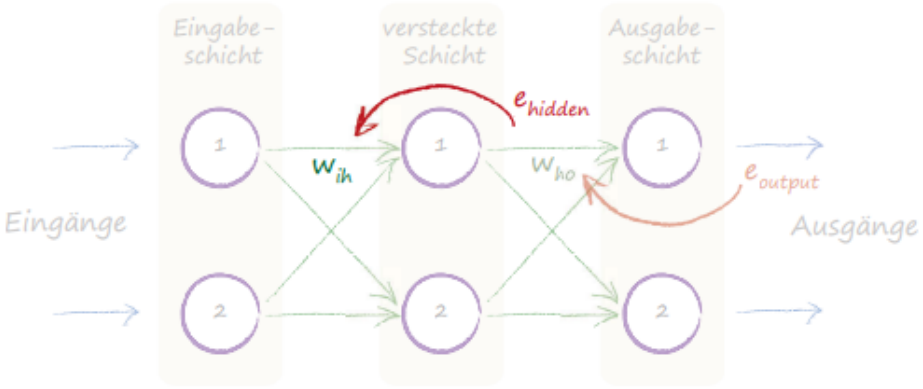

Abbildung 1-68: Aufteilung des Fehlers bis zurück zur versteckten Schicht

Besteht das neuronale Netz aus noch mehr Schichten, wenden wir das gleiche Konzept wiederholt auf jede Schicht an, wobei wir von der Ausgabeschicht aus rückwärtsgehen. Die Rückführung der Fehlerinformationen ist einleuchtend. Auch hier wird klar, warum man von Fehler-*Backpropagierung* spricht.

Wenn wir zuerst den Fehler in der Ausgabe der Ausgabeschichtknoten e_{output} verwenden, welchen Fehler nehmen wir dann für die Knoten der versteckten Schicht e_{hidden}? Gute Frage, weil ein Knoten in der versteckten (mittleren) Schicht keinen offensichtlichen Fehler hat. Wir wissen aus der Vorwärtsführung der Eingabesignale, dass jeder Knoten in der versteckten Schicht in der Tat einen einzelnen Ausgang besitzt. Wie bereits erwähnt, war das die Aktivierungsfunktion, die auf die gewichtete Summe der Eingänge zu diesem Knoten angewendet wurde. Doch wie berechnen wir den Fehler?

Wir haben weder die Ziel- noch die Sollausgänge für die versteckten Knoten, sondern nur die Zielwerte für die Knoten der (letzten) Ausgabeschicht, und diese stammen von den Trainingsbeispielen. Sehen Sie sich noch einmal Abbildung 1-68 an, um sich inspirieren zu lassen! Vom ersten Knoten in der versteckten Schicht gehen zwei Verknüpfungen aus, die den Knoten mit den beiden Knoten der Ausgabeschicht verbinden. Wir wissen, dass wir den Ausgabefehler entlang jeder dieser Verknüpfungen aufteilen können, wie wir es bereits getan haben. Das heißt, wir haben eine Art Fehler für jede der beiden Verknüpfungen, die aus diesem Knoten der mittleren Schicht entspringen. Wir könnten diese beiden Verknüpfungsfehler kombinieren, um den Fehler für diesen Knoten als eine zweitbeste Annäherung zu bilden, weil wir praktisch keinen Zielwert für den Knoten der mittleren Schicht haben. Abbildung 1-69 veranschaulicht diese Idee.

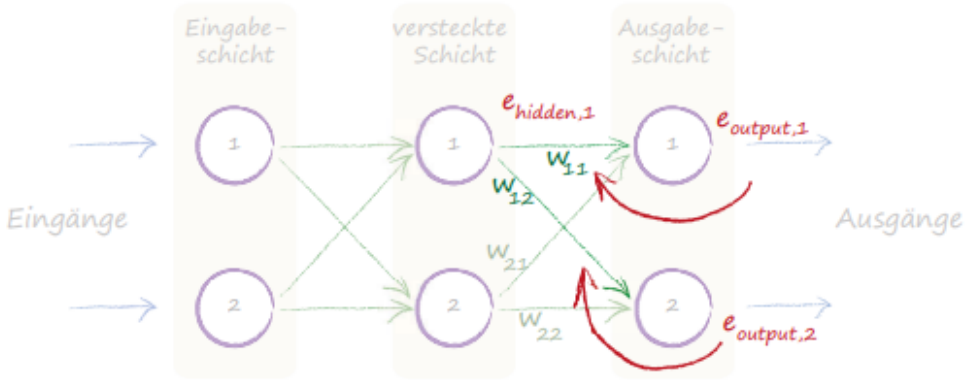

Abbildung 1-69: Aus den Fehlern der Ausgabeknoten die Fehler der Knoten in der versteckten Schicht bilden

Hier ist deutlicher zu sehen, was passiert. Doch wir wollen die Abläufe zur Sicherheit noch einmal durchgehen. Wir brauchen einen Fehler für die Knoten der versteckten Schicht, um mit ihm die Gewichte in der vorhergehenden Schicht aktualisieren zu können. Diese Fehler nennen wir e_{hidden}. Doch wir haben keine offensichtliche Antwort darauf, was die Fehler tatsächlich sind. Wir können nicht sagen, dass der Fehler die Differenz zwischen dem gewünschten Zielwert von diesen Knoten und den tatsächlichen Ausgaben ist, weil unsere Trainingsdatenbeispiele nur Ziele für die allerletzten Ausgabeknoten liefern.

Die Trainingsdatenbeispiele sagen uns lediglich, wie die Ausgaben der allerletzten Knoten sein sollten. Sie sagen uns nicht, wie die Ausgaben von Knoten in irgendeiner anderen Schicht aussehen sollten.

Wir könnten die aufgeteilten Fehler für die Verknüpfungen mithilfe der Fehler-Backpropagierung, die wir bereits kennengelernt haben, zusammenfassen. Somit ist der Fehler im ersten Knoten der versteckten Schicht die Summe der anteiligen Fehler in allen Verknüpfungen, die nach vorn vom selben Knoten ausgehen. In Abbildung 1-69 haben wir einen Anteil vom Ausgabefehler $e_{output,1}$ auf der Verknüpfung mit dem Gewicht w_{11} und außerdem einen Anteil vom Ausgabefehler $e_{output,2}$ vom zweiten Ausgabeknoten auf der Verknüpfung mit dem Gewicht w_{12}.

Wir notieren also:

$$e_{hidden,1} = \text{Summe der aufgeteilten Fehler auf den Verknüpfungen } w_{11} \text{ und } w_{12}$$

$$= e_{output,1} * \frac{w_{11}}{w_{11} + w_{21}} + e_{output,2} * \frac{w_{12}}{w_{12} + w_{22}}$$

Da es hilfreich ist, diese ganze Theorie in Aktion zu sehen, zeigt Abbildung 1-70 die Backpropagierung von Fehlern in einem einfachen dreischichtigen neuronalen Netz mit realen Zahlen.

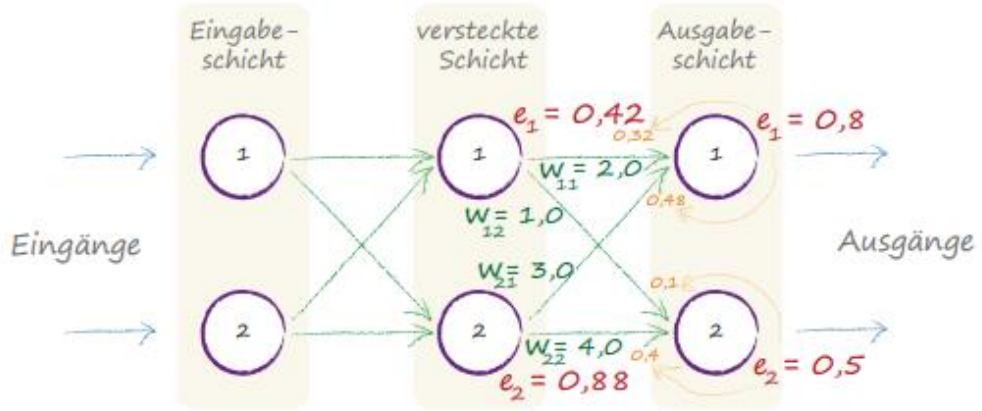

Abbildung 1-70: Das neuronale Beispielnetz mit Fehler-Backpropagierung bis zur verdeckten Schicht

Verfolgen wir einen Fehler zurück. Der Fehler 0,5 beim zweiten Knoten der Ausgabeschicht wird mit 0,1 und 0,4 proportional auf die beiden Verknüpfungen mit den Gewichten 1,0 und 4,0 aufgeteilt. Außerdem zeigt Abbildung 1-70, dass der zusammengefasste Fehler am zweiten Knoten der versteckten Schicht die Summe der verbundenen aufgeteilten Fehler – hier 0,48 und 0,4 – ist, was 0,88 ergibt.

Abbildung 1-71 stellt die gleiche Idee dar, angewandt auf die vorhergehende Schicht, um den Fehler noch weiter zurückzupropagieren.

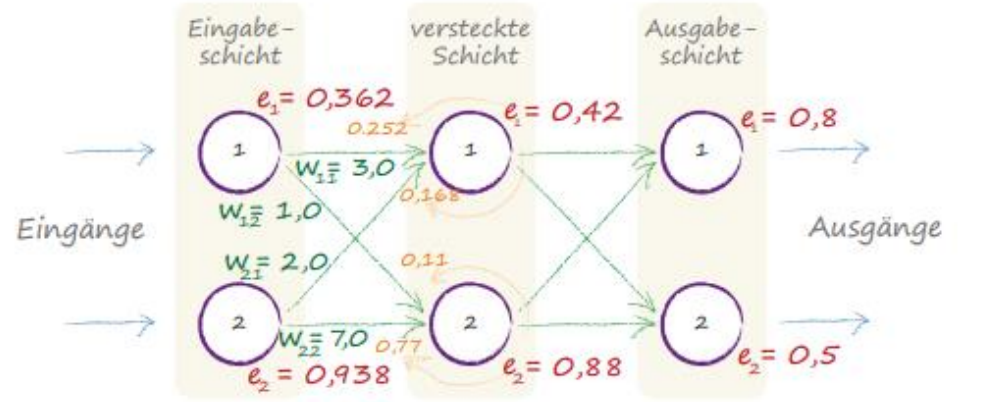

Abbildung 1-71: Das neuronale Beispielnetz mit Fehler-Backpropagierung bis zur Eingabeschicht

Kernideen

- Neuronale Netze lernen, indem ihre Verknüpfungsgewichte verfeinert 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 internen Knoten zugeordnete Fehler ist aber nicht offensichtlich. Eine Möglichkeit ist, die Fehler der Ausgabeschicht proportional zur Größe der verbundenen Verknüpfungsgewichte aufzuteilen und dann die jeweiligen Anteile an jedem internen Knoten zusammenzufassen.

Backpropagierung von Fehlern mit Matrizenmultiplikation

Können wir alle diese aufwendigen Berechnungen per Matrizenmultiplikation vereinfachen? Dieses Verfahren hat bereits weiter oben geholfen, als wir Unmengen von Berechnungen bei der Weiterleitung der Eingabesignale ausgeführt haben.

Um festzustellen, ob sich die Fehler-Backpropagierung mithilfe der Matrizenmultiplikation prägnanter ausdrücken lässt, notieren wir die einzelnen Schritte mit Symbolen. Man sagt auch: Es wird versucht, den Prozess zu *vektorisieren*. Wenn wir in der Lage sind, die zahlreichen Berechnungen in Matrixform auszudrücken, können wir sie viel prägnanter niederschreiben, und Computer können die gesamte Arbeit wesentlich effizienter ausführen, weil sie von den sich wiederholenden Ähnlichkeiten in den erforderlichen Berechnungen profitieren.

Den Ausgangspunkt bilden die Fehler, die sich in der letzten Schicht, der Ausgabeschicht des neuronalen Netzes, ergeben. Im Beispiel haben wir nur zwei Knoten in der Ausgabeschicht, und die Fehler sind e_1 und e_2:

$$error_{output} = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Als Nächstes konstruieren wir die Matrix für Fehler der versteckten Schicht. Da das schwierig zu sein scheint, gehen wir jeden Schritt einzeln durch. Los geht es mit dem ersten Knoten in der versteckten Schicht. Sehen Sie sich noch einmal Abbildung 1-70 an. Hier ist zu erkennen, dass es für den Fehler des ersten versteckten Knotens zwei Verbindungen von der Ausgabeschicht gibt, die Fehleran-

teile liefern. Über diese Verbindungen kommen die Fehlersignale $e_1 \times w_{11} / (w_{11} + w_{21})$ und $e_2 \times w_{12} / (w_{12} + w_{22})$. Auch beim zweiten Knoten der versteckten Schicht sehen wir zwei Verbindungen, die zum Fehler des Knotens beitragen, nämlich $e_1 \times w_{21} / (w_{21} + w_{11})$ und $e_2 \times w_{22} / (w_{22} + w_{12})$. Wir haben bereits weiter oben gesehen, wie diese Ausdrücke ermittelt werden.

Für die versteckte Schicht bekommen wir also die in Abbildung 1-72 gezeigte Matrix. Sie ist ein wenig komplizierter, als ich es mag.

$$error_{hidden} = \begin{pmatrix} \frac{w_{11}}{w_{11} + w_{21}} & \frac{w_{12}}{w_{12} + w_{22}} \\ \frac{w_{21}}{w_{21} + w_{11}} & \frac{w_{22}}{w_{22} + w_{12}} \end{pmatrix} \bullet \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}$$

Abbildung 1-72: Fehlermatrix für die versteckte Schicht

Es wäre großartig, wenn sich diese Form als ganz einfache Multiplikation von Matrizen umschreiben ließe, über die wir bereits verfügen. Das sind die Gewichts-, Vorwärtssignal- und Ausgabefehlermatrizen. Wie bereits erwähnt, ergeben sich damit riesige Vorteile.

Leider können wir dies nicht in eine super einfache Matrizenmultiplikation umwandeln, wie es noch bei der Vorwärtsweiterleitung der Signale möglich war (siehe weiter oben). Die Fehleranteile in dieser großen, voll besetzten Matrix (siehe Abbildung 1-72) sind schwer zu entwirren! Es wäre hilfreich gewesen, wenn wir diese voll besetzte Matrix in eine einfache Kombination der verfügbaren Matrizen hätten aufteilen können.

Was können wir tun? Wir möchten unbedingt diese Vorteile der Matrizenmultiplikation nutzen, um die Berechnungen effizient ausführen zu können.

Zeit also, etwas frecher zu agieren.

Sehen Sie sich noch einmal den Ausdruck in Abbildung 1-72 an. Das Wichtigste ist die Multiplikation der Ausgabefehler e_n mit den verknüpften Gewichten w_{ij}. Je größer das Gewicht, desto stärker wirkt der Ausgabefehler zurück auf die versteckte Schicht. Das ist der entscheidende Teil. Der Nenner dieser Brüche ist eine Art Normalisierungsfaktor. Wenn wir diesen Faktor ignorieren, verlieren wir lediglich die Skalierung der zurückgeführten Fehler. Das heißt, $e_1 \times w_{11} / (w_{11} + w_{21})$ würde zum viel einfacheren Ausdruck $e_1 \times w_{11}$ werden.

Mit dieser Vereinfachung nimmt die Matrizenmultiplikation die in Abbildung 1-73 gezeigte Form an.

Abbildung 1-73: Vereinfachte Gewichtsmatrix

Diese Gewichtsmatrix sieht wie diejenige aus, die wir zuvor konstruiert haben, ist aber entlang einer Diagonalen gespiegelt, sodass das Element rechts oben jetzt links unten liegt und das Element links unten nach rechts oben gewandert ist. Diese Operation heißt *Transponieren* einer Matrix und wird geschrieben als $\mathbf{w}^T$.

Abbildung 1-74 gibt zwei Beispiele für die Transponierung von Matrizen an. Anhand der gewählten Zahlen können Sie deutlich sehen, was passiert. Wie das untere Beispiel zeigt, funktioniert die Transponierung auch für Matrizen, bei denen die Anzahl der Zeilen von der Anzahl der Spalten abweicht.

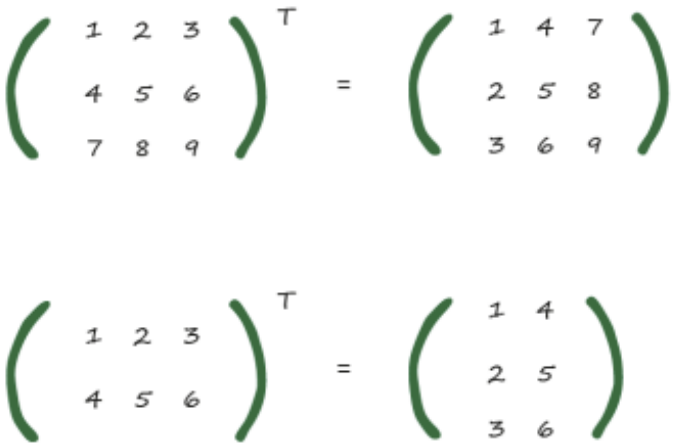

Abbildung 1-74: Beispiele für die Transponierung von Matrizen

Wir haben also das, was wir angestrebt haben – eine Matrixlösung, um die Fehler zurückzuführen (siehe Abbildung 1-75).

Abbildung 1-75: Ausdruck für die Fehler-Backpropagierung in Matrixform

Dies ist zwar großartig, doch war es richtig, diesen Normalisierungsfaktor wegzulassen? Es zeigt sich, dass diese einfachere Rückführung der Fehlersignale genauso gut arbeitet wie die kompliziertere, die wir zuvor ermittelt hatten. Im Blog zu diesem Buch findet sich ein Post-Eintrag, der die Ergebnisse verschiedener Arten der

Fehler-Backpropagierung angibt. Wenn der einfachere Weg wirklich gut funktioniert, behalten wir ihn bei!

Denkt man genauer darüber nach, wird klar, dass sich das Netz selbst bei übermäßig großen oder kleinen zurückgeführten Fehlern während der nächsten Lerniterationen selbst korrigieren wird. Wichtig ist vor allem, dass die zurückgeführten Fehler die Stärke der Verknüpfungsgewichte beachten, weil sich hieraus am besten ableiten lässt, wie der Fehler geteilt werden soll.

Wir haben viel Arbeit bewältigt, wirklich eine riesige Menge!

Kernideen

- Die Fehler-Backpropagierung lässt sich als Matrizenmultiplikation ausdrücken.
- Dadurch können wir ungeachtet der Netzgröße prägnanter formulieren, und Computersprachen, die Matrixberechnungen verstehen, können effizienteren und schnelleren Code erzeugen.
- Das bedeutet, dass sowohl die Signalweiterleitung als auch die Fehlerrückführung mithilfe von Matrixberechnungen effizient ausgedrückt werden können.

Quelle: Neuronale Netze selbst programmieren, Rashid, S. 63-70.