Machine Learning 3: Neuronale Netze V

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

22. November 2021

1 Wie aktualisieren wir die Gewichte?[1]

Bislang haben wir die Frage, wie die Gewichte in einem KNN aktualisiert werden, noch nicht geklärt. Wir haben bis jetzt die Fehler zu jeder Schicht zurückgeführt (Backpropagation). Dies haben wir deshalb gemacht, weil wir uns am Fehler orientieren möchten, wenn es darum geht, die Gewichte zu aktualisieren. Wir möchten die Gewichte so wählen, dass der Fehler (Tatsächlicher Wert minus berechneter Wert) möglichst klein ist (unter der Berücksichtigung aller Trainingsdaten). Zu Beginn der neuronalen Netze haben wir diese Idee mit einem "einfachen" linearen Klassifizierer umgesetzt. Nun haben wir jedoch eine komplexe Struktur: wir summieren Knoten und wenden die Sigmoidfunktion an. Abbildung 1 zeigt dies für die Berechnung von einem Wert am Outputknoten 1.

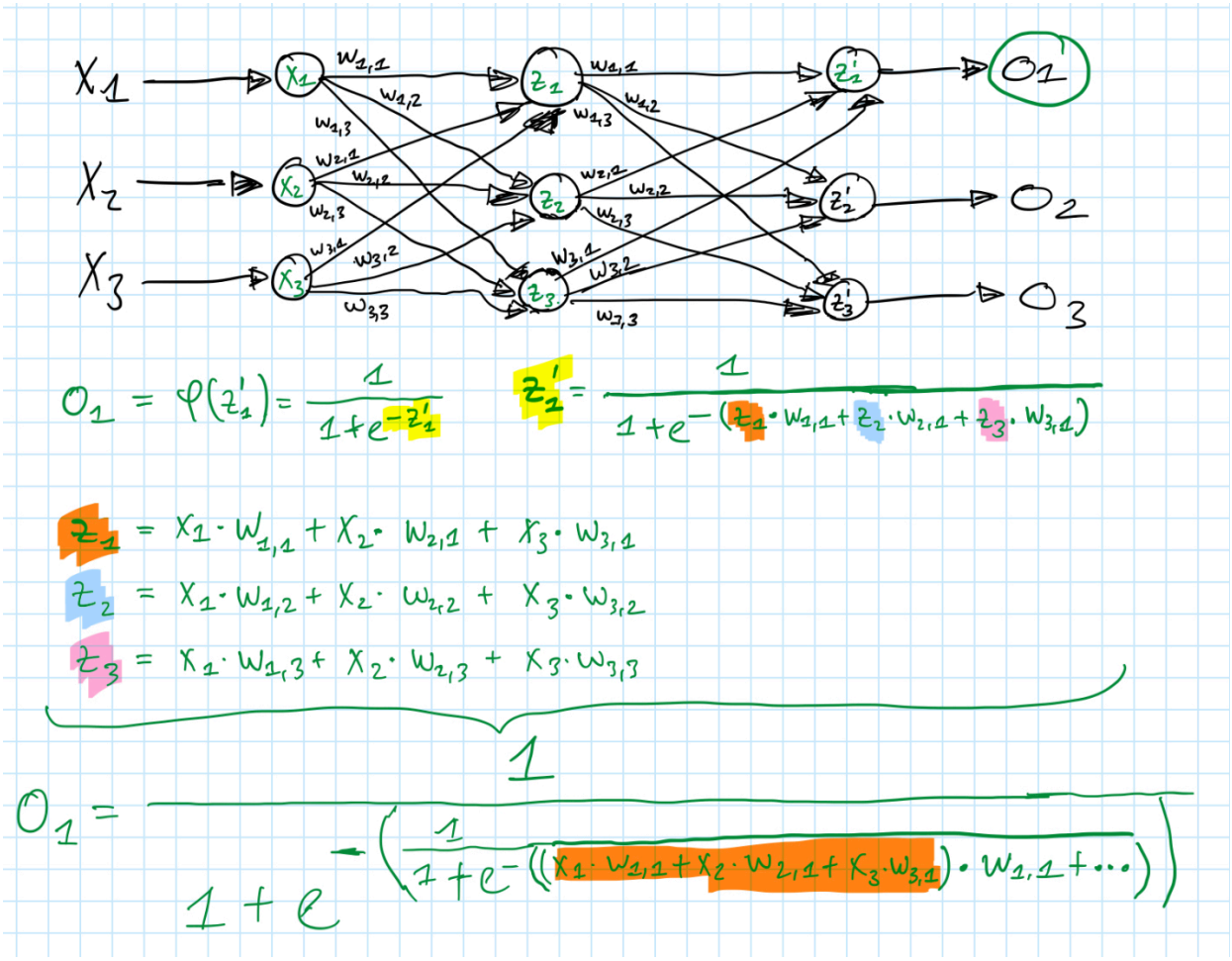

Abbildung 1: Der Exponent der Exponentialfunktion beinhaltet einen Bruch, wobei im Nenner wieder die Exponentialfunktion vorhanden ist. Es ist eine "verschachtelte" Sigmoidfunktion.

Wir müssten für jeden Outputknoten den mathematischen Ausdruck aus Abbildung 1 algebraisch lösen. Wir müssten die Gewichte so wählen, dass der Fehler minimal wird. Dies ist algebraisch zu schwierig, weshalb der "Standardweg" (Minimum suchen durch "Ableiten" und "Null setzen") nicht

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

funktioniert. Man könnte auch verschiedene zufällige Gewichte ausprobieren, jedoch gibt es zu viele Kombinationen. Man kann in keiner vernünftigen Zeit alle Gewichte aus einem gewissen Bereich ausprobieren und dann die besten auswählen. Man hat deshalb eine Methode erfunden, um sich dem Minimum schrittweise zu nähern - das Gradientenverfahren.

2 Gradientenverfahren

Bevor wir uns die Mathematik hinter diesem Verfahren anschauen, erklären wir am Beispiel eines Bergsteigers das Prinzip. Stellen Sie sich vor, wie Sie auf einem Berg stehen und versuchen, den Weg nach unten zu finden. Es ist eine komplizierte Landschaft mit zahlreichen Bergspitzen und Tälern. Ausserdem ist es finster und man kann fast nichts sehen. Eine Taschenlampe kann immer nur die nächste Umgebung sichtbar machen. Die beste Methode in das Tal zu gelangen, ist es nun Schritt-für-Schritt mit der Taschenlampe die Umgebung nach dem steilsten Abstieg zu untersuchen und in diese Richtung zu gehen. Diese Richtung ist am aussichtsreichsten, wenn es darum geht in das Tal zu kommen. Mit diesem Verfahren können Sie, ohne eine vollständige Karte zu besitzen oder im Voraus eine Route geplant zu haben, den Weg ins Tal finden. Es kann aber auch schiefgehen...lokales Minimum...

Ähnlich funktioniert das **Gradientenverfahren** (eng. gradient descent) beim Versuch, das Minimum einer Funktion zu finden. Es ist ein Optimierungsverfahren und arbeitet wie folgt: Das Verfahren beginnt an einer zufälligen Stelle im Parameterraum (dies sind bei uns die Gewichte) und reduziert dann iterativ (schrittweise) den Fehler, bis das Verfahren ein lokales Minimum erreicht. Bei jedem Schritt der Wiederholung bestimmt es die Richtung des steilsten Abstiegs und macht einen Schritt entlang dieser Richtung. Abbildung 2 zeigt ein Beispiel für den 1-D Fall.

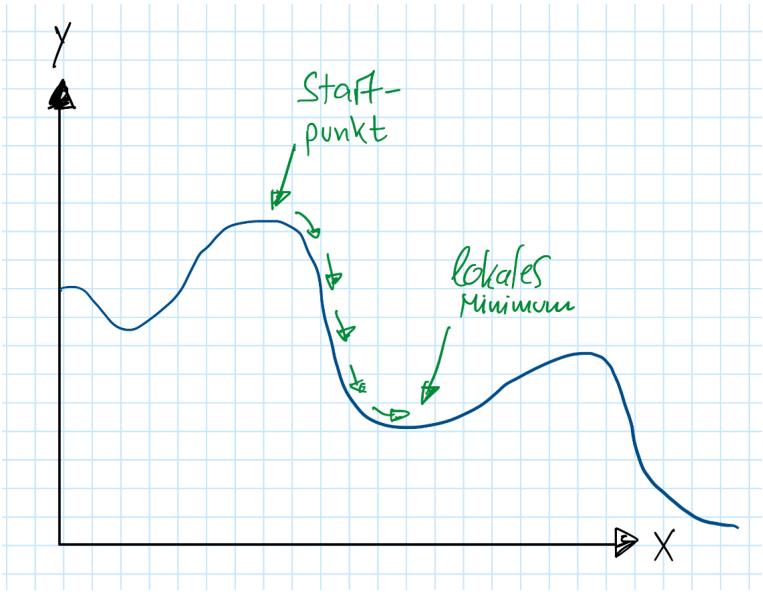

Abbildung 2: Das Gradientenverfahren kann in einem lokalen Minimum festsitzen.

Wir können das Gradientenverfahren nun zur Bestimmung der Gewichte verwenden.

Wichtig!. *Wir ermitteln eine Funktion für den Fehler im KNN. Diese ist analytisch schwierig zu lösen, das heisst, wir können das Minimum nicht exakt bestimmen. Das Gradientenverfahren sucht jedoch schrittweise ein Minimum für eine Funktion. Wir können als die Fehlerfunktion schrittweise in Richtung des steilsten Abstiegs "absteigen" und dadurch die Ausgabewerte verbessern.*

Wir zeigen nun an einem (abstrakten) Beispiel, wie man sich dem Minimum schrittweise nähern kann mit dem Gradientenverfahren. Wir verwenden dabei eine "einfache" Funktion, für die man auch algebraisch das Minimum bestimmen könnte. Wir möchten jedoch nun als Alternative das Gradientenverfahren verwenden. Abbildung 3 zeigt den Graphen (grün) der quadratischen Funktion $f(x) = (x-1)^2 - 2$. Wir verwenden eine vertraute Funktion, damit keine zusätzliche Komplexität entsteht. Bevor wir die Formel für das Gradientenverfahren präsentieren, möchten wir uns mit der

Ableitung der Funktion beschäftigen. Die Ableitung von $f(x)$ lautet $f'(x) = 2 \cdot x + 1$. Die Ableitung "zeigt" immer in "die Richtung", in der die Funktion wächst.

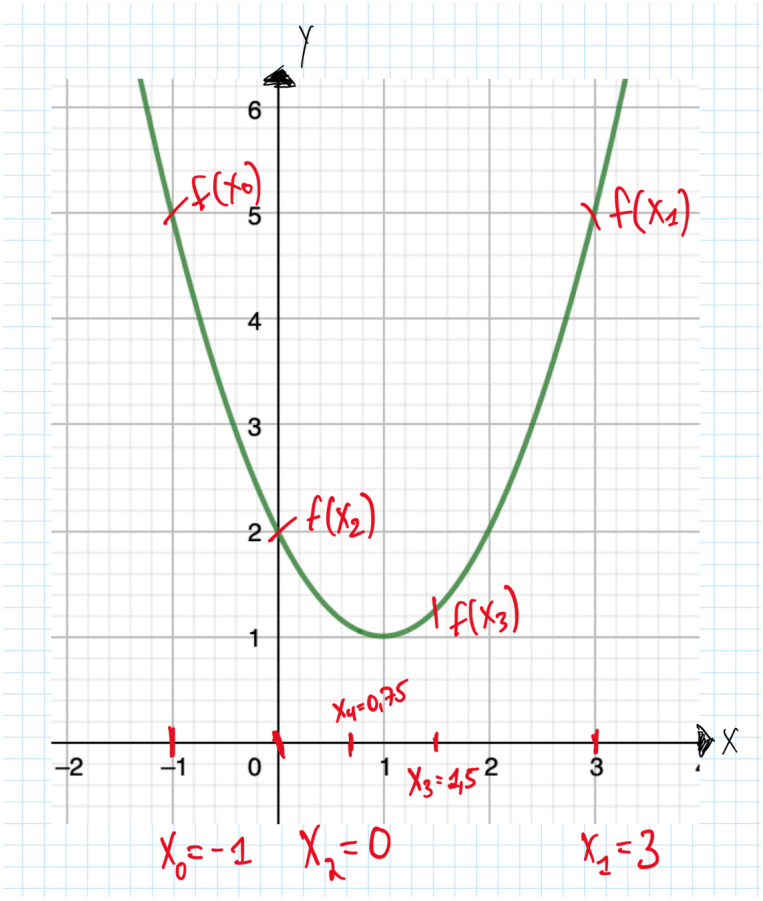

Abbildung 3: Quadratische Funktion mit Annäherung $(x_0, x_1, x_2, \dots)$ an das Minimum bei $x = 1$.

Wenn wir nun die Ableitung bei $x_0 = -1$ auswerten, dann erhalten wir $f'(x_0) = f'(-1) = 2 \cdot (-1) - 2 = -4$. Wenn wir uns nun in "die Richtung der Ableitung" bewegen, muss der Funktionswert grösser werden, da die Ableitung in "die Richtung", in der die Funktion wächst, "zeigt".

Beispiel: Falls $x' = x + f'(x)$, dann muss $f(x')$ grösser sein als $f(x)$. Konkret: $x' = x_0 + f'(x_0) = -1 + f'(-1) = -1 + (-4) = -5$ und $f(x') = f(-5) = (-5 - 1)^2 + 1 = (-6)^2 + 1 = 37$. Da $f(x_0) = f(-1) = 5$ und $f(x') = f(-5) = 37$, wird unsere Beobachtung durch das Beispiel unterstützt.

Da wir uns nun somit mit $f'(x)$ in "die Richtung bewegen, in der die Funktion am grössten wächst" (und somit "weg" vom Minimum), müssen wir für einen neuen X-Wert in "umgekehrter Richtung laufen". Wir erhalten also den nächsten x-Wert dadurch, in dem wir vom aktuellen x-Wert die Ableitung abziehen:

$$x_1 = x_0 - f'(x_0)$$

Wir bewegen uns somit direkt entgegen der Ableitung bei x_0. Dies ist dann die Richtung, bei der die Funktion am grössten "schrumpft" - somit auf dem Weg zu einem (lokalen) Minimum. Wir möchten jedoch den nächsten x-Wert durch eine Schrittgrösse α_0 steuern. Diese soll dazu dienen, dass wir uns mit grösseren Schritten dem Minimum nähern, wenn wir weiter weg sind und mit kleineren Schritten nähern, wenn wir nahe beim Minimum sind. Wir erhalten somit:

$$x_1 = x_0 - \alpha_0 \cdot f'(x_0)$$

Die Wahl von α_0 ist nicht einfach. Während der Suche soll sich α_0 ebenfalls anpassen können. Wir verwenden im folgenden Beispiel für den Start $\alpha_0 = 1$ und reduzieren α_0 um $0,25$, falls wir "über das Minimum hinaus geschossen sind". Das Gradientenverfahren im 1-Fall lautet somit:

$$x_{k+1} = x_k - \alpha_k \cdot f'(x_k)$$

Wir zeigen nun Schritt-für-Schritt wie man die x-Werte aus Abbildung 3 berechnet.

x_k	$f(x_k) = (x_k - 1)^2 + 1$	α_k	$f'(x_k) = 2 \cdot x_k - 2$
$x_0 = -1$	$f(x_0) = 5$	$\alpha_0 = 1$	$f'(x_0) = -4$
$x_1 = x_0 - \alpha_0 \cdot f'(x_0) = -1 - 1 \cdot (-4) = 3$	$f(x_1) = 5$	$\alpha_1 = 0,75$	$f'(x_1) = 4$
$x_2 = x_1 - \alpha_1 \cdot f'(x_1) = 3 - 0,75 \cdot 3 = 0$	$f(x_2) = 2$	$\alpha_2 = 0,75$	$f'(x_2) = -2$
$x_3 = x_2 - \alpha_2 \cdot f'(x_2) = 0 - 0,75 \cdot -2 = 1,5$	$f(x_3) = 1,25$	$\alpha_3 = 0,75$	$f'(x_3) = 1$
$x_4 = x_3 - \alpha_3 \cdot f'(x_3) = 1,5 - 0,75 \cdot 1 = 0,75$	$f(x_4) = 1,0625$	$\alpha_4 = 0,75$	$f'(x_4) = -0,5$
$x_5 = x_4 - \alpha_4 \cdot f'(x_4) = 1,125$	$f(x_5) = 1,015625$	$\alpha_5 = 0,75$	$f'(x_5) = 0,25$

Tabelle 1: Wir reduzieren α in der 2. Zeile, da sich der Funktionswert nicht verändert hat. Wir sind vielleicht über das Minimum "hinausgeschossen". Danach "sinkt" der Funktionswert $f(x_k)$ und wir belassen α - wir nähern uns dem Minimum.

Da es sich um ein iteratives Verfahren zur Suche eines (lokalen) Minimums handelt, müssen wir ein Abbruchkriterium festlegen. Hier kann man z.B. $f'(x_k) < \epsilon$ mit $\epsilon > 0$ festlegen. Man kann z.B. mit $\epsilon = 0,1$ sagen, dass man stoppt, wenn die Ableitung bei x_k weniger als $0,1$ beträgt. Man hat also quasi "keine Steigung" mehr - ein Minimum. Tabelle 1 zeigt die ersten 5 Schritte des Verfahrens.

Bei mehreren Dimensionen müssen wir von jeder Dimension die Ableitung bilden - wir erhalten den Gradienten. Dies ist ein Vektor der in die Richtung zeigt, in der die Funktion am grössten wächst. Wir können unsere bisherigen Überlegungen einfach auf mehrere Dimensionen übertragen:

$$\overrightarrow{x_{k+1}} = \vec{x_k} - \alpha_k \cdot \nabla f(x_k)$$

Der "Nabla"-Operator ∇ ist ein mathematischer Operator für "das Ableiten im mehrdimensionalen Raum". Wir erhalten dadurch den Gradientenvektor.

Beispiel: Wir haben eine mehrdimensionale Funktion $f(x, y) = 2 \cdot x^2 - y^2$ und können davon die partiellen Ableitungen $4 \cdot x$ und $-2 \cdot y$ bilden. Wir erhalten den Gradientenvektor $\nabla f(x, y) = \begin{pmatrix} 4 \cdot x \\ -2 \cdot y \end{pmatrix}$

Wir können also mit dem Gradientenverfahren ein Minimum schrittweise auch für mehrdimensionale Funktionen ermitteln. Es kann jedoch sein, dass wir in einem lokalen Minimum "festsitzen". Deshalb wählt man unterschiedliche Startwerte.

3 Gradientenverfahren und KNN

Die Ausgabe eines neuronalen Netzes ist eine komplexe und schwierige Funktion, deren Rückgabewert von vielen Gewichten beeinflusst wird. Können wir überhaupt mit dem Gradientenverfahren die richtigen Gewichte ermitteln? Ja, sofern wir uns für die richtige Fehlerfunktion entscheiden.

Die Ausgabefunktion eines KNN ist selbst keine Fehlerfunktion. Doch wir wissen, dass wir sie leicht in eine Fehlerfunktion umwandeln können, weil sich der Fehler aus der Differenz zwischen den Solltrainingswerten und den tatsächlichen Ausgabewerten berechnet.

Wir verwenden dabei das Quadrat $(Soll - Ist)^2$ als Fehlerfunktion, da ...

- ... die Algebra für die Bestimmung des Gradienten leicht genug ist.
- ... die Funktion stetig verläuft, was einen reibungslosen Verlauf des Gradientenverfahrens ermöglicht.
- ... der Gradient in der Nähe des Minimums kleiner wird, so dass es weniger wahrscheinlich wird, dass wir über das Minimum "hinausschiessen".

Für einen Fehler e_k ergibt sich nun folgende Fehlerfunktion:

$$f(e_k) = (t_k - o_k)^2$$

Abbildung 4 zeigt ein einfaches neuronales Netz mit eingezeichneten Fehlerberechnungen.

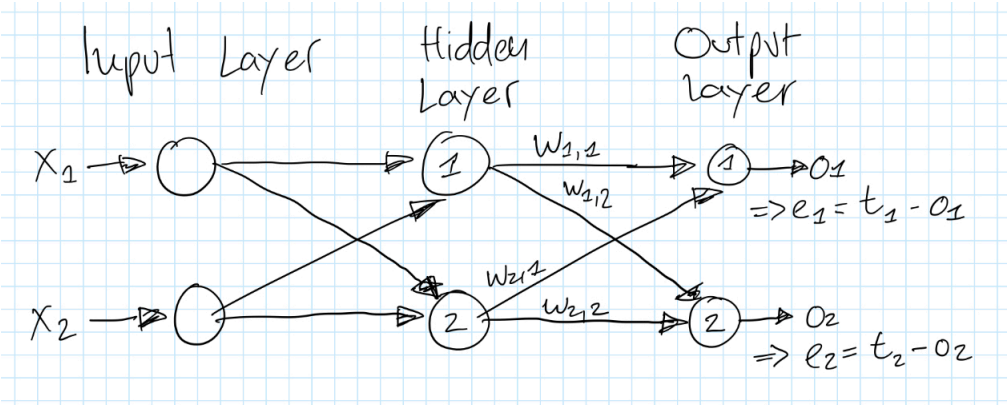

Abbildung 4: Pro Knoten im Output Layer kann man einen Fehler berechnen.

Wir versuchen nun die Gewichte zwischen Hidden Layer und Output Layer zu bestimmen, in dem wir die Fehlerfunktion für einen Fehler e_k nach w_{jk} ableiten. Wir interessieren uns dafür, wie sich der Fehler ändert, wenn wir das Gewicht ändern. Deshalb leiten wir nach w_{jk} ab. Wir müssen mehrere Gewichte beim Ableiten berücksichtigen, da o_k aus mehreren Gewichten gebildet wird:

$$\begin{aligned}
\frac{\partial f(e_k)}{\partial w_{jk}} &= \frac{\partial f(e_k)}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{jk}} && \text{(Kettenregel)} \\
&= \frac{\partial (t_k - o_k)^2}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{jk}} && \text{(Funktion einsetzen)} \\
&= \frac{\partial (t_k^2 - 2 \cdot t_k \cdot o_k + o_k^2)}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{jk}} && \text{(Binom ausrechnen)} \\
&= (-2 \cdot t_k + 2 \cdot o_k) \cdot \frac{\partial o_k}{\partial w_{jk}} && (o_k \text{ ableiten}) \\
&= (-2 \cdot t_k + 2 \cdot o_k) \cdot \frac{\partial (\phi(z))}{\partial w_{jk}} && (o_k \text{ ist Sigmoid von gewichteter Summe } z) \\
&= (-2 \cdot t_k + 2 \cdot o_k) \cdot \phi(z) \cdot (1 - \phi(z)) \cdot \frac{\partial (z)}{\partial w_{jk}} && \text{(Sigmoid ableiten)}
\end{aligned}$$

Da $z = w_{1k} \cdot o_{1h} + w_{2k} \cdot o_{2h} + \cdots + w_{jk} \cdot o_{jh} + \cdots + w_{nk} \cdot o_{nh}$ ergibt $\frac{\partial (z)}{\partial w_{jk}}$ gerade o_{jh}. Hier ist zu beachten, dass o_{jh} die Ausgabe nach der versteckten Schicht darstellt (h für "hidden"). Wir können die konstanten Faktoren streichen, da wir primär an der Richtung des Anstiegs der Fehlerfunktion interessiert sind. Somit erhalten wir für den Gradienten der Fehlerfunktion folgendes Ergebnis:

$$\frac{\partial f(e_k)}{\partial w_{jk}} = -(t_k + o_k) \cdot \phi(z) \cdot (1 - \phi(z)) \cdot o_{jh}$$

$$\phi(z) = \sum_{j=1}^{n} w_{jk} \cdot o_{jh}$$

Wir aktualisieren die Gewichte nun gemäss dem Gradientenverfahren:

$$\begin{aligned} w_{jk}^{neu} &= w_{jk}^{alt} - L \cdot \frac{\partial f(e_k)}{\partial w_{jk}} \\ &= w_{jk}^{alt} - L \cdot (-(t_k + o_k) \cdot \phi(z) \cdot (1 - \phi(z)) \cdot o_{jh}) \\ \phi(z) &= \sum_{j=1}^{n} w_{jk} \cdot o_{jh} \end{aligned}$$

Wir können diese Berechnung auch für die Gewichte zwischen dem Input Layer und dem Hidden Layer durchführen. Da das Problem symmetrisch ist, kommen wir zum gleichen Ergebnis. Wir können nun noch das Ergebnis in der üblichen Vektor-Matrix-Schreibweise notieren, damit eine einfache Eingabe im Programm erfolgen kann.

$$\begin{aligned} \vec{W}_{\text{hidden\_output}} &= \vec{W}_{\text{hidden\_output}} + L \cdot \overrightarrow{e_{\text{out}}} \star \vec{o} \star (1 - \vec{o}) \cdot \vec{o}_{\text{hidden}}^{T} \\ \vec{W}_{\text{input\_hidden}} &= \vec{W}_{\text{input\_hidden}} + L \cdot \overrightarrow{e_{\text{hidden}}} \star \vec{o}_{\text{hidden}} \star (1 - \vec{o}_{\text{hidden}}) \cdot \vec{x}^{T} \end{aligned}$$

Wobei $\vec{x}$ der Eingabevektor mit den Samplewerten darstellt und $\star$ die komponentenweise Multiplikation.