

Inhaltsverzeichnis

1	Numerische Ableitung	3
1.1	Einleitung	3
1.2	Problemstellung	3
1.2.1	Was ist Lernen?	3
1.2.2	Der Parameterraum	4
1.2.3	Das Optimierungsproblem	6
1.2.4	Der Backpropagation Algorithmus	7
1.3	Lösung	8
1.3.1	Finite Differenzen Methode	8
1.3.2	Taylor Methode	9
1.4	Resultate	11
1.5	Folgerungen	13

Kapitel 1

Numerische Ableitung

1.1 Einleitung

Dieses Kapitel befasst sich mit der numerischen Ableitung im Speziellen mit der Finite Differenzen Methode (FDM). Bei dieser Methode handelt es sich um ein Verfahren zur Bestimmung der Ableitung. Die Methode war schon Gauss und Boltzmann bekannt, war aber bis in die 1940er nicht sehr verbreitet um angewandte Probleme zu lösen. Das Verfahren ist mathematisch sehr einfach zu erklären und ist ebenfalls einfach umzusetzen. Die Einfachheit der Methode ermöglicht ein schwieriges Beispiel zu verwenden, da die Umsetzung in Code unkompliziert ist.

Als Einführung der Methode wird aus diesem Grunde der Gradient im neuronalen Netzwerk berechnet, welcher von zentraler Rolle für den Lernprozess ist. Neuronale Netzwerke sind durch die kostengünstig verfügbare Computerleistung in den letzten Jahrzehnten sehr populär geworden. Moderne Forschung in diesem Bereich konzentriert sich auf eine Vielzahl von Problemen, meistens im Bereich der Bild, Signal oder Mustererkennung, aber auch Sprachsynthese und Übersetzung können von neuronalen Netzwerken profitieren. Eine weitere Konsequenz der günstigen Computerleistung ist, dass die Netzwerke immer grösser werden und durch die wachsende Tiefe auch anfälliger auf numerische Probleme sind.

1.2 Problemstellung

Die Optimierung von neuronalen Netzen wird auch als 'lernen' bezeichnet und funktioniert mittels Gradientenabstiegsverfahren. Nach jeder vollständigen Iteration über den gesamten Datensatz (Epoche) wird der Gradient neu berechnet, um den Fehler zu minimieren. Die Gradientenberechnung funktioniert in den meisten neuronalen Frameworks mittels Backpropagation. Die Stärke des Backpropagation-Algorithmus ist, dass die Gradientenberechnung auf moderner Hardware sehr rasch vollzogen werden kann. Gleichzeitig ist aber ein Nachteil, dass die errechnete Ableitung anfällig auf Rauschen der Daten sein kann. Aus diesem Grund ist es spannend, einen anderen Ansatz zu wählen, welcher in den kommenden Abschnitten beschrieben wird.

1.2.1 Was ist Lernen?

Anders als das menschliche Lernen lernt das neuronale Netzwerk durch den Vergleich einer errechneten Ausgabe mit der erwarteten Ausgabe. Neuronale Netzwerke sind in der Regel 'supervised

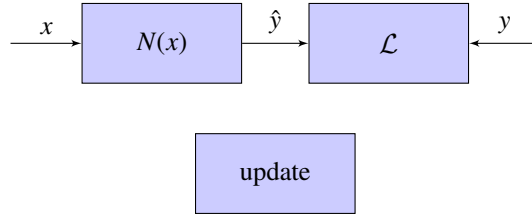


Abbildung 1.1: Trainieren eines neuronalen Netzwerks

learning'-Algorithmen, diese Gruppe von Algorithmen benötigen einen Datensatz, welcher aus einem Input x und einem erwarteten Output y besteht. Das neuronale Netzwerk als Blackbox versucht aus dem Input x einen brauchbaren Output \hat{y} zu erzeugen, welcher dann mittels einer Fehlerfunktion \mathcal{L} zum bestehenden Output y verglichen wird. Der Lernprozess kann der Abbildung ?? entnommen werden. Nach jeder Trainingsiteration werden die inneren Parameter des neuronalen Netzwerks $N(x)$ optimiert um das bestmögliche Resultat für die gegebenen Datenpaare (x, y) zu errechnen. Die Bewertung des aktuellen Lernstands ist mittels der Fehlerfunktion

$$\mathcal{L}(y_i, \hat{y}_i) \quad (1.1)$$

gegeben. Dabei gilt, dass

$$\hat{y} = N(x) \quad (1.2)$$

dem berechneten Output des neuronalen Netzwerks $N(x)$ entspricht. Dies lässt sich in die Fehlerfunktion

$$\mathcal{L}(y, N(x)) \quad (1.3)$$

einsetzen. Die Fehlerfunktion \mathcal{L} wird in der Literatur auch oft *Loss*-Funktion genannt. Das Resultat ist eine qualitative Aussage über den Fehler, auch *loss* genannt. Als einfaches Beispiel für eine solche Fehlerfunktion \mathcal{L} kann die mittlere quadratische Abweichung

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2} \sum_i (\hat{y} - y)^2 \quad (1.4)$$

(MSE) verwendet werden. Diese hat die Eigenschaft, dass sie Vorzeichen unabhängig ist und bei grösserer Distanz zwischen y und \hat{y} der Fehler auch dementsprechend grösser wird. Die Aufgabe des Lernprozesses ist es nun die bestmögliche Lösung für den Fehler zu finden. Entsprechen gilt es für

$$\min(\mathcal{L}(y, \hat{y})) \quad (1.5)$$

eine optimale Lösung zu finden. Da x und y durch den Datensatz gegeben sind, muss nun auf den inneren Parameterraum des neuronalen Netzwerks eingegangen werden.

1.2.2 Der Parameterraum

Das neuronale Netzwerk wird durch eine sehr hohen Anzahl an Parametern charakterisiert, welche im Verlauf des Lernprozesses optimiert werden müssen. Um zu verstehen, woher diese Parameter stammen, muss nun der innere Aufbau eines neuronalen Netzwerks betrachtet werden. Anhand eines

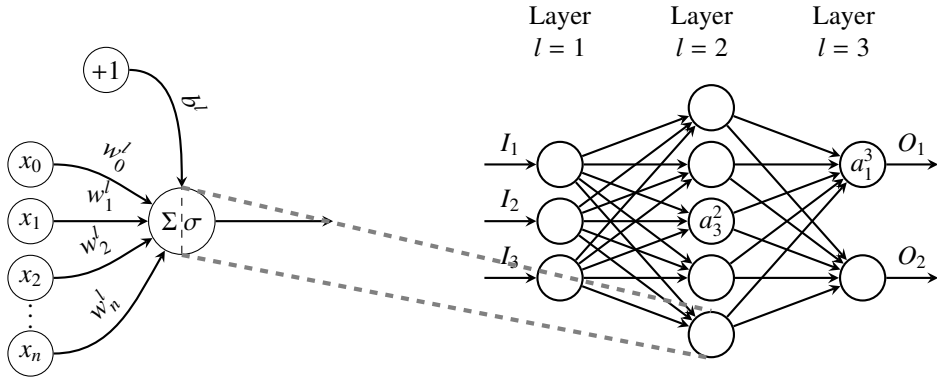


Abbildung 1.2: Übersicht eines Neuronalen Netzwerks

einfachen neuronalen Netzwerks gemäss Abbildung 1.2 können die Komponenten eingeführt werden. W sind die Gewichte, b die Offsets, oft auch *bias* genannt, und σ ist die Aktivierungsfunktion. Die Aktivierungsfunktion wird benötigt um die Linearität der verschiedenen Schichten zu brechen. In der letzten Schicht wird die Aktivierungsfunktion σ zusätzlich verwendet um den Output auf das vorliegende Problem abzubilden. Eine lineare Ausgabe zwischen $[-\infty, \infty]$ ist erwünscht, sofern ein Regressionsproblem vorliegt. Bei einem stochastischen Problem werden die Outputwerte als Wahrscheinlichkeiten interpretiert und sollten daher den Wertebereich $[0, 1]$ annehmen, entsprechend wird eine lineare Funktion für σ gewählt. Die Gleichung

$$a_j^l = \sigma \left(\sum_k w_{jk}^l \cdot a_k^{l-1} + b_j^l \right) \quad (1.6)$$

beschreibt somit für jedes Neuron a_j^l aus Abbildung 1.2 den Zusammenhang zwischen den Gewichten und den vorhergehenden Neuronen a_k^{l-1} . Der Input Layer

$$I_k = a_k^0 \quad (1.7)$$

nimmt einen Daten x_i des Datensatzes x entgegen. Da die Neuronen mit den vorangegangenen Schichten gekoppelt sind, kann man die Gleichung (1.6) für die verschiedenen Schichten ineinander einsetzen und abkürzend

$$\hat{y}(x) = \sigma^l(W^l \sigma^{l-1}(W^{l-1} \dots \sigma^1(W^1 x + b^1) + b^{l-1} \dots)) \quad (1.8)$$

schreiben. Dementsprechend resultiert daraus das Minimalproblem

$$\min \{ \mathcal{L}(y_i, \sigma^l(W^l \sigma^{l-1}(W^{l-1} \dots \sigma^1(W^1 x + b^1) + b^{l-1} \dots)) \}. \quad (1.9)$$

Wobei w und b variabel sind und x und y durch den Trainings-Datensatz gegeben sind. Um die Nomenklatur zusammenzufassen gelten folgende Variablen und Parameter in einem neuronalen Netzwerk:

- x : Input ins Neuronale Netzwerk

- y : Erwarteter Output aus dem Neuronalen Netzwerk gem. dem Trainingsdatensatz
- \hat{y} : Effektiver gerechneter Output aus den neuronalen Netzwerk
- \mathcal{L} : Die Verlustfunktion (*loss*) welche den Fehler zwischen dem gerechneten und tatsächlichem Output berechnet.
- w_{jk}^l : Das Gewicht des k -ten Neurons in der Schicht $(l - 1)$ zum j -ten Neuron.
- b_j^l : Das Offset-Gewicht (bias) für das j -te Neuron in der Schicht l .
- σ : Die Aktivierungsfunktion zur Brechnung der Linearität.

Das Trainieren oder Lernen eines neuronalen Netzwerks ist im Grunde nichts anderes als das bestimmen des Minimums des Fehlers mit Hilfe der Parameter w und b für die gegebenen Problemstellung.

1.2.3 Das Optimierungsproblem

Wie bereits erwähnt, ist durch die extrem hohe Anzahl an Parametern ein Durchprobieren aller Parameterkombinationen nicht möglich. Es würde kein Resultat in nützlicher Frist gefunden werden können. Aus diesem Grund werden andere Verfahren verwendet, um das neuronale Netzwerk zu optimieren. Alle diese Verfahren basieren auf dem Gradientenabstieg und werden in der Literatur oft auch 'Optimizer' genannt. Näheres zum Gradientenabstieg kann in Kapitel xyz gefunden werden. Ein einfacher Algorithmus ist der stochastische Gradientenabstieg. Der Abstiegsschritt

$$p_{new} := p - \eta \nabla \mathcal{L}_i(p) \quad (1.10)$$

findet Parameterwerte p_{new} , für die der Fehler kleiner ist. Die Parameter p stehen für sämtliche inneren Gewichte w und Offsetwerte b . Die Schrittweite η muss experimentel ermittelt und wird oft auch als *learning rate* bezeichnet. Die Kunst liegt darin η so zu wählen, dass lokale Minimas und Rauschen die Konvergenz nicht behindern. Der Algorithmus bricht erst ab, wenn ein Minima erreicht ist oder die Anzahl an Iterationen ausgeschöpft ist. Die Fehlerfunktion

$$\mathcal{L}(w_1 \cdots w_n, b_1 \cdots b_n) = \mathcal{L}(p) = \sum_{i=1}^n \mathcal{L}_i(p) = \sum_{i=1}^n \mathcal{L}(\hat{y}_i - y_i)^2 \quad (1.11)$$

berechnet den Fehler, welcher durch die w und b beeinflusst wird. Aus Gründen der Einfachheit wird hier das vollständiges Einsetzen des Ausdrucks für das neuronale Netzwerk aus (1.8) weggelassen. Um das Beispiel zu veranschaulichen, wird ein einfaches neuronales Netzwerk mit zwei Parametern gewählt. Das Netzwerk

$$\hat{y} = \sigma(w_1 \cdot x + b_1) \quad (1.12)$$

mit den Parametern w_1 und b_1 soll hier als einfaches Beispiel dienen.

Dieses Netzwerk kann eingesetzt werden in die Fehlerfunktion \mathcal{L} . Die anschließende Differenziation nach den inneren Parametern w und b führt zum Gradienten der beiden Parametern. Die neuen Parameter

$$\begin{bmatrix} w_1 \\ b_1 \end{bmatrix}_{new} := \begin{bmatrix} w_1 \\ b_1 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial}{\partial w_1} (\sigma(w_1 x_i + b_1) - y_i)^2 \\ \frac{\partial}{\partial b_1} (\sigma(w_1 x_i + b_1) - y_i)^2 \end{bmatrix} = \begin{bmatrix} w_1 \\ b_1 \end{bmatrix} - \eta \begin{bmatrix} 2((\sigma(w_1 x_i + b_1) - y_i) \cdot \sigma(w_1 x_i + b_1) \cdot (w_1 x_i + b_1) x_i) \\ 2((\sigma(w_1 x_i + b_1) - y_i) \cdot \sigma(w_1 x_i + b_1) \cdot (w_1 x_i + b_1)) \end{bmatrix} \quad (1.13)$$

werden nach vollziehen des Gradientenabstiegs in jeder Iteration aktualisiert.

Es ist unschwer zu erkennen, dass mit steigender Netzkomplexität die Terme noch viel größer werden. Dies ist auf die Kettenregel $u'(v(x)) = u'(v(x)) \cdot (v'(x))$ zurückzuführen. Mit diesem Verfahren ist der Gradientenabstieg zwar möglich, jedoch sehr rechenintensiv. Aus diesem Grunde verwenden die meisten Frameworks den Backpropagation Algorithmus, welcher etwas effizienter ist.

1.2.4 Der Backpropagation Algorithmus

Der Backpropagation Algorithmus kann als einfache Matrixmultiplikation

$$\mathcal{L}(y_i, \sigma^l(W^l \sigma^{l-1}(W^{l-1} \dots \sigma^1(W^1 x + b^1) + b^{l-1} \dots))) \quad (1.14)$$

verstanden werden. Sofern die Fehlerfunktion \mathcal{L} einen Skalar als Resultat liefert und die einzelnen Schichten l des Netzwerks jeweils nur mit ihrer nachfolgenden Schicht $l + 1$ verbunden sind.

Daraus resultiert, dass die partiellen Ableitungen $\partial \mathcal{L} / \partial w_{jk}^l$ und $\partial \mathcal{L} / \partial b_j^l$ berechnet werden müssen. Das Interesse dieser partiellen Ableitung liegt eigentlich auf der Fehleränderung δ in jedem Neuron. δ_j^l ist somit die Änderung des Fehlers im Neuron j in der Schicht l . Somit ist Backpropagation ein Algorithmus um δ_j^l zu berechnen. In der letzten Schicht

$$\delta_j^L = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L) \quad (1.15)$$

kann die Änderung des Fehlers gilt (Index $l = \mathbf{last}$). Analog dazu in Matrixschreibweise

$$\delta^L = \nabla_a \mathcal{L} \odot \sigma'(z^L) \quad (1.16)$$

mit \odot als komponentenweise Multiplikation der Matrix (Hadamard Produkt). Es kann für die Fehlerfunktion \mathcal{L} eine beliebige Funktion verwendet werden. Oft wird der L_1 oder L_2 Fehler verwendet. Für dieses Beispiel verwenden wir die quadratische Fehlerfunktion (L_2) auch bekannt als Mean-Square-Error.

$$\mathcal{L} = \frac{1}{2} \sum_j (\hat{y} - y)^2 \quad (1.17)$$

Somit resultiert nach der partiellen Ableitung von \mathcal{L} und einsetzen von a^L die Gleichung:

$$\delta^L = (\hat{y} - a^L) \odot \sigma'(z^L) \quad (1.18)$$

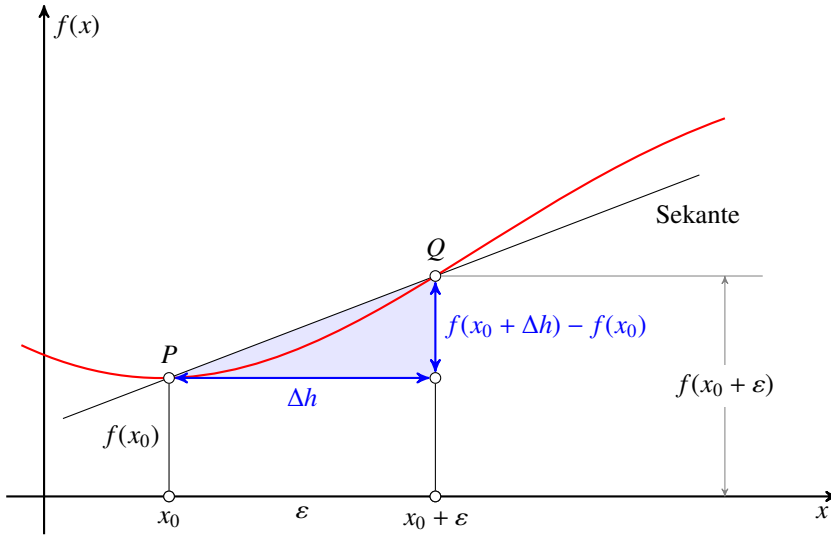
Mit diesen Annahmen wurde nun die Änderung des Fehlers abhängig von den Parametern der letzten Schicht berechnet. Der nächste Schritt wird sein herauszufinden, wie dieser durch das Netzwerk rückwärts gerechnet werden kann — back propagiert.

$$\delta^l = ((w^{l+1} \delta^l + 1) \cdot \sigma'(z^l)) \quad (1.19)$$

Die Transponierte der Matrix w^{l+1} kann man sich intuitiv vorstellen, als das 'Rückwärtsrechnen' des gewichteten Fehlers, da $w^{l+1} \cdot (w^{l+1})^T = I$. Dies ermöglicht die Änderung des Fehlers pro Neuron in der vorherliegenden Schicht zu bestimmen. Abschließend müssen noch die Gradienten

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial b} &= \delta \\ \frac{\partial \mathcal{L}}{\partial w} &= a_k^{l-1} \delta_j^l = a_{in} \delta_{out} \end{aligned} \quad (1.20)$$

berechnet werden.



1.3 Lösung

Wie bereits erklärt, basiert die Lösung darauf, dass ein geeignetes numerisches Verfahren anstelle des Backpropagation-Algorithmus verwendet wird um den Gradienten zu berechnen. Der Gradientenabstieg kann somit weiterhin vom Framework übernommen werden, obschon die Gradientenberechnung ersetzt wurde. Dies ermöglicht zu einem späteren Zeitpunkt einen Vergleich beider Verfahren.

1.3.1 Finite Differenzen Methode

Die Finite Differenzen Methode ist motiviert durch die klassische Ableitung. Die Ableitung (Tangente) wird als Sekante zweier Punkte der Funktion berechnet, wobei beide Punkte einen Abstand von $\Delta h \rightarrow 0$ annehmen. Dieses Konzept ist als Differentialquotient

$$f'(x_0) = \lim_{\Delta h \rightarrow 0} \frac{f(x_0 + \Delta h) - f(x_0)}{\Delta h} \quad (1.21)$$

bekannt. In der Numerik existiert die Möglichkeit von $\Delta h \rightarrow 0$ nicht, da die Datenpunkte diskretisiert sind. Stattdessen ist der kleinstmögliche Abstand zwei benachbarte Datenpunkte. Entsprechend kann man die numerische Ableitung (Differenzenquotient) als eine der Differentialquotienten

$$\begin{aligned} f'(x_0) &= \frac{f(x_0 + h) - f(x_0)}{h} \\ f'(x_0) &= \frac{f(x_0) - f(x_0 - h)}{h} \\ f'(x_0) &= \frac{f(x_0 + h) - f(x_0 - h)}{2h} \end{aligned} \quad (1.22)$$

schreiben, wobei h hier den Abstand zweier benachbarter Datenpunkte annimmt. Die Genauigkeit ist limitiert durch den Abstand der zwei Stützstellen.

1.3.2 Taylor Methode

Mithilfe der Taylor Reihe lässt sich jede beliebige analytische Funktion $f(x)$ approximieren. Dies machen wir uns für die Herleitung einer etwas stabileren numerischen Ableitung zu Nutze.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots \quad (1.23)$$

Um eine höhere Genauigkeit zu erreichen, kann die Anzahl der Stützstellen erhöht werden. Dies lässt sich mithilfe der Taylorreihe herleiten. Im Grunde wird ein Polynom höherer Ordnung gesucht, welches Stützstellen in einem uniformen Abstand hat. Die Ableitung dieses Polynoms sollte dann eine höhere Genauigkeit aufweisen, da durch die höhere Ordnung die Ursprungsfunktion besser um den Punkt der Ableitung approximiert werden kann. Als Beispiel soll die erste Ableitung mit 5 Stützstellen in einem uniformen Abstand h berechnet werden:

$$\frac{\delta f}{\delta x} \approx Af(x-2h) + Bf(x-h) + Cf(x) + Df(x+h) + Ef(x+2h) \quad (1.24)$$

Die jeweiligen Stützstellen können nun mittels Taylor-Reihe erweitert werden:

$$\begin{aligned} Af(x-2h) &\approx Af(x) + Af'(x)(-2h) + A\frac{1}{2}f''(x)(-2h)^2 + A\frac{1}{6} + f'''(x)(-2h)^3 + A\frac{1}{24} + \\ &\quad f''''(x)(-2h)^4 + A\frac{1}{120}f'''''(x)(-2h)^5 \\ Bf(x-h) &\approx Bf(x) + Bf'(x)(-h) + B\frac{1}{2}f''(x)(-h)^2 + B\frac{1}{6} + f'''(x)(-h)^3 + B\frac{1}{24} + \\ &\quad f''''(x)(-h)^4 + B\frac{1}{120}f'''''(x)(-h)^5 \\ Cf(x) &= Cf(x) \\ Df(x+h) &\approx Df(x) + Df'(x)(+h) + D\frac{1}{2}f''(x)(+h)^2 + D\frac{1}{6} + f'''(x)(+h)^3 + D\frac{1}{24} + \\ &\quad f''''(x)(+h)^4 + D\frac{1}{120}f'''''(x)(+h)^5 \\ Ef(x+2h) &\approx Ef(x) + Df'(x)(+2h) + E\frac{1}{2}f''(x)(+2h)^2 + E\frac{1}{6} + f'''(x)(+2h)^3 + E\frac{1}{24} + \\ &\quad f''''(x)(+2h)^4 + E\frac{1}{120}f'''''(x)(+2h)^5 \end{aligned} \quad (1.25)$$

Bei einsetzen dieser Gleichungen in die erste Gleichung (siehe oben) ist ersichtlich, dass nur die Koeffizienten der 2. Ordnung bestehen bleiben müssen und folglich 1 ergeben müssen. Der Rest sollte verschwinden und 0 ergeben. Zur Unterstützung der Lesbarkeit können die Terme nach Ableitungsgrad sortiert werden:

$$\begin{aligned} f(x) \cdot (\quad A + \quad B + \quad C + \quad D + \quad E) &= 0 \\ \frac{1}{2}f'(x)(h) \cdot (\quad -2A - \quad B \quad + \quad D + \quad 2E) &= \frac{2}{h} \\ \frac{1}{6}f''(x)(h^2) \cdot (\quad 14A + \quad B \quad + \quad D + \quad 4E) &= 0 \\ \frac{1}{24}f'''(x)(h^3) \cdot (\quad -8A - \quad B \quad + \quad D + \quad 8E) &= 0 \\ \frac{1}{120}f''''(x)(h^4) \cdot (\quad 16A + \quad B \quad + \quad D + \quad 16E) &= 0 \end{aligned} \quad (1.26)$$

Dies lässt sich auch in Matrix Form wie folgt schreiben:

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \\ -8 & -1 & 0 & 1 & 8 \\ 16 & 1 & 0 & 1 & 16 \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \frac{1}{h} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (1.27)$$

Diese Matrix lässt sich nach

$$\begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \frac{1}{h} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -2 & -1 & 0 & 1 & 2 \\ 4 & 1 & 0 & 1 & 4 \\ -8 & -1 & 0 & 1 & 8 \\ 16 & 1 & 0 & 1 & 16 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 2 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \frac{1}{h} \begin{bmatrix} \frac{1}{12} \\ \frac{2}{3} \\ 0 \\ \frac{2}{3} \\ \frac{1}{12} \end{bmatrix} \quad (1.28)$$

invertieren. Dies ergibt die erste Ableitung

$$f'(x) \approx \frac{1f(x-2h) - 8f(x-h) + 8f(x-h) - 1f(x+2h)}{12h} \quad (1.29)$$

angenähert durch 5 Stützstellen. Analog kann dies natürlich für höhere Ableitungen oder non-uniforme Stützstellen gemacht werden.

Fehlerabschätzung

Zur Berechnung der ersten Ableitung mit 5 Stützstellen werden die ersten 5 Terme der Taylorreihe verwendet. Die Ungenauigkeit dieser Methode ist somit abhängig durch die Rest Terme

$$\frac{f(x-2h) - 8f(x-h) + 8f(x-h) - f(x+2h)}{12h} = f'(x) - \frac{1}{30}f^{(5)}(x)h^4 + R_n(x). \quad (1.30)$$

Der Restterm R_n ist abhängig von der Taylorreihe der zu ableitenden Funktion. Funktionen lassen sich somit mit mehr Stützstellen genauer approximieren, da der Rest R_n mit steigender Taylor-Ordnung kleiner wird. Wenn aber die Terme höherer Ordnung schneller anwachsen als die Fakultät

$$f^{(n)} > n! \quad (1.31)$$

so ist die Approximation nicht genau. Aus der Fehlerabschätzung ergeben sich grundsätzlich zwei Typen an Problemen, die interessant sind zu untersuchen. Die Terme des ersten Types werden mit zunehmender Ordnung kleiner, während bei dem zweiten Typ die Terme immer grösser werden. Gleichzeitig muss aber ein guter Kompromiss gefunden werden, welcher den Abstand h nicht zu klein werden lässt. Ein sehr kleines h führt zu mehr Rauschanfälligkeit, da bei Benachbartenpunkten ein Rauschen von $\pm \epsilon$ höher ins Gewicht fällt.

Als Beispiel für eine Funktion, bei welcher die Terme kleiner werden, können die trigonometrischen Funktionen verwendet werden. Im speziellen hat der Sinus

$$\sin = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} (x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + R_n \quad (1.32)$$

eine Taylorreihe welche immer alternierend ist. Der Fehler

$$R_n < \left| \frac{x^{11}}{11!} \right| \quad (1.33)$$

kann somit auf einem Intervall von $-1 < x < 1$ nicht grösser werden als $\approx 2.5 \cdot 10^{-8}$. Auf der anderen Seite erweist sich die geometrische Reihe


$$\frac{1}{1-x} = \sum_{n=0}^{\infty} 1 + x + x^2 + x^3 + x^4 + R_n \quad (1.34)$$

als sehr schwer zu approximieren. Der Fehler R_n läuft ab $x > 1$ nach ∞ . Für die Experimente werden aus diesem Grund verschiedene Abstände h verwendet und eine unterschiedliche Anzahl Stützstellen, um aufzuzeigen, wie sich das Resultat verändern kann.

1.4 Resultate

Die Resultate dieser Arbeit sollen hauptsächlich zeigen, ob die Finite Differenzen Methode als Option zur Optimierung neuronaler Netzwerke geeignet ist. Es ist klar, dass implementationsbedingt die Trainingsgeschwindigkeit nicht dem Backpropagation Algorithmus das Wasser reichen kann. Viel wichtiger ist eine qualitative Aussage treffen zu können, ob die Methode eine interessante Variante darstellt und ob das Gedankenexperiment funktioniert.

Um dies an einem einfachen Beispiel zu verdeutlichen, wird versucht eine logische XOR Verknüpfung zu trainieren. Ein XOR-Gate ist ein elektronisches Bauteil, welches im Prinzip eine 'Entweder-Oder' Logik abbildet. Im gewählten Fall hat das Bauteil 2 digitale Eingänge und sollte entweder der Erste oder der Zweite aktiv sein, so ist der Ausgang ebenfalls aktiv. Sollte keiner der beiden Ausgänge oder beide Ausgänge aktiv sein, so ist der Ausgang inaktiv. Dies kann man der Logiktable in Abbildung 1.3 entnehmen. *Anmerkung: Wie zum Teufel wird das GATE gefixed, Linien nicht parallel!*



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

Abbildung 1.3: Das XOR-Gate und die dazugehörige Logiktable

Das gewählte Netzwerk in Abbildung 1.4 besteht aus 3 Schichten. Die Mittelschicht (Hidden Layer) wird benötigt, da sich mit nur einer Schicht keine Funktion finden lässt welche die Punkte in der Ebene trennt. Dies wird in Abbildung 1.5 noch einmal verdeutlicht. Das Experiment wurde basierend auf dem PyTorch implementiert und trainiert. Das Netzwerk wurde jeweils mit der Finite Differenzen Methode und dem Backpropagation Algorithmus trainiert. Bei der FDM wurde eine verschiedene Anzahl an Stützstellen (Support) gewählt und unterschiedliche Abstände h verwendet. Die Resultate sind in den Abbildungen 1.6, 1.7, 1.8 ersichtlich, die y-Achse stellt den quadratischen Fehler zwischen \hat{y} und y dar, während die x-Achse die Entwicklung über die Anzahl Iterationen aufzeigt.

Es ist erstaunlich, dass bereits bei diesem kleinen Netzwerk numerische Effekte im Backpropagation Algorithmus in einem etwas höheren Fehler resultieren. Der Fehler von $\approx 10^{-14}$ im besten Fall (FDM) ist Faktor 10^3 kleiner als die des Backpropagation Algorithmus.

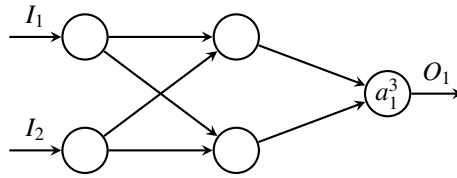
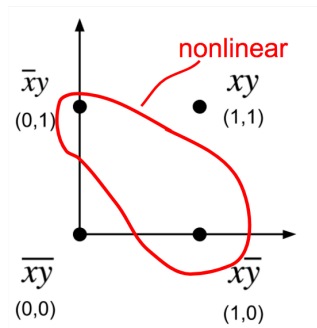
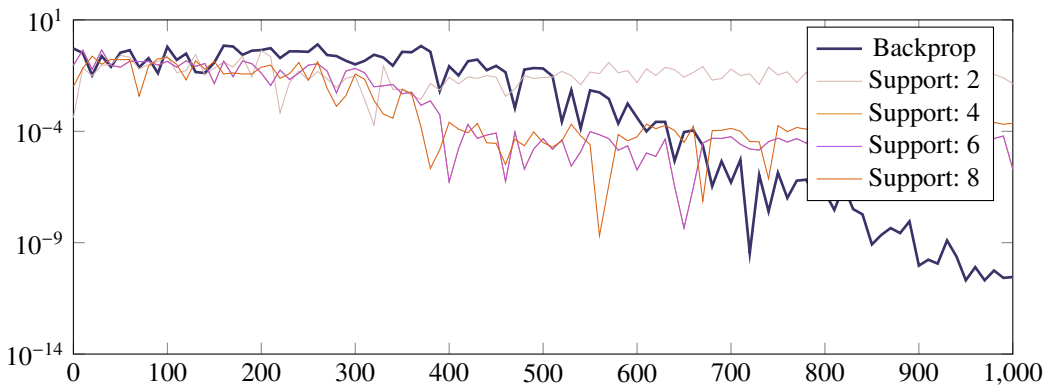
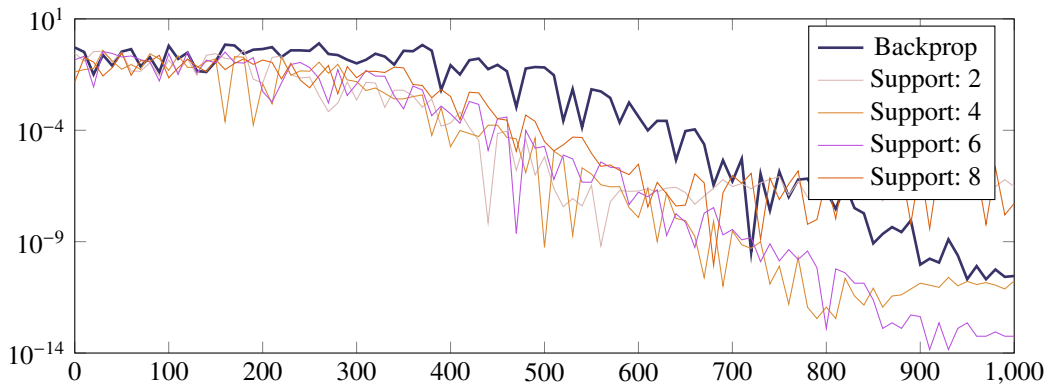
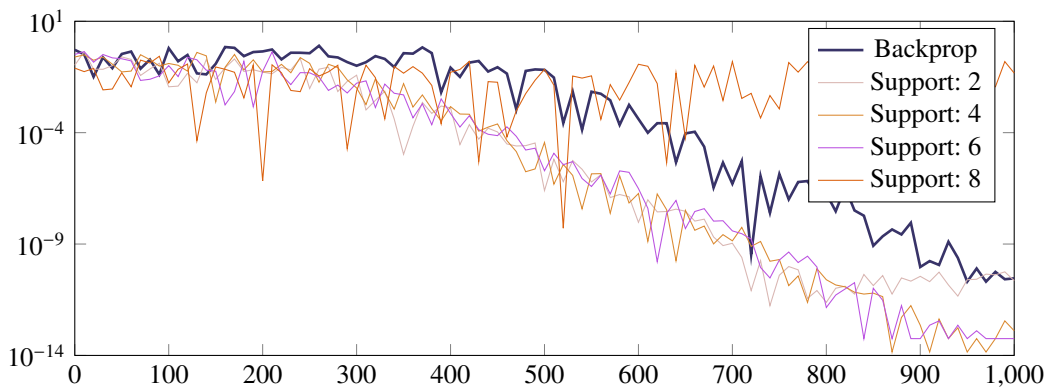


Abbildung 1.4: Das Neuronale Netzwerk zum Erlernen des XOR Logikoperators

Abbildung 1.5: Visualisierung der Punkte in einer xy -Ebene und deren Separierbarkeit.Abbildung 1.6: Resultate des Gradientenabstiegs mit Stützstellenabstand $h = 1$.

Abbildung 1.7: Resultate des Gradientenabstiegs mit Stützstellenabstand $h = 0.1$.Abbildung 1.8: Resultate des Gradientenabstiegs mit Stützstellenabstand $h = 0.01$.

1.5 Folgerungen