Einführung in die Neuroinformatik SoSe 2019

Institut für Neuroinformatik

PD Dr. F. Schwenker

5. Aufgabenblatt (Abgabe bis 11. Juni 2019 zur Vorlesung)

Aufgabe 1 (2 Punkte): Grundprinzip des Backpropagation-Verfahrens [Pen and Paper]

Wir haben bereits gesehen, wie ein einzelnes Neuron anhand von Beispielen mit Hilfe des Gradientenverfahrens lernen kann. Wenn wir unser Netzwerk um mehrere Neuronen in unterschiedlichen Schichten erweitern, lernen wir prinzipiell immer noch auf die gleiche Art und Weise: anhand von Beispielen werden die Gewichte des Netzwerkes iterativ angepasst, um eine definierte Fehlerfunktion zu minimieren. Wir haben jedoch für einen Input erst bei der Netzwerkausgabe das Lehrersignal als Vergleichswert zur Verfügung. Insbesondere ist kein direkter Abgleich für Neuronen aus der Zwischenschicht möglich. Wir berechnen daher den Fehler in der Ausgabeschicht (unter Verwendung des Lehrersignals) und leiten daraus die Fehler in den vorherigen Schichten ab. Das Lernen im Netzwerk läuft dann in vier Phasen ab:

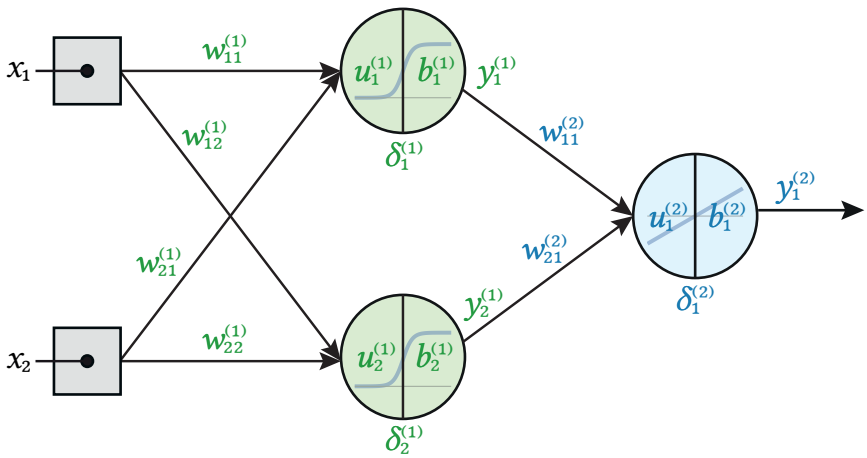
I. Zuerst wird das Eingabesignal vorwärts von der Eingabe- bis zur Ausgabeschicht propagiert und die Netzwerkausgabe berechnet.

II. Mit Hilfe der Netzwerkausgabe und des Lehrersignals können wir den Fehler in der Ausgabeschicht bestimmen.

III. Nun gehen wir noch einmal rückwärts durch das Netzwerk und propagieren den Fehler von der Ausgabe- zur Eingabeschicht (Backpropagation).

IV. Schließlich werden die Regeln zur Adaptierung der Gewichte angewendet.

Wie genau die Berechnung dabei von statten geht, was dafür benötigt wird und in welcher Reihenfolge wir sie durchführen müssen, wollen wir uns zunächst anhand eines einfachen Netzwerkes veranschaulichen:



Wir haben hier zwei Eingabe- und ein Ausgabeneuron sowie zwei Neuronen in der Mittelschicht. Gezeigt sind die Gewichte $w_{ij}^{(s)}$, die Schwellwerte $b_i^{(s)}$, die dendritischen Potenziale $u_i^{(s)}$, die Ausgabe der Neuronen $y_i^{(s)}$ sowie die Fehler $\delta_i^{(s)}$. Die Indizes i und j bezeichnen dabei Neuronen aus der s-ten Schicht. Zusätzlich definieren wir mit $f^{(1)}$ die Transferfunktion in der Zwischen- und mit $f^{(2)}$ die Funktion in der Ausgabeschicht. η bezeichnet die globale Lernrate.

Ein vollständiger Durchlauf des Backpropagation-Lernalgorithmus wäre selbst für dieses kleine Netzwerk etwas zu viel Rechenaufwand, um es per Hand durchzuführen. Daher wollen wir nur eine Rechnung auf der Index-Ebene erstellen und uns so die Abfolge der Rechenschritte verdeutlichen. Konkret wollen wir uns ansehen, was alles zum Update des Gewichtes $w_{11}^{(1)}$ notwendig ist, wenn wir den Datenpunkt (x_1, x_2) mit dem Lehrersignal T_1 in den Backpropagation-Lernalgorithmus geben. Zeige dabei mit Hilfe von Tabelle 1, wie die Variablen

$$u_1^{(1)}, u_1^{(2)}, y_1^{(1)}, y_1^{(2)}, \delta_1^{(1)}, \delta_1^{(2)}, \quad \text{sowie} \quad w_{11}^{(1)} \tag{1}$$

berechnet werden. Alle anderen Variablen seien als gegeben angenommen.

1. Lege zuerst die Reihenfolge der Berechnung fest. Ordne dazu jeder der in (1) genannten Variablen einem Berechnungsschritt zu (2. Spalte)?
2. Gib für jede Variable in (1) an, welche anderen Variablen zur Berechnung als Eingaben benötigt werden (3. Spalte).
3. Ordne jedem Berechnungsschritt (Zeile) einer der oben erwähnten Phasen (I–IV) zu (4. Spalte)?

Aufgabe 2 (8 Punkte): Backpropagation [Python]

Wir sind nun gewappnet, den Backpropagation-Lernalgorithmus im ganzen Umfang zu implementieren. Wir wollen dazu ein Netzwerk entwickeln, welches eine zwei-dimensionale sinusoidale Funktion approximiert. Hier können wir uns leicht in einem Gitter Trainingsdaten generieren und optisch überprüfen, wie gut die Approximation ist. Die Genauigkeit anhand einer separaten Testmenge wollen wir hier noch nicht überprüfen, wir werden aber den Netzwerkfehler

$$E = \sum_{\mu=1}^{M} \left\| T_\mu - y_\mu \right\|_2^2 \tag{2}$$

über alle M Trainingssamples im Auge behalten. Verwende für das Netzwerk die folgenden Hyper-Parameter[1]:

[1] *Hyper-Parameter* ist ein Begriff, der alle Konfigurationsmöglichkeiten des Netzwerkes umfasst und zur Abgrenzung der lernenden Parameter (Gewichte) verwendet wird.

Tabelle 1: Übersicht über die Berechnung der in (1) genannten Variablen. In jedem Schritt wird eine Variable unter Zuhilfenahme anderer Eingangsvariablen berechnet. Die letzte Spalte ordnet jedem Berechnungsschritt einer der Phasen des Backpropagation-Algorithmus zu.

Schritt	Berechnung	Benötigte Variablen	Phase
1	$u_1^{(1)}$		
2			
3			
4			
5			
6			
7			

- Der Tangens hyperbolicus als Transferfunktion in der Zwischen- und lineare Neuronen in der Ausgabeschicht, d. h. $f^{(1)}(x) = \tanh(x)$ und $f^{(2)}(x) = x$. Für $\tanh(x)$ auch die entsprechende Ableitung definieren. $f^{(2)}(x)$ (und die dazugehörige Ableitung) muss jedoch nicht definiert oder an Funktionen übergeben werden. Hier einfach direkt die entsprechenden Werte verwenden.
- Eine Lernrate von $\eta = 0.01$.
- 100 Neuronen in der Zwischenschicht.
- Wir wollen das Netzwerk über 1000 Epochen im Online-Modus trainieren (in jeder Epoche werden einmal alle Trainingssamples präsentiert).

Zu Beginn noch ein allgemeiner Hinweis: für diese Programmieraufgabe ist es essentiell, in Python effizient zu programmieren. Insbesondere sollten `for`-Schleifen vermieden und stattdessen lieber Matrizen- und Vektoroperationen verwendet werden. Es ist möglich, mit nur zwei Schleifen auszukommen (diese befinden sich dabei auch beide in der Funktion `training`). Selbst dann kann die Berechnung jedoch, abhängig von der Rechenleistung, noch einige Sekunden (oder vielleicht sogar Minuten) dauern.

Damit steht uns nichts mehr im Weg und wir können mit der Implementierung beginnen. Im Wesentlichen orientieren sich die folgenden Punkte dabei an die Beschreibung im Skript auf Seite 101ff. Benutze die Datei `Backpropagation.ipynb` als Vorlage und beachte auch Tabelle 2 für das Testskript.

1. (*Schritt 1: Initialisierung*) Wir wollen mit der Gewichtsinitialisierung beginnen. Die Anfangsgewichte definieren den Startpunkt in unserer Fehlerlandschaft. Von dort aus machen wir uns auf den Weg zum Minimum. Eine kluge Wahl der initialen Gewichte ist ausschlaggebend für den Lernerfolg des Netzwerkes und durchaus ein eigenes Thema für sich (darauf kommen wir in einem späteren Übungsblatt nochmal zurück). Hier wollen wir für die Gewichte (Pseudo-)Zufallszahlen aus dem Intervall $[-0.5; 0.5]$ wählen. Nimm dazu die korrekte Initialisierung der Gewichtsmatrizen sowie der Schwellwertvektoren in der Funktion `initialize_weights` vor.

2. (*Schritt 2: Vorwärtsphase*) Wir benötigen die aktuelle Netzwerkausgabe, um daraus den Fehler abzuleiten, den unser Netzwerk noch macht. Implementiere dazu die Funktion `forward`. Es soll dabei möglich sein, die Netzwerkausgabe für ein oder mehrere Eingabedaten gleichzeitig zu ermitteln. Da wir im Online-Modus trainieren, führen wir die Vorwärtsphase während des Trainingsprozesses immer nur für einen einzelnen Datenpunkt aus. Später in der Fehlerberechnung (siehe Schritt 7) benötigen wir die Netzwerkausgabe jedoch für alle Datenpunkte auf einmal.

3. (*Schritt 3 & 4: Fehlerberechnung*) Anhand der Netzwerkausgabe und des Lehrersignals können wir den Fehler in der Ausgabeschicht berechnen und daraus die Fehler in der vorherigen Schicht ableiten. Implementiere dazu die Funktion `prop_error`.

4. (*Schritt 5 & 6: Lernen*) Implementiere unter Verwendung der bisherigen Funktionen den Trainingsalgorithmus in der Funktion `training`. In jeder Epoche sollen die Muster dabei zufällig präsentiert werden, d. h., pro Epoche wählen wir eine andere Reihenfolge der Trainingssamples. Hierfür ist die Funktion `np.random.permutation` hilfreich.

5. (*Schritt 7: Fehlerbrechnung*) Berechne den Netzwerkfehler E gemäß Gleichung 2 nach jeder Trainingsepoche und speichere das Ergebnis. Beachte, dass zur Fehlerbrechnung erneut die Netzwerkausgabe ermittelt werden muss, da sich die Parameter geändert haben.

6. Um das Netzwerk zu „testen“, nehmen wir einfach erneut die Trainingsdaten, d. h. die Gitterpositionen der sinusoidalen-Funktion und ermitteln dafür jeweils die Netzwerkausgabe. Im Idealfall erhalten wir dann eine Approximation der originalen sinusoidalen-Funktion. Zeige beide Ergebnisse, einen Plot mit dem Absolutfehler zwischen den Funktionen sowie den Verlauf des Netzwerkfehlers an. Ein mögliches Ergebnis ist in Abbildung 1 zu sehen.

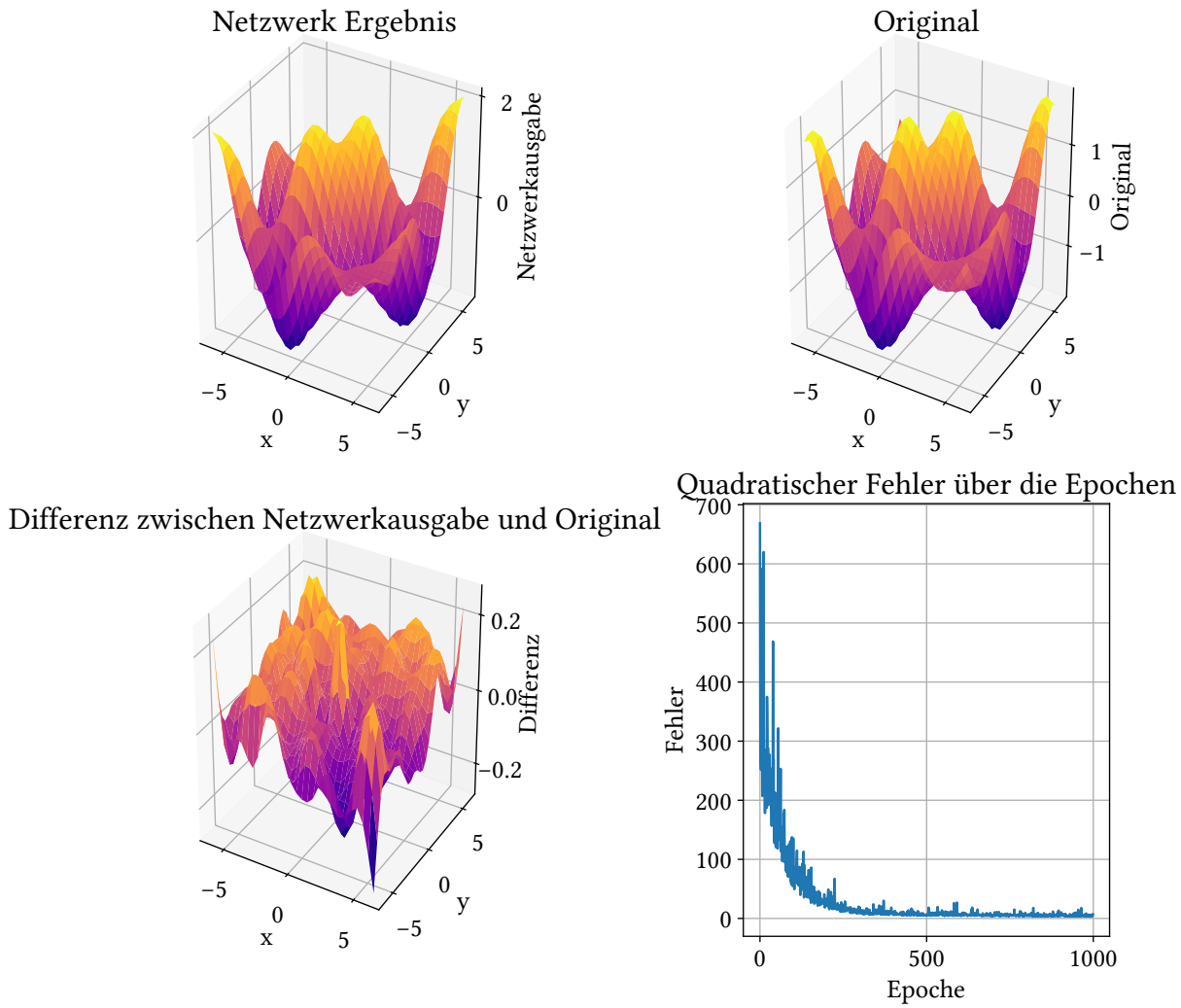


Abbildung 1: Ergebnis des Backpropagation-Lernalgorithmus. Die Netzwerkausgabe ist oben links und die originale sinusoidale-Funktion oben rechts dargestellt. Die absolute Differenz beider Funktion is in der Grafik unten links gezeigt. Unten rechts ist der Fehler E (Gleichung 2) des Netzwerkes im Trainingsverlauf abgebildet.

Tabelle 2: Variablennamen für das Python-Skript `Backpropagation.ipynb` aus Aufgabe 2. Die Variablen müssen exakt so benannt werden, damit das Testskript deren Inhalt überprüfen kann. Neben den Variablen werden auch die einzelnen Funktionen anhand von Testeingaben überprüft.

Variablenname	Typ	Beschreibung
`w1`	Matrix	Gewichtsmatrix zwischen Input- und versteckter Schicht.
`w2`	Matrix	Gewichtsmatrix zwischen versteckter und Ausgabeschicht.
`b1`	Vektor	Bias Vektor der versteckten Schicht
`b2`	Vektor	Bias Vektor der Ausgabeschicht.
`hiddenNeuronen`	Skalar	Anzahl der Neuronen in der Zwischenschicht.
`lernRate`	Skalar	Lernrate η.
`epochen`	Skalar	Anzahl der Trainingsepochen.
`error`	Vektor	Fehler des Netzwerkes E (Gleichung 2) über alle Trainingsepochen hinweg.