Vorlesung
Adaptive Modellierung und Simulation

Prof. Dr. R. Brause
R.Brause@cs.uni-frankfurt.de

Übungsblatt 2 Ausgabe: 5.5. Abgabe: 12.5.

In diesem Aufgabenblatt wollen wir uns ansehen, wie sich Neuronale Netze dazu einsetzen lassen, ein uns unbekanntes System zu modellieren.

Aufgabe 2.1 Modellierung mit neuronalen Netzen und Backprop (10 Pkte)

Als erstes betrachten wir ein kleines "`Spielzeugproblem"', an dem wir den Umgang mit Neuronalen Netzen zur Modellierung üben und das Verständnis der Funktionsweise etwas vertiefen wollen. Dazu wollen wir die Aufgabe lösen, den Verlauf einer unbekannten Funktion nur mit Hilfe von Daten zu erschließen. Die Datei `aprx.dt` enthält dazu zwei Spalten mit den Koordinaten von 16 Datenpunkten einer unbekannten reellen Funktion f(x) im Intervall x aus [-2,2]. Die erste Spalte besteht aus den Argumenten und die zweite Spalte aus den Funktionswerten. Da wir die Funktion nicht kennen, jedoch Zugriff auf eine Reihe von Eingabe-Ausgabe-Paaren besitzen, stellt die unbekannte Funktion eine Black-Box dar, an deren Modellierung wir interessiert sind.

Wir werden uns im Folgenden zwei kleine Szenarien ansehen, die einige generelle Eigenschaften Neuronaler Netze bei der Modellierung aufzeigen.

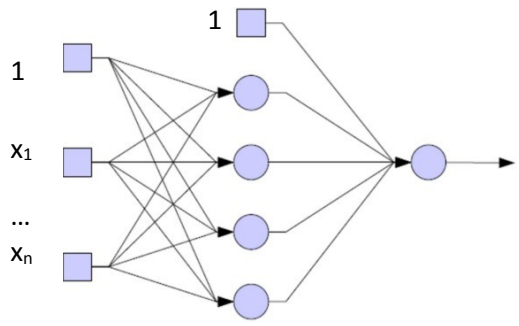
a) Lesen Sie die 16 Datenpunkte der Datei `aprx.dt` ein. Trainieren Sie dann ein Netz aus zwei Schichten mit der stochastischen Variante des Backpropagation-Algorithmus darauf, den unbekannten funktionalen Zusammenhang zu erlernen. Anschließend können Sie mit Hilfe des Netzes die Funktionswerte in den Zwischenräumen der einzelnen Datenpunkte interpolieren.

b) Um Ihre Ergebnisse überprüfen zu können, enthält die Datei `aprx_org.dt` weitere knapp 1000 Datenpunkte, aus denen sich der tatsächliche Kurvenverlauf der unbekannten Funktion leicht ablesen lässt. Lesen Sie die Datenpunkte dieser Datei ein und stellen Sie den Kurvenverlauf der von Ihrem Netz gelernten Funktion, sowie der tatsächlichen Funktion zusammen mit den 16 Datenpunkten der Trainingsmenge in einem Plot graphisch dar.

Was stellen Sie fest? Welche Eigenschaft können Sie an Ihrem trainierten Netz beobachten, unter dem Aspekt, dass Sie zum Training lediglich 16 verschiedene Datenpunkte verwendet haben? (6 Pkte)

Tipps:

- Verwenden Sie 1 Hidden-Schicht bestehend aus 64 Neuronen und je 1 feste Eingabe in der Eingabe und in der Hidden-Schicht.



- Die nichtlineare Ausgabefunktion der Neuronen in der Hidden-Schicht soll durch die Funktion $S_T(z) = \tanh(z) = 1 - 2S_F(z)$ gegeben sein, wobei S_F die in der Vorlesung angegebene Fermi-Funktion ist. In der Ausgabeschicht können Sie die lineare Funktion S(z) = z verwenden. Die Startgewichte sollen zufällig aus dem Intervall [-1/2,+1/2] gewählt werden. Als Lernrate empfiehlt sich $\gamma = 0.01$ und ca. 5000 Trainingsepochen (1 Epoche = 1 Präsentation der gesamten Trainingsmenge)

- Initialisieren Sie Ihre Gewichte so, dass am Anfang jede Schicht lernen kann und nicht im Sättigungsbereich liegt.
  Konkret: Haben Sie Muster, bei denen eine Komponente eins ist, so sollten die Gewichte so gewählt werden (z.B. zufällig und sehr klein), dass die damit erzeugte Aktivität einer Fermi-Ausgabefunktion gerade SF(0) = 0,5 ist. Bei 7 Eingaben null und einer eins ist so der Bias (Mittelwert der Eingabe) auf -0,5) und damit sollte man alle anderen Gewichte zu 0,5 wählen. Haben Sie die Aktivität der Hidden-Schicht im Mittel auf 0,5 gesetzt, so sollten die Anfangsgewichte der zweiten Schicht so gewählt werden, dass die Ausgabe der zweiten Schicht ebenfalls in der Mitte der möglichen Ausgabewerte liegt. Bei drei Ausgaben des Hidden layer sind also alle drei Eingaben 0,5, so dass ein Bias von -0,25 und Gewichte mit 0,5 gerade eine Ausgabe von 3*0,5*0,5-0,25 = 0,5 erzeugt.

- Wählen Sie die Muster aus der Trainingsmenge per Zufall aus, so dass bei jedem Trainingslauf die Reihenfolge unterschiedlich ist. Da der Lerneffekt i.A. von der Musterreihenfolge abhängt, wird so eine möglichst ausgewogene Stochastik beim Lernen bewirkt und das Lernen verallgemeinert.

- Haben Sie nur wenige Muster, so ist die Trainingsmenge bald abgearbeitet. Verwenden Sie in diesem Fall die Trainingsmenge öfters (aber in zufälliger Reihenfolge). Ein Durchgang durch die Trainingsmenge wird als Epoche bezeichnet.

- Für die laufende Überwachung des Lernens ist es hilfreich, die Werte für die Aktivitätskorrektur (das Delta) sowie für die Gewichtskorrektur (den aktuellen Wert des Gradienten) auszudrucken bzw. als Graph zu plotten.

- Es ist wichtig, dass Sie Vertrauen zu Ihrem Programm haben, bevor Sie damit Probleme lösen. Setzen Sie also nach der Programmierung zuallererst einen Test auf. Ein guter, einfacher Test besteht darin, als Trainingsmenge einen Sinus zu erzeugen und diese Funktion lernen zu lassen. Das Eingabemuster kann dabei aus mehreren, zeitlich aufeinander folgenden Werten bestehen, deren Folgewert geschätzt werden soll, oder aus verrauschten Sinuswerten innerhalb eines Intervalls [-pi/2,+pi/2], dessen Originalfunktion gelernt werden soll, oder irgendetwas anderes, was Sie leicht nachprüfen können.

- **NumPy** und **SciPy** – erleichtert in Python die Arbeit mit mathematischen Strukturen wie Vektoren, Matrizen etc. und Operationen (http://www.numpy.org/und http://scipy.org/).

- Plots können Sie mit Hilfe einer Bibliothek Ihrer gewählten Programmiersprache erzeugen, etwa die **MatPlotLib** – Bibliothek in Python für die graphische Darstellung von Funktionen, Punktmengen, Geraden und ähnlichem… (http://matplotlib.org/). Tutorial unter http://matplotlib.org/users/pyplot_tutorial.html

c) So richtig interessant wird das Interpolationsproblem allerdings erst, wenn wir es mit verrauschten Daten zu tun haben und wir damit die dahinterstehende Funktion approximieren sollen. Die Datei `aprx_noise.dt` enthält 128 Punkte der unbekannten Funktion, allerdings mit sehr stark verrauschten Funktionswerten. Lesen Sie die Datenpunkte aus der Datei `aprx_noise.dt` ein und wiederholen Sie damit das obige Experiment. Nach 1000 Trainingsepochen können Sie das Training stoppen. Was beobachten Sie und über welche zweite Eigenschaft scheint Ihr Netz zu

verfügen? Verwenden Sie zum Training eine feste Reihenfolge der Trainingsdaten und fertigen Sie nach dem Training wieder einen Plot vom Approximationsergebnis an, in dem die verrauschten Datenpunkte, die Originalfunktion und die vom Netz berechnete Funktion dargestellt sind. (4 Pkte)

Aufgabe 2.2 Overfitting (6 Punkte)

Ein typisches Problem beim Training zu der Approximation verrauschter Daten oder einer geringen Anzahl verfügbarer Trainingsdaten ist das sogenannte *Overfitting*, eine Überanpassung an die zu lernenden Daten. Wollen wir Neuronale Netze zur Modellierung nichtlinearer Zusammenhänge verwenden, so benötigen wir eine Methode, diesen Effekt zu vermindern. Wie sich *Overfitting* genau äußert und wie es sich reduzieren lässt, wollen wir uns in dieser Aufgabe ansehen.

a) Um diesen wichtigen Effekt zu untersuchen, wiederholen Sie das Experiment aus Aufgabe 2.1 c). Initialisieren Sie die Gewichte im Backpropagation-Algorithmus mit zufälligen Werten aus [-4; +4]. Führen Sie dann 5000 Trainingsepochen durch. Stellen Sie den Kurvenverlauf der von Ihrem Netz gelernten Funktion, den tatsächlichen Kurvenverlauf und die verrauschten Datenpunkte der Trainingsmenge dann wieder zusammen in einem Plotfenster dar. Die Ausgabe des Netzes sollte nun sehr starken Schwankungen unterworfen sein. Dies ist der *Overfitting*-Effekt. Können Sie sich vorstellen, warum dieser Effekt in der Praxis so dramatisch ist? (2 Pkte)

b) Eine Möglichkeit zur Reduzierung des *Overfitting*-Effektes ist das *stopped training* mit Hilfe einer zusätzlichen Validierungsmenge. Allerdings wird sie nicht zum Training selbst herangezogen. Sie dient während der Trainingsphase der Messung der Leistung eines Netzes bei Eingabe von unbekannten Daten und damit als Maß für die Generalisierungsfähigkeit. Die Datei `aprx_noised_validationset.dt` enthält weitere 128 verrauschte Datenpunkte der unbekannten Funktion aus Aufgabe 2.1. Die Datenpunkte hierin sind zufällig gewählt und auf die gleiche Weise verrauscht, wie die Daten der Trainingsmenge. Wiederholen Sie Aufgabe 2.2 a). Trainieren Sie wieder Ihr Netz mit Hilfe der Trainingsmenge darauf, die verrauschte Funktion zu lernen, aber berechnen Sie während des Trainings nach jedem Trainingsschritt den durchschnittlichen quadratischen Fehler, den Ihr Netz bei Eingabe der Werte aus der Validierungsmenge macht. Nach Abschluss des Trainings können Sie dann die Neuronengewichte verwenden, die den kleinsten Validierungsfehler während des Trainings hatten. Stellen Sie dann wie zuvor den entsprechenden Plot, auf dem das Approximationsergebnis zu erkennen ist und stellen Sie auch die Datenpunkte der Validierungsmenge dar. Fertigen Sie auch einen Plot an, in dem der Kurvenverlauf des durchschnittlichen quadratischen Fehlers Ihres Netzes bezüglich der Trainingsmenge *und* der Validierungsmenge während der 2000 Iterationen zu erkennen ist.
Was stellen Sie fest, wenn Sie sich die Approximation Ihres Netzes mit bzw. ohne Verwendung der Validierungsmenge ansehen? Welches Verhalten weisen die Kurven für den Fehler bezüglich der Trainingsmenge und der Validierungsmenge auf und was schließen Sie daraus? (4 Pkte)