Blatt 06 - Funktionsapproximation mit MLP

Abgabe: Bis Mittwoch, 10.Dezember 2014, 10:00 Uhr
An: Tobias Rothenberger, <rothenb@informatik.uni-frankfurt.de>

Aufgaben:	Aufgabe 6.1			Aufgabe 6.2			Aufgabe 6.3			Aufgabe 6.4		
	1a	1b	1c	2a	2b	2c	3a	3b	3c	4a	4b	4c

Aufgabe 6.1 - Aufbau interner Repräsentationen

Bevor wir in der nächsten Aufgabe ein Multilayer-Perzeptron (MLP) für ein komplizierteres Problem einsetzen, wollen wir hier zunächst ein recht simples Problem lösen und uns dabei einige Eigenschaften der Funktionsweise von MLPs anschauen.

Eine interessante Eigenschaft von MLPs ist, daß die Neuronen der *hidden layer* meist als Merkmalsextraktoren funktionieren, die sich darauf spezialisieren, bestimmte Eigenschaften der Trainingsdaten zu erfassen. Darüberhinaus neigen Sie dazu, selbständig nützliche interne Repräsentationen der Eingabe zu erzeugen. Dabei werden in der Regel recht kompakte Repräsentationen aufgebaut, die in einem wichtigen Zusammenhang mit der zu lernenden Abbildung stehen, die dem Netz antrainiert werden soll.

Wir betrachten dazu das 8:3:8 Encoder/Decoder-Problem. Trainieren Sie ein MLP mit einem *hidden layer,* bestehend aus 3 Neuronen, mit Hilfe des Backpropagation-Algorithmus darauf, die 8 Eingabevektoren

$$\begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\\0\\0\\0\\1\end{bmatrix}$$

wieder auf sich selbst abzubilden. Das Netz soll also die Identische Abbildung auf dieser Menge lernen und jeden Eingabevektor wieder genauso ausgeben.

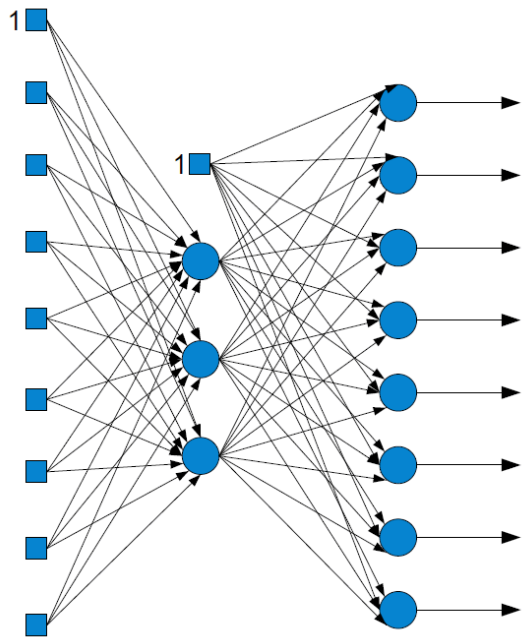


Abbildung Das 8:3:8-Encoder-Netzwerk.

Das Netz befindet sich auf diese Weise in dem Dilemma, daß es alle 8 Vektoren beim Durchlaufen des Netzes irgendwie durch die Ausgabe von nur 3 Neuronen des Hiddenlayers wieder vollständig rekonstruieren muß. Diese stellen gewissermaßen einen Flaschenhals für die durch das Netz transportierte Information dar.

a. Implementieren Sie den Backpropagation-Algorithmus und testen Sie Aktivität und Lernen mit Hilfe der in der Einleitung beschriebenen Methoden. (Testmuster, Fehler- und Gradientenanzeige) auf seine Korrektheit.

   Verwenden Sie dabei in der Eingabe- und der Hidden-Schicht jeweils ein Bias-Neuron sowie die Fermifunktion $S(z) = \frac{1}{1+\exp(-z)}$ als nichtlineare Ausgabefunktion aller Neuronen. Trainieren Sie das Netz 100.000 Epochen lang im Online-Modus, aber mit einer konstanten Lernrate von 1. Für den BP-Algorithmus benötigen Sie noch die Ableitung der Ausgabefunktion. Für die Fermifunktion ist dies S'(z) = (1-S(z))S(z).

   Berechnen Sie anschließend die Ausgabe des Netzes bei Eingabe der 8

Vektoren und weisen Sie nach, dass das Netz die geforderte Abbildung korrekt gelernt hat.

nach oben

b. Berechnen Sie nun die Ausgabe der Hidden-Neuronen bei Eingabe der einzelnen Trainingsvektoren. Dies ist die interne Repräsentation, die das Netz für seine Eingabe entwickelt hat.
Was stellen Sie fest? Wohin scheinen die Werte bei weiterem Training ihrer Meinung nach zu konvergieren? Welche Art der Repräsentation scheint das Netz im *hidden layer* zur Berechnung der Ausgabe automatisch zu entwickeln?

nach oben

c. Trainieren Sie nun das Netz sowohl einmal mit nur zwei als auch mit nur einem *hidden*-Neuron. Was für eine Kodierung vermuten Sie nun?

nach oben

Aufgabe 6.2 - Interpolation und Overfitting

Ein großer Anwendungsbereich von Neuronalen Netzen besteht in der Interpolation von Funktionen. In der Tat können viele praktische Probleme auf ein Interpolationsproblem zurückgeführt werden. In diesem Aufgabenteil wollen wir uns daher einmal etwas näher mit dieser Fähigkeit beschäftigen.

a. Im nachfolgenden Aufgabenteil benotigen wir den Backpropagation-Algorithmus zum Training eines Multi-Layer-Perzeptrons. Implementieren Sie diesen Algorithmus und testen Sie die Aktivitat und das Lernen mit den in der Einleitung beschriebenen Methoden (Testmuster, Fehler- und Gradientenanzeige) auf seine Korrektheit. Als Trainingsmuster generieren Sie sich eine einfache Funktion (z.B. sin(x)) und Testen Sie die Funktion des trainierten Netzes an wenigen

Eingabewerten x.

Lesen Sie die Datei `aprx.dt` ein. Sie besteht aus 2 Spalten und enthält 16 Datenpunkte einer unbekannten Funktion f(x) im Intervall x aus [-2,2]. Die erste Spalte enthält jeweils das Argument x und die zweite Spalte den zugehörigen Funktionswert f(x).

Ihre Aufgabe besteht nun darin, einem Multilayer-Perzeptron mit der Eingabe aus dem Intervall [-2,2] diese Funktion beizubringen und anschließend mit Hilfe des Netzes die Funktionswerte in den Zwischenräumen der einzelnen Datenpunkte zu interpolieren.

Haben Sie dies getan, können Sie zum Vergleich einige weitere Datenpunkte der Originalfunktion aus der Datei `aprx_org.dt` einlesen. Sie enthält etwa 1000 Datenpunkte, so daß sich der Kurvenverlauf sehr gut erkennen läßt. Erstellen Sie dann einen Plot, in dem die 16 Datenpunkte, die Originalfunktion und die Approximation des Netzes zu erkennen sind.

nach oben

b. So richtig interessant wird das Interpolationsproblem allerdings erst, wenn man lediglich über verrauschte Daten verfügt und damit die dahinterstehende Funktion approximieren soll.

   Lesen Sie dazu die Datenpunkte der Datei `aprx_noised.dt` ein und verwenden Sie wieder das Multilayer-Perzeptron, das Sie mit Hilfe des Backpropagation-Algorithmus im *Online*-Modus darauf trainieren, dieses Problem zu lösen. Es handelt sich dabei um eine verrauschte Version der Funktion f(x) aus dem vorigen Aufgabenteil.

   Verwenden Sie zufällige Anfangsgewichte aus dem Bereich [+1/2,-1/2], eine feste Lernrate von 0,01, einen einzigen *hidden layer* mit 64 Neuronen und einer Bias-Eingabe, sowie etwa 1000 Lernschritte. Die Neuronen der Ausgabeschicht sollen dabei lineare Ausgabefunktionen besitzen und die Neuronen der Hiddenschicht die `tanh`-Funktion (mit a=1) als Ausgabefunktion.

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.

nach oben

c. Ein Problem bei der Approximation verrauschter Daten oder einer geringen Anzahl verfügbarer Trainingsdaten stellt das sogenannte *Overfitting* dar. Dabei handelt es sich um eine Überanpassung des Netzes an die Trainingsdaten, was einen Verlust der Generalisierungsfähigkeiten (korrekte Ausgaben auch bei unbekannten Eingaben) des Netzes zur Folge hat. Da *Overfitting* bei der Lösung sehr vieler praktischer Probleme auftreten kann, wollen wir uns nun eine nützliche Methode ansehen, diesen Effekt etwas zu begrenzen.

   Lesen Sie wieder die Daten aus der Datei `aprx_noised.dt` ein und verwenden Sie sie als Trainingsmenge Ihres Netzes, so wie sie dies im vorigen Aufgabenteil gemacht haben. Verwenden Sie nun allerdings 64 Neuronen im *hidden layer*, und initialisieren Sie die Gewichte im Backpropagation-Algorithmus mit zufälligen Werten aus [-4; +4]. Führen Sie dann 5000 Trainingsepochen durch. Stellen Sie dann das Ergebnis wie im vorigen Aufgabenteil graphisch dar.
   Die Approximation Ihres Netzes sollte jetzt sehr starken Schwankungen unterlegen sein. Dies ist der *Overfitting*-Effekt.

   Eine Methode zur Minderung des Overfitting ist die Einbindung einer weiteren Menge während des Trainings, der sogenannten *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.
   Lesen Sie nun zusätzlich die Daten aus der Datei `aprx_noised_validationset.dt` als Validierunsgmenge ein. Die Datenpunkte hierin sind zufällig gewählt und auf die gleiche Weise verrauscht, wie die Daten der Trainingsmenge.

Trainieren Sie dann wie eben 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 verwenden Sie dann die Gewichtseinstellung mit dem kleinsten Fehler bezüglich der Validierungsmenge.

Erstellen 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 Traingsmenge und der Validierungsmenge auf und was schließen Sie daraus?

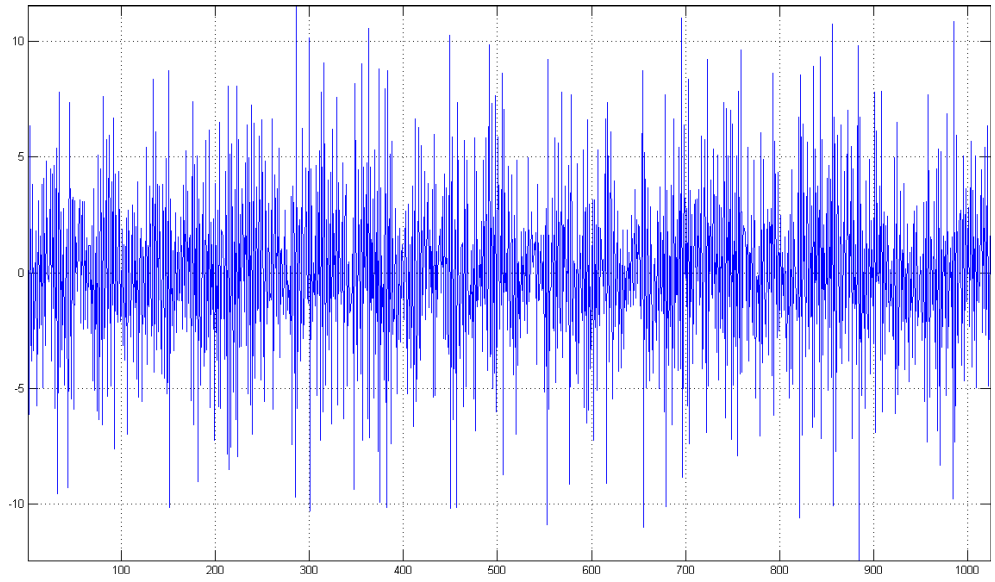
Bemerkung: Wir haben im Übrigen auch noch eine weitere Methode zur Vemeidung des Overfittings kennengelernt. Im vorigen Aufgabenteil haben wir gesehen, daß eine Begrenzung der Trainingsepochen diesen Effekt ebenfalls unterdrücken kann. Diese Vorgehensweise bezeichnet man auch als *early stopping*.

nach oben

Aufgabe 6.3 - Neuronale Netze zur Signalrekonstruktion

Die Multi-Layer-Perzeptrons lassen sich recht vielseitig einsetzen. Ein schönes Problem, bei dem wir die Leistungsfähigkeit dieser Netze sehen können, ist die Rekonstruktion verrauschter Signale. Die Dateien `noise01.dt`, `noise02.dt`, `noise03.dt` und `noise04.dt` enthalten stark rauschende Signale, in denen jeweils ein unbekanntes

Rechtecksignal versteckt ist. Ziel dieser Aufgabe ist es, ein Multilayer-Perzeptron (MLP) so zu trainieren, daß es lernt, diese Rechtecksignale trotz des Rauschens zu erkennen und wiederherzustellen.



Ein verrauschtes Signal.

Die Rechtecksignale bestehen dabei aus Blöcken von je 8 gleichen, aufeinander folgenden Werten aus {+1,-1}. Die einzelnen Blöcke sind jeweils durch 8 Nullen voneinander getrennt. Das gesamte Signal kann dabei beliebig verschoben sein, d.h. vor dem ersten Block stehen zwischen 0 und 8 Nullen.

Beispiel:

$$[0,0,0,+1,+1,+1,+1,+1,+1,+1,+1,0,0,0,0,0,0,0,0,-1,-1,-1,-1,-1,-1,-1,-1,...]$$

Die Rechtecksignale sind stark verrauscht und mit maximal 20 hochfrequenten Kosinussignalen der Form $a \cdot \cos(t \cdot f + \varphi)$ überlagert. Es ist bekannt, daß für die Frequenz f und die Amplitude a dieser Wellen $100 \leq f \leq 200$ und $|a| \leq 2$ gilt. Der Phasenwinkel φ ist unbekannt und jede der Kosinuswellen kann unterschiedliche Parameter besitzen. Dabei ist t ein Zeitparameter, wobei ein Datenpunkt im Signal genau einer Zeiteinheit entspricht.

a. Trainieren Sie nun ein MLP mit Hilfe des Backpropagation-Algorithmus und versuchen Sie, die Rechtecksignale möglichst gut wiederherzustellen. Dazu ist es erforderlich, daß Sie selbständig eine geeignete Trainingsmenge erstellen, mit

der Sie Ihr Netz trainieren können. Verwenden Sie neben der Trainingsmenge außerdem noch eine Validierungsmenge, um die Generalisierungsfähigkeiten ihres Netzes während des Trainings überprüfen zu können. Benutzen Sie zum Training lediglich eine lineare Ausgabefunktion $S(z) = z$ in der Ausgabeschicht Ihres Netzes.

Erstellen Sie auch einen Plot vom Trainingsverlauf (den durchschnittlichen Fehler, den Ihr Netz in den einzelnen Trainingsepochen bei Eingabe eines Musters gemacht hat sowie die Länge des Gradienten) für die Trainingsmenge und die Validierungsmenge. Sie können alle Kurven in den selben Graphen zeichnen. Anschließend stellen Sie jeweils das ursprüngliche (verrauschte) Eingabesignal und das von ihrem Netz rekonstruierte Signal graphisch dar.

nach oben

b. Da Ihr MLP-Netzwerk eine kontinuierliche Ausgabe besitzt, die Rechtecksignale aber einen diskreten Wertebereich von {-1,0,+1} aufweisen, sollte man zur Verbesserung der Netzleistung die Netzausgabe noch in eine diskrete Folge von Werten umwandeln.

   Überlegen Sie sich daher eine geeignete nichtlineare Ausgabefunktion für die letzte Schicht ihres Netzes, die die kontinuierliche Ausgabe in ein passendes Rechtecksignal umwandelt, so daß die Approximationsgüte Ihres Netzes verbessert wird. Stellen Sie jeweils das von ihrem Netz rekonstruierte Rechtecksignal, mit und ohne nichtlinearer Ausgabefunktion in der letzten Schicht, graphisch dar. Die beiden Kurven können sie in einem Plot zusammenfassen.

nach oben

c. Um zu demonstrieren, daß das Netz tatsächlich das Vorhandensein von Rechtecksignalen erkennt und nicht nur irgendwelche zufälligen Rechteckimpulse von sich gibt, für den Fall daß *doch keines* vorhanden ist, ist in einem der verrauschten Signale tatsächlich *kein* Rechtecksignal eingebettet. Welches ist es?

nach oben

Aufgabe 6.4 - Time Series Prediction und Ensemble Averaging

Neuronale Netze sind auch recht gut darin, zeitliche Abhängigkeiten zu lernen. In dieser Aufgabe werden wir uns ansehen, wie gut ein AdaLinE und ein Multilayer-Perzeptron den nächsten Wert einer Sequenz zeitlich aufeinanderfolgender Werte vorhersagen können. Die zeitlichen Abhängigkeiten werden dabei stark nichtlinear sein, was sich in einem großen Unterschied in der Prädiktionsqualität zwischen AdaLinE und MLPs ausdrücken wird.

a. Wir betrachten die Folge x_n, die gegeben ist durch $x_n = \sin(n+\sin(n^2))$, n=0,1,2,... . Nehmen Sie an, daß Sie die ersten 1000 Werte der Folge bereits kennen. Trainieren Sie nun ein AdaLinE mit der linearen Ausgabefunktion S(z) = z darauf, zu einer beliebigen Teilfolge bestehend aus 20 zeitlich unmittelbar aufeinanderfolgenden Werten $y_1,...,y_{20}$, die sich an einer *beliebigen unbekannten* Stelle innerhalb der Folge x_n befinden kann, den nächsten Wert y_{k+1} vorherzusagen.

   Verwenden Sie anschließend das trainierte Netz, um für 1000 <= n <= 2000 den jeweiligen Wert x_n der Folge vorherzusagen. Dabei können Sie annehmen, daß Ihnen die Werte $x_1,...,x_{n-1}$ bekannt sind.
   Berechnen Sie dann den durchschnittlichen betragsmäßigen Fehler, den Ihr Netz für diese Werte gemacht hat und erstellen Sie einen Plot, der für 1000 <= n <= 2000 die Abweichung der Prädiktion zum tatsächlichen Wert der Folge angibt. Erstellen sie auch eine Kurve, die die ersten 100 vorherzusagenden Werte zeigt und fügen Sie eine Kurve der von Ihnen vorhergesagten Werte hinzu.

nach oben

b. Verwenden Sie nun ein Multilayer-Perzeptron, bestehend aus einem Hiddenlayer und der Fermi-Funktion als Ausgabefunktion der Hidden- und der Ausgabeschicht, um wie in Aufgabenteil a) für 1000 <= n <= 2000 das nächste Folgenglied vorherzusagen. Berechnen Sie wieder den durchschnittlichen betragsmäßigen Fehler und fertigen Sie einen Plot des Prädiktionsfehlers, sowie der vorherzusagenden und der vorhergesagten Werte an. Vergleichen Sie das

Ergebnis mit dem aus Aufgabenteil a).

nach oben

c. Das Ergebnis des vorigen Aufgabenteils läßt sich noch etwas verbessern. Trainieren Sie nun nacheinander 8 Multilayer-Perzeptrons, so wie Sie dies in Aufgabenteil b) für ein einzelnes Netz getan haben. Verwenden Sie zum Training aber unterschiedliche Anfangsgewichte für jedes Netz. Nach dem Training geben Sie die 8 Ausgaben der Netze dann jeweils einem Neuron mit Gewichtsvektor **w** = [0,0125 0,0125 0,0125 0,0125 0,0125 0,0125 0,0125 0,0125] und linearer Ausgabefunktion als Eingabe. Die Ausgabe des so zusammengesetzten Netzes besteht dann in der Ausgabe dieses Neurons. Verfahren Sie dann wieder wie in den Aufgabenteilen a) und b). Was hat sich geändert?

nach oben