Einführung in die Neuroinformatik SoSe 2019

Institut für Neuroinformatik

PD Dr. F. Schwenker

11. Aufgabenblatt (Abgabe bis 23. Juli 2019 zur Vorlesung)

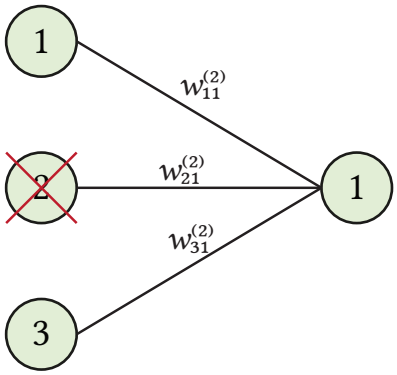
Aufgabe 1 (7 Punkte): *Dropout*

Neben der bereits kennen gelernten L_2-Regularisierung gibt es noch eine weitere verbreitete Technik, um *overfitting* zu vermeiden und die Generalisierungsfähigkeit des Netzwerkes zu steigern: *dropout*. Dabei wird während des Trainierens zufällig ein Anteil von p Neuronen von der Eingabe- oder den Zwischenschichten (nicht jedoch von der Ausgabeschicht) deaktiviert. Für das Testen werden wiederum alle Neuronen verwendet. In dieser Aufgabe wollen wir uns diese Technik etwas genauer ansehen.

1. [Pen and Paper] Ließ dir den zugehörigen Abschnitt[1] über *dropout* aus dem Buch *Neural Networks and Deep Learning* von Michael A. Nielsen durch und beantworte die folgenden Fragen:

   a) Erkläre in eigenen Worten, welche Intuition(en) es dafür gibt, dass *dropout* funktioniert.

   b) Nehmen wir an, wir arbeiten mit dem folgenden Teilnetzwerk, in dem drei Neuronen in der ersten Zwischenschicht mit einem Neuron in der zweiten Zwischenschicht verbunden sind:



   Wir wenden *dropout* auf die Neuronen in der ersten Zwischenschicht mit einer Wahrscheinlichkeit von $p = 1/3$ an (d. h., in jedem Batch-Durchlauf wird eines der drei Neuronen deaktiviert). Mit welchem Faktor müssen wir die Gewichte $w_{ij}^{(2)}$ nach dem Trainingsvorgang multiplizieren, um die erhöhte Anzahl aktiver Neuronen beim Testen zu kompensieren?

[1] http://neuralnetworksanddeeplearning.com/chap3.html#other_techniques_for_regularization

2. [Python] Analog zur Regularisierungsaufgabe wollen wir *dropout* auf ein Regressionsproblem anwenden. Wir werden dazu wieder ein Netzwerk mit und eines ohne *dropout* trainieren und dann die Ergebnisse vergleichen.

   a) Die Beispieldaten, die wir hier verwenden wollen, sind in der Datei `data.pickle` abgelegt. Deren Inhalt kann via

   ```
   import pickle
   data = pickle.load(open('data.pickle', 'rb'))
   ```

   in das Programm geladen werden. `data` ist eine $\mathbb{R}^{N\times 2}$ Matrix mit N zweidimensionalen Datenpunkten, wobei wir die erste Dimension als Eingabe und die zweite Dimension als Lehrersignal verwenden wollen.

   b) Da wir auch in dieser Aufgabe wieder mehrere Netzwerke erstellen, bietet es sich an, die Konfiguration und das Trainieren der Netzwerke in eine Funktion `train_network(dropout_rate)` auszulagern (mit der zu verwendenden *dropout rate* p).

      i. Setze die in Abbildung 1 gezeigte Netzwerkhierarchie um.

      ii. Trainiere mit dem *Adam optimizer* und einer Lernrate von $\eta = 0.02$ über 3000 Epochen.

   c) Trainiere einmal ein Netzwerk ohne ($p = 0$) und einmal eines mit *dropout* ($p = 0.3$).

   d) Berechne für beide Netzwerke die Ausgabe für diskrete Eingabedaten aus dem Bereich $x \in [-2; 2]$ mit einer Schrittweite von $\Delta x = 0.01$.

   e) Zeige die Datenpunkte sowie die Netzwerkausgabe mit und ohne *dropout* in einer Grafik an (siehe Abbildung 2).

3. Normalerweise ist *dropout* nur während des Trainingsvorganges aktiv und beim Testen wird das vollständige Netzwerk verwendet (wie in der vorherigen Teilaufgabe). Es gibt jedoch einen interessanten Ansatz [1], der *dropout* auch während des Testens aktiv lässt, um so Informationen darüber zu bekommen, wie sicher sich das Netzwerk mit seiner Ausgabe ist. Dies sehen wir uns in dieser Aufgabe etwas genauer an.

   Das Ziel ist es, neben der Netzwerkausgabe selbst noch einen Unsicherheitsbereich in Form eines Erwartungswertes μ und einer Standardabweichung σ zu bekommen. Diese Information kann genutzt werden, um beispielsweise zu unterscheiden, ob die Ausgabe eher geraten wurde oder durch eine begründete Vorhersage entstand. „geraten“ meint in diesem Fall, dass die Eingabe dem Netzwerk so unbekannt ist, dass es keine andere Wahl hat, als einen fast schon zufälligen Wert zurückzugeben.

   Die Idee dabei ist, den Testvorgang nicht nur einmal, sondern mehrmals zu wiederholen. Dadurch, dass *droput* auch beim Testen verwendet wird, sind jedes Mal andere Neuronen aktiv und wir erhalten dadurch auch immer unterschiedliche Ausgaben. Von diesen Ausgaben können wir dann die statistischen Größen ableiten.

Um uns dieses Vorgehen zu verdeutlichen, wenden wir das Verfahren auf das zuvor trainierte *dropout*-Netzwerk an.

a) Um *dropout* auch bei der Berechnung der Netzwerkausgabe zu verwenden, müssen wir die Vorwärtsphase selbst implementieren. Im Wesentlichen benötigen wir dafür die drei Gleichungen

$$\begin{aligned} Y_1 &= \text{dropout}_p\left(\text{ELU}_\alpha\left(X \cdot W_1 + B_1\right)\right) \\ Y_2 &= \text{dropout}_p\left(\text{ELU}_\alpha\left(Y_1 \cdot W_2 + B_2\right)\right) \\ Y_3 &= Y_2 \cdot W_3 + B_3 \end{aligned} \tag{1}$$

mit der Eingabematrix X, den Gewichten W_i, den Bias-Matrizen B_i sowie den Ausgaben Y_i in den jeweiligen Schichten.

i. [Pen and Paper] Im Wesentlichen kommt es bei der Implementierung auf die korrekte Dimension der Matrizen an. Grundsätzlich steht dabei in jeder Zeile ein Eingabewert und in jeder Spalte der dazugehörige Ausgabewert des jeweiligen Neurons und ein Gewicht w_{kl} aus einer der Gewichtsmatrizen W_i gibt die Verbindungsstärke vom k-ten (Eingabe) zum l-ten (Ausgabe) Neuron an.

   Um mit diesem Konzept vertraut zu werden, ist es hilfreich, sich zuerst die Dimensionen der einzelnen Matrizen zu überlegen. Gib dazu für eine Eingabe $X \in \mathbb{R}^{1001\times 1}$ die unterschiedlichen Dimensionen an (Tabelle 1).

ii. [Python] Extrahiere die Gewichte und Bias-Werte aus dem trainierten *dropout*-Netzwerk.

iii. [Python] Implementiere eine Funktion `act(X)`, welche die ELU_α-Funktion für jeden Wert in `X` berechnet ($\alpha = 0.2$).

iv. [Python] Implementiere zur Berechnung von $\text{dropout}_p(X)$ eine Funktion `dropout(X)`. Dabei soll zufällig ein Anteil p der Spalten der Matrix X auf 0 gesetzt werden ($p = 0.3$). Dadurch werden effektiv einzelne Neuronen in den Schichten deaktiviert. Die Funktion `np.random.randint` könnte sich hier als hilfreich erweisen.

v. [Python] Implementiere nun eine Funktion `forward(X)`, die Gleichung 1 berechnet.

b) [Python] Berechne insgesamt 1000 Netzwerkausgaben und speichere alle in einer Matrix ab. Verwende dazu wiederholt die diskreten Eingabedaten

$$X = (-5, -4.99, \ldots, 4.99, 5) \in \mathbb{R}^{1001\times 1}.$$

c) [Python] Berechne für jeden Eingabewert x_i den Erwartungswert μ_i und die Standardabweichung σ_i anhand aller Netzwerkausgaben für diese Eingabe. Hierfür gibt es bereits Methoden, die direkt auf der Matrix mit den Netzwerkausgaben angewandt werden können.

d) [Python] Zeige die Datenpunkte, die Erwartungswerte $\boldsymbol{\mu}$ sowie den Bereich $[\boldsymbol{\mu} - \boldsymbol{\sigma}; \boldsymbol{\mu} + \boldsymbol{\sigma}]$ (eine Standardabweichung um den Erwartungswert) an. Für Letzteres ist die Funktion `fill_between` aus `matplotlib` hilfreich. Eine mögliche Ausgabe ist in Abbildung 3 zu sehen.

e) [Pen and Paper] Interpretiere das Ergebnis. Von was hängt die Größe des Unsicherheitsbereiches im Wesentlichen ab?

f) [Pen and Paper] In einer Animation[2] kann für einen ähnlichen Datensatz wie hier das Trainieren des Netzwerkes interaktiv beobachtet werden. Auch die Unsicherheit des Netzwerkes wird grafisch dargestellt. Starte die Animation und füge im linken oder rechten Bereich ein paar neue Punkte hinzu. Wie verändert sich dann die Unsicherheit in diesem Bereich und warum?

Tabelle 1: Zusammenfassung der Dimensionen, welche sich für die Matrizen in Gleichung 1 für Beispieleingabedaten ergeben (Aufgabe 1).

Matrix	Zeilen	Spalten
X	1001	1
W_1		
B_1		
Y_1		
W_2		
B_2		
Y_2		
W_3		
B_3		
Y_3		

[2] http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html#NPGcanvas

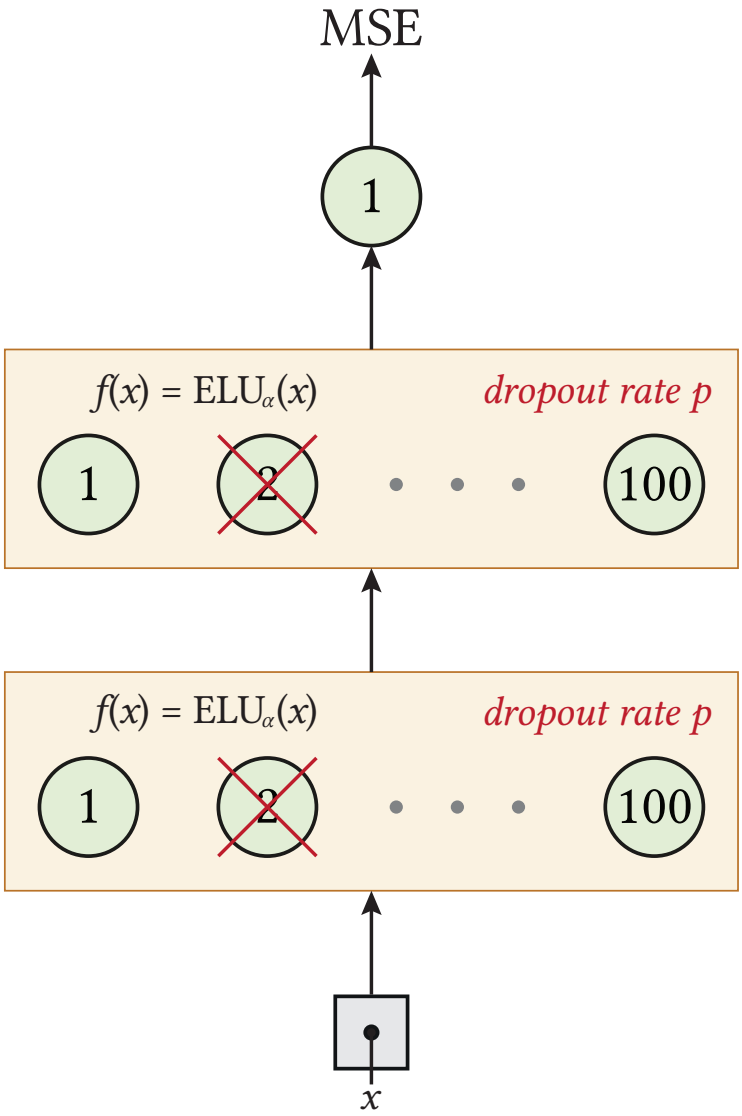


Abbildung 1: Netzwerkhierarchie für Aufgabe 1. Nach dem Eingabeneuron folgen zwei Schichten mit jeweils 100 Neuronen. Diese verwenden $\mathrm{ELU}_\alpha(x)$ mit $\alpha = 0.2$ als Transferfunktion und eine *dropout rate* $p = 0.3$. Auf die Eingabeschicht wird kein *dropout* angewendet, da es in diesem Beispiel nur ein Eingabeneuron gibt. Ein einzelnes abschließendes Ausgabeneuron fasst die Ergebnisse zusammen. Das Netzwerk wird mit Hilfe der quadratischen Fehlerfunktion trainiert. Zur übersichtlicheren Darstellungen sind nicht alle Verbindungen zwischen den Neuronen eingezeichnet.

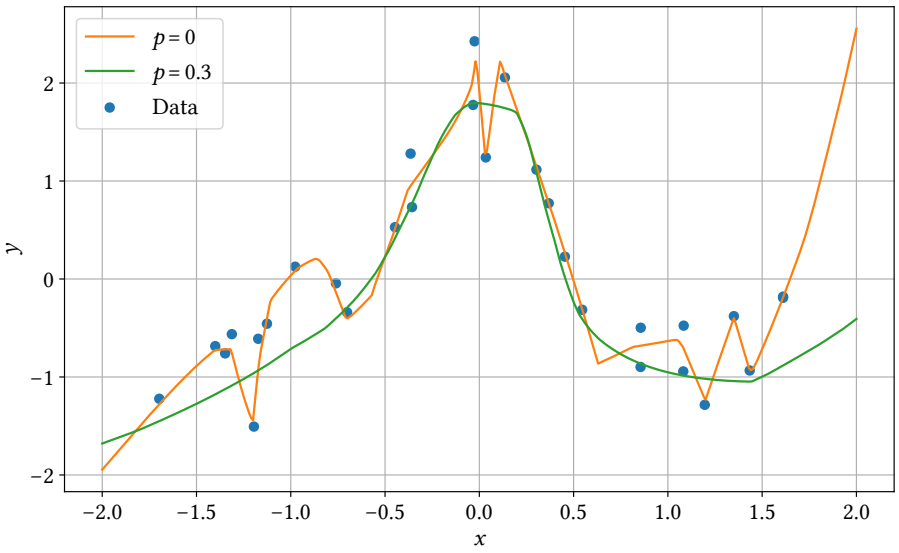


Abbildung 2: Vergleich der Netzwerkausgabe mit und ohne *dropout* für das Netzwerk aus Abbildung 1 (Aufgabe 1).

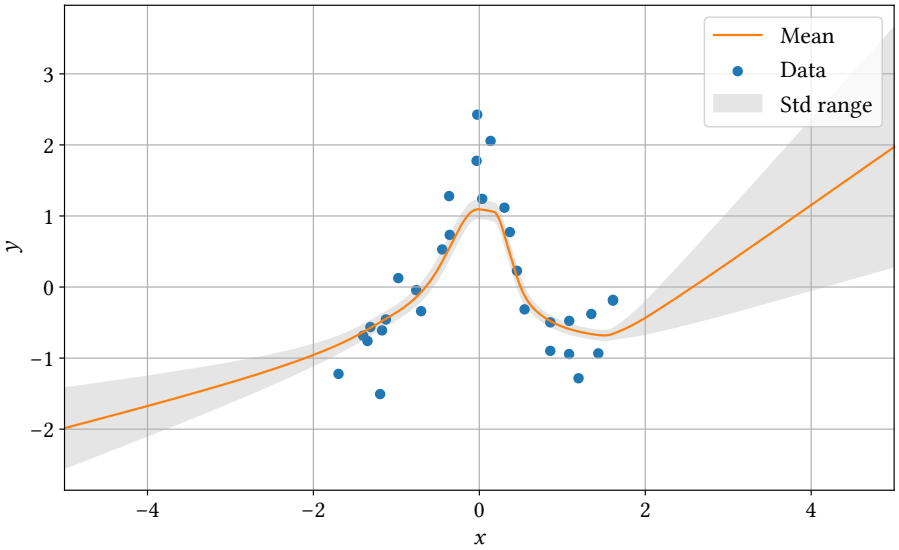


Abbildung 3: Unsicherheitsbereich in Form von einer Standardabweichung um den Erwartungswert herum für ein Regressionsproblem (Aufgabe 1).

Aufgabe 2 (3 Punkte): Radiale Basisfunktionen [Pen and Paper]

In dieser Aufgabe wollen wir uns ansehen, wie wir mit Hilfe von radialen Basisfunktionen zwischen Datenpunkten interpolieren können. Die Zielfunktion muss dabei exakt durch jeden Punkt verlaufen. Die Grundidee ist, um jede Stützstelle x_μ herum eine radialsymmetrische Basisfunktion

$$h(r) = h\left(\left\|x_\mu - x\right\|\right) = \frac{1}{\left(\left\|x_\mu - x\right\|^2 + 0.001\right)^\alpha} \tag{2}$$

zu platzieren. In diesem Fall verwenden wir dazu eine inverse multiquadratische Funktion und setzen den Parameter zuerst auf $\alpha = -1$. Die Interpolationsfunktion

$$g(x) = \sum_{v=1}^{p} w_v \cdot h\left(\|x_v - x\|\right) \tag{3}$$

können wir dann als eine Linearkombination der Basisfunktionen betrachten. Die Gewichte w_v werden dabei so gewählt, dass $g(x)$ durch die p Datenpunkte (x_μ, T_μ) verläuft. Dies wird durch das Gleichungssystem

$$g(x_\mu) = T_\mu \quad \mu = 1, \dots, p, \tag{4}$$

gewährleistet, welches auch kompakt in Matrixnotation $\boldsymbol{Hw} = \boldsymbol{T}$ notiert werden kann (siehe Skript). Als Beispiel wollen wir die folgenden Daten verwenden

$$\begin{aligned} x_1 &= 0, \quad x_2 = 1, \quad x_3 = 2 \\ T_1 &= 1, \quad T_2 = 4, \quad T_3 = -1. \end{aligned} \tag{5}$$

Unser Ziel ist es, für diese Daten die Interpolationsfunktion $g(x)$ zu berechnen.

1. Die Matrix $\boldsymbol{H}$ sei gegeben als

$$\boldsymbol{H} = \begin{pmatrix} h(\|x_1 - x_1\|) & h(\|x_1 - x_2\|) & h(\|x_1 - x_3\|) \\ h(\|x_2 - x_1\|) & h(\|x_2 - x_2\|) & h(\|x_2 - x_3\|) \\ 4.001 & h(\|x_3 - x_2\|) & h(\|x_3 - x_3\|) \end{pmatrix}.$$

   Berechne die fehlenden Einträge.

2. Für die ersten beiden Gewichte erhalten wir

$$w_1 = 1.752 \quad \text{und} \quad w_2 = -8.004.$$

   Berechne das fehlende Gewicht w_3 anhand von Gleichung 4. Hinweis: die Inverse $\boldsymbol{H}^{-1}$ braucht nicht berechnet zu werden.

3. Nun können wir $g(x)$ verwenden, um beliebige Funktionswerte zu berechnen.

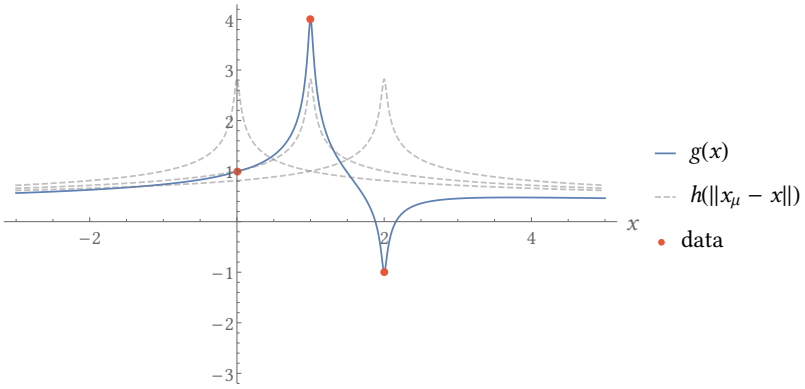
   a) Berechne $g(3)$.

   b) Handelt es sich bei $g(3)$ um einen interpolierten Funktionswert?

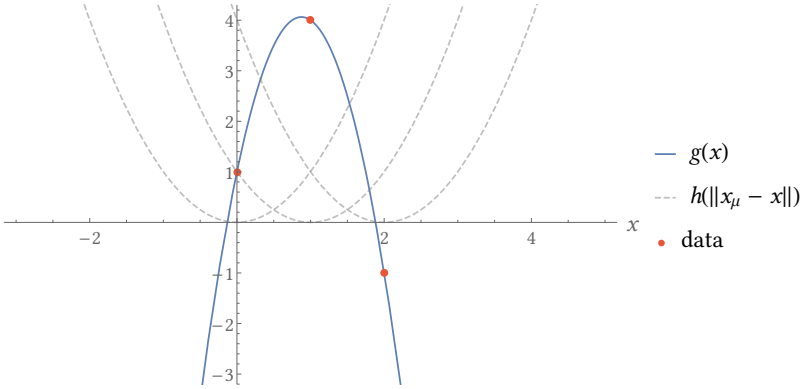
4. In Abbildung 4 sind für die Parametern $\alpha \in \{-1, -0.3, 0.15\}$ drei Graphen der Funktion $g(x)$ gezeigt. Ordne jede Grafik (A, B oder C) einem α-Wert zu und begründe deine Entscheidung.

Literatur

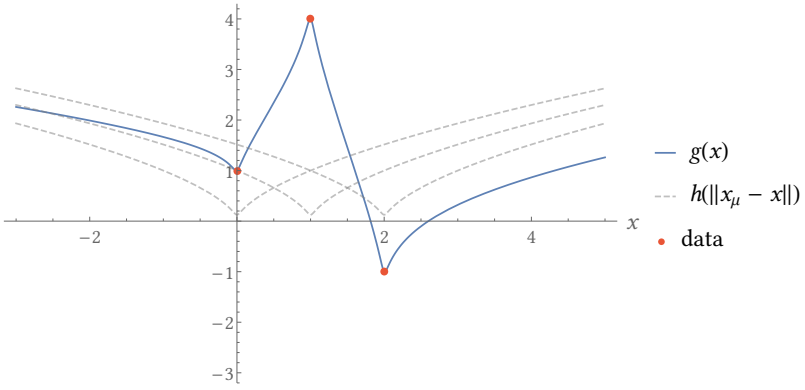
[1] Yarin Gal. *What my deep model doesn't know...* 3. Juli 2015. URL: http://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html (besucht am 17. 06. 2018).



(a) Grafik A



(b) Grafik B



(c) Grafik C

Abbildung 4: Funktion $g(x)$ (Gleichung 3) für die Beispieldaten aus Gleichung 5 mit unterschiedlichen Werten für den Parameter α. Neben der Interpolationsfunktion $g(x)$ sind die Basisfunktionen $h(r)$ aus Gleichung 2 sowie die Datenpunkte gezeigt.