Einführung in die Neuroinformatik SoSe 2019

Institut für Neuroinformatik
PD Dr. F. Schwenker

6. Aufgabenblatt (Abgabe bis 18. Juni 2019 zur Vorlesung)

Aufgabe 1 (5 Punkte): ***Cross Entropy***

Wir haben bis jetzt die quadratische Fehlerfunktion verwendet, um den Netzwerkfehler zu berechnen und daraus Lernregeln abzuleiten. Diese eignet sich gut für den Einstieg, da sie intuitiv zu interpretieren ist und Lernregeln leicht hergeleitet werden können. Gerade für Regressionsprobleme ist dies auch durchaus eine gebräuchliche Wahl als Fehlerfunktion.

Bei einem Klassifikationsproblem sieht dies jedoch anders aus. Hierbei ist jede Eingabe $x_\mu \in \omega_i$ Teil einer Klasse ω_i und das Lehrersignal T_μ repräsentiert das Klassenlabel (für gewöhnlich als Zahl $0, 1, \ldots, M-1$ bei einem M-Klassenproblem codiert). Die Netzwerkausgabe y_μ hat dann das Ziel, sich möglichst gut dem Klassenlabel anzunähern. Die quadratische Fehlerfunktion und die Verwendung eines einzelnen Ausgabeneurons ist nun weniger geeignet, da die Differenzen nicht mehr sinnvoll zu interpretieren sind.

Eine mögliche Lösung besteht darin, dafür zu sorgen, dass wir vom Netzwerk nicht mehr das Klassenlabel selbst, sondern eine diskrete Wahrscheinlichkeitsverteilung als Ausgabe bekommen. Diese gibt dann zu jedem Klassenlabel an, wie wahrscheinlich es ist, dass es sich bei der Eingabe um die jeweilige Klasse handelt. Wir werden uns in Aufgabe 2 ansehen, wie wir dafür konkret das Netzwerk aufbauen müssen.

Um Abschätzen zu können, wie gut die Wahrscheinlichkeitsverteilung des Netzwerkes ist, benötigen wir eine neue Kostenfunktion. Wir werden uns daher in dieser Aufgabe die *cross entropy*-Funktion ansehen. Diese ist für den Vergleich von zwei Wahrscheinlichkeitsverteilungen ausgelegt und wie der Name bereits vermuten lässt, spielt auch die Informationstheorie hier eine Rolle. Mathematisch ist sie definiert als

$$H_Q(P) = \sum_x P(x) \cdot \log_2\left(\frac{1}{Q(x)}\right) \tag{1}$$

und vergleicht die Verteilung $Q(x)$ mit der Verteilung $P(x)$. Das Ergebnis $H_Q(P)$ wird dabei in Bits gemessen. Wir benötigen auch die Entropie

$$H(P) = \sum_x P(x) \cdot \log_2\left(\frac{1}{P(x)}\right), \tag{2}$$

die uns den mittleren Informationsgehalt in Bits für ein Alphabet liefert, das gemäß der Wahrscheinlichkeitsverteilung $P(x)$ generiert wurde.

1. [Pen and Paper] Für ein $M = 3$ Klassenproblem seien für die beiden Eingaben x_1 und x_2 die Netzwerkausgabe y_μ sowie das Lehrersignal T_μ gegeben:

Eingabe	T_μ	y_μ
x_1	1	3
x_2	1	2

Erkläre anhand dieses Beispiels, warum die quadratische Fehlerfunktion und die Verwendung von nur einem (skalaren) Ausgabewert y_μ hier nicht geeignet ist.

2. [Pen and Paper] Lies dich in den Artikel *Visual Information Theory*[1] von Christopher Olah ein. Hier wird ein grafischer Einstieg in das Thema Codierungen und Entropie geboten. Nicht alles ist relevant, aber wenigstens die Abschnitte *Calculating Entropy* und *Cross-Entropy* sollten gelesen werden. Wir arbeiten hier mit den dort verwendeten Beispielen.

3. [Pen and Paper] Nehmen wir an, dass Alice und Bob die vier möglichen Wörter x_1, x_2, x_3 und x_4 gemäß den folgenden Wahrscheinlichkeitsverteilungen wählen

$$A(x_1, x_2, x_3, x_4) = \left(\frac{1}{8}, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}\right) \quad \text{und} \quad B(x_1, x_2, x_3, x_4) = \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right).$$

Die Entropie beträgt in beiden Fällen $H(A) = H(B) = 1.75$.

a) Wir wollen die beiden Wahrscheinlichkeitsverteilungen $A(x)$ und $B(x)$ anhand der *cross entropy* (Gleichung 1) vergleichen. Berechne, wie viele Bits (im Schnitt) notwendig sind, wenn Bob Nachrichten mit der Kodierung von Alice versendet, d. h. $H_A(B)$.

b) Betrachten wir nun den umgekehrten Fall $H_B(A)$. Wie viele Bits sind im Schnitt notwendig, wenn Alice Nachrichten mit der Codierung von Bob sendet?

c) Wie viele Bits sind im Schnitt notwendig, wenn Bob Nachrichten gemäß seiner eigenen Codierung versendet ($H_B(B)$)? Hinweis: hierfür muss nichts berechnet werden.

d) Die *cross entropy* $H_Q(P)$ nimmt ihr Minimum bei $Q = P$ an. In diesem Fall sind beide Wahrscheinlichkeitsverteilungen identisch. Das Minimum entspricht dabei genau der Entropie $H(Q)$. Interessant ist daher die Differenz

$$D_Q(P) = H_Q(P) - H(P),$$

welche auch als *Kullback–Leibler-Divergenz* bekannt ist. Damit messen wir, wie viele Bits im Schnitt mehr für Nachrichten generiert aus $P(x)$ benötigt werden, da die Codierung $Q(x)$ verwendet wird (und nicht $P(x)$). Berechne $D_A(B)$, $D_B(A)$ sowie $D_B(B)$.

4. [Pen and Paper] Analog zu Distanzmaßen, welche für Datenpunkte definiert sind, soll für ein Distanzmaß $d_Q(P) : X \times X \longrightarrow \mathbb{R}_+$ zwischen zwei diskreten Wahrscheinlichkeitsverteilungen $Q, P \in X$ (X bezeichne die Menge aller diskreten Wahrscheinlichkeitsverteilungen) drei Eigenschaften gelten

[1] https://colah.github.io/posts/2015-09-Visual-Information/

I. $d_Q(P) \geq 0$ für alle $Q, P \in X$.

II. $d_Q(P) = d_P(Q)$ für alle $Q, P \in X$.

III. $d_Q(Q) = 0$ für alle $Q \in X$.

Zeige, dass zwei der drei Eigenschaften von der Kullback–Leibler-Divergenz $D_Q(P)$ erfüllt werden. Für die 1. Eigenschaft könnte sich die Gibbs-Ungleichung als hilfreich erweisen.

5. Um genauer zu verstehen, wie die Kullback–Leibler-Divergenz $D_Q(P)$ sich verhält, fügen wir eine neue Person Charles ein, der seine Wörter gemäß der Verteilung

$$C_t(x_1, x_2, x_3, x_4) = \left(t \cdot \frac{3}{4}, (1-t) \cdot \frac{3}{4}, \frac{1}{8}, \frac{1}{8}\right)$$

mit dem Parameter $t \in [0; 1]$ wählt. Wir wollen nun sehen, wie sich die Divergenz $D_C(B)$ verändert, wenn wir die Codierung von Charles anpassen und dabei Wörter gemäß $B(x)$ generieren.

a) [Pen and Paper] Für welchen Wert von t erwartest du ein Minimum von $D_C(B)$? Hierbei ist keine Herleitung notwendig. Es reicht, eine Begründung abzugeben. Tipp: Ergebnis aus Aufgabenteil 3 d).

b) [Python] Simuliere diesen Sachverhalt für unterschiedliche Werte für t. Beginne bei $t_{\text{start}} = 0.01$ und ende bei $t_{\text{end}} = 0.99$ (jeweils einschließlich) bei einer Schrittweite von $\Delta t = 0.01$ und zeige das Ergebnis in einem Plot an, in dem auch das Minimum markiert ist. Eine mögliche Ausgabe ist in Abbildung 1 abgebildet. Für das Testskript bitte auch Tabelle 1 beachten.

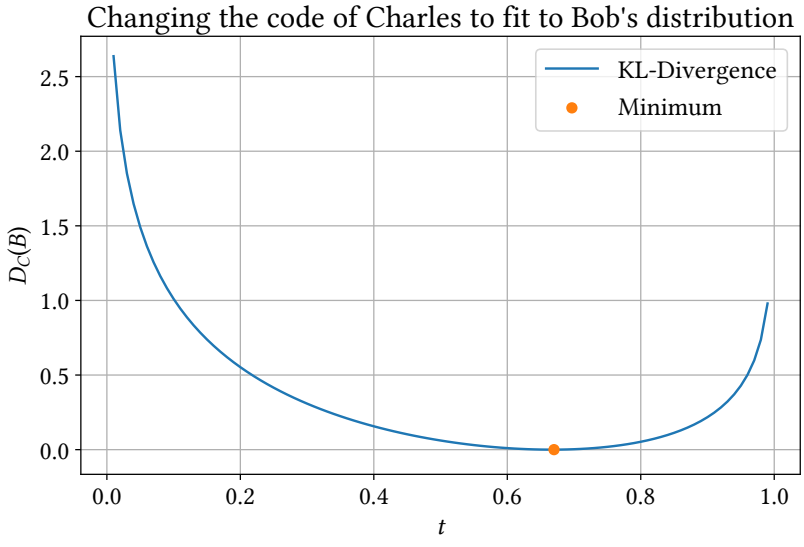


Abbildung 1: Kullback–Leibler-Divergenz $D_C(B)$ unter der Anpassung der Codierung von Charles C_t mit dem Parameter t.

Tabelle 1: Variablennamen für das Jupyter-Notebook `CrossEntropy.ipynb` aus Aufgabe 1. Die Variablen müssen exakt so benannt werden, damit das Testskript deren Inhalt überprüfen kann.

Variablenname	Typ	Beschreibung
`bob`	Vektor	Die Werte $B(\boldsymbol{x})$ als Vektor gespeichert.
`tValues`	Vektor	Diskrete Zeitwerte $t \in \{0.01, 0.02, \dots, 0.99\}$.
`KLdivergences`	Vektor	Berechnete Werte für die Kullback–Leibler-Divergenz $D_C(B)$ (für alle Parameterwerte t).

Aufgabe 2 (5 Punkte): ***Cross Entropy* als Kostenfunktion [Pen and Paper]**

Nachdem wir uns in Aufgabe 1 mit den Grundlagen der *cross entropy*-Funktion vertraut gemacht haben, wollen wir uns nun ansehen, wie sich diese im Kontext von neuronalen Netzwerken verwenden lässt. Zudem überlegen wir uns, wie wir das Netzwerk dazu bringen, eine Wahrscheinlichkeitsverteilung auszugeben und was eine veränderte Kostenfunktion für Auswirkungen auf die Lernregeln hat.

Wie bereits in der vorherigen Aufgabe erwähnt, soll es dabei um Klassifikationsprobleme gehen und das Netzwerk gibt nicht das Klassenlabel selbst, sondern eine diskrete Wahrscheinlichkeitsverteilung aus. Konkret gibt die Ausgabe $y_2 = 0.8$ des 2. Ausgabeneurons beispielsweise an, dass das Netzwerk die Eingabe zu 80 % für ein Objekt der Klasse ω_2 hält.

Die *cross entropy*-Funktion hilft uns dabei, die ausgebende Wahrscheinlichkeitsverteilung des Netzwerkes zu bewerten, indem es mit der korrekten Verteilung gemäß des Lehrersignals verglichen wird. Das Lehrersignal muss daher selbst auch als Wahrscheinlichkeitsverteilung vorliegen, ansonsten können wir Gleichung 1 nicht anwenden. Jeder Datenpunkt besitzt aber erst einmal nur ein Label. Um daraus nun eine gültige Verteilung zu machen, greift man auf sogenannte *one-hot*-Vektoren $\boldsymbol{t} = (t_1, t_2, \dots, t_n)$ zurück, welche an der i-ten Position eine 1 und sonst nur mit 0en gefüllt sind. Die 1 landet dabei an genau der Stelle, welche dem Klassenlabel entspricht. Bei einem 3-Klassenproblem wird beispielsweise

$$\boldsymbol{t} = (0, 1, 0)$$

für das Label der Klasse ω_2 verwendet.

Damit die Netzwerkausgabe $\boldsymbol{y} = (y_1, y_2, \dots, y_n)$ als Wahrscheinlichkeitsverteilung interpretiert werden kann, benötigen wir spezielle Neuronen in der Ausgabeschicht. Eine übliche Wahl ist dabei einen *softmax*-Layer zu verwenden. Hierbei werden alle n Ausgabeneuronen gemäß der Vorschrift

$$y_i = \frac{e^{c \cdot u_i}}{\sum_{j=1}^{n} e^{c \cdot u_j}} \tag{3}$$

mit dem Parameter $c \in \mathbb{R}$ skaliert. $u_i = w_i \cdot x + b_i$ bezeichnet dabei das dendritische Potenzial des i-ten Ausgabeneurons.

1. Wir wollen als nächstes verstehen, warum man bei Gleichung 3 von einer *softmax*-Funktion spricht und warum sie dafür sorgt, dass unser Netzwerk eine gültige Wahrscheinlichkeitsverteilung ausgibt. Zur Vereinfachung gehen wir in diesem Teil der Aufgabe von einem 3-Klassenproblem aus. Unser Netzwerk besitzt in der Ausgabeschicht demnach $n = 3$ Neuronen.

   a) Zeige zunächst, dass die Netzwerkausgabe $\boldsymbol{y}$ nach Gleichung 3 eine gültige diskrete Wahrscheinlichkeitsverteilung ist. Prüfe dazu die beiden Bedingungen

   $$\sum_{i=1}^{n} y_i = 1 \quad \text{und} \quad \forall i : y_i \geq 0.$$

   b) Zeige, dass für unsere 3 Ausgabeneuronen sich die Ausgabe des 1. Neurons y_1 darstellen lässt als

   $$y_1 = \frac{1}{1 + e^{c\cdot(u_2-u_1)} + e^{c\cdot(u_3-u_1)}}.$$

   c) Interessant ist das Ergebnis, was wir bekommen, wenn wir den Parameter c größer werden lassen. Berechne dazu den Grenzwert $\lim_{c\to\infty} y_1$ für die Fälle

      i. $u_1 > u_2 > u_3$,

      ii. $u_2 > u_1 > u_3$,

      iii. $u_2 > u_3 > u_1$.

   d) Wie das *soft* im Namen bereits vermuten lässt, bekommen wir durch Gleichung 3 eine gedämpfte Version der dendritischen Potenziale u_i. Wie stark diese Dämpfung ist, hängt dabei vom Parameter c ab. In einer Animation[2] wird das Ergebnis der *softmax*-Funktion für Beispielwerte gezeigt. Beschreibe die Ergebnisverteilung für die drei Fälle

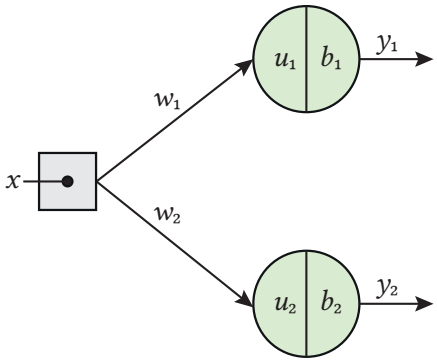
      i. $c > 0$,

      ii. $c = 0$,

      iii. $c < 0$.

2. Wenn wir eine andere Kostenfunktion verwenden, ändern sich für gewöhnlich auch die Lernregeln für das Netzwerk. Wir wollen uns jetzt ansehen, wie diese für die *cross entropy*-Funktion aussehen. Dazu betrachten wir ein einfaches Netzwerk, bei dem die beiden Ausgabeneuronen den *softmax*-Layer bilden und ihren Input nur von einem Eingabeneuron bekommen:

[2] https://milania.de/showcase/The_softmax_function_in_the_output_layer_of_neural_networks



Da wir nur 2 Ausgabeneuronen haben, kann unser Netzwerk auch nur zwei Klassen abbilden. Für die Fehlerfunktion nehmen wir die *cross entropy*-Funktion aus Gleichung 1. Aus Vereinfachungsgründen ignorieren wir jedoch die Summe über alle Datenpunkte M. Zudem verwenden wir zur besseren Abstimmung mit Gleichung 3 $\ln(x)$ anstatt $\log_2(x)$. Damit ergibt sich für unser Beispielnetzwerk die Fehlerfunktion[3]

$$E = D_{\boldsymbol{y}}(\boldsymbol{t}) = H_{\boldsymbol{y}}(\boldsymbol{t}) = -t_1 \cdot \ln\left(y_1\left[u_1(w_1), u_2(w_2)\right]\right) - t_2 \cdot \ln\left(y_2\left[u_1(w_1), u_2(w_2)\right]\right).$$

Konkret wollen wir uns hier die Ableitung nach dem Gewicht w_2 ansehen. Die Verwendung der *softmax*-Funktion führt dazu, dass die Ausgabe $y_i(u_1, u_2)$ nicht nur vom eigenen, sondern auch vom dendritischen Potenzial aller anderen Neuronen abhängt. Dadurch tauchen beide Gewichte jeweils in den $\ln(x)$-Komponenten auf. Entsprechend ist auch die Ableitung nach Anwendung der Kettenregel etwas umfangreicher

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial u_2}\frac{\partial u_2}{\partial w_2} + \frac{\partial E}{\partial y_2}\frac{\partial y_2}{\partial u_2}\frac{\partial u_2}{\partial w_2}. \tag{4}$$

Das Ergebnis der Ableitung wollen wir nun schrittweise herleiten. Den Dämpfungsparameter setzen wir auf $c = 1$.

a) Beginnen wir mit den Ableitungen nach den Netzwerkausgaben y_i. Bestimme dazu die beiden Komponenten

$$\frac{\partial E}{\partial y_1} = \dots$$
$$\frac{\partial E}{\partial y_2} = \dots$$

b) Als nächstes wollen wir ermitteln, wie sich die Netzwerkausgaben y_i abhängig vom dendritischen Potenzial u_2 des 2. Neurons verändern (die Ableitung findet in beiden Fällen nach u_2 statt, da nur hier das Gewicht w_2 auftaucht). Zeige, dass diese sich

[3] Die Entropie von einem *one-hot*-Vektor ist immer 0, daher fällt der Entropieterm $H(\boldsymbol{t}) = 0$ weg.

wie folgt ergeben:

$$\frac{\partial y_1}{\partial u_2} = -y_1 \cdot y_2$$
$$\frac{\partial y_2}{\partial u_2} = y_2 \cdot (1 - y_2)$$

c) Zu guter Letzt bestimme noch die Ableitung des dendritischen Potenzials $u_2 = w_2 \cdot x + b_2$ nach dem Gewicht w_2

$$\frac{\partial u_2}{\partial w_2} = \dots$$

d) Nun ist es an der Zeit, unsere Ergebnisse in Gleichung 4 einzusetzen. Zeige, dass sich die Ableitung vereinfachen lässt zu

$$\frac{\partial E}{\partial w_2} = (y_2 - t_2) \cdot x. \tag{5}$$

Hinweis: $t_1 + t_2 = 1$.

e) Vergleiche das Ergebnis aus Gleichung 5 mit dem der quadratischen Fehlerfunktion (Aufgabenblatt 03). Worin besteht der wesentliche Unterschied in den Ableitungsvorschriften (abgesehen von der fehlenden Summe über die Datenpunkte)?