Fakultät für Informatik
Übung Grundlagen der künstlichen Intelligenz
Wintersemester 2024/25

Ulrich Oberhofer
ulrich.oberhofer@kit.edu
Sebastian Pütz
sebastian.puetz@kit.edu

Übungsblatt 3

Neuronale Netzwerke und Backpropagation

Abgabe online auf ILIAS bis 5. Dezember 2024, 12:00 Uhr

Für die Abgabe bitte sowohl die handgeschriebene Lösung zu Aufgabe 1, als auch das bearbeitete Jupyter Notebook zu Aufgabe 2 im Ilias hochladen.

Aufgabe 1 *Backpropagation Algorithmus* (9 Punkte)

In Aufgabe 2 dieses Übungsblattes möchten wir den Backpropagation Algorithmus selbst implementieren, um damit Bilder von handgeschriebenen Ziffern zu klassifizieren. Als Vorarbeit müssen wir jedoch zunächst die notwendigen Formeln für den Backpropagation Algorithmus herleiten.

Wir betrachten ein vollständig verbundenes (fully-connected) neuronales Netz mit H Layern, wobei jeder Layer eine beliebige Anzahl von Neuronen besitzen kann. Wie schon in der Vorlesung beschrieben wird die Aktivierung von Layer $L \in \{1, \dots, H\}$ folgendermaßen berechnet:

$$a_m^L = h^L \left(\underbrace{\sum_i w_{im}^L \cdot a_i^{L-1} + b_m^L}_{\equiv z_m^L} \right) \tag{1}$$

Die Gewichtsmatrix (Weight Matrix) $\boldsymbol{w}^L$ und der Bias Vektor $\boldsymbol{b}^L$ werden also verwendet, um, gegeben die Aktivierungen $\boldsymbol{a}^{L-1}$ von Layer $L-1$, die Aktivierung $\boldsymbol{a}^L$ von Layer L zu bestimmen. Die Funktion $h^L(\cdot)$ bezeichnet die Aktivierungsfunktion des Layers L . Für die hidden layers $(0 < L < H)$ verwenden wir die **Sigmoid-Funktion**: $\sigma(z_j) = \frac{1}{1+\exp(-z_j)}$, also $h^L(\boldsymbol{z}) = \sigma(\boldsymbol{z})$.

Für das Input-Layer $(L = 0)$ setzen wir die Eingabedaten $\boldsymbol{a}^0 = \boldsymbol{x}$. Das Output-Layer $(L = H)$ liefert die Ausgabe des Netzwerks über die Aktivierung der Output-Neuronen $\boldsymbol{a}^H$

Der Output des Netzwerkes soll Wahrscheinlichkeiten für jede mögliche handgeschriebene Ziffer (0 bis 9, also 10 Klassen) darstellen. Für ein solches Multiclass-Klassifikationsproblem bietet sich die **Softmax-Funktion** als output activation an, um normalisierte Wahrscheinlichkeiten zu erhalten:

$$\text{softmax}(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{10} e^{z_k}} \tag{2}$$

Wir verwenden also $h^H(\mathbf{z}) = \text{softmax}(\mathbf{z})$ als finale Aktivierungsfunktion.

Um die Fehler des Modells zu bewerten, nutzen wir die **Cross-Entropy-Lossfunktion** (siehe Vorlesung). Sei $\boldsymbol{y}$ der wahre Label-Vektor (als One-Hot-Codierung) und $\boldsymbol{a}^H$ der Output unseres Modells, dann ist der Loss für eine einzelne Eingabe definiert als:

$$\mathcal{L}(\boldsymbol{y}, \boldsymbol{a}^H) = -\sum_{i=1}^{10} y_i \cdot \log\left(a_i^H\right) \tag{3}$$

Beim Training mit Batches berechnen wir den mittleren Loss über alle Samples im Batch.

Nun haben wir alle Komponenten unseres neuronalen Netzes definiert. Um dieses nun mit (Mini-batch) gradient descent trainieren zu können benötigen wir die Ableitung unserer Lossfunktion nach den Gewichten (Weights) und Biases unseres Netzes (siehe Vorlesung). Zunächst benötigen wir dafür etwas Vorarbeit:

---

a) Zeigt, dass die Ableitung der Sigmoidfunktion $\sigma(z) = \frac{1}{1+\exp(-z)}$ durch die folgende Beziehung ausgedrückt werden kann: (2 Punkte)

$$\sigma'(z) = \sigma(z) \cdot (1 - \sigma(z)) \tag{4}$$

---

b) Zeige, dass für die Ableitung der Softmaxfunktion $\text{softmax}(\mathbf{z})_j$ Folgendes gilt: (2 Punkte)

$$\frac{\partial \text{softmax}(\mathbf{z})_i}{\partial z_j} = \text{softmax}(\mathbf{z})_i \cdot (\delta_{ij} - \text{softmax}(\mathbf{z})_j), \tag{5}$$

$$\text{mit } \delta_{ij} = \begin{cases} 1 & \text{falls} \quad i = j \\ 0 & \text{falls} \quad i \neq j \end{cases} \tag{6}$$

Wir möchten nun Ableitungen unserer Lossfunktion L nach den Parametern w_{ij}^L und b_i^L bestimmen:

$$\frac{\partial \mathcal{L}}{\partial w_{mn}^L} = \frac{\partial \mathcal{L}}{\partial z_n^L} \cdot \frac{\partial z_n^L}{\partial w_{mn}^L} = \frac{\partial \mathcal{L}}{\partial z_n^L} \cdot \frac{\partial \sum_i w_{in}^L \cdot a_i^{L-1} + b_n^L}{\partial w_{mn}^L} = \underbrace{\frac{\partial \mathcal{L}}{\partial z_n^L}}_{\equiv \delta_n^L} \cdot a_m^{L-1} \tag{7}$$

$$\frac{\partial \mathcal{L}}{\partial b_n^L} = \frac{\partial \mathcal{L}}{\partial z_n^L} \cdot \frac{\partial z_n^L}{\partial b_n^L} = \frac{\partial \mathcal{L}}{\partial z_n^L} \cdot \underbrace{\frac{\partial \sum_i w_{in}^L \cdot a_i^{L-1} + b_n^L}{\partial b_n^L}}_{=1} = \underbrace{\frac{\partial \mathcal{L}}{\partial z_n^L}}_{\equiv \delta_n^L} \tag{8}$$

Tipp: Zum besseren Verständnis kann es helfen einmal den computational Graph des forward pass aufzuzeichnen.

Wir müssen also nur $\delta_n^L \equiv \frac{\partial \mathcal{L}}{\partial z_n^L}$ bestimmen, dann können wir sämtliche Ableitungen die wir benötigen damit ausdrücken.

c) Zeigt, dass folgende Formel für δ_n^H gilt, also für δ_n des Output-Layers: (2 Punkte)

$$\delta_n^H = \frac{\partial \mathcal{L}}{\partial z_n^H} = a_n^H - y_n \tag{9}$$

Dabei ist y_n unser Traininglabel, also das onehot-encoding der richtigen Klasse. Beachte hierbei, dass die finale Aktivierungsfunktion die Softmax-Funktion ist. Bei dieser Aufgabe ist das Ergebnis von Aufgabenteil b) nützlich.

Nun haben wir δ_n^H für das letzte Layer bestimmt. Von hier können wir jetzt rekursiv rückwärts die δ_n^L der vorigen Layer bestimmen:

d) Zeigt, dass folgende Rekursionsformel gilt, um die δ_n^L der vorigen Layer zu bestimmen: (3 Punkte)

$$\delta_n^L = h'\left(z_n^L\right) \sum_m \delta_m^{L+1} \cdot w_{nm}^{L+1} \tag{10}$$

Folgender Start für deinen Beweis könnte hilfreich sein:

$$\delta_n^L = \frac{\partial \mathcal{L}}{\partial z_n^L} = \sum_m \frac{\partial \mathcal{L}}{\partial z_m^{L+1}} \cdot \frac{\partial z_m^{L+1}}{\partial z_n^L} \tag{11}$$

Mit dieser Rekursionsformel können wir nun in Aufgabe 2 den vollständigen, im Folgenden kurz beschriebenen Backpropagation Algorithmus implementieren:

- Forward pass: Verwende Gleichung 1, um für jedes Sample der aktuellen Batch die Aktivierungen für jedes Layer zu bestimmen. Hierbei müssen wir für jedes Sample für jedes Layer z_m^L zwischenspeichern, da wir sonst die Rekursionsformel der Backpropagation nicht anwenden können.

- Backward pass
  - Bestimme δ_n^H für das letzte Layer mit Gleichung 9
  - Bestimme rekursiv (vom letzten Layer zum ersten Layer) für alle Layer δ_n^L mit Gleichung 10
  - Mit den so gewonnenen δ_n^H kann über Gleichung 7 und 8 für jedes Sample in der Batch die Ableitung des Loss nach den Gewichten bestimmt werden.
- Minibatch gradient descent: Update die Gewichte des Netzwerks mit Hilfe der Ableitung des Loss nach den Gewichten (gemittelt über die Batch)

Aufgabe 2 *Implementierung Backpropagation* (11 Punkte)

Bitte löst für diese Aufgabe das im Ilias zur Verfügung gestellte Jupyter Notebook.