Backpropagation, das lernen eines Neuronalen Netzes aus Daten

Felix Reinbott

March 20, 2020

1 Backpropagation

Definition 1. *(Feed Forward Neural Network mit einer Hidden Layer)*
Ein Neuronales Netz mit einer Hidden Layer der Größe l mit Inputs $X \in \mathbb{R}^n$ und Outputs $Y \in \mathbb{R}^d$ ist eine Funktion der Form

$$f_W \colon \mathbb{R}^n \to \mathbb{R}^d, x \mapsto f(x) := \sigma_2(W_2\sigma_1(W_1x)) \quad W_1 \in \mathbb{R}^{l\times n}, W_2 \in R^{d\times l} \tag{1}$$

Wobei hier σ_1, σ_2 Vektorfelder (auch genannt Aktivierungsfunktionen) auf dem entsprechenden Vektorraum sind.

Definition 2. *(Mean Squared Error)*
Die Verlustfunktion des Mean Squared Errors ist gegeben als

$$d_{MSE} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, (x, y) \mapsto \|x - y\|_2^2 \tag{2}$$

Satz 3. *(Geschlossene Form des Gradienten bezüglich den Gewichten)*
*Seien Daten $(X_i, Y_i)_{i=1,...,m}$ gegeben und seien die Vektorfelder σ_i in jeder Komponente mit der Funktion $\sigma(x) = \frac{1}{1+exp(-x)}$ gegeben, d.h. $\sigma_i(x) = (\sigma(x_1), ..., \sigma(x_n)^T$. Der Gradient eines Neuronalen Netzes der Form (**??**) hat bezüglich dem Minimierungsproblem*

$$\frac{d}{dW_{j,k,.}} d_{MSE}(f_W(X_i), Y_i) \quad i \in \{1, ..., m\} \tag{3}$$

eine geschlossene Form.

Beweis. Wir berechnen zuerst die Jakobimatrix der Vektorfelder

$$J_{\sigma_i}(x) = \begin{pmatrix} \frac{d}{dx_1}\sigma(x_1) & ... & 0 \\ ... & ... & ... \\ 0 & ... & \frac{d}{dx_n}\sigma(x_n) \end{pmatrix} = \begin{pmatrix} exp(-x_1)\sigma(x_1)^2 & ... & 0 \\ ... & ... & ... \\ 0 & ... & exp(-x_n)\sigma(x_n)^2 \end{pmatrix}$$

Als nächstes berechnen wir den Gradienten bezüglich $W_{2,k,.}$ und stellen fest, dass in der zweiten Matix eine Zeile nur auf einen Wert in der Zielvariable zeigt und dementsprechend beim Differenzieren nach der i-ten Zeile auch bloß die i-te Komponente des Outputs eine Rolle spielt. Es folgt für $k = 1, ..., d$ und $h := \sigma_1(W_1x)$

$$
\begin{aligned}
&\frac{d}{dW_{2,k,.}} d_{MSE}(f_W(X_i), Y_i) \\
=&\frac{d}{dW_{2,k,.}} \|f_W(X_i) - Y_i\|_2^2 \\
=&2\big((f_W(X_i))_k - Y_{i,k}\big) \frac{d}{dW_{2,k,.}} (f_W(X_i))_k \\
=&2\big((f_W(X_i))_k - Y_{i,k}\big) \frac{d}{dW_{2,k,.}} \sigma_2(W_2 h)_k \\
=&2\big((f_W(X_i))_k - Y_{i,k}\big) exp(-W_{2,k,.}h)\sigma(W_{2,k,.}h)^2 \frac{d}{dW_{2,k,.}} W_{2,k,.}h \\
&= 2\big((f_W(X_i))_k - Y_{i,k}\big) exp(-W_{2,k,.}h)\sigma(W_{2,k,.}h)^2 h
\end{aligned}
$$

Nun betrachten wir den Gradienten bezüglich den Zeilen der Gewichtematrix W_1 für $k = 1, ..., l$ und $\mathbb{1}_d = (1, ..., 1)^T \in \mathbb{R}^d$

$$
\begin{aligned}
&\frac{d}{dW_{1,k,.}} d_{MSE}(f_W(X_i), Y_i) \\
=&2\big(f_W(X_i) - Y_i\big)^T J_{\sigma_2}(W_2 h) \mathbb{1}_d \frac{d}{dW_{1,k,.}} (\sigma_1(W_1 X_i))^T \\
=&2\big(f_W(X_i) - Y_i\big)^T J_{\sigma_2}(W_2 h) \mathbb{1}_d exp(-W_{1,k,.}h)\sigma(W_{1,k,.}h)^2 \frac{d}{dW_{1,k,.}} X_i^T W_{1,k,.}^T \\
=&2\big(f_W(X_i) - Y_i\big)^T J_{\sigma_2}(W_2 h) \mathbb{1}_d exp(-W_{1,k,.}h)\sigma(W_{1,k,.}h)^2 X_i^T
\end{aligned}
$$

□

Bemerkung 4. *Damit können wir mit der Eigenschaft, dass* $-\nabla_W f_W(X_i)$ *für einen Datenpunkt* (X_i, Y_i) *in die Richtung der zu optimierenden Gewichte schrittweise mit einem Verfahren in Richtung der "bessere" Gewichte gehen. Dies Motiviert folgenden Algorithmus*

(i) berechne für jede Zeile der Gewichtematrizen den Gradienten

(ii) aktualisiere für $\lambda \in \mathbb{R}_+$ *klein genug die Zeilen der Gewichtematrizen durch*

$$W^{z+1}_{j,k,\cdot} = W^z_{j,k,\cdot} - \lambda \frac{d}{dW_{j,k,\cdot}} d_{MSE}(f_W(X_i), Y_i)$$

(iii) Wiederhole für ein zufällig gezogenes Paar an Inputs und Outputs (X_i, Y_i) *bis die Verlustfunktion sich nicht mehr wesentlich verbessert*

In dieser Form sind einfache Neutronale Netze also mit Methoden der Analysis II in einer einfachen Form zu verstehen und man kann einen naiven Trainingsalgorithmus aufstellen der das Minimierungsproblem gegeben den Daten approximativ löst.

Code: `https://github.com/FelixRb96/mnist_for_calc1`