Prof. Dr. Andreas Hotho, Albin Zehe, Jonas Kaiser
Data Science Chair

11./12.07.2024

8. Assignment in "Machine Learning for Natural Language Processing"

Summer Term 2024

1 General Questions

1. Describe two tasks that are better suited for RNN models than for MLP models and provide a short explanation.

   a) Recommending items in an online store based on user click paths. RNN models explicitly account for the order of clicks and thus are better suited to extract temporal patterns from the click path. Additionally, they can easily accommodate sessions of different lengths.

   b) Language modelling. The order of words is important to correctly model sentences. RNNs are a natural fit, because they explicitly model sequences.

2.
   - What are the problems in Machine Translation with simple RNNs that can be solved by Sequence to Sequence models?
   - What additional problems can be solved with the Attention Mechanism?

   - Expected output of different length than input, sometimes words that occur later in the sentence are important for translating a word that occurs earlier.
   - Attention solves the "information bottleneck" produced by encoding the entire input into one vector representation and enables the decoder to focus on the most important parts for each output step.

3. What are the key parts of the self-attention module in the transformer architecture? Explain each of these components.

We can think of the self-attention mechanism as some kind of database request. We have an input **query** that is matches with items that consist of a **key** and a **value**. The more similar the query to the key, the more important the value. The self-attention computes these three components from the input.

2 Attention

Given a Sequence to Sequence model with an encoder and decoder RNN, that translates text from English to German. Apply the (Luong-)attention mechanism to the hidden states of the encoder RNN.

For the input "I love Language Processing", the encoder produces the following hidden states h_i:

$$h_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}, h_2 = \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}, h_3 = \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix}, h_4 = \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}.$$

The current output of the decoder is "Ich", the decoder's hidden state is

$$s_1 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}.$$

Calculate the context vector c, which is used to predict the next word.

1. First, the dot product of s_2 with all h_i is calculated.

$$h_1^T s_1 = 2, \quad h_2^T s_1 = 4, \quad h_3^T s_1 = 2, \quad h_4^T s_1 = 2$$

2. These values are then normalized using the softmax function. This results in attention weights of

$$a_1 = 0.09625514, \quad a_2 = 0.71123459, \quad a_3 = 0.09625514, \quad a_4 = 0.09625514$$

3. Finally, the hidden states are weighted using the attention weights and summed up. This results in the context vector

$$c = 0.09625514 \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} + 0.71123459 \cdot \begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} + 0.09625514 \cdot \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix} + 0.09625514 \cdot \begin{pmatrix} 0 \\ 2 \\ 1 \end{pmatrix}$$

$$= \begin{pmatrix} 0.80748973 \\ 1.80748973 \\ 0.38502054 \end{pmatrix}$$

3 Recap: Backpropagation

Given the network depicted below, perform backpropagation to get the gradients for all weight matrices!

The input is the *row vector* $x = [1, 0.5, 0.75, 0.25]$, the target output is $t = 0$.

The initial weights are

$$W_1 = \begin{pmatrix} 0.1 & 0.2 & 0.6 & 0.9 \\ 0.7 & 0.5 & 0.3 & 0.6 \\ 0.5 & 0.75 & 0.6 & 0.3 \\ 1. & 0.1 & 0.1 & 0.2 \end{pmatrix}$$

$$W_2 = \begin{pmatrix} 0.6 & 0.6 & 0.6 & 0.2 \\ 1. & 0.3 & 0.1 & 0.1 \\ 0.2 & 0.5 & 0.1 & 0.7 \\ 0.75 & 0.3 & 0.5 & 0.9 \end{pmatrix}$$

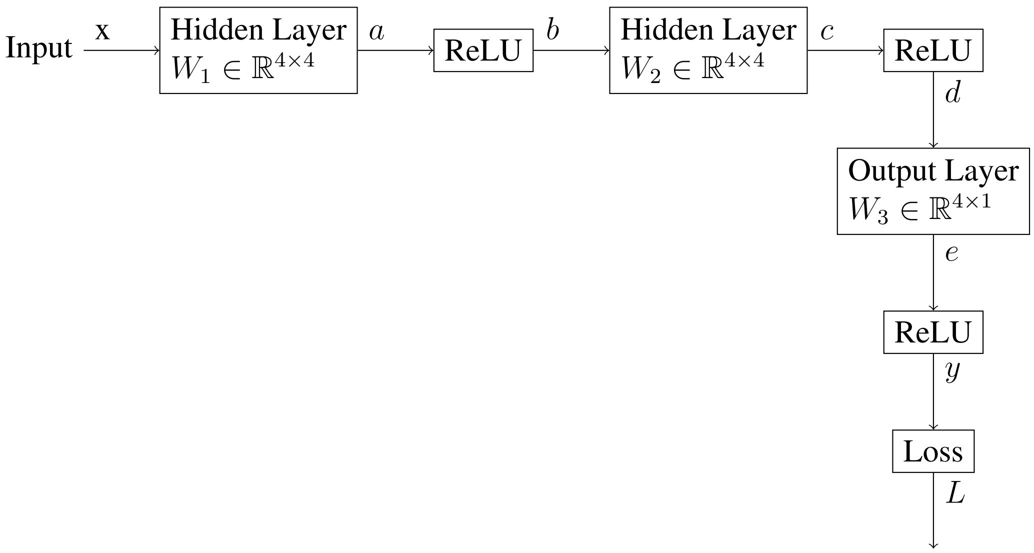

Figure 1: Neural Network with two hidden layers

$$W_3 = \begin{pmatrix} 0.6 \\ 0.7 \\ 0.2 \\ 0.9 \end{pmatrix}$$

No bias is used in the network. As loss function, we use the squared error, $L = \frac{1}{2}(t-y)^2$, where t is the true label.

Use the vectorised version of backpropagation introduced in the lecture.

> First the forward pass:
>
> $$a = xW_1 = \begin{pmatrix} 1.075 & 1.0375 & 1.225 & 1.475 \end{pmatrix}$$
>
> Since all these values are > 0, the ReLU does "nothing" and
>
> $$b = a.$$
>
> $$c = bW_2 = \begin{pmatrix} 3.03375 & 2.01125 & 1.00875 & 2.50375 \end{pmatrix}$$

Again, the ReLU does nothing:

$$d = c$$
$$e = dW_3 = \begin{pmatrix} 5.80325 \end{pmatrix}$$
$$y = e$$

Now for the backward pass, starting at the derivative of the loss by the loss, which is simple:

$$\frac{\partial L}{\partial L} = 1$$

For the loss function, we already know that $l'(y, t) = y - t$:

$$\frac{\partial L}{\partial y} = y - t = 5.80325$$

Since the input to the ReLU is > 0, its local gradient is 1;

$$\frac{\partial L}{\partial e} = \frac{\partial y}{\partial e}\frac{\partial L}{\partial y} = 1 \cdot 5.80325$$

For the next step, we use the formulas from the lecture:

$$\frac{\partial L}{\partial d} = \frac{\partial L}{\partial e} \cdot W_3^T = \begin{pmatrix} 3.48195 & 4.062275 & 1.16065 & 5.22925 \end{pmatrix}$$

$$\frac{\partial L}{\partial W_3} = d^T \frac{\partial L}{\partial e} = \begin{pmatrix} 17.6056 \\ 11.6718 \\ 9.33598 \\ 14.5299 \end{pmatrix}$$

This repeats for the next layers:

$$\frac{\partial L}{\partial c} = \frac{\partial L}{\partial d}\frac{\partial d}{\partial c} = \frac{\partial L}{\partial d}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial c} \cdot W_2^T = \begin{pmatrix} 6.26751 & 5.33899 & 6.49964 & 9.1111025 \end{pmatrix}$$

$$\frac{\partial L}{\partial W_2} = b^T \frac{\partial L}{\partial c} = \begin{pmatrix} 3.74309625 & 4.36694562 & 1.24769875 & 5.61464438 \\ 3.61252313 & 4.21461031 & 1.20417438 & 5.41878469 \\ 4.26538875 & 4.97628687 & 1.42179625 & 6.39808313 \\ 5.13587625 & 5.99185562 & 1.71195875 & 7.70381438 \end{pmatrix}$$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial b} \cdot 1$$

$$\frac{\partial L}{\partial W_1} = x^T \frac{\partial L}{\partial b} = \begin{pmatrix} 6.26751 & 5.33899 & 6.49964 & 9.1111025 \\ 3.133755 & 2.669495 & 3.24982 & 4.55555125 \\ 4.7006325 & 4.0042425 & 4.87473 & 6.83332688 \\ 1.5668775 & 1.3347475 & 1.62491 & 2.27777563 \end{pmatrix}$$

We do not need to calculate $\frac{\partial L}{\partial x}$, since the gradient for the input is not necessary for training.