EDAN20 Language Technology
-Lab 6.

Nils Romanus

September 2022

1 Character Level Recurrent Translation

Using a recurrent, LTSM-based, encoder-decoder architecture one may create a basic program for translating text.

1.1 Vectorizing text

The input text that is to be translated can be vectorized by performing a sequence of operations. First, the text is split in to characters. Secondly, every character is given an index, i.e a number corresponding to that character. Using this index one may one-hot-encode the input text, thus creating a matrix(tensor) representation of it. This tensor has three dimensions. One for wich piece of text we are looking at, i,e the batch dimension, one for where in the sequence a certain character occurs and finally one corresponding to the character index created before. The values in the matrix are 0 except for at the place corresponding to that index that are 1, creating a very sparse matrix representation. Since the sequences are of different length they are padded to the maximum sequence length in order to fit the dimensions of the model. The padding character is therefore also given an index in order to fit the one-hot-encoding scheme.

1.2 A Stacked LSTM Encoder

In the initial part of the lab the encoder module consisted of one LSTM, in the second part another one was stacked on top of it. Stacking, in this context, refers to the output sequence from the first LSTM beeing fed to the second LSTM in the encoder. The hidden states of the final LSTM in the encoder is subsequently fed to the decoder. A visual representation of the architecture is displayed in figure 1.

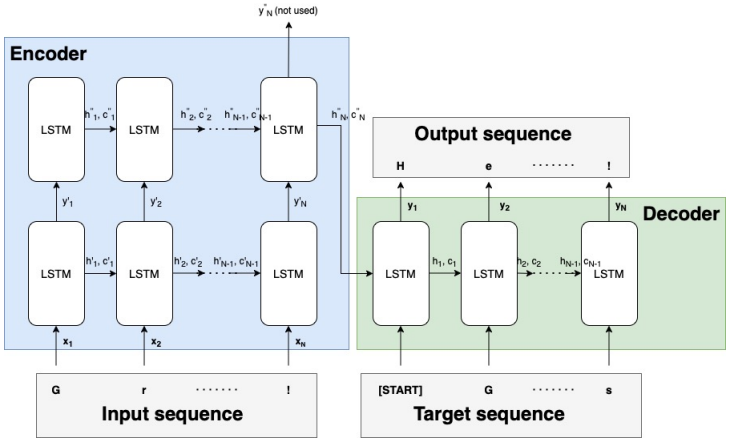

Figure 1: Stacked LSTM Architecture

1.3 Results Using Different Architectures

For the non-stacked architecture, a couple of different architectures were tried out. Initially, no drop-out-layers were used and secondly two configurations with different drop-out parameters. The performance, in terms of validation-loss, and accuracy, are displayed in figure 2.

Method	Parameters	Val Accuracy	Val Loss
LSTM	no dropout	0.8747	0.4510
LSTM	dropout=0.5, recurrent_dropout=0.2	0.8376	0.5659
LSTM	dropout=0.5, recurrent_dropout=0.5	0.8378	0.5717

Figure 2: Non-stacked architecture performance

For the stacked architecture a validation loss of 0.6047, and a validation accuracy of 0.8268, were achieved. The learning curve for the stacked version is displayed in figure 3. A sample of translated sentences, using the stacked architecture, is displayed in figure 4.

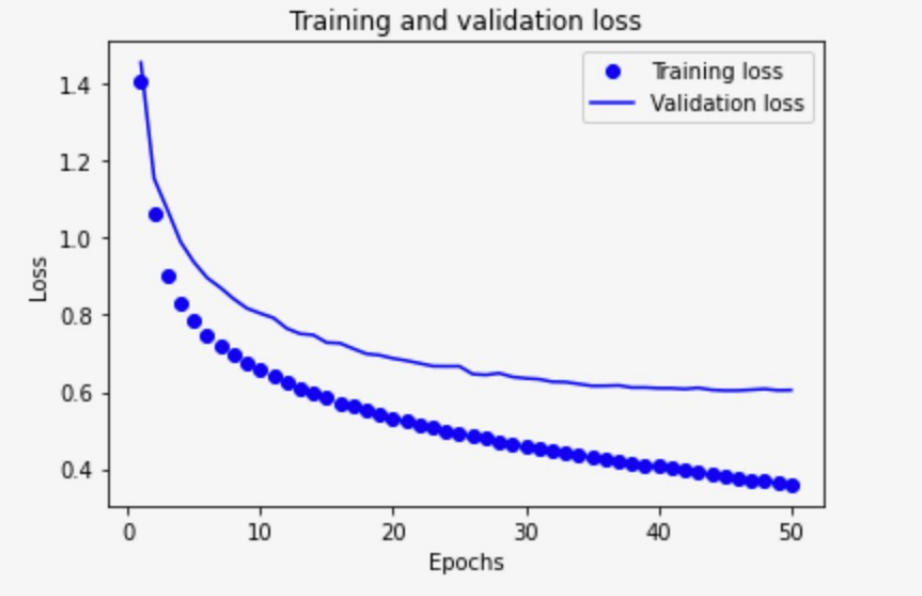

Figure 3: Learning curve for the stacked architecture

```
-
Input sentence: I ran.
Decoded sentence: Jag sprang.

-
Input sentence: I see.
Decoded sentence: Jag ser mig.

-
Input sentence: I see.
Decoded sentence: Jag ser mig.

-
Input sentence: I won!
Decoded sentence: Jag vann ha den.

-
Input sentence: Relax.
Decoded sentence: Slupp av mig.

-
Input sentence: Smile.
Decoded sentence: Le.

-
Input sentence: Attack!
Decoded sentence: Anväll mig!
```

Figure 4: Learning curve for the stacked architecture

2 Transformes and LSTMs in Machine Translation

The transformer architecture firstly introduced in the *Attention is all you need* paper differs from the recurrent LSTM-based encoder-decoder architecture. Firstly, a transformer is not recurrent and hence does not compute matrices in a sequence. This is advantageous as it favors parallelisation in the compute-heavy training of the machine, which is beneficial since it can be implemented using software that is optimized for parallel matrix multiplications on multiple GPUs. The name of the paper refers to an important concept in the transformer architecture, attention. Mask-less self-attention in the encoder allows the model to create representatons of the semantic similarity between elements of the input sequence, regardless of were in the input sequence these elements are located. The inability of an RNN or LTSM to capture relationships between elements that are 'far away' in the input sequence is one of the main draw back of recurrent architectures and this might be one of the reasons why the transformer is now state-of-the-art. Another difference between the transformer based architecture and our LSTM-based one is that the transformer makes use of other types of modules. This includes regularization techniques such. as normalization layers, dropouts and label smoothing. It also uses residual connections and positional encodings which are not present in our stacked LSTM architecture.

During training the encoder part of the network can not be given information about the ground-truth future tokens as this defeats the purpose of training the network. This since we with any autoregressive model aim to compute the joint probability $P(x_i|x_{j<i};\theta)$ for a sequence $x_{1\ldots n}$. In order to hide the x_j where

$j \geq i$ we use masking. In practice this is done by adding a triangular matrix filled with negative infinity in between the matrices in the attention module. The softmax in the attention model will then kill these entries masking the GT information. This is not needed in a recurring architecture since the RNNs or LTSMs are only fed information from past entries in the sequence.