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

19./20.06.2024

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

Summer Term 2024

1 General Questions

1. What are the differences between a recurrent neural network that processes inputs of length $l = 3$ and a multilayer perceptron (fully-connected network) that has three layers?

a) Each layer in the MLP has an independent weight matrix. In contrast, the RNN shares its weight matrices across the three timesteps.

b) The RNN can process sequences of inputs, thus explicitly modelling the input order, while the MLP can only take one input vector, e.g.the input is concatenated. This models the order only implicitly.

c) ...

❓ Something to think about

2. Two problems in Machine Translation that still remain with biRNN Seq2Seq models are a) unknown words (e.g., names) and b) forgetting/duplicating parts of the input. Think about possible solutions to these problems!

a) Usually, an Encoder-Decoder architecture works roughly like this: The encoder reads the entire input and generates an internal representation. The decoder then starts to generate the output from this internal representation by sampling from its output vocabulary. If a word in the input sentence is not in the output vocabulary, the decoder has no chance to predict it. A possible solution to this is a Copy mechanism, as introduced by Gu et al. (`https://arxiv.org/abs/1603.06393`): At each prediction step, give the decoder the choice to either sample a word from

the output vocabulary or copy a part from the input. That way the decoder only needs to identify the position where the unknown word must be inserted.

Another possibility would be to use character level models which can also predict unknown words (but could still suffer from problems with unknown character, for example in foreign names).

A new approach is to use subword tokens, so characters that occur together often are joined. A new word might then be represented as a combination of multiple subwords, which can be translated separately. You will learn about this strategy in the lecture.

b) forgetting/duplicating parts of the input. Sometimes Seq2Seq networks tend to either forget parts of the input or insert them into the output multiple times. Tu et al. (`https://arxiv.org/abs/1601.04811`) propose the Coverage mechanism to solve this problem. Coverage is basically "cumulative Attention": The network keeps track of how much attention has been paid to each input token and can therefore recognise which words it has yet to translate.

? Something to think about

3. Propose a neural network architecture that would be suited for generating image descriptions, that is, given an image as input, creates a sentence describing the content of the image! Provide reasoning why your architecture is a good choice for this task.

It would be reasonable to use a combination of CNNs and RNNs, for example designing an Encoder-Decoder architecture with a CNN as encoder and an RNN as decoder. The CNN generates a representation of the input image, which is then used as initialisation for the RNN.
This architecture makes sense as CNNs are known to be very good at creating image representations. RNNs are a natural choice for sentence generation, due to their ability to create sequential data.

For example, there is, as usually, a paper by Andrej Karpathy that does just that. See `https://cs.stanford.edu/people/karpathy/deepimagesent/`.

2 Beam Search

In the lecture, you learned that Beam Search can sometimes lead to better translations than simple Greedy Search, where the decoder always chooses the locally most likely token.

To show that this can indeed happen, look at the following situation: Given an Encoder-Decoder network trained to translate from English to German. The encoder has read the sentence "I won't tell you nothing!" and produced an internal representation h that encodes this sentence. This representation is now fed to the decoder and a translation is generated (see Figure **??**). The decoder will return the following probabilities for output tokens:

$$P(\text{Ich}|\text{<s>}) = 0.9$$
$$P(\text{Er}|\text{<s>}) = 0.01$$
$$P(\text{verrate}|\text{<s>,Ich}) = 0.3$$
$$P(\text{verrate}|\text{<s>,Er}) = 0.05$$
$$P(\text{sage}|\text{<s>,Ich}) = 0.5$$
$$P(\text{sage}|\text{<s>,Er}) = 0.08$$
$$P(\text{euch}|...,\text{verrate}) = P(\text{euch}|...,\text{sage}) = 0.3$$
$$P(\text{dir}|...,\text{verrate}) = P(\text{dir}|...,\text{sage}) = 0.45$$
$$P(\text{nichts}|...,\text{euch}) = P(\text{nichts}|...,\text{dir}) = 0.35$$
$$P(\text{nicht}|...,\text{euch}) = P(\text{nicht}|...,\text{dir}) = 0.4$$
$$P(!|...,\text{nichts}) = 0.8$$
$$P(!|...,\text{nicht}) = 0.1$$
$$P(\text{nichts}|...,\text{nichts}) = 0.01$$
$$P(\text{nichts}|...,\text{nicht}) = 0.2$$

1. Perform a greedy search to get the output of the decoder network!
2. Perform a beam search with $B = 2$ to get the output of the decoder network!
3. Compare the two outputs.

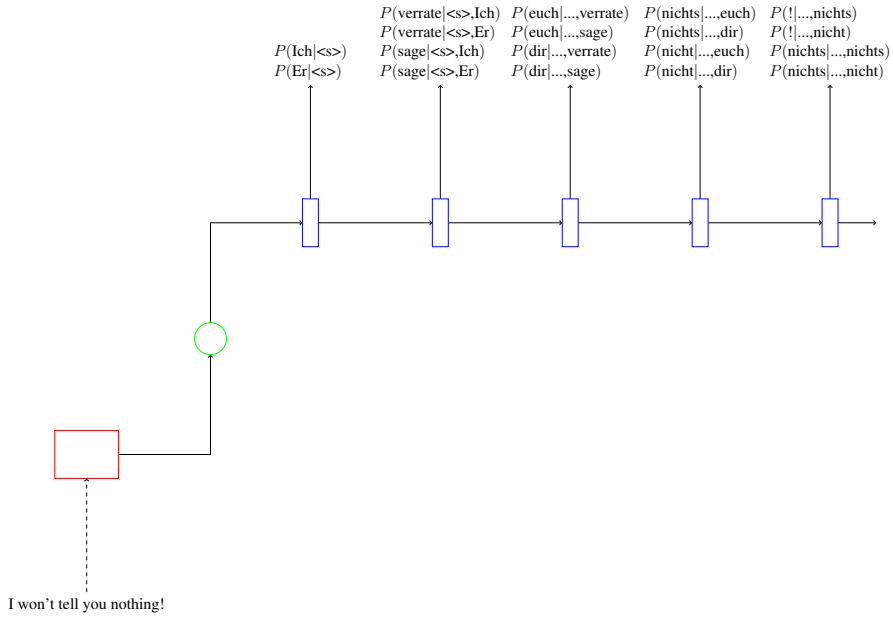

Figure 1: Visualisation of the Encoder-Decoder network used in this task

We can see that greedy search would produce “Ich sage dir nicht nichts”, while beam search finds the better and globally more likely translation “Ich sage dir nichts!”.

3 Python

3.1 Implementing Neural Networks Part 5 — Embeddings

In this assignment, you will implement a neural network "library" yourself, using `python` and `numpy`.

This week, you will use your implementation to train your own word embeddings on a small corpus.

For this, download the dataset of English sentences from `https://data.deepai.org/text8.zip` and extract the .txt file.

1. Extract training samples from the given corpus: Slide a context window over the corpus, creating tuples of the form $\big(\underbrace{w_{i+2}}_{\text{center word}}, \underbrace{(w_i, w_{i+1}, w_{i+3}, w_{i+4})}_{\text{context words}} \big)$
2. Implement a simplified version of the skip-gram model: Your model should have the following layers:
   - Input: a 1-hot vector representing the center word of each sample
   - Hidden Layer: a layer of size 32
   - Output: a layer of size $|V|$, where V is the vocabulary of the corpus
3. Train the network on your samples: Use the center word as the input and train the network to predict the "4-hot" vector of the context words, that is, a vector that is zero everywhere except at the positions corresponding to the output words. (Note that this is not how the original Word2Vec is calculated. We change the procedure as you have not implemented the Cross Entropy loss function.)
4. After training, where will the embeddings be found?

Feel free to try different optimisers, activation functions, layer sizes, maybe apply Dropout, and look into the resulting word embeddings!