NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING

Speech and Natural Language Processing
Spring Semester 2022-23

3rd Laboratory Exercise

1 Introduction

In the lab preparation, you designed simple architectures to classify texts, using pretrained word embeddings. Specifically, the architecture that you were asked to make, as depicted in Figure 1, is the following:

$$e_i = embed(x_i) \qquad \text{embedding of each word} \tag{1}$$

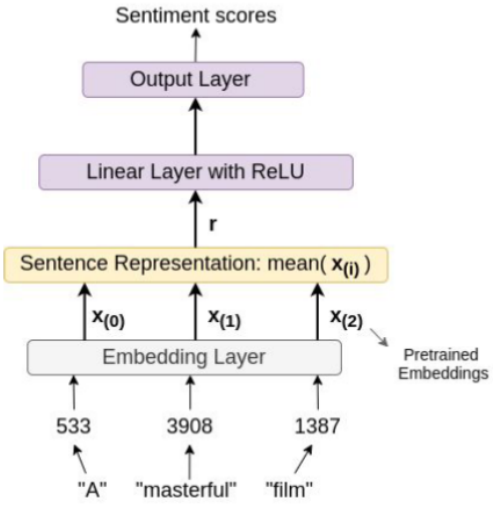
$$u = \frac{1}{N}\sum_{i=1}^{N} e_i \qquad \text{representation of text: Average of embeddings} \tag{2}$$

$$r = ReLU(Wu + b) \qquad \text{non-linear transformation} \tag{3}$$

where r is the vector representation of a text (feature vector).
The purpose of this laboratory exercise is to create better representations by:

1. concatenating the mean-pooling and the max-pooling representations (over embedding dimensions) of the word embeddings for each sentence
2. using Recurrent Neural Networks and specifically LSTMs (Long Short-Term Memory networks)
3. applying a Self-Attention mechanism on the word embeddings
4. applying a MultiHead-Attention mechanism on the word embeddings
5. developing and applying a Transformer-Encoder on the word embeddings
6. using Pre-Trained Transformers to perform sentiment classification
7. fine-tuning Pre-Trained Transformers to perform sentiment classification

Figure 1: Preparation-lab architecture



2 Theoretical Background

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs), in contrast to other architectures, have the capacity of forming connections with feedback. Most networks require fixed dimension inputs while RNNs can easily process variable length data.

This flexibility makes them ideal for processing sequences which arise in natural language processing problems. The way they work resembles how humans process language (text, speech), i.e. serially.

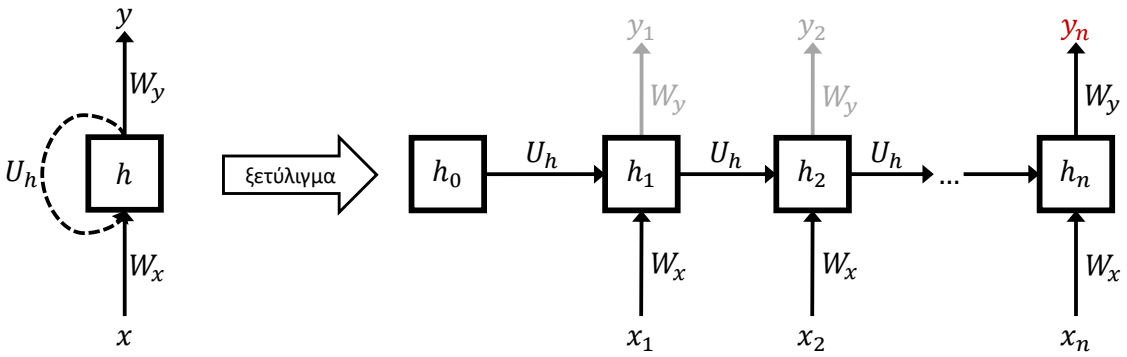


Figure 2: Operation Diagram of an RNN. There are two ways to think about how an RNN works. In the left figure the recurrent operation of an RNN is shown. The RNN accepts one after another the elements of the sequence and updates its inner or hidden state. Another way of representation is with the "unrolling" of the network with respect to time. Essentially the RNN is a feed-forward NN (FFNN) with feedback. In the unfolded form, the RNN looks like a multilevel FFNN.

The basic functionality of an RNN is as follows: it accepts as input a sequence of vectors $x = (x_1, x_2, ..., x_n)$ and it produces a unique vector y as the output. However, the critical difference is that at each timestep, to produce the result, it takes into account the result of the previous step. The simple RNN, which we are considering now, has certain variants [Hopfield, 1982, Elman, 1990, Jordan, 1997]. The variant we are considering here is the Elman [Elman, 1990] network.

Figure 2 clarifies the way of operation of an RNN. As shown by the figure, the RNN can accept a sequence of any length. After processing the items one by one, it produces the end result y_n, which is a single *constant vector representation* for the whole sequence.

Example Let's look at an ordinary example of using an RNN to process natural language. In a text classification problem, the RNN processes the words of the document, one after another, and in the end it produces the *vector representation of the document* y_n. This representation is used as feature vector to classify the text, e.g. based on the emotional content.

More specifically, the document is represented by a word sequence. Each word is represented by a vector (word embedding) x_i, with $x_i \in R^E$, where E the dimensions of the word vectors. Therefore we have the sequence $X = (x_1, x_2, ..., x_T)$, where T is the count of words in the document. The RNN processes the words serially, keeping internally, a summary of what has been read up to time t. At the end, it contains a summary of all document information which is the final vector representation for the document.

2.1.1 Bidirectional RNN

A bidirectional RNN (BiRNN) consists of a combination of two different RNNs, where each one processes the sequence in a different direction. The motivation of this technique, is the creation of a document's summary from both directions, to form a better representation. Thus we have a clockwise RNN $\overrightarrow{f}$, which reads a sentence from x_1 to x_T and a counterclockwise RNN $\overleftarrow{f}$, which reads a sentence from x_T to x_1. Therefore, at every time t, we have:

$$h_i = \overrightarrow{h_i} \parallel \overleftarrow{h_i}, \quad h_i \in R^{2N} \tag{4}$$

where $\parallel$ denotes the act of concatenating two vectors and N are the dimensions of each RNN.

2.1.2 Deep RNNs

As with the simple FFNN we can stack an RNN for many layers to create deep networks.

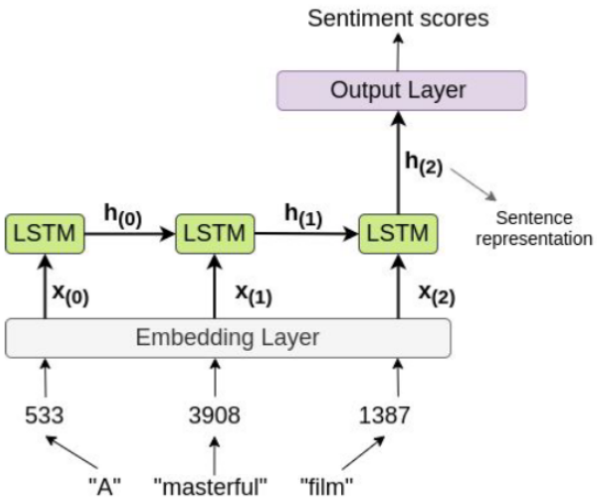
2.1.3 Long Short-Term Memory (LSTM)

The simple RNN is no longer used. However its variations are used which attempt to overcome certain problems, with the most important being that of vanishing gradients during network training [Bengio et al., 1994, Hochreiter et al., 2001]. The most popular variant is the Long Short-Term Memory Network (LSTM) [Hochreiter and Schmidhuber, 1997].

It uses a sophisticated mechanism, that allows it to overcome the limitations of the RNN, regarding the recognition of long dependencies. The LSTM has two crucial differences from the simple RNN:

- It *does not* apply an activation function to recurrent connections. This means that updates will be linear. This guarantees that errors (gradients), will not disappear from the repetitive application of updates (backpropagation through-time). This ensures the flow of information to the network.
- Gating mechanism. This mechanism introduces gates, which regulate how much each network vector will be updated (internal state, output etc.). This way the network assimilates and maintains the most important information in a better way.

Figure 3: Total architecture that utilizes an LSTM



More information with regards to LSTM functionality can be found here[1]. The whole architecture that utilizes LSTM to perform sentiment classification is presented in Figure 3.

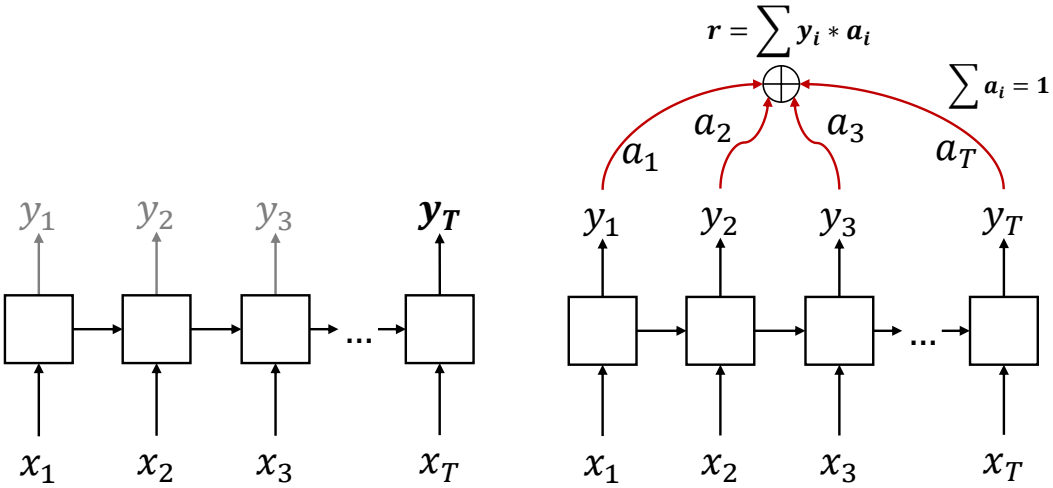
2.2 Attention mechanism

Attention mechanism revolutionized Natural Language Processing field, because of its ability to identify the information in an input which is the most to accomplish a task. It was introduced by [Bahdanau et al., 2014], authors of the paper "Neural Machine Translation by Jointly Learning to Align and Translate", as a technique that allowed the decoder to focus on the appropriate words at each time step. By utilizing an RNN as a sequence encoder, it shortened the path between two distant but similar words with the mechanism that is depicted in Figure 4.

2.3 Transformer

In a groundbreaking paper, [Vaswani et al., 2017] suggested that "Attention Is All You Need". They managed to create an architecture called the Transformer, which significantly improved the state of the art in Neural Machine Translation without using any recurrent or convolutional layers, just attention mechanisms (plus some other pieces like normalization layers etc.). This architecture can be seen in Figure 5 along with some notes with regards to its modules. More details for the operation of a Transformer can be found here[2].

[1] https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[2] https://jalammar.github.io/illustrated-transformer/



(a) Simple RNN. The final representation of the sequence is the last output of the RNN y_n.

(b) RNN with attention mechanism. The final representation of the sequence is the weighted sum of intermediate outputs y_i.

Figure 4: Comparison between a simple RNN and an RNN with attention. In the RNN with attention, the final representation r is the weighted sum of all the outputs of the RNN. The weights of each step a_i, are defined by the attention layer.

A detailed tutorial by Andrej Karpathy, named "Let's build GPT: from scratch, in code, spelled out", can be found here[3], where several concepts about Transformers are clarified. Picking out some notes by A. Karpathy from the above:

- Attention is a communication mechanism. Can be seen as nodes in a directed graph looking at each other and aggregating information with a weighted sum from all nodes that point to them, with data-dependent weights.
- There is no notion of space. Attention simply acts over a set of vectors. This is why we need to positionally encoded tokens.
- "self-attention" just means that the keys and values are produced from the same source as queries. In "cross-attention", the queries still get produced from x, but the keys and values come from some other, external source (e.g. an encoder module)

2.4 Pre-Trained Transformers

In recent years, several pre-trained models have become available on platforms such as HuggingFace[4]. By properly utilizing packages like the transformers[5], one can use thousands of pretrained models to perform tasks on different modalities such as text, vision, and audio.

These models can be applied on:

- Text, for tasks like text classification, information extraction, question answering, summarization, translation, text generation, in over 100 languages.
- Images, for tasks like image classification, object detection, and segmentation.
- Audio, for tasks like speech recognition and audio classification.

[3] https://youtu.be/kCc8FmEb1nY
[4] https://huggingface.co/
[5] https://github.com/huggingface/transformers

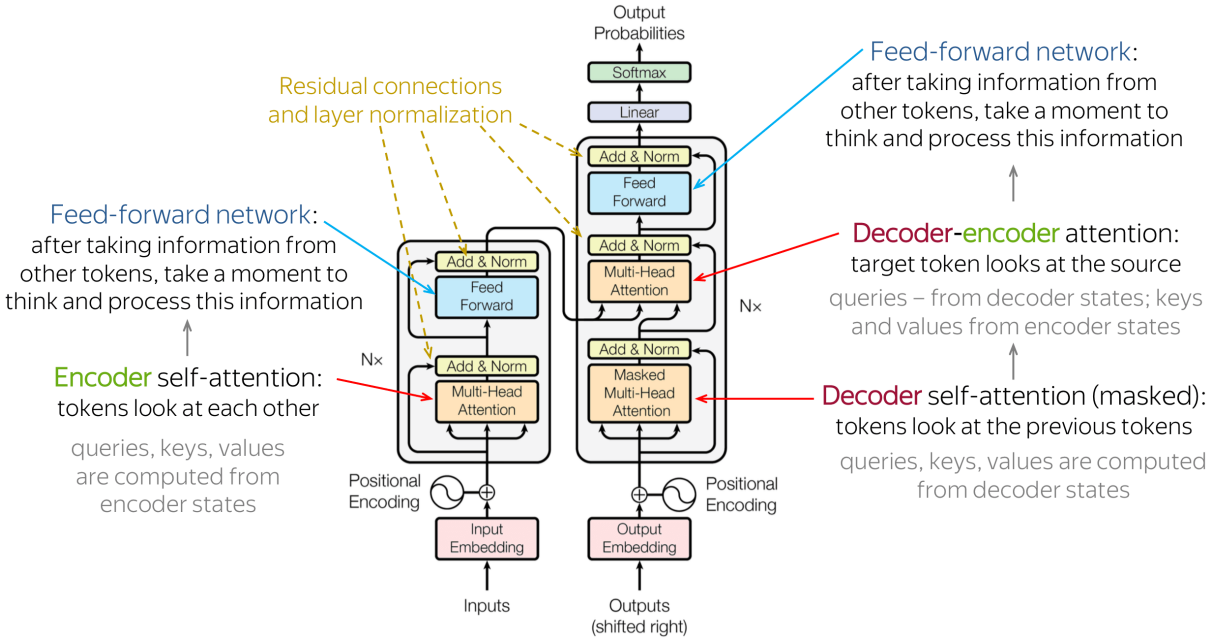


Figure 5: Transformer architecture

3 Questions

For your implementation, rely on the available code that can be found here[6].

Question 1

1.1 Calculate the representation of each sentence u (Equation 2) as the concatenation of the average (mean pooling) and the maximum per dimension (max pooling) of the word embeddings of each sentence, $E = (e_1, e_2, ..., e_N)$.

$$u = [mean(E)||max(E)] \tag{5}$$

1.2 What difference does this representation have with the original? What more information could be extracted? Answer briefly.

Question 2

On this question you should use an LSTM to encode the sentence. The LSTM will read the word embeddings e_i and it will produce a new representation for every word h_i, which will take into account the context as well. You can skip the non-linear transformation (Equation 3).

2.1 At first, use the function `training.torch_train_val_split()` to create a validation set from the training set. By using the class `early_stopper.EarlyStopper`, stop the training when the validation loss is constantly increasing for 5 epochs.
2.2 Use the last output of the LSTM h_N as the representation of the text u.
Caution: You must use the real last timestep, excluding the zero-padded timesteps. Use the actual sentence lengths, just as you calculated the average in the exercise of the lab preparation.
2.3 Using the `bidirectional` parameter in the class constructor, re-run your experiments using bidirectional LSTM. Report the performance of the model on the test set (accuracy, recall, f1-score).

Question 3

On this question you will use an attention mechanism for sentiment classification.

[6] https://github.com/slp-ntua/slp-labs/tree/master/lab3

3.1 Utilize the given model `attention.SimpleSelfAttentionModel`, fill in the blanks, and measure its performance on the test set. Apply average-pooling to the word representations before the last layer, to derive the sentence representation. What is the performance of the model?
3.2 What are the queries, keys and values that exist in the class `attention.Head` and the `position_embeddings` defined in `attention.SimpleSelfAttentionModel`? You can refer to the "Let's build GPT: from scratch, in code, spelled out" tutorial for your answer.

Question 4
On this question you will use a MultiHead-attention mechanism for sentiment classification.
Fill in the blanks in the `attention.MultiHeadAttentionModel` model. You should rely on `SimpleSelfAttentionModel` but make use of `MultiHeadAttention` for the attention mechanism. What is the performance of the model?

Question 5
On this question you will use a Transformer-Encoder for sentiment classification. Keep the average-pooling method to extract the representation for each sentence.
Fill in the blanks in the `attention.TransformerEncoderModel` model. How is it different from `MultiHeadAttentionModel`? Experiment with different hyper-parameter values. What are their default values in the classic Transformer architecture? How does the model perform on the test set?

Question 6
On this question you will use Pre-Trained Transformer models for sentiment classification. Complete the file `transfer_pretrained.py` and run it in order to evaluate the pre-trained models. Choose at least 3 models, from here[7], for each dataset and extend the code accordingly. Compare their performance and report the results in tables.

Question 7
On this question you will train/fine-tune Pre-Trained Transformer models for sentiment classification. Use the code that can be found in the file `finetune_pretrained.py`. Run it locally with a limited number of samples for a few seasons (as given). Transfer the code to a notebook in Google Colab[8], convert it appropriately and fine-tune it for optimal performance. Experiment with at least 3 models for each dataset and report the results in tables.

Question 8 (bonus)
The code for the aforementioned "Let's build GPT: from scratch, in code, spelled out" tutorial is available here[9]. Login to ChatGPT[10] and request:

- to explain the code to you in detail
- to evaluate the code
- to rewrite some parts of it (refactoring)

Report examples from the dialogue and comment on its performance in the above tasks.

[7] https://huggingface.co/models
[8] https://colab.research.google.com/
[9] https://colab.research.google.com/drive/1JMLa53HDuA-i7ZBmqV7ZnA3c_fvtXnx-?usp=sharing
[10] https://chat.openai.com/

Deliverables

For questions 1-5 you can use one of the two datasets which are referred to the lab preparation. Choose whichever you want. Just mention it on your report. For every variant of the model, report the performance of the model for: accuracy, F1_score (macro average), recall (macro average).

In terms of grading, you will not be evaluated with respect to your model's performance, but with respect to the correctness of your answers. You are free to choose the values you want for the hyperparameters of the model. You should deliver the following:

1. Brief report (in pdf) that will contain the answers to each question and the corresponding results.

2. Python Code, accompanied by brief comments.

Gather (1) and (2) in a .zip file and submit it before the due date at helios.

References

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*. 02127.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Elman, 1990] Elman, J. L. (1990). Finding Structure in Time. *Cognitive Science*, 14(2):179–211. 08621.

[Hochreiter et al., 2001] Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Hopfield, 1982] Hopfield, J. J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558. 18706.

[Jordan, 1997] Jordan, M. I. (1997). Serial Order: A Parallel Distributed Processing Approach. In *Advances in Psychology*, volume 121, pages 471–495. Elsevier. 01033.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.