NATIONAL TECHNICAL UNIVERSITY OF ATHENS
FACULTY OF ELECTRICAL & COMPUTER ENGINEERING

Speech and Natural Language Processing
Spring Semester **2024-25**

2nd Laboratory Exercise

1 Introduction

In the preparation, you designed simple neural network architectures for text categorization using pre-trained vector word embeddings. In particular, the architecture you have been asked to build, as shown in Figure 1, is as follows:

$$e_i = embed(x_i) \quad \text{vector representation of each word (embedding)} \quad (1)$$

$$u = \frac{1}{N} \sum_{i=1}^{N} e_i \quad \text{text representation: average embeddings} \quad (2)$$

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

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

1. by pooling the representations obtained through mean pooling and max pooling of the word embeddings of each sentence
2. using Recurrent Neural Networks and in particular LSTMs (Long Short-Term Memory networks)
3. applying simple self-attention mechanism on the word embeddings
4. applying MultiHead-Attention mechanism on the word embed- dings
5. developing and implementing the Encoder part of a Transformer (Transformer- Encoder) on the word embeddings
6. using available pre-trained models (Pre-Trained Transformers) categorise emotion
7. fine-tuning pre-trained models for the categorisation of emotion

2 Theoretical Background

2.1 Feedback-driven Neural Networks

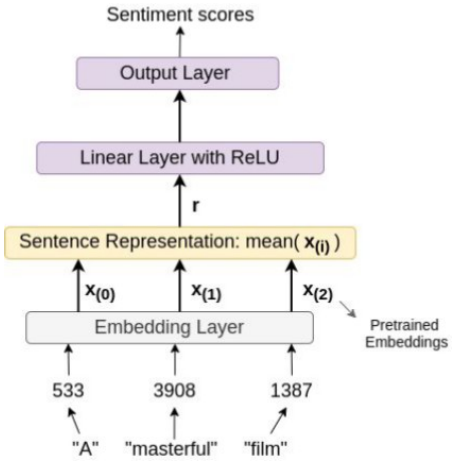
Recurrent Neural Networks (RNNs), unlike other architectures, have the property that they form connections with feedback. Most networks require fixed-length inputs, while RNNs can easily process variable-length data.

This flexibility makes them ideal for processing sequences, such as in natural language processing problems. Their mode of operation is similar to the way humans process language (text, speech), i.e. serially.

The basic operation of an RNN is as follows: it takes as input a sequence of vectors $x = (x_1, x_2, ..., x_n)$ and produces as output a unique vector y. However, the crucial difference is that at each step, the result of previous step is taken into account to produce the result. The simple RNN, which we now consider, has some variants [Hopfield, 1982; El- man, 1990; Jordan, 1997]. The variant which we consider here is the Elman network [Elman, 1990].

Figure 2 clarifies how an RNN works. As can be seen from the figure, an RNN can accept a sequence of any length. After processing the elements one by one, it produces the final result y_n, which is *a constant vector representation* for the whole sequence.

Figure 1: Overall Prefab architecture



Example Let's look at a common example of applying an RNN to natural language processing. In a text categorization problem, the RNN processes the words in the document, one by one, and at the end produces *the vector representation of the document* $y_{(n)}$. This representation is used as a feature vector categorize the text, e.g., based on its emotional orientation.

More specifically, the document is represented by a sequence of words. Each word is represented by a word embedding vector x_i, with $x_i \in R^E$, where E are the dimensions of the word vectors. Thus we have the sequence $X = (x_1, x_2, ..., x_T)$, where T is the number of words in the document. The RNN processes the words sequentially, keeping inside it, a summary of what it has read up to time t. At the end, it contains the summary of all information in the document and from this it forms the final vector representation for the document.

2.1.1 Bidirectional **RNN**

A bidirectional RNN (BiRNN) consists of a combination of two different RNNs, each processing the sequence in a different direction. The motivation of this technique is to create a summary of the document from both directions, so that to form a better representation. So 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. Thus, at each time t, we have:

$$h_i = \overrightarrow{h_{(i)}} \,||\, \overleftarrow{h_i}, \quad h_i \in R^{2N} \tag{4}$$

where $||$ represents the operation of joining two vectors and N are the dimensions of each vector. RNN.

2.1.2 Deep **RNN**

As with simple FFNNs we can stack an RNN into layers to create deep networks.

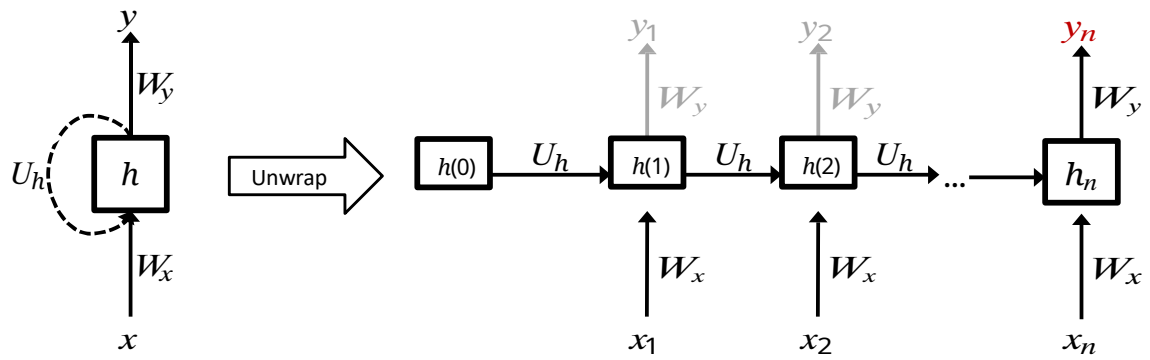


Figure 2: Functional diagram of an RNN. There are two ways to think about how an RNN works. The left figure shows the recursive operation of the RNN. The RNN receives the sequence elements one by one and updates its internal or hidden state. Another way of representing it is by "unrolling" the network in time. Essentially an RNN is a feed-forward NN (FFNN) with feedback. In the unfolded form, an RNN looks like a multilevel FFNN.

2.1.3 Long Short-Term Memory (LSTM)

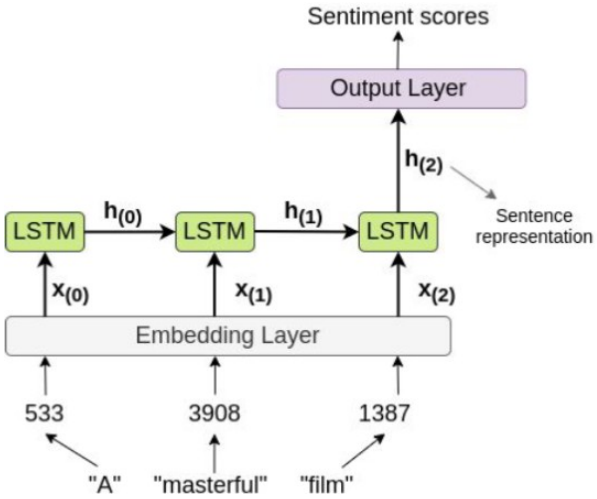
Nowadays, the simple RNN is hardly used at all but its variants, which attempt to overcome some problems, the most important one being the vanishing gradient during network training [Bengio et al., 1994, Hochreiter et al., 2001]. The most popular variant is the Long Short-Term Memory (LSTM) network [Hochreiter and Schmidhuber, 1997].

It uses a sophisticated mechanism,
o which allows it to overcome the RNN problem of identifying remote dependencies.

LSTM has two main differences from the simple RNN:

- *It does not* apply an activation function to recursive connections. This means that the updates will be linear. This guarantees that gradients will not be eliminated by the backpropagation through-time the updates. therefore ensures the flow of information in the network.
- Mechanism with doors. This mechanism introduces ports, which control how much each vector of the network is updated (internal state, external state, and the amount of information that is updated).

Figure 3: Overall architecture using LSTM



road, etc.). In this way, the network assimilates and retains the most important better.

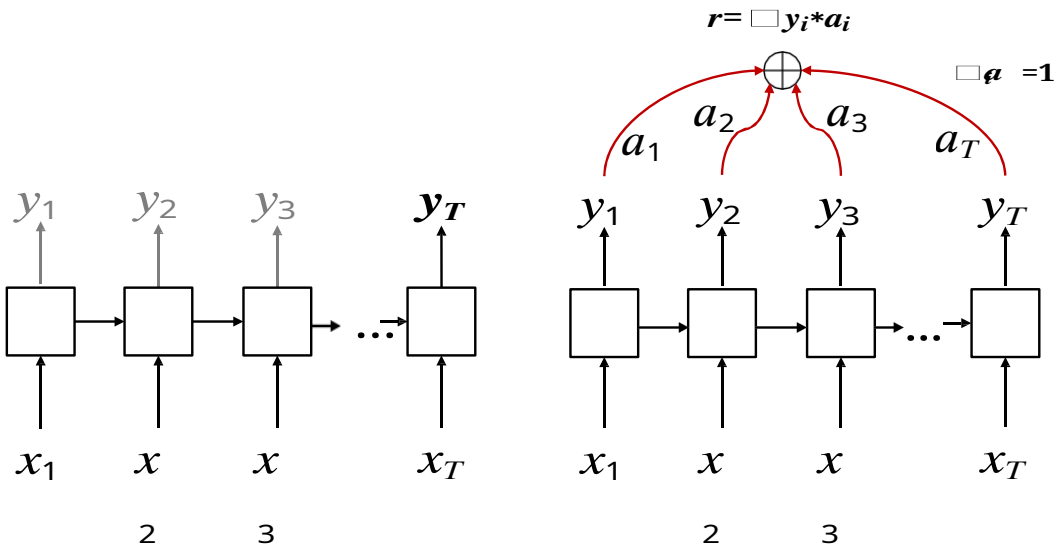
More information on the operation of the LSTM can be found here[1]. In Figure 3

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

presents the overall architecture that will result from using LSTM to categorize the synoptic.

2.2 Attention Mechanism (Attention)

The attention mechanism radically changed the field of natural language processing because of its ability to recognize information in an input that is most appropriate for accomplishing a task. It was practiced by [Bahdanau et al., 2014] in the article entitled "Neural Ma- chine Translation by Jointly Learning to Align and Translate" as a mechanism that allowed the decoder to focus on the most appropriate words at each step. By using an RNN to encode the steps of the sequence, he achieved the reduction of the distance between two "distant" but related words in the sequence, with the mechanism shown in Figure 4.



(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 the intermediate outputs y_i.

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

2.3 Transformer

In a pioneering article, the authors [Vaswani et al., 2017] suggested that "Attention Is All You Need". They managed to create an architecture they called Transformer, which greatly improved the results in text translation without using convolutional or recursive networks, but only attention mechanisms (plus some other elements like normalization layers etc.). This architecture is shown in Figure 5 along with some comments on its individual modules. More information about the operation of the Transformer can be found here[2].

A very detailed tutorial by Andrej Karpathy entitled "Let's build GPT: from scratch, in code, spelled out" can be watched here[3], in which many concepts about Transformers are clarified. Picking some of A. Karpathy's notes from the above:

[2] https://jalammar.github.io/illustrated-transformer/
[3] https://youtu.be/kCc8FmEb1nY

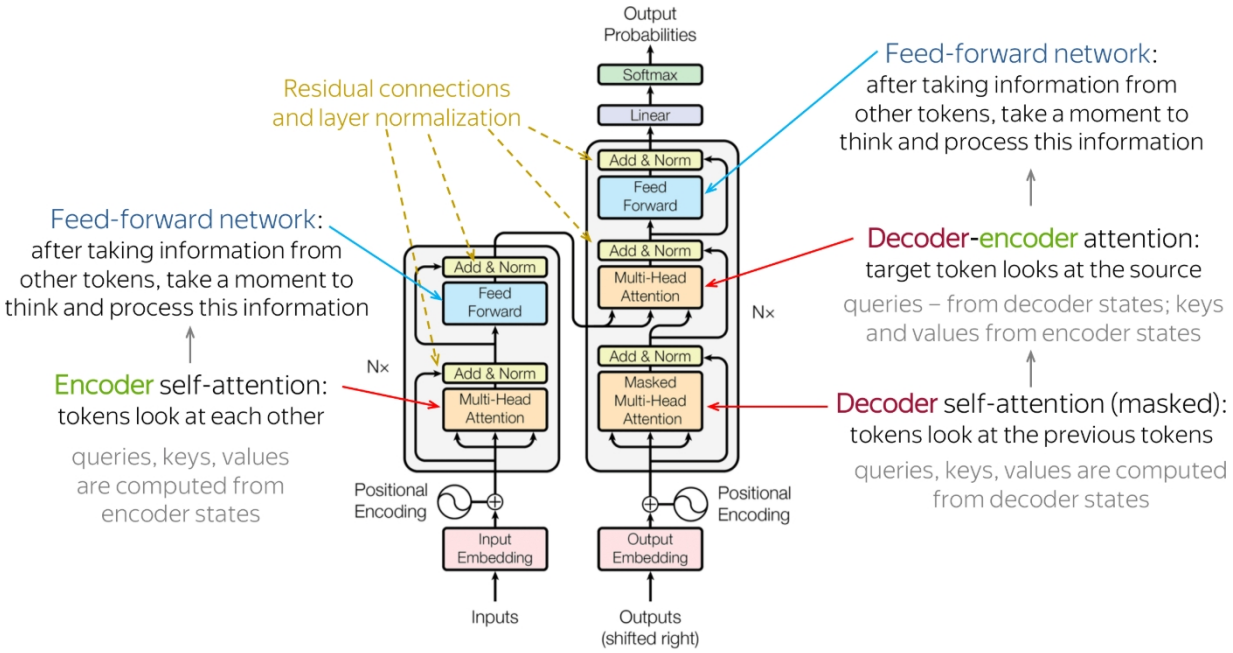


Figure 5: Transformer architecture

- the Attention mechanism is a communication mechanism. It can seen as nodes of a directed graph that are connected to each other and information is gathered by a weighted sum of all nodes pointing to each node with data-dependent weights.
- There is no concept of space. The attention mechanism simply acts on a set of vectors. This is why we add a position encoding.
- The self-attention mechanism simply means that the keys and values are generated from the same source as the queries. In the cross-attention mechanism, the queries are generated from one source but the keys and values come from another (e.g. an encoder).

2.4 Pre-Trained Transformers

In recent years, several pre-trained models have become available on platforms such as HuggingFace[4]. With proper use of packages such as transformers[5], one can use thousands of pre-trained models to perform tasks in text, image and sound.

Indicatively, the models can be applied to:

- Text, for tasks such as sorting text, extracting information, answering questions, summarising, translating, creating text, in more than 100 languages.
- Images, for tasks such as image classification, object detection and segmentation.
- Audio, for tasks such as speech recognition and audio classification.

[4] https://huggingface.co/

[5] https://github.com/huggingface/transformers

3 Questions

For your implementation, rely on the code available here[6].

Question 1

1.1 Compute the representation of each sentence u (Equation 2) as the concatenation of the mean pooling and 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(s) does this representation have to the original? What additional information could it extract? Answer briefly.

Question 2

In this query you should use an LSTM to encode the sentence. The LSTM will read the word embeddings e_i and produce a new representation for each word h_i, which will take context into account. You omit the non-linear transformation (Equation 3).

2.1 First, use the `training.torch_train_val_split()` function to create a validation set from the training set. Using the `early_stopper.EarlyStopper` class, stop training when the validation loss is continuously increasing for 5 epochs.
2.2 Use the last output of LSTM h_N as the representation of the text u.
Rely on the `models.LSTM` implementation provided.
Caution: you must use the actual last timestep, excluding zero-padded timesteps. Use the actual lengths of the sentences as you them in the prep exercise.
2.3 Using the `bidirectional` parameter in the class constructor, re-run the
your experiments using bidirectional LSTM. Report the performance of the model on the test set (accuracy, recall, f1-score).

Question 3

In this question you will use an attention mechanism to categorize emotion.
3.1 Utilize the given model `attention.SimpleSelfAttentionModel`, fill in the
gaps and measure its performance on the test set. Apply average-pooling to word representations before the last layer to extract the sentence representation. What is the performance of the model?
3.2 What are the queries, keys and values that exist in the `attention.Head` class and the `position_embeddings`
defined in `attention.SimpleSelfAttentionModel`; You can consult the tu- torial "Let's build GPT: from scratch, in code, spelled " for your answer.

Question 4

In this question you will use a MultiHead-attention mechanism for emotion categorization.
Fill in the blanks in the `attention.MultiHeadAttentionModel`. Rely on the `SimpleSelfAttentionModel` but use `MultiHeadAttention` for the attention mechanism.
What is the performance of the model?

Question 5

In this query you will use a Transformer-Encoder to categorize

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

emotion. Maintain average-pooling to extract the representation of each prediction.
Fill in the blanks in the `attention.TransformerEncoderModel`. What is its relation with respect to the `MultiHeadAttentionModel`? Experiment with different values of the parameters. What are their default values in the classic Transformer architecture?
What is the performance of the model in the test set?

Question **6**

In this question we will use Pre-Trained Transformer models for emotion categorization. Complete the file `transfer_pretrained.py` and run it to evaluate the pre-trained models. Select at least 3 models, from here[7], for each dataset and extend the code appropriately. Compare their performance and tabulate the results.

Question **7**

In this question you will fine-tune Pre-Trained Transformer models for emotion categorization. Leverage the code found in the `finetune_pretrained.py` file and run locally with a limited dataset for a few seasons (as given). Transfer the code to notebook in Google Colab[8], convert it appropriately and proceed to fine-tune it for optimal performance. Test at least 3 models for each dataset and report the results in tables.

Question **8**

The code for the tutorial "Let's build GPT: from scratch, in code, spelled out", mentioned above, is available here[9]. Log in to ChatGPT[10] and ask:

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

Cite examples of the dialogue you have given and comment on the performance of the above.

Deliverables

In questions 1-5 you use one of two datasets mentioned in the preparation. Choose whichever one you want, just mention it in your report. For each model, report in your report its performance on the metrics: accuracy, F1-score (macro average), recall (macro average).

In terms of grading the exercise, you will not be evaluated on the performance of your model, but on the correctness of your answers. You are free to choose whatever values you want for the hyperparameters of the model. You must provide the following:

1. A short report (in pdf format) containing the answers to each question and the corresponding results.
2. Python code, accompanied by short comments.

Assemble (1) and (2) in a .zip file and submit it to helios by the deadline.

[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/

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.

, Y., Simard, P., and Frasconi, P. (1994): Learning long-term depen- dencies 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, S. 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, L- ., and Polosukhin, I. (2017). attention is all you need. in *Advances in Neural Information Processing Systems*, pages 5998-6008.