NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL & COMPUTER ENGINEERING

Speech and Natural Language Processing
Spring Semester 2021-22

3rd Laboratory Exercise

1 Introduction

In the lab preparation, you designed simple architectures to classify texts, using pretrained word embeddings. The purpose of this laboratory exercise is to dive deeper using Recurrent Neural Networks, transfer learning and attention mechanisms.

In the lab preparation, the architecture that you were asked to make is the following:

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

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

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

where r is the vector representation of a text (feature vector). The goal is to create better representations using the aforementioned techniques.

2 Theoretical Background

2.1 Neural Networks

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.

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.

Operation More specifically the RNN processes the elements of a sequence serially (one by one), performing the *same* transformation, for each element of the sequence. Also, it maintains an internal state, which acts like a kind of memory, in the form of a vector h, which is called *hidden state*. At each step, the RNN updates h, taking into account the value of the current element x_t and the previous value of h. Thus for every time t, we have:

$$h_t = f_h(W_x x_t + U_h h_{t-1} + b_h) \tag{4}$$

$$y_t = f_y(W_y h_t + b_y) \tag{5}$$

where:

- h_t is the hidden state (or hidden vector), at time t.
- x_t is the vector (input) of the element of the sequence at time t.

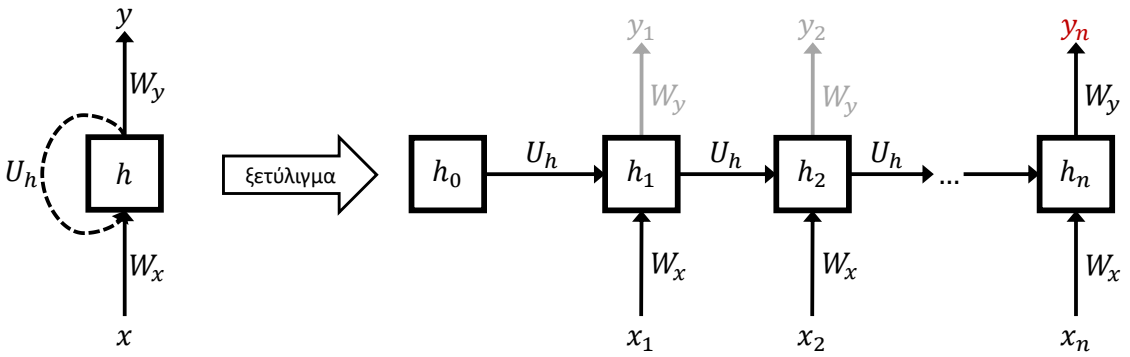


Figure 1: Operation Diagram of an RNN. There are two ways to think about how an RNN works. In the left figure the recurrent operation of an RNN is shown. The RNN accepts one after another the elements of the sequence and updates its inner or hidden state. Another way of representation is with the "unrolling" 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.

- y_t the output (vector) at time t.
- W_x, U_h, W_y the weights[1] of the network for x, h and y respectively.
- b_h the bias for h.
- f_x, f_h the activation functions (usually tanh).

Figure 1 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. You can think of this representation like the summary of the sequence. In this case, only the hidden state updates are performed (Equation (4)) and in the end the output is produced (Equation (5)).

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:

[1] Usually W is used to declare weights in the normal connections of a network and U is used to declare weights in recurrent connections.

$$h_i = \overrightarrow{h_i} \parallel \overleftarrow{h_i}, \quad h_i \in R^{2N} \tag{6}$$

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.

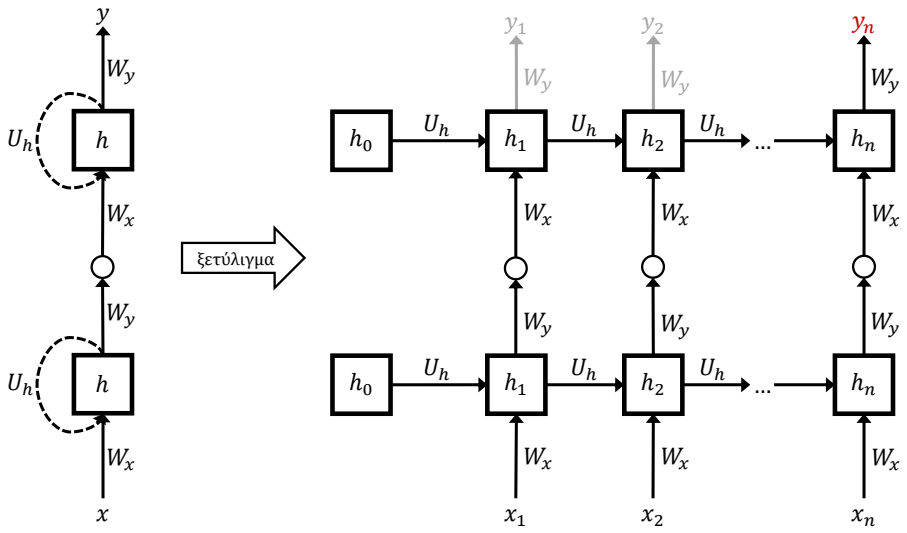


Figure 2: A Deep RNN with two layers.

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.

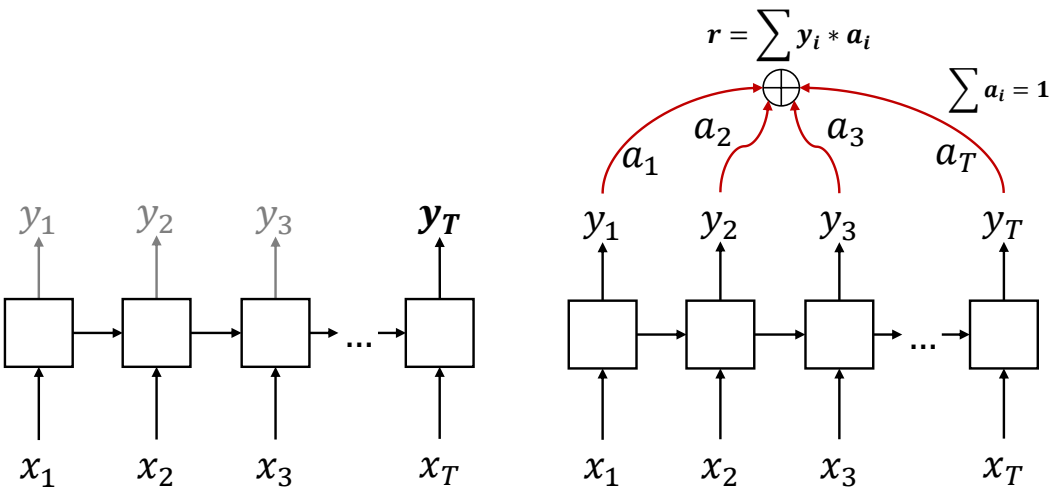
- Gating mechanism. This mechanism introduces gates, which regulate how much each network vector will be updated (internal state, output etc.). This way the network assimilates and maintains the most important information in a better way.

2.2 Attention mechanism

As we saw above, the RNN uses the last value of its inner state as the vector representation of the entire sequence. This representation is updated as the RNN reads the sequence and at the end it contains a summary of the sequence. Nevertheless, especially when the sequence is long enough, there is a case of the network not being able to contain all the important information

in its internal state. To deal with the problem in which the RNN tends to be "myopic", we can use an *attention* mechanism which attempts to enhance the contribution of the most important elements to the final representation.

This is performed by utilizing all the intermediate representations of an RNN. Thus at every time t, the output y_i is used as the "meaning" of word x_i (in the text), as the RNN encodes each word based on the context. To produce the vector representation of the entire text, we use the weighted sum of word "meanings" [Graves, 2013,Bahdanau et al., 2014], where the weight factor of each word is determined by the attention mechanism. Essentially it calculates a probability distribution over y_i and the value a_i corresponds to the importance of each word. The attention mechanism is a network layer which is trained along with the rest.



(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 3: 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.

Examples. When the sequence is a text, this means that the mechanism pays more attention to the most important words in the text. In the context of sentiment analysis, the network learns to give greater weights to the words which are crucial to identifying the emotion in a text. Respectively, in the problem of producing descriptions from pictures [Xu et al., 2015], we want to emphasize different areas of an image, depending on the meaning we want to describe. Similarly, in the context of machine translation [Bahdanau et al., 2014], when we produce the translated text, we want after each translated word to emphasize different words of the source text.

2.3 Transfer Learning

During Transfer Learning we train a statistical model at a machine learning problem and apply it (after appropriate retraining) in another application. In the context of neural networks, it was observed that the weights of models trained in huge image recognition datasets (ImageNet), could be used to initialize networks with the aim of solving other image analysis problems. These networks had learned to recognize very general features in their first layers, which were useful in many different problems. In the simplest case one can use a pretrained network for a machine learning problem, and adapt it by training it for a few epochs for another application, in which there are usually not enough training data available.

The same approach can be applied to natural language processing problems where we have few training data. The use of pretrained embeddings is a form of transfer learning. But we can also transfer the entire network [Baziotis et al., 2018], which has additionally learned to encode the context of words.

3 Questions

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, $E = (e_1, e_2, ..., e_N)$.

$$u = [mean(E)||max(E)] \tag{7}$$

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

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 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.
2.2 Use as a representation of the text u: the concatenation of the last output of the LSTM h_N, the mean of the LSTM outputs, and the maximum per dimension of the LSTM outputs.

$$u = [h_N||mean(E)||max(E)] \tag{8}$$

Question 3
In this question, we will use an attention mechanism. You can implement your own mechanism or use an existing implementation[2].
3.1 Use an attention mechanism to calculate the representation of a text, as the weighted sum of word embeddings.

$$v_i = tanh(W\boldsymbol{e_i} + b) \tag{9}$$

$$a_i = \frac{exp(v_i)}{\sum_{t=1}^{N} exp(v_t)} \tag{10}$$

$$u = \sum_{i=1}^{N} a_i \boldsymbol{e_i} \tag{11}$$

3.2 Use an attention mechanism, to calculate the representation of a text, as the weighted sum of the outputs of an LSTM.

$$v_i = tanh(W\boldsymbol{h_i} + b) \tag{12}$$

$$a_i = \frac{exp(v_i)}{\sum_{t=1}^{N} exp(v_t)} \tag{13}$$

$$u = \sum_{i=1}^{N} a_i \boldsymbol{h_i} \tag{14}$$

[2] https://gist.github.com/cbaziotis/94e53bdd6e4852756e0395560ff38aa4

Question 4
On this question you will implement the tasks of question 3 but using a **bidirectional** LSTM to encode the sentence. Caution with respect to zero-padded timesteps. The last timestep will be calculated separately per direction!
4.1 As in question 2.2, but using a bidirectional LSTM.
4.2 As in question 3.2, but using a bidirectional LSTM.

Question 5
5.1 Save on the disk a checkpoint of the model with the lowest loss in the validation set. Use the stored checkpoint and calculate the network predictions for the validation set, saving the results in a .TXT file.
5.2 Use the stored checkpoint and visualize the weights of the distributions of attention. You can use NeAt-vision[3].
5.3 Compare the results of the distributions of attention between outputs of a model trained as in question 3.1 and 3.2, respectively. What do you notice? Find 3 interesting examples and comment on them.

Question 6
Use Bag-of-Words (BoW) features combined with the features extracted from the neural network (representation). For the calculation of BoW characteristics, you can use scikit-learn[4] (CountVectorizer or TfidfVectorizer).
6.1 Calculate the final representation as a function of the two individual representations (in any way you like). Also, you are free to create in any way you want the BoW characteristics.
6.2 (bonus) In what cases could the BoW characteristics lead to better results compared to representations such as the average word embeddings?

Question 7 (bonus)
In this question we will apply transfer learning to improve performance of a model for calculating emotion. As source dataset, you will use that of Semeval 2017 Task4-A[5] [Rosenthal et al., 2017] (from the lab preparation) and as target dataset those of EI-reg (emotion intensity regression) Subtask from SemEval-2018 Task 1: Affect in Tweets[6]. See [Baziotis et al., 2018].
7.1 Implement a regression model for EI-reg Subtask. Use the architecture from question 4.2.
7.2 Run Transfer Learning to each of the emotions of EI-reg Subtask (anger, fear, joy, and sadness). Initialize the weights of your neural network, except of the last layer [7], by a model pretrained in Semeval 2017 Task4-A.

Additional Material

The following material consists of blog posts, which are quite abstract and will help you understand the concepts briefly covered in Theoretical Background.

- Operation of RNNs [Karpathy, 2015].
- Operation of LSTMs [Olah, 2015].
- Attention mechanisms [Olah, 2016].
- Transfer Learning for natural language processing [Ruder, 2018a]. Also consult the relevant work of our team [Baziotis et al., 2018].

[3] https://github.com/cbaziotis/neat-vision
[4] https://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html
[5] http://alt.qcri.org/semeval2017/task4/index.php?id=data-and-tools
[6] https://competitions.codalab.org/competitions/17751
[7] That is, only the: Embedding, LSTM, Attention layers

- Historical Review of natural language processing using deep neural networks [Ruder, 2018b]. The specified article is not related with the work, but it will give you a picture of how the techniques you used are connected with recent progress in the field.

Deliverables

For questions 1-6 you can use one of the two datasets which refer 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). The questions 6.2, 7.1, 7.2 are optional.

In terms of grading this exercise, 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 or jupyter notebook) 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.

[Baziotis et al., 2018] Baziotis, C., Nikolaos, A., Chronopoulou, A., Kolovou, A., Paraskevopoulos, G., Ellinas, N., Narayanan, S., and Potamianos, A. (2018). Ntua-slp at semeval-2018 task 1: Predicting affective content in tweets with deep attentive rnns and transfer learning. In *Proceedings of The 12th International Workshop on Semantic Evaluation*, pages 245–255.

[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.

[Graves, 2013] Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv:1308.0850 [cs]*. 00570.

[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.

[Karpathy, 2015] Karpathy, A. (2015). The unreasonable effectiveness of recurrent neural networks. https://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[Olah, 2015] Olah, C. (2015). Understanding lstm networks. https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[Olah, 2016] Olah, C. (2016). Attention and augmented recurrent neural networks. https://distill.pub/2016/augmented-rnns/.

[Rosenthal et al., 2017] Rosenthal, S., Farra, N., and Nakov, P. (2017). SemEval-2017 task 4: Sentiment analysis in Twitter. In *Proceedings of the 11th International Workshop on Semantic Evaluation*, SemEval '17, Vancouver, Canada. Association for Computational Linguistics.

[Ruder, 2018a] Ruder, S. (2018a). Nlp's imagenet moment has arrived. https://ruder.io/nlp-imagenet/.

[Ruder, 2018b] Ruder, S. (2018b). A review of the recent history of natural language processing. https://ruder.io/a-review-of-the-recent-history-of-nlp/.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., and Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *ICML*, volume 14, pages 77–81. 01037.