Exercises on Natural Language Processing with Convolutional Neural Networks and Transformers

Ion Androutsopoulos, 2022–23

***TA course*: No exercises to be handed in as assignments in this part of the course.**

***NLP course*: Submit as a group of 2–3 members a report (max. 10 pages, PDF format) for exercise 2 or exercise 3. Each group should select exactly one exercise. Include in your report all the required information, especially experimental results. Do not include code in the report, but include a link to a Colab notebook with your code. Make sure to divide fairly the work of your group to its members and describe in your report the contribution of each member. The contribution of each member will also be checked during the oral examination of your submission. For delayed submissions, one point will be subtracted per day of delay.**

1. Write down (as on slides 37–41) the equations of the CNN of slide 36, also specifying the dimensions of all the matrices and vectors involved.

Answer: The dimensionality of the word embeddings is $d = 5$. We can think of the two bigram filters as a matrix $W^{(2)} \in \mathbb{R}^{2\times 2d} = \mathbb{R}^{2\times 10}$ and a bias terms vector $b^{(2)} = \mathbb{R}^2$ (similarly to slide 39, where we have three bigram filters). Similarly, we can think of the two trigram filters as a matrix $W^{(3)} \in \mathbb{R}^{2\times 3d} = \mathbb{R}^{2\times 15}$ and a bias terms vector $b^{(3)} = \mathbb{R}^2$; and the two 4-gram filters as a matrix $W^{(4)} \in \mathbb{R}^{2\times 4d} = \mathbb{R}^{2\times 20}$ and a bias terms vector $b^{(4)} = \mathbb{R}^2$.

The embeddings of each bigram of the input text can be thought of as a vector $x^{(2)} \in \mathbb{R}^{2d}$. Applying the two bigram filters to the *i*-th bigram $x_i^{(2)}$ of the input text produces:

$$h_i^{(2)} = \text{ReLU}\left(W^{(2)}x_i^{(2)} + b^{(2)}\right) \in \mathbb{R}^2, \qquad i = 1, \ldots, 6$$

where we assumed that we use 'narrow convolutions', i.e., that the filters do not move out of the words of the input text (to partially overlap with padding tokens).

Max-pooling over $h_1^{(2)}, \ldots, h_6^{(2)}$ produces a vector:

$$h^{(2)} = \langle \max_i h_{i,1}^{(2)}, \max_i h_{i,2}^{(2)} \rangle^T \in \mathbb{R}^2$$

Similarly, applying the two trigram filters to the *i*-th trigram $x_i^{(3)} \in \mathbb{R}^{3d}$ of the input text and the two 4-gram filters to the *i*-th 4-gram $x_i^{(4)} \in \mathbb{R}^{4d}$ produces:

$$h_i^{(3)} = \text{ReLU}\left(W^{(3)}x_i^{(3)} + b^{(3)}\right) \in \mathbb{R}^2, \qquad i = 1, \ldots, 5$$
$$h_i^{(4)} = \text{ReLU}\left(W^{(4)}x_i^{(4)} + b^{(4)}\right) \in \mathbb{R}^2, \qquad i = 1, \ldots, 4$$

Max-pooling over $h_1^{(3)}, \ldots, h_5^{(3)}$ and over $h_1^{(4)}, \ldots, h_4^{(4)}$ produces:

$$h^{(3)} = \langle \max_i h_{i,1}^{(3)}, \max_i h_{i,2}^{(3)} \rangle^T \in \mathbb{R}^2$$
$$h^{(4)} = \langle \max_i h_{i,1}^{(4)}, \max_i h_{i,2}^{(4)} \rangle^T \in \mathbb{R}^2$$

The feature vector of the input text is the concatenation $h = \left[h^{(2)};\ h^{(3)};\ h^{(4)}\right]^T \in \mathbb{R}^6$.

We pass on h to a classifier, e.g., a logistic regression layer, i.e., a dense layer $W^{(P)} \in \mathbb{R}^{|C| \times 6}$ with a bias vector $b^{(P)} \in \mathbb{R}^{|C|}$ and a softmax activation function, to obtain a probability distribution $\vec{o}$ over the classes $c_1, \ldots, c_{|C|} \in C$:

$$\vec{o} = \langle P(c_1), \ldots, P(c_{|C|}) \rangle^{\mathrm{T}} = \mathrm{softmax}(W^{(P)} h + b^{(P)})$$

2. Repeat Exercise 1 of Part 4 (NLP with RNNs), now (i) using a stacked CNN with n-gram filters (e.g., $n = 2, 3, 4$), residuals, and max-pooling at the top layer (slide 42), all implemented (by you) in Keras/TensorFlow or PyTorch, **and** (ii) by fine-tuning a pre-trained BERT model.[1] Tune the hyper-parameters (e.g., values of n, number of stacked convolutional layers) on the development subset of your dataset. Monitor the performance of your models on the development subset during training to decide how many epochs to use. In (i), you may optionally add an extra CNN layer to produce word embeddings from characters, concatenating each resulting character-based word embedding with the corresponding pre-trained word embedding (e.g., obtained with Word2Vec); or you may optionally add a pre-trained language model (e.g., ELMo, see the slides of Part 4) as an extra layer to obtain context-sensitive word embeddings.[2] In (ii), if the texts of your experiments exceed BERT's maximum length limit, you may want to truncate them at the maximum allowed length of BERT or use a BERT-like model that can handle longer texts (e.g., Longformer).[3] Include experimental results of a baseline majority classifier, as well as experimental results of your best classifiers from exercise 15 of Part 2, exercise 9 of Part 3, and exercise 1 of Part 4 (if you selected those exercises), now treated as additional baselines. Otherwise, the contents of your report should be as in exercise 1 of Part 4, but now with information and results for the experiments of this exercise.

3. Repeat Exercise 2 of Part 4 (NLP with RNNs), now (i) using a stacked CNN with n-gram filters (e.g., $n = 2, 3, 4$), residuals, and a dense layer (the same at all word positions) with softmax at the top layer (slide 43), implemented (by you) in Keras/TensorFlow or PyTorch, **and** (ii) by fine-tuning a pre-trained BERT model. Tune the hyper-parameters (e.g., values of n, number of stacked convolutional layers) on the development subset of your dataset. Monitor the performance of your models on the development subset during training to decide how many epochs to use. In (i), you may optionally add a character-level CNN to produce word embeddings from characters, concatenating each resulting character-based word embedding with the corresponding pre-trained word embedding (e.g., obtained with Word2Vec); or you may optionally add a pre-trained language model (e.g., ELMo, see the slides of Part 4) as an extra layer to obtain context-sensitive word embeddings. In (ii), if your sentences exceed BERT's maximum length limit, you may want to truncate them at the maximum allowed length of BERT or use a BERT-like model that can handle longer texts (e.g., Longformer). Include experimental results of a baseline that tags each word with the most frequent tag it had in the training data; for words that were not encountered in the training data, the baseline should return the most frequent tag (over all words) of the training data. Also include experimental results of your best method from exercise 10 of Part 3 and exercise 2 of Part 4 (if you selected those exercises), now treated as additional baselines. Otherwise, the contents of your report should be as in exercise 2 of Part 4, but now with information and results for the experiments of this exercise

[1] You can use, for example, https://huggingface.co/transformers/.
[2] See https://allennlp.org/elmo.
[3] See https://huggingface.co/docs/transformers/model_doc/longformer.

4. Consider the following LSTM-based machine translation model (see also exercise 5 of Part 4 – NLP with RNNs).

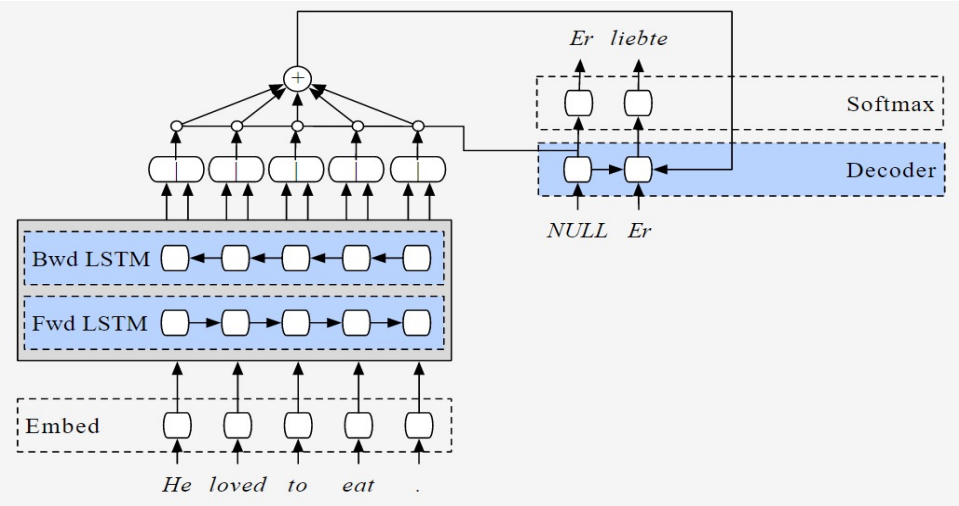
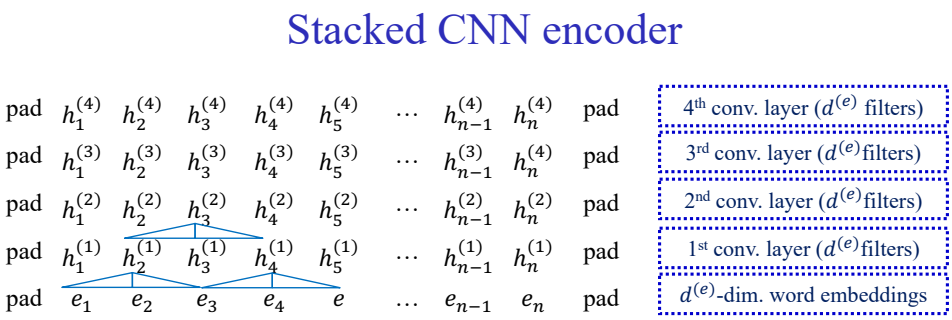


Image from Stephen Merity's http://smerity.com/articles/2016/google_nmt_arch.html

We wish to replace the BiLSTM encoder of the model above by the stacked CNN-based encoder with trigram filters illustrated below, retaining the encoder-decoder attention and the LSTM decoder of the original model.



Let V, V' be the vocabularies of the source language (English) and target language (German), respectively. Each training instance is a pair consisting of (i) a sequence of one-hot vectors:

$$x_1, x_2, x_3, \dots, x_n \in \{0,1\}^{|V|}$$

corresponding to an English sentence (each vector shows the position of the corresponding word in V) and (ii) a sequence of one-hot vectors:

$$y_1, y_2, y_3, \dots, y_m \in \{0,1\}^{|V'|}$$

corresponding to a German sentence that is the correct (gold) translation of the English one (each vector shows the position of the corresponding word in V'). For simplicity, we assume all the English sentences are n words long, and all the German sentences are m words long.

Let $E \in \mathbb{R}^{d^{(e)} \times |V|}$ and $E' \in \mathbb{R}^{d^{(e)} \times |V'|}$ contain the word embeddings of the source and target language, respectively. Notice that word embeddings have $d^{(e)}$ dimensions in both languages, and that all the convolution layers of the CNN encoder also use $d^{(e)}$ filters.

The following formulae describe how the new model works and how the loss (L) is computed, given a training instance. **Fill in the blanks** (for the solution, they have been filled in in red). The notation $[\ldots;\ldots]$ denotes concatenation and f, g denote activation functions.

Encoder: ($i \in \{1, 2, 3, \ldots, \mathrm{n}\}$, $l \in \{2, 3, 4\}$)

$e_i = E\, x_i \in \mathbb{R}^{d^{(e)}}$ (The embedding of the English word at position *i*.)

(Assume that $e_0 = e_{n+1}$ is always an all-zeros embedding of the padding token.)

$h_i^{(1)} = \mathrm{ReLU}\left(W^{(1)}[e_{i-1}; e_i; e_{i+1}] + b^{(1)}\right) + e_i \in \mathbb{R}^{d^{(e)}}$

$W^{(1)} \in \mathbb{R}^{d^{(e)} \times 3 \cdot d^{(e)}}$

$b^{(1)} \in \mathbb{R}^{d^{(e)}}$

$h_i^{(l)} = \mathrm{ReLU}\left(W^{(j)}\left[h_{i-1}^{(l-1)}; h_i^{(l-1)}; h_{i+1}^{(l-1)}\right] + b^{(l)}\right) + h_i^{(l-1)} \in \mathbb{R}^{d^{(e)}}$

$W^{(j)} \in \mathbb{R}^{d^{(e)} \times 3 \cdot d^{(e)}}$

$b^{(j)} \in \mathbb{R}^{d^{(e)}}$

Decoder: ($i \in \{1, 2, 3, \ldots, \mathrm{n}\}$, $j \in \{1, 2, 3, \ldots, m\}$)

$t_j = E' y_j \in \mathbb{R}^{d^{(e)}}$ (The embedding of the correct German word at position *j*.)

$z_j = \mathrm{LSTM}\left(z_{j-1}, [t_{j-1}; c_j]\right) \in \mathbb{R}^{d^{(e)}}$ $z_0 \in \mathbb{R}^{d^{(e)}}$, $t_0 \in \mathbb{R}^{d^{(e)}}$

$\tilde{a}_{i,j} = v^T \cdot f\left(W^{(a)}\left[h_i^{(4)}; z_{j-1}\right] + b^{(a)}\right) \in \mathbb{R}$ $W^{(a)} \in \mathbb{R}^{d^{(a)} \times 2 \cdot d^{(e)}}$

$b^{(a)} \in \mathbb{R}^{d^{(a)}}$, $v \in \mathbb{R}^{d^{(a)}}$

$a_{i,j} = \dfrac{\exp(\tilde{a}_{i,j})}{\sum_{i'} \exp(\tilde{a}_{i',j})}$

$c_j = g\left(\sum_i a_{i,j} h_i^{(4)} + b^{(c)}\right) \in \mathbb{R}^{d^{(e)}}$ $b^{(c)} \in \mathbb{R}^{d^{(e)}}$

$\tilde{o}_j = W^{(o)} z_j + b^{(o)} \in \mathbb{R}^{|V'|}$ $W^{(o)} \in \mathbb{R}^{|V'| \times d^{(e)}}$

$b^{(o)} \in \mathbb{R}^{|V'|}$

$o_{j,k} = \dfrac{\exp(\tilde{o}_{j,k})}{\sum_{k=1}^{|V'|} \exp(\tilde{o}_{j,k})}$ (How probable the model believes it is for the *k*-th word of the German vocabulary to be the correct word for the *j*-th position of the translation.)

$r_j = \mathrm{argmax}_l\, y_{j,l}$ (According to the 1-hot y_j, the correct word for the *j*-the position of the translation is the r_j-th word of the German vocabulary.)

$L = -\sum_j \log o_{j,r_j}$ (By minimizing L, we maximize the likelihood of the correct German word, at every position of the German translation.)

5. (a) We were given a BERT model pre-trained on a generic English corpus and we want to use it to build a Machine Reading Comprehension (MRC) system. The MRC system will be given a question and a paragraph (as shown in the figure) and will aim to predict the spans (sequences of tokens) of the paragraph that answer the question. The first token of each answer span should be classified as B (begin), the other tokens of the answer span as I (inside), and all the other tokens of the paragraph as O (outside). Let h_i be BERT's top-level representation of the i-th token of the paragraph and let $p_i \in [0,1]^3$ be the probability distribution over the three classes (B, I, O) produced by the MRC model for the same token. We add a task-specific dense layer on top of BERT to obtain the p_i distribution for each token of the paragraph from the corresponding h_i. **Write a formula showing how p_i is obtained from h_i**, assuming $h_i \in \mathbb{R}^{128}$. Also **write down the dimensions of all the matrices and vectors** used in the formula.

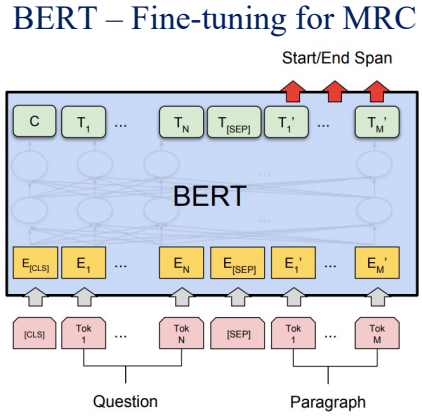


Figure from Devlin et al., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", 2018.

Formula: $p_i = \text{softmax}(Wh_i + b)$, where $p_{i,j} = \frac{[\exp(Wh_i+b)]_j}{\sum_{k=1}^{3}[\exp(Wh_i+b)]_k}$ and $j \in \{1,2,3\}$.

Dimensions: $W \in \mathbb{R}^{3\times128}, b \in \mathbb{R}^3$

(b) Write a <u>detailed formula</u> showing how you would compute the overall loss (L) for an input training instance when training the model (jointly fine-tuning BERT and training the task specific dense layer on top). Assume for simplicity that in all the training instances, the question is N_Q tokens long and the paragraph is N_P tokens long. Call t_i the correct (gold) output probability distribution for the i-th token of the paragraph. **Do not assume that every t_i is 1-hot**. For example, t_i may be a gold per-token probability distribution over the classes, based on the opinion of multiple human annotators; we may have three annotators, two of them may have said that the i-th token is a B, and the third annotator may have said it is an O, in which case the gold distribution for the token over B, I, O is 2/3, 0, 1/3.

Loss: $L = -\sum_{i=1}^{N_p}\sum_{j=1}^{3} t_{i,j} \log p_{i,j}$

(c) Now assume that every t_i is 1-hot. Show **how the formula of the previous sub-question can be simplified**. Clearly explain the steps of the simplification.

Solution:

Now for every token (at position i) of the paragraph, $t_{i,j} = 1$ if the correct class of the token is the j-th one, and $t_{i,j} = 0$ otherwise. Let $r(i)$ be the (index of the) correct class of the i-th token. Then the loss becomes:

$$L = -\sum_{i=1}^{N_p} t_{i,r(i)} \log p_{i,r(i)} = \sum_{i=1}^{N_p} \log p_{i,r(i)}$$

i.e., we maximize the log-likelihood of the correct classes of the paragraph's tokens.

(d) The MRC system of questions (a)–(c) will actually be used in the **biomedical domain**. We have **3k annotated training instances** (question-paragraph pairs with gold answer spans) **from the biomedical domain**, along with **500k additional plain text paragraphs of biomedical text** (without any questions and answer spans). The BERT model we have was pre-trained (using a masked language modeling and a next sentence prediction loss) on millions of plain text inputs, but from a generic corpus that contains very few biomedical documents. **How could we use the additional 500k plain text biomedical paragraphs to improve the performance of our MRC system?** You do not need to provide any formulae for this sub-question, but **your answer must be sufficiently detailed for an experienced colleague** (e.g., another student of the course) **to understand and implement your idea(s).**

One possible answer:

We can take the BERT model that is already pre-trained on millions of plain text inputs and further pre-train it (with masked language modeling loss and next sentence prediction loss) on the 500k additional biomedical plain text paragraphs to tailor it to the biomedical domain. We could then fine tune the pre-trained model on the 3k annotated MRC training instances of the biomedical domain, as in questions (a)–(c).