Exercises on Natural Language Processing with Convolutional Neural Networks

Ion Androutsopoulos, 2025–26

Submit as a group of 2–3 members (unless specified otherwise in the lectures) a report (max. 10 pages, PDF format) for exercises 2 and 3. 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 using a stacked CNN with *n*-gram filters (e.g., $n = 2, 3, 4$), residual connections, and global max-pooling at the top layer (slide 42), all implemented (by you) in PyTorch. 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. You may optionally add an extra CNN layer to produce word embeddings from characters (slide 44), concatenating each resulting character-based word embedding with the corresponding pre-trained word embedding (e.g., obtained with Word2Vec). Include experimental results of a baseline majority classifier, as well as experimental results of your best classifiers from exercise 11 of Part 2, exercise 4 of Part 3, and exercise 1 of Part 4. 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. You may optionally wish to try ensembles (e.g., majority voting of the best checkpoints, temporal averaging of the weights of the best checkpoints, combining RNN and CNN classifiers).

→ **3**. Repeat Exercise 2 of Part 4 (NLP with RNNs), now using a stacked CNN with *n*-gram filters (e.g., $n = 2, 3, 4$), residual connections, and a dense layer (the same at all word positions) with softmax at the top layer (slide 43), implemented (by you) in PyTorch. 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. 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). 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 5 of Part 3 and exercise 2 of Part 4. 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. You may optionally wish to try ensembles.

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

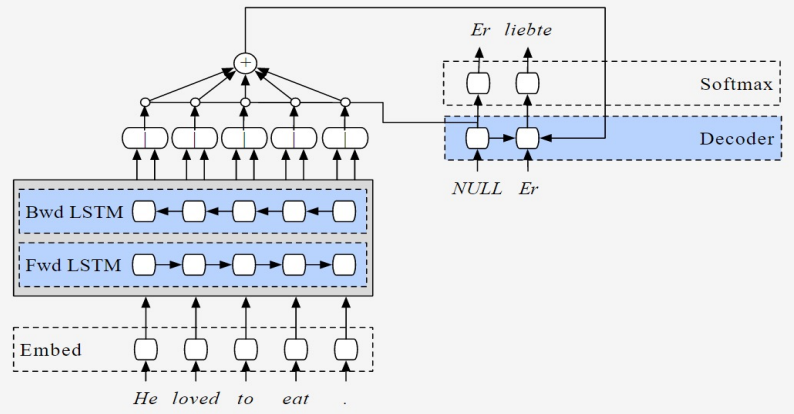
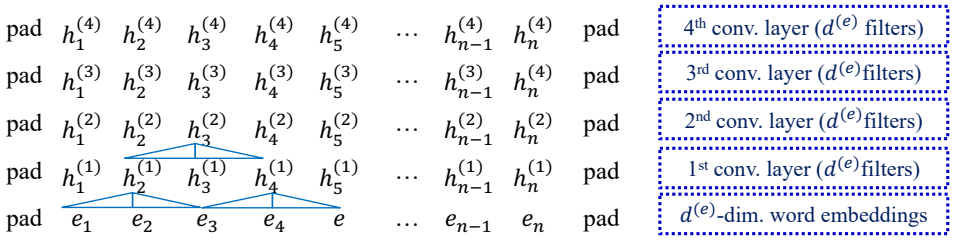


Image from Stephen Merity's http://smerity.com/articles/2016/google_nmt_arch.html

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.

Stacked CNN encoder



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:

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

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

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

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*.)

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

(We assume we are in training mode and that we use teacher forcing, hence we use the correct previous German word as the previous word, which has embedding t_{j-1}.)

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

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

$a_{i,j} = \dfrac{\exp(\tilde{a}_{i,j})}{\sum_{i'=1}^{n} \exp(\tilde{a}_{i',j})}$

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

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

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

$o_{j,k} = \dfrac{\exp(\tilde{o}_{j,k})}{\sum_{k=1}^{|V'|} \exp(\tilde{o}_{j,k})}$ (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.)

$L = -\sum_{j=1}^{m} \log o_{j,r_j}$ (By minimizing *L*, we maximize the likelihood of the correct German word, at every position of the German translation.)