Exercises on Natural Language Processing with Recurrent Neural Networks

Ion Androutsopoulos, 2021–22

Submit as a group of 2–3 members a report (max. 10 pages, PDF format) for exercise 1 or exercise 2. (Each group should select exactly one of these two exercises.) You may optionally submit also exercise 6 for extra bonus (your report may then be up to 20 pages). 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. Repeat exercise 9 of Part 3 (text classification with MLPs), now using a bi-directional stacked RNN (with GRU or LSTM cells) and a self-attention MLP (slide 16), all implemented (by you) in Keras/TensorFlow or PyTorch. Tune the hyper-parameters (e.g., number of stacked RNNs, number of hidden layers in the self-attention MLP, dropout probability) on the development subset of your dataset. Monitor the performance of the RNN on the development subset during training to decide how many epochs to use. You may optionally add an extra RNN layer to produce word embeddings from characters (e.g., as on slide 19), concatenating each resulting character-based word embedding with the corresponding pre-trained word embedding (e.g., obtained with Word2Vec). You may optionally add a pre-trained language model (e.g., ELMo, slides 34–36) as an extra layer to obtain context-sensitive word embeddings.[1] Include experimental results of a baseline majority classifier, as well as experimental results of your best probabilistic classifier from exercise 15 of Part 2 and your MLP classifier from exercise 9 of Part 3 (if you chose that exercise), now treated as baselines. Include in your report:

- Curves showing the loss on training and development data as a function of epochs.
- Precision, recall, F1, precision-recall AUC scores for each class and classifier, separately for the training, development, and test subsets, as in exercise 15 of Part 2.
- Macro-averaged precision, recall, F1, precision-recall AUC scores (averaging the corresponding scores of the previous bullet over the classes), for each classifier, separately for the training, development, and test subsets, as in exercise 15 of Part 2.
- A short description of the methods and datasets you used, including statistics about the datasets (e.g., average document length, number of training/dev/test documents, vocabulary size) and a description of the preprocessing steps that you performed.

2. Repeat exercise 10 of Part 3 (text classification with MLPs), now using a bi-directional stacked RNN (with GRU or LSTM cells) implemented (by you) in Keras/TensorFlow or PyTorch. Tune the hyper-parameters (e.g., number of stacked RNNs, dropout probability) on the development subset. Monitor the performance of the RNN on the development subset during training to decide how many epochs to use. You may optionally add an extra RNN layer to produce word embeddings from characters (e.g., as on slide 19), concatenating each resulting character-based word embedding with the corresponding pre-trained word embedding (e.g., obtained with Word2Vec). You may optionally add a pre-trained language model (e.g., ELMo, slides 34–36) as an extra layer to obtain context-sensitive word embeddings. 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.

[1] See https://allennlp.org/elmo.

Also include experimental results of your best method of exercise 10 of Part 3 (if you chose that exercise), now treated as an additional baseline. Include in your report:

- Curves showing the loss on training and development data as a function of epochs.
- Precision, recall, F1, precision-recall AUC scores for each class (tag) and classifier, separately for the training, development, and test subsets, as in exercise 15 of Part 2.
- Macro-averaged precision, recall, F1, precision-recall AUC scores for each classifier, separately for the training, development, and test subsets, as in exercise 15 of Part 2.
- A short description of the methods and datasets you used, including statistics about the datasets (e.g., average document length, number of training/dev/test documents, vocabulary size) and a description of the preprocessing steps that you performed

3. Write down the equations for a modified version of the "RNN with deep self-attention" (slide 16), where the uni-directional RNN with GRU cells is replaced by a stacked bi-directional RNN with GRU cells. Use the notation $\text{GRU}(h_{t-1}, \tau_t)$ to denote the new state of a GRU cell with previous state h_{t-1} and input τ_t.

Answer: At the first layer of the GRU RNN, we have (for $t = 1, \ldots, k$):

$$\vec{h}_t^{(1)} = \text{GRU}\left(\vec{h}_{t-1}^{(1)}, x_t\right)$$

$$\overleftarrow{h}_t^{(1)} = \text{GRU}\left(\overleftarrow{h}_{t+1}^{(1)}, x_t\right)$$

$$h_t^{(1)} = [\vec{h}_t^{(1)}; \overleftarrow{h}_t^{(1)}]$$

where $\vec{h}_0^{(1)}$ is the initial state of the left-to-right GRU RNN of the first layer, $\overleftarrow{h}_{k+1}^{(1)}$ is the initial state of the right-to-left GRU RNN of the first layer, `;' denotes concatenation, and $x_1, \ldots, x_k$ are the word embeddings of the input word sequence.

Similarly, at the m-th layer of the GRU RNN:

$$\vec{h}_t^{(m)} = \text{GRU}\left(\vec{h}_{t-1}^{(m)}, h_t^{(m-1)}\right)$$

$$\overleftarrow{h}_t^{(m)} = \text{GRU}\left(\overleftarrow{h}_{t+1}^{(m)}, h_t^{(m-1)}\right)$$

$$h_t^{(m)} = [\vec{h}_t^{(m)}; \overleftarrow{h}_t^{(m)}]$$

The other equations remain as on slide 16.

4. Modify the equations of the neural network of the previous exercise to support *multi-label classification*, i.e., cases where the same text (e.g., tweet) may belong in multiple classes (labels). Use a separate *label-specific self-attention-head* for each class, which will produce a different distribution of attention scores $a_{c,1}, \ldots, a_{c,k}$ (where k is again the length of the input text, counted in words) and a different $h_{sum,c}$ for each class c. Feed the $h_{sum,c}$ of each class c to a separate (different per class) dense layer with a sigmoid to produce the probability that the input text should be assigned class c.

Answer: Let C be the set of possible classes (labels). We modify the self-attention MLP of slide 16, so that $a_t^{(l)} \in \mathbb{R}^{|C|}$, i.e., $a_t^{(l)}$ is now a vector (not a scalar) containing $|C|$ attention scores $a_{1,t}, \ldots, a_{|C|,t}$ for word position t, one for each possible class. To achieve this, we

modify the dimensions of $W^{(l)}$ and $b^{(l)}$ of layer l of the self-attention MLP, to be $|C| \times d$ and $|C|$, respectively, where d is the dimensionality of the previous layer $a_t^{(l-1)}$.

The softmax of slide 16 is now applied *label-wise*, on the attention scores of a particular class, i.e., for each possible class c:

$$a_{c,t} = \text{softmax}(a_{c,t}^{(l)}; a_{c,1}^{(l)}, \ldots, a_{c,k}^{(l)})$$

We form a separate weighted sum $h_{sum,c}$ for each possible class c:

$$h_{sum,c} = \sum_{t=1}^{k} a_{c,t} h_t^{(M)}$$

where M is the number of stacked GRU RNNs of the previous exercise, and we feed each $h_{sum,c}$ to a separate dense layer $W_{p,c}$ (with bias term $b_{p,c}$) per class c, to compute the probability of the corresponding class:

$$P(c|x_1, \ldots, x_k) = \sigma(W_{p,c} h_{sum,c} + b_{p,c})$$

The other equations of the neural network remain as in the previous exercise.

5. We train the following neural machine translation model.

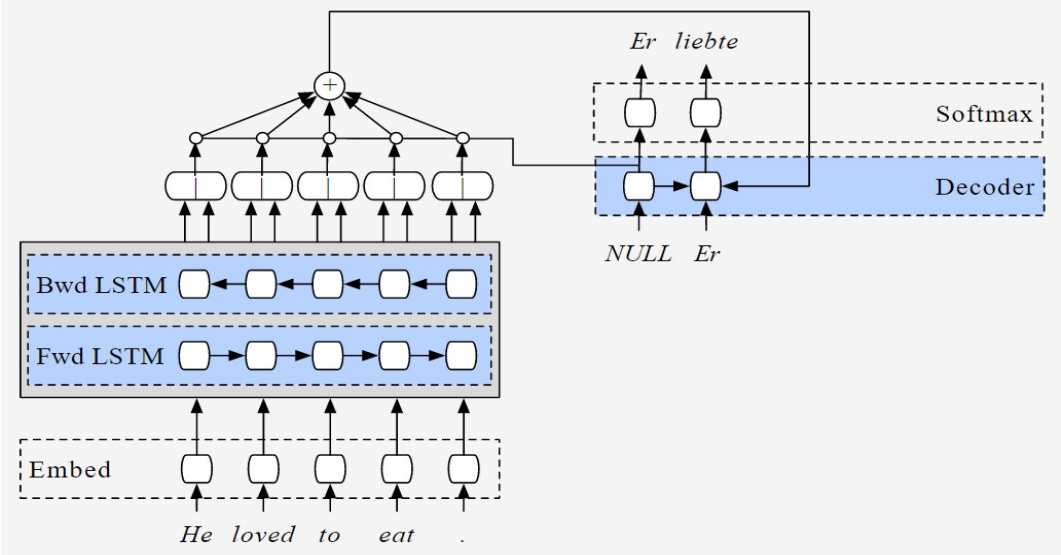

Image from Stephen Merity's http://smerity.com/articles/2016/google_nmt_arch.html

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').

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. (Word embeddings in both languages have $d^{(e)}$ dimensions.)

The following formulae describe how the 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 $[\dots;\dots]$ denotes concatenation and f, g denote activation functions.

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

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

$\vec{h}_i = \mathrm{LSTM}(\vec{h}_{i-1}, e_i) \in \mathbb{R}^{d^{(h)}}$ $\quad \vec{h}_0 \in \mathbb{R}^{d^{(h)}}$

$\overleftarrow{h}_i = \mathrm{LSTM}(\overleftarrow{h}_{i+1}, e_i) \in \mathbb{R}^{d^{(h)}}$ $\quad \overleftarrow{h}_{n+1} \in \mathbb{R}^{d^{(h)}}$

$h_i = [\vec{h}_i; \overleftarrow{h}_i] \in \mathbb{R}^{2 \cdot d^{(h)}}$

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

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

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

$\tilde{a}_{i,j} = v^T \cdot f(W^{(a)} h_i + U^{(a)} z_{j-1} + b^{(a)}) \in \mathbb{R}$ $\quad W^{(a)} \in \mathbb{R}^{d^{(z)} \times 2 \cdot d^{(h)}}$, $U^{(a)} \in \mathbb{R}^{d^{(z)} \times d^{(z)}}$, $b^{(a)} \in \mathbb{R}^{d^{(z)}}$, $v \in \mathbb{R}^{d^{(z)}}$

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

$c_j = W^{(c)} \cdot g(\sum_i a_{i,j} h_i + b^{(c)}) \in \mathbb{R}^{d^{(e)}}$ $\quad W^{(c)} \in \mathbb{R}^{d^{(e)} \times 2 \cdot d^{(h)}}$, $b^{(c)} \in \mathbb{R}^{2 \cdot d^{(h)}}$

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

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

6. [**optional**] Repeat exercise 3 of Part 1 (*n*-gram language models and context-aware spelling correction) now using an RNN language model, instead of an *n*-gram language model. Compare the WER and CER scores you obtain using the RNN language model to those you had obtained with the bigram and trigram language models.