Deep Learning for NLP 2019
Home Exercise 05

TECHNISCHE UNIVERSITÄT DARMSTADT

Due on Tuesday, 21.05. at 13:00

Submission Guidelines for all Home Exercises

- When submitting multiple files, submit one **zip-archive**.
- Submit python code as plain python scripts (**.py**). **Must** be runnable in the given Docker container.
- Submit answers to non-code assignments in **one PDF** file. Scans are permitted, if readable.
- Guidelines specific to neural network code:
  - Please submit your training/testing results (a copy of your console output is fine). Reasoning: Your network might train much slower on the tutor's system than on yours.
  - If you are aware that your network never stops training, please be honest and add a short statement saying so. Thank you!

Problem 1 Mandatory Paper (1P)

Read this week's mandatory paper[1]. State the three issues from which traditional feature template approaches suffer and explain each issue in one sentence.

Problem 2 Poetry Generation (9P)

As a part of this exercise, you will develop a MLP baseline for this year's DL4NLP shared task on *poetry generation*. An explanation of the task and the relevant dataset formats can be found in section Problem 3 on the last page of this home exercise.

Note: You may work in groups on the shared task. This home exercise is still individual work for everyone!

Problem 2.1 Preparation (1P)

a) Create an account on CodaLab competitions: `https://competitions.codalab.org/`

b) The DL4NLP shared task is not publicly listed. Apply for access to the DL4NLP 2019 shared task via this URL: `https://competitions.codalab.org/competitions/22850?secret_key=3f6256dc-611f-439d-8c8e-cee07e13e758`

c) Upload the sample submission included in `hex05_data.zip` on CodaLab for the "trial phase".

d) State your CodaLab username in your solution to this home exercise.

[1] `https://www.aclweb.org/anthology/D14-1082`

Problem 2.2 Data Input and Preprocessing (3P)

Refer to section Problem 3 for a description of the data format.
Since the task is a generation task, both the inputs and the outputs of your system will be sequences. There exist neural architectures specifically for this purpose, but those will be covered at a later point in the semester. Instead, you will be using n-grams and a multi-layer perceptron to deal with sequential data in this home exercise. The data is included in `hex05_data.zip`.

a) Implement a function with the path to a data file that contains poems as a parameter. It should read the poems of the data file and replace the newline after every verse with a special end-of-line token "`<eol/>`", except for the last verse of a poem, which should end with an end-of-sequence token "`<eos/>`". Additionally it should split the poems at whitespace characters to obtain a list of tokens. Also all tokens should be lowercased. The function should return a list of tokenized poems. (1P)

b) Implement a preprocessing function which, given some tokenized poems and an integer n as parameters, returns the poems as a flat list of (n-gram, label) tuples, where the label is the next token, after the n-gram ends. An n-gram is defined as a list of words $(w_t, w_{t+1}, \ldots, \ldots, w_{t+n-1}, w_{t+n})$. If a poem should have less than $n+1$ tokens, it should be ignored. (2P)

Demonstrate that your code works by printing the preprocessed version of the first training poem for $n = 7$.

Poem
Lorem ipsum dolor sit amet , consectetur adipiscing elit .
Sed do eiusmod
⋮

(a) Input

N-Gram					Label
(	lorem	ipsum	<eol/>	)	dolor
(	ipsum	<eol/>	dolor	)	sit
		⋮			⋮
(	adipiscing	elit	.	)	<eos/>
		⋮			⋮

(b) Output

Table 1: Example for $n = 3$

Problem 2.3 Embedding Preparation (2P)

The tokens should be represented by embedding vectors in the neural network. This subtask deals with steps necessary to make pretrained embeddings work in TensorFlow / Keras.

Problem 2.3.1 Download Pretrained Embeddings (0P)

Download the pretrained, uncased GloVe embeddings with 6B tokens (`glove.6B.zip`) from Stanford: `https://nlp.stanford.edu/projects/glove/` For performance reasons, we will only use the 50-dimensional embeddings (`glove.6B.50d.txt`).

Problem 2.3.2 Embedding Integration Steps (2P)

Write code which performs the following steps:

a) Read the pretrained embeddings for words that occur in the training data with gensim or by hand. This should give you the embedding matrix and a dictionary which maps from a token to the row index of its vector in the embedding matrix.

b) Define embedding vectors for your "<eol/>" and "<eos/>" tokens. The vectors should be randomly generated. Append the vectors to the matrix and add corresponding entries to the dictionary.

c) For simplicity, define a single out-of-vocabulary token (for example "<oov/>") and also choose a random vector as its embedding vector. Again append the vector to the matrix and add an entry to the dictionary.

d) Any time you feed data to your network (i.e. at training and test time): Replace every token in your input data with its respective index from your dictionary. If a token is out-of-vocabulary, use the vector of the out-of-vocabulary token.

Hint: Many guides can be found for using pretrained embeddings with TensorFlow and Keras, for example this one[2] or this one[3].
Demonstrate that your code works by printing the indices of all (n-gram, label) tuples of the first training poem ($n = 7$).

Problem 2.4 MLP Architecture (2P)

Implement a multilayer perceptron with either TensorFlow or Keras (your choice). You may reuse your code from home exercise 3 (or the code from the sample solution).

As the input, your MLP should accept an embedded n-gram. Use the embedding layer from TensorFlow / Keras [4] for this purpose. After the embedding layer, add one hidden layer with $n \cdot 500$ units. The output layer should be softmax and the loss should be cross-entropy loss. The network output should then represent the probabilities of each word in the vocabulary. The word with the highest probability should hopefully have the index of the label.

Train your model on the first 3000 poems from the training data. Use $n = 7$ for the n-grams and `glove.6B.50d.txt` for the pretrained embeddings. You may chosse the remaining parameters (activation function, optimizer, batch size, learning rate, ...) freely. Train your network for a reasonable number of epochs.

Problem 2.5 Generate Poems (2P)

Use your trained model to generate 1000 new poems. To start the generation process a seed text is required, which is used as the first input sequence to your language model. Use the first n-grams of 1000 randomly selected poems from the other half of the training data as your seed texts. The generation process of a poem then roughly follows this cycle:

a) Get the prediction of the current n-gram.

b) Append the word that corresponds to the prediction to the generated text.

c) Append the predicted index to the last $n-1$ indices of the current n-gram and use the resulting n-gram as the input to the next cycle.

d) Repeat until the "<eos/>" token occurs.

To prevent the case, that a model generates an infinitely long poem because it doesn't produce the "<eos/>" token, make sure to stop the generation process manually when 60 words are generated.
Implement a method which writes your generated poems into a file of the required format (see Problem 3). Put the resulting file in a zip-archive and upload your predictions to CodaLab for the trial phase.

[2] TensorFlow embedding guide: `https://www.tensorflow.org/versions/master/programmers_guide/embedding`
[3] A Keras embedding guide: `https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/`
[4] TensorFlow embedding layer: `https://www.tensorflow.org/api_docs/python/tf/nn/embedding_lookup`
Keras embedding layer: `https://keras.io/layers/embeddings/`

Problem 2.6 Bonus: Baseline Optimization (1P)

You can earn a bonus point by tuning your multilayer perceptron via random hyperparameter optimization (similar to home exercise 03). For this, you will have to split the given training dataset into a training split and a development split (for example with a ratio of 90:10).

Report 10 hyperparameter configurations and their result (perplexity) on your development split. Then, train your network on the full training data (training split + development split) and upload your prediction for the given development data to CodaLab.

Problem 3 Abbreviated Description of the DL4NLP Shared Task (0P)

Poetry Generation

Poetry is a unique artifact of the human language faculty, with its defining feature being a strong unity between content and form. Poetry Generation is a problem from the Natural Language Generation domain, that focuses on the automatic generation of poems.
In this shared task, we want to explore various methods revolving around neural networks to create poems artificially.

Data Formats

Training / Development Data

The training data file contains several *poems*. Poem boundaries are represented by empty lines. A poem itself consists of several lines, where every row contains a *verse*. A verse contains several *tokens*, i.e. words, which are separated by whitespaces.

CodaLab Submission Format

For submissions, CodaLab expects the following file format:

```
<submission>.zip
└ poems.txt
```

The ZIP filename is not relevant. The ZIP file must not contain any subfolders. The name of the prediction file must literally be "`poems.txt`". The prediction file must contain exactly 1000 poems.
The submitted poems must be of the same format as the training data. However files that end with an empty newline are also valid.
A sample submission file is provided for comparison.