Deep Learning for NLP 2020
Home Exercise 6

Due on Monday, 01.06. at 18:00

May 24, 2020

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!

In this exercise, you will develop a simple baseline for this year's DL4NLP shared task, which deals with **semantic textual similarity under low-level adversarial attacks**.

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

1 Mandatory Paper (1P)

Please read this week's mandatory paper which you can find in the Moodle course. Explain in one sentence how characters are embedded in the paper using continuous vectors. Which kind of similarity do the embeddings capture?

2 Semantic Textual Similarity (7P)

In this task, we define *semantic textual similarity* (STS) as a **supervised** regression task in which the semantic similarity of two pieces of text (typically sentences) should be determined.

2.1 Data Formats (1P)

The labeled datasets for this task contain entries that comprise a real-numbered similarity score between 0 and 1, a first sentence, and a second sentence. The unlabeled datasets' entries comprise only the two sentences. In our case, each line in the datasets corresponds to a single entry, and the score and the sentences are separated by tabs '\t'.

Later on, in the shared task, you will submit .txt files in which every row contains the real-numbered similarity score between 0 and 1 for the same row in the corresponding unlabeled test dataset.

1. Implement a reader function for the labeled datasets, which for a given filename returns a list of scores, a list of first sentences, and a list of second sentences.
2. Implement a reader function for the unlabeled datasets, which for a given filename returns a list of first sentences and a list of second sentences.
3. Implement a writer function, which for a given filename and list of real-numbered similarity scores writes these scores to the file.

Hint: Set the encoding to 'utf8'and use strip() to remove whitespace from the beginning and ending of the sentences.

2.2 Embedding the Sentences (3P)

We will use the averages of the words' FastText[1] embeddings to embed both sentences.

1. Download the 'wiki-news-300d-1M.vec.zip'embeddings and read them into a Python dictionary that maps every token to the corresponding vector. Represent the vectors as NumPy arrays. It is fine to load only the first 20000 or 40000 lines of the file.
2. Implement a function that tokenizes the sentences using nltk.word_tokenize(...).
3. Implement a function that maps the tokens to their corresponding embedding vectors. If a token does not exist in FastText's vocabulary, embed this token as a 0-vector with the same dimension as the FastText embeddings.
4. Finally, implement a function that embeds each sentence as the average of the embeddings of its tokens.

2.3 Scoring the Similarity (3P)

We will train a simple multi-layer perceptron to score the similarity of the two sentences. The MLP should concatenate both inputs and have two hidden layers with dropout. The first hidden layer should have 300 dimensions and ReLu activation, and the second hidden layer (the output layer) should have 1 dimension and sigmoid activation.

The precise definition is as follows:

- Two inputs, one for each sentence embedding. **Hint:** You may use tensorflow.keras.layers.Input(...).
- A layer that concatenates both inputs. **Hint:** You may use tensorflow.keras.layers.Concatenate(...).
- A dropout layer with probability 0.3. **Hint:** You may use tensorflow.keras.layers.Dropout(...).
- A dense layer with 300 dimensions and relu activation.
- A dropout layer with probability 0.3.
- A dense layer with 1 dimension and sigmoid activation.

Hint: You may use tensorflow.keras.Model(<input>,<output>) to define the model.

1. Briefly explain why we are using sigmoid instead of softmax as the activation function in the final layer.
2. Implement the model described above.
3. Compile the model with the Adam optimizer and mean squared error as the loss function. Also, choose mean squared error as a metric.
4. Train the model with batch size 100 for 300 epochs on the prepared training dataset. Observe the mean squared error on the development dataset.
5. Report the model's final mean squared error on the development dataset.

[1] `https://fasttext.cc/docs/en/english-vectors.html`

3 Low-Level Adversarial Attacks (2P)

- Finally, evaluate your trained model on the test set and report its mean squared error **(1P)**.

You may notice that it performs worse than on the development set. That is because the test set has been perturbed with so-called *adversarial attacks* (cf. the mandatory paper).

Adversarial attacks are modifications to the input of a model that do not change the label or score. Usually, their goal is to fool a deep learning model but not humans. Thus, attacks are often chosen in such a way that humans are robust to them.

In our test set, these attacks are:

(i) **disemvoweling:** Some of the vowels have been removed.

(ii) **visual attacks:** Similar-looking characters have been replaced by each other.

- Suggest (in at most two sentences) a method to make your model more robust to adversarial inputs **(1P)**.

The test sets for the shared task will introduce additional adversarial attacks. Your task will be to design models that can deal with corrupted inputs in a similar way as humans can.