INF5820, Fall 2018 Assignment 3: Sentiment Analysis with Convolutional Neural Networks

UiO Language Technology Group

Deadline 26 October, at 22:00, to be delivered in Devilry

Goals

1. Learn to apply convolutional neural networks to sentence classification
2. Get more experience in performing sensitivity analysis for different hyper-parameters of deep neural networks
3. Learn to employ functional models in *Keras*
4. Try subword-level word embedding models for inferring representations for out-of-vocabulary words.

Introduction

This obligatory assignment aims at using **Convolutional Neural Networks** (CNNs) in sentence-level sentiment analysis: that is, detecting the emotional attitude of the author, based on the text. The primary and recommended machine learning framework in this course is *Keras* with *TensorFlow* as a backend. These are provided out-of-the-box in the LTG environment on the Abel cluster[1], and we strongly encourage you to train your models using Abel. If you would like to use any other deep learning software library, it is absolutely allowed, provided that your code is available.

Make sure you read through the entire assignment before you start. Solutions must be submitted through Devilry[2] **by 22:00 on October 26**. Please upload a single 3-7 pages PDF file with details on your experiments and your answers to the questions posed in the assignment.

Your heavily commented code and data files should be available in a separate private repository in the UiO in-house Git platform[3] (we remind that the official programming language of the INF5820 course is *Python 3*). The code must be self-sufficient, meaning that it should be possible to run it directly on the data. If you use some additional data sources, these datasets should be put in the repository as well. Make the repository private, but allow full access to

[1] `https://www.uio.no/studier/emner/matnat/ifi/INF5820/h18/setup.html`
[2] `https://devilry.ifi.uio.no/`
[3] `https://github.uio.no/`

INF5820 teaching staff[4]. The PDF in Devilry should contain a link to your repository. If your data files or models exceed file size limit of the UiO Git platform, leave them in your home directory on Abel, and provide the path to them in the report. If you use some non-standard *Python* modules, this should be documented in your report as well.

If you have any questions, please raise an issue in the INF5820 Git repository[5] or email us at *inf5820-help@ifi.uio.no*. Make sure to take advantage of the group sessions before the deadline.

Recommended reading

1. **Neural network methods for natural language processing.** Goldberg, Y., 2017.[6]
2. **A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification**. Ye Zhang and Byron Wallace, 2017.[7]
3. **Stanford Sentiment Treebank** (`https://nlp.stanford.edu/sentiment/`)
4. `https://keras.io/`
5. `https://www.tensorflow.org/`

The assignment is divided into 4 parts:

1. **Implementing a baseline sentiment prediction system**;
2. **Controlling randomness**;
3. **Sensitivity analysis: tuning hyperparameters**;
4. **Testing the influence of word embeddings**;

1 Baseline sentiment analysis

In its basic form, sentence-level sentiment analysis is a supervised classification task: we want to label text instances (sentences) with two or more class labels, for example, '*positive*', '*negative*' and '*neutral*'. To achieve this, one can train a neural classifier on a manually annotated collection of sentences. In this obligatory assignment, we will use a subset of the Stanford Sentiment Treebank or **SST** (see **Recommended Reading** for more details on this dataset). You can find training and development (validation) parts of this dataset at `/projects/nlpl/teaching/uio/inf5820/2018/SST/` directory on Abel. The training and development sets contain about 6 500 and 800 sentences respectively. All sentences are labeled with one of two sentiment categories: '*positive*' or '*negative*' (so this a binary classification task). There is also a held-out test

[4] `https://github.uio.no/orgs/inf5820/people`
[5] `https://github.uio.no/inf5820/course2018/issues`
[6] `https://www.morganclaypool.com/doi/10.2200/S00762ED1V01Y201703HLT037`
[7] `https://www.aclweb.org/anthology/I/I17/I17-1026.pdf`

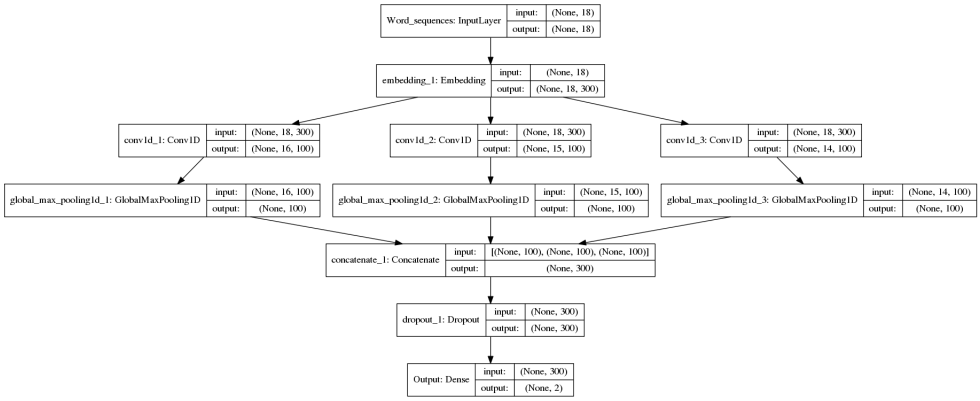
set of about 1 700 sentences, which will be published only after the deadline for this assignment is over.

The datasets consist of 5 columns. '`sent_id`' and '`phrase_id`' are technical identifiers from the original dataset, and you don't need them. The '`label`' column contains the sentiment labels, while the '`tokens`' column contains the sentences themselves (they are already tokenized, with each token separated by spaces, including punctuation marks). The '`lemmatized`' column contains the same sentences, automatically lemmatized and PoS-tagged with the *UDPipe* tagger. You are free to somehow use or completely ignore this last column.

Your task is to train a neural classifier similar to the ones you implemented in the previous assignments, but this time employing convolutional neural networks applied to word sequences. CNNs is a well-known method of feature selection for linguistic sequences, which is actively used for sentiment analysis (since sentiment is often expressed with word n-grams). You must first reproduce the architecture from the baseline model described in the Table 1 from the '*A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification*' paper (see **Recommended Reading**). Its hyperparameters include:

1. categorical cross-entropy loss;
2. pre-trained 300-D *word2vec* embeddings trained on the Google News corpus; you can use the PoS-tagged version of this model from the NLPL repository (identifier 1), or find the original version on the Web;
3. single channel (sequence of **static** word embeddings);
4. window sizes 3, 4 and 5;
5. 100-D filters (per each window size);
6. ReLU as the activation function in the convolution layers;
7. 1-max pooling;
8. dropout of 0.5 after the convolution layers.

As a result, you should have an architecture looking approximately like the one on the plot below (is uses word sequences of length 18, but you can choose another length to pad and truncate the sentences from the dataset):



The most convenient way to reproduce this architecture in *Keras* is to use its **functional API** (`https://keras.io/getting-started/functional-api-guide/`). Report the accuracy of this baseline model on the provided validation set (both global and per-class scores).

2 Controlling randomness

Due to random parameter initialization, one and the same neural architecture can yield different results from run to run. Assess the variance of your model results: train it 10 times and report the average, minimum and maximum values.

One can almost completely eliminate this randomness by fixing the random seed in the *Python* code. Try to do this and repeat the experiment from the previous section. Make sure that the variance did decrease to negligible values or disappeared completely.

3 Sensitivity analysis: tuning hyperparameters

The performance of any machine learning system can be sensitive to various values of hyperparameters. You have to perform a **sensitivity analysis** of your model: pick at least three hyperparameters to tune, with at least three different values for each. For example, you can experiment with:

- activation functions;
- convolutional window sizes;
- numbers of filters per window size;
- regularization terms;
- pooling strategies;
- using raw or lemmatized and PoS-tagged versions of the data; make sure that your word embedding model vocabulary matches your input data (it does not make sense to look for inflected word forms in the word embedding model trained on lemmatized texts);
- using several channels for your convolution layers, for example, words and PoS tags, or words and punctuation marks.

For simplicity, you can ignore the issue of co-dependent values. Keep all other settings the same as in your 'baseline' model, and just change the value of a single hyperparameter at one time. Motivate your choices. Evaluate the resulting models, report and discuss your results.

4 Testing the influence of word embeddings

Pick at least 2 of the subtasks below and complete them.

4.1 Influence of algorithms

Experiment with using pre-trained models from different distributional algorithms (*word2vec*, *fastText*, *GloVe*). You can find plenty of models in the NLPL repository (recall that you don't have to download them, since they are available directly from Abel) and on the Web. Do you see any clear differences between them? Report and discuss your results.

4.2 Influence of vocabulary size

Choose several word embedding models with different vocabulary sizes (or use the `limit` parameter in Gensim[8] to change the number of loaded words in one and the same model). Find out how many words in the Stanford Sentiment dataset are out-of-vocabulary (OOV) in each case. Draw a plot showing how the number of OOV words is dropping as the vocabulary size of the word embedding models is increasing.

4.3 Inferring vectors for OOV words

The *fastText* models use character n-gram embeddings to infer vectors for unknown words (*Gensim* does this automatically when you try to lookup an OOV word). Use this ability to get rid of OOV tokens in your training data. Does it improve the performance of your sentiment analysis system?

4.4 Influence of fine-tuning word embeddings

For a given pre-trained word embedding model of your choosing, experiment with keeping the word embeddings static (*frozen embedding layer*, the default mode) VS allowing them to be fine-tuned during training (*unfrozen embedding layer*). NB: training word embedding weights along with other model weights can consume lots of RAM, especially in the case of embedding models with large vocabularies. Report and discuss your results.

5 Conclusion

Your best sentiment analysis system should be published in your UiO Github repository as a saved *Keras* model. If this system used a word embedding model not present in the NLPL repository, this model must be made available in your home directory on Abel.

Also, you have to provide an `eval_on_test.py`[9] script. It should take as an input a saved *Keras* model, word embedding file and a test dataset (in the same format as the training dataset), apply all the data transformations you used, load the model, make predictions and return accuracy scores for each class in the test set, as well as the overall accuracy.

Good luck and happy coding!

[8] `https://github.com/RaRe-Technologies/gensim/blob/master/gensim/models/keyedvectors.py#L1422`

[9] Similar to the one at `https://github.uio.no/inf5820/course2018/blob/master/obligatories/1/solution/eval_on_test.py`