Deep Learning Workshop - TensorFlow

July 13, 2016

Exercise 1 - Simple binary-classification on 2d-data

For k classes, we can define multinomial logistic regression as

$$P[c = j] = \frac{e^{\beta_j^\top \mathbf{x}}}{\sum_i e^{\beta_i^\top \mathbf{x}}}$$

Instead of k individual terms $\beta_j^\top \mathbf{x}$, we can think of a vector $\mathbf{W}\mathbf{x} + \mathbf{b} \in \mathbb{R}^k$ that parametrizes the underlying normalization function. In the language of neural networks, we use a linear layer on the input $\mathbf{x}$.

In tensorflow, we can express this using a softmax. The classification loss can be measured in terms of cross-entropy loss.

$$\text{input} \rightarrow \text{linear} \rightarrow \text{softmax} \rightarrow \text{cross-entropy loss}$$

Your task

- Implement the suggested graph and run the training routine. You should see the train and the test error decrease.
- Start tensorboard using

  ```
  tensorboard --logdir=/your/logdir/path
  ```

  [1] and go to `http://localhost:6006` (Chrome recommended but Firefox should work, too). Open the Graph section and try to follow the flow of information in the main graph. Match it to your code.
  The "gradients" box under auxiliary nodes can be expanded to show the graph necessary to compute the gradients.
- Add another layer on top of the first and retrain. Does the performance increase?
- To optimize the parameters more sensibly, introduce a linear learning rate decay of the form

  $$\text{lr} = \text{initial\_lr}\left(1 - \frac{t}{t_{\text{max}}}\right)$$

  where t is the current time (e.g. global_step provided by `optimizer.minimize(loss, global_step = global_step)` and t_{max} is the total number of steps you are planning to do in all epochs. initial_lr is a tuning parameter.

Exercise 2 - Classify MNIST digits

Dataset We now turn to the classification of handwritten digits from the MNIST dataset that contains 55,000 training, 5,000 validation and 10,000 testing images. We will be creating a multilayer convolution neural network for this task.

CNN We provide a python script train_and_test_MNIST.py which does the following:

- Downloads the data from the Internet unless the data is already present in a directory called MNIST_data.
- Creates a neural network with two convolutional layers, each followed by a pooling and a non-linearity layer, which can be found in TwoLayerCNN.py
- Trains and evaluates the model using the training, development and test set.
  The training function we provided samples a mini-batch at each iteration, instead of a single data point. This is a common practice that consists in sampling a small set of datapoints which reduces the variance of the estimated gradient. This operation can usually be parallelized and thus made relatively cheap.

Familiarize yourself with the code and run it with the default parameters. This will run the code for 50 epochs and print the accuracy for the test set.

[1] You can set the `logdirPath` variable at the beginning of the file

Your task

- Visualization. The provided code creates scalar summaries for the variables and image summaries for the filters of the first layer. Run tensorboard and visualize the filters. Examine the graph and check how the loss and accuracy progress with training for more epochs.
- Creating summaries. Add an image summary for visualizing the filters of the second layer.
- Regularization. Modify the loss function such that you include ℓ_2 penalty on the norm of the weights and the bias for the fully connected layer (the variables are W_fc2 and b_fc2).
- Regularization. Add dropout at the penultimate layer. Make sure that the dropout keep probability is set to 1.0 during the validation and test step.

Exercise 3 - CNN Sentiment Classification

In this exercise we will build a text classifier system based on the sentiment. The task is to predict if a tweet message used to contain a positive :) or negative :(smiley, by considering only the remaining text.

Dataset We provide a large set of training tweets, one tweet per line. The dataset is accessible from

`https://inclass.kaggle.com/c/cls-text-classification/data`.

All tweets in the file train_pos.txt (and the train_pos_full.txt counterpart) used to have positive smileys, those of train_neg.txt used to have a negative smiley. Additionally, the file test_data.txt contains 10,000 tweets without any labels, each line numbered by the tweet-id.
In particular,

- train_pos.txt and train_neg.txt - are a small set of training tweets for each of the two classes. You can first test your code using these files to make sure everything is correct.
- train_pos_full.txt and train_neg_full.txt - are a complete set of training tweets for each of the two classes, about 1M tweets per class
- test_data.txt - is the test set, that is the tweets for which you have to predict the sentiment label
- sampleSubmission.csv - is a sample submission file in the correct format, note that each test tweet is numbered. (submission of predictions: -1 = negative prediction, 1 = positive prediction)

Your task is to predict the labels of the test tweets, and upload the predictions to Kaggle. Your submission file for the 10,000 tweets must be of the form <tweet-id>, <prediction> (see sampleSubmission.csv). Note that all tweets have already been pre-processed so that all words (tokens) are separated by a single whitespace. Also, the smileys (labels) have been removed.
In the previous exercise we created a two layer Convolutional Neural Network. We will now implement a model similar to Kim Yoon's Convolutional Neural Networks for Sentence Classification [1].

Model The model is shown in Figure 1. The first layer embeds words into low-dimensional vectors. The next layer performs convolutions over the embedded word vectors using multiple filter sizes. For example, sliding over 3, 4 or 5 words at a time. Next, we max-pool the result of the convolutional layer into a long feature vector, add dropout regularization, and classify the result using a softmax layer. We will use only one channel for the convolution.

Getting started We provide you with a starter code that loads the data which you can find in the train.py script. For this you need to unpack the twitter dataset in the twitter-dataset folder. After the data has been loaded, we pad all the tweets with a <PAD> token to the length of the longest sentence. This allows us to use mini-batches since all the tweets will be of the same size. Finally, we convert each word from every tweet to the corresponding index of the word in the vocabulary. Essentially, each tweet becomes a vector of integers.
The first (embedding) layer of the CNN receives as input a batch of tweets in such form, and replaces each index with a word embedding for the particular word. These word embeddings can be randomly initialized, but typically for NLP tasks it is better to set their initial value to some pretrained word embeddings. For this you can either use the 300-dimensional word2vec embeddings or the 50-dimensional GloVe embeddings (depending on your RAM limitations). You can download the word embeddings you would like to use from the Kaggle website and place them in the word-embeddings folder. In the sample scrip we have included a code that initializes the embedding matrix from the CNN. Make sure you have gensim installed (`https://radimrehurek.com/gensim/install.htm`).

Your task Your task is to complete the code for the CNN (in text_cnn.py) and implement the functions for training and evaluation on the development set (in train_cnn.py). The code for predicting on the test is provided to you (in predict_test.py). We have included a function create_submission_file in data_helper.py that given a list of the predictions and a file name creates a submission file in the required format.

- Add summaries for your variables, the loss and the accuracy and visualize them in tensorboard.
- Disable the training of the embedding matrix. This means that during the training the word embeddings will be fixed and should not be updated. See how the accuracy changes.
- Submit your predictions to Kaggle to get a misclassification error score. Recall that your submission file for the 10,000 tweets must be of the correct form, see sampleSubmission.csv. Try to tune your system for best evaluation score. You can also use a local separate validation set to get faster feedback on the accuracy of your system.

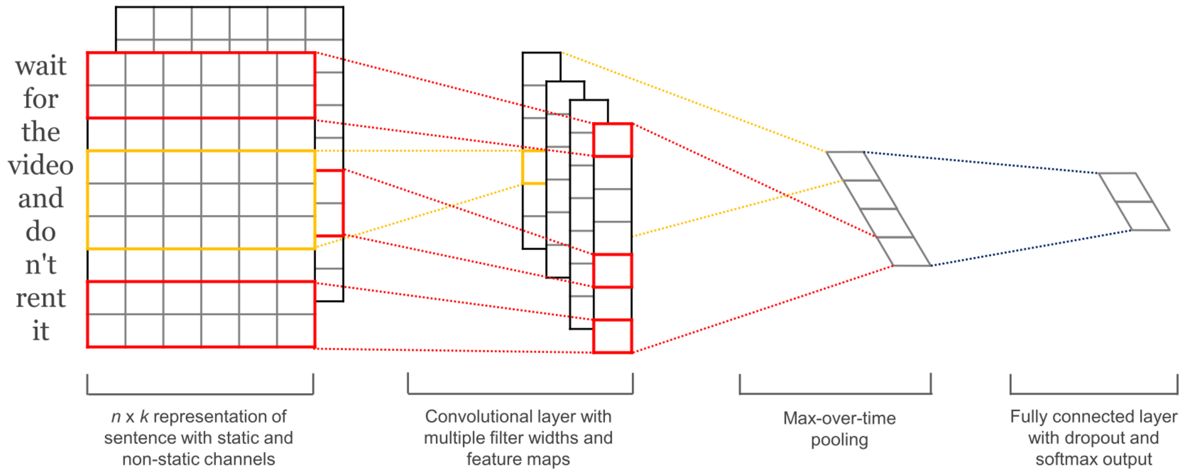
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: CNN model for sentiment classification

Exercise 4 - RNN Sentiment Classification

As shown in the slides, RNNs can solve the same sentiment classification task by iteratively building up a summary of the sentence. With your CNN implementation at hand, it's actually only a few lines of codes to implement the RNN.

Your task

- Take a look at `train_rnn.py`. Up to the construction of the `TextRNN` and the references to `rnn` it's the same as `train_cnn.py`. The implementation of the `TextRNN` in `text_rnn.py` differs from `TextCNN` only between the indicates lines. Use the information on the slides, to replace `placeholder` by a set of feature vectores generated by an RNN. To this end, use `BasicRNNCell` and `tensorflow.python.ops.rnn.rnn` as described.

  Note: The CNN approach expected the input as a single tensor of shape `[None, sequence_length, embedding_size]`. In contrast, the rnn expects a sequence (a python list) of tensors of shape `[None, embedding_size]`
- Follow the call of `rnn.rnn` and find the code section where the actual loop is happening.
- Open the graph in tensorboard and find the chained `BasicRNNCells` in the graph.
- Use `tf.nn.rnn_cell.MultiRNNCell` to create a multi-layer RNN. Does the performance increase?

References

[1] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.