Deep Learning for NLP

Assignment 1: Twitter sentiment classification using a feed-forward neural network with pre-trained word embeddings

originally from Sharid Loáiciga's edition of this course
(based on content from M. Silfverberg and H. Celikkanat)
with minor adapations by Philipp Sadler

1 Introduction

In this assignment, you will implement a sentiment classifier for tweets. You will get a set of tweets, some starter code, and a size-wise pruned version of Google's pre-trained embeddings.

2 Getting started

You should start by unpacking the archive `assignment1.zip`. This results in a directory `assignment1` with the following structure:

```
- data/
--- development.gold.txt
--- development.input.txt
--- test.gold.txt
--- test.input.txt
--- training.txt
-model/
--- GoogleNews-pruned2tweets.bin
- src/
--- data_semeval.py
--- assignment1.py
--- paths.py
```

The folder includes a pruned version of the Google's pretrained word embeddings, restricted only to the words in the dataset (17M as oppposed to 1.5G).

You should start by copying `src/assignment1.py` to a new name `src/assignment1_LASTNAME.py`. For example, Sharid will copy the file to `assignment1_LOAICIGA.py`. This is the file that you will edit and finally upload to Moodle after you are done with the assignment.

Try to run the Python program. It requires PyTorch, NumPy and NLTK (for tokenization). You should see the output below.

```
$ python assignment1.py
[nltk_data] Downloading package punkt to /Users/sharid/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
epoch: 0, loss: 0.0000
epoch: 1, loss: 0.0000
epoch: 2, loss: 0.0000
```

```
epoch: 3, loss: 0.0000
epoch: 4, loss: 0.0000
...
epoch: 28, loss: 0.0000
epoch: 29, loss: 0.0000
test accuracy: 41.85
```

The program will run 30 epochs (an epoch is a complete run over the entire training data) of the training algorithm and output accuracy on the test set. However, the implementation of the model does not do much yet, it simply returns neutral for all tweets. It is your task to actually implement the model, training and classification properly.

3 Twitter Data

You will be working with datasets from the 2014 SemEval shared task "Sentiment analysis in Twitter". If you are interested in background information or want to compare your own system to systems which participated in the competition, please have a look at:

S. Rosenthal, A. Ritter, P. Nakov and V. Stoyanov. *SemEval-2014 Task 9: Sentiment Analysis in Twitter.* SemEval 2014.

The provided code will take care of reading and writing data for your. However, if you are interested in the datasets, you can take a look at the `data` directory. Each data file consists of lines with three fields: a Twitter ID number, a sentiment class (positive, neutral or negative), and a tweet. Here are a few lines from `training.txt`:

```
101450548439560192 negative @stoney16 @JeffMossDSR I'd recommend just turning it off and waiting for Verlander tomorrow.
103158179306807296 positive RT @MNFootNg It's monday and Monday Night Football is on my mind. RT if you love football! #MNFootNg.
100259220338905089 neutral All Blue and White fam, we r meeting at Golden Corral for dinner tonight at 6pm....
104230318525001729 positive @DariusButler28   Have a great game agaist Tampa Bay tonight.
```

The files `development.input.txt` and `test.input.txt` do not show the correct sentiment label for the tweets. Instead, they have a dummy `unknown` label in the second field. If you want to see the correct labels, you can look at `development.gold.txt` and `test.gold.txt`.

The development dataset is usually used to fine-tune the hyperparameters of the model. The test set should be only used for the final evaluation run. If you ever target a benchmark dataset, then you will not have access to the test data at all. In this assignment, we allow you to simplify and to evaluate the model on the test set repeatedly.

4 Starter Code

Let's look at the main program in `assignment1.py`. It ...

1. reads in the Twitter sentiment datasets using `read_semeval_datasets()`,
2. loads the pre-trained embeddings using the Gensim library,
3. initializes a `Feed-Forwardd Neural Network` model, loss function and optimizer, and
4. trains the model for 30 epochs on the training data and then uses the model to classify the test data.

The function `read_semeval_dataset` returns a Python `dict` object `data` which has 5 keys:

- training – the training set.
- development.input – the unlabeled development set,
- development.gold – the development set with gold standard labels,
- test.input – the unlabeled test set,
- test.gold – the test set with gold standard labels.

Each of these, for example `data["training"]`, is a list of tweets which are themselves Python `dict` objects, for example:

```
{"ID": "264183816548130816",
"SENTIMENT": "positive",
"BODY": ["Gas", "by", "my", "house", "hit", "$", "3.39", "!", "!",
"!", "!", "I", ""m", "going", "to", "Chapel", "Hill", "on", "Sat", ".", ":)"]}
```

4.1 Loading pre-trained embeddings

The pre-trained word embedddings by Googled are loaded using the Gensim library[1]

```
#--- data loading ---
data = read_semeval_datasets(data_dir)
gensim_embeds = gensim.models.KeyedVectors.load_word2vec_format(
                os.path.join(model_dir, embeddings_file), binary=True)
pretrained_embeds = gensim_embeds.vectors

# To convert words in the input tweet to indices of the embeddings matrix:
word_to_idx = {word: i for i, word in enumerate(gensim_embeds.vocab.keys())}
```

Note that `pretrained_embeds` is a numpy matrix of shape (num_embeddings, embedding_dim). You must use this matrix in your FFNN model to initialize your embeddings layer. Furthermore, you will need to convert the tokens in the tweets into numbers, which you can use as indices to the embedding layer. For this, you need to use the `word_to_idx` dictionary. The index of a word given as returned by this dictionary gives the location of the word's embedding within the `pretrained_embeds`. (Note however, that you are free to decide where to do this token$\rightarrow$index conversion, e.g., within the model itself, or before invoking the model during the training/testing pass).

4.2 The `FFNN` class

A feed-forward neural network model for tweet classification is parametrized by the weights and biases of its neurons. E.g., for a feed-forward network with two hidden layers (notation is modified from Goldberg, p. 43):

[1] Gensim went through a major refactoring by version 3.3.0, and the code below is incompatible with the old versions.

$$\begin{aligned} \text{FFNN}(x; W^1, b^1, W^2, b^2, W^3, b^3) = y \\ h^1 = g^1(xW^1 + b^1) \\ h^2 = g^2(h^1W^2 + b^2) \\ y = g^3(h^2W^3 + b^3) \end{aligned} \tag{1}$$

where x is a tweet represented by the sum over its word embeddings, y is the output predicted by the network, W^n is the weight matrix of layer n, b^n is the bias vector of layer n, g^n is the activation function used in layer n, h^n denotes the output of hte n^th hidden layer. Note that the output layer does not have to emply an activation funtion, but in the case of a classification task, it makes things easier to use the `log_softmax/softmax` function.

The implementation of the the FFFNN in the skeleton code looks like this:

```
#--- model ---

class FFNN(nn.Module):
# Feel free to add whichever arguments you like here.
# Note that pretrained_embeds is a numpy matrix of shape (num_embeddings,
embedding_dim)
def __init__(self, pretrained_embeds, n_classes, extra_arg_1=None,
extra_arg_2=None):
super(FFNN, self).__init__():
# WRITE CODE HERE
pass

def forward(self, x):

    # WRITE CODE HERE
    pass
```

There are two methods in the `FFNN` class. The first method `__init__` is supposed to initialize the model parameters. For representing the hidden layer(s), you will need weight matrices and bias vectors associated with that hidden layer. The most straightforward way is again using `torch.nn.Linear`. Remember that `torch.nn.Linear` is basically $\mathbf{XW} + \mathbf{b}$ so that you do not have to implement matrix multiplication on your own.

Please use at least one hidden layer! We do not require a specific number of neurons in the hidden layer. It may be a good idea to start with a small number when developing your code, which will save some time in the initial experiments. Feel free with trying different numbers of hidden layers and neurons.

The concept of looking for "good" configurations is also called architecture-search or hyper-parameter optimization. A model with too much parameters or a too high capacity is likely to overfit the training data, whereas a model with too low capacity might not learn anything at all.

The second method forward computes the probabilities for our classes. In our case, the output y is a 1 $\times$ 3 tensor $(\log p_{NEG}, \log p_{NEU}, \log p_{POS})$. In this function, you can specify what kind of activation functions you would like your network to use for the hidden layers. You are welcome to play around with different activation functions. For g^3, since we are working on a classification task again, use the `log_softmax/softmax`.

4.3 Loss function and optimizer

Now that the `FFNN` class is working, we need to implement a loss function and initialize the optimizer. You can do this in the main part of the program. The current skeleton code looks like this:

```
#--- set up ---
# WRITE CODE HERE
model = FFNN(pretrained_embeds, n_classes)
loss_function = None
optimizer = None
```

You can add any extra arguments you would like to the FFNN class constructor. For example:

```
model = FFNN(pretrained_embeds, n_classes, n_hidden_1, n_hidden_2)
```

You should use the `torch.nn.NLLLoss` loss function, which will be appropriate given that forward in the `FFNN` class returns a tensor of logarithms of probabilitites.

Please use a plain SGD optimizer. Have a look into the `torch.optim` package or search in the torch docs for usage and default parameters.

4.4 Training

The current training algorithm in the skeleton code looks like this:

```
 #--- training ---
  for epoch in range(n_epochs):
    total_loss = 0
    for tweet in data['training']:
      gold_class = label_to_idx(tweet['SENTIMENT'])

      # WRITE CODE HERE
      None

    if ((epoch+1) % report_every) == 0:
      print('epoch: %d, loss: %.4f' %
            (epoch, total_loss/len(data['training'])))
```

The algorithm runs several epochs over the training data and trains on a single tweet at a time. You can see that the code has no validation epoch. Nevertheless, currently, the code does nothing at all.

Your task is to evaluate the loss for `tweet` given the current model parameters and the gold standard class `gold_class`. Then you need to perform a parameter update using the optimizer. Do not forget to start by zeroing the gradient of optimizer so that your update will correctly reflect the current training example. You should also update the total_loss which tracks the loss over the entire training set. Just add the instance to total_loss.

4.5 Testing

Finally, now that you are able to train your model, you should implement classification. The current code for testing the model looks like this:

```
#--- test ---
  correct = 0
  with torch.no_grad():
    for tweet in data['test.gold']:
      gold_class = label_to_idx(tweet['SENTIMENT'])

      # WRITE CODE HERE

      predicted = label_to_idx('neutral')
      correct += torch.eq(predicted,gold_class).item()

      if verbose:
        print('TEST DATA: %s, OUTPUT: %s, GOLD LABEL: %d' %
              (tweet['BODY'], tweet['SENTIMENT'], predicted))

    print('test accuracy: %.2f' %
          (100.0 * correct / len(data['test.gold'])))
```

Currently, this simply returns neutral for every tweet. It also evaluates the accuracy on the held-out test set.

It is your task to use the forward function in the FFNN class to set predicted to the maximally probable class for each test example.

You should be able to reach sentiment classification accuracy around 60-65% or higher, although depending on your exact settings the number can also be different. if you get a too low accuracy, do check if everything runs as expected, and that your network is being trained properly.

5 Submission

Upload your src/assignment1_LASTNAME.py file on Moodle.