Assignment 4 Neural Networks

In this assignment, we will experiment with feedforward neural networks on a German named entity recognition task using TensorFlow, one of the mainstream deep learning packages. This assignment does NOT involve heavy implementation. However, we do need patience for the experiments as training neural networks takes time.

0. Preparation

TensorFlow is already installed on our server. If you want to work with your own computer, please visit https://www.tensorflow.org/versions/r0.11/get_started/os_setup.html#pip-installation for more information on installation. Make sure that TensorFlow is correctly installed in your system. Start a python interactive session and type:

```python
>>> import tensorflow as tf
```

If it is imported normally, you can exit and continue.

Enter your working directory and download the skeleton code for the assignment via:

```bash
cd /your/working/directory
cp -r /local/kurs/ml/Assignment4 ./
cd Assignment4
```

In order to have some basic knowledge of the NER task, let us open the data file by:

```bash
less data/NER-de-dev.tsv
```

We can see that the sentences are separated by blank lines. The second column contains the tokens as our input. In this assignment, we only predict the NER tags in the third column. If you want to know more about the data set, check: https://sites.google.com/site/germeval2014ner/

Now, if you use Python 3, run:

```bash
python3 ner.py
```

If you use Python 2, run:

```bash
python ner.py
```

This will take quite a few minutes. Finally, you will get an F1-score of around 0.675 on the test set. To be fair, the accuracy is already somewhat reasonable. Nonetheless, we will start from this very basic model and incrementally improve it in this assignment.

Now open `ner.py` in your favourite python script editor (pyCharm, Vim, Emacs, etc), try to understand the structure and functionality of the script. We want to predict the NER label of a word by the concatenated embeddings of itself and its neighbouring words using a feedforward neural network.

If the code doesn’t make any sense to you, please read through the basic MNIST example at https://www.tensorflow.org/versions/r0.11/tutorials/mnist/beginners/index.html#mnist-for-ml-beginners.

1. Define Variables (10 p)

Let’s take a closer look at how the computational graph is defined in `ner.py`. In our current simple model, the concatenated word embeddings are directly fed into a softmax(as referred by the activation parameter) layer to generate the output. Now open `layers.py` and check how this DenseLayer is defined. Note that tf.matmul(a, b) multiplies matrix a by matrix b and produces a * b in the `__call__()` function.

Does it look familiar to you? Yes, it is very similar to linear perceptron without the bias. The difference is that the weights here is a matrix and the output is therefore a vector instead of a single number. We use
softmax as the activation function to normalise the multiplication product into a probability distribution with respect to the predicting tags. Now let’s recover it to the standard linear model by adding the bias.

First, define the bias variable using `tf.get_variable()`. More information about `tf.get_variable()` can be found at [https://www.tensorflow.org/versions/r0.10/api_docs/python/index.html](https://www.tensorflow.org/versions/r0.10/api_docs/python/index.html). You can also refer to the weights variable to see how it is defined, but please bear in mind that the shapes of the weights and the bias are different. Additionally, the bias should use `tf.constant_initializer(0.0)` as the initializer instead of `rand_uniform_init`.

In the `__call__()` function, we should add the defined bias to the multiplication product under the `if self.bias:` statement and comment out the `NotImplemented` exception.

Now go back to `ner.py` and change the bias parameter in the DenseLayer to True and run the script again. Does adding the bias somehow affect the accuracy?

2. Adding Layers (10 p)

Now, let’s try to build up our feedforward neural network by adding a hidden layer to the model. First, make sure you understand how the DenseLayer is defined and called. We put our hidden layer after the Embedding layer and before the softmax layer. Assign the activation parameter with ‘linear’ or leave it as None. The output dimension of the hidden layer should be equal to `hidden_layer_size`. Since all the layers are linearly stacked in our neural network, apart from the EmbeddingLayer, the output dimension of the previous layer is always equal to the input dimension of the current layer. We should then be able to figure out the input dimension of our hidden layer. We also need to change the input dimension of the softmax layer accordingly. It is also recommended to give the hidden layer a name through the name parameter so that we can easily keep track of it.

Now let’s run `ner.py` again. Is the accuracy significantly improved? Why or why not?

3. Non-linear Activations (10 p)

In order to learn non-linear decision boundaries and obtain further improvement, we need to employ non-linear activations such as hyperbolic tangent or sigmoid function. Read the descriptions of `tanh` and `sigmoid` functions at [https://www.tensorflow.org/versions/r0.10/api_docs/python/nn.html#activation-functions](https://www.tensorflow.org/versions/r0.10/api_docs/python/nn.html#activation-functions). We can choose one of the two as our non-linear activation. Now go back to `layers.py` and add your favoured non-linear activation in the `__init__()` of the DenseLayer. It should be added similarly as the softmax function.

Now change the activation function of the added hidden layer from linear to the name of the chosen non-linear function via the activation parameter. Run `ner.py` and check if the model performs better this time.

4. Word Embeddings (20 p)

Our neural network uses pre-trained word embeddings as the core feature. Open the embedding file under `embeddings` folder to see how it looks. Now, pick your favourite word in the embedding file and compare its vector representations with the other words’. Find its most similar word in terms of cosine similarity. Are they intuitively similar to you? Why or why not? (use a German dictionary or Google translator to look up the words if you do not know German)

For efficiently computing the cosine similarity, we can reuse the function `readEmbeddings()` in `reader.py` and store the vectors in numpy arrays. The cosine similarity between two vectors can be computed by:

```python
>>>from scipy import spatial
>>>similarity = 1 - spatial.distance.cosine(v1, v2)
```

Let’s change the weights parameter of the EmbeddingLayer to None so that the model will use random initialised embeddings instead of the pre-trained ones. Run `ner.py` and describe what you observe. Change the weights back to embeddings when the run is finished.
5. Optimisers (10 p)

The standard gradient descent optimiser tf.train.GradientDescentOptimizer() is so far used for all the experiments with the constant learning rate 0.5. Take a look at https://www.tensorflow.org/versions/r0.10/api_docs/python/train.html#optimizers for some other optimisers. Choose one additional optimiser from the list and substitute tf.train.GradientDescentOptimizer() and test to see if it works differently in terms of accuracy and training time. For some optimisers, probably a smaller learning rate is required. If you use tf.train.AdamOptimizer() or tf.train.AdadeltaOptimizer(), leave the learning rates as default and do not pass any value to the learning rate parameter.

You may notice that some optimisers make the training time significantly longer. For efficiency, let’s change our optimiser back to tf.train.GradientDescentOptimizer() for the following experiments.

6. Dropout (15 p)

First, let’s change number_of_epochs to 100 so that the model will run for more iterations. Apart from the F1-score on the development set, this time we also inspect the accuracy on the training set after each epoch (this is already implemented in the code, just delete the commend mark #). How do the F1-scores on the development set and training set change after each epoch? Do they change differently? Try to explain the reason behind and why using a development set for validation is important.

Dropout is a commonly used regularisation technique. There is a dropout wrapper implemented for you in layers.py. Figure out how it works and apply it to your hidden layer. Dropout is only applied at the training stage while it should be turned off when decoding. The best way to integrate dropout in the model is to define a tf.placeholder() and add it to batch.train() and batch.predict() along with the dropout value if needed.

Try different dropout rates and see how they affect the training of our model.

7. Hyper-Parameter Tuning (20 p)

There are many hyper-parameters that influence the performance of our neural network. Try to play around with them by:

- Changing window_size
- Changing hidden_layer_size
- Adding more hidden layers
- Changing batch_size
- Changing learning_rate
- Changing dropout rate
- Running for more epochs

First, change the parameters individually and then try to find the best combination. What F1-score does your best model achieve? Try to interpret why the corresponding configuration works better than the others. Save your best model in ner.py. Briefly introduce the parameter tuning procedure.

8. Further Development (5 p)

The state-of-the-art accuracy on the test data is around 0.775 (do not try to beat it within this assignment) in terms of F1-score. Review the lecture notes, reading materials and summarise your experimental results as well as read some relevant papers if you think necessary. Discuss what else can be done to obtain further improvements to the best of your knowledge.

Submission

You should submit a short written report (3-5 pages) together with your modified ner.py and layers.py. Try to solve all the problems and answer the questions in your report.

You need at least 70 points to pass this assignment and 90 points to pass with distinction (VG). Please send all the required materials in a packed zip file with your identification to yan.shao@lingfil.uu.se before January 13, 2017.