Advanced Natural Language Processing

Assignment 2

Tejasvi Chebrolu

2019114005

—

Implementing Embedding from Language Model

For implementing `ELMO` from scratch we need to understand the architecture of the language model. The architecture of the language model is the same as the TensorFlow implementation of the pretrained biLM used to compute ELMo representations. The model was further trained on the Switchboard dataset.

The command to fine-tune the model further is as follows:

```
python bin/run_test.py --test_prefix='swb/dev/*' --vocab_file 'swb/vocab.txt' --save_dir='swb/checkpoint'
```

To run the predictions and get the embeddings, run the Jupyter notebook `Assignment_2.ipynb`. The embeddings are saved in `embeddings.txt` and are of the form `word vector`. The embeddings were generated by the following code:

```
with open('embeddings.txt', 'w') as f:
   f.write('{} {}\n'.format(len(vocabulary), len(weights[1])))
   for i in range(len(vocabulary)):
    f.write('{}'.format(vocabulary[i]))
    for weight in weights[i]:
      f.write(' {}'.format(weight))
    f.write('\n')
```

PreProcessing

The data was preprocessed in the file `trial.py`. The training data was randomly split into many training files, each containing one slice of the data. Each file contains pre-tokenized and white space separated text, one sentence per line. White space separated full stops were also added. The training required multiple files with one text sentence per line, we created 20K training files by writing 6 sentences per file. The vocabulary file is a text file with one token per line. It must also include the special tokens `<S>`, `</S>` and `<UNK>` (case sensitive) in the file. This was also done.

Hyperparameters

- `batch_size` : The batch size for the training was set to 128.
- `num_epochs` : The number of epochs was set to 10.
- `vocabulary` : The vocabular was set to 19558.
- `embedding_size` : The embedding size was set to 1024.
- `activation_function` : The activation function was set to `relu`.

Calculating Cosine and Euclidean Distances

The embeddings were chosen at random using the `random` function. The pairs selected were:

- (we, one)
- (but, and)
- (that, right)
- (yes, we)
- (exactly, him)

> The Euclidean distance is calculated as follows:

```
def euclidean(a, b):
  return math.sqrt(sum((a[i] - b[i]) ** 2 for i in range(len(a))))
```

> The Cosine distance is calculated as follows:

```
 from sklearn.metrics.pairwise import cosine_similarity
def cosine(a, b):
  return cosine_similarity([a], [b])[0][0]
```

Distances

Pairs	Euclidean Distances	Cosine Devices
(we, one)	16.423919642809537	0.22874095
(but, and)	16.191433749407427	0.27082092
(that, right)	16.69056389253577	0.2051402
(yes, we)	15.753354185592876	0.28037643
(exactly, him)	17.274845122675785	0.16618866

References

- Model
- Blog
- Docs

—
