# Assignment 5

Marius Seritan

February 19, 2019

## 1 Character-based convolutional encoder for NMT

(a) Explain one reason why the embedding size used for character-level embeddings is typically lower than that used for word embeddings

When using character-level embeddings the size of the vocabulary is smaller than the size of the word-level vocabulary and multiple character embeddings are combined together to build a word. As such the size of each embedding can be smaller.

(b) Total number of parameters

For character-level embeddings the number of parameters is

$$N_{char} = V_{char} * e_{char} + e_{word} * e_{char} * k + e_{word} + 2 * (e_{word} * e_{word} + e_{word})$$

For word-level embeddings the number of parameters is

$$N_{word} = V_{word} * e_{word}$$

For $k = 5$, $V_{word} = 50000$, $V_{char} = 96$, $e_{word} = 256$ and $e_{char} = 50$

$$N_{char} = 200384$$

$$N_{word} = 50000 * e_{word} = 12800000$$

The ratio is

$$\frac{N_{word}}{N_{char}} = 63.87$$

For this specific use based there are $\approx 64$ times as many parameters in the word embeddings as in the character embeddings.

(c) One advantage of using a convolutional architecture rather than a recurrent architecture for the character based embedding models

When a 1d convnet computes features from character embeddings the features are given the same importance no matter where the original characters are placed in the world. This allows the convnet to focus on subwords/n-grams and learn the most relevant sections of the word. In contrast an RNNs always look at the whole world which makes it hard to discard the prefix, for example. Therefore for longer words convnets are more likely to extra the more relevant sub-word which could be specially for languages with joined words.

(d) Compare max-pooling and average pooling

Both pooling layers are generally applied after convolution and allow to reduce the variance and size of the data. For each individual block we pick just one value through pooling. For a given application one pooling method may work better.

In the case of the character based encoding we first use character embeddings to which we apply a convolution and then take the max-pool. The benefits of max-pooling in this approach is that the learned word embedding will focus on the subword that is the most relevant for describing the given word. Max-pooling also beneficial for same reasons when processing images.

One of the benefits of average pooling is that is keeps more information about the overall block. I would consider using average pooling in later parts of an content understanding pipeline, for example to identify which sentence of a paragraph may be more relevant to answer a query.

(h) Testing the highway implementation

In order to test my highway implementation I wrote the `unittest` based `test/highway_test.py`. This test class uses a helper method in the `Highway` instance to set the weights to know values.

- I validated the projection path by setting gate weights to 10.0 - when passed through the `sigmoid` this will set the gate to $\approx 1.0$ and disable the skip connection. We used two value for the linear projection weights to 0.3 and 20.0, without bias. We validated that when passing 1.0 as input the output is equal to the projection parameter.
- I validated the skip connection by keeping the projection path as above and setting the gate weights to $-10$. The expected output in this case is 1.0 (equal to the input). Note that we had to reduce the precision from $1e-5$ to $1e-3$.
- We tested training of part of the network. For this we adapted the `polynomial` subproject from the pytorch examples to validate that I can train part of the network - removed the `relu` layer and setup the gate to positive numbers as above. See the `run-highway.py` file. The output is

```
==> Actual function: y = +10.86 x^4 +5.07 x^3 -3.35 x^2 -3.40 x^1 -7.35
==> Learned projection: y = +10.93 x^4 +5.11 x^3 -3.37 x^2 -3.40 x^1 -7.42
==> Learned gate: y = +1.37 x^4 +4.08 x^3 +2.22 x^2 +3.64 x^1 +4.41
```

The one interesting fact is that the gate parameters remained relatively high after the training, otherwise comparing the learned function with the actual one would have been a bit useless.

I believe that the tests above are sufficient because we are only interested to test the highway specific code. More specifically the highway adds a skip connection and the first tests above test the operation with and without this connection. Testing with more complex input would only test `pytorch` specific code. The additional training use case is an exploration to allow me to get familiar with testing strategies for DNNs.

(i) Testing the cnn implementation

In order to test my cnn implementation I wrote the `unittest` based `test/cnn_test.py`. To enable this I wrote a helper method in the `Cnn` instance to set the weights to know values.

The test compares the implemention against manual testing.

# 2 Character-based LSTM decoder for NMT

BLEU score:

## 3 Analyzing NMT Systems