# Assignment 5

Marius Seritan

February 23, 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 the number of letters, so the overall size will be 100 or less for english. The word word-level vocabulary has to be significantly larger (50k or more) and as such the size of the word embeddings has to be larger.

(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 word. This allows the convnet to focus on subwords/n-grams and learn the most relevant sections of the word.

In contrast an RNNs the unit of processing is the whole world. This makes it hard to discard the prefix, for example.

Therefore for longer words convnets are more likely to extract the more relevant sub-word. This is specially important for languages with joined words (german).

(d) Compare max-pooling and average pooling

Both pooling layers are 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 than the other.

When applied to character based encoding we apply max-pool after the convolution of character embeddings. 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 a 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 tests in `test/highway_test.py`. To faciliate reproducible tests I added 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 implementation against manual testing:

- using a CNN with 1 input, 1 output and a kernel size of 2
- initialize the weights to 1.0 and bias to 0.0
- on input $1, 2, 3$ the output should be $3, 5$

## 2 Character-based LSTM decoder for NMT

BLEU score:

# 3 Analyzing NMT Systems

(a) Verb conjugation in spanish

We examine `vocab.json` to figure out which of the forms of `traducir` verb are part of the vocabulary. See results below:

| Spanish form | Present |
|---|---|
| traducir | yes |
| traduzco | no |
| traduces | no |
| traduce | yes |
| traduzca | no |
| traduzcas | no |

As we can see 2/3 of the inputs are missing.

For a word-based NMT system this is a bad situation because the generated output will contain `<UNK>` words for most of the conjugations of this particular verb. This is likely to apply to all verbs unless we increase the size of the vocabulary many times over.

This situation is improved a lot with character-based models. A character-based NMT will learn embeddings for all the words that are present in the training set. Based on these embeddings the NMT will be able to generate output for both words that are in the training set and also generalize to other words that it has not seen before. The number of unknown words will decrease a lot.

Note: before taking this class I was not aware of the fact that character embedding perform so well in practice. Many thanks for making me aware of their benefits

(b) Word embeddings

(i) Nearest neighbors for Word2Vec 10k

| Center word | Nearest | Close by |
|---|---|---|
| financial | economic | business, markets, banking |
| neuron | nerve | neural, cells, brain |
| Francisco | san | jose, diego, antonio |
| naturally | occurring | readily, humans, arise |
| expectation | norms | assumptions, policies, inflation |

(ii) Nearest neighbors for CharCNN

| Center word | Nearest | Close by |
|---|---|---|
| financial | vertical | informal, physical, cultural |
| neuron | Newton | George, NBA, Delhi, person |
| Francisco | France | platform, tissue, Foundation |
| naturally | practically | typically, significantly, mentally |
| expectation | exception | indication, integration, separation |

(iii) Compare

Based on the data above Word2Vec models relationships between words where these relationships can be the domain (for `financial` and `neuron`), geographical (for francisco), synonyms (`expectation` and `norms`) or more general word co-occurrence (`naturally occurring` is a frequent expression).

The CharCNN relationships are a lot more about words having the same shape than the relationships between them. In 3 of the 5 examples above the grouping is done based on prefix (end of the shape). In English this groups together adjectives (`financial`, `vertical`), adverbs (`naturally`, `practically`) or nouns (`expectation` and `exception`) but this grouping superficial - it is not derived from any grammatical knowledge, just shape similarity. The other two groupings seem to be less useful.

The differences above can be explained by the training methodology. In the case of Word2Vec we predict how a word will occur based on its context in a sentence and this exposes more complex relationships. In the case of CharCNN the focus is on individual words and the patterns we learn are related to the inner structure.

(c) Analyze outputs