CS 224n Assignment #5

Yoshihiro Kumazawa

July 18, 2020

1. Character-based convolutional encoder for NMT

(a) Convolutional architectures can operate over variable length input too since convolutional layers slide fixed-sized windows over input unlike linear layers.

(b) The size of the padding should be 1 so that the padded vector will have size at least 5. Indeed, m_{word} could be 1 if all words in a batch happen to be some characters of length 1 like 'a', in which case we have $\mathbf{x}'_{\text{padded}} \in \mathbb{Z}^3$.

(c) The highway layer makes it possible to combine local features and global features. In other words, it matches our intuition that we can sometimes understand the meaning of a word by just looking at a little chunk of consecutive characters at a time but it is sometimes better to consider the whole characters in it at once. In order to simplify the network semantics in the beginning of training, I would initialize $\boldsymbol{b}_{\text{gate}}$ to be negative.

(d) Transformers are easier to parallelize and faster to train.

(e) See `vocab.py`.

(f) For the highway network implementation, see `highway.py`. I added a function `question_1f_sanity_check()` in `sanity_check.py` to test the following expected properties.

- The output size is correct for a given input.
- $\boldsymbol{x}_{\text{highway}} = \boldsymbol{x}_{\text{conv\_out}}$ when $\boldsymbol{x}_{\text{gate}} = 0$, which is checked by making $\boldsymbol{b}_{\text{gate}} = -\infty$.
- $\boldsymbol{x}_{\text{highway}} = \boldsymbol{x}_{\text{proj}}$ when $\boldsymbol{x}_{\text{gate}} = 1$, which is checked by making $\boldsymbol{b}_{\text{gate}} = \infty$.
- $\boldsymbol{x}_{\text{highway}} = \boldsymbol{x}_{\text{conv\_out}}$ when the projection layer is the identity function.

In addition, I checked if $\boldsymbol{x}_{\text{gate}}$ is initialized to be negative by computing the mean 4 times. (see my answer for 1 (c) above).

(g) For the convolutional network implementation, see `cnn.py`. I added a function `question_1g_sanity_check()` in `sanity_check.py` to test the following expected properties.

- The sizes of input channels, output channels, kernels and padding of the convolutional layer are correct.
- The output size is correct for a given input.

(h) See `model_embeddings.py`. I do not provide any additional test for it.

(i) See `nmt_model.py`.

(j) See `outputs/test_outputs_local_q1.txt`. The BLEU score is 99.67.

2. Character-based LSTM decoder for NMT

(a) See `char_decoder.py`.

(b) See `char_decoder.py`.

(c) See `char_decoder.py`.

(d) See `outputs/test_outputs_local_q2.txt`. The BLEU score is 40.92.

(e) See `outputs/test_outputs.txt`. The BLEU score is 34.99. So would I get 0 point here? So sad...

3. Analyzing NMT Systems

(a) Traducir, traduzco and traduce are in `vocab.json`, while traduces, traduzca and traduzcas are not. This is bad for word-based NMT models from Spanish to English because they will not directly translate such words, which contain not only the essential meanings but also information about subjects or tense. Our character-based model will possibly overcome the problem since it is based on hidden states of the LSTM, which expectedly have some information about the right words to produce. Moreover, as in the above example of Spanish words some of which are in the vocabulary and others are not, words that mean similar things tend to be made of similar characters, where our model would conceivably work well.

(b) i. Below are the (word → closest neighbor) pairs.
- financial → economic
- neuron → nerve
- Francisco → san
- naturally → occurring
- expectation → norms

ii. Below are the (word → closest neighbor) pairs.
- financial → vertical
- neuron → Newton
- Francisco → France
- naturally → practically
- expectation → exception

iii. The CharCNN models similarity in spelling whereas Word2Vec models semantic similarity. This is due to how those models are trained. With the CharCNN, similarly spelled words will have similar representations after convolutions, whereas semantically similar words will be embedded closely with Word2Vec since it tries to model words by nearby words they are placed with in sentences.

(c)