CS 224n Assignment #5

Jai Gupta

February 22, 2019

Q (1.a): Entries arranged in an embedding space are supposed to be far apart if there exists no relation (of some predefined form – usually semantic relationships), and nearby if there is. As we increase the number of dimensions, we allow more space for entries to be able to correctly position themselves in the embedding space. If the number of dimensions are low, then the words might be forced to depict interactions/relations that they don't really have.

The number of characters to arrange in the embedding space is much smaller than the number of words, and hence can easily depict the needed relations/interactions in a much smaller number of dimensions.

Q (1.b): Parameters in character based embedding model are:

- Embedding table: $V_{char} * e_{char}$
- Convolutional NN (taking $f = e_{word}$): $e_{word} * (k * e_{char} + 1)$
- Highway Network: 2 linear transformation of gate and proj $\rightarrow 2 * (e_{word}^2 + e_{word})$

Hence, total number of parameters = $V_{char} e_{char} + e_{word}(k e_{char} + 1) + 2(e_{word}^2 + e_{word})$

Parameters in word based embedding model are only due to the embedding table: $V_{word} * e_{word}$

For our case, the number of parameters in:

- char based embedding model: $96 * 50 + 256 * (5 * 50 + 1) + 2(256^2 + 256) = 200640$
- word based embedding model: $50000 * 256 = 12800000$

The number of parameters in word based embedding model is greater and approximately 64 times that of character based embedding model.

Note that we ignore all hyperparameters (like dropout) during these calculations.

Q (1.c): When using a CNN, the model is able to look at many characters at the same time. In case of using RNN, a combined hidden state produced after processing the surrounding characters is used. It is possible to increase the size of the hidden state of the RNN so that it can encode sufficient information from each state, followed by a point-wise feed forward network

to get the same effect, but that does increase the number of parameters required by the model significantly.
Further, CNNs can only look at a fixed window, while RNNs do have a choice to remember information from longer term.

Q (1.d): Max-pooling: Works great when operating with activations (or other similar features) where the existence of a feature in a huge population is required. For example, in the case of char-embedding, we are interested in whether a given activation was found in output of any of the sweeping windows of the conv-nn. Similarly, in case of convolution over images, we are usually looking for the presence of a pattern anywhere in the max-pool window. In contrast, if we used mean pooling, it will look for average value of the feature which is not what we want.

Mean-pooling: Max pooling loses a lot of information about the presence of low activations. It focuses on just the input with max value and hence is also sensitive to presence of outliers. Mean-pooling on the other hand keeps a mean of the complete population, and hence produces a much smoother output. For examples, the output required to be produced is a smoothened out version of the input (e.g. an image), mean-pooling would work better.

Q (1.h): The following tests were performed to test the implementation in highway.py. Significant hint for the tests were taken from this assignment's existing sanity_check.py file and the test files from previous assignments.

- Firstly, added documentation for the class and each function clearly defining the params and the return value.
- Added a function in a new test file 'sanity_check_extra.py' to add tests for this file. The test can be run using the command $python sanity\_check\_extra.py 1h$
- Created a new instance of Highway module with dim = 3.
- Verified that the projection nn.Linear instance 1) is not None, 2) is an instance of nn.Linear type, 3) has weights with shape (dim, dim), and 4) has bias of shape (dim,)
- Initialized the weight and bias weights filling it with 0.3 using torch.Tensor.data.fill_(args) method. I have also tried using randomly generated weights. randomly generated weights are potentially more robust, but using a constant of 0.3 makes the intermediate values easier to inspect.
- Created a sample input for testing the highway module of shape (3, 4, 3). The values were manually curated to include 0, negative and positive values.
- Expected output was calculated using numpy.
- Passed the input through the highway network to get the result.
- Used np.allclose(...) to verify that the output is close to the expected values.
- I also added print statements to verify that intermediate shapes look good, and also modified the test to pass ones and zeros (which made the output easy to inspect) to validate manually before writing the above test.
- I also add assert statements in the original code itself during development to validate that the shape of each output and input matches the expectations. These assert statements are very cheap and a lot of robustness to the code. I have commented them out from the code to ensure it does not hinder the auto-grader.
- Also validated failures by passing tensors of shape different than what is expected by the module. Passed None, empty tensor, etc to validate that we get proper error messages.

At this point, I have validated all the inputs, and verified the shape of the output after each step. I also validated the output after generating inputs of zeros, ones and a manually curated one.

Hence, I think this is sufficiently tested.

Q (1.i): A major fraction of the steps performed for testing CNN module are same as Highway too:

- Added documentation for inputs and output
- New tests can be executed using $python sanity\_check\_extra.py 1h$
- Created an instance of CNN module and verified that it had a Conv1d initialized in its children using cnn.children()
- initialized the weights of cnn module's all children recursively.
- Tested the module on inputs of size (2, 3, 5, 3) where the values were initialized to 1) manually with negative, zero and positive entries, 2) all zeros, and 3) all ones.
- Additionally, tested the failures on invalid input sizes and verified the failures due to assert added in code itself. As in the previous case, these asserts were commented out to make sure it does not interfere with the autograder.

Similar to the question above, I have validated all the inputs, and verified the shape of the output after each step. I also validated the output after generating inputs of zeros, ones and a manually curated one. Hence, I think this is sufficiently tested.

Q (2.f): BLEU Score: 24.175

Q (3.a): Forms that are present in vocab.py:

- traducir
- traduce

Forms that are missing from vocab.py:

- traduzco
- traduces
- traduzca
- traduzcas

All missing forms of the verb will be translated to <unk> token in word based embedding, causing the Spanish to English translations to perform badly.

The character aware model can learn about inflections in verb and their semantic meaning by operating on a large number of words with similar pattern. It can internally learn a representation to check for patterns at the end of the word (or elsewhere) and understand the semantic meaning associated with the inflection or other types of patterns.

Q (3.b.i): Nearest words found from "Word2Vec All". "Word2Vec 10k" was missing words below, hence I am using "Word2Vec All" instead.

- financial → economic
- neuron → neurons
- Francisco → san (note that Word2Vec is uncased, so the result is for "francisco")
- naturally → occurring
- expectation → operator

Q (3.b.ii): Nearest words found from our char-embedding model.

- financial → vertical
- neuron → Newton
- Francisco → France
- naturally → practically
- expectation → exception

Q (3.b.iii): Word2Vec focuses on finding semantic similarity between words. Since the training objective is to guess nearby words from a context word (or vice-versa), the word embedding model tries to put words with similar meaning nearby in the embedding space. This is explained by the fact that words with similar meaning usually have similar neighbors.
In case of char-embedding model, we are using CNN to extract features that try to look for patterns within the window they operate on. Hence, we are essentially training our embedding model to group words with similar patterns together in the embedding space.

Q (3.c): Acceptable substitution:

- *Spanish*: Qu les parece pensar en insistir para que las familias puedan elegir entre varias ciudades que estn compitiendo para atraer a nuevos residentes?
- *Reference*: What kind of an idea is it to think about insisting that every family have a choice of several cities that are competing to attract new residents?
- *A4*: What seems to you to think about <unk> so that families can choose between various cities that are competing for attracting new <unk>
- *A5*: What does it look like to think about insisting so that families can choose between various cities that are competing to attract new international?
- *Acceptable*: The character based model had learned the semantic meaning of the the pattern 'insist-' from various other inflections of the same word present in the training set. Additionally, it must have also learned about the semantic meaning of the patter '-ir' from the training set. Recognizing these two patterns help the character based model to correctly identify the translation for the word 'insistir'.

Incorrect substitution:

- *Spanish*: Un amigo mo hizo eso – Richard Bollingbroke.
- *Reference*: friend of mine did that – Richard Bollingbroke.
- *A4*: friend of mine did that – Richard <unk>.
- *A5*: A friend of mine did that – Richard Brooklyn.
- *Not Acceptable*: At the core, our character based embedding model is still doing pattern matching. Hence, for certain words like proper nouns, the semantic relationships are hard to capture for the character based embedding. In such cases, pattern matching is only used. Still certain amount of semantic context is still preserved in the above substitution as a proper noun was replaced with another proper noun.