CS224n Assignment 5

SUNet ID: maxspero
Name: Max Spero

1. Character-based convolutional encoder for NMT

Written Questions

a. The size of character embeddings are typically smaller than the size of word embeddings simply because characters carry less information than words: the vocabulary for characters is extremely limited compared to words, so there is not too much nuanced meaning to convey in a single character compared to a word. It would also be hugely inefficient to create character embeddings the size of word embeddings since there are many more characters in a sentence than there are words in a sentence – training anything would take much longer.

b. A lookup-based word embedding model must contain $V_{word} \times e_{word}$ parameters.

The character-based embedding model contains $V_{char} \times e_{char}$ parameters for the embedding lookup, $e_{char} \times k \times e_{word} + e_{word}$ parameters in the conv1D, and $2 \times e_{word} \times e_{word} + 2 \times e_{word}$ parameters in the highway layer. In total, that is:

$$(V_{char} \times e_{char}) + (e_{char} \times k \times e_{word}) + (2 \times e_{word} \times e_{word}) + (3 \times e_{word})$$

parameters.

Given $k = 5$, $V_{word} = 50,000$, $V_{char} = 96$, $e_{word} = 256$, and $e_{char} = 50$, the lookup-based word embedding model has roughly $12,800,000$ parameters and our character-based model has roughly $96 \times 50 + 50 \times 5 \times 256 + 2 \times 256 \times 256 + 3 \times 256 = 200,640$ which is roughly $1 \sim 2\%$ of the size of the original word embedding model. Tiny!

c. Because an RNN in this case would compute on character embeddings one at a time, either front to back or bidirectionally, characters on the ends will have more weight on the final output/hidden state than characters in the middle (this is especially true for long sequences). In contrast, correctly-padded and strided convolutions by nature give equal weighting to every cell that they compute on. This is an advantage for CNNs in sequences like characters where information could otherwise be lost in RNNs on long words.

d. Max pooling will reduce a tensor along one dimension, outputting the maximum values across that dimension, while average pooling will do the same reduction but output the average values across a dimension.

One advantage to average pooling is that highly negative values are factored into the final result. This can be important because a highly negative value could be much

more relevant to the final output than a slightly positive value, which is what would get chosen in max pooling.

One advantage of max pooling is the way gradients pass through it. Since only the maximum value is chosen, gradient only flows through the max pool layer to the cell that contained the maximum value. This is generally a good thing for neural networks because it allows the system to tune one value that affected output without tuning the less relevant ones (closer to 0 or negative), similar to how ReLU will not pass gradient through for values less than 0, allowing us to leave irrelevant weights untouched. This is opposed to average pooling, where gradient passes through all inputs regardless of how relevant they actually were to the output, making changes to these weights regardless.

Implementation

h. I have tested my code and I believe it is correct. I did several tests:

(a) I wrote docstrings for all my functions that included shapes of inputs and outputs and how the parameters of __init__ affect it.

(b) I wrote a file called test_highway.py to test my code: in it I specified a batch size and input size and sent a tensor of ones through it.

(c) I printed all shapes of the intermediate and final tensors, and they checked out.

(d) I printed all values of the intermediate and final tensors, and they also checked out. They were consistent in definitions: $x_{proj} \geq 0$ because it has a relu and $0 \leq x_{gate} \leq 1$ because it is the output of a sigmoid function.

(e) I also checked the computation of the output tensor compared to the intermediate tensors and the math checked out.

I believe this is sufficient because every line of code I wrote is tested in at least two ways and I have a strong understanding of what my code does and what output I expect, and the code meets that expectation.

i. I have tested my code and I believe it is correct. I did several tests:

(a) I wrote docstrings for all my functions that included shapes of inputs and outputs and how the parameters of __init__ affect it.

(b) I wrote a file called test_highway.py to test my code: in it I specified a batch size and e_{char} size and m_{word} size, and sent a tensor of ones through it.

(c) I printed all shapes of the intermediate and final tensors, and they checked out exactly with what I should expect from the equations.

(d) I examined the sizes of the weight matrices and they looked correct.

(e) I printed all values of the intermediate and final tensors, and they also checked out. They were consistent in successfully performing a max pool and not having any values below 0 (because of the ReLU).

I believe this is sufficient because every line of code I wrote is tested in at multiple ways and I have a strong understanding of what my code does, and the code matches that in tests.