CS224n Winter 2019 Homework 4

SUNet ID: 05794739
Name: Luis Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Problem 1

(a) The embedding must capture the essence of the item being embedded, and this is reflected by the dimensionality of the embedding – in other words, the higher the information content (number of unique items), the larger the embedding dimesion should be. Considering that a typical vocabulary size of words, $|\mathcal{V}|$, can be in the thousands or hundreds of thousands (with the number of possible words far higher), while the size of the character set for most languages, $|\mathcal{C}|$, is typically a few orders of magnitude smaller (in the hundreds), it is reasonable that a character embedding of 50 suffices.

(b) The number of parameters for the word-based lookup embedding model is trivial to compute (where we treat the embedding itself as trainable). We have the number of parameters as:

$$V_{\text{word}} \times e_{\text{word}} = 12.8M$$

The number of parameters for the character-based embedding model is a little more involved to compute, but can nonetheless still be done. We have:

$$\begin{aligned} V_{\text{char}} \times e_{\text{char}} &= 4,800 && \text{(Character Embedding Parameters)} \\ e_{\text{word}} \times e_{\text{char}} \times k + e_{\text{word}} &= 64,256 && \text{(Convolution Parameters)} \\ 2 \times [e_{\text{word}} \times e_{\text{word}} + e_{\text{word}}] &= 131,584 && \text{(Highway Network Parameters)} \end{aligned}$$

This gives a final expression for the number of parameters as:

$$V_{\text{char}} \times e_{\text{char}} + e_{\text{word}} \times e_{\text{char}} \times k + e_{\text{word}} + 2 \times [e_{\text{word}} \times e_{\text{word}} + e_{\text{word}}] = 200,640$$

From the above calculations, it is clear that the word-embedding model has more parameters, by a factor of 64 (almost two orders of magnitude).

(c) The convolutional architecture computes a set of features over the same window, meaning that each filter can learn to detect different patterns, no matter where they occur in the input word (translationally invariant). By contrast, typical RNNs are more sensitive to the absolute position of particular features, given their left (or left-right for bidirectional) processing. This makes CNNs more useful for longer sequences where interaction are local (rather than non-local), and where absolute positions are not as relevant.

(d)
- When data goes through an average-pooling layer, the average over the input is taken, which leads to smoothing of the input features over the window. This means that no data is lost from the input, since every input point contributes to the average. This is an advantage in comparsion to max-pooling, where a signficant portion of the input is completely ignored.
- When data goes through a max-pooling layer, a single value is selected (the maximum) over the window for each input layer. This leads to a max-pooling layer sending stronger gradient signals to the selected input (the entirety of the loss gradient flows through the selected value), and can help with extracting distinct/orthoganal features from the data.

(e) In "vocab.py".

(f) In "utils.py".

(g) In "vocab.py".

(h) Coding in "highway.py". These are the following checks that took place. Please see associated code file for details.

- Testing began by simply checking that the dimensions of the output from the highway network would be as expected. In this scenario, we do not check any of the weights or the data, simply set-up a random input with given dimensions, and verify the output was the expected dimension. Specifically, the input should be $(batch\_size, embed\_size)$ and the output $(batch\_size, embed\_size)$.
- We checked the dimensions of all intermediate computations. In particular, we set-up the Highway module so that it stores the dimensions of the projection, the gate, and the highway output. We verify these dimensions match what we wanted to compute.
- We also checked using a small example of $(batch\_size, embed\_size)$ where $batch\_size = 1$ and $embed\_size = 3$.
- We added tests for a few edge-cases. We verified that the correct values were computed (for small-cases) consisting of edge-case dimensions. In particular, when the batch_size is 1 and not 1, when the embed_size is 1 and not 1, and when the batch_size and embed_size are the same.
- Finally, we added more complex cases (randomly generated input of random generated size) where the gate output was all 0, meaning that the input and output should exactly match. For these cases, we disabled dropout.

(i) We did similar checks as above – checking dimensions. In particular, the input dimensions are $(batch\_size, char\_embed\_size, max\_word\_length)$ and the output dimensions are $(batch\_size, word\_embed\_size)$. The testing actually caught a bug where we were misuing the 'squeeze()' function (not passing a dim= parameter) when we tested with

all equal dimensions. For the CNN, we did not add any randomly generated test-cases, as this proved too much work.

(j) In "model_embeddings.py". Similar checks as before.

(k) In "nmt_model.py". Similar checks as before.

(l) Ran locally and overfit the data successfully.

Problem 2

(a) In "char_decoder.py".

(b) In "char_decoder.py".

(c) In "char_decoder.py".

(d) In "char_decoder.py"

(e) Results in "outputs/tesr_outputr_locar_q2.txt"

(f) Training took 40611.18 sec (11.28 hours) with a final test BLEU score of 24.56565986092025. Sampled results can be found in "outputs/tesr_outputs.txt".

Problem 3

(a) We summarize the presence of each word in Table 1. The fact that four of the six

Word	In vocab.json?
traducir	Yes
traduzco	No
traduces	No
traduce	Yes
traduzca	No
traduzcas	No

Table 1: Summary for whether a word is present in 'vocab.json'.

common forms of *traducir* do not appear in our vocabulary is problematic since these will all map to "<UNK>" when translating from Spanish to English, thereby giving poor translations. However, our new character based model will definitely help in overcoming this problem, since all of the above words will have different embeddings, and hopefully, related words will have embeddings which lead to similar translations. Given that this information is now input into the network, with the char-based decoder for any "<UNK>" tokens, we can expect our model to actually generate a sequence of characters which closely matches the learned translations.

(b) (i) We have a table of the nearest words according to Word2Vec All in Table 2:

Word	Nearest Neighbord
financial	economic
neuron	neurons
Franscisco	San
naturally	occurring
expectation	operator

Table 2: Nearest Cosing Neighbors in Word2VecAll.

(ii) We have a table of the nearest words according to the provided character embeddings in Table 3.

Word	Nearest Neighbord
financial	vertical
neuron	Newton
Franscisco	France
naturally	practically
expectation	exception

Table 3: Nearest Cosing Neighbors in Word2VecAll.

(iii) From looking at the above words and their nearest neighbors, the similarities modeled by Word2Vec appear to be what I term 'replacebility' similarity – that is to say, two words are similar if they can be replaced, without making the sentence incorrect (note, this is not the same as synonyms, since antonyms can be used as replacements).

The similarity modeled by the CharCNN is more of a lingusitic or word-level similarity where words with similar suffixes (for 'financial' and 'naturally') are clustered closer together – however, this is not necessarily always the case, as is shown by the neighbors of 'neuron' and 'Franscisco'.

These difference can be explained by the methodology used to achieve the given embeddings. The embeddings for Word2Vec are word-level embeddings, which means that the character-level information has no affect on the model, and infact, the model is completely oblivious to this information and as such any word-level, denotational similarity is, even if present, not causal. In contrast, the the CharCNN, embeddings for each word are directly derived from the characters composing that word (run through an RNN), and as such, it is reasonable to expect that words composed of similar characters would lead to similar embeddings.

Furthermore, the training objective for each embedding is different. When training Word2Vec, the objective is to match the probability distribution of a word given its context – as such, words which are likely to occur in the same context

(with the same surrounding words) will have similar embeddings. However, for the CharCNN, we've trained the embeddings on a translation task, which predicts next words or next characters. As such, for individual words, it makes sense that words with similar endings would have similar embeddings.

(c) We provide two cases below. One where the new model did the right thing, and one where it did not.

- (1) Yo estaba asombrada.
  (2) I was in awe.
  (3) I was <unk>
  (4) I was amazed.
  (5) This is an acceptable translation as provided by the model. The character based model was correctly able to initialize the character decoder based on the context of the sentence, and generate a plausible adjective. It also likely benefitted from the fact that both "awe" and "amazed" begin with 'a', but this is unconfirmed.
- (1) La intersexualidad adopta muchas formas.
  (2) Intersex comes in a lot of different forms.
  (3) <unk> adopt many ways.
  (4) Interviewers adopted many ways.
  (5) This is not an acceptable translation of the provided sentence. The character-based model likely made this error due to two main reasons: (1) the word "intersex" is relatively rare, (2) the sentence is short. (2) leads to a situation where the amount of context that the model can generate when parsing the input Spanish sentence is relatively little. This leads to a relatively weak signal being fed into the character decoder. With this weak signal, the character decoder is likely to pick a word close to "insersex", but due to (1), since intersex likely doesn't occur with enough frequency in the training data, it instead opted to produce a far more probably, yet near-neghbor, "interviewers" (note both words begin with the same prefix).