Assignment 5 - Written: Sub-Word Modeling Neural Machine Translation

1. Character-based convolutional encoder for NMT

(a) In Assignment 4 we used 256-dimensional word embeddings ($e_{word} = 256$), while in this assignment, it turns out that a character embedding size of 50 suffices ($e_{char} = 50$). In 1-2 sentences, explain one reason why the embedding size used for character-level embeddings is typically lower than that used for word embeddings.

Solution: One reason the embedding size for character-level embeddings is lower is because the character space will most always be smaller than any word space. There's simply more words than characters so we are afforded smaller embedding spaces.

(b) Write down the total number of parameters in the character-based embedding model (Figure 2), then do the same for the word-based lookup embedding model (Figure 1). Write each answer as a single expression (though you may show working) in terms of e_{char}, k, e_{word}, V_{word} (the size of the word-vocabulary in the lookup embedding model) and V_{char} (the size of the character-vocabulary in the character-based embedding model).
Given that in our code, $k = 5$, $V_{word} \approx 50,000$ and $V_{char} = 96$, state which model has more parameters, and by what factor (e.g. twice as many? a thousand times as many?)

Solution:

1. **character-based embedding**: For the embedding lookup layer we have a weight matrix of size $V_{char} \times e_{char}$. In the convolution layer portion we have a weight matrix of size $f \times e_{char} \times k$ and a

bias of size f. Note f will be set to e_{word} in our application. Lastly we take into account the number of parameters in the projection and gate layers. Here we have project/gate matrices and biases of size: $e_{word} \times e_{word}$ and e_{word}, respectively. So in all we'll have

$$\begin{aligned}
\text{total number of parameters} &= (V_{char} \times e_{char}) + ((f \times e_{char} \times k) + f) + ((e_{word} \times e_{word}) + e_{word}) \\
&= (V_{char} \times e_{char}) + ((e_{word} \times e_{char} \times k) + e_{word}) + ((e_{word} \times e_{word}) + e_{word}) \\
&= V_{char}e_{char} + ke_{word}e_{char} + e_{word} + e_{word}e_{word} + e_{word} \\
&= \mathbf{V_{char}e_{char} + ke_{word}e_{char} + e_{word}^2 + 2e_{word}} \\
&= (96 * 50) + (5 * 256 * 50) + (256^2) + (2 * 256) \\
&= \mathbf{134,848}
\end{aligned}$$

1. **word-based lookup embedding**: Note that the word-based embedding model uses a single Embedding layer given a vocabulary and embedding vector size. So in all we'll have:

$$\begin{aligned}
\text{total number of parameters} &= \mathbf{V_{word} * e_{word}} \\
&= (50,000 * 256) \\
&= \mathbf{2,800,000}
\end{aligned}$$

So in all the word embedding model has roughly 20 times more parameters than the char-based one.

(c) In step 3 of the character-based embedding model, instead of using a 1D convnet, we could have used a RNN instead (e.g. feed the sequence of characters into a bidirectional LSTM and combine the hidden states using max-pooling). Explain one advantage of using a convolutional architecture rather than a recurrent architecture for this purpose, making it clear how the two contrast.

Solution: A 1D convnet allows us to naturally work with character level n-grams by constricting the receptive field (kernel) size of the convolution to n. This allows us to extract local morphemic features of a word that are not influenced by the characters outside of the field's current position over said word. In contrast, an RNN can only visit characters of a word sequentially from beginning-to-end and/or end-to-beginning. This suggests that hidden character embeddings outside of a morphemic n-gram region can leak into the region's representation even though they will most likely lack any substructural word affinity. Therefore building up word embeddings from character sequences through an RNN may not embed the morpheme-like representations we'd enjoy with the 1D conv.

(d) In lectures we learned about both max-pooling and average-pooling. For each pooling method, please explain one advantage in comparison to the other pooling method. For each advantage, make it clear how the two contrast, and write to a similar level of detail as in the example given in the previous question.

Solution:

max-pooling: Max pooling compresses an input by extracting the most amplified activation signals in each windowed region of the input. By choosing the strongest signals, we ensure that the most prominent features in a region of a sequence get taken upstream as *the* region's representative. The advantage is that the max signal will, in many cases, provide the best summary (most important feature) of its region and in turn produce better word representations that are locally invariant to changes. In contrast, average pooling does not always pass on the strongest possible signal. They can get "watered" down considerably by small neighboring activations (attack of the outliers!).

average-pooling: Average pooling compresses an input by extracting the average activation signals of each windowed region of the input. It has the advantage that it can take regions with a majority of large feature signals and output a smooth summary of the neighborhood so as to maintain some information provided by a minority of weak signals. On the other hand, max pooling would completely discard such rare signals, even if they are somewhat helpful in depicting the region.

(h) Highway Network: Once you've finished testing your module, write a short description of the tests you carried out, and why you believe they are sufficient.

Solution: Testing was conducted by checking that the input and output shapes, batch dimension included, stayed the same after applying the layer, as required by highway networks (see Section 2. of the paper Highway Networks). I believe this is sufficient because the highway network is a rather simple linear transformation.

(i) CNN: Once you've finished testing your module, write a short description of the tests you carried out, and why you believe they are sufficient.

Solution: Testing was performed by checking shapes for the convolution activations prior to max-

pooling to ensure the one-dimensional convolution layer produced the necessary shapes.

2. Character-based LSTM decoder for NMT

(f) Please report the model's corpus BLEU Score.

```
(machine-learning) code [master●●] % sh run.sh test
load test source sentences from [./en_es_data/test.es]
load test target sentences from [./en_es_data/test.en]
load model from model.bin
Decoding: 100%|████████████████████████████████████████████████| 8064/8064 [06:29<00:00, 20.71it/s]
Corpus BLEU: 24.505559386486702
```

Figure 1: The model's corpus BLEU Score is roughly 24.51

3. Analyzing NMT Systems

(a)

Infinitive	traducir	to translate
Present	traduzco	I tranlsate
	traduces	you translate
	traduce	he/she translates
Subjunctive	traduzca	that I translate
	traduzcas	that you translate

Use vocab.json to find (e.g. using grep) which of these six forms are in the word-vocabulary, which consists of the 50,000 most frequent words in the training data for English and for Spanish. Superstrings don't count (e.g. having traducen in the vocabulary is not a hit for traduce). State which of these six forms occur, and which do not. Explain in one sentence why this is a bad thing for word-based NMT from Spanish to English. Then explain in detail (approximately two sentences) how our new character-aware NMT model may overcome this problem.

Solution: Only two of the six forms are accounted for in the word-vocabulary:

1. "traducir"
2. "traduce"

This is bad for word based models under a Spanish to English task because it suggests that these words will always transfer over to English as out of vocabulary (<UNK>) words. The fact that character models can pick up graphemic structures give us hope that <UNK> words can be resolved with in the model's decoder. Common irregular verbs can be reconstructed from pieces of their lemmas without having all conjugations explicitly stored in the vocabulary so long as they are common in the data set.

(b)

i. In Assignments 1 and 2, we investigated word embeddings created via algorithms such a Word2Vec, and found that for these embeddings, semantically similar words are close together in the embedding

space. In this exercise, we'll compare this with the word embeddings constructed using the Char-CNN trained in our NMT system. Go to `https://projector.tensorflow.org/`. The website by default shows data from Word2Vec. Look at the nearest neighbors of the following words (in cosine distance). • financial • neuron • Francisco • naturally • expectation For each word, report the single closest neighbor. For your convenience, for each example take a screenshot of all the nearest words (so you can compare with the CharCNN embeddings).

Solution:
• financial → economic
• neuron → nerve
• Francisco → san
• naturally → occurring
• expectation → norms

ii. Download the character-based word embeddings obtained from our implementation of the character-aware NMT model from this link. Navigate to `https://projector.tensorflow`. org/, select Load Data, and upload the files character-embeddings.txt (the embeddings themselves) and metadata.txt (the words associated with the embeddings). Now look at the nearest neighbors of the same words. Again, report the single closest neighbors and take screenshots for yourself.

Solution:
• financial → vertical
• neuron → Newton
• Francisco → France
• naturally → practically
• expectation → norms

iii. Compare the closest neighbors found by the two methods. Briefly describe what kind of similarity is modeled by Word2Vec. Briefly describe what kind of similarity is mod- eled by the CharCNN. Explain in detail (2-3 sentences) how the differences in the methodology of Word2Vec and a Char-CNN explain the differences you have found.

Solution:
Word2Vec models similarity by context proximity so that words that are distributed frequently together in sentences will be positioned closely, more similar, in the embedding space. This seems to better measure semantic relations between words. On the other hand CharNN models similarity by spelling proximity so that words with graphemes

(c) As in Assignment 4, we'll take a look at the outputs of the model that you have trained! The test set translations your model generated in 2(f) should be located in the outputs directory at: outputs/test outputs.txt. We also provided translations from a word-based model from our Assignment 4 model in the file outputs/test outputs a4.txt. Find places where the word-based model produced <UNK>, and compare to what the character- based decoder did. Find one example where the character-based decoder produced an acceptable translation in place of <UNK>, and one example where the character-based decoder produced an incorrect translation in place of <UNK>. As in Assignment 4, 'acceptable' and 'incorrect' doesn't just mean 'matches or doesn't match the reference translation' – use your own judgment (and Google Translate, if necessary). For each of the two examples, you should:

- Write the source sentence in Spanish. The source sentences are in en es data/test.es.

- Write the reference English translation of the sentence. The reference translations are ii en es data/test.en.
- Write the English translation generated by the model from Assignment 4. These translations are in outputs/test outputs a4.txt. Underline the <UNK> you are talking about.
- Write your character-based model's English translation. These translations are in outputs/test outputs.txt. Underline the CharDecoder-generated word you are talking about.
- Indicate whether this is an acceptable or incorrect example. Give a brief possible explanation (one sentence) for why the character-based model performed this way.

Solution:

a. Example where the character-based decoder produced an acceptable translation in place of <UNK>

- Spanish Reference: "Tenemos la idea de que ser mujer es tener identidad femenina".
- English Reference: "So we have the concept that what it means to be a woman is to have a female identity".
- Word-Based Output: "We have the idea that being a woman is to have <unk> identity".
- CharNN Output: "We have the idea that being woman is to have female identity".

This fill-in is acceptable as it correctly constructs 'female' in place of <unk>. Note that it is not an exact translation of the spanish word 'feminina', i.e. 'feminine', but it gives nearly the same sense in this context. Most likely the char decoder learned to build up 'female' from its graphemic similarity to 'feminina' but couldn't reach 'feminine'.

b. Example where the character-based decoder produced an incorrect translation in place of <UNK>.

- Spanish Reference: "Una mujer autista llamada Zosia Zaks dijo una vez".
- English Reference: "An autistic [man] named Zosia Zaks once said".
- Word-Based Output: "A autistic woman named <unk> <unk> once said".
- CharNN Output: "An autistic woman named Mosor Zamis once said".

This is an incorrect example as the name of the woman was modified/lost in the translation. Although the output name was incorrect, the CharNN plugged in a proper noun in the right place.