CS224N: Assignment 4

Mark Thornburg

1. Neural Machine Translation with RNNs (45 points)

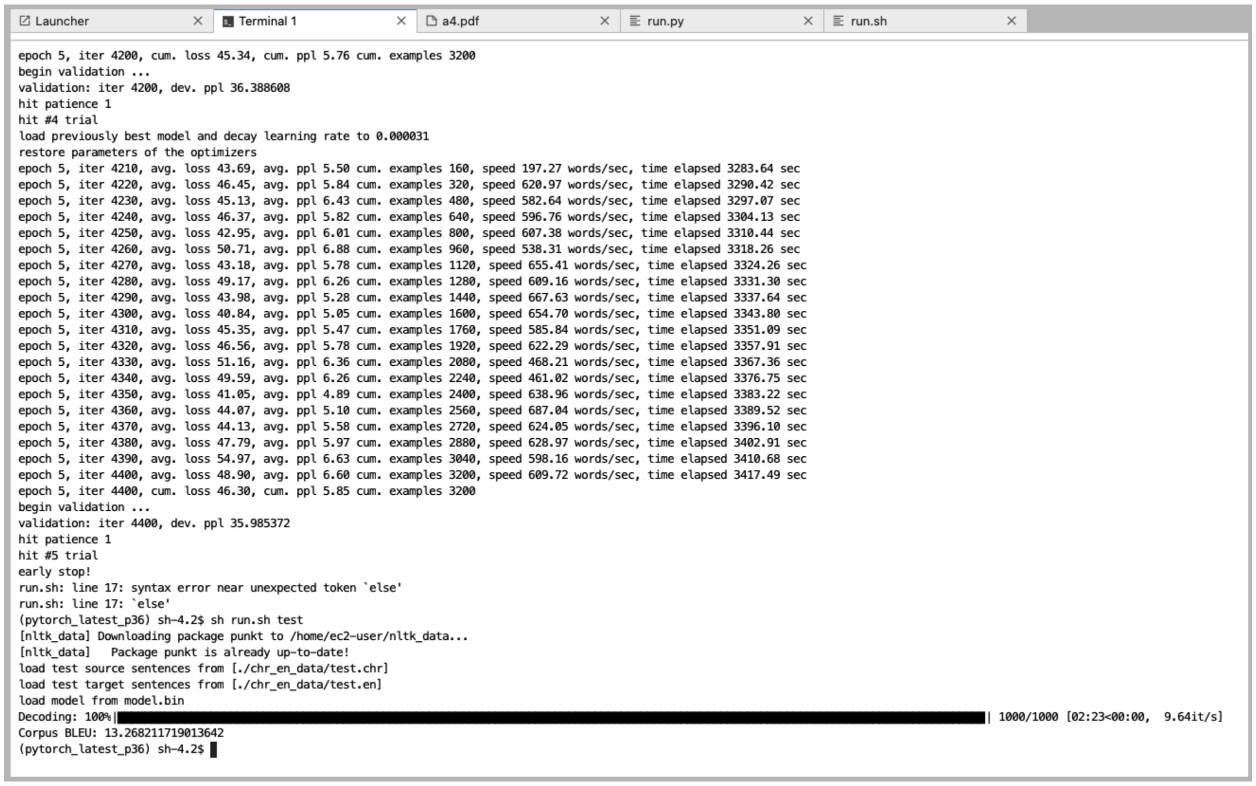
1G

The masks set all raw attention scores in the attention layer to -infinity for any encoder position where the token was a padding value. After applying softmax to the raw attention outputs, this results in an attention distribution of 0 for these positions.

This is needed as there won't be "valid" hidden states to use at these positions when computing the attention output. Therefore, this ensures that our attention output is only a weighted combination of encoder hidden states that were produced at positions with non padded inputs.

1H

My model's bleu score was 13.268211719013642 .



```
epoch 5, iter 4200, cum. loss 45.34, cum. ppl 5.76 cum. examples 3200
begin validation ...
validation: iter 4200, dev. ppl 36.388608
hit patience 1
hit #4 trial
load previously best model and decay learning rate to 0.000031
restore parameters of the optimizers
epoch 5, iter 4210, avg. loss 43.69, avg. ppl 5.50 cum. examples 160, speed 197.27 words/sec, time elapsed 3283.64 sec
epoch 5, iter 4220, avg. loss 46.45, avg. ppl 5.84 cum. examples 320, speed 620.97 words/sec, time elapsed 3290.42 sec
epoch 5, iter 4230, avg. loss 45.13, avg. ppl 6.43 cum. examples 480, speed 582.64 words/sec, time elapsed 3297.07 sec
epoch 5, iter 4240, avg. loss 46.37, avg. ppl 5.82 cum. examples 640, speed 596.76 words/sec, time elapsed 3304.13 sec
epoch 5, iter 4250, avg. loss 42.95, avg. ppl 6.01 cum. examples 800, speed 607.38 words/sec, time elapsed 3310.44 sec
epoch 5, iter 4260, avg. loss 50.71, avg. ppl 6.88 cum. examples 960, speed 538.31 words/sec, time elapsed 3318.26 sec
epoch 5, iter 4270, avg. loss 43.18, avg. ppl 5.78 cum. examples 1120, speed 655.41 words/sec, time elapsed 3324.26 sec
epoch 5, iter 4280, avg. loss 49.17, avg. ppl 6.26 cum. examples 1280, speed 609.16 words/sec, time elapsed 3331.30 sec
epoch 5, iter 4290, avg. loss 43.98, avg. ppl 5.28 cum. examples 1440, speed 667.63 words/sec, time elapsed 3337.64 sec
epoch 5, iter 4300, avg. loss 40.84, avg. ppl 5.05 cum. examples 1600, speed 654.70 words/sec, time elapsed 3343.80 sec
epoch 5, iter 4310, avg. loss 45.35, avg. ppl 5.47 cum. examples 1760, speed 585.84 words/sec, time elapsed 3351.09 sec
epoch 5, iter 4320, avg. loss 46.56, avg. ppl 5.78 cum. examples 1920, speed 622.29 words/sec, time elapsed 3357.91 sec
epoch 5, iter 4330, avg. loss 51.16, avg. ppl 6.36 cum. examples 2080, speed 468.21 words/sec, time elapsed 3367.36 sec
epoch 5, iter 4340, avg. loss 49.59, avg. ppl 6.26 cum. examples 2240, speed 461.02 words/sec, time elapsed 3376.75 sec
epoch 5, iter 4350, avg. loss 41.05, avg. ppl 4.89 cum. examples 2400, speed 638.96 words/sec, time elapsed 3383.22 sec
epoch 5, iter 4360, avg. loss 44.07, avg. ppl 5.10 cum. examples 2560, speed 687.04 words/sec, time elapsed 3389.52 sec
epoch 5, iter 4370, avg. loss 44.13, avg. ppl 5.58 cum. examples 2720, speed 624.05 words/sec, time elapsed 3396.10 sec
epoch 5, iter 4380, avg. loss 47.79, avg. ppl 5.97 cum. examples 2880, speed 628.97 words/sec, time elapsed 3402.91 sec
epoch 5, iter 4390, avg. loss 54.97, avg. ppl 6.63 cum. examples 3040, speed 598.16 words/sec, time elapsed 3410.68 sec
epoch 5, iter 4400, avg. loss 48.90, avg. ppl 6.60 cum. examples 3200, speed 609.72 words/sec, time elapsed 3417.49 sec
epoch 5, iter 4400, cum. loss 46.30, cum. ppl 5.85 cum. examples 3200
begin validation ...
validation: iter 4400, dev. ppl 35.985372
hit patience 1
hit #5 trial
early stop!
run.sh: line 17: syntax error near unexpected token `else'
run.sh: line 17: `else'
(pytorch_latest_p36) sh-4.2$ sh run.sh test
[nltk_data] Downloading package punkt to /home/ec2-user/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
load test source sentences from [./chr_en_data/test.chr]
load test target sentences from [./chr_en_data/test.en]
load model from model.bin
Decoding: 100%|████████████████████████████████████████████████████████████████████████████████████████████| 1000/1000 [02:23<00:00,  9.64it/s]
Corpus BLEU: 13.268211719013642
(pytorch_latest_p36) sh-4.2$ █
```

1l

Dot product attention scores compute “similarity” between encoder hidden states and decoder hidden state via the dot product function. Multiplicative attention uses a large parameter learned matrix to linearly project the encoder hidden state + decoder state to compute similarity.

The main advantage of these scores compared to multiplicative attention is that they are fast to compute (meaning more gradient updates for a fixed amount of training) as well as provide math interpretable attention scores (higher scores *generally* mean more similar vectors, lower scores mean more dissimilar vectors).

The disadvantage of dot product attention is that it requires the encoder hidden states and decoder hidden state to have information relevant to decide which parts of the source sentence need to be attended too. This is a lot for an LSTM to do as now hidden states need to be predictive of its immediate output, encode information from previous timesteps within an encoder/decoder, and have information relevant for attention lookbacks.

Multiplicative attention helps with this by allowing a lot of the information relevant for deciding attention lookbacks between encoder hidden states and decoder hidden state to be stored in an attention matrix that can be learned and updated with lots of training data.

Additive attention uses a small neural network layer to project encoder hidden state + decoder state into an attention score.

The advantage of additive attention versus multiplicative attention is that a neural network layer is more flexible in the types of projections it can perform on the encoder + decoder hidden states into an attention score (no longer has to be a linear projection!) which can lead to better performance. The disadvantage is that updating this neural network can potentially take more compute and needs to be regularized potentially to prevent overfitting.

2. Analyzing NMT Systems (30 points)

2a

Cherokee is a polysnethetic language, which means that the “word” corpus is extremely high dimensional and composed from lower level morphemes that may not stand alone (e.g. high dimensional conjugation set that is used with a high dimensional noun set to form lots of different concepts and words). Subword modeling (as opposed to word modeling) in this case will assist during training (and improve our NMT system’s ability to generalize to unseen examples) by giving our model the chance to learn when/how to decompose a sentence into these morphemes. This helps translation generalization when you see morphemes you’ve seen before but in different contexts.

2b

Character-level and subword-level embeddings are often smaller than whole word embeddings because these subwords need to capture less information as they will be combined with other subwords in a sequence to form a concept.

Take the word biology for instance. If you were to model that from a word level, you would need both information that bio refers to the living world and ology refers to the study of something in that one specific word embedding for biology.

Whereas if you modeled this as two different subwords and embeddings, each embedding could have information about just one of these concepts within them. So each embedding could be smaller as it needs to hold less information. Then it could be up to the language model to learn that when it sees “biology” that it means the study of the living world.

2c

We can think of multilingual training as a form of pre-training & transfer learning. By training on different languages, our model will be forced to first learn architecture and parameters that can take sequences of embeddings of concepts and combine them together to translate into a new sequence of target embeddings and concepts.

Then the NMT system can fine-tune on a smaller subset of Cherokee translations to learn how to adjust to new words and embeddings from the Cherokee language while still utilizing a lot of the parameters and architectures that know how to combine concepts into a target language (specifically English here that was seen in the other multilingual translation tasks!)

Another way to put it is that you can think of cherokee to English translation as a composition of these tasks:

1. Decompose Cherokee source sentence into sequence of subwords with meaningful embeddings of concepts.
2. Learn how to map a sequence of embeddings of concepts to the best sequence of English specific embeddings.

Task 2 is required in any X_to_english translation task, so by using multi-lingual training, you give your NMT system the ability to learn 2 really well from that outside data!

2D

I

1. The error that the NMT system is making is that it is failing to translate “a crown of daisies” in the source sentence to the equivalent in the target language (instead mapping it to “her hair”). This may be an issue with the attention mechanism as “in her hair” occurs later in the source sentence.
2. If it’s a problem with underperforming attention, try adjusting the model capacity of the attention mechanism (e.g. try increasing additive attention with possibly a couple neural network layers if our attention mechanism is underfitting. If we are overfitting, try dot product attention). To determine if our attention mechanism is underfitting or overfitting, look to see if we encounter similar errors on training examples.

II

1. The error that the NMT system is making is that it is failing to translate “she is” from the Cherokee source sentence into its gendered equivalent pronoun in English (and instead translating it to “It is”). One hypothesis that I have for this is that there may be a bias in the low record Cherokee translation training set that includes not enough translations involving women.
2. In order to solve the problem described above (if it is indeed the issue), we can use synthetic data techniques to take valid translations involving men (aka He/Him) and create additional training samples with female pronouns (He went to the pond ⇒ She went to the pond.)

III

1. 1. The error that the NMT system is making is that it is failing to translate Littlefish into a proper noun and instead translating it to the raw concept instead: “a little fish”. This is probably due to a subword modeler that hasn’t seen that proper noun Littlefish enough (either in Cherokee or English). The modeler might instead be decomposing it to “little” and “fish” and looking up those two word’s corresponding embeddings before sending it into the model, or vice versa (might not be a valid word in the target word domain).
2. 2. To solve this, increase the domain size of the subword modeler (both source and target) and make sure you have enough translations with Littlefish in it. Make sure that Littlefish has a source embedding and a target word embedding in both the english / cherokee language. Additionally, you could add synthetic translations into training by taking human proper nouns in other translations and substituting Littlefish into it.

2E

I

The long “correct” translation that I found was line 64 in test_outputs.txt (I didn’t find any verbatim correct translations):

Translation: And behold, a voice came out of heaven, saying, Thou art my beloved son, who is good.
Target: and lo, a voice out of the heavens, saying, This is my beloved Son, in whom I am well pleased.

I didn’t find this target exactly in the training file `train.en`:

```
train_targets_df = pd.read_csv('./chr_en_data/test.en', delimiter='\n', header=None)
train_targets_df.columns = ['target']
sum(train_targets_df['target'] == 'and lo, a voice out of the heavens, saying, This is my beloved Son, in whom I am well pleased.')
0
```

Had this target shown up in the training file (or something very similar), we could have argued that it was due to seeing a very similar example during training and memorization by the NMT system. However, since I couldn’t find anything very similar in the set of training labels, we could argue this is a good generalization by our NMT system.

II

The translation that I found that started off correct (for a sequence of 4 words) but then diverges is line 42:

Translation: Salute one another with love. Amen. Amen. Amen.
Target Label: Salute one another with a kiss of love. Peace be unto you all that are in Christ.

This may suggest that our NMT decoder might have higher accuracy for shorter sentences. It also might suggest that our NMT decoder has good accuracy for predicting the next word in a sequence assuming it correctly predicted the previous words - but once it starts to make mistakes, those mistakes compound and cause it to diverge quickly from the target.

2F

I

For our first example:

$c_1 =$ the love can always do
$r_1 =$ love can always find a way
$r_2 =$ love makes anything possible

First, let's compute p_0^{c1} and p_1^{c1} :

$$\text{1-grams}(c1) = \{the, love, can, always, do\}$$
$$p_1^{c1} = \frac{\sum_{ngram \in \text{1-grams}(c_1)} \min(\max_{i \in \{1,2\}} count_{r_i}(ngram), count_{c_1}(ngram))}{\sum_{ngram \in \text{1-grams}(c_1)} count_{c1}(ngram)}$$
$$p_1^{c1} = \frac{0+1+1+1+0}{1+1+1+1+1}$$
$$p_1^{c1} = \frac{3}{5}$$
$$\text{2-grams(c2)} = \{\text{the love, love can, can always, always do}\}$$
$$p_2^{c1} = \frac{\sum_{ngram \in \text{2-grams}(c_1)} \min(\max_{i \in \{1,2\}} count_{r_i}(ngram), count_{c_1}(ngram))}{\sum_{ngram \in \text{2-grams}(c_1)} count_{c1}(ngram)}$$
$$p_2^{c1} = \frac{0+1+1+0}{1+1+1+1}$$
$$p_{2c1} = \frac{1}{2}$$

Now let's compute the brevity penalty BP:

$$len(c1) = 5, len(r1) = 6, len(r2) = 5$$
$$5 = len(c1) > len(r|c1) = 4$$
$$BP = 1$$

Therefore our bleu score for candidate c_1 with respect to r_1, r_2 is:

$$BLEU = BP * exp(\sum_{n=1}^{n=2} 0.5 * ln(p_n^{c1}))$$

since $\lambda_n = 0.5$ for all n. This comes out to:

$$BLEU = 1 * exp(0.5 * ln(\frac{3}{5}) + 0.5 * ln(\frac{1}{2}))$$

$$BLEU = 0.54772$$

$$p_1^{c2} = \frac{\sum_{ngram \in \text{1-grams}(c_2)} \min(\max_{i \in \{1,2\}} count_{r_i}(ngram), count_{c_2}(ngram))}{\sum_{ngram \in \text{1-grams}(c_2)} count_{c2}(ngram)}$$

The second candidate translation gives us a better bleu score of 0.632455 versus 0.54772. I agree with the score as “love can make anything possible” is very close to one of the reference translations “love makes anything possible” (versus the first candidate “the love can always do” which doesn’t really make sense).

II

$\text{len}(c_2) = 5 \quad \text{len}(r_1) = 6 \quad \text{len}(r_2) = 4$

$5 = \text{len}(c_2) > 4 = \text{len}(r)$

$BP = 1$

This gives us our final bleu score:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{n=2} 0.5 \cdot \log p_n\right)$$

$$= 1 \cdot \exp(0.5 \cdot \log 0.5 + 0.5 \ln(0.5))$$

$$= 0.632455$$

The second NMT translation

III Let's start with c_1. Since r_2 was lost, we have the new brevity penalty:

$\text{len}(r) = 6 \quad \text{len}(c_1) = 5$

$BP = \exp\left(1 - \frac{6}{5}\right) = 0.81873$

Now let's recomput $p_i^{c_1}$:

$$p_1^{c_1} = \frac{0+1+1+1+0}{1+1+1+1+1} = 3/5$$

$$p_2^{c_1} = \frac{0+1+1+0}{1+1+1+1} = 1/2$$

So our new Bleu Score is:

$$\text{Bleu} = 0.81873 \cdot \exp\left(0.5 \cdot \ln(3/5) + 0.5 \cdot \ln(0.5)\right)$$
$$= 0.44843$$

Similarity for candidate translation c^2:

$$BP = 0.81873$$
$$p_1^{c_2} = \frac{1+1+0+0+0}{1+1+1+1+1} = 2/5$$
$$p_2^{c_2} = \frac{1+0+0+0}{1+1+1+1} = 1/4$$

$$\text{Bleu} = 0.81873 \cdot \exp\left(0.5 \cdot \ln(2/5) + 0.5 \ln(1/4)\right)$$
$$= 0.258905$$

Now the first candidate translation has a higher BLEU of 0.2589 versus the second of 0.44843. I actually agree with this assuming that reference translation 1 was a perfect translation of the Cherokee source sentence as “the love can always do” seems to be closer to “love can always find a way” then “love makes anything possible”. But I can still see arguments for or against, especially if we assume reference translation 2 was a better actual translation.

III

If we are only computing BLEU with a single reference translation, then our bleu scores for candidate translations can be very noisey depending on the quality of the single reference translation. Additionally, sometimes there are truly multiple valid ways to translate a sentence/ With only a single reference translation, it’s impossible to reward a candidate translation for mapping to one of the multiple valid ways (which isn’t a problem if we have a reference translation for each valid way since bleu subcomponents often compare the candidate translation to the “closest” reference translation!).

IV

Advantages:
BLEU scores are somewhat more interpretable to use, especially when diagnosing why a particular candidate translation has a low score (e.g. we can determine what the brevity penalty was and what the ngram precision scores were to break down why a prediction is bad). Human evaluation won't necessarily give that information in a standard way.

BLEU scores may be a lot less expensive to "compute" as it just requires reference translations. Technically human evaluation requires 1) the human to translate the sentence (either in paper or in their head) and then 2) give a more subjective score of how close that translation is.

Disadvantages:

One disadvantage of bleu score is that you need to determine good n-gram precision weights before computing scores - which may heavily depend on the language you are translating and might take a lot of time to get good default values for. This isn't required with human translators.

BLEU scores are really only capturing n-gram precision and closeness in length to reference translations. This means bleu scores can be very invariant potentially due to bad n-gram sequence ordering of a translation. "The park went to I" gets a pretty decent bleu score compared to "I went to the park" (even though they are conceptually opposites). Human translators. however, can take more into account.

Also BLEU score subcomponents (n-gram precision + brevity penalty) are computed to the "closest reference translation". So if you have a ton of reference translations and one of them is very bad, a lot of candidate translations that are very close to the singular bad reference translation will get very high bleu scores. This contrasts with human translators where 9/10 translators would have given that bad candidate translation a bad score.

Additional BLEU Example

```
e^(1 - (20/5)) * (1) = .04978706837
>>> reference
[['I', 'went', 'to', 'the', 'park', 'and', 'went', 'swimming', 'with',
'my', 'dog', 'alfred', 'who', 'is', 'very', 'cute', 'and', 'likes', 'to',
'draw']]
>>> hypothesis
[['I', 'went', 'to', 'the', 'park']]
>>> nltk.translate.bleu_score.sentence_bleu(reference, hypothesis[0])
0.049787068367863944
```