Project Report **Zeren Xing**
CAP6610 Team Number: 51

Problem Statement

Neural Machine translation is a heated topic nowadays. The basic underlying idea is to learn semantic information and structures from sentence in a complicated or even deep neural network.

We are going to do sentence-to-sentence machine translation. Each sentence is as an entry of data sequence. We code each word to a vector, plus a "SENTENCE_END_SIGN" at the end of each sentence and get a large matrix representing our sentence. All our computation is based on matrix algebra.

Model Theory

Word to Vector

Tokenization

We read in all sentences line by line in our sample corpus and break lines into "tokens", such as words and punctuations, by the "nltk" package in Python.

Vocabulary

We calculate frequency of each token in the corpus and sort tokens from the most frequent to the least one. Our vocabulary size $C = 2000$, comprises of the top 1999 frequent tokens both in the input language, English, and in the output language, German, and a "UNKNOWN_TOKEN", to which any tokens outside the top 1999 ones belongs.

Word2Vec

We use the one-hot vector to represent the token, which means the ith word in the vocabulary will a vector with all 0's but a 1 at the ith position.

Model Framework

Our model is based on RNN Encoder-Decoder, proposed by [1] and [2].

Encoder

In the Encoder-Decoder framework, an encoder reads the input sentence, a sequence of vectors $\mathbf{x} = (x_1, \ldots, x_{T_x})$, into a vector c[3]. Our RNN encoder works in this way

$$h_t = f(W_e \cdot x_t + U_e \cdot h_{t-1}) \tag{1}$$

$$c = h_{T_x} \tag{2}$$

where $h_t \in \mathbb{R}^H$ is a hidden state with hidden layer size H of the encoder at time t, and f is a nonlinear activation function.

Decoder

An decoder is trained to predict the next word $y_{t'}$ given the context vector c and the previously predicted words $\{y_1, \ldots, y_{t'-1}\}$.

In details, it predicts a probability distribution $o_{t'}$ over the vocabulary for the next word, and we select the desired next word $y_{t'}$ with the highest predicted probability. Our RNN decoder works in this way

$$s_1 = f(V \cdot c) \tag{3}$$

$$s_{t'} = f(W_d \cdot y_{t'-1} + U_d \cdot s_{t'-1} + V \cdot c),\ t' > 1 \tag{4}$$

$$o_{t'} = softmax(P \cdot s_{t'}) \tag{5}$$

where $s_{t'} \in \mathbb{R}^H$ is a hidden state with hidden layer size H (same as the encoder hidden layer size) of the decoder at time t', and f is the same nonlinear activation function as the one in the encoder.

Notice that the decoder takes different $y_{t'}$ in training process and translation process. In training process, the decoder takes true $\mathbf{y}$ from our training samples, which means it won't really select a $y_{t'}$ from the output $o_{t'}$. In translation process, there won't be known $\mathbf{y}$ available so it always select a $\hat{y}_{t'}$ from the estimated $o_{t'}$ and use it as an input for $s_{t'+1}$

Language Model

The language model underlying the RNN Encoder-Decoder model is that we are trying to estimate the joint probability of a sentence $\mathbf{y}$ in the target language by decomposing the joint probability into the ordered conditionals[3]:

$$p(\mathbf{y}) = \prod_{t'=1}^{T_y} p(y_{t'} | \{y_1, \ldots, y_{t'-1}\}, \mathbf{x}) \tag{6}$$

where $\mathbf{y} = (y_1, \ldots, y_{T_y})$, $\mathbf{x} = (x_1, \ldots, x_{T_x})$.

Model Evaluation

Loss Function

We use the cross entropy error as the loss function. If we have N training samples in the target language (tokens from the corpus), the loss with respect to our predicted probability o and the true word y is given by:

$$L_t(y_t, o_t) = y_t \log(o_t) \tag{7}$$

$$L = -\frac{1}{N} \sum_{t=1}^{N} L_t \tag{8}$$

In the case of random prediction without any prior knowledge, the probability distribution of the vocabulary of size C will be evenly $\frac{1}{C}$ for each word. Then our baseline of the complete loss over any corpus will be $-\frac{1}{N} \sum_{t=1}^{N} \log \frac{1}{C} = \log C$.

Training Algorithm

We train the model with stochastic gradient descent and back propagation through time.

BPTT

Let $z_t^d = W_d y_{t-1} + U_d s_{t-1} + Vc$, $z_t^e = W_e x_t + U_e h_{t-1}$, $net_t = Ps_t$. To improve the model as much as possible in each training step, we set

$$\frac{\partial L_t}{\partial net_t} = y_t - o_t$$

. First we do back propagation through time from step t back to the first step in our decoder. For step t and each step i, $1 \leq i < t$ in the decoder, we have this chain rule through time:

$$\begin{aligned}
\frac{\partial L_t}{\partial P} &= \frac{\partial L_t}{\partial net_t}\frac{\partial net_t}{\partial P} \\
&= (y_t - o_t) \otimes s_t \\
\frac{\partial L_t}{\partial z_t^d} &= \frac{\partial L_t}{\partial net_t}\frac{\partial net_t}{\partial s_t}\frac{\partial s_t}{\partial z_t^d} \\
&= P^T(y_t - o_t)f'(z_t^d) \\
\frac{\partial L_t}{\partial z_i^d} &= \frac{\partial L_t}{\partial z_{i+1}^d}\frac{\partial z_{i+1}^d}{\partial s_i}\frac{\partial s_i}{\partial z_i^d} \\
&= U_d^T \frac{\partial L_t}{\partial z_{i+1}^d} f'(z_i^d)
\end{aligned}$$

Then in each step i, $1 < i \leq t$ in the decoder, we have

$$\begin{aligned}
\frac{\partial L_t}{\partial z_i^d}\frac{\partial z_i^d}{\partial W_d} &= \frac{\partial L_t}{\partial z_i^d} \otimes s_{i-1} \\
\frac{\partial L_t}{\partial z_i^d}\frac{\partial z_i^d}{\partial U_d} &= \frac{\partial L_t}{\partial z_i^d} \otimes y_{i-1} \\
\frac{\partial L_t}{\partial z_i^d}\frac{\partial z_i^d}{\partial V} &= \frac{\partial L_t}{\partial z_i^d} \otimes c \\
\frac{\partial L_t}{\partial z_i^d}\frac{\partial z_i^d}{\partial c} &= V^T \frac{\partial L_t}{\partial z_i^d}
\end{aligned}$$

For step 1, we have

$$\begin{aligned}
\frac{\partial L_t}{\partial z_1^d}\frac{\partial z_1^d}{\partial W_d} &= 0 \\
\frac{\partial L_t}{\partial z_1^d}\frac{\partial z_1^d}{\partial U_d} &= 0 \\
\frac{\partial L_t}{\partial z_1^d}\frac{\partial z_1^d}{\partial V} &= \frac{\partial L_t}{\partial z_1^d} \otimes c \\
\frac{\partial L_t}{\partial z_1^d}\frac{\partial z_1^d}{\partial c} &= V^T \frac{\partial L_t}{\partial z_1^d}
\end{aligned}$$

Then we have the equations for derivative of L respect to all the parameters in the decoder:

$$\begin{aligned}
\frac{\partial L}{\partial W_d} &= \sum_{t=1}^{T_y} \frac{\partial L_t}{\partial W_d} \\
&= \sum_{t=1}^{T_y} \sum_{i=1}^{t} \frac{\partial L_t}{\partial z_i^d} \frac{\partial z_i^d}{\partial W_d} \\
\frac{\partial L}{\partial U_d} &= \sum_{t=1}^{T_y} \sum_{i=1}^{t} \frac{\partial L_t}{\partial z_i^d} \frac{\partial z_i^d}{\partial U_d} \\
\frac{\partial L}{\partial V} &= \sum_{t=1}^{T_y} \sum_{i=1}^{t} \frac{\partial L_t}{\partial z_i^d} \frac{\partial z_i^d}{\partial V}
\end{aligned}$$

And for c, we need

$$\frac{\partial L_t}{\partial c} = \sum_{i=1}^{t} \frac{\partial L_t}{\partial z_i^d} \frac{\partial z_i^d}{\partial c}$$

Similarly in the encoder, for step T_x and each step j, $1 \leq i < T_x$we have

$$\begin{aligned}
\frac{\partial L_t}{\partial z_{T_x}^e} &= \frac{\partial L_t}{\partial c} \frac{\partial c}{\partial z_{T_x}^e} \\
&= \frac{\partial L_t}{\partial c} f'(z_{T_x}^e) \\
\frac{\partial L_t}{\partial z_j^e} &= \frac{\partial L_t}{\partial z_{j+1}^e} \frac{\partial z_{j+1}^e}{\partial h_j} \frac{\partial h_j}{\partial z_j^e} \\
&= U_e^T \frac{\partial L_t}{\partial z_{j+1}^e} f'(z_j^e)
\end{aligned}$$

Then in each step j, $1 < j \leq T_x$ in the encoder, we have

$$\begin{aligned}
\frac{\partial L_t}{\partial z_j^e} \frac{\partial z_j^e}{\partial W_e} &= \frac{\partial L_t}{\partial z_j^e} \otimes h_{j-1} \\
\frac{\partial L_t}{\partial z_j^e} \frac{\partial z_j^e}{\partial U_e} &= \frac{\partial L_t}{\partial z_j^e} \otimes x_j
\end{aligned}$$

At step 1, we have

$$\begin{aligned}
\frac{\partial L_t}{\partial z_1^e} \frac{\partial z_1^e}{\partial W_e} &= 0 \\
\frac{\partial L_t}{\partial z_1^e} \frac{\partial z_1^e}{\partial U_e} &= \frac{\partial L_t}{\partial z_1^e} \otimes x_1
\end{aligned}$$

So we have the equations for derivative of L respect to all the parameters in the encoder:

$$\begin{aligned}\frac{\partial L}{\partial W_e} &= \sum_{t=1}^{T_y} \frac{\partial L_t}{\partial W_e} \\ &= \sum_{t=1}^{T_y} \sum_{j=1}^{T_x} \frac{\partial L_t}{\partial z_j^e} \frac{\partial z_i^d}{\partial W_e} \\ \frac{\partial L}{\partial U_e} &= \sum_{t=1}^{T_y} \sum_{j=1}^{T_x} \frac{\partial L_t}{\partial z_i^e} \frac{\partial z_i^e}{\partial U_e}\end{aligned}$$

For each parameter in the model, W_e, U_e, W_d, U_d, V, P, we have the above partial derivative of the loss function respect to them. In implementation, we may not back propagate until the starting step of the RNN since the sentence could be long and very early steps may only have modest influence on very later steps. Thus we truncate the bptt process at a fixed step. More details of model implementation will be explained in the following part.

Model Implementation

Initialization

We initialize the weights with uniform distribution in the interval from $\left[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right]$ where n is the number of incoming connections from the previous layer. We also set the truncating step at 5.

Loss Calculation

We only count in the estimated probability of the desired word y_t from our training examples for the loss function in training process. The complete loss of the training corpus will be the average loss over the total number of tokens in the corpus.

Stochastic Gradient Descent

We will update all parameters after each pair of sentences from the source corpus (in English) and the target corpus (in German) is read in for training. There isn't much sense in mini-batch algorithm in this model, since all sentences vary significantly in length and words in them. We will do multiple epochs with one set of sample corpus and see how the complete loss over the corpus will change. When the loss function increases after some epoch, we will decrease the learning rate.

Gradient Check

We need to check whether our implementation of the BPTT algorithm is correct. We compare the gradients of each parameter from our BPTT to the estimation of the partial derivative of the loss function respect to each parameter by checking the relative error $\frac{|x-y|}{|x|+|y|}$ between them.

We estimate the partial derivative in this way:

$$\begin{aligned}\frac{\partial L}{\partial \theta} &= \lim_{h \to 0} \frac{L(\theta + h) - L(\theta - h)}{2h} \\ &\approx \frac{L(\theta + h) - L(\theta - h)}{2h} \text{ with small } h\end{aligned}$$

To do the gradient check we only need a mini size model with very few hidden layer nodes and input/output dimension. We set the $h = 0.001$ and the threshold of the relative error to 0.01.

Gradient check results proves that we do implement the BPTT algorithm correctly and we can start to train our model.

Primary Training Result

Due to limited calculation capacity of our own laptop, we only use 1000 pairs of sentences as our training set and train the model with 10 epochs and 50 hidden layer nodes. The baseline of loss now is $\log 2000 = 7.600902$. The training result of our model is:

```
2017-04-17 18:29:23: Loss after 0 examples used in model and epoch=0: 7.596586
2017-04-17 18:29:53: Loss after 1000 examples used in model and epoch=1: 5.469779
2017-04-17 18:30:21: Loss after 2000 examples used in model and epoch=2: 5.420758
2017-04-17 18:30:49: Loss after 3000 examples used in model and epoch=3: 5.328672
2017-04-17 18:31:18: Loss after 4000 examples used in model and epoch=4: 5.266854
2017-04-17 18:31:46: Loss after 5000 examples used in model and epoch=5: 5.247842
2017-04-17 18:32:14: Loss after 6000 examples used in model and epoch=6: 5.234273
2017-04-17 18:32:43: Loss after 7000 examples used in model and epoch=7: 5.221078
2017-04-17 18:33:12: Loss after 8000 examples used in model and epoch=8: 5.207045
2017-04-17 18:33:41: Loss after 9000 examples used in model and epoch=9: 5.192672
```

Apparently the loss is decreasing with each epoch, although the most significant decrease only happens once with the first epoch.

Translation Problem

When we tried to translate some sentences from our corpus which hasn't been used for training, we found that the model usually put the highest probability on the same word, mostly the "the" word in the target language. So we can't get a real sentence if we always select the word with the highest estimated probability at each time step. This also leads to difficulty in predicting the ending sign of a sentence and terminating the translation process.

At first we tried to do sampling according to the estimated probability rather than directly select the token with the highest probability. Whenever "UNKNOWN_TOKEN" is selected we will do resampling until "UNKNOWN_TOKEN" is get rid of.

We also put an upper bound of sentence length so translation process will be forced to stop after a fixed number of steps. Here I set the upper bound to 30 as I don't want our translated sentences longer than 30.

Common Words v.s. Meaningful words

It is obvious that the natural frequency, or the prior distribution of words has overwhelmed influence on estimation compared to the semantic context inside a sentence. The model tend to choose the most frequent word, "the", especially when training set is not large enough so data information can't have significant influence on the posterior distribution.

We need to force the model to pay more attention to those less common but more meaningful tokens. At first I just tried increasing training data set size to 5000 sentences, however it didn't make any significant changes apart from significantly slow down the training time. It could be understood that whatever the size is, the natural frequency of words won't change much.

TF-IDF

Next we would try the statistics, "tf-idf", short for term frequencyinverse document frequency to tackle the problem. For each token, we would calculate its frequency in the whole corpus, which is the term frequency, and frequency of sentences containing it, which is our document frequency here. So the tf and itf is defined as:

$$tf(\text{some token}) = \frac{\#\{\text{this token in the whole corpus}\}}{\#\{\text{total tokens of the whole corpus}\}}$$
$$idf(\text{some token}) = \log\left(\frac{\#\{\text{total sentences in the corpus}\}}{\#\{\text{sentences containing this token}\}}\right)$$

At first I tried $tf * idf$ as weight for each token in the vocabulary on the loss function in bptt, but it turned out that it still decreased with word frequency from high to low.

```
tf = words_freq_target/float(np.sum(words_freq_target))
idf = np.log(doc_size/(sentence_freq_target.astype(float)+1))
tf_idf =tf*idf
# Print tf_idf value for the top 20 frequent tokens
print tf_idf[:20]
```

These are the tf_itf value for the first 20 tokens in our German vocabulary, the first to fourth ones are "die", "SENTENCE_END", "." and "das" respectively. "die" and "das" are "the" in German with feminine and neuter gender respectively.

```
[  3.21717349e-02  -9.33142365e-06   2.72031746e-03   3.58719279e-02
   2.63439508e-02   3.66969105e-02   2.41776904e-02   2.38460809e-02
   1.81743099e-02   2.01894652e-02   1.83317942e-02   1.90800832e-02
   1.87701278e-02   1.85686597e-02   1.84003765e-02   1.60827613e-02
   1.87222103e-02   1.65210876e-02   1.68614493e-02   1.67660548e-02]
```

Basically the term frequency dominates the inverse document frequency, so $tf - itf$ can't play a good role in balancing natural frequency. Then we decided that we only used the itf statistic to achieve our goal, which is to assign more emphasis on those less frequent but semantically more important words. In training process, we put weights on gradients respect to each token i at the first beginning:

$$\frac{\partial L_t(i)}{\partial net_t(i)} = idf(i)(y_t(i) - o_t(i)), \quad i = 1, \ldots, C$$

However we are not going to modify the loss formula to a weighted one. Our goal is just to emphasize semantically important words, but there is no difference of priority on the discrepancy of probability between our estimated one and the desired one, while the loss function is based on the Kullback Leibler divergence between two distributions. Here is the training result with our weighted gradients:

```
2017-04-18 18:22:30: Loss after 0 examples used in model and epoch=0: 7.596586
2017-04-18 18:23:12: Loss after 1000 examples used in model and epoch=1: 5.458221
2017-04-18 18:23:50: Loss after 2000 examples used in model and epoch=2: 5.411062
2017-04-18 18:24:30: Loss after 3000 examples used in model and epoch=3: 5.309658
2017-04-18 18:25:10: Loss after 4000 examples used in model and epoch=4: 5.231831
2017-04-18 18:25:44: Loss after 5000 examples used in model and epoch=5: 5.193296
2017-04-18 18:26:18: Loss after 6000 examples used in model and epoch=6: 5.156344
2017-04-18 18:26:54: Loss after 7000 examples used in model and epoch=7: 5.117667
2017-04-18 18:27:33: Loss after 8000 examples used in model and epoch=8: 5.078299
2017-04-18 18:28:12: Loss after 9000 examples used in model and epoch=9: 5.041027
```

This result and the previous result of unweighted gradients are generated on the same random seed, which means they have the same initial value of parameters and loss function. We can see this new training results in quicker decrease of loss function, though not significantly different. This shows that our adjustment work to some extent.

Nevertheless the translation still can't work very well. When it comes to the training process, it may still give incline to predict the most common words. Thus we have to keep using the resampling method and generate sentence. Here is our samples of 2 groups. The first sentence is our source sentence, the second one is our translated sentence and the third one is the target sentence.

we need our governments to put up the financial resources needed to give us effective control and we need them also to stop UNKNOWN_TOKEN their UNKNOWN_TOKEN when it comes to maritime safety .
die geht herr auch ihres jeweiligen wie ich die mitgliedstaaten unserer was themas uns in die ändern ! jetzt einer hilfe 20 den wollen fall untersuchen abgeordnete – equal
wir müssen unsere regierungen dazu UNKNOWN_TOKEN daß sie ausreichend mittel für eine wirksame kontrolle UNKNOWN_TOKEN und wir müssen dafür sorgen daß sie sich UNKNOWN_TOKEN für die sicherheit auf see einsetzen .

the erika cost no human UNKNOWN_TOKEN but it may have UNKNOWN_TOKEN many human UNKNOWN_TOKEN . just as the sea UNKNOWN_TOKEN just as the UNKNOWN_TOKEN did .
die anwendung und sozialen maßnahmen gebiete das tun ich danken eine eine ist sprechen die ein kollegen die kommission des aber .
zwar hat die erika keine UNKNOWN_TOKEN UNKNOWN_TOKEN sie hat jedoch ebenso wie vor jahren die UNKNOWN_TOKEN UNKNOWN_TOKEN und die UNKNOWN_TOKEN u. u. viele menschen in den wirtschaftlichen UNKNOWN_TOKEN UNKNOWN_TOKEN

Discussion

Flaws and Incapacity

I think one factor that might affect the unbalanced probability is the amount of information from resources sentences. Currently we only wrap all information from a target sentence into the final hidden state from the encoder and input it to every hidden state of the decoder. We leave out the location information of each word in the target sentence and not distinguish between phrases. Both of those two missing could lead to poor interpretation and low capacity of our model.

Suggested Solution

RNN with LSTM[2] and GRU[1] mechanism may help treat phrases as integrated parts. They attempt to make the model to "remember" relevant previous information and forget the irrelevant one.

Double direction RNN[3] could help to relate words both before and after some word.

I am also interested in the advanced NMT (Neural Machine Translation) proposed by Than Luong[4][5].He suggested that rare words should be directly "copied" from source sentences to target sentences, which reduces vocabulary size and increase translation efficiency simultaneously.

He also suggested that attention mechanism[4] for long sentence translation. In the attention mechanism, rather than use the same c from an encoder to each time step of a decoder, they use a different context vector c_t for each t, a weighted summation of a (local or global) scope of hidden states from the encoder. The weights are based on similarity of each hidden states from the encoder h_i in the selected scope to the current hidden state in the decoder s_t. Based on different definition of "similarity", different context vectors can be applied. This context vector c_t enables the model to absorb more information centered at the same location in the source sentence when predicting each word in the target sentence. It's prove that this attention mechanism could increase BLEU (bilingual evaluation understudy) performance.

References

[1] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[5] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.