HW3: Translation

Alex Lin
alexanderlin01@college.harvard.edu

Melissa Yu
melissayu@college.harvard.edu

Team Name: PUDGE

March 2, 2018

1 Introduction

With the increasing globalization of our world, translation has become more and more important. Prior to 2014, many automatic translation systems were either based on linguistic rules or statistical models. However, with the emergence of deep learning, neural machine translation systems with end-to-end capabilities now dominate the field.

In this problem set, we examine two such models: 1) a baseline sequence-to-sequence model and 2) an attention-driven deep learning model. Our overall objective is to create translation systems that can translate German sentences to English sentences. For an out-of-sample German test set comprised of 800 sentences, we generate 100 of the most probable 3-gram English starting-sentence translations as determined by our models.

2 Problem Description

Let $\mathcal{V}_X$ be the German vocabulary and $\mathcal{V}_Y$ be the English vocabulary. Let each word in a vocabulary be represented by a one-hot encoded vector with length equal to the vocabulary. Given a German sentence (i.e sequence of words) $\boldsymbol{x} = [x_1, x_2, \dots, x_S]$ and an English partial translation $[y_1, y_2, \dots, y_{t-1}]$, the objective of translation is to accurately predict the distribution of the next English word

$$y_t | y_1, \dots, y_{t-1}, x$$

for $t = 1, 2, \dots, T$ where T is the length of y. Let q be our model's predictive distribution. Let $(\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)})$ be a single pair of German-English sentences that mean the same thing in our training set, for $n = 1, 2, \dots, N$. In training a model, we wish to adjust the model's parameters to minimize the *average loss*, as defined by

$$L = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{T} \sum_{t=1}^{T^{(n)}} \left(\ln q(y_t^{(n)} | y_1^{(n)}, \dots, y_{t-1}^{(n)}, x^{(n)}) \right) \cdot y_t^{(n)}$$

where $\cdot$ denotes the inner dot product and $T^{(n)}$ denotes the length of $\boldsymbol{y}^{(n)}$. The standard metric used for evaluation is called *perplexity*, as defined by

$$PPL = \exp L$$

3 Model and Algorithms

We trained the three different models specified in the instructions. These included (1) a baseline sequence-to-sequence model and (2) an attention-drive sequence-to-sequence model.

3.1 Baseline Sequence-to-Sequence

The baseline sequence-to-sequence model is comprised of an encoder LSTM and a decoder LSTM. The encoder reads an input sentence $\boldsymbol{x}$ with length T_x into a fixed-length context vector c, as follows:

$$\begin{aligned} h_t &= f(x_t, h_{t-1}) \quad \forall t = 1, \ldots, T_x \\ c &= j(\{h_1, \ldots, h_{T_x}\}) \end{aligned}$$

where h_t is the hidden state of the encoder with $h_0 = \mathbf{0}$, and j is a function of the hidden states. In this case, we simply let $j(\{h_1, \ldots, h_{T_x}\}) = h_{T_X}$ (i.e. the last hidden state). Next, we let $s_0 = c$ be the first previous hidden state of the decoder. The decoder follows the recursion:

$$s_t = g(y_t, s_{t-1}) \quad \forall t = 1, \ldots, \tilde{t}$$

where y_t is the t-th word of the supplied *partial* output sentence with length $\tilde{t}$. To predict the probability distribution over the $(\tilde{t}+1)$-th word in the output, we simply have

$$q(y_{\tilde{t}+1} | y_1, \ldots, y_{\tilde{t}}, \boldsymbol{x}) = \text{softmax}(W s_t + b)$$

where W is a $|V_Y| \times n_{\text{hid}}$ matrix and b is an n_{hid}-dimensional vector. Note that

$$n_{\text{hid}} = \dim s_t = \dim h_t$$

is the number of hidden states in each of the two LSTMs.

3.2 Sequence-to-Sequence with Attention

Next, we incorporate attention into the baseline model. We once again experiment with vanilla and bidirectional LSTM's for both the encoder and decoder networks.

The encoder network consists of an embedding layer, $M^{(e)} \in \mathbb{R}^{m_e \times |\mathcal{V}_s|}$, followed by an $N^{(e)}$-layer LSTM with hidden size $H^{(e)}$. The encoder returns the hidden and cell states $(h_S, c_S) \in \mathbb{R}^{N^{(e)} \times H^{(e)}}$ for the last word in the input, along with the the outputs $o_s \in \mathbb{R}^{H^{(e)}}$ at the last LSTM layer for all words in the input.

The decoder network also consists of an embedding layer, $M^{(d)} \in \mathbb{R}^{m_d \times |\mathcal{V}_t|}$, followed by an $N^{(d)}$-layer LSTM with hidden size $H^{(d)}$. The LSTM hidden and cell states $(h_0, c_0) \in \mathbb{R}^{N^{(d)} \times H^{(d)}}$ are initialized using the last "forward" direction encoder hidden and cell states (h_S, c_S). We let $o_t \in \mathbb{R}^{H^{(d)}}$ be the output of the LSTM. We compute the dot product attention over the inputs for each timestep t by taking

$$\alpha_t = \mathcal{S}(f(o_{1:S}, o_t)),$$

where $f = (\boldsymbol{W} o_{1:S} + b) \cdot o_t$; the linear layer $(\boldsymbol{W}, b)$ over the encoder output is necessary to transform its hidden size to match that of the decoder. Using the attention weights, we construct a context vector c_t:

$$c_t = \sum_{s=1}^{S} \alpha_{ts} o_s$$

Then, the probability distribution over our target vocabulary words returned by the decoder is

$$p(y_t \mid y_{<t}, \boldsymbol{x}) = \mathcal{S}(\boldsymbol{W}'[o_t, c_t]^{\mathsf{T}} + b')$$

4 Experiments

We performed several experiments to tune the hyper-parameters of our models. The attention model seemed to perform the best. In-depth explanations of our tuning procedure and associated results can be found in this section. Note that all perplexities reported in table 1 are test set perplexities.

Model	Validation Perplexity	Kaggle Score
BASELINE SEQUENCE-TO-SEQUENCE	13.831	0.24806
SEQUENCE-TO-SEQUENCE WITH ATTENTION	7.55	0.27141

Table 1: Language perplexities and Kaggle scores of our models.

4.1 Baseline Sequence-to-Sequence

The main parameters that we tuned were (1) the embedding dimensions of the two vocabularies, (2) the number of hidden states n_{hid}, (3) the number of hidden layers in each LSTM, (4) the dropout rate, and (5) whether or not to reverse the input sentence $\boldsymbol{x}$.

The model that obtained the best validation perplexity of 13.831 had embedding dimensions of 1000 each, $n_{\text{hid}} = 1000$, 2 hidden layers in each LSTM, a dropout rate of 0.2, and reversed input. We used a stochastic gradient descent optimizer that ran for 16 epochs with an initial learning rate of 1 that was successively halved during the 7th and 14th epochs. The batch size was 32 and the maximum norm of any gradient used during training was set as 5.

Here are our results from varying embedding dimensions (assume all other variables are as listed above). We found that increasing the embedding dimensions too much led to overfitting (i.e. low train perplexity, but higher test perplexity).

Embedding Dimension	100	200	500	1000	1500
Val. Perplexity	15.82	14.52	14.01	13.83	15.27

Here are our results from varying n_{hid}.

n_{hid}	100	200	500	1000	1500
Val. Perplexity	14.11	14.15	13.87	13.83	13.94

Here are our results from varying the number of hidden layers in the two LSTMs.

Number of Hidden Layers	1	2	3	4
Val. Perplexity	18.81	13.87	14.66	14.67

Here are our results from varying the dropout rate.

Dropout Rate	0.1	0.2	0.3	0.4
Val. Perplexity	14.08	13.87	13.92	13.91

And here are our results for deciding whether or not to reverse the input sentence x.

	No Reverse	Yes Reverse
Val. Perplexity	13.99	13.87

To generate predictions for the Kaggle competition, we used beam search with a beam of 100. Our Kaggle score obtained from the baseline sequence-to-sequence model trained with the most optimal parameters is 0.24806.

4.2 Sequence-to-Sequence with Attention

We train the attenton model using full teacher forctthe am optimizer with learning rate 0.001, decaying the rate by 0.5 each epoch after 8 epochs. The choice of optimizer was important, leading to a drop in validation perplexity from 12.83 to 7.55, all else held constant. All gradients are clipped to a maximum total norm of 10. We train for a maximum of 30 epochs, using the validation set perplexity as criteria for early stopping.

We experimented with initializing the embeddings matrices with pretrained vectors (e.g., GloVe, fastText), but found that this did not give a significant performance improvement. Likewise, limiting the max norm for embedding values did not lead to performance gains.

Our final model uses single-direction 2-layer LSTM's with a hidden size of 200 for both the encoder and decoder and a dropout of 0.3 during training. This model achieves a training set perplexity of 8.67 and a validation set perplexity of 7.55. Surprisingly, bidirectional LSTM's performed poorly on this task, overfitting to the training target data.

Figure **??** shows sample attention plots for our final model.

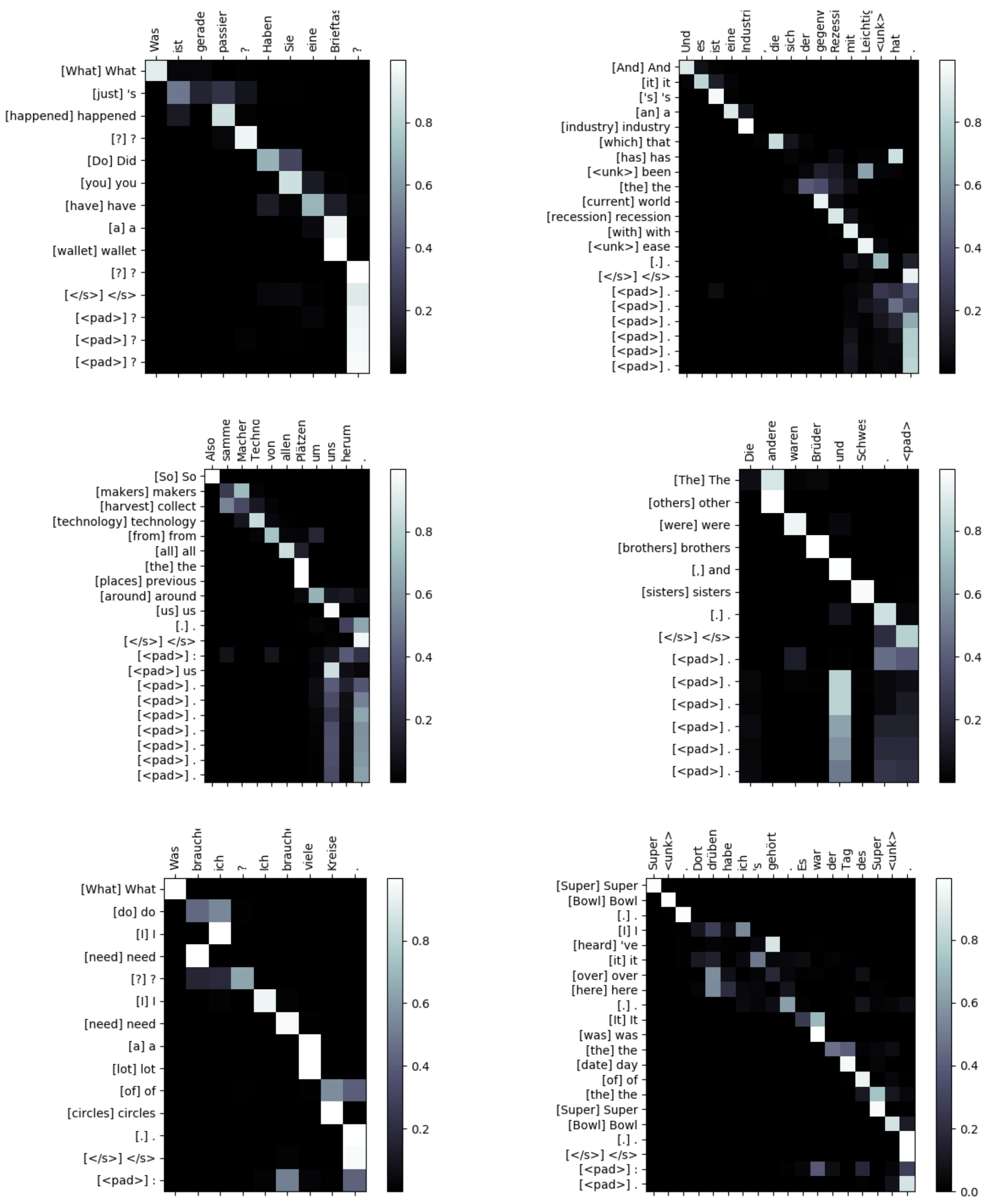

Figure 1: Attention plots for a selection of sentences from the training dataset. The source (German) sentences are shown on the x axis, and the targets are shown on the y axis. The predicted target words are shown perpended with the true word in braces.

5 Conclusion

In this practical, we implemented two models to predict the most likely translations: A baseline LSTM sequence-to-sequence model and a version of this model with dot product attention. We also implemented a beam search algorithm to make predictions using our models. Our experiments on the German-English dataset show that the model with attention performs quite well, achieving a Kaggle public BLEU score of 0.27141 and a validation set perplexity of 7.55.

6 Conclusion

In this practical, we implemented two models to predict the most likely translations: A baseline LSTM sequence-to-sequence model and a version of this model with dot product attention. We also implemented a beam search algorithm to make predictions using our models. Our experiments on the German-English dataset show that the both the baseline model and the model with attention perform quite well, scoring well above the Kaggle baseline.

References

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Zaremba, W., Sutskever, I., and Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.