Attention in Sequence to Sequence Modeling

Dillon Laird

Introduction

We implement a sequence to sequence model using an attention mechanism similar to (Luong et. al., 2015) and try to reproduce their results. We start with an overview of Attention based sequence to sequence models and then describe experimental results as well as technical details of our implementation.

In a neural machine translation model we use a neural network to directly model the conditional probabilities $p(y|x)$ where $y = y_1, \ldots, y_m$ is the target sentence and $x = x_1, \ldots, x_m$ is the source sentence. We train our model to minimize the cross-entropy:

$$J(\theta) = \sum_{(x,y)} -\log p(y|x)$$

Our encoder is a multi-layered recurrent neural network (RNN). We specifically use a long short term memory network for this task (LSTM) but any RNN will work such as a gated recurrent unit (GRU). The RNN takes as input the current word, x_t and the previous hidden states, $\bar{h}_{t-1}$ and spits out a new hidden state, $\bar{h}_t^f$.

$$\bar{h}_t^f = \text{RNN}_{\text{encoder}}(x_t, \bar{h}_{t-1})$$

Here h_t^f is the final hidden state outputed by the last RNN layer while $\bar{h}_{t-1}$ contains the previous hidden states of all the layers. We run the encoder on the entire source sentence and get $[\bar{h}_1^f, \ldots, \bar{h}_T^f]$, $\bar{h}_T$. When running the decoder RNN we initialize it with this final hidden state so:

$$h_1^f = \text{RNN}_{\text{decoder}}(y_1, h_T)$$

During train we feed the correct target word into the decoder while during test time we feed the predicted word back into the decoder. During decoding we caclulate scores:

$$[h_{t'}^T \bar{h}_1, \ldots, h_{t'}^T \bar{h}_T]$$

Where t' is the current decoding time. We used this dot product to calculate the scores instead of using concat $(v_a^T \tanh(W_a[h_{\prime t}, \bar{h}_t]))$ as the model ran much faster and (Luong et. al. 2015) achieved better results using it. We let $a_{t'}(t)$ be a softmax over the above vector. The context vector is then defined as:

$$c_{t'} = \sum_{t=1}^{T} a_{t'}(t)\bar{h}_t$$

Our final output hidden state after attention is computed by $\tilde{t}_{t'} = \tanh(W_c[c_{t'}; h_{t'}])$.

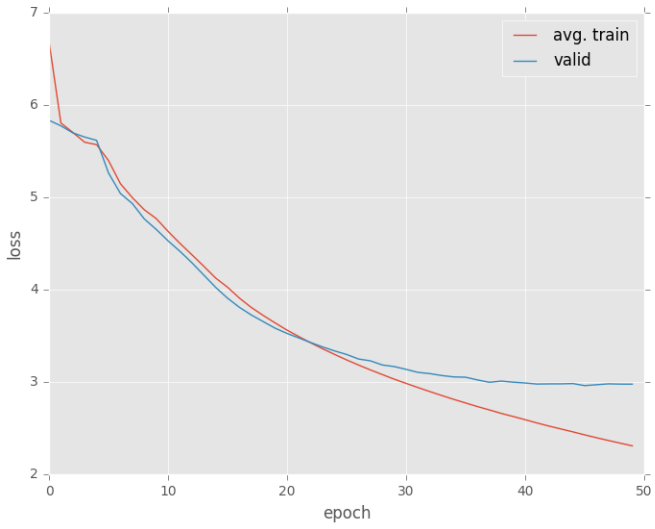
Experiments

For our experiments we used a smaller model than the one used in (Luong et. al. 2015) to cope with memory constraints. We also used an extra matrix projection in both the input and output to help reduce memory. Thus to get our input vector for the encoder and decoder we did $W_{eh}(W_{ie}x + b_{ie}) + b_{eh}$ where the input to embedding matrix was $W_{ie} \in \mathbb{R}^{|V| \times E}$ and our embedding to hidden was $W_{eh} \in \mathbb{R}^{E \times H}$. We do a similar projection when going from the output hidden vectors to the output logits where we go from hidden to embedding size to output logit. This reduces the number of parameters from $|V| \times H$ to $|V| \times E + E \times H$ where $E < H$.

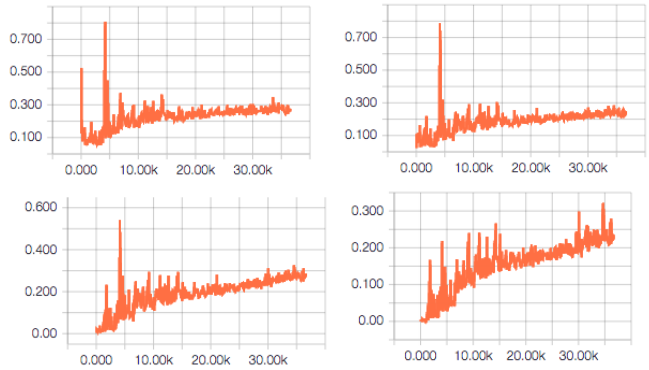
We trained on the Vietnamese dataset. We used a maximum sentence length of 30 and prune the dataset of all sentences with length greater than 30. We trained on $93,913$ data points and used a validation set of 500. We use an embedding size of 256, a hidden size of 512 with 4 layers. We use a batch size of 124 and a dropout of 0.2. We ran the model for a total 50 epochs. We start the learning rate at 1.0 adn then start decaying by 0.75 every 5 epochs after 10 epochs. We decay until the learning rate goes below 0.0025.

For training we reversed the words in the sentence and pre padded the sequence so it was length 30. We then used a weighted cross entropy loss to give pads a weight of 0.

We reached an average training error of 2.31 on our last epoch and our best validation error was 2.96*. We should note that the reason the validation error is so much lower than the test error is because we had to use the tensorflow graph for training to calculate the validation error. So the validation error was from feeding in the true outputs rather than feeding in the predicted outputs. This should not affect saving the best model based on validation that much because we would expect it to perform better than other validation losses regardless of whether it is being fed the true labels after prediction or the predicted labels. We achieve a test loss of 7.78 on tst2013 and a BLEU score of 8.29.



The graph above shows the training loss vs the validation. The reason the validation is sometimes lower than the training is because the size of the validation set is small and validation error was calculated by feeding the true labels instead of the predicted, as explained above. We used 2 very basic learning rate strategies. One in which we kept the learning rate constant at 1.0 and another where we annealed by 0.75 every 5 epochs after 10 epochs until it dipped below 0.00025. Both yieled similar results.

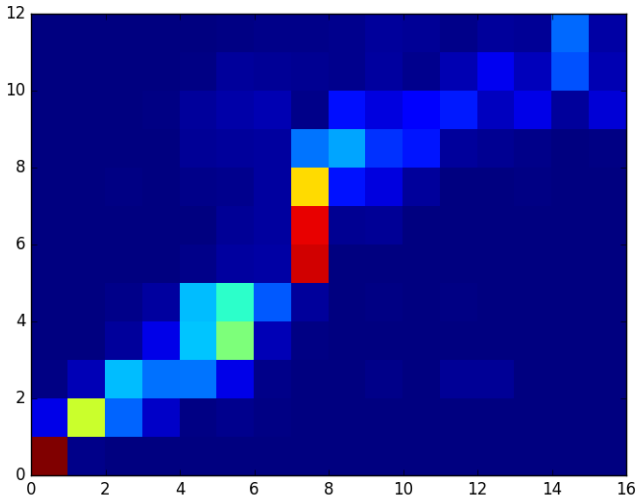


Hình 1: Decoder Gradient Norms

The figure above shows the gradient norms for all the layers of the decoder over the number of iterations. You can see here we didn't suffer from vanishing or exploding gradients. Gradients were cut off at 5 when their norm exceeding 5 but it seems this never happened.

We used the model to translate the following sentence, "The answer to the ultimate question of life, the universe, and everything"and got "Câu trả lời cho các câu hỏi chính là sự thay đổi của thế giới và sự khác biệt"which roughly translates to "The answer to the main question is the change of the world and the differences.".

Our model that got the best BLEU score was trained with 512 hidden units, 256 embedding size and only 1 hidden layer. It was trained for 50 epochs and achieved a BLEU score of 17.31. Here's a heatmap of the attention probabilities for a simple example. The x axis represents the source sentence and the y axis represents the target sentence.



References

[Luong et. al.2015] Minh-Than Luong, Hieu Pham, Christopher D. Manning. 2015. Effective Approaches to Attention-based Neural Machine Translation.