HW4: All about Attention

Jiafeng Chen Yufeng Ling Francisco Rivera

April 2, 2019

1 Introduction

In this writeup we consider neural machine translation—given a source sentence in one language, can we generate a target sentence in another? We work with the prevailing encoder-decoder architecture, where one neural network, the *encoder*, maps the source sentence into a numerical tensor—which is supposed to be a latent representation of the source sentence. Another network, the *decoder*, is a language model that takes the encoded sentence as an input, and generates a sentence in the target language.

2 Problem Description

In this writeup, we consider the problem of machine translation. Let

$$\boldsymbol{x}_i = [x_{i1}, \ldots, x_{iS}] \tag{1}$$

be a source sentence, where each word belongs to some source vocabulary, $x_{it} \in \mathcal{V}_s$. Let

$$\boldsymbol{y}_i = [y_{i1}, \ldots, y_{iT}] \tag{2}$$

be a target sentence, with each target word belonging to some target vocabulary, $y_{it} \in \mathcal{V}_t$. Our goal is to learn $p(\boldsymbol{y}_i \mid \boldsymbol{x}_i)$. We often treat the prediction like a language model—i.e. we consider the sequential conditional distributions $p(y_{it} \mid y_{i1}, \ldots, y_{i,t-1}, \boldsymbol{x}_i)$.

3 Model and Algorithms

3.1 Sequence-to-Sequence (Seq2Seq)

In Seq2Seq (Sutskever et al. (2014)), we have a encoder-decoder network architecture. The *encoder*, in the vanilla Seq2Seq implementation, is an LSTM network that takes an embedding of the source sentence $\mathsf{embed}(\boldsymbol{x}_{i1}), \ldots, \mathsf{embed}(\boldsymbol{x}_{iS})$ and outputs a list of hidden states $\boldsymbol{h}_{i1}, \ldots, \boldsymbol{h}_{iS}$ and cell state $\boldsymbol{c}_{iS}$.

In Seq2Seq, the decoder is another LSTM which is initialized with $\boldsymbol{h}_{iS}$ and $\boldsymbol{c}_{iS}$. At training time, the decoder LSTM takes embeddings of the ground truth target $\mathsf{embed}(y_{it})$ and output probability predictions for $y_{i,t+1}$. These predictions are penalized with the usual cross entropy loss at training time. At prediction time, the decoder LSTM gets passed the start-of-sentence token <s> and outputs predictions for the first word. We then iteratively pass in its top predictions to obtain predictions for future words. For instance, a greedy algorithm to generate a sentence would be taking the top prediction every time and pass the predicted sentence so far into the decoder to obtain the next word—stopping when the end-of-sentence token </s> is the top prediction.[1]

3.2 Attention

The attention model (Vaswani et al. (2017)) builds upon the Sequence-to-Sequence model described in section (3.1). Instead of only using the end state of the encoded input as input, we take advantage of a bi-directional RNN and generate target sentence by taking in a weighted input of all the states of the encoder. The intuition is that when we are translating a sentence, the adjacent words of the current word also help inform how we should translate the current word. Such weighted average is aptly named "context". To formalize this, we let the sequential conditional distribution be given by

$$p(y_{it} \mid y_{i1}, \ldots, y_{i,t-1}, \boldsymbol{x}_i) = g(y_{i,t-1}, s_t, c_t) \tag{3}$$

[1] This corresponds to beam search with beam size 1.

where s_t is the hidden state from BiRNN at time t defined by

$$s_t = f(s_{t-1}, y_{i,t-1}, c_t). \tag{4}$$

As mentioned above, the context vector c_t is a weighted average given by

$$c_t = \sum_{j=1}^{T_x} \alpha_{tj} h_j \tag{5}$$

where the annotations $\{h_j\}$ are the concatenated hidden states of the BiRNN and $\{\alpha_{tj}\}$ are the weights, which usually center around the current state t. The weights

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T_x} \exp(e_t k)} \tag{6}$$

are generated by softmaxing over

$$e_{tk} = a(s_{t-1}, h_k) \tag{7}$$

for some function a. In our model, we take a to be the dot product between the two arguments.

3.3 Beam Search

After we have trained a model, since the outputs are conditional probabilities over the entire vocabulary, we need to have an algorithm that generates the top k most likely sentences. One can use the greedy algorithm by selecting the local MLE conditional on the previously selected words. However, this can easily run into problems like stucking at a local optimum.

Beam search is proposed based on this idea. Instead of selecting the MLE, we select the top B words with the largest conditional probabilities. The parameter B is known as the "beam width". At every step, we choose the top B words with the highest joint probability conditional on the B previously generated sentenses $\{\boldsymbol{y}_{i,1:t-1}^{(b)}\}_{b=1}^{B}$. That is

$$\arg\max_{y_{it},b} p(y_{it}, y_{i,t-1}^{(b)}, \ldots, y_{i1}^{(b)}, \boldsymbol{x}_i) = \arg\max_{y_{it},b} p(y_{it} \mid y_{i,t-1}^{(b)}, \ldots, y_{i1}^{(b)}, \boldsymbol{x}_i) p(y_{i,t-1}^{(b)}, \ldots, y_{i1}^{(b)}, \boldsymbol{x}_i).$$

In the last step we choose the top k to report as results.

3.4 Transformer extension

We also implemented the *transformer* architecture, following Vaswani et al. (2017). Each transformer layer is a self-attention layer,[2] a source-attention layer (only in the decoder), and a fully-connected feedforward network, connected by residual connections[3] and layer normalization.[4] We transform the input via a dense embedding and a *positional embedding*—treating position in a sentence as tokens in a vocabulary, and map to a dense embedding. We concatenate the embeddings and feed into the transformer layers. The model architecture is diagramed in **??**.

We trained two transformer models, which we name `Megatron` and `Soundwave`, after beloved *Transformers* antagonists. The `size` parameter is the number of dimensions of the dense embeddings, input to layers, and output from layers (so that the layers plug into each other). The `pos`, `emb` parameters are the number of dimensions in the positional and the word embedding

```
# Megatron
size = 200
pos = 20
emb = 180
head = 4
nlayers = 3
dropout = .2

# Soundwave
size = 200
pos = 20
emb = 180
```

[2]The attention layers are *multiheaded*, which means that we transform the (query, key, value) inputs by a linear layer before using dot-product attention (and normalization by $\sqrt{d}$. We then concatenate the resulting context vectors and feed into one last linear layer).

[3]A residual connection of a layer adds the input to the layer output. This helps with gradient propagation

[4]Layer normalization standardizes a layer's output along dimension of the dense embedding (i.e. the dimension other than the batch and seqlen dimensions).

```
head = 3
nlayers = 4
dropout = .1
```

4 Experiments

We trained two models and document hyperparameters in Table 1. Results are displayed in Table 2. The performance of the Sequence-to-Sequence model in Sutskever et al. (2014) is reasonable, but understandabely weaker due to its simple structure of combining two LSTMs without any attention layer applied to the source sentence.

In the attention model proposed in Bahdanau et al. (2014), we used 1/5 of the number of parameters compared to the paper due to the constraint of training speed. The main difference is that instead of a 5-layer decoder, we used one layer. Compared to the previous Sequence-to-Sequence model without attention, the performance improved significantly as seen in Table 2. We experimented normalizing the log-attention weights by factor of $\sqrt{\text{embedding size}}$ before proceeding to the softmax function to calculate the attention weights; this is because with more dimensions, dot product tends to be larger, and the softmax tends to be sharper, which would result in small gradients. However, this resulted in uniform weights across all source words and very poor performance. In the properly trained model, we can observe the attention weights reflecting our intuition. See Figure **??** and **??** for visualization of the attention weights.

It is also somewhat surprising to us that transformer model in Vaswani et al. (2017) does not perform well relative to LSTM-based models, contradicting established wisdom. We suspect that the reason is due to either poor implementation/optimization or short sentences and insufficient computing power. Attention is powerful when sentences are long, since it allows the model to look at arbitrarily long dependencies, and attention scales well with GPU hardware and multi-GPU training—we would get neither of these advantages with a small dataset and Google Colab computing power (Training `Megatron` takes 15 minutes per epoch and training `Soundwave` takes over 20 minutes, which does not leave room for hyperparameter tuning).

model name	specifications
`Seq2Seq`	100 embedding, 100 hidden, no dropout
`Seq2SeqAttn`	Bi-directional LSTM, 100 embedding, 100 hidden, no dropout

Table 1: Both models used only one layer in the decoder and trained with Adam and learning rate 10^{-3} *over 10 epochs. For* `Seq2Seq`*, decrease in validation loss flattened at epoch 8. For* `Seq2SeqAttn`*, training stopped at epoch 7 after loss starting going up.*

model name	Loss	Perplexity	BLEU
`Seq2Seq`	3.23	25.36	6.32
`Seq2SeqAttn`	2.71	15.05	17.77
`Megatron`	2.7	15	9.68
`Soundwave`	2.8	16	7.08

Table 2: Performance metrics for different models

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008.

A Model implementation