Neural Machine Translation by Jointly Learning to Align and Translate

1 Citation

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).

`https://arxiv.org/pdf/1409.0473.pdf`

2 Abstract

The encoder-decoder network is good for translation, but it is limited by the fact that the encoder must produce a fixed-length vector. We remove this limitation with an attention mechanism that lets the decoder focus on relevant parts of the input sentence. Our model is comparable with the state of the art Statistical Machine Translation (SMT) system and qualitatively does well on long sentences.

3 Introduction

Prior work showed that the encoder-decoder network struggles with long sentences. Our model's encoder creates a variable length annotations and uses an attention mechanism to help the decoder decide which annotations to focus on. Our system is trained end-to-end on English to French translation.

4 Background: Neural Machine Translation

We seek to find $p(\mathbf{y}|\mathbf{x})$ for input sentence $\mathbf{x}$ and output sentence $\mathbf{y}$. We train on parallel training corpus and we use beam search at test time. The encoder-decoder system is used to solve this problem. On its own, it does really well. When used to re-rank SMT phrase pairs, it sets state of the art.

5 RNN Encoder-Decoder

The encoder turns input sequence $\mathbf{x} = (x_1, ..., x_{T_x})$ into vector c with:

$$h_t = f(x_t, h_{t-1}) \text{ and } c = q(\{h_1, ..., h_{T_x}\})$$

Previous work uses a Long Short-Term Memory (LSTM) for f and sets $c = h_{T_x}$. The decoder then does this:

$$p(\mathbf{y}) = \prod_{i=1}^{T} p(y_t|\{y_1, ..., y_{t-1}\}, c) = \prod_{i=1}^{T} g(y_{t-1}, s_t, c)$$

6 Learning to Align and Translate

We model our decoder with $p(y_i|\{y_1, ..., y_{i-1}\}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$. We set $s_i = f(s_{i-1}, y_{i-1}, c_i)$. Notice we have one c_i per output word, rather than one shared c. Basically, the encoder computes an annotation sequence (one annotation per input word): $h_1, ..., h_{T_x}$. We then compute:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

where $\alpha_{ij} = \text{softmax}(e_{ij})$ and $e_{ij} = a(s_{i-1}, h_j)$. The alignment model a is just a feedforward network. Notice the α_{ij} make up a probability distribution over the annotations.

Our encoder is a bidirectional Recurrent Neural Network (RNN). It has a forward piece $\overrightarrow{f}$ that reads the sentence and produces $(\overrightarrow{h}_1, ..., \overrightarrow{h}_{T_x})$ and a backward piece $\overleftarrow{f}$ that reads the sentence backwards and produces $(\overleftarrow{h}_1, ..., \overleftarrow{h}_{T_x})$. We then concatenate to get $h_j = [\overrightarrow{h}_j; \overleftarrow{h}_j]$.

7 Experiment Settings

We use the WMT'14 English to French dataset. We use a special dataset selection technique to pick the dataset of 348M words. Some work pretrains on an all-English or all-French dataset, but we only use the parallel dataset. Our vocabulary has 30K words.

We train an encoder-decoder and our NMT system (each with a 30 and 50 variant, which represents the maximum sentence size in training). We have 1000 hidden units. We use a maxout hidden layer and softmax to compute output probabilities. Beam search identifies the best translation.

8 Results

Our 50-word NMT system matches the Moses SMT on BLEU score. It beats encoder-decoder and hugely outperforms it on long sentences. Looking at a visualization of the alignment probabilities - they make sense.

9 Related Work

Prior work used an attention mechanism on handwriting synthesis. Our approach is richer because we generate one annotation per word, but it may be harder to scale to extremely long input sequences.

Others have used neural networks to rescore SMT phrase pairs.

10 Conclusion

We make an encoder-decoder with an attention mechanism instead of a fixed-length context vector. Our model matches SMT performance and does great on long sentences.

11 Appendix A: Model Architecture

We use the gated hidden unit (it's similar to an LSTM) as our f function. That is,

$$
\begin{aligned}
s_i = f(s_{i-1}, y_{i-1}, c_i) &= (1 - z_i) \odot s_{i-1} + z_i \odot \tilde{s}_i \\
\tilde{s}_i &= \tanh\left(We(y_{i-1}) + U[r_i \odot s_{i-1}] + Cc_i\right) \\
z_i &= \sigma(W_z e(y_{i-1}) + U_z s_{i-1} + C_z c_i) \\
r_i &= \sigma(W_r e(y_{i-1}) + U_r s_{i-1} + C_r c_i)
\end{aligned}
$$

where $e(y_{i-1})$ is the embedding of y_{i-1}. z_i and r_i are the update and reset gates, respectively. Embedding is just done by multiplying an embedding matrix E with the one-hot encoded word.

To turn our decoder's hidden state into an output distribution, we use a hidden layer of maxout units followed by a softmax.

Our alignment model needs to be fast, so we use a small multilayer perceptron (we can precompute $U_a h_j$).

$$
a(s_{i-1}, h_j) = v_a^T \tanh\left(W_a s_{i-1} + U_a h_j\right)
$$

The forward RNN in our encoder is defined as follows (the backwards RNN is defined analogously):

$$
\begin{aligned}
\overrightarrow{h}_i &= (1 - \overrightarrow{z}_i) \odot \overrightarrow{h}_{i-1} + \overrightarrow{z}_i \odot \overrightarrow{p}_i \\
\overrightarrow{p}_i &= \tanh\left(\overrightarrow{W}\bar{E}x_i + \overrightarrow{U}[\overrightarrow{r}_i \odot \overrightarrow{h}_{i-1}]\right) \\
\overrightarrow{z}_i &= \sigma(\overrightarrow{W}_z\bar{E}x_i + \overrightarrow{U}_z\overrightarrow{h}_{i-1}) \\
\overrightarrow{r}_i &= \sigma(\overrightarrow{W}_r\bar{E}x_i + \overrightarrow{U}_r\overrightarrow{h}_{i-1})
\end{aligned}
$$

Our decoder has:

$$
\begin{aligned}
s_i &= (1 - z_i) \odot s_{i-1} + z_i \odot \tilde{s}_i \\
\tilde{s}_i &= \tanh\left(WEy_{i-1} + U[r_i \odot s_{i-1}] + Cc_i\right) \\
\tilde{z}_i &= \sigma(W_z E y_{i-1} + U_z s_{i-1} + C_z c_i) \\
\tilde{r}_i &= \sigma(W_r E y_{i-1} + U_r s_{i-1} + C_r c_i)
\end{aligned}
$$

The probability of a target word is modeled by a maxout layer and softmax:

$$
\begin{aligned}
p(y_i | s_i, y_{i-1}, c_i) &= \text{softmax}(y_i^T W_o t_i) \\
t_i &= [\max\left(\tilde{t}_{i,2j-1}, \tilde{t}_{i,2j}\right)]_{j=1\ldots l}^T \\
\tilde{t}_i &= U_o s_{i-1} + V_o E y_{i-1} + C_o c_i
\end{aligned}
$$

12 Appendix B: Training Procedure

Recurrent weight matrices are initialized as random orthogonal matrices. Alignment model parameters are initialized from zero-mean Gaussians. Bias vectors (omitted in equations for conciseness) are initialized at zero. Other weights come from zero-mean Gaussian.

We train with Adadelta ($\epsilon = 10^{-6}, \rho = 0.95$). We clip gradients to have L2 norm = 1. We used minibatches of 80 sentences. Since runtime is proportional to sentence length, we sampled 1600 sentences, sorted them by length, and created 20 minibatches so that shorter sentences wouldn't have to wait too long for longer sentences to finish processing.