End-to-end Attention-based Large Vocabulary Speech Recognition

1 Citation

Bahdanau, Dzmitry, et al. "End-to-end attention-based large vocabulary speech recognition." Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on. IEEE, 2016.

`https://arxiv.org/pdf/1508.04395.pdf`

2 Abstract

We make an end-to-end neural network for speech recognition that is almost state of the art. It uses an attention mechanism (made faster by considering only the most promising frames and information from neighboring frames) to figure out the alignment between utterance and output. We also integrate a language model into our system.

3 Introduction

Traditional speech recognition systems are GMM-HMMs. Recent work have replace the GMM with an deep neural network (DNN). To train the DNN, we need a training set where the utterance is aligned with the per-frame outputs. The GMM-HMM is applied to the input to generate the alignment. Two ways to get around this limitation is to use an Attention-based Recurrent Sequence Generator (ARSG) or to train with the Connectionist Temporal Classification (CTC) loss.

Our neural network does speech recognition with attention (sped up with pooling and windowing) and an integrated langugage model (based on Weighted Finite State Transducers (WFST)).

4 Attention-based Recurrent Sequence Generators for Speech

We use an encoder-decoder Recurrent Neural Network (RNN), where the attention mechanism is in the decoder (i.e it is an ARSG). An RNN turns an input sequence $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$ into $(\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T)$. We can then stack a softmax layer to make predictions (note g is usually tanh). In our case, we use a Gated Recurrent Unit (GRU) instead of a vanilla RNN. The equations are:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{xz}\mathbf{x}_t + \mathbf{U}_{hz}\mathbf{h}_{t-1}) \tag{1}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr}\mathbf{x}_t + \mathbf{U}_{hr}\mathbf{h}_{t-1}) \tag{2}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{U}_{rh}(\mathbf{r}_t \otimes \mathbf{h}_{t-1})) \tag{3}$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t)\mathbf{h}_{t-1} + \mathbf{z}_t\tilde{\mathbf{h}}_t \tag{4}$$

To incorporate both future and past information, we have a bidirectional RNN (i.e. we have one GRU run from start to and and another run the opposite direction and we concatenate their hidden state vectors h_t).

We also stack these bidirectional RNNs. That is, $(\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_T)$ becomes the input sequence to another bidirectional RNN.

Now let's look at the encoder-decoder architecture, which lets us turn a variable length into a variable length output. Our encoder is simply a stacked bidirectional RNN, which produces a hidden state for each input vector in the utterance. This ends up being a lot of vectors, so we use pooling to cut the number of hidden states in half.

The decoder is an ARSG. The decoder has an attention mechanism, which basically means that it computes a vector $\mathbf{c}_t = \sum_l \alpha_{tl} \mathbf{h}_l$ and then feeds $\mathbf{c}_t$ to a fully connected layer and softmax layer. The α_t vector is a probability distribution over the encoder's hidden states, indicating what we should pay attention to. The equations to compute α_t are (here $\mathbf{s}_{t-1}$ is the previous hidden state of the decoder and $*$ is convolution - this convolution step is critical):

$$\mathbf{F} = \mathbf{Q} * \alpha_{t-1} \tag{5}$$

$$e_{tl} = \mathbf{w}^T \tanh\left(\mathbf{W}\mathbf{s}_{t-1} + \mathbf{V}\mathbf{h}_t + \mathbf{U}\mathbf{f}_l + \mathbf{b}\right) \tag{6}$$

$$\alpha_{gl} = \exp(e_{tl}) / \sum_{l=1}^{L} \exp(e_{tl}) \tag{7}$$

The problem with the above approach is that summing over $1...L$ is expensive. To mitigate this, we only consider the window $[m_{t-1} - w_l, m_{t-l} + w_r]$, where m_{t-1} is the median of probability distribution α_{t-1} and w_l and w_r are window offsets. At the start of training, we need to give the system a hint about what a good window looks like, so we start it with $[s_{min} + tv_{min}, s_{max} + tv_{max}]$, where s_{min} (s_{max}) are the lower (upper) bound on the number of leading silent frames and v_{min} (v_{max}) are lower (upper) bounds on ratio of transcript and encoded input lengths (estimated from training data).

5 Integration with a Language Model

Language models are defined over words, our our ASRG outputs characters. To merge the two, we use a WFST, which lets us build a character level language model from a word level language model. A WFST is basically a state machine with learnable transition parameters that figures out all the possible ways produce the output characters given the input characters. Once we do this, we can then have our test-time decoding algorithm (beam search) aim to find the sequence that minimizes (for utterance length T):

$$L(x, y) = -\log p_{EncDec}(y|x) - \beta \log p_{LangMod}(y) - \gamma T \tag{8}$$

6 Related Work

CTC has worked well for speech recognition. It can also be merged with a language model. An RNN Transducer extends CTC to an encoder-decoder setting (no attention mechanism though). Unlike an ARSG, an RNN Transducer does not explicitly compute an alignment.

7 Experiments

We train and evaluate on the Wall Street Journal corpus (81 hours of speech). We compute 40 mel-scale filterbank coefficients (and energy) along with their first and second temporal derivatives (so each vector in the utterance is a 123-length feature vector). The utterance is broken into 10 ms chunks. We consider a 20K word vocabulary, a bigram and trigram language model, and a 32 character output vocabulary.

Our model has 4 layers with 250 units in each of the GRUs (forward and backward). THe top two layers only read every other hidden state of the layers below (this is our pooling). We initialize weights from an isotropic Gaussian and train with Adadelta with gradient clipping. We set $w_L = w_R = 100$. To regularize, we use the column norm 1 contraint on all weight matrices. For beam search, we use a beam size of 200 (using a beam size of 10 only gets a 10% relative degradation though).

We compare against CTC systems. The CTC systems benefit muchh more from a language model than our system does because an ARSG implicitly learns a language model.

8 Discussion

Since our dataset was small, we think our ARSG's implicit language model may have been overfitted. Training on a larger corpus should fix this. Our model does not beat the CTC models, but does pretty well.

9 Conclusion

We use an encoder-decoder network with an attention mechanism (with pooling and windowing) integrated with a language model (via WFST) to do speech recognition almost as well as CTC models.