Deep Speech 2: End-to-End Speech Recognition in English and Mandarin

1 Citation

Amodei, Dario, et al. "Deep speech 2: End-to-end speech recognition in english and mandarin." International Conference on Machine Learning. 2016.

`https://arxiv.org/pdf/1512.02595.pdf`

2 Abstract

Our neural network does speech recognition for English and Mandarin and is competitive with Amazon Mechanical Turk workers. We describe how to efficiently train and deploy this model.

3 Introduction

Traditional speech recognition systems don't generalize to other languages, get poor accuracy, and have many complex pieces - our neural net does not have any of these problems.

We stack a softmax on fully connected layers on Recurrent Neural Net (RNN) layers (with batch normalization) on convolutional layers on spectrogram input and train with Connectionist Temporal Classification (CTC) loss. We train on almost 12K hours of English and 10K hours of Mandarin (we also do data augmentation). We have an optimized system that trains this on multiple GPUs. Our Batch Dispatch technique helps deploy these models (they run in 67 ms).

4 Related Work

People have used Convolutional Neural Nets (CNN) and Long Short-Term Memory (LSTM) RNNs for speech recognition. The encoder-decoder architecture (with or without attention) is popular. The alternative to encoder-decoder is training with the CTC loss and outputting the grapheme (instead of phoneme). Usually, you need to initialize this network with a frame-level alignment (from a GMM-HMM), but we don't need to do that in our case. People augment their dataset with additive noise (or simulating changes in vocal tract length). Once trained, a model can be used to align a larger dataset and then train on that dataset (bootstrapping).

5 Model Architecture

We need many layers in our models, so we use Batch Normalization and a novel optimization curriculum called SortaGrad. We use bidirectional RNNs, but unidirectional RNNs also work well (and you need these for realtime speech recognition because you don't have the full audio sequence).

Our input is $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^{N}\}$ where $(\mathbf{x}_t^{(i)})_{t=1}^{T^{(i)}}$ is the sequence of spectograms from the utterance and $\mathbf{y}^{(i)}$ is the grapheme (e.g. a, b, c, ..., z, space, apostrophe, blank).

Let $h_0 = \mathbf{x}$ be the network input. We apply conv layers: $h_{t,i}^l = f(w_t^l \odot h_{t-c:t+c}^{l-1})$ where l is the layer and i is the activation. Our nonlinearity, f, is clipped ReLU $f(x) = \min(\max(x, 0), 20)$. After the conv layers, we can apply a bidirectional RNN: $\overrightarrow{h}_t^l = g(h_t^{l-1}, \overrightarrow{h}_{t-1}^l)$ (we define $\overleftarrow{h}_t^l$ analogously). We combine this backward and forward RNN with $h^l = \overleftarrow{h}^l + \overrightarrow{h}^l$. g can be an LSTM or Gated Recurrent Unit (GRU). After the bidirectional RNN layers, we apply fully connected layers: $h_t^l = f(W^l h_t^{l-1} + b^l)$. The output is simply a softmax layer

$$p(l_t = k|\mathbf{x}) = \frac{\exp(w_k^L \cdot h_t^{L-1})}{\sum_j \exp(w_j^L \cdot h_t^{L-1})} \tag{1}$$

We train with CTC loss. We decode with a language model and beam search. To speed up training, we use Batch Normalization (this is theorized to mitigate internal covariate shift) on the recurrent part of the bidirectional RNN, like this:

$$\overrightarrow{h}_t^l = f(\mathcal{B}(W^l h_t^{l-1}) + \overrightarrow{U}^l \overrightarrow{h}_{t-1}^l) \tag{2}$$

$$\mathcal{B}(x) = \gamma \frac{x - E[x]}{(Var(x) + \epsilon)^{1/2}} - \beta \tag{3}$$

for learned γ, β, fixed ϵ, and $E[x]$, $Var(x)$ computed on the minibatch. At deployment time, we don't have a minibatch, so we keep a running mean and variance at each neuron.

The CTC cost is defined as follows (where $Align(x, y)$ is all possible alignments of utterance x and grapheme sequence y).

$$\mathcal{L}(x, y; \theta) = -\log \sum_{l \in Align(x,y)} \prod_t^T p_{ctc}(l_t|x; \theta) \tag{4}$$

Since this cost increases with T, we do the first epoch of training in order of increasing sequence length. Then, we use a random order for later epochs. We call this SortaGrad and it improves Word Error Rate (it's complementary with Batch Norm).

For g, we found that a GRU trains faster and converges better than an LSTM. Our GRU equations are (for σ = sigmoid and f = clipped ReLU):

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{5}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{6}$$

$$\tilde{h}_t = f(W_h x_t + r_t \odot U_h h_{t-1} + b_h) \tag{7}$$

$$h_t = (1 - z_t) h_{t-1} + z_t \tilde{h}_t \tag{8}$$

We use one to three layers of convolution. We use both frequency convolution (2D invariant) and time convolution (1D invariant). The former does better on noisy data. We have stride on the convolution to reduce the length of the sequences. This can hurt English accuracy, so we predict bigrams rather than graphemes and we segment our words into bigrams, suffixing with a pad if they stand alone. So the grapheme sequence (t, h, e, ¡space¿, m, o, t, h) would become (th, e ¡pad¿, ¡space¿ ¡pad¿, mo, th).

Since a bidirectional RNN requires a full audio sequence, it can't be used in realtime applications. So, we can use a unidirectional RNN that looks forward only τ timesteps. Our feature matrix is thus:

$h_{t:t+\tau} = [h_t, h_{t+1}, ..., h_{t+\tau}]$. Then, we use our usual matrix W, but we compute the activations with a row convolution as follows (this row convolution follows every RNN layer):

$$r_{t,i} = \sum_{j=1}^{\tau+1} W_{i,j} h_{t+j-1,i} \text{ for } 1 \leq i \leq d \tag{9}$$

Although our neural net learns an implicit language model, we combine it with pretrained Kneser-Ney smoothed 5-gram models. At decoding time, we use beam search to find the sequence that maximizes:

$$Q(y) = \log(p_{ctc}(y|x)) + \alpha \log(p_{lm}(y)) + \beta \text{wordCount}(y) \tag{10}$$

6 System Optimization

We wrote an efficient GPU deep learning library that runs on 8 NVIDIA Titan GPUs. We use data parallelism and synchronous SGD (since nondeterminism is a good way to detect bugs, we don't use asynchronous SGD). We wrote our own all-reduce implementation to exchange data between GPUs. We also implement our own CTC loss function GPU implementation so it can be parallelized easily. We also wrote our own memory allocator to fallback to external memory for long utterences that don't fit in GPU memory. We also have a faster memory allocation algorithm.

7 Training Data

We train on about 12K hours of English and 10K hours of Mandarin - both are internal company datasets. The English data consists of audio that can sometimes be many minutes to hours long, so we need to split into pieces. So, we apply an existing bidirectional RNN model trained with CTC to compute the best alignment with the Viterbi algorithm:

$$l^* = \text{argmax}_{l \in Align(x,y)} \prod_t^T p_{ctc}(l_t|x, \theta) \tag{11}$$

With this alignment, we cut the sequence whenever we find a "long-enough" sequence of ¡blank¿ tokens. We tune "long-enough" so we get sequences of about 7 seconds long. Next, we need to remove the bad samples, so we train a linear classifier to mark a sample as good or bad (ground truth comes from crowdsourcing a few thousand samples). Our features are CTC cost, CTC cost normalized by sample length (or transcript length), edit distance, etc.

For data augmentation, we add noise to 40% of the utterances.

To understand how more data improves the model, we plot word error rate (WER) as a function of dataset size. We find that each 10x increase in dataset size comes with about a 40% relative decrease in WER. Having different speakers, background noises, microphones, etc in the dataset is very important.

8 Results

We evaluate on internal test sets and public benchmarks. We train for 20 epochs with SGD with momentum (.99). The minbatch size is 512. We cross validate to pick the learning rate and anneal it. Beam size is 500 for English and 200 for Mandarin.

Our best English model has 3 conv layers, 7 RNN layers, and 1 fully-connected layers. We measure it's word error rate by comparing it against ground truth collected from Amazon Mechanical Turk. For extremely deep models, a regular RNN beats GRU and LSTM.

We consider a dataset of audiobooks and news reports. We are competitive with human performance. This speech is very clear, so it's not hard to do well on.

Voxforge has accented speech. Human performance beats our model for everything except the Indian accent. The CHiME dataset has speech from busy (noisy) environments or with synthetic noise. Our model does better on synthetic noise, but human performance wins in both kinds of noisy speech.

9 Deployment

The system described so far is not good for realtime applications because it uses bidirectional RNNs, uses a big beam (expensive) for beam search, and normalizes spectograms with power (requires whole utterance). The power normalization is solved by learning parameters to do it (like for batch norm).

Processing one utterance at a time is wasteful, so we have a Batch Dispatch system that buffers user requests into batches and sends batches to the GPU for processing all at once.

We use 16-bit (rather than 32-bit) floats at test-time, which saves memory. We also use smaller batches. We wrote custom code to figure out how to do the matrix math most efficiently and what to store in the cache.

Our beam search requires lookups in the n-gram language model, which can be expensive. So, instead of considering all possible character additions in the beam, we only consider character sequence additions that have a cumulative probability of $p = 0.99$ (but no more than 40 characters in an added sequence).

We use our unidirectional RNN (with a short look into the future) instead of the bidirectional RNNs.

These optimizations come with a 5% relative degradation in word-error rate. We did all these optimizations for the Mandarin systems.

10 Conclusion

Our end-to-end neural network for speech recognition trained on massive English and Mandarin datasets achieves almost human level performance. We also describe a bunch of optimizations to make the system fast at train and test time.