COMPARISON OF OPTIMIZATION ALGORITHMS OF CONNECTIONIST TEMPORAL CLASSIFIER FOR SPEECH RECOGNITION SYSTEM

Yedilkhan Amirgaliyev1,2, Kuanyshbay Darkhan1,2, Aisultan Shoynbek2
1Institute Information and Computational Technologies CS MES RK, Almaty, Kazakhstan, 2Suleyman Demirel University, Almaty, Kazakhstan

Abstract. This paper evaluates and compares the performances of three well-known optimization algorithms (Adagrad, Adam, Momentum) for faster training the neural network of CTC algorithm for speech recognition. For CTC algorithms recurrent neural network has been used, specifically Long-Short-Term memory. LSTM is effective and often used model. Data has been downloaded from VCTK corpus of Edinburgh University. The results of optimization algorithms have been evaluated by the Label error rate and CTC loss.

Keywords: recurrent neural network, search methods, acoustic, systems modeling language

PORÓWNANIE ALGORYTMÓW OPTYMALIZACJI KLASYFIKATORA CZASOWEGO DO SYSTEMU ROZPOZNAWANIA MOWY

Streszczenie. W artykulecto dokonano oceny i porównania wydajności trzech znanych algorytmów optymalizacyjnych (Adagrad, Adam, Momentum) w celu przyspieszenia treningu sieci neuronowej algorytmu CTC do rozpoznawania mowy. Dla algorytmów CTC wykorzystano rekurencyjną sieć neuronową, w szczególności LSTM, która jest efektywnym i często używanym modelem. Dane zostały pobrane z wydziału VCTK University w Edynburgu. Wyniki algorytmów optymalizacyjnych zostały ocenione na podstawie wskaźników Label error i CTC loss.

Słowa kluczowe: rekurencyjna sieć neuronowa, metody wyszukiwania, akustyka, język modelowania systemów

Introduction

There have been many techniques for recognizing speech and variety of tasks like voice pattern recognition, which helps to identify human by his voice [12]. Considering the fact that speech data is complicated in terms of segmentation that is [1], it is difficult to build a model with a simple structure. The state-of-the-art technique for ASR (Automatic speech recognition) is always been HMM model [7], which involves other pre-trained models like acoustic model, language model etc. However, recent researches have shown that by using recurrent neural networks [9], we can build such architecture of neural network, which will require only speech data (.wav) and transcription (.txt) to train the model completely, whereas traditional models (HMM) [7] would require data for training language model and acoustic model. This advanced algorithm called Connectionist-Temporal-Classifier [8], the heart of which is RNN. One of the most common and crucial steps in neural network is training. It is important that the model will train fast and at the same time does not overfit or underfit, especially with speech data. Labelling an unsegmented data is very common and often difficult problem in the sequence-to-sequence models. Straightforward way to solve this problem is to label each segment of a sentence (for example wave file) manually. However, considering that there are so many words in speech, not counting the sentences, which brings a certain transformations time-consuming, boring and hard to do. To avoid this kind of issues traditional ASR system uses Language model like in [4], which predicts the probability of last word given the sentence and Acoustic model using a progress like in [3], which gives the phoneme representation of the given speech (Fig. 1).

Labelling an unsegmented data is very common and often difficult problem in the sequence-to-sequence models. Straightforward way to solve this problem is to label each segment of a sequence (for example wave file) manually. However, considering that there are so many words in speech, not counting the sentences, which brings a certain transformations time-consuming, boring and hard to do. To avoid this kind of issues traditional ASR system uses Language model like in [4], which predicts the probability of last word given the sentence and Acoustic model using a progress like in [3], which gives the phoneme representation of the given speech (Fig. 1).

Fig. 1. Traditional ASR system

1. Encoder and decoder

The RNN encoder-decoder is a neural network model that directly computes the conditional probability of the output sequence given the input sequence without assuming a fixed alignment, i.e. \( P(y_1, \ldots, y_T | x_1, \ldots, x_T) = \prod_{t=1}^{T} P(y_t | x_1, \ldots, x_t, y_1, \ldots, y_{t-1}) \). From the previous output symbols and the context vector, the decoder computes:

\[
P(y_1, \ldots, y_T | x_1, \ldots, x_T) = \prod_{t=1}^{T} P(y_t | x_1, \ldots, x_t, y_1, \ldots, y_{t-1})
\]
Since the probability \( P(y_{1,...,y_T} | x_{1,...,x_T}) \) is conditioned on the previous outputs as well as the context vector, an RNN can be used to compute this probability which implicitly remembers the history using a recurrent layer.

Let \( y_o \) be a vector representation of the output symbol \( y_o \), where \( y_o \) is a one-hot vector indicating one of the words in the vocabulary followed by a neural projection layer for dimension reduction. The posterior probability of \( y_o \) is computed as:

\[
P(y_o | y_1,...,y_{o-1},c_o) = g(y_o, y_1,...,y_{o-1}, c_o),
\]

where so denotes the output of a recurrent hidden layer \( f(\cdot) \) with inputs \( y_{o-1}, c_{o-1} \), and \( c_o \). \( g(\cdot) \) is a softmax function with inputs \( y-o \) and so. We condition both \( f(\cdot) \) and \( g(\cdot) \) on the context vector to encourage the decoder to be heavily reliant on the context from the encoder. The previous output \( y_{o-1} \), is also fed to the softmax function \( g(\cdot) \) to capture the bigram dependency between consecutive words [3]. We have also investigated a simpler output function without the dependence on the previous output \( y_{o-1} \), i.e. \( P(y_o | y_1,...,y_{o-1}, c_o) = g(s_o, c_o) \).

**Encoder**

As discussed above, the computation of the conditional probability relies on the availability of the context vector \( c_o \) for each output \( y_o \). The context vector is obtained from the encoder which reads the input sequence and generates a continuous space representation. The context vector \( c_o \) is obtained by the weighted average of all the hidden representations of a bidirectional RNN (BiRNN) [8]:

\[
c_o = \sum_t \alpha_t h_t
\]

where \( \alpha_t \in [0, 1] \) and \( \sum_t \alpha_t = 1 \); \( h_t = (h_t^+, h_t^−) \) and \( h_t^+, h_t^− \) denote the hidden representations of \( x_t \) from the forward and backward RNNs respectively. The context vector \( c_o \) is global, for instance, \( c_o = hT \). This means the context vector does not depend on the index \( o \), meaning that the whole input sequence is encoded into a fixed vector representation. This approach has produced state-of-the-art results in machine translation when the dimension of the vector is relatively large [14]. When the model size is relatively small, however, the use of a dynamic context vector has been found to be superior, especially for long input sequences.

The weight \( \alpha_t \) is computed by a learned alignment model for each \( o \), which is implemented as a neural network such that:

\[
\begin{align*}
\alpha_t &= \text{exp}(\alpha_t) \\
\sum_t \alpha_t &= 1
\end{align*}
\]

where \( \alpha(\cdot) \) is a feedforward neural network that computes the relevance of each hidden representation \( h_t \) with respect to the previous hidden state of RNN decoder so. The alignment model is a single-hidden-layer neural network:

\[
\alpha(s_o, h_t) = W^T \tanh(W s_o +Uh_t)
\]

where \( W \) and \( U \) are weight matrices, and \( v \) is a vector so that the output of \( \text{a}(\cdot) \) is a scalar. More hidden layers can be used in the alignment model.

In the case of using a fixed context vector using an RNN to map the whole input sequence into the context vector is necessary because this vector must represent all the relevant information in the input sequence.

**2. Connectionist temporal classifier**

The CTC algorithm considers the order of the output labels of RNNs with ignoring the alignments by introducing a blank label, \( b \). For the set of target labels, \( L \), and its extended set with the additional CTC blank label, \( L' = L \cup \{b\} \), the path, \( \pi \), is defined as a sequence over \( L' \), that is, \( \pi \subseteq L' \), where \( T \) is the length of the input sequence, \( x \). Then, the output sequence, \( z \in L \subseteq T \), is represented by \( z = F(\pi) \) with the sequence to sequence mapping function \( F \). \( F \) maps any path \( \pi \) with the length \( T \) into the shorter sequence of the label, \( z \), by first merging the consecutive same labels into one and then removing the blank labels. Therefore, any sequence of the raw RNN outputs with the length \( T \) can be decoded into the shorter labelling sequence, \( z \), with ignoring timings. This enables the RNNs to learn the sequence mapping, \( z = G(x) \), where \( x \) is the input sequence and \( z \) is the corresponding target labelling for all \( x, z \) in the training set, \( S \). More specifically, the gradient of the loss function \( L(x, z) = -\ln p(z|x) \) is computed and fed to the RNN through the softmax layer, of which the size is \( |L'| \).

The CTC algorithm employs the forward-backward algorithm for computing the gradient of the loss function, \( L(x, z) \). Let \( z' \) be the sequence over \( L' \) with the length of \( |\pi|+|\lambda| \) where \( z'_u \) is for odd \( u \) and \( z'_u = z_{u+1} \) for even \( u \). Then, the forward variable, \( a \), and the backward variable, \( b \), are initialized by:

\[
\begin{align*}
a(1, u) &= \begin{cases} y^1_h & \text{if } u = 1 \\
y^2_h & \text{if } u = 2 \\
0 & \text{otherwise}
\end{cases} \\
b(T, u) &= \begin{cases} 1 & \text{if } u = |z'| \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

where \( y^k_h \) is the soft max output of the label \( k \in L' \) at time \( t \). The variables are forward and backward propagated as:

\[
\begin{align*}
a(t, u) &= \begin{cases} y^1_h & \text{if } z'_u = b \text{ or } z'_{u-2} = z'_u \\
y^2_h & \text{otherwise}
\end{cases} \\
b(t, u) &= \begin{cases} y^1_h & \text{if } z'_u = b \text{ or } z'_{u+2} = z'_u \\
y^2_h & \text{otherwise}
\end{cases}
\end{align*}
\]

With boundary conditions:

\[
\begin{align*}
a(t, 0) &= \ln v, b(t, |z'| + 1) = 0, \forall t
\end{align*}
\]

Then, the error gradient with respect to the input of the softmax layer at time \( t \), \( a_t^T a_t d_t \), is computed as:

\[
\frac{\partial L(x, z)}{\partial a_t} = \begin{cases} \frac{1}{|z|^2} \sum_{u \in \text{seq}(z,x)} a(t, u) \beta(t, u) & \text{if } z'_u = b \text{ or } z'_{u+2} = z'_u \\
0 & \text{otherwise}
\end{cases}
\]

where \( B(z, x) = \{u: z'_u = b \} \) and \( p(z|x) = \alpha(T, |z'| - 1) \).

**Beam search**

This algorithm iterates through the NN output and creates text candidates (called beams) which are scored. Figure 2 shows an illustration of the evolution of beams: we start with the empty beam, then add all possible characters (we only have “a” and “b” in this example) to it in the first iteration and only keep the best scoring ones. The beam width controls the number of surviving beams. This is repeated until the complete NN output it processed.

![Fig. 2. Beam search](image)

**3. Optimization algorithms**

Gradient descent [14] is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent. These algorithms, however, often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by.

Gradient descent is a way to minimize an objective function \( f(\theta) \) parameterized by a model’s parameters \( \theta \in \mathbb{R}^d \) by updating the parameters in the opposite direction of the gradient of the objective function \( \nabla f(\theta) \) w.r.t. to the parameters. The learning rate \( \eta \) determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.
Stochastic Gradient Descent

SGD in [10] updates model parameters \( \theta \) in the negative direction of the gradient \( g \) by taking a subset or a mini-batch of data of size \( m \):

\[
g = \frac{1}{m} \nabla \theta \sum_i L(f(x^{(i)}; \theta), y^{(i)}),
\]

\[
\theta = \theta - \eta \times g.
\]

Adagrad

This method simply allows the learning Rate -\( \eta \) to adapt based on the parameters. Therefore, it makes big updates for infrequent parameters and small updates for frequent parameters. For this reason, it is well suited for dealing with sparse data.

Adagrad uses a different learning rate in [5] for every parameter \( \theta(i) \) at every time step \( t \), we first show Adagrad’s per-parameter update, which we then vectorise. Briefly, we set \( g(t, i) \) to be the gradient of the loss function w.r.t. to the parameter \( \theta(i) \) at time step \( t \).

\[
\theta_{t+1, i} = \theta_{t, i} - \eta \frac{1}{\sqrt{M_{i,t}} + \epsilon} \cdot g_{t,i}
\]

Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.

Momentum in [2] is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction \( \gamma \) of the update vector of the past time step to the current update vector.

\[
\dot{\theta}_t = \gamma \dot{\theta}_{t-1} + \eta \nabla J(\theta)
\]

\[
\theta = \theta - \dot{\theta}_t
\]

The momentum term \( \gamma \) is usually set to 0.9 or a similar value. Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity, if there is air resistance, i.e. \( \gamma < 1 \)). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Adam

Adam stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter [15]. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients \( M(t) \), similar to momentum:

\[
m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t
\]

\[
v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2
\]

\( m_t \) and \( v_t \) are estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As \( m_t \) and \( v_t \) are initialized as vectors of 0’s, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. \( \beta_1 \) and \( \beta_2 \) are close to 1).

\[
\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}
\]

They then use these to update the parameters:

\[
\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t
\]

The authors propose default values of 0.9 for \( \beta_1 \), 0.999 for \( \beta_2 \), and \( 10^{-8} \) for \( \epsilon \). They show empirically that Adam works well in practice and compares favourably to other adaptive learning-method algorithms.

4. Results

After training the model three times with different optimization algorithms, we see the following outcomes (Table 1):

Table 1. Result of each optimizer

<table>
<thead>
<tr>
<th>Name</th>
<th>Training cost</th>
<th>Validation cost</th>
<th>Training LER</th>
<th>Validation LER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adagrad</td>
<td>166.774</td>
<td>242.164</td>
<td>0.970</td>
<td>0.980</td>
</tr>
<tr>
<td>Momentum</td>
<td>2.405</td>
<td>71.289</td>
<td>0.000</td>
<td>0.500</td>
</tr>
<tr>
<td>Adam</td>
<td>166.774</td>
<td>242.164</td>
<td>0.970</td>
<td>0.980</td>
</tr>
</tbody>
</table>

Fig. 3. CTC loss and LER of Adagrad

As we can see in Figure 3 CTC loss at the beginning of each iteration starts to decrease as it should, but after that, the loss value starts to hesitate drastically between 500 and 50 (approximately). At the same time, LER right from the beginning start to hesitate between 1 and 0.9 (approximately), which does not allow the model to learn.

Fig. 4. CTC loss and LER of Momentum
The visualization we see in Figure 4 shows that Momentum works a lot better than Adagrad. LER and CTC loss are continuously decreasing for training and validation sets. Because of learning rate is equal to 0.005 decreasing process slows down little bit. Other than that, learning process is doing well.

As shown in Figure 5 CTC loss decreases at the beginning of the gradient steps and once again as in Adagrad optimizer starts to hesitate between two numbers with a big difference. LER on the other hand decreases to 1 after few iterations and after that does not change for a long gradient steps. Right after it reaches about 750 iteration LER starts to hesitate between 1 and 0.5 (approximately), which is not a well performance.

5. Conclusion

This paper shows a clear benefit of Momentum optimizer over Adam and Adagrad for CTC algorithm for speech recognition. The experiment showed that model with Momentum optimizer learns faster decreasing the CTC loss and LER after each gradient step, whereas Adagrad and Adam optimizers performed very poorly, showing a hesitation of errors from big number to small. Other than that, this paper shows the advanced algorithm called Connectionist Temporal Class for speech recognition in action. It also describes the clear benefits of this algorithm over the traditional method, which is HMM based model, which is the simplicity and effectiveness.

References


Ph.D. Yedilkhan Amirgaliev
email: amir_ed@mail.ru
Doctor of technical sciences, professor, head of the laboratory of mathematical and computer modelling of the Institute of Information and computing technologies of the Science Committee of RK MES.

M.Sc. Kuanyshbay Darkhan
email: darkhan.kuanyshbay@sdu.edu.kz

ORCID ID: 0000-0002-6528-0619

M.Sc. Aisultan Shoiynbek
email: aisultan.shoiynbek@sdu.edu.kz

ORCID ID: 0000-0001-5952-8609