A Review on Recent Advances in Recurrent Neural Networks

Prithviraj Rathore And Ritika
Department of Information Technology
JECRC Foundation
Jaipur

Ms. Shikha Shrivastava
Assistant professor
Department of Information Technology
JECRC Foundation
Jaipur

Abstract - Recurrent neural networks (RNNs) are capable of learning features and long term dependencies from sequential and time-series data. The RNNs have a stack of non-linear units where at least one connection between units forms a directed cycle. A well-trained RNN can model any dynamical system; however, training RNNs is mostly plagued by issues in learning long-term dependencies. In this paper, we present a survey on RNNs and several new advances for newcomers and professionals in the field. The fundamentals and recent advances are explained and the research challenges are introduced.

Keywords - Deep learning, long-term dependency, recurrent neural networks, time-series analysis.

I. INTRODUCTION

Speech is a complex time-varying signal with complex correlations at a range of different timescales. Recurrent neural networks (RNNs) contain cyclic connections that make them a more powerful tool to model such sequence data than feed-forward neural networks. RNNs have demonstrated great success in sequence labeling and prediction tasks such as handwriting recognition and language modeling. In acoustic modeling for speech recognition, however, where deep neural networks (DNNs) are the established state-of-the-art, recently RNNs have received little attention beyond small scale phone recognition tasks, notable exceptions being the work of Robinson [1], Graves [2], and Sak [3].

LSTM and conventional RNNs have been successfully applied to various sequence prediction and sequence labeling tasks. In language modeling, a conventional RNN has obtained significant reduction in perplexity over standard n-gram models [6] and an LSTM RNN model has shown improvements over conventional RNN LMs [7]. LSTM models have been shown to perform better than RNNs on learning context-free and context-sensitive languages [8]. Bidirectional LSTM (BLSTM) networks that operate on the input sequence in both directions to make a decision for the current input have been proposed for phonetic labeling of acoustic frames on the TIMIT speech database [9]. For online and offline handwriting recognition, BLSTM networks used together with a Connectionist Temporal Classification (CTC) layer and trained from unsegmented sequence data, have been shown to outperform a state-of-the-art Hidden-Markov-Model (HMM) based system [10]. Similar techniques with a deep BLSTM network have been proposed to perform grapheme-based speech recognition [11]. BLSTM networks have also been proposed for phoneme prediction in a multi-stream framework for continuous conversational speech recognition [12]. In terms of architectures, following the success of DNNs for acoustic modeling [13, 14, 15, 16], a deep BLSTM RNN combined with a CTC output layer and an RNN transducer predicting phone sequences has been shown to reach state-of-the-art phone recognition accuracy on the TIMIT database [17].

II. LSTM Network Architectures

The LSTM contains special units called memory blocks in the recurrent hidden layer. The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block in the original architecture contained an input gate and an output gate. The input gate controls the flow of input activations into the memory cell. The output gate controls the output flow of cell activations into the rest of the network. Later, the forget gate
Deep input to hidden: One of the basic ideas is to bring the multi-layer perceptron (MLP) structure into the transition and output stages, called deep transition RNNs and deep output RNNs, respectively. To do so, two operators can be introduced. The first is a plug operator, which receives two vectors, the input vector $x$ and hidden state $h$, and returns a summary as

$$h = x \oplus h.$$ 

This operator is equivalent to the Eq. (1). The other operator is a predictor denoted as $a$, which is equivalent to the Eq. (3) and predicts the output of a given summary $h$ as

$$y = ah.$$ 

Higher level representation of input data means easier representation of relationships between temporal structures of data. This technique has achieved better results than feeding the network with original data in speech recognition [43] and word embedding [45] applications. Structure of a RNN with an MLP in the input to hidden layers is presented in Figure 5a. In order to enhance long-term dependencies, an additional connection makes a short-cut between the input and hidden layer as in Figure 5b [44].

Deep hidden to hidden and output: The most focus for deep RNNs is in the hidden layers. In this level, the procedure of data abstraction and/or hidden state construction from previous data abstractions and new inputs is highly non-linear. An MLP can model this non-linear function, which helps a RNN to quickly adapt to fast changing input modes while still having a good memory of past events. A RNN can have both an MLP in transition and an MLP before the

$$h = x \oplus h.$$ 

where the $W$ terms denote weight matrices (e.g. $W_{ix}$ is the matrix of weights from the input gate to the input), $W_{ic}$, $W_{if}$, $W_{io}$ are diagonal weight matrices for peephole connections, the $b$ terms denote bias vectors ($b_f$ is the input gate bias vector), $\sigma$ is the logistic sigmoid function, and $i$, $f$, $o$ and $c$ are respectively the input gate, forget gate, output gate and cell activation vectors, all of which are the same size as the cell output activation vector $m$, is the element-wise product of the vectors, $g$ and $h$ are the cell input and cell output activation functions, generally and in this paper $tanh$, and $\varphi$ is the network output activation function, softmax in this paper.

III Deep RNNs with Multi-Layer Perceptron

Deep architectures of neural networks can represent a function exponentially more than shallow architectures. While recurrent networks are inherently deep in time given each hidden state is a function of all previous hidden states [43], it has been shown that the internal computation is in fact quite shallow [44]. In [44], it is argued that adding one or more nonlinear layers in the transition stages of a RNN can improve overall performance by better disentangling the underlying variations the original input. The deep structures in RNNs with perceptron layers can fall under three categories: input to hidden, hidden to hidden, and hidden to output [44].

IV Bidirectional RNN

Conventional RNNs only simply looking at previous context is sufficient in many applications such as speech recognition, it is also useful to explore the future context as well [43]. Previously, the use of future information as context for current prediction have been attempted in the basic architecture of RNNs by delaying the output by a certain number of time frames. However, this method required a handpicked optimal delay to be chosen for any implementation. A bi-directional
RNN (BRNN) considers all available input sequence in both the past and future for estimation of the output vector [46]. To do so, one RNN processes the sequence from start to end in a forward time direction. Another RNN processes the sequence backwards from end to start in a negative time direction as demonstrated in Figure 6. Outputs from forward states are not connected to inputs of backward states and vice versa and there are no interactions between the two types of state neurons [46]. The forward and backward hidden sequences are denoted by \( h_f \) and \( h_b \), respectively, at time \( t \). The forward hidden sequence is computed as

\[
y_{t+1} = f_{	ext{ff}}(x_t, y_t, h_{f, t})
\]

BPTT is one option to train BRNNs. However, the forward and backward pass procedures are slightly more complicated because the update of state and output neurons can no longer be conducted one at a time [46]. While simple RNNs are constrained by inputs leading to the present time, the BRNNs extend this model by using both past and future information. However, the shortcoming of BRNNs is their requirement to know the start and end of input sequences in advance. An example is labeling spoken sentences by their phonemes.

V Recurrent Convolutional Neural Networks

The rise in popularity of RNNs can be attributed to its ability to model sequential data. Previous models examined have augmented the underlying structure of a simple RNN to improve its performance on learning the contextual dependencies of single dimension sequences. However, there exists several problems, which require understanding of contextual dependencies over multiple dimensions. The most popular network architectures use convolutional neural networks (CNNs) to tackle these problems.

CNNs are very popular models for machine vision applications. CNNs may consist of multiple convolutional layers, optionally with pooling layers in between, followed by fully connected perceptron layers [11]. Typical CNNs learn through the use of convolutional layers to extract features using shared weights in each layer. The feature pooling layer (i.e., sub-sampling) generalizes the network by reducing the resolution of the dimensionality of intermediate representations (i.e., feature maps) as well as the sensitivity of the output to shifts and distortions. The extracted features, at the very last convolutional layer, are fed to fully connected perceptron model for dimensionality reduction of features and classification.

Incorporation of recurrent connections into each convolutional layer can shape a recurrent convolutional neural network (RCNN) [47]. The activation of units in RCNN evolve over time, as they are dependent on the neighboring unit. This approach can integrate the context information, important for object recognition tasks. This approach increases the depth of model, while the number of parameters is constant by weight sharing between layers. Using recurrent connections from the output into the input of the hidden layer allows the network to model label dependencies and smooth its own outputs based on its previous outputs [48]. This RCNN approach allows a large input context to be fed to the network while limiting the capacity of the model. This system can model complex spatial dependencies with low inference cost. As the context size increases with the built-in recurrence, the system identifies and corrects its own errors [48]. Quadrilateral 2-dimensional RNNs can enhance CNNs to model long range spatial dependencies [49]. This method efficiently embeds the global spatial context into the compact local representation [49].

VI Long-Short Term Memory

Recurrent connections can improve performance of neural networks by leveraging their ability to understand sequential dependencies. However, the memory produced from the recurrent connections can severely be limited by the algorithms employed for training RNNs. All the models thus far have fallen victim to exploding or vanishing gradients during the training phase, resulting in the network failing to learn long-term sequential dependencies in data. The following models are specifically designed to tackle this problem, the most popular being the long-short term memory (LSTM) RNNs. LSTM is one of the most popular and efficient methods for reducing the effects of vanishing and exploding gradients [54]. This approach changes the structure of hidden units from “sigmoid” or “tanh” to memory cells, in which their inputs and outputs are controlled by gates. These gates control flow of information to hidden neurons and preserve extracted features from previous timesteps.

It is shown that for a continual sequence, the LSTM model’s internal values may grow without bound [55]. Even when continuous sequences have naturally reoccurring properties, the network has no way to detect which information is no longer relevant. The forget gate learns weights that control the rate at which the value stored in the memory cell decays [55].
For periods when the input and output gates are off and the forget gate is not causing decay, a memory cell simply holds its value over time so that the gradient of the error stays constant during back-propagation over those periods [21]. This structure allows the network to potentially remember information for longer periods.

LSTM suffers from high complexity in the hidden layer. For identical size of hidden layers, a typical LSTM has about four times more parameters than a simple RNN [6]. The objective at the time of proposing the LSTM method was to introduce a scheme that could improve learning long-range dependencies, rather than to find the minimal or optimal scheme [21]. Multi-dimensional and grid LSTM networks have partially enhanced learning of long-term dependencies comparing to simple LSTM.

![Distributed Training: Scaling up to Large Models with Parallelization](image)

We chose to implement the LSTM RNN architectures on multicore CPU rather than on GPU. The decision was based on CPU’s relatively simpler implementation complexity, ease of debugging and the ability to use clusters made from commodity hardware. For matrix operations, we used the Eigen matrix library [21]. This templated C++ library provides efficient implementations for matrix operations on CPU using vectorized instructions. We implemented activation functions and gradient calculations on matrices using SIMD instructions to benefit from parallelization.

We use the truncated backpropagation through time (BPTT) learning algorithm [22] to compute parameter gradients on short subsequences of the training utterances. Activations are forward propagated for a fixed step time \( T_{bptt} \) (e.g. 20). Cross entropy gradients are computed for this subsequence and backpropagated to its start. For computational efficiency each thread operates on subsequences of four utterances at a time, so matrix multiplies can operate in parallel on four frames at a time. We use asynchronous stochastic gradient descent (ASGD) [23] to optimize the network parameters, updating the parameters asynchronously from multiple threads on a multicore machine. This effectively increases the batch size and reduces the correlation of the frames in a given batch. After a thread has updated the parameters, it continues with the next subsequence in each utterance, preserving the LSTM state, or starts new utterances with reset state when one finishes. Note that the last subsequence of each utterance can be shorter than \( T_{bptt} \) but is padded to the full length, though no gradient is generated for these padding frames.

This highly parallel single machine ASGD framework described in [3] proved slow for training models of the scale we have used for large scale ASR with DNNs (many millions of parameters). To scale further, we replicate the single-machine workers on many (e.g. 500) separate machines, each with three, synchronized, computation threads. Each worker communicates with a shared, distributed parameter server [23], which stores the LSTM parameters. When a worker has computed the parameter gradient on a minibatch (of 3 4 \( T_{bptt} \) frames), the gradient vector is partitioned and sent to the parameter server shards which each add the gradients to their parameters and respond with the new parameters. The parameter server shards aggregate parameter updates completely asynchronously. For instance, gradient updates from workers may arrive in different orders at different shards of the parameter server. Despite the asynchrony, we observe stable convergence, though the learning rate must be reduced, as would be expected because of the increase in the effective batch size from the greater parallelism.

**Experiment**

We evaluate and compare the performance of LSTM RNN architectures on a large vocabulary speech recognition task – the Google Voice Search task. We use a hybrid approach [24] for acoustic modeling with LSTM RNNs, wherein the neural networks estimate hidden Markov model (HMM) state posteriors. We scale the state posteriors by the state priors estimated as the relative state frequency from the training data to obtain the acoustic frame likelihoods. We deweight the silence state counts by a factor of 2.7 when estimating the state frequencies.

**Result**

We summarize the results for various LSTM and LSTMP RNN architectures. We observe that the conventional LSTM RNNs with a single layer do not perform very well for this large scale acoustic modeling task. With two layers of LSTM RNNs, the performance improves but still it is not very good. The LSTM RNN with five layers approaches the performance of the best model. We see that training an LSTM RNN with seven layers is hard, the model starts converging after a day of training. From the table, one can see that the LSTMP RNN models with a single layer and a large number of memory cells tends to overfit the training data. Increasing the number of LSTMP RNN layers seems to alleviate this problem.
of memorization and to result in better generalization to held-out data. The LSTMP RNN models give slightly better results than the LSTM RNN model with 5 layers. We see that increasing the number of parameters in the LSTMP RNN models more than 13M by having more layers or more memory cells does not give property.

Figure 3 compares the frame accuracies on training and held-out sets for various LSTM and LSTMP architectures. The overfitting problem with LSTMP RNN architecture with large number of memory cells (2048) can be seen clearly. We observe that LSTMP RNN architectures converges faster than LSTM RNN architectures. It is clear that having more layers helps generalization but makes training harder and convergence slower.

Table 2 shows how the performance of networks with deep LSTMP RNN architecture changes with the depth and number of model parameters. We see that increasing the number of parameters over 13M does not improve performance. We can also decrease the number of parameters substantially without hurting performance much. The deep LSTMP RNN architecture with two layers each with 800 cells and 512 recurrent projection units converges mostly in 48 hours and gives 10.9% WER on the independent test set. Training this model for 100 hours improves the WER to 10.7% and for 200 hours to 10.5%. In comparison, our best DNN models with 85M parameters gives 11.3% at the same beam and training takes a few weeks.

Conclusions –

We showed that deep LSTM RNN architectures achieve state- of-the-art performance for large scale acoustic modeling. The proposed deep LSTMP RNN architecture outperforms standard LSTM networks and DNNs and makes more effective use of the model parameters by addressing the computational efficiency needed for training large networks. We also show for the first time that LSTM RNN models can be quickly trained using ASGD distributed training.

References -


[72] J. Weston, A. Bordes, S. Chopra, and T.