GREEDY LAYER-WISE TRAINING OF LONG SHORT TERM MEMORY NETWORKS

Kaisheng Xu[†], *Xu Shen*[†], *Ting Yao*[‡], *Xinmei Tian*[†], *Tao Mei*[‡]

[†] CAS Key Laboratory of Technology in Geo-Spatial Information Processing and Application System,
University of Science and Technology of China, Hefei, Anhui, China
[‡] Microsoft Research, Beijing, China

{ksxu, shenxu}@mail.ustc.edu.cn, tingyao.ustc@gmail.com, xinmei@ustc.edu.cn, tmei@microsoft.com

ABSTRACT

Recent developments in Recurrent Neural Networks (RNNs) such as Long Short Term Memory (LSTM) have shown promising potential for modeling sequential data. Nevertheless, training LSTM is not trivial when there are multiple layers in the deep architectures. This difficulty originates from the initialization method of LSTM, where gradient-based optimization often appears to converge to poor local solutions. In this paper, we explore an unsupervised pretraining mechanism for LSTM initialization, following the philosophy that the unsupervised pretraining plays the role of a regularizer to guide the subsequent supervised training. We propose a novel encoder-decoder-based learning framework to initialize a multi-layer LSTM in a greedy layer-wise manner in which each added LSTM layer is trained to retain the main information in the previous representation. A multi-layer LSTM trained with our method outperforms the one trained with random initialization, with clear advantages on several tasks. Moreover, the multi-layer LSTMs converge 4 times faster with our greedy layer-wise training method.

Index Terms— Layer-wise training, Long Short Term Memory

1. INTRODUCTION

Understanding sequential data is a fundamental problem in the field of Artificial Intelligence. Recurrent Neural Networks (RNNs) such as Long Short Term Memory (LSTM) have been successfully used to capture sequential information by mapping sequences to other sequences in many learning tasks, including speech recognition, machine translation, image and video captioning and action recognition [1].

While promising results have been achieved using LSTM for various tasks, initializing LSTM parameters is very challenging, as the training criterion is non-convex and involves many local minima. A general empirical work has demonstrated the effect of parameter initialization in deep architectures [2]. With hundreds of different random initializations, gradient descent converges to a different local minimum each time. More importantly, purely supervised training starting from random initialization may become worse in those architectures with multiple layers. Therefore, the main challenge is how to effectively initialize the LSTM parameters.

Inspired by greedy unsupervised pretraining in Deep Belief Networks (DBNs) [3] and stacked autoencoders [4], this paper describes an investigation of greedy layer-wise pretraining to initialize an LSTM. The greedy layer-wise pretraining strategy starts by pretraining one layer at a time in a greedy manner, using an unsupervised learning process for each layer to preserve input information. Then, the whole network is fine tuned with respect to an ultimate criterion with a gradient-based optimization. Networks pretrained in this way are expected to reach better local minima and support better generalization; thus, the parameters learned in this phase can function as a better method of network initialization in subsequent supervised learning tasks.

By incorporating the greedy layer-wise pretraining method into LSTM training, we present a novel encoder-decoder-based learning architecture for multi-layer LSTM initialization. Specifically, an encoding LSTM is used to learn the representation of the input sequence. Subsequently, the decoding LSTM can reconstruct the input sequence from that representation. Then, we recursively use hidden states from the previous encoding LSTM as the input of the next encoding-decoding LSTM layer to initialize a desired number of LSTM layers. Finally, we connect the hidden output of the last LSTM layer to a supervised layer and fine tune all the parameters using supervised costs. A multi-layer LSTM initialized with our approach noticeably outperforms one trained using random initialization on several tasks. Moreover, a multi-layer LSTM trained using our approach converges much faster.

The main contributions of this paper are as follows:

- We study the initialization problem of LSTM networks. To the best of our knowledge, this paper represents one of the first attempts from the artificial intelligence community to target this type of network using greedy layer-wise training.
- A novel encoder-decoder-based learning framework is proposed to initialize multi-layer LSTMs in a greedy

layer-wise manner. The resulting LSTMs are better than those trained using random initialization in terms of convergence speed and recognition performance.

- The proposed strategy is evaluated on four diverse tasks, *i.e.*, regression, recognition of handwritten digits, video classification and machine translation, and LSTMs trained with our method consistently outperform LSTMs trained using random initialization.

2. RELATED WORK

Greedy layer-wise unsupervised learning was first introduced for training DBNs [3]. It consists of two steps: unsupervised layer-wise pretraining and supervised fine tuning. After layer-wise pretraining, the network is represented by a stack of Restricted Boltzmann Machines (RBMs), in which the learned feature activations of the previous RBM are used as the "data" to train the next RBM in the stack. Similarly, autoencoders can also be used as the building blocks for layer-wise pretraining of multi-layer neural networks [4]. Theoretical and empirical analyses of the greedy layer-wise training method for deep networks were presented in [4, 2, 5]. This empirical analysis confirmed that greedy layer-wise unsupervised training primarily helps by initializing weights in a region near a good local minimum that supports better generalization [4, 5]. Erhan *et al.* clarified that layer-wise pretraining acts similarly to a regularizer [2].

Recently, unsupervised or semi-supervised training of RNNs has mostly been accomplished by training encoder-decoder-based architectures [6, 7, 8, 1]. A combination of an autoencoder model and a future predictor model was used for unsupervised LSTM training in [1]. In [6], the RNN was connected to two linear decoders that reconstructed the input and predicted its supervised classification. Similarly, an LSTM encoder was trained to provide representations used by two decoders: one LSTM decoder that reconstructs the input sequence in reverse order and one classifier for classification purposes [7]. In [8], the recurrent sequence autoencoder relearns the sequence prediction and is then used as a "pre-training" algorithm for a later supervised learning method.

Inspired by the success of greedy layer-wise training in fully connected networks and the LSTM autoencoder method for unsupervised learning, in this paper, we propose to improve the performance of multi-layer LSTMs by greedy layer-wise pretraining. This is one of the first attempts to use greedy layer-wise training for LSTM initialization.

3. OUR APPROACH

Deep architectures trained with random initialization can easily converge to poor local minima, leading to much slower convergence and reduced performance. Unsupervised layer-wise pretraining using the stacked autoencoder method has shown success in solving such problems with deep, fully connected neural networks. Meanwhile, an LSTM autoencoder demonstrates a good power to learn sequence representations. To achieve better initialization for deep LSTM models, we aim to leverage the LSTM autoencoder to perform layer-wise pretraining of each LSTM in the deep model. In the following, we will first present the related components and the training algorithm for fully connected networks, and then, describe the components and proposed pretraining algorithm for deep LSTM models.

3.1. Autoencoder

An autoencoder is trained to encode the input $\mathbf{x}$ into some representation $\mathbf{c}(\mathbf{x})$ such that the input can be reconstructed from that representation $\mathbf{f}(\mathbf{c}(\mathbf{x}))$, where $\mathbf{c}(\cdot)$ denotes the encoder and $\mathbf{f}(\cdot)$ denotes the decoder. Generally, the loss function of such an autoencoder can be defined as the cross entropy error $-\sum_i p_i \log \hat{p}_i - \sum_i (1-p_i)\log(1-\hat{p}_i)$, or the Euclidean distance, $\sum_i (x_i - \hat{x}_i)^2$.

3.2. Training Stacked Autoencoders

Stacked autoencoders can be used to initialize a deep multi-layer network [4].The training procedure is briefly described as follows:

1. Train the first layer as an autoencoder to minimize the reconstruction error of the raw input.
2. The outputs of that autoencoder are used as the input for the next layer. This next layer is also trained to be an autoencoder.
3. Iterate over step 2 to initialize the desired number of additional layers.
4. Feed the output of the last hidden layer into a new supervised layer.
5. Fine tune all the parameters of this deep architecture by a supervised or unsupervised cost.

3.3. The Long Short Term Memory (LSTM) Algorithm

RNNs have achieved great success in sequence learning tasks. However, the main known problem with RNNs is the difficulty of modeling long-term dependencies due to the vanishing or exploding problems. One of the most effective methods to address this problem is to use LSTM networks [9]. LSTM networks introduce a new architecture termed the *memory cell* to store long term dependencies. Memory cells have three main elements: an input gate, a forget gate and an output gate. The input gate is designed to control adding the input information to memory, while the forget gate and output gate determine whether information will be kept or released from the memory at each decision point. As presented in [10], LSTM variants do not exhibit large differences in performance. Therefore, we adopt the commonly used LSTMs as

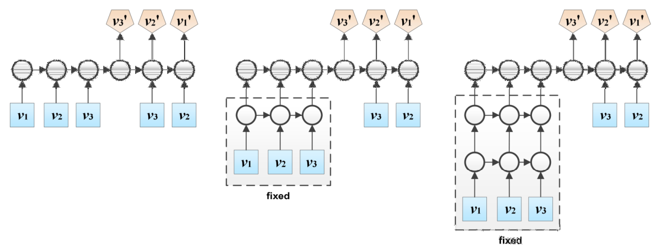

Fig. 1. Greedy layer-wise pretraining framework for sequence classification. The encoder LSTM reads in the hidden states of the previous layer, and the decoder LSTM reconstructs the input sequence in reverse order. During training, we fix the weights of all the previous layers; the parameters for only the current layer are learned. The input sequence is $\{v_1, v_2, v_3\}$. The circles represent LSTM cells.

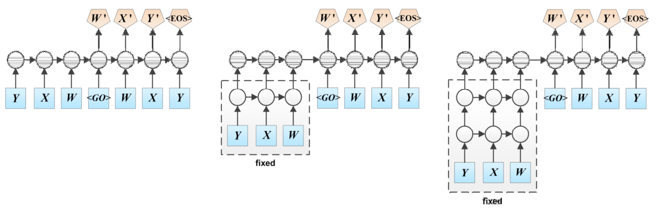

Fig. 2. Greedy layer-wise pretraining framework for sequence-to-sequence learning. In sequence learning, the encoder LSTM reads the input sequence in reverse order, but the decoder LSTM is trained to predict future elements in the original sequence. When training a new LSTM decoder, the previous layers are fixed. Here, $< GO >$ is the start marker for a sequence and $< EOS >$ is the end marker of the sequence. The input sequence is $\{W, X, Y\}$. The circles represent LSTM cells.

described in [11]. The gates, cell values and hidden outputs are computed as follows:

$$i^t = sigmoid(W^{xi}x^t + W^{hi}h^{t-1} + b^i) \tag{1}$$

$$o^t = sigmoid(W^{xo}x^t + W^{ho}h^{t-1} + b^o) \tag{2}$$

$$f^t = sigmoid(W^{xf}x^t + W^{hf}h^{t-1} + b^f) \tag{3}$$

$$g^t = \tanh(W^{xg}x^t + W^{hg}h^{t-1} + b^g) \tag{4}$$

$$c^t = f^t \odot c^{t-1} + i^t \odot g^t \tag{5}$$

$$h^t = o^t \odot \tanh(c^t), \tag{6}$$

where the Ws are weights and the bs are corresponding bias vectors, x^t, h^t and c^t represent the input, output and cell states at timestep t, respectively. h^{t-1} and c^{t-1} are the output and cell states of timestep $t-1$, while i^t, o^t and f^t are the input, output and forget gates, respectively. The $\odot$ represents a dot product operation, and $sigmoid(x) = 1/(1+e^{-x})$ and $tanh(x) = (e^x - e^{-x})/(e^x + e^{-x})$ are element-wise nonlinear activation functions.

3.4. LSTM Autoencoder

Generally, an LSTM autoencoder consists of two LSTMs, one for encoding and one for decoding. The input to the model is a sequence of vectors (features or video frames). Encoder LSTM will read in all the input sequences and encode them into a fixed length hidden output and cell states. Then, the cell states and hidden outputs of the encoder LSTM are copied over to the decoding LSTM [1], which outputs a decoding sequence that is a prediction for the input sequence. The decoding sequence should be the same as the input sequence in raw or reverse order.

Reversing the target sequence should be easier because the model needs to capture correlation in only a small range. Consequently, we adopt this architecture to perform unsupervised pretraining for sequence classification tasks. In contrast, reconstructing the input sequence in the original order requires the model to retain the general structure and long range correlations of the input sequence; we introduce this model to learn the initialization for sequence-to-sequence learning tasks.

3.5. Greedy Layer-wise LSTM Training

Training deep neural networks from random initialization with standard gradient descent is difficult, because variance in the activations and gradients across layers can easily cause vanishing or exploding problems when the singular values of the Jacobian associated with each layer are far from 1 [12]. If we denote the output of layer $i+1$ as $\mathbf{z}^{i+1}$ and that of layer i as $\mathbf{z}^i$, the Jacobian matrix associated with layer i is defined as:

$$\mathbf{J}^i = \frac{\partial \mathbf{z}^{i+1}}{\partial \mathbf{z}^i} = \{W^i; \mathbf{f}\}, \tag{7}$$

where the W^i values represent the weights of layer i and $\mathbf{f}$ is the activation function. In practice, if W^is are not properly initialized according to different $\mathbf{f}$ ($\mathbf{J}^i$ near 1), the gradients may exhibit different magnitudes at different layers, which leads to poor conditioning and slower training. There are two directions of gradients flows in the deep LSTM model: inner LSTM and inter LSTMs. That is, in the LSTM layer l, we have

$$\mathbf{J}_l = \frac{\partial \mathbf{h}_l^{t+1}}{\partial \mathbf{h}_l^t} = \{W_l; \mathbf{f}\}, \tag{8}$$

while between LSTM layer l and LSTM layer $l+1$, we have

$$\mathbf{J}_l^{l+1} = \frac{\partial \mathbf{h}_{l+1}^t}{\partial \mathbf{h}_l^t} = \{W_l; W_{l+1}; \mathbf{f}\}. \tag{9}$$

Here W_l and W_{l+1} are the weights in layers l and $l+1$, respectively, and $\mathbf{h}_{l+1}^t$ and $\mathbf{h}_l^t$ are the hidden states of layesr $l+1$ and l in timestep t, respectively. As shown in Equation 6, the relationships between $\mathbf{h}_l^{t+1}$ and $\mathbf{h}_l^t$ and between $\mathbf{h}_{l+1}^t$ and $\mathbf{h}_l^t$ are non-trivial, and they cannot be expressed as an explicit function. Consequently, we cannot derive the proper random initialization of weights to avoid the vanishing or exploding problems in gradient flows such as in deep feed forward neural networks [12].

In order to obtain proper weight initialization in deep LSTM models, we propose to first leverage the LSTM autoencoder to learn weights that ensure a constant gradient and activation flow in a single LSTM layer. Then, the hidden states of the previous layer are recursively used as input for the next LSTM autoencoder. The weights will be tuned to ensure inter LSTM layer constant gradient flows through an overall tuning step. Finally, gradient descent learning of supervised tasks from this initialization is expected to escape from vanishing or exploding problems; consequently, the model should learn better and faster than random initialized models.

In addition, this layer-wise training process influences each LSTM layer to memorize the previous representations of the sequence and to reconstruct the original input from that representation. Because the extracted information grows increasingly abstract from the lower layers to the higher layers, the model must retain the most useful and compact information and drop irrelevant noise from the input. This helps the model to escape from poor local optima in the weight space. Therefore, we believe that this procedure is better approach than random initialization for training deep LSTM networks.

The procedure to train a multi-layer LSTM is similar to that of stacked autoencoders:

1. Train the first LSTM layer as an LSTM autoencoder. The input sequence is also used as the input for the decoding LSTM.
2. The hidden states of the encoding LSTM are used as the input for the next LSTM autoencoder. To assist the model in learning the input sequence, we require the decoding LSTM to recover the original input sequence.
3. Iterate step 2 to initialize the desired number of additional LSTM layers.
4. Use the hidden outputs of the last LSTM layer as the inputs for a supervised layer.
5. Fine tune all the parameters of this deep architecture by a supervised cost.

Following the pipeline described above, our framework for layer-wise pretraining in sequence classification and the sequence-to-sequence learning task are shown in Fig. 1 and Fig. 2, respectively. For the sequence classification task, the input signals or features $\{v_1, v_2, v_3\}$ are read into the encoder LSTM in raw order, and the decoder LSTM is required to reconstruct the input in reverse order $\{v_3', v_2', v_1'\}$ because it is easier to reconstruct signals within a small range (Fig. 1). For the sequence-to-sequence learning task, the goal is to predict the next element in the target sequence; therefore, the decoding output is required to be in the same order as the ground truth sequence $\{W, X, Y\}$; however, to reduce the length of dependencies between the input and the predicted output $\{W', X', Y'\}$, we reverse the input sequence as discussed in [13] (Fig. 2).

Table 1. Performance of LSTMs on the Adding dataset.

Methods	Error
Random initialized (1 layer)	6.2%
Layer-wise initialized (1 layer)	5.3%
Random initialized (2 layers)	5.6%
Layer-wise initialized (2 layers)	4.8%
Random initialized (3 layers)	5.7%
Layer-wise initialized (3 layers)	4.8%

Table 2. Performance of LSTMs on the MNIST dataset.

Methods	Error
Random initialized (1 layer)	0.89%
Layer-wise initialized (1 layer)	0.76%
Random initialized (2 layers)	0.75%
Layer-wise initialized (2 layers)	0.62%
Random initialized (3 layers)	0.80%
Layer-wise initialized (3 layers)	0.60%

4. EXPERIMENTS

To evaluate the effectiveness of our proposed layer-wise pretraining of LSTM, we conduct extensive experiments on the Adding, MNIST, UCF-101 and WMT'14 datasets. These four datasets are representative of *regression*, *image recognition*, *video recognition* and *sequence-to-sequence learning* tasks. Since our layer-wise pretraining is the first attempt for multilayer LSTM initialization without using any extra unlabeled data, our only baseline is random initialized LSTM. Experimental results show that layer-wise pretraining can generally improve the performance of LSTM networks in sequence learning tasks. The details are presented in the following sections.

In the pretraining phase, we use the same learning rate, random initialization and batch size as those used in the fine tuning phase. Reconstruction loss is defined as the Euclidean distance between the input and predicted sequences on the Adding, MNIST and UCF-101 datasets, while in WMT'14 dataset, our model is pretrained by maximizing the log probability of the correct input sentence; therefore, the objective is $1/|\mathcal{S}| \sum_{S \in \mathcal{S}} \log p(\hat{S}|S)$. Empirically, we found that the model typically converges quite well after only one exposure to all the training samples in the pretraining stage. Therefore, we pretrained the model for only one epoch in all the experiments.

4.1. The Adding Problem

The adding problem is a toy benchmark used to evaluate the power of recurrent models in learning long-term dependencies [9]. The input to the LSTM is a two-dimensional sequence. One dimension is sampled from a uniform distribution in the range $[0, 1]$ and the other is a mask signal. The mask is usually 0 but it has a value of 1 at two random steps. The LSTM is trained to predict the sum of the two numbers

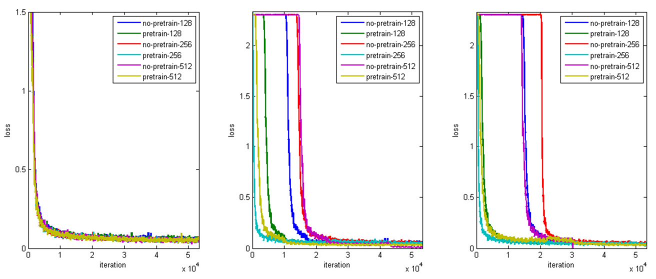

Fig. 3. Evolution of training loss on MNIST without pretraining and with pretraining w.r.t. different layer sizes. Left: LSTM with 1 layer. Middle: LSTM with 2 layers. Right: LSTM with 3 layers. (Due to the space limitation, the full size figure is given in the supplementary material.)

at the points where the mask is 1.

To evaluate our layer-wise training of LSTM, we generated $100,000$ training examples and $10,000$ test examples using sequences of length $T = 200$. The positions of the two masks that have a value of 1 are randomly selected from $[0, T/10]$ and $[9T/10, T]$. Test samples with absolute prediction errors larger than 0.04 are regarded as incorrect predictions. The results of randomly initialized LSTMs and layer-wise trained LSTMs are summarized in Table 1., which shows that layer-wise training helps to improve the performance of LSTMs on the sequence regression task.

4.2. MNIST Classification

In this experiment, we transformed the MNIST inputs into a sequence classification problem by converting the pixels into binary representations and presenting the image to the LSTM row by row. Therefore, the 28×28 images generate sequences of 28 bit vectors of dimension 28. The network is required to predict the next digit after all 28 vectors have been presented to the LSTM. In all the experiments, LSTM models were trained on the data with 256 states and a learning rate between 10^{-1} and 10^{-5}. For the randomly initialized LSTM, the weights were initialized by $U[-0.003, 0.003]$. Table 2 lists the results. We can confirm that layer-wise pretraining of LSTMs can also improve their performances on sequence classification tasks.

To further investigate the effectiveness of layer-wise pretraining on the evolution of LSTMs learning, we present all the training losses and test errors during the entire learning process w.r.t. different learning rates and layer sizes, as shown in Fig. 3 and Fig. 4. These results indicate that layer-wise pretraining improves LSTM models in the following three aspects:

- With pretraining, the model converges faster because pretraining can cause the weights of LSTM to be located closer to good local minima.
- With pretraining, the model achieves reduced test error levels under comparable training losses, demonstrating that layer-wise pretraining improves the generalization ability of the LSTM model.
- For different layer sizes and different learning rates, the LSTM with layer-wise pretraining shows consistent superiority in terms of both training losses and testing errors. This result verifies that pretraining reduces the model's sensitivity to parameters.

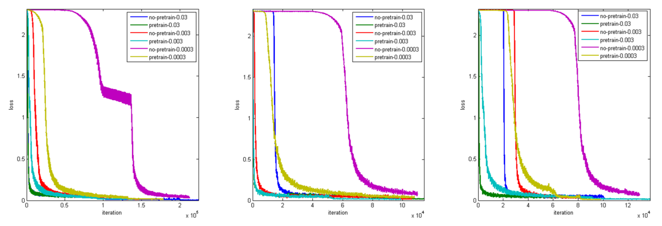

Fig. 4. Evolution of training loss on MNIST without pretraining and with pretraining w.r.t. different learning rates. Left: LSTM with 1 layer. Middle: LSTM with 2 layers. Right: LSTM with 3 layers. (Due to the space limitation, the full size figure is given in the supplementary material.)

Table 3. Performance of LSTMs on UCF-101 split 3 dataset.

Methods	Accuracy
Random initialized (1 layer)	78.67%
Layer-wise initialized (1 layer)	79.35%
Random initialized (2 layers)	79.01%
Layer-wise initialized (2 layers)	79.67%
Random initialized (3 layers)	78.93%
Layer-wise initialized (3 layers)	79.74%

4.3. UCF-101 Video Classification

The UCF-101 dataset contains $13,320$ videos with an average length of 6.2 sec [14]. Each video belongs to one of 101 different action categories. Approximately $9,500$ videos are used for training and the rest for testing. This dataset has three different training and testing splits for evaluation purposes. In this paper, we conducted experiments on split 3.

The LSTM models were trained on data with $1,024$ states, and the learning rate selected was between 10^{-1} and 10^{-5}. For the random initialized LSTM, the weights were initialized from a uniform distribution of $U[-0.003, 0.003]$. Batch size was set to 10, the gradient clip was set to 35 and the momentum was set to 0.9. The test results are presented in Table 3. We can conclude that layer-wise pretraining helps LSTMs achieve much better performances on the video recognition task.

4.4. Machine Translation

In the machine translation task, we compare our layer-wise pretraining initialization and random initialization scheme against the model proposed in [13]. We train and test LSTMs

Table 4. Performance of LSTMs on WMT'14 dataset.

Methods	BLEU
Random initialized (1 layer)	24.33
Layer-wise initialized (1 layer)	25.37
Random initialized (2 layers)	26.49
Layer-wise initialized (2 layers)	27.38
Random initialized (3 layers)	28.17
Layer-wise initialized (3 layers)	29.03

on a subset of the $12M$ sentences in the WMT'14 English to French dataset. This dataset consists of $304M$ English words and $348M$ French words. The most frequently occurring $160,000$ words in English sentences and $80,000$ words in French sentences were used to build the vocabulary. Every out-of-vocabulary word is replaced with a special "_UNK" token. The beginning and ending of sentences are marked with "_GO" and "_EOS" tokens, respectively.

All the layers in the LSTMs had $1,000$ states, and all the input words were embedded as a dense representation with $1,000$ dimensions. Sentences were inputted to the encoding LSTM in reverse order. A naive *softmax* layer with $8,000$ outputs was used to predict each word in the target sentence. All the weights were initialized to $U[-0.008, 0.008]$. The learning rate was initialized to 0.5 and decreased by 0.99 when no improvement occurred over the previous $3,000$ iterations. We chose a batch size of 64. Gradients in each update were clipped to a norm of 5. To compromise between constructing a graph for every pair of lengths and padding to a single length, we used a number of buckets with lengths of $[(5,10),(10,15),(20,25), and (40,50)]$ and padded each sentence to the length of the bucket with a "_PAD" token. Performances of the models are evaluated by BLEU scores using *multi-bleu.pl* the ground truth and predictions. Experimental results are listed in Table 4. The results indicate that sequence-to-sequence learning tasks also benefit from greedy layer-wise pretraining of LSTMs.

5. CONCLUSIONS

In this paper, we proposed a novel unsupervised layer-wise approach to initializing LSTM networks. This learning strategy generally improves the performance of LSTMs on regression, image recognition, video recognition and sequence-to-sequence learning tasks. Our analysis implies that, with layer-wise pretraining, LSTMs converge faster, have reduced test error and are less sensitive to hyperparameters. In future work, we plan to investigate the importance of pretraining for each layer in the deep LSTM models, and which memory and sequence features are learned after layer-wise pretraining.

6. ACKNOWLEDGEMENT

This work was supported by National Key R&D Program of China 2017YFB1002203, NSFC No.61572451, Youth Innovation Promotion Association CAS CX2100060016, and Fok Ying Tung Education Foundation WF2100060004.

7. REFERENCES

[1] Nitish Srivastava, Elman Mansimov, and Ruslan Salakhutdinov, "Unsupervised learning of video representations using lstms," in *ICML*, 2015, pp. 843–852.

[2] Dumitru Erhan, Pierre-Antoine Manzagol, Yoshua Bengio, Samy Bengio, and Pascal Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *AISTATS*, 2009, pp. 153–160.

[3] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[4] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle, "Greedy layer-wise training of deep networks," in *NIPS*, pp. 153–160. 2007.

[5] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio, "Why does unsupervised pre-training help deep learning?," *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[6] Jason Tyler Rolfe and Yann LeCun, "Discriminative recurrent sparse auto-encoders," in *ICLR*, 2013.

[7] Félix G. Harvey and Christopher J. Pal, "Semi-supervised learning with encoder-decoder recurrent neural networks: Experiments with motion capture sequences," *CoRR*, vol. abs/1511.06653, 2015.

[8] Andrew M. Dai and Quoc V. Le, "Semi-supervised sequence learning," *CoRR*, vol. abs/1511.01432, 2015.

[9] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015.

[11] W. Zaremba and I. Sutskever, "Learning to execute," *arXiv preprint*, vol. arXiv:1410.4615, 2014.

[12] Xavier Glorot and Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V Le, "Sequence to sequence learning with neural networks," in *NIPS*, Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, Eds., pp. 3104–3112. 2014.

[14] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah, "Ucf101: A dataset of 101 human actions classes from videos in the wild," *CoRR*, vol. abs/1212.0402, 2012.