A pipelined Pre-training algorithm for DBNs

Zhiqiang Ma [1[0000-0003-0006-2044]] Tuya Li [1] Shuangtao Yang [1] Li Zhang [1]

College of Information Engineering, Inner Mongolia University of Technology, Hohhot, CHINA
{675898486,2297854548,60130107,2550896731}@qq.com

Abstract. Deep networks have been widely used in many domains in recent years. However, the pre-training of deep networks is time consuming with greedy layer-wise algorithm, and the scalability of this algorithm is greatly restricted by its inherently sequential nature where only one hidden layer can be trained at one time. In order to speed up the training of deep networks, this paper mainly focuses on pre-training phase and proposes a pipelined pre-training algorithm because it uses distributed cluster, which can significantly reduce the pre-training time at no loss of recognition accuracy. It's more efficient than greedy layer-wise pre-training algorithm by using the computational cluster. The contrastive experiments between greedy layer-wise and pipelined layer-wise algorithm are conducted finally, so we have carried out a comparative experiment on the greedy layer-wise algorithm and pipelined pre-training algorithms on the TIMIT corpus, result shows that the pipelined pre-training algorithm is an efficient algorithm to utilize distributed GPU cluster. We achieve a 2.84 and 5.9 speed-up with no loss of recognition accuracy when we use 4 slaves and 8 slaves. Parallelization efficiency is close to 0.73.

Keywords: component; deep networks; pre-training; greedy layer-wise; RBM; pipelined.

1 Introduction

Recently, deep networks have been widely used in many domains because of its powerful modeling capacity, including speech recognition [1], image recognition [2] and natural language processing [3]. However, deep neural networks have not been discussed much in machine learning literature before Hinton et al introduced a greedy layer-wise unsupervised pre-training algorithm to train such multi-layer neural networks, a reasonable explanation is that there were no efficient algorithms to train such deep neural networks, since gradient-based optimization starting from random initialization appears helpless [4]. The greedy layer-wise unsupervised pre-training algorithm can quickly find a fairly good set of parameters by greedily training one layer at a time, even with millions of parameters and many hidden layers. Reference [5] successfully trained a deep belief networks for MNIST digit classification and achieved better digit classification than any discriminative learning algorithms. Then this

greedy layer-wise algorithm was analyzed and extended in [6] which made it become a general algorithm to initialize deep networks.

Since Greedy layer-wise pre-training algorithm was proposed, it proved to be effective by lots of successful practices. However, scalability of greedy layer-wise pre-training algorithm is greatly restricted by its inherently sequential nature and only one hidden layer can be trained at one time because of data dependency. As the training data and the depth of deep networks continue to grow, the pre-training of deep networks becomes more and more time-consuming, despite the use of high performance GPU and other optimization strategies [7].

In order to speed up the training of deep networks, this paper proposes a pipelined pre-training algorithm for distributed cluster, which can significantly reduce the pre-training time at no loss of recognition accuracy. It's more efficient than the greedy layer-wise pre-training algorithm and it speeds up the training of deep networks obviously by using the computational resources of distributed cluster. The outline of this paper is as follows: Section 2 mainly discusses previous works on parallelization of deep networks training. Section 3 briefly introduces restricted Boltzmann machine deep belief networks and greedy layer-wise pre-training algorithm. In section 4 there is a detail description of pipelined pre-training algorithm. In section 5, the experiments are conducted and discussed. Section 6 summarizes the works in this paper and gives some suggestions on future works.

2 Relate Work

Many efforts have been devoted to using the distributed cluster in order to accelerate the training of deep networks. In previous works, parallelization of deep networks training mainly includes model parallelism and data parallelism [8].

To facilitate the training of super large deep networks, Google developed a framework that is called DistBelief, it supports parallel training of super large deep networks on CPU cluster [9]. The success of DistBelief shows that the parallel performance advantage depends on the model's connection structure and computing requirements. Reference [10] developed a neural-net training framework which adopted different versions of data parallelism. Each slave node in the framework has a copy of the entire model and has a different randomly selected subset of SGDs on the copy. After all slaves process a fixed number of training data, the copies across all slaves will be averaged and re-distributed to each slave for further training until all training cases are processed.

GPU-based distributed parallelization is more common compared to CPU-based distributed parallelization, because the use of a smaller number of GPU cards can achieve satisfactory acceleration. However, a number of existing deep learning frameworks do not support distributed GPU parallelization across multiple machines currently, most of them just can take advantage of GPUs on the same machine, such as Torch [9] and Theano [10]. In order to enhance these distributed GPU popular framework. Spark Net [11] was proposed, which supports to train deep networks in Spark and it achieves a 4-5 times speedup with 10 machines equipped GPU cards. In

order to overcome the difficulty of parallelize back-propagation algorithm, asynchronous stochastic gradient descent algorithm (ASCD) was used in fine-tune the DNN on multi-GPU cards in [12]. Each GPU computes gradient on the latest parameters independently and updates the parameters asynchronously. In this way, a 3.2 times speedup was achieved with 4 GPU cards without any performance loss. In order to speed up the training of Multilingual DNN on multiple GPU cards, [13] proposes two distribution frameworks which is called DistModel and DistLang. Each GPU trains an instance of the multilingual DNN by a part of training data in DistModel, and these parallel model instances are averaged periodically after a predefined number of mini-batches. The multilingual DNN instances are trained separately by languages without any communications in DistLang. In essence, these two distribution frameworks are both belong to data parallelism.

Besides model parallelism and data parallelism, there are also other parallel strategies have been proposed for deep networks training. Reference [14] found that the pipelined back-propagation can update models with delayed gradient and allow training layers parallel. Experiments showed that the pipelined BP is an efficient way of utilizing multiple GPUs in a single machine. It achieved 1.9 and 3.3 times speed-up with 2 and 4 GPUs at no loss of recognition accuracy. However, this strategy is only suitable for discriminative training. Different from above acceleration strategies, [15]diverted its attention to the per-training phase and proposed a synchronized greedy layer-wise algorithm. The synchronized algorithm allows training different layer parallel by multiple threads running on different cores with regular synchronization. Experiments on dimensionality reduction of MNIST showed that this algorithm achieved 26% speed-up compared to greedy layer-wise pre-training algorithm with the same reconstruction accuracy.

3 DBN and Greedy layer-wise algorithm

DBN is a deep generative model with many layers of hidden causal. It can be learned efficiently by stacking multiple RBMs from bottom to top with greedy layer-wise pre-training algorithm. The pre-training of DBNs mainly include three steps as shown in Figure 1 (from left to right).

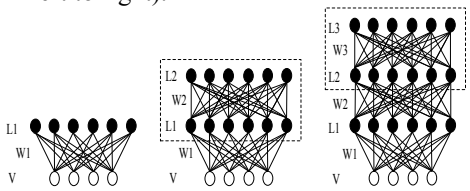

Fig.1. Greedy Layer-wise Pre-training of DBNs.

Left: construct a RBM with an input layer v and a hidden layer L_1.The number of units in input layer depends on the dimensionality of input data, then use CD-1algorithm to train this RBM. It will obtain one layer representation from input data after the training is finished, and the representation will be used as input for second

RBM. In this paper, the activations of each hidden unit are chosen as the representations.

Middle: Hidden layer L_2 is newly added layer and the second RBM is constructed by L_1 and L_2.Then train it like the first RBM, the only difference is that input for the second RBM is computed according to equation (1) from input data according to RBM.

$$p(h_j = 1|v) = \sigma(b_j + \sum_i v_i w_{ij}) \qquad (1)$$

Right: continue to stack new hidden layers on the top and train it as previous until the DBNs are all trained.

Since greedy layer-wise algorithm was proposed, it has been proved to be effective through lots of successful practices [7]. However, scalability of greedy layer-wise algorithm is greatly restricted. L_2 has to wait until L_1 have finished all training task, because L_2 needs the output from L_1as input. The greedy layer-wise algorithm is quite simple, as illustrated in Algorithm 1.Function RBM update means the training of RBM with CD-1.

Algorithm 1: Greedy Layer-wise algorithm
ϵ: learning rate for CDK algorithm
L: number of hidden layers
W^i: weight matrix
b^i : bias vector for hidden layer

```
split dataset into batches[]
initialize
For = 1 to do
  Initialize  =0, =0
  For e = 1 to epoch do
For mini-batch in batches
RBMupdate (mini-batch,  ϵ, W^i , b^i , b^(i−1) )
End for
  End for
End for
```

4 Pipelined pre-training algorithm

The data dependency between adjacent hidden layers makes it hard to parallel the pre-training of DBNs. In order to achieve the parallelization, it is necessary to overcome the inherently sequential nature of Greedy Layer-wise algorithm. This paper proposes a pipelined pre-training algorithm with this intent, which is more efficient than the greedy layer-wise pre-training algorithm and it speeds up the training of deep networks obviously by using the computational resources of distributed cluster.

A new concept called middle result is proposed in pipelined pre-training algorithm, that is shown in Figure 2.

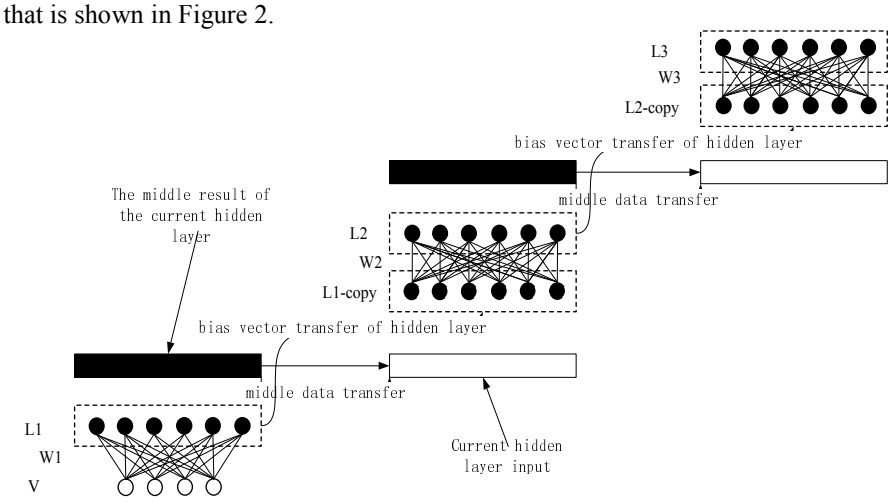

Fig.2. Data transfer in DNN Pipelined pre - training

When using CD-1 to train the first RBM in Figure 1, the middle result is calculated. It computes the activation probability of each hidden unit via equation (1) again, and get the binary state vector h1, which is the reconstruction for h0. At this point, all the parameters can be updated via equation (2), (3), (4).

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}) \quad (2)$$

$$\Delta a_i = \epsilon(\langle v_i \rangle_{data} - \langle v_i \rangle_{recon}) \quad (3)$$

$$\Delta b_j = \epsilon(\langle h_j \rangle_{data} - \langle h_j \rangle_{recon}) \quad (4)$$

The middle result can be used as input for the second RBM. Thus, hidden layer L_1 and L_2 are trained in parallel. In order to make a detailed description of the proposed algorithm, we make following statements:

$L_i$is the i^{th} hidden layer, L_0 is the input layer.
$B_i$is the bias vector of L_i .
L_{i-copy} is the copy of hidden layerL_i .
$L_{i-1}$and form RBM_i that is trained at M_i .
L_{i-copy}andL_{i+1} form RBM_{i+1} that is trained at M_{i+1} .

Generally, the pipelined pre-training algorithm mainly does three things:

1. Segment DBNs into multiple RBMs and assign each RBM to an exclusive computer. For example, as shown in Figure 3 (left), RBM1 and RBM2 are adjacent and share the hidden layer L_1. In order to make RBM1 and RBM2 independent of each

other, we use the copy of L_1 instead of L_1 to construct RBM2, thus RBM2 is formed by $L_{i-copy}$and L_2 .

2. Start pre-training from RBM1 which is formed by input layer and hidden layerL_1. Transmit B_1 and the middle result to RBM2 after RBM1 finishes a training of K mini-batches, initializing the bias of L_{1-copy} with B_1 at the first when RBM2 receives them and then start the training of RBM2 with middle result. The transmission of biases and middle results is achieved by message communication between computers, which is indicated by dotted line in Figure 3 (right).

3. Train RBM_{i+1} with B_i and the middle result from RBM_i until all hidden layers are trained. Collecting all parameters in each RBM after pre-training is finished and begin to fine tune the entire network.

It should be pointed out that RBM_i is composed of hidden layer L_i and hidden layer L_{i-1} of the DBN. The pseudo-code of pipelined pre-training algorithm is shown in Algorithm2.

Algorithm 2: Pipelined pre-training algorithm
ϵ: learning rate for CDK algorithm
L: number of hidden layers
W^i: weight matrix
b^i : bias vector for hidden layer
o_e^i:middle result of hidden layer at epoch e

```
Master:
Initialize cluster
split DBNs to slaves
Slave:
Fori = 1 to L parallel do
Initialize W^i , b^i
  For e = 1 to epoch do
If i = 1
batches[] = split from dataset
   Initialize b^0
Else
receive o_e^(i-1) , b^(i-1) from L_(i-1)
batches[] = split from o_e^(i-1)
For mini-batch in batches
RBMupdate (mini-batch, ϵ, W^i , b^i ,b^(i-1) )
End for
Send o_e^i , b^i to L_(i+1)
    End for
End for
```

The training process of pipelined pre-training algorithm is shown in Figure 3(Right), all hidden layers are trained parallel in the computational cluster. As shown in Figure 3, in order to pre-train a deep belief networks containing 5 hidden layers (L1, L2, L3, L4, L5) with pipelined pre-training algorithm, the deep belief networks are segmented into 5 RBMs, and each RBM is trained on its special computer in the distributed clus-

ter. For example, RBM1 formed by input layer and hidden layer L1, it is trained on machine M1, RBM2 formed by the copy of hidden layer L1 (represented by L1-copy) and hidden layer L2, it is trained on Machine M2.

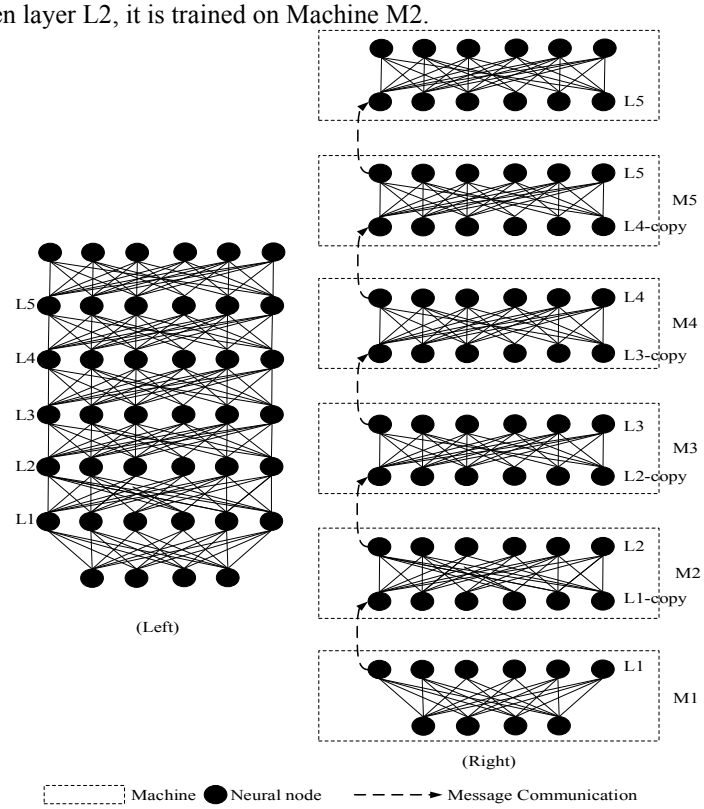

Fig. 3 Pipelined Per-training of DBNs

In pipelined pre-training, when RBM1 finishes a training of K mini-batches every time, it will produce a number of middle results, and update B_1(biases of hidden layer L1) K times. Then, these middle result andB_1will be transmitted to RBM2 through message communication. It first sets the biases of L1-copy toB_1according to the rule that biases of the copy should always be consistent with the biases of its original after RBM2 receives the middle result andB_1, and then begins the training with received middle result. Actually, every time RBM2 finishes a training of K mini-batches, it also produces some middle results, and updates the biases of L1-copy. But these updates will not be transmitted backward to RBM1 and the updated biases of L1-copy only survive to next communication. The biases of the L1-copy will always be set toB_1 that received from RBM1, because RBM1 has been trained with more training data, the biases of L1 is more suitable to training data.

5 Experiment

In order to better understand the advantage brought by pipelined pre-training algorithm, experiments are performed on TIMIT corpus with the task of speech recognition. TIMIT is a popular corpus of speech recognition, and many recognition experiments have been conducted on this popular corpus, and greedy layer-wise algorithm is also evaluated on it. So, it's ideal for evaluating pipelined pre-training algorithm.

In the TIMIT corpus experiment, this paper carries on the contrast experiment of the greedy layer-wise unsupervised pre-training algorithm and the pipelined pre-training algorithm in different hidden layers. The experimental data for TIMIT are shown in Table 1

Table 1. Import parameters.

parameter name	parameter value
activation function	tanh
number of neural units in hidden layer	1024
initial learning rate of pre-training	0.015
the final learning rate	0.002
K value of CD-K algorithm	1
Pre-training cycle times	20
Mini-batch size	256

In order to conduct pipelined pre-training of DBNs, a distributed parallel framework oriented distributed cluster is introduced, and the architecture is shown in Fig 4. This paper only makes a brief introduction about this framework. It adopts master-slave structure, the master node is responsible for the schedule and monitor of entire DBNs during pre-training, slaves are responsible for the training of each hidden layer. In each slave, training work is done by Theano, and middle results are transmitted through message communication. All slaves (eight slaves) are equipped with four 3.20GHz CPU and one GPU card GeForce GTX 660. All distributed experiments are conducted in this framework.

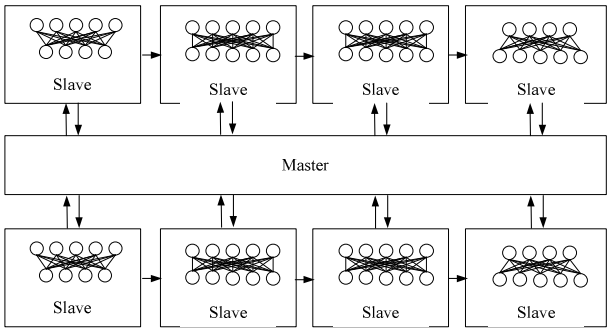

Fig. 4 Distributed experiment framework

5.1 Recognition Accuracy

Three deep belief networks are pre-trained using greedy layer-wise and pipelined pre-training algorithm respectively, and the sentence recognition error rate shown in Table 2.

Table 2.Recognition Error rate of each DBN

platform	algorithm	the number of layers of hidden layers	sentence recognition error rate
Theano	greedy layer-wise algorithm	4	22.81%
		5	19.92%
		6	18.35%
pipelined pre - training framework	pipelined pre-training algorithm	4	23.43%
		5	20.02%
		6	18.72%
Kaldi	greedy layer-wise algorithm	4	22.81%
		5	19.92%
		6	18.35%

5.2 Time Complexity of Pipelined Pre-training Algorithm

Under the premise of no loss of recognition accuracy, pipelined pre-training algorithm is more efficient and it speeds up the pre-training of deep networks obviously by using the computational cluster. Time of all pre-trainings is shown in Table 3. Before discussion, it should be pointed out that all pre-trainings using pipelined pre-training algorithm are trained with enough computers, which means that each hidden layer is trained on an exclusive computer.

Table 3. Time of each pre-training.

M	L				
	4	5	6	7	8
1	68.68	90.3	113.67	139.78	166.17
1	67.89	90.12	112.71	139.82	168.02
4	24.17				
5		25.09			
6			25.95		
7				26.69	
8					28.15

In Table 3, L indicates the number of hidden layers in deep belief networks and M indicates the number of computers in distributed cluster. The first row (red units) is the pre-training time for DBNs on a single machine (single GPU) using greedy layer-

wise pre-training algorithm. Second row (yellow units) is the pre-training time for DBNs on a single machine (single GPU) using pipelined pre-training algorithm. Other rows (green units) are the pre-training time for DBNs in a distributed cluster with pipelined pre-training algorithm, each slave in the distributed cluster has only one GPU card. These blank units in Table 3 means there are no experiments conducted.

As shown in Table 3, the pre-training time grows sharply with the increase of hidden layers when using greedy layer-wise algorithm to pre-train DBNs on a single computer. A DBN having 4 hidden layers can be trained with 68.68 minutes, but it needs 166.17 minutes to pre-train a 8-hidden layers DBN, the pre-training time increases by 97.49 minutes. Pipelined pre-training can significantly reduce the pre-training time compared to greedy layer-wise algorithm, increasing the number of hidden layers does not lead to a dramatic increase in the training time anymore. And the increase in training time using pipelined pre-training algorithm is slow and gentle. For example, pre-training time of a DBN with 4 hidden layers is 24.17 minutes and it's 28.15 in a 8-hidden layers DBN, there is only a 3.98 minutes-growth.

When using pipelined pre-training algorithm to pre-train a deep belief networks with N hidden layers (represented by L1, L2,…, LN), training process is shown in Figure 5. As shown in Figure 5, Blank rectangle indicates a training of K mini-batches in current RBM. Solid line with arrow indicates the beginning of next K mini-batches pre-training in current RBM. Dotted line indicates the transmission of middle result and biases between original hidden layer and its copy. The trainings between these RBMs are not completely parallel. Upper layer will have a short delay compared to lower layer, because it has to wait for the lower layer to finish a K mini-batches training, transmit middle results and biases. For example, R2 has a short delay compared to R1, R3 has a short delay compared to R2, and the delay nearly equals to the time of a K mini-batches pre-training, this is why there exists a tiny growth in Table 3.

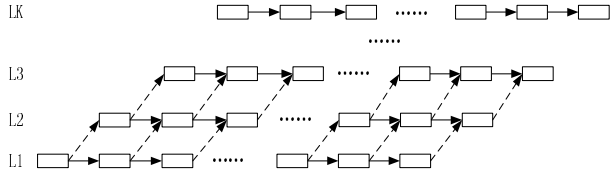

Fig. 5 Pipelined Pre-training of a k Hidden Layers

5.3 Speed-up of Pipelined Pre-training Algorithm

The speed up of pipelined pre-training algorithm is shown in Fig 6, which is calculated from Table 3.

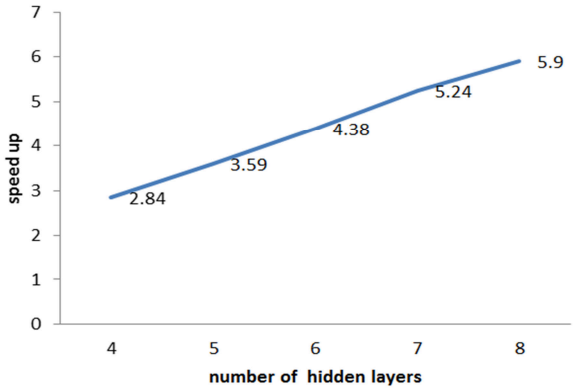

Fig. 6 Speed up of Pipelined pre-training algorithm

The acceleration of pipelined pre-training algorithm mainly benefits the simple connectivity structure. Each slave node only connects two other slave nodes, which greatly reduce the communication overheads of the cluster during the per-training. There is few data needs to be transmitted between slaves compared to other parallelisms. The data that needs to be transmitted between slaves in every communication only contains K mini-batches middle results and a bias vector, which are surprisingly few compared to the entire model.

6 Conclusion

In order to accelerate the training of deep networks, many efforts have been devoted to leveraging distributed cluster to speed up the training of deep networks, but previous works mainly concentrate on the fine-tuning phase. In order to overcome the inherently sequential nature of greedy layer-wise algorithm, a pipelined pre-training algorithm is proposed, which is more efficient than the greedy layer-wise pre-training algorithm and it speeds up the training of deep networks obviously by using distributed cluster. For the above experimental results through the TIMIT corpus, we know that speed up of the proposed algorithm get a linear increase with the number of slave node and the parallelization efficiency is close to 0.73 compared to greedy layer wise algorithm. In addition, Pipelined pre-training algorithm is more suitable for distributed cluster, it's easy to implement. Although all experiments in this paper are conducted on GPU, the proposed algorithm also supports CPU pre-training well.

However, there are still lots of works to do in the future. We will mainly concentrate on two things: (1) Expands pipelined pre-training algorithm to other types of deep networks. (2) Parallel the training of a hidden layer to more computers rather a single machine.

Acknowledgments

Funding project: National Natural Science Foundation of China (61650205). Inner Mongolia Autonomous Region Natural Sciences Foundation project (2014MS0608). Inner Mongolian University of Technology key Fund (ZD201118).

References

1. Dahl, George E., et al. "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition." Audio, Speech, and Language Processing, IEEE Transactions on 20.1 (2012): 30-42
2. Krizhevsky, Alex, IlyaSutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.
3. Sarikaya, Ruhi, Geoffrey E. Hinton, and AnoopDeoras. "Application of deep belief networks for natural language understanding." Audio, Speech, and Language Processing, IEEE/ACM Transactions on 22.4 (2014): 778-784.
4. Bengio, Yoshua. "Learning deep architectures for AI." Foundations and trends® in Machine Learning 2.1 (2009): 1-127.
5. Hinton, Geoffrey E., Simon Osindero, and Yee-WhyeTeh. "A fast learning algorithm for deep belief nets." Neural computation 18.7 (2006): 1527-1554.
6. Bengio, Yoshua, et al. "Greedy layer-wise training of deep networks." Advances in neural information processing systems 19 (2007): 153.
7. Dean, Jeffrey, et al. "Large scale distributed deep networks." Advances in Neural Information Processing Systems. 2012.
8. Seide, Frank, et al. "On parallelizability of stochastic gradient descent for speech DNNs." Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.
9. Collobert, Ronan, KorayKavukcuoglu, and Clément Farabet. "Torch7: A matlab-like environment for machine learning." BigLearn, NIPS Workshop. No. EPFL-CONF-192376. 2011.
10. Bergstra, James, et al. "Theano: Deep learning on gpus with python." NIPS 2011, BigLearning Workshop, Granada, Spain. 2011.
11. Moritz, Philipp, et al. "SparkNet: Training Deep Networks in Spark." arXiv preprint arXiv:1511.06051 (2015).
12. Zhang, Shanshan, et al. "Asynchronous stochastic gradient descent for DNN training." Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.
13. Miao, Yajie, Hao Zhang, and Florian Metze. "Distributed learning of multilingual DNN feature extractors using GPUs." (2014).
14. Chen, Xie, et al. "Pipelined Back-Propagation for Context-Dependent Deep Neural Networks." INTERSPEECH. 2012.
15. Santara, Anirban, et al. "Faster learning of deep stacked autoencoders on multi-core systems using synchronized layer-wise pre-training." arXiv preprint arXiv:1603.02836 (2016).