Predicting the Training Time of Deep Neural Networks

Jitendra Yasaswi Suresh Purini C. V. Jawahar

IIIT Hyderabad, India

***Abstract*—In this work, we propose a method to estimate the time required in terms of epochs, to train a deep neural network to achieve a given accuracy. We perform a time series analysis over the observed accuracy values from few initial epochs and learn a function using regression. We report various functions that are learned over the observed data. We present an in-depth analysis on the behavior of the learned function. The proposed method is able to make predictions accurately even at early stages of training. We demonstrate the utility of our proposed method and show promising experimental results on challenging image classification datasets.**

I. Introduction

Deep neural networks (DNNs) have become a great success in large-scale visual recognition tasks [1]–[5]. However, training deep neural networks has never been easy [6]. A thorough investigation on the difficulty of training deep neural networks is presented in [7]. Training deep neural networks has become possible because of the availability of large data sets like ImageNet [8] and high-performance computing systems like GPUs and efficient training schemes such as dropout [9], batch normalization [10] etc. In general, a gradient descent algorithm minimizes a predefined cost function during the training process. Deep neural networks are trained with batches of data where each batch contains a fixed number of data samples. Every sample is used to advance the minimization of the cost function. The optimization proceeds slowly that in many cases one stops training after a predetermined number of iterations. Training requires user parameters that are difficult to set and their effect on the convergence is crucial: learning rate and its decay, momentum, mini-batch size, etc. [11]. The above mentioned training process demands heavy computations and lot of time. In deep learning, the optimization method usually used is stochastic gradient descent methods (SGDs). Recent work on SGDs focuses on adaptive strategies for the learning rates [12] or improving SGD convergence by approximating second-order information [13]. A thorough survey on optimization methods for deep learning is presented in [14].

When it comes to training, a general strategy is to train DNNs using GPU systems [1], [2], [15]. In [15], the authors developed general principles for massively parallelizing unsupervised learning tasks using graphics processors. The other approach is to train deep neural networks using CPU cores. In [16], the authors developed a framework called *DistBelief* that enables model parallelism and data parallelism, where multiple replicas of a model are used to optimize a single objective. In [17], the authors present an alternative approach for training large scale networks that leverage inexpensive computing power in the form of GPUs. Here, they present a deep learning framework with COTS HPC systems: a cluster of GPU servers with Infiniband interconnects and MPI. Even with the availability of the state-of-the-art high-performance computing systems, it requires days to train deep neural networks. For example, it takes six days to train AlexNet [1] on two GTX 580 3GB GPUs.

In this work, we address the problem of estimating the time required to train a deep neural network to achieve a given accuracy. We come up with a predictive analysis to estimate the training time (in number of epochs) required for a deep neural network to achieve a given accuracy. We propose a method which involves an iterative algorithm that takes the observed accuracy values of n initial epochs as input during training and estimate the time required to achieve a given accuracy. We perform a time series analysis over the observed accuracy values from few initial epochs and learn a function using regression. We formulate the task of learning a function as an optimization problem and solve it. Our method can give a good estimate to answer queries such as "how many epochs are required to train a deep neural network to achieve 95% accuracy?" while training online. We can estimate the actual training time by taking the product of the number of epochs and the time taken by each epoch. Since the structure of computations across epochs remains the same, the time taken by each epoch does not vary much. And it can be noted during the training process itself. The effectiveness of our proposed method lies in its simplicity, the ability to make predictions at early stages of training (say at third epoch itself, it is able to predict) and the accuracy of the predictions. We demonstrate the utility of our training time prediction algorithm by applying it to standard image classification tasks and argue that this would benefit the recent deep learning frameworks such as caffe [18], torch [19] etc.

The problem statement and evaluation measures are discussed in Section II. The main training time prediction technique is presented in Section II-B. In Section III, we provide experimental results and

analysis demonstrating the utility of our method and finally conclude our work in Section IV.

II. Problem Statement

Consider the online scenario of training a deep neural network. Given the training accuracy at each epoch, up to epoch n, our goal is to estimate (predict) the epoch at which a specified training accuracy is achieved. Towards that, we perform time series analysis over the observed accuracy values from few initial epochs and learn various functions using regression. Let $(x_i, y_i), i = 1, \ldots, m$, be the input data pairs where y_i is the epoch number and x_i is the corresponding accuracy value at epoch y_i. We call these input data pairs as the observed data. These recorded epoch numbers to reach various accuracies will serve as the ground truth epochs. Each of the functions learned over the observed data has the form $f(x, \alpha)$, where 'α' is a k-dimensional parameter vector. The parameters of a learned function are adjusted so as to minimize the difference between y_i and the predicted epoch value by the function. This is called residual error, which is

$$e_i = y_i - f(x_i, \alpha). \tag{1}$$

The goodness of the functional fit is measured using the sum of squared errors (SSE) which is calculated as

$$S = \sum_{i=1}^{m} e_i^2, \tag{2}$$

where e_i are the residual errors as in Eq 1. Given a new accuracy value X to achieve, choose and use one of the learned function to predict the epoch Y at which this accuracy is achieved. The above mentioned process is repeated for various trials (here trial means a stage of training, say for $n = 5, 10, 12, \ldots$). The criteria up on which a function is chosen is described in Section II-B.

The utility of our proposed method lies in its ability to give an estimate of training time beforehand while training deep neural networks online. It can be incorporated as a functionality that would benefit the recent deep learning frameworks such as torch, caffe etc. Many previous attempts to train deep neural networks from random initializations have likely failed due to poor initialization schemes [7]. Choosing a careful choice hyper-parameters like learning rate, momentum etc., choosing an architecture (i.e. number of layers), particularly when training with a limited computation budget makes it a challenging task. With our predictive analysis during training, one can gain insight into a deep neural network's behavior and can decide whether training shall be continued further. Training involves lot of computations. If training is not proceeding as desired, one can save lot of computational time and thereby computation resources. Experiments demonstrating the utility are discussed in detail in Section III.

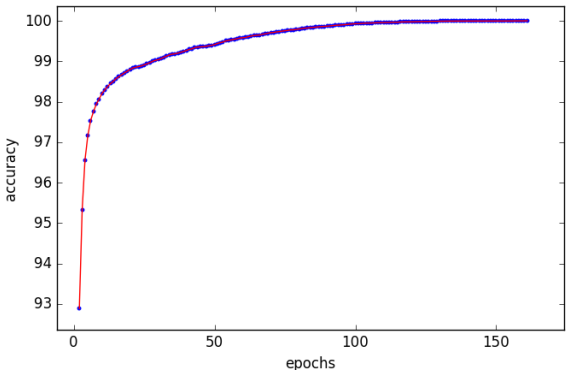


Fig. 1. A plot showing the accuracy vs. epoch, for LeNet-5 trained on MNIST data set.

A. Evaluation measures

The goodness of prediction is validated by taking the sum of squared differences between the predictions and observed ground truth values at each trial. It is given by the equation

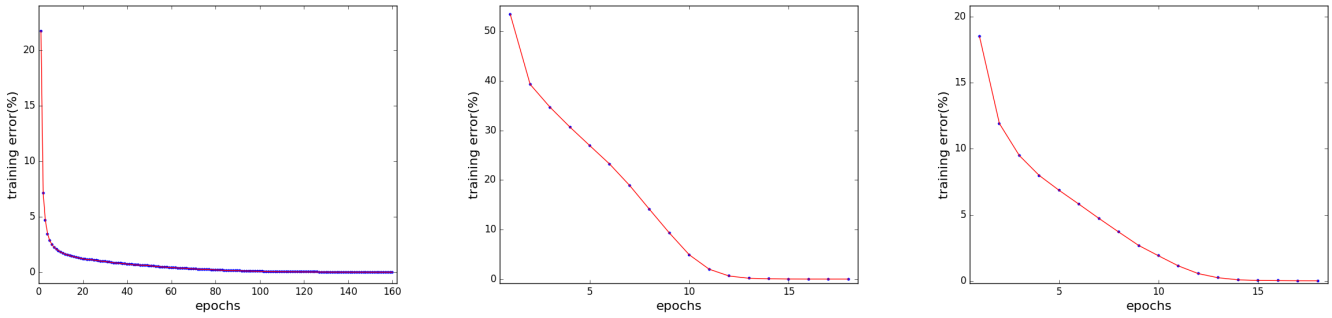
$$E = \sum_{i=1}^{t} \frac{(O_i - y_i)}{t}, \tag{3}$$

where y_i is the ground truth value, O_i is the predicted value and t is the number of valid predictions in a trial (i.e., at different stages of training, say for n = 5, 10, 12,...) . A prediction in a trial is said to be valid, if at the trial being conducted, the accuracy to achieve has not already surpassed during training. For example, the cell value in the fifth row, first column in table II is invalid (represented by -). It is invalid at epoch eight, to do a prediction to achieve 92% accuracy, as the network has already reached 92% accuracy in the 4^{th} epoch itself. So, for the prediction trial at epoch eight, the value of t will be 4. Based on the predictions, we present the function that gives best prediction at a given trial. Table I explains the calculation of our evaluation measure, which corresponds to fourth row in table II.

B. Predicting Training Time

We train a suitable deep neural network up on each of the datasets that are mentioned in the next section. The training accuracy values along with their respective epoch numbers are recorded. Fig. 1 shows the training accuracy vs. epoch for training LeNet-5 up on MNIST data set. These recorded epoch numbers to reach various accuracies will serve as the ground truth values, against which we validate our predictions.

1) Solution based on regression: Given the training accuracy at each epoch, up to epoch n, the task is to estimate epoch (predict) when accuracy is greater than or equal to X%. Our proposed algorithm takes as input the observed data pairs $(x_i, y_i), i = 1, \ldots, m$, where y_i is the epoch number and x_i is the corresponding accuracy value at epoch y_i. The first step is to fit the observed



(a) On MNIST (b) On CIFAR-10 (c) On SVHN

Fig. 2. Training error vs. epochs, of like Convolutional Neural Networks, trained on different datasets. (a) demonstrates the training of LeNet-5 on MNIST dataset, (b) and (c) demonstrate the training of convolutional network like LeNet-5 up on CIFAR-10 and SVHN datasets respectively. Note the rapid convergence of training in (b) and (c).

data points with a variety of functions using regression.

The function with the minimum SSE value is chosen to predict the epochs Y required to reach unseen accuracies X. The output of the model is the prediction value along with the function that predicts this value. The above mentioned process is repeated for different trials (i.e., at different stages of training, say for $n = 5, 10, 12, ...$).

Various functions like exponential, power and polynomial (sixth and seventh degree) are learnt using regression, over the observed accuracy values for the first few epochs. A single-term exponential function of the form, as shown below

$$y = a \cdot e^{b \cdot x} \tag{4}$$

is learned, where x is the accuracy value and y is the epoch. Similarly, a power function of the form, as shown below

$$y = a \cdot x^{b} \tag{5}$$

is learned. Similarly, polynomial functions (sixth and seventh degree), piece-wise polynomial function are also learnt. From Eq 4 and 5, 'a' and 'b' are the real valued parameters that are learned up on the data observed. It is the values and sign of these parameters that define the nature of the learned function. For example, the positive value of 'b' in Eq 1 gives an exponential growth function whereas the negative value of 'b' gives an exponential decay function.

In the next section, we show the effectiveness of various regression models in predicting training time of various DNNs on standard image classification datasets such MNIST, CIFAR-10, SVHN and ImageNet.

III. Experiments

A. Datasets

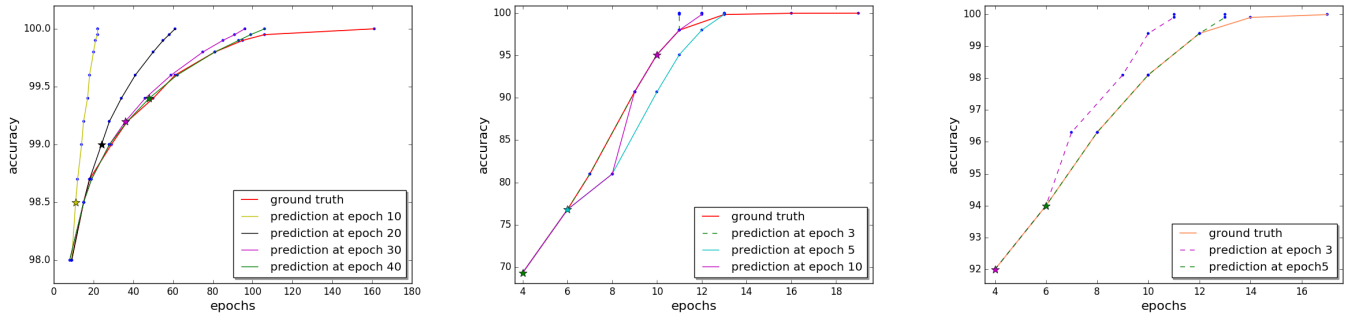
The datasets we adopt are the MNIST [20] handwritten digits dataset, the CIFAR-10 dataset [21], the Street View House Numbers (SVHN) [22] and ImageNet ILSVRC-2010 [23] dataset. All these datasets corresponds to the task of image classification.

The MNIST dataset consists of grey scale handwritten digit images of size 28×28, in 10 classes from 0 to 9. The training dataset consist of 60,000 images and test set contains 10,000 images. The original black and white images from NIST were size normalized to fit in a 20×20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. The images were centered in a 28×28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28×28 field. For our purpose, we use 32×32 images, which are result of extending to 28×28 images with background pixels.

The CIFAR-10 dataset consists of $60,000$ 32×32 RGB images in 10 classes: *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck*, with 6000 images per class. There are $50,000$ training images and $10,000$ test images. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. It is a subset of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton.

The SVHN is a real-world image dataset obtained from house numbers in Google Street View images. The digit images are MNIST-like 32×32 in size but comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). The dataset contains ten classes, digit '1' has label 1, '9' has label 9 and '0' has label 10. The training set contains $73,257$ images and test set contains $26,032$ images. SVHN is obtained from house numbers in Google Street View images.

The ILSVRC-2010 (ImageNet Large Scale Visual Recognition Challenge 2010) is a subset of ImageNet with roughly 1000 high-resolution images in each of 1000 categories, where



(a) Case I (b) Case II (c) Case III

Fig. 3. Case I shows the predictions by an exponential function on MNIST dataset. Case II shows the predictions by a piece-wise polynomial function on CIFAR-10 dataset. Case III shows the predictions by a second degree polynomial function on SVHN dataset at different trials. For each curve, the portion after the colored star represents the prediction. The performance of prediction at early stages can be observed clearly.

1.2 million images are for training, $50,000$ images are for validation and the rest $150,000$ are for testing. Each of these RGB image is of size 256×256. On ImageNet, it is customary to report two error rates: top-1 and top-5, where the top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the model.

B. Experimental Details

1) On MNIST *dataset:* We train LeNet-5 [20], a Convolutional Neural Network to carry out our experiments. LeNet-5 comprises seven layers (four convolutional and three fully connected). The input is a 32×32 image and output is a 10 dimensional feature vector. Torch framework is used to train the LeNet-5 model [24]. The original MNIST dataset is converted from their original LUSH format to Torch's internal format. LeNet-5 is trained for 200 epochs, with fixed batch size of 10 and at a constant learning rate of 0.001. Fig. 2 shows the training error vs. epochs. Using the learned function, the epoch predictions are carried out for the training to achieve accuracy of 98, 98.5, 98.7, 99, 99.2, 99.4, 99.6, 99.8, 99.9, 99.95 and 100 percent respectively. Fig. 3a shows the prediction of epochs, at various trials by the exponential function.

2) On CIFAR-10 *dataset:* A convolutional neural network from torch demos [25] is trained on CIFAR-10 dataset, with a fixed batch size of 1 and at a constant learning rate of 0.001. Using the learned function, the epoch predictions are carried out for the training to achieve accuracy of 69.33, 76.8, 81.0, 90.7, 95.1, 98.0, 99.84, 99.99 and 100 percent respectively. Fig. 3b shows the prediction of epochs, at various trials (i.e predictions carried out at different stages of training) by the piece-wise polynomial function.

3) On SVHN *dataset:* A convolutional neural network from torch demos [26] is trained on SVHN dataset, with a fixed batch size of 1 and at a constant learning rate of 0.001. Using the learned function, the epoch predictions are carried out for the training to achieve accuracy of 92, 94, 96.3, 98.1, 99.4, 99.9 and 99.99 percent respectively. Table II shows the predictions of the epochs as estimated by a piece-wise polynomial function. Fig. 3c shows the prediction of epochs, at various trials by the second degree polynomial function.

TABLE I
TABLE SHOWING THE EVALUATION MEASURE

Accuracy to achieve	92	94	96.3	98.1	99.4	99.9	99.99
Ground truths	4	6	8	10	12	14	17
Prediction at epoch 5	-	6	8	10	11	12	19
Squared error	-	0	0	0	1	4	4
Error (normalized SSE)	1.5						

4) On ImageNet dataset: We trained AlexNet on the subset of ImageNet used in ILSVRC-2010 competition. Training is carried as mentioned in [27], [28] along with batch normalization up on three GPUs (two Titan X and one GTX 970). The batch-size we used is 512 with a hard-coded learning rate schedule as mentioned here [29]. We trained till 55 epochs where the top-1 training accuracy is 28.23, 36.25, 40.96, 44.14, 47.27, 48.06, 49.2, 52.02, 54.05, 55.07, 56.1, 56.51, 57.03 and 57.06 percent respectively. Using learned function the epoch predictions are carried out for training to achieve of percent respectively. Fig. 4 shows the prediction of epochs at various trials by the single term exponential function. The performance of prediction at early stages (at epoch 15) can be observed clearly.

5) Know before you use: Choosing a careful choice hyper-parameters like learning rate, momentum etc., particularly

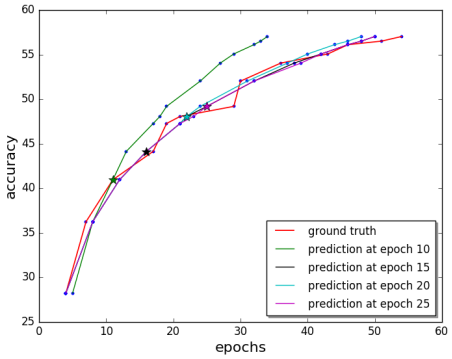


Fig. 4. The predictions by an exponential function on ImageNet dataset. For each curve, the portion after the colored star represents the prediction.

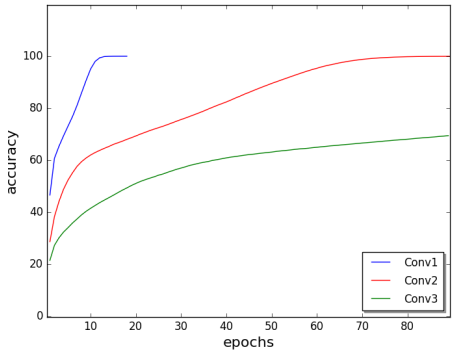


Fig. 5. Convergence behavior of a convolutional neural network with different parameter initializations, trained up on CIFAR-10 dataset.

TABLE II
PREDICTION OF THE EPOCHS AS ESTIMATED BY THE PIECE-WISE POLYNOMIAL FUNCTION, UP ON SVHN DATASET.

	Accuracy to achieve							
At epoch	92	94	96.3	98.1	99.4	99.9	99.99	Error
3	4	6	7	9	10	11	11	7.28
5	-	6	8	10	11	12	19	1.5
8	-	-	-	10	12	13	13	4.25
10	-	-	-	-	12	13	13	5.66
12	-	-	-	-	-	14	13	8
14	-	-	-	-	-	-	15	4

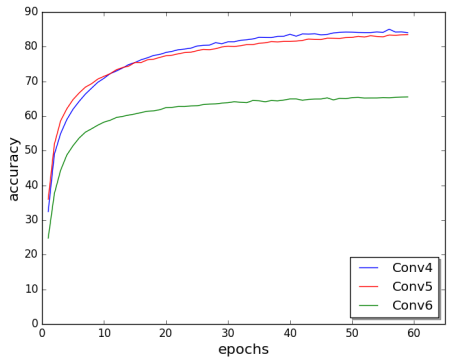


Fig. 6. Accuracy vs. epochs for training three different Convolutional Neural Networks on CIFAR-10 dataset.

when training with a limited computation budget makes it a challenging task. With the application of our proposed predictions, one can know the convergence behavior of a DNN, hence decide whether training shall be continued further. Fig. 5 shows convergence behavior of a convolutional neural network up on CIFAR-10 dataset. The network is trained thrice (represented as Conv1, Conv2 and Conv3), each time initialized with a different set of parameters for 90 epochs. Conv1 is initialized with a learning rate of 0.001 and a batch-size of 1. Conv2 is initialized with a learning rate of 0.001 and a batch-size of 10. Whereas, Conv3 is initialized with a learning rate of 0.05 and a batch-size of 50. We can observe that Cov1 is converges very quickly as it achieves 75% accuracy in less than 10 epochs. Whereas Conv3 never achieves 75% training accuracy within 90 epochs. The curve training accuracy curve corresponding to Conv3 gets saturated as training proceeds. Availability of predictions at hand during the training ensures ceasing the training of Conv3 resulting in a time as well as computational gains.

6) Choosing an architecture: Here we demonstrate the utility of our method to choose a DNN architecture. We have considered three Convolutional Neural Networks (namely Conv4, Conv5 and Conv6) that are different from each other. Each of these networks are alike LeNet-5, with some minor variations. Conv4 has two convolutional layers and two fully connected layers. The convolution filters are of size 5×5. Conv5 also has two convolutional layers and two fully connected layers. However, the convolutional filters are of size 3×3. We use dropout in all the layers. Conv6 has three convolutional layers and two fully connected layers. Here the convolution filters are of size 3×3. We use dropout in all the layers. All the above mentioned networks are initialized with a learning rate of 0.001, batch-size of 1 and trained using Stochastic Gradient Descent (SGD) up on CIFAR-10 dataset. Fig. 6 shows the training curves for above mentioned networks. The green curve corresponding to Conv6 saturates as the training proceeds. While training is at epoch 25, we predicted at which epoch shall Conv6 achieve a training accuracy of 80%. According to the prediction made by a power function, Conv6 is expected to achieve 80% accuracy after 357 epochs, which signifies that Conv6 is not a good architecture. Hence, in similar situations, one can halt training. Such early predictions can avoid unnecessary computation and time invested in training, otherwise which is demanded by a not so good architecture like Conv6.

C. Discussion

The exponential and power functions gives good estimates up on MNIST dataset, when n is around 30 to 50. Refer green and magenta curves in Fig. 3a. Polynomial and piece-wise

polynomial functions gives good estimates, particularly when the data available at hand is less (during trials at initial epochs), refer Fig. 3c and tables in supplementary results. From table II, most of the estimates are an underestimate of the actual values. From Fig. 1, we can observe that the rate of increase in accuracy value diminishes as the training progresses. This makes the task of estimation using regression, a bit challenging. The exponential and power function fails when the data available is less. When n is less than or equal to 25 (i.e., data up to 25 epochs is available), the exponential and power function fails to give a reasonable estimate. But the polynomial function (seventh degree) gives a good estimate. This observation drives the message that, sufficient data till some specified epoch (here data more than 25 epochs) is required to capture the behavior of growth trend in accuracy. Or alternatively n should the value where, the rate of increase in accuracy is greater than some threshold 't'. As the value of n increases, the data follows an exponential growth trend. Even at the first trial(prediction at epoch 3), the predictions are reasonably good. Refer dotted green curve in Fig. 3c (the trial at epoch 3) which traces the ground truth red curve. While learning function at each prediction trail, the observed data is considered in a cumulative fashion. Consider two prediction trails at epochs 20 and 30 respectively. The observed data at epoch 30 is super-set of the observed data at epoch 20. Instead, one can consider observed data and its gradient locally (i.e. observed data for trail at epoch 30 is the data from epochs 21 to 30), and decide the kind of function that can be learnt. Tables showing the ground truth values, i.e the actual epochs taken to reach various accuracies, the performance of various functions and the function that gives best estimate at various stages are presented in supplementary material.

IV. Conclusion and Future Works

In this work, we propose a method to estimate the time required to train a deep neural network to achieve a given accuracy. Our experimental results show that the proposed method is capable of achieving accurate results at early stages of training, on complex DNNs such as AlexNet, trained on a challenging dataset ImageNet. The utility of our method to aid in selecting parameters and network architecture is notable, thus enabling time and computation gain. We also the studied the effectiveness of our proposed method while training various convolutional neural networks. The effect of batch-size and learning rate on our prediction accuracy is to be analyzed. In future, we would like to demonstrate the working of our approach on other deep neural network architectures such as recurrent neural networks etc. and on larger datasets, in a distributed setting.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *NIPS*, 2012. 1

[2] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *ECCV*, 2014. 1

[3] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. Lecun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *ICLR*, 2014. 1

[4] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *NIPS*, 2014. 1

[5] ——, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015. 1

[6] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *AISTATS*, 2010. 1

[7] I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, "On the importance of initialization and momentum in deep learning," *ICML*, 2013. 1, 2

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," *CVPR*, 2009. 1

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *JMLR*, 2014. 1

[10] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015. 1

[11] "Neural networks: Tricks of the trade, reloaded," *Lecture Notes in Computer Science (LNCS)*, vol. 7700, 2012. 1

[12] S. Shalev-Shwartz, Y. Singer, and N. Srebro, "Pegasos: Primal estimated sub-gradient solver for svm," *ICML*, 2007. 1

[13] A. Bordes, L. Bottou, and P. Gallinari, "Careful quasi-newton stochastic gradient descent," *JMLR*, 2009. 1

[14] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning," 2011. 1

[15] R. Raina, A. Madhavan, and A. Y. Ng, "Large-scale deep unsupervised learning using graphics processors," *ICML*, 2009. 1

[16] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," *NIPS*, 2012. 1

[17] A. Coates, B. Huval, T. Wang, D. J. Wu, B. C. Catanzaro, and A. Y. Ng, "Deep learning with COTS HPC systems," *ICML*, 2013. 1

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014. 1

[19] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," *BigLearn, NIPS Workshop*, 2011. 1

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, pp. 2278–2324, 1998. 3, 4

[21] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images." [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features2009-TR.pdf 3

[22] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf 3

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, 2015. 3

[24] https://github.com/torch/demos/blob/master/train-a-digit-classifier/. 4

[25] https://github.com/torch/demos/blob/master/train-on-cifar/train-on-cifar.lua. 4

[26] https://github.com/torch/demos/blob/master/train-on-housenumbers/. 4

[27] https://github.com/soumith/imagenet-multiGPU.torch. 4

[28] A. Krizhevsky, "One weird trick for parallelizing convolutional neural networks," *arXiv preprint arXiv:1404.5997*, 2014. 4

[29] https://github.com/soumith/imagenet-multiGPU.torch. 4