STUDY OF NEURAL NETWORK INTERNALS

Gaurav Kumar Jain

Master of Technology Thesis

December 2016

ज्ञानमुत्तमम्

International Institute of Information Technology, Bangalore

STUDY OF NEURAL NETWORK INTERNALS

Submitted to International Institute of Information Technology,
Bangalore
in Partial Fulfillment of
the Requirements for the Award of
Master of Technology

by

Gaurav Kumar Jain
EMT2013007

International Institute of Information Technology, Bangalore
December 2016

Dedicated to

My parents, My Wife Divya and My sweet daughter Samartha

Thesis Certificate

This is to certify that the thesis titled **Study of Neural network internals** submitted to the International Institute of Information Technology, Bangalore, for the award of the degree of **Master of Technology** is a bona fide record of the research work done by **Gaurav Kumar Jain**, **EMT2013007**, under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. G.Srinivasaraghavan

Bangalore,

The 30th of December, 2016.

STUDY OF NEURAL NETWORK INTERNALS

Abstract

Neural networks generated lot of interest and constantly created new bench marks in recent times which were never imagined. Certain cases it is able to pass human level performance. However it comes with lot of challenges and specifically optimization of these networks poses lot of challenges. In this work we will try to uncover these aspects. Network optimization aspects are studied in great detail in this work and experimentations with many possible configurations are performed on datasets such as MNIST, CIFAR10, CIFAR 100. Experimental results are analysed and on their basis recommendations are suggested which could help in better convergence performance and able to generate robust classifiers.

Acknowledgements

Put your acknowledgements here...

"Intellectual and practical assistance, advice, encouragement and sources of monetary support should be acknowledged. It is appropriate to acknowledge the prior publication of any material included in the thesis either in this section or in the introductory chapter of the thesis."

— MUN School of Graduate Studies

List of Publications

[1] John Doe John Doe, and Some Guy. Journal article SWGC title. Journal of Sample Journals, 1(12):1000–1024, 2002.

Contents

List of Figures

List of Tables

List of Abbreviations

DNN .... Deep Neural Network

X_r .... Training Data

IIITB .... International Institute of Information Technology Bangalore

CHAPTER 1

INTRODUCTION AND OVERVIEW

Deep neural networks have demonstrated excellent results in many machine learning tasks [REFERENCES:TBD] and became a default choice for machine learning researchers irrespective of the end result they achieved. Computer vision , Natural language and Speech processing tasks have scaled to the next level of accuracies using these networks.

Structure: The core structure of Neural network is the Neurons arranged in a layer manner also known as hidden layer and stack of these layers with interconnected Neurons provides depth to the network. This arrangement of layers is known as network architecture. A simple Neural Network is shown in Figure FC1.1. Several networks are proposed till date for the range of machine learning tasks.[REFERENCES:TBD]

Training: Network architecture remains passive till it is trained and parameters of the networks are estimated for the desired level of prediction performance on data samples outside training data set, also known as Generalization performance. The most important part is the training strategy which encompass choosing the hyper-parameters to intermediate updates along with learning algorithm. The number of hyper-parameters termed as annoying knobs to be adjusted [Bengio 2012: Practical recommendations] and famously known as nuisance parameters are quite high. On top of this range of

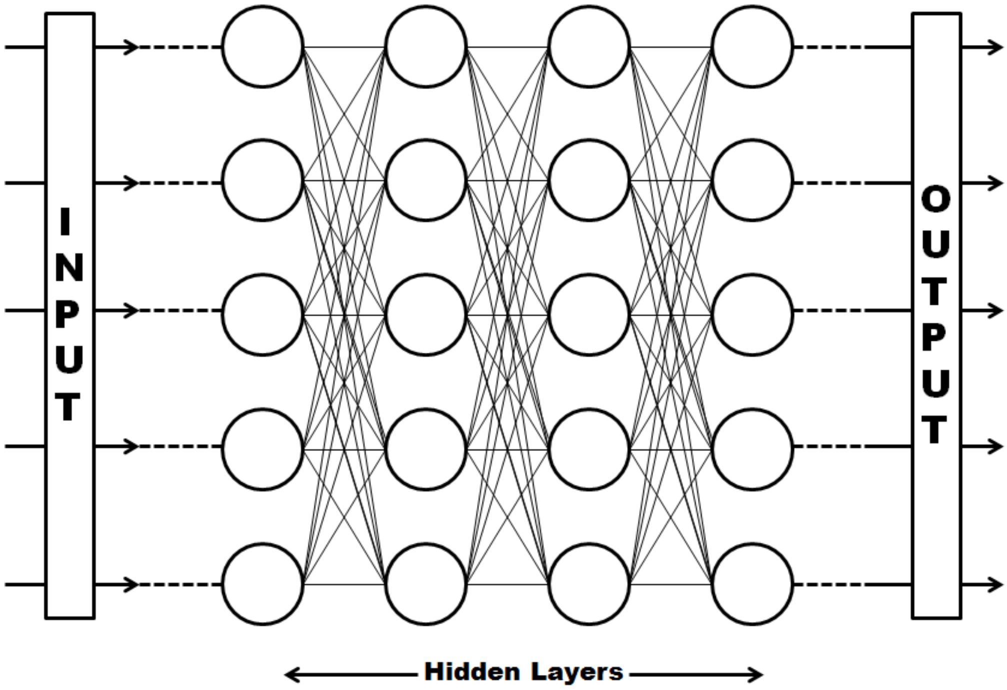


Figure FC1.1: A Simple Neural Network

choices for these hyper-parameters make exhaustive search impractical. Once these parameters are chosen, training can be proceed and the algorithm, which is more often than not is Stochastic gradient descent [References] along with Back propagation [Reference: Rumelhart] as a default choice allows network to evolve from untrained to train network.Training stops as per stopping criterion.

Considering all these background the **Training Life Cycle** of DNN's has 3 main stages as shown in Figure FC1.2

1. Choosing network structure
2. Selection of hyper parameters and training algorithm
3. Network updates during training and stopping criterion

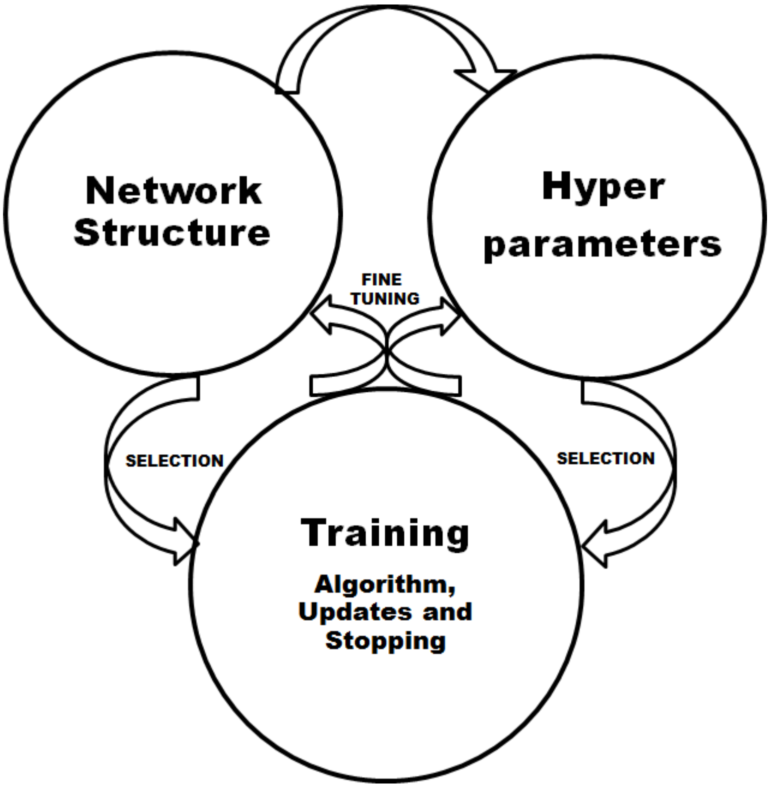


Figure FC1.2: DNN Training Lifecycle

1.1 Chapters Organization

This study is divided into 6 chapters, Chapter 2, discusses **Network architecture**, Chapter 3 discusses **Hyper-parameters**, Chapter 4 discusses **Training the Network**, Chapter 5 discusses the **Insights and Recommendations** from this study. Chapter 6 **concludes** and poses some **open questions** for future work.

1.1.1 Network types

Chapter 2 details different architectures, which can be seen as small survey on the state of the art networks used in deep learning.We will keep them for study purposes.

As we studied the details and interdependencies among the training parameters,so we have used our own network, we call it **GsNet**. We recommend to use them for comparative study of this kind. Table TC1.1 shows the layers configuration of GsNet-2,GsNet-3 and GsNet-5.

Table TC1.1: GsNet architectures

GsNet-2	**GsNet-3**	**GsNet-5**
conv 3x3x64 pool 2x2	conv 3x3x64 pool 2x2	conv 3x3x64 pool 2x2
conv 3x3x64 pool 2x2	conv 3x3x64 pool 2x2	conv 3x3x64 pool 2x2
	conv 3x3x64 pool 2x2	conv 3x3x64 pool 2x2
		conv 3x3x64 pool 2x2
		conv 3x3x64 pool 2x2
dense,128	dense,128	dense,128
softmax,c	softmax,c	softmax,c

We have used these networks for our experiments and their analysis. This may bring how depth affects learning. Different architectures in practice are described, which can be seen as small survey on state of the art network architectures in chapter 2.

1.1.2 Hyper-parameters

Hyper-parameters are discussed in chapter 3.Main parameters, which we studied are as following

1. Batch Size
2. Optimizations
3. Initializations

Firstly we describe all the different prescribed available techniques for these parameters which can be seen as a small survey of the available studies, experimental results and techniques.

Secondly we present results of almost exhaustive set of parameters configuration. Then best of parameters and configurations are chosen for the next set of experiments.

1.1.3 Training the network

Chapter 4 discusses training the network and study which describes different techniques used in training.We will also explain our training set up which is used for our experiments.

1.1.4 Insights and Recommendations

Chapter 5 provides all insights and analysis of our results. This includes well performing strategies as well as strategies which may didn't perform well. Based on these we will describe our recommendations. Also we will explain novel technique which perform well and provide more stability to the learning system.

1.1.5 Conclusion

Chapter 6 concludes with the summary of our study and future direction of this work.

1.2 Notations used

This section explains the notations used through out this study.

1.2.1 DNN Setting

DNN goal is to approximate a target function g^* for the unknown distribution input $X^* = (x_1, x_2, ...x_d)^T \in \mathbb{R}^d$. The target value is $Y^* = (y_1, y_2, ...y_s) \in \mathbb{R}^s$. if θ^* is the parameters associated, then

$$Y^* = g^*(X^*, \theta^*) \qquad \text{(Eqn 1.1)}$$

X^* represents entire input data for the underlying input distribution. Getting X^* is almost impossible, so generally g^* is approximate using the representative input X of size N which is sampled from X^* and hoped to have same distribution as the original input distribution. Let Y is the target value for X.

So DNN problem reduces to approximating g^* using (X, Y) of size N. $\theta = (\theta_1, \theta_2 ... \theta_m) \in \mathbb{R}^m$ represents the parameters which gives best approximation for target function. Finally the DNN has to learn the best θ such that $g(X, \theta) \sim g^*$.

g represents a chained function in context of DNN as it flows from input to output via hidden layers as shown in FC1.1. g as chained function flowing via hidden layers can be written as

$$g(X) = g^K(g^{K-1}(g^{K-2}......(g^2(g^1(X))).......)) \quad \text{(Eqn 1.2)}$$

where K represents total number of hidden layers and $\{g^k, k = 1....K\}$ is output of k^{th} layer.Let $\hat{Y} = g(X)$ then lets define a loss function $\mathscr{L}(\hat{Y}, Y)$ as the cost it incurs using $g(X)$ to approximate $g^*(X)$ and hence it is also known as cost function.

The gradient of $\mathscr{L}$ with respect to θ is denoted as $\nabla_{\theta_t}\mathscr{L}(\theta_t)$ at t^{th} iteration. For simplicity we denote this as ∇_{θ_t}, where θ_t denotes the network parameters at iteration/time t.

Network Parameters

1.3 Experimental set up

We have performed exhaustive set of experiment using standard datasets on Nvidia-Tesla K80 GPU.Regularly results are analyzed to reduce the experiment space and become basis for the next set of experiments.

1.3.1 Databases

Following is the list of datasets used in the experiments:

1. MNIST [1]
2. CIFAR-10 [2]
3. CIFAR 100 [2]

1.3.2 MNIST

MNIST dataset has 60000 training samples and 10000 testing samples.It is database of handwritten digits. Sample examples are shown in fig.??

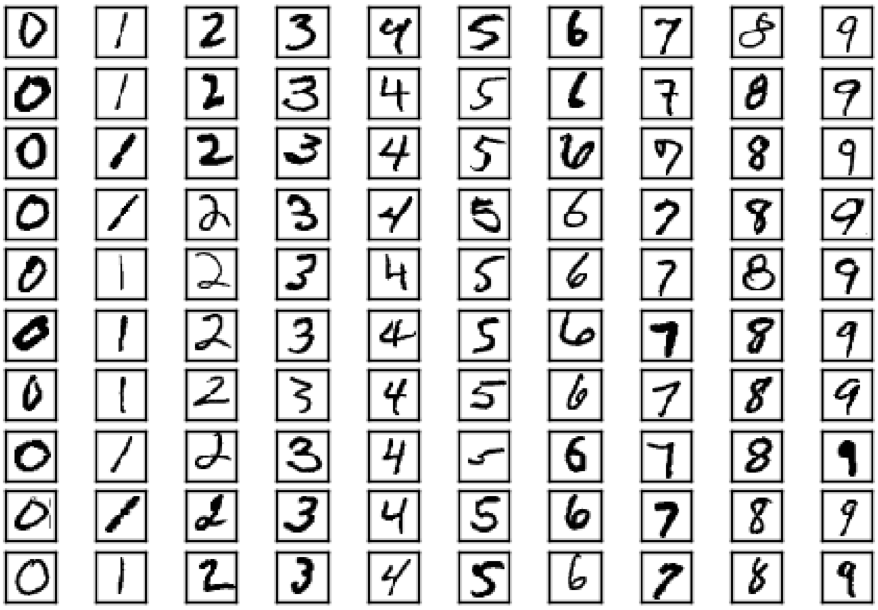


Figure FC1.3: MNIST dataset input images example

MNIST sample images having handwritten digits from 0-9, image size is 28x28 and images are grayscale, shown samples are randomly chosen, 10 for each class.

1.3.3 CIFAR10

MNIST dataset has50000 training samples and 10000 testing samples.It is database of different objects present in the images. Sample examples are shown in fig.??

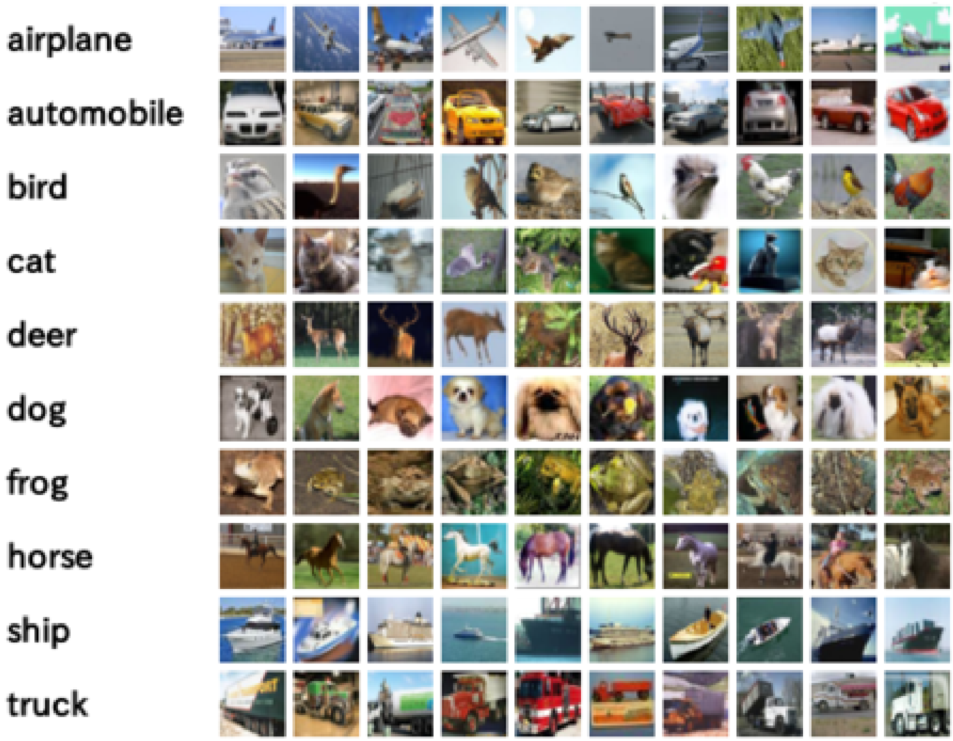


Figure FC1.4: CIFAR-10 dataset input images example

CIFAR-10 sample images having different objects present in the images, image size is 32x32 and images are color

1.3.4 CIFAR100

CIFAR100 dataset has 50000 training samples and 10000 testing samples.It is database of 100 different object classes. Sample examples are shown in fig.**??**

Figure FC1.5: CIFAR-100 dataset input images example

CIFAR-100 sample images consist of database with 100 different object classes, image size is 32x32 and images are color

1.3.5 Software

Keras [3] is mainly used for almost all the experiments. Theano [4] is used as main backend for Keras.Python is used as main programming language.

CHAPTER 2

NETWORK STRUCTURE

This chapter discusses different type of networks in practice, their usage and prominent network architectures. We will discuss only few networks which have achieved significant success in recent times. Primarily we will discuss CNN(Convolutional Neural network) as that is the main network used through out this work.

2.1 Networks

The introduction to Neural network has started long ago with Frank Rosenblatt, famous MLP(Multi Layer Perceptron) [5].Working of neural nets today are precisely captured by Rosenblatt. It talks about pathways to connect to output so that for particular input associated pathway gets activated and produces corresponding output. Following that several networks are suggested directed for specific tasks. For example for image to extract neighborhood relationship, convolution neural networks are suggested. For time series data Recurrent neural networks are suggested. LSTM is recent state of the art for handling time series tasks.

Auto Encoders are suggested as unsupervised nets, which generates low level representation of inputs using only input data. Restricted boltzmann machines are another type of network which focuses on convergence by lowering the energy. In Hopfield network every neuron connects to every other neuron in the network.

Other types of networks are Radial Basis Network(RBN), Gated Recurrent Unit(GRU), Deep belief network(DBN), Generative adversial network(GAN).

2.1.1 Perceptron

The perceptron has been introduced to handle perceptual recognition, generalization and hence the name Perceptron. In this landmark paper Rosenblatt nicely maps the neurons development in human being to perceptron. For instance connection of nervous system are assumed as random and in neural nets at start mostly random initializations are used. The original system has said to be capable of plasticity, which allows other neuron output to change over time seeing stimulus applied. And if same or similar stimuli is seen large number of times, will tend to form pathways to same sets of responding cells. This almost sums up the current neural nets, however the network construction, random initializations may differ a lot.

2.1.2 RNN

Recurrent neural nets are having connections to same hidden layer neurons, and thus capable of storing time series data or feedback to be used with next set of input in sequence.

CHAPTER 3

HYPER-PARAMETERS

This chapter introduces hyper-parameters. They are not directly related to the machine learning algorithm parameters, but responsible for overall algorithm evolution. For example learning rate dictates the update strength per iteration, choice of initializer can lead to slow or fast convergence, choice of optimizers provide way to update learning parameters(θ).

Other than direct algorithm parameters, we will consider everything else as hyper-parameter. Mainly we will consider following hyper-parameters, which we will discuss and explain the experiments performed with different choices of these parameters and their effect in overall convergence.

1. **Initializer**
2. **Optimizer**
3. **Batch size**
4. **Total parameters**
5. **Number of Epochs**

3.1 Initializer

Initial network condition is described as *"At birth the construction of the most important networks is largely random, subject to a minimum number of constraints"* in [5] suggests that random initializations can be used to initialize network parameters and constraints could be range of these parameters.

Uniform initialization, assigns initial weights from $U[-r,\ r]$, where U is uniform distribution, Mostly r is used as small value ~ 0.1. Another random initialization scheme is **Normal Initialization**, based on sampling initial weights from normal distribution, $N(0,1)$. These are simple methods which are getting used.

Uniform initialization using fan in [6] suggests, using fan-in to determine standard deviation σ_i and sampling initial weights from $N(0,\sigma_l)$. value of σ_l is dependent on fan-in, which is number of inputs to a hidden unit. It is suggested to be chosen as per Eqn 3.1

$$\sigma_l = \mathbf{m}^{-1/2}, where,\ \mathbf{m}\ is\ the\ fan\ in \qquad \text{(Eqn 3.1)}$$

Normalized initialization [7] suggest to use uniform weight initialization as per Eqn 3.2

$$U[-\frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}},\frac{\sqrt{6}}{\sqrt{n_l+n_{l+1}}}] \qquad \text{(Eqn 3.2)}$$

where n_l and n_{l+1} are total number of units in l^{th} and $(l+1)^{th}$ hidden layer respectively. This work on paradigm of maintaining activation variances in feed forward direction and back propagated gradient variances in both the directions.

For very deep models and to support activation ReLU/PReLU [8] suggests to use slight modification in considering the variance from [7] and the **initialization scheme becomes zero-mean Gaussian with standard deviation as** $\sqrt{2/n_l}$, where n_l is number of hidden units in l^{th} layer.

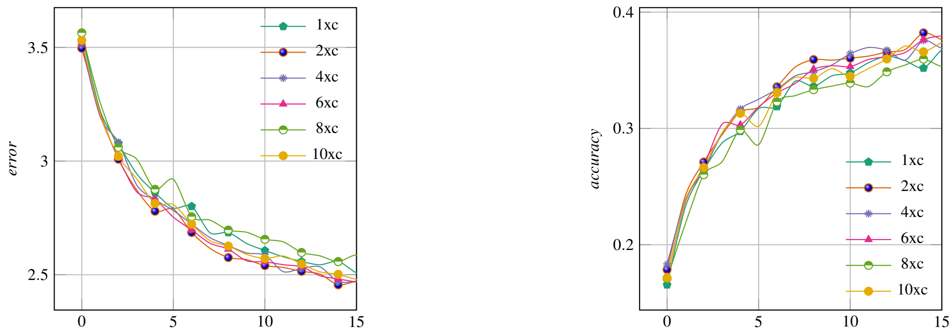
Orthogonal random initializations [9] suggests simple initialization scheme where in initial weights are chosen from the random orthogonal matrix satisfying $W^TW = I$. This yield depth independent learning times, which means as depth increases learning time remains same as oppose to suggested initializations in [6] [7]

3.1.1 Experiments

In this section we analyze the effect of different initializers based on our experiment results.

3.1.1.1 CIFAR100

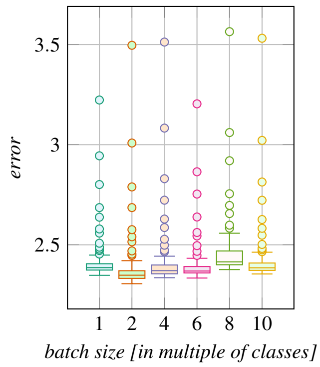
Two Layer, opti=adagrad, batch size=1xc



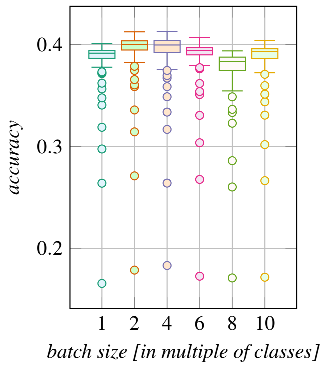
(a) Error for initial epochs:INIT (b) Accuracy for initial epcohs

Figure FC3.1: *Different batch results for starting 15 epochs*

to write



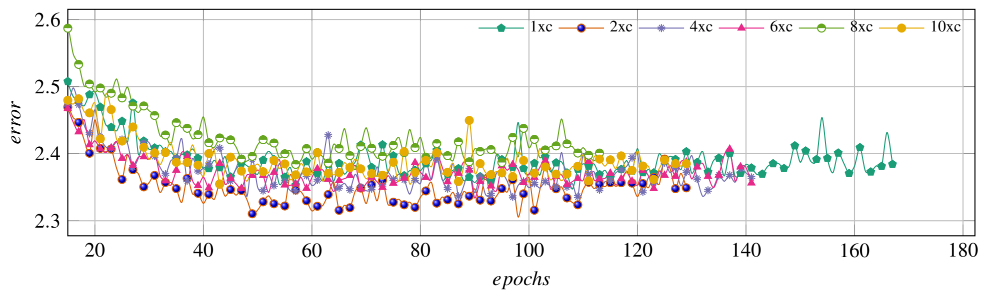
(a) Error variation



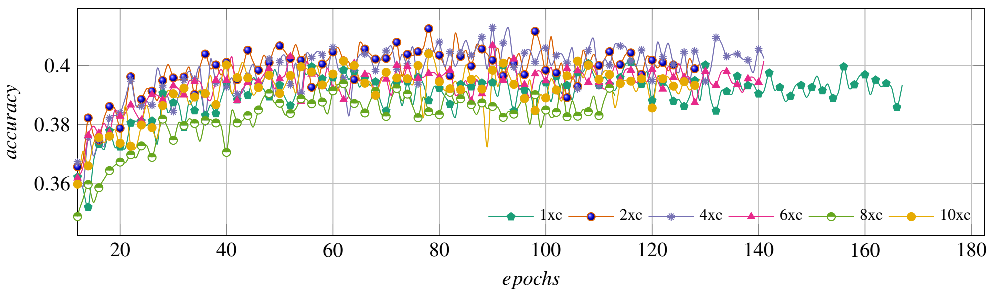
(b) Accuracy variation

Figure FC3.2: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

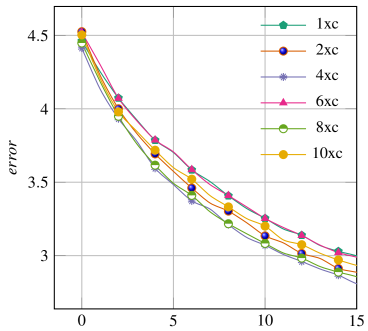


(b) Accuracy for later epochs

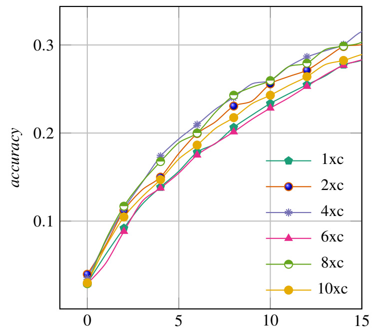
Figure FC3.3: *Different batch results for later epochs*

to write

Two Layer, opti=SGD with nesterov momentum, batch size=1xc



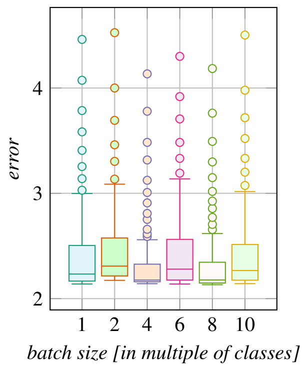
(a) Error for initial epochs:INIT



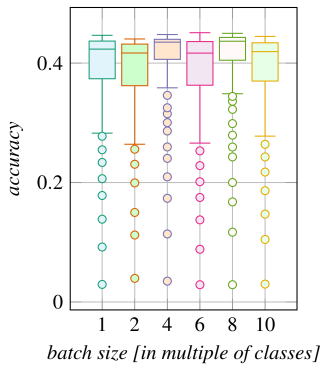
(b) Accuracy for initial epcohs

Figure FC3.4: *Different batch results for starting 15 epochs*

to write



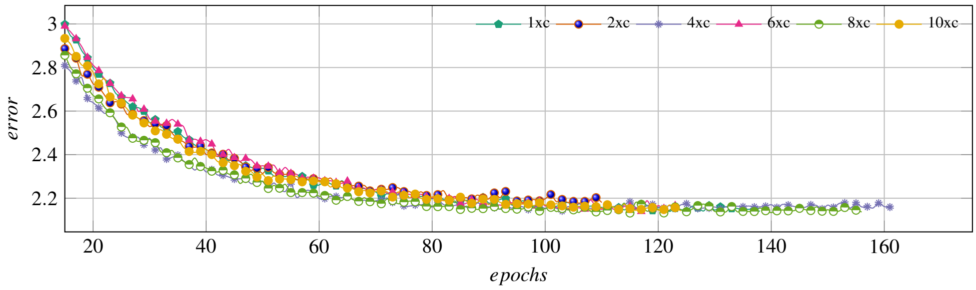
(a) Error variation



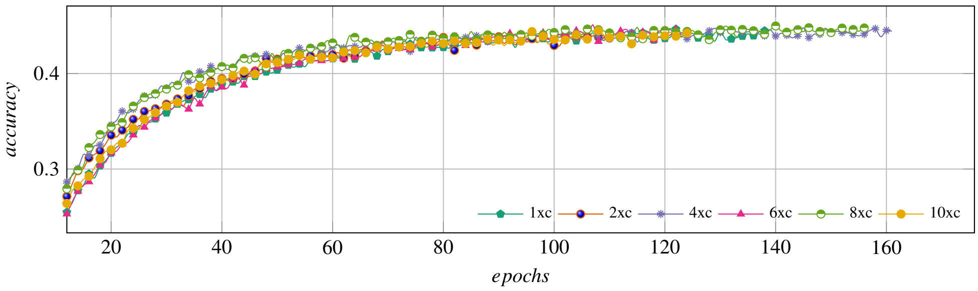
(b) Accuracy variation

Figure FC3.5: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs



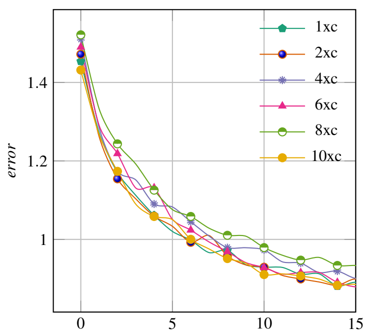
(b) Accuracy for later epochs

Figure FC3.6: *Different batch results for later epochs*

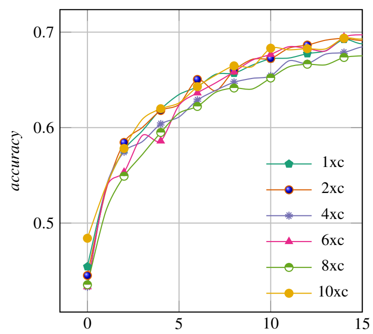
to write

3.1.1.2 CIFAR-10

Five Layer, opti=adagrad, batch size=2xc



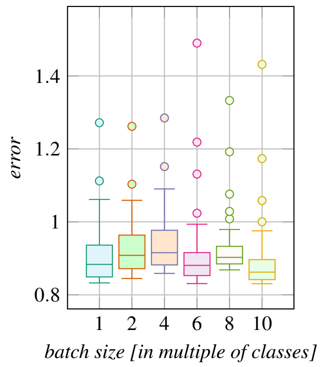
(a) Error for initiaNITCIFAR10



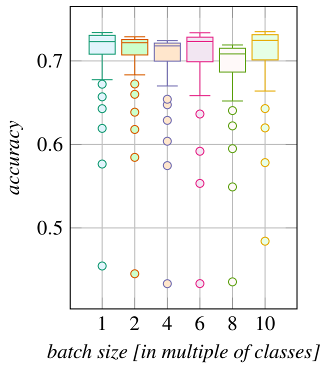
(b) Accuracy for initial epcohs

Figure FC3.7: *Different batch results for starting 15 epochs*

to write



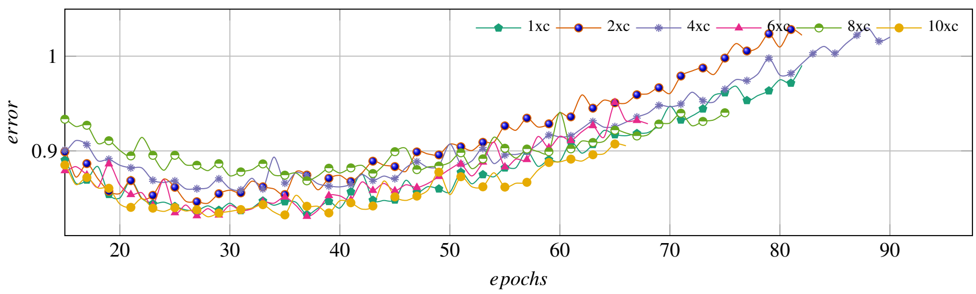
(a) Error variation



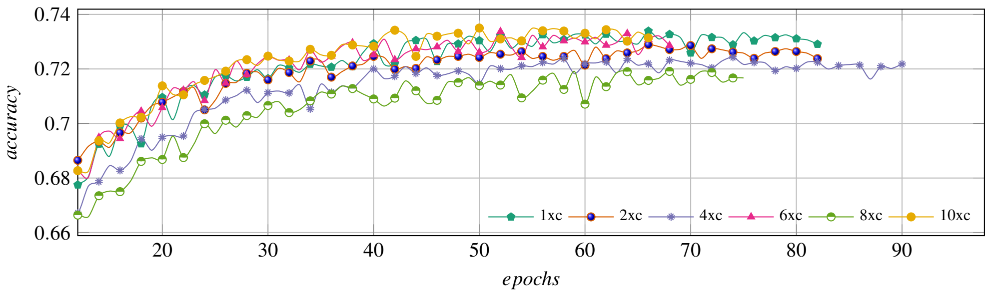
(b) Accuracy variation

Figure FC3.8: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs



(b) Accuracy for later epochs

Figure FC3.9: *Different batch results for later epochs*

to write

3.1.1.3 MNIST

Two Layer, opti=nadam, batch size=1xc

3.2 Optimizers

Optimizers are main learning algorithm or routine which is responsible for changes in the network as and when update takes place. The updates are not limited to parameters update alone, even hyperparameters can also get updated as per underlying optimizer routines.

Here we will discuss different optimizers, mainly gradient/sub gradient methods,their analysis and effect from experimental results. Detailed survey of these methods can be found at [10] and [11]

GD(Gradient descent) is one of the important and robust optimization algorithm. The gradient of the function to be optimized is computed with respect to the parameters. In deep learning setting the function to be optimized usually is loss function $\mathscr{L}$ and parameters are θ_t at t^{th} iteration. The gradient of $\mathscr{L}$ with respect to θ is denoted as $\nabla_{\theta_{t-1}}\mathscr{L}(\theta_{t-1})$, which gets computed at $(t-1)$ iteration and θ_t is updated as per Eqn 3.3

$$\theta_t = \theta_{t-1} - \eta(\nabla_{\theta_{t-1}}\mathscr{L}(\theta_{t-1})), \textit{where } \eta \textit{ is known as learning rate} \qquad \text{(Eqn 3.3)}$$

In deep learning due to large training size gradient descent, also known as **batch learning** is almost impossible. So for large scale learning reduced size is considered and network parameters are updated, this batch will not be used till new repetition of dataset starts. This repetition is known as an **epoch**. The size of batch used in one single update is known as **minibatch**. The minibatch and epochs are considered later in this chapter. We will see batch learning in chapter 4.

The strategy to update network parameters after every $minibatch\ size = 1$ is known as **SGD (Stochastic Gradient Descent)**, which is also known as online learning. Commonly $minibatch\ size \gg 1$ is used in deep learning with large datasets. We will use SGD for minibatch or online learning.

Eqn 3.3 is applicable as it is to SGD also, only difference is computation of loss gradient is restricted to the current minibatch instead of full training dataset. We will discuss its details in chapter 4. Another optimizers explained ahead are variants of SGD, which is essentially differs in the way network parameters get adjusted as learning progresses.

Classical Momentum [12] remembers previous gradient update vector and fraction of it is added to the next parameter update. The momentum term is computed as per Eqn 3.4 and update takes place as per Eqn 3.5.

$$m_t = \mu m_{t-1} + \nabla_{\theta_{t-1}} \mathscr{L}(\theta_{t-1}), \quad \text{(Eqn 3.4)}$$

$$\theta_t = \theta_{t-1} - \eta m_t \quad \text{(Eqn 3.5)}$$

where μ is known as momentum term

NAG (Nesterov accelerated gradient) is accelerated gradient descent which converges faster than classical momentum or SGD. The gradient is calculated on possible future update without using gradient and then using it to update network parameter. Acceleration is achieved as it can be seen as looking into future as this can prevent slows gradient update to move uphill and accelerate if it is moving downhill. The updates are calculated as per Eqn 3.6

$$\begin{aligned} m_t &= \mu m_{t-1} + \eta \nabla_{\theta_{t-1}} \mathscr{L}(\theta_{t-1} - \mu m_{t-1}) \\ \theta_t &= \theta_{t-1} - m_t \end{aligned} \quad \text{(Eqn 3.6)}$$

Adagrad (Adaptive subgradient descent) [13] adapts the learning rate as per the parameters updates. So this optimizer adjusts learning rate hyper parameter and falls under category where it updates parameters as well as hyper parameters. Basic premise of Adagrad is to have larger updates for less frequent parameters and smaller updates for frequent ones.

The updated learning rate for each parameter thus varies based on their earlier gradient updates individually. The updates take place as per Eqn 3.7

$$n_t = n_{t-1} + (\nabla_{\theta_{t-1}} \mathscr{L}(\theta_{t-1}))^2$$
$$\theta_t = \theta_{t-1} - \eta \frac{\nabla_{\theta_{t-1}} \mathscr{L}(\theta_{t-1})}{\sqrt{n_t + \varepsilon}} \quad \text{(Eqn 3.7)}$$

Adadelta [14] counters the effect of increasing norm n_t of Adagrad as that can reduce learning rate monotonically, which can be easily seen in Eqn 3.7, where n_t increases as iteration progresses. Adadelta restricts the size of gradients which get accumulated to a sliding window of fixed size. The sum of gradients are maintained as running average $E[g^2]_t$ of previous gradient average squared $E[g^2]_{t-1}$ and current gradient g_t squared as per Eqn 3.8

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho){g_t}^2, where\ \rho\ is\ a\ decay\ constant \quad \text{(Eqn 3.8)}$$

Other than maintaining the running average of gradients sqaured, running average $E[\nabla\theta^2]$ of previous parameter updates squared are also maintained and this is compute in similar way of running average gradient computation of Eqn 3.8. It is given in Eqn 3.9

$$\nabla\theta_t = -g_t \frac{\sqrt{E[\theta^2]_{t-1} + \varepsilon}}{\sqrt{E[g^2]_t + \varepsilon}}$$
$$E[\nabla\theta^2]_t = \rho E[\nabla\theta^2]_{t-1} + (1-\rho){\nabla\theta_t}^2, \quad \text{(Eqn 3.9)}$$

Now updates of Adadelta takes place as per Eqn 3.10

$$\theta_t = \theta_{t-1} - \nabla\theta_t \quad \text{(Eqn 3.10)}$$

if instead of using running average of parameters update, we use η in Eqn 3.9 to calculate $\nabla\theta_t$ given in Eqn 3.11 and update happens as per Eqn 3.10, then this optimizer is known as **RMSprop**.value of ρ as proposed by the author is 0.95.

$$\nabla\theta_t = -g_t \frac{\eta}{\sqrt{E[g^2]_t + \varepsilon}} \qquad \text{(Eqn 3.11)}$$

Adam(Adaptive moment estimation) [15] combines momentum and norm based optimizer. It computes first and second moment estimate as per Eqn 3.12. $\hat{m}_t$ and $\hat{v}_t$ are known as first and second moment estimates respectively,

$$\begin{aligned}\hat{m}_t &= \frac{\beta_1 m_{t-1} + (1-\beta_1)g_t}{(1-\beta_1{}^t)} \\ \hat{v}_t &= \frac{\beta_2 v_{t-1} + (1-\beta_2)g_t{}^2}{(1-\beta_2{}^t)}\end{aligned} \qquad \text{(Eqn 3.12)}$$

Adam updates then takes place as per Eqn 3.13

$$\theta_t = \theta_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}} \qquad \text{(Eqn 3.13)}$$

Adamax [15] uses same updates as Adam other than it uses l_∞ norm instead of l_2 norm.

3.2.1 Experiments

In this section we analyze the effect of different optimizers based on our experiment results.

3.3 Number of Epochs

Number of epochs is the parameter which governs how many time full dataset will undergo training progress. As in minibatch learning batch size $\ll$ full training data. To see full data one time several minibatch undergoes update but not used later till full training data is used. One epoch thus sees full data set once and then it starts the process again.

Number of epochs are required to average out noise which is introduced due to stochastic minibatch updates. We have used this parameters in conjunction with Early stopping, which means set epoch value to a large number and use stopping criterion automatically based on certain performance parameter. In ur experiments we have used validation accuracy as performance parameter to monitor for maximum patience of 50 epochs, which means if there is no improvement from last 50 epochs on validation accuracy, training halts automatically.

3.4 Batch Size

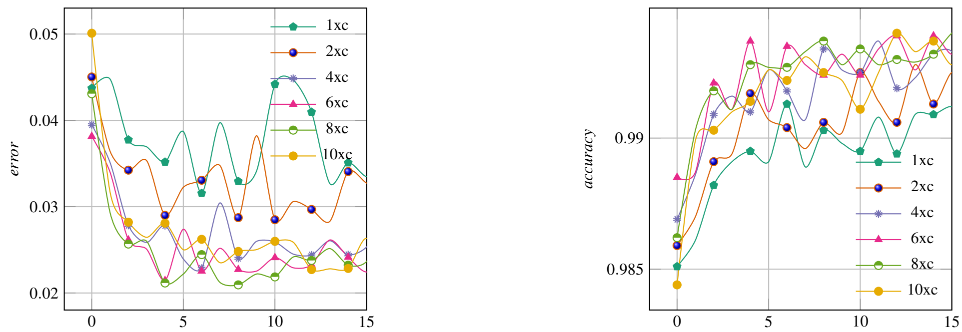
Batch size or minibatch size is the total number of samples chosen from the dataset, which are part of single network update. Often this parameter is chosen arbitarily based on memory availability o the system or capability o underlying mechanism. There is not much study available on the batch size recommendations.As a general rule [16] suggests to use batch size=32, as values above 10 can take advantage of fast matrix multiplications over vector matrix products. [17] in context of RBM(Restricted Boltzmann Machines) also suggest to use batch size greater than 10 for speed up, but strongly against making the size too big when using stochastic gradient descent.

We can understand this bit more in context of network updates where each mini batch is responsible for a single update, so total number of updates in an epoch depends on the size of mini batch. Bigger the size less number of updates it has in an epoch. So weight updates decreases as batch size increases. [17] suggest to use batch size equal to number of classes in case of uniform class data with small number of classes. Also it suggest to have sample from each class in a batch. However there is no experimental evidences provided which supports the suggestions for different networks. Also study of mini batch size with respect to classes seems interesting. So our major work here is experiments the relationship of this kind which is largely neglected and provide a recommendations for their choice.

3.4.1 Experiments

3.4.1.1 MNIST

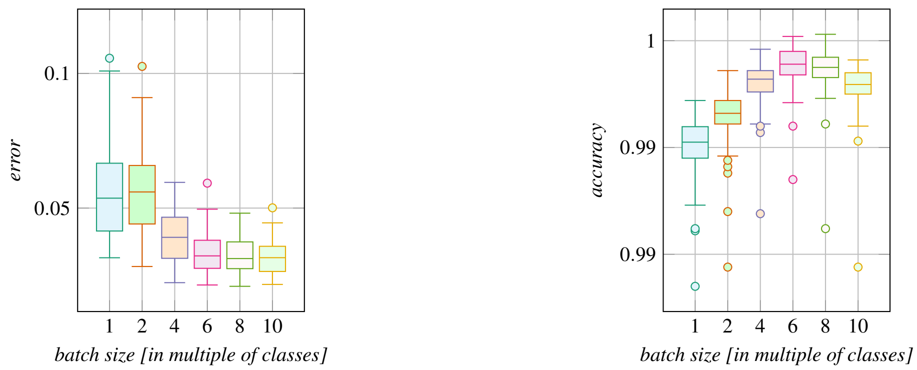
Two Layer, opti=nadam, init=hessian uniform This section provides comprehensive study on the choice of mini batch size, its relationship with number of classes in classification task.



(a) Error for initiaNITCIFAR10 (b) Accuracy for initial epcohs

Figure FC3.10: *Different batch results for starting 15 epochs*

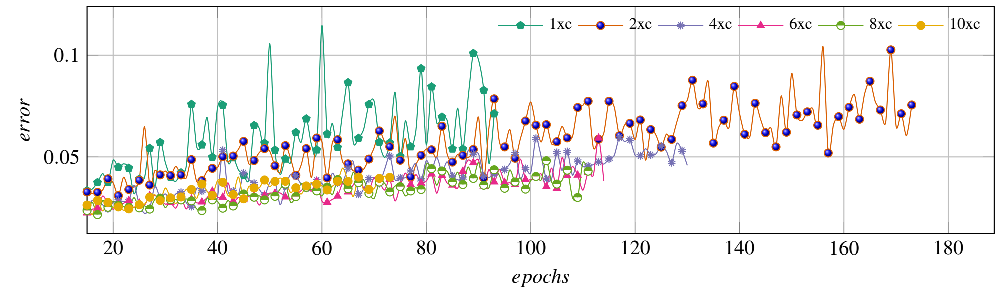
to write



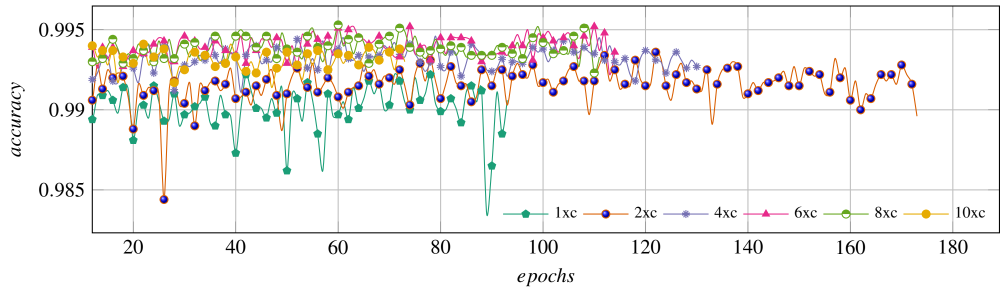
(a) Error variation (b) Accuracy variation

Figure FC3.11: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

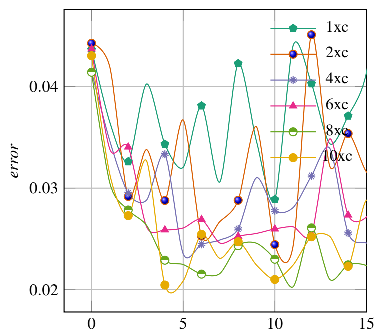


(b) Accuracy for later epochs

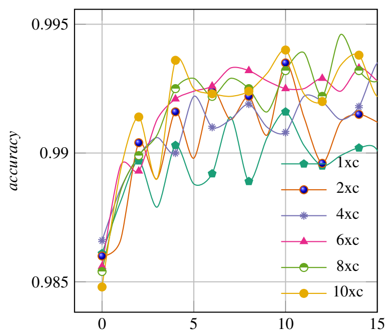
Figure FC3.12: *Different batch results for later epochs*

to write

Two Layer, opti=nadam, init=glorot uniform



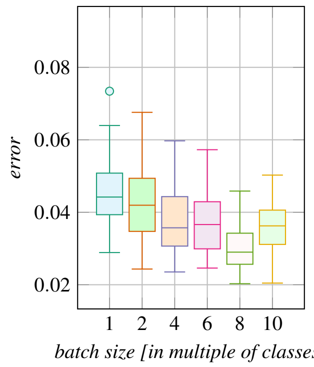
(a) Error for initiaNITCIFAR10



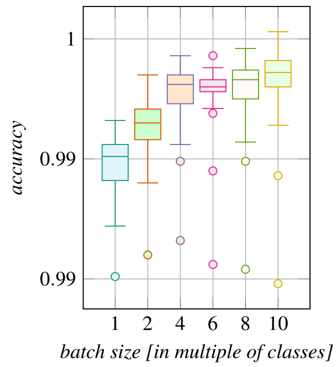
(b) Accuracy for initial epcohs

Figure FC3.13: *Different batch results for starting 15 epochs*

to write



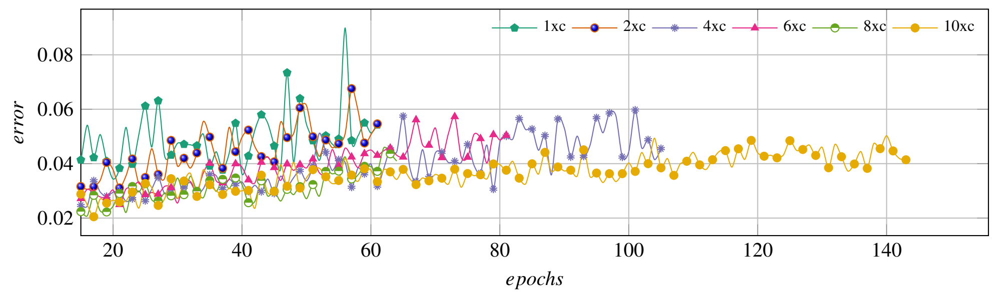
(a) Error variation



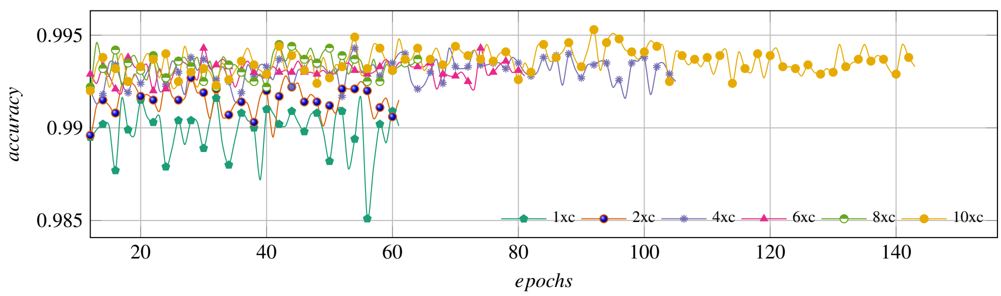
(b) Accuracy variation

Figure FC3.14: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

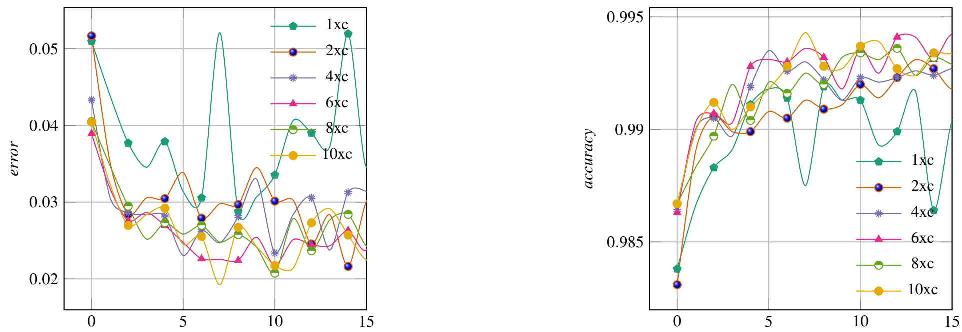


(b) Accuracy for later epochs

Figure FC3.15: *Different batch results for later epochs*

to write

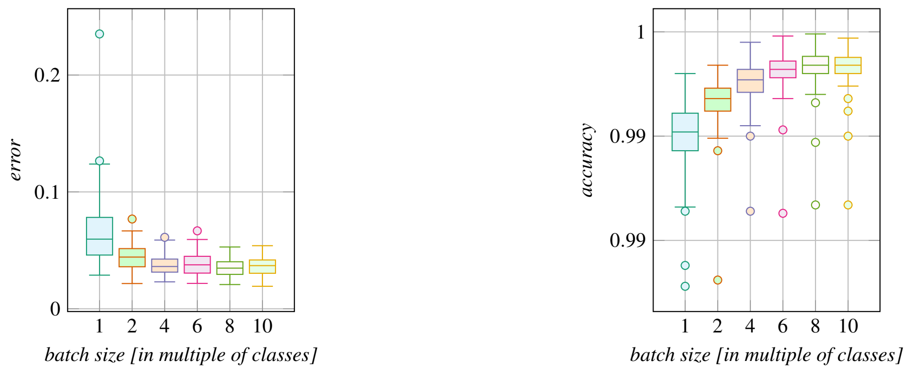
Two Layer, opti=nadam, init=uniform



(a) Error for initiaNITCIFAR10 (b) Accuracy for initial epcohs

Figure FC3.16: *Different batch results for starting 15 epochs*

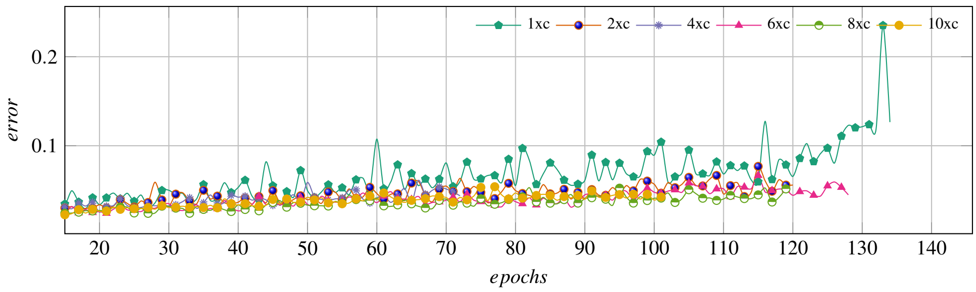
to write



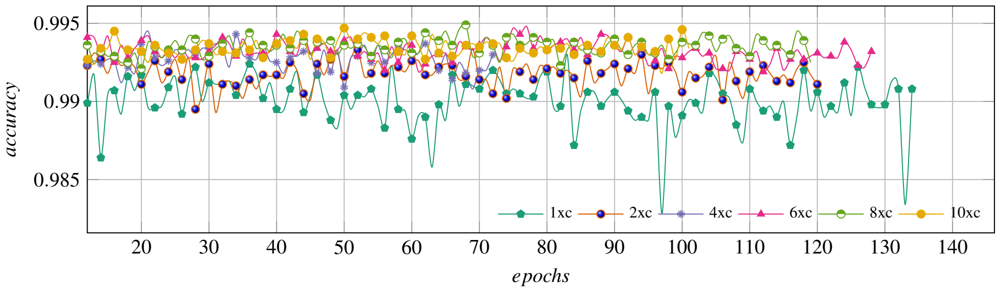
(a) Error variation (b) Accuracy variation

Figure FC3.17: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

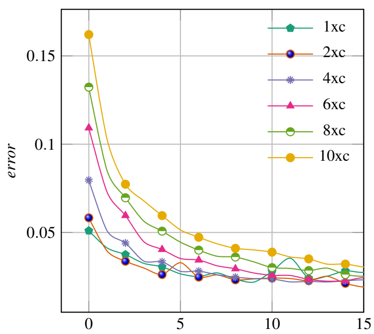


(b) Accuracy for later epochs

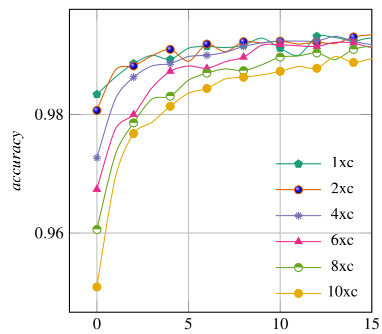
Figure FC3.18: *Different batch results for later epochs*

to write

Two Layer, opti=sgd with momentum, init=hessian uniform



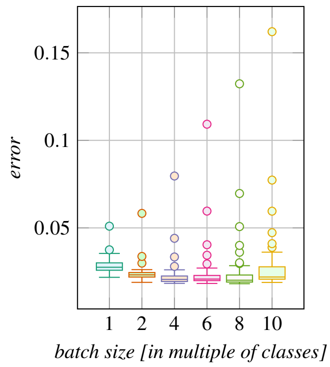
(a) Error for initiaNITCIFAR10



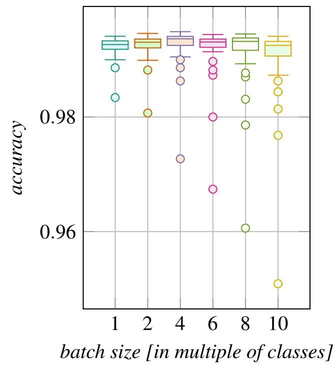
(b) Accuracy for initial epcohs

Figure FC3.19: *Different batch results for starting 15 epochs*

to write



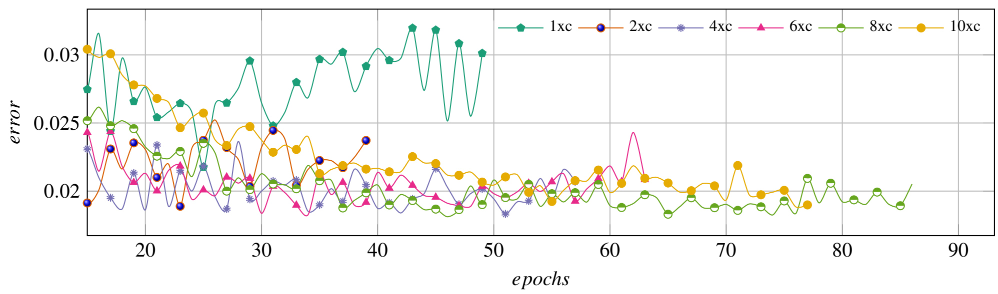
(a) Error variation



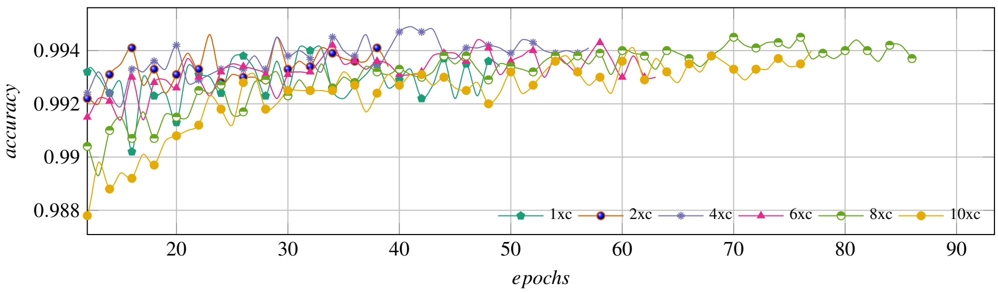
(b) Accuracy variation

Figure FC3.20: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

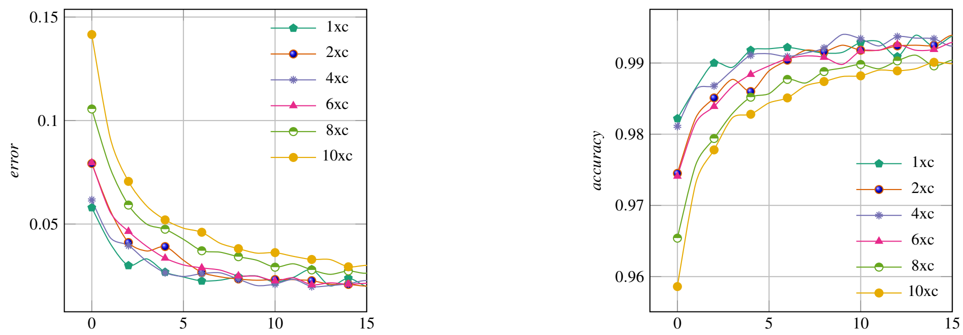


(b) Accuracy for later epochs

Figure FC3.21: *Different batch results for later epochs*

to write

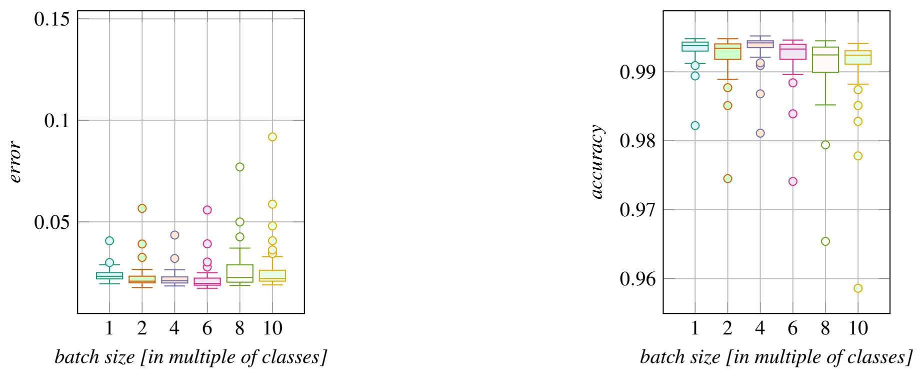
Two Layer, opti=sgd with momentum, init=uniform



(a) Error for initiaNITCIFAR10 (b) Accuracy for initial epcohs

Figure FC3.22: *Different batch results for starting 15 epochs*

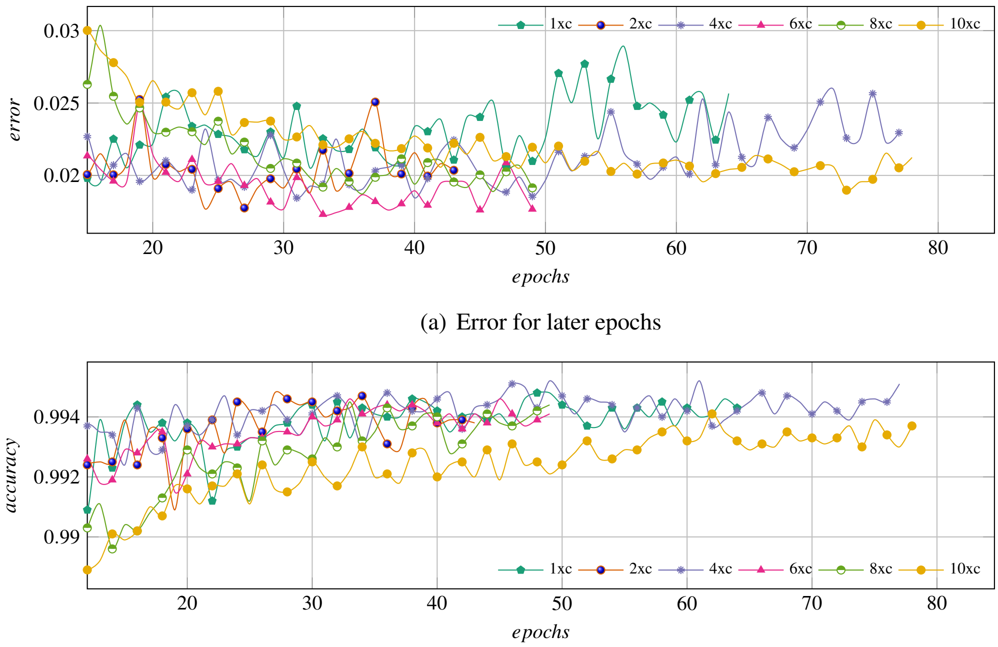
to write



(a) Error variation (b) Accuracy variation

Figure FC3.23: *accuracy and error plot for full training epochs*

to write



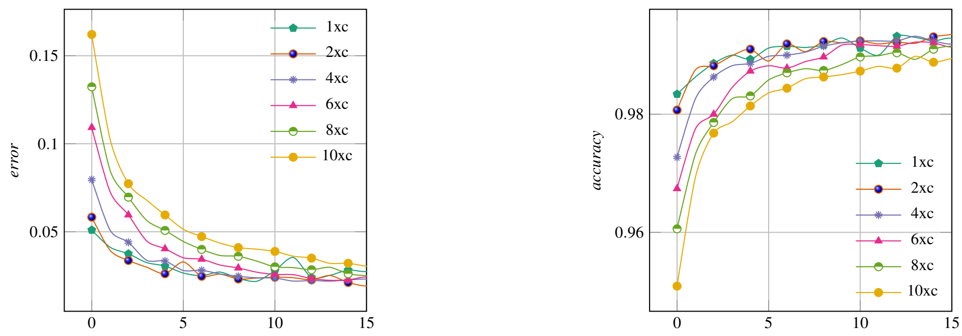
(a) Error for later epochs

(b) Accuracy for later epochs

Figure FC3.24: *Different batch results for later epochs*

to write

Three Layer, opti=sgd with momentum, init=uniform

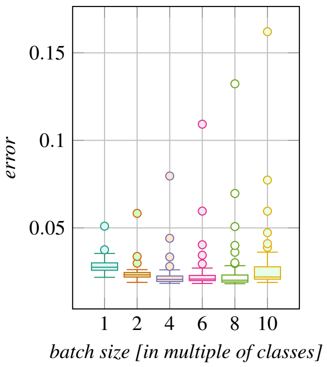


(a) Error for initiaNITCIFAR10

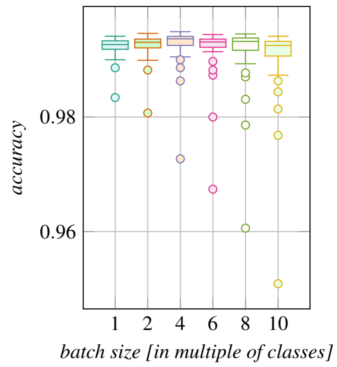
(b) Accuracy for initial epcohs

Figure FC3.25: *Different batch results for starting 15 epochs*

to write



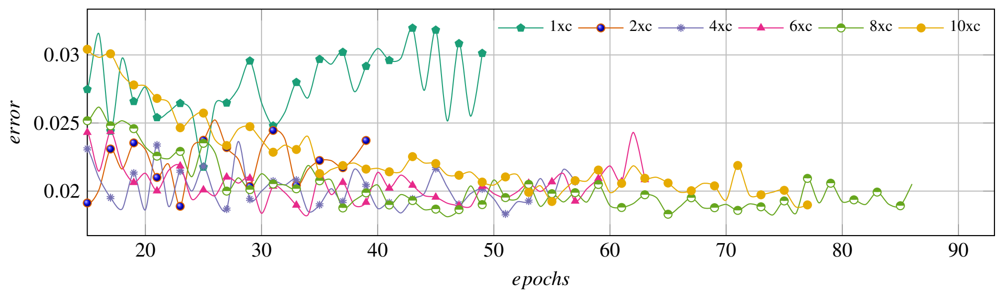
(a) Error variation



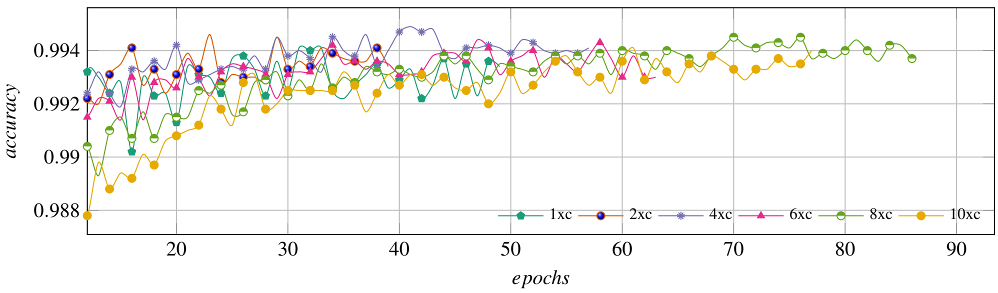
(b) Accuracy variation

Figure FC3.26: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs



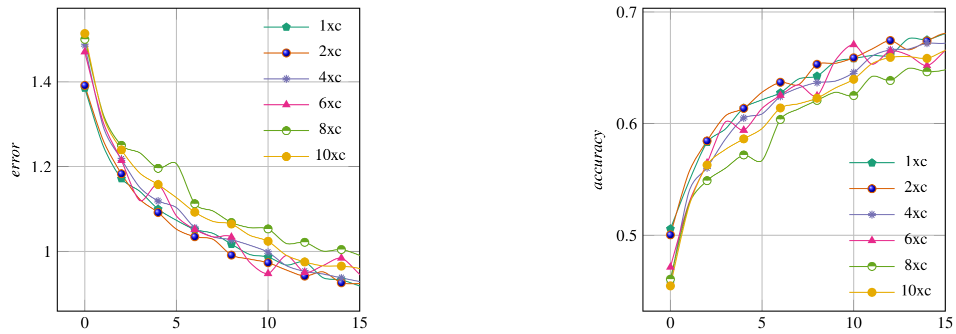
(b) Accuracy for later epochs

Figure FC3.27: *Different batch results for later epochs*

to write

3.4.1.2 CIFAR10

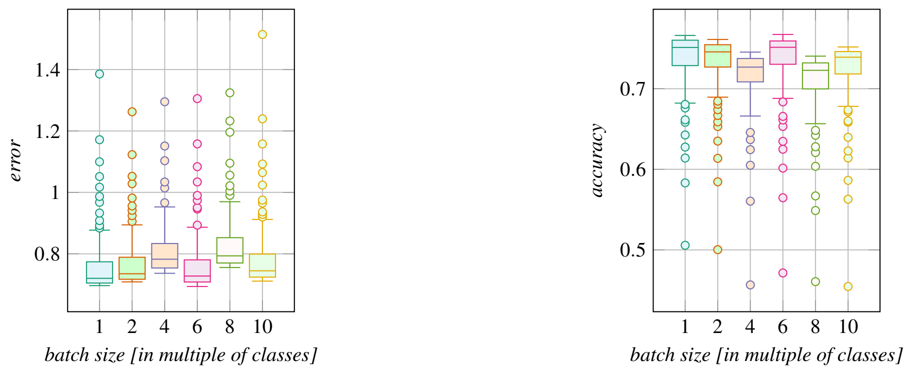
Three Layer, opti=adagrad, init=uniform



(a) Error for initiaNITCIFAR10 (b) Accuracy for initial epcohs

Figure FC3.28: *Different batch results for starting 15 epochs*

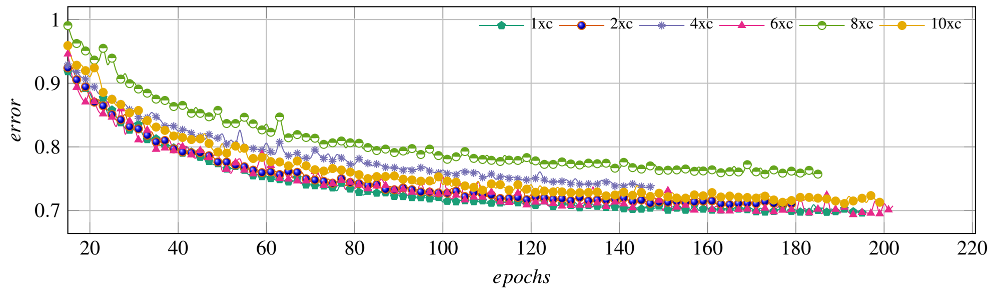
to write



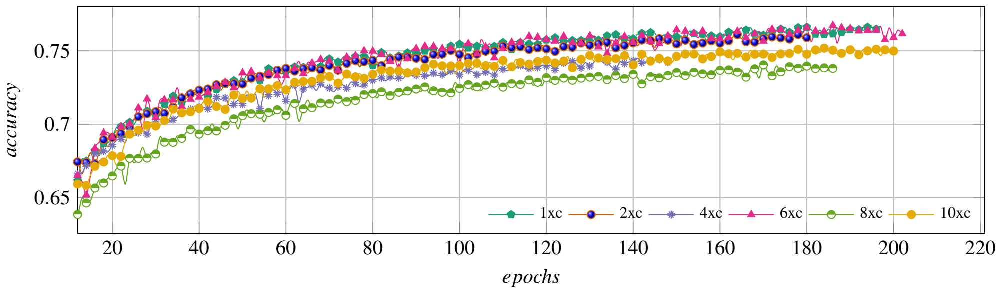
(a) Error variation (b) Accuracy variation

Figure FC3.29: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

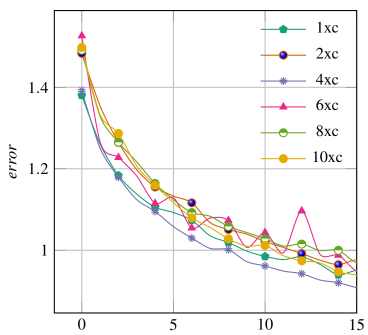


(b) Accuracy for later epochs

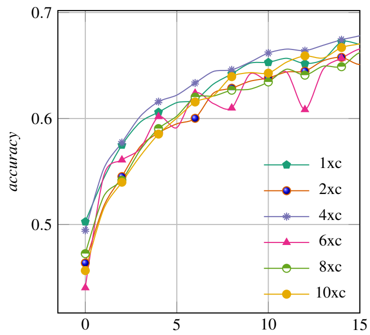
Figure FC3.30: *Different batch results for later epochs*

to write

Three Layer, opti=adagrad, init=glorot normal



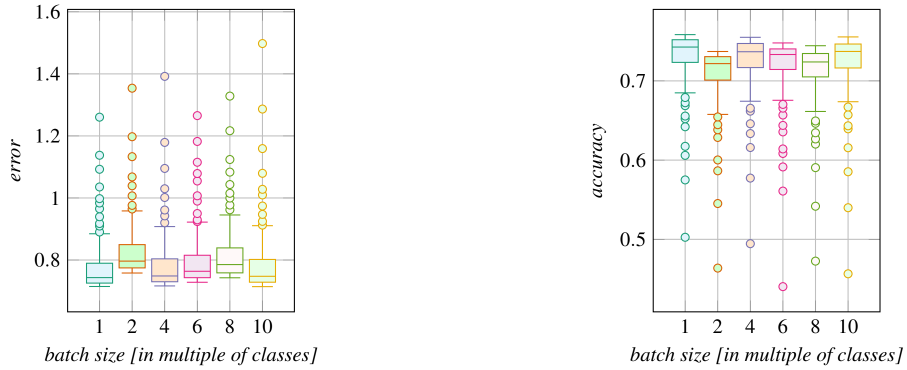
(a) Error for initiaNITCIFAR10



(b) Accuracy for initial epcohs

Figure FC3.31: *Different batch results for starting 15 epochs*

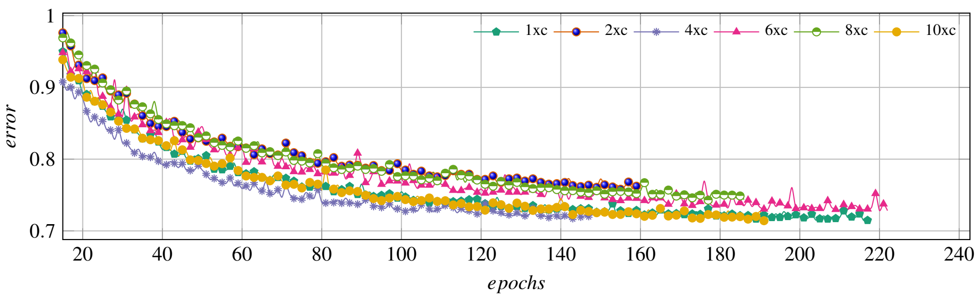
to write



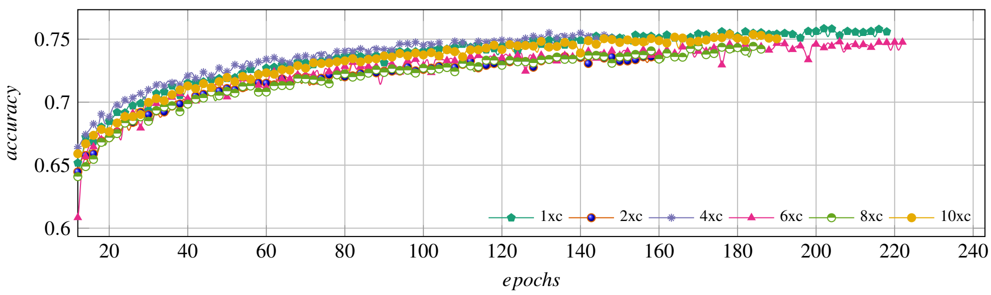
(a) Error variation (b) Accuracy variation

Figure FC3.32: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

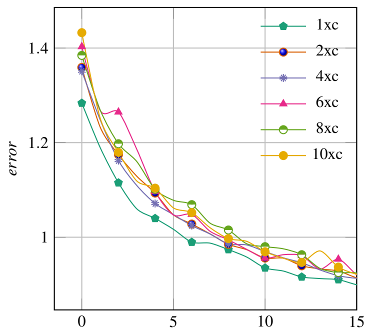


(b) Accuracy for later epochs

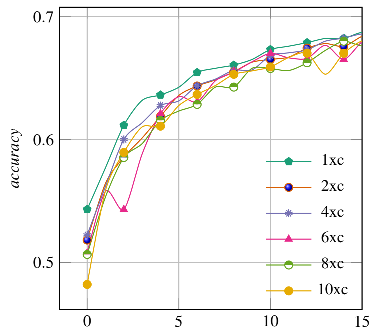
Figure FC3.33: *Different batch results for later epochs*

to write

Two Layer, opti=adagrad, init=normal



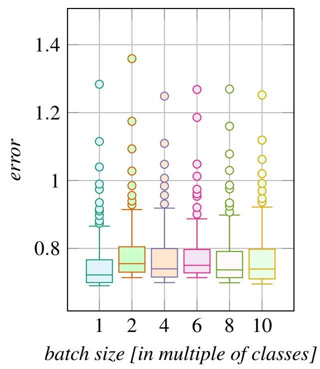
(a) Error for initiaNITCIFAR10



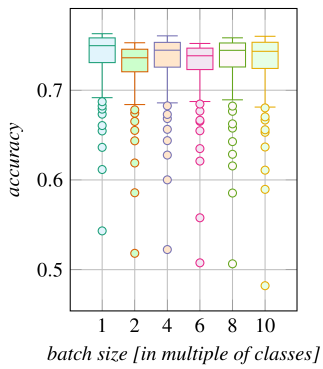
(b) Accuracy for initial epcohs

Figure FC3.34: *Different batch results for starting 15 epochs*

to write



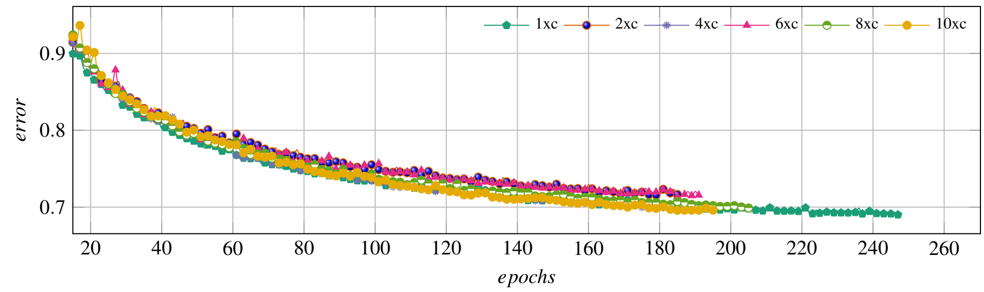
(a) Error variation



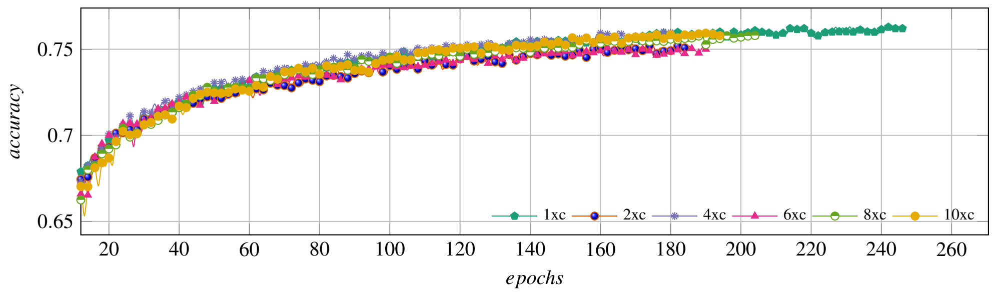
(b) Accuracy variation

Figure FC3.35: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs



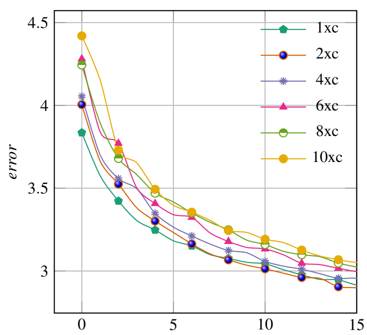
(b) Accuracy for later epochs

Figure FC3.36: *Different batch results for later epochs*

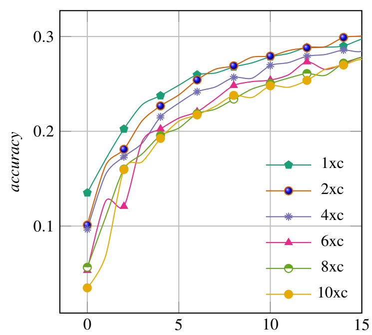
to write

3.4.1.3 CIFAR100

Two Layer, opti=adagrad, init=hessian uniform



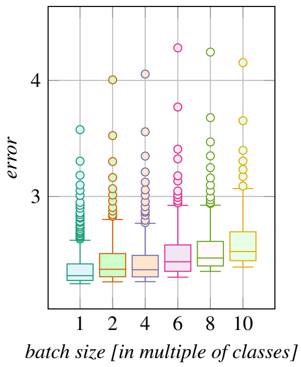
(a) Error for initiaNITCIFAR10



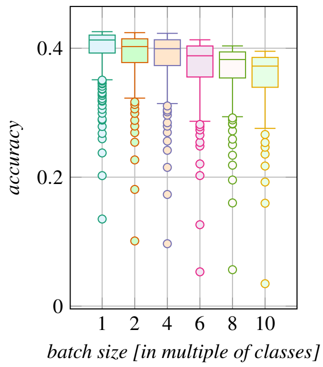
(b) Accuracy for initial epcohs

Figure FC3.37: *Different batch results for starting 15 epochs*

to write



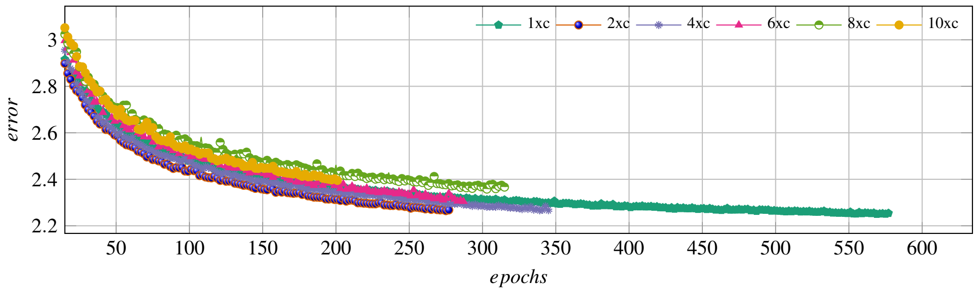
(a) Error variation



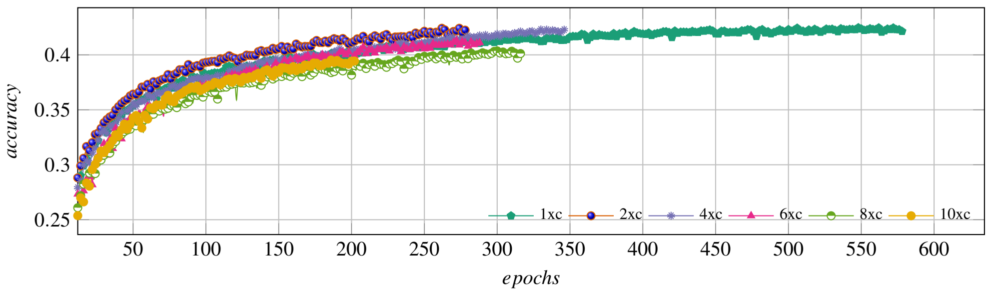
(b) Accuracy variation

Figure FC3.38: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs

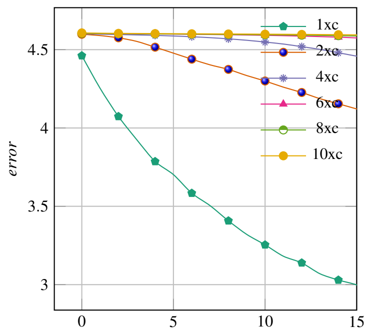


(b) Accuracy for later epochs

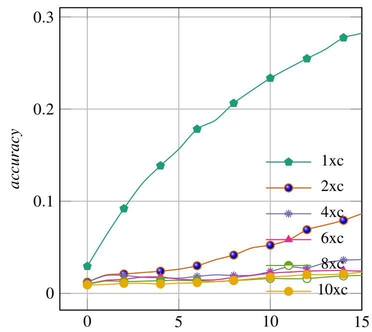
Figure FC3.39: *Different batch results for later epochs*

to write

Two Layer, opti=sgd with nesterov momentum, init=glorot normal



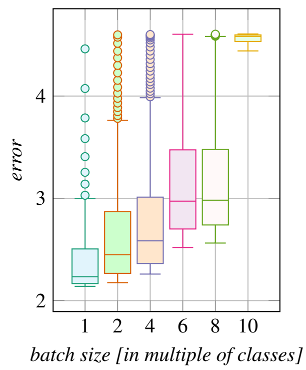
(a) Error for initiaNITCIFAR10



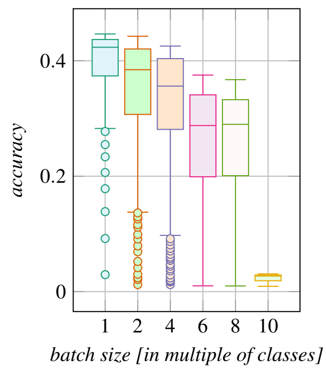
(b) Accuracy for initial epcohs

Figure FC3.40: *Different batch results for starting 15 epochs*

to write



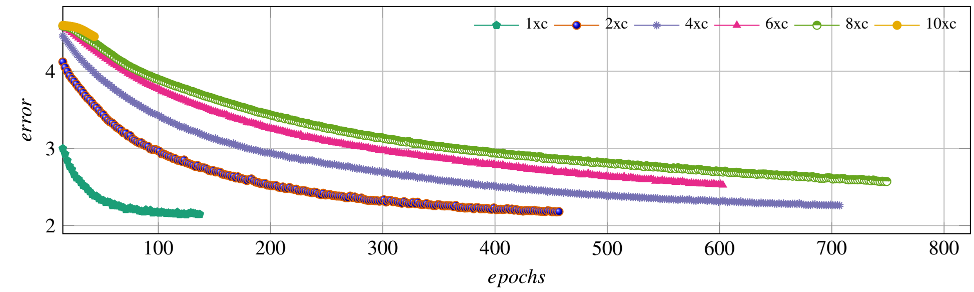
(a) Error variation



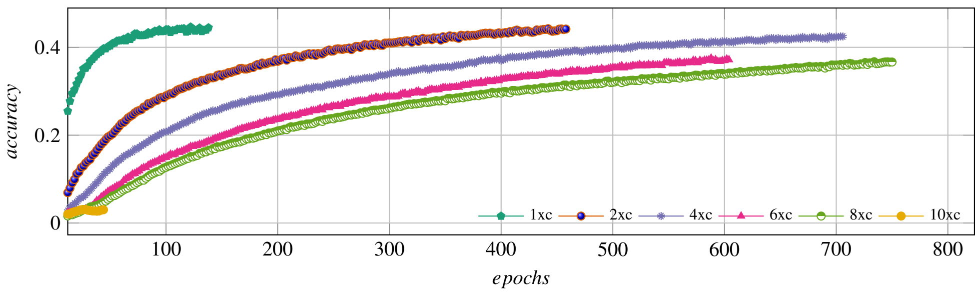
(b) Accuracy variation

Figure FC3.41: *accuracy and error plot for full training epochs*

to write



(a) Error for later epochs



(b) Accuracy for later epochs

Figure FC3.42: *Different batch results for later epochs*

to write

CHAPTER 4

TRAINING THE NETWORK

In this chapter network training is studied in detail. We will see available methods of training the network, their working algorithms and stopping criterion.

CHAPTER 5

RECOMMENDATIONS

BibTEX can be used to handle all your bibliographic needs. Simply add references to the file `ref.bib` and BibTEX will take care of the rest. An example of a BibTEX book, conference paper and journal article are given in the sample `ref.bib` file. Many online journals have links to BibTEX citations that you can download and incorporate into the `ref.bib` file. *Do not change the name of the file* `ref.bib`.

The order of the fields is unimportant. BibTEX will display them in the correct order when constructing your bibliography. Also note that you can specify information about a reference that may not even be included in the actual bibliography. For example, the ISBN field is not required by the bibliography, but you can, if you want, put the ISBN to the BibTEX entry.

We can cite a journal article [**?**] and a conference paper [18] in the same way as a book citation. More information can be found in [19].

CHAPTER 6

CONCLUSIONS

That’s all folks!

Bibliography

[1] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[2] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[3] Franois Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[4] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[5] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[6] Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag.

[7] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[9] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013.

[10] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[11] Timothy Dozat. Incorporating Nesterov Momentum into Adam.

[12] B.T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17, 1964.

[13] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[14] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[16] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *CoRR*, abs/1206.5533, 2012.

[17] Geoffrey E. Hinton. *A Practical Guide to Training Restricted Boltzmann Machines*, pages 599–619. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[18] FirstName LastName, FirstName I. LastName, and F.N. LastName Jr. Conference paper MUN title. In *Proceedings of the Conference of Sample Conferences*, pages 100–110, Apr. 1996.

[19] Leslie Lamport. *LaTeX: A Document Preparation System*. Addison-Wesley Publishing Company, second edition, 1994.

APPENDIX A

APPENDIX: HOW TO ADD AN APPENDIX

This is Appendix A.

You can have additional appendices too, (*e.g.*, `apdxb.tex`, `apdxc.tex`, *etc.*). These files need to be included in `thesis.tex`.

If you don't need any appendices, delete the appendix related lines from `thesis.tex`.

A.1 Equations

An example mathematical formulae is show in Eqn 1.1.

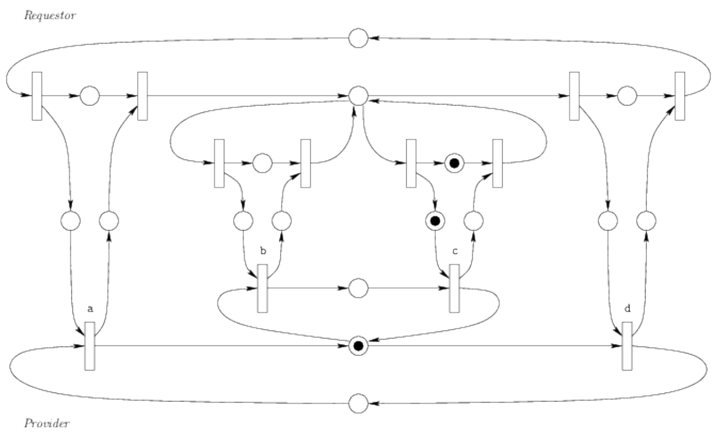


Figure FA1.1: Image of a deadlocked Petri net at 40% scaling.

Table TA1.1: Fall Semester Enrollment

	Undergraduate			Graduate		
	F/T	P/T	Total	F/T	P/T	Total
2004	13,191	2,223	15,414	1,308	879	2,187
2005	13,184	2,143	15,327	1,375	920	2,295
2006	12,809	2,224	15,033	1,373	899	2,272
2007	12,634	2,155	14,789	1,403	899	2,302
2008	12,269	2,208	14,477	1,410	1,005	2,415
2009	12,382	2,323	14,705	1,567	1,106	2,673

$$\sum_{i=0}^{n} i^2 \qquad \text{(Eqn 1.1)}$$

APPENDIX B

APPENDIX: HOW TO ADD ANOTHER ONE

This is Appendix B.