COURANT INSTITUTE OF MATHEMATICS SCIENCES, NEW YORK UNIVERSITY

# Deep Learning Assignment 1

Wenliang Zhao & Xuanang Chen

February 11, 2016

## 1 PROBLEM 1

### 1.1

Let $f(x_{in}) = \frac{1}{1+\exp^{-x_{in}}}$

$$\begin{aligned}
\frac{\partial E}{\partial x_{in}} &= \frac{\partial E}{\partial f(x_{in})}\frac{\partial f(x_{in})}{x_{in}} \\
&= \frac{\partial E}{\partial x_{out}}\frac{\partial f(x_{in})}{\partial x_{in}} \\
\frac{\partial f(x_{in})}{\partial x_{in}} &= \frac{\partial}{\partial x_{in}}\frac{1}{1+\exp^{-x_{in}}} \\
&= \frac{\exp^{-x_{in}}}{(1+\exp^{-x_{in}})^2} \\
\Longrightarrow \frac{\partial E}{\partial x_{in}} &= \frac{\partial E}{\partial x_{out}}\frac{\exp^{-x_{in}}}{(1+\exp^{-x_{in}})^2}
\end{aligned} \tag{1.1}$$

1.2

For $i = j$

$$\begin{aligned}
\frac{\partial (x_{out})_i}{\partial (x_{in})_j} &= \frac{\partial}{\partial (x_{in})_i} \frac{\exp^{-\beta (x_{in})_i}}{\sum_k \exp^{-\beta (x_{in})_k}} \\
&= (\frac{\partial}{\partial (x_{in})_i} \exp^{-\beta (x_{in})_k}) \times \frac{1}{\sum_k \exp^{-\beta (x_{in})_k}} + (\frac{\partial}{\partial (x_{in})_i} \frac{1}{\sum_k \exp^{-\beta (x_{in})_k}}) \times \exp^{-\beta (x_{in})_i} \\
&= (-\beta) \exp^{-\beta (x_{in})_i} \frac{1}{\sum_k \exp^{-\beta (x_{in})_k}} - \exp^{-\beta (x_{in})_i} \frac{1}{(\sum_k \exp^{-\beta (x_{in})_k})^2} (-\beta) \exp^{-\beta (x_{in})_i} \\
&= -\beta \frac{\exp^{-\beta (x_{in})_i}}{\sum_k \exp^{-\beta (x_{in})_k}} (1 - \frac{\exp^{-\beta (x_{in})_i}}{\sum_k \exp^{-\beta (x_{in})_k}})
\end{aligned} \tag{1.2}$$

For $i \neq j$

$$\begin{aligned}
\frac{\partial (X_{out})_i}{\partial (X_{in})_j} &= e^{-\beta (X_{in})_i} \frac{\partial}{\partial (X_{in})_j} \frac{1}{\sum_k e^{-\beta (X_{in})_k}} \\
&= e^{-\beta (X_{in})_i} \times - \frac{1}{(\sum_k -\beta (X_{in})_k)^2} \frac{\partial}{\partial (X_{in})_j} e^{-\beta (X_{in})_j} \\
&= e^{-\beta (X_{in})_i} \times - \frac{1}{(\sum_k -\beta (X_{in})_k)^2} (-\beta e^{-\beta (X_{in})_j}) \\
&= \frac{\beta e^{-\beta ((X_{in})_i + (X_{in})_j)}}{(\sum_k e^{-\beta (X_{in})_k})^2}
\end{aligned} \tag{1.3}$$

# 2 Torch (MNIST Handwritten Digit Recognition)

## 2.1 Introduction

Image recognition is one of the key tasks in machine learning area. In this project, we test two models - 2-layer neural network which is used as the base line and a three-stage convolutional neural network on MNIST dataset. Then we play some tricks on our convolutional neural network model and analyzes how those tricks influence the performance and efficiency of our model. Finally, we give our experimental results and conclusion.

## 2.2 Dataset and Models

We test our models on the MNIST dataset of handwritten digits, which has a training set of $60,000$ examples, and a test set of $10,000$ examples. All these black and white digits are size normalized, and centered in a fixed-size image with $28 \times 28$ pixels. Thus the dimensionality of of each page is image sample vector is $28 * 28 = 784$.

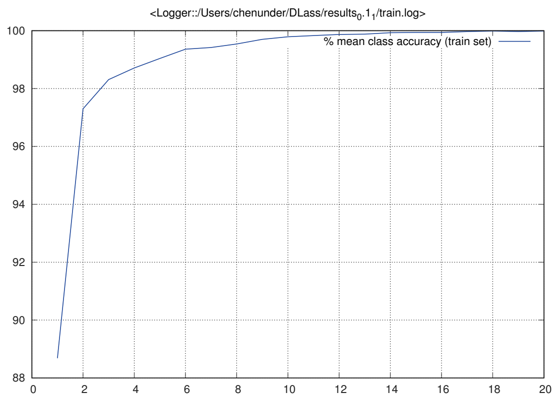
For comparison, we first test the dataset on simple 2-layer neural network with one hidden layer to get the baseline result. Then we try several tricks on the convolutional neural network architecture.

The ConvNet architecture of our experiment is composed of two repeated stacked stage and one fully-connected 2-layer neural network. Each stacked stage contains a convolutional module, a non-linear module, a pooling module and a normalization module.

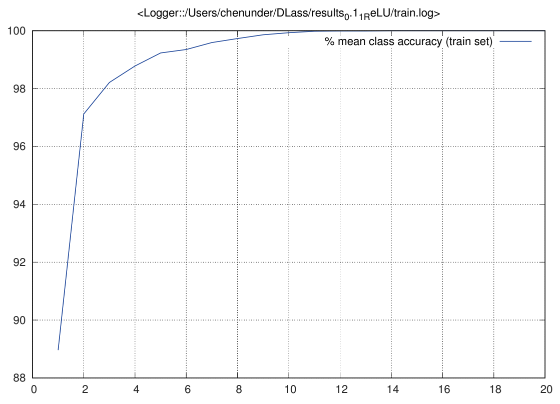
## 2.3 Experiment settings

### 2.3.1 non-linear module

On non-linear module, we test both the sigmoid function (Tanh) and ReLU. We use 50000 training data and 10000 validate data. And we find that learning capacity on ReLU model is severely worse than Tanh model as shown in the following figures. We assume that the dropping of negative parts causes such weak learning capacity. So we use Tanh function as non-linear module in the following tests.

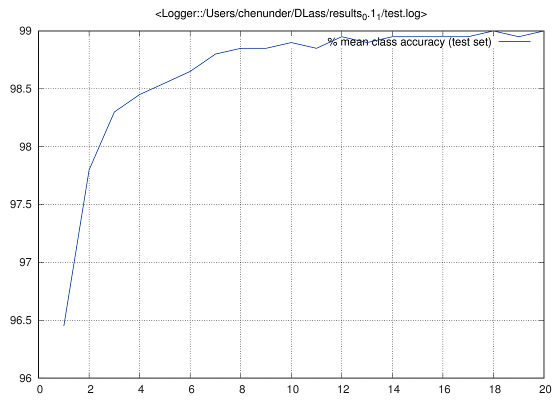


(a) Tanh

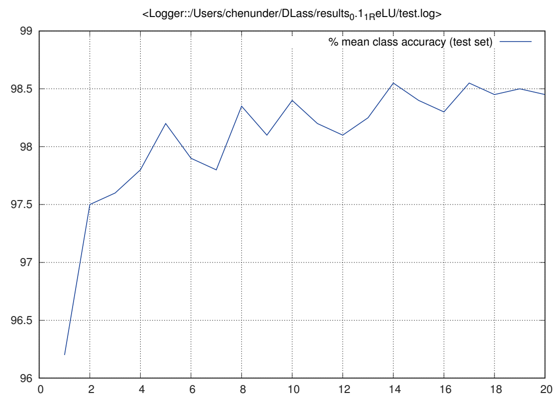


(b) ReLU

Figure 2.1: training accuracy



(a) Tanh

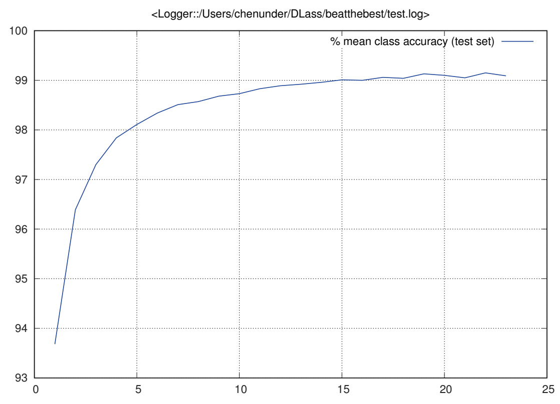


(b) ReLU

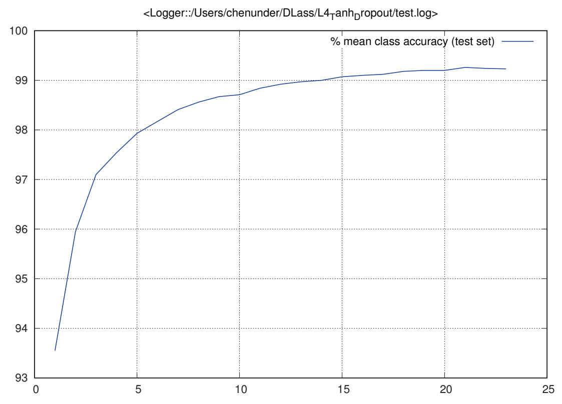
Figure 2.2: testing accuracy

### 2.3.2 Pooling

On pooling modules, we try max-pooling and LP-pooling. For max-pooling, we first test the original $2 \times 2$ non-overlapping situation, then we set the filter size to 3, stride to 2, paddle to 1 in order to test the overlapping pooling situation. For LP-pooling, we compare the result of $p = 1, 2, 4, 8$ on our validation set and find $p = 2$ and $p = 4$ gives the best result.

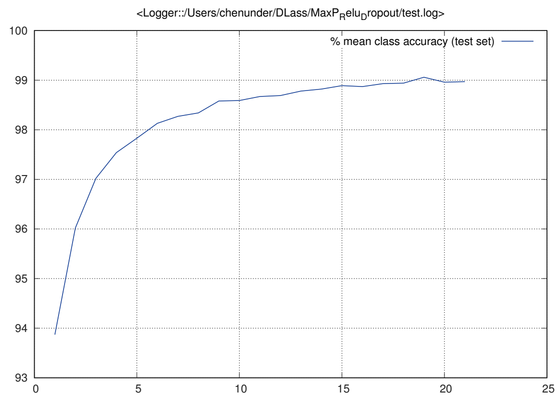


(a) L2 pooling

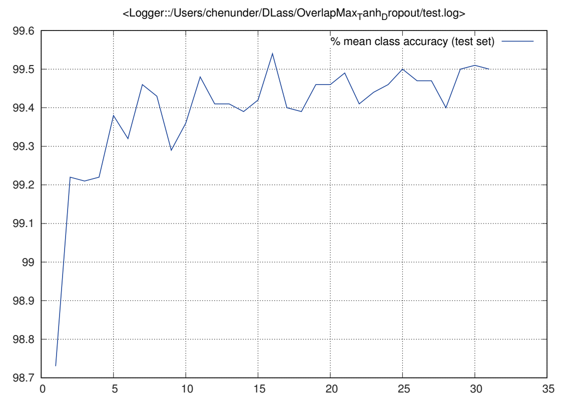


(b) L4 pooling

Figure 2.3: LP pooling



(a) Max pooling



(b) Overlapping max pooling

Figure 2.4: Max pooling

### 2.3.3 **Momentum**

During the optimization using gradient decent, when there is long narrow local minimal in the objective value space (steep slope in 1 dimension and fairly flat in others), it is usually slower in convergence. The optimization takes more time for decent and jumpping out of the local minimal. When gradient direction is parallel to the a steep slope, the convergence speed is almost doubled; when in a flatter area or gradient direction chages frequently, the updating of weights slow down to search more possibilities nearby. It is usually coupled with a larger laerning rate in order to accelerate convergence significantly.

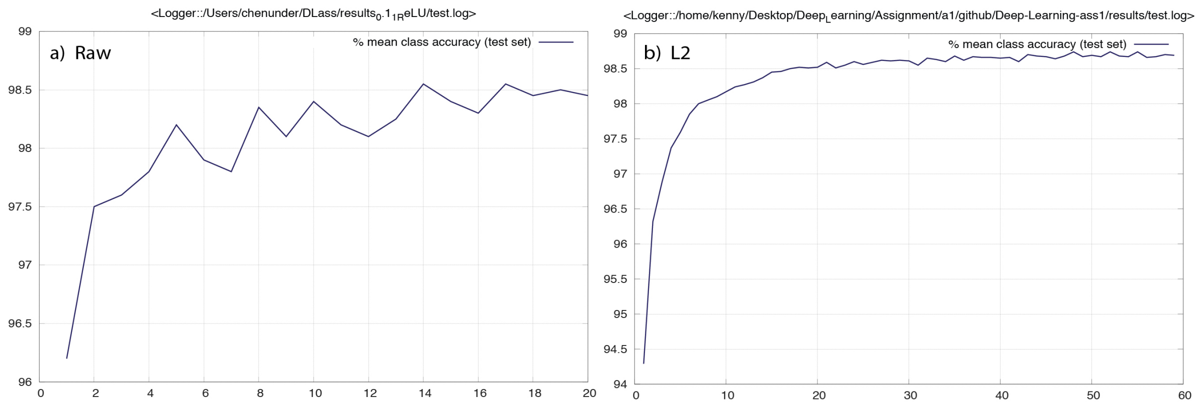


Figure 2.5: The effect of regularization

### 2.3.4 REGULARIZATION

Although being shown to have strong ability of prediction in supervised learning, over fitting is a serious problem in convolutional neural network. The direct way of visualizing it is comparing the accuracy as a function of epoch in training and testing. When there is no significant growth in testing but training accuracy is still increasing slowly, this usually indicate certain amount of over fitting. An straight forward way of preventing from this is so called **"early stopping"** - to stop training when testing accuray does not improve any more. In practice, we found to set number of epochs at 20 – 30, usually can obtain a stable result.

### 2.3.5 NORM REGULARIZATION

Norm based regularization is commonly used in statistics, machine learning. The typical way of doing regularization in optimization is adding a penalty term behind the objective function, which is a multiplication of a hyperparameter $\lambda$ and the norm of weights. There many kinds of variations of regularization such as regularized least square, regularized logistic regression and regularized support vector machine. In these regularization, if the penalty is $L1$ norm, it is usually called ridge regression. If the penalty is $L2$ norm, it is called lasso. If one combines ridge regression with lasso, and assign ratio to both of them, the derived regularization is called elastic net. In pratice, ridge regression is noise is high or the system is rank deficient. The derived result is usually more smooth and robust. When the model is in very high dimension and relatively lack of input data, there needs some technique to obtain sparcity of model. Lasso is used in this situation and usually very useful for feature selection. Below is example using that the $L2$ norm regularization reduces over fitting significantly. $a$) shows a testing curve using ReLU activate function, and it starts oscillating after epoch 5 which suggests over fitting occurs - prediction is unstable even training score is very high. $b$ shows by using regularization ($C = 0.2, \lambda = 0.01$), the learning curve is much moother suggesting the result is mor robust than the one without regularization.

### 2.3.6 DROPOUT

Another way of doing regularization is dropout which was proposed by Hintom et al. in 2012. In the network, every element of a layer's outputs is kept with certain probability $p$. It avoids the case that some node's effect only depends on the existing of another node - by random sampling, it is likely that no any two nodes will always in the network through different iterations. Dropout significantly increases the generalization ability of convolutional neural network and improves test performance.

In this study, we trained several models using the dropout tchnique. We add dropout processes after each pooling layer. In the first dropout, we drop nodes with probility of 0.2, and in the second we st the probability to 0.5. One interesting observation is that, after dropout, the testing accuracy is even higher than training which is not very common is the regularization techniques introduced above. Empirically, by using $L1/L2$ regularization, the testing score is usually lower than training.

## 2.4 RESULTS AND FUTURE WORK

### 2.4.1 HYPER-PARAMETERS

The hyper-parameters here consist of learning rate, batch size, momentum and regularization parameters.

We choose 0.001 as our learning rate. For batch size, we test several batch sizes that are factor of two. We finally choose batch size 16 since we both want efficiency and stochasticity for out model.

| Batch size | ms per sample train | ms per sample test |
|---|---|---|
| 1 | 32.23 | 17.79 |
| 2 | 26.78 | 16.43 |
| 4 | 17.63 | 20.94 |
| 8 | 16.26 | 18.54 |
| 10 | 15.51 | 17.15 |
| 16 | 14.41 | 17.28 |
| 32 | 15.83 | 24.16 |
| 128 | 13.18 | 18.41 |

Table 2.1: batch size efficiency

We test different momentum parameter - 0, 0.1, 0.5, 0.7, 0.8 for the MNIST data set. From the speed of training, we do not see significance by importing the momentum parameter. A possible reason of this is the data set is relatively clean, so that the test score is high even in the first iteration suggesting a fairly flat and smooth surface so the algorithm quickly finds the region of optimal, and search slowly and locally after the first epoch. Although no obvious difference, we still set $p$ to 0.7 as recommended by previous works.

Cross validation is the typical method for tuning the parameters in regularization. However, we haven't finished the cross validation code for this study. We build an elastic net procedure which contains two tuning parameters, $\lambda$ - the scale parameter for the norm penalty, and $C$ - indicates the portion of lasso. For example $C = 0.5$ means, we do $50\%L1$ and $50\%L2$ regularization. In practice, we find $L1$ norm seems to be not very useful for this study. The possible explanation is system is not very complicated so that sparcity doesn't help a lot in this perticular case. We thus always set $C = 0$ indicating we ignore the $L1$ norm regularization. As for $L2$ norm regularization, we select several values for $\lambda$ based on empirical information. The selected values are 0.01, 0.005, 0.001, 0.0005.

### 2.4.2 Experiments result

Here is he final result table we get for this experiment.

| Model | Test error rate |
|---|---|
| simple 2 layer nn | 4.6 |
| L2 pooling + Tanh | 0.69 |
| L2 pooling + ReLU | 0.81 |
| Max pooling + Tanh | 0.94 |
| L2 pooling + Tanh + dropout | 0.67 |
| L4 pooling + Tanh + dropout | 0.74 |
| **Overlapping max pooling + Tanh + dropout** | **0.51** |

Table 2.2: Results table

### 2.4.3 **FUTURE WORK**

Out study gives the best result using overlapping max pooling model, which has more learning features than other models. This suggests that our architecture is still lack of capacity and we can research more on the architecture in the future. This time we spend much time tuning the hyper-parameters and it turns out that such parameters will be trivial if our architecture is not that capable.

Also, one interesting observation is that ReLU which is generally thought to improve the performance do not perform well in our model. Assumably, we attribute this to the weak capacity of our model. Since ReLU simply drop all the negative parts, although better sparsity we get, it might have under-fitting problem and thus have a worse result. We will conduct more experiments on this issue in the future.