Homework 3

CSCI 5525: Machine Learning

Due on Nov 7th 11:00am (before the class)

Please type in your info:

- **Name**: Chih-Tien Kuo
- **Student ID**: 5488927
- **Email**: kuo00013@umn.edu
- **Collaborators, and on which problems:**

Homework Policy. (1) You are encouraged to collaborate with your classmates on homework problems, but each person must write up the final solutions individually. You need to fill in above to specify which problems were a collaborative effort and with whom. (2) Regarding online resources, you should **not**:

- Google around for solutions to homework problems,
- Ask for help on online.
- Look up things/post on sites like Quora, StackExchange, etc.

Submission. Submit a PDF using this LaTeX template for written assignment part and submit .py files for all programming part. You should upload all the files on Canvas.

Written Assignment

Instruction. For each problem, you are required to write down a full mathematical proof to establish the claim.

Problem 1. VC Dimension.

(**5 points**) Prove this claim formally: If $\mathcal{F}$ is finite, then $\text{VCD}(\mathcal{F}) \leq \log(|\mathcal{F}|)$.

Answer.

Proof. Suppose $\mathcal{F}$ can shatter $\text{VCD}(\mathcal{F})$ points, then the hypothesis class $\mathcal{F}$ can separate the data to at least $2^{\text{VCD}(\mathcal{F})}$ dichotomies.

$$2^{\text{VCD}(\mathcal{F})} \leq |\mathcal{F}|$$
$$\Rightarrow \text{VCD}(\mathcal{F}) \leq log(|\mathcal{F}|)$$

□

Problem 2. Uniform convergence.

(**10 points**)

In this problem, we will prove a stronger generalization error bound for the binary classification that uses more information about the distribution. Let us say that P is a distribution over (X, Y) pairs where $X \in \mathcal{X}$ and $Y \in \{+1, -1\}$. Let $\mathcal{H} \subset \mathcal{X} \to \{+1, -1\}$ be a finite hypothesis class and let ℓ denote the zero-one loss $\ell(\hat{y}, y) = 1\{\hat{y} \neq y\}$. Let $R(h) = \mathbb{E}\,\ell(h(X), Y)$ denote the risk and let $h^* = \min_{h \in \mathcal{H}} R(h)$. Given n samples, let $\hat{h}_n$ denote the empirical risk minimizer. Here, we want to prove a sample complexity bound of the form:

$$R(\hat{h}_n) - R(h^*) \leq c_1 \sqrt{\frac{R(h^*) \log(|\mathcal{H}|/\delta)}{n}} + c_2 \frac{\log(|\mathcal{H}|/\delta)}{n} \tag{1}$$

for constants c_1, c_2. If $R(h^*)$ is small, this can be a much better bound than the usual excess risk bound. In particular, if $R(h^*) = 0$, this bound recovers the $1/n$-rate.

a) To prove the result, we will use Bernstein's inequality, which is a sharper concentration result.

Theorem 1. *(Bernstein's inequality). Let $X_1, ..., X_n$ be i.i.d real-valued random variables with mean zero, and such that $|X_i| \leq M$ for all i. Then, for all $t > 0$*

$$\mathbb{P}[\sum_{i=1}^{n} X_i \geq t] \leq \exp\left(-\frac{t^2/2}{\sum_{i=1}^{n} \mathbb{E}[X_i^2] + Mt/3}\right) \tag{2}$$

Using this inequality, show that with probability at least $1 - \delta$

$$|\bar{X}| \leq \sqrt{\left(\frac{2\,\mathbb{E}\,X_1^2 \log(2/\delta)}{n}\right)} + \frac{2M \log(2/\delta)}{3n} \tag{3}$$

where $\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ and X_i's satisfy the conditions of Bernstein's inequality.

b) Using Eq. (3) and the union bound, show Eq. (1)

Answer.

(a) Since $X_1, ..., X_n$ is *i.i.d* real-valued random variables with mean zero, from Eq. (2) we can also get:

$$\mathbb{P}[\sum_{i=1}^{n} X_i \leq -t] \leq \exp\left(-\frac{t^2/2}{\sum_{i=1}^{n} \mathbb{E}[X_i^2] + Mt/3}\right) \tag{4}$$

Let $\exp\left(-\frac{t^2/2}{\sum_{i=1}^{n} \mathbb{E}[X_i^2] + Mt/3}\right) = \delta/2$, then

$$\Rightarrow \ln\frac{2}{\delta} = \frac{t^2/2}{\sum_{i=1}^{n} \mathbb{E}[X_i^2] + Mt/3} \tag{5}$$

$$\Rightarrow \frac{t^2}{2} - \frac{Mt}{3}\ln\frac{2}{\delta} - \ln\frac{2}{\delta}\sum_{i=1}^{n} \mathbb{E}[X_i^2] = 0 \tag{6}$$

We then can solve the quadratic function for t

$$t = \frac{Mt}{3}\ln\frac{2}{\delta} + \sqrt{\left(\frac{Mt}{3}\ln\frac{2}{\delta}\right)^2 + 2\ln\frac{2}{\delta}\sum_{i=1}^{n}\mathbb{E}[X_i^2]} \tag{7}$$

$$\leq \frac{Mt}{3}\ln\frac{2}{\delta} + \sqrt{\left(\frac{Mt}{3}\ln\frac{2}{\delta}\right)^2} + \sqrt{2\ln\frac{2}{\delta}\sum_{i=1}^{n}\mathbb{E}[X_i^2]} \tag{8}$$

$$= \frac{2Mt}{3}\ln\frac{2}{\delta} + \sqrt{2\ln\frac{2}{\delta}\sum_{i=1}^{n}\mathbb{E}[X_i^2]} \tag{9}$$

We can now substitute $\frac{\delta}{2}$ and t into Eq. (2) and Eq. (4), which become

$$\mathbb{P}\left[\sum_{i=1}^{n}X_i > \frac{2Mt}{3}\ln\frac{2}{\delta} + \sqrt{2\ln\frac{2}{\delta}\sum_{i=1}^{n}\mathbb{E}[X_i^2]}\right] \leq \frac{\delta}{2} \tag{10}$$

$$\mathbb{P}\left[\sum_{i=1}^{n}X_i < -\frac{2Mt}{3}\ln\frac{2}{\delta} - \sqrt{2\ln\frac{2}{\delta}\sum_{i=1}^{n}\mathbb{E}[X_i^2]}\right] \leq \frac{\delta}{2} \tag{11}$$

Combining Eq. (10) and Eq. (11), we can get the following:

$$\mathbb{P}\left[|\sum_{i=1}^{n}X_i| > \frac{2Mt}{3}\ln\frac{2}{\delta} + \sqrt{2\ln\frac{2}{\delta}\sum_{i=1}^{n}\mathbb{E}[X_i^2]}\right] \leq \delta \tag{12}$$

$$\Rightarrow\mathbb{P}\left[|\bar{X}| \leq \frac{2Mt}{3}\ln\frac{2}{\delta} + \sqrt{2\ln\frac{2}{\delta}\mathbb{E}[X_1^2]}\right] \geq 1-\delta \tag{13}$$

Eq. (13) derived is by dividing Eq. (12) by n to get the mean of X

(b)

Problem 3. Model selection.

(**10 points**) In problem 2, we proved the $R(h^*)$ bound. The goal in this problem is to do this simultaneously while doing structural risk minimization. Specifically, given a family of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 ... \subset \mathcal{H}_L$ of sizes $N_1 \leq N_2 \leq ... \leq N_L < \infty$, a loss function bounded on [0,1] and a sample of size n, design an algorithm that guarantees

$$R(\hat{h}) \leq \min_{i\in[\mathrm{L}]}\min_{h^*\in\mathcal{H}_i}\left\{R(h^*) + c_1\sqrt{\frac{R(h^*)\log(LN_i/\delta)}{n}} + c_2\frac{\log(LN_i/\delta)}{n}\right\}$$

for $n \geq 2$. Your algorithm may use ERM (so need not be efficient) and your constants may vary.

You may find it useful to use the following *empirical Bernstein inequality.*

Theorem 2. *Let $X_1, ..., X_n$ be iid random variables from a distribution P supported on [0,1] and define the sample variance $V_n = \frac{1}{n(n-1)}\sum_{1\leq i<j\leq n}(X_i - X_j)^2$. Then for any $\delta \in (0,1)$ with probability at least 1 - δ*

$$\mathbb{E}\,X - \frac{1}{n}\sum_{i=1}^{n}X_i \leq \sqrt{\frac{2V_n\log(2/\delta)}{n}} + \frac{7\log(2/\delta)}{3(n-1)} \tag{14}$$

Your answer.

Problem 4. Neural network.

(**10 points**) Consider a neural neural network with input $x \in R^{d_i,1}$ and the number of classes being d_o. The network can be given by $\hat{y} = \text{softmax}(W_2(ReLU(W_1x + b_1)) + b_2)$ where $W_1 \in R^{d_1,d_i}$, $b_1 \in R^{d_1,1}$, $W_2 \in R^{d_o,d_1}$, $b_2 \in R^{d_o,1}$ and Loss $L = \text{cross\_entropy}(y, \hat{y})$

Cross entropy loss between $y, \hat{y}$ is given by $-\sum_i^m y_i \log(\hat{y}_i)$ and softmax clamps x to [0,1] range using $\frac{e^{x_i}}{\sum_i e^{x_i}}$.

Derive expressions for partial derivatives (be precise and clear) $\frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_2}, \frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial b_1}$

Answer. Set $Z_2 = W_2(ReLU(W_1x + b_1)) + b_2$, $Z_1 = W_1x + b_1$, and $H_1 = ReLU(W_1x + b_1)$

1.

$$\begin{aligned}\frac{\partial L}{\partial W_2} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z_2} \frac{\partial Z_2}{\partial W_2} \\ &= \left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \left(\begin{bmatrix} \sigma(Z_{2,1})(1-\sigma(Z_{2,1})) & -\sigma(Z_{2,2})\sigma(Z_{2,1}) & \dots & -\sigma(Z_{2,d_0})\sigma(Z_{2,1}) \\ -\sigma(Z_{2,1})\sigma(Z_{2,2}) & \sigma(Z_{2,2})(1-\sigma(Z_{2,2})) & \dots & -\sigma(Z_{2,d_0})\sigma(Z_{2,2}) \\ \vdots & \vdots & \ddots & \vdots \\ -\sigma(Z_{2,1})\sigma(Z_{2,d_0}) & -\sigma(Z_{2,2})\sigma(Z_{2,d_0}) & \dots & \sigma(Z_{2,d_0})(1-\sigma(Z_{2,d_0})) \end{bmatrix} \right) \\ &\quad \Big(ReLU(W_1x + b_1) \Big)\end{aligned}$$

$$= \boxed{\left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \Big[\sigma(j)(\delta_{ij} - \sigma(i)) \Big] \Big(ReLU(W_1x + b_1) \Big)}$$

, where $\sigma(\cdot)$ is softmax function and δ_{ij} is Kronecker delta

2.

$$\begin{aligned}\frac{\partial L}{\partial b_2} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z_2} \frac{\partial Z_2}{\partial b_2} \\ &= \left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \Big[\sigma(j)(\delta_{ij} - \sigma(i)) \Big] (1) \\ &= \boxed{\left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \Big[\sigma(j)(\delta_{ij} - \sigma(i)) \Big]}\end{aligned}$$

3.

$$\begin{aligned}\frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z_2} \frac{\partial Z_2}{\partial H_1} \frac{\partial H_1}{\partial Z_1} \frac{\partial Z_1}{\partial W_1} \\ &= \boxed{\left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \Big[\sigma(j)(\delta_{ij} - \sigma(i)) \Big] (W_2) \left(\begin{cases} 1, \text{if } Z_1 > 0 \\ 0, \text{if } Z_1 < 0 \end{cases} \right) (x^\intercal)}\end{aligned}$$

4.

$$\begin{aligned}
\frac{\partial L}{\partial W_1} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial Z_2} \frac{\partial Z_2}{\partial H_1} \frac{\partial H_1}{\partial Z_1} \frac{\partial Z_1}{\partial b_1} \\
&= \left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \Big[\sigma(j)(\delta_{ij} - \sigma(i)) \Big] (W_2) \left(\begin{cases} 1, \text{if } Z_1 > 0 \\ 0, \text{if } Z_1 < 0 \end{cases} \right) (1) \\
&= \boxed{\left(-\sum_{j=1}^{d_0} y_j \frac{1}{\hat{y}} \right) \Big[\sigma(j)(\delta_{ij} - \sigma(i)) \Big] (W_2) \left(\begin{cases} 1, \text{if } Z_1 > 0 \\ 0, \text{if } Z_1 < 0 \end{cases} \right)}
\end{aligned}$$

Programming Assignment

Instruction. For each problem, you are required to report descriptions and results in the PDF and submit code as python file (.py) (as per the question).

- **Python** version: Python 3.
- Please follow PEP 8 style of writing your Python code for better readability in case you are wondering how to name functions & variables, put comments and indent your code
- **Packages allowed**: pytorch, numpy, pandas, matplotlib
- **Submission**: Submit report, description and explanation of results in the main PDF and code in .py files.
- Please PROPERLY COMMENT your code in order to have utmost readability

Programming Common

This programming assignment focuses on implementing neural network for handwritten digits. This problem is about multi class classification and you will use MNIST dataset. You already have an idea of the dataset by now.

You will use pytorch to implement neural network. The implementation has to be for the CPU version only - No GPU's or MPI parallel programming is required for this assignment. Follow the installation instructions at https://pytorch.org in case you want to use your local machine, we recommend using conda environment.

Here is the pytorch tutorial. If you are new to pytorch, we recommend you to go through the tutorial here. https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html

Problem 5. Neural Network Implementation.

(**25 Points**)

1. (**No Points**) You can directly download MNIST in pytorch using torchvision.datasets.MNIST. It will allow you to

```
trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True,
transform=transform)
testset = torchvision.datasets.MNIST(root='./data', train=False, download=True,
transform=transform)
```

2. (**8 Points**) First, implement a multi-layer fully connected network:
   - Input: 1-channel input, size 28x28
   - Keep batch size as 32.
   - Fully connected layer 1: Input with bias; output - 128
   - ReLu Layer
   - Fully connected layer 2: input - 128; output - 10
   - Softmax layer
   - Use cross entropy as loss function

- Use SGD as optimizer.

Train using mini-batches of the given batch size. Plot loss and training accuracy for every epoch. At the end of the training, save your model with the name "mnist-fc". Load the saved model and report testing accuracy on the reloaded model. We should be able to reload your trained model and test.

epoch: An epoch tells you the number of times the learning algorithm sees the whole training data set. When it has seen all the training samples, you say 1 epoch is over and you start iterating in the next epoch.

3. (**10 Points**) Implement a convolutional neural network with the following specifications.
   - Input: 1-channel input, size 28x28
   - Keep batch size as 32.
   - Convolution layer: Convolution kernel size is (3, 3) with stride as 1. Input channels - 1; Output channels - 20
   - Max-pool: 2x2 max pool
   - ReLu Layer
   - Flatten input for feed to fully connected layers
   - Fully connected layer 1: flattened input with bias; output - 128
   - ReLu Layer
   - Fully connected layer 2: input - 128; output - 10
   - Softmax layer as above
   - Use cross entropy as loss function
   - Use SGD as optimizer.

   Train using mini-batches of the given batch size. Plot loss and training accuracy for every epoch. At the end of the training, save your model with the name "mnist-cnn". Load the saved model and report testing accuracy on the reloaded model. We should be able to reload your trained model and test.

4. (**4 Points**) For the convolutional network implemented above, vary the batch sizes as [32, 64, 96, 128] and plot the convergence run time vs batch sizes.

5. (**3 Points**) "torch optim" provides many optimizers. Try the following optimizers in your last implementation - "SGD", "ADAM", "ADAGRAD". (SGD is already covered in the class, for ADAM and ADAGRAD refer to https://arxiv.org/abs/1412.6980and http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf respectively). Plot loss vs epochs for each optimizer. Briefly describe.

Submission: Submit all plots requested and explanation in latex PDF. Submit your program in a file named (hw3_mnistfc.py and hw3_mnistcnn.py).

Answer.

1. None

2. Fully connected network: **Testing accuracy: 94.14%** (learning rate: 0.3 , 20 epochs)

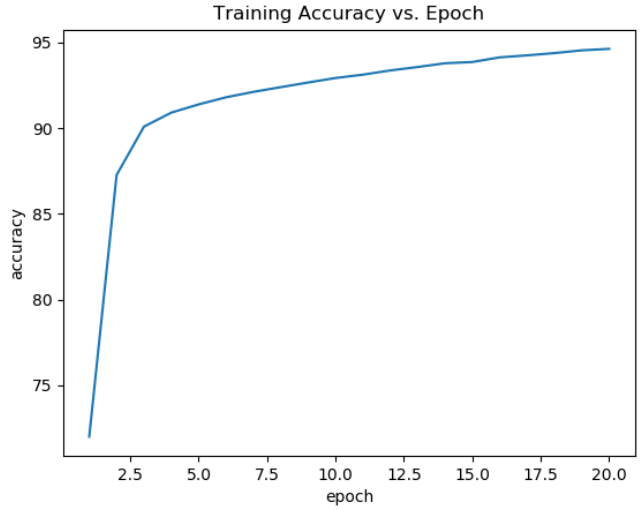

Figure 1: FC network training loss

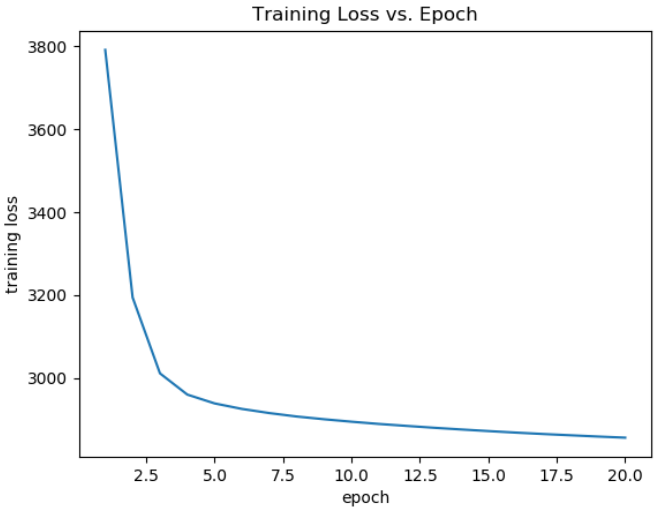

Figure 2: FC network training accuracy

```
epoch  1: [loss: 3791.58, accuracy: 71.998%]
epoch  2: [loss: 3194.61, accuracy: 87.262%]
epoch  3: [loss: 3011.51, accuracy: 90.085%]
epoch  4: [loss: 2960.31, accuracy: 90.902%]
epoch  5: [loss: 2938.99, accuracy: 91.382%]
epoch  6: [loss: 2925.72, accuracy: 91.795%]
epoch  7: [loss: 2915.74, accuracy: 92.112%]
epoch  8: [loss: 2907.43, accuracy: 92.387%]
epoch  9: [loss: 2900.83, accuracy: 92.658%]
epoch 10: [loss: 2894.94, accuracy: 92.920%]
epoch 11: [loss: 2889.55, accuracy: 93.115%]
epoch 12: [loss: 2884.89, accuracy: 93.362%]
epoch 13: [loss: 2880.39, accuracy: 93.562%]
epoch 14: [loss: 2876.26, accuracy: 93.780%]
epoch 15: [loss: 2872.37, accuracy: 93.853%]
epoch 16: [loss: 2868.67, accuracy: 94.125%]
epoch 17: [loss: 2865.28, accuracy: 94.242%]
epoch 18: [loss: 2862.23, accuracy: 94.373%]
epoch 19: [loss: 2859.19, accuracy: 94.535%]
epoch 20: [loss: 2856.46, accuracy: 94.620%]
Accuracy of the network on the 10000 test images: 94.140%
```

Figure 3: FC network testing accuracy

3. For convolutional neural network, I split the training dataset into training set (80%) and validation set (20%), and applied early stopping when the accuracy in validation set decreases (check every epoch). **Testing accuracy: 97.04%** (learning rate: 0.3, 18 epochs)

4. The convergence time for different batch size is shown in Fig. (7).

5. The training loss plot is shown in Fig. (8). From the plot we can see that both optimizers have much lower training losss than SGD optimizer, and ADAM has slightly better performance than ADAGRAD.

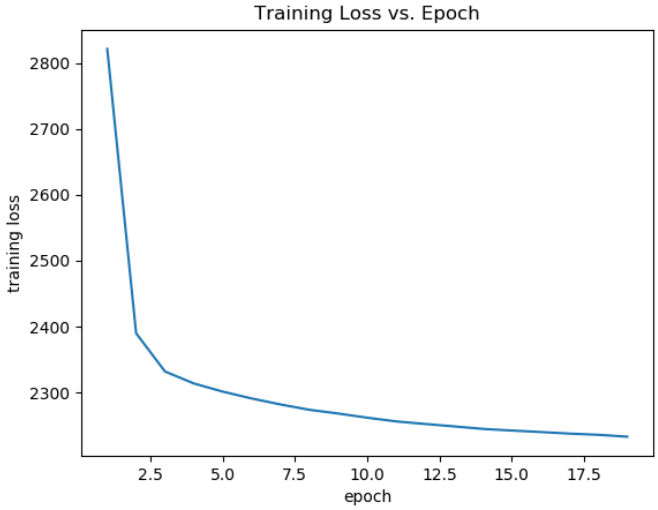

Figure 4: CNN network training loss

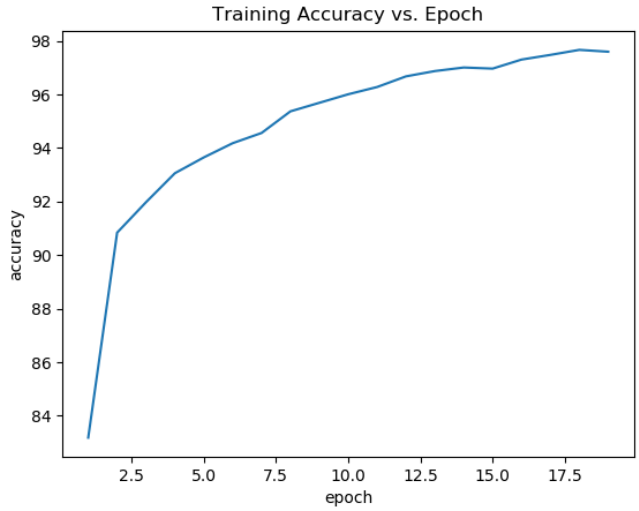

Figure 5: CNN network training accuracy

```
epoch  1: [training loss: 2821.55, validation loss: 615.99, training accuracy: 83.183%]
epoch  2: [training loss: 2389.93, validation loss: 586.89, training accuracy: 90.840%]
epoch  3: [training loss: 2331.93, validation loss: 582.09, training accuracy: 91.977%]
epoch  4: [training loss: 2313.75, validation loss: 577.95, training accuracy: 93.060%]
epoch  5: [training loss: 2301.40, validation loss: 575.14, training accuracy: 93.650%]
epoch  6: [training loss: 2290.99, validation loss: 572.82, training accuracy: 94.181%]
epoch  7: [training loss: 2281.90, validation loss: 571.55, training accuracy: 94.562%]
epoch  8: [training loss: 2273.81, validation loss: 568.48, training accuracy: 95.371%]
epoch  9: [training loss: 2268.12, validation loss: 567.84, training accuracy: 95.690%]
epoch 10: [training loss: 2261.82, validation loss: 566.86, training accuracy: 96.008%]
epoch 11: [training loss: 2256.21, validation loss: 565.89, training accuracy: 96.279%]
epoch 12: [training loss: 2252.30, validation loss: 564.46, training accuracy: 96.677%]
epoch 13: [training loss: 2248.65, validation loss: 563.56, training accuracy: 96.875%]
epoch 14: [training loss: 2244.83, validation loss: 563.44, training accuracy: 97.008%]
epoch 15: [training loss: 2242.33, validation loss: 563.02, training accuracy: 96.967%]
epoch 16: [training loss: 2240.09, validation loss: 561.60, training accuracy: 97.304%]
epoch 17: [training loss: 2237.71, validation loss: 561.31, training accuracy: 97.479%]
epoch 18: [training loss: 2235.86, validation loss: 560.77, training accuracy: 97.667%]
epoch 19: [training loss: 2233.03, validation loss: 560.89, training accuracy: 97.598%]
Accuracy of the network on the 10000 test images: 97.040%
```

Figure 6: CNN network testing accuracy

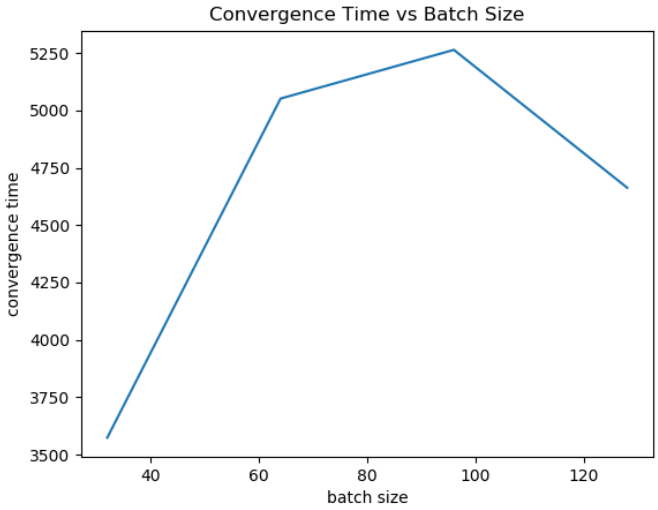

Figure 7: CNN convergence time for different batch size

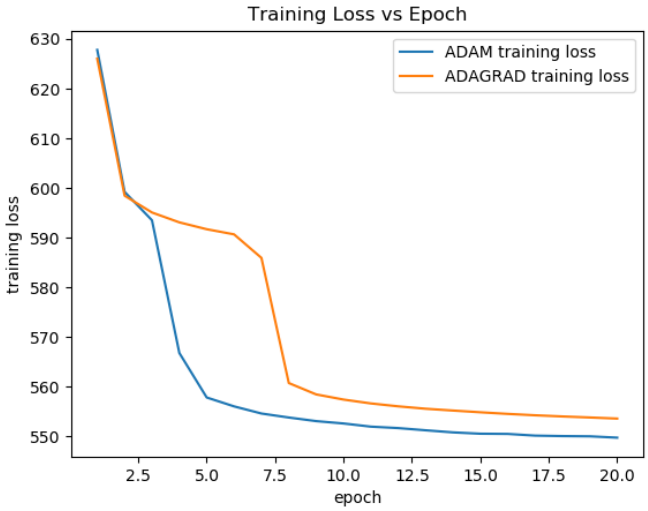

Figure 8: training loss of ADAM and ADAGRAD optimizer