Comparison of Performances of Models on Fashion-MNIST Dataset

Weifan Ma, Zikai Li, Zhenya Liu

Dec 6, 2022

1 Introduction

For decades, the MNIST dataset with handwritten digits has been a classic in the machine learning community. Many factors contribute to its popularity, including its simplicity and accessibility. However, another dataset has recently emerged as an alternative to MNIST: Fashion-MNIST [1].The research group from Zalando, a German fashion commerce company, created this dataset, and wants it to replace the MNIST for reasons such as the original data too overused.

In this project, we will train machine learning models with Zalando's Fashion-MNIST dataset, which contains 70,000 images of fashion that belong to 10 classes. And our goal is to test the performance of these traditional models on this dataset.

For the description of Fashion-MNIST dataset, each sample is a 28×28 gray-scale image with totally 784 pixels. These images are associated with a label from the following 10 classes:

- 0 T-shirt/top

- 1 Trouser
- 2 Pullover
- 3 Dress
- 4 Coat
- 5 Sandal
- 6 Shirt
- 7 Sneaker
- 8 Bag
- 9 Ankle boot

As each image contains a single item of clothing in the same size and color, all images are centered and pre-segmented. As a result, we are able to load and reshape the images into a single color channel. A pixel has one pixel-value (an integer), which ranges from 0 to 255. A higher value indicates more darkness, while a lower value indicates more lightness.

2 Problem statement

We first separate these images into a training set with $60,000$ images and a testing set of $10,000$ images, and then train a Deep Neural Networks (DNN) and a Convolutional Neural Networks (CNN) to compare the results. Our goal is to test the performance of "these" traditional models on this dataset. We are trying to explore the different models' performance for images that are in a good alignment in terms of their accuracies and efficiencies

In addition, we also implement more conventional statistical learning algorithms based on multinomial logit. We use two variants. First, we train multinomial logit models on the raw data. Secondly, we try preprocessing the image data using principal

component analysis before feeding them into our algorithm. We compare the accuracy rates of both approaches as well as how they compare to the neural network models. We are interested in how reductions in the size of the training set affects the relative efficacy of these two broad approaches (neural networks vs. multinomial logit).

3 Informal description of the algorithms

3.1 Deep Neural Networks (DNN)

[2] Linear models predict the output by modeling the predictive probabilities as

$$\hat{y} = \sigma(\phi(x)^T W)$$

Here $\hat{y}$ is the predicted output, x is the input, and W is the parameter matrix to be learned. In this way, $\phi(x)$ helps us select the feature vectors beforehand. To determine what features to use, we use Deep Neural Networks (DNN).
In DNN, we predict the output by modeling the probability as

$$\hat{y} = \sigma(\sigma(x^T W^{(1)})^T$$

Note that σ is the activation function, and here we choose Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

as our activation function. In this step, we have obtained parameter matrices. Our data is trained using DNN by adding an additional layer between the input and output. The nodes between two adjacent layers are fully connected.

Between the input and output layers, we can add additional hidden layers. For simplicity, in this project, we also are just adding one hidden layer.

$$\hat{w} := \arg\min_{w} \left[\ell_{NN}(w) := \sum_{s=1}^{n} \ell(y_s, \hat{y}(x_s; w)) \right]$$

To train our DNN model, we are training the parameter $W = [W^1, W^2]$. Given training examples $(x_1, y_1), \cdots, (x_n, y_n)$, our training result should gives us the optimal parameter $\hat{W}$ that minimizes the training loss, and we choose the square loss l to be cross-entropy loss:

$$\ell(y_s, \hat{y}(x_s; w)) = -\sum_{i=1}^{\kappa} \mathbf{1}(y_s = \text{class } i) \log \hat{y}_i(x_s; w)$$

Since it is a non-convex optimization problem, we choose to use stochastic gradient descent (SGD) to find the local minimum. Since computing the full gradient is very expensive, SGD only selects a set of training data and computes the partial gradient:

$$\ell_{NN;I}\ell(w_t) := \sum_{i \in I} \nabla_w \ell(y_s, \hat{y}_i(x_s; w_t))$$

The complete algorithm of SGD will be:

Input: $\mathbf{w_0}$ (inital parameter); T (number of iterations); $(\eta_t)_{t \geq 1}$, (non-increasing weights in $(0, 1]$); $n_0 \in \{1, ..., n\}$ (size of subsample)

1: **for** $t = 1, ..., T$ **do**:
2: Sample $I \subseteq \{1, \cdots, n\}$ with $I = n_0$ over all subsets of size n_0;
3: $\mathbf{w_t} \leftarrow \mathbf{w_{t-1}} - \eta_t \nabla_w \ell_{NN;I}(\mathbf{W_{t-1}})$
4: **end for**

Output: $\mathbf{w_T}$

3.1.1 Theory of DNN

By adding extra layers to DNN, we can model complex non-linear relationships between inputs and outputs. As a result, DNN can be used to solve the problem of nonlinearity. The extra layers enable the composition of features from previous layers. In the process of training, we use backpropagation. The predictive outputs are compared with the correct values to calculate the loss. By the techniques mentioned in the previous section, we can feed back these errors through the network. Then, the weights are adjusted

accordingly to better predict the outputs. We differentiate the loss function with respect to the nodes in the layers in backpropagation. Therefore, backpropagation can only be applied on networks with differentiable activation functions.

As mentioned, computing the full gradients can be extremely time consuming in DNN. Therefore, we resolved this problem by using the trick of patching. Instead of using all training data in every iteration, we just randomly choose several of them and compute the partial derivatives. It significantly speeds up the computation time and does not affect the result too much.

3.2 Convolutional Neural Networks (CNN)

[3] CNN adds convolutional layers and pooling layers before doing DNN. How they work is as follows. We can think of a convolutional layer as a bunch of filters, each of which captures a pattern of the image. For a filter, we use it as a window sliding on the input. Stride is defined as the step size of each sliding movement. The result of the convolutional layer with input X in $\mathbb{R}^{(c\times a\times b)}$and filters $F_1,...,F_m$ in $\mathbb{R}^{(c\times r\times r)}$ with stride s is

$$X * (\text{filters } F_1,...,F_m) = [X * F_1,...,X * F_m] \in \mathbb{R}^{m\times(\frac{a-r}{s}+1)\times(\frac{b-r}{s}+1)}$$

Then, we are using ReLU to ensure all values are positive. ReLU is defined as $x \to \max(0,x)$. Pooling layer is used to reduce the size of the output from the convolutional layer. We are using max pooling in this project, which keeps the maximum cell value in each patch. A $r \times r$ pooling on $c \times a \times b$ input X with stride s is

$$\text{pool}(X) = \left[\text{pool}\Big(X[d, is : is + r, j : j + r]; o \le d < c, 0 \le i \le \frac{a-r}{s}, 0 \le j \le \frac{b-r}{s}\Big)\right]$$

In this project, we have two convolutional layers and two pooling layers. In each convolutional layer, the size of each filter is 8×8 with stride 1. In each pooling layer, the size of pooling is 2×2 with stride 2. After the convolutional layers and pooling layers, we are basically using the stochastic gradient descent to train our parameters as discussed

in DNN. The overall process of CNN can be shown as:

$$\text{Input} \rightarrow \text{Conv1} + \text{ReLU} + \text{maxpool} \rightarrow \text{Conv2} + \text{ReLu} + \text{maxpool}$$
$$\rightarrow \text{Dense1} + \text{ReLU} \rightarrow \text{Dense2} + \text{Softmax} \rightarrow \text{Output}$$

3.2.1 Theory of CNN

A convolutional neural network consists of an input layer, hidden layers (convolutional layers, pooling layers, and dense layers) and an output layer. [4] [5] We may consider CNN as an input-output mapping. Because of its local-shared special structure, CNN has a unique advantage when it comes to image classification. The layout of the network is similar to that of biological neural networks. As a result of weight sharing, the complexity of the network is reduced. In particular, the images of multi-dimensional input vectors can be directly input into the neural network, which avoids the complexity of data reconstruction in the process of feature extraction and classification. CNN is mainly used to identify displacement, scaling and other forms of distortion invariant two-dimensional graphics. Since the feature detection layer of CNN learns through training data, it avoids explicit feature extraction and learns implicitly from training data when using CNN. Moreover, since the weight of neurons on the same feature mapping surface is the same, the network can learn in parallel, which is also a major advantage of convolutional networks compared with the network with connected neurons.

3.3 Multinomial logit

In this section, we discuss the algorithms for implementing the multinomial logit models [6]. The multinomial logit probability that an observation with a vector of features, x,

is labelled as k, is given by:

$$P(k|x) = \frac{1}{\sum_{j=0}^{k-1} e^{[\theta_j \cdot x/\tau]-c}} \begin{bmatrix} e^{[\theta_0 \cdot x/\tau]-c} \\ e^{[\theta_1 \cdot x/\tau]-c} \\ \vdots \\ e^{[\theta_{k-1} \cdot x/\tau]-c} \end{bmatrix},$$

where τ is a temperature parameter and $c = \max_j \theta_j \cdot x/\tau$. The addition of c ensures the stability of the model. This is the formula for the `probs_func` function in our code.

The cost function is given by:

$$J(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=0}^{k-1}[[y_i == j]]\log\frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=0}^{k-1} e^{\theta_l \cdot x^{(i)}/\tau}}\right] + \frac{\lambda}{2}\sum_{i=0}^{d-1}\sum_{j=0}^{k-1}\theta_{ij}^2$$

We use gradient descent to train our model. Write $\frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=0}^{k-1} e^{\theta_l \cdot x^{(i)}/\tau}}$ as $p_{i,j,\theta}$. Then

$$\frac{\partial p_{i,j,\theta}}{\partial \theta_a} = \frac{x^{(i)}}{\tau} p(y^{(i)} = a|x^{(i)}, \theta)[1 - p(y^{(i)} = a|x^{(i)}, \theta)]$$

The partial derivative of the first half of the cost function $J(\theta)$ is

$$\begin{aligned} &\frac{\partial}{\partial \theta_a}\left[\sum_{j=0}^{k-1}[[y^{(i)} == j]]\log\frac{e^{\theta_j \cdot x^{(i)}/\tau}}{\sum_{l=0}^{k-1} e^{\theta_l \cdot x^{(i)}/\tau}}\right] \\ &= \sum_{j=0, j\neq a}^{k-1}\left[[[y^{(i)} == j]][-\frac{x^{(i)}}{\tau} p(y^{(i)} = a|x^{(i)}, \theta)]\right] + \\ &\quad [y^{(i)} == a]\frac{x^{(i)}}{\tau}[1 - p(y^{(i)} = a|x^{(i)}, \theta)] \end{aligned} \tag{1}$$

The full gradient is

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_a} &= \frac{\partial}{\partial \theta_a}\left[-\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{j=0}^{k-1}[[y^{(i)} == j]]\log p\left(y^{(i)} = j \mid x^{(i)}, \theta\right)\right] + \frac{\lambda}{2}\sum_{j=0}^{k-1}\sum_{i=0}^{d-1}\theta_{ji}^2\right] \\ &= -\frac{1}{\tau n}\sum_{i=1}^{n}\left[x^{(i)}\left([[y^{(i)} == a]] - p\left(y^{(i)} = m \mid x^{(i)}, \theta\right)\right)\right] + \lambda\theta_a \end{aligned}$$

The test accuracy is simply the proportion of the observations in the test set that

are correctly classified using the trained model.

This softmax style multinomial function outputs probabilities for a categorical distribution for the labels in our data. Computationally the algorithm is faster to run than neural networks with more than a few layers. It might also be more robust when the amount of data is not large, something we explore in this project.

4 Results

4.1 DNN's results

We begin our tests by initializing one DNN model.

```
# set the net structure
node_num_list = [784, 500, 200, 10]
active_obj = Sigmoid
loss_obj = CrossEntropy()
opt_obj = SGD
# init the net
net = DNN(node_num_list, active_obj, loss_obj, opt_obj, name="DNN")
\\
```

Listing 1: Initialize DNN

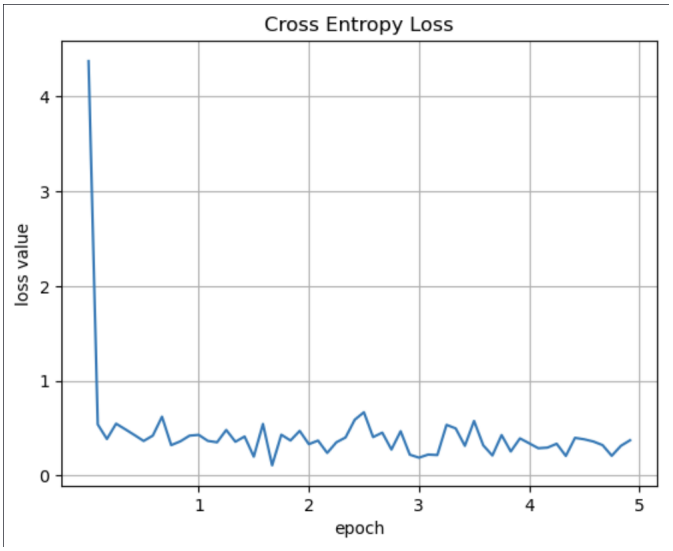
Here we train our model with 5 epoch, batch size is 50, and 500 test number, and get following results:

```
1200it [00:08, 139.43it/s]
epoch: 1, loss: 0.6083
train data accuracy: 91.0%
test  data accuracy: 78.0%
1200it [00:11, 103.79it/s]
epoch: 2, loss: 0.4010
train data accuracy: 90.0%
test  data accuracy: 82.0%
1200it [00:10, 117.46it/s]
```

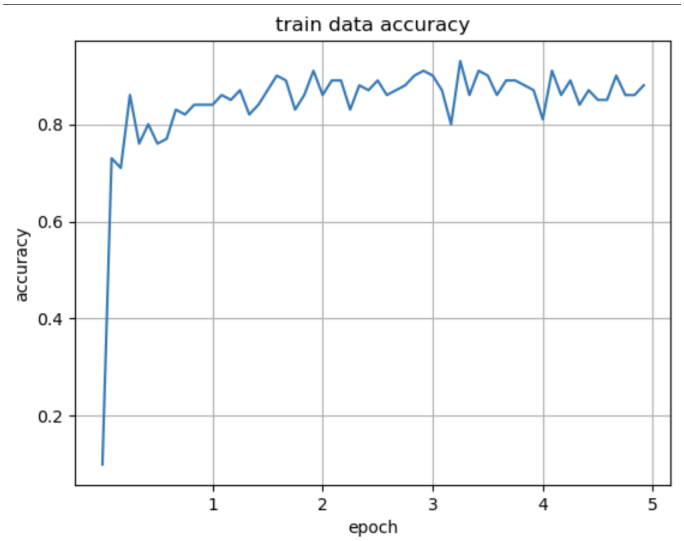
```
epoch: 3, loss: 0.25555
train data accuracy: 88.0%
test  data accuracy: 87.0%
1200it [00:09, 120.05it/s]
epoch: 4, loss: 0.25289
train data accuracy: 84.0%
test  data accuracy: 86.0%
1200it [00:11, 105.50it/s]
epoch: 5, loss: 0.25085
train data accuracy: 88.0%
test  data accuracy: 85.0%
```

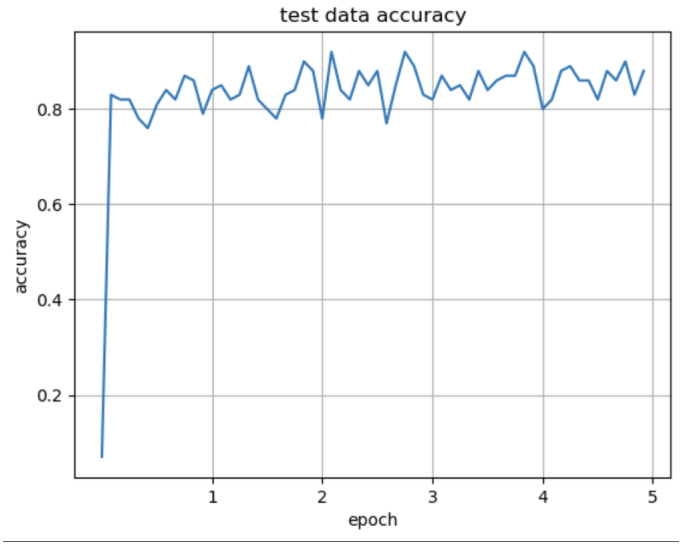
It is reasonable to check that generally train data accuracy (average is 88.2 %) is higher than test data accuracy (average is 83.6 %). The average time cost for each epoch is about 10 seconds in this case.

The cross entropy loss of DNN is shown below:



The training and testing accuracy are shown below respectively:



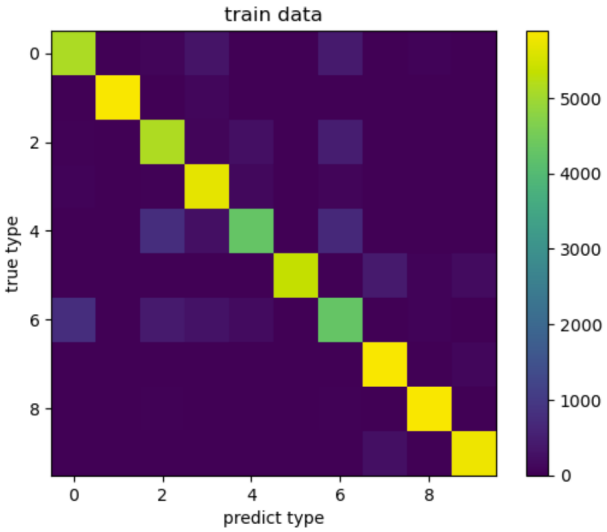


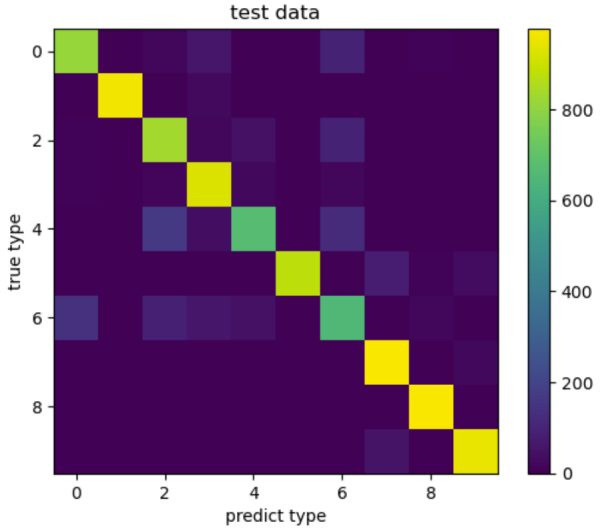
We also collect the accuracy for individual label:

```
type T-shirt/top: train accu=85.18%, test accu=81.20%
type Trouser    : train accu=97.77%, test accu=96.20%
type Pulloverm  : train accu=85.68%, test accu=83.20%
type Dressm     : train accu=94.48%, test accu=92.50%
type Coat       : train accu=71.57%, test accu=66.80%
type Sandal     : train accu=89.05%, test accu=87.60%
type Shirt      : train accu=71.38%, test accu=65.00%
type Sneaker    : train accu=97.97%, test accu=97.50%
type Bag        : train accu=98.15%, test accu=97.70%
type Ankle boot : train accu=95.92%, test accu=94.70%
```

The train and test accuracy for Coat and Shirt are significantly worse than other types.

Translating this data into the confusion matrix, it looks like:





4.2 CNN's results

To test the dataset with our CNN model, first initializing one as below:

```
# set the net structure
conv_size = [8, 8]
pool_size = [2, 2]
linear_size = [100]
active_obj = Sigmoid
loss_obj = CrossEntropy()
opt_obj = SGD
```

```
# init the net
net = CNN(conv_size, pool_size, linear_size, active_obj, loss_obj, opt_obj,
    name="CNN")
net
```

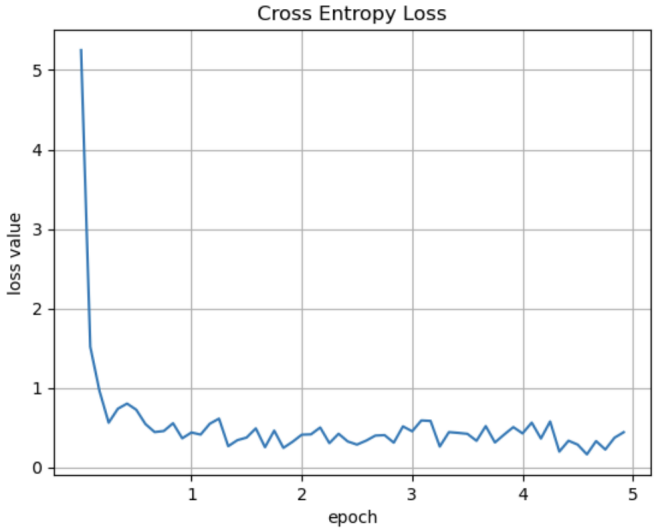
Listing 2: Initialize CNN

Here again we train our model with 5 epoch, batch size is 50, and 500 test number, and get following results:

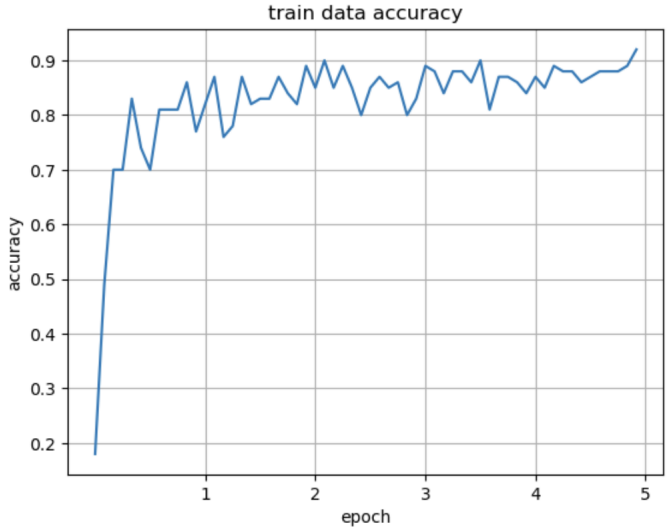
```
1200it [02:07,  9.42it/s]
epoch: 1, loss: 0.7886
train data accuracy: 91.0%
test  data accuracy: 82.0%
1200it [02:08,  9.32it/s]
epoch: 2, loss: 0.4571
train data accuracy: 85.0%
test  data accuracy: 86.0%
1200it [02:04,  9.62it/s]
epoch: 3, loss: 0.25951
train data accuracy: 92.0%
test  data accuracy: 83.0%
1200it [01:55, 10.257it/s]
epoch: 4, loss: 0.25611
train data accuracy: 88.0%
test  data accuracy: 85.0%
1200it [02:23,  8.37it/s]
epoch: 5, loss: 0.25398
train data accuracy: 86.0%
test  data accuracy: 83.0%
```

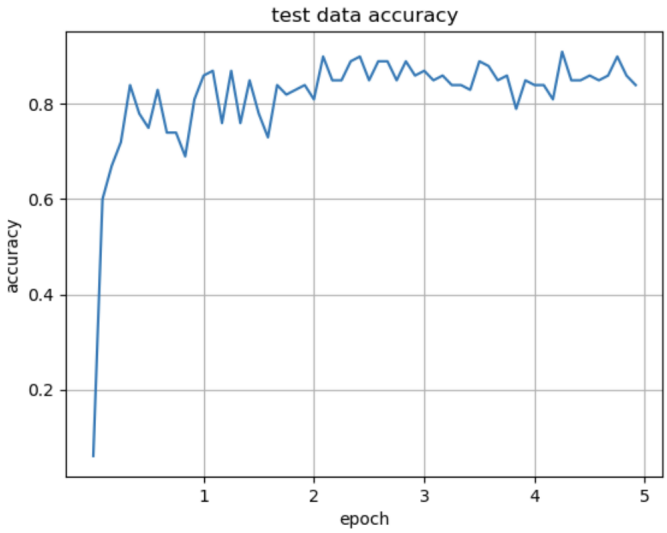
Similarly to training results of DNN, the train data accuracy (average is 88.4 %) is generally higher than the test data accuracy (average is 83.8 %). The average test accuracy for DNN is slightly higher than the average test accuracy for CNN. However, the average time cost for CNN is about 120 seconds, which is 12 times of the time cost for DNN.

The cross entropy loss of CNN is shown below:



The training and testing accuracy are shown below respectively:

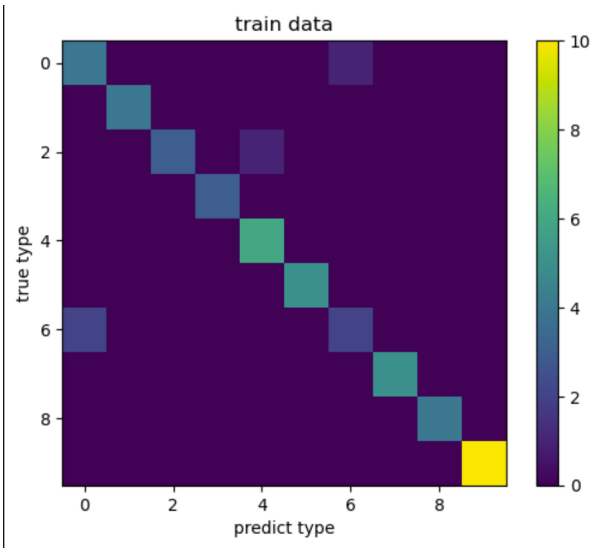


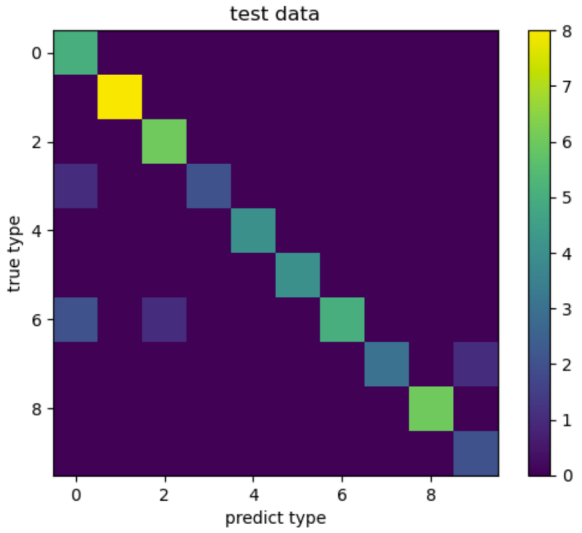


Similarly, we collect the CNN accuracy for individual label:

```
type T-shirt/top: train accu=90.60%, test accu=88.50%
type Trouser    : train accu=97.70%, test accu=96.90%
type Pulloverm  : train accu=76.47%, test accu=73.70%
type Dressm     : train accu=86.60%, test accu=86.00%
type Coat       : train accu=85.08%, test accu=84.00%
type Sandal     : train accu=95.07%, test accu=94.70%
type Shirt      : train accu=63.75%, test accu=59.20%
type Sneaker    : train accu=96.22%, test accu=96.60%
type Bag        : train accu=97.97%, test accu=98.10%
type Ankle boot : train accu=96.25%, test accu=95.80%
```

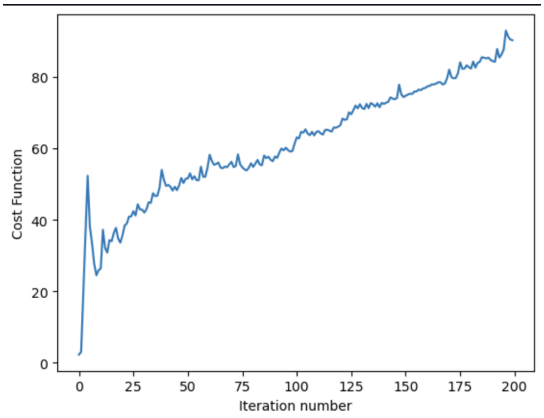
Translating this data into the confusion matrix, it looks like:

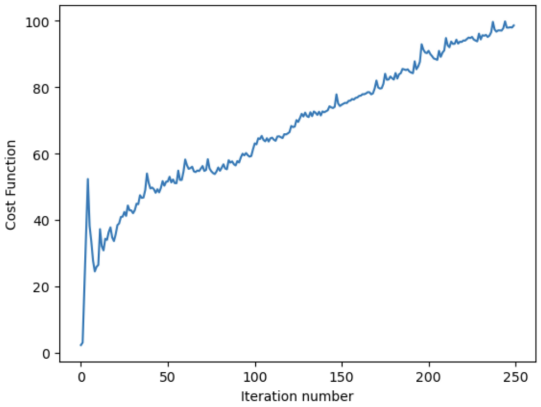


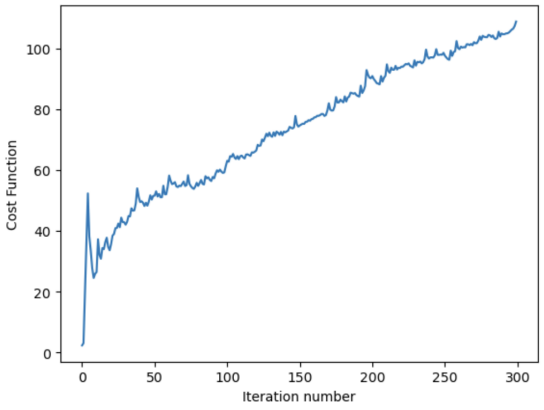


4.3 MNL's results

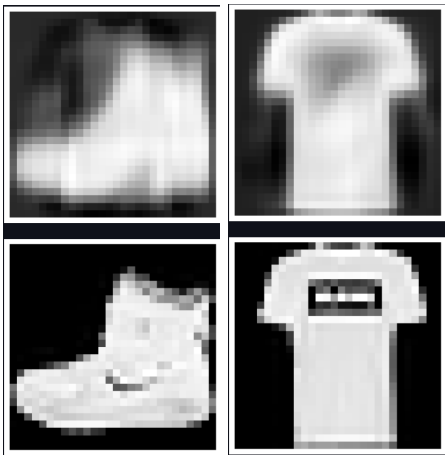
Test errors after 200 iterations, 250 iterations, and 300 iterations are shown below respectively:


Cost Function
Iteration number
0
20
40
60
80
0
25
50
75
100
125
150
175
200

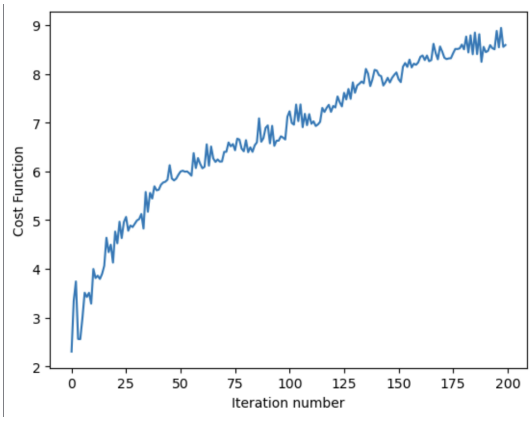

Cost Function
Iteration number
0
20
40
60
80
100
0
50
100
150
200
250

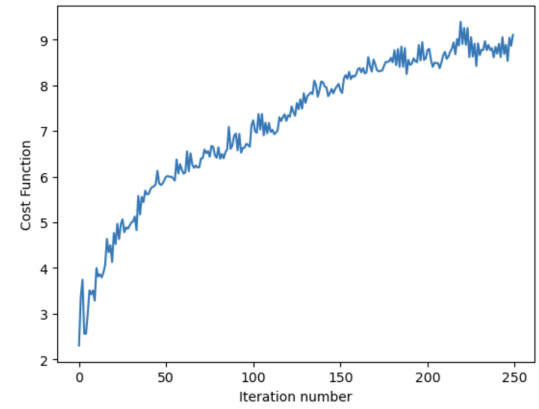

Cost Function
Iteration number
0
20
40
60
80
100
0
50
100
150
200
250
300

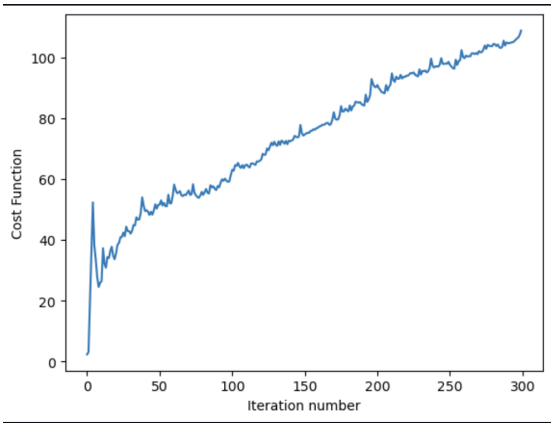
Use the reconstruct PC to show the first and second MNIST images using the 20 first principal components. The second image is the original in the data set.



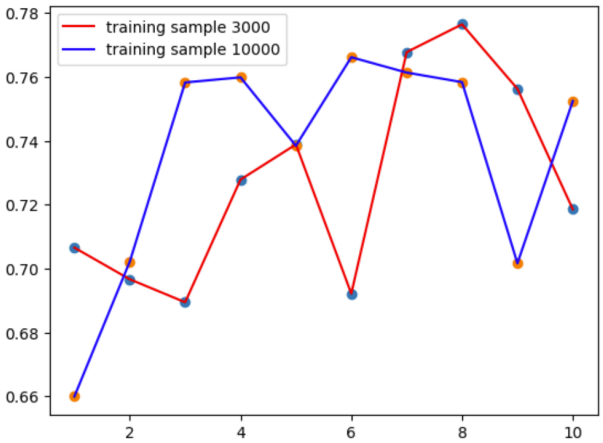
With PCA, Test errors after 200 iterations, 250 iterations, and 300 iterations are shown below respectively:







We are also interested in the performance of MNL with a small training set. Here we train this model with random 3000 and 10000 samples from the original training sets, and test it with the whole test set. This test is independently processes in 10 times, with the result shown below:



The graph shows that there is no big difference on performance of CNN when the

training sample size is small, and the test accuracy is unstable with an value between 66 % and 78 %.

For the multinomial logit model, using a temperature parameter (τ in our equations above) of 0.5, $\alpha = 0.2$, $\lambda = 1e-4$, and 300 iterations, we were able to train a model that correctly predicts about 80% of the labels in our test set. This is significantly lower than the test accuracy we get using CNNs or DNNs.

We also tried preprocess our data with principal component analysis (PCA) before feeding them into our algorithm. We used the first 20 projections and the same settings for the multinomial logit algorithm as above. This resulted in a slight reduction of the accuracy to 70%. But with smaller data sets, preprocessing data using PCA might be useful for guarding against noise in the data that algorithms sometimes inevitably pick up in the training process.

5 Conclusion

In this project, we train different models to classify observations in the fashion image data set. We firs train a Deep Neural Networks (DNN) and a Convolutional Neural Networks (CNN), two approaches that have proved effective in obtaining high accuracy rates in classifying images across various domains. As a reference point, we also implement more conventional statistical learning algorithms based on multinomial logit and investigate how much more accurate the neural networks can be compared to conventional approaches.

It is challenging to compare these different classes of models since different specifications, especially different settings of meta-parameters, can return different results. But we believe our approaches roughly approximates the maximum accuracy that is reachable by these different classes of models. Overall, the DNNs and CNNs are more accurate than the MNL models, but not by much (83 85% vs. 80%). The neural network models, however, are much more computationally expensive than the MNL models.

We also know that DNNs and CNNs are data-hungry, i.e., their accuracy drops quickly in the number of data points in the training set. We did not have a strong prior about how data-hungry MNL models can be. We test empirically using the data set. We train this model with independent draws of 3000 and 10000 data points from the original training set. The accuracy rates drop slightly but not by much. Thus, the MNL model appears to be robust to a reduction in the size of the training set. In general, based on our results, we recommend using CNNs and DNNs when computational power and the size of training sets are secondary considerations, but would recommend data scientists consider MNL-based models when computational power and/or the size of the training set are limited.

On the other hand, we can see that as the value of epoch increases, the loss decreases. DNNs will result in a slightly better confusion matrix than CNNs. DNNs has good performance on images that are well-aligned. And CNNs usually has a better performance when dealing with noise and padding, but this ability is not clearly shown in our project. One of the main reason is due to initial setting for such epoch numberïŒ batch size and test number. Though the overall accuracies are sufficiently good, we can see our models are prone to predict some item incorrectly (e.g. pullover with accuracy only around 80 percent). We deduce that it might be attributed to the fact that they have similar shapes with other items.

In the future, we will try to increase their accuracy by training with more representative data images or other methods that we will research on. We will increase the value of entropy, our model will have better performance, also this process will result in much more processing time. And we will also show that CNNs has better performance as its been well-trained.

References

[1] URL: https://www.kaggle.com/datasets/zalando-research/fashionmnist?resource=download.

[2] URL: https://en.wikipedia.org/wiki/Feedforward_neural_network.

[3] URL: https://en.wikipedia.org/wiki/Deep_learning.

[4] URL: http://deeplearning.stanford.edu/tutorial/.

[5] URL: https://www.researchgate.net/publication/331540139_A_State-of-the-Art_Survey_on_Deep_Learning_Theory_and_Architectures.

[6] Dan Jurafsky. "Chapter 7 Logistic Regression". en. In: *Speech and Language Processing*. Unpublished manuscript. URL: https://www.semanticscholar.org/paper/Speech-and-Language-Processing.-Chapter-7-Logistic-Jurafsky/dda48d8a91f7927c27d6e76627221a2a0cf74968 (visited on 12/01/2022).