Deep Learning and Neural Networks – Multilayer Perceptron and CNN

Qing Zhou[1]

[1]Leiden University, PO Box 9513, NL-2300 RA Leiden, the Netherlands

1 Introduction

The motivation of this assignment is to get familiar with Multilayer Perceptrons (MLPs) and Convolutional Networks (CNNs), and their advantages respectively, by constructing models using Tensorflow to solve classification problems. This report is laid out as follows: in section 2 we introduce the basics of the MLPs and CNNs, and details of parameter optimisation, by testing different models on the two well-known datasets: MNIST – a handwritten digit database, and Fashion_MNIST – a dataset of Zalando's article images.

1.1 MLPs and CNNs

A MLP is a feedforward network, which consists of at least three layers: an input layer, a hidden layer and an output layer. It utilises backpropagation, and activation is non-linear. A CNN is a network which normally takes input images and assign weights to different aspects of the image, in an attempt to distinguish one from another. The architecture of the CNN is inspired by that of the visual cortex, where individual neurons respond to stimuli only in a restricted region. Unlike MLPs, CNNs by default has spatial and temporal dependencies, which is why it is superior in image classification tasks.

2 The Basics of Keras and Tensorflow

The corresponding python files for this section are
'task1_main.py','task1_CNN_plot.py','task1_DNN_plot.py'.

To get an idea of the impact of changing different hyperparameters and network architectures, we set the network as listed in Table 1. To test the impact of each parameter, we allow one to vary each time, and the others are fixed as their default settings. Fig. 1 shows the overall results of different runs implementing the MLP networks, giving a total of 40 runs, demonstrating the impacts of changing 6 different hyperparameters(architectures). It can be seen from Fig. 1, when changing the learning rate, the loss rate decreases with the decrease of η, for both of the datasets. The optimum learning rate is 0.001. A model of 3 layers, each having 32 neurons turns out to be best suited for solving our classification problem, for both datasets. Moreover, increasing the number of neurons improves the shape of the loss function greater than increasing the number of layers (comparing layer $[16, 16, 16]$ with $[32, 32]$). The best optimiser, for both datasets with MLP networks, is the Adam optimiser, which implements a stochastic gradient descendent algorithm. When changing the activation function, it is found both 'relu' and 'selu' perform well on the two datasets, with 'selu'

Table 1: Default hyperparameters and model architectures. Unless otherwise specified, in the parameter optimisation part we always allow only one to change and fix the rest. Other choices for paramters are: (learning rate) $\eta = (0, 1, 0.01)$; Layer = $[16, 16], [16, 16, 16]$; Optimiser = 'uniform'; Activation = 'softmax'; Regulariser = 'l1','l1_l2','l2'. For detailed documentation please refer [2].

Network	Dataset	η	Layers	Optimiser	Activation	Initialiser	Regulariser
MLP	MNIST	0.001	32-32-32	Adam	selu	xavier	L1L2
	FaMNIST	0.001	32-32-32	Adam	selu	truncated normal	L1L2
CNN	MNIST	0.001	32-32	Adam	relu	xavier	L1L2
	FaMNIST	0.001	32-32	Adam	relu	xavier	L1L2

having a slight advantage. Only in initialiser the optimum solution differs for the two datasets. Truncated normal is best suited for the fashion dataset while xavier is the optimum solution found for the MNIST dataset. The best solution for the regulariser is L1L2 for both of the datasets. From Fig. 2 we can learn that, in overal, a CNN converges in a shorter epoch, as compared to an MLP network. Likewise for the MLP network, the best suited learning rate is decided to be 0.001. The model architecture however, differs. With a CNN we need fewer layers to reach the same accuracy as delivered by an MLP network. Here the best solution is found to be using 2 layers each having 32 neurons. The best solution for the optimiser is in this case the same as before, the Adam optimiser, and also applies for both of the datasets. For a CNN it is found there is a slight advantage when changing the activation function from 'selu' to 'relu', unlike the cases using MLP network. The initialiser and regulariser are both found to be the same for the two datasets, their best solutions are: xavier and L1L2. After finding the sets of best solutions, we set the default parameter setting as the best solutions and trained the networks again. Thus our best solution happens to be that listed in Table 1.

3 The impact of obfuscating data

The corresponding python file for this section are 'task2_plot.py'.

To test the performance of the above two networks, we reset the training process with obfuscated data. To obfuscate the datasets we first flatten the input images, and random permutating the pixels with fixed seed, and then reshape the images back to their original ones. To get an idea of how the obfuscated inputs look like, Fig. 3 shows two images from both datasets are selected, with their pixel-permuted image on the right. Since CNNs involve convolutional kernels which perform a dimension reduction on the input data, the details of the input(images) are thus combined as clusters and then feed onto the network; whereas MLP networks, in the inputs can be flattened. It can be expected, with obfuscated data, the MLPs should not be greatly affected, while CNNs should be must worse. Fig. 4 shows the loss function for both datasets, with MLP networks. Fig. 5 shows the loss function for both datasets, with CNNs. By comparing the gaps between the loss function for the raw data and for the obfuscated data, it can be seen, with obfuscated data, indeed the CNN is much worse, with accuracies dropping from 98.44% to 95.89% for the MNIST dataset, and from 90.43% to 84.70% on the fashion MNIST dataset, a relative error of 2.59% and 6.33%, respectively. MLP performs as expected, with accuracies dropping from 96.63% to 96.23% for the MNIST dataset, and from 84.92% to 82.80% on the fashion MNIST dataset, a relative error of 0.41% and 0.25%, respectively.

4 A "Tell-the-time" network

The corresponding python files for this section are
'task3_hour.py','task3_minute.py','load.py','model.py','model1.py','task3_plot.py'.

To construct a model which can "tell-the-time", we have experimented with different model architectures. At first we used a simple 4 layer CNN network. After trials of various possible solutions, none of them is able to deliver acceptable error in both hour and minute. Thus we decided to train the network separately, one for deciding the hour and the other for deciding the minute. In the case with both hour and minute, the actual loss function should be a combination of the two, with a weighting parameter deciding whose faction of impact is larger on the loss. With separate networks this is no longer a problem. The top panels of Fig. 6 shows the loss function of the two network, with the left one on the training sets, and the right one on the test set. Both networks are allowed to train within 100 epochs. It can be seen, the best training result is 25 epoch, after taking both networks into consideration (see top right panel). After combining the predicted time (hour and minute) given by the two networks, a total error can be calculated. The middle left panel shows the predicted time vs. the actual time. It can be seen, apart from the correctly predicted ones (located along the diagonals), there are two line features emerging, which are parallel to the diagonal. This behaviour can be seen more clearly from the middle right panel, which is the histogram of the total error. The secondary peaks at ~ 50 (minutes) corresponds to the two lines parallel to the diagonal. It means about $\sim 30\%$ percent of the test samples have an error of ~ 50 minutes. By looking at the errors from hour and from minute separately, we get a better clue where this secondary peak comes from: it is mostly from the error in hour (see the second bar in the bottom left histogram). About $\sim 30\%$ of test samples have an error of 1 hour. This leads to a total mean error of about 30.5 minutes. And lastly, the error in minute is well behaved, with a mean value of 7.56 minutes.

5 Conclusion

We have learnt the basics of MLPs and CNNs, by training the networks on two simple datasets to solve a classification problem. We have understood the impacts of various parameters on the performance of the networks, by choosing different sets of parameters. We have gained a deeper understanding of the CNN, by implementing it on an "hour-minute" time dataset. It it found the average error in minute is 7.56 minutes, and the mean total error is found to be 30 minutes. The secondary peak in the error distribution in minute comes mainly from the samples such as $1:55$. So by looking solely at the hour, the network tends to predict a "2", rather than a "1", thus leading to a large total error. To reduce the total error we can go back to what we have proposed in the beginning of the previous section. Since we have well behaved results on predicting the minute, we can first train the network to decide minute only, until it converges. Then build a loss function which take into consideration of both hour and minute, but assign a weight on each contribution. And by allowing the minute to "correct" the prediction on the hour, we can have a chance to reduce the total error.

References

[1] A. Géron, Hands-on Machine Learning With Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. Sebastopol, CA, USA: O'Reilly, 2017

[2] Tensorflow: Keras, `https://www.tensorflow.org/api_docs/python/tf/keras/`

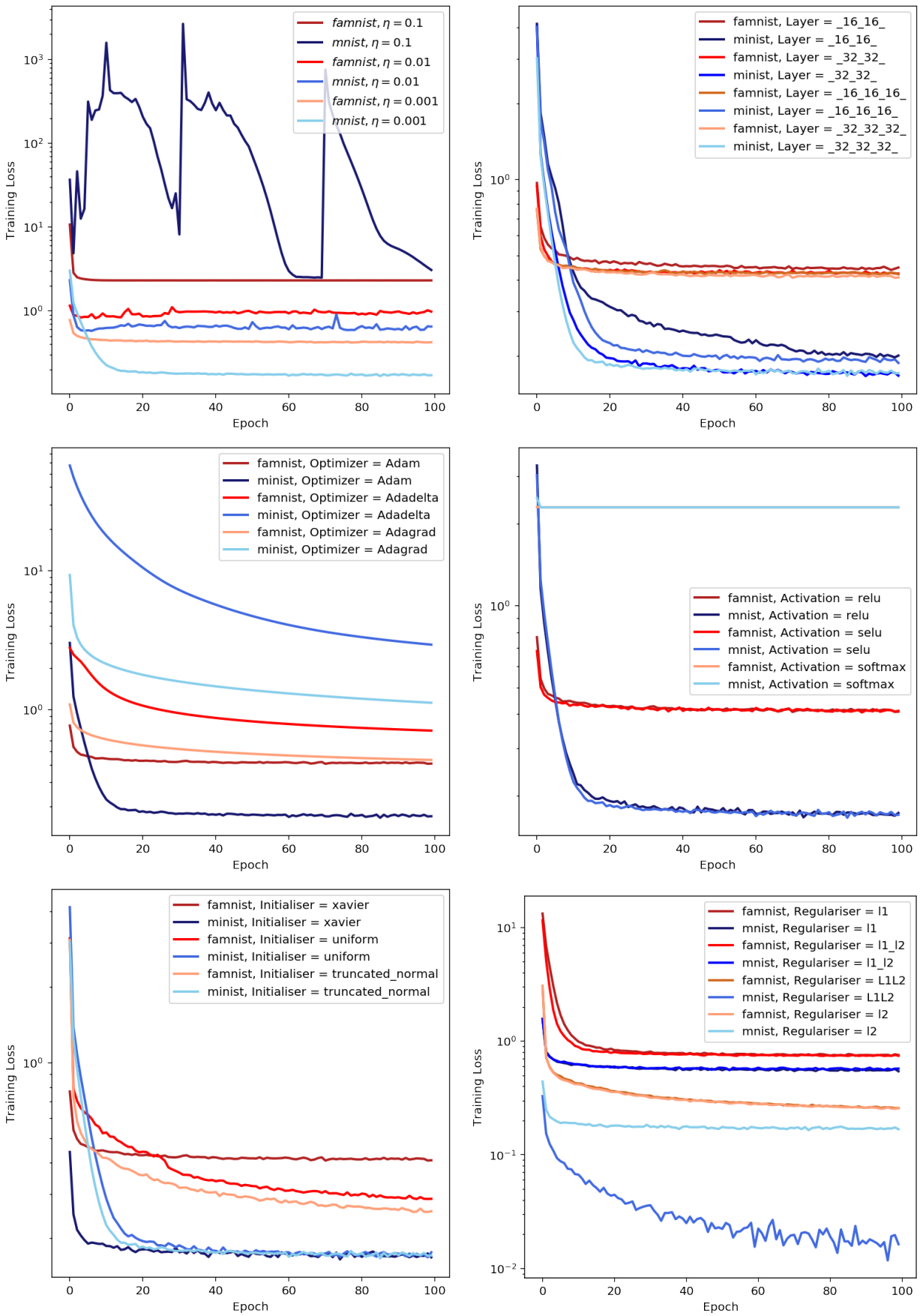

Figure 1: Loss function for 40 runs implementing the MLP network, experimenting on two datasets: MNIST and fashion MNIST, with each subpanel representing the effects of changing one parameter. Each run is allowed for 100 epochs. Top left: the impact of changing the learning rate η; top right: the impact of changing the model architecture; middle left: the impact of changing the optimizer; middle right: the impact of chaning the activation function; bottom left: the impact of changing the initialiser; bottom right: the impact of changing the regulariser. In each subfigure the red lines are runs using the fashion MNIST dataset, and the blue ones are using the MNIST dataset.

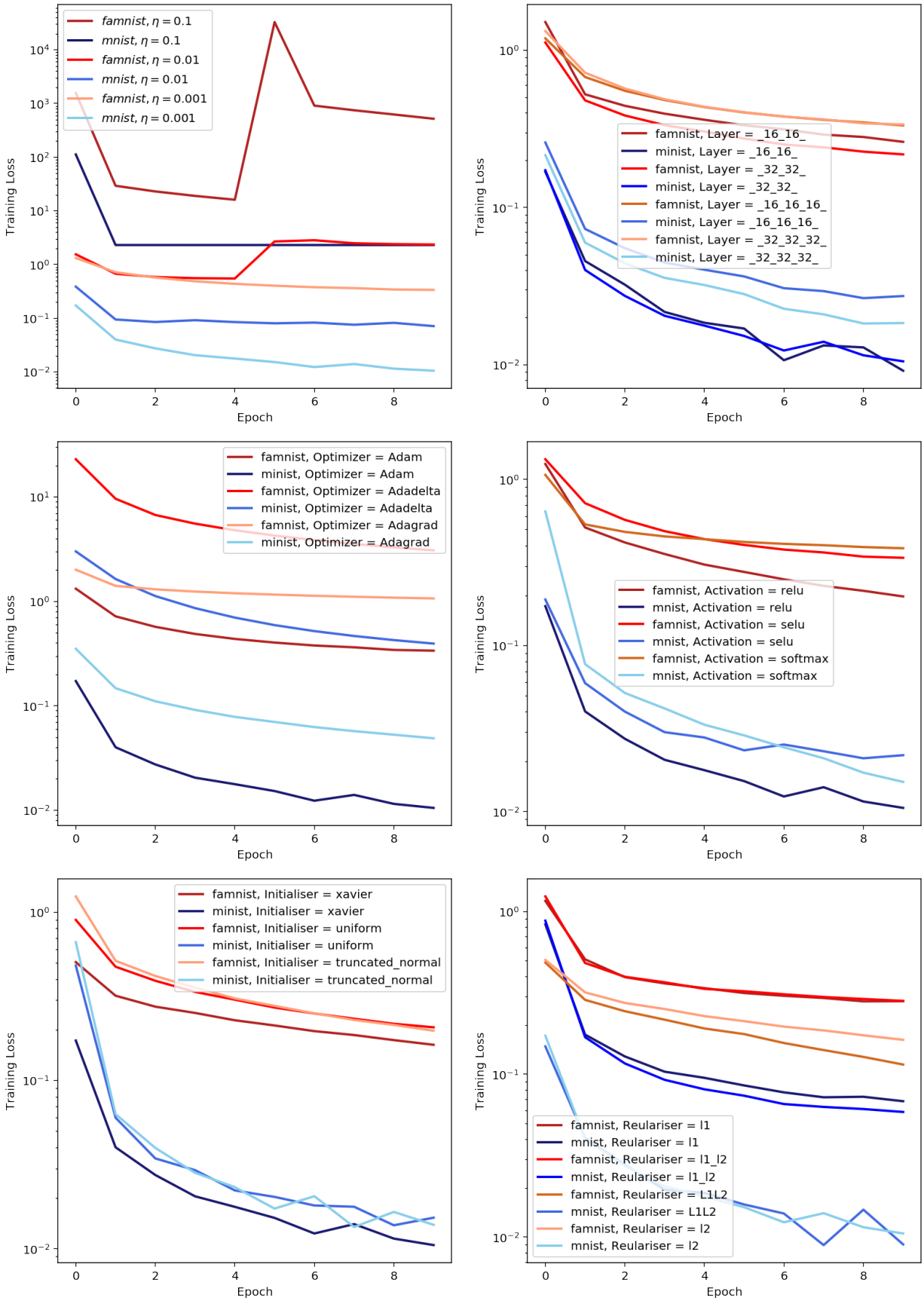

Figure 2: Same as Fig. 1, but now implementing the CNNs. Each run is allowed for 10 epochs.

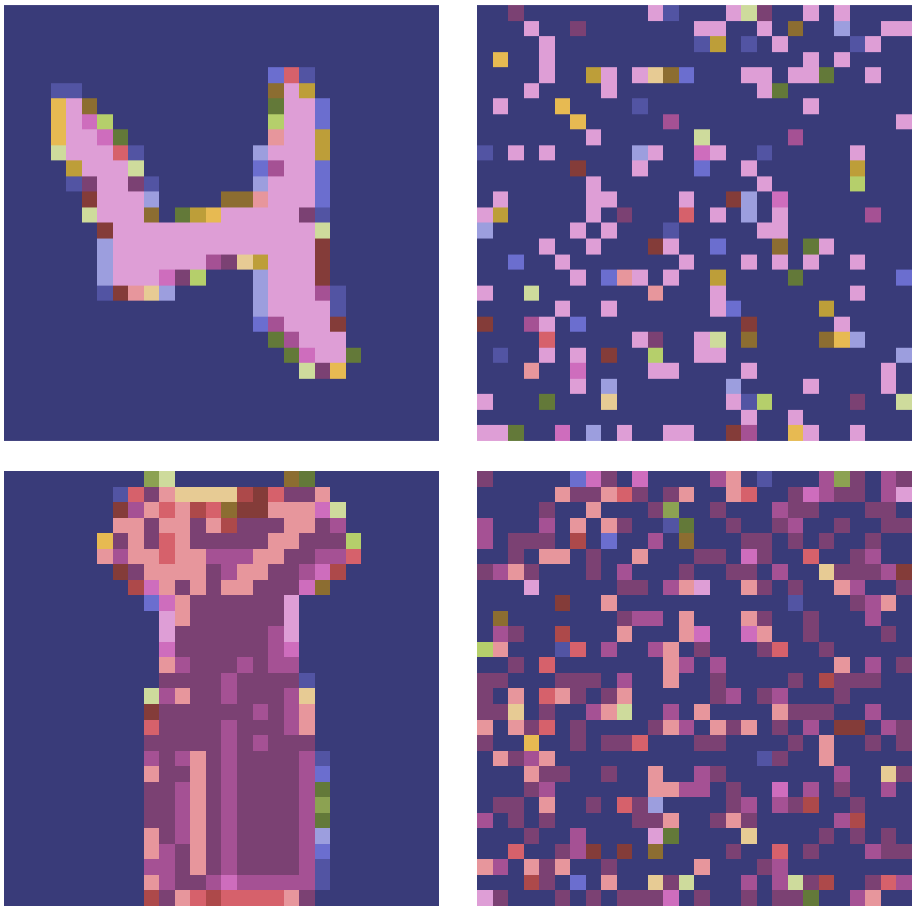

Figure 3: The original images from both datasets, plotted against their pixel-permuted images. Left column: original images; Right column: permuted images.

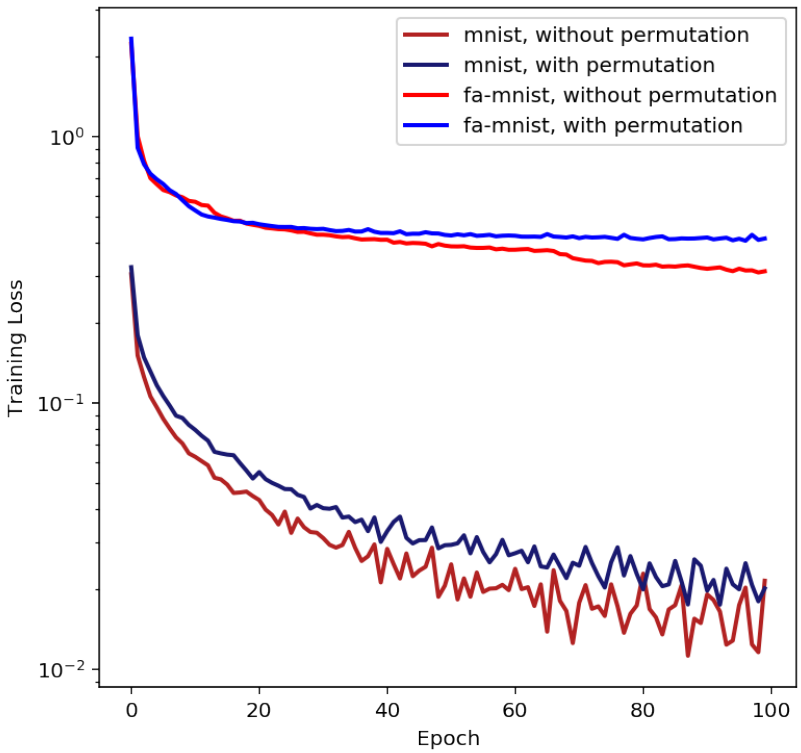

Figure 4: Loss function with MLP network. The red lines represent the loss functions with raw data, and the blue lines represent the loss functions with the obfuscated data.

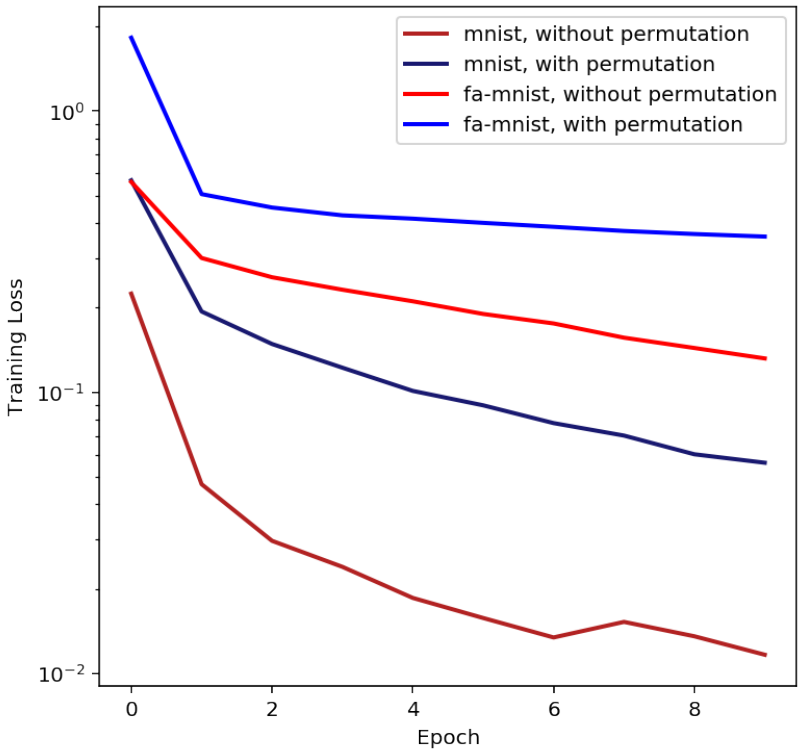

Figure 5: Loss function with convolutional network. The red lines represent the loss functions with raw data, and the blue lines represent the loss functions with the obfuscated data.

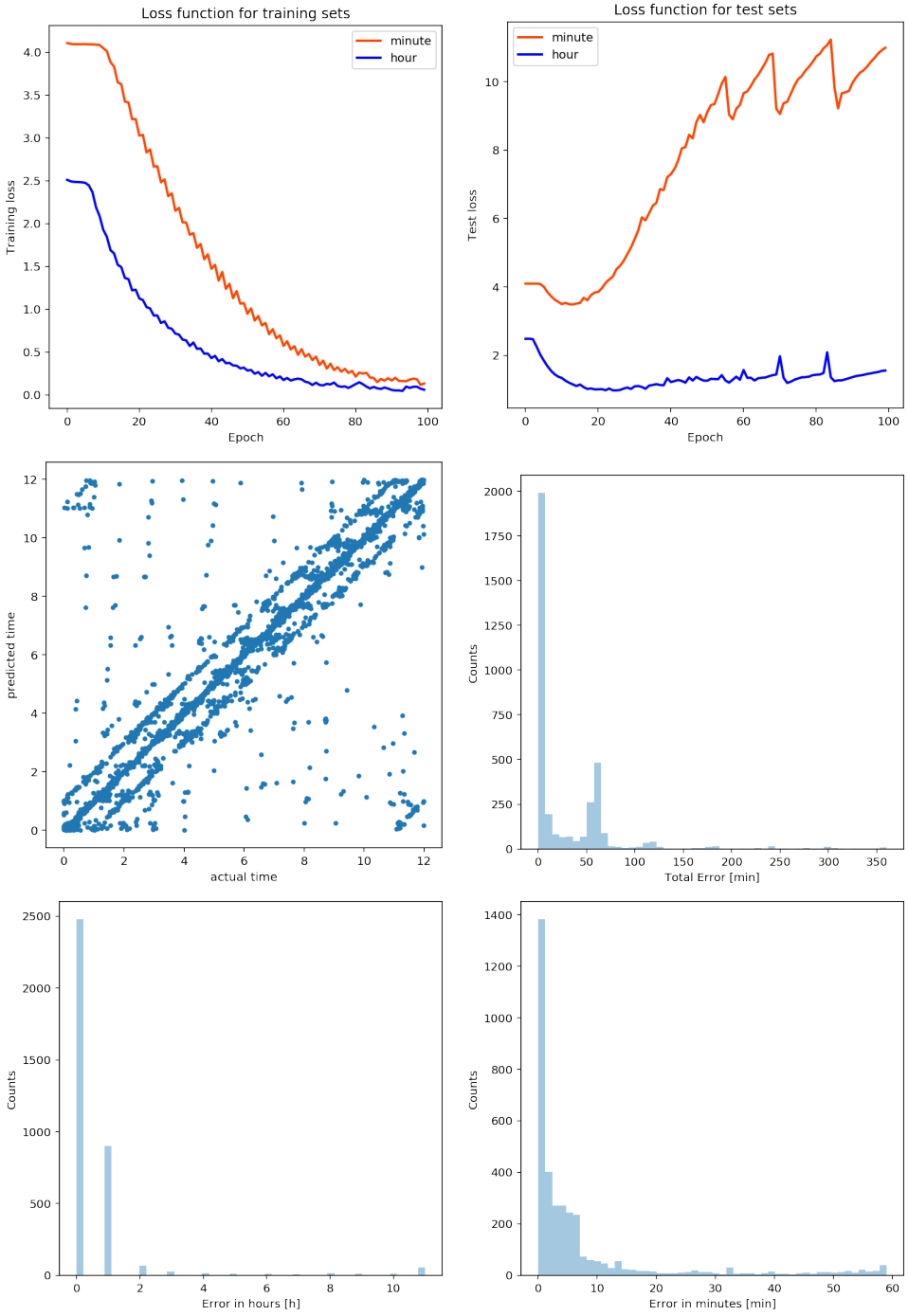

Figure 6: "Tell-the-time" network. Top panel: the loss function on both the training sets and test sets, for both of the networks (hour and minute). The blue lines represent the loss functions for the hour network and the red lines represent the results of the minute network. Middle panel: the predicted time plotted against the actual time; the total error distribution. Bottom panel: the distribution of the error in hour; the distribution of the error in minute.