Report of Deep Learning Training on MINST Dataset

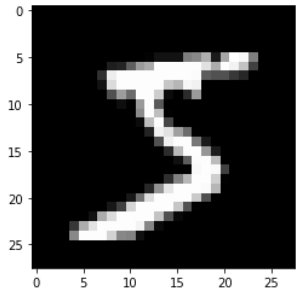
Wenhui Fang
GR 5241 Statistical Machine Learning
Department of Statistics
Columbia University
05/08/2022

Introduction

The MINST dataset containing handwritten digits is one of the most commonly used dataset for training various image processing systems and machine learning algorithms. It is composed by two parts: training examples and test set. The training examples have 60,000 cases, and test set have 10,000 cases. The digites have been size-normalized and centered in a fixed size image. In this report, we construct many neural network models to train with MINST dataset to analyze the difference of these models with different parameters and initializations.

Data Processing

1. Examples of Training Set.



   For this graph, the corresponding label is 5. It matches with the label 5.

2. Dimension of the set and Normalization
   The dimension of of X_train is (60000, 28, 28), which contains 60000 examples of images of (28,28) shape. Y_train set is composed by 60,000 numbers. They has been normalized by using sklearn preprocessing function.

3. One Hot Encoding
   We noticed that Y take values of integers, but we are dealing with a classification problem. Therefore, one-hot-encoding allows us to change numerical values for computers to learn. It prevents the possibilities that computer treat Y as integer values and effects generated by higher values. For example, the class labeled with 6 does not has more wights than class labeled with 1.

Deep Learning Models

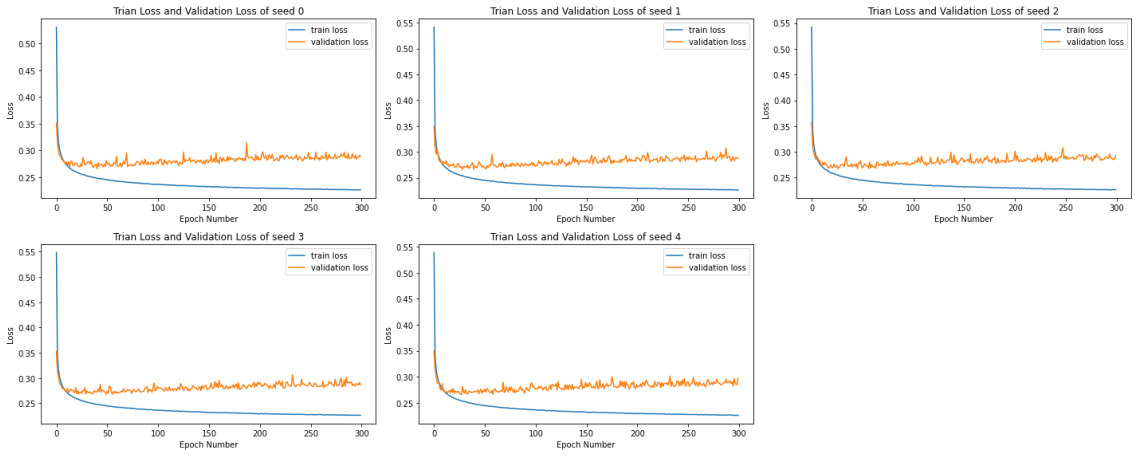
In this part, we construct three neural network models to compare the performance. For all models, the loss function for gradient descent is cross entropy function, and the metric is accuracy. Labels are processed by one-hot-encoding for reason listed in previous part. For each model, to try different initialization on weights, we train each model 5 times with different random seed and at least 150 epoch for sign of overfitting. Also, to analyze the difference of parameters, all models are also trained with different learning rate and momentum.

1. One Layer ANN:

   In this part, we will construct an artificial neural by using one layer with 100 hidden layers. We first dense it to 100 unites, and dense it to 10 units for output with softmax function, since we have 10 classes. This ANN model are trained with 300 epochs with learning rate 0.1. The architecture is below:

```
ann = Sequential([keras.layers.Dense(100),
                  keras.layers.Dense(10, activation="softmax")
                  ])
```

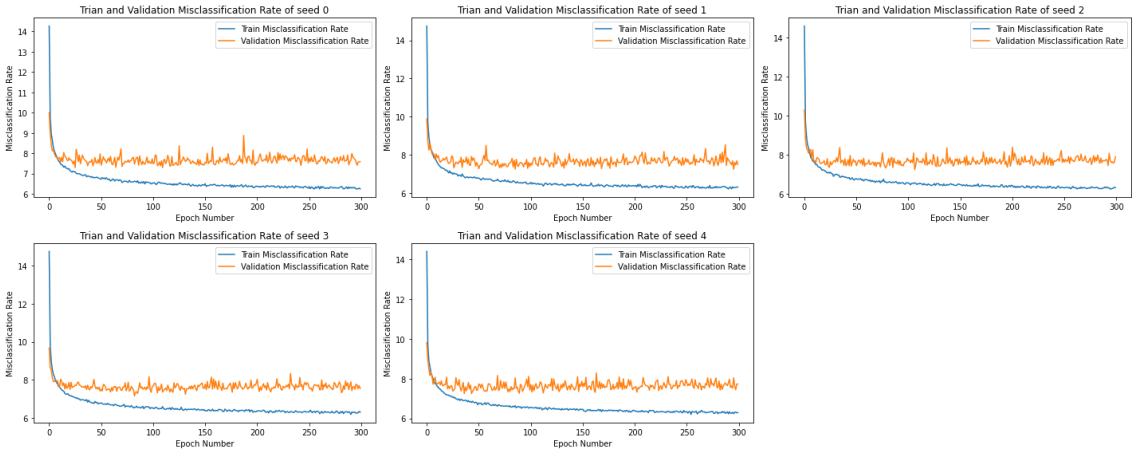
   (a) Training and Test Loss Comparison



   By comparing five graphs, the overall shape and trend are the same, and different initial weights does not have much effects on model performance. If we zoom in a bit on y-axis, the start point of loss is slightly differed from 0.52 to 0.55, which is also a small gap. Furthermore, we can see that the training loss convergent at a fast rate, it drops to 0.2 from 0.5 within 20 epochs and decrease steady in the rest of epochs. For the validation loss, it starts at about 0.35 which is smaller than

training loss, and stay higher than training loss. It is reasonable that the training loss is lower than validation loss because the model are trained on training set and it adjust the weights to more fit on that set. Therefore, the training loss will continue to decrease, but the validation loss will stay on if the model stopped to constructively improve itself. It also has an obvious upward trend after about 150 epochs. Thus, we conclude that one-layer ANN model is overfitted after about 150 epochs. Overall, to compare these five models, we sum up validation loss, which is the same meaning of using average validation loss, and select model with seed 4 for best model with the least validation summation loss for 84.4941. The best model has accuracy of 92.27%

(b) Training and Test Misclassification Rate

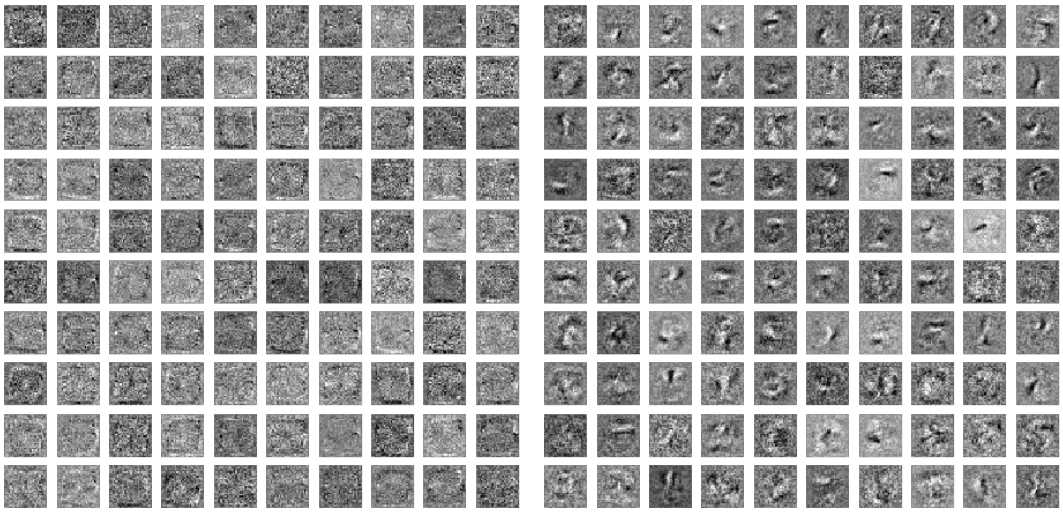


In the graph for five training examples, if we evaluate model based on its accuracy, performances of five models are basically the same with 92% accuracy on validation set at the end of training. Again, different initial weights do not have significant impact on model training except on the first one or two epochs. By comparing this plot with the plot in the previous part, we can see that the trend and the shape are basically identical for each training example except that the missclassification rate does not has an obvious upward trend. Therefore, we can conclude that the loss has great impact on the accuracy of the model. Since the model is trying to find the minimal of loss, it also find the minimal error rate.

(c) Visualization of Learned W for the Best Model

After analyzing the performance of model, we analyze the weight of the model. Since the neural networks are use the linear combination of features for each

sample before activation, the weight of the first layer can help us identify the feature importance. The best model is selected to the model with seed 4 in part(a). We present two visualization of first layer weight to explain the weight for feature importance and model accuracy.

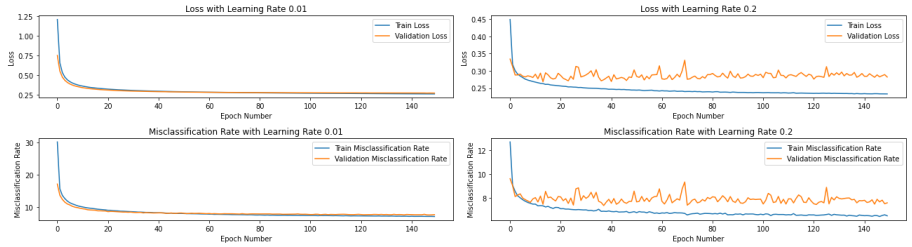


The left graph is 100 weight visualization of weight comes from the model with accuracy 92%. The right graph is from a model with accuracy 98% which added relu activation function. The weights in the left graph does not vary that much, or does not have clear graph in it, just vague circles. On the other graph, with the model accuracy of 98%, the weight has clear shape in every visualization. It means that weights are higher in this region. We can see that these obvious shape in black are all in the center part of each graph. Recall that we present an example of a training example in the beginning, and the number is in the center of the graph. By the concept of neural networks, it captures more importance features in image identification problem. We conclude that the model on the right captures more on the image feature and put more weight on these parts and leads to higher accuracy.

(d) Different parameters

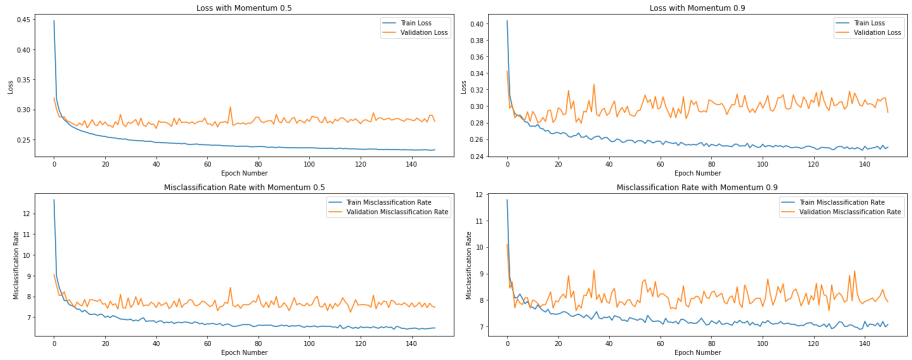
In this part, we will explore the influences of different parameters to the model performance by changing the learning rate and momentum. To compare the influences, it is sufficient to choose two different learning rate and two momentum and control other parameters to be the same.

i. Different Learning Rate



We compare the model performance on learning of 0.01 and 0.2. Learning rate is used in gradient descent, and it controls the evolving of weights. In this graph, the right column is the loss and the misclassification rate of learning rate 0.01. We can see that it has a steady decrease on every curve. Since the learning rate is low, the model adjusts its weights slow and steady. The left column is from learning rate 0.2. Since it is relatively bigger, the decrease of the loss and accuracy is not stable and bumpy. But it the loss convergent faster than learning rate 0.01. This is because the weights evolve faster than the model before, and it approach to the "correct" weights faster. After adjusted to acceptable weights (does not improve that much), curve for big learning rate still bumpy because the weights changes more than weights in the model with learning rate 0.01. As we stated in previous part that compares loss and misclassification rate, these curves behave in the same way for each model.

ii. Different Momentum



By the concept of Nesterov's Accelerated Gradient, the effect of momentum is that it it gives the information about where the momentum is about to take us, which is called "look ahead" position. Therefore, the gradient descent process can have a general idea where it will going in next few moves. With the "about-to-be" direction pointed by momentum, it helps the gradient vectors into the correct direction, leading to faster convergence.

With the concept of momentum, we can see that the model with small momentum convergent slower but steady. The model with larger momentum convergent faster and the shape of the curve is bumpy. The reason behind this is similar with the concept of learning rate. The gradient descent process is accelerated with larger learning rate and larger momentum

iii. Conclusion on Learning Rate and Momentum

From the analysis above, by control other parameters to be the same, the effect of these two parameters can be analyzed easily and clearly. For this model, since it is not complicated and convergent fast at first 10 epochs, to accelerate the learning process, we can choose larger learning rate and larger momentum. In this case, we choose learning rate to be 0.1 and momentum to be 0.5. The bumpy gradient descent won't have too much effects on the overall accuracy rate in this long training process of 150 epoch.

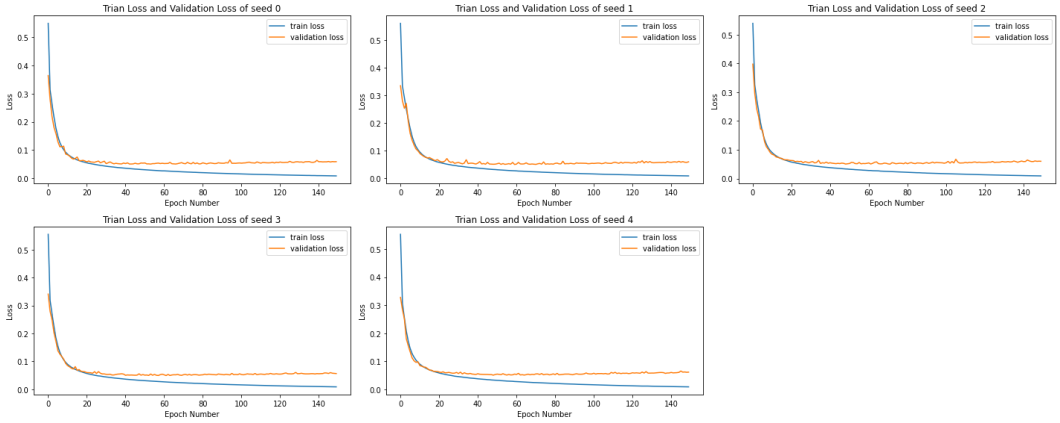
2. Simple CNN:

We construct a simple CNN and train five times with different initialization and 150 epochs. Learning rate is set to be 0.1 and moment is 0.

The architecture of this model is below:

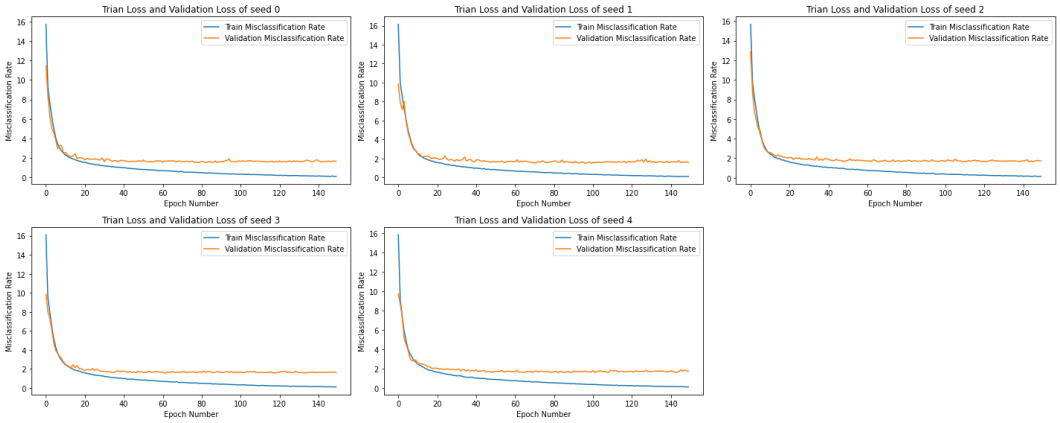
```
cnn = Sequential([Conv2D(32, kernel_size=3,
                  activation="relu",
                  input_shape=(28,28,1)), # One CNN layer
                  MaxPooling2D(2,2), # MaxPooling, reducing shape
                  Flatten(),
                  Dense(10, activation="softmax")] # For output
                  )
```

(a) Training and Test Loss Comparison



The shape of curve for training and validation loss are similar with the curve generated by ANN. The difference between them is that CNN has a more steady convergence rate and the curve is more smooth. Loss starting point is slightly lower than before. But the ending point is at about 0.05 which is much lower than 0.2 in ANN. It indicates a better performance for CNN. For overfitting problem, before training for 150 epoch, we tried early stopping function to have a idea that at what number of epoch the model will stop improving. The model will stop at about 50 epochs. Therefore, it is enough to train each model 150 epochs. Even though we did not see an obvious upward trend in the graph, the model has been trained well and enough. For the comparison of training and test loss curves, since it has similar shape with ANN, it has same interpretation. By summing up the validation loss, the model with seed 3 has been selected to be the best model. The best model has accuracy of 98.38%

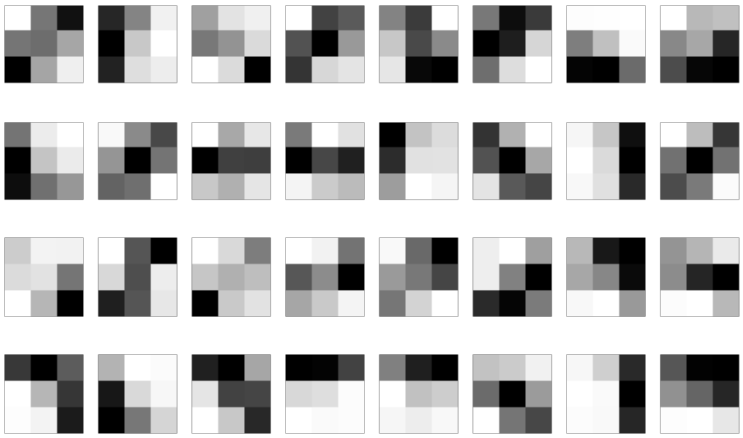
(b) Training and Test Misclassification Rate



From the results, the error of the model is about 2%, which is much smaller than the error rate in ANN. The general shape of each curve is similar with curves in

ANN, and the interpretation is the same. The behaviour of the error rate is very same to the loss curve.

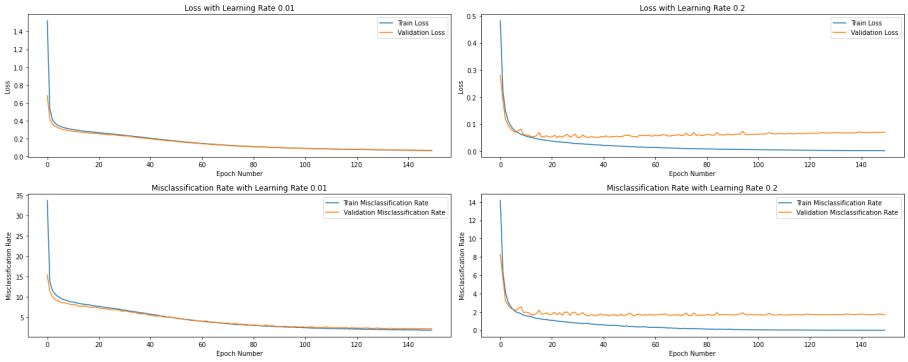
(c) Visualization of Learned W in the Best Model



Since the first layer is Conv2D with 32 filters and kernel with shape 3×3. The weight of this layer is also the same dimension. From these visualization we can see that the blocks with darker color are clustered. We speculate that the filter recognize the graph important feature are all in one place (cluster together). In this case, the important feature is in the center of the graph. Since it is only a 3×3 visualization, we cannot conclude it has a clear shape or not.

(d) Different Parameters

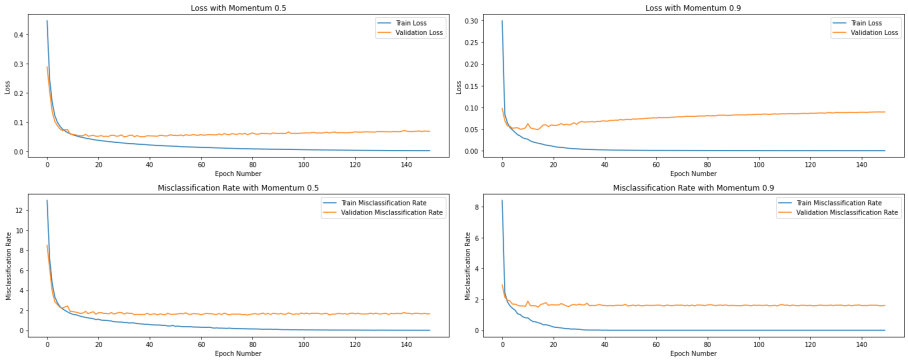
i. Different Learning Rate



By changing the learning rate to 0.01 and 0.2, we see a different behavior than the pattern in ANN. For learning rate lower than 0.1, the test loss continue

to decrease in 150 epochs and no sign of overfitting. It still can be improved by epochs. On the other hand, the model with learning rate 0.2 stopped to improve after about 20 epochs, and it has a clear upward trend. Therefore, we conclude that different learning rate has significant influence on the model performance. The accuracy is also about 0.6 percent higher in learning rate 0.01 model with 150 epochs. Even though it is not a big number, but the model with 0.01 learning rate indicate that it still can be improved, while the other one cannot. The difference of steady and bumpy curves has the same meaning in the previous ANN part.

ii. Different Momentum



By the figure, the curve for each momentum are smoother than curve in ANN part, which means that the model is more stable than ANN. Even though it is hard to see, but the curve with higher momentum is more bumpy with same reason in ANN. We also see that there is an obvious upward trend for test loss with 0.9 momentum. Since the momentum accelerate the process of gradient descent, we speculate that it does need too many epochs for large momentum for a model to adjust to the right parameters, and more epochs will lead to overfitting problem.

iii. Conclusion:

Since the accuracy of each model is about the same, we consider the stability and training time for parameters selection. In this CNN model, we select learning rate to be 0.01 since this model can be improved by smaller learning rate, and momentum to be 0.5 to accelerate the process without obvious overfitting.

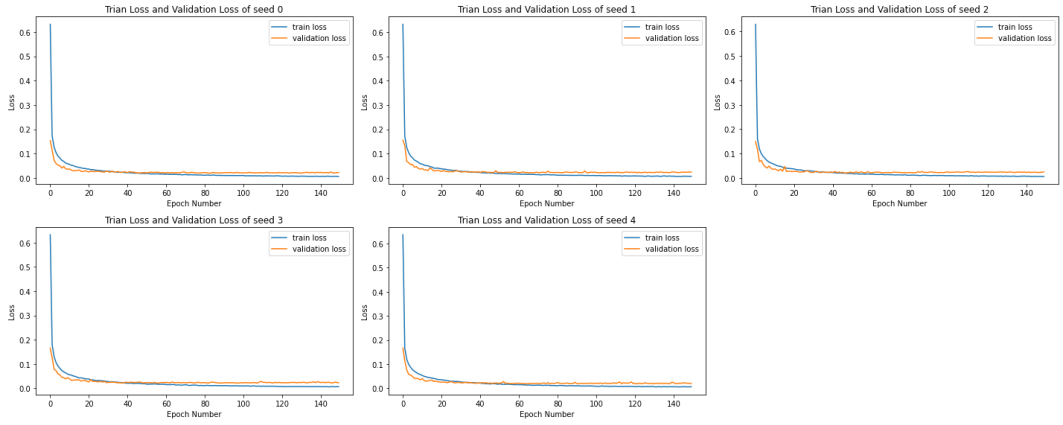
3. Deep CNN

   To obtain a higher accuracy, we add more layers to previous CNN and try to beat the performance of SVM model with error rate 1.4%. Learning rate is set to be 0.1 and moment is 0.

   The architecture of this model is below:

```
cnn = Sequential([Conv2D(32, (3, 3), padding="same",
                         activation = "relu",
                         input_shape = (28, 28, 1)),
                  MaxPooling2D(2, 2),
                  Conv2D(64, (3, 3), padding="same", activation =
                         "relu"), # Adding more CNN layer
                  MaxPooling2D(2, 2),
                  Dropout(0.5),
                  Flatten(),
                  Dense(64, activation = "relu"), # Full-connected
                  Dense(10,activation="softmax")])
```

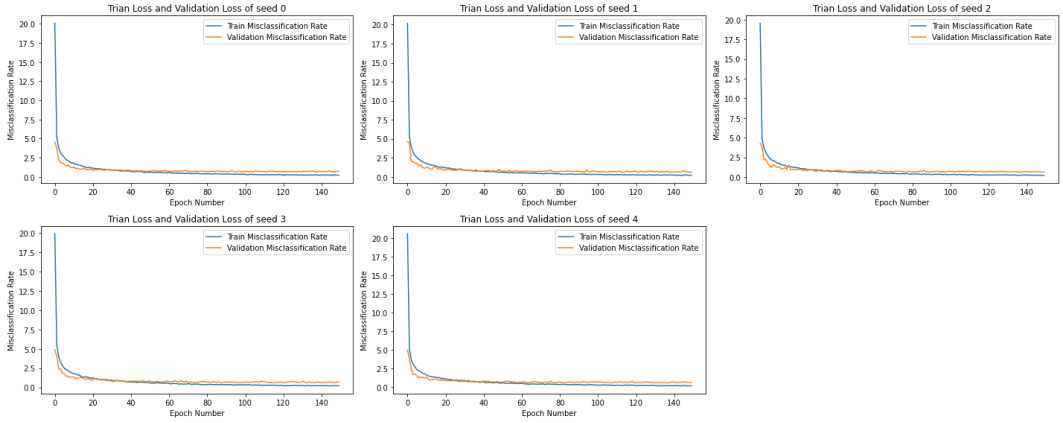
   (a) Training and Test Loss Comparison



   The general shape for training and test loss curve is the same with curves from previous two model. The starting point of training loss is about 0.6 which is slightly higher, but it drops fast to lower than 0.1 within 5 epochs. The ending point of the training loss is also lower than previous model, which indicates a better performance. We did not see an obvious sign of overfitting, but the early stopping function stops the training at about 20 epochs. Then we conclude the model is well trained. By summing up the test loss, we select model with seed 4 as our best model and the accuracy is 99.36%. We sucessfully construct a model

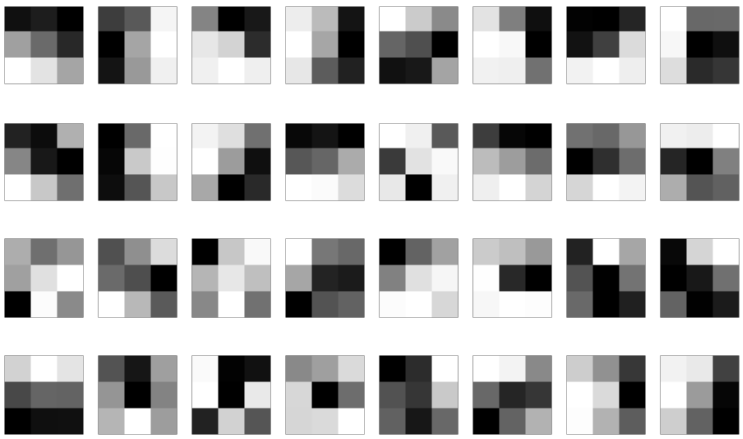
that beat the SVM model.

(b) Training and Test Misclassification



The error rate curve shape is also similar with the curves generated in previous two models. The ending error rate is 0.64%, which is the smallest in all three models. The behavior of error rate and loss is the same for each seed.

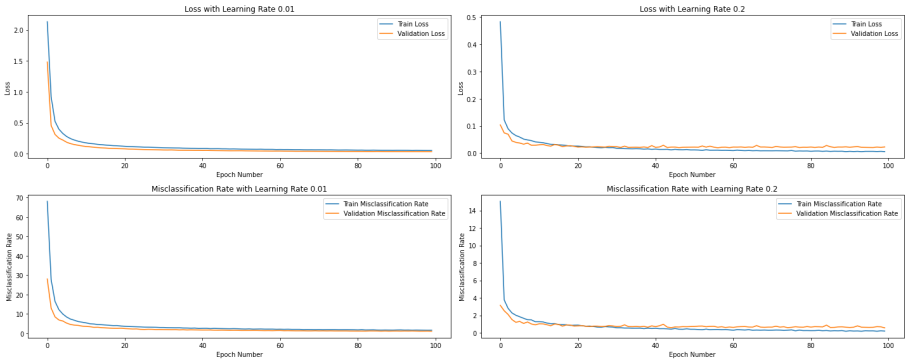
(c) Visualization of Learned W in the Best Model



Since we have similar structure with CNN in previous part, the weight of the first layer does not change much. The darker block also cluster together. The interpretation is the same in previous model.

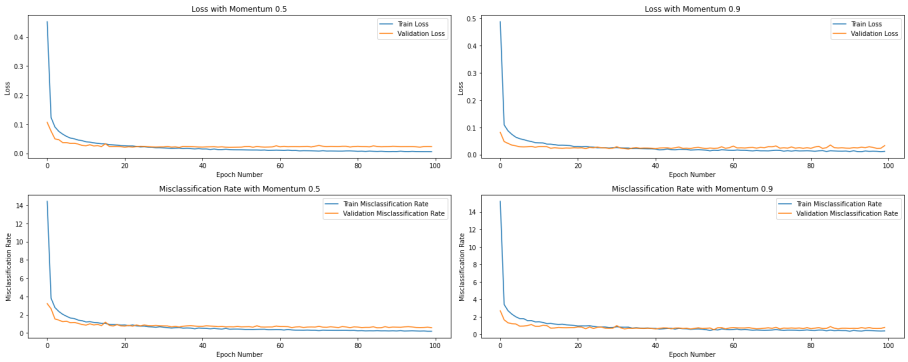
(d) Different Parameters

i. Different Learning Rate



The general shape of loss and error rate curves behave the same with the curves generated by previous CNN model. The interpretation is the same. However, we do not see an obvious upward trend in model with learning rate 0.2, and model with 0.01 can be improved with more epochs.

ii. Different Momentum



By the graph, the general shape behave the same with the curve the learning rate part. We see no sign of overfitting, and the oscillation of the curve is really small for both momentum graphs. It indicates that the gradient descent problem is more complicated than previous two models, and larger momentum can effectively raise the rate of convergence. The interpretation of different momentum is the same with the one in previous two models.

iii. Conclusion:

Since the larger learning rate and momentum do not make the model overfit-

ting, and the model can still be improved, we select learning rate to be 0.2, and momentum to be 0.9 to accelerate the training process.

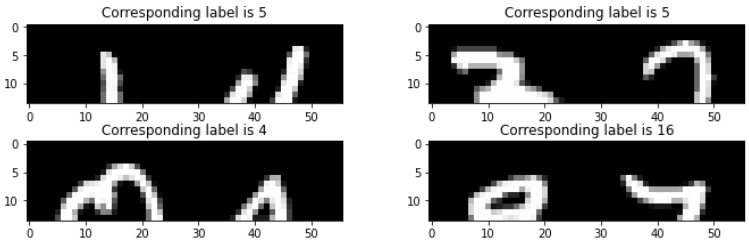
Addition of two MINST digits Training

After constructing CNN with accuracy higher than 99%, we change the dataset that composed by the combination of two hand-written digits, and the label for each case is the addition of them.

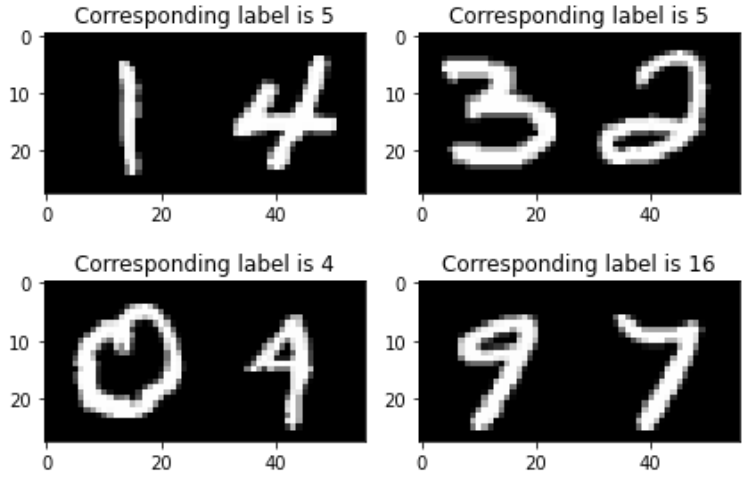
1. Shape of the dataset and each case

   We have 20,000 images in training set, 5000 images for both validation set and test set. The validation set is used in model training, and the test set is used to compare the model accuracy. For each case, it is a $1-d$ array with 1568 numbers. We divide them into 2 parts with 784 numbers each and reshape both of them into 28×28 for image showing.

   Furthermore, We find out that the pixels is scanned out in row-major by plotting the first 784 numbers since it only gives the upper half.



2. Example of the dataset



The corresponding y value is labeled in the graph.

3. Data Preprocessing

   For each cases, it has been reshaped into $(28, 56)$ and normalized. One-hot encoding for the label has been applied for the reason same with the first part.

Deep Learning Models

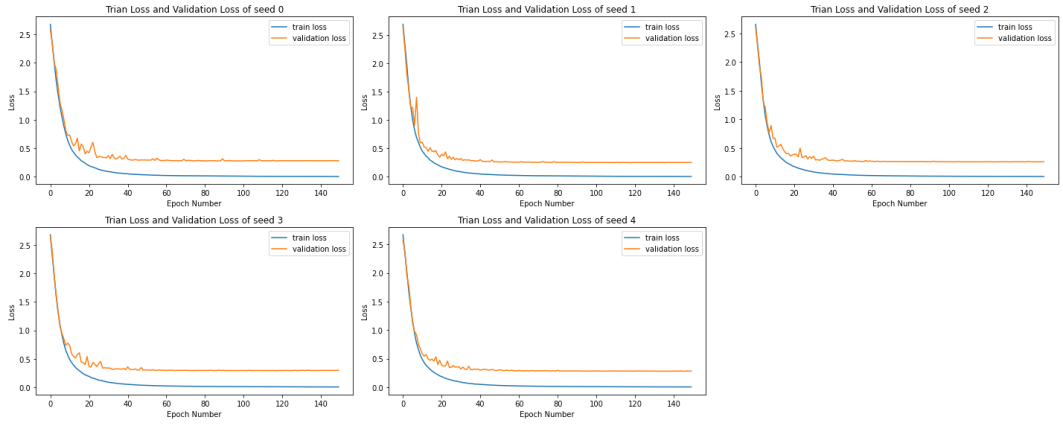
In this part, we will try to construct two models for training addition dataset. The process of training and analyzing is the same as before

1. LeNet:

   LeNet5 is a model that often are used in MINST dataset training. This LeNet5 model are trained with 150 epochs with learning rate 0.1. The architecture is below:

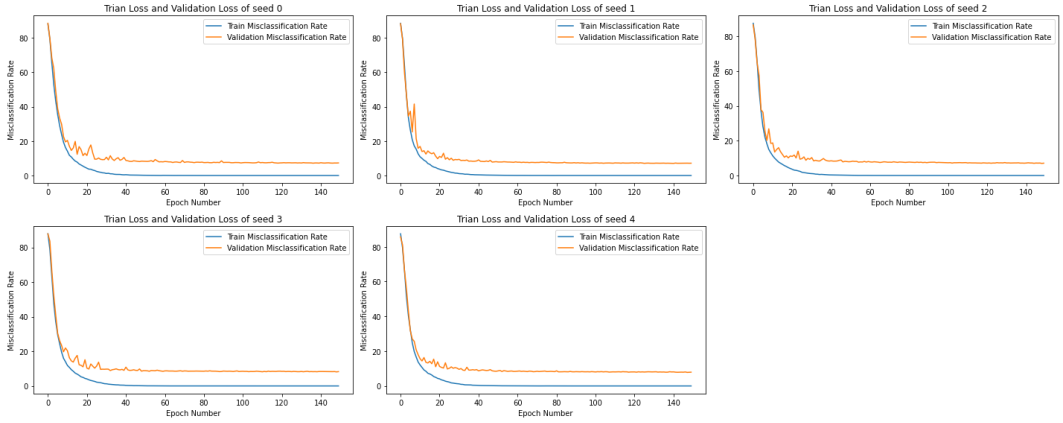
```
cnn = Sequential([Conv2D(6, (5, 5), padding="same",
                         activation = "tanh",
                         input_shape = (28, 28*2, 1)),
                  MaxPooling2D(2, 2),
                  Conv2D(16, (5, 5), padding="same",
                         activation = "tanh"),
                  MaxPooling2D(2, 2),
                  Conv2D(120, (5, 5), padding="same",
                         activation = "tanh"),
                  Dropout(0.2),
                  Flatten(),
                  Dense(85, activation="tanh"),
                  Dense(19,activation="softmax")])
```

   (a) Train and Test Loss Comparison



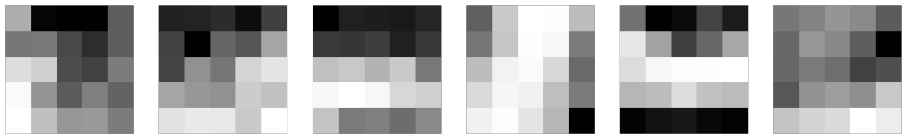
From the results, we can see that the behavior of these curves are similar with curves generated by models with different dataset. However, the loss starting point and ending point is much higher than the curves in previous part, which indicates that the model here does not have a good performance as before. From these plot, we do not see the sign of overfitting, but it is well-trained. The interpretation of difference between train loss and test loss is the same as before. The best model is selected by the same critrion, and the accuracy on test set is 93.52%

(b) Train and Test Misclassification Comparison



The general pattern is the same as the models in previous part, and the interpretation is the same. The shape of loss and error rate curves are the same for each seed. Furthermore, since we have higher loss, the error rate is also higher about 8% here.

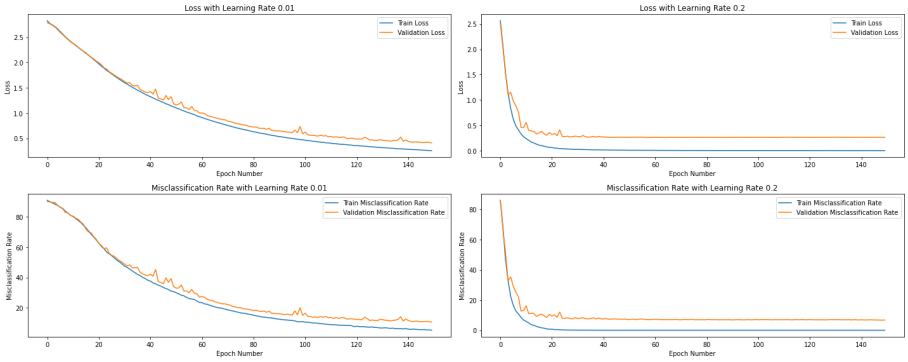
(c) Visualization of Learned W in the Best Model



Since the fisrt layer of our LeNet structure is a ConV2D with 6 filter and a 5×5 size kernel. The visualization of W also has the same dimension. Since we have a larger picture, we can see that the darker blocks cluster together and form its own region. It is similar with the first 100 weights graph we draw. Every filter highlights the important feature in the graph.

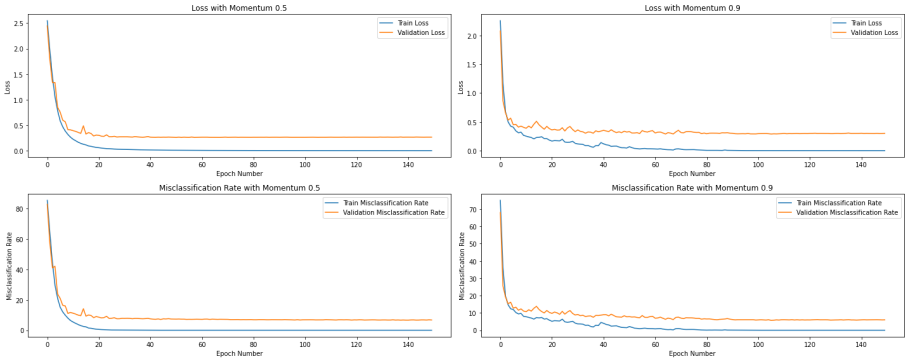
(d) Different Parameters

i. Different Learning Rate



In this part, we see a complete different curve than before in the model with learning rate 0.01. It converges really slow and steady, and the ending point of training loss is about 0.2 of learning rate 0.01, and 0.1 for learning rate 0.2. Therefore, we cannot conclude that the model is finished training with 150 epochs because the loss can continue to decrease. The relationship between learning rate and convergent rate is explained in previous parts. The error rate and loss curves behaves the same.

ii. Different Momentum



General shape of loss and error rate curves are the same with previous parts, and the interpretation is the same. We did not see the sign of overfitting here.

iii. Conclusion:

Since we see that the model is not well-trained with 150 epochs and 0.01 learning rate, we can select this learning rate with higher epochs and high

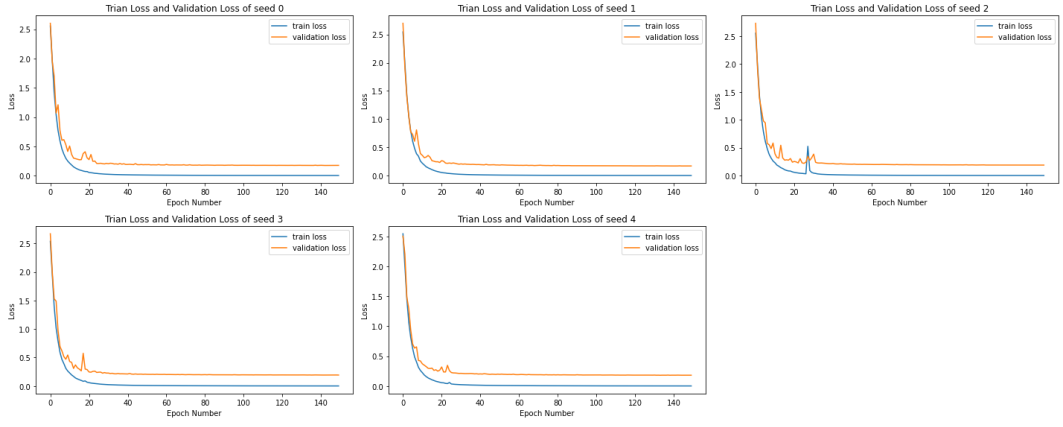
momentum, it might give us a higher accuracy due to low learning rate If the time is important, we can select learning rate to be 0.2 and momentum to be 0.5 to accelerate the process.

2. 6-layer CNN

   This model is composed by 6 CNN layer to see if more CNN layer will help in image recognition. This CNN model is trained with 150 epochs with learning rate 0.1. The architecture is below:

```
cnn = Sequential([Conv2D(32, (5, 5), padding="same",
                         activation = "tanh",
                         input_shape = (28, 28*2, 1)),
                  Conv2D(32, (5, 5), padding="same",
                         activation = "tanh"),
                  Conv2D(32, (5, 5), padding="same", strides=2,
                         activation = "tanh"),
                  Conv2D(32, (5, 5), activation = "tanh"),
                  Conv2D(32, (5, 5), padding="same",
                         activation = "tanh"),
                  Conv2D(32, (5, 5), padding="same", strides=2,
                         activation = "tanh"),
                  Flatten(),
                  Dense(256, activation="tanh"),
                  Dense(19,activation="softmax")])
```

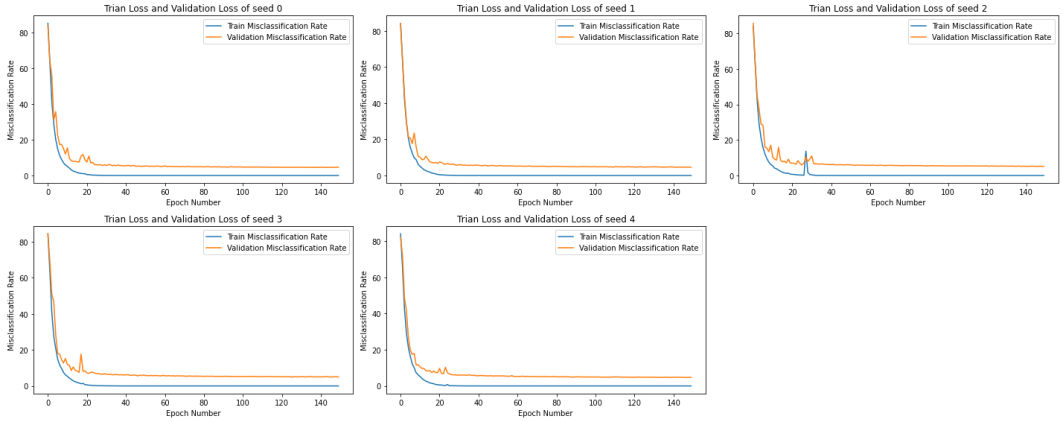
   (a) Train and Test Loss Comparison



   The shape of these curves are the same as before. We noticed that there is an unusual uptick in the model of seed 2, it probably because the initialization is

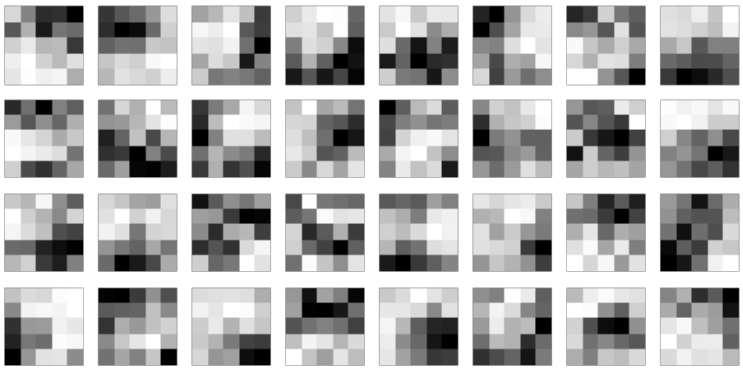
very bad that the gradient descent did not find the global minimal values. The interpretation of difference between train and test loss is the same as before. We did not see an obvious sign of overfitting. The best model selected is the model of seed 1. The accuracy of this model is 95.99%

(b) Train and Test Misclassification



The general pattern of this graph is the same as before, and the interpretation of the error rate comparison for train and test is the same. We noticed that the starting point for error rate is very high and reached over 80 percent. It corresponding loss starting point is also high, and decreases very fast.

(c) Visualization of Learned W for the Best Model

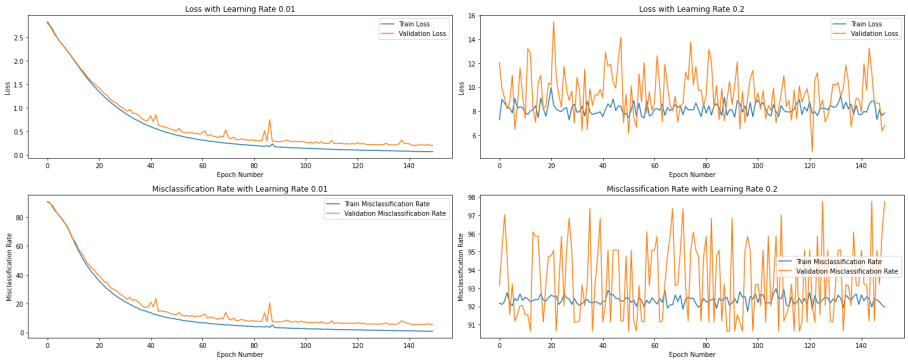


Since the first layer is Conv2D layer with 32 filters and 5×5 kernel size, the visualization of W has the same dimension. The interpretation of this is the same

with previous model.

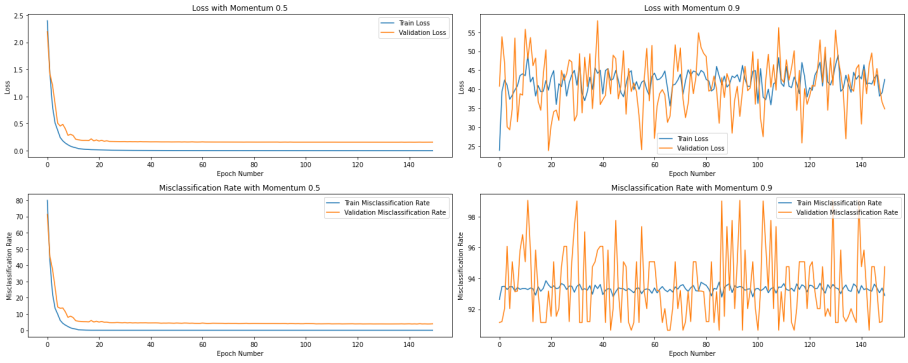
(d) Different Parameters

i. Different Learning Rate



With this model, we see a very different shape of loss function in the model of 0.2 learning rate. Because the learning rate is too big, it does not converge at all. The train loss does not decrease as expected, and test loss oscillated randomly. The model with learning rate 0.01 decrease slow and steady like the one in previous model.

ii. Different Momentum



The model with momentum 0.5 has been successfully accelerated and has the shape similar with models in the previous part. But, again, the model of momentum 0.9 does not converge at all. It has the same reason with the one in the learning rate part.

iii. Conclusion

To select parameters, we should not make it too high because it will not converge. The training time for this model of 150 epoch is about 15 minutes. Therefore, we need to accelerate the process by choosing learning rate to be 0.1 and momentum to be 0.5 without overfitting and the loss function should converge.

3. Model Comparison

LeNet has accuracy of 93.52% and 6-layer CNN has 95.99%. Thus, CNN has a better performance on the problem of digits addition. However, it takes much shorter to train LeNet, and it is computational cheaper. If we change parameters such as learning rate and momentum, it might have a better performance than the current one. CNN should also be trained with different parameters to accelerate the training process.

Both of them cannot achieve 99% accuracy on this dataset. Possible reason is that this dataset is the addition of two digits, but the machine only recognize the image feature. It does not learn to add digits by this model. Also, there are many combinations of digits to have the same label. For example, 3 + 7 = 4 + 6 = 5 + 5 = 10. Machine will not treat them as the same because it is not programmed to add, and it might makes the problem confusion. Overall, to solve the addition problem, it is not enough to only give machine images of two digits to learn the feature of that image.

Conclusion

In this report, we use two dataset of handwritten digits dealing with two different problems. The first one is number recognition and the second one is number addition. In the first part, we use three neural network models and achieve over 99% of accuracy. But in the second part, we only achieve over 95% accuracy. To have a better performance in the future study, we can try different combination of learning rate and momentum or change to other models such as RNN.