Multilayer Perceptron

– Classification on Handwritten Digit Data Set

Abstract

The optimal model we choose for one hidden layer has the parameters as following: learning rate 0.3, weight decay 0.001, batch size 200 and max epochs 50 with Softmax Cross Entropy loss and ReLU activation. The accuracy on the test set reaches to 0.9768. For two hidden layers, we tested both Softmax Cross Entropy loss and Euclidean loss with ReLU activation. Their accuracy scores both reach 0.98, among which MSE is slightly better with an accuracy at 0.9811.

The balance between batch size, max epoch, learning rate and weight decay, has decisive influences over the learning results. Neuron amount in one hidden layer, that is how wide a layer/model is, also has a huge impact over the training results. If less number of neurons is chosen it will lead to underfitting and high statistical bias. Whereas if we choose too many neurons it may lead to overfitting, high variance, and increases the time it takes to train the network.

In our experiment, ReLU works generally better than Sigmoid. sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. x>0 elementwise in f=wTx+b)), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights.

With more time, we could try to use early-stop strategy and Adam in the MLP and see how the network behaves. Besides, given the two-hidden-layer networks with softmax and MSE have similar accuracy on the test set, we should run tests to check which one is better with statistical significance.

1 Introduction

The MNIST handwritten digit recognition dataset is one of the most commonly used datasets in the field of image classification. It contains 60,000 training images and 10,000 test images. The numbers in the images are all scaled to the same size and placed in the center of the image, the image size is 28×28. The numbers in the MNIST data set include 0 to 9, a total of 10 classes.

The goal of the experiment is to construct a multilayer perceptron network for these digit classification. The base framework has been provided, so that we only need to fill in loss calculation, network forward and backward propagation and SGD optimizer with weight decay.

2 Programming Modules

Numpy and Pandas are used for data processing. Matplotlib is used for analyzing extracted features and training result visualization. We invoke Tensorflow for data preprocessing and stepping.

One problem encountered is that tensorflow will cause a crash on the kernel if a certain dynamic library has multiple instances. The error message suggests that “Initializing libiomp5md.dll, but found libiomp5md.dll already initialized”. A solution is to delete this library under the environment path folder[1]. \

3 Data Preprocessing

Grayscale information for each pixel is normalized and the ground truth labels from 0-9 are converted to one-hot matrices.

The training set is splitted into training and validation sets. Parameter tuning and network choosing are on these two sets. No information about the test set is introduced into the training process.

4 Parameter Tuning for MLP with Euclidean Loss

In Section 4 and 5, we will compare different hyperparameter settings and how they work with MLP with only one hidden layer. The MLP network models used are:

1. MLP with Euclidean Loss and ReLU Activation
2. MLP with Euclidean Loss and Sigmoid Activation
3. MLP with Softmax Cross Entropy Loss and ReLU Activation
4. MLP with Softmax Cross Entropy Loss and Sigmoid Activation

4.1 Batch Size

Batch sizes for testing are: 50, 100, 200, 500, 1000.

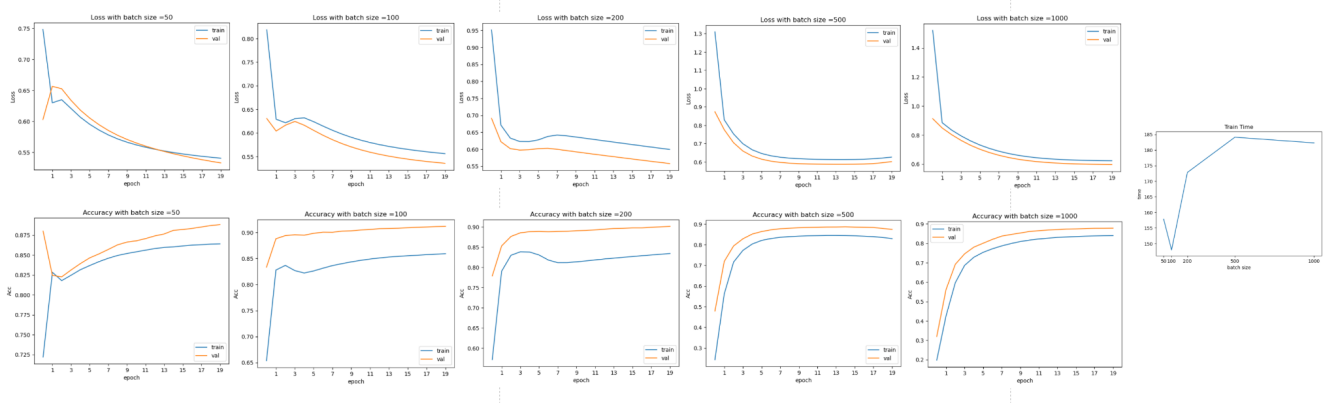
The number of training examples used in the estimate of the error gradient is a hyperparameter for the learning algorithm called the “batch size,” or simply the “batch”.

Smaller batch sizes are normally used for:

1. Smaller batch sizes are noisy, offering a regularizing effect and lower generalization error;
2. Smaller batch sizes make it easier to fit one batch worth of training data in memory. Large batch sizes usually suggest more stable gradients[2].

We will start with max epoch 20, learning rate 0.1 and weight decay 0.01 as default configurations.

4.1.1 MLP with Euclidean Loss and Sigmoid Activation

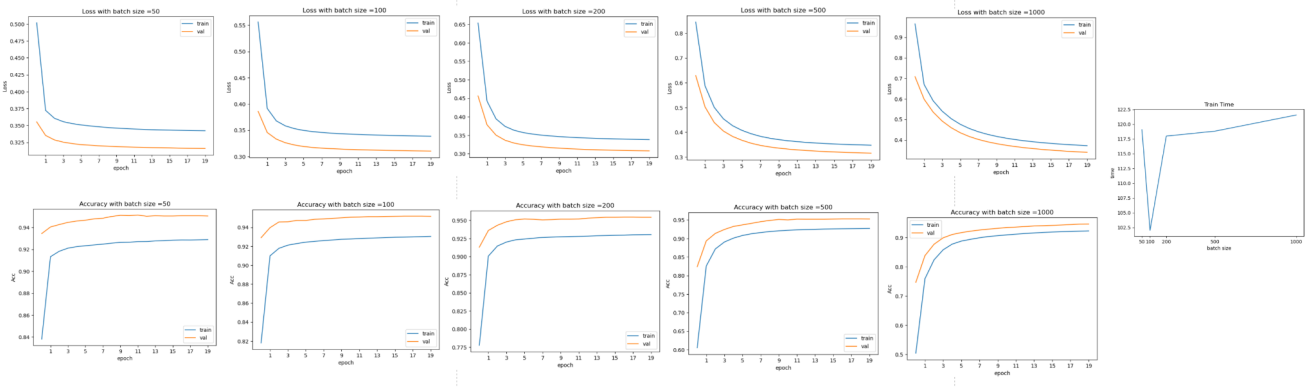


Almost no underfitting happens in the group of experiments.

Before batch size is beyond 500, local oscillation happens at quite an early stage of training(small epoch). Loss on the train set and validation set both decrease, then increase and decrease again. One possible explanation is that the optimizer pushes the network out of the minimum and then the loss decreases healthily again towards a different local minimum. And the second downward trend leads to a much lower loss and higher accuracy[3]. This means the first local minimum may be nowhere near the global minimum.

When batch size is 500, the loss starts to grow after epoch reaches 13, and at the same time, the accuracy drops. This is overfitting, due to we train too much on the train set. Good convergence happens when batch size is 1000 as the loss tends to be stable. However, the loss from batch size 100 is much lower than batch size 1000 and still keeps decreasing. Also, taking into account the train budget(time), we will use batch size 100 for later tests.

4.1.2 MLP with Euclidean Loss and ReLU Activation



Using ReLU as an activation function does not lead to local minimums as much as using Sigmoid. One reason may be ReLu has benefits that reduced likelihood of the gradient to vanish. In contrast, the gradient of sigmoids becomes increasingly small and the model will have difficulty in converge to global optimal.

ReLU is also more computationally efficient to compute than Sigmoid-like functions since Relu just needs to pick max(0,x) and not perform expensive exponential operations as in Sigmoids[5].

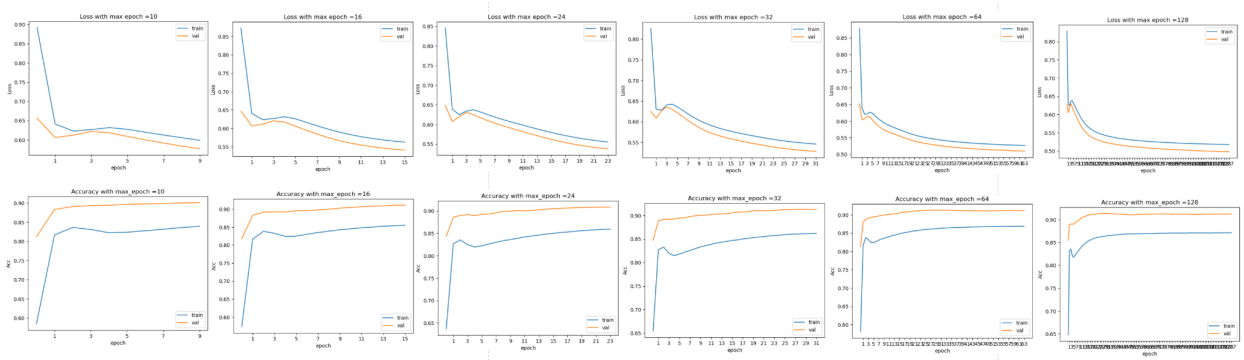
Batch size 200 has the smoothest loss curves and accuracy curves and second lowest training cost. The convergence happens fast on this model. Therefore, we will use it for tuning epoch settings.

4.2 Max Epoch

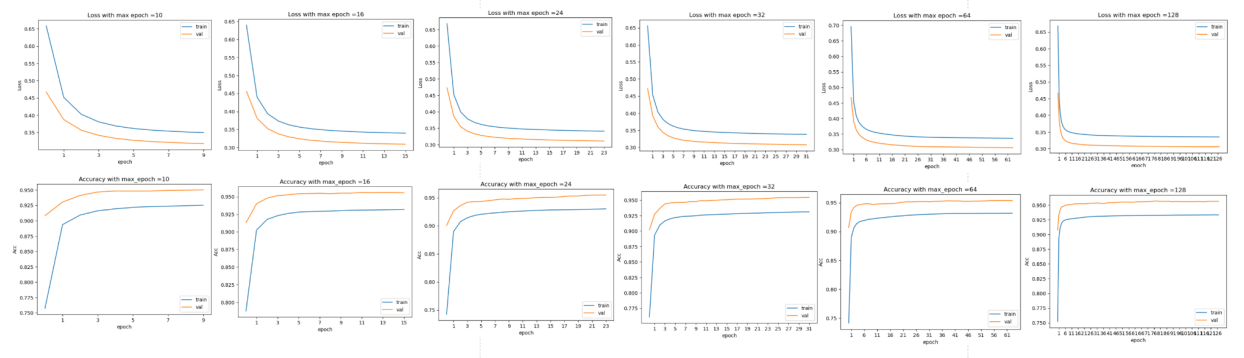
Observed from the last section, we could notice that non global optimal happens before epoch reaches 10. Thus, we will start testing max epoch from this anchor. Batch sizes for testing are: 10, 16, 24, 32, 64.

Overfitting can happen if the model trains so much on the training data. We can also get a glimpse of this through diagrams from the previous section. Take 2.1.1 with batch size 500, the loss starts to increase and accuracy starts to drop after the epoch reaches 15. We need to joint control the model with epoch and batch size. One example is epoch usually increases when the batch size grows.

4.2.1 MLP with Euclidean Loss and Sigmoid Activation (Batch Size:100)



4.2.2 MLP with Euclidean Loss and ReLU Activation (Batch Size:200)



Using ReLU as the activation function has faster convergence, lower loss and higher accuracy. The best performance is MLP with Euclidean Loss and ReLU Activation when batch size is 200 and epoch is around 43.

The network with ReLU converges fast before epoch 6 and becomes stable(almost converged) at around epoch 20.

After reaching a local minima, the loss curves always follow a downward trend as the epoch grows. As the max epoch reaches above 64, the accuracy starts to be stable. This is potentially because our network starts to memorize data, and we need to consider some forms of drop out[6]. In other words, our network has already converged and we need to consider other ways for regularization for better learning.

In addition to the above paragraph, sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. x>0 elementwise in f=wTx+b)), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights[7].

In conclusion, using a one-hidden-layer MLP with ReLU activation and MSE works better on our data set. We will use this combination for later testing on neural unit amount, learning rate and weight decay.

4.3 Learning Rate, Weight Decay and Neuron Amount

Below listed some some typical rules for choosing neuron amount:

1. The number of hidden neurons should be between the size of the input layer and the size of the output layer.
2. The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
3. The number of hidden neurons should be less than twice the size of the input layer [8].

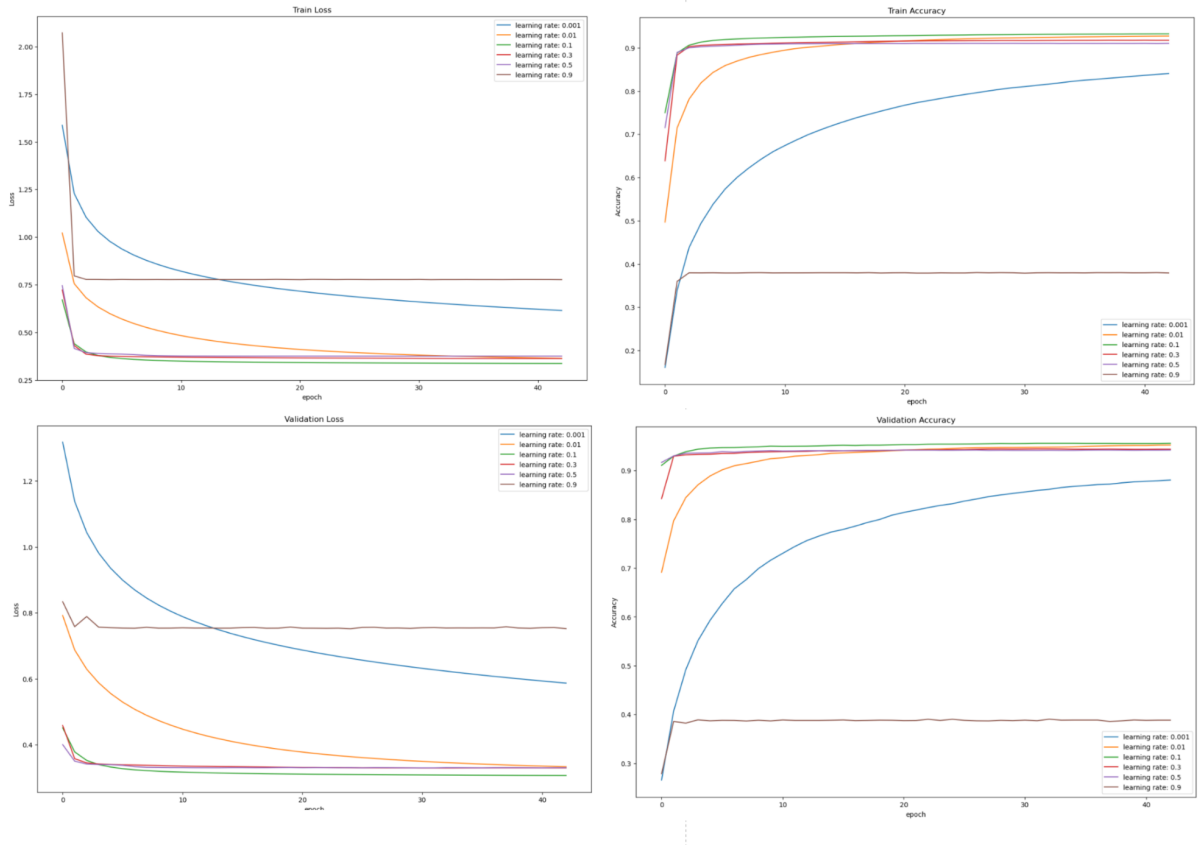
$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

Too fewer neurons will always result in underfitting, and too much can lead to overfitting. Image above is a formula for reference. Therefore, we will test [30, 60, 75, 100, 128, 150] for neuron amounts.

4.3.1 Learning Rate

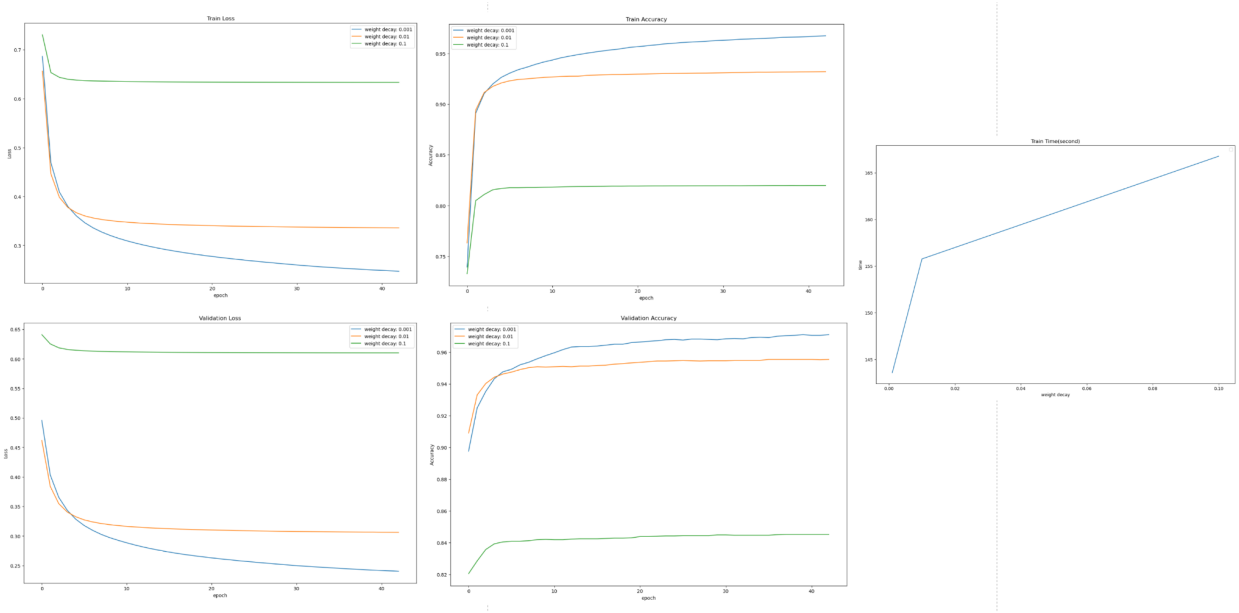
Learning rates at 0.001 and 0.9 carry out the worst performances on our data set. Learning rate 0.001 makes the network slow to converge and 0.9 makes the loss curves and accuracy fluctuates on the validation set. Too large learning rates will make the weight oscillate around the optimal, but always skip the best solution.

Networks' behaviors with learning rate 0.01, 0.1, 0.3 and 0.5 are close. Among which 0.1 has the best learning results. It converges fast and has the highest prediction accuracy. Therefore, we will use 0.1 as the learning rate. One thing needs to be noted is that we will still use max epoch 43 to tuning weight decay and neuron amount settings. This is because all parameters work together in the training. Even if our current settings(batch size: 200, max epoch: 43, weight decay: 0.01, neuron amount: 128) converge at around epoch 10, we could not be sure that this early convergence will remain the same when other parameters get changed.



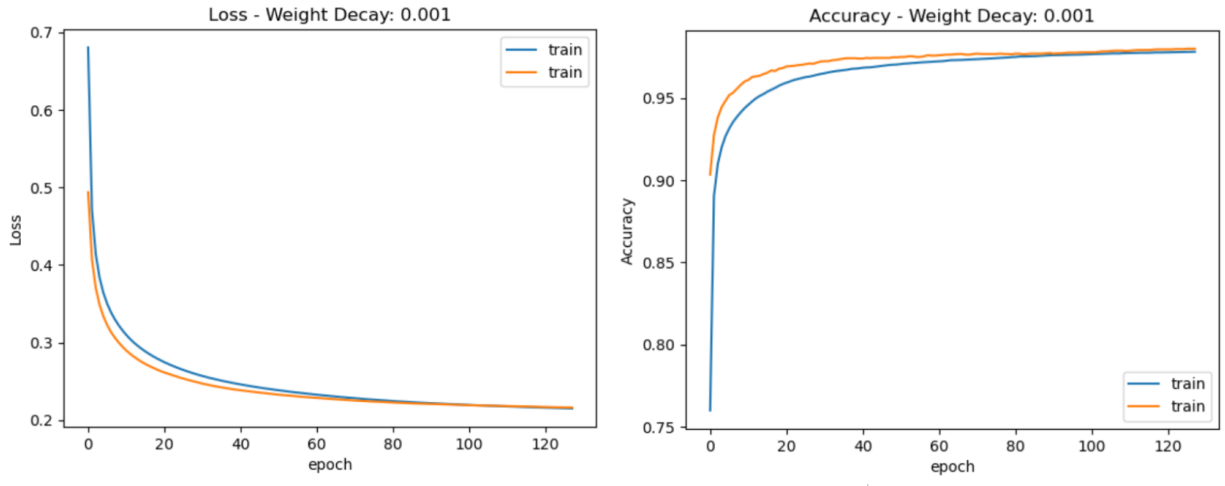
4.3.2 Weight Decay

Weight decay, sometimes referred to as L2 normalization, is a common way to regularize neural networks. It helps the neural networks to learn smoother or simpler functions which most of the time generalizes better compared to spiky, noisy ones. Weight decay here acts as a method to lower the model's capacity such that an over-fitting model does not overfit as much and gets pushed towards the sweet spot. This also shows that weight decay will have a negative impact if the model is originally operating in the under-fitting region[9].



Observations from the above diagrams are:

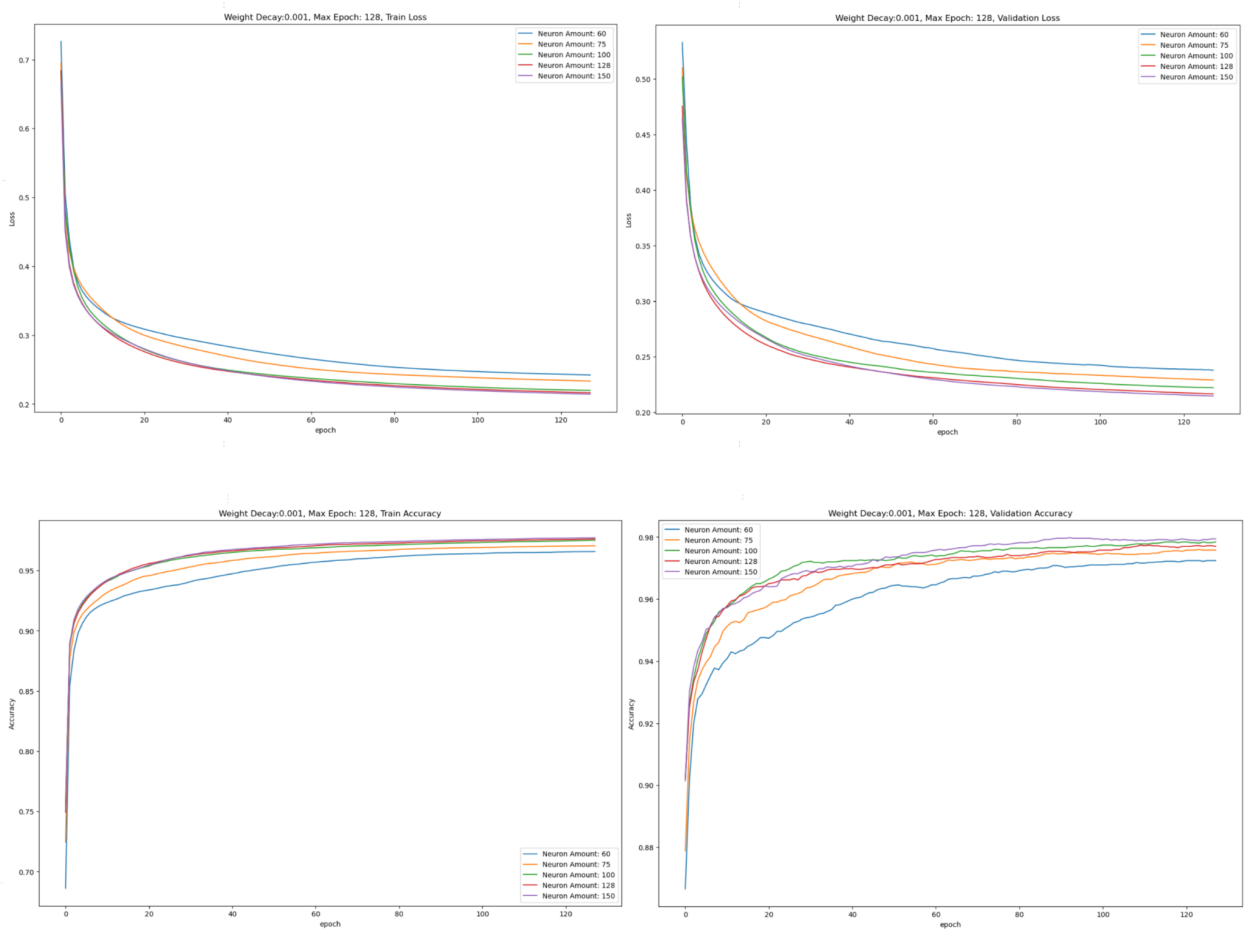
1. Weight decay at 0.1 has the worst performance among the values we test. Too large weight decay factors usually make the network underfit and have small weights that cannot capture the complexity of the data and classification rules.
2. Diagrams for weight decay at 0.001 suggests that the model has still not converged at the last epoch we run. The associated accuracy on the validation set fluctuates a lot. When the weight decay factor is too small, the network may still overfit and have large weights, that is, not converge.
3. Weight decay 0.01 has the smoothest loss and accuracy. The accuracy is around 0.95112 and loss is as low as 0.31840.
4. We will increase the epoch to 128 and see how the network converges with weight decay 0.001.



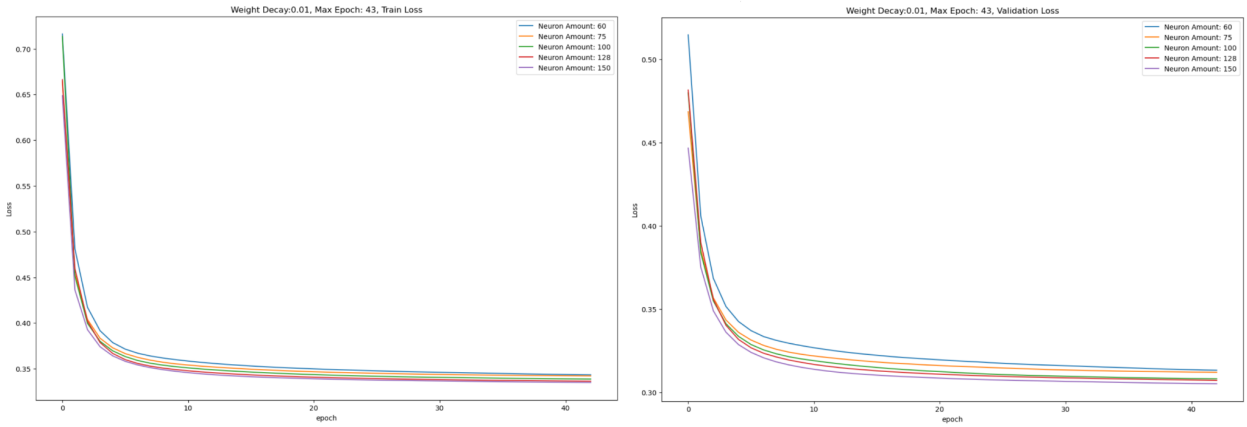
From the above diagram(orange is for validation performances), we could see that weight decay 0.001 has the lowest loss 0.24141 and highest accuracy 0.97282. Therefore, we will use it and the weight decay 0.01 to test neuron amount.

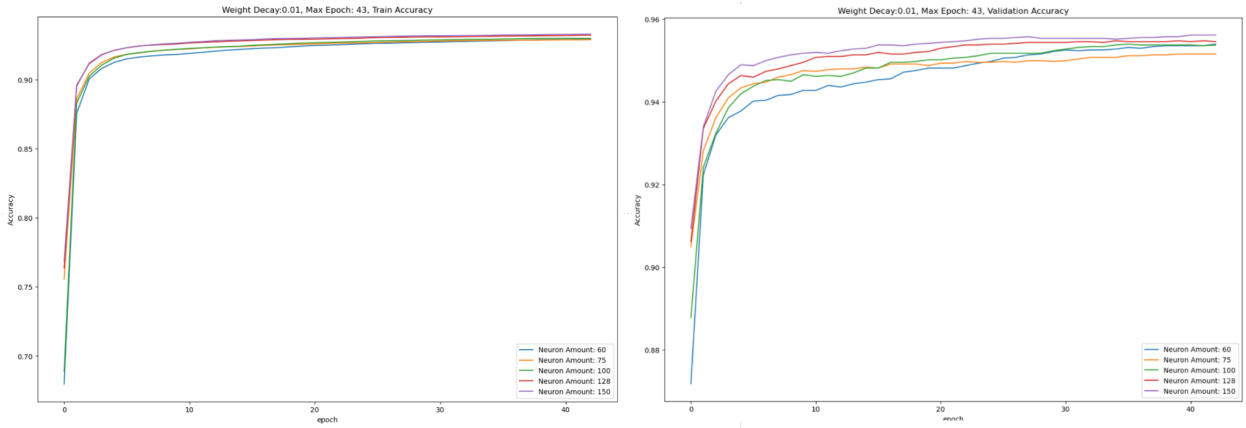
4.3.3 Neuron Amount

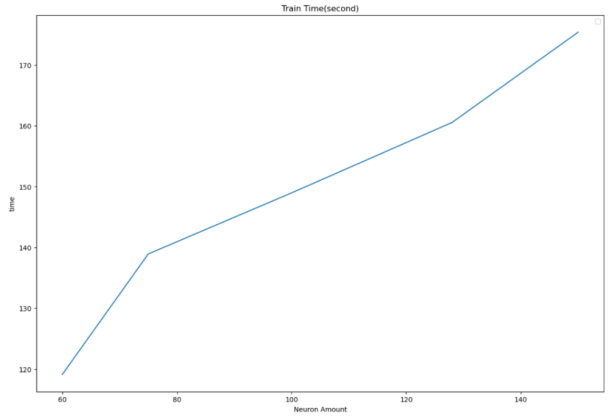
Batch Size: 200, Max Epoch: 128, Weight Decay: 0.001



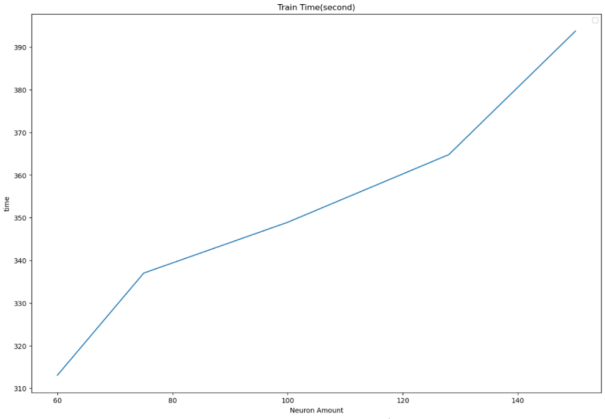
Batch Size: 200, Max Epoch: 43, Weight Decay: 0.01







Batch Size: 200, Max Epoch: 43, Weight Decay: 0.01



Batch Size: 200, Max Epoch: 128, Weight Decay: 0.001

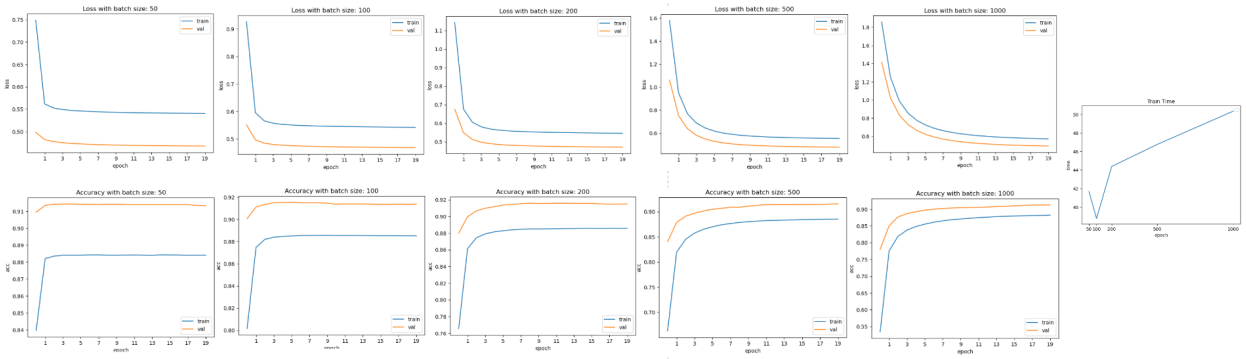
The best performance is with 150 neurons in the hidden layer with 128 epochs and weight decay at 0.001, despite it having the highest training cost.

5 Parameter Tuning for MLP with Softmax Cross Entropy Loss

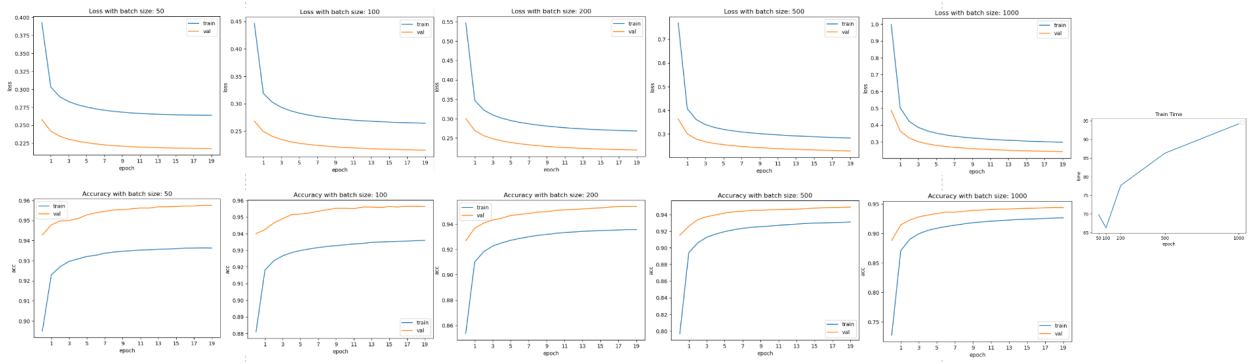
We will follow the same process of parameter tuning in Section 2. Some analysis will be omitted since we have already analyzed quite some details in Section 2.

5.1 Batch Size

5.1.1 MLP with Softmax Cross Entropy Loss and Sigmoid Activation



5.1.2 MLP with Softmax Cross Entropy Loss and ReLU Activation



ReLU activation takes more training cost but carries out better performance than Sigmoid activation. Losses with ReLU are down to 0.3 and Sigmoid are mostly above 0.5.

The larger the batch size, the slower the network converges. The size of mini-batches is essentially the frequency of updates: the smaller mini batches the more updates. Geometrically, several updates is better because you are drawing several segments, each in the direction of the (approximated) gradient at the start of each segment. While a single big update is a single segment from the very start in the direction of the (exact) gradient. It's better to change direction several times even if the direction is less precise[10].

As the batch size increases, performance on the validation set and the performance on the train set start to approximate each other. A network with larger mini batch sizes usually has better generalization, that is, can estimate errors on the validation set better.

Smaller batch size with Sigmoid converges at around epoch 5-7, with a comparative high loss and low accuracy, due to potential underfitting.

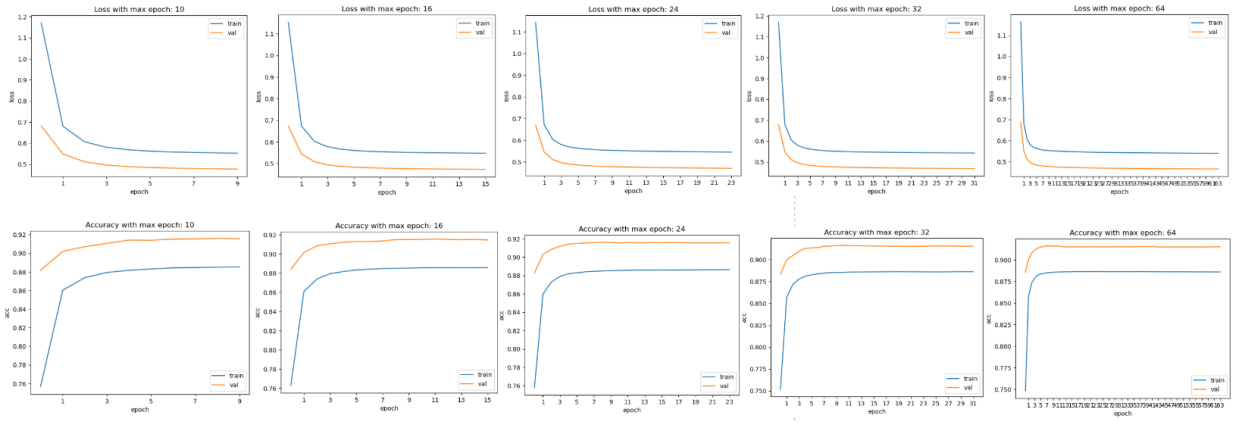
The ReLU activation network has not fully converged within 20 epochs since loss curves in the diagrams all show a continuous downward trend.

Best performance for Sigmoid is with batch size 200. It has the lowest training/validation loss and highest accuracy. Also, its curves are much smoother than with batch size 100, which has similar loss and accuracy on the train set.

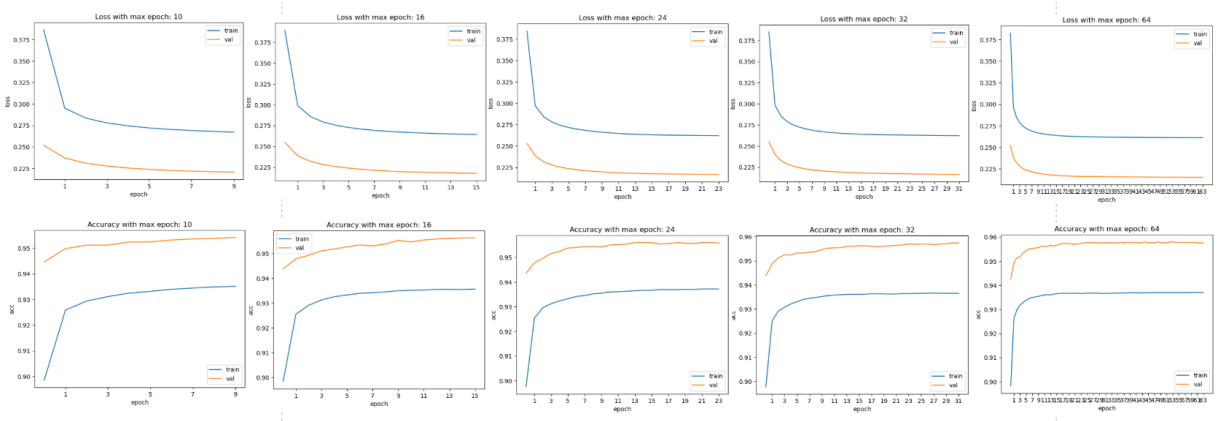
ReLU's best result is with batch size 50. However, the rest of the networks haven't converged. Therefore, we will also test batch size 200 with higher epoch.

5.2 Max Epoch

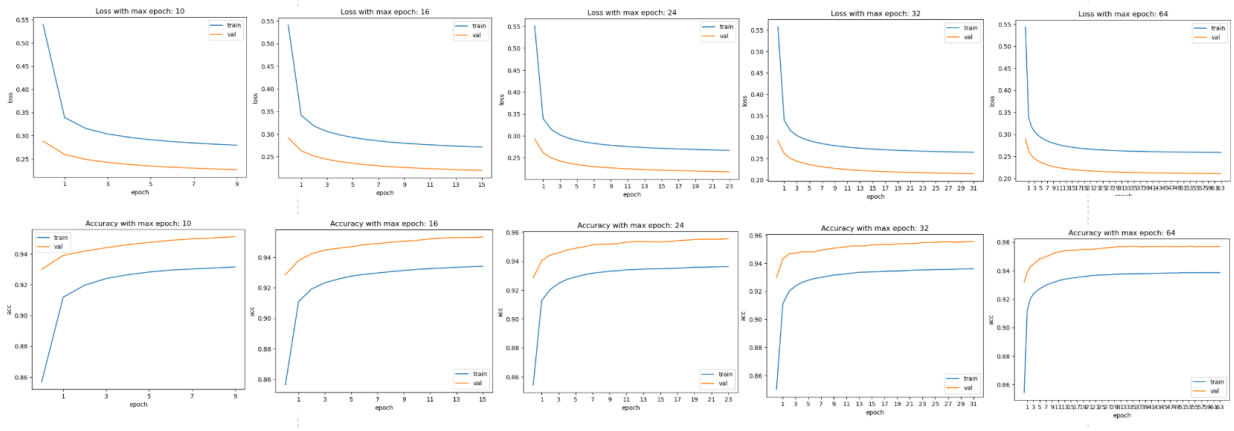
5.2.1 MLP with Softmax Cross Entropy Loss and Sigmoid Activation (Batch Size: 200)



5.2.2 MLP with Softmax Cross Entropy Loss and ReLU Activation (Batch Size: 50)



5.2.3 MLP with Softmax Cross Entropy Loss and ReLU Activation (Batch Size: 200)



For Sigmoid activation, the loss and accuracy stop to change after convergence. It carries out the similar training result as in the previous section.

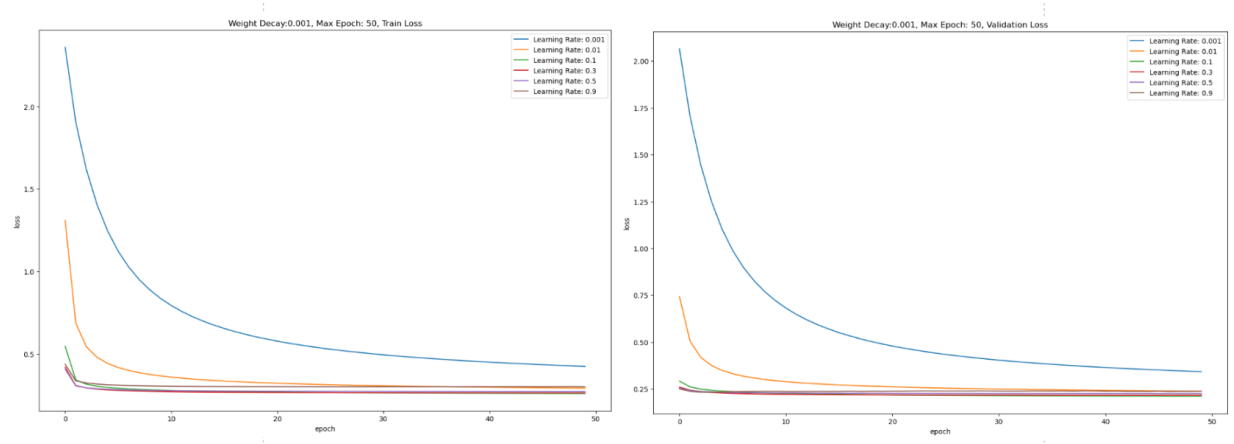
ReLU activation still has better learning results than Sigmoid.

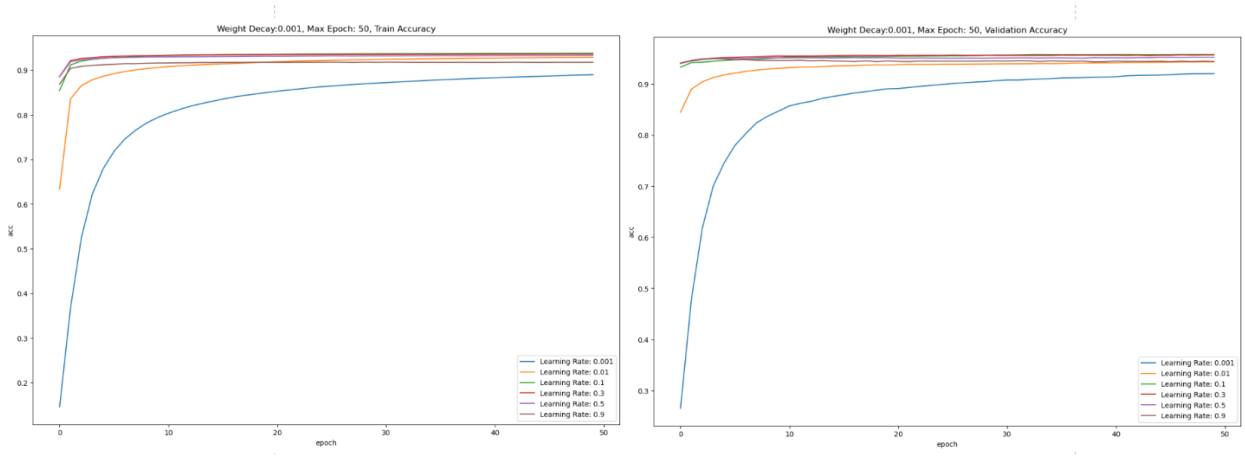
Although the loss is down to 0.225 with batch size 50 using ReLU, much lower than batch size 200, batch size 200 still has an accuracy curve with far less fluctuations. Convergence for batch size 200 roughly happens around epoch 48.

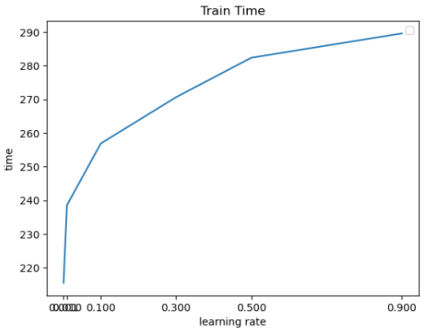
In conclusion, using a one-hidden-layer MLP with ReLU activation and softmax loss works better on our data set. We will use this combination for later testing on neural unit amount, learning rate and weight decay.

5.3 Learning Rate, Weight Decay and Neuron Amount

5.3.1 Learning Rate



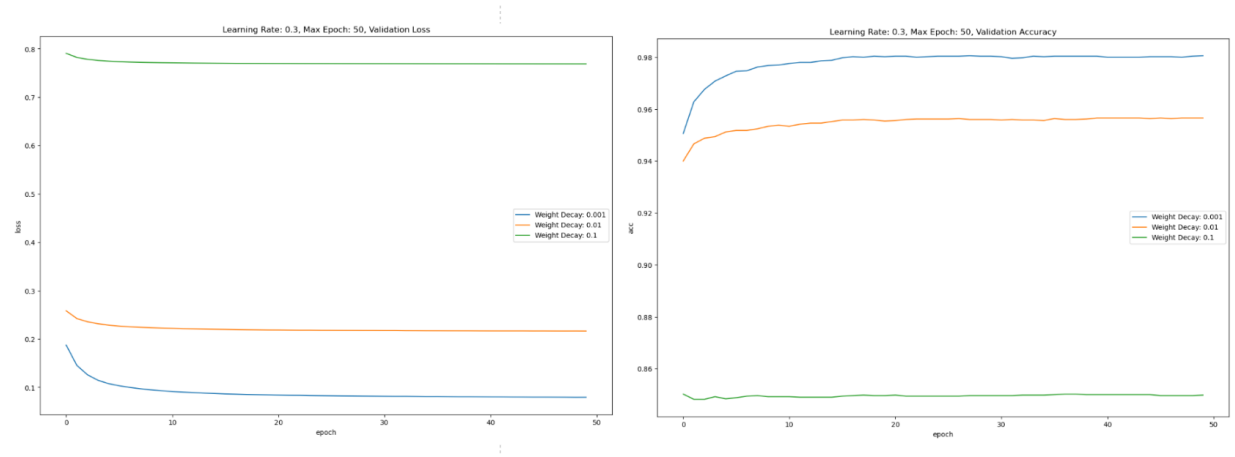




Learning rate at 0.001 carries out the worst performances on our data set. It makes the network slow to converge. Other learning rates' results are close.

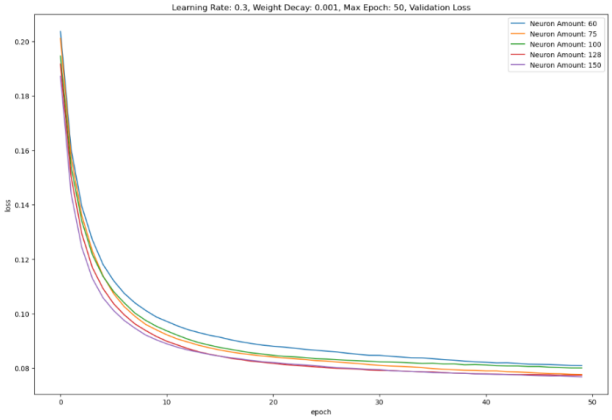
Among all tested, learning rate 0.3 has the best learning results. It converges fast and has the highest prediction accuracy (mean: 0.955124). Therefore, we will use 0.3 as the learning rate for later testing.

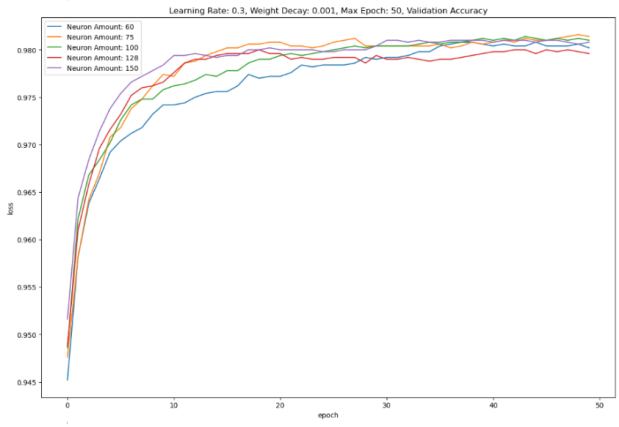
5.3.2 Weight Decay



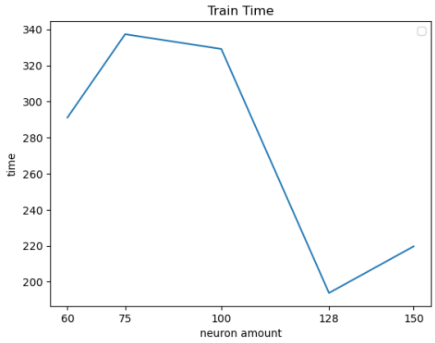
Weight decay 0.001 has the best performance. The decay value is small since our learning rate is comparatively high, and we need to do a proper trade-off to guarantee both the convergence and the network behavior.

5.3.3 Neuron Amount





Same as the MLP with MSE and ReLU, neuron amount 150 has the lowest loss. However, neuron amount 75 has similar results and a bit higher accuracy at large epochs. Mean validation accuracy with 150 neurons is 0.97734 and with 75 neurons is 0.978. 75 neurons also has the highest training cost.



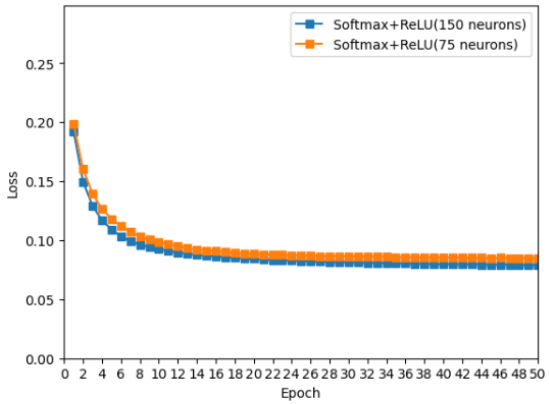
We will use both of these performances to compare the best performance with MSE.

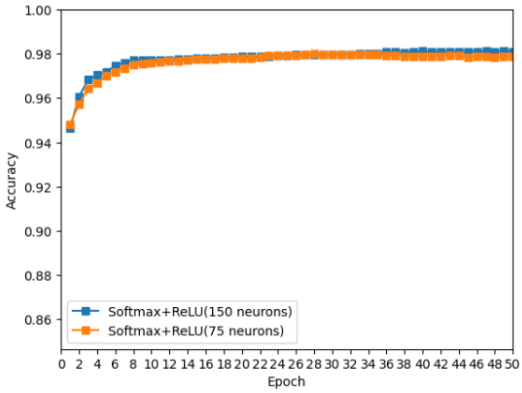
6 Compare Best Performances of One Hidden Layer

We selected the networks with best performances in section 4 and 5 for compare and contrast in this section. They are:

1. MSE+ReLU, Neuron 150, batch 200, epoch 128, learning rate 0.1, weight decay 0.001
2. Softmax+ReLU, Neuron 150, batch 200, epoch 50, learning rate 0.3, weight decay 0.001
3. Softmax+ReLU, Neuron 75, batch 200, epoch 50, learning rate 0.3, weight decay 0.001

Below are diagrams for comparing two networks using Softmax Cross Entropy Loss. The only difference between these two networks is the neuron amount in the hidden layer.





We also construct a list of performance data for 3 networks above.

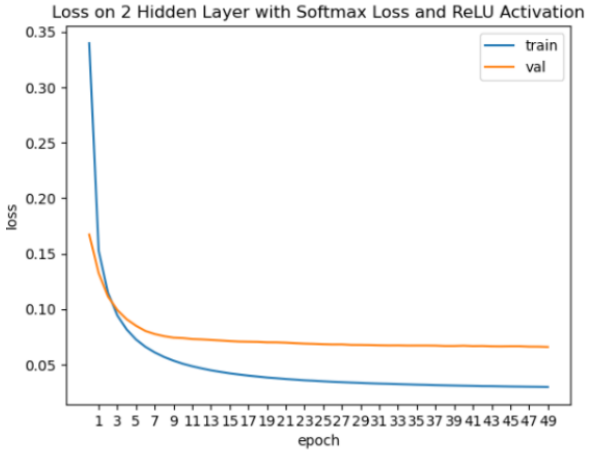
		Train Time (in second)	Mean Validation Loss	Mean Validation Accuracy	Test Accuracy
MSE+ReLU	Neuron 150, batch 200, epoch 128, learning rate 0.1, weight decay 0.001	332.312384	0.244303	0.971234	0.9729
Softmax+ReLU	Neuron 150, batch 200, epoch 50, learning rate 0.3, weight decay 0.001	252.68668	0.0893919	0.9777600	0.9768
Softmax+ReLU	Neuron 75, batch 200, epoch 50, learning rate 0.3, weight decay 0.001	317.891104	0.0951156	0.9766039	0.9738

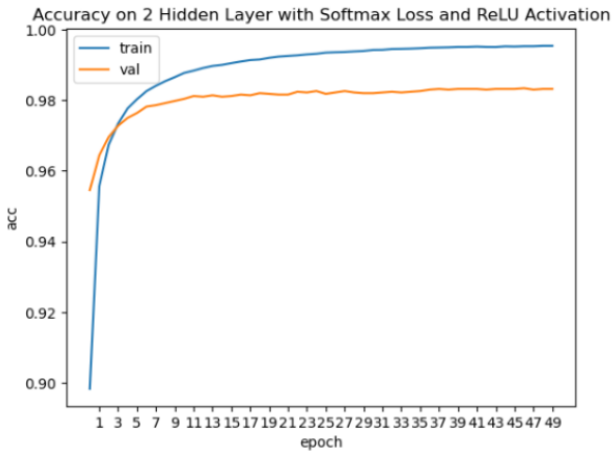
Analyzed from the diagrams above, Softmax+ReLU(Neuron 150, batch 200, epoch 50, learning rate 0.3, weight decay 0.001) has the best performance and lowest training cost. We wull use this one for conclusion, and the test set has accuracy at 0.9768.

7 MLP with Multiple Hidden Layers

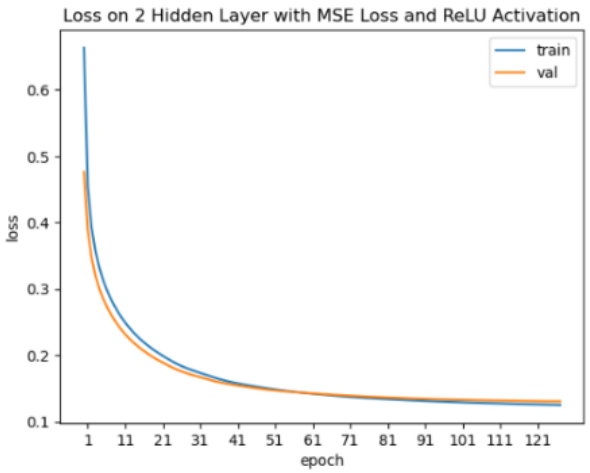
7.1 Two Hidden Layer with Softmax and ReLU Activation

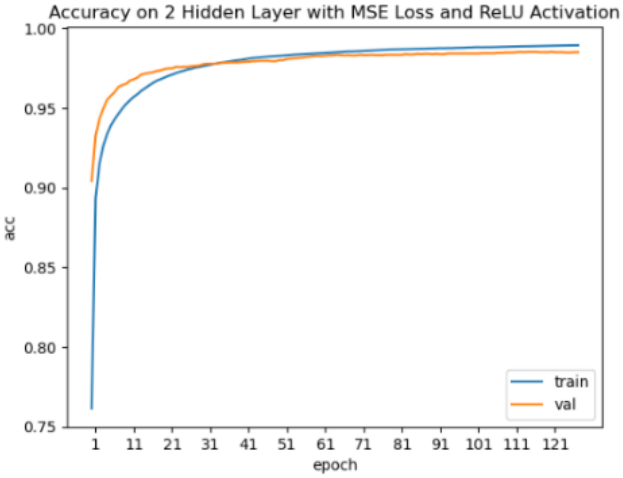
```
reluMLP = Network()
# 使用FCLayer和SigmoidLayer构建多层感知机
# 128为隐含层的神经元数目
reluMLP.add(FCLayer(784, 150))
reluMLP.add(ReLULayer())
reluMLP.add(FCLayer(150, 75))
reluMLP.add(ReLULayer())
reluMLP.add(FCLayer(75, 10))
```





7.2 Two Hidden Layer with MSE and ReLU Activation





With two hidden layers, the network converges much faster than one hidden layer, at about 33. The accuracy and loss also improves. The loss on the train set starts to approximate 0.05 and accuracy starts to approximate 1.0

The accuracy reaches 0.9801 on the test set. We also tested two hidden layers with MSE loss and ReLU activation, and the test accuracy reached 0.9811.

The model with more number of neurons per layer and more number of layers has higher representational capacity, in turn, is capable to map any complicated function or model is capable to extract any level complex features from out input datasets. The advantage of multiple layers is that they can learn features at various levels of abstraction. If we build a very wide, very deep network, we run the chance of each layer just memorizing what we want the output to be, and we end up with a neural network that fails to generalize to new data[11].

8 Conclusion

The optimal model we choose for one hidden layer has the parameters as following: learning rate 0.3, weight decay 0.001, batch size 200 and max epochs 50 with Softmax Cross Entropy loss and ReLU activation. The accuracy on the test set reaches to 0.9768. For two hidden layers, we tested both Softmax Cross Entropy loss and Euclidean loss with ReLU activation. Their accuracy scores both reach 0.98, among which MSE is slightly better with an accuracy at 0.9811.

The balance between batch size, max epoch, learning rate and weight decay, has decisive influences over the learning results. Neuron amount in one hidden layer, that is how wide a layer/model is, also has a huge impact over the training results. If less number of neurons is chosen it will lead to underfitting and high statistical bias. Whereas if we choose too many neurons it may lead to overfitting, high variance, and increases the time it takes to train the network.

In our experiment, ReLU works generally better than Sigmoid. sigmoid outputs are not zero-centered. This is undesirable since neurons in later layers of processing in a Neural Network (more on this soon) would be receiving data that is not zero-centered. This has implications on the dynamics during gradient descent, because if the data coming into a neuron is always positive (e.g. x>0 elementwise in f=wTx+b)), then the gradient on the weights w will during backpropagation become either all be positive, or all negative (depending on the gradient of the whole expression f). This could introduce undesirable zig-zagging dynamics in the gradient updates for the weights.

With more time, we could try to use early-stop strategy and Adam in the MLP and see how the network behaves. Besides, given the two-hidden-layer networks with softmax and MSE have similar accuracy on the test set, we should run tests to check which one is better with statistical significance.

9. References

[1] About OMP: Error #15: Initializing libiomp5md.dll, but found libiomp5md.dll already initialized
https://zhuanlan.zhihu.com/p/371649016

[2] How to Control the Stability of Training Neural Networks With the Batch Size
https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

[3] Training loss decreases, then suddenly increases, then decreases lower than the first time
https://stats.stackexchange.com/questions/512101/training-loss-decreases-then-suddenly-increases-then-decreases-lower-than-the

[4] Understanding Local Minima in Neural-Network Training
https://www.allaboutcircuits.com/technical-articles/understanding-local-minima-in-neural-network-training/

[5] What are the advantages of ReLU over sigmoid function in deep neural networks?
https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks

[6] Tensorflow: loss decreasing, but accuracy stable
https://stackoverflow.com/questions/43499199/tensorflow-loss-decreasing-but-accuracy-stable

[7] Commonly used activation functions
https://liyin2015.medium.com/commonly-used-activation-functions-9821ed348217

[8] How to choose the number of hidden layers and nodes in a feedforward neural network?
https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

[9] Weight Decay and Its Peculiar Effects
https://towardsdatascience.com/weight-decay-and-its-peculiar-effects-66e0aee3e7b8

[10] How does batch size affect convergence of SGD and why?
https://stats.stackexchange.com/questions/316464/how-does-batch-size-affect-convergence-of-sgd-and-why

[11] Why we need Multi Layer Neural Network (MLP)?
https://medium.com/@Rohit_Varma/why-we-need-multi-layer-neural-network-mlp-d50425b8f37d