Two-Layer Neural Network

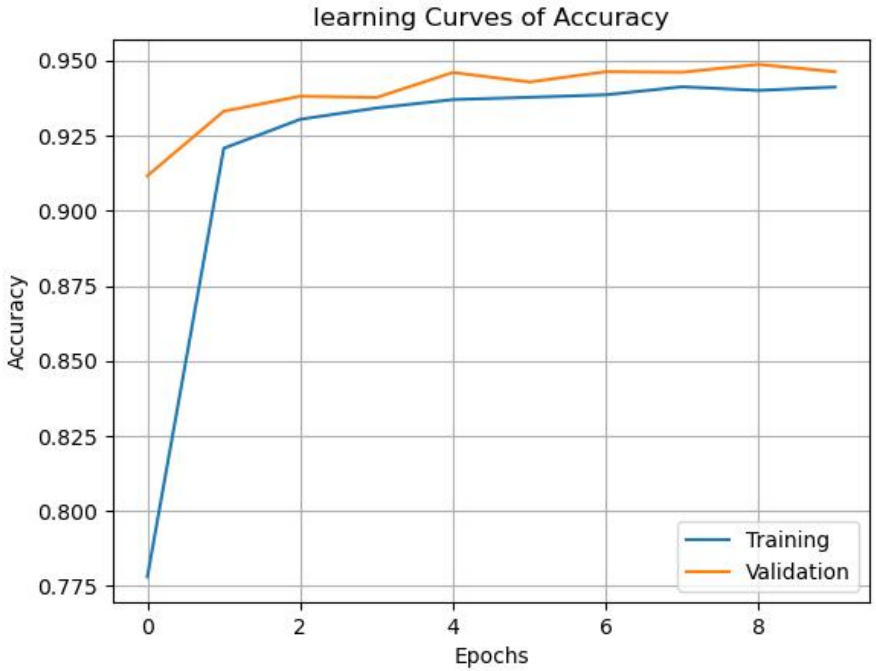
1. Learning Rates

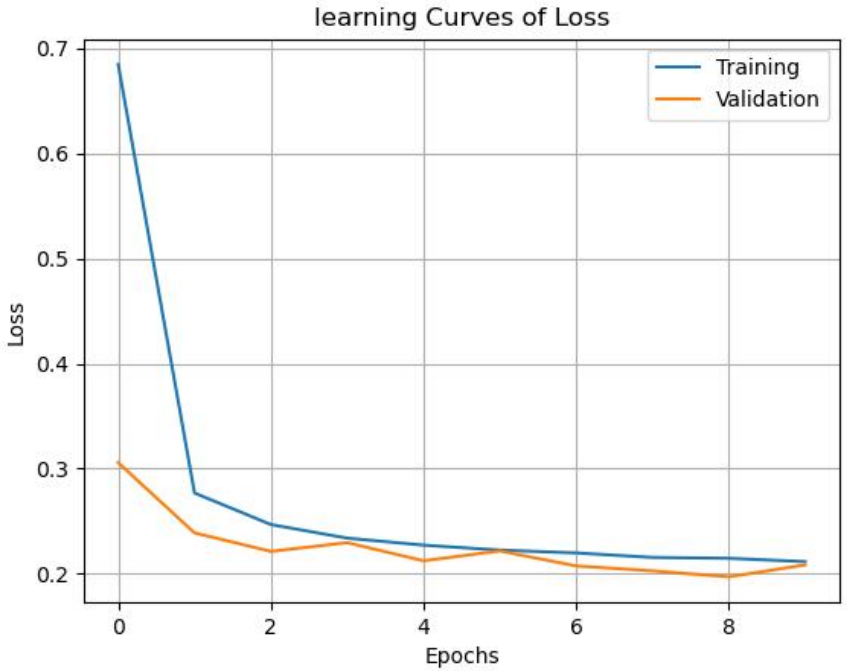
Tune the learning rate of the model with all other default hyper-parameters fixed. Fill in the table below:

	lr=1	lr=1e-1	lr=5e-2	lr=1e-2
Training Accuracy	0.9413	0.9210	0.9080	0.7298
Test Accuracy	0.9480	0.9253	0.9129	0.7561

1. Learning Curve

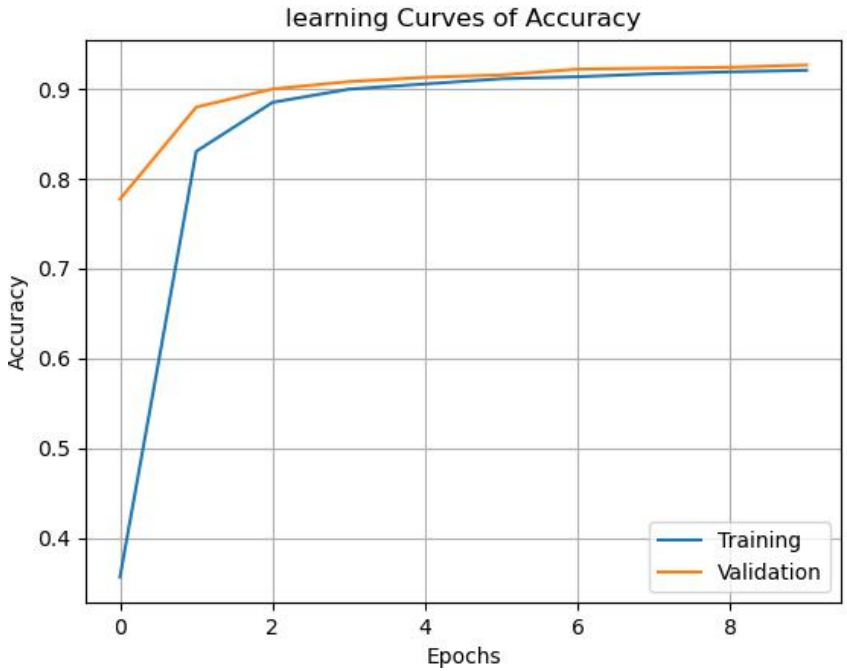
Learning Rate = 1

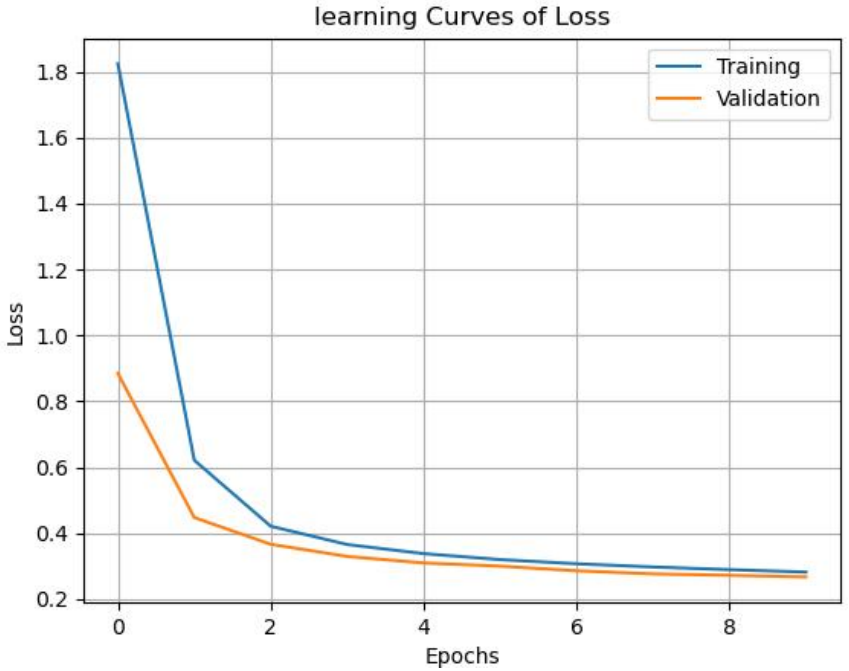




1. Learning Curve

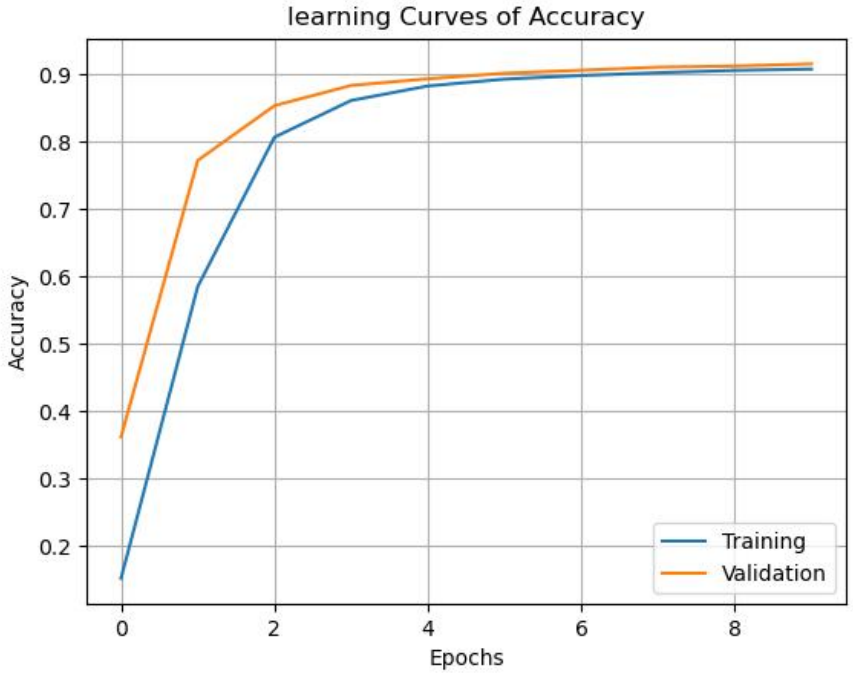
Learning Rate = 0.1

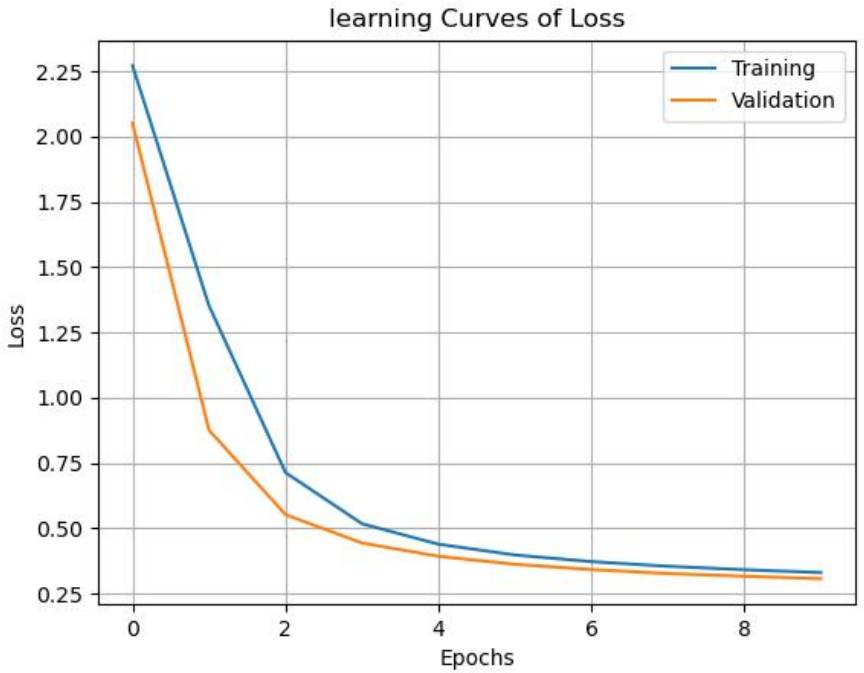




1. Learning Curve

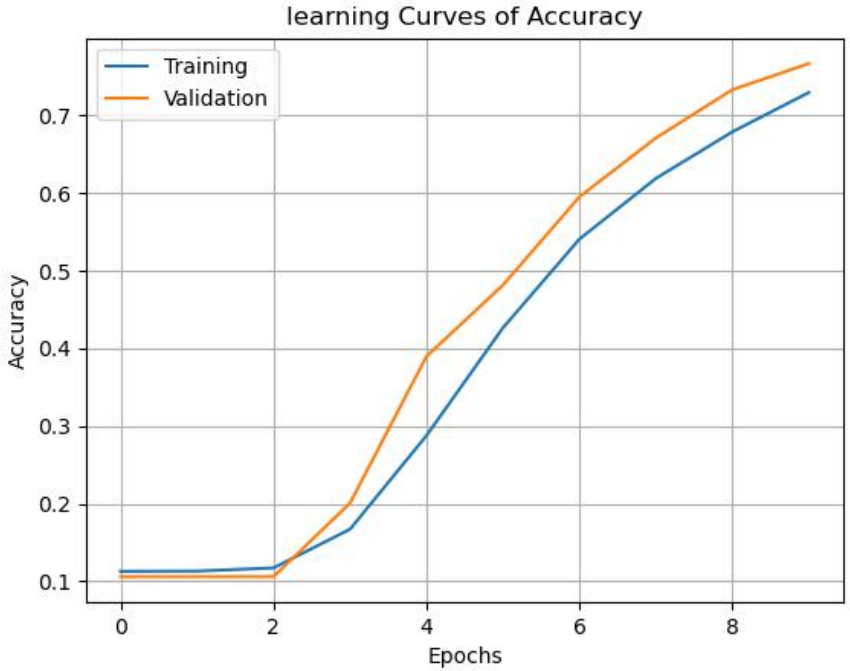
Learning Rate = 0.05

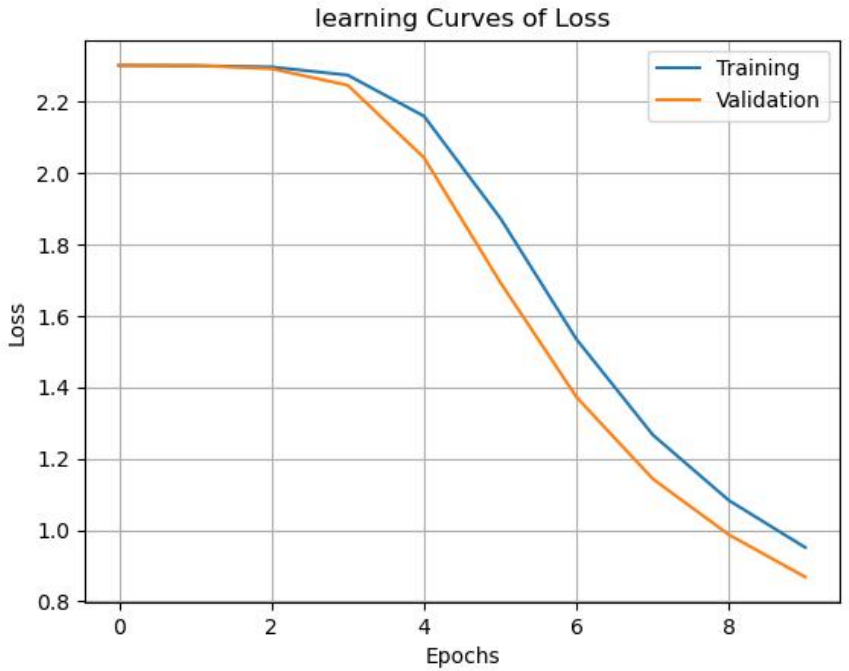




1. Learning Curve

Learning Rate = 0.01





1. Learning Rates

Observation 1: As the learning rate increases from 0.01 to 1, both the training accuracy and test accuracy increases .

Explanation 1:

The vanilla SGD optimizer update rule is as follows:

$$\theta^{t+1} = \theta^t - \eta \nabla_\theta J(\theta)$$

η is the learning rate, θ is the model weight, and $\nabla_\theta J(\theta)$ stands for the gradient for the weight. We can see that for each batch, the weight updates to the opposite direction of the gradient proportionally. The bigger the learning rate is, the more dramatic the update is. The large learning rate let the model be able to learn fast and update the weight quickly according to the gradient. That's why a bigger learning rate shows higher training and test accuracy.

Observation 2: As the epoch number increases, both training and validation accuracy increase, and both the training and validation loss decrease as the epoch increases.

Explanation 2:

As the epoch increase, the model is trained again and again. Each time the weight moves to the opposite direction of the average loss. Over time, the loss will decrease and training and validation accuracy increases.

1. Learning Rates

Observation 3: Compare all learning rate curves, the bigger the learning rate is, the faster the accuracy increases at the beginning, and later on the increasing speeds slow down.
Explanation 3:
Like in explanation 1, a bigger learning rate updates the weight faster, and the learning process speeds up. But over time the loss decrease and the gradient get smaller, the increase of accuracy will slow down and finally converges.

Observation 4: Compare the validation curve and training curve in the same plot, the validation accuracy is higher than the training accuracy, and the validation loss is lower than the training loss.
Explanation 4:
In each epoch, we use the training set to train and update the model weight first, then use the validation set to evaluate the model again. Because the weights are updated in the training process, the validation set will have a lower loss and higher accuracy.

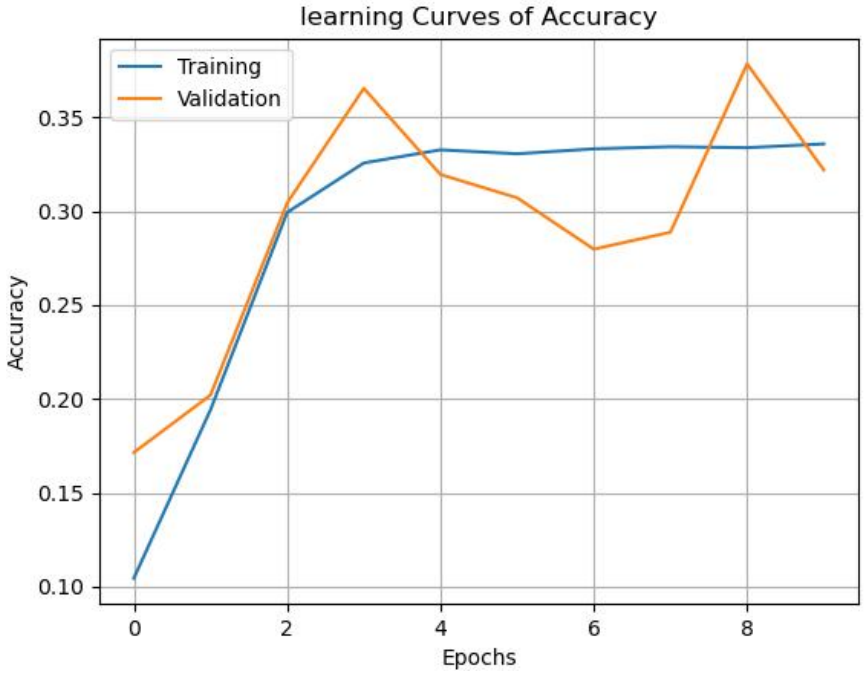
2. Regularization

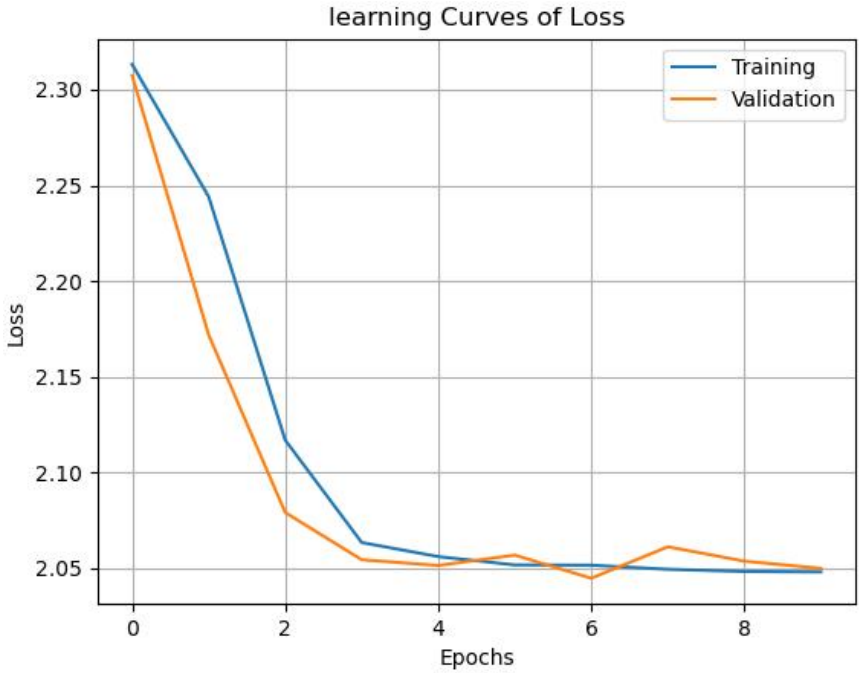
Tune the regularization coefficient of the model with all other default hyper-parameters fixed. Fill in the table below:

	alpha=1	alpha=1e-1	alpha=1e-2	alpha=1e-3	alpha=1e-4
Training Accuracy	0.1008	0.3358	0.8840	0.9210	0.9297
Validation Accuracy	0.0995	0.3219	0.8947	0.9268	0.9331
Test Accuracy	0.1028	0.3846	0.8922	0.9253	0.9326

2. Regularization

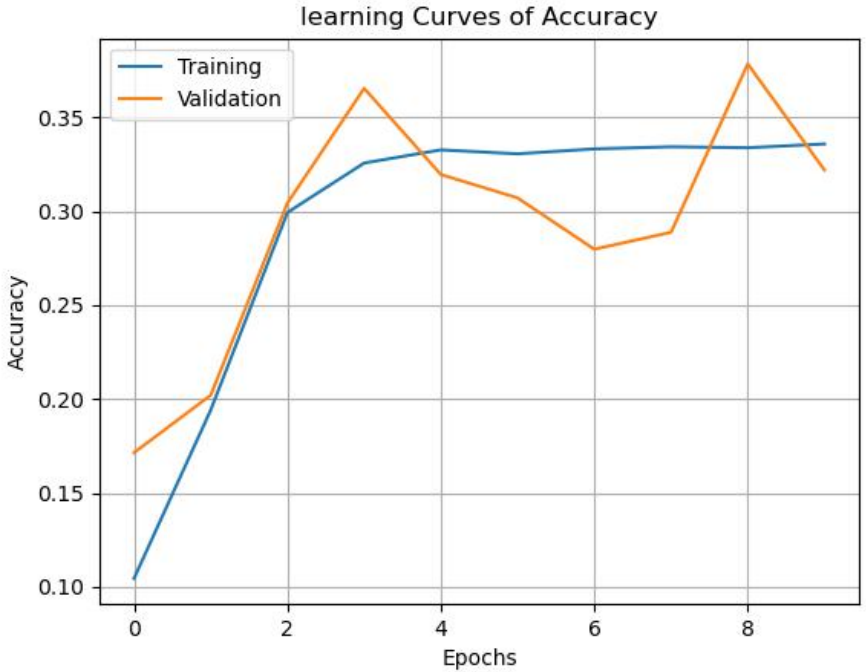
Regularization = 1

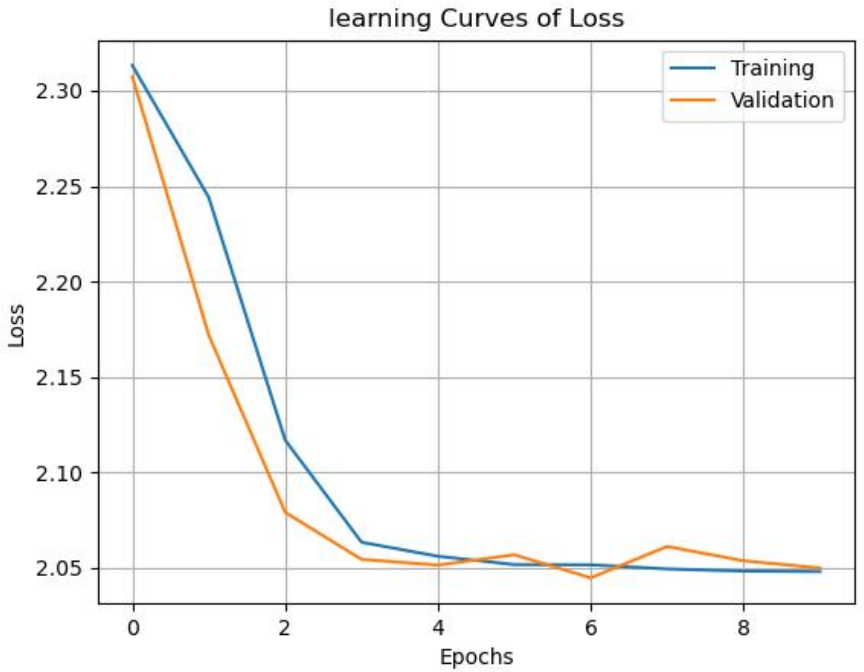




2. Regularization

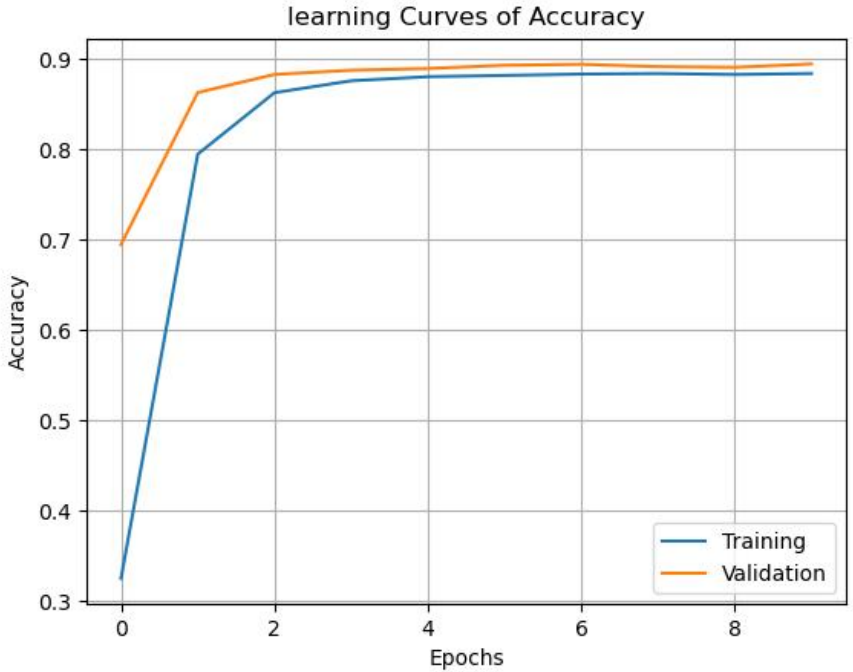
Regularization = 0.1

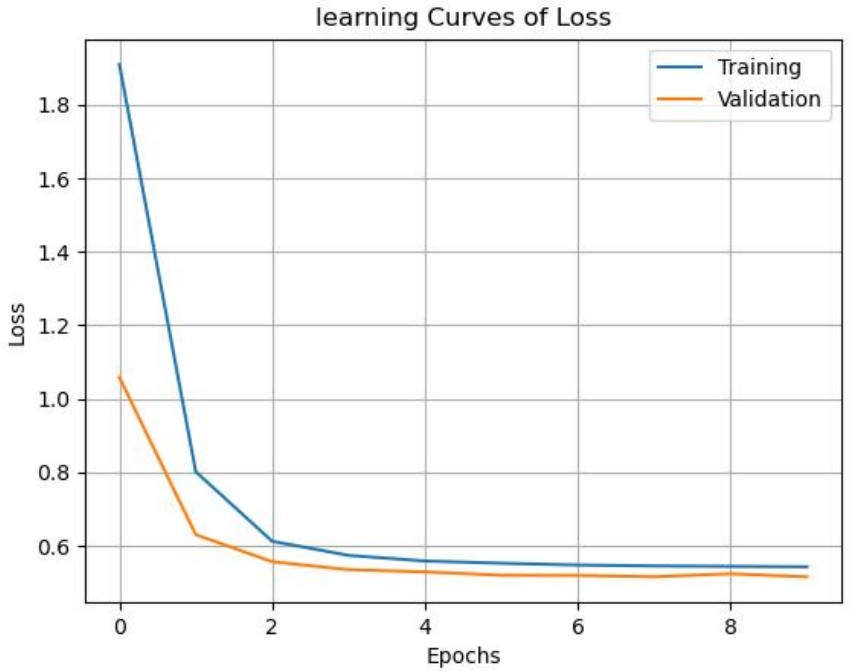




2. Regularization

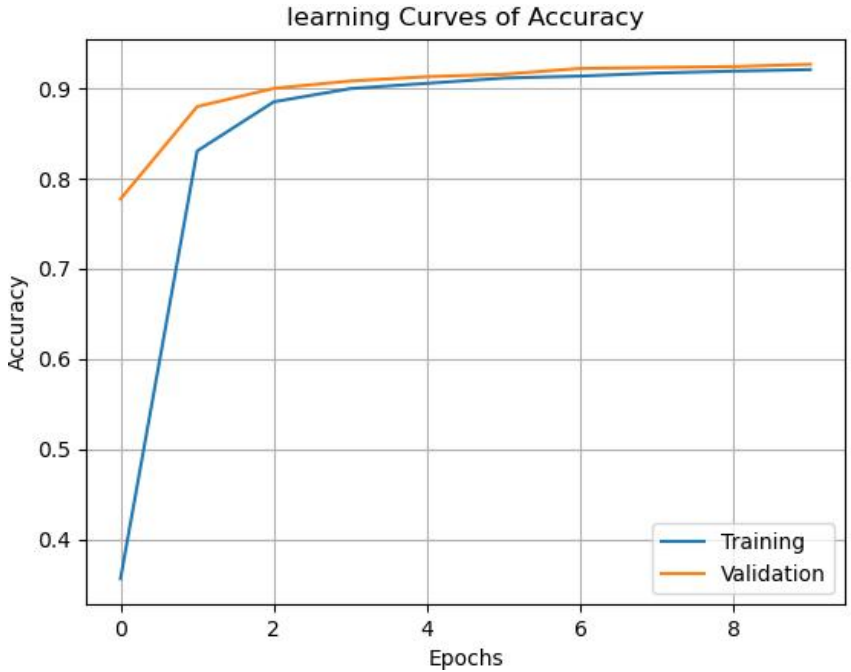
Regularization = 0.01

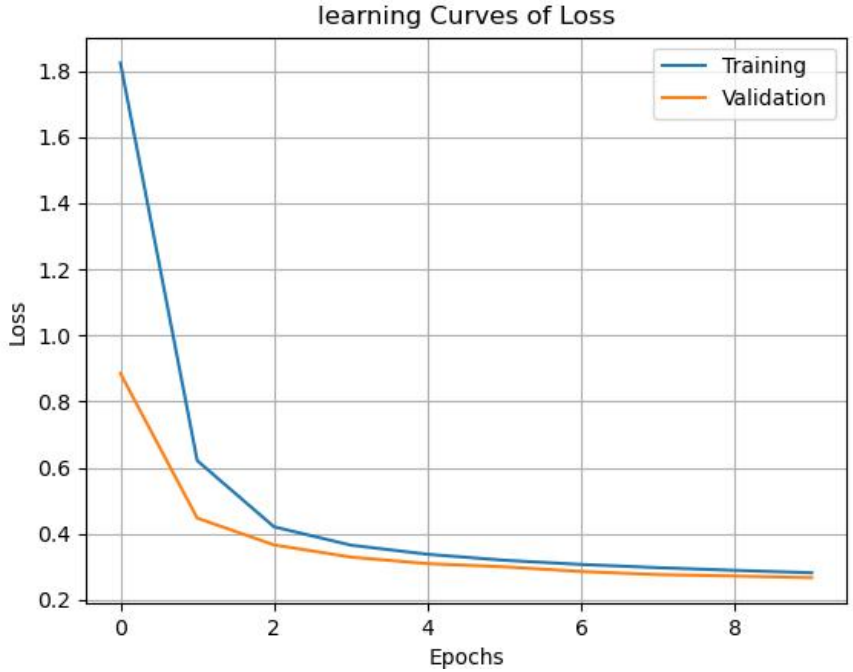




2. Regularization

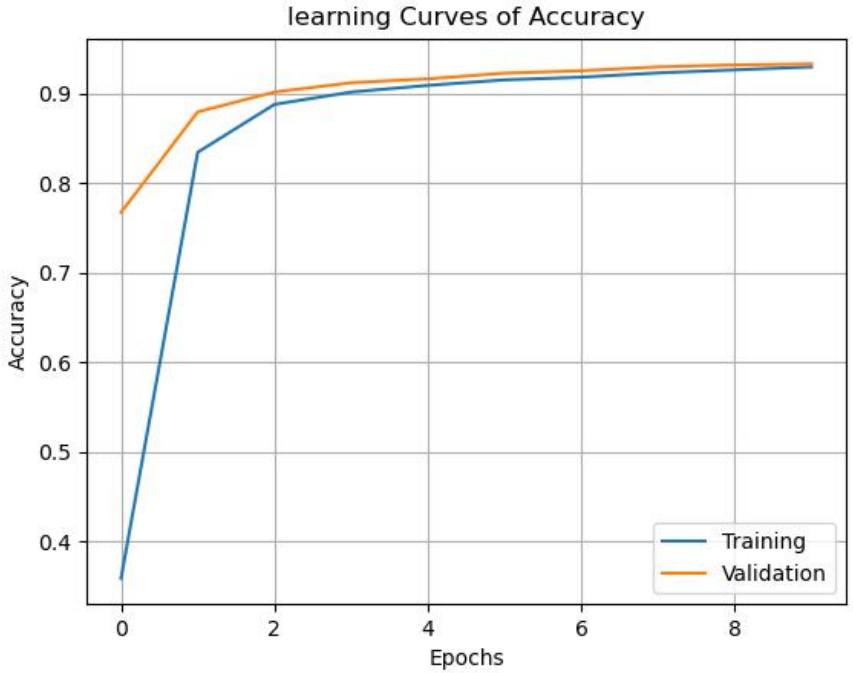
Regularization = 0.001

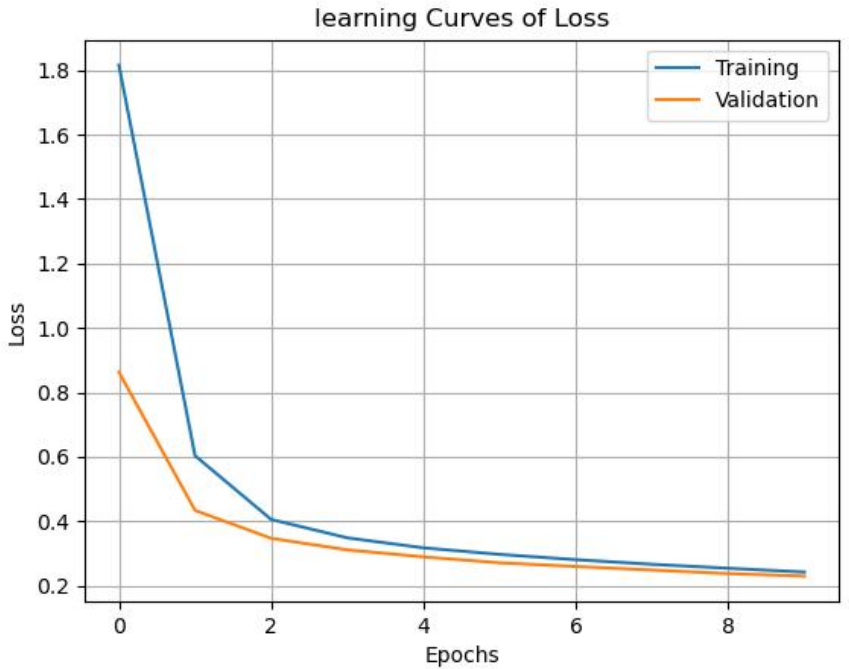




2. Regularization

Regularization = 0.0001





2. Regularization

Observation: As the regularization value decreases from 1 to 0.0001, the training/validation/test accuracy increases .

Explanation:

Before updating the model weights, the optimizer applies L2 regularization on the model by rule:

$$J = L_{CE} + \frac{1}{2}\lambda \sum_{i=1}^{N} {\omega_i}^2$$

The gradient of the parameter $\nabla_\theta J(\theta) = \nabla_\theta L_{CE}(\theta) + \frac{1}{2}\lambda * 2\theta = \nabla_\theta L_{CE}(\theta) + \lambda * \theta$

With the vanilla SGD optimizer update rule, the weight update will be:

$$\theta^{t+1} = \theta^t - \eta(\nabla_\theta L_{CE}(\theta) + \lambda * \theta^t)$$

$\nabla_\theta L_{CE}(\theta)$ is the gradient of the weight from cross-entropy, λ is the regularization value, and θ^t is the weight before the update. From the update rule, we can see that weight will move in the opposite direction of the loss gradient, but the regularization value adds a penalty $\lambda * \theta$ on the weight gradient. The bigger the λ is, the bigger the penalty is. The purpose of L2 regularization is to try to reduce model overfitting by keeping the weight values small, but when the λ is too high, the learning effect is diminished, the model will be too simple and cause the data underfitting. That's the reason why the smaller regularization values have higher training/test accuracy.

Other similar observations as in 1.learning rate part are explained in the learning rate part.

3. Hyper-parameter Tuning

Tune any hyper-parameters for better accuracy.

	lr=0.01 alpha=0.01	lr=0.1 alpha=0.001	lr=1 alpha=0.0001			
	batch size = 64			batch size = 128	batch size = 32	batch size = 16
Training Accuracy	0.6719	0.9209	0.9788	0.9722	0.9804	0.9749
Validation Accuracy	0.7113	0.9259	0.9720	0.9668	0.9713	0.9663
Test Accuracy	0.7008	0.9248	0.9737	0.9680	0.9749	0.9707

Best Accuracy

3. Hyper-parameter Tuning

Firstly, based on the previous experiments on learning rate and regularization value, I get the weight updating rule:

$$\theta^{t+1} = \theta^t - \eta(\nabla_\theta L_{CE}(\theta) + \lambda * \theta^t) = (1 - \eta\lambda)\theta^t - \eta\nabla_\theta L_{CE}(\theta)$$

The weight updates are influenced by both learning rate and regularization value, to study the relationship between these two, I tested 3 groups of hyper-parameter with the same batch size = 64. The parameters are {η=0.01, λ=0.01},{η=0.1, λ=0.001},{η=1, λ=0.0001}. The accuracy results show that the performance ranking for the parameter groups is {η=1, λ=0.0001}>{η=0.1, λ=0.001}>{η=0.01, λ=0.01}, which is what I expected. Because all parameter groups have the same penalty factor on weights ($\eta\lambda$ = 0.0001), and larger learning rate can quickly update the weights according to the loss gradients. So the parameter group {η=1, λ=0.0001} has the best performance.

Then I tuned the batch size parameter of the model, I set the experiment with η=1, λ=0.0001, and set varied batch size 128, 64, 32, 16. After running the experiment, the training and test accuracy show that the performance rank for different batch size is 32 > 64 > 16 > 128. The result is what I expected, the batch size is important in network training. A large batch size usually makes larger gradient steps and it can lead to poor generalization. A small batch size keeps noisy data and offers a regularizing effect which lowers the generalization error, but too small a batch size can also be too noisy and prevent the descent from fully converging to the optima. So a relatively small but not too small batch size can be the most suitable.

3. Hyper-parameter Tuning

Best work learning curve:
learning rate = 1, alpha = 0.0001, batch size = 32, epoch = 10