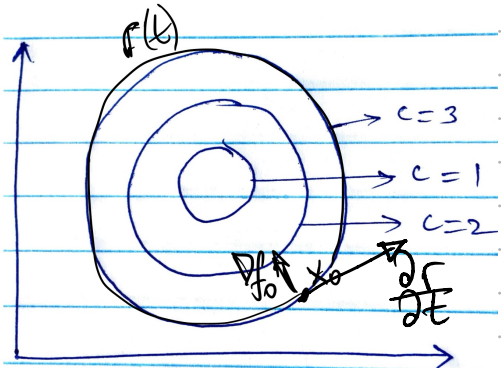
J COLLABORATORS:

Zhaokin Li

2.1

$r(t) = [x_1(t), x_2(t), \dots x_d(t)]$ lies within the level surface and passes through the gradient vector ∇f_0 i.e., $r(t) \in L_{f(x_0)}$ then:



intuitively, we know that the tangent will point out of the surface in the tangent direction of it, in the same plane as the level surface, while the gradient will point to the the direction of greatest increase of the function $\Rightarrow$ orthogonal

formally: $\frac{\partial r}{\partial t} = \left[\frac{d}{dt}x_1(t), \frac{d}{dt}x_2(t), \dots \frac{d}{dt}x_d(t)\right]$

$$\nabla f_0 = \left[\frac{\partial f_0}{\partial x_1}, \frac{\partial f_0}{\partial x_2}, \dots \frac{\partial f_0}{\partial x_d}\right]$$

at $t = t_0$: $\frac{\partial r(t_0)}{\partial t} = \left[\frac{d}{dt}x_1(t_0), \frac{d}{dt}x_2(t_0), \dots \frac{d}{dt}x_d(t_0)\right] =$

$$= \frac{\partial x_0}{\partial t} = \left[\frac{dx_{01}}{dt}, \frac{dx_{02}}{dt}, \frac{dx_{0d}}{dt}\right]$$

$$\left\langle \frac{\partial x_0}{\partial t}, \nabla f_0 \right\rangle = \left[\frac{dx_{01}}{dt}\frac{\partial f_0}{\partial x_1}, \frac{dx_{02}}{dt}\frac{\partial f_0}{\partial x_2}, \dots \frac{\partial x_{0d}}{\partial t}\frac{\partial f_0}{\partial x_d}\right]$$

$$= [0, 0, \dots, 0] \Rightarrow \frac{\partial x_0}{\partial t} \text{ \& } \nabla f_0 \text{ are ortho-gonal}$$

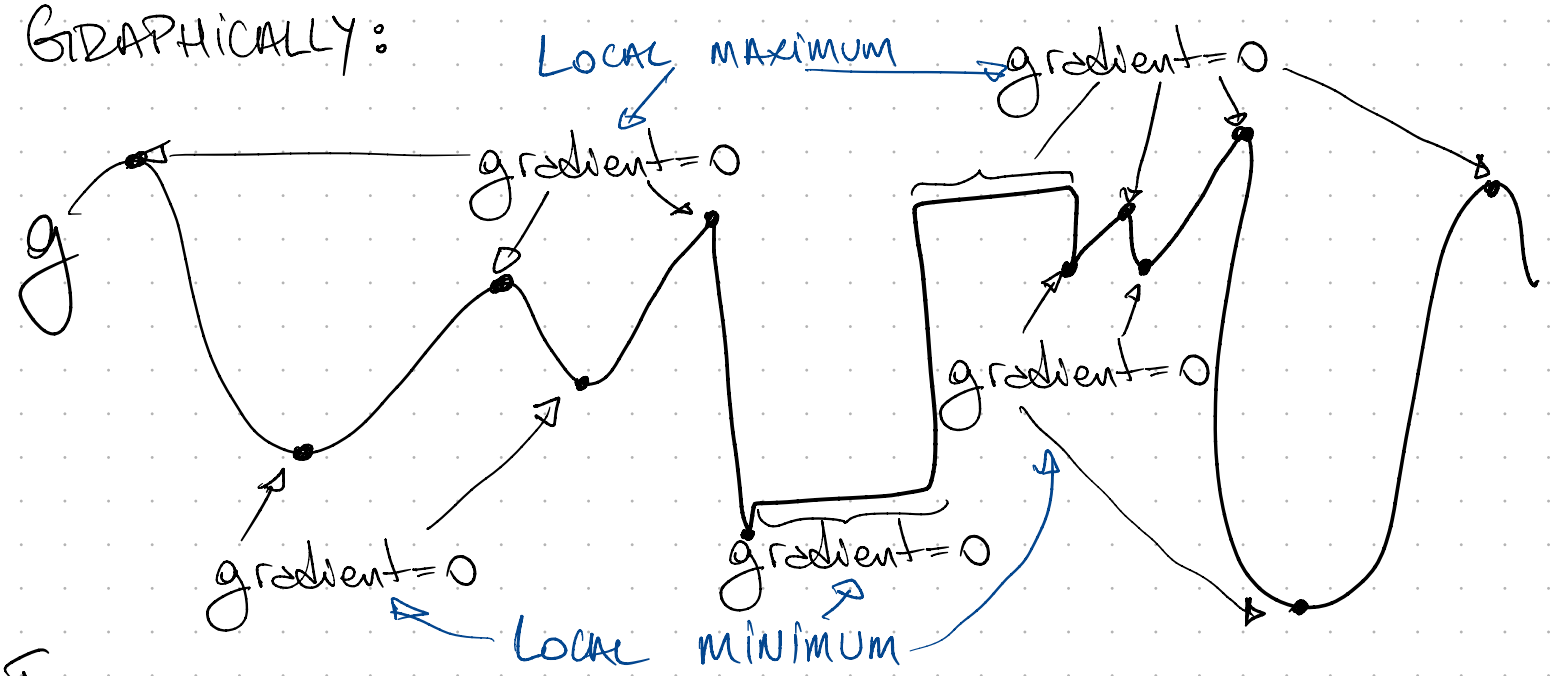
2.1 — continued

I've just proven that the gradient of an n-d function points towards where the func. increases / decreases the most, i.e. where there is most change per step relative to level surfaces.

This is key in DL because it all boils down to optimization, which is basically finding minima. With that info in hand we know that the gradient of a function is what we want to compute to be able to find said minima/um.

2.2 if $\exists\ \gamma > 0$ s.t. $\|w^T - w\|_2 < \gamma\ \forall\ w^T \in \mathbb{R}^d$ and $\|w^T - w\|_2 < \gamma \Rightarrow g(w^T) \leq g(w) \Rightarrow \exists$ local minimum

GRAPHICALLY:



FORMALLY:

let $w^T = [w_1, w_2, \dots w_n]^T$ take the gradient:

$$\nabla g(w^T) = \begin{bmatrix} g_{w_1}(w_1, w_2, \dots w_n) \\ g_{w_2}(w_1, w_2, \dots w_n) \\ \vdots \\ g_{w_n}(w_1, w_2, \dots w_n) \end{bmatrix}$$

if g has local minimum at w^T, by the definition of derivatives & minima:

$$\nabla g(w^T) = \begin{bmatrix} g_{w_1}(w_1, w_2, \dots w_n) \\ g_{w_2}(w_1, w_2, \dots w_n) \\ \vdots \\ g_{w_n}(w_1, w_2, \dots w_n) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

2.2 continued

which proves that if $\exists$ local minimum at some w^T then the gradient at $w^T = 0$.

the contrary is not true because $\nabla g(w^T) = 0$ at local maximums and saddle points, e.g.:

$g = -w^2 \longrightarrow w^T = 0 \Rightarrow \nabla g = -2w \Rightarrow \nabla g = 0$

$\therefore \nabla g(w^T) = 0$ is sufficient, but does not guarantee that w^T is a local minimum ■

2.3 We know the definition of local minimum irrespective of convexity from the previous question:

if $\exists\ \gamma > 0$ s.t. $\|w^* - w\|_2 < \gamma\ \forall\ w^T \in \mathbb{R}^d$ and $\|w^* - w\|_2 < \gamma \Rightarrow g(w^*) \leq g(w) \Rightarrow \exists$ local minimum

$\nabla g(w^*) = 0$ (conv. properties $\downarrow$ $\forall\ \delta \in (0, 1]$:

$$g((1-\delta)w^* + \delta w) \leq (1-\delta)g(w^*) + \delta g(w)$$

$$< (1-\delta)g(w^*) + \delta g(w^*) = g(w^*)$$

for small $\delta \Rightarrow \|(1-\delta)w^* + \delta w - w^*\| < \gamma \Rightarrow$ global minimum ■

2.4 $S(z)$: $s_i = \frac{e^{z_i}}{\sum_k e^{z_k}}$ Derive $\frac{\partial S}{\partial z}$

$$\frac{\partial s_i}{\partial z_j} = \frac{\partial \frac{e^{z_i} g}{\sum_k e^{z_k} h}}{\partial z_k} \longrightarrow \quad g' = \begin{cases} e^{z_k} & \text{if } i=j \\ 0 & \text{otherwise} \end{cases} \qquad f' = \frac{g'h - h'g}{h^2}$$

$$h' = e^{z_k}$$

$\frac{\partial s_i}{\partial z_j} \neq$ (Jacobian) MATRIX. Diag. $\Rightarrow$ $\Rightarrow i=j$

$$= \frac{g' \sum_k e^{z_k} - e^{z_k} e^{z_i}}{\left(\sum_k e^{z_k}\right)^2} \quad (I)$$

First, let's consider $i=j$

then from (I) and g': $\frac{\partial s_i}{\partial z_j} = \frac{e^{z_j} \sum_k e^{z_k} - e^{z_j} e^{z_i}}{\left(\sum_k e^{z_k}\right)^2}$

$$= \frac{e^{z_j}\left(\sum_k e^{z_k} - e^{z_i}\right)}{\left(\sum_k e^{z_k}\right)^2} = \underbrace{\frac{e^{z_j}}{\sum_k e^{z_k}}}_{\text{softmax}(j)} \cdot \underbrace{\frac{\sum_k e^{z_k} - e^{z_i}}{\sum_k e^{z_k}}}_{1 - \text{softmax}(i)}$$

$\frac{\partial s_i}{\partial z_j}$ (for $i=j$) $= (1 - s(j))\, s(i)$ $\rightsquigarrow$ Diag. of Jacobian

For $i \neq j$:

$$\frac{\partial s_j}{\partial z_j} = \frac{e^{z_i} \cancel{\sum_k e^{z_k}}^{\to 0} - e^{z_j} e^{z_i}}{\left(\sum_k e^{z_k}\right)^2} = -\underbrace{\frac{e^{z_j}}{\sum_k e^{z_k}}}_{\text{softmax}(j)} \cdot \underbrace{\frac{e^{z_i}}{\sum_k e^{z_k}}}_{\text{softmax}(i)} = -s(j)\, s(i)$$

2.4 — continued

$$\therefore \frac{\partial s}{\partial z} = \begin{cases} (1 - s(j))\, s(i) & \text{if } i = j \\ -s(j)\, s(i) & \text{else} \end{cases}$$

3.6

$G = (V, E)$; $V = \{v_1, v_2, \dots v_i, \dots v_n\}$; $E = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in V\}$

$G_+ = G + v_{n+1}$ is also DAG. Simmilarly, $G_- = G - v_n$ is also DAG

$G_+ = G - v_{n+1} \Rightarrow$ Edge $e_{n,n+1} = (v_n, v_{n+1})$ has to be added to E, now that V was added with v_{n+1}

$G_- = G - v_{n-1} \Rightarrow e_{n-1,n} = (v_{n-1}, v_n)$ has to be deleted from E because V was updated by deleting v_n

$G_- = G - v_{i-1} \Rightarrow e_{i-2,i-1} = (v_{i-2}, v_{i-1})$ and $e_{i-1,i} = (v_{i-1}, v_i)$ are deleted ; $e_{i-2,i} = (v_{i-2}, v_i)$ is added

Let $i = 1 \Rightarrow e_{0,1}$ and $e_{1,2}$ are deleted

$v_0 \notin V \Rightarrow e_{0,2} = (v_0, v_2)$ can not be added $\Rightarrow$

$\Rightarrow G_-$ doesn't have incomming edge $\Rightarrow$ there is topological order ■

3.7 Given the previous proof:

- each time a node is added, E_+ increases
- " " " " deleted, E_- decreases

the same goes for V_+ & V_- (in # of elements)

∴ When a node is added, it has a topological place. If it did not then it would be possible that the added edges are cyclical, e.g. $e_{new} = (v_i, v_i) \Rightarrow G$ is a DAG ■

CS 4644/7643: Deep Learning
Spring 2024
HW1 Solutions

Arthur Scaquetti do Nascimento

February 5, 2024

4.8)

- Key contributions: Even though this is not the first attempt to optimize NN architectures, the authors bring a neuroscientific perspective and leverage facts that strong inductive biases can lead to innate learning to develop an evolution-based search strategy for optimal NN architectures.

- Strengths: The optimization is done on a network-design level, and using genetic algorithms heuristics; therefore, there is no need to compute gradients, which is one of the bottlenecks in Deep Learning. Moreover, this is done grounded in neuroscience findings, instead of a common practice trial-and-error approach.

- Weaknesses: Even though the approach is novel, this is not the first work to tackle the need for smarter network designing, and does not outperform all previous work, such as Zhou, et al. (2019) [1]. The authors also could have incorporated pruning methods and the formulation of search for architectures in a differentiable manner to enhance their evolutionary search strategy.

4.9) I am curious to see how this method would perform in more complex RL environments (e.g., 7DoF manipulator, Deep Brain Stimulation, Minecraft, etc.) either for the direct generation of the policy as they did in the 2D environments, or as a "warm-start", namely the search for the best architecture to encapsulate the behavior of a Q-function. Especially with respect of training time and robustness to out-of distribution priors (could be treated as zero-shot). If it works well, it could mean that this should be step no.1 for RL practitioners.
Moreover, the authors criticize meta-learning, but I also wonder how applicable this is to the MAML model and other meta-learned policies.

[1] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. arXiv preprint arXiv:1905.01067, 2019. https://arxiv.org/abs/1905.01067.

In [ ]:
```
#Cell 1
from google.colab import drive
drive.mount('/content/drive')
```

In [ ]:
```
#Cell 2
%cd /content/drive/MyDrive/'hw1.zip (Unzipped Files)'/hw1/student_version/data # or your custom p
!sh get_data.sh
%cd ..
```

In [ ]:
```
# Cell 3
# Run all local tests in this block
# If you get an error saying test not found, add an __init__.py file in the
# tests directory
!python -m unittest tests.test_network
```

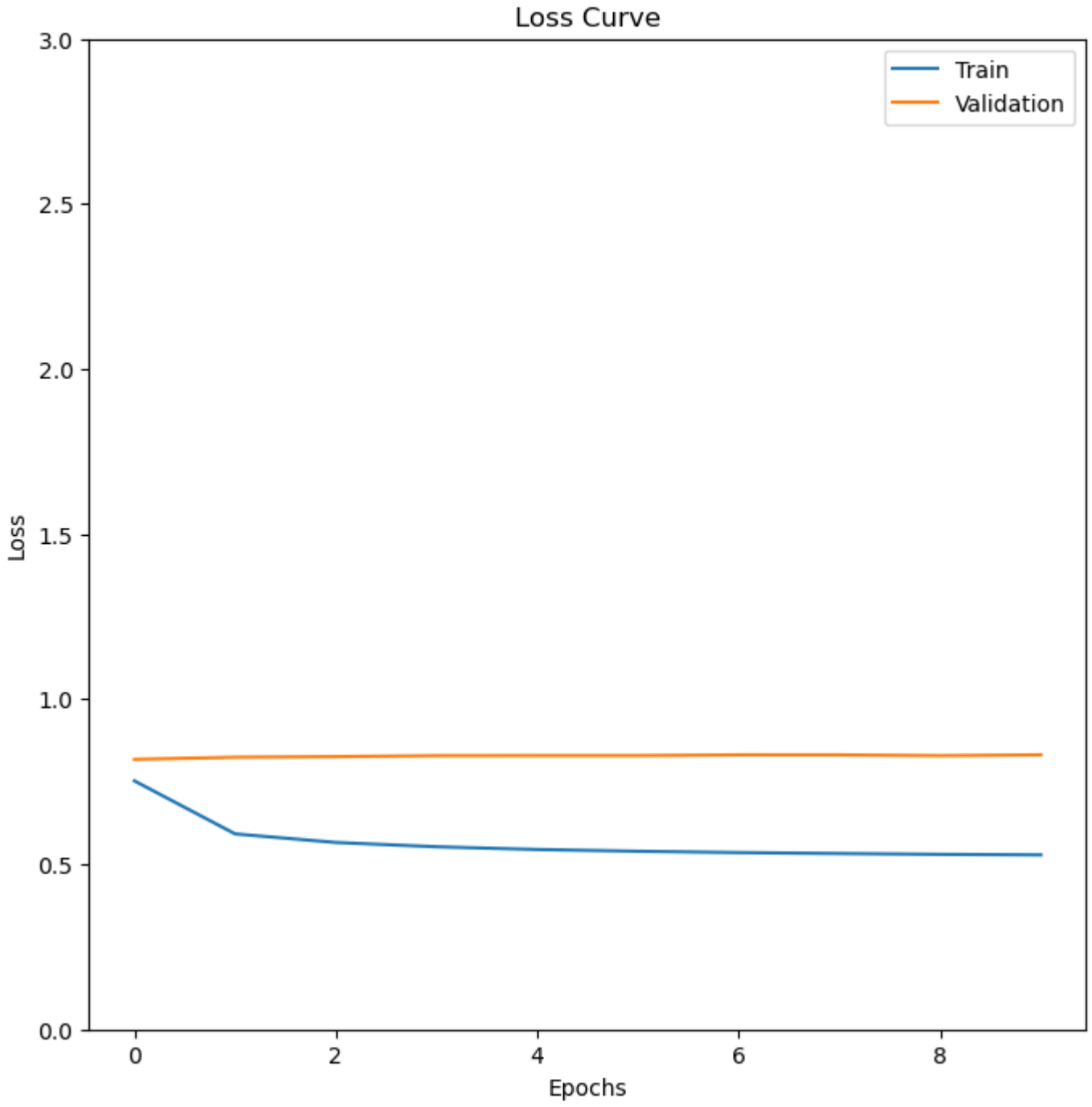
In [ ]:
```
#Cell 4
import yaml
import copy

from models import TwoLayerNet, SoftmaxRegression
from optimizer import SGD
from utils import load_mnist_trainval, load_mnist_test, generate_batched_data, train, evaluate, p
```

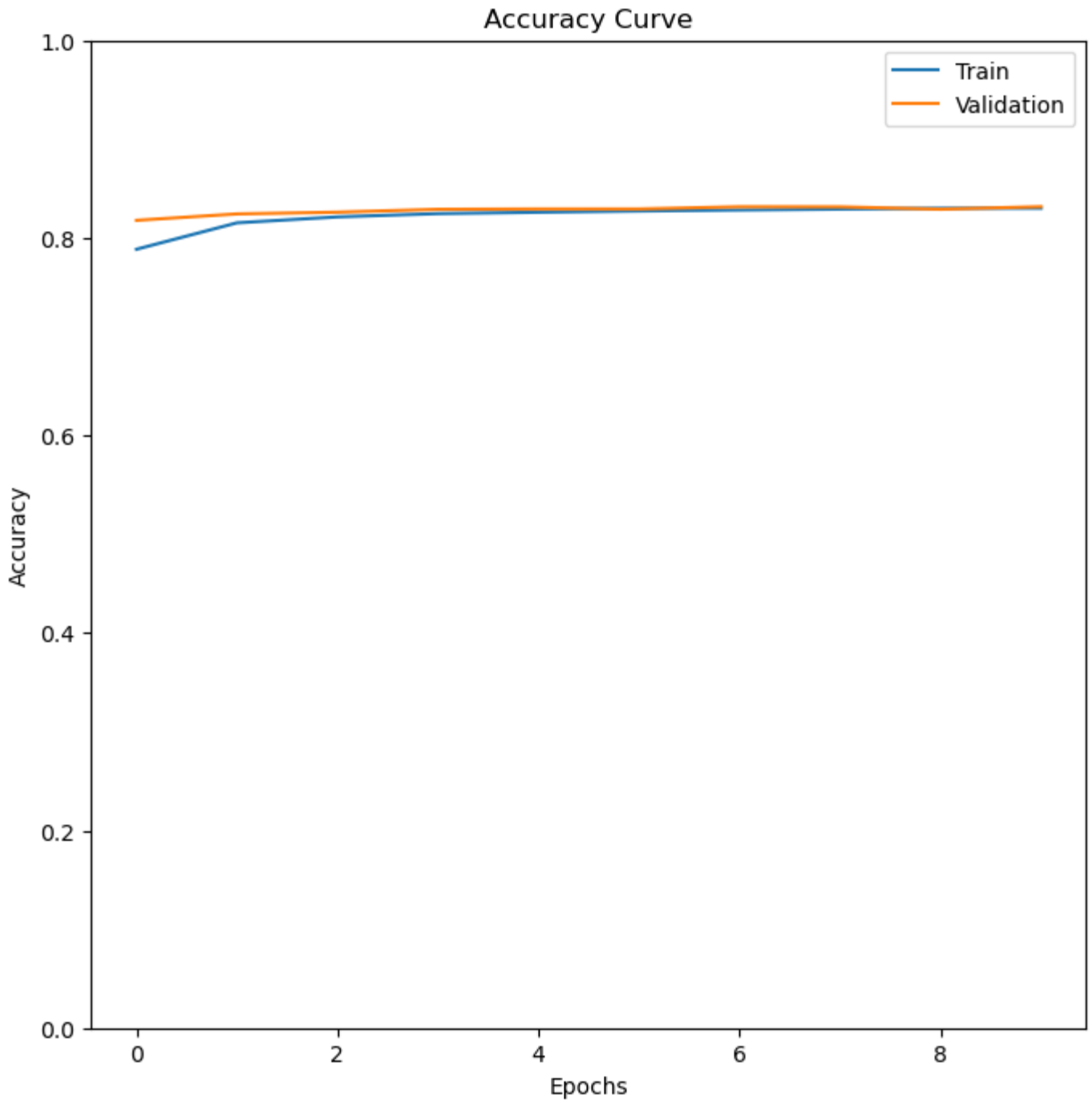
In [ ]:

```
# Cell 5
%matplotlib inline
def train_model(yaml_config_file):
  args = {}
  with open(yaml_config_file) as f:
      config = yaml.full_load(f)

  for key in config:
      for k, v in config[key].items():
          args[k] = v

  # Prepare MNIST data
  train_data, train_label, val_data, val_label = load_mnist_trainval()
  test_data, test_label = load_mnist_test()

  # Prepare model and optimizer
  if args["type"] == 'SoftmaxRegression':
      model = SoftmaxRegression()
  elif args["type"] == 'TwoLayerNet':
      model = TwoLayerNet(hidden_size=args["hidden_size"])
  optimizer = SGD(learning_rate=args["learning_rate"], reg=args["reg"])

  # Training Code
  train_loss_history = []
  train_acc_history = []
  valid_loss_history = []
  valid_acc_history = []
  best_acc = 0.0
  best_model = None
  for epoch in range(args["epochs"]):
      batched_train_data, batched_train_label = generate_batched_data(train_data, train_label, ba
      epoch_loss, epoch_acc = train(epoch, batched_train_data, batched_train_label, model, optim

      train_loss_history.append(epoch_loss)
      train_acc_history.append(epoch_acc)
      # evaluate on test data
      batched_test_data, batched_test_label = generate_batched_data(val_data, val_label, batch_s
      valid_loss, valid_acc = evaluate(batched_test_data, batched_test_label, model, args["debug
      if args["debug"]:
          print("* Validation Accuracy: {accuracy:.4f}".format(accuracy=valid_acc))

      valid_loss_history.append(valid_loss)
      valid_acc_history.append(valid_acc)

      if valid_acc > best_acc:
          best_acc = valid_acc
          best_model = copy.deepcopy(model)

  #Testing Code
  batched_test_data, batched_test_label = generate_batched_data(test_data, test_label, batch_size
  _, test_acc = evaluate(batched_test_data, batched_test_label, best_model) # test the best model
  if args["debug"]:
      print("Final Accuracy on Test Data: {accuracy:.4f}".format(accuracy=test_acc))

  return train_loss_history, train_acc_history, valid_loss_history, valid_acc_history
```

In [ ]:
```
# Cell 6
# train softmax model
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

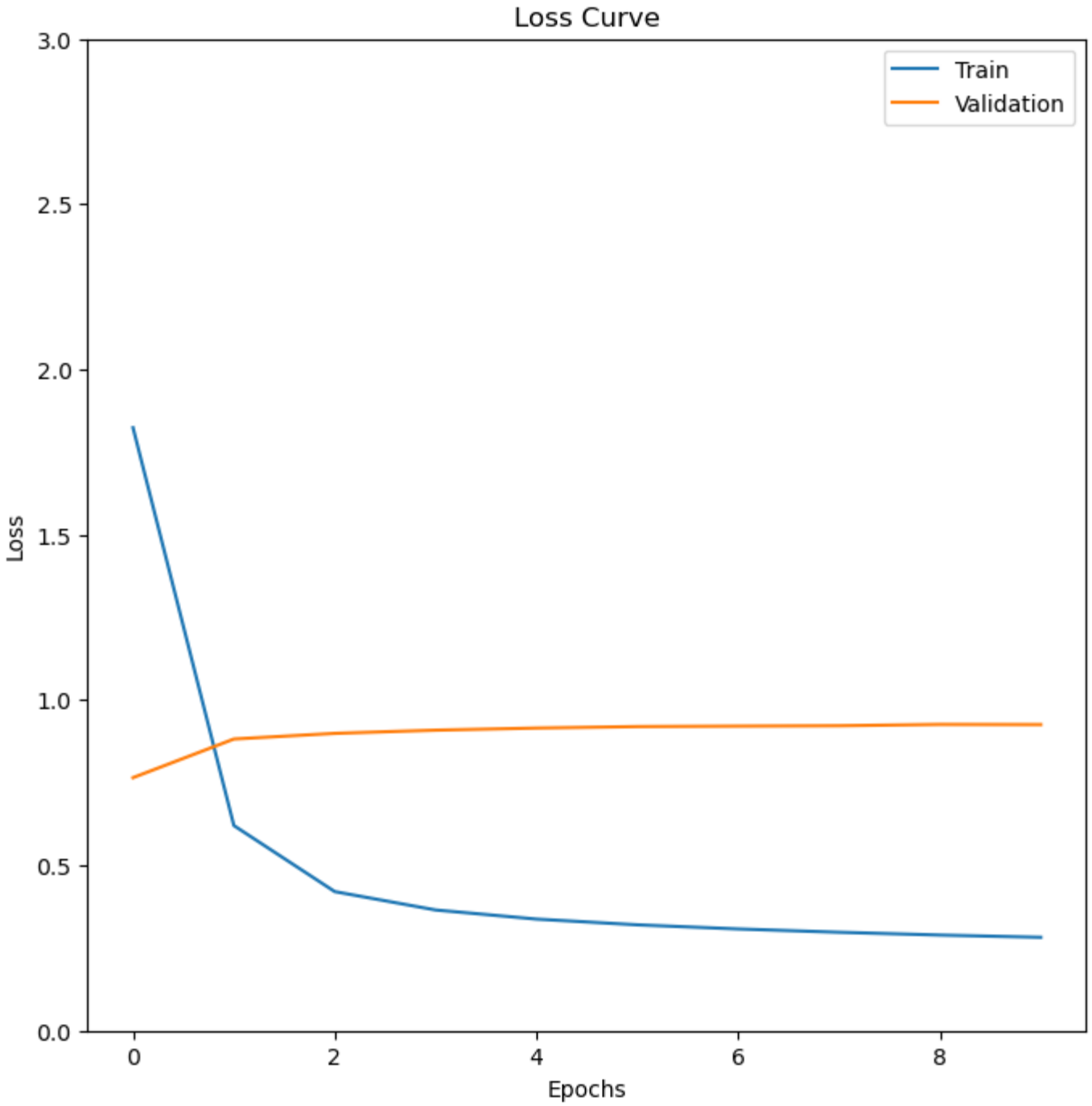
In [ ]:
```
# Cell 7
# plot results for softmax model
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

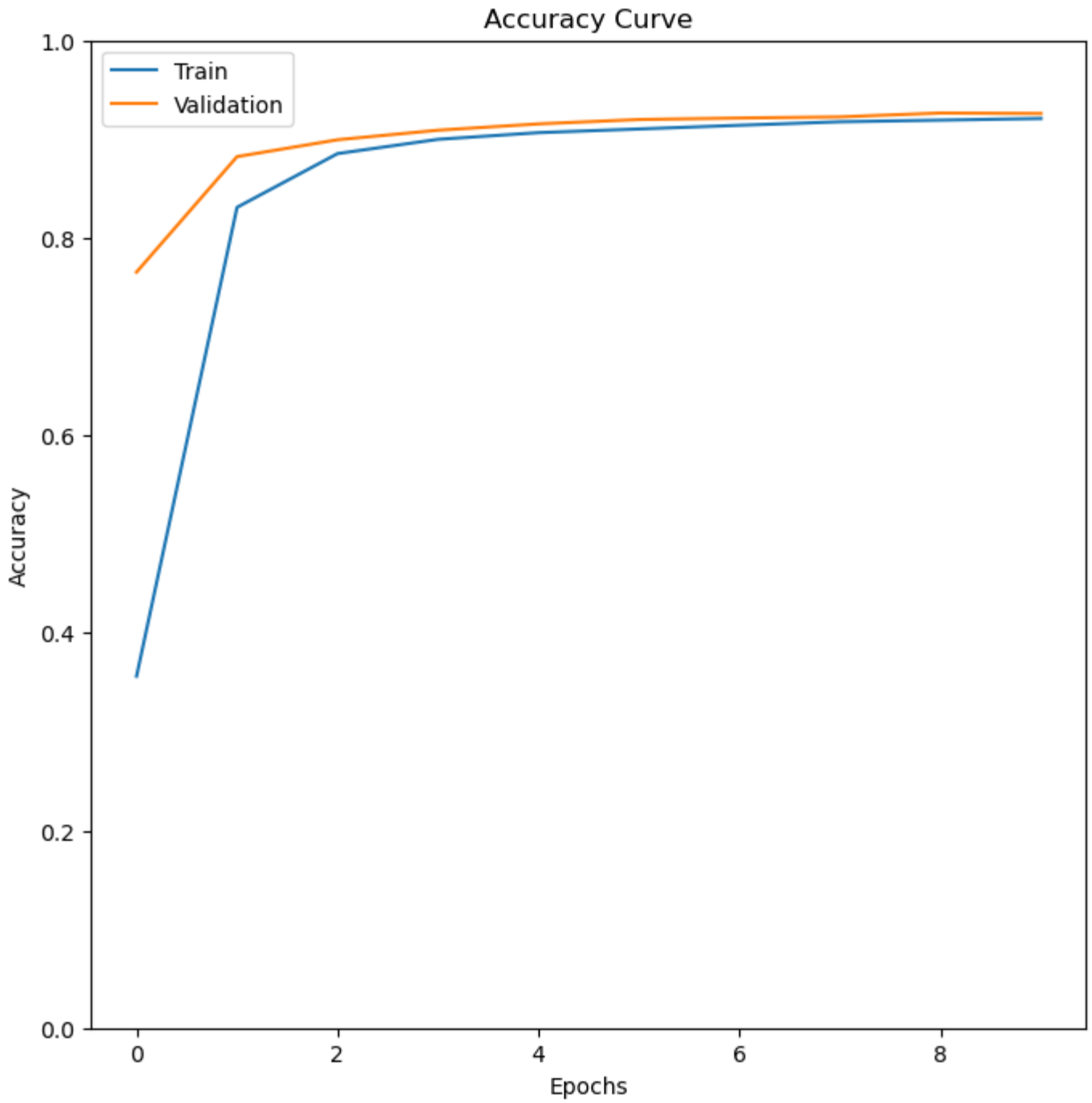




In [ ]:
```
# Cell 8
# train two layer neural network
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

In [ ]:
```
# Cell 9
# plot two layer neural network
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```


Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



Assignment 1 Writeup

- Name: Arthur Scaquetti do Nascimento
- GT Email: anascimento7@gatech.edu
- GT ID: 903721549

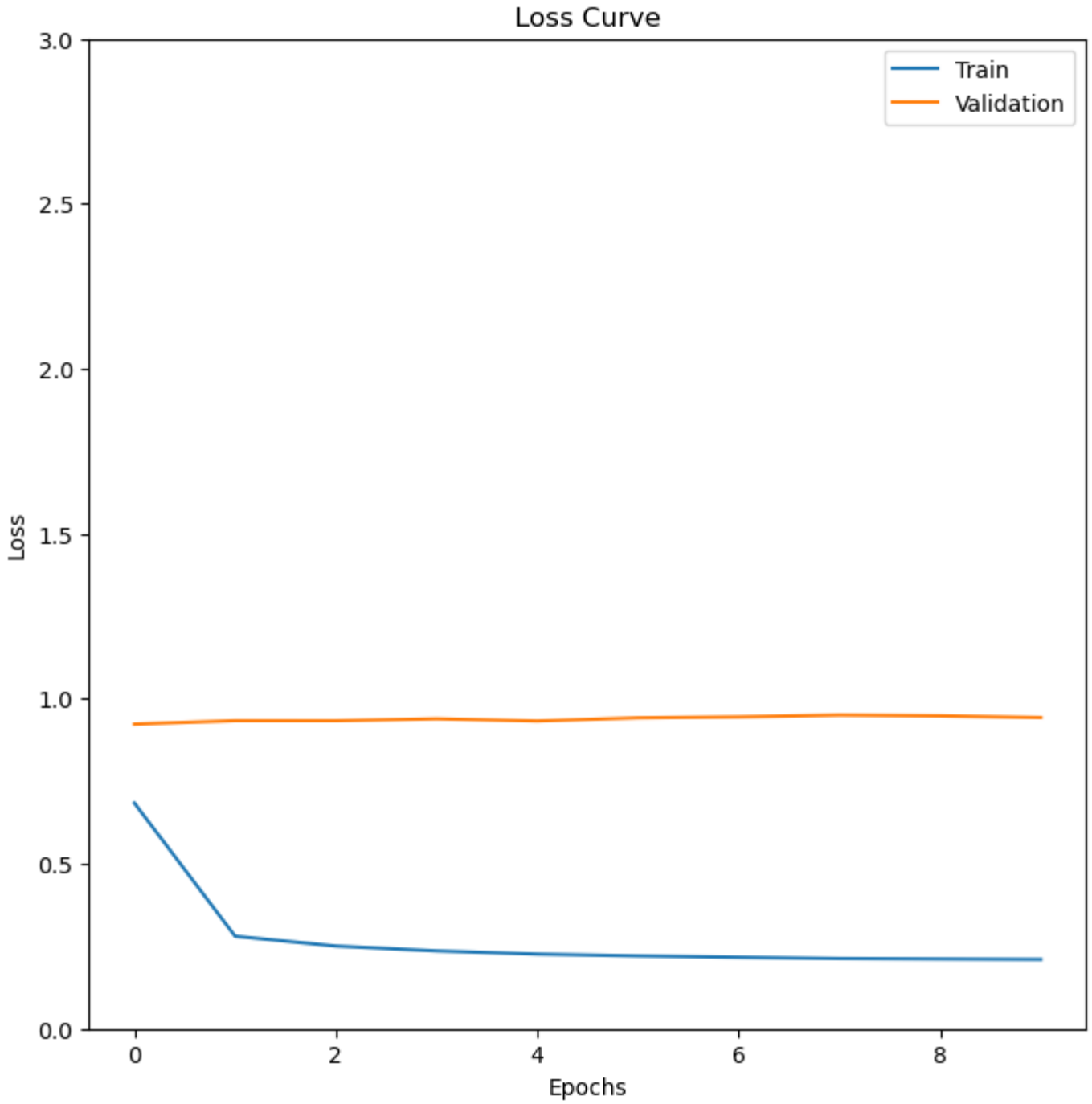
Two Layer Neural Network

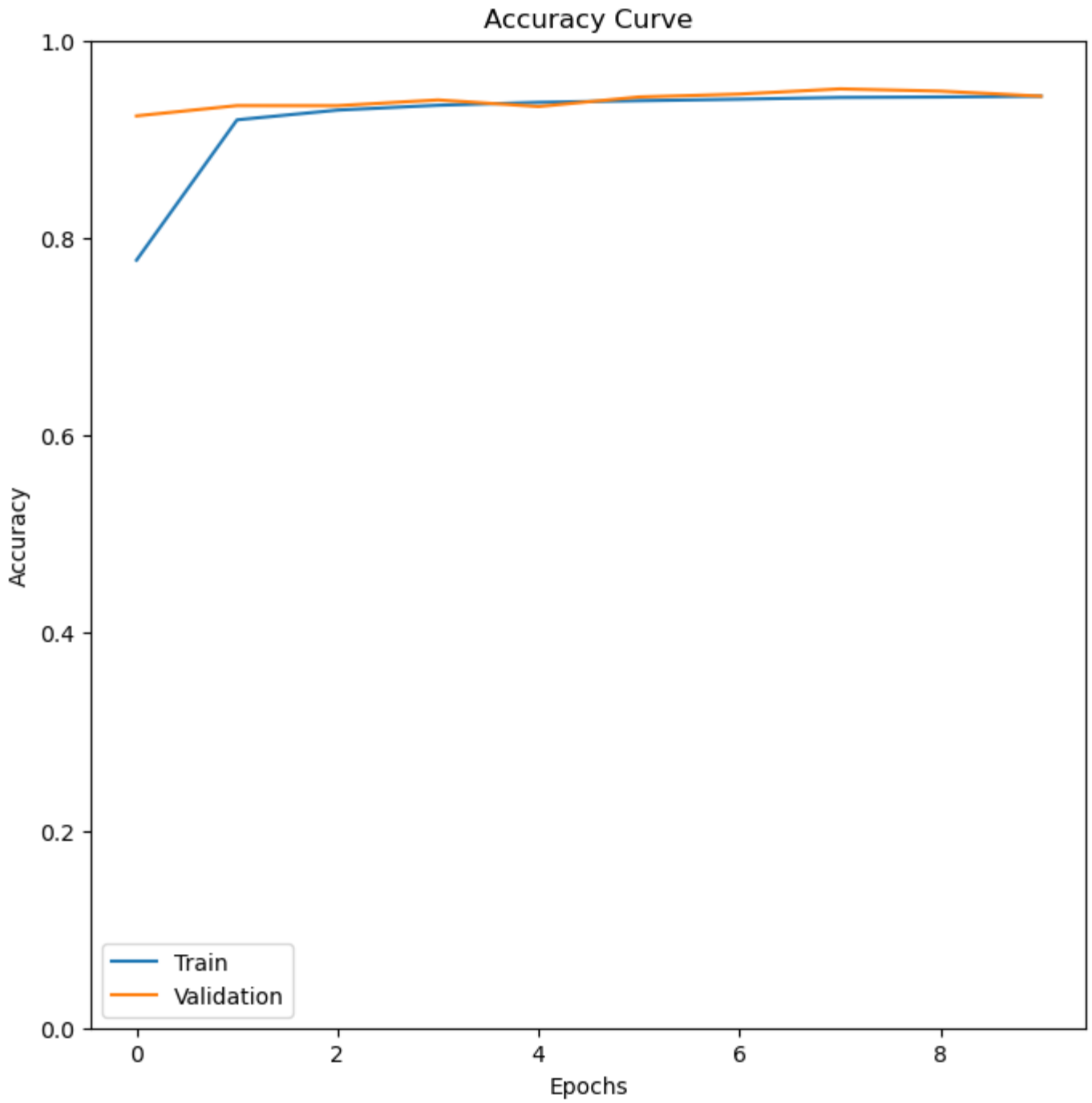
Learning Rates

- Tune the Two Layer Neural Network with various learning rates (while keeping all other hyperparameters constant) by changing the config file.
  - lr = 1
  - lr = 1e-1
  - lr = 1e-2
  - lr = 5e-2

In [ ]:
```
# Cell 10
# Change lr to 1 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

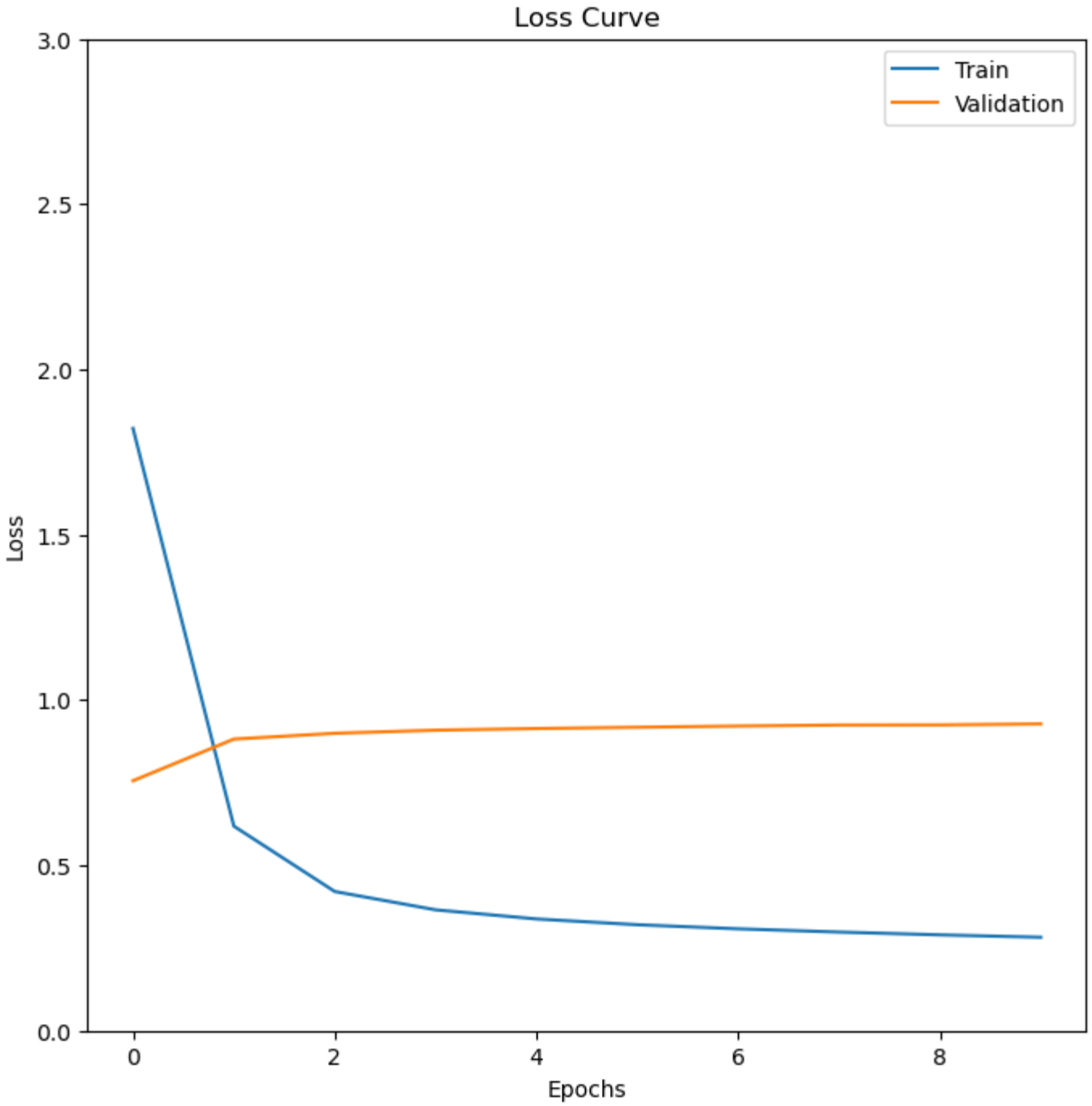
In [ ]:
```
# Cell 11
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

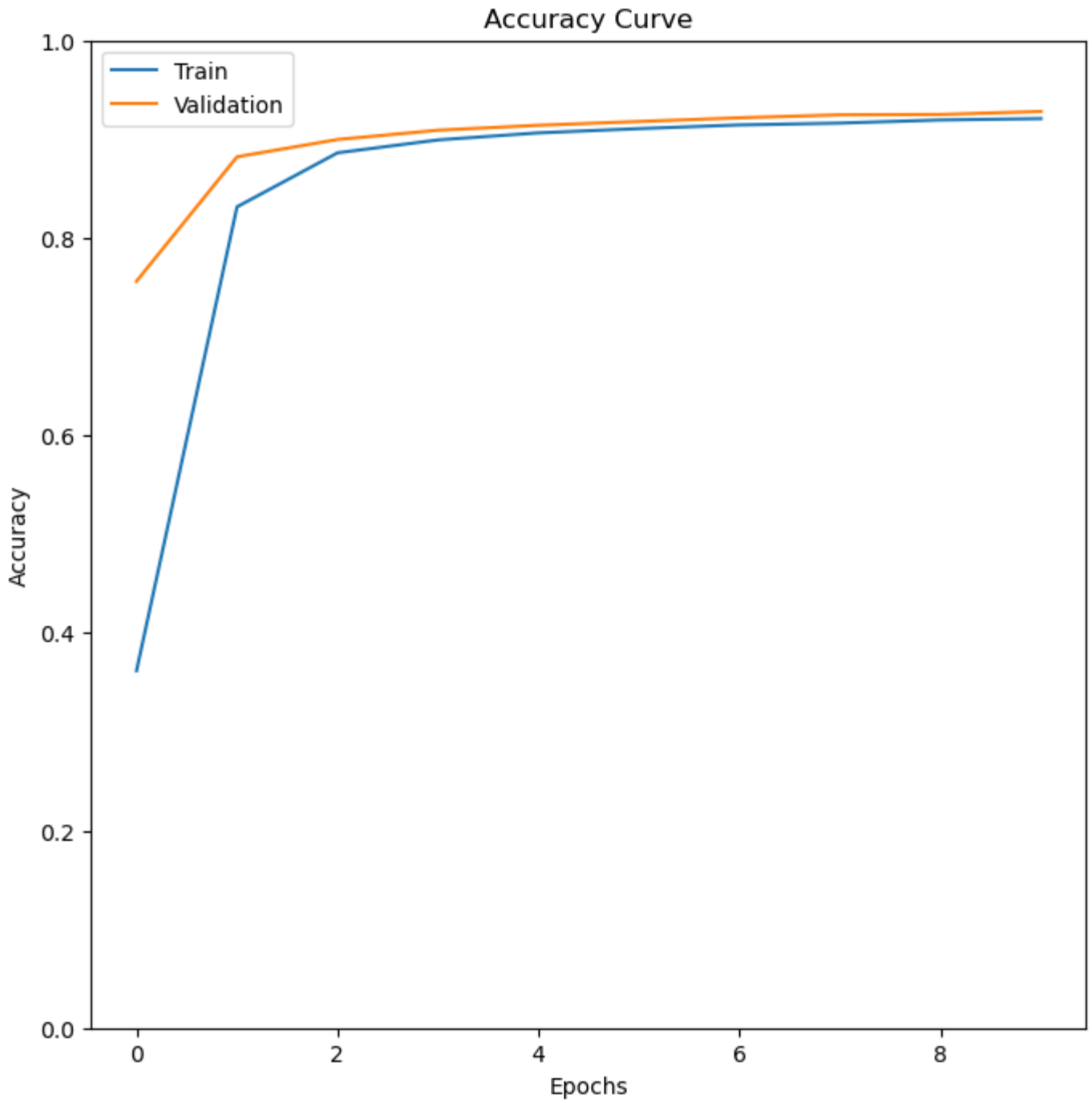

Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 12
# Change lr to 1e-1 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

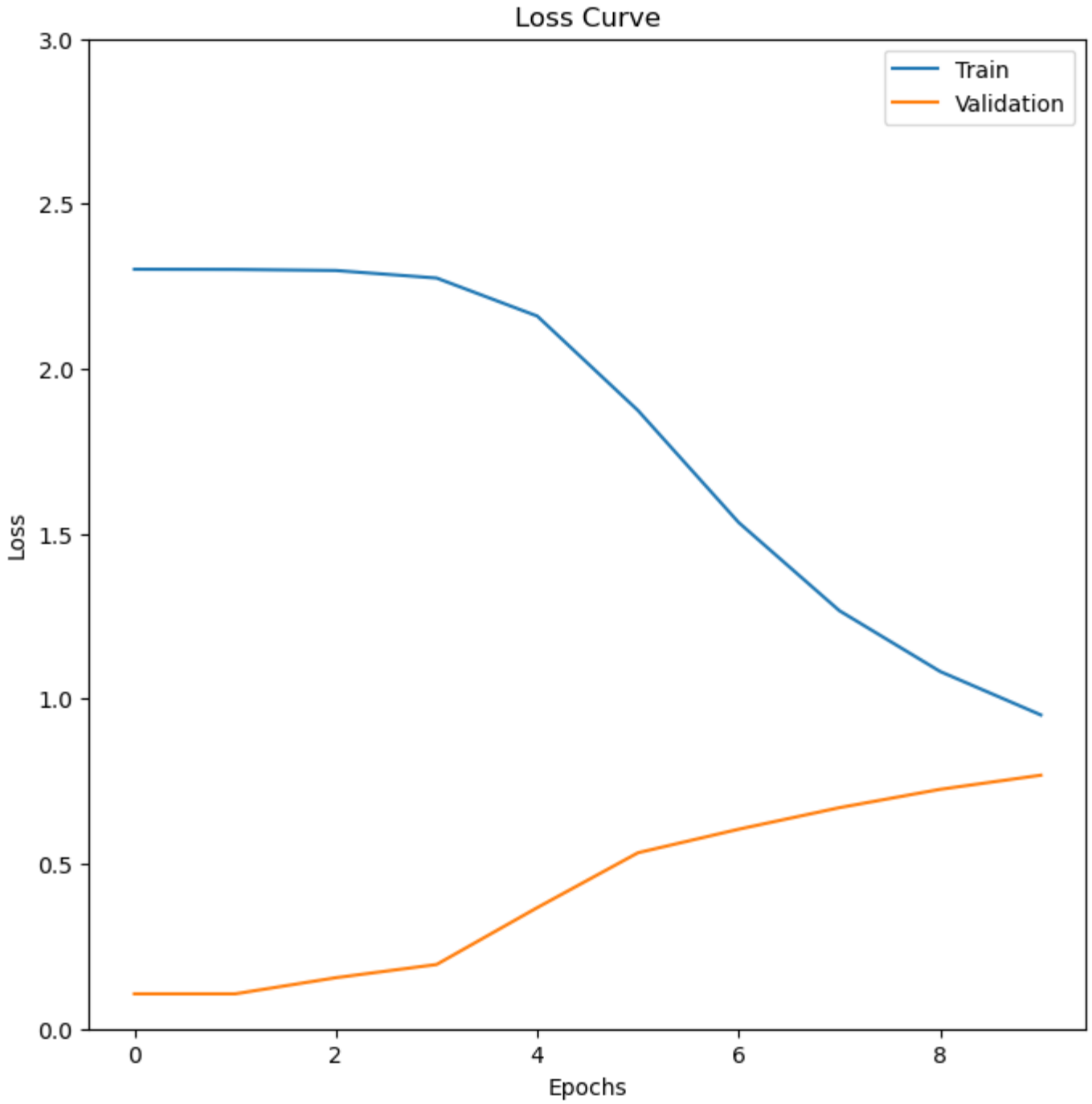
In [ ]:
```
# Cell 13
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

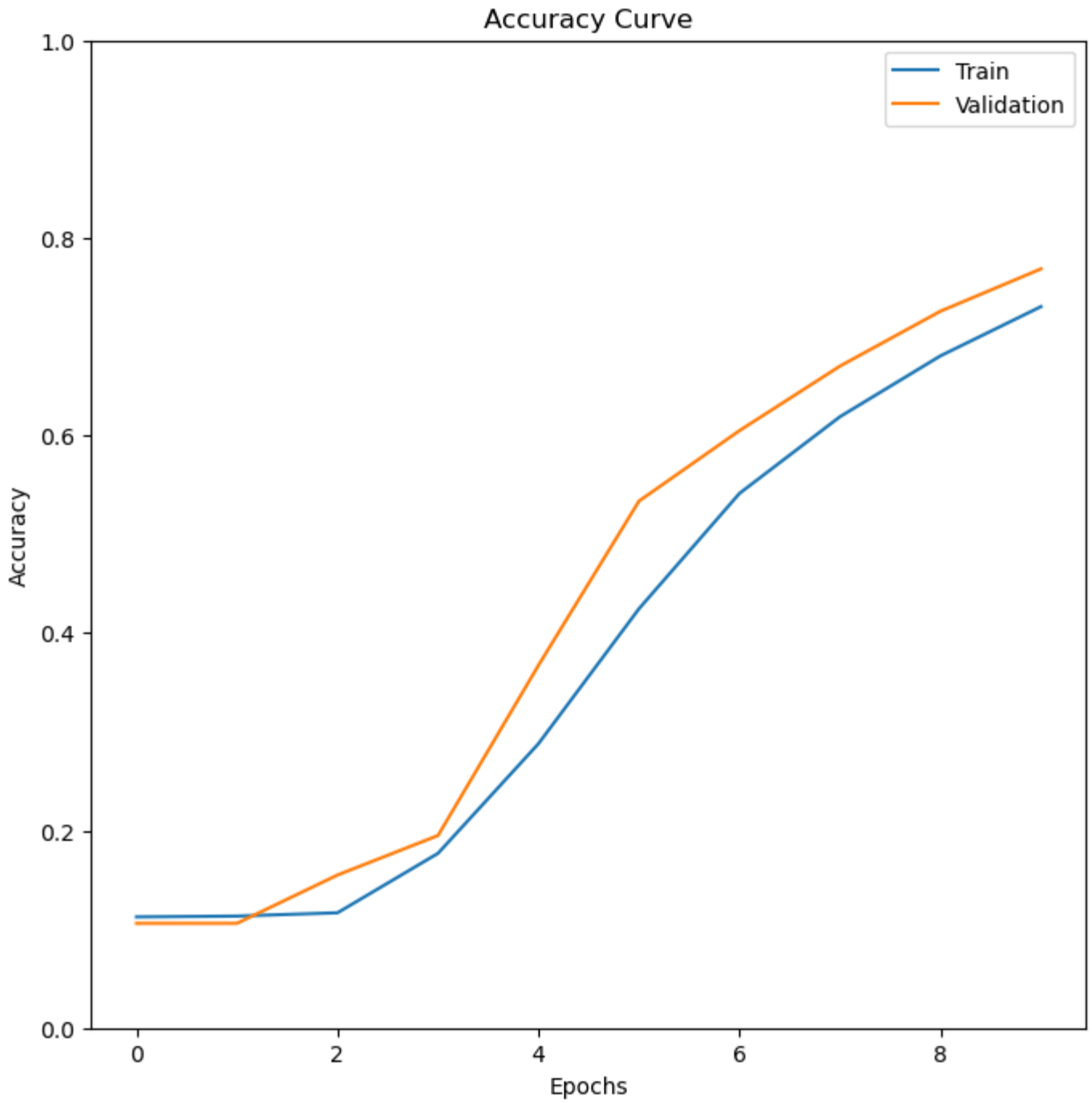

Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 14
# Change lr to 1e-2 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

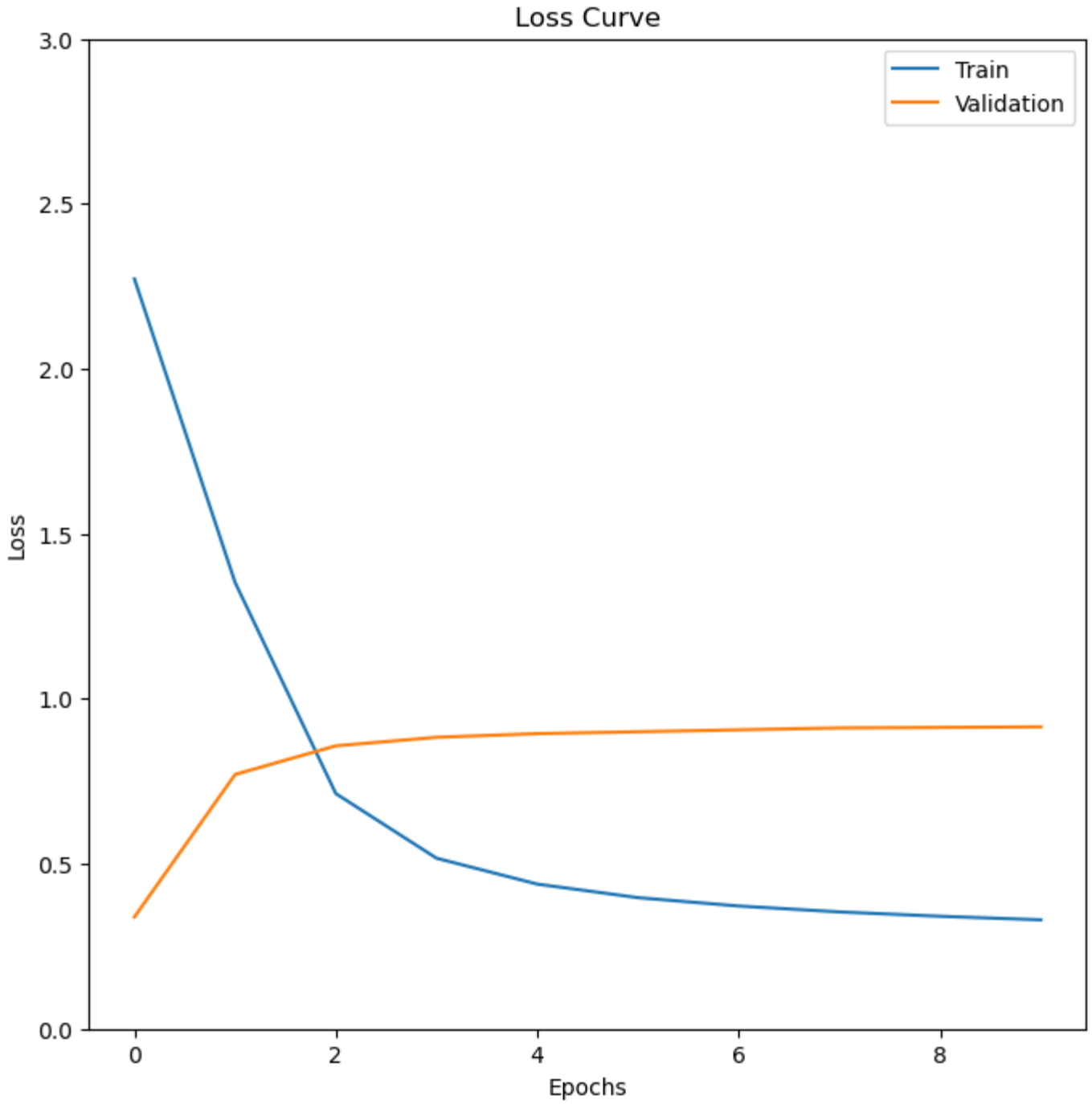
In [ ]:
```
# Cell 15
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

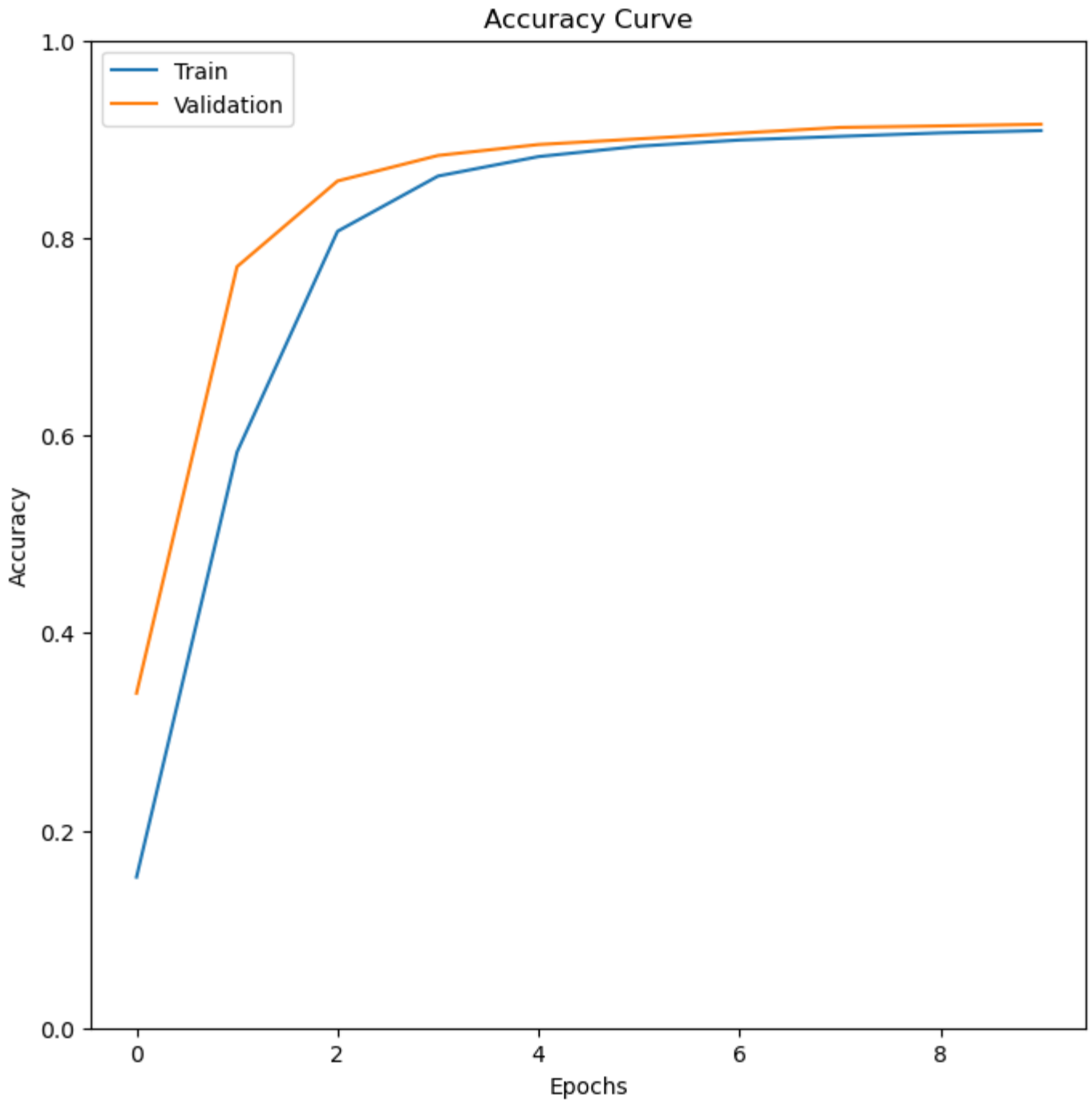

Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 16
# Change lr to 5e-2 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

In [ ]:
```
# Cell 17
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```


Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



Describe and explain your findings here:

Experimentally (as shown in the plots above), we observe that as the learning rate (η) decreses, the training and validation accuracies, and training loss slowly decrease, while the validation loss keeps at about the same level -- but just up to a certain point: if the learning rate is too low, then the MLP is not able to learn properly in the given amount of epochs.

That is because the loss function ($J(\theta)$, where θ are the weights of the model) is well-behaved and the weights updates given by the gradient descent algorithm, i.e., $\theta^{t+1} = \theta^t - \eta \nabla_\theta J(\theta)$, will be minimal once the learning rate is low enough, either requiring more epochs, or being completely insufficient.

Regularization

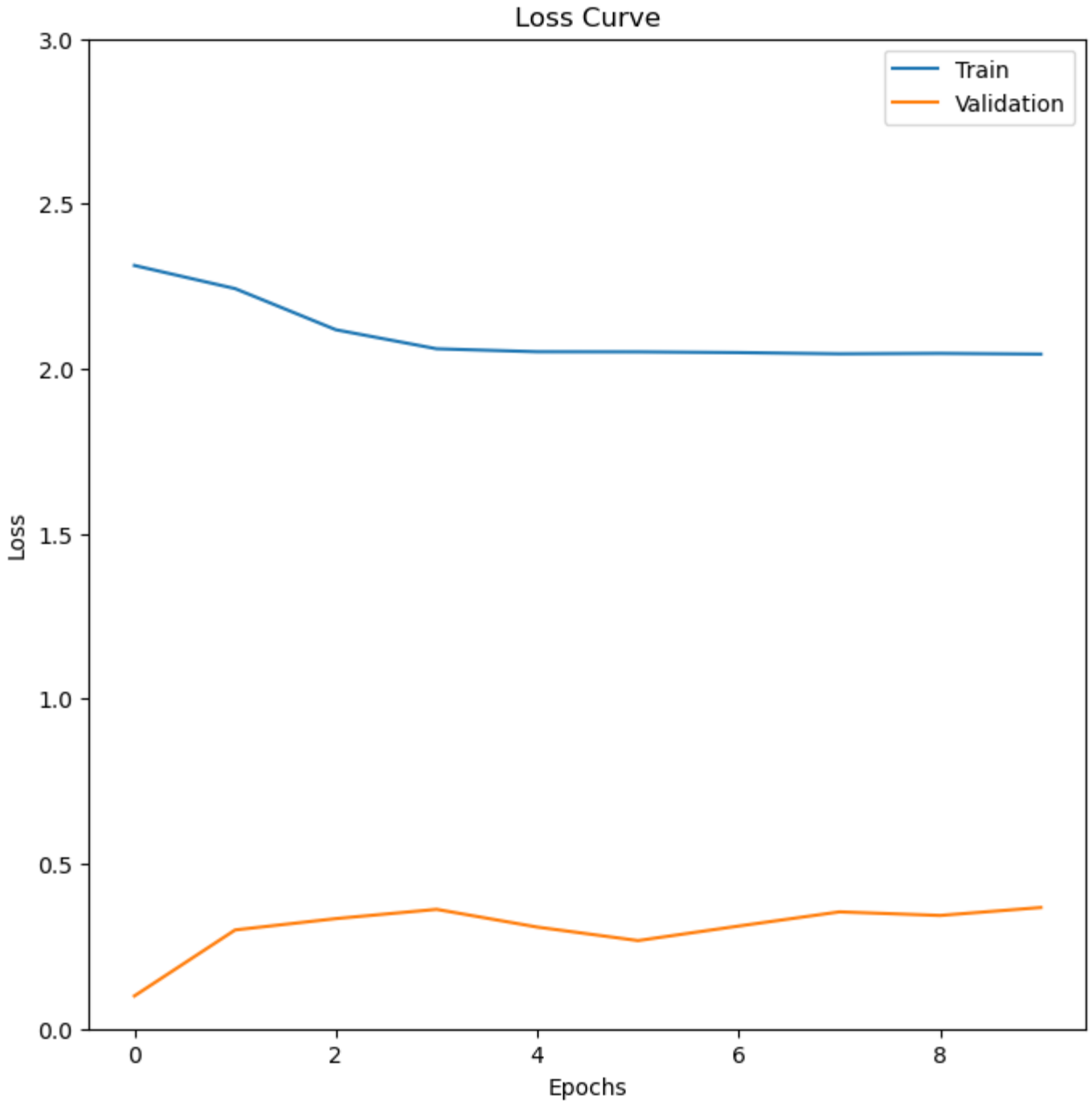
- Tune the Two Layer Neural Network with various regularization coefficients (while keeping all other hyperparameters constant) by changing the config file.
    - reg = 1e-1
    - reg = 1e-2
    - reg = 1e-3
    - reg = 1e-4
    - reg = 1

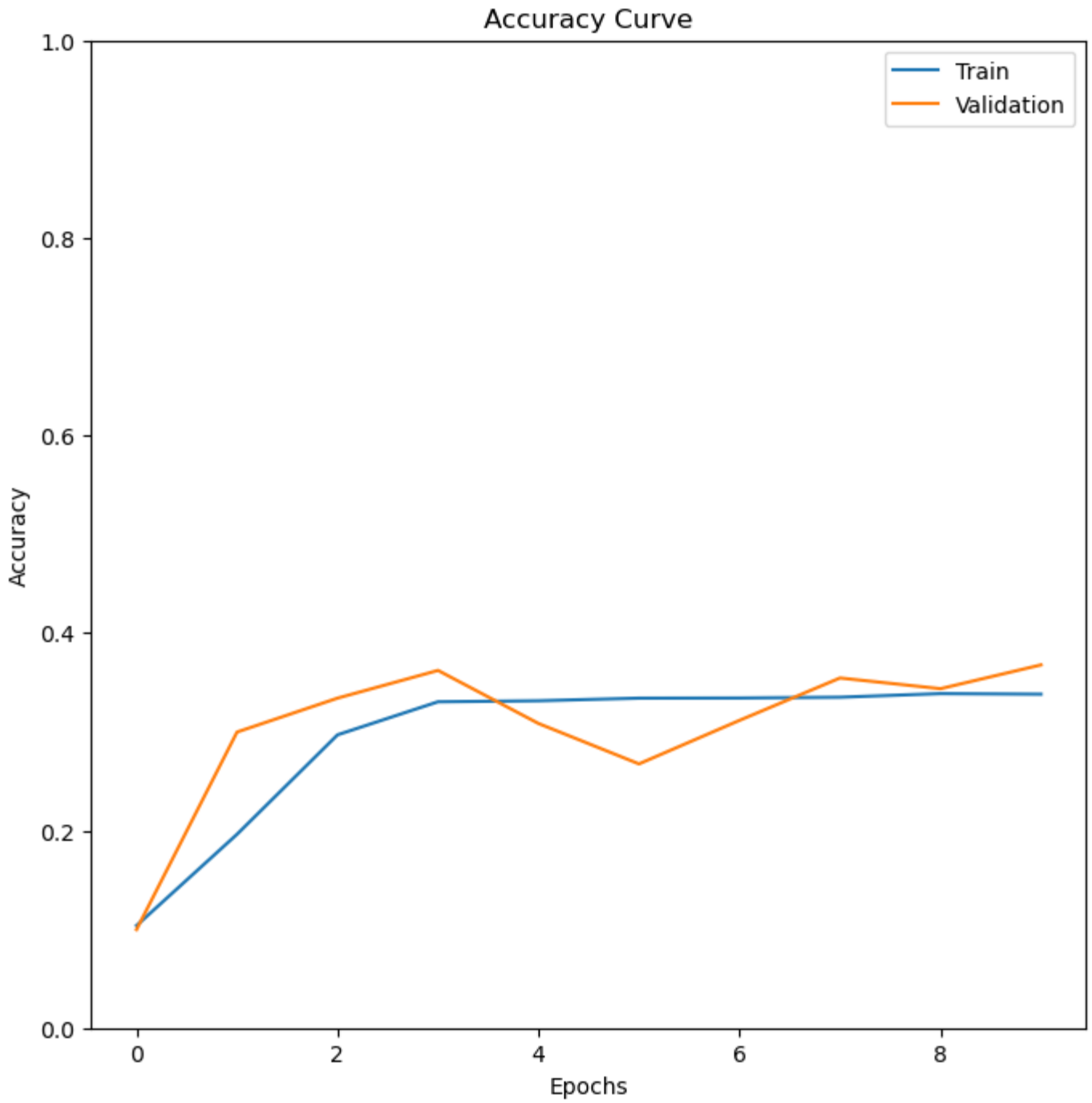
Arthur's NOTE: keeping lr = 0.1 (original)

Original reg = 0.001

In [ ]:
```
# Cell 18
# Change reg to 1e-1 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

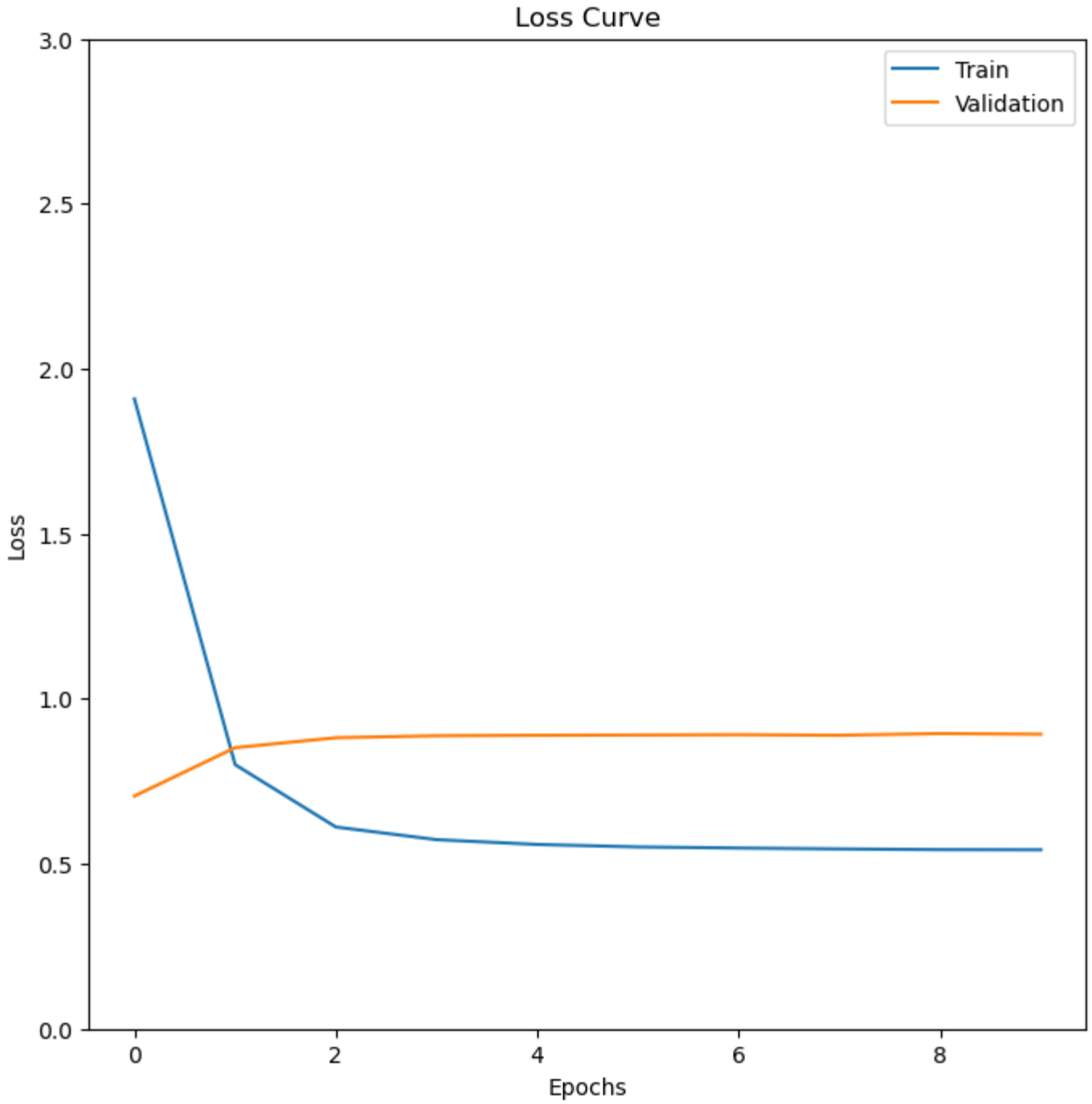
In [ ]:
```
# Cell 19
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

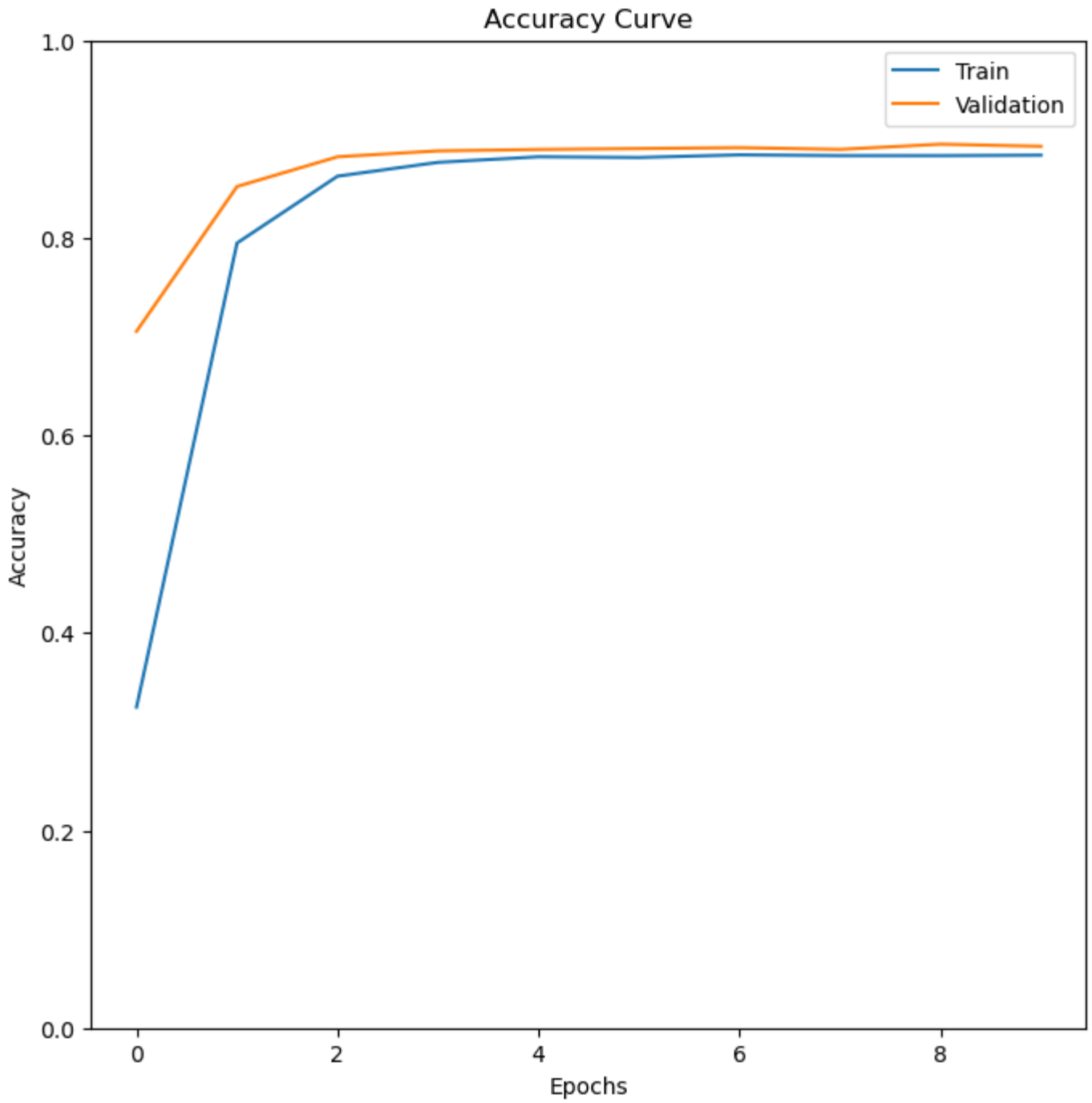

Loss Curve
Train
Validation
Loss
Epochs
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8



In [ ]:
```
# Cell 20
# Change reg to 1e-2 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

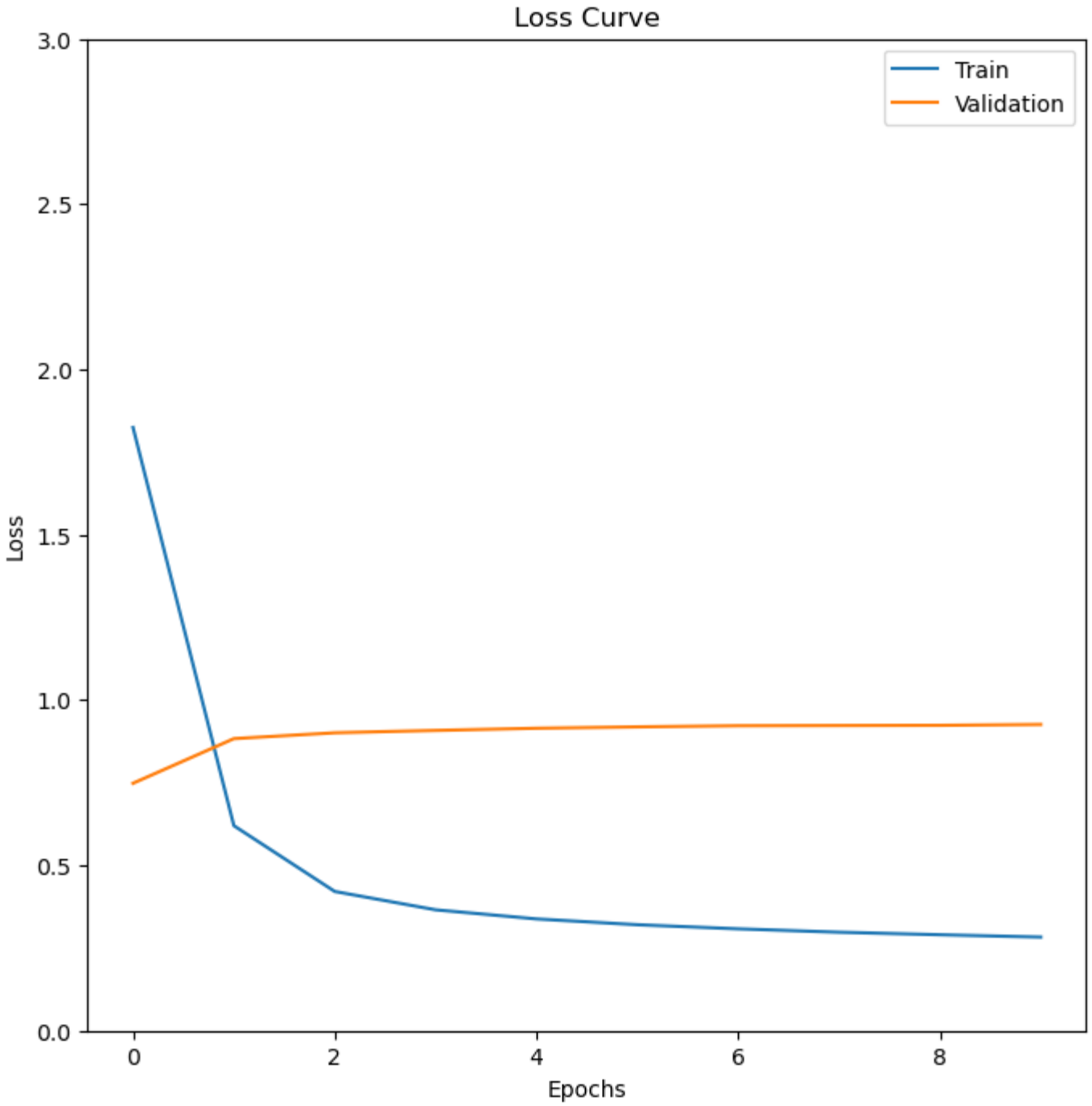
In [ ]:
```
# Cell 21
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

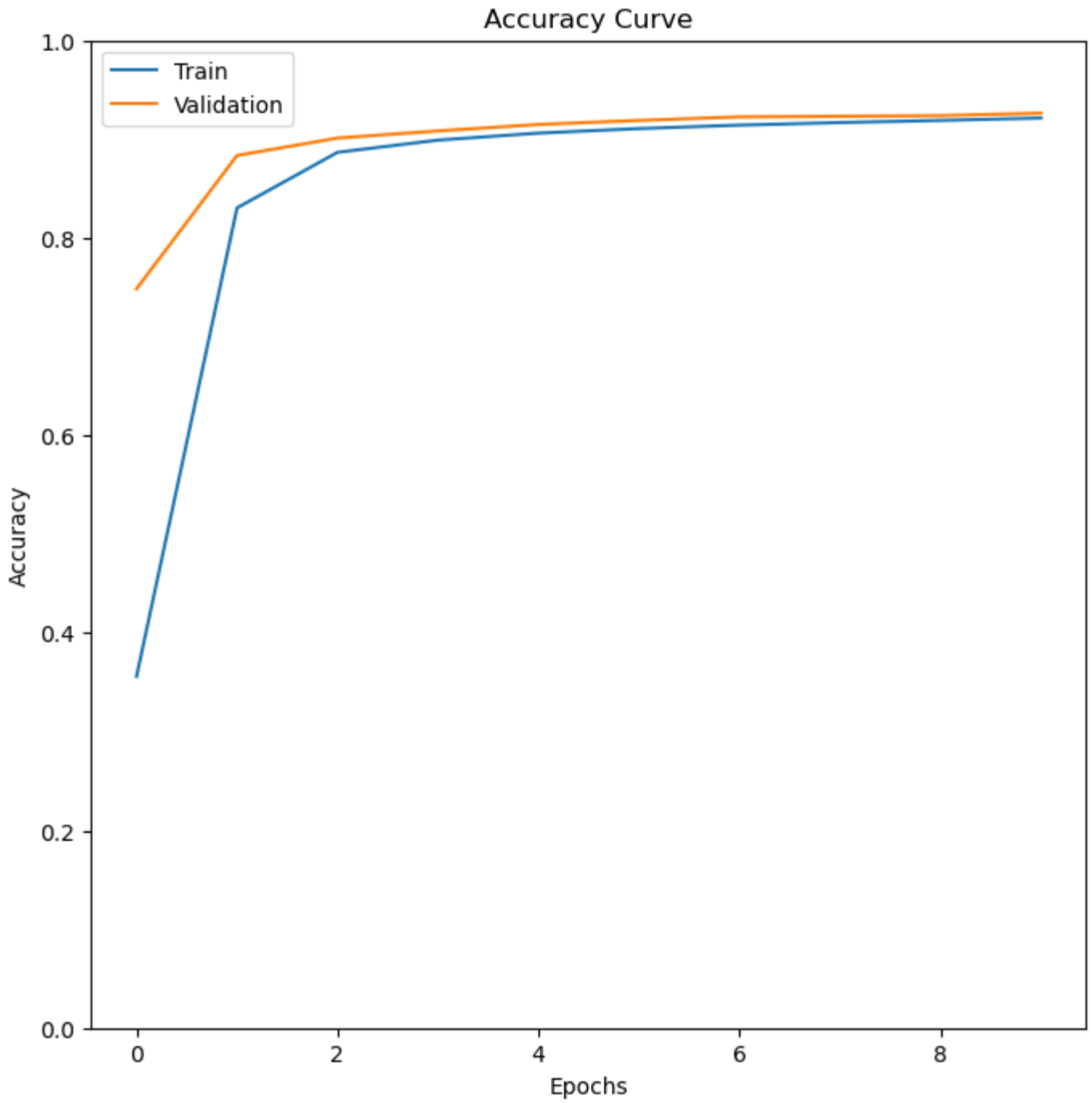

Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 22
# Change reg to 1e-3 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

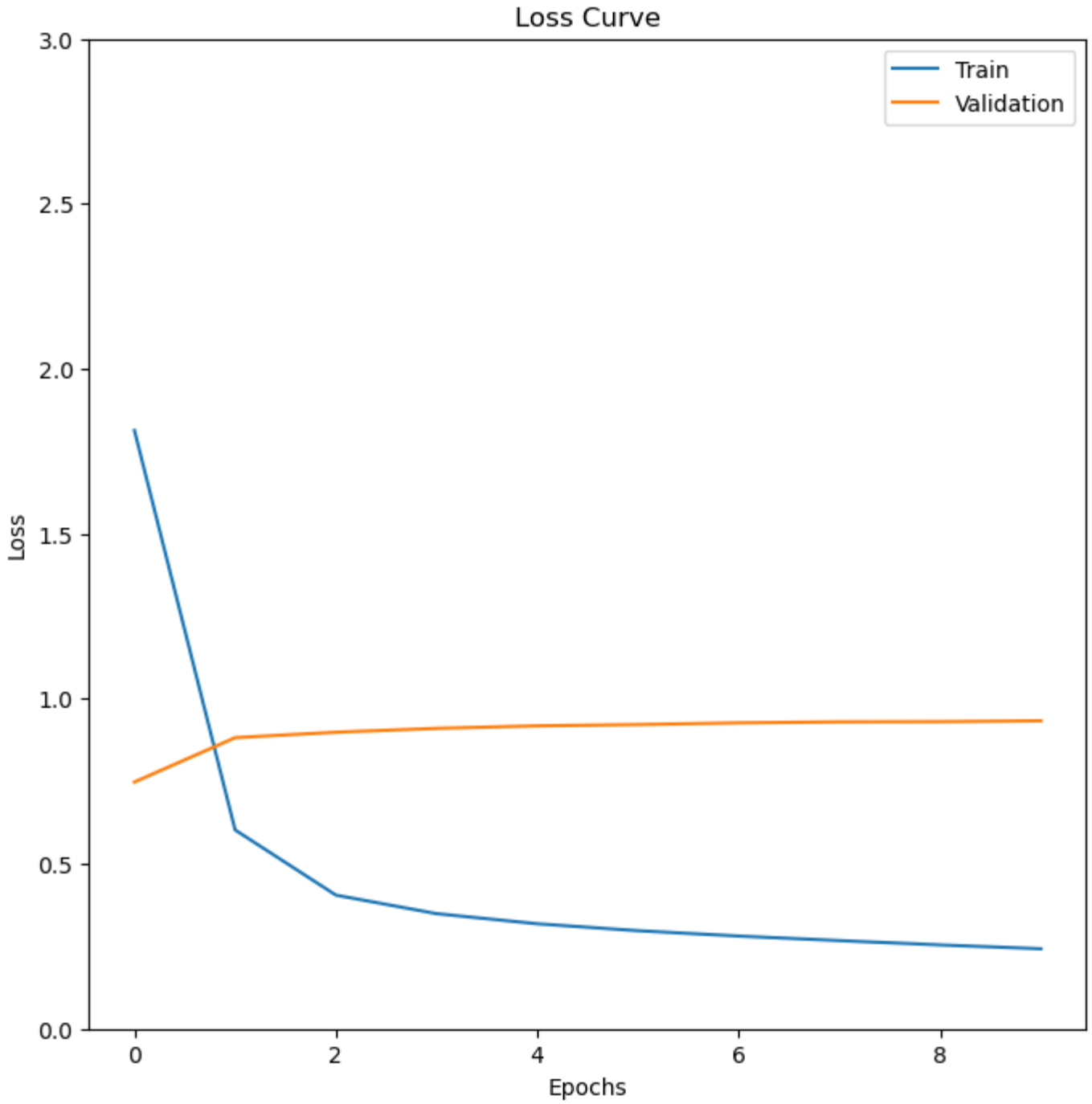
In [ ]:
```
# Cell 23
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

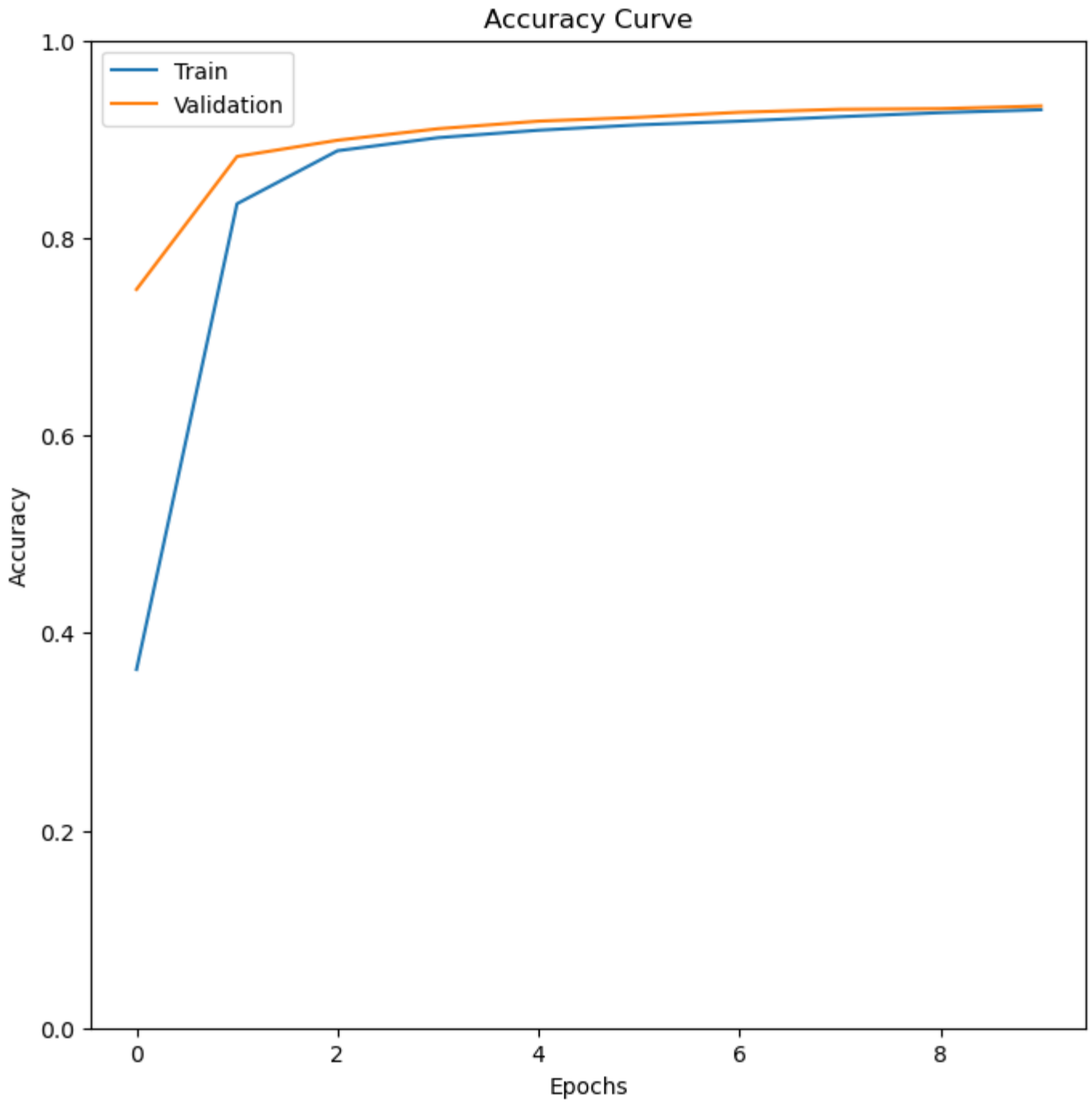

Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 24
# Change reg to 1e-4 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

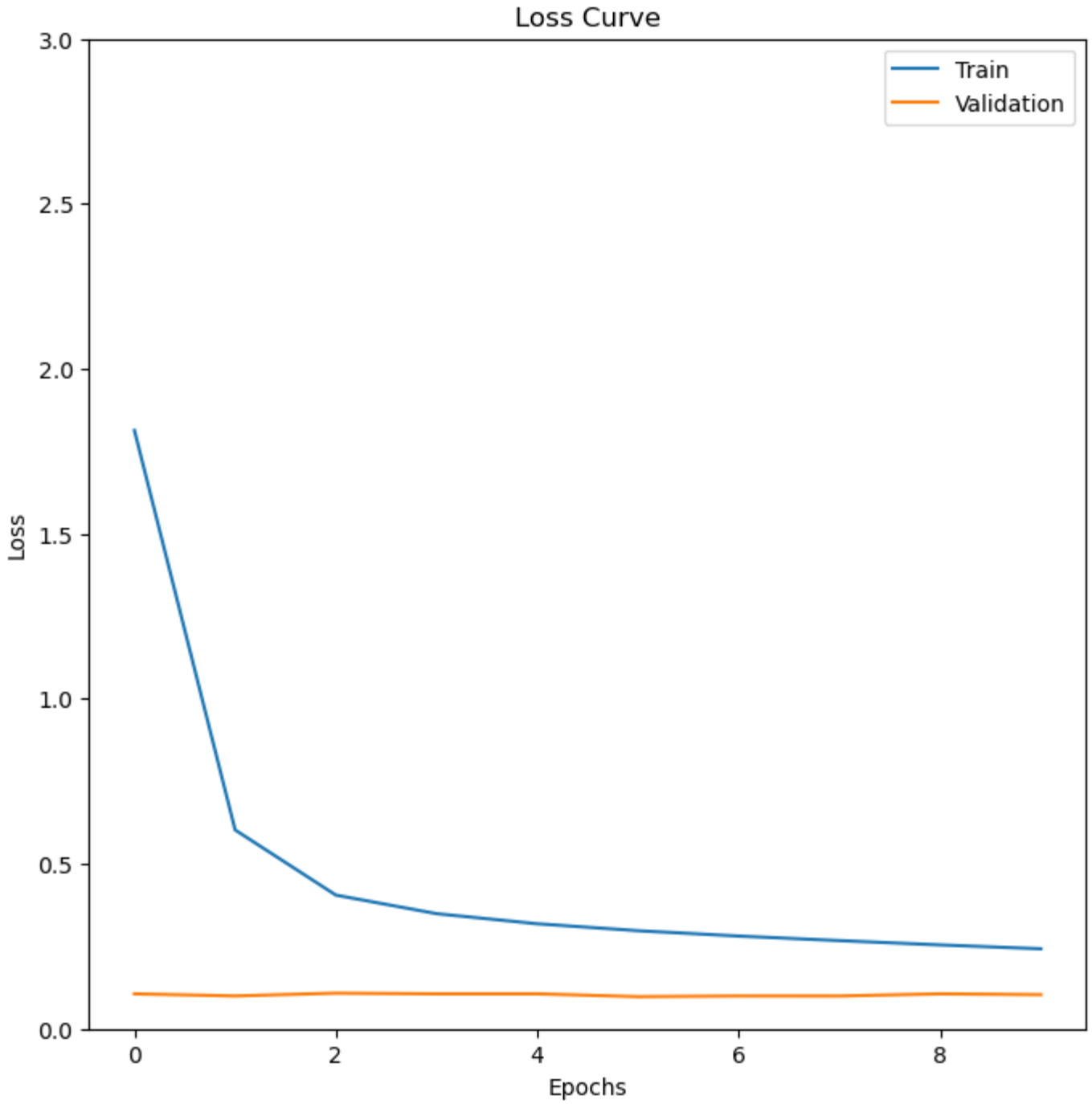
In [ ]:
```
# Cell 25
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```

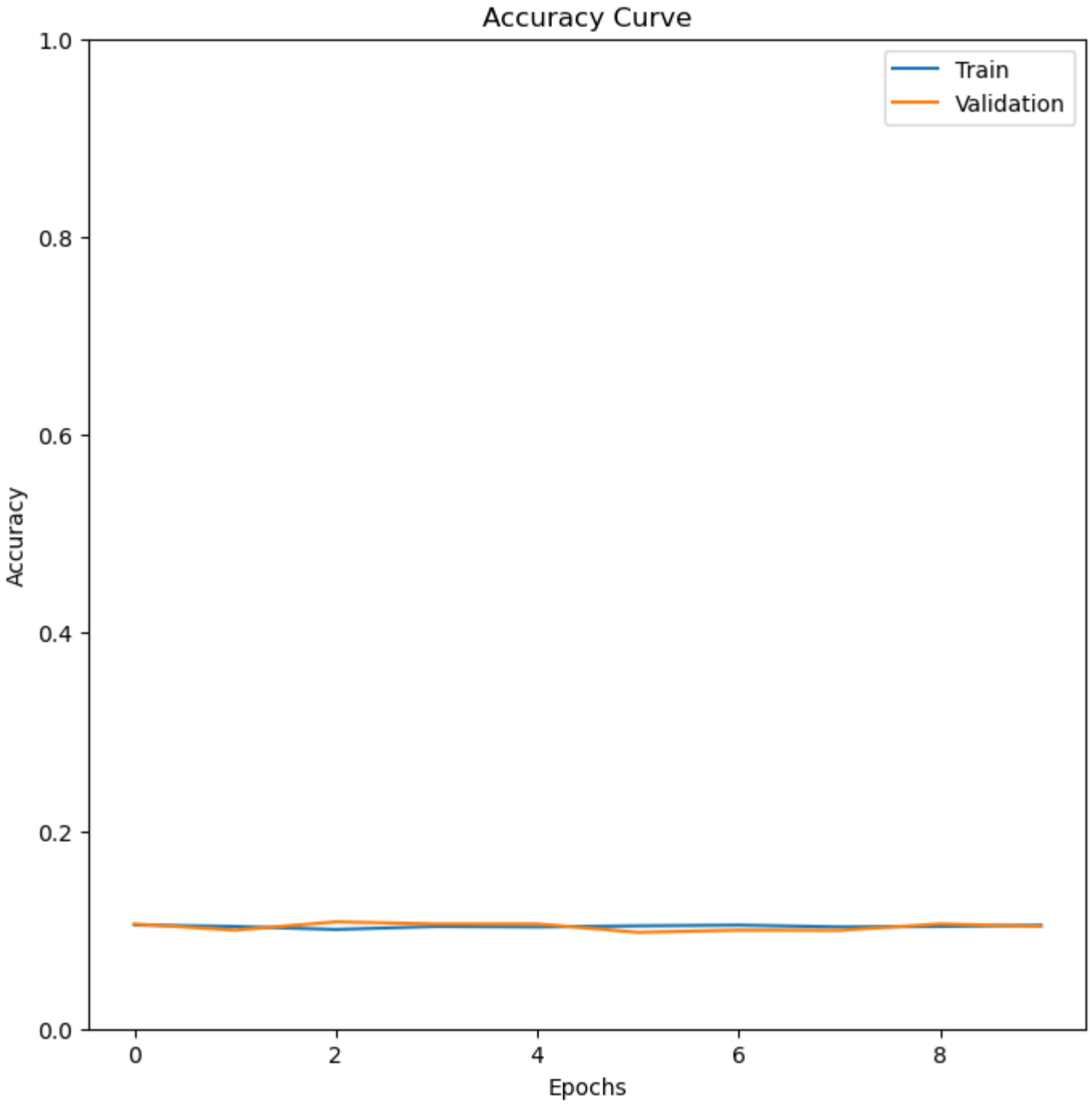

Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 26
# Change reg to 1 in the config file and run this code block
train_loss_hi64story, train_acc_history, valid_loss_history, valid_acc_history = train_model("co
```

In [ ]:
```
# Cell 27
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```


Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs


Accuracy Curve
Train
Validation
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
0
2
4
6
8
Epochs

Describe and explain your findings here:

With the plots above, we empirically observe that the higher the regularization coefficient (λ) is, the worse the model performs. That is up to a certain point, i.e. there isn't much difference between $\lambda = 0.001$ and $\lambda = 0.0001$.

This is explained because the regularization term is supposed to give the model some "slack" or "room to play" during the optimization phase. It essentially means that it can sacrifice increasing a little bit the settling loss to in return be more generalizable. This is evident when looking at the regularization formula :

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

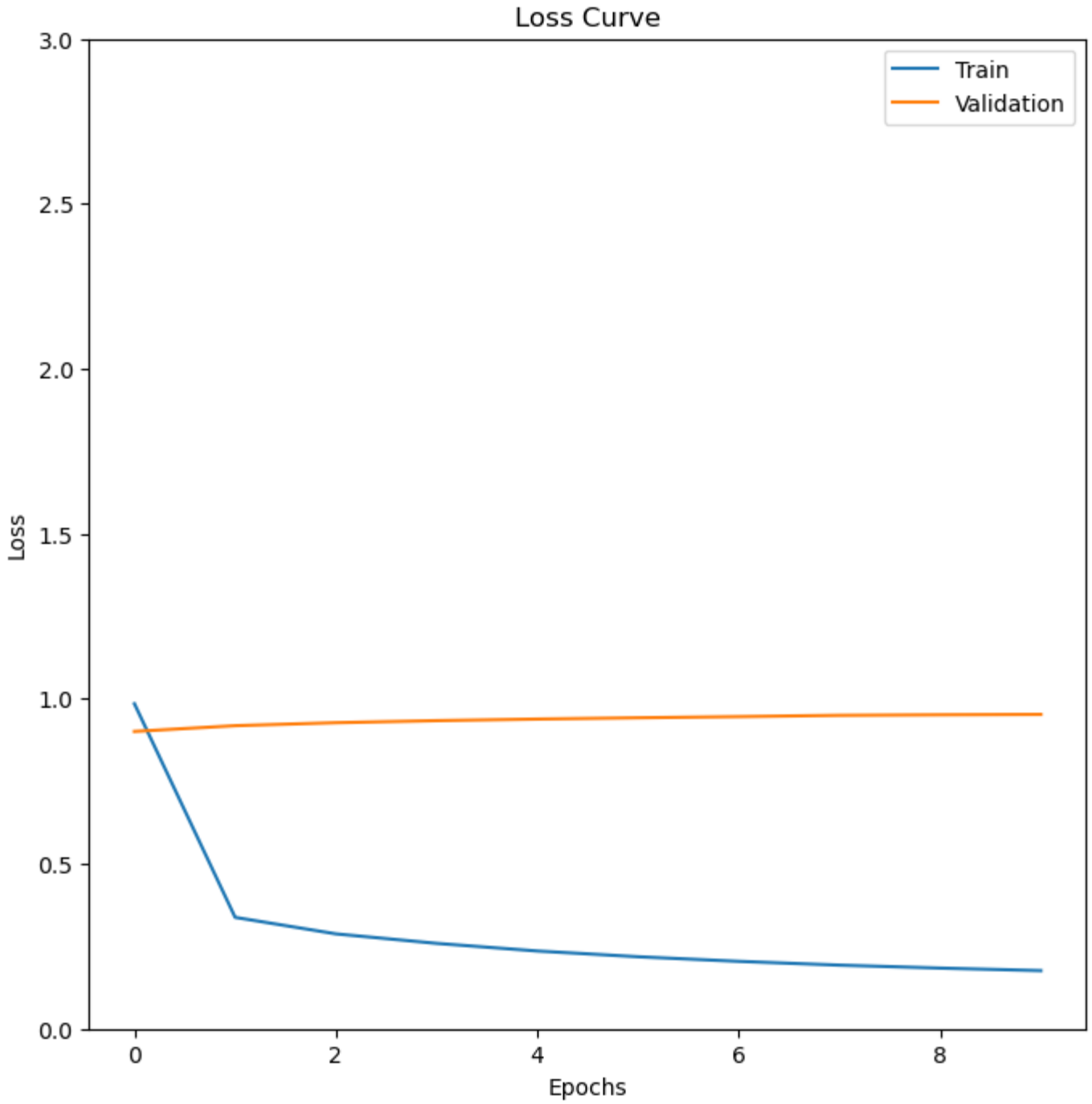
We can see that if λ is not low enough, we are summing numbers that might be too high, therefore compromizing the fidelity of the value of the loss to its actual meaning.

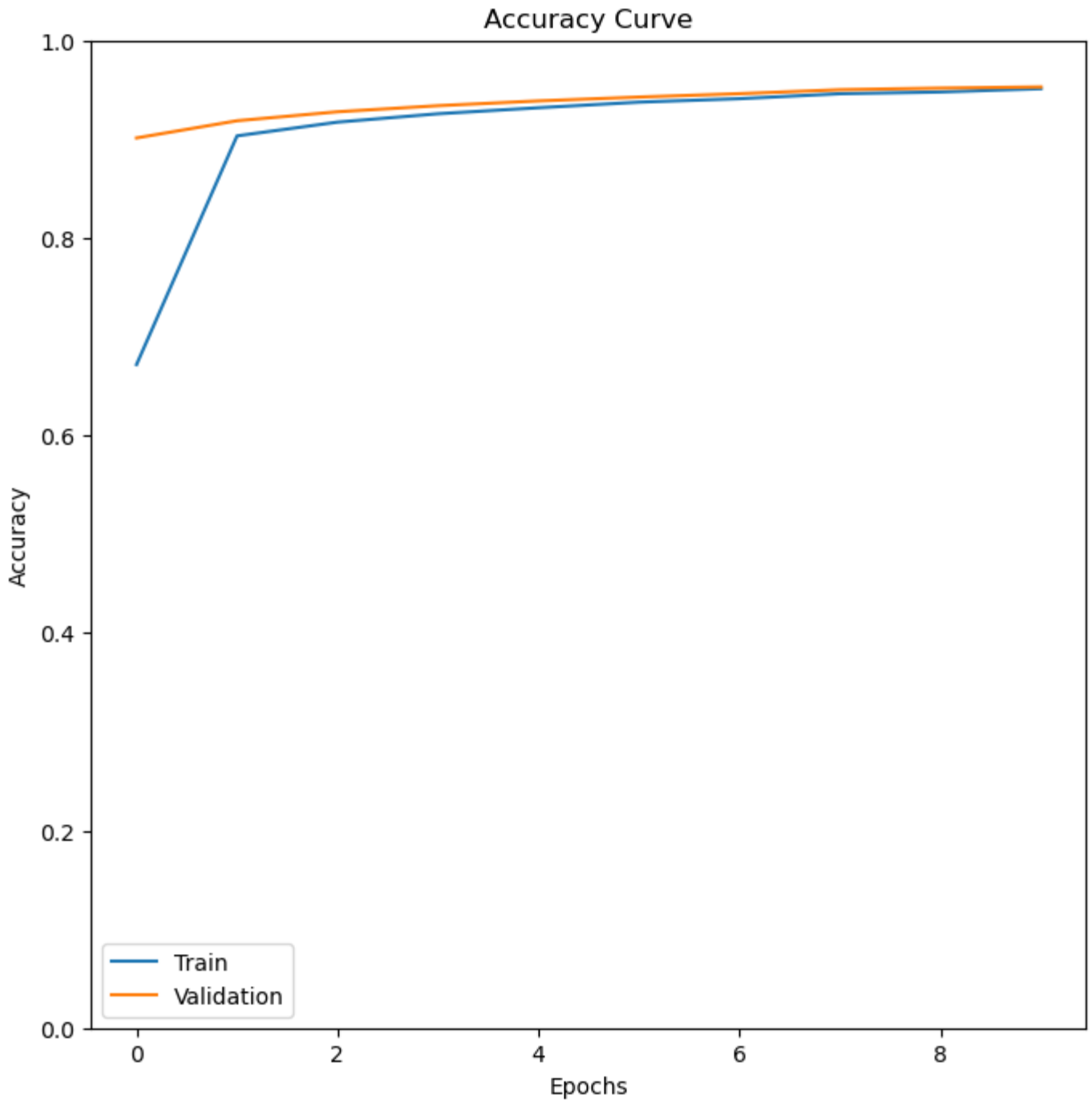
Hyper-parameter Tuning

You are now free to tune any hyperparameters for better accuracy. In this block type the configuration of your best model and provide a brief explanation of why it works.

In [ ]:
```
# Cell 28
# hyperparameter tuning
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history = train_model("conf
```

In [ ]:
```
# Cell 29
plot_curves(train_loss_history, train_acc_history, valid_loss_history, valid_acc_history)
```


Loss Curve
Train
Validation
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Loss
0
2
4
6
8
Epochs



In [ ]:
```
# Cell 30
#To collect submission
!sh collect_submission.sh
```

Describe and explain your findings here:

Best model: $\eta = 0.3$; $\lambda = 0.0005$.

Important to note: I only played with those two variables since they were the only two that the notebook had us play with. On that note, I think that given enough epochs, many different combinations of η and λ would end up performing just as good; batch size would not affect the results that much (if kept between 16 and 64), but would increase the computation time for samller batches; momentum should not be a deciding factor, but is in a good level; and changing the hidden size would make the comparisons unfair given that we would essitially be changing the architecture (network's width).

As seen in previous experiments, the regularization coefficient λ that worked better was somewhere between 10^{-3} and 10^{-4}, so that was the range which was more promissing. Simmilarly, we observed that lower learning rates η for the same number of epochs were not as well-performing, so I chose to play in the range of 0.1 to 1, while avoiding taking steps too large.