main

January 29, 2025

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[36]:
```
#Cell 2
import os
# TODO: Enter the relative path in your Google Drive to the unzipped folder for␣
 ↪hw1_code_submission.zip
FOLDERNAME = 'hw1' # e.g. 'cs7643/hw1/Code'

assert FOLDERNAME is not None, "[!] Enter the foldername."
working_directory = os.path.join("/content/drive/MyDrive/", FOLDERNAME)
assert os.path.exists(working_directory), "Make sure your FOLDERNAME is correct"
%cd $working_directory/data
!sh get_data.sh
%cd ..
```

```
/content/drive/MyDrive/hw1/data
--2025-01-29 08:33:31--  https://pjreddie.com/media/files/mnist_train.csv
Resolving pjreddie.com (pjreddie.com)… 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 109575994 (104M) [text/csv]
Saving to: ‘mnist_train.csv.3’

mnist_train.csv.3   100%[===================>] 104.50M  31.1MB/s    in 3.9s

2025-01-29 08:33:35 (27.0 MB/s) - ‘mnist_train.csv.3’ saved
[109575994/109575994]

--2025-01-29 08:33:35--  https://pjreddie.com/media/files/mnist_test.csv
Resolving pjreddie.com (pjreddie.com)… 162.0.215.52
Connecting to pjreddie.com (pjreddie.com)|162.0.215.52|:443… connected.
HTTP request sent, awaiting response… 200 OK
Length: 18289443 (17M) [text/csv]
```

```
Saving to: ‘mnist_test.csv.3’

mnist_test.csv.3    100%[===================>]  17.44M  12.7MB/s    in 1.4s

2025-01-29 08:33:37 (12.7 MB/s) - ‘mnist_test.csv.3’ saved [18289443/18289443]

/content/drive/MyDrive/hw1
```

[3]:
```
# 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_training
```

```
Loading training data…
Training data loaded with 60000 images
Loading testing data…
Testing data loaded with 10000 images
…Loading training data…
Training data loaded with 60000 images
Loading testing data…
Testing data loaded with 10000 images
.
----------------------------------------------------------------------
Ran 4 tests in 44.984s

OK
```

[10]:
```
#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, plot_curves
```

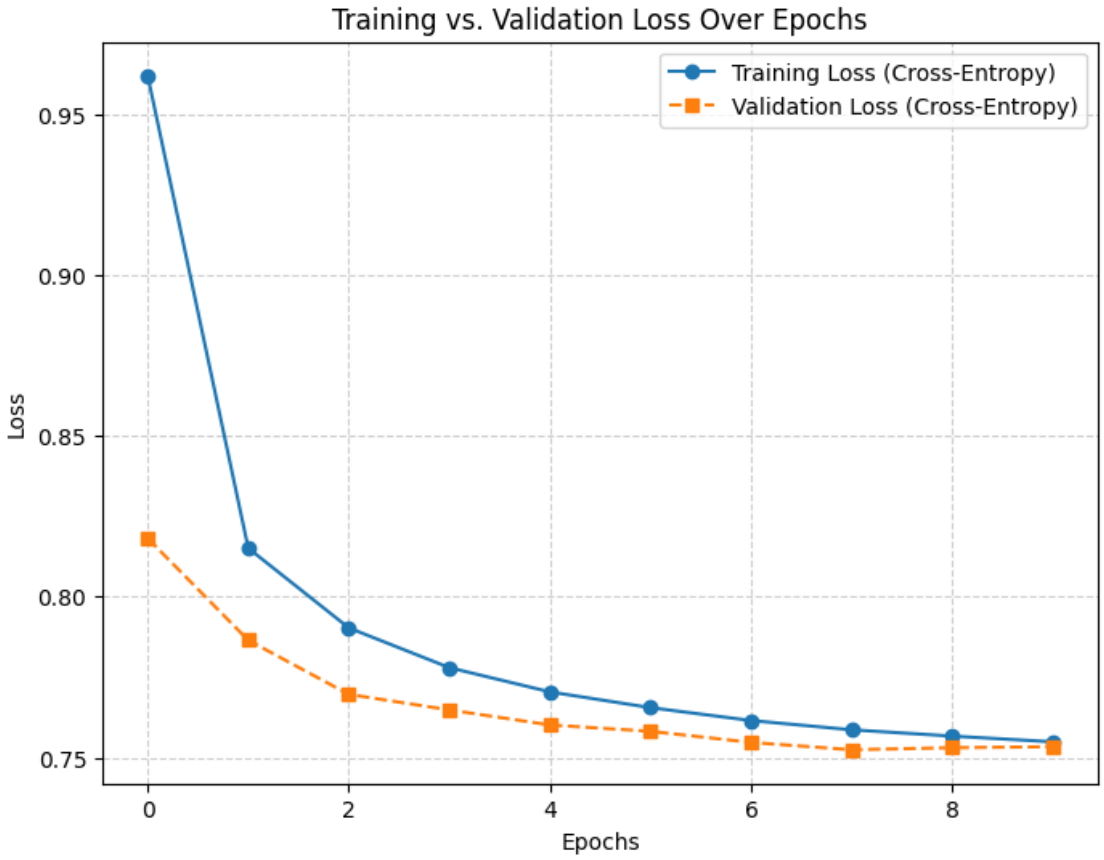
[5]:
```
# Cell 5
%matplotlib inline
def train_model(yaml_config_file):
  args = {}
  with open(yaml_config_file) as f:
      config = yaml.full_load(f)

  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,␣
↪batch_size=args["batch_size"], shuffle=True)
    epoch_loss, epoch_acc = train(epoch, batched_train_data,␣
↪batched_train_label, model, optimizer, args["debug"])

    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_size=args["batch_size"])
    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=args["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
```

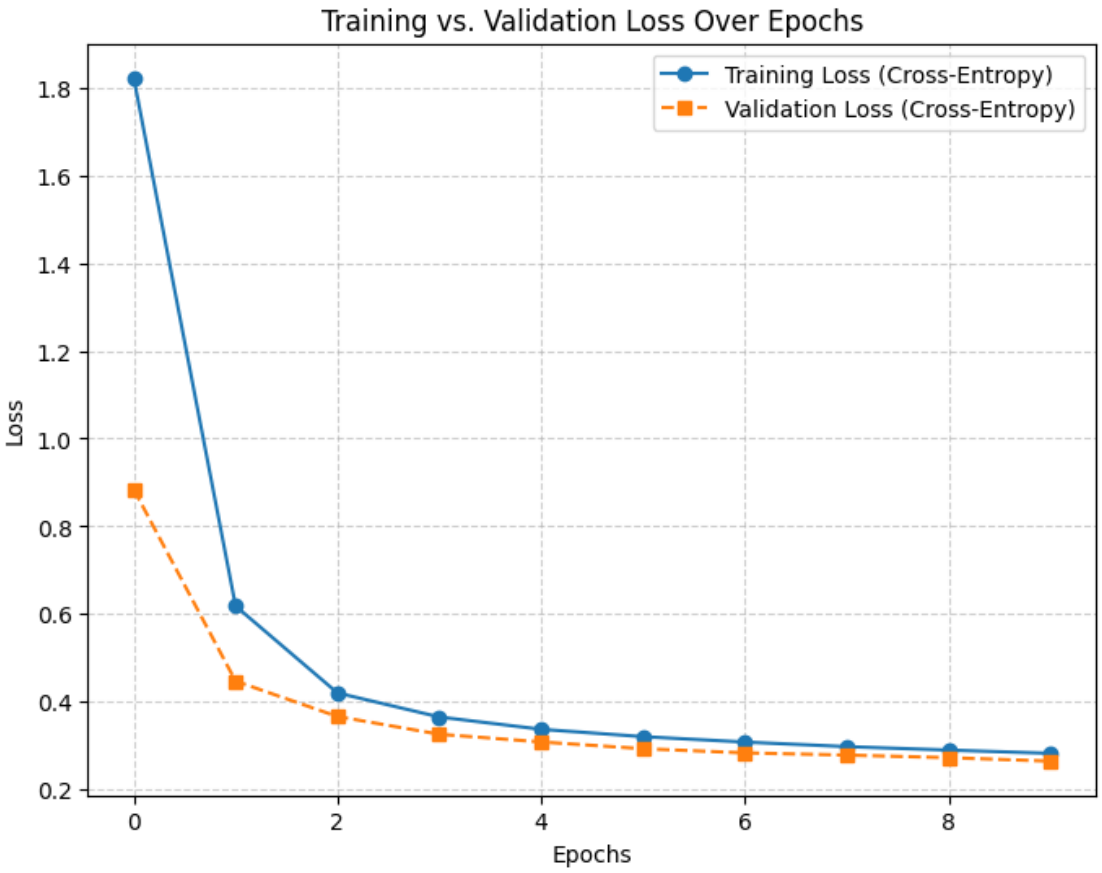
[ ]:
```
# Cell 6
# train softmax model
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history =␣
 ↪train_model("configs/config_softmax.yaml")
```

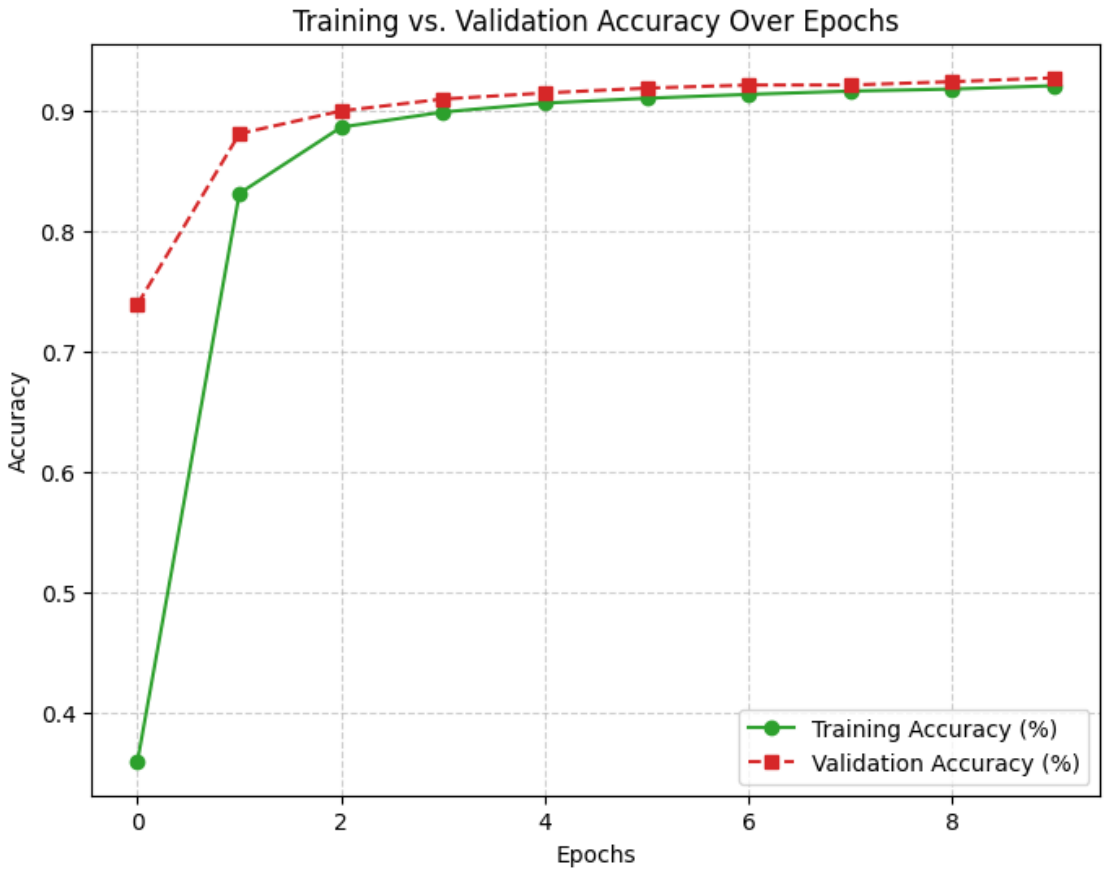
[7]:
```
# Cell 7
# plot results for softmax model
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
 ↪valid_acc_history)
```



[ ]:
```
# Cell 8
# train two layer neural network
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history =␣
↪train_model("configs/config_twolayer.yaml")
```

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


Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8



1 Assignment 1 Writeup

- Name: Adrian Kukla
- GT Email: akukla6@gatech.edu
- GT ID: 904056948

1.1 Two Layer Neural Network

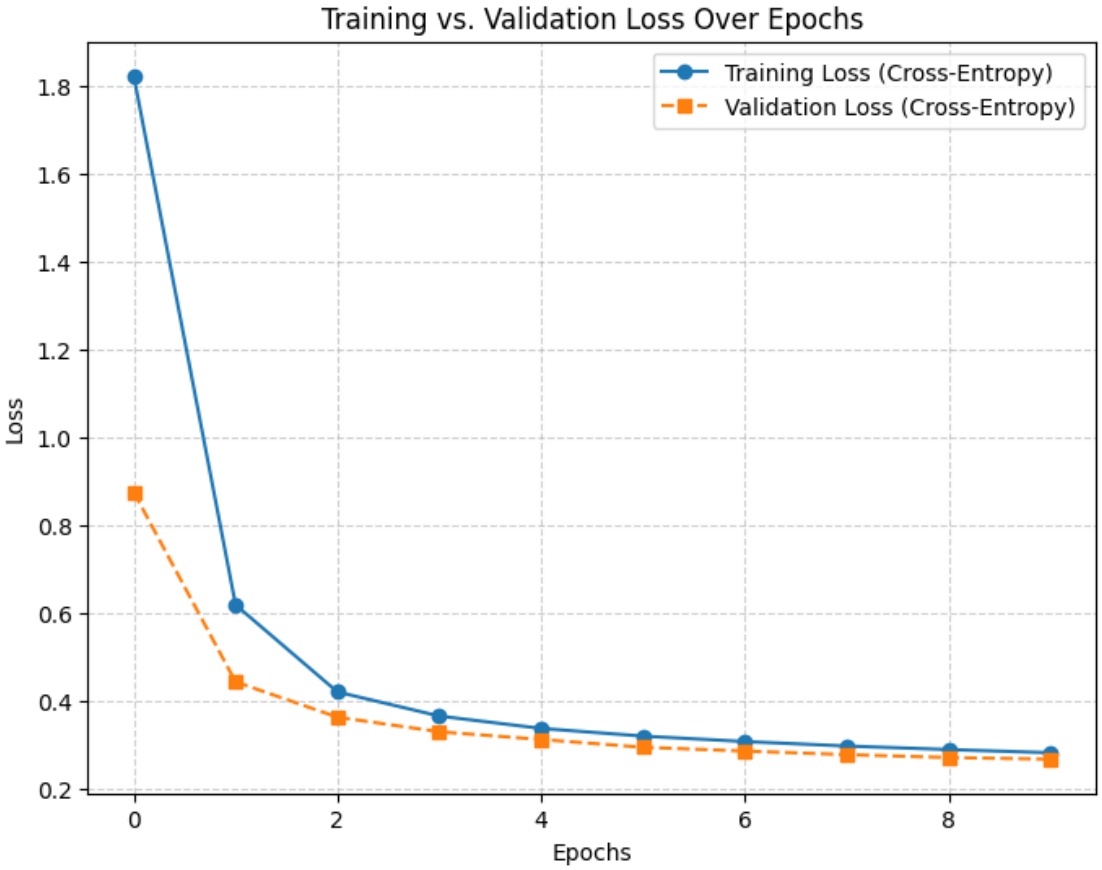
2 Learning Rates

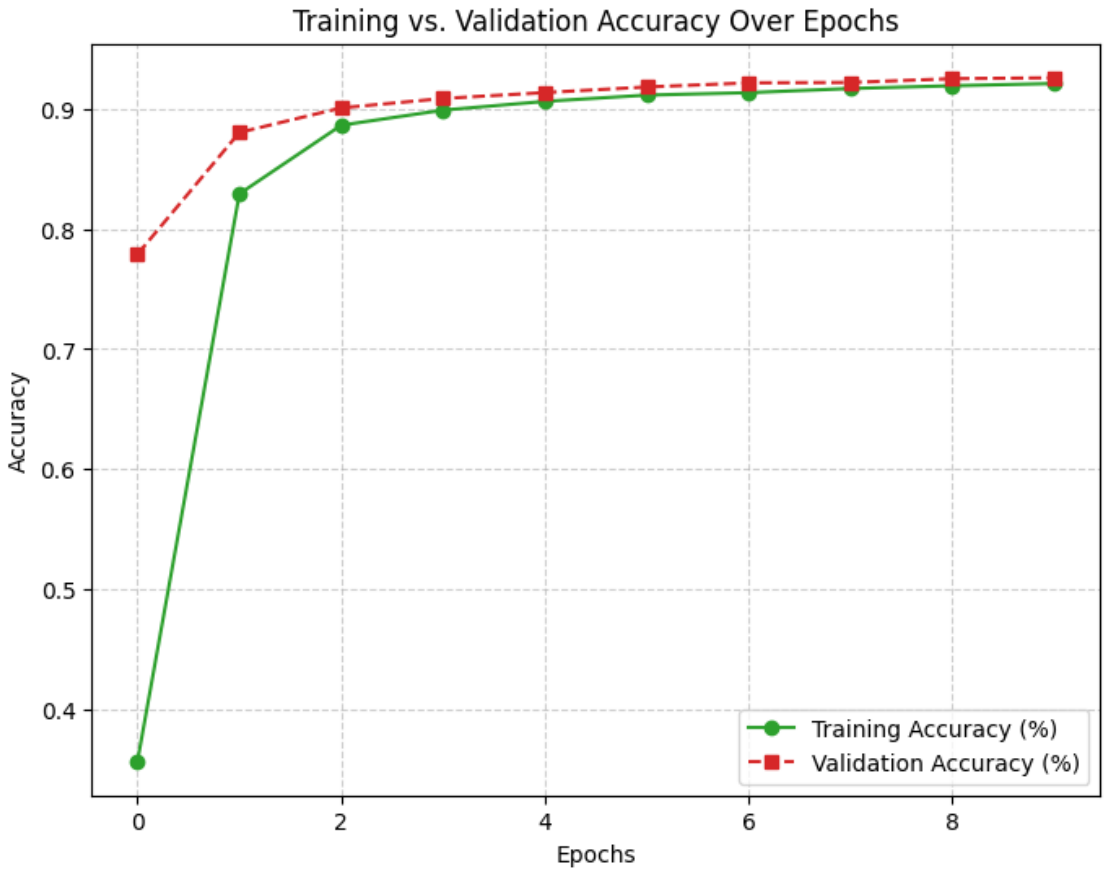
- Tune the Two Layer Neural Network with various learning rates (while keeping all other hyperparameters constant) by changing the config file.
  - lr = 1
  - lr = 1e-1
  - lr = 1e-2
  - lr = 5e-2

[ ]:
```
# 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("configs/config_exp.yaml")
```

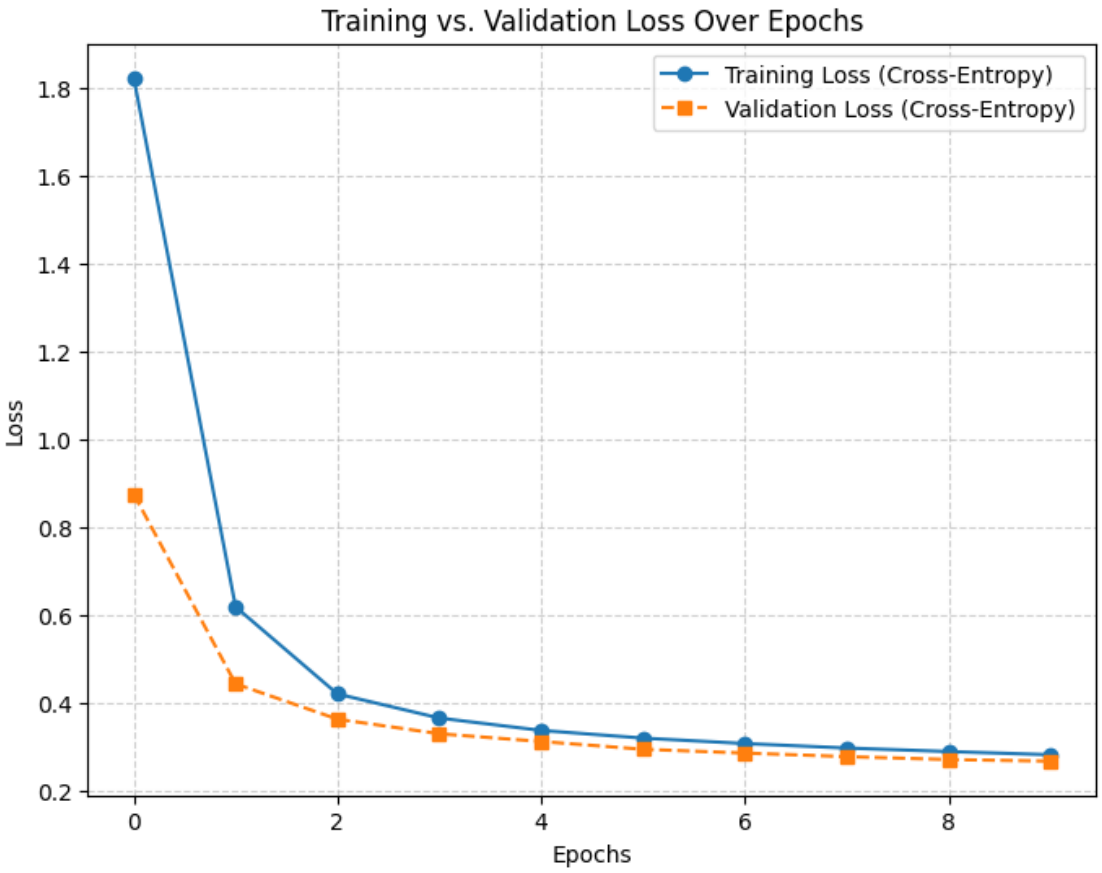
[15]:
```
# Cell 11
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

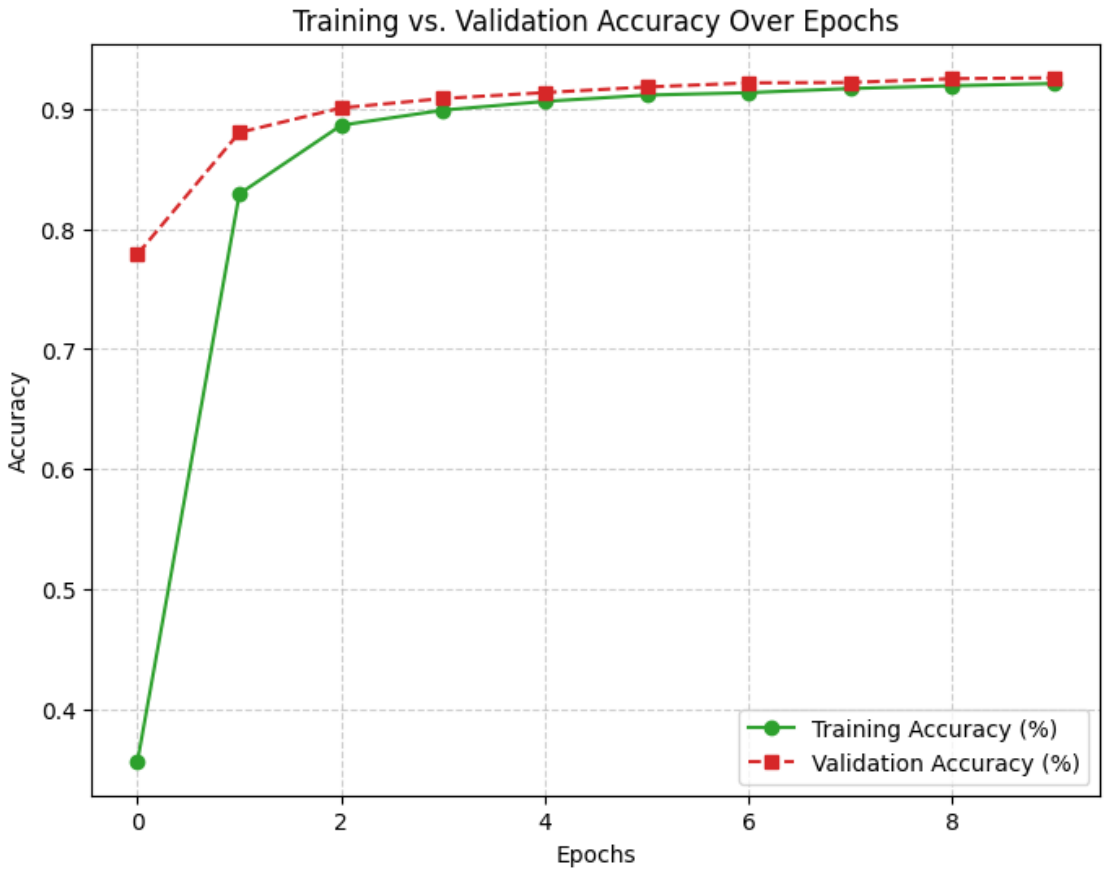




[ ]:
```
# Cell 12
# Change lr to 1e-1 in the config file and run this code block
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history =␣
↪train_model("configs/config_exp.yaml")
```

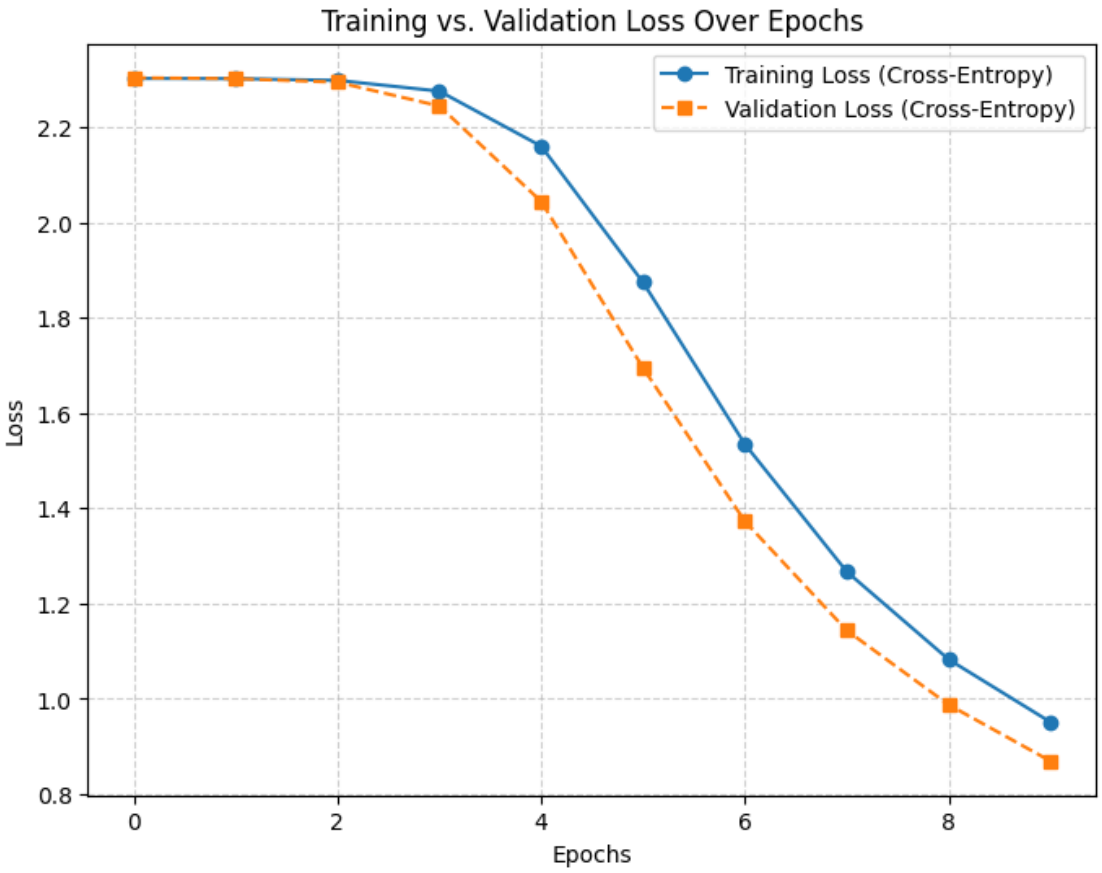
[14]:
```
# Cell 13
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

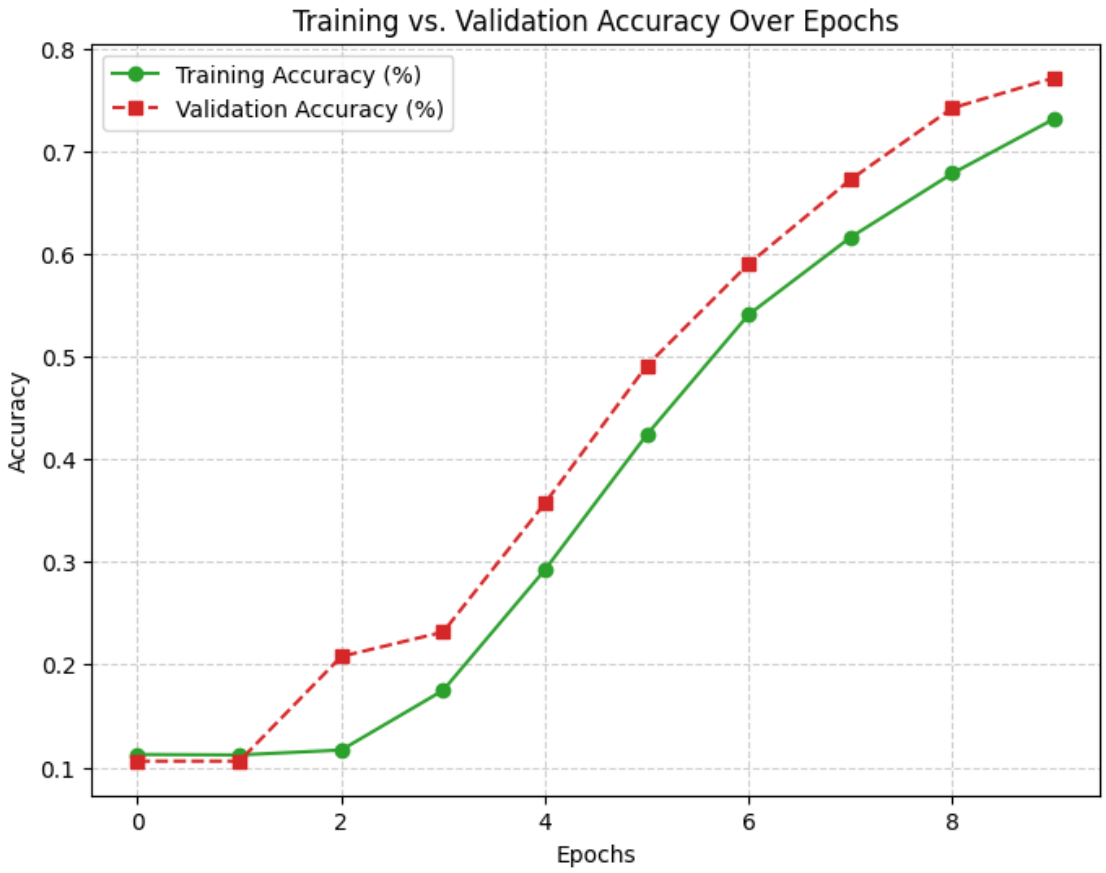

Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8



[ ]:
```
# 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("configs/config_exp.yaml")
```

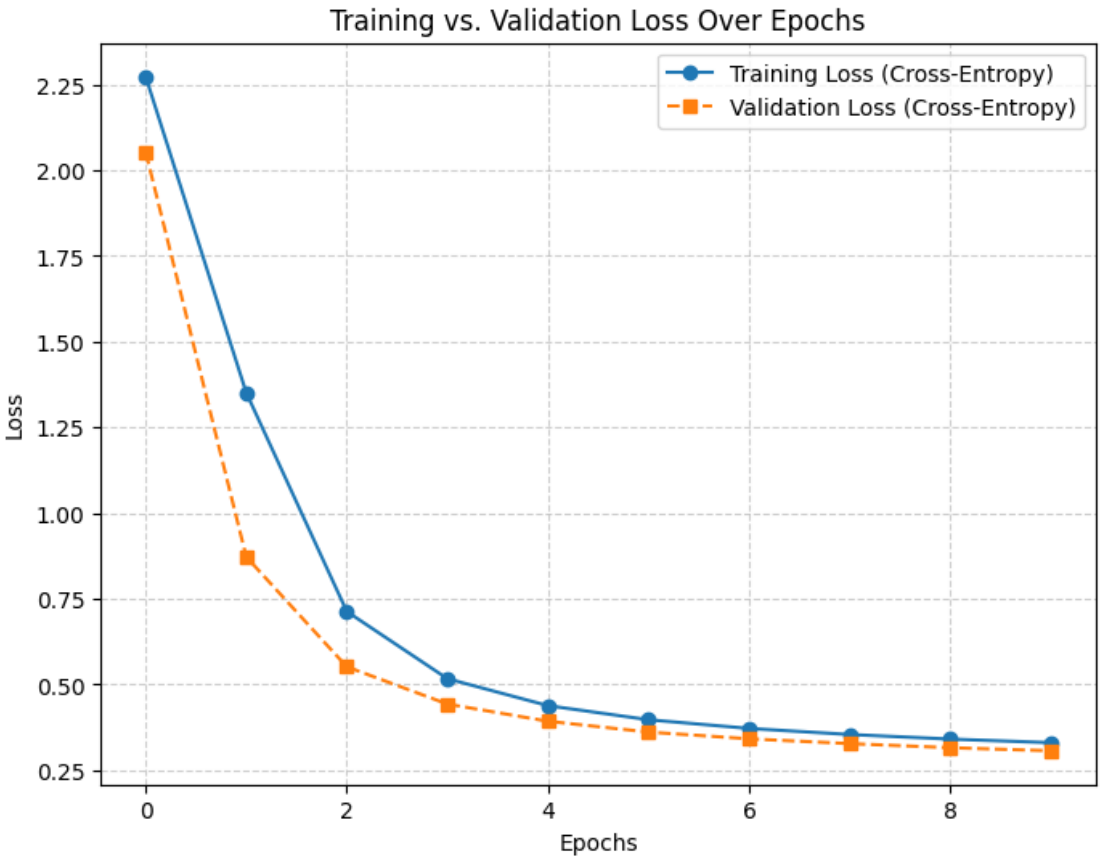
[17]:
```
# Cell 15
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

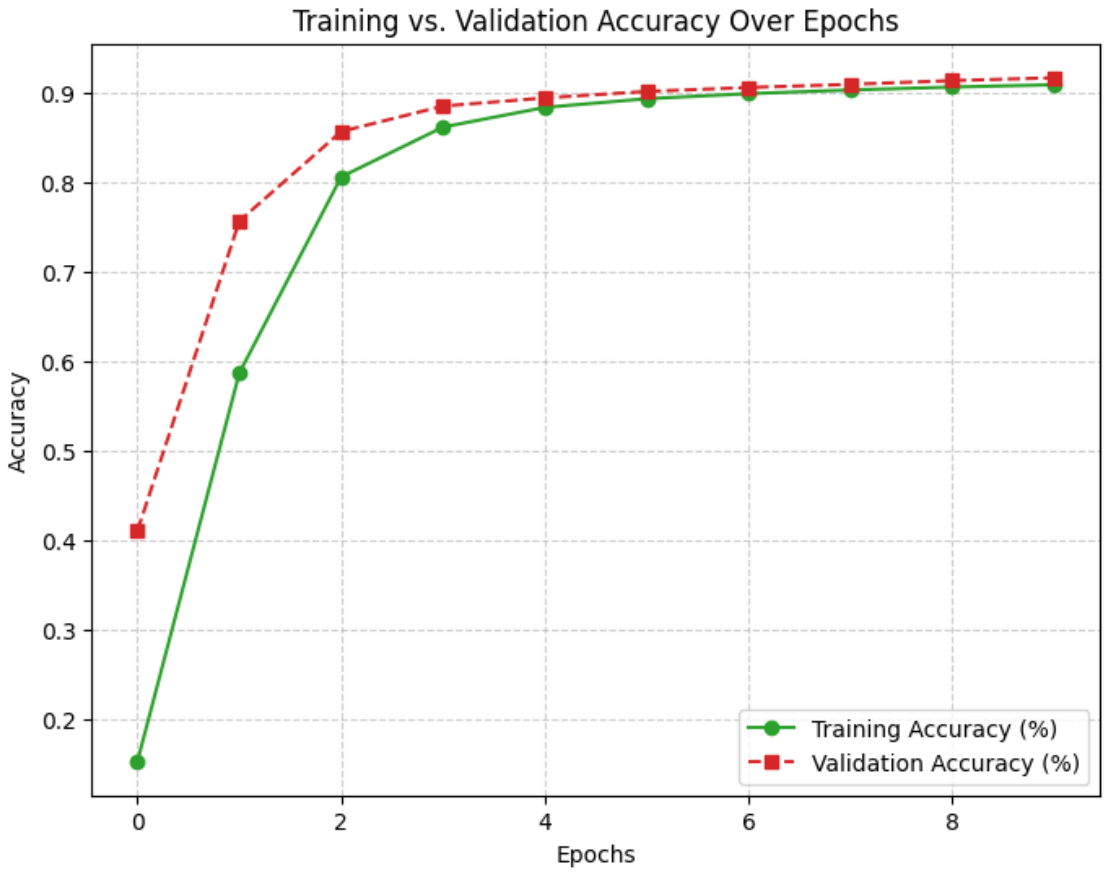

Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
2.2
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0
2
4
6
8



[ ]:
```
# 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("configs/config_exp.yaml")
```

[19]:
```
# Cell 17
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```


Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
2.25
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
2
4
6
8



Describe and explain your findings here: A high learning rate allows quick weight updates making the model to converge rather quickly (within 3 epochs) with little change thereafter. A large learning rate risks moving around the optimal point, however since MNIST is relatively easy to classify, so the use of a higher learning rate does allow the model to converge quickly to a viable solution without risking of divergence.

A higher learning rate is potentially performing better because gradients in sigmoid networks can be small and a higher learning rate helps to counter slow updates and stagnation in the early stages of training - while at learning rate 0.001 the learning is very slow initially potentially due to this problem.

For deeper networks or datasets with more noise using a high learning rate of 1 may cause divergence from optimal solution, but here the dataset is relatively well behaved.

Moreover, use of a L2 regularization rate helps counter overfitting which may be caused by a higher learning rate when using a low number of epochs as opposed to a smaller learning rate. Also use of a relatively moderate/large batch size of 64 and 128 in the layers respectively helps to smooth out gradient updates causing a learning rate of 1 to perform well. With larger batch sizes gradient estimates are typically more stable so may work well with higher learning rates.

A lower learning rate of 0.1 still converges relatively quickly but is seen as being more stable. The test accuracy is worse (this could be random chance or the model weights aren't as optimized).

The model may potentially be getting stuck in a local minimum. A longer training duration would potentially improve the accuracy.

A smaller learning rate of 0.01 does not look like it converged likely because it learns too slowly. This could be due to the use of a sigmoid activation function which suffers from vanishing gradients, where the gradients are too small which could be due to a small learning rate. This implies that the gradient updates may not be significant enough to optimize weights effectively.

A learning rate of 0.05 converges slower than 0.1 or 1 which makes sense as the updates are smaller. The accuracy is worse than 1 or 0.1 as there may not be a high enough number of epochs to fully optimize the weights.

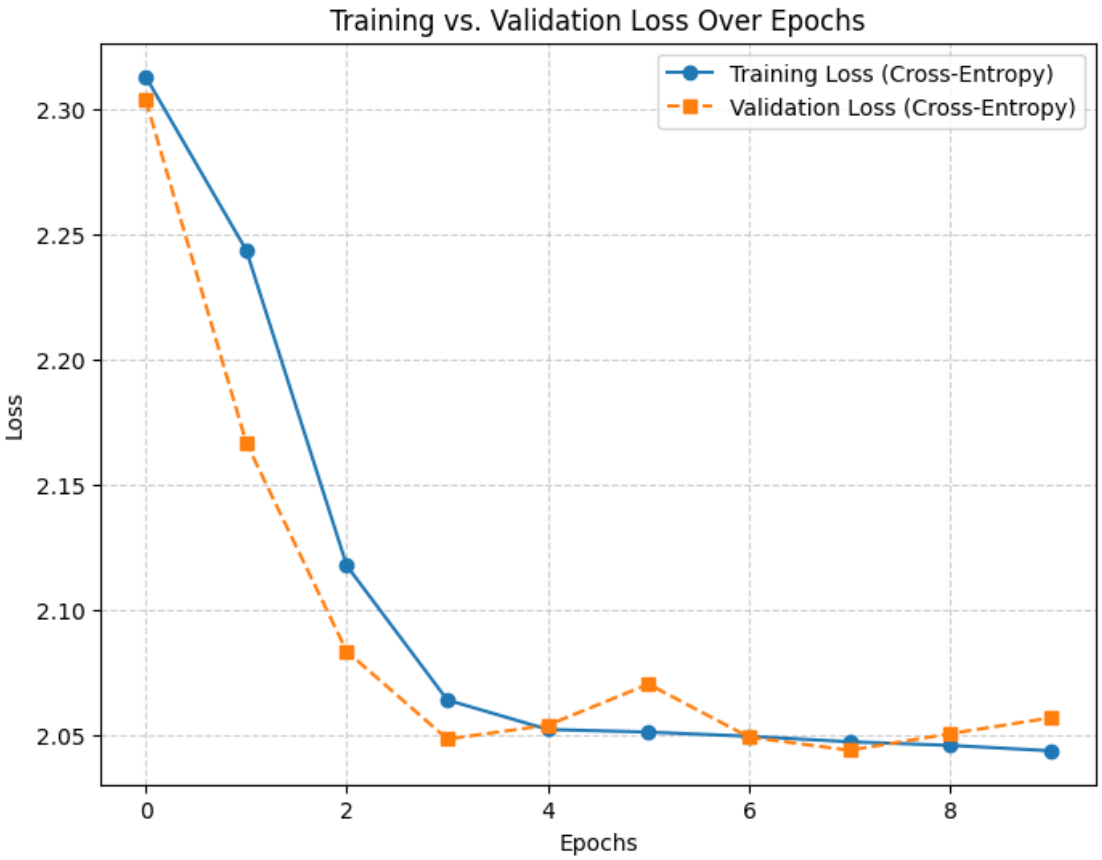
3 Regularization

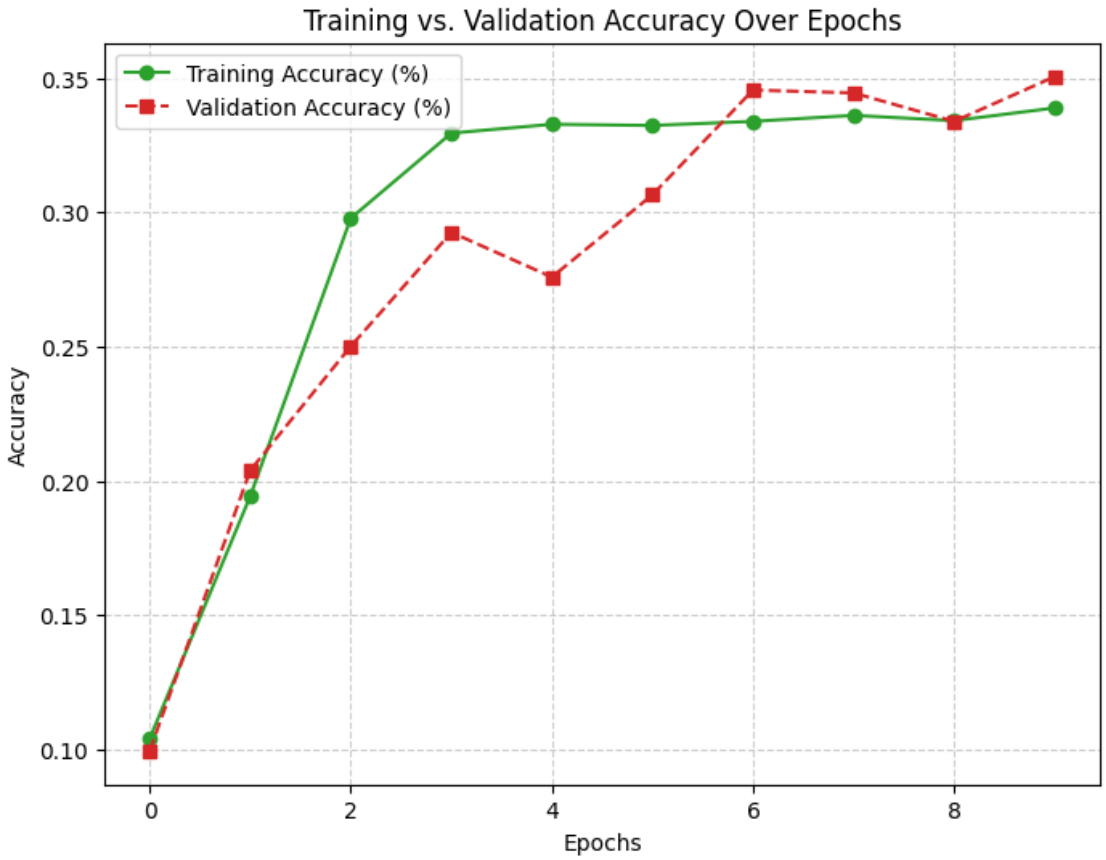
- Tune the Two Layer Neural Network with various regularization coefficients (while keeping all other hyperparameters constant) by changing the config file.
  - reg = 1e-1
  - reg = 1e-2
  - reg = 1e-3
  - reg = 1e-4
  - reg = 1

When you are making changes to the regularization/learning rate values in the `.yaml` files, please do not use scientific notation in the `.yaml` files i.e. instead of writing `1e-1` please write `0.1`. You may create multiple config files for tuning the learning rate and the regularization strength.

[ ]:
```
# 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("configs/config_exp.yaml")
```

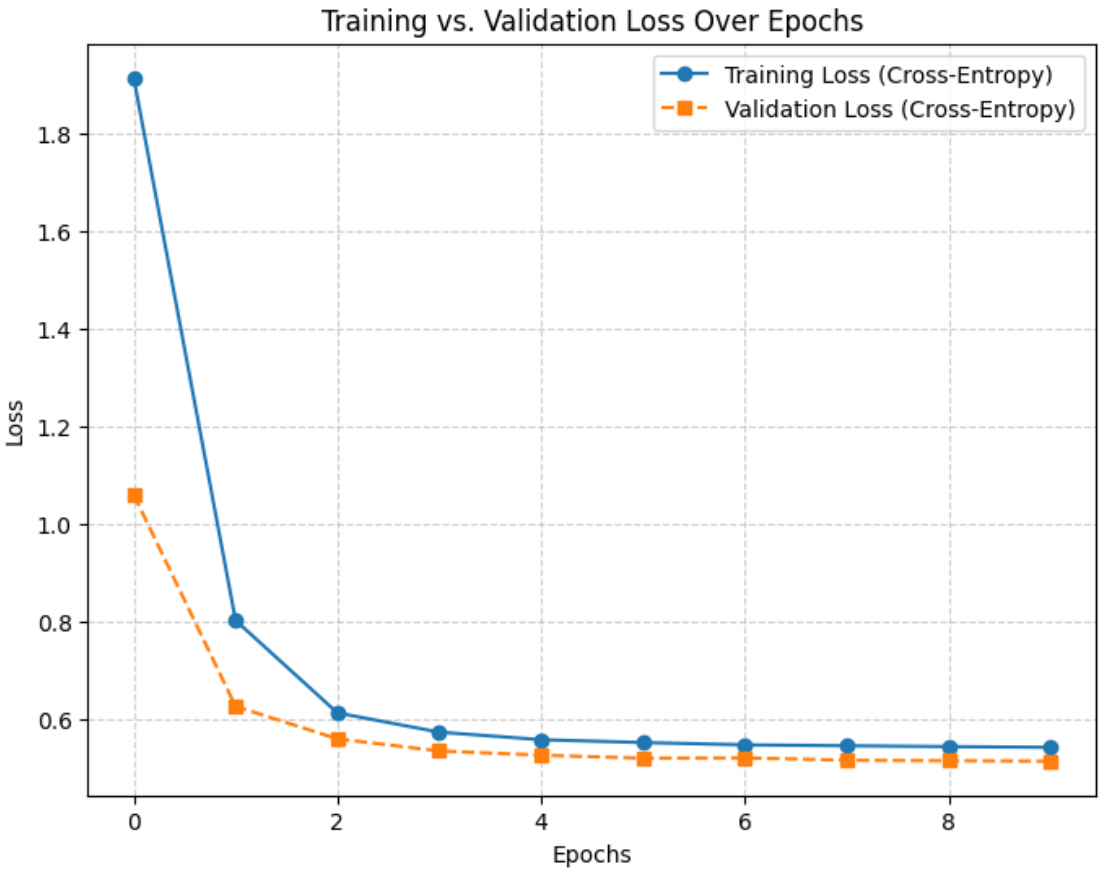
[21]:
```
# Cell 19
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

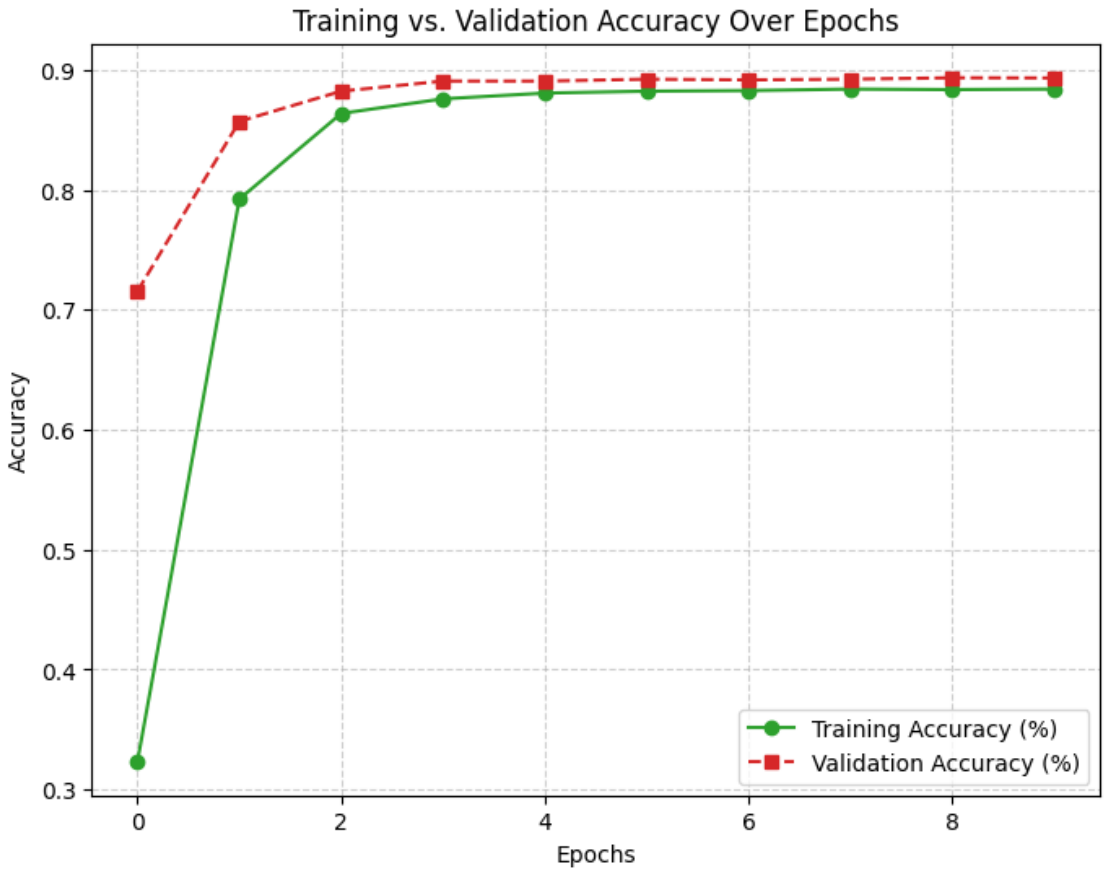

Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
2.30
2.25
2.20
2.15
2.10
2.05
0
2
4
6
8



[ ]:
```
# 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("configs/config_exp.yaml")
```

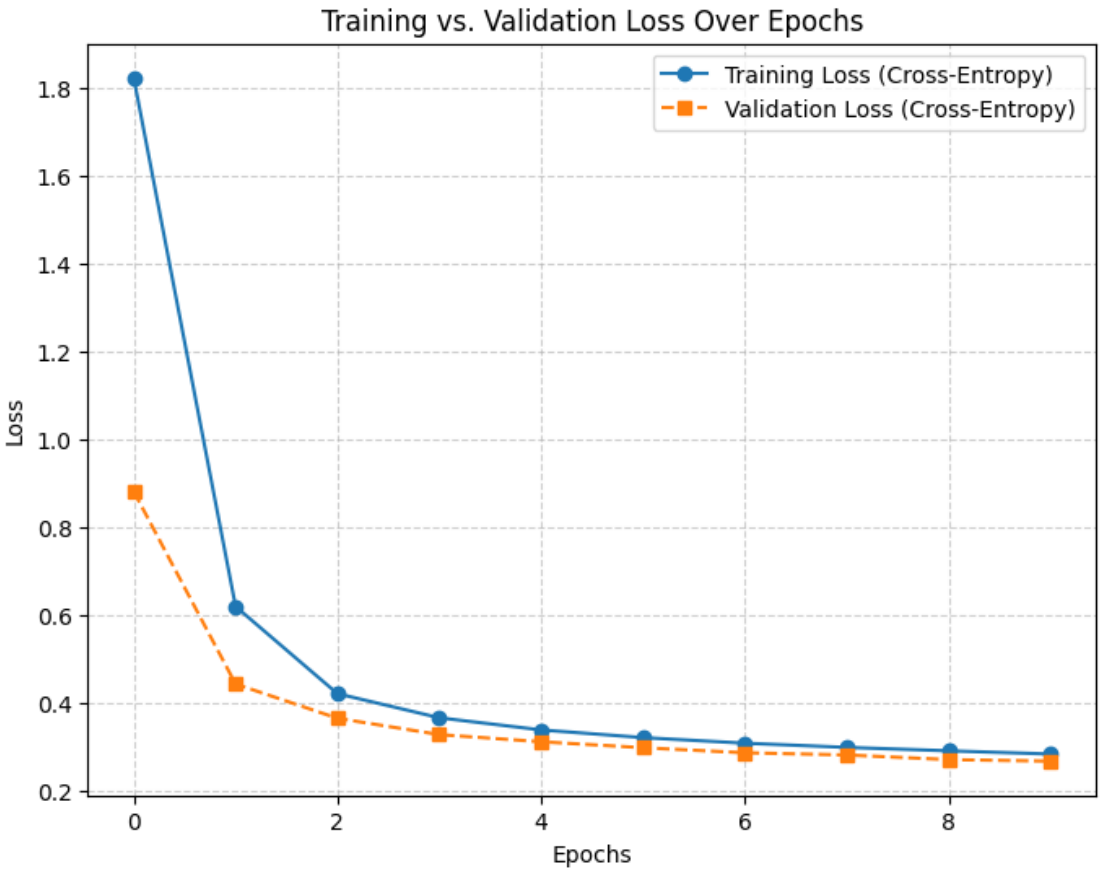
[23]:
```
# Cell 21
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

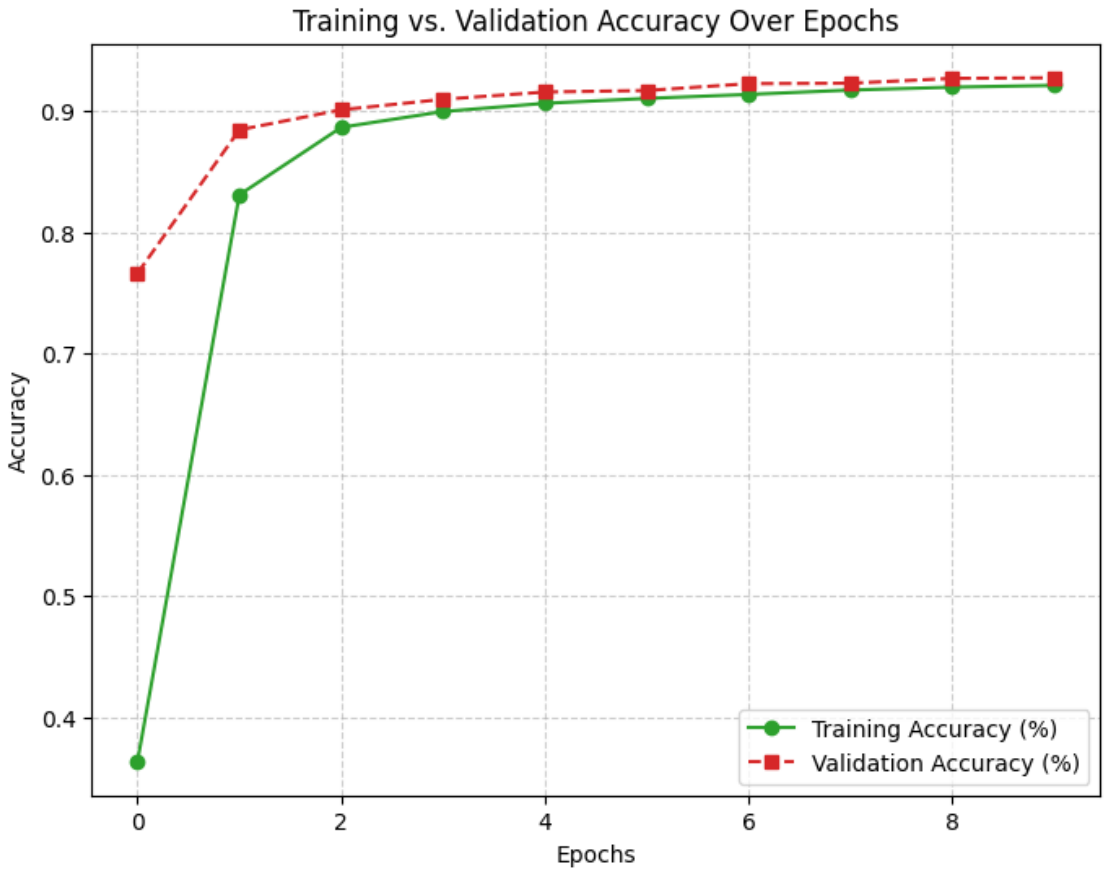

Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0
2
4
6
8



[ ]:
```
# 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("configs/config_exp.yaml")
```

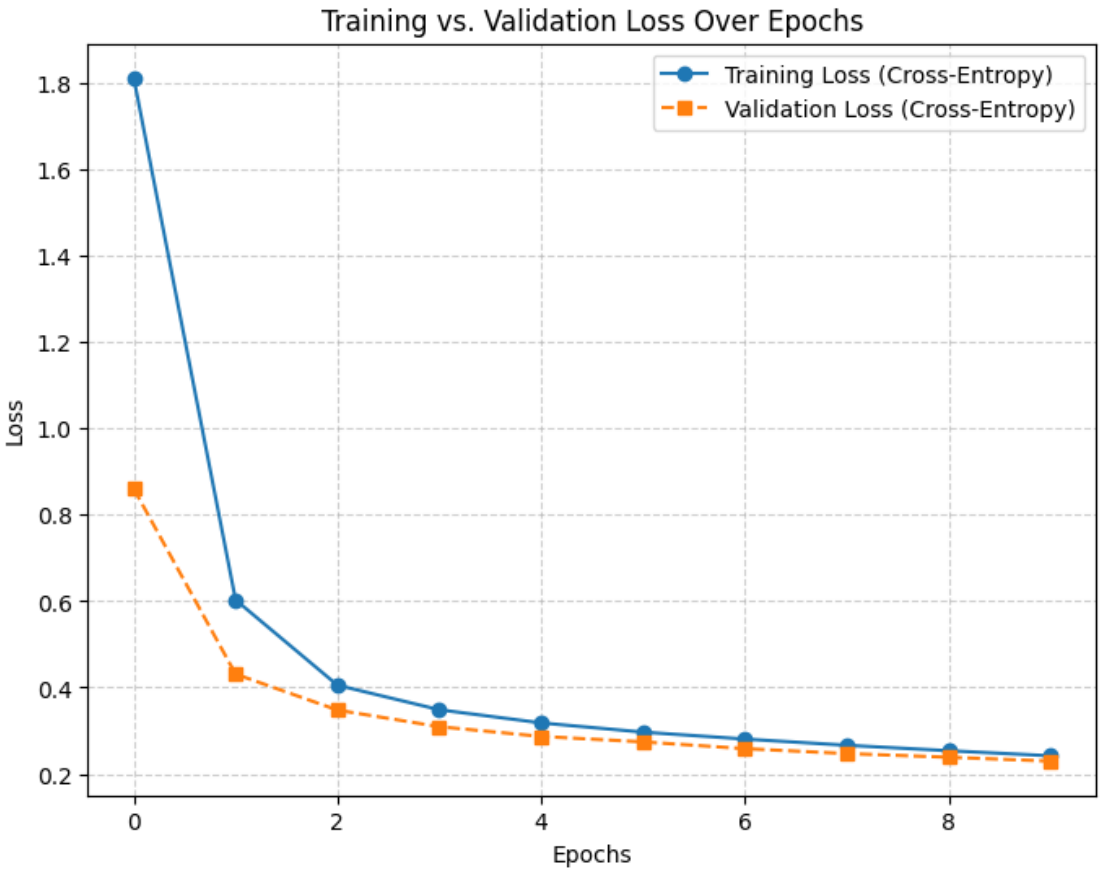
[25]:
```
# Cell 23
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

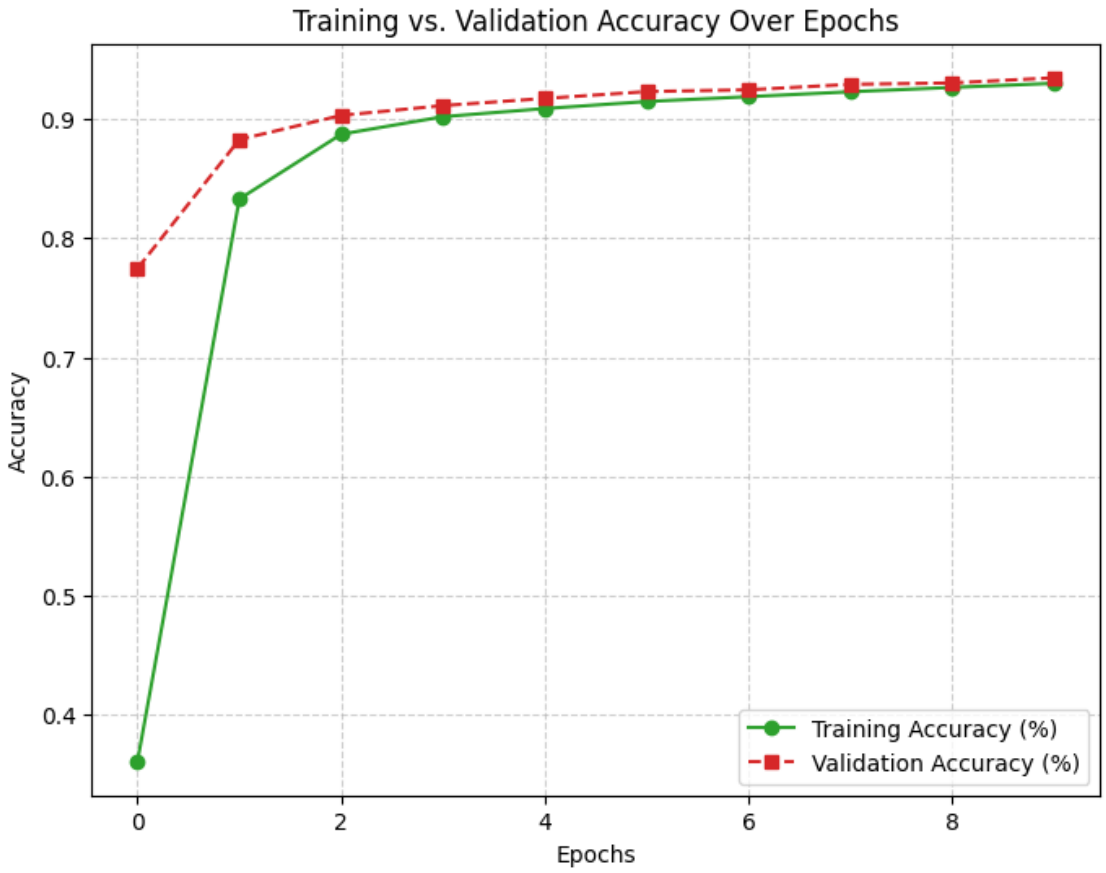

Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8



[ ]:
```
# 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("configs/config_exp.yaml")
```

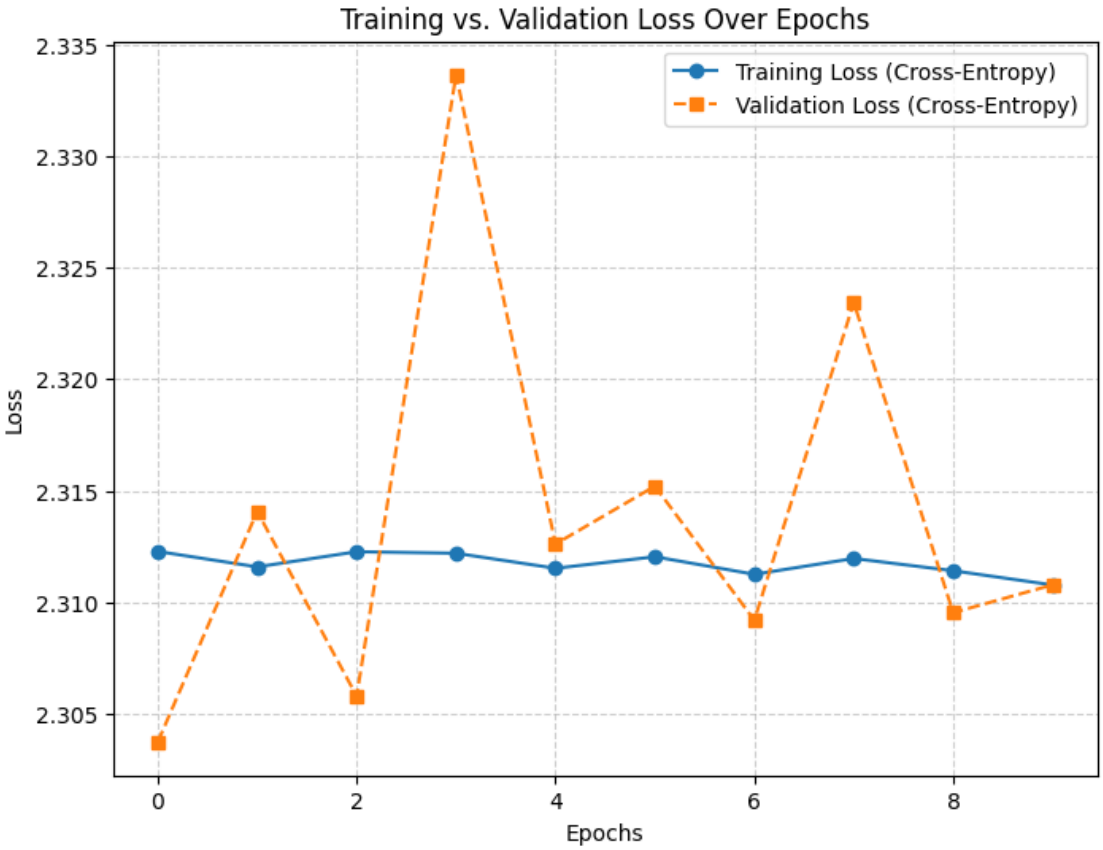
[27]:
```
# Cell 25
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```

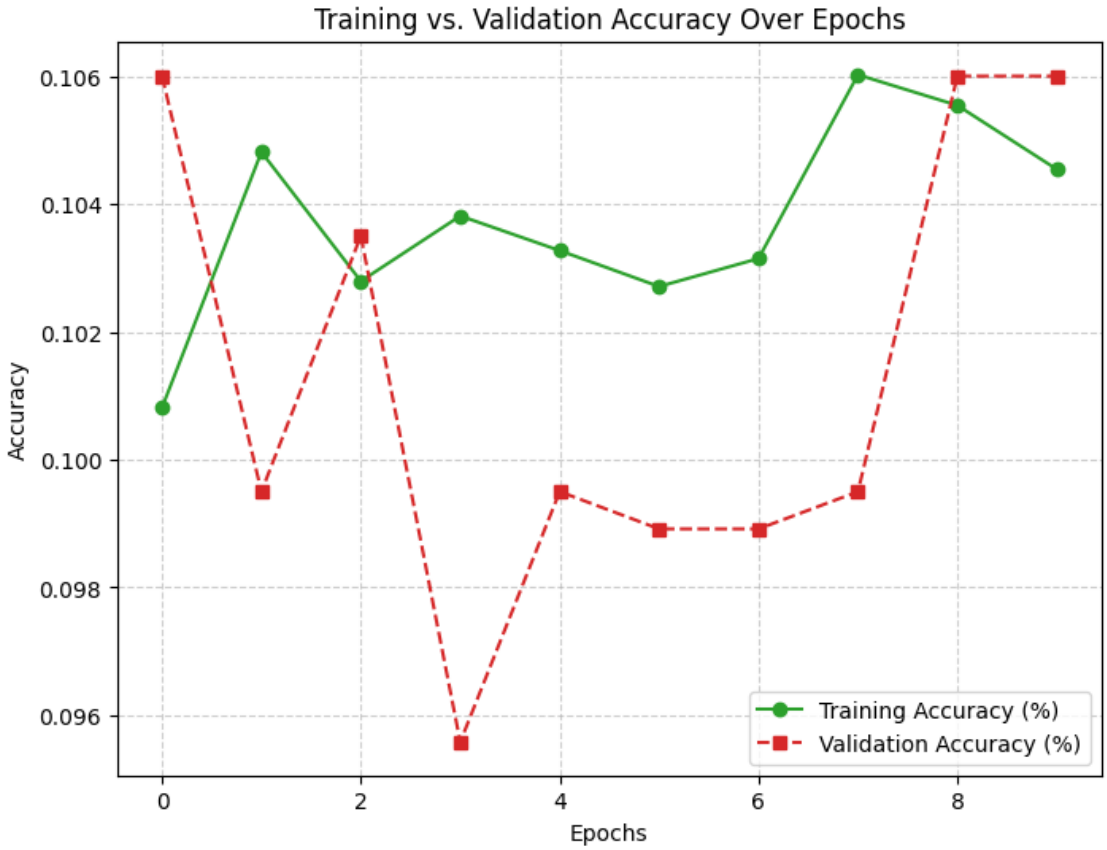

Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0
2
4
6
8



[ ]:
```
# Cell 26
# Change reg 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("configs/config_exp.yaml")
```

[29]:
```
# Cell 27
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```


Training vs. Validation Loss Over Epochs
Training Loss (Cross-Entropy)
Validation Loss (Cross-Entropy)
Loss
Epochs
2.335
2.330
2.325
2.320
2.315
2.310
2.305
0
2
4
6
8



Describe and explain your findings here: The difference in performance has primarily to do with the concept of overfitting and underfitting. A high regularization of 1 or 0.1 perform poorly because regularization penalizes large weights, meaning it encourages the neural network to learn small magnitude parameters. With a high regularization parameter weight updates are very constrained, which prevents the model from fitting patterns to the dataset. This leads to underfitting. Validation accuracy fluctuates because the model can't minimize loss due to very small weights.

Moreover, since the neural network uses a single hidden layer with 128 neurons, combined with a sigmoid - it requires moderately large weights to avoid vanishing gradients.

Lower regularization parameter values work better because MNIST is a relatively well structured dataset so a small amount of weight decay is enough to prevent overfitting. As lambda gets lower and lower the L2 regularization effect diminishes so the difference in accuracy is small.

In conclusion: high values of lambda cause weight constraints to be too high leading to underfitting. While a low regularization parameter could lead to overfitting - not in this case due to the well structured nature of MNIST dataset.

3.1 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 the previous section, you should have achieved around 70% and 90% validation accuracy (+/- 10%) with the softmax and two-layer networks, respectively. For your tuning, we expect at least 65% validation accuracy, but try to beat your best numbers from the previous section.

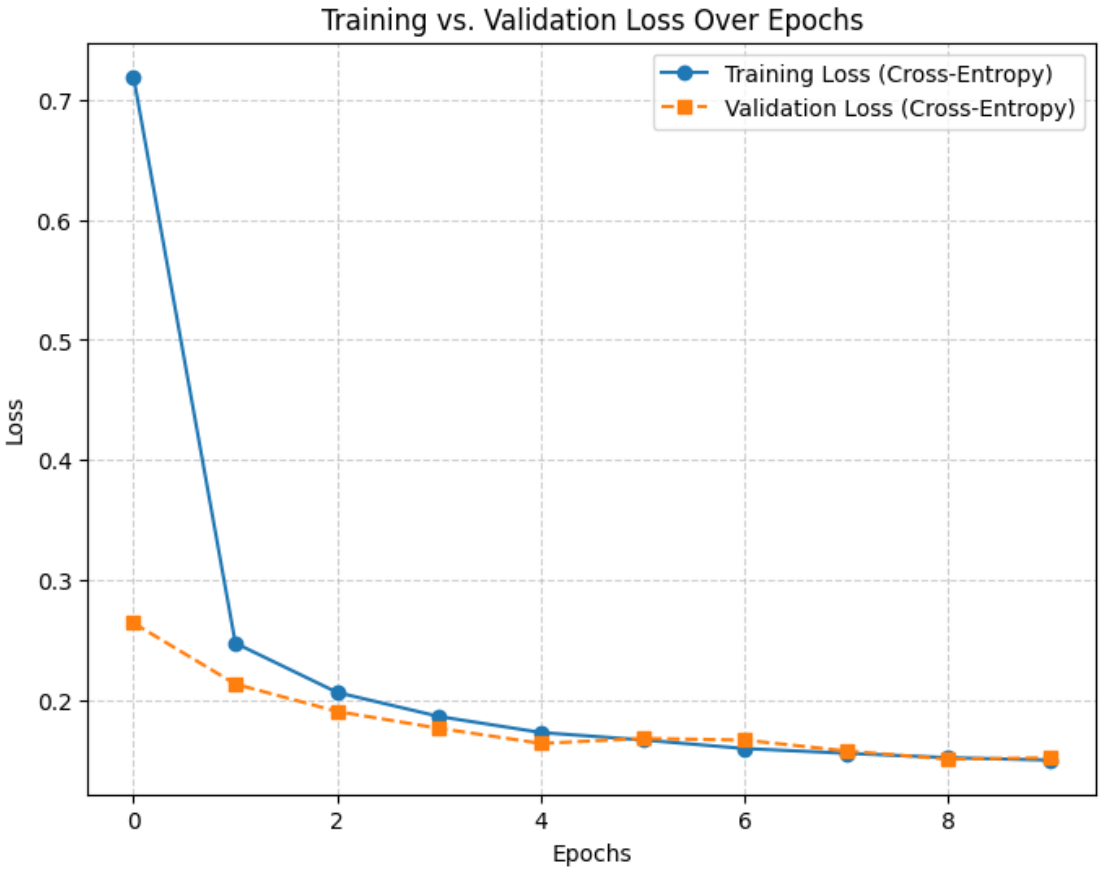
Findings below: I reduced the regularization parameter lambda from 0.001 in default setting to 0.0005 allowing the model to learn richer features - since the dataset is well structured and the batch sizes are of moderate size - there isn't as much risk of overfitting.

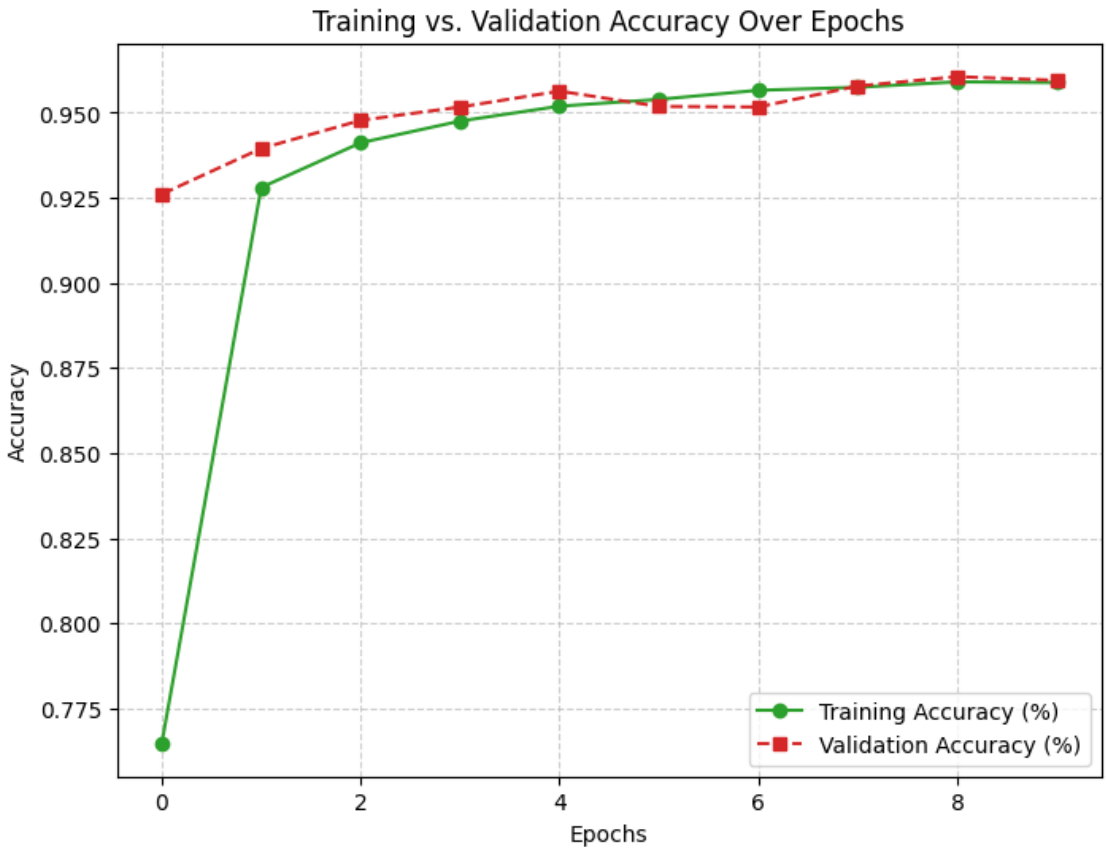
When I used a larger batch size of 128 and 256 for respective layers the model didn't generalize as well and the performance on test set worsened. Large batch sizes reduce gradient noise and cause more determinsistic updates and so potentially causing overfitting.

I increased the learning rate to 1 (generally this may not be a good idea for noisier datasets, but for a well behaved dataset like MNIST a higher learning rate allowed faster convergence and optimization of learning, where the risk of divergence is smaller than for other datasets).

[ ]:
```
# Cell 28
# hyperparameter tuning
train_loss_history, train_acc_history, valid_loss_history, valid_acc_history =␣
↪train_model("configs/config_exp.yaml")
```

[34]:
```
# Cell 29
plot_curves(train_loss_history, train_acc_history, valid_loss_history,␣
↪valid_acc_history)
```




Training vs. Validation Accuracy Over Epochs
Accuracy
Epochs
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0
2
4
6
8
Training Accuracy (%)
Validation Accuracy (%)