Setup Colab

Here we setup Colab, and import some useful packages.

```
from google.colab import drive
drive.mount('/content/gdrive')
```

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

```
import os
os.chdir('/content/gdrive/MyDrive/Assignment1/Startpkg_A1')
```

```
import random
import numpy as np
import matplotlib.pyplot as plt
from data_process import get_CIFAR10_data
import math
from scipy.spatial import distance
from models import Perceptron, Softmax
from kaggle_submission import output_submission_csv
%matplotlib inline
```

Loading CIFAR-10

In the following cells we determine the number of images for each split and load the images.

```
# You can change these numbers for experimentation
# For submission we will use the default values
TRAIN_IMAGES = 49000
VAL_IMAGES = 1000
TEST_IMAGES = 5000  # Keep this default as 5000 for your submission
```

```
data = get_CIFAR10_data(TRAIN_IMAGES, VAL_IMAGES, TEST_IMAGES)
X_train, y_train = data['X_train'], data['y_train']
X_val, y_val = data['X_val'], data['y_val']
X_test, y_test = data['X_test'], data['y_test']
```

Convert the sets of images from dimensions of **(N, 3, 32, 32) -> (N, 3072)** where N is the number of images so that each **3x32x32** image is represented by a single vector.

```
X_train = np.reshape(X_train, (X_train.shape[0], -1))
X_val = np.reshape(X_val, (X_val.shape[0], -1))
X_test = np.reshape(X_test, (X_test.shape[0], -1))
```

Get Accuracy

Double-click (or enter) to edit

This function computes how well your model performs using accuracy as a metric.

Double-click (or enter) to edit

```
def get_acc(pred, y_test):
    return np.sum(y_test==pred)/len(y_test)*100
```

Perceptron

Perceptron has 2 hyperparameters that you can experiment with:

- **Learning rate** - controls how much we change the current weights of the classifier during each update. We set it at a default value of 0.5, but you should experiment with different values. We recommend changing the learning rate by factors of 10 and observing how the performance of the classifier changes. You should also try adding a **decay** which slowly reduces the learning rate over each epoch.
- **Number of Epochs** - An epoch is a complete iterative pass over all of the data in the dataset. During an epoch we predict a label using the classifier and then update the weights of the classifier according the perceptron update rule for each sample in the training set. You should try different values for the number of training epochs and report your results.

You will implement the Perceptron classifier in the **models/Perceptron.py**. You may directly edit it by open it from the Files icon located on the left sidebar.

The following code:

- Creates an instance of the Perceptron classifier class
- The train function of the Perceptron class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

Train Perceptron

```
alpha_list = [0.01]
epochs_list = [200]
decay = 0.8
i = 0

#variables to capture Optimal Hyper-parameters
best_alpha = None
best_epochs = None
best_predict = None
best_accuracy = 0
best_test_accuracy = 0

# nested for loop to use all possible alphas & epochs
for alpha in alpha_list:
  for epochs in epochs_list:
    i+=1 #counter

    print(f"Experiment {i} with alpha={alpha}, epochs={epochs}, decay={decay}")
    percept_ = Perceptron(alpha=alpha, epochs=epochs,decay=decay)
    percept_.train(X_train, y_train,X_val,y_val)

    pred_percept_train = percept_.predict(X_train)
    print('Training Accuracy: %f' % (get_acc(pred_percept_train, y_train)))
    pred_percept_val = percept_.predict(X_val)
    print('Validation Accuracy: %f' % (get_acc(pred_percept_val, y_val)))
    pred_percept_test = percept_.predict(X_test)
    print('Testing Accuracy: %f' % (get_acc(pred_percept_test, y_test)))

    # Plot loss vs number of iterations
    plt.figure(figsize=(12, 5))
    plt.plot(range(epochs), percept_.losses, label='Loss')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.title(f'Loss vs No of Iterations - (alpha={alpha}, epochs={epochs})')
    plt.suptitle(f'Experiment 3 Perceptron, vyeruban')
    plt.legend()
    plt.show()

    # Plot training and validation accuracy vs number of iterations
    plt.figure(figsize=(12, 5))
    plt.plot(range(epochs), percept_.train_accuracies, label='Training Accuracy')
    plt.plot(range(epochs), percept_.val_accuracies, label='Validation Accuracy')
    plt.xlabel('Iterations')
    plt.ylabel('Accuracy')
    plt.title(f'Accuracy vs No of Iterations - (alpha={alpha}, epochs={epochs})')
    plt.suptitle(f'Experiment 3 Perceptron, vyeruban')
    plt.legend()
    plt.show()

    # Check if this is the best accuracy we've encountered
    if get_acc(pred_percept_test, y_test) > best_test_accuracy:
```

```
        best_test_accuracy = get_acc(pred_percept_test, y_test)
        best_alpha = alpha
        best_epochs = epochs
        best_predict = pred_percept_test
```

```
Experiment 1 with alpha=0.01, epochs=200, decay=0.8
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-8-107c4162bfc9> in <cell line: 14>()
     18     print(f"Experiment {i} with alpha={alpha}, epochs={epochs}, decay={decay}")
     19     percept_ = Perceptron(alpha=alpha, epochs=epochs,decay=decay)
---> 20     percept_.train(X_train, y_train,X_val,y_val)
     21 
     22     pred_percept_train = percept_.predict(X_train)

/content/gdrive/MyDrive/Assignment1/Startpkg_A1/models/Perceptron.py in train(self, X_train, y_train, X_val, y_val)
     35             correct = 0
     36             for i in range(len(y_train)):
---> 37                 pred = np.argmax(np.dot(self.w, X_train[i]))
     38                 if pred != y_train[i]:
     39                     self.w[pred] -= self.alpha * X_train[i]

KeyboardInterrupt: 
```

```
pred_percept = percept_.predict(X_train)
print('The training accuracy is given by : %f' % (get_acc(pred_percept, y_train)))
```

```
The training accuracy is given by : 39.489796
```

Validation

```
pred_percept = percept_.predict(X_val)
print('The validation accuracy is given by : %f' % (get_acc(pred_percept, y_val)))
```

```
The validation accuracy is given by : 30.200000
```

Test Perceptron

```
pred_percept = percept_.predict(X_test)
print('The testing accuracy is given by : %f' % (get_acc(pred_percept, y_test)))
```

```
The testing accuracy is given by : 31.000000
```

Perceptron Kaggle Submission

Once you are satisfied with your solution and test accuracy output a file to submit your test set predictions to the Kaggle for Assignment 1 Perceptron. Use the following code to do so:

```
output_submission_csv('perceptron_submission.csv', percept_.predict(X_test))
```

Softmax Classifier (with SGD)

Next, you will train a Softmax classifier. This classifier consists of a linear function of the input data followed by a softmax function which outputs a vector of dimension C (number of classes) for each data point. Each entry of the softmax output vector corresponds to a confidence in one of the C classes, and like a probability distribution, the entries of the output vector sum to 1. We use a cross-entropy loss on this sotmax output to train the model.

Check the following link as an additional resource on softmax classification: http://cs231n.github.io/linear-classify/#softmax

Once again we will train the classifier with SGD. This means you need to compute the gradients of the softmax cross-entropy loss function according to the weights and update the weights using this gradient. Check the following link to help with implementing the gradient updates: https://deepnotes.io/softmax-crossentropy

The softmax classifier has 3 hyperparameters that you can experiment with :

- **Learning rate** - As above, this controls how much the model weights are updated with respect to their gradient.
- **Number of Epochs** - As described for perceptron.
- **Regularization constant** - Hyperparameter to determine the strength of regularization. In this case, we minimize the L2 norm of the model weights as regularization, so the regularization constant is a coefficient on the L2 norm in the combined cross-entropy and regularization objective.

You will implement a softmax classifier using SGD in the **models/Softmax.py**

The following code:

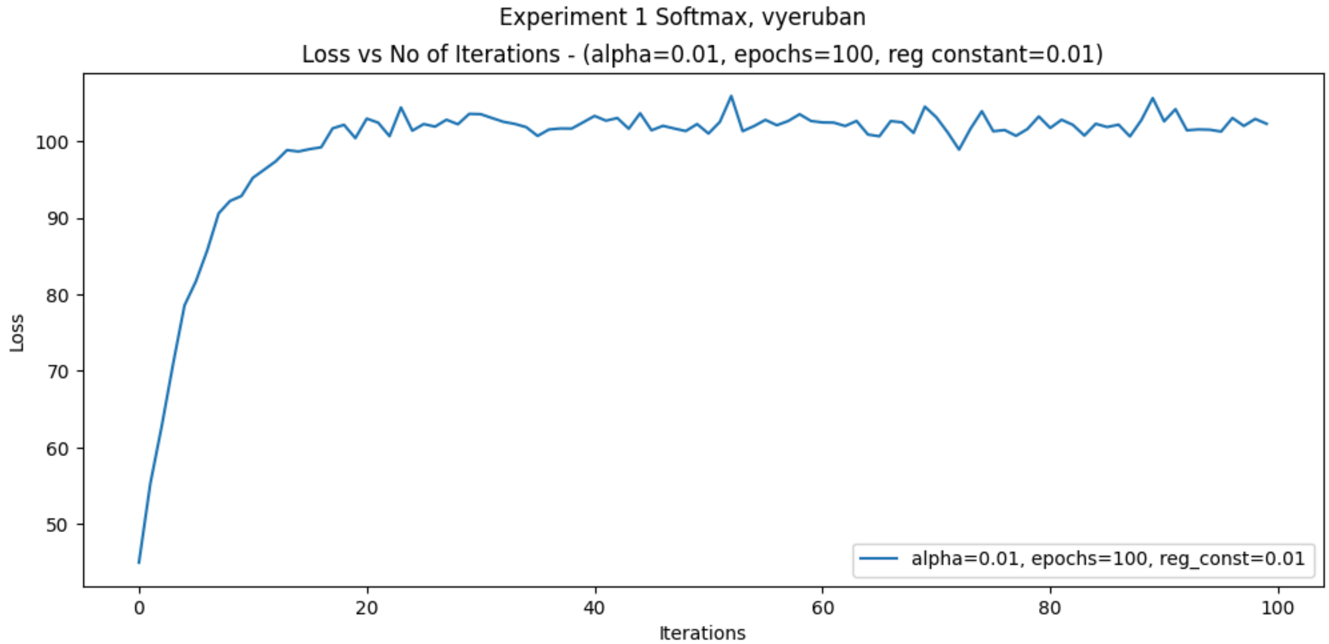
- Creates an instance of the Softmax classifier class
- The train function of the Softmax class is trained on the training data
- We use the predict function to find the training accuracy as well as the testing accuracy

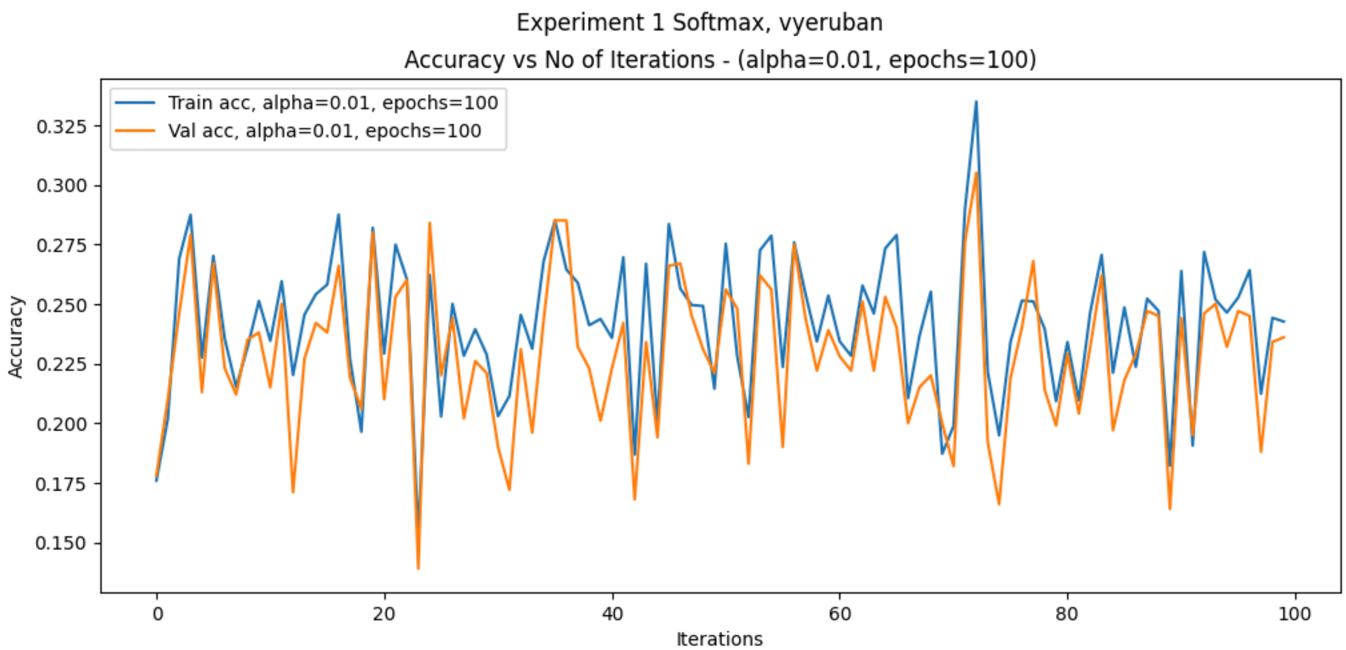
Train Softmax

```
alphas = [0.01, 0.1]
epochs_list = [100, 200]
reg_consts = [0.01, 0.05]
i=0

#variables to capture Optimal Hyper-parameters
best_alpha = None
best_epochs = None
best_reg_const = None
best_pred = None
best_accuracy = 0

# nested for loop to use all possible alphas & epochs
for alpha in alphas:
  for epochs in epochs_list:
      for reg_const in reg_consts:

          name = 'vvaddi2' if alpha == 0.01 else 'vyeruban'

          # Create a Softmax classifier
          softmax = Softmax(alpha=alpha, epochs=epochs, reg_const=reg_const)
          train_loss, train_acc, val_acc = softmax.train(X_train, y_train, X_val, y_val)

          i+=1 #counter
          print(f"Experiment {i} with alpha={alpha}, epochs={epochs}, reg_const={reg_const}")

          #Training Loss vs no of Iterations
          plt.figure(figsize=(12, 5))
          plt.plot(np.arange(len(train_loss)), train_loss, label=f"alpha={alpha}, epochs={epochs}, reg_const={reg_const}")
          plt.xlabel('Iterations')
          plt.ylabel('Loss')
          plt.title(f'Loss vs No of Iterations - (alpha={alpha}, epochs={epochs}, reg constant={reg_const})')
          plt.suptitle(f'Experiment {i} Softmax, vyeruban')
          plt.legend()
          plt.show()

          #Training & Validation Accuracy vs no of Iterations
          plt.figure(figsize=(12, 5))
          plt.plot(range(len(train_acc)), train_acc, label=f"Train acc, alpha={alpha}, epochs={epochs}")
          plt.plot(range(len(val_acc)), val_acc, label=f"Val acc, alpha={alpha}, epochs={epochs}")
          plt.title(f'Training and Validation Accuracy vs Number of Iterations (alpha={alpha}, epochs={epochs}, reg_const={reg_co
          plt.xlabel('Iterations')
          plt.ylabel('Accuracy')
          plt.title(f'Accuracy vs No of Iterations - (alpha={alpha}, epochs={epochs})')
          plt.suptitle(f'Experiment {i} Softmax, vyeruban')
          plt.legend()
          plt.show()

          print("------------------------------------")
          train_pred = softmax.predict(X_train)
          train_accuracy = get_acc(train_pred, y_train)
          print(f'Training Accuracy: {train_accuracy}')
          val_pred = softmax.predict(X_val)
          val_accuracy = get_acc(val_pred, y_val)
          print(f'Validation Accuracy: {val_accuracy}')
          test_pred = softmax.predict(X_test)
          test_accuracy = get_acc(test_pred, y_test)
```

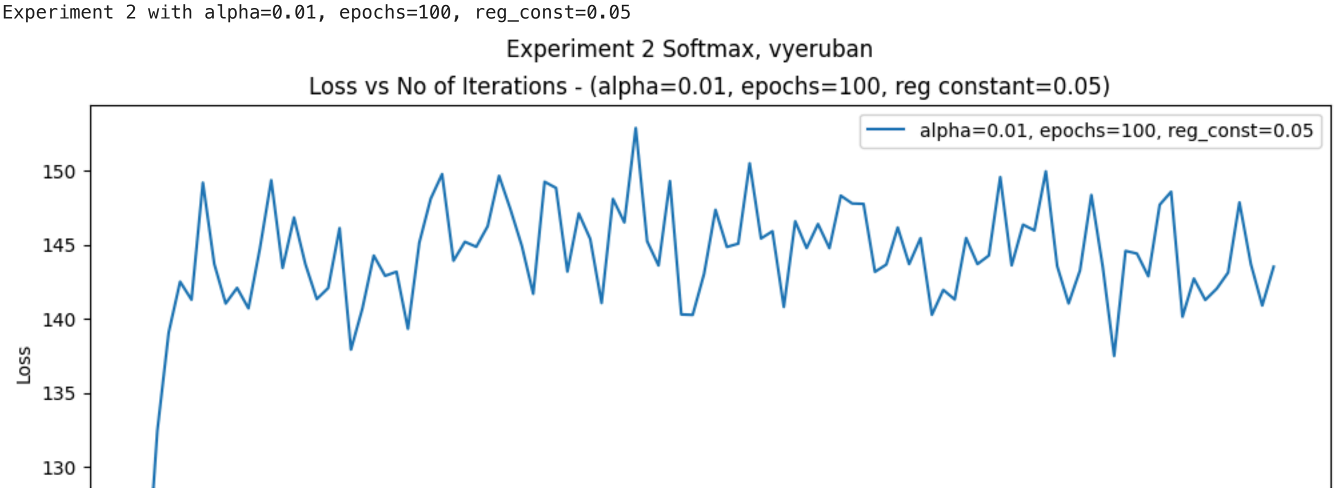
```
          test_accuracy = get_acc(test_pred, y_test)
          print(f'Test Accuracy:{test_accuracy}')
          print("-----------------------------------")

          #Optimal Hyper-parameters for Kaggle
          if test_accuracy > best_accuracy:
              best_accuracy = test_accuracy
              best_alpha = alpha
              best_reg_const = reg_const
              best_epochs = epochs

print(f"Best alpha: {best_alpha}, Best epochs: {best_epochs}, Best accuracy: {best_accuracy}, Best Regularization Constant: {best_
```

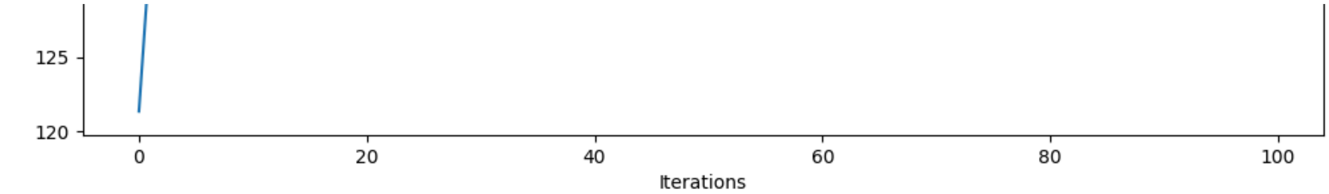
```
Experiment 1 with alpha=0.01, epochs=100, reg_const=0.01
```

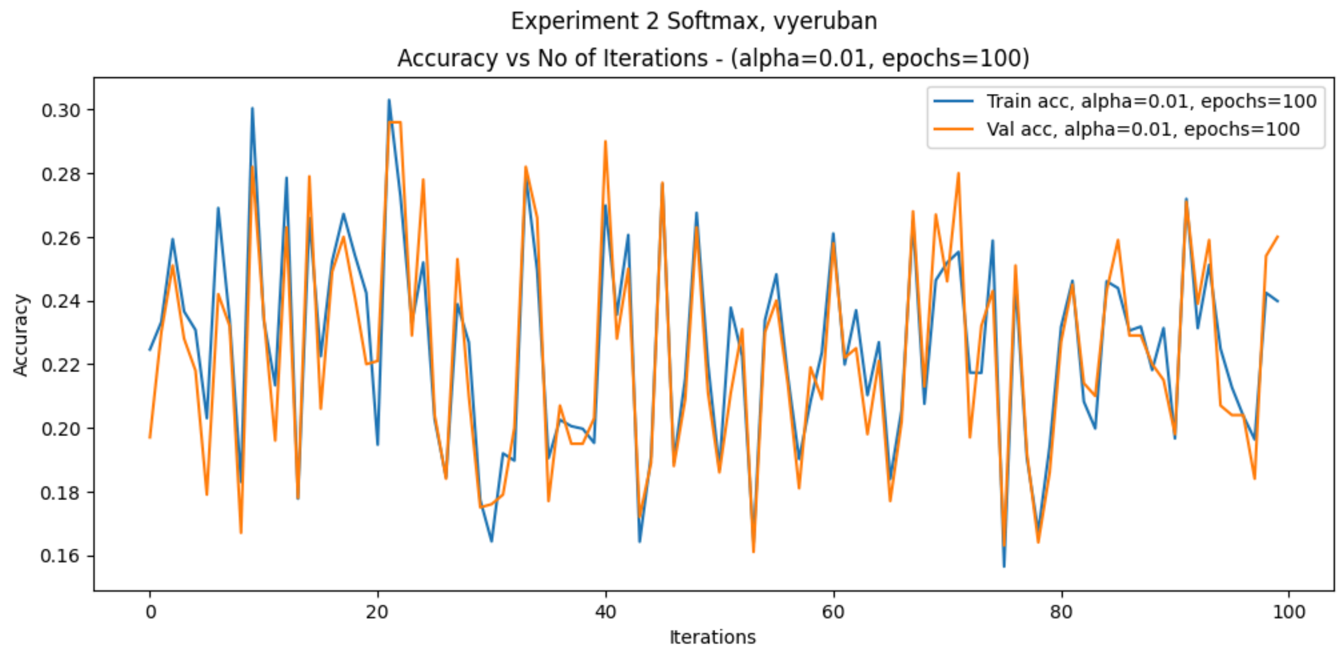




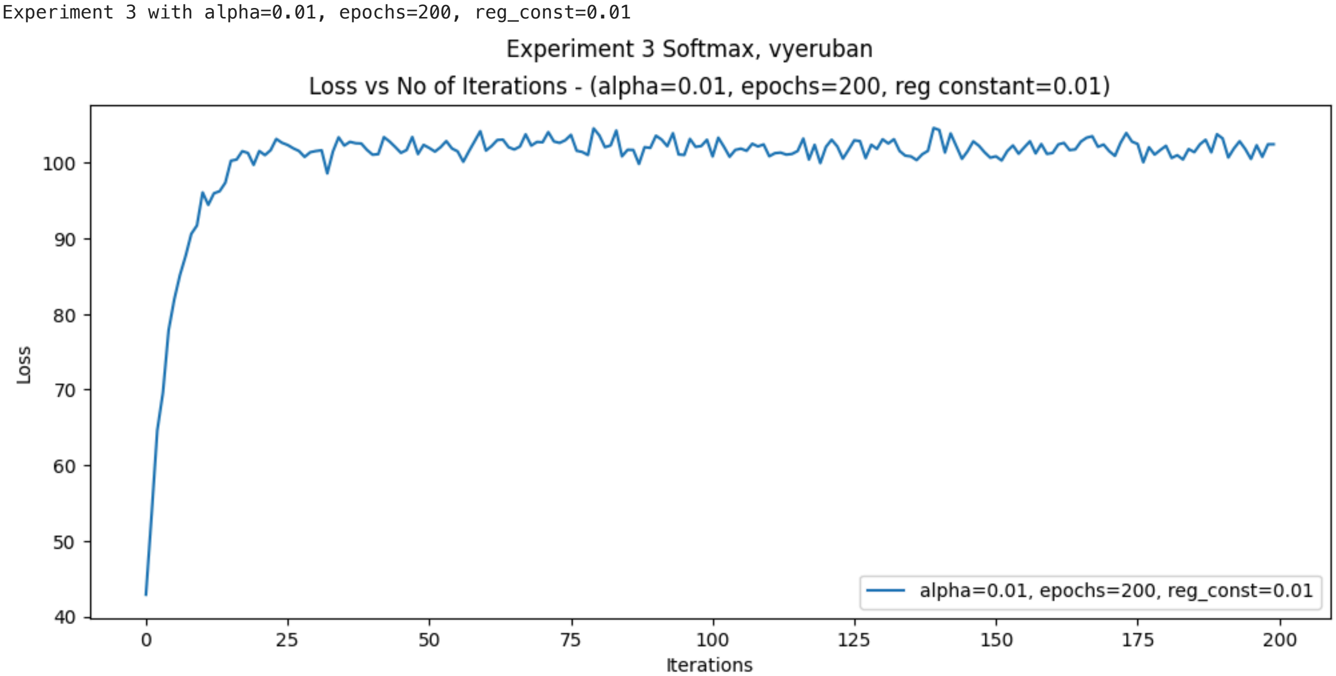

```
------------------------------------
Training Accuracy: 24.259183673469387
Validation Accuracy: 23.599999999999998
Test Accuracy:23.3
------------------------------------
Experiment 2 with alpha=0.01, epochs=100, reg_const=0.05
```

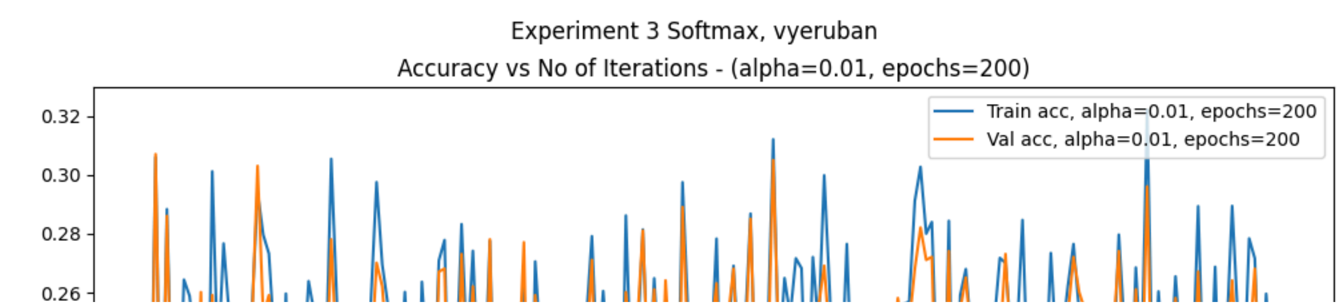


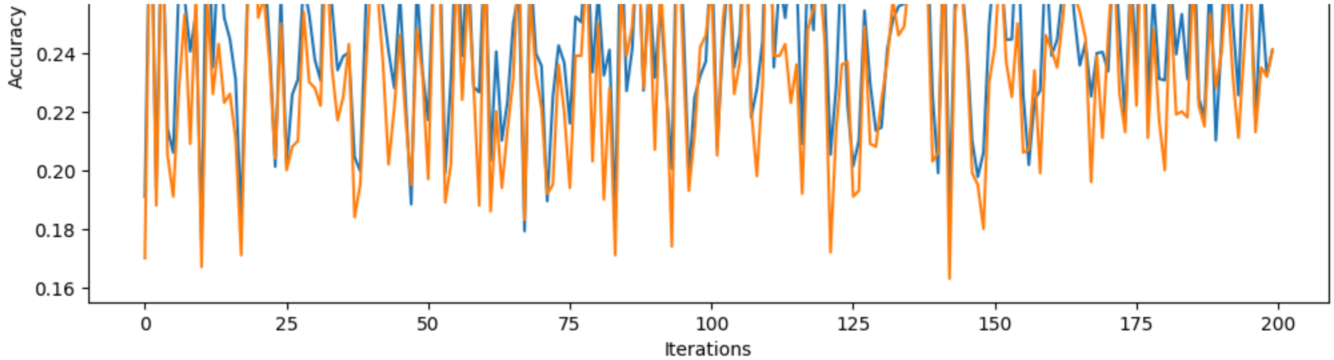




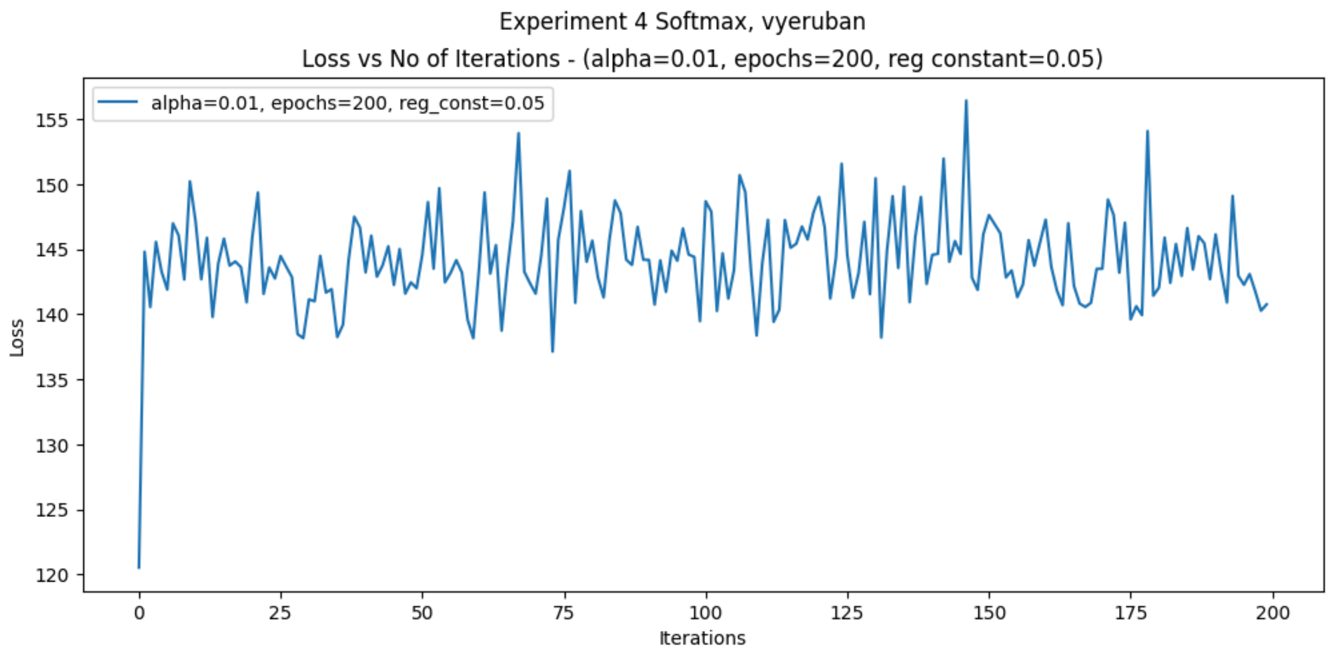

```
-----------------------------------
Training Accuracy: 23.981632653061226
Validation Accuracy: 26.0
Test Accuracy:23.7
-----------------------------------
Experiment 3 with alpha=0.01, epochs=200, reg_const=0.01
```

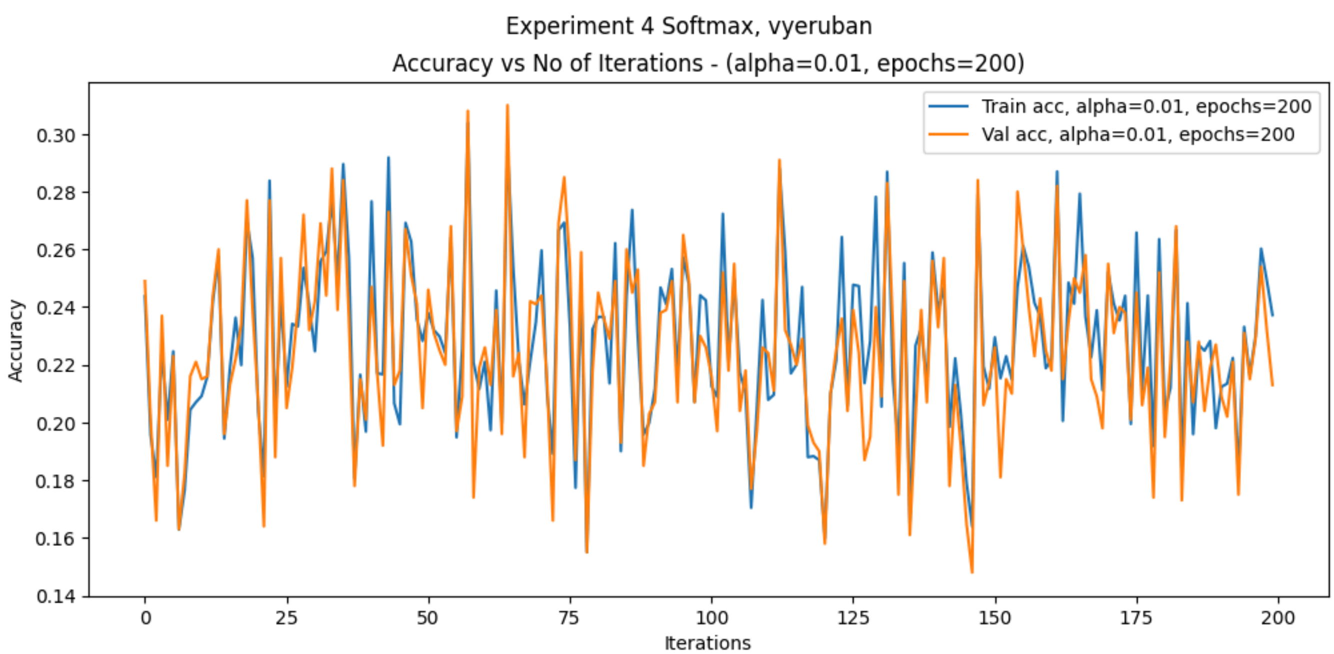






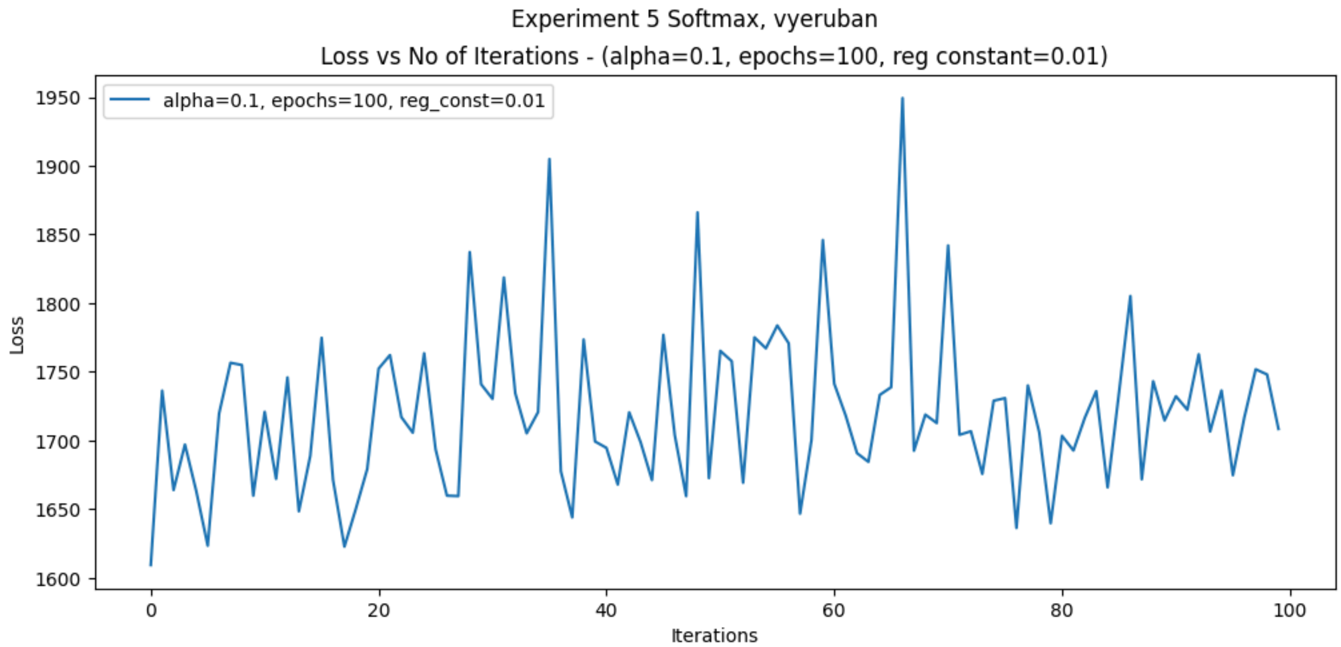

```
-----------------------------------
Training Accuracy: 24.10408163265306
Validation Accuracy: 24.099999999999998
Test Accuracy:22.48
-----------------------------------
Experiment 4 with alpha=0.01, epochs=200, reg_const=0.05
```

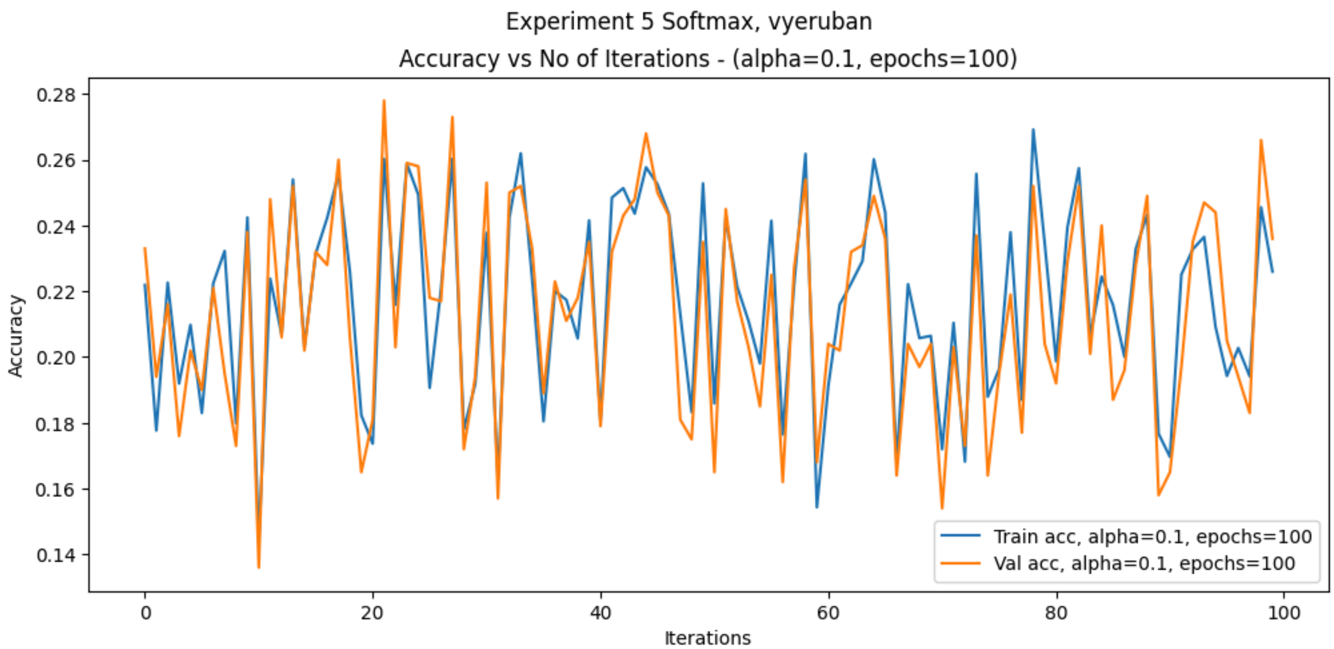




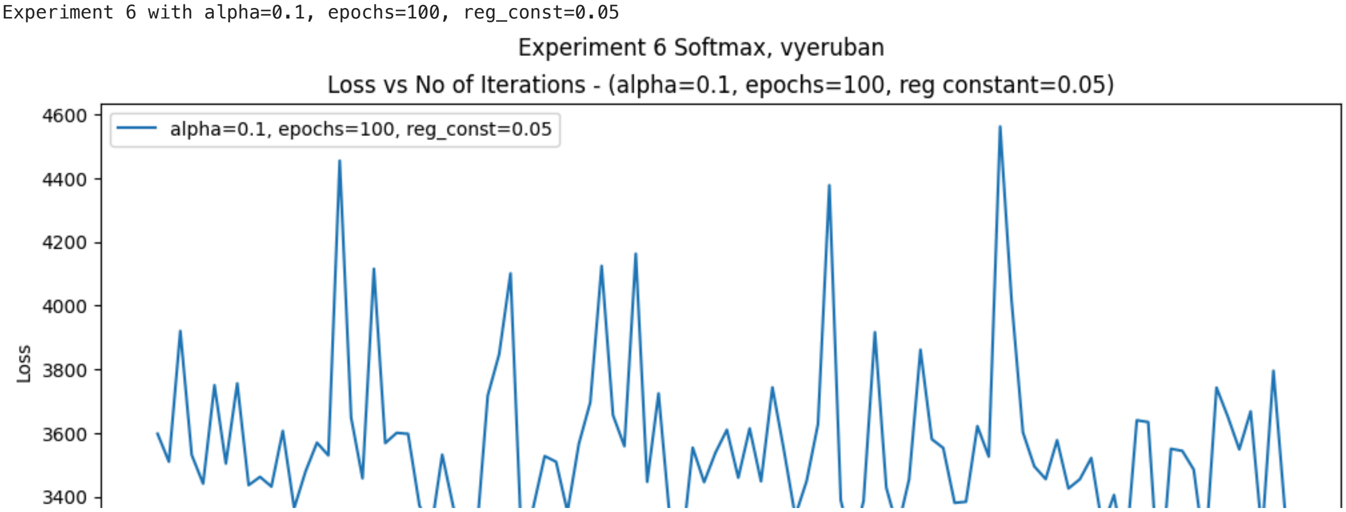

```
-----------------------------------
Training Accuracy: 23.72448979591837
Validation Accuracy: 21.3
Test Accuracy:22.54
-----------------------------------
Experiment 5 with alpha=0.1, epochs=100, reg_const=0.01
```

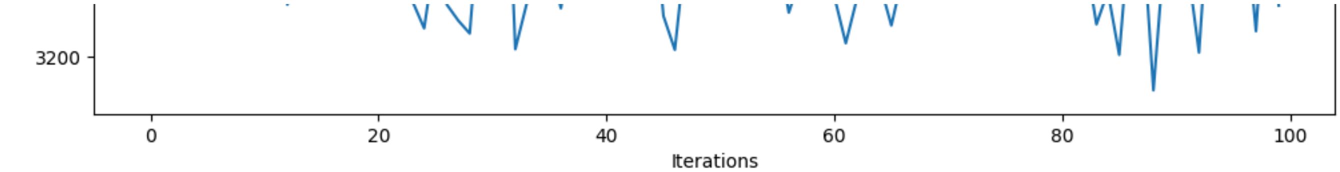
```
Experiment 5 with alpha=0.1, epochs=100, reg_const=0.01
```

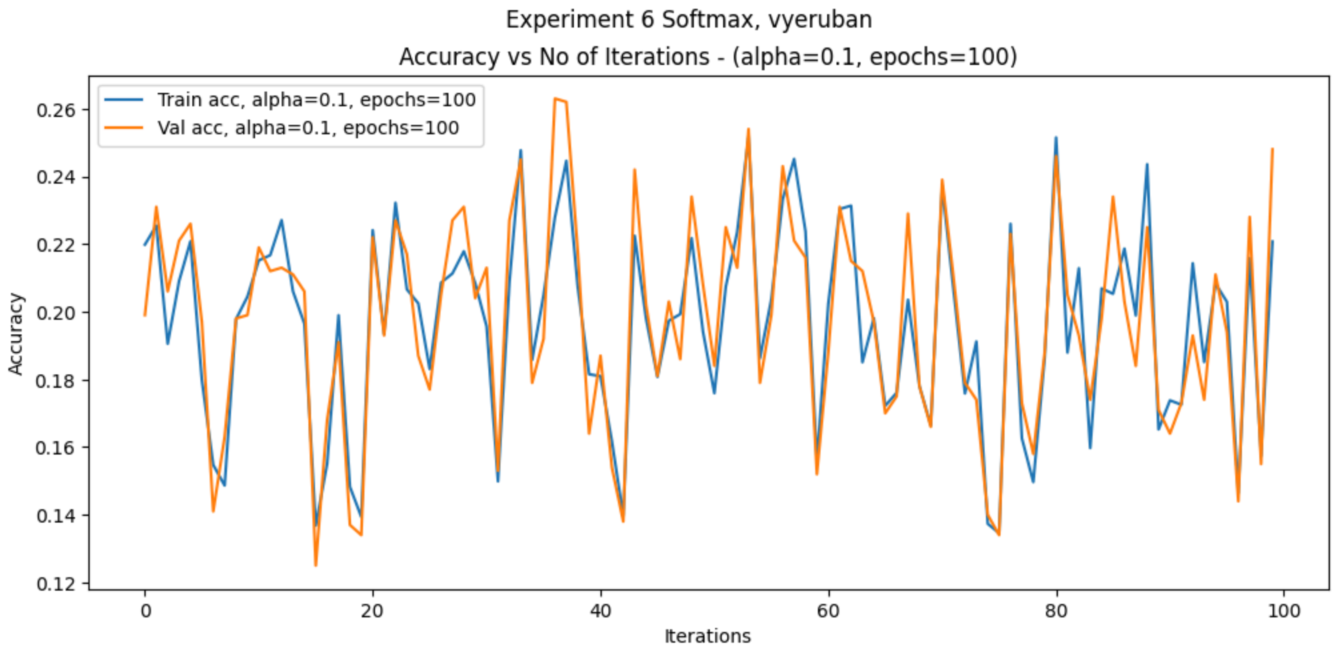




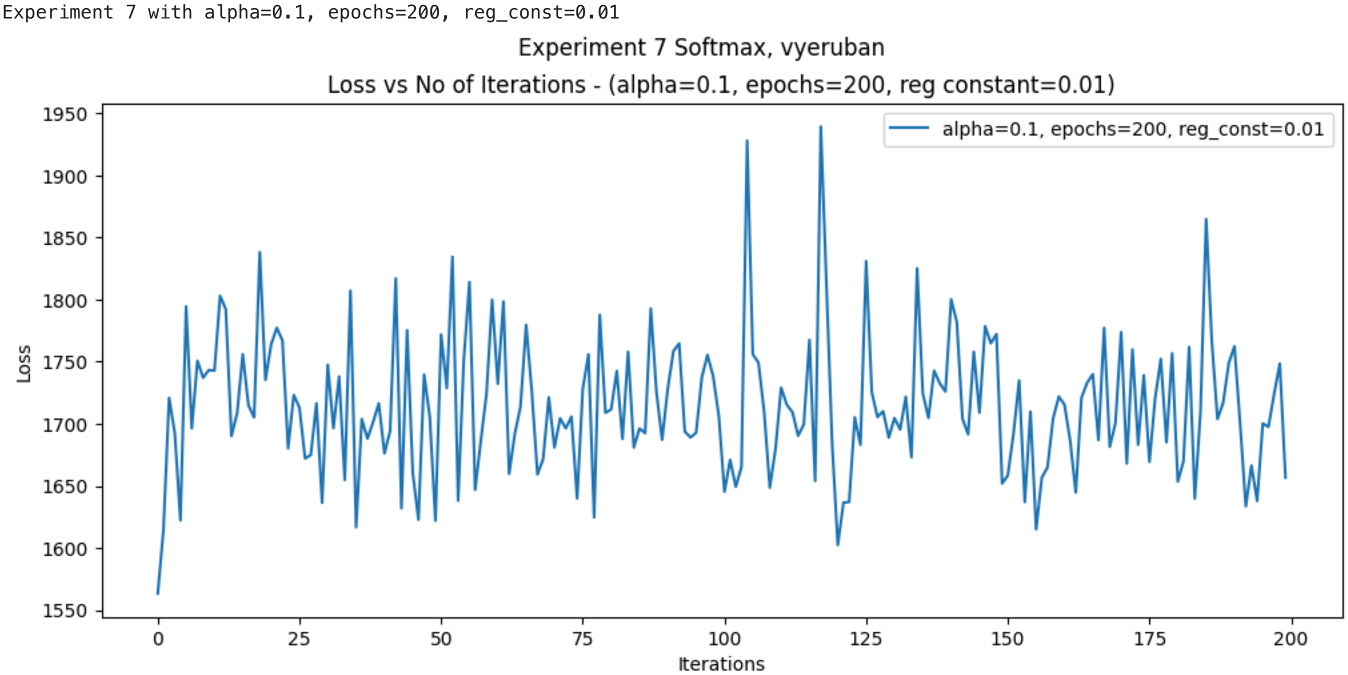

```
------------------------------------
Training Accuracy: 22.6
Validation Accuracy: 23.599999999999998
Test Accuracy:23.02
------------------------------------
Experiment 6 with alpha=0.1, epochs=100, reg_const=0.05
```

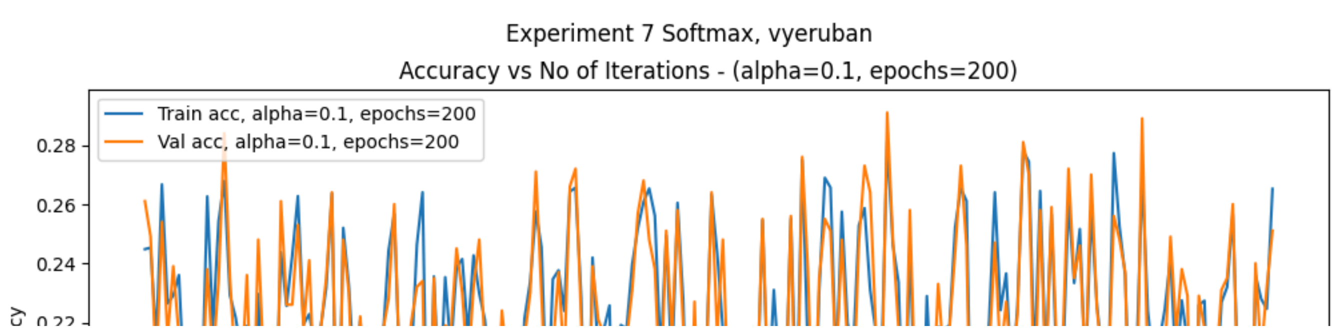


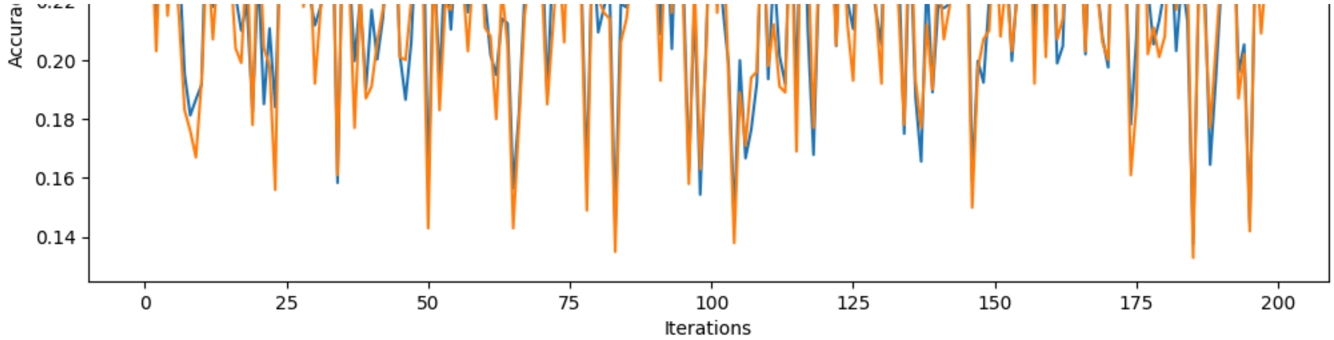





```
-----------------------------------
Training Accuracy: 22.0734693877551
Validation Accuracy: 24.8
Test Accuracy:21.6
-----------------------------------
Experiment 7 with alpha=0.1, epochs=200, reg_const=0.01
```








```
----------------------------------
Training Accuracy: 26.518367346938774
Validation Accuracy: 25.1
Test Accuracy:25.319999999999997
----------------------------------
Experiment 8 with alpha=0.1, epochs=200, reg_const=0.05
```