Assignment 2 :

Part 1: Neural Network optimization with SGD and Adam

```
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf

from sklearn.feature_extraction.text import CountVectorizer

(x_train, y_train_bow), (x_test, y_test_bow) = 
tf.keras.datasets.imdb.load_data(
    path='imdb.npz',
    num_words=10000,
    skip_top=0,
    maxlen=None,
    seed=113,
    start_char=1,
    oov_char=2,
    index_from=3,
)

print(x_train.shape, y_train_bow.shape, x_test.shape, 
y_test_bow.shape)
```

```
(25000,) (25000,) (25000,) (25000,)
```

Create the BoW feature vectors (10 points)

Create the word vectors using Bag of Words (BoW) representation. You can use the following code to get the BoW representation of the dataset. You can read more about BoW here

```
vectorizer = CountVectorizer(max_features=10000)

word_index = tf.keras.datasets.imdb.get_word_index()
reverse_word_index = dict([(value, key) for (key, value) in 
word_index.items()])

# x_train_text = [' '.join([str(word) for word in x]) for x in 
x_train]
x_train_text =  [' '.join([reverse_word_index.get(i - 3, '?') for i in 
sequence]) for sequence in x_train]
x_train_bow = vectorizer.fit_transform(x_train_text).astype('float32')
```

```python
#.toarray()#.astype('float32')
# x_train_bow = np.array(x_train_bow)

features = vectorizer.get_feature_names_out()
print(features)
```

```
['00' '000' '10' ... 'zoom' 'zorro' 'zu']
```

```python
# print(x_train_bow.shape, x_train_bow, type(x_train_bow))
print(x_train_bow)
```

```
  (0, 230)	1.0
  (0, 312)	1.0
  (0, 387)	3.0
  (0, 412)	1.0
  (0, 429)	3.0
  (0, 456)	1.0
  (0, 463)	9.0
  (0, 583)	3.0
  (0, 632)	3.0
  (0, 676)	2.0
  (0, 868)	1.0
  (0, 888)	2.0
  (0, 896)	1.0
  (0, 919)	2.0
  (0, 979)	1.0
  (0, 1132)	1.0
  (0, 1144)	1.0
  (0, 1201)	3.0
  (0, 1298)	1.0
  (0, 1332)	1.0
  (0, 1440)	1.0
  (0, 1593)	2.0
  (0, 1936)	1.0
  (0, 2070)	1.0
  (0, 2152)	1.0
  :	:
  (24999, 7513)	1.0
  (24999, 7742)	1.0
  (24999, 7744)	1.0
  (24999, 7753)	1.0
  (24999, 7758)	1.0
  (24999, 7875)	1.0
  (24999, 8055)	1.0
  (24999, 8066)	1.0
  (24999, 8338)	1.0
  (24999, 8443)	2.0
  (24999, 8713)	1.0
  (24999, 8763)	10.0
  (24999, 8783)	1.0
```

```
  (24999, 8805)	2.0
  (24999, 8886)	4.0
  (24999, 9035)	1.0
  (24999, 9081)	1.0
  (24999, 9388)	1.0
  (24999, 9431)	1.0
  (24999, 9446)	1.0
  (24999, 9463)	1.0
  (24999, 9472)	1.0
  (24999, 9541)	1.0
  (24999, 9632)	1.0
  (24999, 9668)	1.0
```

```python
# x_test_text = [' '.join([str(word) for word in x]) for x in x_test]
x_test_text =  [' '.join([reverse_word_index.get(i - 3, '?') for i in
sequence]) for sequence in x_test]
x_test_bow = vectorizer.fit_transform(x_test_text).astype('float32')
# x_test_bow = np.array(x_test_bow)
# x_test_bow =
vectorizer.transform(x_test_text).toarray().astype('float32')

features = vectorizer.get_feature_names_out()
print(features)
```

```
['00' '000' '10' ... 'zoom' 'zorro' 'zu']
```

```python
print(x_test_bow.shape, x_test_bow)
```

```
(25000, 9725)   (0, 387)	1.0
  (0, 400)	1.0
  (0, 405)	1.0
  (0, 462)	2.0
  (0, 1142)	4.0
  (0, 1428)	1.0
  (0, 2059)	1.0
  (0, 2672)	1.0
  (0, 3289)	1.0
  (0, 3464)	3.0
  (0, 3792)	2.0
  (0, 3832)	1.0
  (0, 4073)	1.0
  (0, 4085)	1.0
  (0, 4191)	2.0
  (0, 4292)	1.0
  (0, 4668)	1.0
  (0, 4922)	2.0
  (0, 5312)	2.0
  (0, 5570)	1.0
  (0, 5640)	2.0
  (0, 6053)	1.0
```

```
  (0, 6093)	1.0
  (0, 6095)	2.0
  (0, 6163)	1.0
  :	:
  (24999, 7674)	1.0
  (24999, 7786)	1.0
  (24999, 7821)	2.0
  (24999, 7825)	1.0
  (24999, 7905)	1.0
  (24999, 7996)	1.0
  (24999, 8320)	1.0
  (24999, 8328)	1.0
  (24999, 8329)	1.0
  (24999, 8717)	1.0
  (24999, 8722)	1.0
  (24999, 8724)	8.0
  (24999, 8734)	1.0
  (24999, 8744)	2.0
  (24999, 8766)	2.0
  (24999, 8846)	2.0
  (24999, 8882)	1.0
  (24999, 8905)	1.0
  (24999, 9185)	1.0
  (24999, 9223)	1.0
  (24999, 9275)	1.0
  (24999, 9421)	1.0
  (24999, 9444)	1.0
  (24999, 9587)	4.0
  (24999, 9696)	1.0
```

```
y_train_bow = y_train_bow.astype('float32')
y_test_bow = y_test_bow.astype('float32')

# y_train_bow = y_train_bow.astype('float32')
# y_test_bow = y_test_bow.astype('float32')
print(y_test_bow, y_train_bow)
```

```
[0. 1. 1. ... 0. 0. 0.] [1. 0. 0. ... 0. 1. 0.]
```

Therefore Bag Of Words (BoW) vector created on the input data set - x_train and x_test

Implement the models (10 points)

You need to implement Logistic Regression, MLP and CNN models.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense, Dropout, Conv2D, MaxPooling2D
from keras.optimizers import Adam, SGD
from keras.regularizers import l2

from keras import backend as K

from keras.datasets import mnist, cifar10
```

Logistic Regression

Logistic Regression for Multi-class for MNIST dataset

```
# function to build and train a logistic regression model
def build_logistic_regression_model(optimizer_name, trial,
learning_rate,
                                    beta_1, beta_2 ):

    model = Sequential()
    model.add(Flatten(input_shape=(784,)))
    model.add(Dense(10, activation='softmax',
kernel_regularizer=l2(1e-6)))

    if optimizer_name == 'adam':
        # optimizer = Adam(learning_rate=learning_rate, beta_1=0.9,
beta_2=0.999, epsilon=1e-8)
        optimizer = Adam(learning_rate=learning_rate, beta_1=beta_1,
beta_2=beta_2, epsilon=1e-8)
    elif optimizer_name == 'sgd':
        # optimizer = SGD(learning_rate=0.001/np.sqrt(45),
momentum=0.9, nesterov=True)
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9,
nesterov=True)
    else:
        raise ValueError("Invalid optimizer name")

    model.compile(optimizer=optimizer,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model
```

Logistic Regression for Binary classes for IMDB review

```
# Create a function to build and train a logistic regression model
def build_logistic_regression_bin_model(optimizer_name, trial,
learning_rate,
                                        beta_1, beta_2):
    model = Sequential()
    model.add(Flatten(input_shape=x_train_bow.shape[1:]))
    model.add(Dropout(0.5))
```

```
    model.add(Dense(1, activation='sigmoid'))

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=learning_rate, beta_1=beta_1,
beta_2=beta_2, epsilon=1e-8)
    elif optimizer_name == 'sgd':
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9,
nesterov=True)
    else:
        raise ValueError("Invalid optimizer name")

    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

    return model
```

Multi-layer Perceptron (MLP)

```
# Create a function to build and train a MLP model
def build_multi_layer_perceptrons_model(optimizer_name, trial,
learning_rate,
                                        beta_1, beta_2, l2_decay):
    # Create an MLP model
    mlp_model = Sequential()
    # mlp_model.add(input(shape=[28, 28]))
    mlp_model.add(Flatten(input_shape=(X_train.shape[1],)))
    # mlp_model.add(Dropout(0.5))
    mlp_model.add(Dense(1000, activation="relu",
kernel_regularizer=l2(l2_decay)))
    mlp_model.add(Dropout(0.5))
    mlp_model.add(Dense(1000, activation="relu",
kernel_regularizer=l2(l2_decay)))
    mlp_model.add(Dropout(0.5))
    mlp_model.add(Dense(10, activation="softmax"))

    # Create a custom learning rate schedule that decreases as
1/sqrt(t)
    # initial_learning_rate = 0.001
    # # decay_steps = len(X_train) // 128  # Adjust as needed
    # decay_steps = epochs
    # lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
    #     initial_learning_rate, decay_steps=decay_steps,
decay_rate=1/np.sqrt(epochs), staircase=False
    # )

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=learning_rate, beta_1=beta_1,
beta_2=beta_2, epsilon=1e-8)
    elif optimizer_name == 'sgd':
```

```
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9,
nesterov=True)
    else:
        raise ValueError("Invalid optimizer name")

    # Compile the model
    mlp_model.compile(optimizer=optimizer,
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])

    # Evaluate the model
    mlp_loss, mlp_accuracy = mlp_model.evaluate(X_test, y_test)
    print(f"MLP - Test Loss: {mlp_loss}, Test Accuracy:
{mlp_accuracy}")

    return mlp_model
```

Convolutional Neural Network (CNN)

```
# Create CNN model
def build_cnn_model(optimizer_name, trial, learning_rate,
                          beta_1, beta_2, l2_reg):
    # Define the CNN architecture
    cnn_model = Sequential([
        Conv2D(32, (5, 5), activation='relu', input_shape=(32, 32, 3),
padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Conv2D(64, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Conv2D(128, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Flatten(),
        Dense(1000, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')  # 10 classes in CIFAR-10
    ])

    if optimizer_name == 'adam':
        # optimizer = Adam(learning_rate=0.01, beta_1=0.9,
beta_2=0.999, epsilon=1e-8)
        optimizer = Adam(learning_rate=learning_rate, beta_1=beta_1,
beta_2=beta_2, epsilon=1e-8)
    elif optimizer_name == 'sgd':
        # optimizer = SGD(learning_rate=0.0001, momentum=0.9,
nesterov=True)
        optimizer = SGD(learning_rate=learning_rate, momentum=0.9,
nesterov=True)
    else:
        raise ValueError("Invalid optimizer name")
```

```
    cnn_model.compile(optimizer=optimizer, 
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return cnn_model
```

Use SGD and Adam optimizers with Optuna to find the best hyperparameters for the models. (20)

```
!pip install --quiet optuna
```

```
zsh:1: command not found: pip
```

```
import optuna
import math
from sklearn.metrics import accuracy_score
from keras.callbacks import Callback

optuna.__version__
```

```
/Users/banani/Library/Python/3.9/lib/python/site-packages/tqdm/
auto.py:21: TqdmWarning: IProgress not found. Please update jupyter
and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

```
'3.3.0'
```

MNIST Dataset

Load MNIST data

```
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Preprocess the data
X_train = X_train / 255.0
X_test = X_test / 255.0

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

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-
keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

Log_Reg for MNIST

```python
# Define a custom callback to track validation loss
class ValidationLoss(Callback):
    def __init__(self):
        super(ValidationLoss, self).__init__()
        self.validation_losses = []

    def on_epoch_end(self, epoch, logs=None):
        self.validation_losses.append(logs['val_loss'])

# Custom learning rate suggestion function with rate decay
def custom_learning_rate_schedule(epoch):
    t = epoch  # Use trial number as 't' or replace it with epoch
number
    lr = 0.001 / math.sqrt(t + 1)  # Initial LR 0.1, rate decay
1/sqrt(t+1) to avoid division by zero
    return lr

def objective(trial):

  epochs = 200
  # Create and compile the logistic regression model
  optimizer_name = trial.suggest_categorical('optimizer', ['adam',
'sgd'])
  learning_rate = trial.suggest_float('learning_rate', 1e-3, 1e-2,
log=True)
  # learning_rate = custom_learning_rate_schedule(trial)
  # dropout_rate = trial.suggest_float('dropout_rate', 0.0, 0.5)
  beta_1 = trial.suggest_float('beta_1', 0.0, 0.9)  # Vary beta_1
within [0, 0.9]
  beta_2 = trial.suggest_float('beta_2', 0.99, 0.9999)  # Vary beta_2
within [0.99, 0.9999]
  decay = trial.suggest_float('decay', 1e-6, 1e-2, log=True)
  decay_steps = trial.suggest_int('decay_steps', 1, len(X_train) //
128)
  # epsilon = trial.suggest_float('epsilon', 1e-9, 1e-7)
  epsilon = 1e-8
  # momentum = trial.suggest_float('momentum', 0.9, 0.99)
  momentum = 0.9
  l2_reg = trial.suggest_float('l2_reg', 1e-6, 1e-3, log=True)  # L2
regularization strength

  # Implement the learning rate schedule for Adam
  # def learning_rate_schedule(epoch, lr):
      # t = epoch + 1  # Current epoch
```

```
      # return initial_learning_rate / math.sqrt(t)

  # decay_steps = len(X_train) // 128  # Adjust as needed
  # decay_steps = epochs  # Adjust as needed
  # lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
  #       learning_rate, decay_steps=decay_steps,
decay_rate=1/np.sqrt(45.0), staircase=False
  # )

  # Define the custom learning rate scheduler
  class CustomLRScheduler(tf.keras.callbacks.Callback):
      def on_epoch_begin(self, epoch, logs=None):
          new_lr = 0.001 / np.sqrt(epoch+1)
          K.set_value(self.model.optimizer.lr, new_lr)
          print(f'Epoch {epoch + 1}: Learning Rate = {new_lr}')

  model = build_logistic_regression_model(optimizer_name, trial,
learning_rate,
                                          beta_1, beta_2)
  # model = build_logistic_regression_model(optimizer_name, trial,
learning_rate,
  #                                       beta_1, beta_2)


  # Create a custom callback to track validation loss
  val_loss_callback = ValidationLoss()
  lr_scheduler = CustomLRScheduler()

  # Train the model
  history = model.fit(X_train, y_train, epochs=epochs, batch_size=128,
                      validation_data=(X_test, y_test), verbose=0,
                      callbacks=[val_loss_callback, lr_scheduler])

  # y_pred = model.predict_classes(X_test)
  # accuracy = accuracy_score(y_test, y_pred)
  # evaluation_score = model.evaluate(x_valid, y_valid, verbose=0)

  # Get the minimum validation loss
  min_val_loss = min(val_loss_callback.validation_losses)

  return min_val_loss

# Create an Optuna study
study = optuna.create_study(direction='minimize')

# Optimize hyperparameters
# study.optimize(objective, n_trials=45)
study.optimize(objective, n_trials=1)

# Get the best trial and hyperparameters
best_trial = study.best_trial
```

```python
best_optimizer = best_trial.params['optimizer']
best_lr = best_trial.params['learning_rate']
best_beta_1 = best_trial.params['beta_1']
best_beta_2 = best_trial.params['beta_2']

# Print the best hyperparameters
print(f'Best Optimizer: {best_optimizer}')
print(f'Best Learning Rate: {best_lr}')
```

```
[I 2023-10-07 16:07:44,227] A new study created in memory with name:
no-name-e8101cda-534c-43cf-944f-5f9caf381059
```

```
Epoch 1: Learning Rate = 1.0
Epoch 2: Learning Rate = 0.7071067811865475
Epoch 3: Learning Rate = 0.5773502691896258
Epoch 4: Learning Rate = 0.5
Epoch 5: Learning Rate = 0.4472135954999579
Epoch 6: Learning Rate = 0.4082482904638631
Epoch 7: Learning Rate = 0.3779644730092272
Epoch 8: Learning Rate = 0.35355339059327373
Epoch 9: Learning Rate = 0.3333333333333333
Epoch 10: Learning Rate = 0.31622776601683794
Epoch 11: Learning Rate = 0.30151134457776363
Epoch 12: Learning Rate = 0.2886751345948129
Epoch 13: Learning Rate = 0.2773500981126146
Epoch 14: Learning Rate = 0.2672612419124244
Epoch 15: Learning Rate = 0.2581988897471611
Epoch 16: Learning Rate = 0.25
Epoch 17: Learning Rate = 0.24253562503633297
Epoch 18: Learning Rate = 0.23570226039551587
Epoch 19: Learning Rate = 0.22941573387056174
Epoch 20: Learning Rate = 0.22360679774997896
Epoch 21: Learning Rate = 0.2182178902359924
Epoch 22: Learning Rate = 0.21320071635561041
Epoch 23: Learning Rate = 0.20851441405707477
Epoch 24: Learning Rate = 0.20412414523193154
Epoch 25: Learning Rate = 0.2
Epoch 26: Learning Rate = 0.19611613513818404
Epoch 27: Learning Rate = 0.19245008972987526
Epoch 28: Learning Rate = 0.1889822365046136
Epoch 29: Learning Rate = 0.18569533817705186
Epoch 30: Learning Rate = 0.18257418583505536
Epoch 31: Learning Rate = 0.1796053020267749
Epoch 32: Learning Rate = 0.17677669529663687
Epoch 33: Learning Rate = 0.17407765595569785
Epoch 34: Learning Rate = 0.17149858514250882
Epoch 35: Learning Rate = 0.1690308509457033
Epoch 36: Learning Rate = 0.16666666666666666
Epoch 37: Learning Rate = 0.1643989873053573
Epoch 38: Learning Rate = 0.16222142113076254
```

```
Epoch 39: Learning Rate = 0.16012815380508713
Epoch 40: Learning Rate = 0.15811388300841897
Epoch 41: Learning Rate = 0.15617376188860607
Epoch 42: Learning Rate = 0.1543033499620919
Epoch 43: Learning Rate = 0.15249857033260467
Epoch 44: Learning Rate = 0.15075567228888181
Epoch 45: Learning Rate = 0.14907119849998599
Epoch 46: Learning Rate = 0.14744195615489714
Epoch 47: Learning Rate = 0.14586499149789456
Epoch 48: Learning Rate = 0.14433756729740646
Epoch 49: Learning Rate = 0.14285714285714285
Epoch 50: Learning Rate = 0.1414213562373095
Epoch 51: Learning Rate = 0.14002800840280097
Epoch 52: Learning Rate = 0.1386750490563073
Epoch 53: Learning Rate = 0.13736056394868904
Epoch 54: Learning Rate = 0.13608276348795434
Epoch 55: Learning Rate = 0.13483997249264842
Epoch 56: Learning Rate = 0.1336306209562122
Epoch 57: Learning Rate = 0.13245323570650439
Epoch 58: Learning Rate = 0.13130643285972254
Epoch 59: Learning Rate = 0.13018891098082389
Epoch 60: Learning Rate = 0.12909944487358055
Epoch 61: Learning Rate = 0.12803687993289598
Epoch 62: Learning Rate = 0.1270001270001905
Epoch 63: Learning Rate = 0.1259881576697424
Epoch 64: Learning Rate = 0.125
Epoch 65: Learning Rate = 0.12403473458920847
Epoch 66: Learning Rate = 0.12309149097933272
Epoch 67: Learning Rate = 0.12216944435630522
Epoch 68: Learning Rate = 0.12126781251816648
Epoch 69: Learning Rate = 0.1203858530857692
Epoch 70: Learning Rate = 0.11952286093343936
Epoch 71: Learning Rate = 0.11867816581938533
Epoch 72: Learning Rate = 0.11785113019775793
Epoch 73: Learning Rate = 0.11704114719613057
Epoch 74: Learning Rate = 0.11624763874381928
Epoch 75: Learning Rate = 0.11547005383792514
Epoch 76: Learning Rate = 0.11470786693528087
Epoch 77: Learning Rate = 0.11396057645963795
Epoch 78: Learning Rate = 0.11322770341445956
Epoch 79: Learning Rate = 0.1125087900926024
Epoch 80: Learning Rate = 0.11180339887498948
Epoch 81: Learning Rate = 0.1111111111111111
Epoch 82: Learning Rate = 0.11043152607484653
Epoch 83: Learning Rate = 0.10976425998969035
Epoch 84: Learning Rate = 0.1091089451179962
Epoch 85: Learning Rate = 0.10846522890932808
Epoch 86: Learning Rate = 0.10783277320343841
Epoch 87: Learning Rate = 0.10721125348377948
```

```
Epoch 88: Learning Rate = 0.10660035817780521
Epoch 89: Learning Rate = 0.105999788000636
Epoch 90: Learning Rate = 0.10540925533894598
Epoch 91: Learning Rate = 0.10482848367219183
Epoch 92: Learning Rate = 0.10425720702853739
Epoch 93: Learning Rate = 0.10369516947304253
Epoch 94: Learning Rate = 0.10314212462587934
Epoch 95: Learning Rate = 0.10259783520851541
Epoch 96: Learning Rate = 0.10206207261596577
Epoch 97: Learning Rate = 0.10153461651336192
Epoch 98: Learning Rate = 0.10101525445522107
Epoch 99: Learning Rate = 0.10050378152592121
Epoch 100: Learning Rate = 0.1
Epoch 101: Learning Rate = 0.09950371902099892
Epoch 102: Learning Rate = 0.09901475429766744
Epoch 103: Learning Rate = 0.09853292781642932
Epoch 104: Learning Rate = 0.09805806756909202
Epoch 105: Learning Rate = 0.09759000729485333
Epoch 106: Learning Rate = 0.09712858623572641
Epoch 107: Learning Rate = 0.09667364890456635
Epoch 108: Learning Rate = 0.09622504486493763
Epoch 109: Learning Rate = 0.09578262852211514
Epoch 110: Learning Rate = 0.09534625892455924
Epoch 111: Learning Rate = 0.0949157995752499
Epoch 112: Learning Rate = 0.0944911182523068
Epoch 113: Learning Rate = 0.09407208683835973
Epoch 114: Learning Rate = 0.0936585811581694
Epoch 115: Learning Rate = 0.09325048082403138
Epoch 116: Learning Rate = 0.09284766908852593
Epoch 117: Learning Rate = 0.09245003270420485
Epoch 118: Learning Rate = 0.09205746178983235
Epoch 119: Learning Rate = 0.09166984970282113
Epoch 120: Learning Rate = 0.09128709291752768
Epoch 121: Learning Rate = 0.09090909090909091
Epoch 122: Learning Rate = 0.09053574604251853
Epoch 123: Learning Rate = 0.09016696346674323
Epoch 124: Learning Rate = 0.08980265101338746
Epoch 125: Learning Rate = 0.08944271909999159
Epoch 126: Learning Rate = 0.0890870806374748
Epoch 127: Learning Rate = 0.08873565094161139
Epoch 128: Learning Rate = 0.08838834764831843
Epoch 129: Learning Rate = 0.08804509063256238
Epoch 130: Learning Rate = 0.08770580193070293
Epoch 131: Learning Rate = 0.0873704056661038
Epoch 132: Learning Rate = 0.08703882797784893
Epoch 133: Learning Rate = 0.086710996952412
Epoch 134: Learning Rate = 0.08638684255813601
Epoch 135: Learning Rate = 0.08606629658238704
Epoch 136: Learning Rate = 0.08574929257125441
```

```
Epoch 137: Learning Rate = 0.0854357657716761
Epoch 138: Learning Rate = 0.08512565307587486
Epoch 139: Learning Rate = 0.08481889296799709
Epoch 140: Learning Rate = 0.08451542547285165
Epoch 141: Learning Rate = 0.0842151921066519
Epoch 142: Learning Rate = 0.08391813582966891
Epoch 143: Learning Rate = 0.08362420100070908
Epoch 144: Learning Rate = 0.08333333333333333
Epoch 145: Learning Rate = 0.08304547985373997
Epoch 146: Learning Rate = 0.0827605888602368
Epoch 147: Learning Rate = 0.08247860988423225
Epoch 148: Learning Rate = 0.08219949365267865
Epoch 149: Learning Rate = 0.08192319205190406
Epoch 150: Learning Rate = 0.08164965809277261
Epoch 151: Learning Rate = 0.08137884587711594
Epoch 152: Learning Rate = 0.08111071056538127
Epoch 153: Learning Rate = 0.08084520834544433
Epoch 154: Learning Rate = 0.08058229640253803
Epoch 155: Learning Rate = 0.08032193289024989
Epoch 156: Learning Rate = 0.08006407690254357
Epoch 157: Learning Rate = 0.07980868844676221
Epoch 158: Learning Rate = 0.079555728417573
Epoch 159: Learning Rate = 0.07930515857181442
Epoch 160: Learning Rate = 0.07905694150420949
Epoch 161: Learning Rate = 0.07881104062391006
Epoch 162: Learning Rate = 0.07856742013183861
Epoch 163: Learning Rate = 0.07832604499879574
Epoch 164: Learning Rate = 0.07808688094430304
Epoch 165: Learning Rate = 0.0778498944161523
Epoch 166: Learning Rate = 0.07761505257063328
Epoch 167: Learning Rate = 0.07738232325341368
Epoch 168: Learning Rate = 0.07715167498104596
Epoch 169: Learning Rate = 0.07692307692307693
Epoch 170: Learning Rate = 0.07669649888473704
Epoch 171: Learning Rate = 0.07647191129018725
Epoch 172: Learning Rate = 0.07624928516630233
Epoch 173: Learning Rate = 0.07602859212697055
Epoch 174: Learning Rate = 0.07580980435789034
Epoch 175: Learning Rate = 0.07559289460184544
Epoch 176: Learning Rate = 0.07537783614444091
Epoch 177: Learning Rate = 0.07516460280028289
Epoch 178: Learning Rate = 0.07495316889958614
Epoch 179: Learning Rate = 0.07474350927519359
Epoch 180: Learning Rate = 0.07453559924999299
Epoch 181: Learning Rate = 0.07432941462471664
Epoch 182: Learning Rate = 0.07412493166611012
Epoch 183: Learning Rate = 0.07392212709545729
Epoch 184: Learning Rate = 0.07372097807744857
Epoch 185: Learning Rate = 0.07352146220938077
```

```
Epoch 186: Learning Rate = 0.07332355751067665
Epoch 187: Learning Rate = 0.07312724241271307
Epoch 188: Learning Rate = 0.07293249574894728
Epoch 189: Learning Rate = 0.07273929674533079
Epoch 190: Learning Rate = 0.07254762501100116
Epoch 191: Learning Rate = 0.07235746052924216
Epoch 192: Learning Rate = 0.07216878364870323
Epoch 193: Learning Rate = 0.07198157507486945
Epoch 194: Learning Rate = 0.07179581586177382
Epoch 195: Learning Rate = 0.0716114874039433
Epoch 196: Learning Rate = 0.07142857142857142
Epoch 197: Learning Rate = 0.07124704998790965
Epoch 198: Learning Rate = 0.07106690545187015
Epoch 199: Learning Rate = 0.07088812050083358
Epoch 200: Learning Rate = 0.07071067811865475

[I 2023-10-07 16:12:10,739] Trial 0 finished with value:
0.28927814960479736 and parameters: {'optimizer': 'sgd',
'learning_rate': 0.0036132298044764073, 'beta_1': 0.26108547972311,
'beta_2': 0.9986691704338064, 'decay': 4.216458440951901e-06,
'decay_steps': 152, 'l2_reg': 3.6376854898462428e-06}. Best is trial 0
with value: 0.28927814960479736.

Best Optimizer: sgd
Best Learning Rate: 0.0036132298044764073

# Retrieve and print the best hyperparameters
best_hyperparameters = best_trial.params
print("Best Hyperparameters for logistic Regression on MNIST
dataset:")
for param_name, param_value in best_hyperparameters.items():
    print(f"{param_name}: {param_value}")

Best Hyperparameters for logistic Regression on MNIST dataset:
optimizer: sgd
learning_rate: 0.0036132298044764073
beta_1: 0.26108547972311
beta_2: 0.9986691704338064
decay: 4.216458440951901e-06
decay_steps: 152
l2_reg: 3.6376854898462428e-06

# Get the best trial and hyperparameters
best_trial = study.best_trial
best_optimizer = best_trial.params['optimizer']
best_lr = best_trial.params['learning_rate']
best_beta_1 = best_trial.params['beta_1']
best_beta_2 = best_trial.params['beta_2']

# Build and train the final model with the best hyperparameters
best_model = build_logistic_regression_model(best_optimizer,
```

```python
best_trial, best_lr,
                                             best_beta_1, best_beta_2)
history = best_model.fit(X_train, y_train, epochs=45, batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

# adam_history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
#                         validation_data=(X_test, y_test), verbose=1)
# sgd_history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
                        # validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/45
469/469 [==============================] - 2s 4ms/step - loss: 0.8113
- accuracy: 0.8033 - val_loss: 0.4917 - val_accuracy: 0.8805
Epoch 2/45
469/469 [==============================] - 1s 3ms/step - loss: 0.4655
- accuracy: 0.8795 - val_loss: 0.4085 - val_accuracy: 0.8921
Epoch 3/45
469/469 [==============================] - 1s 3ms/step - loss: 0.4107
- accuracy: 0.8899 - val_loss: 0.3740 - val_accuracy: 0.8996
Epoch 4/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3832
- accuracy: 0.8958 - val_loss: 0.3553 - val_accuracy: 0.9028
Epoch 5/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3659
- accuracy: 0.9000 - val_loss: 0.3419 - val_accuracy: 0.9056
Epoch 6/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3536
- accuracy: 0.9024 - val_loss: 0.3317 - val_accuracy: 0.9086
Epoch 7/45
469/469 [==============================] - 2s 4ms/step - loss: 0.3443
- accuracy: 0.9047 - val_loss: 0.3245 - val_accuracy: 0.9113
Epoch 8/45
469/469 [==============================] - 2s 4ms/step - loss: 0.3368
- accuracy: 0.9065 - val_loss: 0.3184 - val_accuracy: 0.9135
Epoch 9/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3306
- accuracy: 0.9084 - val_loss: 0.3142 - val_accuracy: 0.9149
Epoch 10/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3256
- accuracy: 0.9093 - val_loss: 0.3104 - val_accuracy: 0.9146
Epoch 11/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3212
- accuracy: 0.9105 - val_loss: 0.3065 - val_accuracy: 0.9162
Epoch 12/45
469/469 [==============================] - 2s 3ms/step - loss: 0.3174
- accuracy: 0.9113 - val_loss: 0.3044 - val_accuracy: 0.9160
Epoch 13/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3141
```

```
- accuracy: 0.9125 - val_loss: 0.3007 - val_accuracy: 0.9170
Epoch 14/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3110
- accuracy: 0.9138 - val_loss: 0.2985 - val_accuracy: 0.9173
Epoch 15/45
469/469 [==============================] - 2s 4ms/step - loss: 0.3083
- accuracy: 0.9143 - val_loss: 0.2978 - val_accuracy: 0.9170
Epoch 16/45
469/469 [==============================] - 2s 5ms/step - loss: 0.3059
- accuracy: 0.9147 - val_loss: 0.2953 - val_accuracy: 0.9178
Epoch 17/45
469/469 [==============================] - 2s 3ms/step - loss: 0.3036
- accuracy: 0.9152 - val_loss: 0.2935 - val_accuracy: 0.9187
Epoch 18/45
469/469 [==============================] - 1s 3ms/step - loss: 0.3017
- accuracy: 0.9161 - val_loss: 0.2915 - val_accuracy: 0.9181
Epoch 19/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2999
- accuracy: 0.9165 - val_loss: 0.2904 - val_accuracy: 0.9189
Epoch 20/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2982
- accuracy: 0.9172 - val_loss: 0.2891 - val_accuracy: 0.9189
Epoch 21/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2965
- accuracy: 0.9175 - val_loss: 0.2891 - val_accuracy: 0.9193
Epoch 22/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2951
- accuracy: 0.9179 - val_loss: 0.2876 - val_accuracy: 0.9201
Epoch 23/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2936
- accuracy: 0.9183 - val_loss: 0.2864 - val_accuracy: 0.9209
Epoch 24/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2924
- accuracy: 0.9185 - val_loss: 0.2851 - val_accuracy: 0.9195
Epoch 25/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2912
- accuracy: 0.9189 - val_loss: 0.2849 - val_accuracy: 0.9203
Epoch 26/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2899
- accuracy: 0.9191 - val_loss: 0.2843 - val_accuracy: 0.9211
Epoch 27/45
469/469 [==============================] - 3s 7ms/step - loss: 0.2888
- accuracy: 0.9197 - val_loss: 0.2830 - val_accuracy: 0.9215
Epoch 28/45
469/469 [==============================] - 3s 6ms/step - loss: 0.2877
- accuracy: 0.9198 - val_loss: 0.2830 - val_accuracy: 0.9208
Epoch 29/45
469/469 [==============================] - 4s 8ms/step - loss: 0.2868
- accuracy: 0.9199 - val_loss: 0.2821 - val_accuracy: 0.9201
```

```
Epoch 30/45
469/469 [==============================] - 3s 7ms/step - loss: 0.2859
- accuracy: 0.9205 - val_loss: 0.2811 - val_accuracy: 0.9218
Epoch 31/45
469/469 [==============================] - 3s 7ms/step - loss: 0.2850
- accuracy: 0.9207 - val_loss: 0.2808 - val_accuracy: 0.9213
Epoch 32/45
469/469 [==============================] - 3s 7ms/step - loss: 0.2840
- accuracy: 0.9209 - val_loss: 0.2800 - val_accuracy: 0.9201
Epoch 33/45
469/469 [==============================] - 4s 9ms/step - loss: 0.2833
- accuracy: 0.9211 - val_loss: 0.2796 - val_accuracy: 0.9213
Epoch 34/45
469/469 [==============================] - 5s 11ms/step - loss: 0.2825
- accuracy: 0.9215 - val_loss: 0.2793 - val_accuracy: 0.9216
Epoch 35/45
469/469 [==============================] - 3s 5ms/step - loss: 0.2818
- accuracy: 0.9218 - val_loss: 0.2788 - val_accuracy: 0.9217
Epoch 36/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2811
- accuracy: 0.9219 - val_loss: 0.2786 - val_accuracy: 0.9220
Epoch 37/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2804
- accuracy: 0.9220 - val_loss: 0.2778 - val_accuracy: 0.9222
Epoch 38/45
469/469 [==============================] - 3s 5ms/step - loss: 0.2796
- accuracy: 0.9227 - val_loss: 0.2777 - val_accuracy: 0.9219
Epoch 39/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2790
- accuracy: 0.9227 - val_loss: 0.2771 - val_accuracy: 0.9212
Epoch 40/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2785
- accuracy: 0.9226 - val_loss: 0.2766 - val_accuracy: 0.9224
Epoch 41/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2778
- accuracy: 0.9226 - val_loss: 0.2761 - val_accuracy: 0.9224
Epoch 42/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2773
- accuracy: 0.9230 - val_loss: 0.2759 - val_accuracy: 0.9220
Epoch 43/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2767
- accuracy: 0.9232 - val_loss: 0.2755 - val_accuracy: 0.9228
Epoch 44/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2762
- accuracy: 0.9232 - val_loss: 0.2758 - val_accuracy: 0.9221
Epoch 45/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2757
- accuracy: 0.9235 - val_loss: 0.2749 - val_accuracy: 0.9233
```

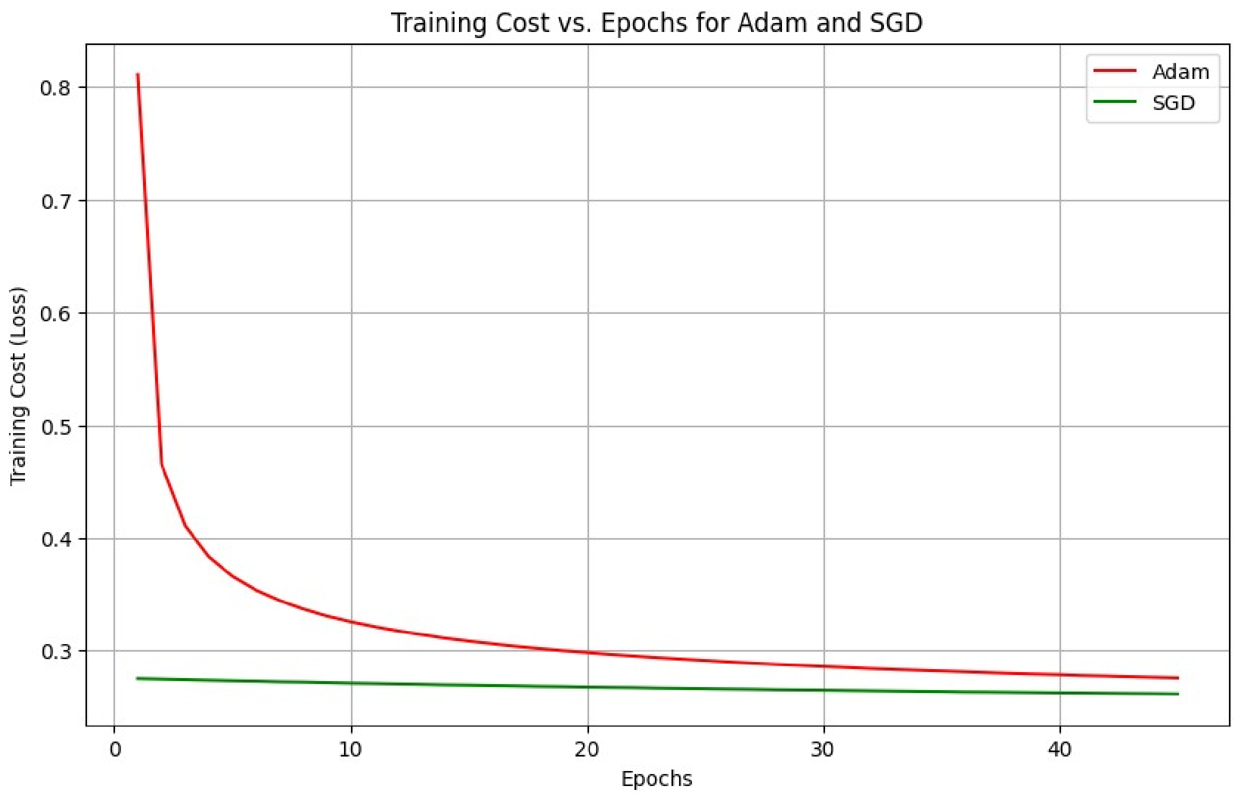
```
sgd_model = build_logistic_regression_model("adam", best_trial,
best_lr,
                                              best_beta_1, best_beta_2)

sgd_history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)
```

```
Epoch 1/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2751
- accuracy: 0.9238 - val_loss: 0.2753 - val_accuracy: 0.9218
Epoch 2/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2746
- accuracy: 0.9239 - val_loss: 0.2749 - val_accuracy: 0.9227
Epoch 3/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2742
- accuracy: 0.9241 - val_loss: 0.2751 - val_accuracy: 0.9216
Epoch 4/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2737
- accuracy: 0.9241 - val_loss: 0.2744 - val_accuracy: 0.9227
Epoch 5/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2732
- accuracy: 0.9243 - val_loss: 0.2743 - val_accuracy: 0.9233
Epoch 6/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2728
- accuracy: 0.9244 - val_loss: 0.2738 - val_accuracy: 0.9234
Epoch 7/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2723
- accuracy: 0.9247 - val_loss: 0.2737 - val_accuracy: 0.9213
Epoch 8/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2720
- accuracy: 0.9249 - val_loss: 0.2735 - val_accuracy: 0.9221
Epoch 9/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2715
- accuracy: 0.9252 - val_loss: 0.2741 - val_accuracy: 0.9218
Epoch 10/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2711
- accuracy: 0.9254 - val_loss: 0.2733 - val_accuracy: 0.9227
Epoch 11/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2708
- accuracy: 0.9252 - val_loss: 0.2734 - val_accuracy: 0.9228
Epoch 12/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2704
- accuracy: 0.9250 - val_loss: 0.2725 - val_accuracy: 0.9228
Epoch 13/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2700
- accuracy: 0.9252 - val_loss: 0.2727 - val_accuracy: 0.9235
Epoch 14/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2696
- accuracy: 0.9252 - val_loss: 0.2723 - val_accuracy: 0.9224
```

```
Epoch 15/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2693
- accuracy: 0.9259 - val_loss: 0.2721 - val_accuracy: 0.9224
Epoch 16/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2690
- accuracy: 0.9257 - val_loss: 0.2721 - val_accuracy: 0.9229
Epoch 17/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2687
- accuracy: 0.9259 - val_loss: 0.2714 - val_accuracy: 0.9229
Epoch 18/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2682
- accuracy: 0.9259 - val_loss: 0.2717 - val_accuracy: 0.9228
Epoch 19/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2680
- accuracy: 0.9262 - val_loss: 0.2719 - val_accuracy: 0.9224
Epoch 20/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2676
- accuracy: 0.9264 - val_loss: 0.2712 - val_accuracy: 0.9230
Epoch 21/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2673
- accuracy: 0.9262 - val_loss: 0.2716 - val_accuracy: 0.9234
Epoch 22/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2672
- accuracy: 0.9263 - val_loss: 0.2717 - val_accuracy: 0.9238
Epoch 23/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2667
- accuracy: 0.9263 - val_loss: 0.2710 - val_accuracy: 0.9231
Epoch 24/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2665
- accuracy: 0.9263 - val_loss: 0.2710 - val_accuracy: 0.9234
Epoch 25/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2662
- accuracy: 0.9265 - val_loss: 0.2701 - val_accuracy: 0.9233
Epoch 26/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2659
- accuracy: 0.9268 - val_loss: 0.2703 - val_accuracy: 0.9241
Epoch 27/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2657
- accuracy: 0.9264 - val_loss: 0.2707 - val_accuracy: 0.9234
Epoch 28/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2653
- accuracy: 0.9272 - val_loss: 0.2698 - val_accuracy: 0.9241
Epoch 29/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2651
- accuracy: 0.9270 - val_loss: 0.2711 - val_accuracy: 0.9242
Epoch 30/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2649
- accuracy: 0.9267 - val_loss: 0.2702 - val_accuracy: 0.9240
Epoch 31/45
```

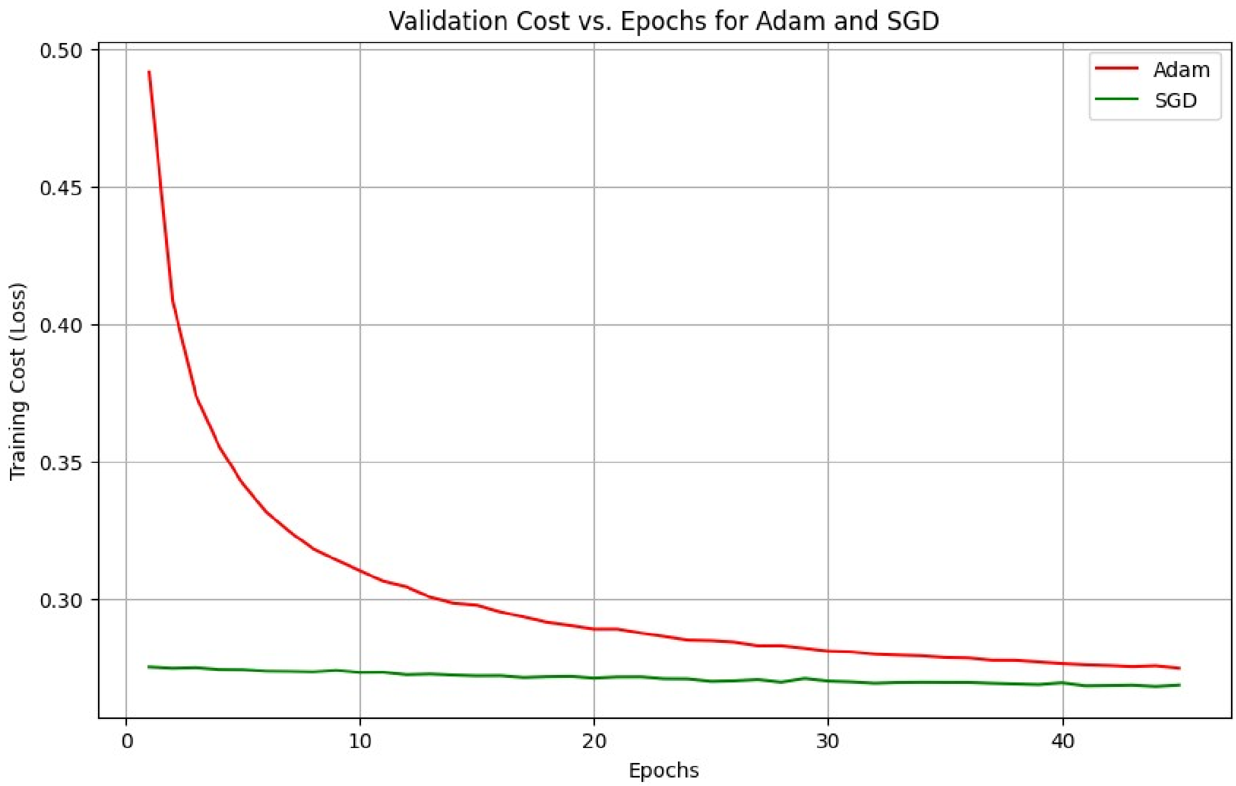
```
469/469 [==============================] - 1s 3ms/step - loss: 0.2646
- accuracy: 0.9269 - val_loss: 0.2699 - val_accuracy: 0.9234
Epoch 32/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2644
- accuracy: 0.9269 - val_loss: 0.2694 - val_accuracy: 0.9230
Epoch 33/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2641
- accuracy: 0.9274 - val_loss: 0.2697 - val_accuracy: 0.9233
Epoch 34/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2638
- accuracy: 0.9271 - val_loss: 0.2698 - val_accuracy: 0.9238
Epoch 35/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2636
- accuracy: 0.9276 - val_loss: 0.2697 - val_accuracy: 0.9237
Epoch 36/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2633
- accuracy: 0.9275 - val_loss: 0.2697 - val_accuracy: 0.9232
Epoch 37/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2632
- accuracy: 0.9275 - val_loss: 0.2694 - val_accuracy: 0.9248
Epoch 38/45
469/469 [==============================] - 2s 5ms/step - loss: 0.2630
- accuracy: 0.9276 - val_loss: 0.2692 - val_accuracy: 0.9240
Epoch 39/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2628
- accuracy: 0.9273 - val_loss: 0.2689 - val_accuracy: 0.9239
Epoch 40/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2625
- accuracy: 0.9275 - val_loss: 0.2696 - val_accuracy: 0.9236
Epoch 41/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2624
- accuracy: 0.9279 - val_loss: 0.2685 - val_accuracy: 0.9239
Epoch 42/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2621
- accuracy: 0.9277 - val_loss: 0.2686 - val_accuracy: 0.9249
Epoch 43/45
469/469 [==============================] - 2s 3ms/step - loss: 0.2619
- accuracy: 0.9277 - val_loss: 0.2687 - val_accuracy: 0.9242
Epoch 44/45
469/469 [==============================] - 1s 3ms/step - loss: 0.2618
- accuracy: 0.9280 - val_loss: 0.2682 - val_accuracy: 0.9242
Epoch 45/45
469/469 [==============================] - 2s 4ms/step - loss: 0.2615
- accuracy: 0.9276 - val_loss: 0.2688 - val_accuracy: 0.9237

# Extract training cost (loss) values from history
adam_training_costs = history.history['loss']
sgd_training_costs = sgd_history.history['loss']
```

```
# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_costs) + 1), adam_training_costs,
label='Adam', color='r')
plt.plot(range(1, len(sgd_training_costs) + 1), sgd_training_costs,
label='SGD', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('Training Cost vs. Epochs for Adam and SGD')
plt.legend()
plt.grid(True)
plt.show()
```

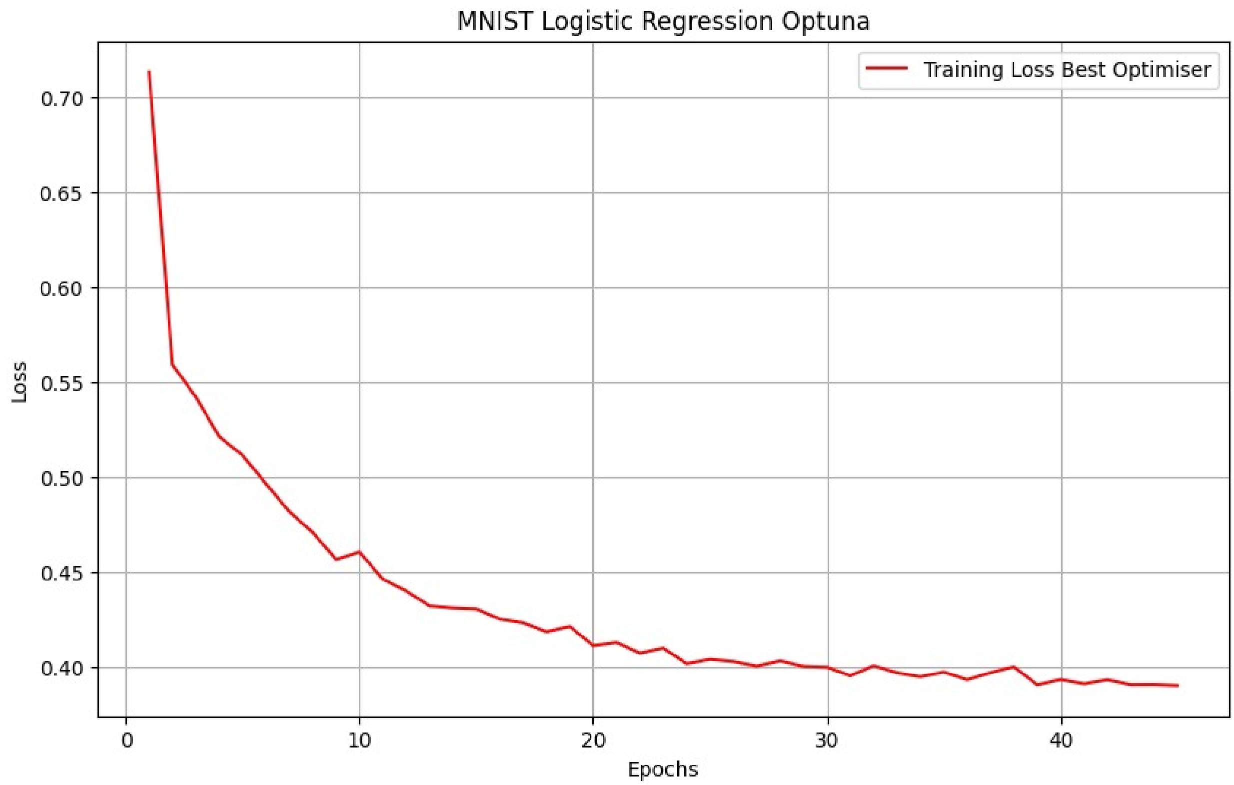



```
# Extract training cost (loss) values from history
adam_training_costs = history.history['val_loss']
sgd_training_costs = sgd_history.history['val_loss']

# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_costs) + 1), adam_training_costs,
label='Adam', color='r')
plt.plot(range(1, len(sgd_training_costs) + 1), sgd_training_costs,
label='SGD', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
```

```
plt.title('Validation Cost vs. Epochs for Adam and SGD')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Plot training cost vs. epochs for the best optimizer
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(history.history['loss']) + 1),
history.history['loss'], label='Training Loss Best Optimiser',
color='r')
# plt.plot(range(1, len(history.history['val_loss']) + 1),
history.history['val_loss'], label='Validation Loss', color='b')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('MNIST Logistic Regression Optuna')
plt.legend()
plt.grid(True)
plt.show()
```



Observation Adam vs SGD

As per the paper, we can also observe that fine tuned SGD with momenetum and best hyperparam values outperformed Adam, although adam is found to be the best optimizer.

SGD without fine tuing and static values was not able to converge properly.

MLP for MNIST

```
def objective(trial):

  epochs = 45
  # Create and compile the logistic regression model
  optimizer_name = trial.suggest_categorical('optimizer', ['adam',
'sgd'])
  learning_rate = trial.suggest_float('learning_rate', 1e-3, 1e-2,
log=True)
  dropout_rate = trial.suggest_float('dropout_rate', 0.5, 0.6)
  beta_1 = trial.suggest_float('beta_1', 0.0, 0.9)  # Vary beta_1
within [0, 0.9]
  # beta_1 = 0.9
  # beta_2 = 0.99
  beta_2 = trial.suggest_float('beta_2', 0.99, 0.9999)  # Vary beta_2
within [0.99, 0.9999]
  epsilon = trial.suggest_float('epsilon', 1e-8, 1e-7)
  # epsilon = 1e-8
```

```
  momentum = trial.suggest_float('momentum', 0.9, 0.99)
  # momentum = 0.9
  l2_reg = trial.suggest_float('l2_reg', 1e-6, 1e-3, log=True)  # L2
regularization strength

  # Implement the learning rate schedule for Adam
  # def learning_rate_schedule(epoch, lr):
      # t = epoch + 1  # Current epoch
      # return initial_learning_rate / math.sqrt(t)

  # decay_steps = len(X_train) // 128  # Adjust as needed
  # decay_steps = epochs  # Adjust as needed
  # lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
  #       learning_rate, decay_steps=decay_steps,
decay_rate=1/np.sqrt(epochs), staircase=False
  # )
  # Define the custom learning rate scheduler
  class CustomLRScheduler(tf.keras.callbacks.Callback):
      def on_epoch_begin(self, epoch, logs=None):
          new_lr = 0.001 / np.sqrt(epoch+1)
          K.set_value(self.model.optimizer.lr, new_lr)
          print(f'Epoch {epoch + 1}: Learning Rate = {new_lr}')

  # Binary class
  model = build_multi_layer_perceptrons_model(optimizer_name, trial,
learning_rate,
                                          beta_1, beta_2, l2_reg)

  # Create a custom callback to track validation loss
  val_loss_callback = ValidationLoss()
  lr_scheduler_callback = CustomLRScheduler()

  # Train the model
  history = model.fit(X_train, y_train, epochs=epochs, batch_size=128,
                      validation_data=(X_test, y_test), verbose=0,
                      callbacks=[val_loss_callback,
lr_scheduler_callback])

  # Get the minimum validation loss
  min_val_loss = min(val_loss_callback.validation_losses)

  return min_val_loss

# Create an Optuna study
study_mlp = optuna.create_study(direction='minimize')

# Optimize hyperparameters
# study.optimize(objective, n_trials=45)
study_mlp.optimize(objective, n_trials=1)
```

```
best_trial_mlp = study_mlp.best_trial

# Print the best hyperparameters
print(f'Best Optimizer: {best_trial_mlp.params['optimizer']}')
print(f'Best Learning Rate: {best_trial_mlp.params['learning_rate']}')
```

```
[I 2023-10-07 17:04:29,555] A new study created in memory with name:
no-name-787c1035-de20-48ee-8af0-12324105ca4c
```

```
313/313 [==============================] - 3s 7ms/step - loss: 2.8608
- accuracy: 0.1231
MLP - Test Loss: 2.860802412033081, Test Accuracy: 0.12309999763965607
Epoch 1: Learning Rate = 0.001
Epoch 2: Learning Rate = 0.0007071067811865475
Epoch 3: Learning Rate = 0.0005773502691896258
Epoch 4: Learning Rate = 0.0005
Epoch 5: Learning Rate = 0.0004472135954999579
Epoch 6: Learning Rate = 0.0004082482904638631
Epoch 7: Learning Rate = 0.0003779644730092272
Epoch 8: Learning Rate = 0.00035355339059327376
Epoch 9: Learning Rate = 0.0003333333333333333
Epoch 10: Learning Rate = 0.00031622776601683794
Epoch 11: Learning Rate = 0.00030151134457776364
Epoch 12: Learning Rate = 0.0002886751345948129
Epoch 13: Learning Rate = 0.0002773500981126146
Epoch 14: Learning Rate = 0.0002672612419124244
Epoch 15: Learning Rate = 0.0002581988897471611
Epoch 16: Learning Rate = 0.00025
Epoch 17: Learning Rate = 0.0002425356250363330
Epoch 18: Learning Rate = 0.00023570226039551587
Epoch 19: Learning Rate = 0.00022941573387056174
Epoch 20: Learning Rate = 0.00022360679774997895
Epoch 21: Learning Rate = 0.0002182178902359924
Epoch 22: Learning Rate = 0.00021320071635561044
Epoch 23: Learning Rate = 0.0002085144140570748
Epoch 24: Learning Rate = 0.00020412414523193154
Epoch 25: Learning Rate = 0.0002
Epoch 26: Learning Rate = 0.00019611613513818404
Epoch 27: Learning Rate = 0.00019245008972987527
Epoch 28: Learning Rate = 0.0001889822365046136
Epoch 29: Learning Rate = 0.0001856953381770519
Epoch 30: Learning Rate = 0.00018257418583505537
Epoch 31: Learning Rate = 0.00017960530202677493
Epoch 32: Learning Rate = 0.00017677669529663688
Epoch 33: Learning Rate = 0.00017407765595569785
Epoch 34: Learning Rate = 0.00017149858514250885
Epoch 35: Learning Rate = 0.0001690308509457033
Epoch 36: Learning Rate = 0.00016666666666666666
Epoch 37: Learning Rate = 0.0001643989873053573
Epoch 38: Learning Rate = 0.00016222142113076255
```

```
Epoch 39: Learning Rate = 0.00016012815380508712
Epoch 40: Learning Rate = 0.00015811388300841897
Epoch 41: Learning Rate = 0.00015617376188860606
Epoch 42: Learning Rate = 0.00015430334996209192
Epoch 43: Learning Rate = 0.00015249857033260467
Epoch 44: Learning Rate = 0.00015075567228888182
Epoch 45: Learning Rate = 0.00014907119849998598

[I 2023-10-07 17:20:49,866] Trial 0 finished with value:
0.09418720006942749 and parameters: {'optimizer': 'adam',
'learning_rate': 0.0030078035155307018, 'dropout_rate':
0.5360351232418614, 'beta_1': 0.060912971518701654, 'beta_2':
0.9976266796097788, 'epsilon': 3.638850233127473e-08, 'momentum':
0.9334336489509198, 'l2_reg': 0.00029757452768789297}. Best is trial 0
with value: 0.09418720006942749.

Best Optimizer: adam
Best Learning Rate: 0.0030078035155307018
```

```
# Get the best trial and hyperparameters
best_optimizer_mlp = best_trial_mlp.params['optimizer']
best_lr_mlp = best_trial_mlp.params['learning_rate']
best_beta_1_mlp = best_trial_mlp.params['beta_1']
best_beta_2_mlp = best_trial_mlp.params['beta_2']
best_l2_reg_mlp = best_trial_mlp.params['l2_reg']

# Retrieve and print the best hyperparameters
best_mlp_hyperparameters = best_trial_mlp.params
print("Best Hyperparameters for MLP on MNIST dataset:")
for param_name, param_value in best_mlp_hyperparameters.items():
    print(f"{param_name}: {param_value}")
```

```
Best Hyperparameters for MLP on MNIST dataset:
optimizer: adam
learning_rate: 0.0030078035155307018
dropout_rate: 0.5360351232418614
beta_1: 0.060912971518701654
beta_2: 0.9976266796097788
epsilon: 3.638850233127473e-08
momentum: 0.9334336489509198
l2_reg: 0.00029757452768789297
```

```
if best_optimizer_mlp == 'adam':
  # Build and train the final model with the best hyperparameters
  best_model = build_multi_layer_perceptrons_model(best_optimizer_mlp,
best_trial_mlp, best_lr_mlp,
                                          best_beta_1_mlp,
best_beta_2_mlp, best_l2_reg_mlp)
  history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)
```

```
  sgd_model = build_multi_layer_perceptrons_model("sgd",
best_trial_mlp, best_lr_mlp,
                                          best_beta_1_mlp,
best_beta_2_mlp, best_l2_reg_mlp)

  sgd_history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_training_mlp_costs = history.history['loss']
  sgd_training_mlp_costs = sgd_history.history['loss']

elif best_optimizer_mlp == 'sgd':
  # Build and train the final model with the best hyperparameters
  best_model = build_multi_layer_perceptrons_model(best_optimizer_mlp,
best_trial_mlp, best_lr_mlp,
                                          best_beta_1_mlp,
best_beta_2_mlp, best_l2_reg_mlp)
  history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_model = build_multi_layer_perceptrons_model("adam",
best_trial_mlp, best_lr_mlp,
                                          best_beta_1_mlp,
best_beta_2_mlp, best_l2_reg_mlp)

  adam_history = best_model.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_training_mlp_costs = adam_history.history['loss']
  sgd_training_mlp_costs = history.history['loss']
else:
    raise ValueError("Invalid optimizer name")
```

```
313/313 [==============================] - 2s 7ms/step - loss: 2.8757
- accuracy: 0.1132
MLP - Test Loss: 2.875709295272827, Test Accuracy: 0.11320000141859055
Epoch 1/45
469/469 [==============================] - 26s 54ms/step - loss:
0.7132 - accuracy: 0.8874 - val_loss: 0.5101 - val_accuracy: 0.9366
Epoch 2/45
469/469 [==============================] - 26s 56ms/step - loss:
0.5590 - accuracy: 0.9297 - val_loss: 0.4724 - val_accuracy: 0.9583
Epoch 3/45
469/469 [==============================] - 25s 53ms/step - loss:
0.5420 - accuracy: 0.9342 - val_loss: 0.4637 - val_accuracy: 0.9556
Epoch 4/45
```

```
469/469 [==============================] - 26s 56ms/step - loss:
0.5211 - accuracy: 0.9354 - val_loss: 0.4258 - val_accuracy: 0.9644
Epoch 5/45
469/469 [==============================] - 25s 53ms/step - loss:
0.5112 - accuracy: 0.9371 - val_loss: 0.6212 - val_accuracy: 0.8941
Epoch 6/45
469/469 [==============================] - 24s 52ms/step - loss:
0.4963 - accuracy: 0.9370 - val_loss: 0.3926 - val_accuracy: 0.9665
Epoch 7/45
469/469 [==============================] - 24s 52ms/step - loss:
0.4815 - accuracy: 0.9392 - val_loss: 0.3873 - val_accuracy: 0.9624
Epoch 8/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4706 - accuracy: 0.9383 - val_loss: 0.3799 - val_accuracy: 0.9636
Epoch 9/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4564 - accuracy: 0.9398 - val_loss: 0.3730 - val_accuracy: 0.9604
Epoch 10/45
469/469 [==============================] - 26s 56ms/step - loss:
0.4603 - accuracy: 0.9374 - val_loss: 0.3710 - val_accuracy: 0.9609
Epoch 11/45
469/469 [==============================] - 26s 55ms/step - loss:
0.4460 - accuracy: 0.9406 - val_loss: 0.3527 - val_accuracy: 0.9663
Epoch 12/45
469/469 [==============================] - 25s 54ms/step - loss:
0.4398 - accuracy: 0.9394 - val_loss: 0.3958 - val_accuracy: 0.9463
Epoch 13/45
469/469 [==============================] - 25s 54ms/step - loss:
0.4319 - accuracy: 0.9408 - val_loss: 0.3437 - val_accuracy: 0.9657
Epoch 14/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4308 - accuracy: 0.9395 - val_loss: 0.4081 - val_accuracy: 0.9442
Epoch 15/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4303 - accuracy: 0.9398 - val_loss: 0.3360 - val_accuracy: 0.9668
Epoch 16/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4250 - accuracy: 0.9397 - val_loss: 0.3357 - val_accuracy: 0.9679
Epoch 17/45
469/469 [==============================] - 24s 51ms/step - loss:
0.4230 - accuracy: 0.9403 - val_loss: 0.3242 - val_accuracy: 0.9671
Epoch 18/45
469/469 [==============================] - 26s 55ms/step - loss:
0.4183 - accuracy: 0.9413 - val_loss: 0.3210 - val_accuracy: 0.9675
Epoch 19/45
469/469 [==============================] - 27s 58ms/step - loss:
0.4210 - accuracy: 0.9395 - val_loss: 0.3308 - val_accuracy: 0.9651
Epoch 20/45
469/469 [==============================] - 26s 55ms/step - loss:
```

```
0.4110 - accuracy: 0.9406 - val_loss: 0.3173 - val_accuracy: 0.9683
Epoch 21/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4126 - accuracy: 0.9403 - val_loss: 0.3296 - val_accuracy: 0.9641
Epoch 22/45
469/469 [==============================] - 24s 51ms/step - loss:
0.4070 - accuracy: 0.9412 - val_loss: 0.3169 - val_accuracy: 0.9673
Epoch 23/45
469/469 [==============================] - 25s 54ms/step - loss:
0.4097 - accuracy: 0.9403 - val_loss: 0.3225 - val_accuracy: 0.9641
Epoch 24/45
469/469 [==============================] - 25s 54ms/step - loss:
0.4015 - accuracy: 0.9404 - val_loss: 0.3357 - val_accuracy: 0.9612
Epoch 25/45
469/469 [==============================] - 26s 56ms/step - loss:
0.4038 - accuracy: 0.9396 - val_loss: 0.3214 - val_accuracy: 0.9629
Epoch 26/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4026 - accuracy: 0.9404 - val_loss: 0.3140 - val_accuracy: 0.9648
Epoch 27/45
469/469 [==============================] - 26s 56ms/step - loss:
0.4002 - accuracy: 0.9403 - val_loss: 0.3239 - val_accuracy: 0.9624
Epoch 28/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4029 - accuracy: 0.9394 - val_loss: 0.3062 - val_accuracy: 0.9671
Epoch 29/45
469/469 [==============================] - 25s 54ms/step - loss:
0.4000 - accuracy: 0.9406 - val_loss: 0.3217 - val_accuracy: 0.9630
Epoch 30/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3996 - accuracy: 0.9399 - val_loss: 0.3032 - val_accuracy: 0.9677
Epoch 31/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3951 - accuracy: 0.9413 - val_loss: 0.2953 - val_accuracy: 0.9685
Epoch 32/45
469/469 [==============================] - 25s 53ms/step - loss:
0.4003 - accuracy: 0.9400 - val_loss: 0.3171 - val_accuracy: 0.9634
Epoch 33/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3966 - accuracy: 0.9406 - val_loss: 0.3461 - val_accuracy: 0.9507
Epoch 34/45
469/469 [==============================] - 27s 58ms/step - loss:
0.3946 - accuracy: 0.9410 - val_loss: 0.3108 - val_accuracy: 0.9626
Epoch 35/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3970 - accuracy: 0.9407 - val_loss: 0.3194 - val_accuracy: 0.9623
Epoch 36/45
469/469 [==============================] - 24s 52ms/step - loss:
0.3931 - accuracy: 0.9419 - val_loss: 0.3198 - val_accuracy: 0.9622
```

```
Epoch 37/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3967 - accuracy: 0.9398 - val_loss: 0.3090 - val_accuracy: 0.9645
Epoch 38/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3998 - accuracy: 0.9384 - val_loss: 0.3095 - val_accuracy: 0.9644
Epoch 39/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3903 - accuracy: 0.9413 - val_loss: 0.2980 - val_accuracy: 0.9672
Epoch 40/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3931 - accuracy: 0.9400 - val_loss: 0.3074 - val_accuracy: 0.9656
Epoch 41/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3909 - accuracy: 0.9408 - val_loss: 0.3161 - val_accuracy: 0.9610
Epoch 42/45
469/469 [==============================] - 26s 56ms/step - loss:
0.3930 - accuracy: 0.9396 - val_loss: 0.2994 - val_accuracy: 0.9658
Epoch 43/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3904 - accuracy: 0.9406 - val_loss: 0.3048 - val_accuracy: 0.9629
Epoch 44/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3905 - accuracy: 0.9421 - val_loss: 0.3040 - val_accuracy: 0.9643
Epoch 45/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3900 - accuracy: 0.9405 - val_loss: 0.3367 - val_accuracy: 0.9508
313/313 [==============================] - 2s 7ms/step - loss: 2.8859
- accuracy: 0.0774
MLP - Test Loss: 2.8858821392059326, Test Accuracy:
0.07739999890327454
Epoch 1/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3926 - accuracy: 0.9404 - val_loss: 0.2854 - val_accuracy: 0.9686
Epoch 2/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3850 - accuracy: 0.9415 - val_loss: 0.3204 - val_accuracy: 0.9565
Epoch 3/45
469/469 [==============================] - 28s 59ms/step - loss:
0.3850 - accuracy: 0.9411 - val_loss: 0.3393 - val_accuracy: 0.9529
Epoch 4/45
469/469 [==============================] - 27s 57ms/step - loss:
0.3899 - accuracy: 0.9389 - val_loss: 0.3188 - val_accuracy: 0.9615
Epoch 5/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3902 - accuracy: 0.9405 - val_loss: 0.2931 - val_accuracy: 0.9673
Epoch 6/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3882 - accuracy: 0.9402 - val_loss: 0.3583 - val_accuracy: 0.9481
```

```
Epoch 7/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3882 - accuracy: 0.9401 - val_loss: 0.3192 - val_accuracy: 0.9613
Epoch 8/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3878 - accuracy: 0.9397 - val_loss: 0.2892 - val_accuracy: 0.9677
Epoch 9/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3825 - accuracy: 0.9414 - val_loss: 0.2974 - val_accuracy: 0.9656
Epoch 10/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3872 - accuracy: 0.9402 - val_loss: 0.2849 - val_accuracy: 0.9714
Epoch 11/45
469/469 [==============================] - 26s 56ms/step - loss:
0.3853 - accuracy: 0.9413 - val_loss: 0.2769 - val_accuracy: 0.9707
Epoch 12/45
469/469 [==============================] - 27s 57ms/step - loss:
0.3850 - accuracy: 0.9396 - val_loss: 0.3021 - val_accuracy: 0.9623
Epoch 13/45
469/469 [==============================] - 27s 57ms/step - loss:
0.3843 - accuracy: 0.9402 - val_loss: 0.3128 - val_accuracy: 0.9605
Epoch 14/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3856 - accuracy: 0.9393 - val_loss: 0.2833 - val_accuracy: 0.9685
Epoch 15/45
469/469 [==============================] - 27s 57ms/step - loss:
0.3817 - accuracy: 0.9399 - val_loss: 0.2909 - val_accuracy: 0.9680
Epoch 16/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3819 - accuracy: 0.9421 - val_loss: 0.2959 - val_accuracy: 0.9657
Epoch 17/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3861 - accuracy: 0.9401 - val_loss: 0.2931 - val_accuracy: 0.9638
Epoch 18/45
469/469 [==============================] - 27s 58ms/step - loss:
0.3862 - accuracy: 0.9386 - val_loss: 0.3037 - val_accuracy: 0.9617
Epoch 19/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3804 - accuracy: 0.9410 - val_loss: 0.2887 - val_accuracy: 0.9671
Epoch 20/45
469/469 [==============================] - 24s 52ms/step - loss:
0.3819 - accuracy: 0.9406 - val_loss: 0.3061 - val_accuracy: 0.9599
Epoch 21/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3834 - accuracy: 0.9407 - val_loss: 0.2874 - val_accuracy: 0.9670
Epoch 22/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3793 - accuracy: 0.9411 - val_loss: 0.2875 - val_accuracy: 0.9675
Epoch 23/45
```

```
469/469 [==============================] - 26s 55ms/step - loss:
0.3846 - accuracy: 0.9395 - val_loss: 0.2902 - val_accuracy: 0.9669
Epoch 24/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3828 - accuracy: 0.9400 - val_loss: 0.2862 - val_accuracy: 0.9680
Epoch 25/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3789 - accuracy: 0.9404 - val_loss: 0.2853 - val_accuracy: 0.9695
Epoch 26/45
469/469 [==============================] - 28s 59ms/step - loss:
0.3794 - accuracy: 0.9406 - val_loss: 0.3076 - val_accuracy: 0.9603
Epoch 27/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3766 - accuracy: 0.9413 - val_loss: 0.2909 - val_accuracy: 0.9661
Epoch 28/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3807 - accuracy: 0.9406 - val_loss: 0.2988 - val_accuracy: 0.9643
Epoch 29/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3811 - accuracy: 0.9398 - val_loss: 0.2885 - val_accuracy: 0.9656
Epoch 30/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3774 - accuracy: 0.9409 - val_loss: 0.2889 - val_accuracy: 0.9685
Epoch 31/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3790 - accuracy: 0.9405 - val_loss: 0.3330 - val_accuracy: 0.9525
Epoch 32/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3753 - accuracy: 0.9416 - val_loss: 0.2809 - val_accuracy: 0.9685
Epoch 33/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3835 - accuracy: 0.9403 - val_loss: 0.2862 - val_accuracy: 0.9661
Epoch 34/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3815 - accuracy: 0.9407 - val_loss: 0.3038 - val_accuracy: 0.9611
Epoch 35/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3776 - accuracy: 0.9409 - val_loss: 0.2876 - val_accuracy: 0.9649
Epoch 36/45
469/469 [==============================] - 26s 56ms/step - loss:
0.3746 - accuracy: 0.9408 - val_loss: 0.2952 - val_accuracy: 0.9633
Epoch 37/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3852 - accuracy: 0.9384 - val_loss: 0.2780 - val_accuracy: 0.9691
Epoch 38/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3797 - accuracy: 0.9404 - val_loss: 0.2837 - val_accuracy: 0.9679
Epoch 39/45
469/469 [==============================] - 26s 55ms/step - loss:
```

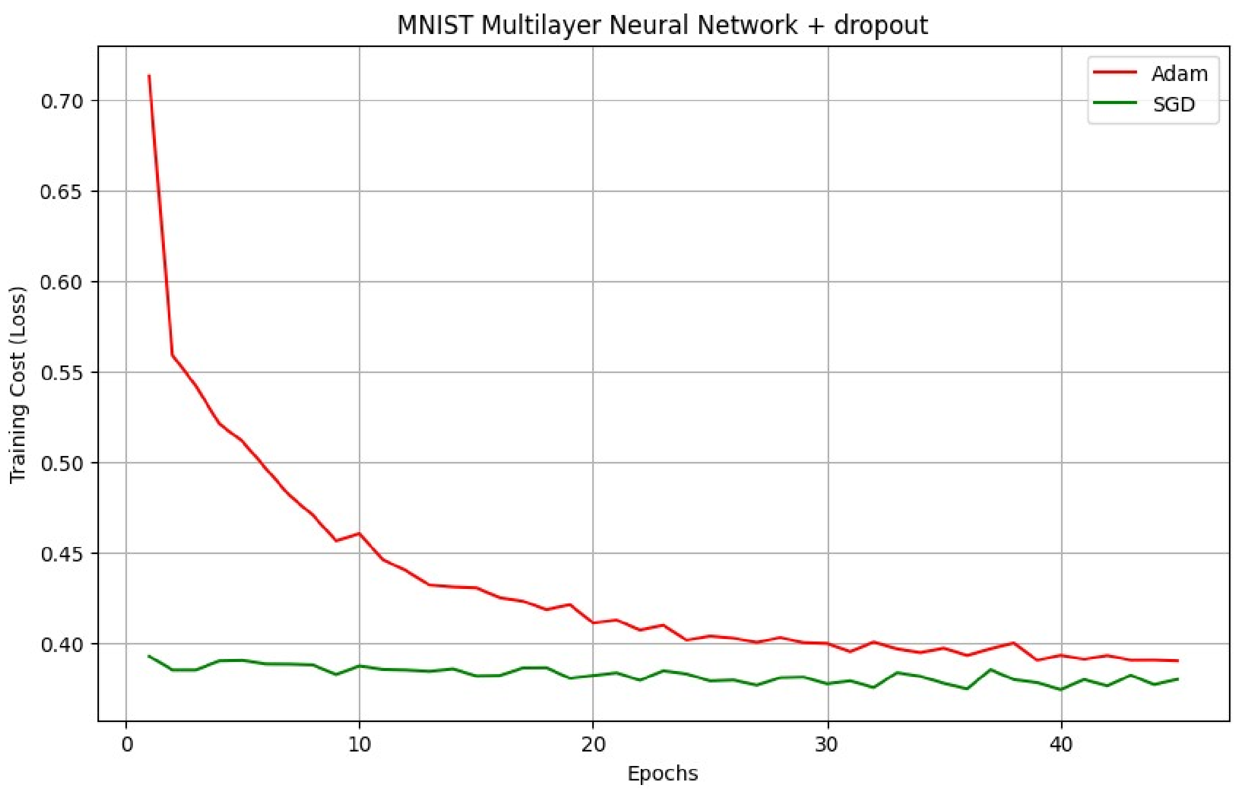
```
0.3780 - accuracy: 0.9406 - val_loss: 0.2946 - val_accuracy: 0.9663
Epoch 40/45
469/469 [==============================] - 27s 57ms/step - loss:
0.3742 - accuracy: 0.9418 - val_loss: 0.3146 - val_accuracy: 0.9569
Epoch 41/45
469/469 [==============================] - 27s 57ms/step - loss:
0.3797 - accuracy: 0.9392 - val_loss: 0.2994 - val_accuracy: 0.9647
Epoch 42/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3762 - accuracy: 0.9405 - val_loss: 0.2915 - val_accuracy: 0.9641
Epoch 43/45
469/469 [==============================] - 25s 53ms/step - loss:
0.3821 - accuracy: 0.9399 - val_loss: 0.3059 - val_accuracy: 0.9576
Epoch 44/45
469/469 [==============================] - 26s 55ms/step - loss:
0.3769 - accuracy: 0.9396 - val_loss: 0.2950 - val_accuracy: 0.9632
Epoch 45/45
469/469 [==============================] - 25s 54ms/step - loss:
0.3798 - accuracy: 0.9401 - val_loss: 0.3024 - val_accuracy: 0.9597
```

```
print(adam_training_mlp_costs)
print(sgd_training_mlp_costs)
```

```
[0.713222861289978, 0.5590003728866577, 0.5419694781303406,
0.5210512280464172, 0.5112354755401611, 0.49628153443336487,
0.48153743147850037, 0.4705853760242462, 0.45636844635009766,
0.4602936804294586, 0.445973902940750l, 0.43979865312576294,
0.43190616369247437, 0.43080469965934753, 0.430274099111557,
0.4249860942363739, 0.4229927659034729, 0.418302983045578,
0.4210262894630432, 0.41097506880760193, 0.41261979937553406,
0.4070376753807068, 0.4096870422363281, 0.4014561176300049,
0.40378621220588684, 0.40258780121803284, 0.40015605092048645,
0.40293174982070923, 0.3999958038330078, 0.39957892894744873,
0.39513319730758667, 0.4003044366836548, 0.39664241671562195,
0.3946174681186676, 0.39701932668685913, 0.39308348298072815,
0.3966973125934601, 0.3998142182826996, 0.39033371210098267,
0.39313000440597534, 0.3908863961696625, 0.39301759004592896,
0.3904017508029938, 0.3904514014720917, 0.3899695575237274]
[0.3926064074039459, 0.38499772548675537, 0.38499656319618225,
0.38994643092155457, 0.39022672176361084, 0.3882436752319336,
0.3881559669971466, 0.3877656161785126, 0.382520467042923,
0.38719266653060913, 0.38529345393180847, 0.384971022605896,
0.38425472378730774, 0.3855842649936676, 0.38170334696769714,
0.3819020688533783, 0.3861245810985565, 0.3862023651599884,
0.3803958296775818, 0.38191646337509155, 0.38342034816741943,
0.3792942762374878, 0.38457125425338745, 0.3827972710132599,
0.37893185019493103, 0.3793991208076477, 0.37657099962234497,
0.3807166814804077, 0.3810654580593109, 0.37735068798065186,
0.37896648049354553, 0.3752952218055725, 0.38350605964660645,
0.38148537278175354, 0.37762027978897095, 0.37458497285842896,
```

```
0.3851820230484009, 0.3796778917312622, 0.37798869609832764,
0.37419936060905457, 0.3796813488006592, 0.3762228190898895,
0.38207679986953735, 0.37692052125930786, 0.37983372807502747]
```

```
# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_mlp_costs) + 1),
adam_training_mlp_costs, label='Adam', color='r')
plt.plot(range(1, len(sgd_training_mlp_costs) + 1),
sgd_training_mlp_costs, label='SGD', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('MNIST Multilayer Neural Network + dropout')
plt.legend()
plt.grid(True)
plt.show()
```



IMDB Dataset

Log Reg for IMDB

```
print(type(x_train_bow), type(x_test_bow))
```

```
<class 'scipy.sparse._csr.csr_matrix'> <class
'scipy.sparse._csr.csr_matrix'>
```

```
from scipy.sparse import csr_matrix

x_train_bow = tf.convert_to_tensor(x_train_bow.toarray(),
dtype=tf.float32)
x_test_bow = tf.convert_to_tensor(x_test_bow.toarray(),
dtype=tf.float32)

# Preprocess the data (pad sequences to a fixed length)
from keras.preprocessing.sequence import pad_sequences
# max_length = 200  # You can adjust this as needed
max_length = max(len(sample) for sample in x_train_bow)
x_train_bow = pad_sequences(x_train_bow, maxlen=max_length,
padding='post')

x_test_bow = pad_sequences(x_test_bow, maxlen=max_length,
padding='post')

print(np.where(x_train_bow != 0.0))
print(np.where(x_test_bow != 0.0))
```

```
(array([    0,     0,     0, ..., 24999, 24999, 24999]), array([ 230,
312,  387, ..., 9541, 9632, 9668]))
(array([    0,     0,     0, ..., 24999, 24999, 24999]), array([ 387,
400,  405, ..., 9444, 9587, 9696]))
```

```
def objective(trial):

  epochs = 45
  # Create and compile the logistic regression model
  optimizer_name = trial.suggest_categorical('optimizer', ['adam',
'sgd'])
  learning_rate = trial.suggest_float('learning_rate', 1e-3, 1e-2,
log=True)
  # dropout_rate = trial.suggest_float('dropout_rate', 0.0, 0.5)
  beta_1 = trial.suggest_float('beta_1', 0.0, 0.9)  # Vary beta_1
within [0, 0.9]
  beta_2 = trial.suggest_float('beta_2', 0.99, 0.9999)  # Vary beta_2
within [0.99, 0.9999]
  decay = trial.suggest_float('decay', 1e-6, 1e-2, log=True)
  decay_steps = trial.suggest_int('decay_steps', 1, len(x_train_bow)
// 128)
  epsilon = trial.suggest_float('epsilon', 1e-9, 1e-7)
  momentum = trial.suggest_float('momentum', 0.9, 0.99)

  # Binary class
  model = build_logistic_regression_bin_model(optimizer_name, trial,
learning_rate,
                                          beta_1, beta_2)
```

```
  # Define the custom learning rate scheduler
  class CustomLRScheduler(tf.keras.callbacks.Callback):
      def on_epoch_begin(self, epoch, logs=None):
          new_lr = 0.001 / np.sqrt(epoch+1)
          K.set_value(self.model.optimizer.lr, new_lr)
          print(f'Epoch {epoch + 1}: Learning Rate = {new_lr}')

  # Create a custom callback to track validation loss
  val_loss_callback = ValidationLoss()
  lr_scheduler = CustomLRScheduler()

  # Train the model
  history = model.fit(x_train_bow, y_train_bow, epochs=epochs,
batch_size=128,
                      validation_data=(x_test_bow, y_test_bow),
verbose=0,
                      callbacks=[val_loss_callback, lr_scheduler])

  # Get the minimum validation loss
  min_val_loss = min(val_loss_callback.validation_losses)

  return min_val_loss

# Create an Optuna study
study_lg_imdb = optuna.create_study(direction='minimize')

# Optimize hyperparameters
# study.optimize(objective, n_trials=45)
study_lg_imdb.optimize(objective, n_trials=1)

# Get the best trial and hyperparameters
best_trial_lg_imdb = study_lg_imdb.best_trial
best_optimizer_lg_imdb = best_trial_lg_imdb.params['optimizer']
best_lr_lg_imdb = best_trial_lg_imdb.params['learning_rate']
best_beta_1_lg_imdb = best_trial_lg_imdb.params['beta_1']
best_beta_2_lg_imdb = best_trial_lg_imdb.params['beta_2']

# Print the best hyperparameters
print(f'Best Optimizer: {best_optimizer_lg_imdb}')
print(f'Best Learning Rate: {best_lr_lg_imdb}')
```

```
[I 2023-10-07 16:17:37,146] A new study created in memory with name:
no-name-a55ef85f-53e1-4f62-8a73-8215047b2c2c

Epoch 1: Learning Rate = 1.0
Epoch 2: Learning Rate = 0.7071067811865475
Epoch 3: Learning Rate = 0.5773502691896258
Epoch 4: Learning Rate = 0.5
Epoch 5: Learning Rate = 0.4472135954999579
Epoch 6: Learning Rate = 0.4082482904638631
```

```
Epoch 7: Learning Rate = 0.3779644730092272
Epoch 8: Learning Rate = 0.35355339059327373
Epoch 9: Learning Rate = 0.3333333333333333
Epoch 10: Learning Rate = 0.31622776601683794
Epoch 11: Learning Rate = 0.30151134457776363
Epoch 12: Learning Rate = 0.2886751345948129
Epoch 13: Learning Rate = 0.2773500981126146
Epoch 14: Learning Rate = 0.2672612419124244
Epoch 15: Learning Rate = 0.2581988897471611
Epoch 16: Learning Rate = 0.25
Epoch 17: Learning Rate = 0.24253562503633297
Epoch 18: Learning Rate = 0.23570226039551587
Epoch 19: Learning Rate = 0.22941573387056174
Epoch 20: Learning Rate = 0.22360679774997896
Epoch 21: Learning Rate = 0.2182178902359924
Epoch 22: Learning Rate = 0.21320071635561041
Epoch 23: Learning Rate = 0.20851441405707477
Epoch 24: Learning Rate = 0.20412414523193154
Epoch 25: Learning Rate = 0.2
Epoch 26: Learning Rate = 0.19611613513818404
Epoch 27: Learning Rate = 0.19245008972987526
Epoch 28: Learning Rate = 0.1889822365046136
Epoch 29: Learning Rate = 0.18569533817705186
Epoch 30: Learning Rate = 0.18257418583505536
Epoch 31: Learning Rate = 0.1796053020267749
Epoch 32: Learning Rate = 0.17677669529663687
Epoch 33: Learning Rate = 0.17407765595569785
Epoch 34: Learning Rate = 0.17149858514250882
Epoch 35: Learning Rate = 0.1690308509457033
Epoch 36: Learning Rate = 0.16666666666666666
Epoch 37: Learning Rate = 0.1643989873053573
Epoch 38: Learning Rate = 0.16222142113076254
Epoch 39: Learning Rate = 0.16012815380508713
Epoch 40: Learning Rate = 0.15811388300841897
Epoch 41: Learning Rate = 0.15617376188860607
Epoch 42: Learning Rate = 0.1543033499620919
Epoch 43: Learning Rate = 0.15249857033260467
Epoch 44: Learning Rate = 0.15075567228888181
Epoch 45: Learning Rate = 0.14907119849998599

[I 2023-10-07 16:21:01,549] Trial 0 finished with value:
16.59992790222168 and parameters: {'optimizer': 'sgd',
'learning_rate': 0.0060180060264529685, 'beta_1': 0.3629224874118513,
'beta_2': 0.9984999619973528, 'decay': 0.004870768621823307,
'decay_steps': 185, 'epsilon': 3.188656906213701e-08, 'momentum':
0.9852339755385909}. Best is trial 0 with value: 16.59992790222168.

Best Optimizer: sgd
Best Learning Rate: 0.0060180060264529685
```

```
if best_optimizer_lg_imdb == 'adam':
  # Build and train the final model with the best hyperparameters
  best_model_lg_imdb =
build_logistic_regression_bin_model(best_optimizer_lg_imdb,
best_trial_lg_imdb, best_lr_lg_imdb,
                                             best_beta_1_lg_imdb,
best_beta_2_lg_imdb)
  history_lg_imdb = best_model_lg_imdb.fit(x_train_bow, y_train_bow,
epochs=45, batch_size=128,
                          validation_data=(x_test_bow, y_test_bow),
verbose=1)

  sgd_model = build_logistic_regression_bin_model("sgd",
best_trial_lg_imdb, best_lr_lg_imdb,
                                              best_beta_1_lg_imdb,
best_beta_2_lg_imdb)

  sgd_history_lg_imdb = best_model_lg_imdb.fit(x_train_bow,
y_train_bow, epochs=45, batch_size=128,
                          validation_data=(x_test_bow, y_test_bow),
verbose=1)

  adam_training_mlp_costs = history_lg_imdb.history['loss']
  sgd_training_mlp_costs = sgd_history_lg_imdb.history['loss']

elif best_optimizer_lg_imdb == 'sgd':
  # Build and train the final model with the best hyperparameters
  best_model_lg_imdb =
build_logistic_regression_bin_model(best_optimizer_lg_imdb,
best_trial_lg_imdb, best_lr_lg_imdb,
                                             best_beta_1_lg_imdb,
best_beta_2_lg_imdb)
  history_lg_imdb = best_model_lg_imdb.fit(x_train_bow, y_train_bow,
epochs=45, batch_size=128,
                          validation_data=(x_test_bow, y_test_bow),
verbose=1)

  adam_model = build_logistic_regression_bin_model("adam",
best_trial_lg_imdb, best_lr_lg_imdb,
                                              best_beta_1_lg_imdb,
best_beta_2_lg_imdb)

  adam_history_lg_imdb = best_model_lg_imdb.fit(x_train_bow,
y_train_bow, epochs=45, batch_size=128,
                          validation_data=(x_test_bow, y_test_bow),
verbose=1)

  adam_training_mlp_costs = adam_history_lg_imdb.history['loss']
  sgd_training_mlp_costs = history_lg_imdb.history['loss']
```

```
else:
    raise ValueError("Invalid optimizer name")

# # Build and train the final model with the best hyperparameters
# best_model_lg_imdb =
build_logistic_regression_bin_model(best_optimizer_lg_imdb,
best_trial_lg_imdb, best_lr_lg_imdb,
#                                              best_beta_1_lg_imdb,
best_beta_2_lg_imdb)
# history_lg_imdb = best_model_lg_imdb.fit(x_train_bow, y_train_bow,
epochs=45, batch_size=128,
#                          validation_data=(x_test_bow, y_test_bow),
verbose=1)

# sgd_model = build_logistic_regression_bin_model("adam",
best_trial_lg_imdb, best_lr_lg_imdb,
#                                              best_beta_1_lg_imdb,
best_beta_2_lg_imdb)

# sgd_history_lg_imdb = best_model_lg_imdb.fit(x_train_bow,
y_train_bow, epochs=45, batch_size=128,
#                          validation_data=(x_test_bow, y_test_bow),
verbose=1)

# sgd_training_mlp_costs = history_lg_imdb.history['loss']
# adam_training_mlp_costs = sgd_history_lg_imdb.history['loss']
```

```
Epoch 1/45
196/196 [==============================] - 7s 34ms/step - loss: 0.5743
- accuracy: 0.7097 - val_loss: 0.6843 - val_accuracy: 0.5598
Epoch 2/45
196/196 [==============================] - 5s 24ms/step - loss: 0.4726
- accuracy: 0.7883 - val_loss: 0.6872 - val_accuracy: 0.5688
Epoch 3/45
196/196 [==============================] - 11s 55ms/step - loss:
0.4459 - accuracy: 0.8064 - val_loss: 0.6948 - val_accuracy: 0.5660
Epoch 4/45
196/196 [==============================] - 9s 45ms/step - loss: 0.4278
- accuracy: 0.8168 - val_loss: 0.7098 - val_accuracy: 0.5604
Epoch 5/45
196/196 [==============================] - 7s 38ms/step - loss: 0.4152
- accuracy: 0.8241 - val_loss: 0.7238 - val_accuracy: 0.5568
Epoch 6/45
196/196 [==============================] - 7s 38ms/step - loss: 0.3987
- accuracy: 0.8347 - val_loss: 0.7463 - val_accuracy: 0.5547
Epoch 7/45
196/196 [==============================] - 7s 37ms/step - loss: 0.3934
- accuracy: 0.8346 - val_loss: 0.7650 - val_accuracy: 0.5502
Epoch 8/45
196/196 [==============================] - 6s 32ms/step - loss: 0.3865
```

```
- accuracy: 0.8370 - val_loss: 0.7624 - val_accuracy: 0.5543
Epoch 9/45
196/196 [==============================] - 5s 25ms/step - loss: 0.3768
- accuracy: 0.8418 - val_loss: 0.7915 - val_accuracy: 0.5456
Epoch 10/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3712
- accuracy: 0.8460 - val_loss: 0.8120 - val_accuracy: 0.5440
Epoch 11/45
196/196 [==============================] - 6s 28ms/step - loss: 0.3681
- accuracy: 0.8460 - val_loss: 0.8335 - val_accuracy: 0.5374
Epoch 12/45
196/196 [==============================] - 6s 30ms/step - loss: 0.3659
- accuracy: 0.8474 - val_loss: 0.8299 - val_accuracy: 0.5419
Epoch 13/45
196/196 [==============================] - 4s 23ms/step - loss: 0.3605
- accuracy: 0.8498 - val_loss: 0.8735 - val_accuracy: 0.5357
Epoch 14/45
196/196 [==============================] - 6s 28ms/step - loss: 0.3524
- accuracy: 0.8538 - val_loss: 0.8654 - val_accuracy: 0.5388
Epoch 15/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3541
- accuracy: 0.8536 - val_loss: 0.8918 - val_accuracy: 0.5346
Epoch 16/45
196/196 [==============================] - 5s 26ms/step - loss: 0.3499
- accuracy: 0.8564 - val_loss: 0.8959 - val_accuracy: 0.5345
Epoch 17/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3440
- accuracy: 0.8550 - val_loss: 0.9129 - val_accuracy: 0.5335
Epoch 18/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3464
- accuracy: 0.8569 - val_loss: 0.9218 - val_accuracy: 0.5333
Epoch 19/45
196/196 [==============================] - 5s 27ms/step - loss: 0.3447
- accuracy: 0.8587 - val_loss: 0.9288 - val_accuracy: 0.5322
Epoch 20/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3405
- accuracy: 0.8617 - val_loss: 0.9493 - val_accuracy: 0.5318
Epoch 21/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3432
- accuracy: 0.8598 - val_loss: 0.9366 - val_accuracy: 0.5346
Epoch 22/45
196/196 [==============================] - 5s 27ms/step - loss: 0.3410
- accuracy: 0.8573 - val_loss: 0.9695 - val_accuracy: 0.5298
Epoch 23/45
196/196 [==============================] - 5s 28ms/step - loss: 0.3343
- accuracy: 0.8601 - val_loss: 0.9802 - val_accuracy: 0.5305
Epoch 24/45
196/196 [==============================] - 5s 25ms/step - loss: 0.3364
- accuracy: 0.8616 - val_loss: 0.9788 - val_accuracy: 0.5310
```

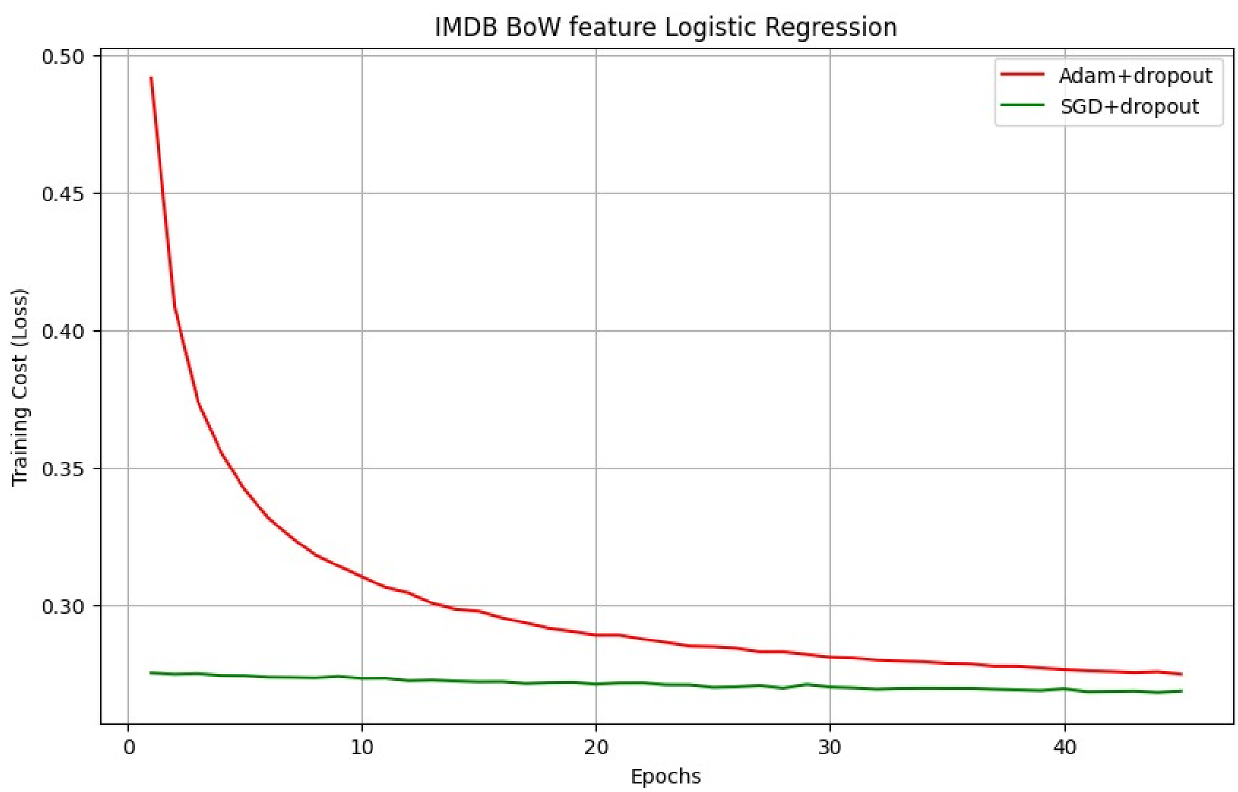
```
Epoch 25/45
196/196 [==============================] - 6s 28ms/step - loss: 0.3356
- accuracy: 0.8596 - val_loss: 1.0192 - val_accuracy: 0.5295
Epoch 26/45
196/196 [==============================] - 5s 25ms/step - loss: 0.3318
- accuracy: 0.8639 - val_loss: 1.0315 - val_accuracy: 0.5282
Epoch 27/45
196/196 [==============================] - 5s 28ms/step - loss: 0.3306
- accuracy: 0.8644 - val_loss: 1.0332 - val_accuracy: 0.5284
Epoch 28/45
196/196 [==============================] - 5s 25ms/step - loss: 0.3328
- accuracy: 0.8633 - val_loss: 1.0192 - val_accuracy: 0.5297
Epoch 29/45
196/196 [==============================] - 4s 21ms/step - loss: 0.3270
- accuracy: 0.8653 - val_loss: 1.0273 - val_accuracy: 0.5286
Epoch 30/45
196/196 [==============================] - 5s 27ms/step - loss: 0.3258
- accuracy: 0.8641 - val_loss: 1.0214 - val_accuracy: 0.5281
Epoch 31/45
196/196 [==============================] - 5s 27ms/step - loss: 0.3224
- accuracy: 0.8678 - val_loss: 1.0342 - val_accuracy: 0.5284
Epoch 32/45
196/196 [==============================] - 5s 23ms/step - loss: 0.3203
- accuracy: 0.8669 - val_loss: 1.0374 - val_accuracy: 0.5305
Epoch 33/45
196/196 [==============================] - 5s 27ms/step - loss: 0.3223
- accuracy: 0.8692 - val_loss: 1.0347 - val_accuracy: 0.5310
Epoch 34/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3230
- accuracy: 0.8667 - val_loss: 1.0600 - val_accuracy: 0.5295
Epoch 35/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3214
- accuracy: 0.8687 - val_loss: 1.0521 - val_accuracy: 0.5306
Epoch 36/45
196/196 [==============================] - 5s 26ms/step - loss: 0.3189
- accuracy: 0.8721 - val_loss: 1.0310 - val_accuracy: 0.5314
Epoch 37/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3167
- accuracy: 0.8698 - val_loss: 1.0799 - val_accuracy: 0.5286
Epoch 38/45
196/196 [==============================] - 5s 25ms/step - loss: 0.3196
- accuracy: 0.8671 - val_loss: 1.0705 - val_accuracy: 0.5303
Epoch 39/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3113
- accuracy: 0.8714 - val_loss: 1.0695 - val_accuracy: 0.5308
Epoch 40/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3166
- accuracy: 0.8692 - val_loss: 1.0832 - val_accuracy: 0.5303
Epoch 41/45
```

```
196/196 [==============================] - 5s 26ms/step - loss: 0.3128
- accuracy: 0.8715 - val_loss: 1.0871 - val_accuracy: 0.5309
Epoch 42/45
196/196 [==============================] - 5s 23ms/step - loss: 0.3108
- accuracy: 0.8730 - val_loss: 1.0854 - val_accuracy: 0.5318
Epoch 43/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3147
- accuracy: 0.8688 - val_loss: 1.0854 - val_accuracy: 0.5297
Epoch 44/45
196/196 [==============================] - 5s 28ms/step - loss: 0.3146
- accuracy: 0.8694 - val_loss: 1.1202 - val_accuracy: 0.5284
Epoch 45/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3109
- accuracy: 0.8734 - val_loss: 1.1320 - val_accuracy: 0.5271
Epoch 1/45
196/196 [==============================] - 7s 36ms/step - loss: 0.3067
- accuracy: 0.8745 - val_loss: 1.1504 - val_accuracy: 0.5250
Epoch 2/45
196/196 [==============================] - 6s 29ms/step - loss: 0.3084
- accuracy: 0.8726 - val_loss: 1.1219 - val_accuracy: 0.5270
Epoch 3/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3093
- accuracy: 0.8723 - val_loss: 1.1527 - val_accuracy: 0.5244
Epoch 4/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3096
- accuracy: 0.8716 - val_loss: 1.1448 - val_accuracy: 0.5250
Epoch 5/45
196/196 [==============================] - 5s 26ms/step - loss: 0.3048
- accuracy: 0.8744 - val_loss: 1.1070 - val_accuracy: 0.5284
Epoch 6/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3097
- accuracy: 0.8706 - val_loss: 1.1319 - val_accuracy: 0.5265
Epoch 7/45
196/196 [==============================] - 5s 28ms/step - loss: 0.3090
- accuracy: 0.8721 - val_loss: 1.1467 - val_accuracy: 0.5249
Epoch 8/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3081
- accuracy: 0.8722 - val_loss: 1.1457 - val_accuracy: 0.5251
Epoch 9/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3045
- accuracy: 0.8772 - val_loss: 1.1701 - val_accuracy: 0.5246
Epoch 10/45
196/196 [==============================] - 5s 26ms/step - loss: 0.3050
- accuracy: 0.8744 - val_loss: 1.1776 - val_accuracy: 0.5242
Epoch 11/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3026
- accuracy: 0.8736 - val_loss: 1.1694 - val_accuracy: 0.5262
Epoch 12/45
196/196 [==============================] - 4s 23ms/step - loss: 0.3022
```

```
- accuracy: 0.8760 - val_loss: 1.1795 - val_accuracy: 0.5247
Epoch 13/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2997
- accuracy: 0.8774 - val_loss: 1.1828 - val_accuracy: 0.5250
Epoch 14/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3031
- accuracy: 0.8766 - val_loss: 1.1661 - val_accuracy: 0.5258
Epoch 15/45
196/196 [==============================] - 5s 25ms/step - loss: 0.3015
- accuracy: 0.8758 - val_loss: 1.1583 - val_accuracy: 0.5270
Epoch 16/45
196/196 [==============================] - 4s 22ms/step - loss: 0.3046
- accuracy: 0.8762 - val_loss: 1.1529 - val_accuracy: 0.5279
Epoch 17/45
196/196 [==============================] - 5s 24ms/step - loss: 0.3006
- accuracy: 0.8753 - val_loss: 1.1720 - val_accuracy: 0.5263
Epoch 18/45
196/196 [==============================] - 7s 37ms/step - loss: 0.3016
- accuracy: 0.8748 - val_loss: 1.1685 - val_accuracy: 0.5268
Epoch 19/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2983
- accuracy: 0.8768 - val_loss: 1.1854 - val_accuracy: 0.5259
Epoch 20/45
196/196 [==============================] - 5s 28ms/step - loss: 0.2980
- accuracy: 0.8766 - val_loss: 1.1760 - val_accuracy: 0.5269
Epoch 21/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2979
- accuracy: 0.8750 - val_loss: 1.1890 - val_accuracy: 0.5270
Epoch 22/45
196/196 [==============================] - 4s 22ms/step - loss: 0.2991
- accuracy: 0.8780 - val_loss: 1.1996 - val_accuracy: 0.5255
Epoch 23/45
196/196 [==============================] - 5s 27ms/step - loss: 0.2963
- accuracy: 0.8799 - val_loss: 1.2049 - val_accuracy: 0.5267
Epoch 24/45
196/196 [==============================] - 4s 23ms/step - loss: 0.2981
- accuracy: 0.8810 - val_loss: 1.1840 - val_accuracy: 0.5279
Epoch 25/45
196/196 [==============================] - 6s 32ms/step - loss: 0.2955
- accuracy: 0.8789 - val_loss: 1.2037 - val_accuracy: 0.5274
Epoch 26/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2991
- accuracy: 0.8765 - val_loss: 1.2064 - val_accuracy: 0.5272
Epoch 27/45
196/196 [==============================] - 5s 23ms/step - loss: 0.2963
- accuracy: 0.8782 - val_loss: 1.2249 - val_accuracy: 0.5272
Epoch 28/45
196/196 [==============================] - 5s 27ms/step - loss: 0.2953
- accuracy: 0.8800 - val_loss: 1.2369 - val_accuracy: 0.5275
```

```
Epoch 29/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2957
- accuracy: 0.8797 - val_loss: 1.2557 - val_accuracy: 0.5230
Epoch 30/45
196/196 [==============================] - 5s 26ms/step - loss: 0.2948
- accuracy: 0.8790 - val_loss: 1.2531 - val_accuracy: 0.5248
Epoch 31/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2880
- accuracy: 0.8823 - val_loss: 1.2600 - val_accuracy: 0.5243
Epoch 32/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2933
- accuracy: 0.8804 - val_loss: 1.2838 - val_accuracy: 0.5233
Epoch 33/45
196/196 [==============================] - 7s 34ms/step - loss: 0.2923
- accuracy: 0.8816 - val_loss: 1.2905 - val_accuracy: 0.5228
Epoch 34/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2932
- accuracy: 0.8792 - val_loss: 1.3118 - val_accuracy: 0.5200
Epoch 35/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2950
- accuracy: 0.8792 - val_loss: 1.3407 - val_accuracy: 0.5190
Epoch 36/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2944
- accuracy: 0.8786 - val_loss: 1.3338 - val_accuracy: 0.5196
Epoch 37/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2907
- accuracy: 0.8808 - val_loss: 1.3585 - val_accuracy: 0.5202
Epoch 38/45
196/196 [==============================] - 5s 26ms/step - loss: 0.2903
- accuracy: 0.8807 - val_loss: 1.3837 - val_accuracy: 0.5199
Epoch 39/45
196/196 [==============================] - 4s 23ms/step - loss: 0.2918
- accuracy: 0.8796 - val_loss: 1.3841 - val_accuracy: 0.5184
Epoch 40/45
196/196 [==============================] - 5s 23ms/step - loss: 0.2897
- accuracy: 0.8815 - val_loss: 1.4052 - val_accuracy: 0.5189
Epoch 41/45
196/196 [==============================] - 6s 30ms/step - loss: 0.2917
- accuracy: 0.8802 - val_loss: 1.4123 - val_accuracy: 0.5184
Epoch 42/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2931
- accuracy: 0.8813 - val_loss: 1.3909 - val_accuracy: 0.5188
Epoch 43/45
196/196 [==============================] - 5s 25ms/step - loss: 0.2891
- accuracy: 0.8850 - val_loss: 1.4054 - val_accuracy: 0.5184
Epoch 44/45
196/196 [==============================] - 5s 24ms/step - loss: 0.2935
- accuracy: 0.8787 - val_loss: 1.4194 - val_accuracy: 0.5184
Epoch 45/45
```

```
196/196 [==============================] - 5s 25ms/step - loss: 0.2896
- accuracy: 0.8802 - val_loss: 1.4446 - val_accuracy: 0.5180


# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_costs) + 1), adam_training_costs,
label='Adam+dropout', color='r')
plt.plot(range(1, len(sgd_training_costs) + 1), sgd_training_costs,
label='SGD+dropout', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('IMDB BoW feature Logistic Regression')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Retrieve and print the best hyperparameters
best_lg_imdb_hyperparameters = best_trial.params
print("Best Hyperparameters for MLP on IMDB dataset:")
for param_name, param_value in best_lg_imdb_hyperparameters.items():
    print(f"{param_name}: {param_value}")

# Get the best trial and hyperparameters
best_trial_lg_imdb = study.best_trial
```

```
best_optimizer_lg_imdb = best_trial.params['optimizer']
best_lr_lg_imdb = best_trial.params['learning_rate']
best_beta_1_lg_imdb = best_trial.params['beta_1']
best_beta_2_lg_imdb = best_trial.params['beta_2']

Best Hyperparameters for MLP on IMDB dataset:
optimizer: sgd
learning_rate: 0.0036132298044764073
beta_1: 0.26108547972311
beta_2: 0.9986691704338064
decay: 4.216458440951901e-06
decay_steps: 152
l2_reg: 3.6376854898462428e-06

# Plot the cost function (loss) versus epoch using Optuna's
visualization API
optuna.visualization.plot_optimization_history(study_lg_imdb)
```

CIFAR10

CNN for Cifar10

```
from keras.datasets import cifar10
import numpy as np
import tensorflow as tf
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam, SGD
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel
values to [0, 1]

# For RGB images (3 color channels)
X_train = X_train.reshape(-1, 32, 32, 3)
X_test = X_test.reshape(-1, 32, 32, 3)

def objective(trial):

  epochs = 3
  # Create and compile the logistic regression model
  optimizer_name = trial.suggest_categorical('optimizer', ['adam',
'sgd'])
  learning_rate = trial.suggest_float('learning_rate', 1e-3, 1e-2,
log=True)
```

```
  dropout_rate = trial.suggest_float('dropout_rate', 0.5, 0.6)
  beta_1 = trial.suggest_float('beta_1', 0.0, 0.9)  # Vary beta_1 
within [0, 0.9]
  # beta_1 = 0.9
  # beta_2 = 0.99
  beta_2 = trial.suggest_float('beta_2', 0.99, 0.9999)  # Vary beta_2 
within [0.99, 0.9999]
  epsilon = trial.suggest_float('epsilon', 1e-8, 1e-7)
  # epsilon = 1e-8
  momentum = trial.suggest_float('momentum', 0.9, 0.99)
  # momentum = 0.9
  l2_reg = trial.suggest_float('l2_reg', 1e-6, 1e-3, log=True)  # L2 
regularization strength

  # Define the custom learning rate scheduler
  class CustomLRScheduler(tf.keras.callbacks.Callback):
      def on_epoch_begin(self, epoch, logs=None):
          new_lr = 0.001 / np.sqrt(epoch+1)
          K.set_value(self.model.optimizer.lr, new_lr)
          print(f'Epoch {epoch + 1}: Learning Rate = {new_lr}')

  # Binary class
  model = build_cnn_model(optimizer_name, trial, learning_rate,
                          beta_1, beta_2, l2_reg)

  # Create a custom callback to track validation loss
  val_loss_callback = ValidationLoss()
  lr_scheduler_callback = CustomLRScheduler()

  # Train the model
  history = model.fit(X_train, y_train, epochs=epochs, batch_size=128,
                      validation_data=(X_test, y_test), verbose=0,
                      callbacks=[val_loss_callback, 
lr_scheduler_callback])

  # Get the minimum validation loss
  min_val_loss = min(val_loss_callback.validation_losses)

  return min_val_loss

# Create an Optuna study
study_cnn = optuna.create_study(direction='minimize')

# Optimize hyperparameters
# study.optimize(objective, n_trials=45)
study_cnn.optimize(objective, n_trials=2)

best_trial_cnn = study_cnn.best_trial

# Print the best hyperparameters
```

```
print(f'Best Optimizer: {best_trial_cnn.params["optimizer"]}')
print(f'Best Learning Rate: {best_trial_cnn.params["learning_rate"]}')
```

```
[I 2023-10-09 20:08:39,909] A new study created in memory with name:
no-name-13356c26-8fdb-4747-8422-48316e4ca53f
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.SGD`.

Epoch 1: Learning Rate = 0.001
Epoch 2: Learning Rate = 0.0007071067811865475
Epoch 3: Learning Rate = 0.0005773502691896258

[I 2023-10-09 20:11:37,789] Trial 0 finished with value:
1.6748086214065552 and parameters: {'optimizer': 'sgd',
'learning_rate': 0.0018450130840638925, 'dropout_rate':
0.5238892437014541, 'beta_1': 0.50153863887955, 'beta_2':
0.9919482056115264, 'epsilon': 1.6498842549319386e-08, 'momentum':
0.9535901110148401, 'l2_reg': 0.00015335809016012424}. Best is trial 0
with value: 1.6748086214065552.
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.

Epoch 1: Learning Rate = 0.001
Epoch 2: Learning Rate = 0.0007071067811865475
Epoch 3: Learning Rate = 0.0005773502691896258

[I 2023-10-09 20:14:20,980] Trial 1 finished with value:
1.0044238567352295 and parameters: {'optimizer': 'adam',
'learning_rate': 0.0010571724712221452, 'dropout_rate':
0.56598479751596, 'beta_1': 0.10332251746349828, 'beta_2':
0.9948522887265507, 'epsilon': 1.3135048718567216e-08, 'momentum':
0.9110865515384468, 'l2_reg': 0.00047931852666593315}. Best is trial 1
with value: 1.0044238567352295.

Best Optimizer: adam
Best Learning Rate: 0.0010571724712221452
```

```
# Get the best trial and hyperparameters
best_optimizer_cnn = best_trial_cnn.params['optimizer']
best_lr_cnn = best_trial_cnn.params['learning_rate']
best_beta_1_cnn = best_trial_cnn.params['beta_1']
best_beta_2_cnn = best_trial_cnn.params['beta_2']
best_l2_reg_cnn = best_trial_cnn.params['l2_reg']

# Retrieve and print the best hyperparameters
best_cnn_hyperparameters = best_trial_cnn.params
print("Best Hyperparameters for CNN on CIFAR10 dataset:")
```

```
for param_name, param_value in best_cnn_hyperparameters.items():
    print(f"{param_name}: {param_value}")
```

```
Best Hyperparameters for CNN on CIFAR10 dataset:
optimizer: adam
learning_rate: 0.0010571724712221452
dropout_rate: 0.56598479751596
beta_1: 0.10332251746349828
beta_2: 0.9948522887265507
epsilon: 1.3135048718567216e-08
momentum: 0.9110865515384468
l2_reg: 0.00047931852666593315
```

For 3 epochs

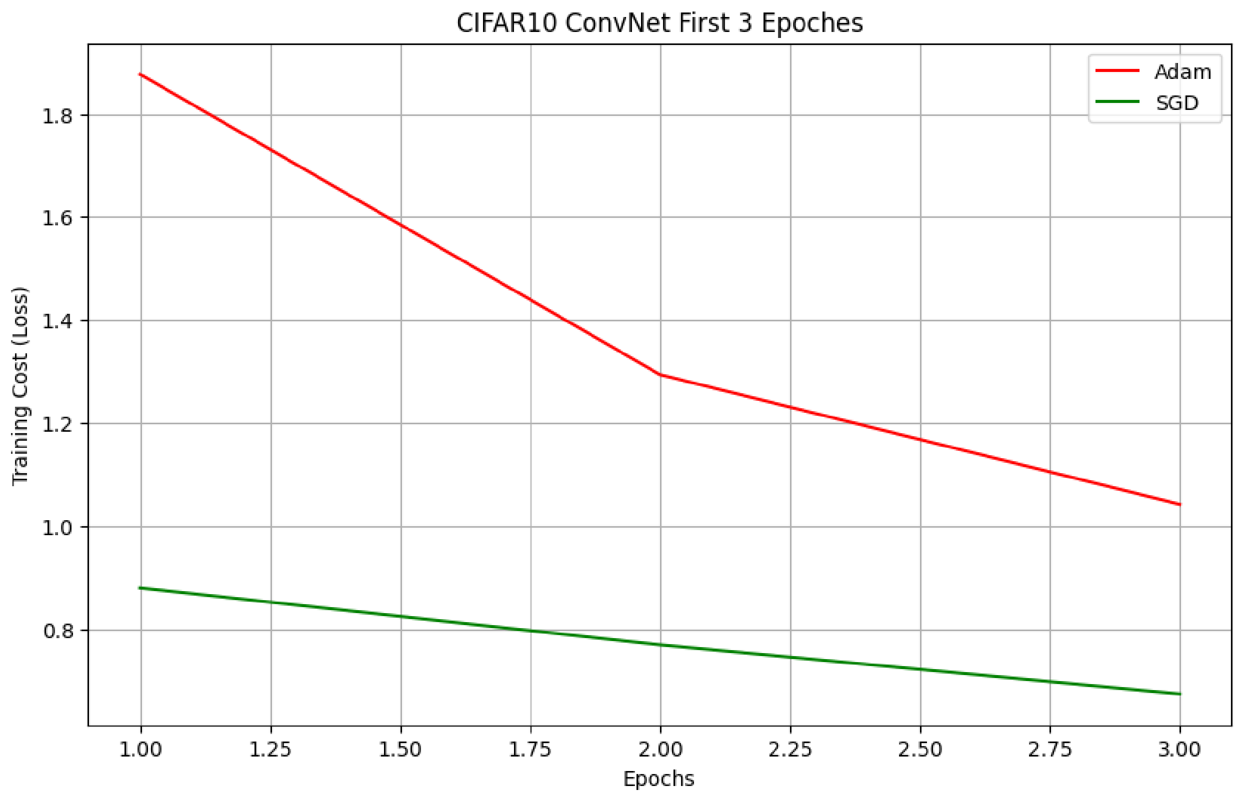
```
if best_optimizer_cnn == 'adam':
  # Build and train the final model with the best hyperparameters
  best_model_cnn = build_cnn_model(best_optimizer_cnn, best_trial_cnn, 
best_lr_cnn,
                                          best_beta_1_cnn, 
best_beta_2_cnn, best_l2_reg_cnn)
  history_cnn = best_model_cnn.fit(X_train, y_train, epochs=3, 
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  sgd_model_cnn = build_cnn_model("sgd", best_trial_cnn, best_lr_cnn,
                                          best_beta_1_cnn, 
best_beta_2_cnn, best_l2_reg_cnn)

  sgd_history_cnn = best_model_cnn.fit(X_train, y_train, epochs=3, 
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_training_cnn_costs = history_cnn.history['loss']
  sgd_training_cnn_costs = sgd_history_cnn.history['loss']

elif best_optimizer_cnn == 'sgd':
  # Build and train the final model with the best hyperparameters
  best_model_cnn = build_cnn_model(best_optimizer_cnn, best_trial_cnn, 
best_lr_cnn,
                                          best_beta_1_cnn, 
best_beta_2_cnn, best_l2_reg_cnn)
  history_cnn = best_model_cnn.fit(X_train, y_train, epochs=3, 
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_model_cnn = build_cnn_model("adam", best_trial_cnn, 
best_lr_cnn,
                                          best_beta_1_cnn, 
best_beta_2_cnn, best_l2_reg_cnn)
```

```
  adam_history_cnn = best_model_cnn.fit(X_train, y_train, epochs=3,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  sgd_training_cnn_costs = history_cnn.history['loss']
  adam_training_cnn_costs = adam_history_cnn.history['loss']
else:
    raise ValueError("Invalid optimizer name")
```

```
WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.

Epoch 1/3
391/391 [==============================] - 57s 145ms/step - loss:
1.8768 - accuracy: 0.3127 - val_loss: 1.3480 - val_accuracy: 0.5118
Epoch 2/3
391/391 [==============================] - 59s 152ms/step - loss:
1.2938 - accuracy: 0.5381 - val_loss: 1.0969 - val_accuracy: 0.6153
Epoch 3/3
391/391 [==============================] - 55s 141ms/step - loss:
1.0421 - accuracy: 0.6346 - val_loss: 0.9784 - val_accuracy: 0.6554

WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.SGD`.

Epoch 1/3
391/391 [==============================] - 55s 141ms/step - loss:
0.8799 - accuracy: 0.6916 - val_loss: 0.9051 - val_accuracy: 0.6874
Epoch 2/3
391/391 [==============================] - 55s 140ms/step - loss:
0.7693 - accuracy: 0.7335 - val_loss: 0.8920 - val_accuracy: 0.6938
Epoch 3/3
391/391 [==============================] - 55s 139ms/step - loss:
0.6741 - accuracy: 0.7650 - val_loss: 0.8271 - val_accuracy: 0.7160
```

```
# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_cnn_costs) + 1),
adam_training_cnn_costs, label='Adam', color='r')
plt.plot(range(1, len(sgd_training_cnn_costs) + 1),
sgd_training_cnn_costs, label='SGD', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('CIFAR10 ConvNet First 3 Epoches')
plt.legend()
```

```
plt.grid(True)
plt.show()
```



45 epochs

```
if best_optimizer_cnn == 'adam':
  # Build and train the final model with the best hyperparameters
  best_model_cnn = build_cnn_model(best_optimizer_cnn, best_trial_cnn,
best_lr_cnn,
                                          best_beta_1_cnn,
best_beta_2_cnn, best_l2_reg_cnn)
  history_cnn = best_model_cnn.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  sgd_model_cnn = build_cnn_model("sgd", best_trial_cnn, best_lr_cnn,
                                          best_beta_1_cnn,
best_beta_2_cnn, best_l2_reg_cnn)

  sgd_history_cnn = best_model_cnn.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_training_cnn_costs = history_cnn.history['loss']
  sgd_training_cnn_costs = sgd_history_cnn.history['loss']
```

```
elif best_optimizer_cnn == 'sgd':
  # Build and train the final model with the best hyperparameters
  best_model_cnn = build_cnn_model(best_optimizer_cnn, best_trial_cnn,
best_lr_cnn,
                                          best_beta_1_cnn,
best_beta_2_cnn, best_l2_reg_cnn)
  history_cnn = best_model_cnn.fit(X_train, y_train, epochs=45,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  adam_model_cnn = build_cnn_model("adam", best_trial_cnn,
best_lr_cnn,
                                          best_beta_1_cnn,
best_beta_2_cnn, best_l2_reg_cnn)

  adam_history_cnn = best_model_cnn.fit(X_train, y_train, epochs=3,
batch_size=128,
                        validation_data=(X_test, y_test), verbose=1)

  sgd_training_cnn_costs = history_cnn.history['loss']
  adam_training_cnn_costs = adam_history_cnn.history['loss']
else:
    raise ValueError("Invalid optimizer name")

WARNING:absl:At this time, the v2.11+ optimizer
`tf.keras.optimizers.Adam` runs slowly on M1/M2 Macs, please use the
legacy Keras optimizer instead, located at
`tf.keras.optimizers.legacy.Adam`.

Epoch 1/45
391/391 [==============================] - 55s 141ms/step - loss:
1.8288 - accuracy: 0.3274 - val_loss: 1.4979 - val_accuracy: 0.4682
Epoch 2/45
391/391 [==============================] - 55s 141ms/step - loss:
1.2645 - accuracy: 0.5503 - val_loss: 1.1576 - val_accuracy: 0.5945
Epoch 3/45
391/391 [==============================] - 55s 141ms/step - loss:
1.0255 - accuracy: 0.6408 - val_loss: 1.0077 - val_accuracy: 0.6498
Epoch 4/45
391/391 [==============================] - 60s 155ms/step - loss:
0.8688 - accuracy: 0.6951 - val_loss: 0.9956 - val_accuracy: 0.6517
Epoch 5/45
391/391 [==============================] - 55s 141ms/step - loss:
0.7569 - accuracy: 0.7356 - val_loss: 0.9022 - val_accuracy: 0.6953
Epoch 6/45
391/391 [==============================] - 55s 140ms/step - loss:
0.6695 - accuracy: 0.7657 - val_loss: 0.9288 - val_accuracy: 0.6891
Epoch 7/45
391/391 [==============================] - 55s 141ms/step - loss:
```

```
0.5827 - accuracy: 0.7954 - val_loss: 0.8624 - val_accuracy: 0.7210
Epoch 8/45
391/391 [==============================] - 55s 141ms/step - loss:
0.5140 - accuracy: 0.8198 - val_loss: 0.8682 - val_accuracy: 0.7177
Epoch 9/45
391/391 [==============================] - 55s 140ms/step - loss:
0.4502 - accuracy: 0.8423 - val_loss: 0.9267 - val_accuracy: 0.7259
Epoch 10/45
391/391 [==============================] - 60s 154ms/step - loss:
0.3897 - accuracy: 0.8612 - val_loss: 0.9637 - val_accuracy: 0.7133
Epoch 11/45
391/391 [==============================] - 55s 140ms/step - loss:
0.3406 - accuracy: 0.8797 - val_loss: 1.0699 - val_accuracy: 0.7182
Epoch 12/45
391/391 [==============================] - 309s 791ms/step - loss:
0.3031 - accuracy: 0.8922 - val_loss: 1.0073 - val_accuracy: 0.7293
Epoch 13/45
391/391 [==============================] - 55s 141ms/step - loss:
0.2686 - accuracy: 0.9063 - val_loss: 1.0592 - val_accuracy: 0.7363
Epoch 14/45
391/391 [==============================] - 54s 139ms/step - loss:
0.2373 - accuracy: 0.9173 - val_loss: 1.1111 - val_accuracy: 0.7233
Epoch 15/45
391/391 [==============================] - 54s 138ms/step - loss:
0.2217 - accuracy: 0.9216 - val_loss: 1.1854 - val_accuracy: 0.7374
Epoch 16/45
391/391 [==============================] - 60s 153ms/step - loss:
0.2040 - accuracy: 0.9291 - val_loss: 1.2607 - val_accuracy: 0.7208
Epoch 17/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1760 - accuracy: 0.9387 - val_loss: 1.3464 - val_accuracy: 0.7295
Epoch 18/45
391/391 [==============================] - 56s 142ms/step - loss:
0.1735 - accuracy: 0.9392 - val_loss: 1.4662 - val_accuracy: 0.7206
Epoch 19/45
391/391 [==============================] - 54s 139ms/step - loss:
0.1644 - accuracy: 0.9424 - val_loss: 1.3653 - val_accuracy: 0.7290
Epoch 20/45
391/391 [==============================] - 54s 139ms/step - loss:
0.1550 - accuracy: 0.9477 - val_loss: 1.6316 - val_accuracy: 0.6948
Epoch 21/45
391/391 [==============================] - 55s 141ms/step - loss:
0.1473 - accuracy: 0.9492 - val_loss: 1.5459 - val_accuracy: 0.7316
Epoch 22/45
391/391 [==============================] - 61s 155ms/step - loss:
0.1459 - accuracy: 0.9512 - val_loss: 1.5747 - val_accuracy: 0.7294
Epoch 23/45
391/391 [==============================] - 56s 142ms/step - loss:
0.1445 - accuracy: 0.9527 - val_loss: 1.5676 - val_accuracy: 0.7137
```

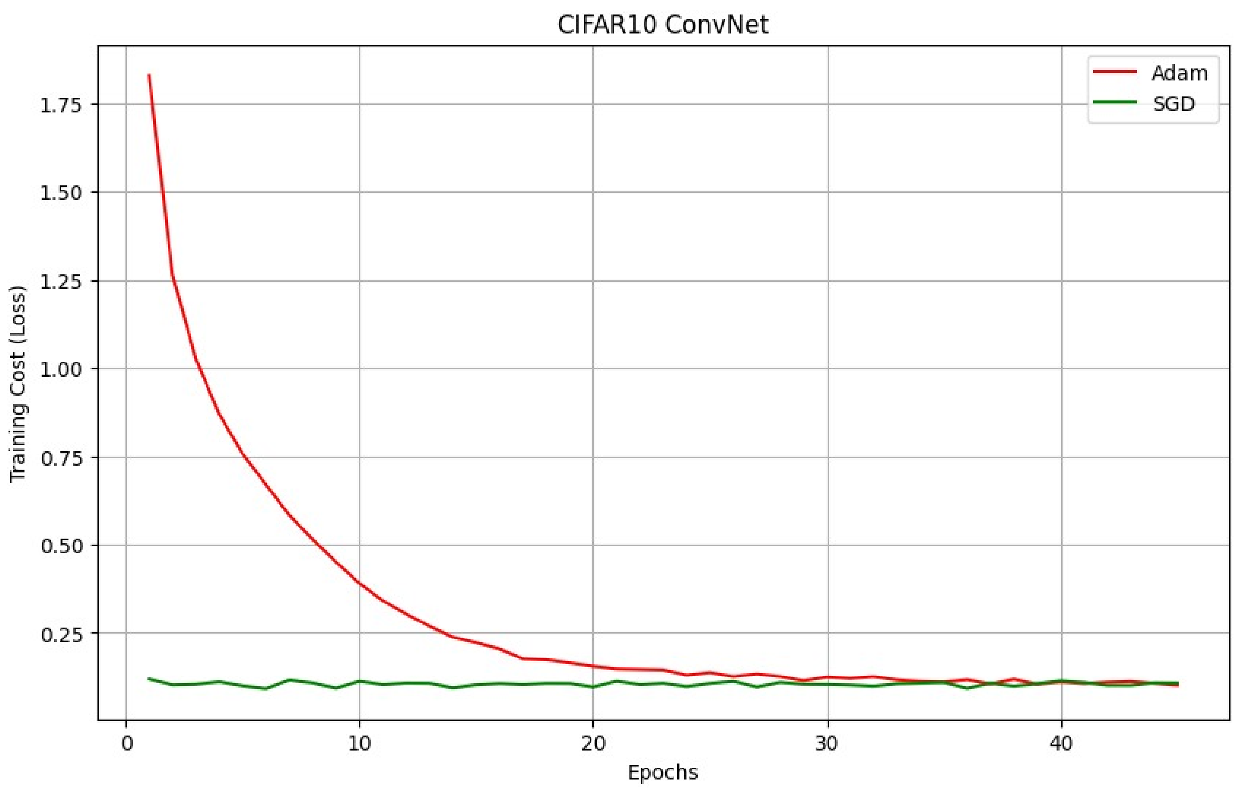
```
Epoch 24/45
391/391 [==============================] - 54s 138ms/step - loss:
0.1299 - accuracy: 0.9562 - val_loss: 1.7822 - val_accuracy: 0.7332
Epoch 25/45
391/391 [==============================] - 55s 141ms/step - loss:
0.1371 - accuracy: 0.9543 - val_loss: 1.5244 - val_accuracy: 0.7298
Epoch 26/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1258 - accuracy: 0.9588 - val_loss: 1.7113 - val_accuracy: 0.7349
Epoch 27/45
391/391 [==============================] - 56s 142ms/step - loss:
0.1332 - accuracy: 0.9567 - val_loss: 1.7163 - val_accuracy: 0.7245
Epoch 28/45
391/391 [==============================] - 58s 149ms/step - loss:
0.1259 - accuracy: 0.9595 - val_loss: 1.6818 - val_accuracy: 0.7298
Epoch 29/45
391/391 [==============================] - 53s 134ms/step - loss:
0.1141 - accuracy: 0.9628 - val_loss: 1.7846 - val_accuracy: 0.7274
Epoch 30/45
391/391 [==============================] - 52s 133ms/step - loss:
0.1244 - accuracy: 0.9604 - val_loss: 1.8789 - val_accuracy: 0.7345
Epoch 31/45
391/391 [==============================] - 52s 133ms/step - loss:
0.1209 - accuracy: 0.9621 - val_loss: 2.1454 - val_accuracy: 0.7150
Epoch 32/45
391/391 [==============================] - 53s 134ms/step - loss:
0.1255 - accuracy: 0.9622 - val_loss: 1.8166 - val_accuracy: 0.7315
Epoch 33/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1165 - accuracy: 0.9637 - val_loss: 1.9884 - val_accuracy: 0.7267
Epoch 34/45
391/391 [==============================] - 59s 152ms/step - loss:
0.1120 - accuracy: 0.9654 - val_loss: 2.0016 - val_accuracy: 0.7184
Epoch 35/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1103 - accuracy: 0.9656 - val_loss: 2.0245 - val_accuracy: 0.7340
Epoch 36/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1168 - accuracy: 0.9639 - val_loss: 1.9585 - val_accuracy: 0.7285
Epoch 37/45
391/391 [==============================] - 54s 139ms/step - loss:
0.1046 - accuracy: 0.9670 - val_loss: 2.2034 - val_accuracy: 0.7287
Epoch 38/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1182 - accuracy: 0.9648 - val_loss: 2.3266 - val_accuracy: 0.7080
Epoch 39/45
391/391 [==============================] - 56s 144ms/step - loss:
0.1039 - accuracy: 0.9688 - val_loss: 2.2347 - val_accuracy: 0.7285
Epoch 40/45
```

```
391/391 [==============================] - 58s 149ms/step - loss: 
0.1102 - accuracy: 0.9678 - val_loss: 2.1457 - val_accuracy: 0.7325
Epoch 41/45
391/391 [==============================] - 55s 141ms/step - loss: 
0.1057 - accuracy: 0.9689 - val_loss: 2.7399 - val_accuracy: 0.7105
Epoch 42/45
391/391 [==============================] - 56s 142ms/step - loss: 
0.1097 - accuracy: 0.9674 - val_loss: 2.5885 - val_accuracy: 0.7160
Epoch 43/45
391/391 [==============================] - 53s 137ms/step - loss: 
0.1119 - accuracy: 0.9673 - val_loss: 2.2469 - val_accuracy: 0.7316
Epoch 44/45
391/391 [==============================] - 55s 140ms/step - loss: 
0.1064 - accuracy: 0.9687 - val_loss: 2.2302 - val_accuracy: 0.7324
Epoch 45/45
391/391 [==============================] - 55s 140ms/step - loss: 
0.1013 - accuracy: 0.9700 - val_loss: 2.4042 - val_accuracy: 0.7248

WARNING:absl:At this time, the v2.11+ optimizer 
`tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the 
legacy Keras optimizer instead, located at 
`tf.keras.optimizers.legacy.SGD`.

Epoch 1/45
391/391 [==============================] - 59s 151ms/step - loss: 
0.1187 - accuracy: 0.9669 - val_loss: 2.3854 - val_accuracy: 0.7371
Epoch 2/45
391/391 [==============================] - 56s 144ms/step - loss: 
0.1022 - accuracy: 0.9708 - val_loss: 2.3934 - val_accuracy: 0.7103
Epoch 3/45
391/391 [==============================] - 57s 146ms/step - loss: 
0.1039 - accuracy: 0.9703 - val_loss: 2.3042 - val_accuracy: 0.7291
Epoch 4/45
391/391 [==============================] - 58s 150ms/step - loss: 
0.1107 - accuracy: 0.9687 - val_loss: 2.5077 - val_accuracy: 0.7238
Epoch 5/45
391/391 [==============================] - 56s 143ms/step - loss: 
0.0995 - accuracy: 0.9720 - val_loss: 2.4923 - val_accuracy: 0.7359
Epoch 6/45
391/391 [==============================] - 56s 144ms/step - loss: 
0.0918 - accuracy: 0.9734 - val_loss: 2.8379 - val_accuracy: 0.7035
Epoch 7/45
391/391 [==============================] - 64s 163ms/step - loss: 
0.1158 - accuracy: 0.9677 - val_loss: 2.7222 - val_accuracy: 0.7289
Epoch 8/45
391/391 [==============================] - 57s 146ms/step - loss: 
0.1074 - accuracy: 0.9709 - val_loss: 2.3702 - val_accuracy: 0.7294
Epoch 9/45
391/391 [==============================] - 56s 144ms/step - loss: 
0.0931 - accuracy: 0.9738 - val_loss: 2.4841 - val_accuracy: 0.7362
```

```
Epoch 10/45
391/391 [==============================] - 56s 144ms/step - loss: 
0.1123 - accuracy: 0.9690 - val_loss: 2.6646 - val_accuracy: 0.7265
Epoch 11/45
391/391 [==============================] - 56s 143ms/step - loss: 
0.1030 - accuracy: 0.9711 - val_loss: 2.7476 - val_accuracy: 0.7401
Epoch 12/45
391/391 [==============================] - 55s 141ms/step - loss: 
0.1072 - accuracy: 0.9701 - val_loss: 2.8117 - val_accuracy: 0.7289
Epoch 13/45
391/391 [==============================] - 59s 150ms/step - loss: 
0.1067 - accuracy: 0.9717 - val_loss: 2.6925 - val_accuracy: 0.7296
Epoch 14/45
391/391 [==============================] - 57s 145ms/step - loss: 
0.0939 - accuracy: 0.9732 - val_loss: 2.6888 - val_accuracy: 0.7382
Epoch 15/45
391/391 [==============================] - 55s 142ms/step - loss: 
0.1025 - accuracy: 0.9723 - val_loss: 2.7644 - val_accuracy: 0.7177
Epoch 16/45
391/391 [==============================] - 55s 140ms/step - loss: 
0.1060 - accuracy: 0.9711 - val_loss: 2.6766 - val_accuracy: 0.7287
Epoch 17/45
391/391 [==============================] - 55s 140ms/step - loss: 
0.1032 - accuracy: 0.9728 - val_loss: 2.7981 - val_accuracy: 0.7283
Epoch 18/45
391/391 [==============================] - 55s 140ms/step - loss: 
0.1063 - accuracy: 0.9720 - val_loss: 2.9637 - val_accuracy: 0.7290
Epoch 19/45
391/391 [==============================] - 57s 147ms/step - loss: 
0.1060 - accuracy: 0.9729 - val_loss: 2.8232 - val_accuracy: 0.7389
Epoch 20/45
391/391 [==============================] - 57s 146ms/step - loss: 
0.0963 - accuracy: 0.9738 - val_loss: 2.9519 - val_accuracy: 0.7343
Epoch 21/45
391/391 [==============================] - 54s 139ms/step - loss: 
0.1125 - accuracy: 0.9722 - val_loss: 2.7521 - val_accuracy: 0.7313
Epoch 22/45
391/391 [==============================] - 54s 138ms/step - loss: 
0.1030 - accuracy: 0.9731 - val_loss: 2.9202 - val_accuracy: 0.7234
Epoch 23/45
391/391 [==============================] - 54s 138ms/step - loss: 
0.1066 - accuracy: 0.9723 - val_loss: 3.0713 - val_accuracy: 0.7329
Epoch 24/45
391/391 [==============================] - 54s 138ms/step - loss: 
0.0975 - accuracy: 0.9739 - val_loss: 3.0304 - val_accuracy: 0.7415
Epoch 25/45
391/391 [==============================] - 57s 145ms/step - loss: 
0.1062 - accuracy: 0.9737 - val_loss: 3.1781 - val_accuracy: 0.7266
Epoch 26/45
```

```
391/391 [==============================] - 57s 146ms/step - loss:
0.1122 - accuracy: 0.9720 - val_loss: 2.9012 - val_accuracy: 0.7355
Epoch 27/45
391/391 [==============================] - 55s 140ms/step - loss:
0.0962 - accuracy: 0.9757 - val_loss: 2.9951 - val_accuracy: 0.7361
Epoch 28/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1090 - accuracy: 0.9727 - val_loss: 3.0639 - val_accuracy: 0.7299
Epoch 29/45
391/391 [==============================] - 54s 138ms/step - loss:
0.1040 - accuracy: 0.9737 - val_loss: 3.1734 - val_accuracy: 0.7357
Epoch 30/45
391/391 [==============================] - 55s 141ms/step - loss:
0.1036 - accuracy: 0.9743 - val_loss: 3.0421 - val_accuracy: 0.7343
Epoch 31/45
391/391 [==============================] - 57s 145ms/step - loss:
0.1017 - accuracy: 0.9745 - val_loss: 3.2885 - val_accuracy: 0.6923
Epoch 32/45
391/391 [==============================] - 57s 146ms/step - loss:
0.0984 - accuracy: 0.9759 - val_loss: 3.1606 - val_accuracy: 0.7402
Epoch 33/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1054 - accuracy: 0.9750 - val_loss: 3.2601 - val_accuracy: 0.7254
Epoch 34/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1069 - accuracy: 0.9741 - val_loss: 3.3834 - val_accuracy: 0.7072
Epoch 35/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1089 - accuracy: 0.9743 - val_loss: 4.0614 - val_accuracy: 0.7077
Epoch 36/45
391/391 [==============================] - 54s 138ms/step - loss:
0.0927 - accuracy: 0.9774 - val_loss: 3.3650 - val_accuracy: 0.7340
Epoch 37/45
391/391 [==============================] - 58s 149ms/step - loss:
0.1068 - accuracy: 0.9754 - val_loss: 3.1501 - val_accuracy: 0.7366
Epoch 38/45
391/391 [==============================] - 57s 145ms/step - loss:
0.0984 - accuracy: 0.9759 - val_loss: 3.5800 - val_accuracy: 0.7222
Epoch 39/45
391/391 [==============================] - 55s 141ms/step - loss:
0.1055 - accuracy: 0.9743 - val_loss: 3.3637 - val_accuracy: 0.7355
Epoch 40/45
391/391 [==============================] - 54s 139ms/step - loss:
0.1138 - accuracy: 0.9746 - val_loss: 3.5807 - val_accuracy: 0.7357
Epoch 41/45
391/391 [==============================] - 55s 140ms/step - loss:
0.1088 - accuracy: 0.9741 - val_loss: 3.3488 - val_accuracy: 0.7403
Epoch 42/45
391/391 [==============================] - 54s 138ms/step - loss:
```

```
0.1010 - accuracy: 0.9758 - val_loss: 3.7869 - val_accuracy: 0.7356
Epoch 43/45
391/391 [==============================] - 57s 146ms/step - loss:
0.1009 - accuracy: 0.9766 - val_loss: 3.4026 - val_accuracy: 0.7375
Epoch 44/45
391/391 [==============================] - 56s 144ms/step - loss:
0.1078 - accuracy: 0.9748 - val_loss: 3.6617 - val_accuracy: 0.7201
Epoch 45/45
391/391 [==============================] - 54s 138ms/step - loss:
0.1072 - accuracy: 0.9763 - val_loss: 3.6586 - val_accuracy: 0.7252

# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_cnn_costs) + 1),
adam_training_cnn_costs, label='Adam', color='r')
plt.plot(range(1, len(sgd_training_cnn_costs) + 1),
sgd_training_cnn_costs, label='SGD', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('CIFAR10 ConvNet')
plt.legend()
plt.grid(True)
plt.show()
```



Observation - Compare the results

Adam is one of the most popular methods for neural network training and usually converges faster than vanilla SGD and SGD with momentum, but generalizes worse. Later, researchers found that welltuned SGD and SGD with momentum outperform Adam in both training error and test error.

As seen above in the plots and when trained using Optuna hyper-parapmeter tuning, the best optimizer was reported to be "SGD" for Logistic regression on the dataset IMDB, MLP oon MNIST as well as CNN on the CIFAR10 dataset. So we saw in the Training Cost(Training loss) vs Epochs graphs for different dataset and model combinations, that although simple vanilla SGD's performance is inferior to that of Adam's, however, when SGD was well tuned with momentum outperformed Adam in both training error and test error.

Thus the advantages of Adam, compared to SGD, are considered to be the relative insensitivity to hyperparameters and rapid initial progress in training. In Adam the effective stepsize decay rate of $1/rootoverT$ did not necessarily diminished even if the step-size and learning rate kept decreasing thus causing some divergence.

Hence, the hyperparameter optimization made the empirical results of the paper less relevant.

MISC: Simple implementation without best optimizer or fine tuning

direct values for hyperparameters used as mentioned in the paper

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss, accuracy_score
from sklearn.model_selection import train_test_split
from keras.datasets import mnist

# from sklearn.linear_model import SGDClassifier

from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.optimizers import Adam, SGD
```

MNIST

```
# Set hyperparameters
learning_rate = 0.001
beta1 = 0.9
beta2 = 0.999
epsilon = 1e-8
num_iterations = 1000

# Loading the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
# Preprocess the data
X_train = X_train / 255.0  # Normalize pixel values to the range [0,
1]
X_test = X_test / 255.0

# Flatten the images for logistic regression
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# function to build and train a logistic regression model
def build_logistic_regression_model(optimizer_name):
    model = Sequential()
    model.add(Flatten(input_shape=(784,)))
    model.add(Dense(10, activation='softmax'))

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=0.001, beta_1=0.9,
beta_2=0.999, epsilon=1e-8)
    elif optimizer_name == 'sgd':
        optimizer = SGD(learning_rate=0.001)
    else:
        raise ValueError("Invalid optimizer name")

    model.compile(optimizer=optimizer,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return model

# Train logistic regression models with both Adam and SGD
adam_model = build_logistic_regression_model('adam')
sgd_model = build_logistic_regression_model('sgd')

adam_history = adam_model.fit(X_train, y_train, epochs=10,
validation_data=(X_test, y_test), verbose=0)
sgd_history = sgd_model.fit(X_train, y_train, epochs=10,
validation_data=(X_test, y_test), verbose=0)

# Extract training cost (loss) values from history
adam_training_costs = adam_history.history['loss']
sgd_training_costs = sgd_history.history['loss']

# Plot training cost vs. epochs (iterations over the dataset) for both
optimizers
plt.figure(figsize=(7, 5))
plt.plot(range(1, len(adam_training_costs) + 1), adam_training_costs,
label='Adam', color='r')
plt.plot(range(1, len(sgd_training_costs) + 1), sgd_training_costs,
label='SGD', color='g')
plt.xlabel('Epochs')
```

```
plt.ylabel('Training Cost (Loss)')
plt.title('Training Cost vs. Epochs for Adam and SGD')
plt.legend()
plt.grid(True)
plt.show()

import numpy as np
import tensorflow as tf
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam, SGD
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel
values to [0, 1]

# For RGB images (3 color channels)
X_train = X_train.reshape(-1, 32, 32, 3)
X_test = X_test.reshape(-1, 32, 32, 3)

# Create CNN model
def build_cnn_model(optimizer_name):
    # Define the CNN architecture
    cnn_model = Sequential([
        Conv2D(32, (5, 5), activation='relu', input_shape=(32, 32, 3),
padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Conv2D(64, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Conv2D(128, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Flatten(),
        Dense(1000, activation='relu'),
        Dropout(0.5),
        Dense(10, activation='softmax')  # 10 classes in CIFAR-10
    ])

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=0.01, beta_1=0.9, beta_2=0.999,
epsilon=1e-8)
    elif optimizer_name == 'sgd':
        optimizer = SGD(learning_rate=0.0001, momentum=0.9,
nesterov=True)
    else:
        raise ValueError("Invalid optimizer name")

    # Compile the model with Adam optimizer
```

```
    # adam_optimizer = Adam()
    cnn_model.compile(optimizer=optimizer,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return cnn_model


# Train CNN models with both Adam and SGD
adam_model = build_cnn_model('adam')
sgd_model = build_cnn_model('sgd')

adam_history = adam_model.fit(X_train, y_train, epochs=3, batch_size =
128, validation_data=(X_test, y_test), verbose=0)
sgd_history = sgd_model.fit(X_train, y_train, epochs=3,
batch_size=128, validation_data=(X_test, y_test), verbose=0)

# Extract training cost (loss) values from history
adam_training_costs = adam_history.history['loss']
sgd_training_costs = sgd_history.history['loss']

# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_costs) + 1), adam_training_costs,
label='Adam+dropout', color='r')
plt.plot(range(1, len(sgd_training_costs) + 1), sgd_training_costs,
label='SGD+dropout', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('CIFAR10 ConvNet First 3 Epoches')
plt.legend()
plt.grid(True)
plt.show()

import numpy as np
import tensorflow as tf
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.optimizers import Adam, SGD
import matplotlib.pyplot as plt

# Load and preprocess the CIFAR-10 dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0  # Normalize pixel
values to [0, 1]

# For RGB images (3 color channels)
X_train = X_train.reshape(-1, 32, 32, 3)
X_test = X_test.reshape(-1, 32, 32, 3)
```

```
# Create CNN model
def build_cnn_model(optimizer_name):
    # Define the CNN architecture
    cnn_model = Sequential([
        Conv2D(32, (5, 5), activation='relu', input_shape=(32, 32, 3),
padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Conv2D(64, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Conv2D(128, (5, 5), activation='relu', padding='same'),
        MaxPooling2D(pool_size=(3, 3), strides=2),
        Flatten(),
        Dense(1000, activation='relu'),
        Dense(10, activation='softmax')  # 10 classes in CIFAR-10
    ])

    # Create a custom learning rate schedule that decreases as
1/sqrt(t)
    initial_learning_rate = 0.001
    decay_steps = len(X_train) // 128  # Adjust as needed
    lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
        initial_learning_rate, decay_steps=decay_steps,
decay_rate=1/np.sqrt(45.0), staircase=False
    )

    if optimizer_name == 'adam':
        optimizer = Adam(learning_rate=lr_schedule, beta_1=0.9,
beta_2=0.999, epsilon=1e-8)
    elif optimizer_name == 'sgd':
        optimizer = SGD(learning_rate=0.0001, momentum=0.9,
nesterov=True)
    else:
        raise ValueError("Invalid optimizer name")

    # Compile the model with Adam optimizer
    # adam_optimizer = Adam()
    cnn_model.compile(optimizer=optimizer,
loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    return cnn_model


# Train CNN models with both Adam and SGD
adam_model = build_cnn_model('adam')
sgd_model = build_cnn_model('sgd')

adam_history = adam_model.fit(X_train, y_train, epochs=45, batch_size
= 128, validation_data=(X_test, y_test), verbose=0)
sgd_history = sgd_model.fit(X_train, y_train, epochs=45,
batch_size=128, validation_data=(X_test, y_test), verbose=0)
```

```python
# Extract training cost (loss) values from history
adam_training_costs = adam_history.history['loss']
sgd_training_costs = sgd_history.history['loss']

# Plot training cost vs. epochs for both optimizers
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(adam_training_costs) + 1), adam_training_costs,
label='Adam+dropout', color='r')
plt.plot(range(1, len(sgd_training_costs) + 1), sgd_training_costs,
label='SGD+dropout', color='g')
plt.xlabel('Epochs')
plt.ylabel('Training Cost (Loss)')
plt.title('CIFAR10 ConvNet')
plt.legend()
plt.grid(True)
plt.show()
```

IMDB dataset

Log_reg from scratch

```python
# # Load your IMDB BoW features and corresponding labels (X, y)

# # Add a bias term to X
# # X_bias = np.c_[np.ones((X.shape[0], 1)), X]  # Assuming X is your
feature matrix

# X = x_train_bow
# y = y_train

# # Convert data to TensorFlow tensors
# X_tensor = tf.constant(X, dtype=tf.float32)
# y_tensor = tf.constant(y, dtype=tf.float32)

# # Initialize model parameters (weights and bias)
# num_features = X.shape[1]
# theta = tf.Variable(tf.zeros((num_features, 1), dtype=tf.float32))
# bias = tf.Variable(0.0, dtype=tf.float32)
# # Convert data to TensorFlow tensors
# X_tensor = tf.constant(X_bias, dtype=tf.float32)
# y_tensor = tf.constant(y, dtype=tf.float32)

# # Initialize model parameters (weights and bias)
# num_features = X_bias.shape[1]
# theta = tf.Variable(tf.zeros((num_features, 1), dtype=tf.float32))

# # Set hyperparameters
# learning_rate = 0.001
```

```
# beta1 = 0.9
# beta2 = 0.999
# epsilon = 1e-8
# num_iterations = 1000

# # Define the logistic regression model
# def logistic_regression_model(X):
#     logits = tf.matmul(X, theta)
#     return tf.sigmoid(logits)

# # Define the cost function (binary cross-entropy)
# def compute_cost(y_true, y_pred):
#     return -tf.reduce_mean(y_true * tf.math.log(y_pred) + (1 -
y_true) * tf.math.log(1 - y_pred))

# # Use the Adam optimizer
# optimizer = tf.optimizers.Adam(learning_rate=learning_rate,
beta_1=beta1, beta_2=beta2, epsilon=epsilon)

# # Lists to store costs for plotting
# costs = []

# # Training loop
# for i in range(num_iterations):
#     with tf.GradientTape() as tape:
#         y_pred = logistic_regression_model(X_tensor)
#         cost = compute_cost(y_tensor, y_pred)

#     gradients = tape.gradient(cost, [theta])
#     optimizer.apply_gradients(zip(gradients, [theta]))

#     costs.append(cost.numpy())

# # Plot cost vs. iterations
# plot.figure(figsize=(10, 6))
# plot.plot(range(1, num_iterations + 1), costs, color='b')
# plot.xlabel('Iterations')
# plot.ylabel('Cost')
# plot.title('Training Cost vs. Iterations')
# plot.grid(True)
# plot.show()
```

Simple Adam optimiser with Log Reg implementation

```
import numpy as np
import tensorflow as tf
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
import matplotlib.pyplot as plt
```

```
# Load the MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Flatten the images (convert 28x28 images to 784-dimensional vectors)
X_train = X_train.reshape(X_train.shape[0], -1)
X_test = X_test.reshape(X_test.shape[0], -1)

# Normalize pixel values to be between 0 and 1
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255

# Define a simple logistic regression model
model = Sequential()
model.add(Dense(10, input_dim=784, activation='softmax'))  # 10
classes for MNIST

# Compile the model
model.compile(loss='sparse_categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=128,
validation_split=0.2)

# Extract training cost (loss) values from history
training_costs = history.history['loss']

# Plot training cost vs. epochs
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(training_costs) + 1), training_costs, color='b')
plt.xlabel('Epochs')
plt.ylabel('Training Loss')
plt.title('Training Loss vs. Epochs')
plt.grid(True)
plt.show()

# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss:.4f}")
print(f"Test Accuracy: {test_accuracy:.4f}")
```