**Lab Assignment 6**

**Neural Network & Deep Learning**

**Learning rate and optimizers**

**Step 1**: Load the wheat seed dataset into your notebook

**Step 2**: Pre-processing and prepare the data for giving to the neural network.

1. Encoding the species names using label encoder.
2. Normalize the features.
3. Split it into train and validate.

**Step 3**: Building the sequential neural network model.

1. You may choose the layers.
2. Use appropriate activation and loss functions.

**Step 4**: Compile and fit the model to the training dataset. Use validation also. Use SGD as optimizer.

**Step 5**: Use learning rates as (0.1, 0.01, 0.001, 0.0001) and train the model. Plot the training and validation accuracy curves and note the best learning rate.

**Step 6**: Use the best learning rate and add momentum to it. Use different momentum values as 0, 0.5, 0.9, 0.99. Train the model and note the best momentum value.

**Step 7:** Add a decay parameter to the optimizer. Use decay values as 1E-1, 1E-2, 1E-3, 1E-4. Note the best results.

**Step 8:** Train the model using Adagrad, adam and rmsprop and conclude which works best.

**Step 9:** Finally infer which optimizer and learning rate works best for your model.

PART B

|  |  |
| --- | --- |
| Roll No: C0009 | Name: Samarth Borade |
| Class : B | Batch : EB1 |
| Date of Experiment: 09/02/24 | Date of Submission |
| Grade : |  |

**B.1 Software Code written by student:**

#Samarth Borade

#C009

#BTI SEM 10

#EXP 6: Learning rate and optimizers

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import SGD, Adagrad, Adam, RMSprop

import matplotlib.pyplot as plt

# Step 1: Load the wheat seed dataset

data = pd.read\_csv('seeds.csv')

# Step 2: Pre-processing and prepare the data

X = data.drop(*columns*=['Type']) # Features

y = data['Type'] # Target

# Encoding species names using label encoder

label\_encoder = LabelEncoder()

y = label\_encoder.fit\_transform(y)

# Normalizing the features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Splitting the data into train and validate

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X, y, *test\_size*=0.2, *random\_state*=42)

# Step 3: Building the sequential neural network model

model = Sequential([

Dense(64, *activation*='relu', *input\_shape*=(X\_train.shape[1],)),

Dense(32, *activation*='relu'),

Dense(3, *activation*='softmax')

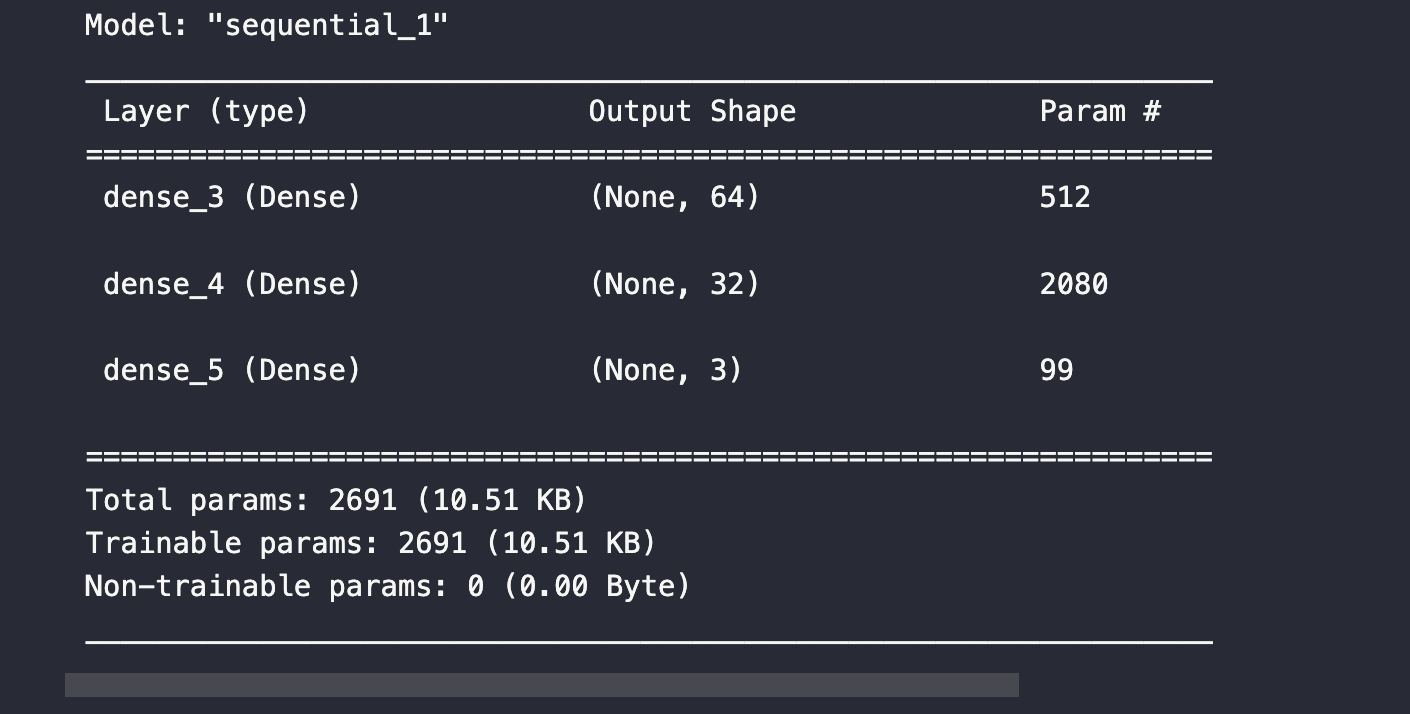
])

# Step 4: Compile and fit the model to the training dataset

model.compile(*optimizer*=SGD(), *loss*='sparse\_categorical\_crossentropy', *metrics*=['accuracy'])

history = model.fit(X\_train, y\_train, *epochs*=50, *validation\_data*=(X\_val, y\_val), *verbose*=0)

model.summary()



# Step 5: Use different learning rates

learning\_rates = [0.1, 0.01, 0.001, 0.0001]

histories = []

for lr in learning\_rates:

model = Sequential([

Dense(64, *activation*='relu', *input\_shape*=(X\_train.shape[1],)),

Dense(32, *activation*='relu'),

Dense(3, *activation*='softmax')

])

optimizer = SGD(*learning\_rate*=lr)

model.compile(*optimizer*=optimizer, *loss*='sparse\_categorical\_crossentropy', *metrics*=['accuracy'])

history = model.fit(X\_train, y\_train, *epochs*=50, *validation\_data*=(X\_val, y\_val), *verbose*=0)

histories.append(history)

# Plot training and validation accuracy curves for different learning rates

plt.figure(*figsize*=(10, 6))

for i, lr in enumerate(learning\_rates):

plt.plot(histories[i].history['accuracy'], *label*=f'Training LR={lr}')

plt.plot(histories[i].history['val\_accuracy'], *label*=f'Validation LR={lr}', *linestyle*='--')

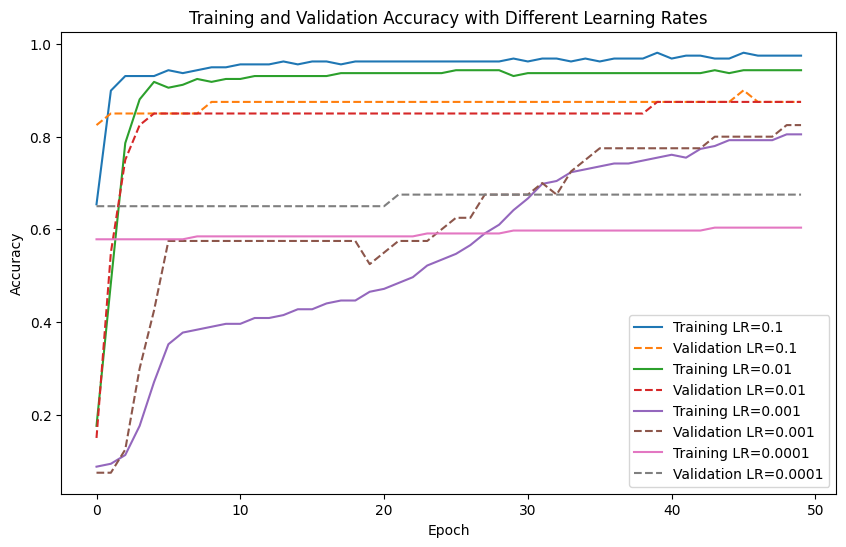
plt.title('Training and Validation Accuracy with Different Learning Rates')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.show()



# Step 6: Use the best learning rate and add momentum to it

best\_lr = 0.01 # Assuming you have identified the best learning rate from Step 5

momentum\_values = [0, 0.5, 0.9, 0.99]

momentum\_histories = []

for momentum in momentum\_values:

model = Sequential([

Dense(64, *activation*='relu', *input\_shape*=(X\_train.shape[1],)),

Dense(32, *activation*='relu'),

Dense(3, *activation*='softmax')

])

optimizer = SGD(*learning\_rate*=best\_lr, *momentum*=momentum)

model.compile(*optimizer*=optimizer, *loss*='sparse\_categorical\_crossentropy', *metrics*=['accuracy'])

history = model.fit(X\_train, y\_train, *epochs*=50, *validation\_data*=(X\_val, y\_val), *verbose*=0)

momentum\_histories.append(history)

# Step 7: Add weight decay (equivalent to decay parameter) using L2 regularization

from tensorflow.keras import regularizers

decay\_values = [1e-1, 1e-2, 1e-3, 1e-4]

decay\_histories = []

for decay in decay\_values:

model = Sequential([

Dense(64, *activation*='relu', *input\_shape*=(X\_train.shape[1],), *kernel\_regularizer*=regularizers.l2(decay)),

Dense(32, *activation*='relu', *kernel\_regularizer*=regularizers.l2(decay)),

Dense(3, *activation*='softmax')

])

optimizer = LegacySGD(*learning\_rate*=best\_lr, *momentum*=0.9)

model.compile(*optimizer*=optimizer, *loss*='sparse\_categorical\_crossentropy', *metrics*=['accuracy'])

history = model.fit(X\_train, y\_train, *epochs*=50, *validation\_data*=(X\_val, y\_val), *verbose*=0)

decay\_histories.append(history)

# Step 8: Train the model using Adagrad, Adam, and RMSprop

optimizers = ['Adagrad', 'Adam', 'RMSprop']

optimizer\_histories = []

for optimizer\_name in optimizers:

model = Sequential([

Dense(64, *activation*='relu', *input\_shape*=(X\_train.shape[1],)),

Dense(32, *activation*='relu'),

Dense(3, *activation*='softmax')

])

if optimizer\_name == 'Adagrad':

optimizer = Adagrad(*learning\_rate*=best\_lr)

elif optimizer\_name == 'Adam':

optimizer = Adam(*learning\_rate*=best\_lr)

elif optimizer\_name == 'RMSprop':

optimizer = RMSprop(*learning\_rate*=best\_lr)

model.compile(*optimizer*=optimizer, *loss*='sparse\_categorical\_crossentropy', *metrics*=['accuracy'])

history = model.fit(X\_train, y\_train, *epochs*=50, *validation\_data*=(X\_val, y\_val), *verbose*=0)

optimizer\_histories.append(history)

Comparison:

# Analyzing the results

# Compare the performance of different momentum values

plt.figure(*figsize*=(10, 6))

for i, momentum in enumerate(momentum\_values):

plt.plot(momentum\_histories[i].history['val\_accuracy'], *label*=f'Momentum={momentum}')

plt.title('Validation Accuracy with Different Momentum Values')

plt.xlabel('Epoch')

plt.ylabel('Validation Accuracy')

plt.legend()

plt.show()

# Compare the performance of different decay values

plt.figure(*figsize*=(10, 6))

for i, decay in enumerate(decay\_values):

plt.plot(decay\_histories[i].history['val\_accuracy'], *label*=f'Decay={decay}')

plt.title('Validation Accuracy with Different Decay Values')

plt.xlabel('Epoch')

plt.ylabel('Validation Accuracy')

plt.legend()

plt.show()

# Compare the performance of different optimizers

plt.figure(*figsize*=(10, 6))

for i, optimizer\_name in enumerate(optimizers):

plt.plot(optimizer\_histories[i].history['val\_accuracy'], *label*=optimizer\_name)

plt.title('Validation Accuracy with Different Optimizers')

plt.xlabel('Epoch')

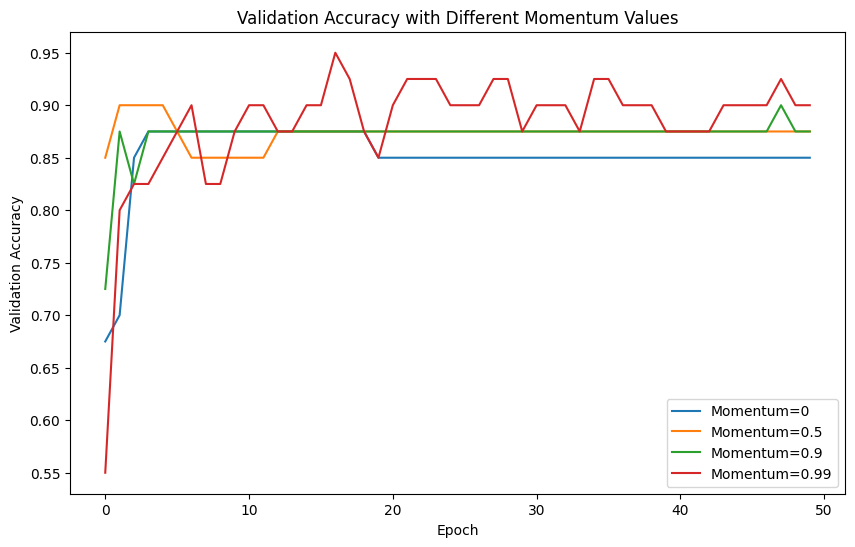
plt.ylabel('Validation Accuracy')

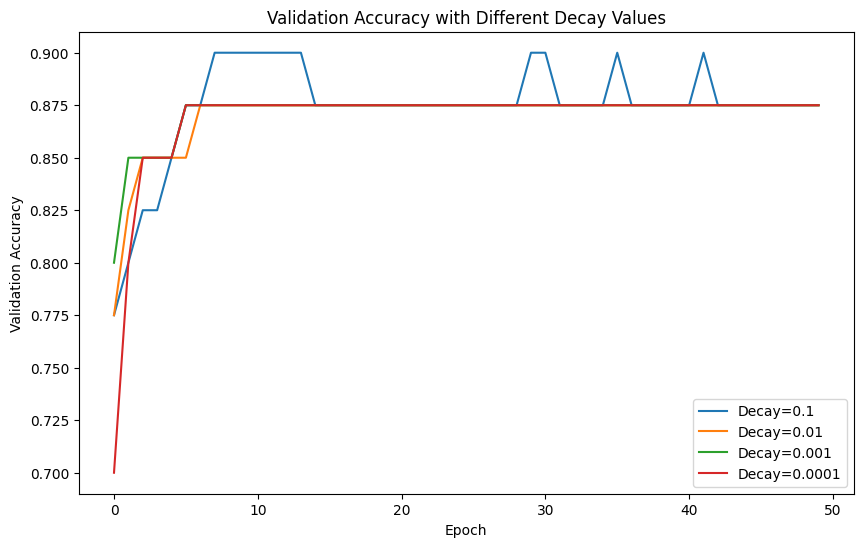
plt.legend()

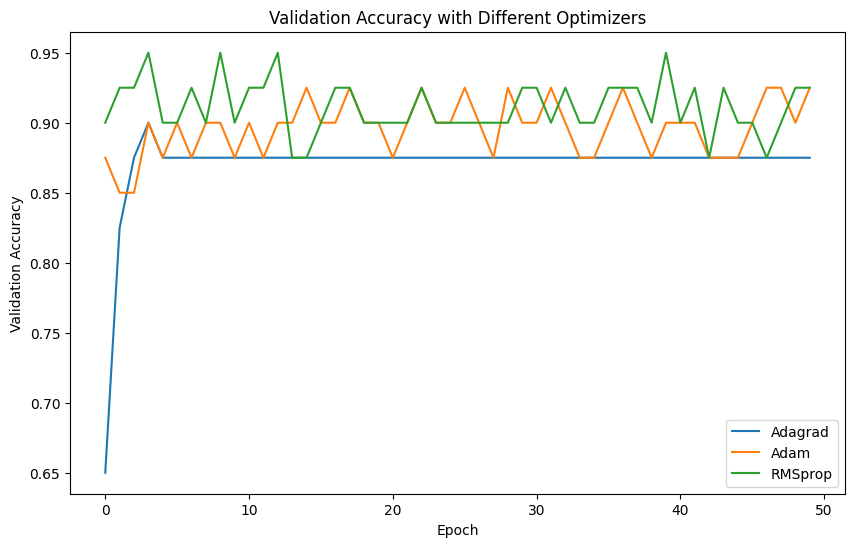
plt.show()

.

**B.3 Observations and learning:**







* Among different momentum values, the highest validation accuracy is achieved with a momentum of 0.99, closely followed by 0.9, indicating that higher momentum values generally lead to better performance.
* For decay values, there is no consistent trend observed, as the highest validation accuracy varies across different decay values, with no clear pattern of improvement or degradation.
* Among the optimizers, RMSprop achieves the highest maximum validation accuracy, followed closely by Adam and then Adagrad, suggesting that RMSprop may be the most suitable optimizer for this dataset.

**B.4 Conclusion:**

Based on the analysis, it can be inferred that higher momentum values tend to yield better validation accuracy, while the impact of decay values on performance is less predictable. Moreover, among the optimizers tested, RMSprop demonstrates superior performance in maximizing validation accuracy. These findings underscore the importance of careful selection and fine-tuning of hyperparameters to optimize the performance of neural network models for specific datasets.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*