|  |  |
| --- | --- |
| **Ex. No: 7**  **Date: 22.02.24** | **MLP with Optimization** |

**Name: S Sowmithaa Sri**

**Reg no: 22011101096**

**Aim:**

To implement a Multi layer perceptron with optimization techniques:

1. Stochastic Gradient Descent
2. Adagrad
3. RMSProp
4. Adagrad

**Dataset description:**

The dataset used is the Fashion-MNIST dataset (a popular benchmark dataset for image classification tasks). It is similar in structure to the original MNIST dataset but consists of images of various clothing items instead of handwritten digits.

1. Size: The dataset contains a total of 70,000 grayscale images.

2. Image Dimensions: Each image is 28x28 pixels.

3. Classes: There are 10 classes in the dataset, representing different categories of clothing items:

- 0: T-shirt/top

- 1: Trouser

- 2: Pullover

- 3: Dress

- 4: Coat

- 5: Sandal

- 6: Shirt

- 7: Sneaker

- 8: Bag

- 9: Ankle boot

4. Split: The dataset is typically split into training and test sets, with 60,000 images used for training and 10,000 images used for testing.

**Code:**

1. **Stochastic Gradient Descent**

import tensorflow as tf

# Load the Fashion MNIST dataset

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

# Normalize the pixel values to be in the range [0, 1]

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Define the architecture of the neural network

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model with the optimizer (gradient descent) and loss function

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

# Train the model

model.fit(X\_train, y\_train, epochs=5)

# Evaluate the model on the test set

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

**Result:**

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>

29515/29515 [==============================] - 0s 0us/step

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz>

26421880/26421880 [==============================] - 0s 0us/step

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz>

5148/5148 [==============================] - 0s 0us/step

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz>

4422102/4422102 [==============================] - 0s 0us/step

Epoch 1/5

1875/1875 [==============================] - 5s 2ms/step - loss: 0.8099 - accuracy: 0.7324

Epoch 2/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.5634 - accuracy: 0.8085

Epoch 3/5

1875/1875 [==============================] - 5s 3ms/step - loss: 0.5050 - accuracy: 0.8271

Epoch 4/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.4725 - accuracy: 0.8352

Epoch 5/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.4503 - accuracy: 0.8431

313/313 [==============================] - 1s 2ms/step - loss: 0.4453 - accuracy: 0.8448

Test Loss: 0.4452683627605438, Test Accuracy: 0.8447999954223633

1. **AdaGrad:**

import tensorflow as tf

# Load the Fashion MNIST dataset

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

# Normalize the pixel values to be in the range [0, 1]

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Define the architecture of the neural network

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model with the Adagrad optimizer and loss function

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

# Train the model

model.fit(X\_train, y\_train, epochs=5)

# Evaluate the model on the test set

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

**Result:**

Epoch 1/5

1875/1875 [==============================] - 6s 3ms/step - loss: 1.1655 - accuracy: 0.6231

Epoch 2/5

1875/1875 [==============================] - 5s 3ms/step - loss: 0.8125 - accuracy: 0.7293

Epoch 3/5

1875/1875 [==============================] - 6s 3ms/step - loss: 0.7289 - accuracy: 0.7576

Epoch 4/5

1875/1875 [==============================] - 5s 3ms/step - loss: 0.6825 - accuracy: 0.7743

Epoch 5/5

1875/1875 [==============================] - 5s 3ms/step - loss: 0.6531 - accuracy: 0.7844

313/313 [==============================] - 1s 2ms/step - loss: 0.6141 - accuracy: 0.7953

Test Loss: 0.6141026020050049, Test Accuracy: 0.7953000068664551

**RMSProp:**

import tensorflow as tf

# Load the Fashion MNIST dataset

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

# Normalize the pixel values to be in the range [0, 1]

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Define the architecture of the neural network

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model with the RMSprop optimizer and loss function

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

# Train the model

model.fit(X\_train, y\_train, epochs=5)

# Evaluate the model on the test set

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

**Result:**

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>

29515/29515 [==============================] - 0s 0us/step

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz>

26421880/26421880 [==============================] - 0s 0us/step

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz>

5148/5148 [==============================] - 0s 0us/step

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz>

4422102/4422102 [==============================] - 0s 0us/step

Epoch 1/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.5367 - accuracy: 0.8096

Epoch 2/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.4083 - accuracy: 0.8547

Epoch 3/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3830 - accuracy: 0.8655

Epoch 4/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3712 - accuracy: 0.8706

Epoch 5/5

1875/1875 [==============================] - 3s 2ms/step - loss: 0.3583 - accuracy: 0.8757

313/313 [==============================] - 0s 1ms/step - loss: 0.4280 - accuracy: 0.8556

Test Loss: 0.4279777407646179, Test Accuracy: 0.8555999994277954

**Adam:**

import tensorflow as tf

# Load the Fashion MNIST dataset

fashion\_mnist = tf.keras.datasets.fashion\_mnist

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

# Normalize the pixel values to be in the range [0, 1]

X\_train, X\_test = X\_train / 255.0, X\_test / 255.0

# Define the architecture of the neural network

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation='softmax')

])

# Compile the model with the Adam optimizer and loss function

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

# Train the model

model.fit(X\_train, y\_train, epochs=5)

# Evaluate the model on the test set

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f'Test Loss: {loss}, Test Accuracy: {accuracy}')

**Result:**

Epoch 1/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.5313 - accuracy: 0.8126

Epoch 2/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.4001 - accuracy: 0.8549

Epoch 3/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3663 - accuracy: 0.8675

Epoch 4/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3441 - accuracy: 0.8735

Epoch 5/5

1875/1875 [==============================] - 4s 2ms/step - loss: 0.3282 - accuracy: 0.8783

313/313 [==============================] - 0s 1ms/step - loss: 0.3583 - accuracy: 0.8693

Test Loss: 0.3583424389362335, Test Accuracy: 0.8693000078201294

**Result Description:**

**---In General:**

SGD is simple and easy to implement, it may suffer from slow convergence. Adagrad and RMSProp address some of SGD's limitations by adaptively adjusting the learning rate, but they have their drawbacks. Adam combines the benefits of both momentum and adaptive learning rates, making it a popular choice for optimizing deep neural networks. However, the choice of optimization algorithm depends on various factors, including the problem at hand, the dataset, and computational resources. It's often recommended to experiment with different optimization algorithms and hyperparameters to find the best-performing configuration for a specific task.

**---Model Performance:**

1. Stochastic Gradient Descent (SGD):

- Test Accuracy: 0.84

- SGD is a classic optimization algorithm widely used in training neural networks. It updates the model's parameters in the opposite direction of the gradient of the loss function with respect to the parameters.

- It performs well in many cases but can be sensitive to the learning rate and may take longer to converge compared to more sophisticated optimizers.

2. Adagrad:

- Test Accuracy: 0.79

- Adagrad adapts the learning rate for each parameter based on the historical gradients. It allocates larger updates to infrequent parameters and smaller updates to frequent parameters.

- While Adagrad performs well in many scenarios, it may suffer from a diminishing learning rate problem where the learning rate becomes too small over time, making it less effective for fine-tuning.

3. RMSProp:

- Test Accuracy: 0.85

- RMSProp addresses the diminishing learning rate problem of Adagrad by using a moving average of squared gradients to normalize the learning rate. It scales down the learning rate adaptively.

- RMSProp is more robust than Adagrad for training deep neural networks and often converges faster. It can handle non-stationary objectives better.

4. Adam (Adaptive Moment Estimation):

- Test Accuracy: 0.86

- Adam combines the advantages of both RMSProp and momentum. It uses moving averages of both the gradients and the squared gradients, and also includes momentum to adaptively adjust the learning rates for each parameter.

- Adam is widely used due to its efficiency, robustness, and ease of use. It often achieves good performance across various tasks and architectures with default hyperparameters.

In summary, Adam generally performs the best among these optimization techniques, followed closely by RMSProp. Adagrad may suffer from a diminishing learning rate issue, leading to slower convergence and potentially lower performance. SGD can perform well, but it may require careful tuning of the learning rate and other hyperparameters to achieve optimal results. Overall, Adam is the most recommended choice for training neural networks due to its strong performance across a wide range of scenarios.

**Conclusion:** Thus the multiclass classification with optimisation techniques like SGD, Adagrad, RMSProp, Adam is successfully implemented and executed.