Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department

Subject/Odd Sem 2023-23/Experiment 2

Name : Prasad Jawale	**Class/Roll No. :** D16AD 20	**Grade :**

Title of Experiment : Apply any of the following learning algorithms to learn the parameters of the supervised single layer feed forward neural network.

- Stochastic Gradient Descent
- Mini Batch Gradient Descent
- Momentum GD
- Nestorev GD
- Adagrad GD
- Adam Learning GD

Objective of Experiment : The objective is to apply various gradient descent-based optimization algorithms to learn the parameters of a supervised single-layer feed-forward neural network. By experimenting with different algorithms, we aim to compare their convergence speeds and overall performance in terms of training the neural network.

Outcome of Experiment : The outcome of this experiment will be a comparison of how different optimization algorithms perform in training a single-layer feed-forward neural network. We'll analyze their convergence rates, final accuracy on the validation set, and potentially identify which optimization algorithm works best for this specific neural network architecture and dataset.

Problem Statement : To implement deep learning algorithms in an neural network

Description / Theory :

1. Stochastic Gradient Descent (SGD):

Stochastic Gradient Descent is the simplest form of gradient descent. In each iteration, it randomly selects a single data point from the training set to compute the gradient and update the model's parameters. SGD introduces a level of randomness that can lead to faster convergence, especially in noisy or non-convex optimization landscapes. However, this randomness might also cause oscillations, and it can be slower to converge when the objective function has a lot of noise.

2. Mini Batch Gradient Descent:

Mini Batch Gradient Descent is a compromise between full-batch GD and SGD. It divides the training dataset into small batches of data points. In each iteration, one batch is used to compute the gradient and update the parameters. Mini-batch GD combines the advantages of both SGD and full-batch GD. It reduces the noise introduced by single data points in SGD and takes advantage of vectorized operations for efficient computation. The batch size can be adjusted to balance between computational efficiency and convergence speed.

3. Momentum Gradient Descent:

Momentum is an enhancement to SGD that addresses the slow convergence issue by adding a momentum term. Instead of updating the parameters directly based on the current gradient, momentum GD also considers the previous gradient updates. This helps accelerate convergence in the direction of the steepest descent and dampens oscillations. It introduces a "velocity" term that accumulates past gradients, making it useful for escaping shallow local minima.

4. Nesterov Accelerated Gradient (NAG):

Nesterov Accelerated Gradient builds upon momentum GD. NAG estimates the gradient's direction based on where the momentum would take the parameters in the next step. It uses this lookahead to calculate the gradient at a "virtual" point ahead of the current position, resulting in more accurate updates. NAG improves convergence by considering the upcoming momentum-driven update and reduces overshooting, making it effective in practice.

5. Adagrad Gradient Descent:

Adagrad adjusts the learning rate for each parameter individually based on the historical gradients. Parameters that receive large gradients get a smaller learning rate, while those with small gradients get a larger learning rate. This adaptivity helps to balance learning rates automatically, allowing the algorithm to make larger updates for infrequently updated parameters and smaller updates for frequently updated parameters. However, Adagrad might accumulate the squared gradients, leading to a diminishing learning rate over time.

6. Adam (Adaptive Moment Estimation) Gradient Descent:

Adam combines the adaptive learning rate of Adagrad with the momentum term of momentum GD. It maintains running averages of both past gradients and past squared gradients, then uses these to compute adaptive learning rates for each parameter. The momentum term helps smooth the parameter updates. Adam's combination of adaptive learning rates and momentum often makes it a suitable choice for a wide range of optimization problems.

Algorithm:

1. Initialize parameters (weights and biases) randomly or using a predefined method.
2. Choose hyperparameters: learning rate, batch size (if applicable), momentum rate (if applicable), etc.
3. Loop for a fixed number of epochs or until convergence:
    - Calculate the gradient of the loss function with respect to parameters using the current batch or all training data.
    - Update the parameters in the opposite direction of the gradient using the learning rate.
    - Apply additional techniques if using specific gradient descent variants (momentum, adaptive learning rates, etc.).
    - Optionally, calculate and store the loss or other metrics for monitoring convergence.

Program :

```
In [3]: import pandas as pd
        import numpy as np
        import tensorflow as tf
        from tensorflow import keras
        from keras import Sequential
        from keras.layers import Dense,Flatten,Dropout

In [4]: from tensorflow.keras.datasets import mnist
        (X_train,Y_train),(X_test,Y_test) = mnist.load_data()
        X_train, X_test = X_train / 255.0, X_test / 255.0

In [5]: X_train.shape

Out[5]: (60000, 28, 28)

In [6]: model = Sequential()
        model.add(Flatten(input_shape=(28, 28)))
        model.add(Dense(128, activation='relu'))
        model.add(Dropout(0.2))
        model.add(Dense(10, activation='softmax'))
```

Stochastic gradient descent

```
In [7]: #Stochastic Gradient Des

In [8]: optimizer = keras.optimizers.SGD(learning_rate=0.1)
        model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer,metrics=['accuracy'])

In [9]: model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10)

        Epoch 1/10
        1875/1875 [==============================] - 5s 2ms/step - loss: 0.3330 - accuracy: 0.9029 - val_loss: 0.1690 - val_accuracy:
        0.9490
        Epoch 2/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.1733 - accuracy: 0.9496 - val_loss: 0.1205 - val_accuracy:
        0.9635
        Epoch 3/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.1344 - accuracy: 0.9598 - val_loss: 0.1003 - val_accuracy:
        0.9705
        Epoch 4/10
        1875/1875 [==============================] - 5s 2ms/step - loss: 0.1149 - accuracy: 0.9655 - val_loss: 0.0931 - val_accuracy:
        0.9701
        Epoch 5/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.0995 - accuracy: 0.9703 - val_loss: 0.0846 - val_accuracy:
        0.9737
        Epoch 6/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.0885 - accuracy: 0.9736 - val_loss: 0.0789 - val_accuracy:
        0.9757
        Epoch 7/10
        1875/1875 [==============================] - 5s 3ms/step - loss: 0.0789 - accuracy: 0.9759 - val_loss: 0.0773 - val_accuracy:
        0.9753
        Epoch 8/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.0739 - accuracy: 0.9773 - val_loss: 0.0690 - val_accuracy:
        0.9780
        Epoch 9/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.0671 - accuracy: 0.9792 - val_loss: 0.0688 - val_accuracy:
        0.9782
        Epoch 10/10
        1875/1875 [==============================] - 4s 2ms/step - loss: 0.0628 - accuracy: 0.9805 - val_loss: 0.0676 - val_accuracy:
        0.9790

Out[9]: <keras.callbacks.History at 0x22824f22730>
```

Stochastic gradient descent with momentum

```
In [10]: #Stochastic Gradient Descent with Momentum
```

```
In [11]: optimizer1 = keras.optimizers.SGD(learning_rate=0.1,momentum=0.9)
         model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer1,metrics=['accuracy'])
```

```
In [12]: model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10)

Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.3711 - accuracy: 0.8974 - val_loss: 0.2673 - val_accuracy:
0.9308
Epoch 2/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2993 - accuracy: 0.9215 - val_loss: 0.1979 - val_accuracy:
0.9449
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2636 - accuracy: 0.9333 - val_loss: 0.2350 - val_accuracy:
0.9470
Epoch 4/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2409 - accuracy: 0.9386 - val_loss: 0.1949 - val_accuracy:
0.9550
Epoch 5/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2307 - accuracy: 0.9433 - val_loss: 0.1972 - val_accuracy:
0.9544
Epoch 6/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2223 - accuracy: 0.9443 - val_loss: 0.1774 - val_accuracy:
0.9602
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.2149 - accuracy: 0.9481 - val_loss: 0.2015 - val_accuracy:
0.9609
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2076 - accuracy: 0.9500 - val_loss: 0.2153 - val_accuracy:
0.9558
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2000 - accuracy: 0.9516 - val_loss: 0.2011 - val_accuracy:
0.9575
Epoch 10/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1992 - accuracy: 0.9521 - val_loss: 0.1905 - val_accuracy:
0.9618

Out[12]: <keras.callbacks.History at 0x2282a1c0460>
```

Nesterov Accelerated Gradient

In [13]:
```
#Nesterov Accelarated Gradient(NAG)
```

In [14]:
```
optimizer3 = keras.optimizers.SGD(learning_rate=0.1, momentum=0.9, nesterov=True)
model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer3,metrics=['accuracy'])
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1809 - accuracy: 0.9560 - val_loss: 0.2385 - val_accuracy:
0.9564
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1738 - accuracy: 0.9576 - val_loss: 0.2000 - val_accuracy:
0.9604
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1652 - accuracy: 0.9596 - val_loss: 0.2272 - val_accuracy:
0.9602
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1715 - accuracy: 0.9590 - val_loss: 0.1914 - val_accuracy:
0.9645
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1538 - accuracy: 0.9631 - val_loss: 0.1950 - val_accuracy:
0.9646
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1433 - accuracy: 0.9650 - val_loss: 0.1936 - val_accuracy:
0.9622
Epoch 7/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1559 - accuracy: 0.9635 - val_loss: 0.2296 - val_accuracy:
0.9622
Epoch 8/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1646 - accuracy: 0.9609 - val_loss: 0.2559 - val_accuracy:
0.9574
Epoch 9/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1573 - accuracy: 0.9633 - val_loss: 0.2158 - val_accuracy:
0.9640
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1569 - accuracy: 0.9642 - val_loss: 0.2307 - val_accuracy:
0.9615
```

Out[14]:
```
<keras.callbacks.History at 0x2285dac9460>
```

Adam Optimiser

In [15]:
```
#Adam Optimizer
```

In [16]:
```
optimizer4 = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer4,metrics=['accuracy'])
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10)
```

```
Epoch 1/10
1875/1875 [==============================] - 5s 2ms/step - loss: 0.2599 - accuracy: 0.9467 - val_loss: 0.2559 - val_accuracy:
0.9562
Epoch 2/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2305 - accuracy: 0.9481 - val_loss: 0.2614 - val_accuracy:
0.9536
Epoch 3/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2091 - accuracy: 0.9515 - val_loss: 0.2508 - val_accuracy:
0.9542
Epoch 4/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2099 - accuracy: 0.9525 - val_loss: 0.2788 - val_accuracy:
0.9566
Epoch 5/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1884 - accuracy: 0.9558 - val_loss: 0.2113 - val_accuracy:
0.9635
Epoch 6/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1870 - accuracy: 0.9568 - val_loss: 0.2501 - val_accuracy:
0.9595
Epoch 7/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1868 - accuracy: 0.9569 - val_loss: 0.2583 - val_accuracy:
0.9580
Epoch 8/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1798 - accuracy: 0.9585 - val_loss: 0.2358 - val_accuracy:
0.9635
Epoch 9/10
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1736 - accuracy: 0.9609 - val_loss: 0.2466 - val_accuracy:
0.9636
Epoch 10/10
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1708 - accuracy: 0.9608 - val_loss: 0.2548 - val_accuracy:
0.9643
```

Out[16]:
```
<keras.callbacks.History at 0x2285ccfa490>
```

Mini Batch Gradient Descent

```
In [17]: #Mini Batch Gradient Descent with Adam optimizer
optimizer5 = keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer5,metrics=['accuracy'])
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10,batch_size=64)
```

```
Epoch 1/10
938/938 [==============================] - 3s 3ms/step - loss: 0.1440 - accuracy: 0.9675 - val_loss: 0.2229 - val_accuracy: 0.9
700
Epoch 2/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1313 - accuracy: 0.9686 - val_loss: 0.2280 - val_accuracy: 0.9
664
Epoch 3/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1242 - accuracy: 0.9696 - val_loss: 0.2485 - val_accuracy: 0.9
660
Epoch 4/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1253 - accuracy: 0.9697 - val_loss: 0.2794 - val_accuracy: 0.9
661
Epoch 5/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1169 - accuracy: 0.9716 - val_loss: 0.2357 - val_accuracy: 0.9
665
Epoch 6/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1120 - accuracy: 0.9726 - val_loss: 0.2423 - val_accuracy: 0.9
663
Epoch 7/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1156 - accuracy: 0.9725 - val_loss: 0.2524 - val_accuracy: 0.9
685
Epoch 8/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1208 - accuracy: 0.9718 - val_loss: 0.2485 - val_accuracy: 0.9
685
Epoch 9/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1171 - accuracy: 0.9721 - val_loss: 0.2394 - val_accuracy: 0.9
697
Epoch 10/10
938/938 [==============================] - 2s 2ms/step - loss: 0.1126 - accuracy: 0.9730 - val_loss: 0.2856 - val_accuracy: 0.9
662
```

```
Out[17]: <keras.callbacks.History at 0x2280d7174c0>
```

AdaGrad

```
In [18]: #AdaGrad Optimizer
optimizer6 = keras.optimizers.Adagrad(learning_rate=0.01)
model.compile(loss='sparse_categorical_crossentropy',optimizer=optimizer6,metrics=['accuracy'])
model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10)
```

```
Epoch 5/10
1875/1875 [==============================] - 2s 1ms/step - loss: 0.0674 - accuracy: 0.9819 - val_loss: 0.2390 - val_accuracy:
0.9725
Epoch 6/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0628 - accuracy: 0.9829 - val_loss: 0.2370 - val_accuracy:
0.9723
Epoch 7/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0651 - accuracy: 0.9822 - val_loss: 0.2360 - val_accuracy:
0.9721
Epoch 8/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0641 - accuracy: 0.9826 - val_loss: 0.2356 - val_accuracy:
0.9728
Epoch 9/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0625 - accuracy: 0.9828 - val_loss: 0.2359 - val_accuracy:
0.9727
Epoch 10/10
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0617 - accuracy: 0.9826 - val_loss: 0.2358 - val_accuracy:
0.9730
```

```
Out[18]: <keras.callbacks.History at 0x2280d2c0040>
```

```
In [ ]:
```

Results and Discussions : In conclusion, the various gradient descent optimization algorithms offer different trade-offs in terms of convergence speed, stability, and adaptability to different optimization landscapes. Stochastic Gradient Descent (SGD) introduces randomness for faster convergence but can be noisy. Mini Batch Gradient Descent balances between efficiency and convergence. Momentum Gradient Descent accelerates convergence by adding momentum terms, while Nesterov Accelerated Gradient refines this approach with lookahead updates. Adagrad adapts learning rates to parameters' historical gradients, and Adam combines adaptive learning rates with momentum for versatility.