SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

COURSE CODE: DJ19DSC501 **DATE:**

COURSE NAME: Machine Learning - II **CLASS: AY 2023-24**

LAB EXPERIMENT NO.3

60009210105

Amitesh Sawarkar

D 12

AIM :

Evaluate and analyze Prediction performance using appropriate optimizers for deep learning models.

THEORY:

Optimizers: Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses. It finds the value of parameters(weights) that minimize the error when mapping inputs to outputs. These optimization algorithms or optimizers widely affect the accuracy of deep learning model and the speed of training of the model.

Types:

1. **Gradient Descent** - most basic but most used optimization algorithm. Gradient descent is a first-order optimization algorithm which is dependent on the first order derivative of a loss function. It calculates that which way the weights should be altered so that the function can reach a minima. Through backpropagation, the loss is transferred from one layer to another and the model's parameters also known as weights are modified depending on the losses so that the loss can be minimized.

2. **Stochastic Gradient Descent** - variant of Gradient Descent. It tries to update the model's parameters more frequently. In this, the model parameters are altered after computation of loss on each training example. So, if the dataset contains 1000 rows SGD will update the model parameters 1000 times in one cycle of dataset instead of one time as in Gradient Descent

3. **Stochastic Gradient descent with momentum** - Momentum was invented for reducing high variance in SGD and softens the convergence. It accelerates the convergence towards the relevant direction and reduces the fluctuation to the irrelevant direction.

4. **Mini-Batch Gradient Descent** - best among all the variations of gradient descent algorithms. It is an improvement on both SGD and standard gradient descent. It updates the model

parameters after every batch. So, the dataset is divided into various batches and after every batch, the parameters are updated.

5. **Adagrad** - This optimizer changes the learning rate - uses different learning rates for each iteration. It changes the learning rate ‘η’ for each parameter and at every time step ‘t’. It’s a type second order optimization algorithm. It works on the derivative of an error function.

6. **RMSProp** - The algorithm mainly focuses on accelerating the optimization process by decreasing the number of function evaluations to reach the local minima. The algorithm keeps the moving average of squared gradients for every weight and divides the gradient by the square root of the mean square.

7. **AdaDelta** - It is an extension of AdaGrad which tends to remove the decaying learning Rate problem of it. Instead of accumulating all previously squared gradients, Adadelta limits the window of accumulated past gradients to some fixed size w. In this exponentially moving average is used rather than the sum of all the gradients.

8. **Adam** - adaptive moment estimation - adam optimizer updates the learning rate for each network weight individually. Adam optimizers inherit the features of both Adagrad and RMS prop algorithms. The intuition behind the Adam is that we don’t want to roll so fast just because we can jump over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta, Adam also keeps an exponentially decaying average of past gradients M(t).

Tasks to be performed:

- **a) Take the MNIST dataset**
- **b) Initialize a neural network basic layers with random weights.**
- **c) Perform practical analysis of optimizers on MNIST dataset keeping batch size, and epochs same but with different optimizers.**
- **d) Compare the results by choosing 8 different optimizers on a simple neural network**

[gradient desecent, Stochastic Gradient Descent, Stochastic Gradient descent with momentum, Mini-Batch Gradient Descent, Adagrad, RMSProp, AdaDelta, Adam]

- **e) List Advantages and Disadvantages of each Optimizer.**

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```
[1] def sigmoid(y_in):
        y_hat = 1 / (1 + np.exp(-y_in))
        return y_hat
```

```
[2] def perceptron(x, w, b):
        y_in = x * w + b
        y_hat = sigmoid(y_in)
        return y_hat
```

```
[3] def grad_w(x, y, w, b):
        y_hat = perceptron(x, w, b)
        db =  (y - y_hat) * y_hat * (1 - y_hat)
        return db

    def grad_b(x, y, w, b):
        y_hat = perceptron(x, w, b)
        dw =  (y - y_hat) * y_hat * (1 - y_hat) * x
        return dw
```

```python
def minibatch(w, b, x, y,a):
    n = 0.1
    epoch = 10
    batch_size = int(input("Enter the batch size: "))
    for i in range(epoch):
        dw, db, sample_no = 0, 0, 0
        for xi, yi in zip(x, y):
            dw += grad_w(w, b, xi, yi)
            db += grad_b(w, b, xi, yi)
            sample_no += 1
            if sample_no % batch_size == 0:
                w = w - dw*a
                b = b - db *a

    return w, b
x=np.array([0.5,2.5])
y=np.array([1.2,0.9])
w=0.0
b=0
a=0.1
new_w, new_b = minibatch(w, 0, x, y,a)

print("Updated w:", new_w)
print("Updated b:", new_b)
```

```
Enter the batch size: 2
Updated w: 0.2663132373855611
Updated b: 0.03190748395173052
```

```python
def momentum_descent(w, b, x, y, alpha,beta, num_epochs):
    v_w, v_b = 0.0, 0.0
    for epoch in range(num_epochs):
        dw,db=0,0
        for xi, yi in zip(x, y):
            dw = grad_w(w, b, xi, yi)
            db = grad_b(w, b, xi, yi)

        v_w = beta * v_w +(1-beta)* dw
        v_b = beta * v_b + (1-beta)* db

        w -= v_w*alpha
        b -= v_b*alpha
    return w, b

x=np.array([0.5,2.5])
y=np.array([1.2,0.9])
w=0.0
b=0
a=0.1
num_epochs = 10
beta=0.9
new_w, new_b = momentum_descent(w, 0.0, x, y, a, beta, num_epochs)

print("Updated w:", new_w)
print("Updated b:", new_b)
```

```
Updated w: 0.060159284696433391
Updated b: 0.00076167207008995
```

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

```
import numpy as np

def adagrad(w, b, x, y, alpha, epsilon, num_epochs):

    for epoch in range(num_epochs):
        for xi, yi in zip(x, y):
            dw = grad_w(w, b, xi, yi)
            db = grad_b(w, b, xi, yi)

            w -= (alpha / (np.sqrt(dw ** 2) + epsilon)) * dw
            b -= (alpha / (np.sqrt(db ** 2) + epsilon)) * db

    return w, b

x=np.array([0.5,2.5])
y=np.array([1.2,0.9])
w=0.0
b=0
a=0.1
num_epochs = 10
eps=0.00001

new_w, new_b = adagrad(w, 0, x, y, a, eps, num_epochs)

print("Updated w:", new_w)
print("Updated b:", new_b)
```

```
Updated w: 0.9995639624032058
Updated b: 0.8992503170804655
```

```
def NAG(w, b, x, y, alpha, beta, num_epochs):
    v_w, v_b = 0.0, 0.0
    for epoch in range(num_epochs):
        for xi, yi in zip(x, y):
            lookahead_dw = grad_w(w - beta * v_w, b - beta * v_b, xi, yi)
            lookahead_db = grad_b(w - beta * v_w, b - beta * v_b, xi, yi)

            v_w = beta * v_w - alpha * lookahead_dw
            v_b = beta * v_b - alpha * lookahead_db

            w += v_w
            b += v_b

    return w, b

x=np.array([0.5,2.5])
y=np.array([1.2,0.9])
w=0.0
b=0
a=0.1
num_epochs = 10
beta=0.9
new_w, new_b = NAG(w, 0, x, y, a, beta, num_epochs)

print("Updated w:", new_w)
print("Updated b:", new_b)
```

```
Updated w: 1.3658099442347835
Updated b: 0.35595057946420394
```

```
def adam(w, b, x, y, alpha, beta1, beta2, epsilon, num_epochs):
    w,b=0.0,0.0
    m_w = np.zeros_like(w)
    m_b, v_b = 0.0, 0.0
    v_w = np.zeros_like(w)
    i=0
    for epoch in range(num_epochs):
        for xi, yi in zip(x, y):
            i += 1
            dw = grad_w(w, b, xi, yi)
            db = grad_b(w, b, xi, yi)
            m_w = beta1 * m_w + (1 - beta1) * dw
            m_b = beta1 * m_b + (1 - beta1) * db
            v_w = beta2 * v_w + (1 - beta2) * (dw ** 2)
            v_b = beta2 * v_b + (1 - beta2) * (db ** 2)
            m_w_hat = m_w / (1 - beta1 ** i)
            m_b_hat = m_b / (1 - beta1 ** i)
            v_w_hat = v_w / (1 - beta2 ** i)
            v_b_hat = v_b / (1 - beta2 ** i)
            w -= (alpha / (np.sqrt(v_w_hat) + epsilon)) * m_w_hat
            b -= (alpha / (np.sqrt(v_b_hat) + epsilon)) * m_b_hat
    return w, b
```

```
x=np.array([0.5,2.5])
y=np.array([1.2,0.9])
w,b,a=0.0,0,0.1
num_epochs = 10
esp=0.0001
beta1,beta2 = 0.9,0.999
new_w, new_b = adam(w, 0, x, y, a, beta1, beta2, eps, num_epochs)
print("Updated w:", new_w)
print("Updated b:", new_b)
```

```
Updated w: 1.2533562939177947
Updated b: 1.0582274871855926
```

```
def adadelta(w, b, x, y, rho, epsilon, num_epochs):
    w, b = 0.0, 0.0
    E_dw, E_db = 0.0,0.0
    delta_w, delta_b =  0.0,0.0

    for epoch in range(num_epochs):
        for xi, yi in zip(x, y):
            dw = grad_w(w, b, xi, yi)
            db = grad_b(w, b, xi, yi)

            E_dw = rho * E_dw + (1 - rho) * (dw ** 2)
            E_db = rho * E_db + (1 - rho) * (db ** 2)

            delta_w = -np.sqrt(delta_w + epsilon) / (np.sqrt(E_dw + epsilon)) * dw
            delta_b = -np.sqrt(delta_b + epsilon) / (np.sqrt(E_db + epsilon)) * db


            w += delta_w
            b += delta_b

    return w, b
```

```
x = np.array([0.5, 2.5])
y = np.array([1.2, 0.9])
w, b, alpha, rho, eps, num_epochs = 0.0, 0.0, 0.1, 0.95, 0.001, 10

new_w, new_b = adadelta(w, b, x, y, rho, eps, num_epochs)
print("Updated w:", new_w)
print("Updated b:", new_b)
```

```
Updated w: 2.7740631226689167
Updated b: 0.8735200645845033
```