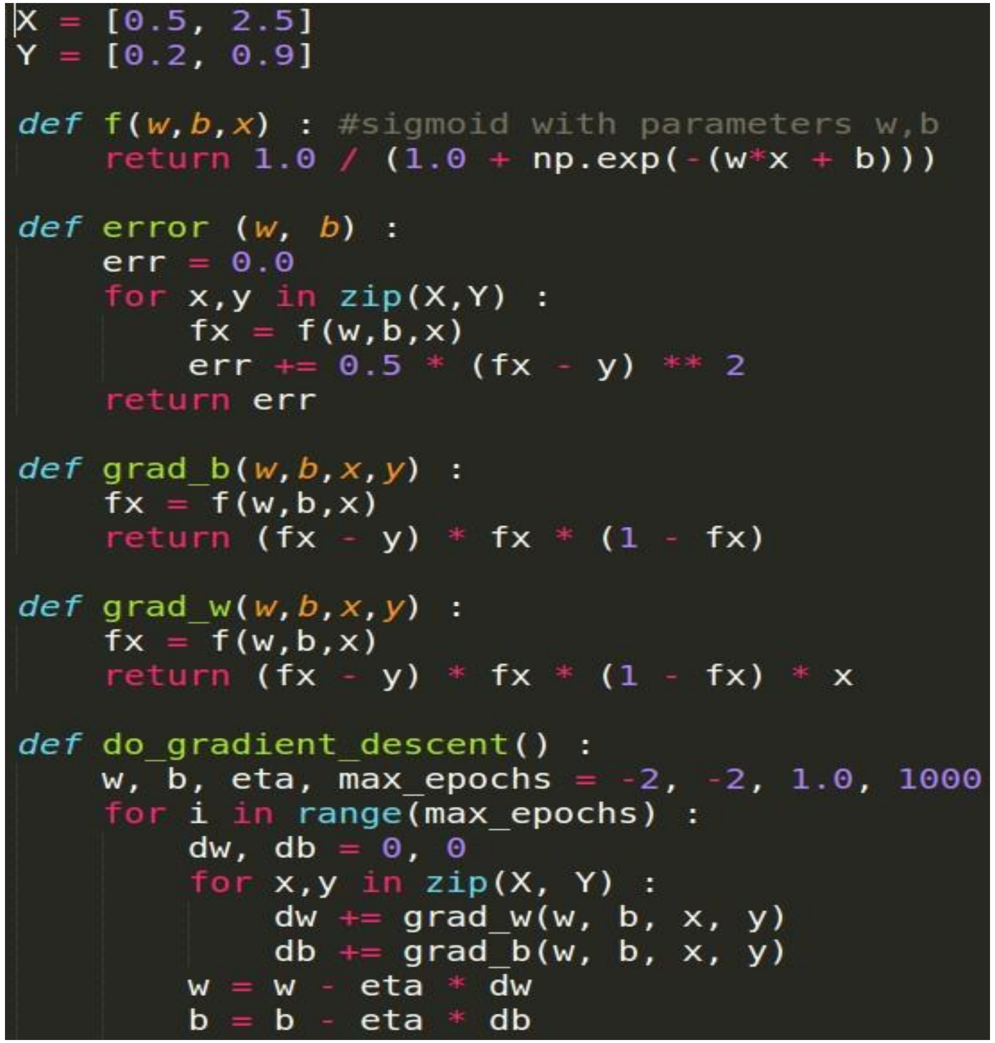
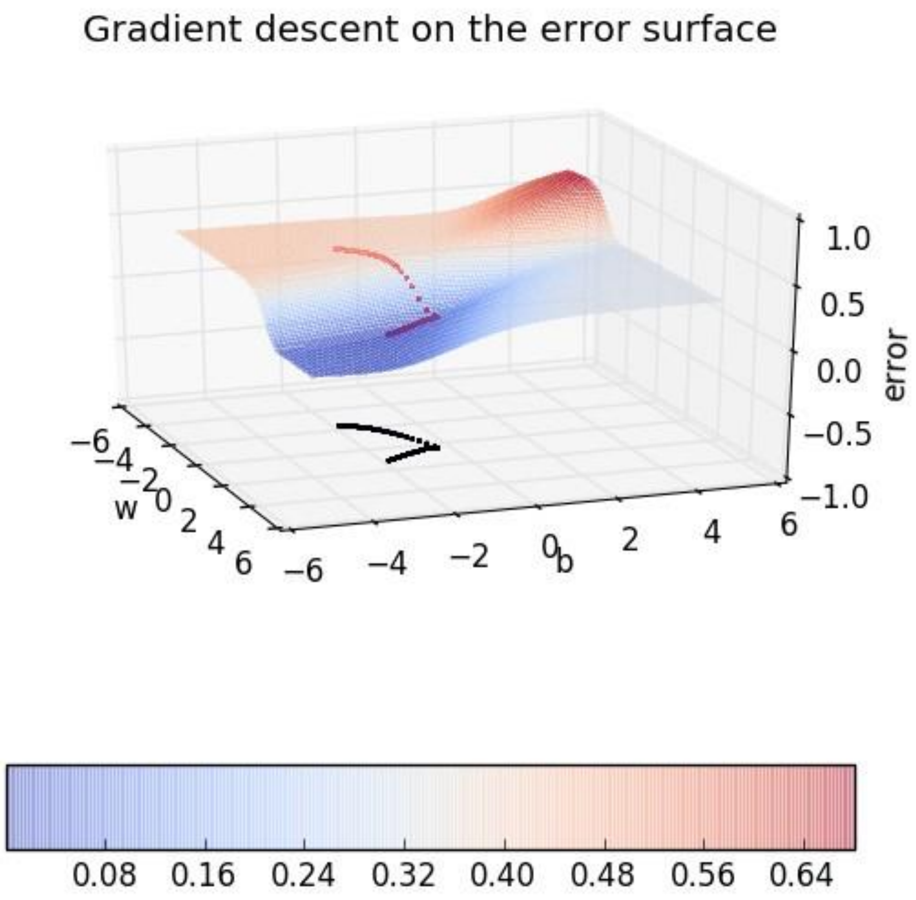
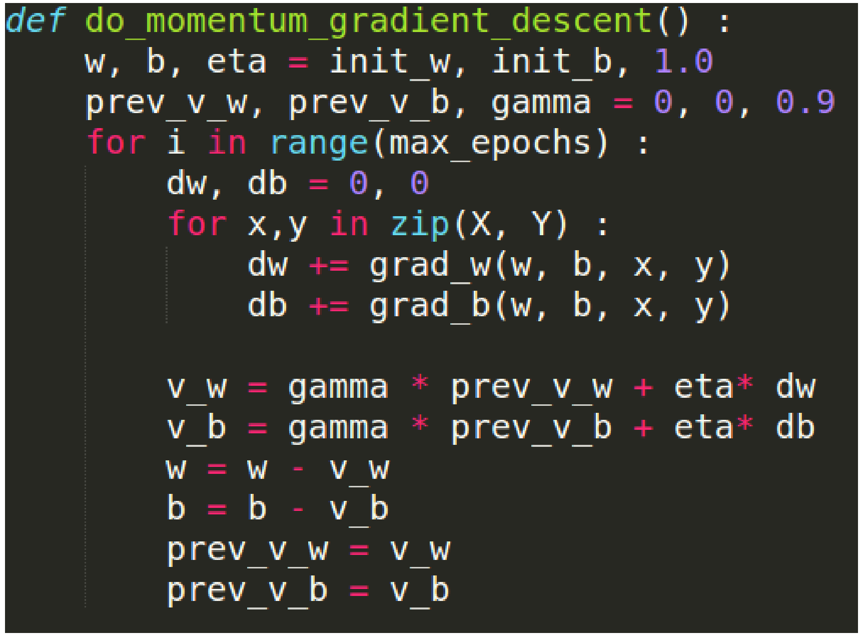
Optimizers(Gradient Descent)



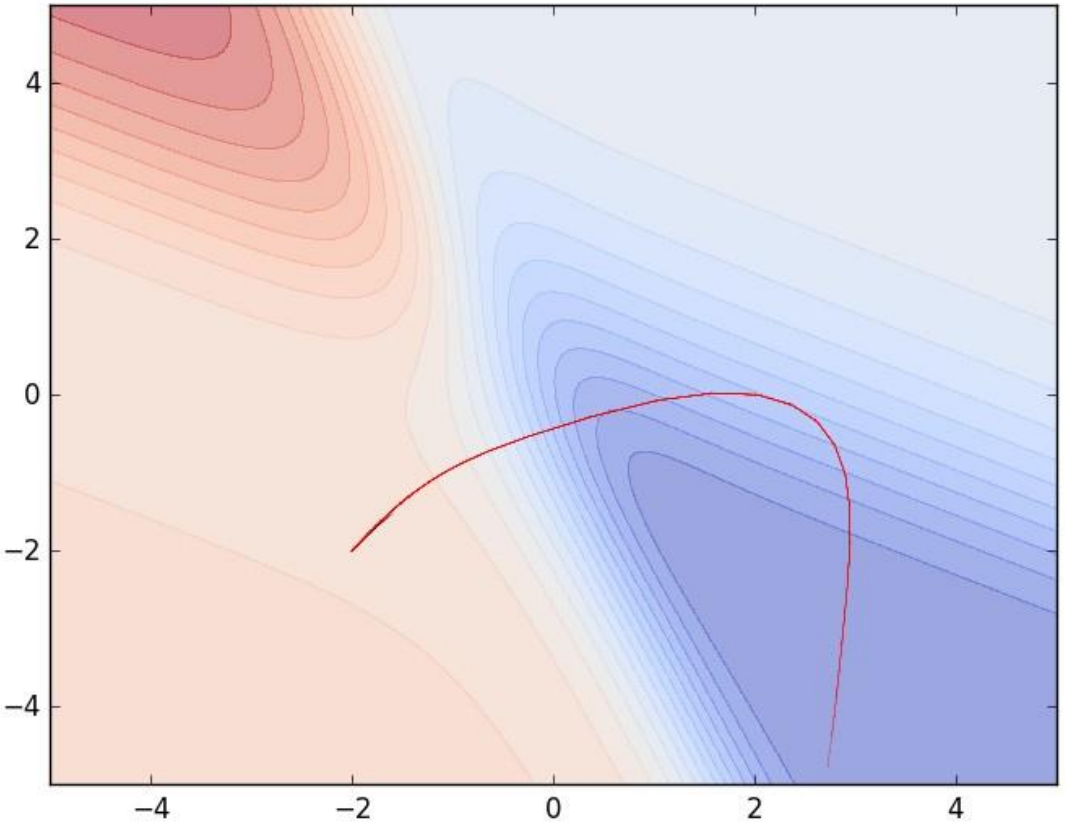
```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```


Gradient descent on the error surface

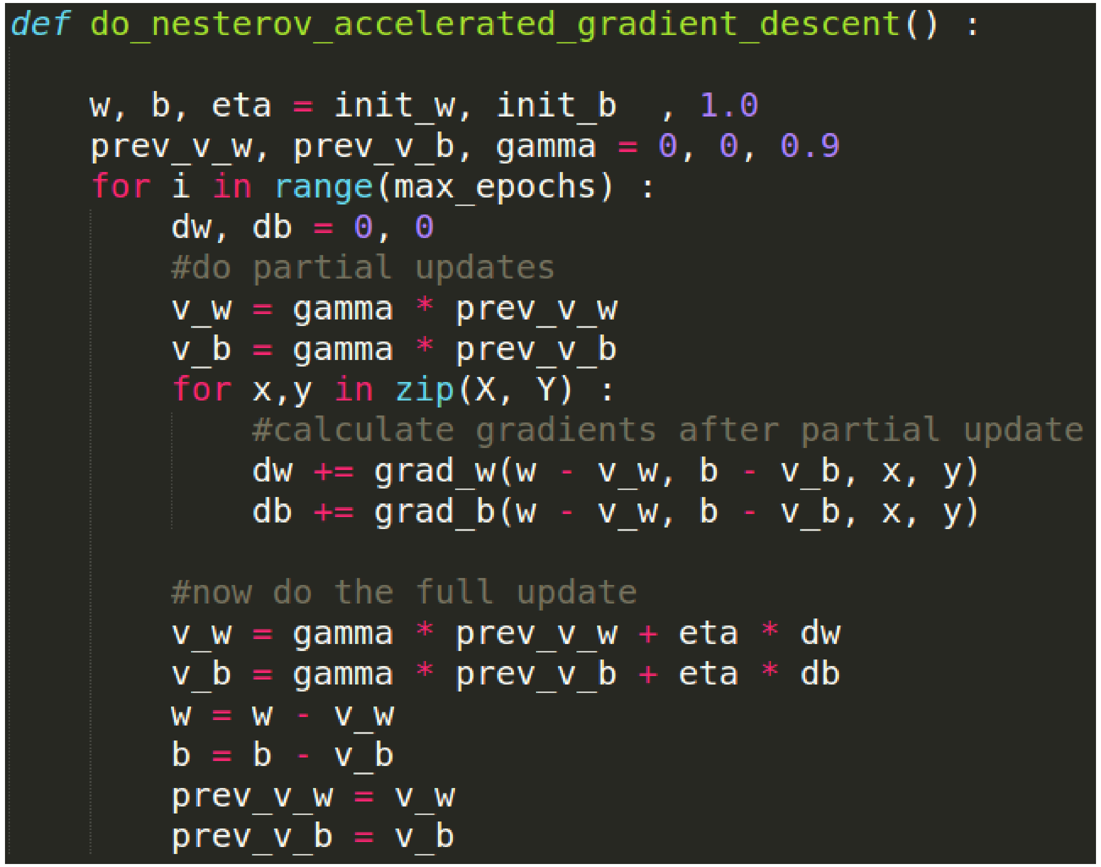
Optimizers(Momentum Gradient Descent)



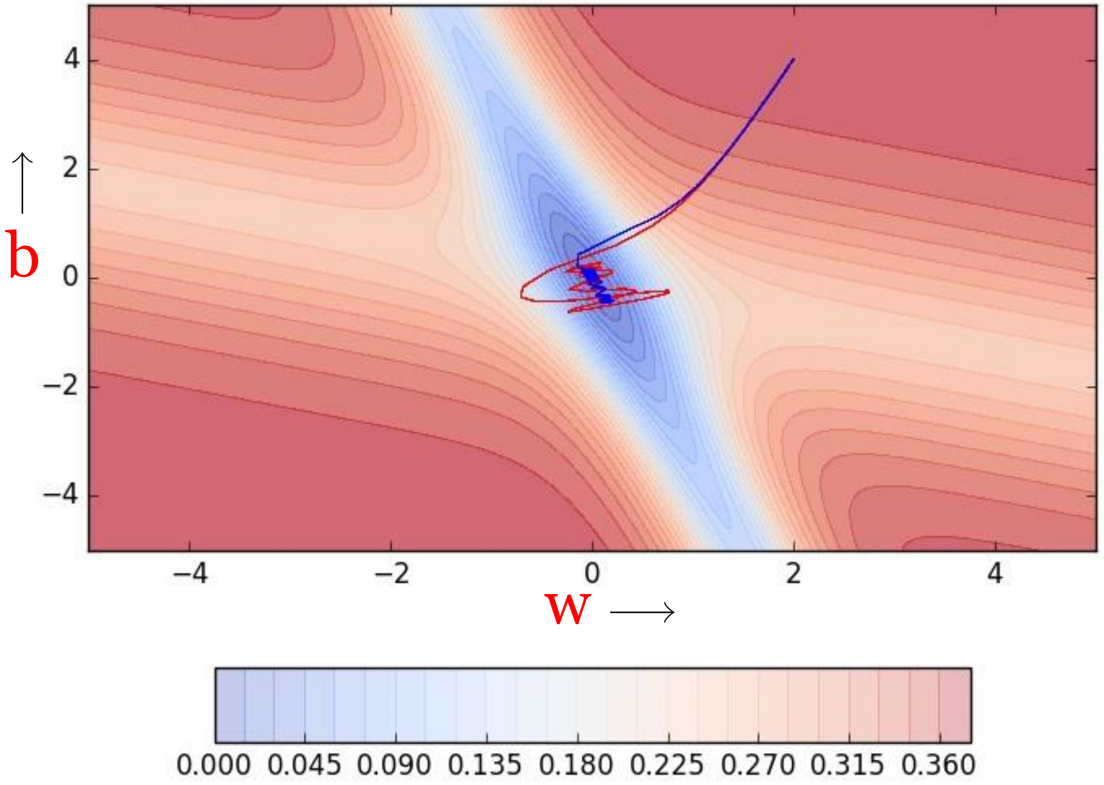
```python
def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



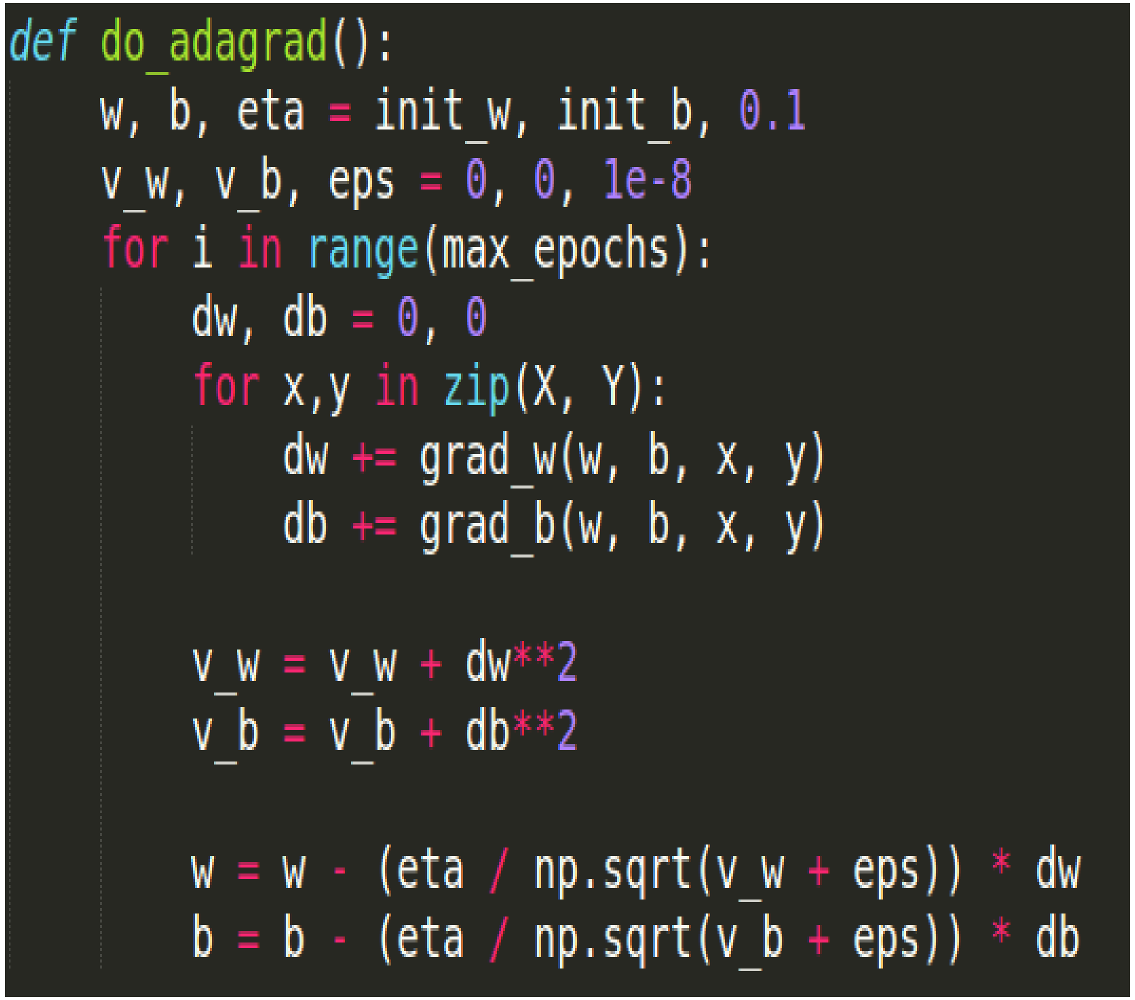
Optimizers(Nesterov Accelerated Gradient Descent)



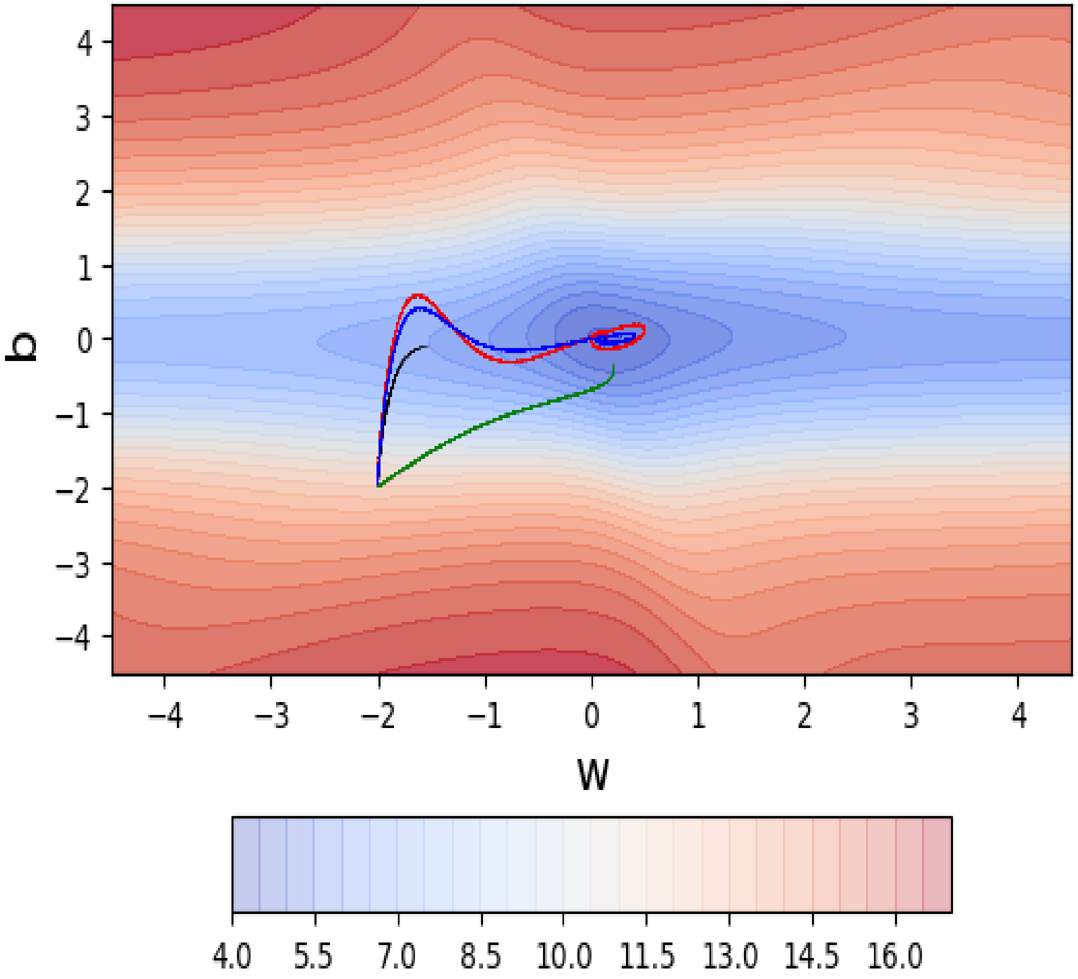
```
def do_nesterov_accelerated_gradient_descent() :

    w, b, eta = init_w, init_b  , 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        #do partial updates
        v_w = gamma * prev_v_w
        v_b = gamma * prev_v_b
        for x,y in zip(X, Y) :
            #calculate gradients after partial update
            dw += grad_w(w - v_w, b - v_b, x, y)
            db += grad_b(w - v_w, b - v_b, x, y)

        #now do the full update
        v_w = gamma * prev_v_w + eta * dw
        v_b = gamma * prev_v_b + eta * db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
```



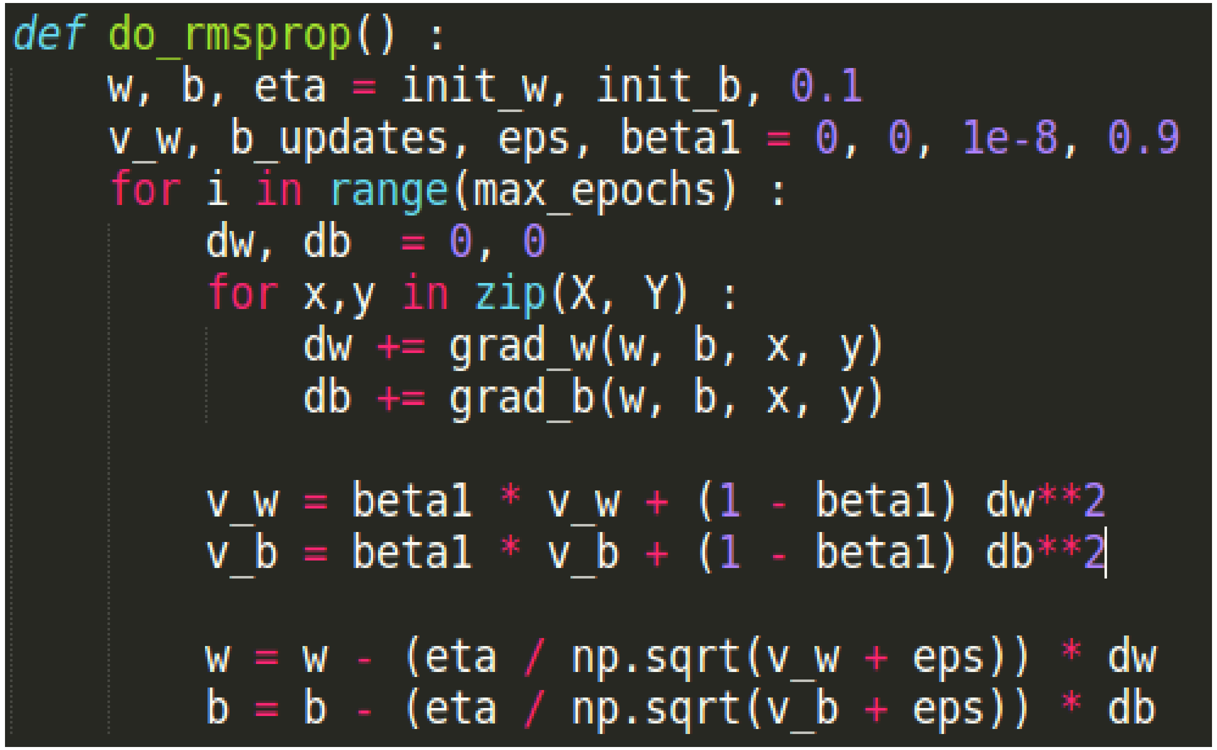
Optimizers(Adagrad)



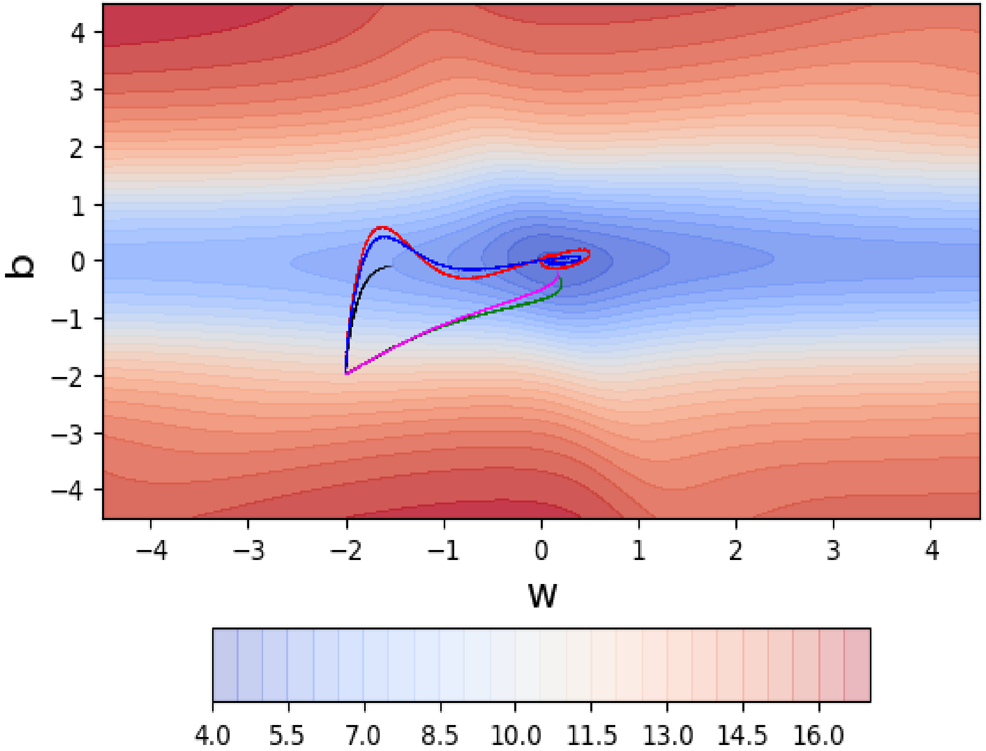
```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



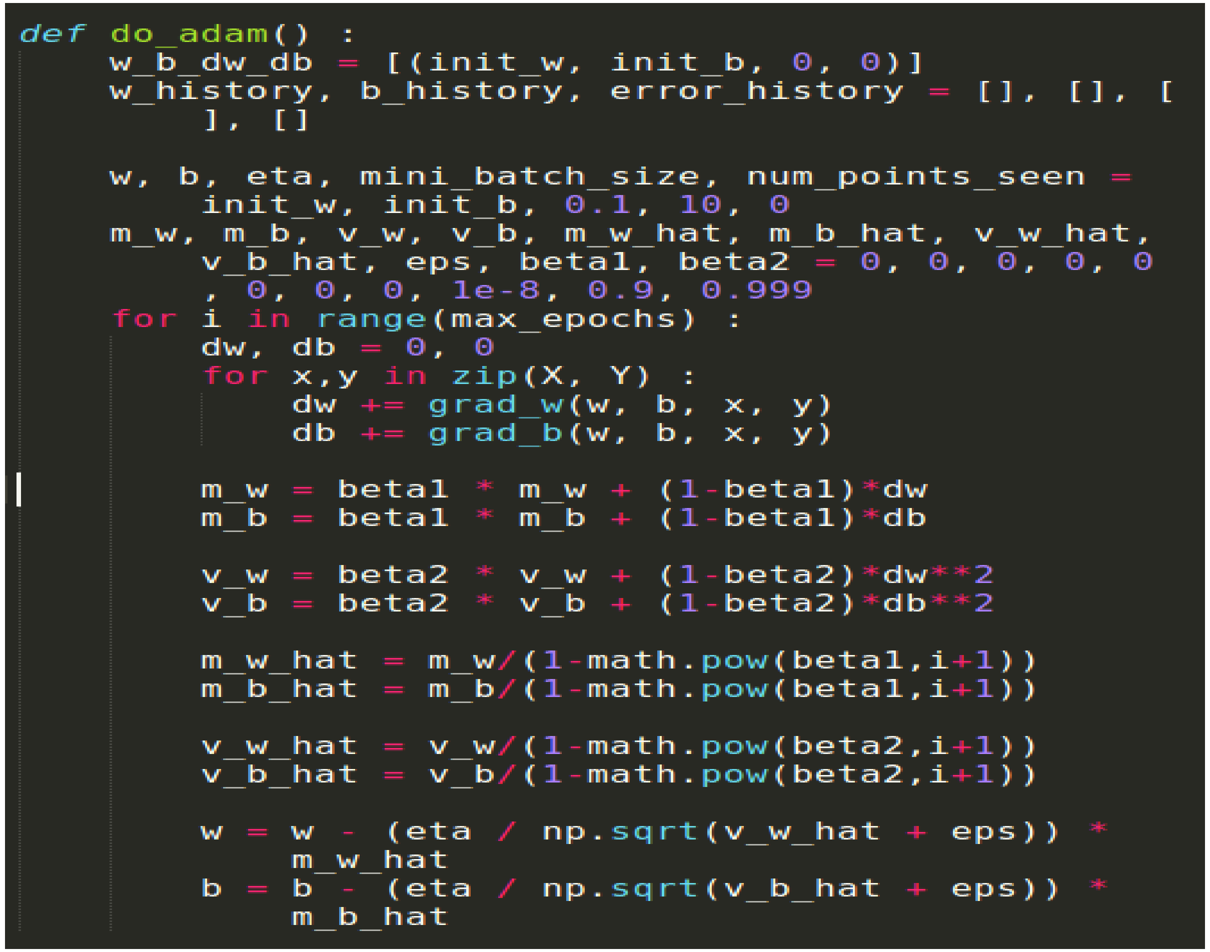
Optimizers(RmsProp)



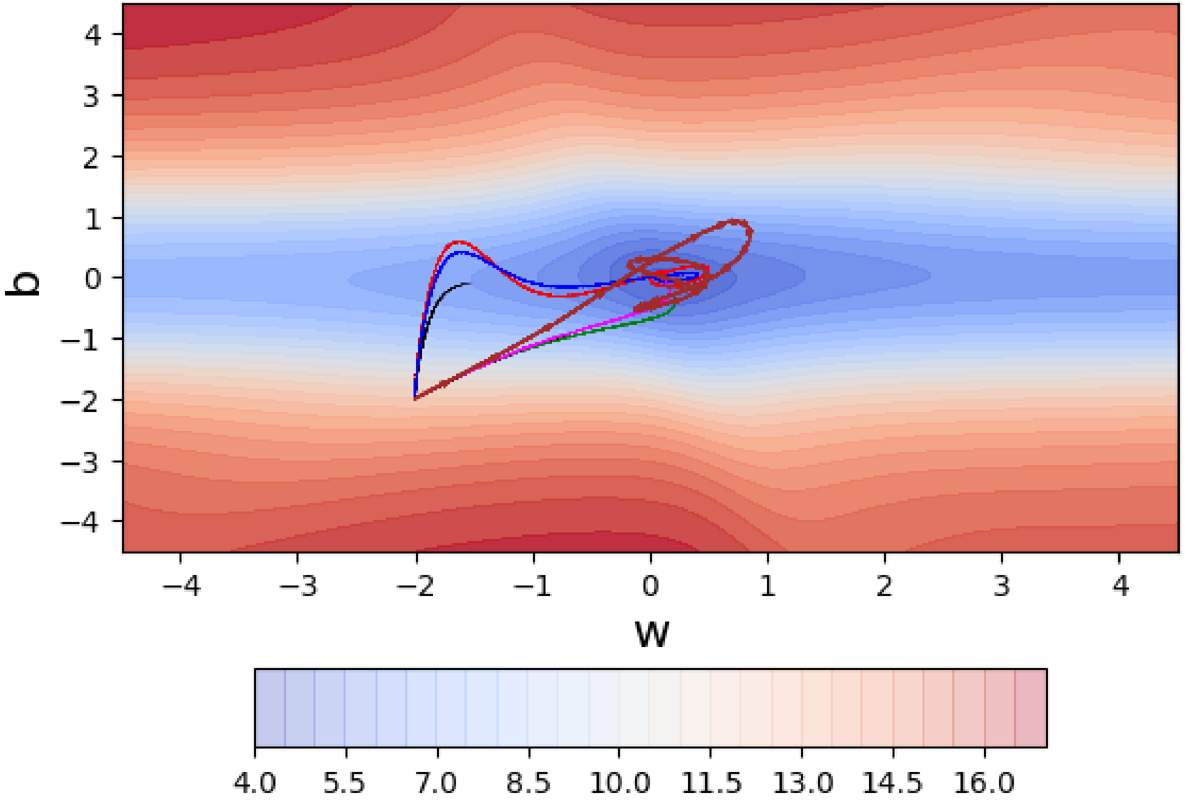
```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db  = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



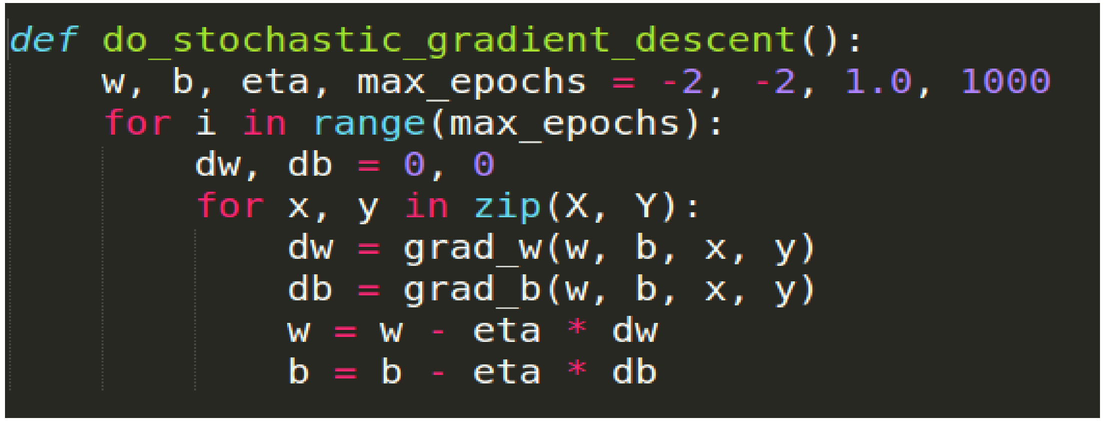
Optimizers(Adam)



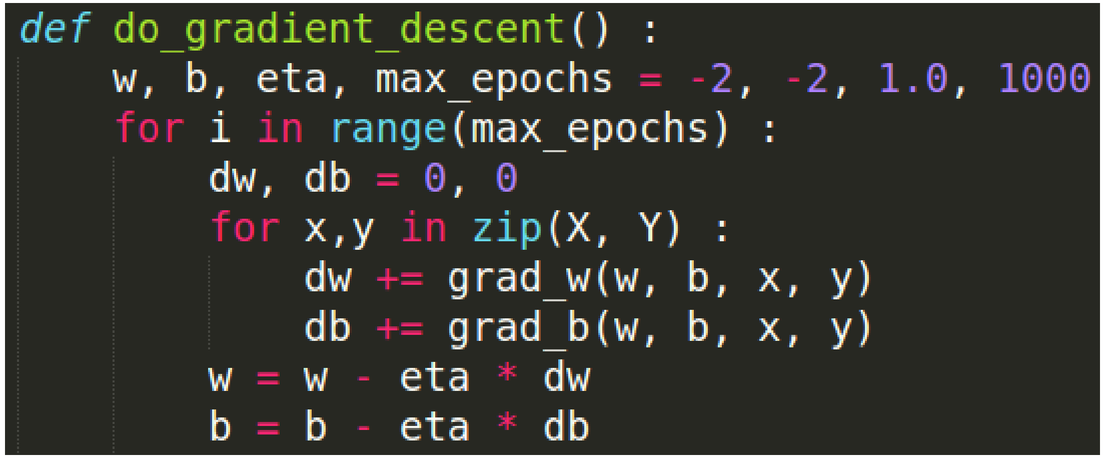
```
def do_adam() :
    w_b_dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [
        ], []

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, m_w_hat, m_b_hat, v_w_hat,
        v_b_hat, eps, beta1, beta2 = 0, 0, 0, 0, 0
        , 0, 0, 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        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-math.pow(beta1,i+1))
        m_b_hat = m_b/(1-math.pow(beta1,i+1))

        v_w_hat = v_w/(1-math.pow(beta2,i+1))
        v_b_hat = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w_hat + eps)) *
            m_w_hat
        b = b - (eta / np.sqrt(v_b_hat + eps)) *
            m_b_hat
```



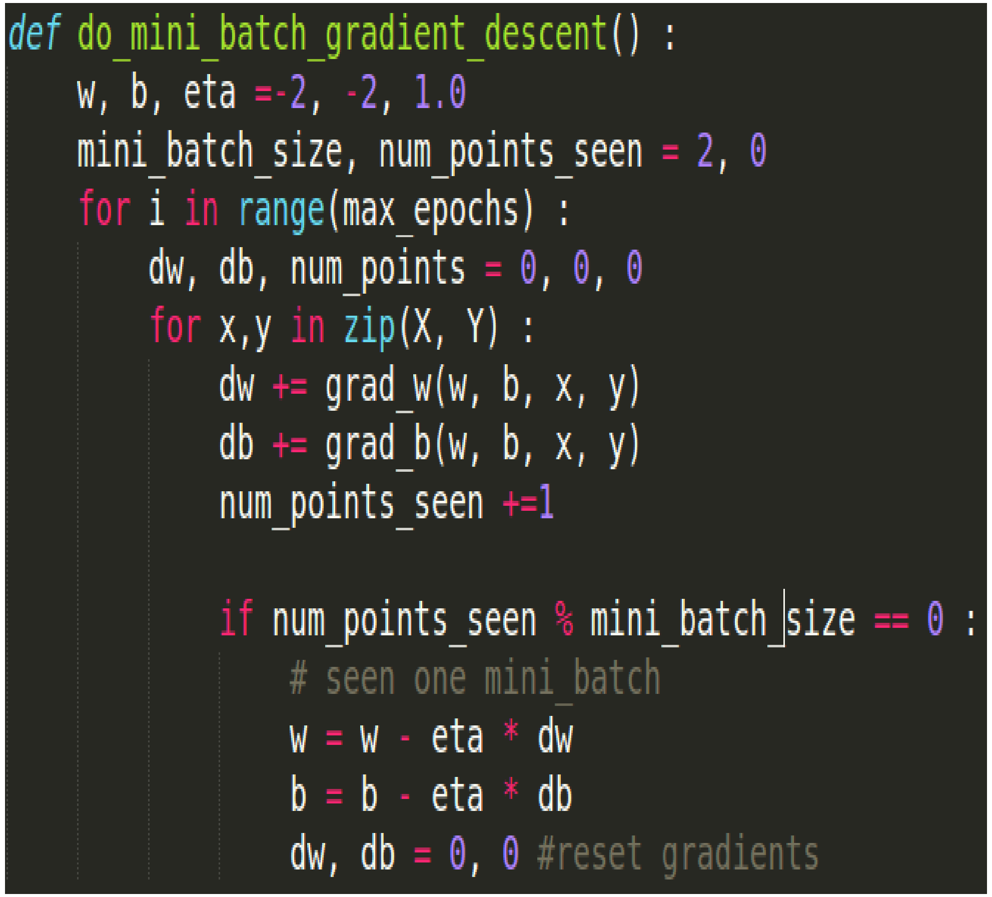
Stochastic ,Batch ,Mini-Batch



```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```



```
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```



```
def do_mini_batch_gradient_descent() :
    w, b, eta =-2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs) :
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen +=1

            if num_points_seen % mini_batch_size == 0 :
                # seen one mini_batch
                w = w - eta * dw
                b = b - eta * db
                dw, db = 0, 0 #reset gradients
```