

```

class MLP():
    def __init__(self, D_in, H, D_out):
        self.w1, self.b1 = np.random.normal(loc=0.0,
                                              scale=np.sqrt(2/(D_in+H)),
                                              size=(D_in, H)), np.zeros(H)
        self.w2, self.b2 = np.random.normal(loc=0.0,
                                              scale=np.sqrt(2/(H+D_out)),
                                              size=(H, D_out)), np.zeros(D_out)

        self.loss = mse
        self.grad_loss = grad_mse

    def __call__(self, x):
        self.h = np.dot(x, self.w1) + self.b1
        y_hat = np.dot(self.h, self.w2) + self.b2
        return self.final_activation(y_hat)

    def final_activation(self, x):
        return x

def mse(output, target):
    return 0.5*(output - target)**2

def grad_mse(output, target):
    return (output - target)

def fit(self, X, Y, epochs = 100, lr = 0.001):
    for e in range(epochs):
        for x, y in zip(X, Y):
            # add batch dimension
            x = x[None,:]
            y_pred = self(x)
            # loss function
            loss = self.loss(y_pred, y).mean()
            # Backprop
            # dl/dy
            dldy = self.grad_loss(y_pred, y)
            # dl/dw2 = dl/dy * dy/dw2
            grad_w2 = np.dot(self.h.T, dldy)
            grad_b2 = dldy
            # dl/dh = dl/dy * dy/dh
            dldh = np.dot(dldy, self.w2.T)
            # dl/dw1 = dl/dy * dy/dh * dh/dw1
            grad_w1 = np.dot(x.T, dldh)
            grad_b1 = dldh
            # Update (GD)
            self.w1 = self.w1 - lr * grad_w1
            self.b1 = self.b1 - lr * grad_b1
            self.w2 = self.w2 - lr * grad_w2
            self.b2 = self.b2 - lr * grad_b2

```

