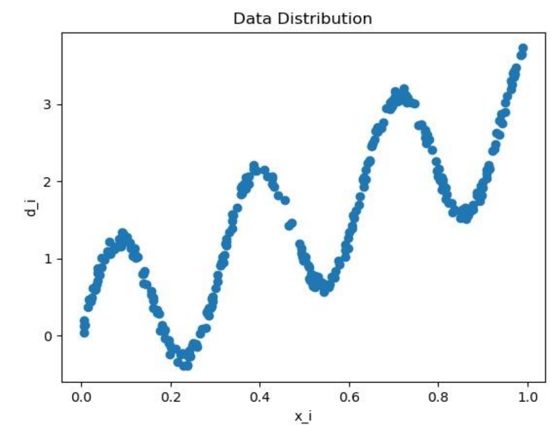
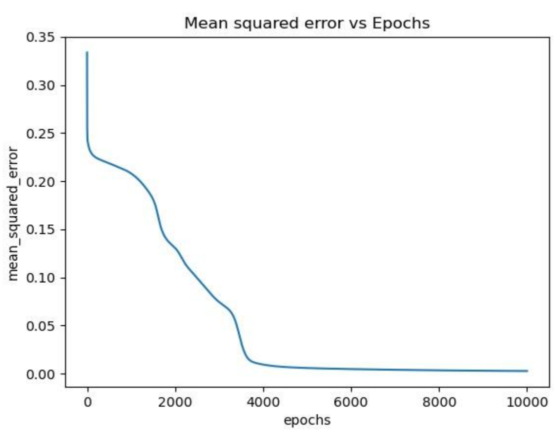
HW4:

Trained the network for 10000 epochs and the loss after 10000 epochs is 0.002583088101414392. The learning rate chosen is 0.01.
For loss I have used
$1/2\ \times (target - desired)^2$ so when calculating gradient we will not get 2.

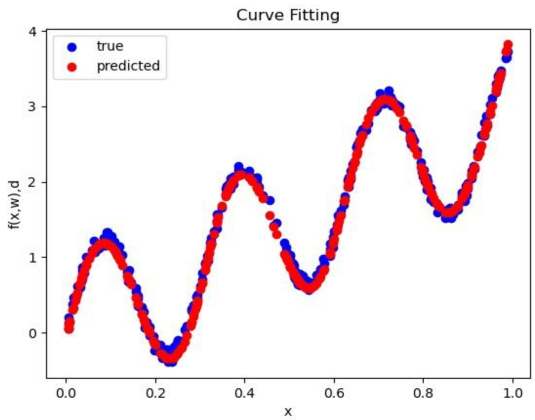
Q3)



Q4)



Q5)



Q6)

$$\text{loss} = \frac{1}{2}(o-y)^2$$

o - output of model

y - true output.

$\phi_1(v) = \tanh v \qquad \phi_2(v) = v$

$$o = \phi_2(\phi_1(w_h \times x + b_h) \times w_o + b_o)$$

$$= \tanh((w_h x + b_h)\, w_o + b_o) \quad \text{— (1)}$$

x - input $\quad w_o$ - weight of output layer

w_h - weight of hidden layer

b_h - bias of hidden layer $\quad b_o$ - bias of output layer

$$x \xrightarrow{w_h} \boxed{h} \xrightarrow{w_o} o$$

Gradients: ~~updates~~

$$\frac{\partial L}{\partial o} = +(o-y)$$

$$\frac{\partial L}{\partial w_h} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial h} \times \frac{\partial h}{\partial w_h}$$

$$\frac{\partial L}{\partial w_h} = (o-y) \times w_o \times \left[1 - \tanh^2(w_h x + b_h)\right] \cdot x \quad \text{— (2)}$$

$$\frac{\partial L}{\partial w_o} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial w_o} = (o-y) \times h \quad - (3)$$

$$\frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial b_o} = (o-y) \times 1 \quad - (4)$$

$$\frac{\partial L}{\partial b_h} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial h} \times \frac{\partial b}{\partial h}$$

$$\frac{\partial L}{\partial b_h} = (o-y) \times w_o \times \left(1-\tanh^2(w_h x + b_h)\right) \quad - (5)$$

Pseudocode:

1. Initialize weights
   w_o, w_h with random values
2. Initialize biases :
   b_o, b_h with random values.

~~epho epochs~~ . ~~lossez0~~
~~Initialize epochs~~

3. Initialize epochs to a fixed number usually a large value so that the network can get loss = 0 ~~(so)~~ ~~the network~~

Initialize learning rate η

4. ~~for epoch = 0 to epochs~~

for epoch = 0 to epoch < epochs :

~~losses~~ loss_per_epoch, η = 0, 0.01

for each point in data :

calculate o using equation 1

$loss = \cancel{loss} + \frac{1}{2}(o-y)^2$

calculate gradients using equation 2, 3, 4, 5

update weights

$w_o = w_o - \eta \times \frac{\partial L}{\partial w_o}$

$w_h = w_n - \eta \times \frac{\partial L}{\partial w_h}$

update bias

$b_o = b_o - \eta \times \frac{\partial L}{\partial b_o}$

$b_h = b_h - \eta \times \frac{\partial L}{\partial b_h}$

Loss_per_epoch += loss

end for

Loss_per_epoch = loss_per_epoch / length of data

if loss-per-epoch == 0 :
end the for loop.
end.

Code:

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
x = np.random.rand(300)
v = np.random.uniform(low=-0.1, high=0.1, size=300)
d = np.sin(20*x)+3*x+v

plt.scatter(x, d)
plt.xlabel("x_i")
plt.ylabel("d_i")
plt.title("Data Distribution")
plt.savefig("Q3.jpg")
plt.show()


class NeuralNetwork:
    def __init__(self, lr):
        np.random.seed(42)
        self.w_h = np.random.randn(24)
        self.w_o = np.random.randn(24)
        self.b_h = np.random.randn(24)
```

```
        self.b_o = np.random.randn(1)
        self.lr = lr
        self.dwo, self.dwh, self.dbh, self.dbo = np.zeros(24), np.zeros(24), np.zeros(24),
np.zeros(1)

    def forward(self, x):
        a = self.w_h * x + self.b_h
        h = np.tanh(self.w_h * x + self.b_h)
        o = np.sum(self.w_o * h) + self.b_o
        return o, h, a

    def backward(self, o, h, a, x, y):
        self.dwo += (o-y) * h
        self.dwh += (o-y) * self.w_o * (1 - (np.tanh(a))**2) * x
        self.dbo += (o-y)
        self.dbh += (o-y) * self.w_o * (1 - (np.tanh(a))**2)

    def zero_grad(self):
        self.dwo, self.dwh, self.dbh, self.dbo = np.zeros(24), np.zeros(24), np.zeros(24),
np.zeros(1)

    def weight_updates(self):
        self.w_o = self.w_o - self.lr * self.dwo
        self.w_h = self.w_h - self.lr * self.dwh
        self.b_o = self.b_o - self.lr * self.dbo
        self.b_h = self.b_h - self.lr * self.dbh

    def update_lr(self):
        self.lr /= 10


def square_loss(x, y):
    return 1/2 * ((x-y)**2)


def train_loop(model, data, target, loss_fn):
    o, h, a = model.forward(data)
    model.backward(o, h, a, data, target)
    loss = loss_fn(o, target)[0]
    model.weight_updates()
    model.zero_grad()
    return loss
```

```
epochs = 10000
model = NeuralNetwork(lr=0.01)
mean_squared_error = []
for epoch in range(0, epochs):
    loss_per_epoch = 0
    for i in range(0, 300):
        loss = train_loop(model, x[i], d[i], square_loss)
        loss_per_epoch += loss
    loss_per_epoch /= 300
    if epoch != 0 and loss_previous_epoch < loss_per_epoch:
        model.update_lr()
    loss_previous_epoch = loss_per_epoch
    mean_squared_error.append(loss_per_epoch)
    loss_previous_epoch = loss_per_epoch
print("Loss after 10000 epochs is ", loss_per_epoch)
plt.plot(range(0, epochs), mean_squared_error)
plt.xlabel("epochs")
plt.ylabel("mean_squared_error")
plt.title("Mean squared error vs Epochs")
plt.savefig("Q4.jpg")
plt.show()

fig, ax = plt.subplots()
blue = ax.scatter(x, d, color="blue")
for i in x:
    o = model.forward(i)[0][0]
    red = ax.scatter(i, o, color="red")
blue.set_label("true")
red.set_label("predicted")
plt.xlabel("x")
plt.ylabel("d")
plt.title("Curve Fitting")
plt.savefig("Q5.jpg")
plt.legend()
plt.show()
```