Answer 1: We pass the input training values to the neural network in **forward propagation**, and predict the corresponding output from it.
We then calculate error by comparing original outputs to predicted outputs through loss function, which is the feedback given to the network.
Then we update the parameters by differentiating the loss function with respect to parameters and subtract the obtained corresponding values and learning rate's product from the parameter. This is **backward propagation**.

Answer 2: x.shape = (n,4)
y.shape = (1,4)
Initialize:
W1= random array of shape (5,n)
B1 = random array of shape (5,1)
W2= random array of shape (1,5)
B2 = random array of shape (1,1)
Forward propagation:
Z1 = (W1*x+B1)
A1 = activation (Z1)
Z2 = (W2*A1+B2)
A2 = activation (Z2)
Backward propagation:
Error = loss(A2,y)
Update-
W1 = W1 - learning_rate* dError/dW1
B1 = B1 - learning_rate* dError/dB1
W2 = W2 - learning_rate* dError/dW2
B2 = B2 - learning_rate* dError/dB2
Generalization:
Forward propagation :
A[i] = activation (W[i]*A[i-1]+B[i])
Backward propagation :
W[i] = W[i] - learning_rate* dError/dW[i]
B[i] = B[i] - learning_rate* dError/dB[i]

Answer 3:

(a) Sigmoid = 1/(1+e^(-x)) = y
Differentiation = y(1-y) = e^-x/(1+e^-x)^2

(b) Relu = {
0 x<0
x x>=0

Differentiation = {
0 x<0
1 x>0

(c) $0.01x \quad x<0$

Leaky Relu = { $x \quad x>=0$

$0.01 \quad x<0$

Differentiation = { $1 \quad x>0$

(d) $\tanh = (e^{x} - e^{-x})/(e^{x} + e^{-x}) = y$

Differentiation = $1 - y^{2}$

(e) softmax (a[i]) = $e^{a[i]} / \sum_{j=0}^{k} e^{a[j]}$

Differentiation = D[j]S[i] = S[i](δij - S[j])