A Gentle Introduction to Neural Network

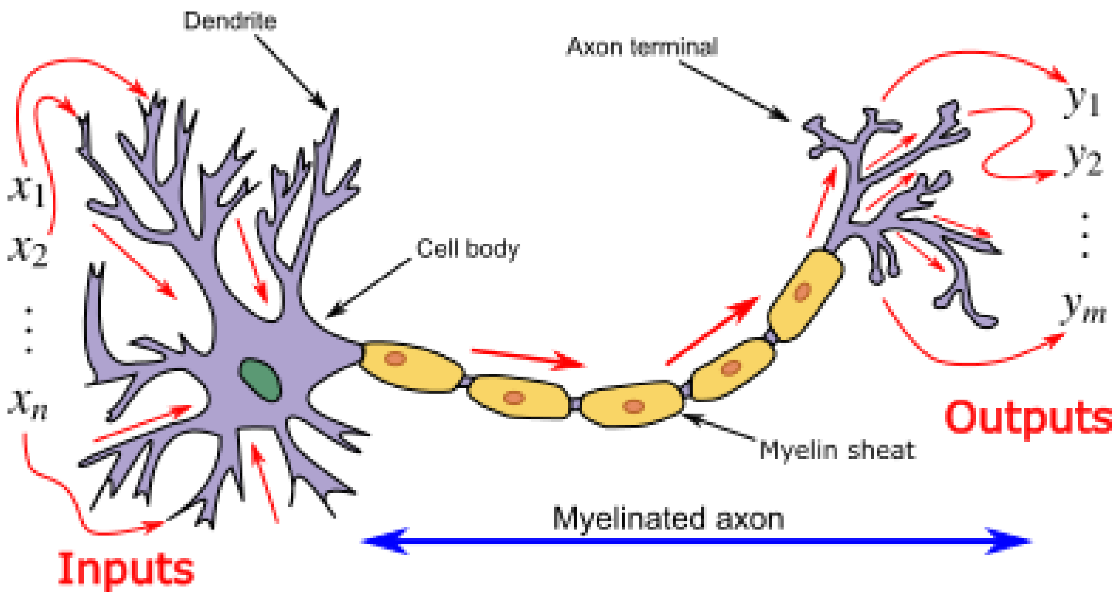
Rajdeep Banerjee

Topics covered

- Idea of perceptron
- A single neuron
- Network topology and assumptions
- Understanding notations
- Forward propagation
- Backward propagation
- Building a neural network from scratch

A biological Neuron

- The dendrites carry information from other neurons, axons send information to other neurons
- The signal carried by dendrites can get strengthened or weakened
- The output signal is a continuous analog signal.

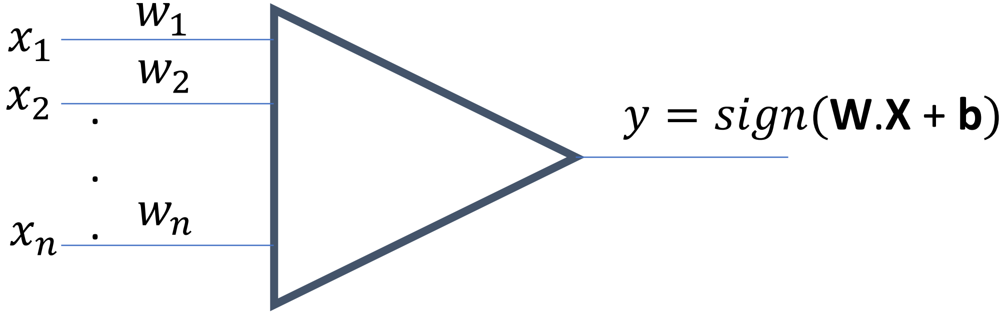


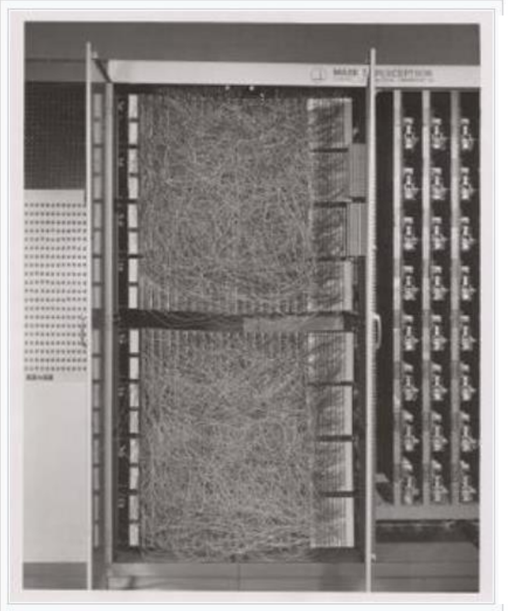
https://en.wikipedia.org/wiki/Biological_neuron_model

The idea of perceptron

- The idea of neural networks is more than 50 years old
- The first mathematical description of a neuron, by Frank Rosenblatt (1957)
- Input: $\mathbf{X} = [x_1 \; x_2 \; \ldots x_n]^T$
- Weights: $\mathbf{W} = [w_1 \; w_2 \; \ldots w_n]$
- Output: $\boldsymbol{y} = \boldsymbol{sign}(\boldsymbol{z}) = \boldsymbol{sign}(\mathbf{W}.\mathbf{X} + \mathbf{b})$
- $sign(z) = 1 \; z > 0$

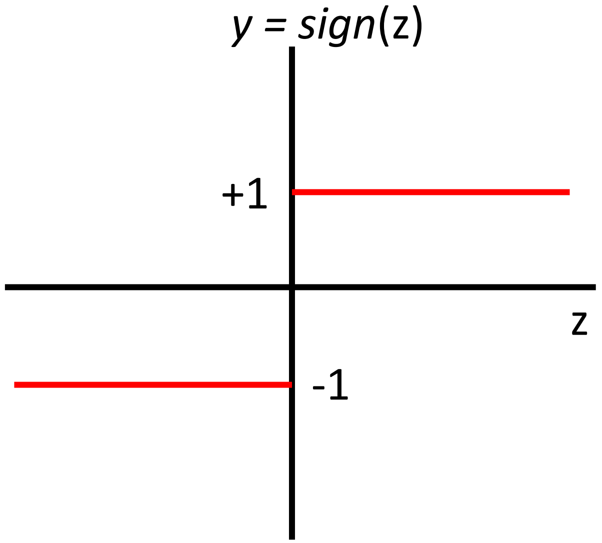
  $= -1 \text{ otherwise}$





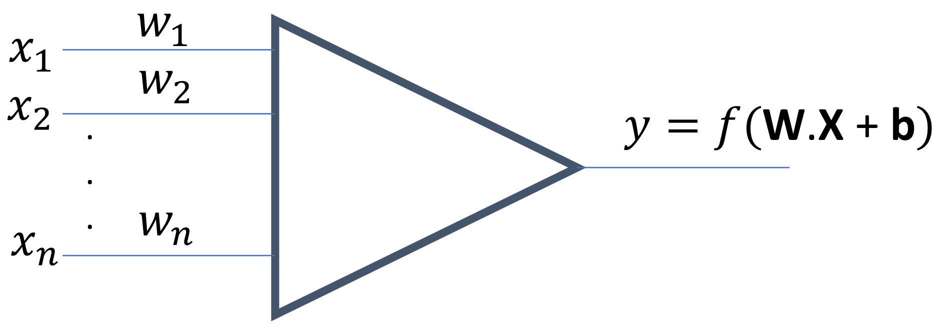
Mark I Perceptron machine, the first implementation of the perceptron algorithm. It was connected to a camera with 20×20 cadmium sulfide photocells to make a 400-pixel image. The main visible feature is a patch panel that set different combinations of input features. To the right, arrays of potentiometers that implemented the adaptive weights.[2]:213

https://en.wikipedia.org/wiki/Perceptron



A single neuron

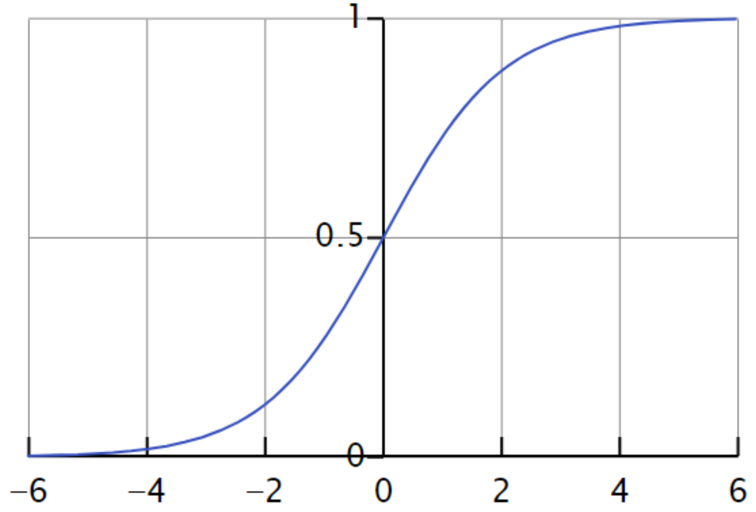
- A single neuron is very similar to perceptron where the function used can have a gradient instead of binary levels.



- *f*(x) is the activation function, there can be different types of activation function used, e.g. sigmoid, ReLU, tanh, etc.

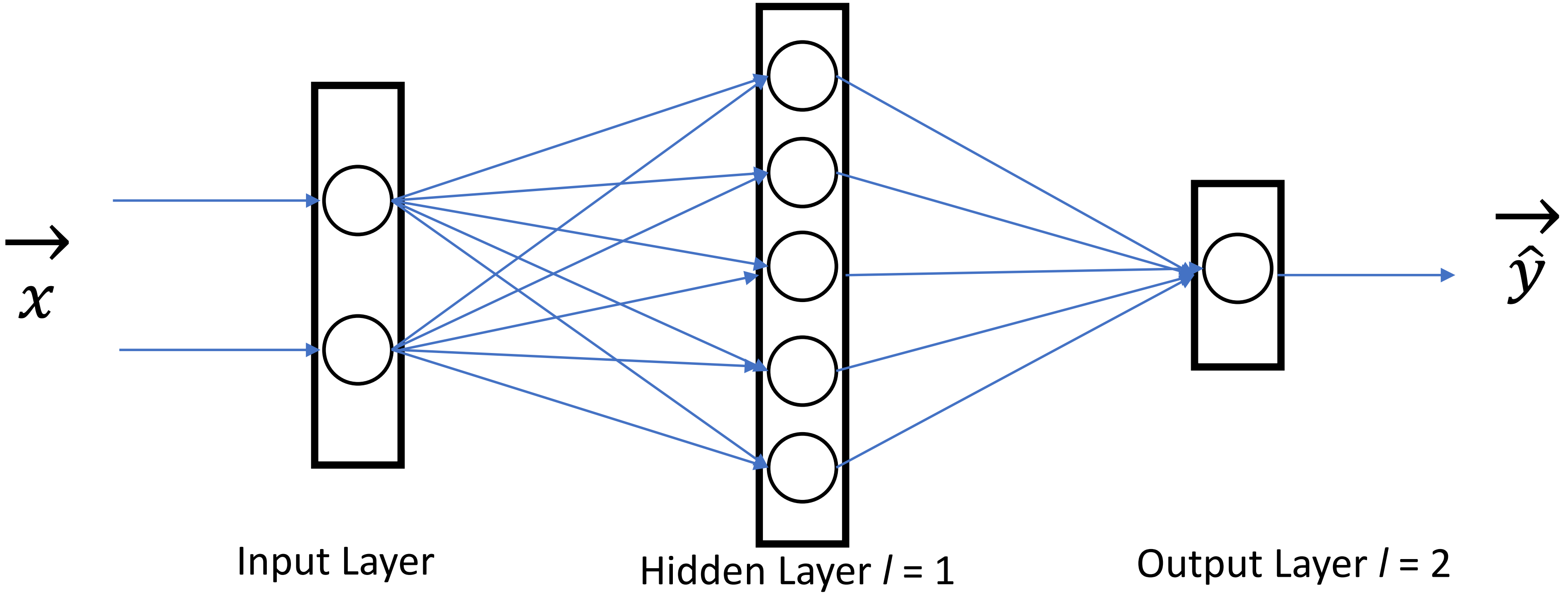
Sigmoid or Logistic function:

$$f(x) = \frac{1}{1 + e^{-x}}$$




```
In [7]: def sigmoid(z):
            return 1 / (1+np.exp(-z))
```

Example of a neural network: topology

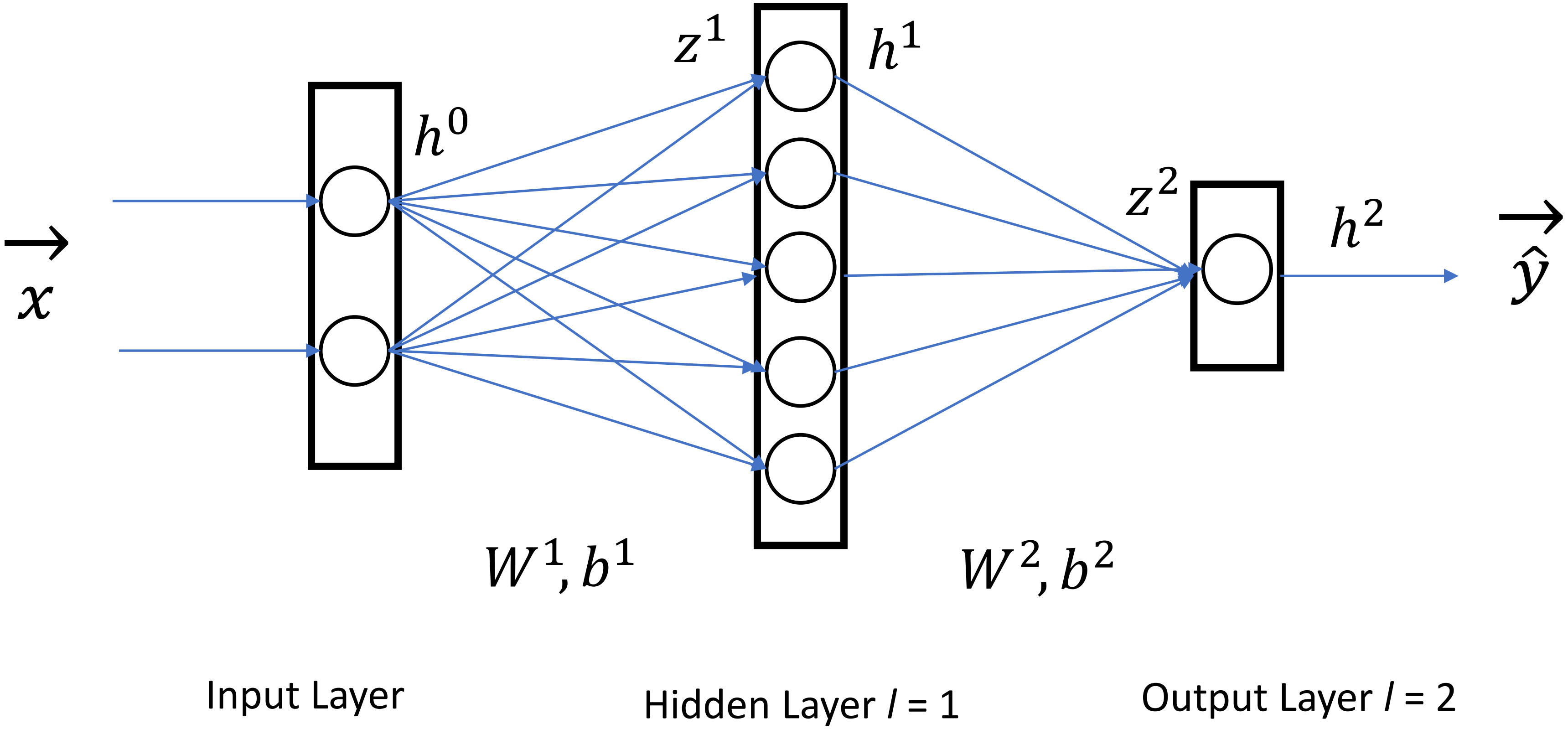


- Neurons are arranged in layers
- First layer is the input layer and the last is the output layer.
- Input layer has as many neurons as the number of features
- Output layer has as many neurons as the number of output levels (classification), 1 in case of regression
- The rest are hidden layers

Simplifying assumptions of NN

- Neurons are arranged in layers and the layers are arrange sequentially
- Neurons within the same layer do not interact with each other
- Neurons in consecutive layers are densely connected (there are ways to reduce number of connections, such as dropouts)
- Every interconnection has a weight associated with it, every neuron has a bias associated with it.
- All neurons in a particular layer use the same activation function.

Example: Fully connected (dense) NN

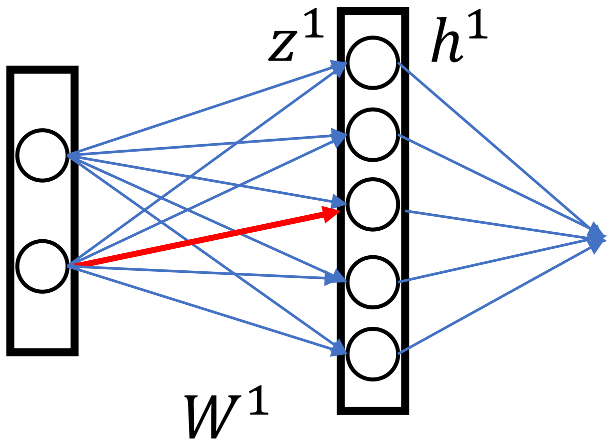


h^l : Output of layer l; a vector in general
z^l: Input to a node of layer l ; a vector in general
W^l : Weight matrix of layer l ; b^l: bias vector of layer l.

The shapes of W, h, z for a datapoint x

$$\mathrm{W}^l = \begin{pmatrix} w^l_{11} & w^l_{12} & \cdots & w^l_{1n} \\ \vdots & & & \\ w^l_{m1} & w^l_{m2} & \cdots & w^l_{mn} \end{pmatrix} \qquad z^l = \begin{bmatrix} z^l_1 \\ z^l_2 \\ \vdots \\ z^l_m \end{bmatrix} \quad h^l = \begin{bmatrix} x^l_1 \\ x^l_2 \\ \vdots \\ x^l_m \end{bmatrix}$$

- w_{ji} : the weight associated with j^{th} neuron of next layer to i^{th} neuron of the previous layer, e.g. weight associated with the connection in red w_{32}
- Shape of W (for each input): # of neurons in next layer (m) × # of neurons in previous layer (n)

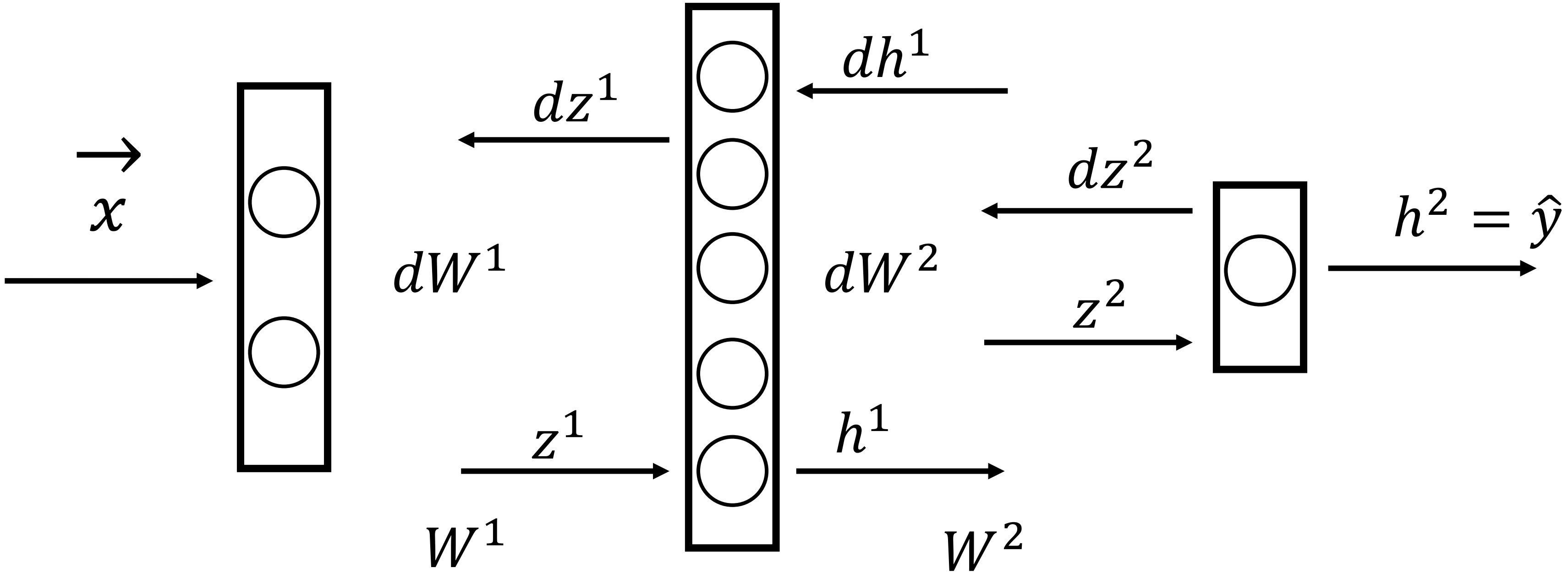


$$\mathrm{W}^1 = \begin{pmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \\ w^1_{31} & w^1_{32} \\ w^1_{41} & w^1_{42} \\ w^1_{51} & w^1_{52} \end{pmatrix} \qquad z^1 = \begin{bmatrix} z^1_1 \\ z^1_2 \\ z^1_3 \\ z^1_4 \\ z^1_5 \end{bmatrix} \quad h^1 = \begin{bmatrix} h^1_1 \\ h^1_2 \\ h^1_3 \\ h^1_4 \\ h^1_5 \end{bmatrix}$$

The Learning Loop: Gradient Descent

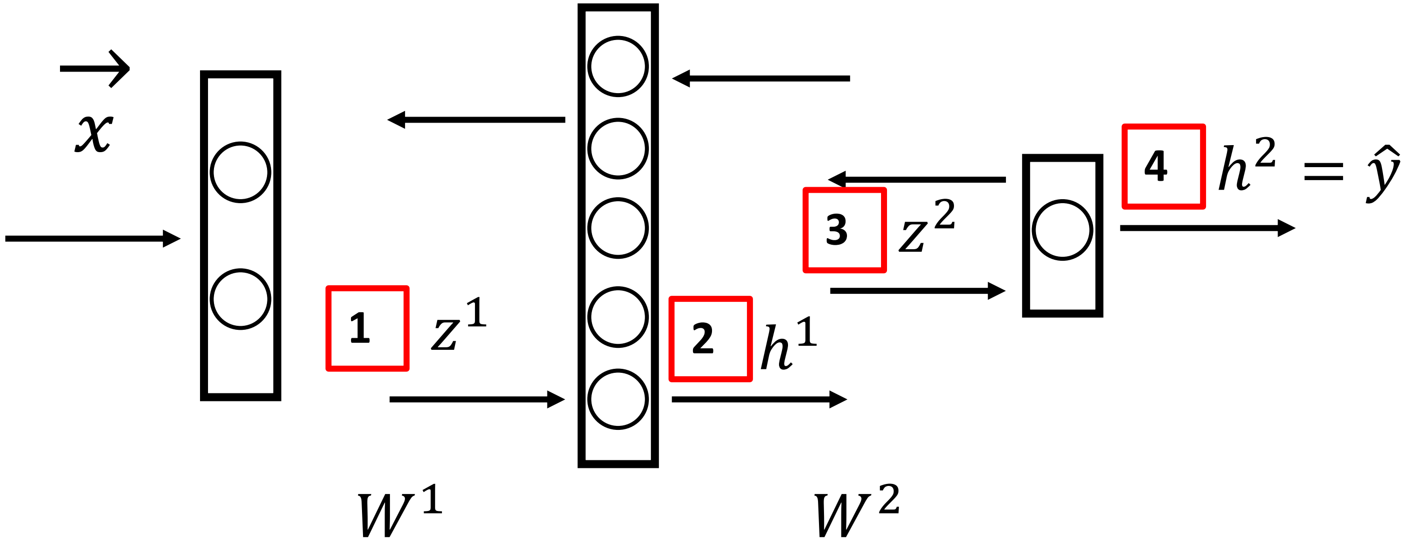
1. Propagate Forward: calculate all the parameters
2. Calculate the objective/cost/loss function
3. Calculate the gradient of loss wrt all parameters, i.e., propagate backward
4. Update the parameters using gradients calculated
5. Go to step 1 with updated parameters
6. An epoch is completed when you complete 1-5 for all the data points.

Forward and back propagation



- $dX = \frac{\partial L}{\partial x}$; $X = X - \alpha dX$ **(Gradient update rule)**
- Back propagation is a way of calculating the gradients *dX* throughout the network
- For simplicity we are only backpropagating *dW*

Forward propagation equations



h^l : Output of layer *l*
z^l: Input to a node of layer *l*
W^l : Weight matrix of layer *l*
b^l: bias vector of layer *l*.

$\boxed{1}$ $$z^1 = W^1.h^0 + b^1 = W^1.x + b^1$$

$\boxed{2}$ $$h^1 = \sigma(z^1) = \sigma(W^1.x + b^1)$$

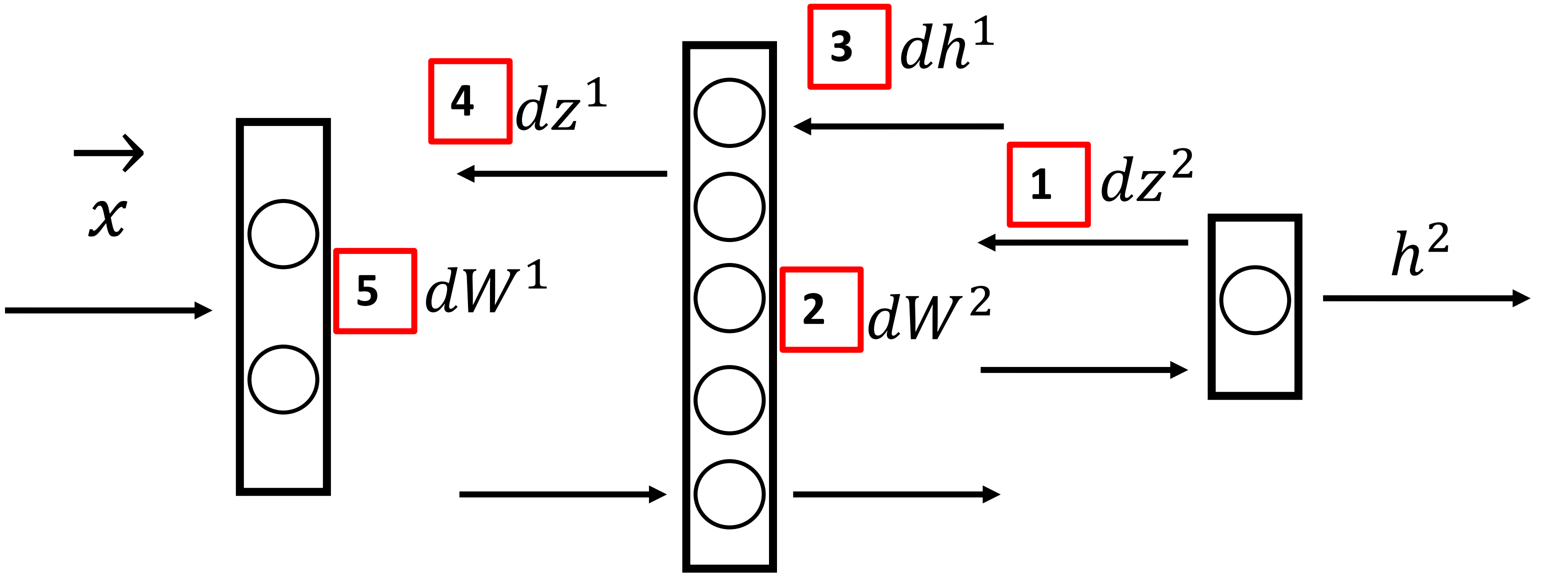
$\boxed{3}$ $$z^2 = W^2.h^1 + b^2 = W^2(\sigma(W^1.x + b^1)) + b^2$$

$\boxed{4}$ $$\hat{y} = h^2 = \sigma(z^2) = \sigma(W^2.h^1 + b^2)$$

In [9]:
```
def forward_prop(x,W1,W2):
    z1 = np.dot(W1,x) + b1
    #print(z1.shape)
    h1 = sigmoid(z1)
    #print(h1.shape)
    z2 = np.dot(W2,h1) + b2
    #print(z2.shape)
    y_hat = sigmoid(z2)
    #print(y_hat.shape)

    return z1,h1,z2,y_hat
```

Back propagation equations



General chain rule of partial differentiation:

$$\frac{\partial f(u,v,w)}{\partial w} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial w}$$

If v = g(w) and u = h(v) = h(g(w))

$\boxed{1}$

$$dz^2 = \frac{\partial L}{\partial z^2}$$
$$= \frac{\partial L}{\partial h^2}\frac{\partial h^2}{\partial z^2}$$
$$= (\hat{y} - y)\sigma'(z^2)$$

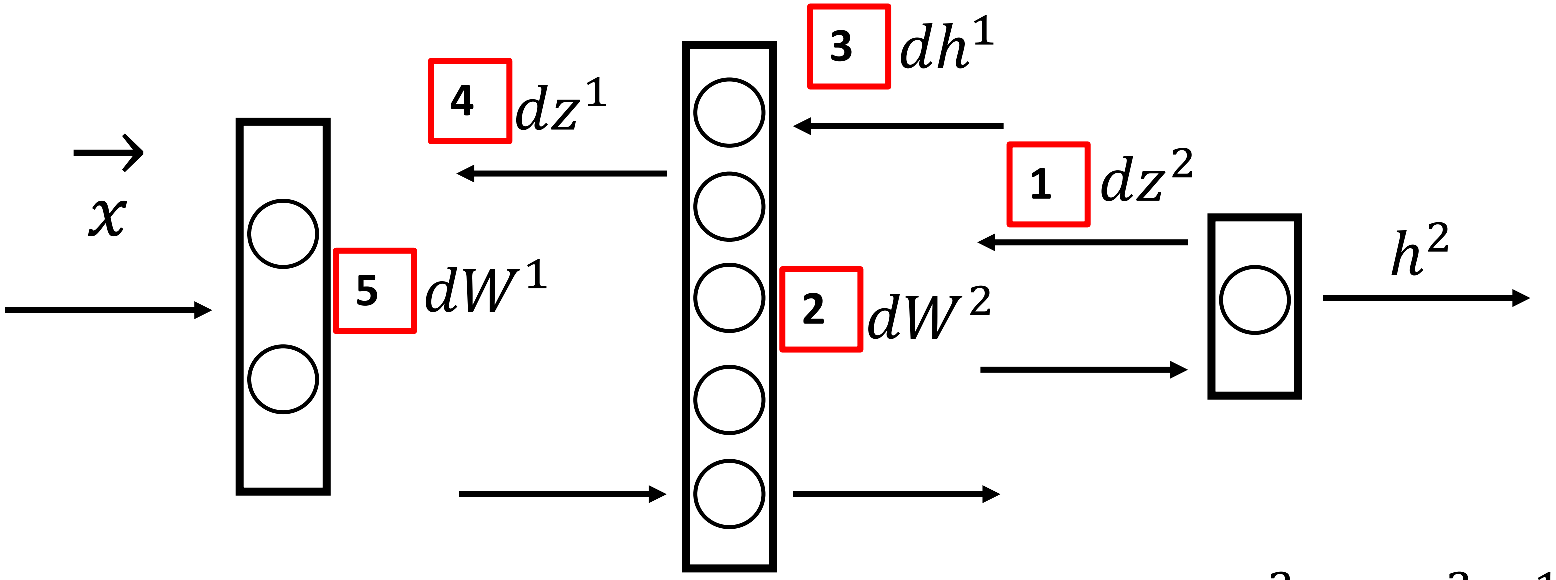
$L = \frac{1}{n}\sum_n(\hat{y} - y)^2$;

Here, n = 2 => $L = \frac{1}{2}\sum_n(\hat{y} - y)^2$

Therefore, $\frac{\partial L}{\partial h^2}$ = $\frac{\partial L}{\partial \hat{y}} = \hat{y} - y$

and, $h^2 = \sigma(z^2)$; $\frac{\partial h^2}{\partial z^2} = \sigma'(z^2)$

Back propagation equations



General chain rule of partial differentiation:

$$\frac{\partial f(u,v,w)}{\partial w} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial w}$$

If v = g(w) and u = h(v) = h(g(w))

$$z^2 = W^2.h^1 + b^2 = w^1 h_1^1 + \cdots + w^5 h_5^1 + b^2$$

$$\frac{\partial z^2}{\partial W^2} = \frac{\partial}{\partial W^2}(w^1 h_1^1 + \cdots + w^5 h_5^1 + b^2)$$

$$= (\frac{\partial}{\partial w^1}(w^1 h_1^1 + \cdots + w^5 h_5^1 + b^2) \ldots$$

$$\frac{\partial}{\partial w^5}(w^1 h_1^1 + \cdots + w^5 h_5^1 + b^2))$$

$$= \left(h_1^1 \ldots h_5^1\right) = (h^1)^T$$

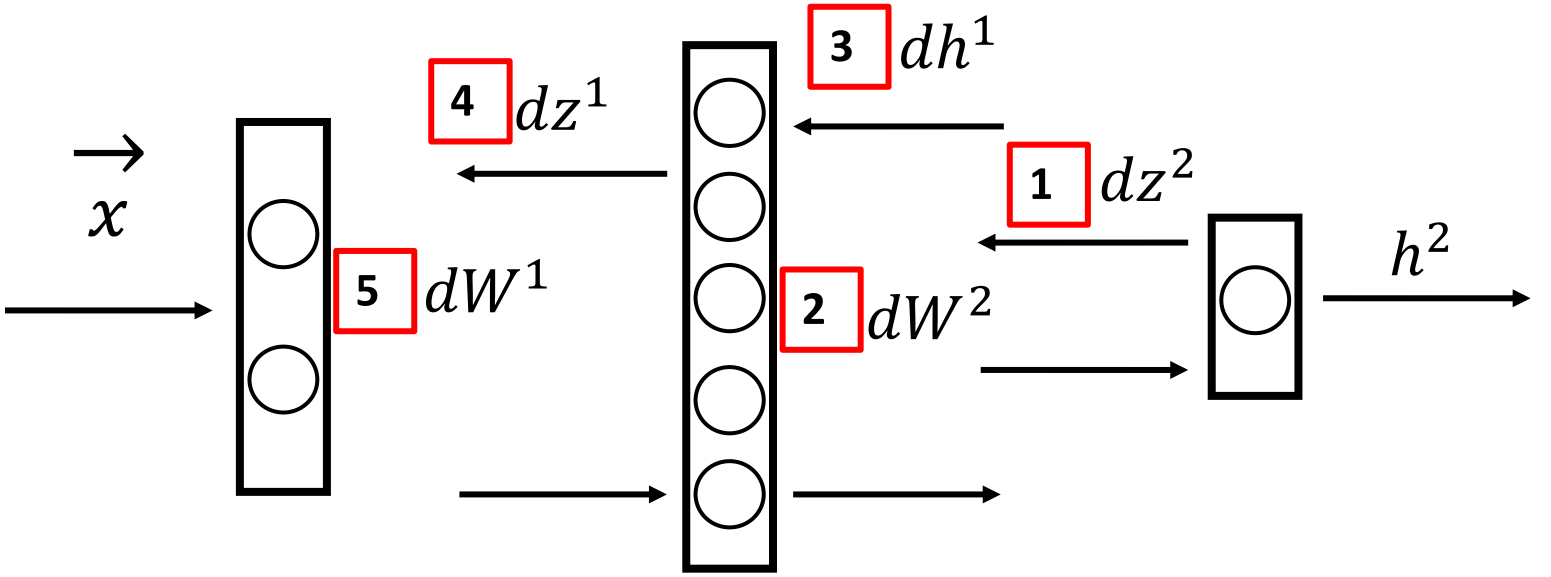
$\boxed{2}$

$$dW^2 = \frac{\partial L}{\partial W^2}$$

$$= \frac{\partial L}{\partial z^2}\frac{\partial z^2}{\partial W^2} = dz^2 (h^1)^T$$

Note: dz^2 was found from $\boxed{1}$

Back propagation equations



General chain rule of partial differentiation:

$$\frac{\partial f(u,v,w)}{\partial w} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial w}$$

If v = g(w) and u = h(v) = h(g(w))

$\boxed{3}$

$$dh^1 = \frac{\partial L}{\partial h^1}$$

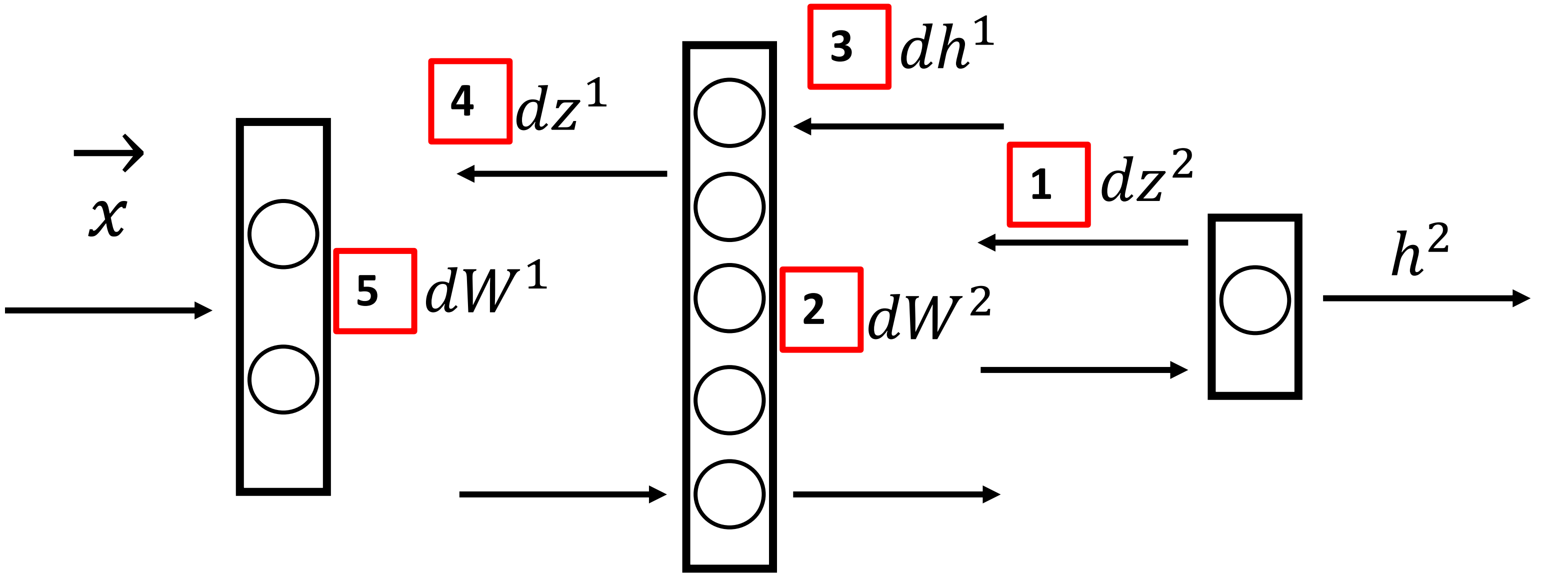
$$= \frac{\partial L}{\partial z^2}\frac{\partial z^2}{\partial h^1} = dz^2 W^2$$

Note: dz^2 was found from $\boxed{1}$

$$z^2 = W^2 . h^1 + b^2$$

$$\frac{\partial z^2}{\partial h^1} = W^2$$

Back propagation equations



General chain rule of partial differentiation:

$$\frac{\partial f(u,v,w)}{\partial w} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial w}$$

If v = g(w) and u = h(v) = h(g(w))

$$\boxed{4} \quad dz^1 = \frac{\partial L}{\partial z^1} = \frac{\partial L}{\partial h^1}\frac{\partial h^1}{\partial z^1}$$

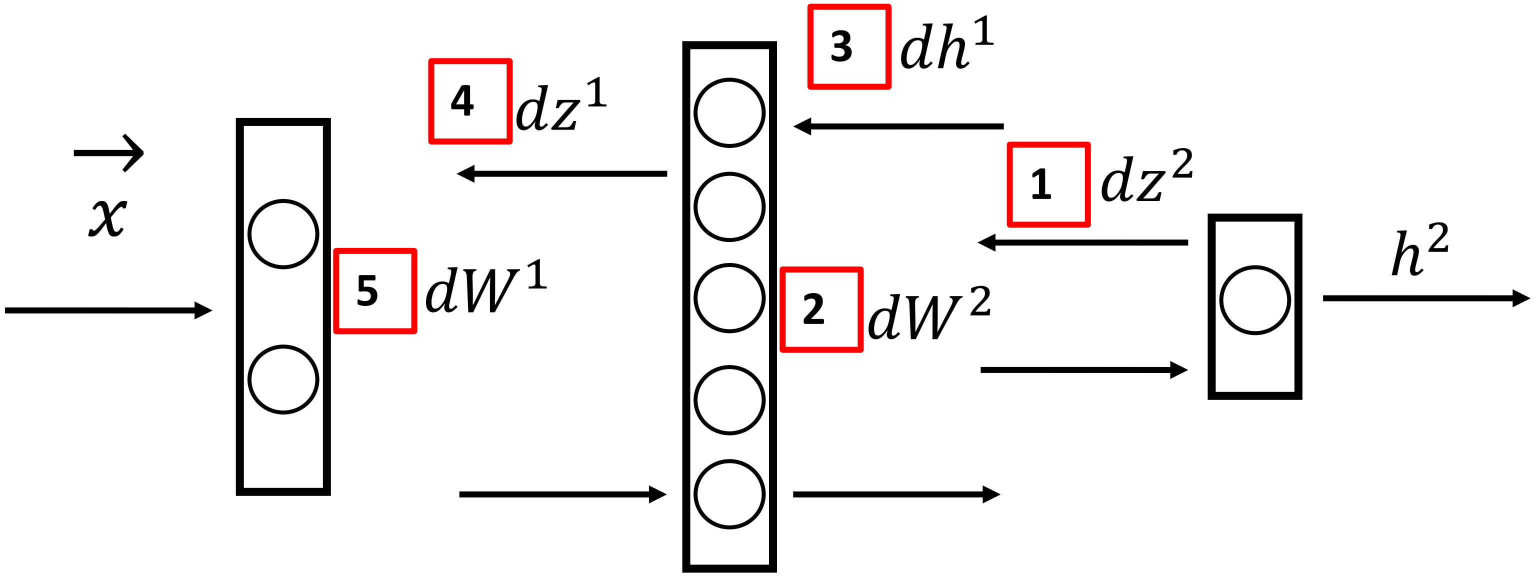
$$= dh^1\,\sigma'(z^1)$$

Note: dh^1 was found from $\boxed{3}$

$$h^1 = \sigma(z^1)$$

$$\frac{\partial h^1}{\partial z^1} = \sigma'(z^1)$$

Back propagation equations



General chain rule of partial differentiation:

$$\frac{\partial f(u,v,w)}{\partial w} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial v}\frac{\partial v}{\partial w}$$

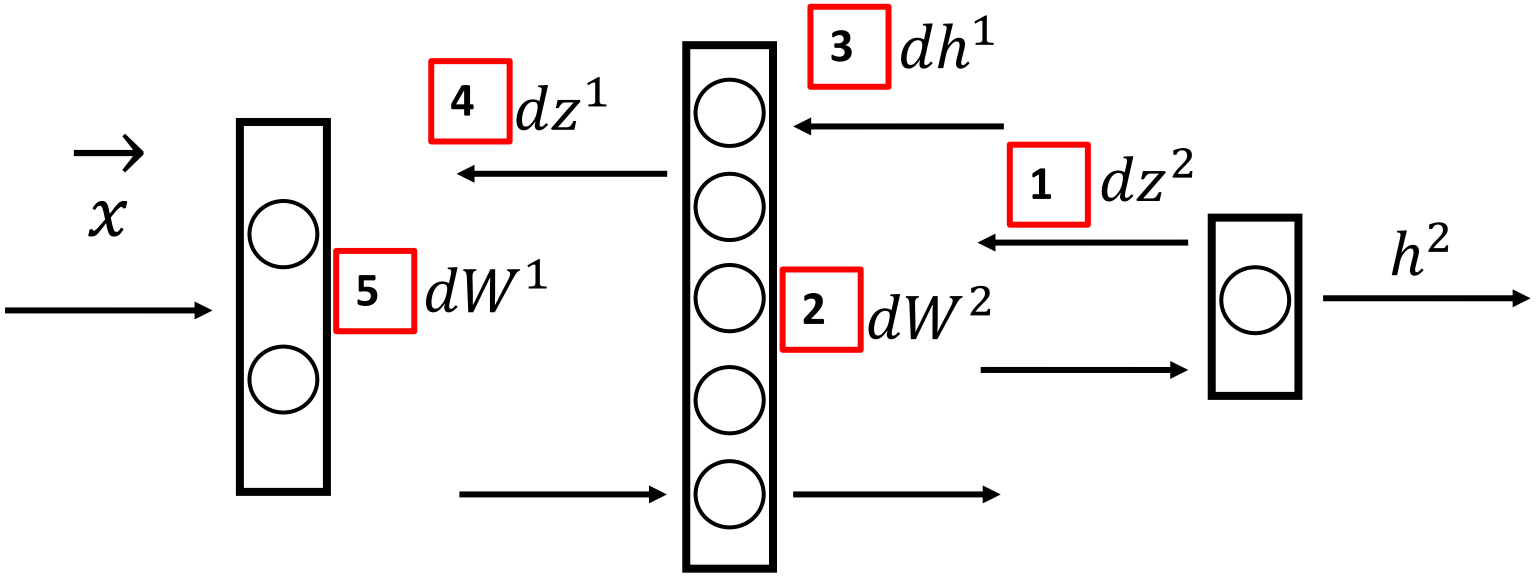
If v = g(w) and u = h(v) = h(g(w))

5 $$dW^1 = \frac{\partial L}{\partial W^1}$$

$$= \frac{\partial L}{\partial z^1}\frac{\partial z^1}{\partial W^1} = dz^1(h^0)^T = dz^1.x^T$$

Note: dz^1 was found from **4**

Back propagation equations (summary)



1 $dz^2 = (\hat{y} - y)\sigma'(z^2)$

2 $dW^2 = dz^2(h^1)^T$

3 $dh^1 = dz^2 W^2$

4 $dz^1 = dh^1\,\sigma'(z^1)$

5 $dW^1 = dz^1.x^T$

In [8]:
```
def sigmoid_derivative(z):
    return np.exp(-z)/((1+np.exp(-z))**2)
```

In [10]:
```
def backword_prop(y_hat, z1, h1, z2):

    dz2 = np.multiply((y_hat-y),sigmoid_derivative(z2))
    dW2 = np.dot(dz2, h1.T)
    dh1 = np.dot(W2.T, dz2)
    dz1 = np.multiply(dh1, sigmoid_derivative(z1))
    dW1 = np.dot(dz1, x.T)

    return dW1, dW2
```

Summary: The Pseudocode (generalized for L hidden layers)

1. $h^0 = x$
2. for *l* in [1,2, …, L]:
   $h^l = \sigma(W^l . h^{l-1} + b^l)$
3. $\hat{y} = h^{L+1}$
4. $L = \frac{1}{n}\sum_n(\hat{y} - y)^2$; n =2 in our case
5. $dz^O = (\hat{y} - y)$; since $\sigma'(z) = 1$ in our case
6. $dW^O = dz^O(h^L)^T$
7. for *l* in [L, L-1, …, 1]:
   1. $dh^l = W^{(l+1)^T} dz^{(l+1)}$
   2. $dz^l = dh^l \otimes \sigma'(z^l)$
   3. $dW^l = dz^l(h^{l-1})^T$

   Loop over hidden layers
8. $W_{new} = W_{old} - \alpha dW$; Update rule (similar for biases)

[54]:
```
alpha = 0.01
num_iterations = 5000
```

[55]:
```
cost = []
for i in range(num_iterations):

    #perform forward propagation and predict output
    z1,h1,z2,y_hat = forward_prop(x,W1,W2)

    #perform backward propagation and calculate gradients
    dW1, dW2 = backword_prop(y_hat, z1, h1, z2)

    #update the weights
    W1 = W1 -alpha * dW1
    W2 = W2 -alpha * dW2

    #compute cost
    c = cost_function(y, y_hat)
    #print(c)
    #store the cost
    cost.append(c)
```

Using computational graph

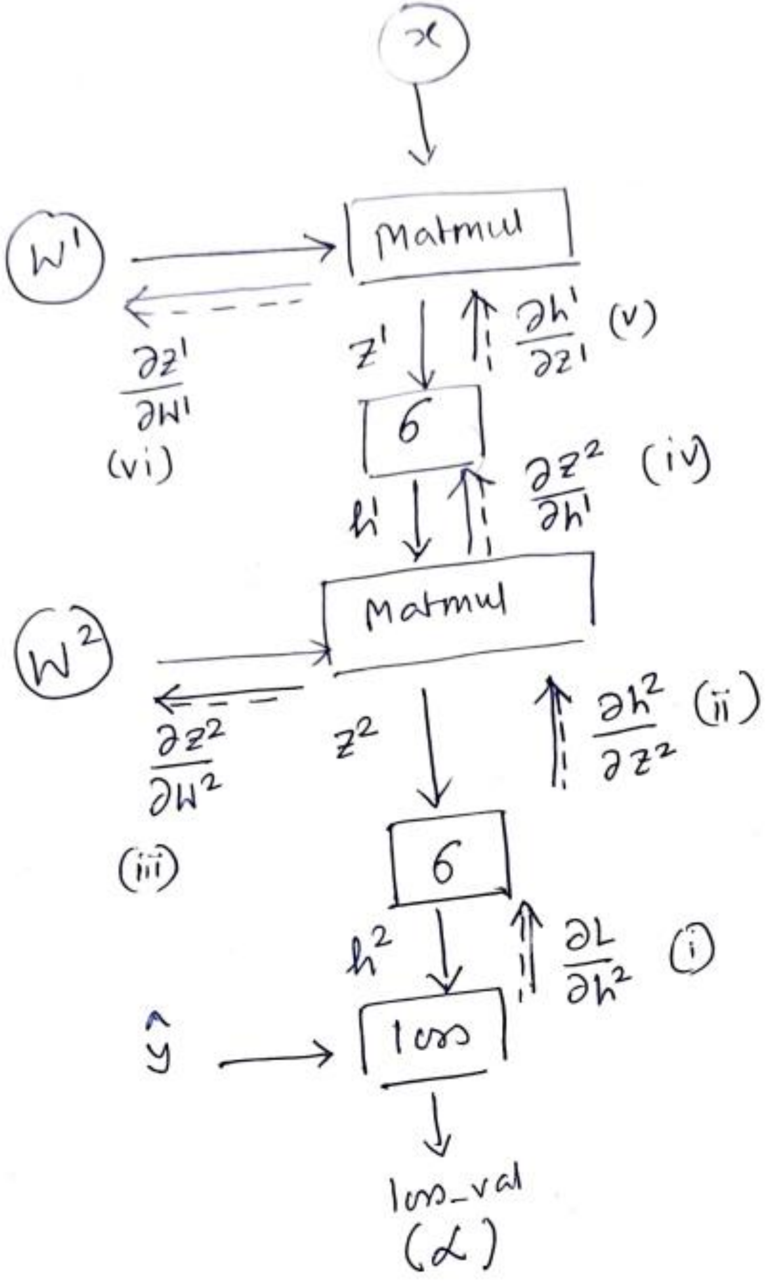
① $dz^2 = \frac{\partial L}{\partial z^2} = \frac{\partial L}{\partial h^2} \cdot \frac{\partial h^2}{\partial z^2} = (i) \times (ii)$

② $dW^2 = \frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial z^2} \cdot \frac{\partial z^2}{\partial W^2} = (i) \times (ii) \times (iii)$

③ $dh^1 = \frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial z^2} \frac{\partial z^2}{\partial h^1} = (i) \times (ii) \times (iv)$

④ $dz^1 = \frac{\partial L}{\partial z^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial z^1} = (i) \times (ii) \times (iv) \times (v)$

⑤. $dW^1 = \frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial z^1} \frac{\partial z^1}{\partial W^1} = (i) \times (ii) \times (iv) \times (v) \times (vi)$

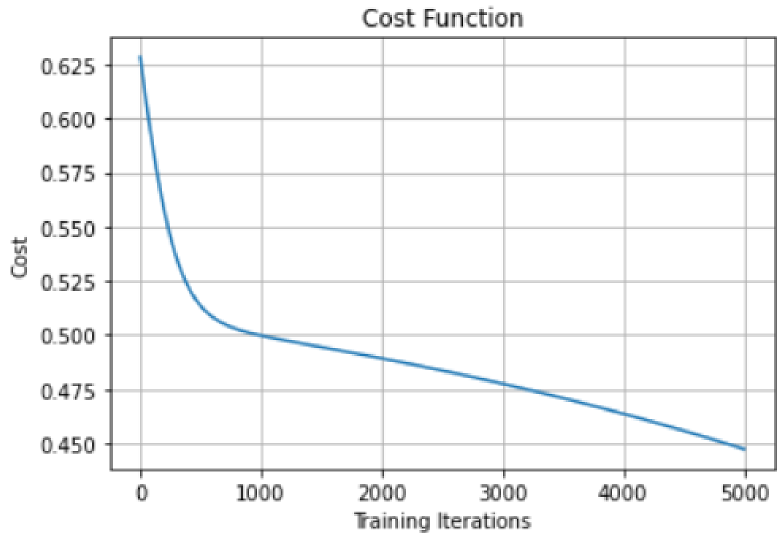


And our little NN works!

```
In [56]: plt.grid()
         plt.plot(range(num_iterations),cost)

         plt.title('Cost Function')
         plt.xlabel('Training Iterations')
         plt.ylabel('Cost')

Out[56]: Text(0, 0.5, 'Cost')
```



- Note: training in batches, batch normalizations, dropouts excluded for time constraint