Shallow Neural Networks

Learning Objectives:

- Describe hidden units and hidden layers
- Use units with a non-linear activation function
- Implement forward and backward propagation
- Apply random initialization to your neural network
- Increase fluency in Deep Learning notations and Neural Network Representations
- Compute the cross entropy loss

Shallow Neural Networks

1 Neural Networks Overview
2 Neural Network Representation
3 Computing a Neural Network's Output
4 Vectorizing across multiple examples
5 Explanation for vectorized implementation
6 Activation functions
7 Why do you need non-linear activation functions?
8 Derivatives of activation functions
9 Gradient descent for neural networks
10 Backpropagation intuition (Optional)
11 Random Initialization

FPT UNIVERSITY

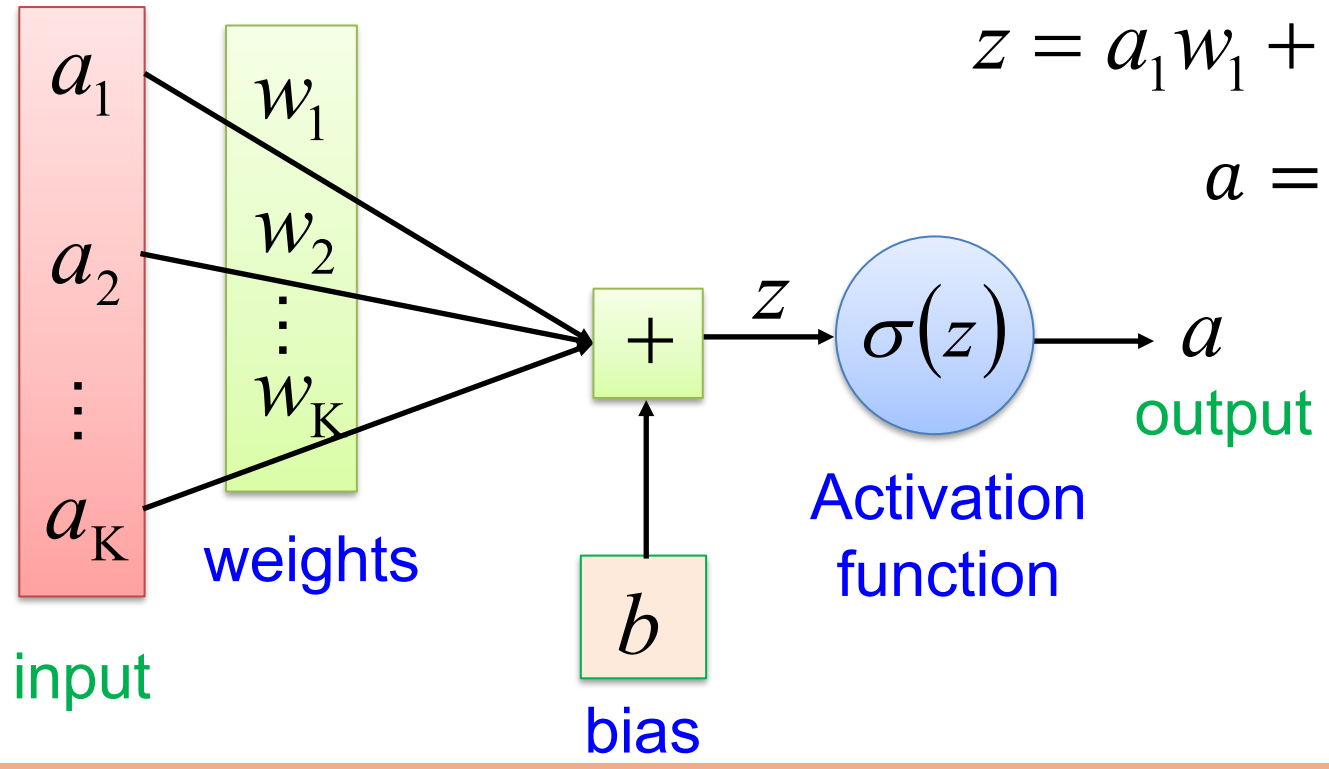
One hidden layer Neural Network

What is a Neural Networks

- A neural network, often referred to as an artificial neural network (ANN), is a computational model inspired by the way biological neural networks, like those found in the human brain, process information.
- It is a powerful machine learning technique capable of learning complex patterns and making predictions or decisions based on input data.
- A neural network is formed by stacking together multiple logistic regression-like computations (sigmoid units).

What is a Neural Networks

- Elements of Neural Networks:
  - Neural networks consist of hidden layers with neurons (i.e., computational units)
  - A single neuron maps a set of inputs into an output number, or $f: R^K \rightarrow R$



$$z = a_1 w_1 + a_2 w_2 + \cdots + a_K w_K + b$$

$$a = \sigma(z)$$

$$\mathcal{L}(a, y)$$

FPT Education

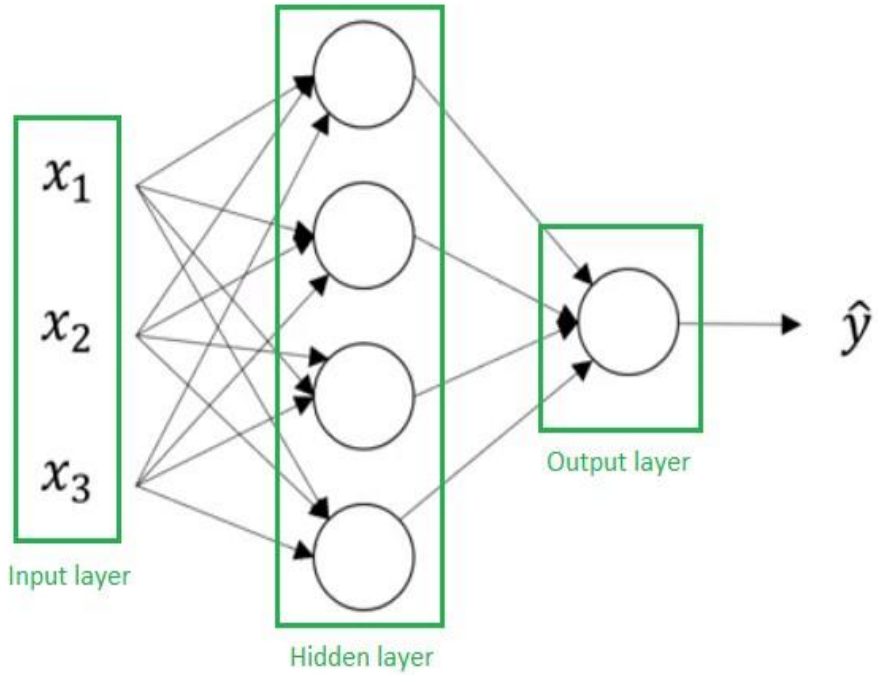
FPT UNIVERSITY

One hidden layer Neural Network

Neural Network Representation

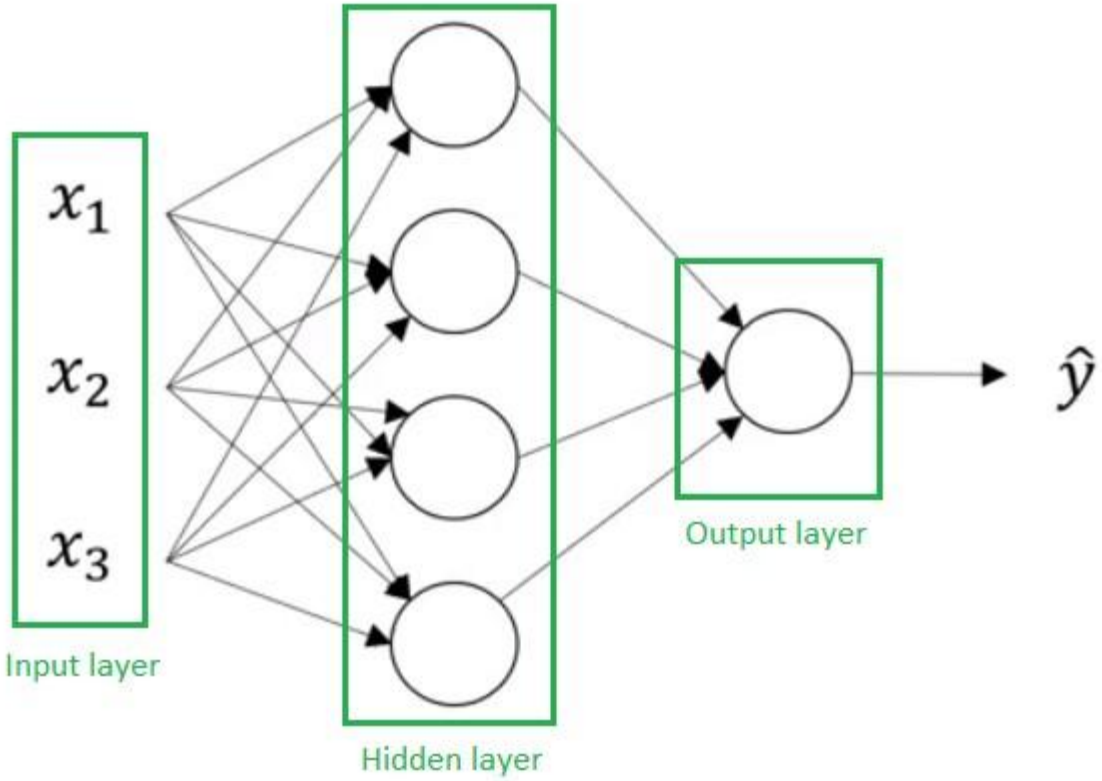
Neural Networks Representation

- Consider a neural network with one hidden layer and one output layer
- The key components of a neural network are the input layer, the hidden layer, and the output layer.
- The hidden layer is so-called because its values are not observed in the training set.



Neural Network Representation

- Elements of Neural Networks:



Weights Biases

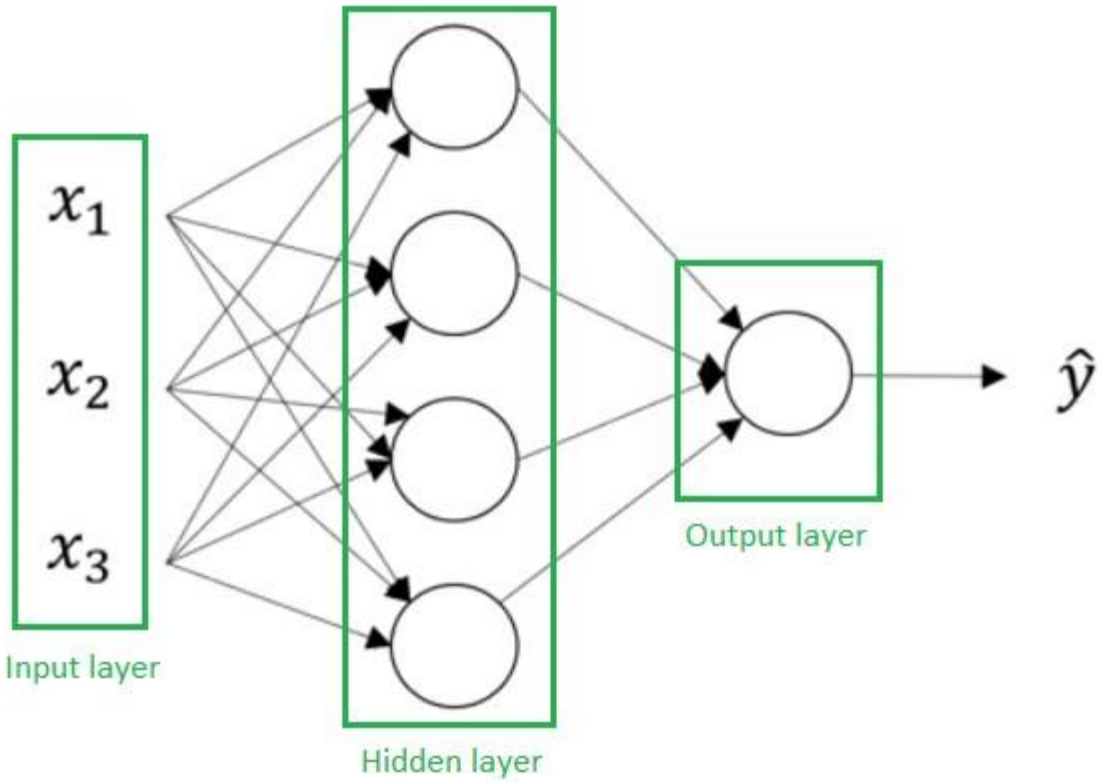
$$hidden\ layer\ h = \sigma(Wx + b)$$

$$output\ layer\ y = \sigma(Wh + b)$$

Activation functions

$a^{[0]} = x$ $a^{[1]} = h$ $a^{[2]} = \hat{y}$

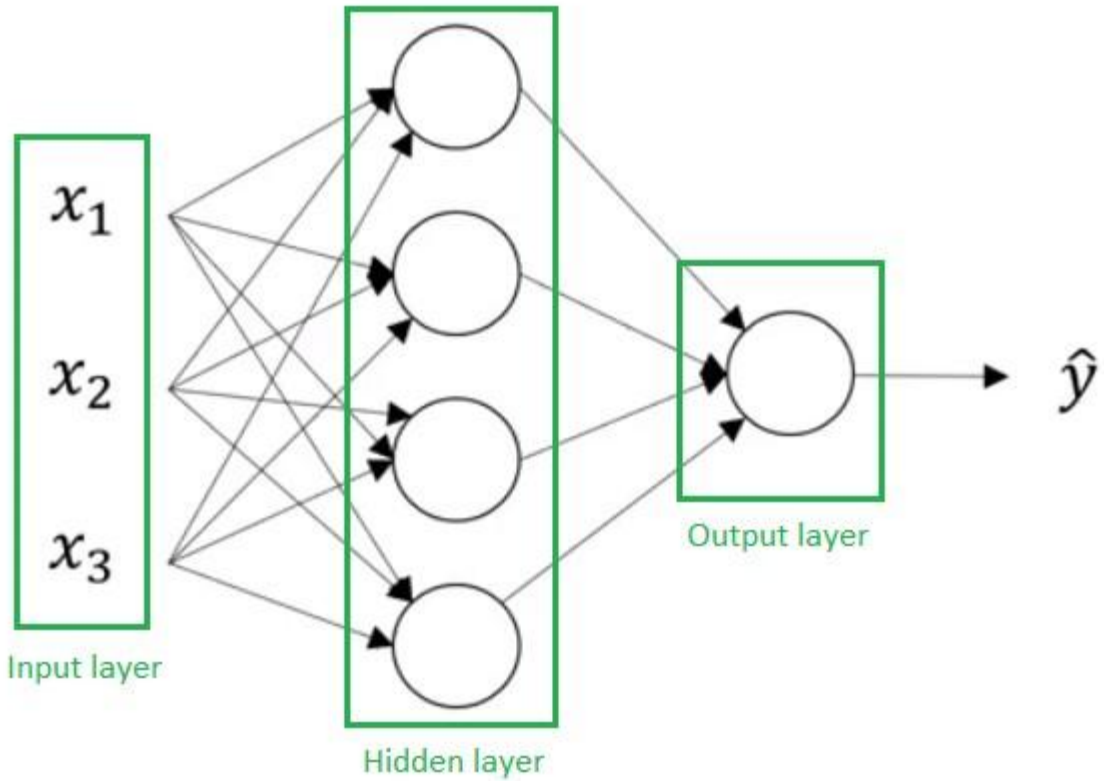
Neural Networks Representation



$\mathbf{a^{[0]}} = \boldsymbol{x}$ $\mathbf{a^{[1]}} = \boldsymbol{h}$ $\mathbf{a^{[2]}} = \hat{y}$

- Activations are introduced as the values passed between layers:
  - $a^{[0]}$ represents input features or layer 0
  - $a^{[1]}$ represents the hidden layer activations or layer 1
  - $a^{[2]}$ represents the output or layer 2.
- A two-layer neural network refers to the number of layers excluding the input layer.
- The parameters associated with each layer (W and b), with their dimensions to be explained further in later slides.

Neural Networks Representation

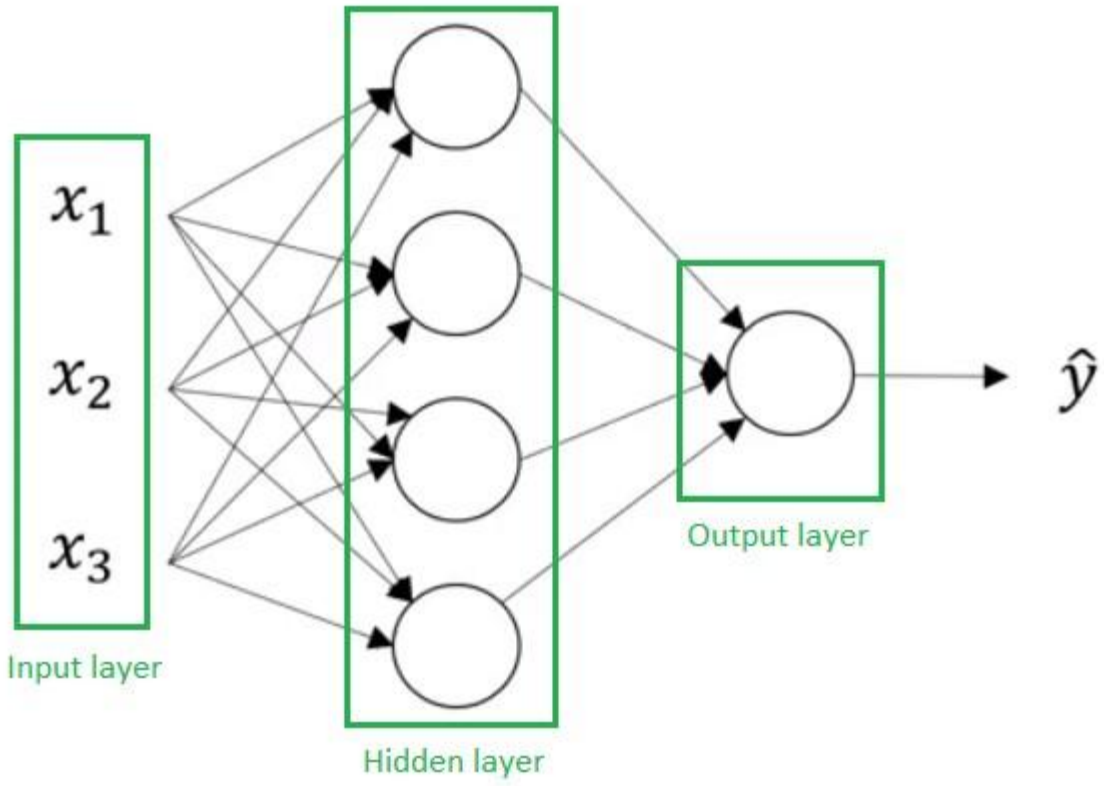


$\mathbf{a^{[0]}} = \boldsymbol{x}$ $\mathbf{a^{[1]}} = \boldsymbol{h}$ $\mathbf{a^{[2]}} = \hat{y}$

- The first hidden layer is associated with parameters w[1] and b[1]. The dimensions of these matrices are:
- - $w^{[1]}$ is (4,3) matrix
- - $b^{[1]}$ is (4,1) matrix
- Parameters w[2] and b[2] are associated with the second layer or actually with the output layer. The dimensions of parameters in the output layer are:
- - $w^{[2]}$ is (1,4) matrix
- - $b^{[2]}$ is a real number

Neural Network Representation

Elements of Neural Networks:



- 4 + 1 = 5 neurons (not counting inputs)
- [3 × 4] + [4 × 1] = 16 weights
- 4 + 1 = 5 biases
- 21 learnable parameters

$\mathbf{a}^{[0]} = \boldsymbol{x}$ $\mathbf{a}^{[1]} = \boldsymbol{h}$ $\mathbf{a}^{[2]} = \hat{y}$

FPT Education

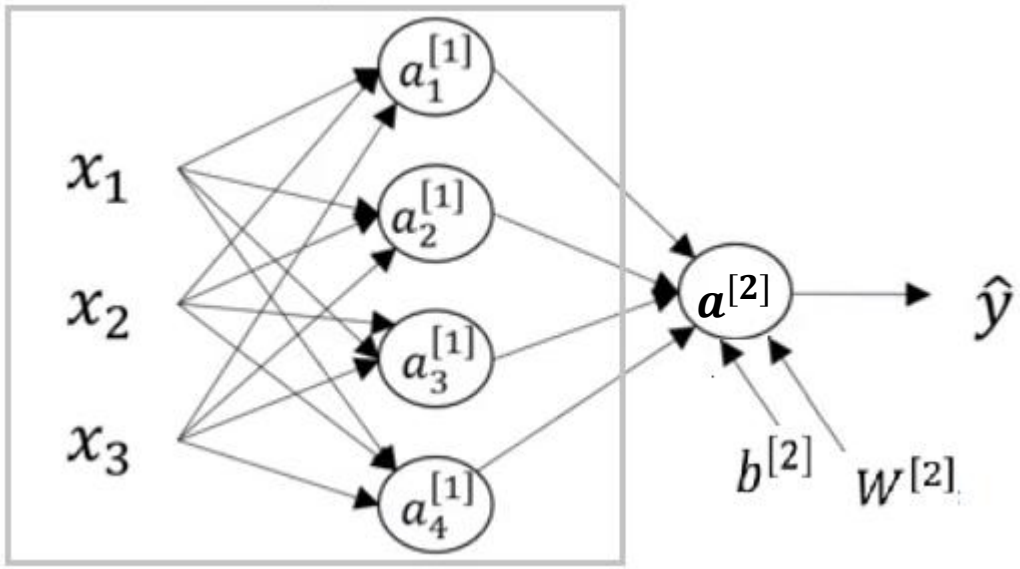
FPT UNIVERSITY

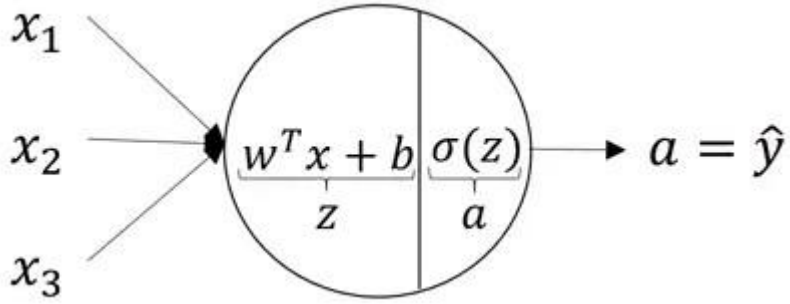
One hidden layer Neural Network

Computing a Neural Network's Output

Computing a Neural Network's Output

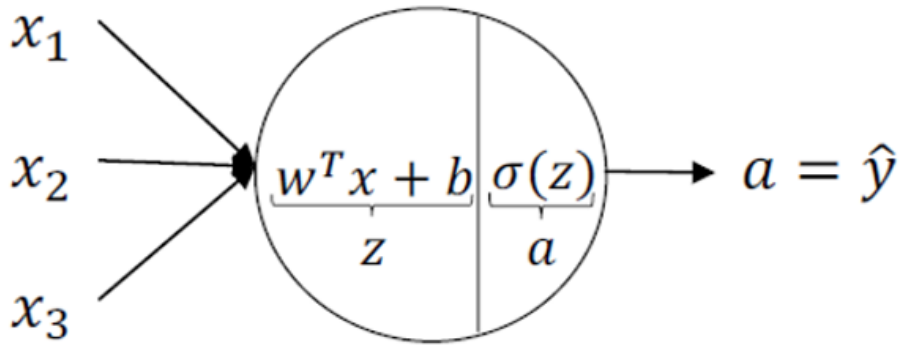
- Equations of Hidden layers:
- Each neuron will receive an input, perform some transformations on them (here, calculation $Z = W^{[T]}x + b$) then apply the sigmoid function (σ):





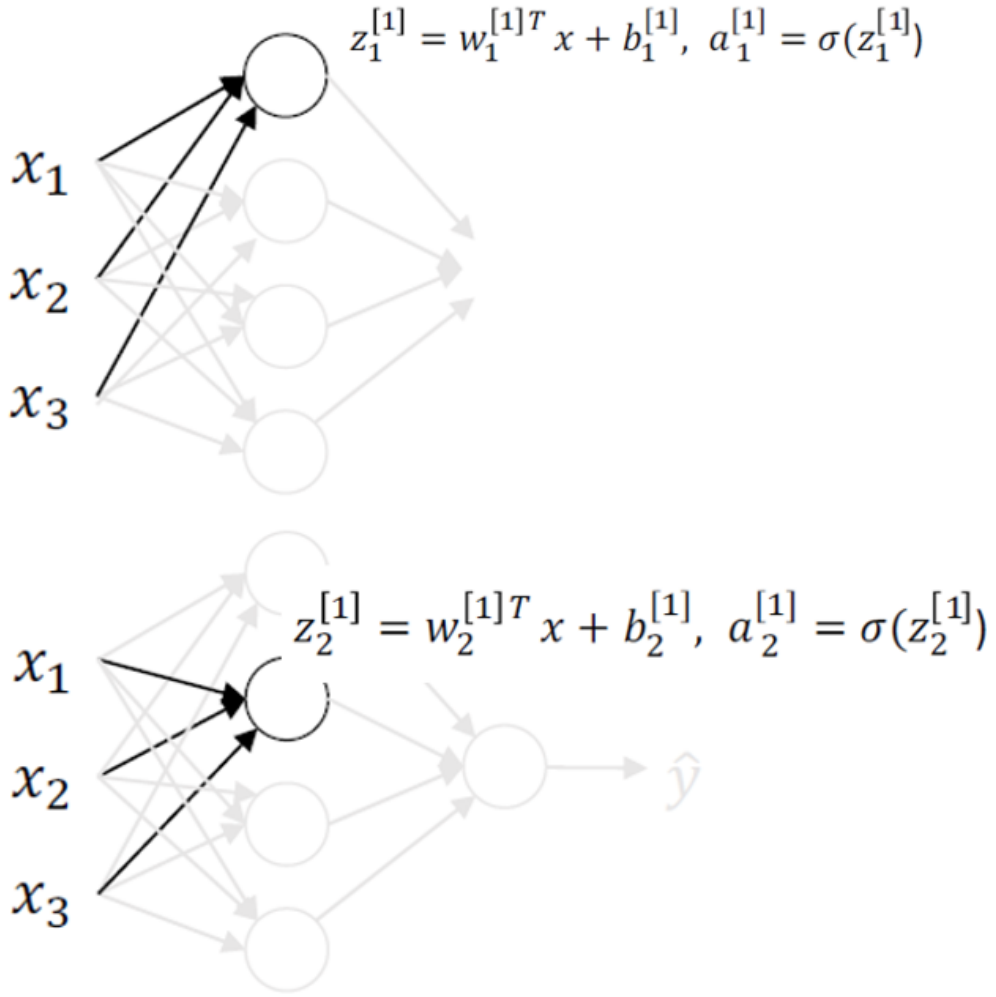
Computing a Neural Network's Output

- Equations of Hidden layers:



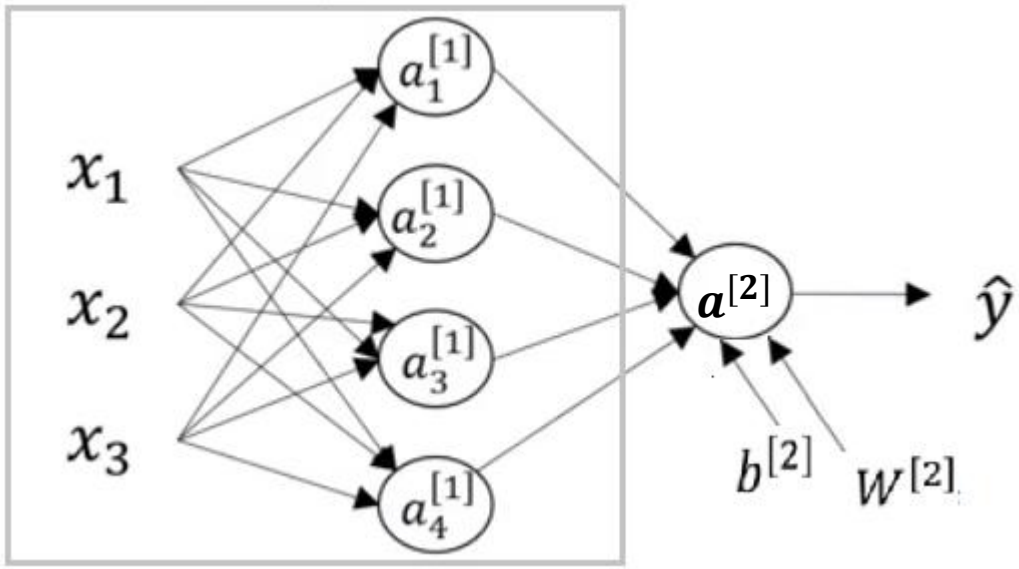
$$z = w^T x + b$$

$$a = \sigma(z)$$



Computing a Neural Network's Output

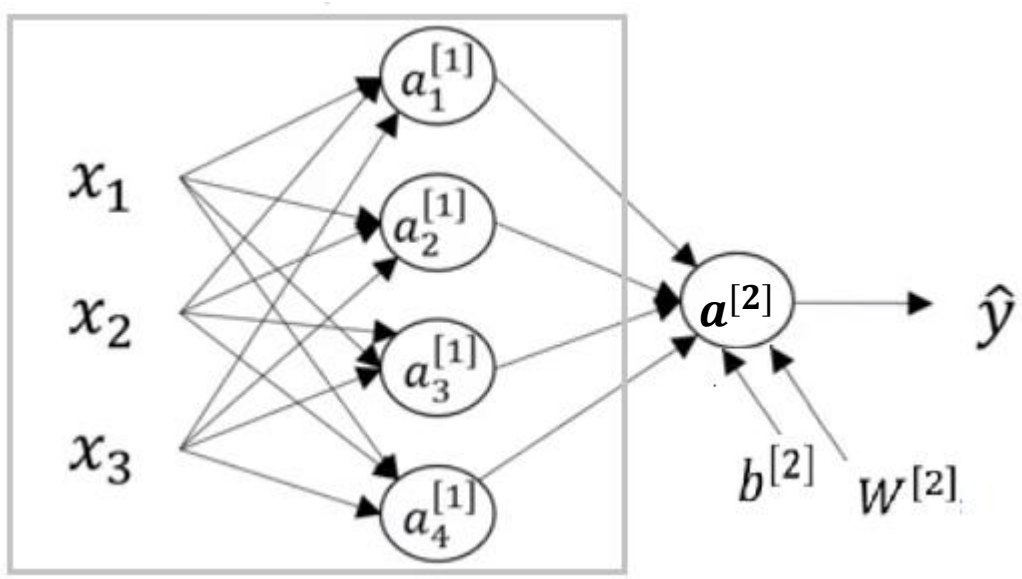
- Equations of Hidden layers:
  - These steps will be performed on each nerve cell. The equations for the first hidden layer with 4 neurons will be:



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

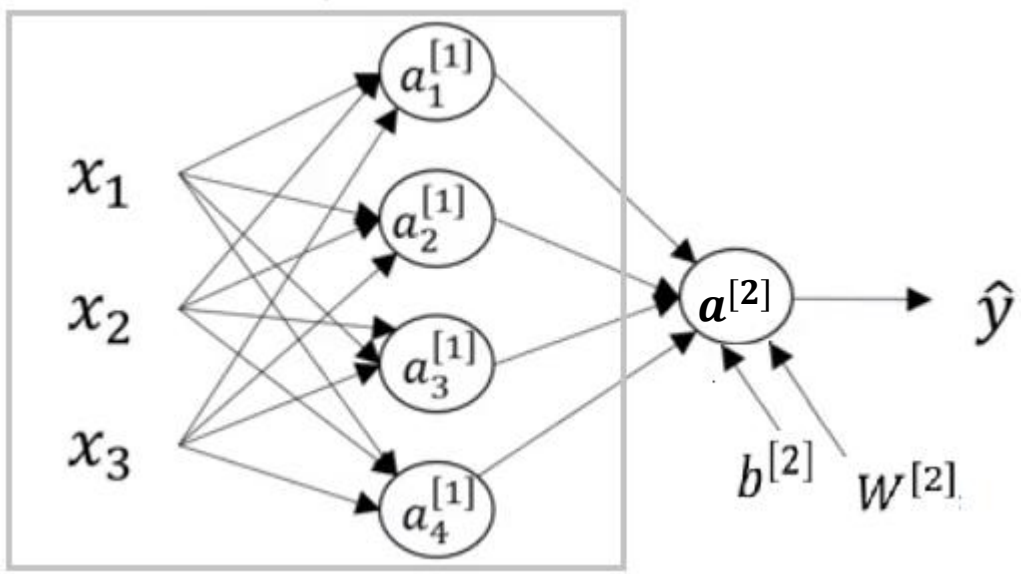
Computing a Neural Network's Output

- Equations of Hidden layers:
  - So, for given input X, the outputs of layer 1 and 2 will be:



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

Computing a Neural Network's Output



Equations of Hidden layers:

- No of Hidden Neurons = 4
- Nx = 3
- Shapes of the variables:

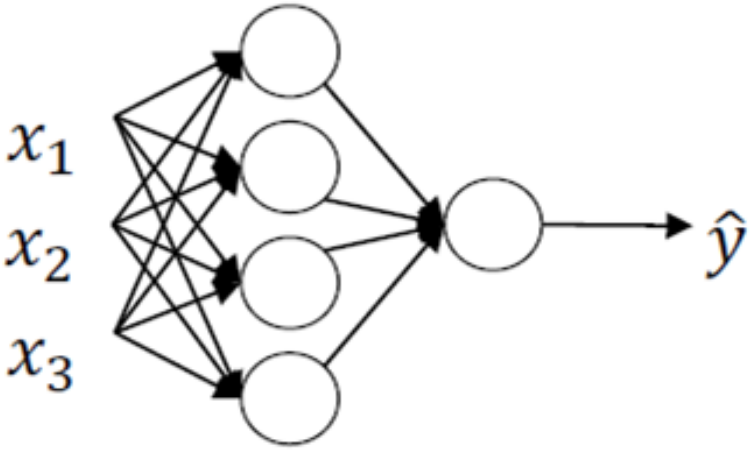
- $W^{[1]}$ is the matrix of the first hidden layer, it has a shape of (noOfHiddenNeurons, Nx)
- $b^{[1]}$ is the matrix of the first hidden layer, it has a shape of (noOfHiddenNeurons,1)
- $z^{[1]}$ is the result of the equation $z^{[1]} = W^{[1]} * X + b$, it has a shape of (noOfHiddenNeurons,1)
- $a^{[1]}$ is the result of the equation $a^{[1]} = \sigma(z^{[1]})$, it has a shape of (noOfHiddenNeurons,1)
- $W^{[2]}$ is the matrix of the second hidden layer, it has a shape of (1,noOfHiddenNeurons)
- $b^{[2]}$ is the matrix of the second hidden layer, it has a shape of (1,1)
- $z^{[2]}$ is the result of the equation $z^{[2]} = W^{[2]} * a^{[1]} + b$, it has a shape of (1,1)
- $a^{[2]}$ is the result of the equation $a^{[2]} = \sigma(z^{[2]})$, it has a shape of (1,1)

FPT Education

FPT UNIVERSITY

Vectorizing across multiple examples

Vectorizing across multiple examples



$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$

Vectorizing across multiple examples

- Four equations from the previous slide are used to compute z[1] , a[1], z[2] , and a[2] .
- By adding a superscript round bracket i, ie (i), to all the variables that depend on the training example, outputs can be computed for all m training examples:

```
for i = 1 to m:
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

Vectorizing across multiple examples

FPT Education
FPT UNIVERSITY

- The goal is to vectorize the computation and get rid of the for loop. To achieve this, Z[1] , A[1] , Z[2] , and A[2] can be computed using matrixes:
  - Matrix Z[1] represents the output of the first layer for all examples
  - Matrix A[1] represents the activations of the first layer for all examples.
  - Matrix Z[2] represents the output of the second layer for all examples
  - Matrix A[2] represents the activations of the second layer for all examples.
- By stacking up the lowercase vectors in different columns, you can obtain the matrices Z and A. The horizontal index in these matrices corresponds to different training examples, while the vertical index corresponds to different nodes in the neural network.

FPT Education

FPT UNIVERSITY

One hidden layer Neural Network

Explanation for vectorized implementation

Explanation for vectorized implementation

FPT Education
FPT UNIVERSITY

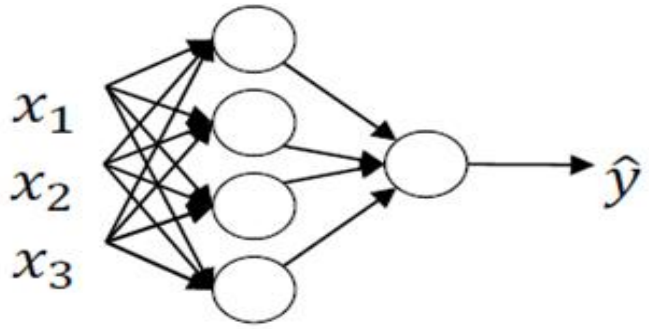
1. Stacking training examples horizontally in the matrix X.
2. The matrix capital Z[1] is formed by taking the vector x(1), x(2) , and x(3) (or more examples) and stacking them vertically.
3. The line z[1] = w[1] x + b[1] is a correct vectorization of the first step of forward propagation.
4. Similar reasoning can be applied to the other lines of code, showing that they are also correct vectorizations.
5. There's a certain symmetry in the equations, which demonstrates that the different layers of a neural network are performing the same computation repeatedly.

Explanation for vectorized implementation

$$W^{[1]}x = \begin{bmatrix} \cdots \\ \cdots \\ \cdots \end{bmatrix} \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ x^{(1)} & x^{(2)} & x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ w^{(1)}x^{(1)} & w^{(1)}x^{(2)} & w^{(1)}x^{(3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} =$$

$$\begin{bmatrix} \vdots & \vdots & \vdots & \vdots \\ z^{[1](1)} & z^{[1](2)} & z^{[1](3)} & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} = Z^{[1]}$$

Recap of vectorizing acrros multiple examples



$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

```
for i = 1 to m
```

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$
$$a^{[1](i)} = \sigma(z^{[1](i)})$$
$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$
$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = \sigma(Z^{[2]})$$

FPT Education

FPT UNIVERSITY

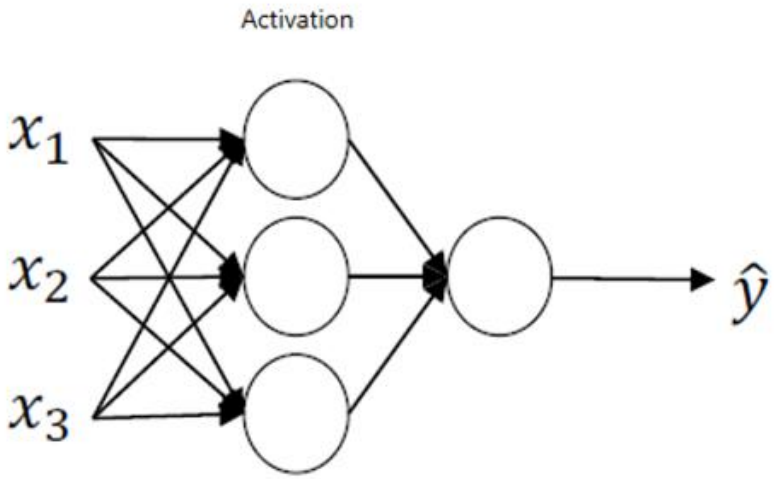
One hidden layer Neural Network

Activation functions

Activation functions

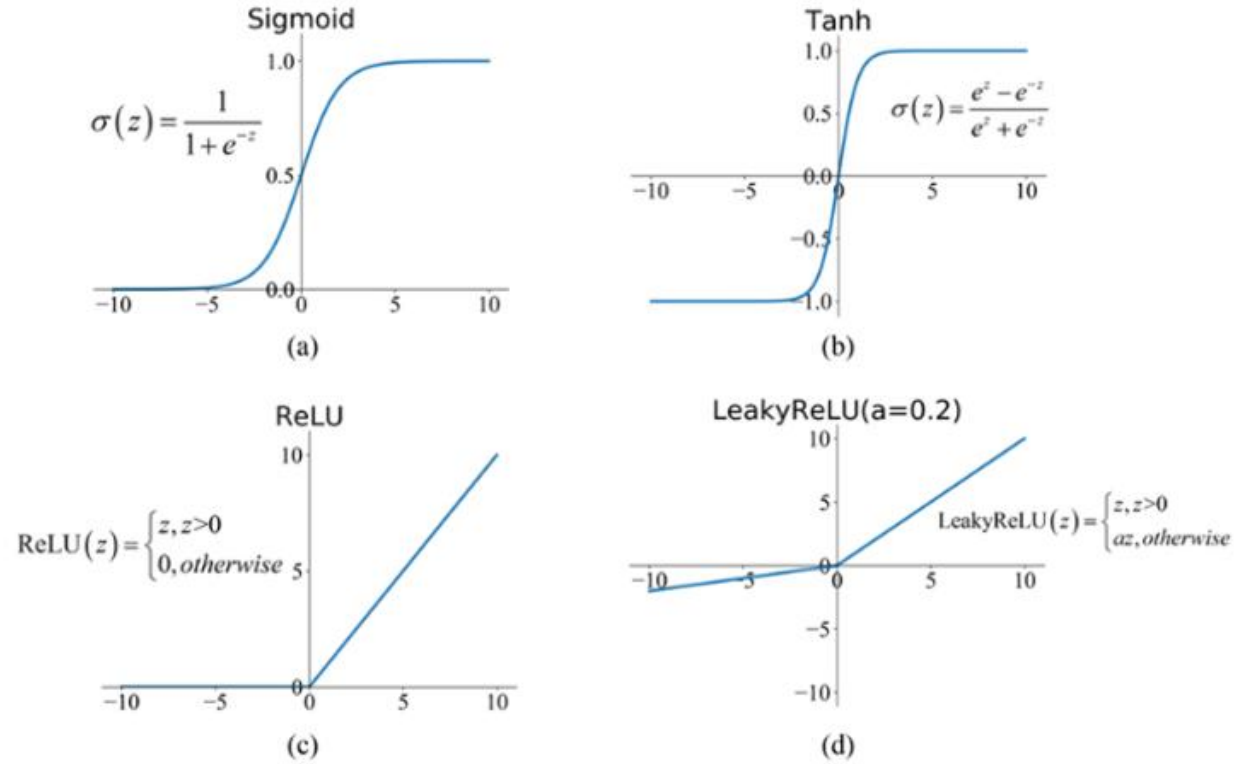
- The sigmoid function was popular in the past, but it has been mostly replaced by other activation functions.
- The tanh function, which is a shifted version of the sigmoid, works better than the sigmoid for hidden layers as it centers the data with a mean close to zero.
- However, for the output layer in binary classification, the sigmoid function is still preferred.
- The rectified linear unit (ReLU) is currently the most popular activation function for hidden layers due to its simplicity and ability to reduce the vanishing gradient problem.
- It is defined as a=max(0,z). The Leaky ReLU, a variant of ReLU, is also used in some cases, with a=max(0.01*z,z) instead.
- It offers a small slope for negative z values, which can help avoid dead neurons.

Activation functions



Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = \sigma(z^{[2]})$$



Commonly used activation functions: (a) Sigmoid, (b) Tanh, (c) ReLU, and (d) LReLU.

Activation functions

- Some rules of thumb for choosing activation functions are:
  - For binary classification output layers, use the sigmoid function.
  - For hidden layers, the default choice is the ReLU function. The tanh function can also be used.
- It is essential to experiment with different activation functions in your specific application to determine the best choice. This helps future-proof your neural network architecture against the idiosyncrasies of different problems and the evolution of algorithms.

FPT Education

FPT UNIVERSITY

One hidden layer Neural Network

Why do you need non linear activation functions?

Why do you need non-linear activation functions?

FPT Education
FPT UNIVERSITY

- If a linear or identity activation function is used, the neural network would only compute a linear function of the input, making hidden layers more or less useless. The composition of two linear functions is itself a linear function, so without a non-linear activation function, the network won't compute more interesting functions even with more layers.
- A linear activation function might be used in the output layer if the problem is a regression problem where y is a real number, but hidden layers should still use non-linear activation functions such as ReLU, tanh, or Leaky ReLU.

FPT UNIVERSITY

Derivatives of activation functions

Derivatives of activation functions

- The computation of slopes or derivatives of activation functions used in backpropagation for neural networks.
- The **Sigmoid** activation function's slope is shown to be equal to $\sigma(x)(1-\sigma(x))$
- The **Tanh** activation function's slope is equal to $(1+tanh(x))(1-tanh(x))$
- The **ReLU** activation function's derivative is 0 if x is less than 0 and 1 if x is greater than 0, and the **Leaky ReLU** activation function's derivative is 0.01 if x is less than 0 and 1 if x is greater than 0. It is noted that the derivative is technically undefined when x is exactly equal to 0, but in practice, it does not matter where the derivative is set.
- **Note**: The shorthand notation f'(x) for the derivative of the function f(x) with respect to the input variable x.

Sigmoid function

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

$$\frac{\partial \sigma(x)}{\partial x} = \frac{\partial}{\partial x}\frac{1}{1+e^{-x}} = \frac{\partial}{\partial x}(1+e^{-x})^{-1} = (1+e^{-x})^{-2}\frac{\partial}{\partial x}(1+e^{-x})$$

$$= \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}} * \frac{e^{-x}}{1+e^{-x}} = \frac{1}{1+e^{-x}} * \frac{1+e^{-x}-1}{1+e^{-x}}$$

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

Tanh function

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

$$\frac{\partial tanh(x)}{\partial x} = \frac{\partial}{\partial x}\frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$= \frac{[\frac{\partial}{\partial x}(e^x - e^{-x})](e^x + e^{-x}) - (e^x - e^{-x})[\frac{\partial}{\partial x}(e^x + e^{-x})]}{(e^x + e^{-x})^2}$$

$$= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} = \frac{4}{(e^x + e^{-x})^2} = \frac{2e^x}{e^x + e^{-x}} * \frac{2e^{-x}}{e^x + e^{-x}}$$

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

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

ReLU and Leaky ReLU function

FPT Education
FPT UNIVERSITY

ReLU $$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases} \qquad f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$$

ReLU $$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \qquad f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

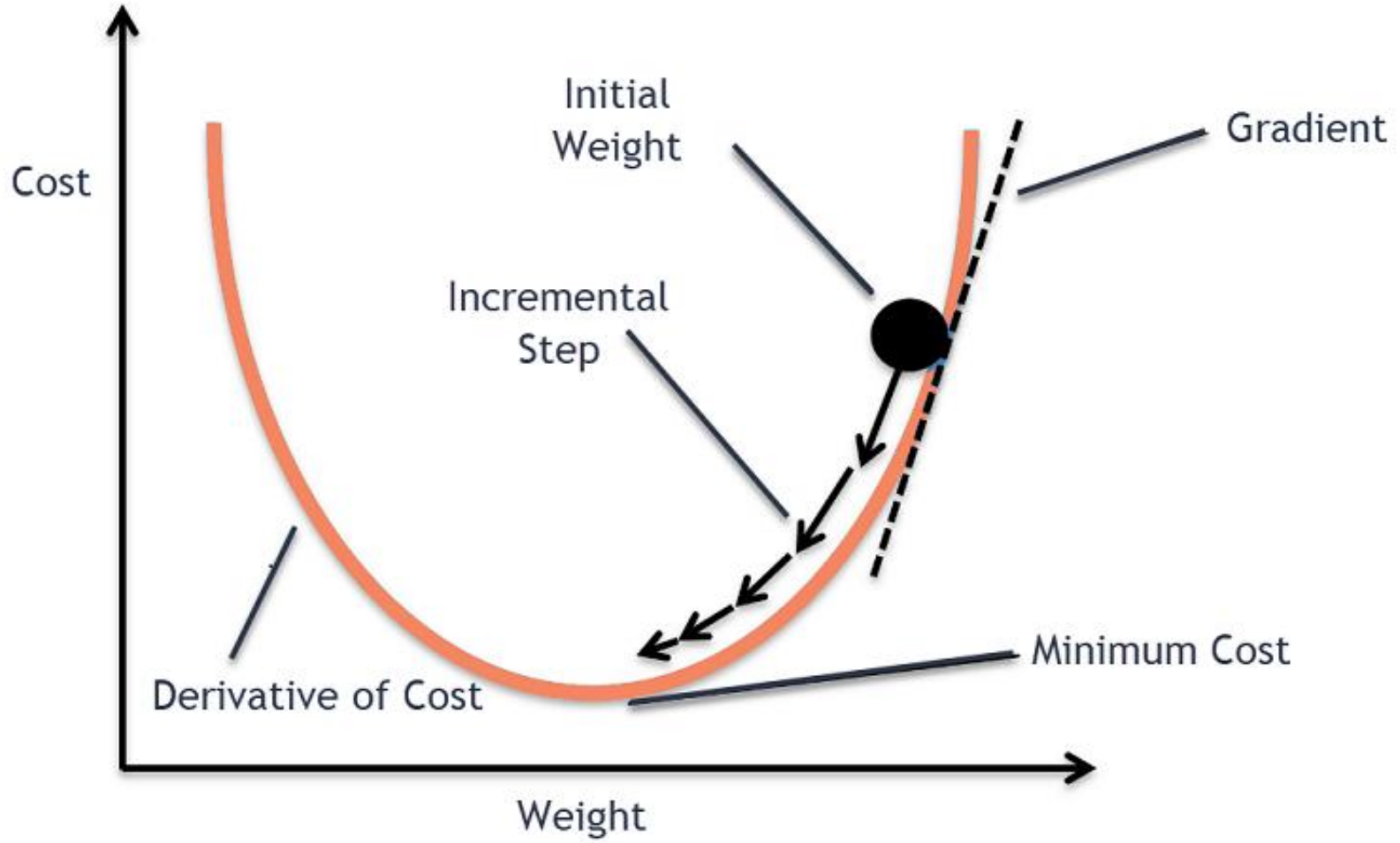
Leaky ReLU $$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases} \qquad f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

FPT UNIVERSITY

One hidden layer Neural Network

Gradient descent for neural networks

Gradient descent for neural networks



Gradient descent for neural networks

Parameter: $w^{[1]}$, $b^{[1]}$, $w^{[2]}$, $b^{[2]}$

$(n^{[1]}, n^{[0]})$ $(n^{[1]}, 1)$ $(n^{[2]}, n^{[1]})$ $(n^{[2]}, 1)$ $n_x = n^{[0]}, n^{[1]}, n^{[2]} = 1$

Cost function: $J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}, y)$

Gradient Descent:

➔ Repeat {

Compute Predicts ($\hat{y}^{(i)}$, i = 1…m)

$dw^{[1]} = \frac{dJ}{dw^{[1]}}$, $db^{[1]} = \frac{\partial J}{\partial\, b^{[1]}}$,…

$w^{[1]} = w^{[1]} - \alpha\, dw^{[1]}$

$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$

$w^{[2]} = \ldots$

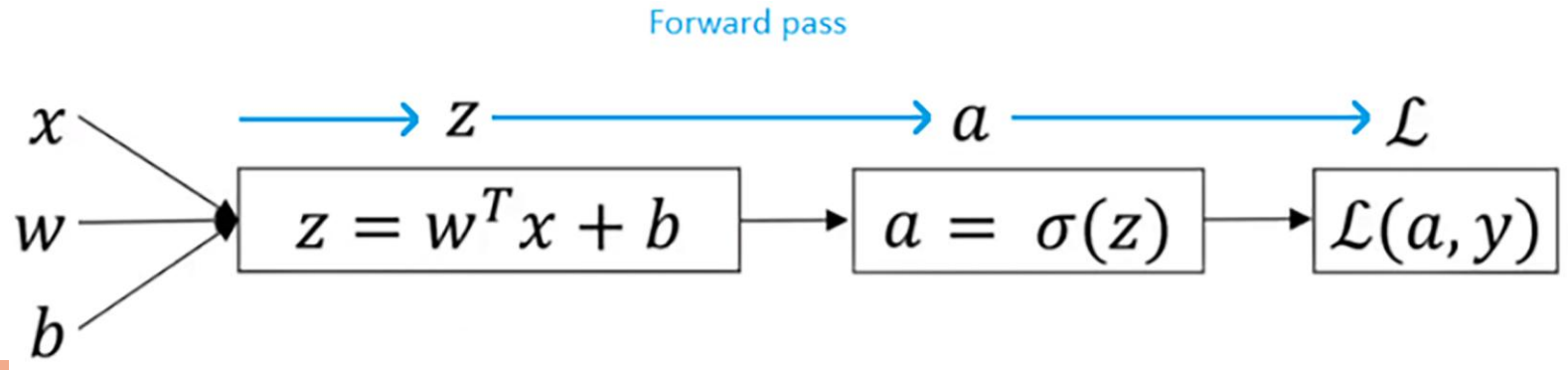
$b^{[2]} = \ldots$

}

Formulas for computing derivatives

- The forward and backpropagation of a 2-layer neural network:
- Forward propagation:

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]}) = \sigma(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1[} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$$



Formulas for computing derivatives

FPT Education
FPT UNIVERSITY

- **Backward propagation**:
- Also known as backpropagation or backprop, **is the process of computing the gradient of the loss function with respect to the weights and biases of a neural network**, so that the weights and biases can be updated during the training process.
- During forward propagation, the input data is passed through the neural network, layer by layer, to produce an output. The output is compared to the expected output, and the difference between them is used to compute the loss function.
- The goal of backpropagation is to compute the gradient of the loss function with respect to each weight and bias in the network.

Formulas for computing derivatives

- Backward propagation:
- To compute the gradients, backpropagation uses the chain rule of calculus to propagate the error back through the layers of the neural network.
  - Starting at the output layer, the gradient of the loss function with respect to the output of each neuron is computed.
  - These gradients are then used to compute the gradients with respect to the input to each neuron in the previous layer.
  - This process is repeated until the gradient with respect to each weight and bias in the network is computed.

Formulas for computing derivatives

- Backward propagation:
- Once the gradients have been computed, they can be used to update the weights and biases of the network using an optimization algorithm, such as stochastic gradient descent, to minimize the loss function.
- This process of forward propagation followed by backpropagation and weight updates is repeated many times during training until the network is able to make accurate predictions on new data.

Formulas for computing derivatives

FPT Education
FPT UNIVERSITY

Forward propagation

$Z^{[1]} = W^{[1]}X + b^{[1]}$
$A^{[1]} = g^{[1]}(Z^{[1]})$
$Z^{[2]} = W^{[2]}A^{[1[} + b^{[2]}$
$A^{[2]} = g^{[2]}(Z^{[2]}) = \sigma(Z^{[2]})$

Backward propagation

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$$

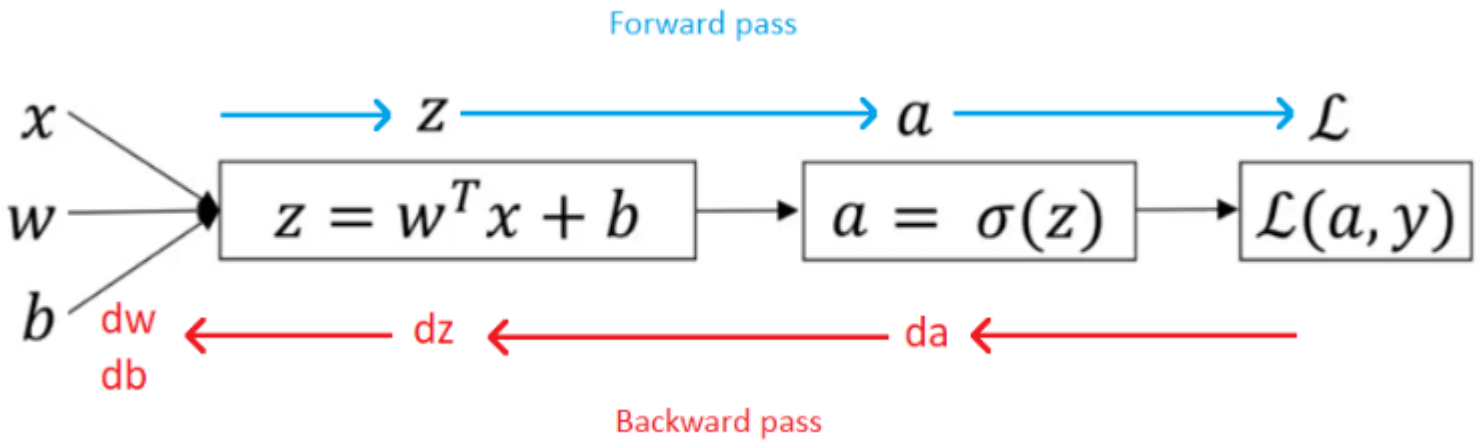
$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]^T} dZ^{[2]} * g^{[1]'}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

Formulas for computing derivatives



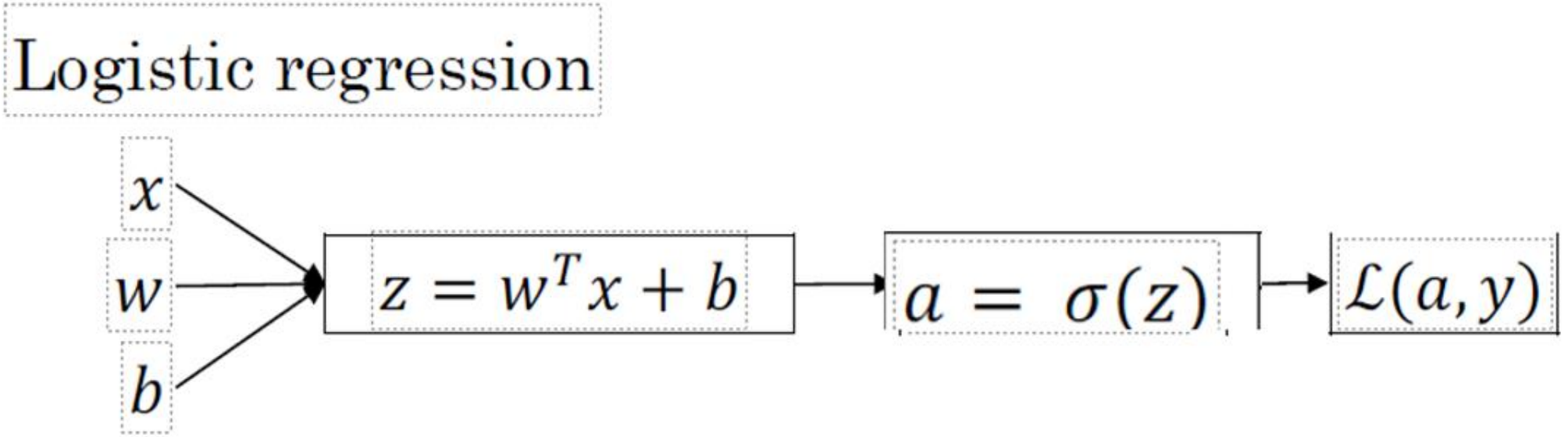
FPT Education

FPT UNIVERSITY

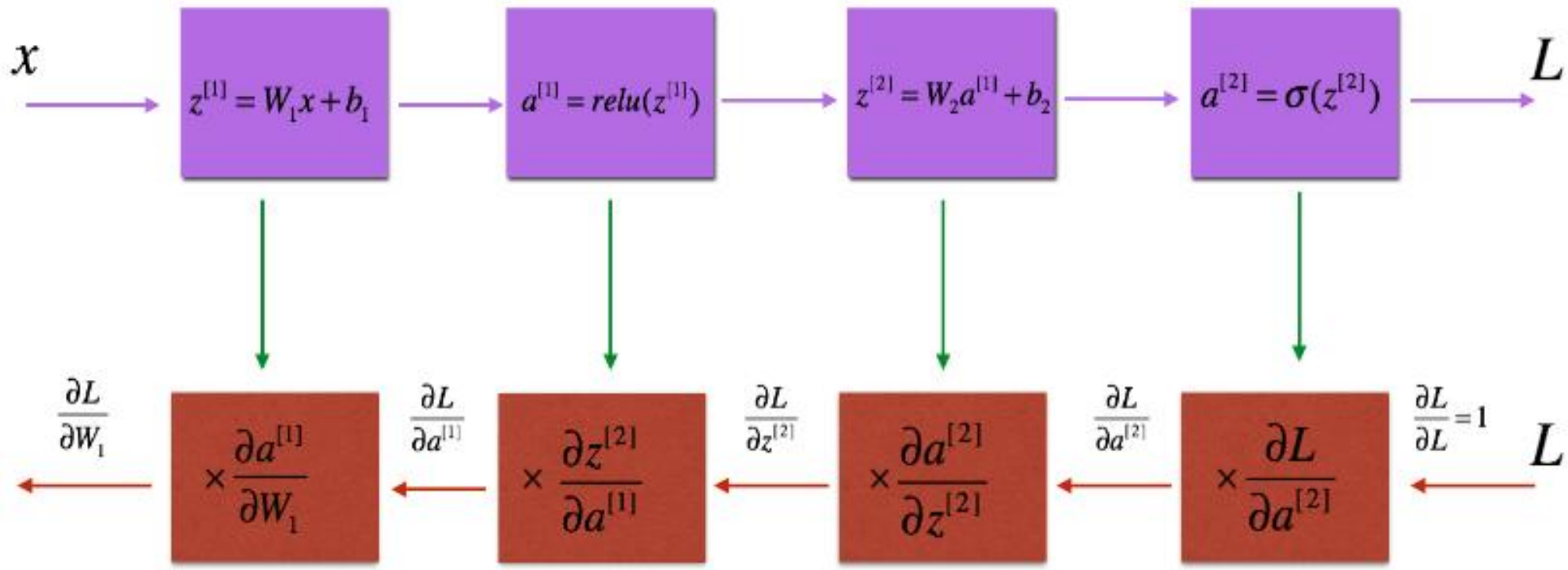
One hidden layer Neural Network

Backpropagation intuition (Optional)

Computing gradients

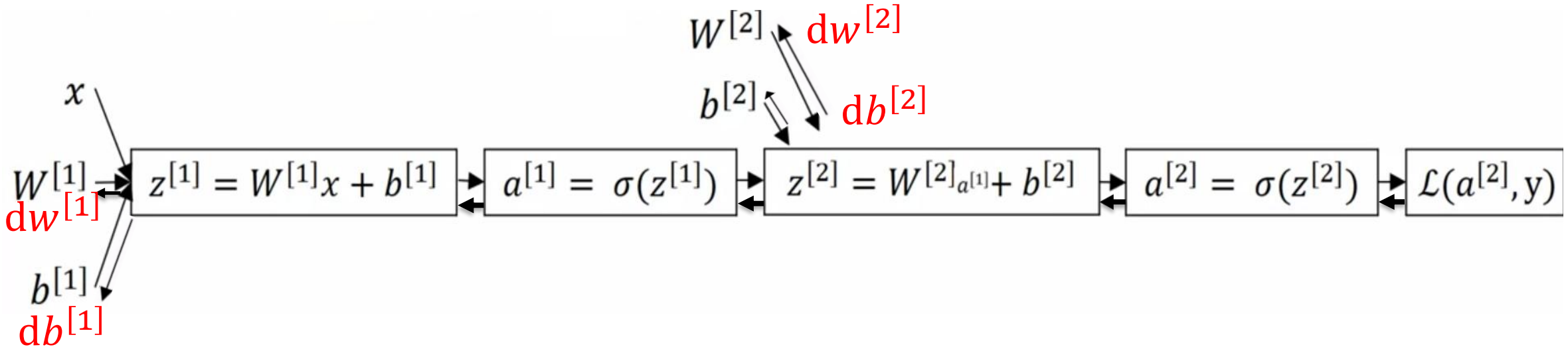


Neural network gradients



$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial W_1}$$

Neural network gradients



Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} x^{T}$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^{T}$$

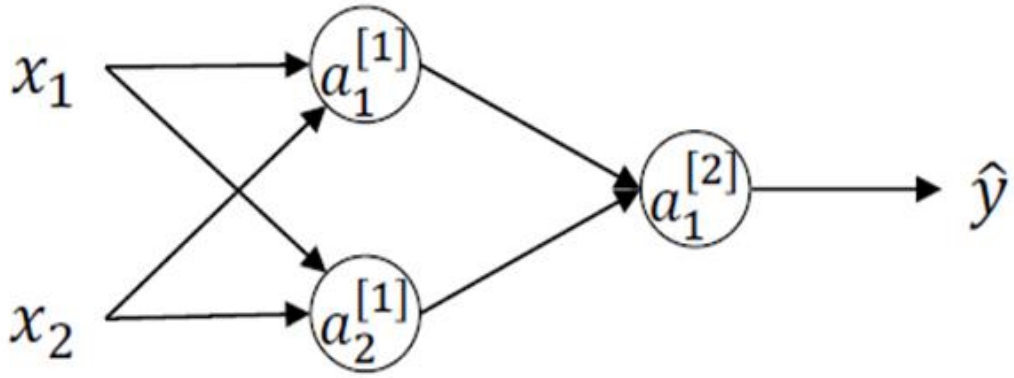
$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

FPT UNIVERSITY

One hidden layer Neural Network

Random Initialization

What happens if you initialize weights to zero?



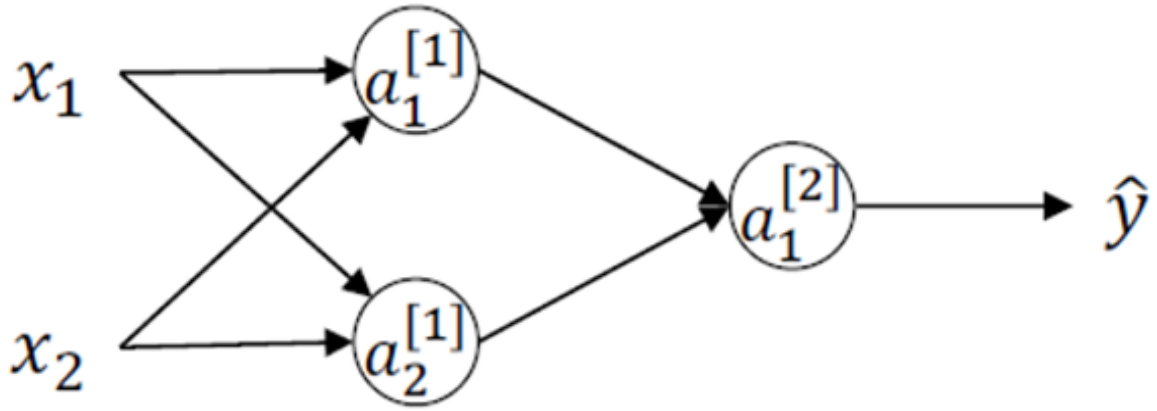
Zero initialization:

If all the weights are initialized to zeros, **the derivatives will remain same for every w in W[l]**. As a result, neurons will learn same features in each iterations. This problem is known as network failing to break symmetry.

Random Initialization

- Initializing neural network weights to all zeros results in symmetry and the hidden units computing the same function, leading to slow learning.
- To solve this, the weights are initialized randomly with small values, such as 0.01, while biases can be initialized to zeros.
- Larger constants than 0.01 may be used for very deep neural networks.

Random Initialization




```
In [15]: w1=torch.randn(m,nh)
         w1.mean(),w1.std()

Out[15]: (tensor(0.0032), tensor(1.0010))
```

Summarization

FPT Education
FPT UNIVERSITY

- Neural Networks are interconnected layers of neurons learning from data for predictions. They organize into input, hidden, and output layers, with neurons computing weighted sums and applying activation functions.
- Computing the network's output involves propagating input through layers using weights, biases, and activations.
- Vectorizing across examples enhances efficiency. Vectorized implementation speeds up training by processing multiple examples simultaneously.
- Activation functions like sigmoid, tanh, and ReLU introduce non-linearity for learning complex patterns. Non-linear activation allows networks to approximate arbitrary functions.
- Derivatives of activation functions are crucial for gradient calculations in backpropagation.
- Gradient descent optimizes by adjusting weights and biases based on gradients to minimize the cost function.
- Backpropagation propagates gradients backward through the network for parameter updates.
- Proper weight initialization is vital for preventing symmetry and ensuring efficient learning.

Questions

FPT Education
FPT UNIVERSITY

1. Does tanh usually outperform sigmoid for hidden units by better centering data (True/False, considering notes)?
2. What is the correct vectorized implementation of forward propagation for layer l (equations for Z[l] and A[l])?
3. What is the correct vectorized implementation of backward propagation for layer ll (equations for Z[l] and A[l])?
4. For a binary, ternary classifier which activations are best for the output layer and why?
5. If A.shape=(4,3), what is B.shape for B=np.sum(A,axis=1,keepdims=True), and why use keepdims?
6. If a neural network's weights and biases are initialized to zero, what happens to neuron computations?
7. Should logistic regression weights be initialized randomly (not zero) to "break symmetry”?
8. What happens if a tanh-activated network's weights are initialized very large (e.g., *1000), and why?
9. For a 1-hidden layer NN 3-2-1, what are the shapes of b[1], W[1], W[2], and b[2]?
10. For the same 1-hidden layer NN, what are the dimensions of Z[1] and A[1] (assuming m examples)?