CIS 678 - Machine Learning

Introduction to Neural Networks

Supervised Models

- kNN
- Linear Regression
- Decision Tree
- Random Forest Regressor
- Boosting Regressor
- Support Vector Regressor (SVRs)

Regression

- kNN
- Logistic Regression
- Decision Tree
- Random Forest Classifier
- Boosting Classifiers
- Support Vector Classifiers (SVCs)
- Naive Bayes

Classification

Supervised Models

- kNN
- Linear Regression
- Decision Tree
- Random Forest Regressor
- Boosting Regressor
- Support Vector Regressor (SVRs)
- Neural Networks (NNs)

Regression

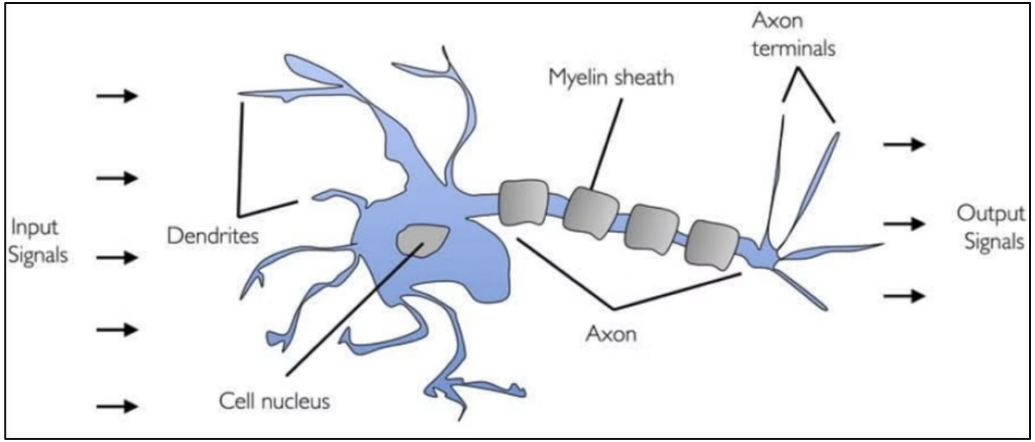
- kNN
- Logistic Regression
- Decision Tree
- Random Forest Classifier
- Boosting Classifiers
- Support Vector Classifiers (SVCs)
- Naive Bayes
- Neural Networks (NNs)

Classification

Neural Networks

Motivation src: Biological neuron

Perceptron was introduced by **Frank Rosenblatt** in 1957.



perceptron

Neural Networks

From Biological Neuron to Artificial NN

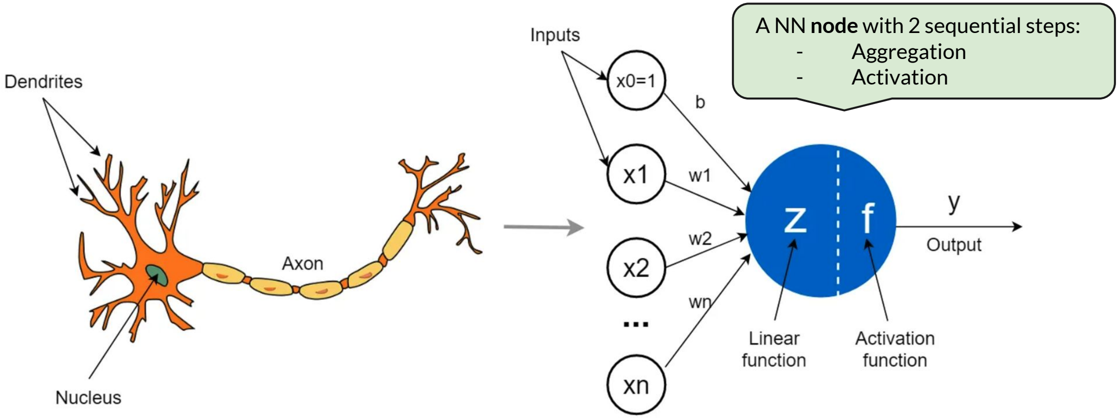


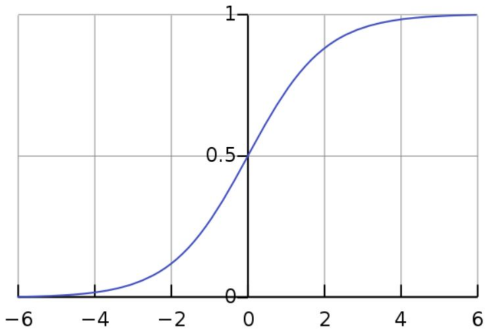
fig-ref

Logistic Regression

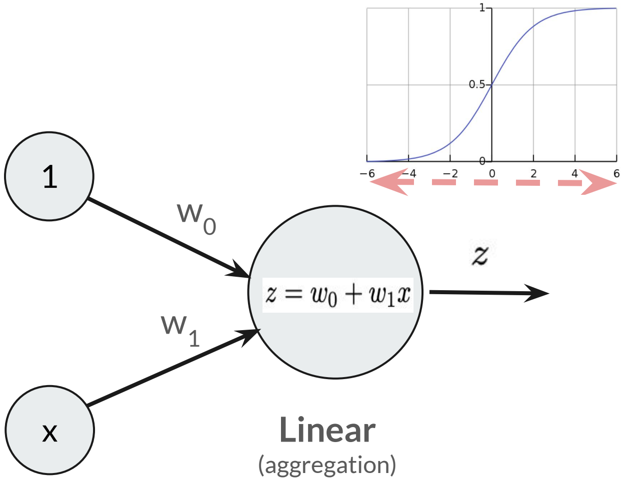
- Probabilistic classifier

$$p(x) = \frac{1}{1+e^{-(w_0+w_1 x)}}$$

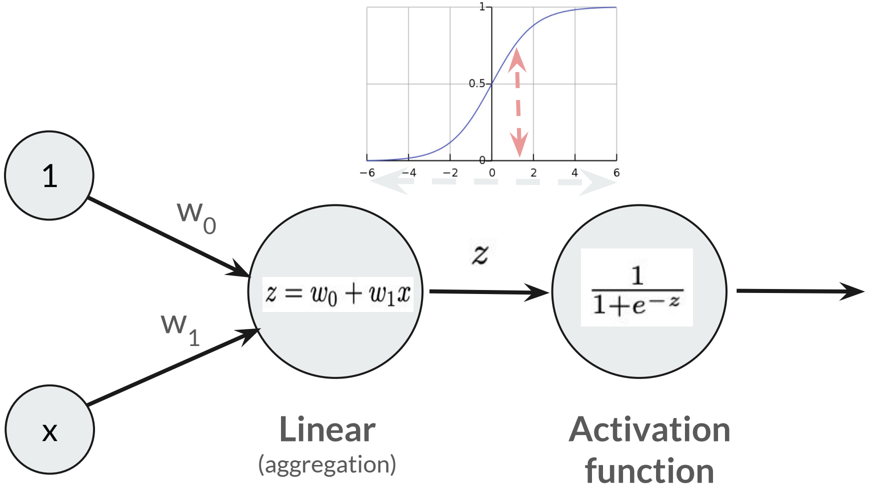
- Sigmoid function



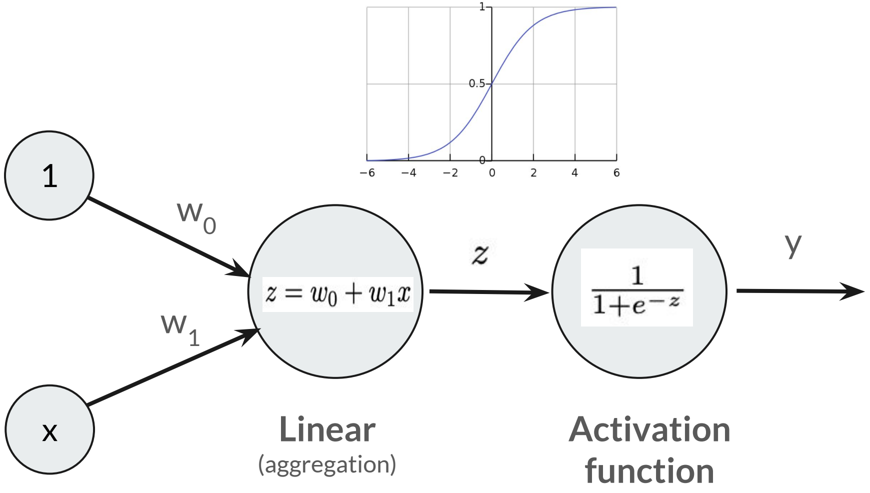
Neural Networks (Node)



Neural Networks (Node)



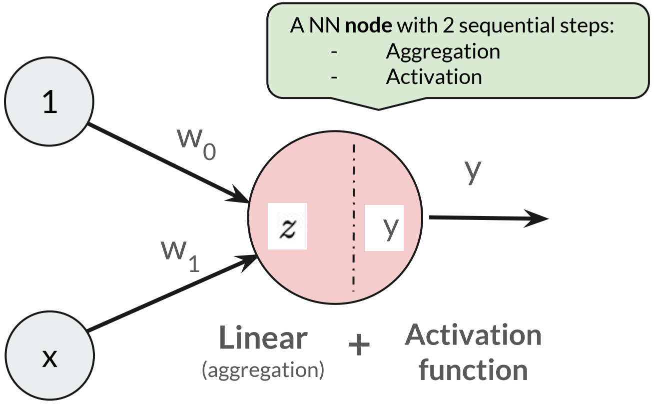
Neural Networks (Node)



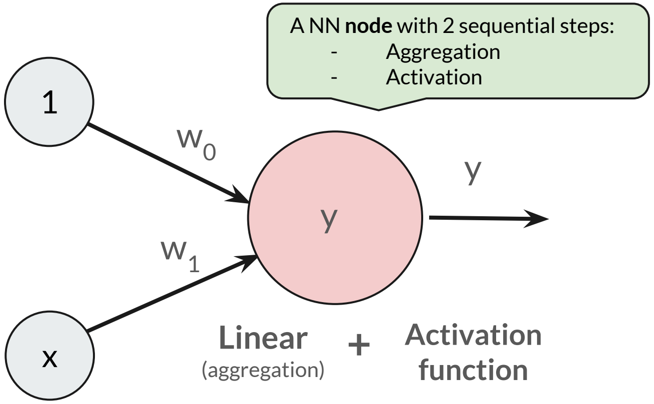
$y \in \{0, 1\}$

Logistic Regression

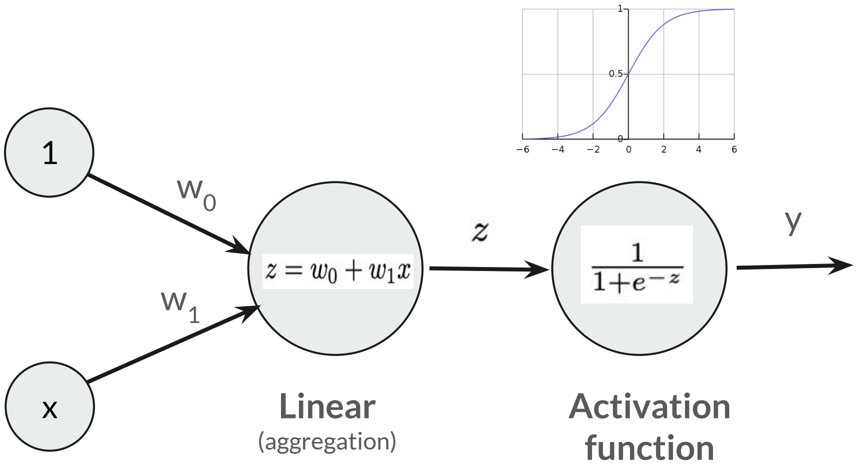
Neural Networks (Node)



Neural Networks (Node)



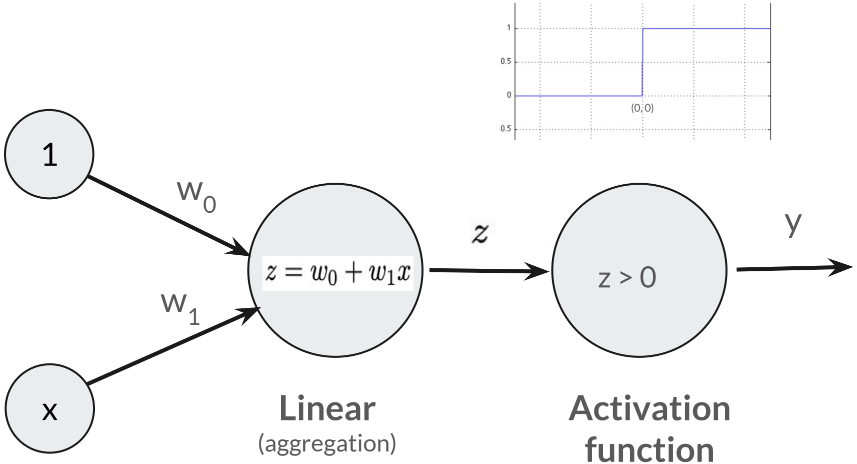
Neural Networks (Node)



$y \in R$

A NN with Sigmoid Activation function

Neural Networks (Node)

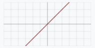
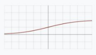
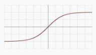


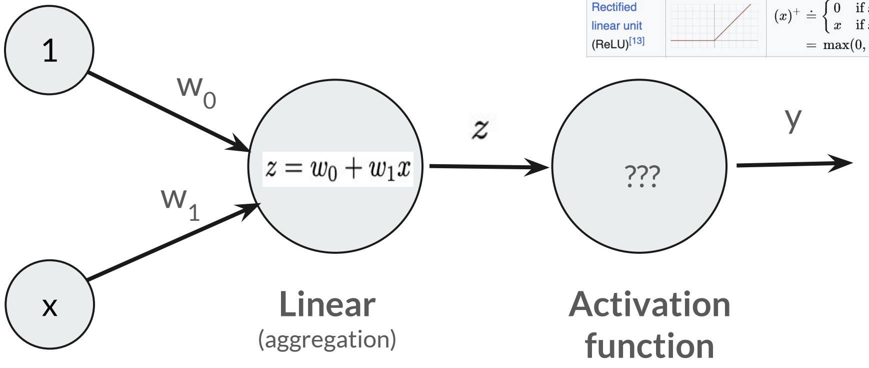
$y \in \{0, 1\}$

A NN with Step Activation function

Frank Rosenblatt's Perceptron

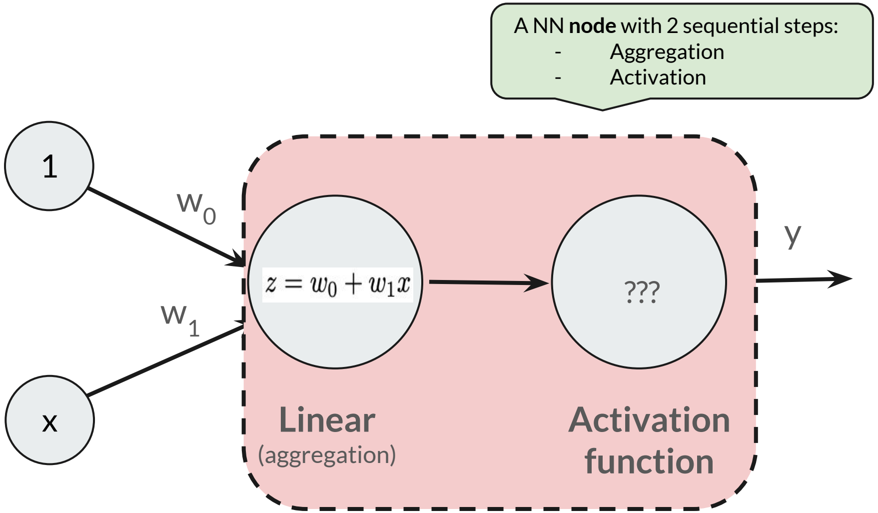
Neural Networks (Node)

Name	Plot	Function, $g(x)$
Identity		x
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$
Logistic, sigmoid, or soft step		$\sigma(x) \doteq \frac{1}{1+e^{-x}}$
Hyperbolic tangent (tanh)		$\tanh(x) \doteq \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Soboleva modified hyperbolic tangent (smht)		$\operatorname{smht}(x) \doteq \frac{e^{ax} - e^{-bx}}{e^{cx} + e^{-dx}}$
Rectified linear unit (ReLU)[13]		$(x)^+ \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$



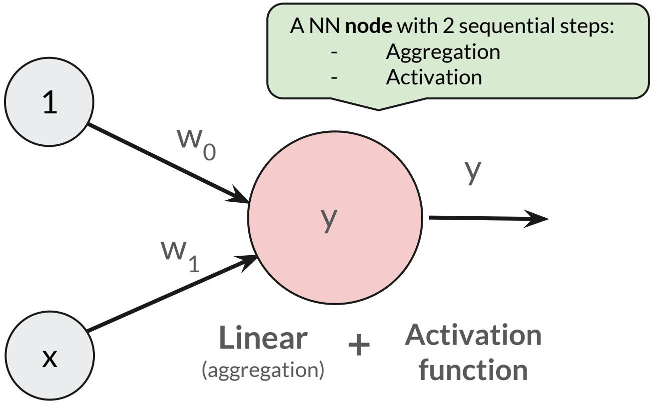
A NN with ???
Activation function

Neural Networks (Node)

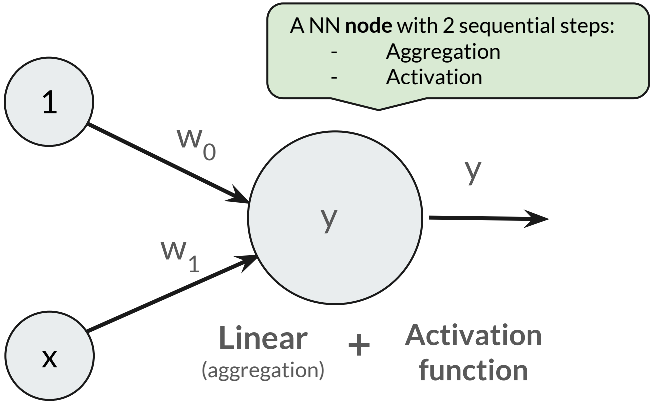


A NN with ???
Activation function

Neural Networks (Node)



Neural Networks (No Hidden Layer)



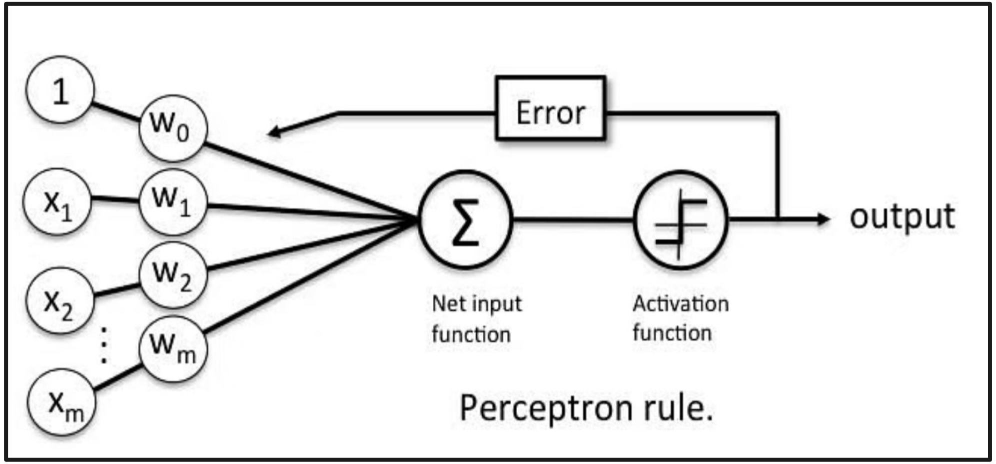
A simple **one leayer NN** with no hidden layer

Perceptron: the first Neural Network

Motivation src: Biological neuron

Perceptron was introduced by **Frank Rosenblatt** in 1957.

A binary classifier



[perceptron]

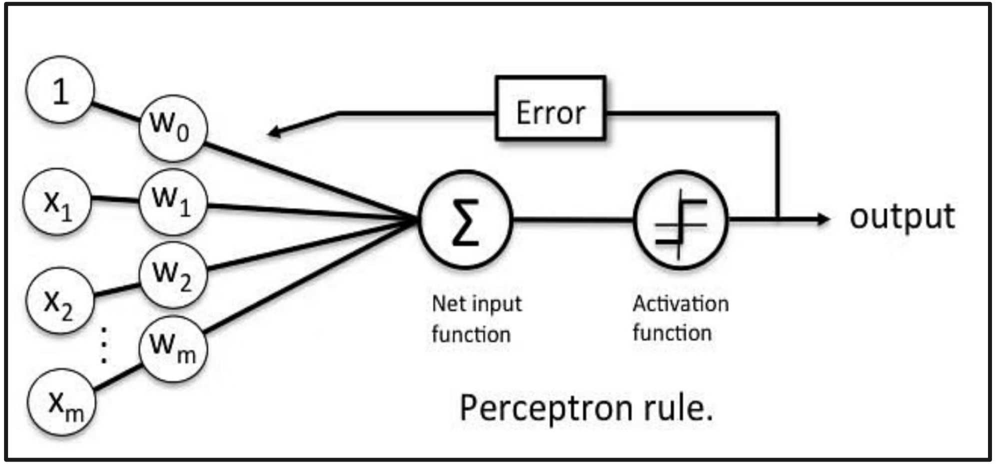
Perceptron: the first Neural Network

Motivation src: Biological neuron

Perceptron was introduced by **Frank Rosenblatt** in 1957.

A binary classifier

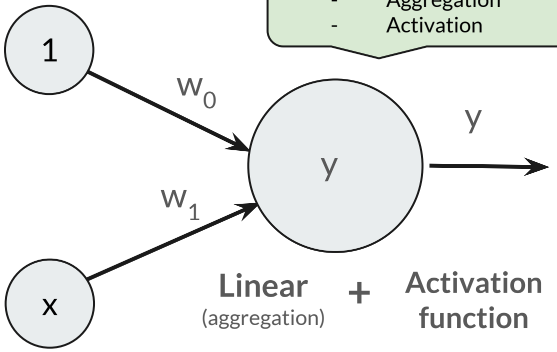
[Professor's perceptron paved the way for AI – 60 years too soon]



perceptron

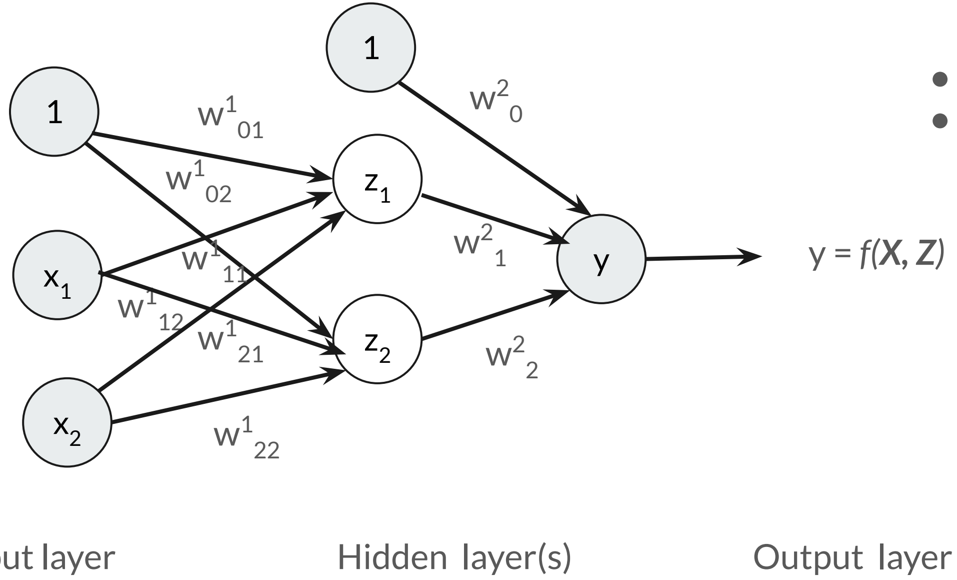
Neural Networks (No Hidden Layer)

A NN **node** with 2 sequential steps:
- Aggregation
- Activation



To take the advantage of the NN, you require at least one Hidden layer

Feed-forward (FF) neural networks



- Feedforward NN
- Perceptron

Feed-forward (FF) neural networks

x_1

x_2

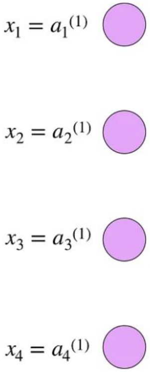
x_3

x_4

Input
(X)

mainly adapted from

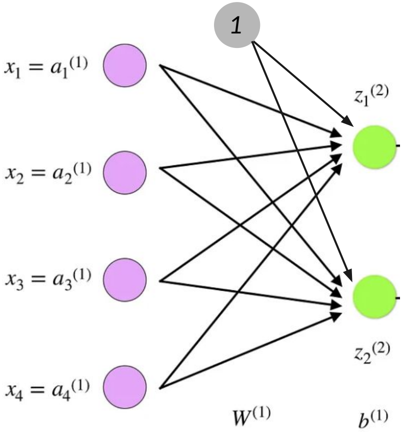
Feed-forward (FF) neural networks



A = X

[mainly adapted from]

Feed-forward (FF) neural networks



Input layer

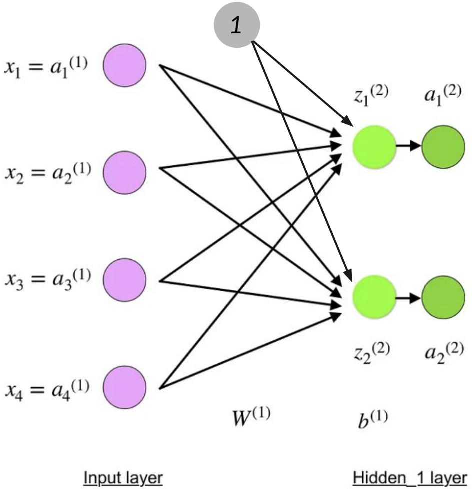
Hidden_1 layer

A = X

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

mainly adapted from

Feed-forward (FF) neural networks



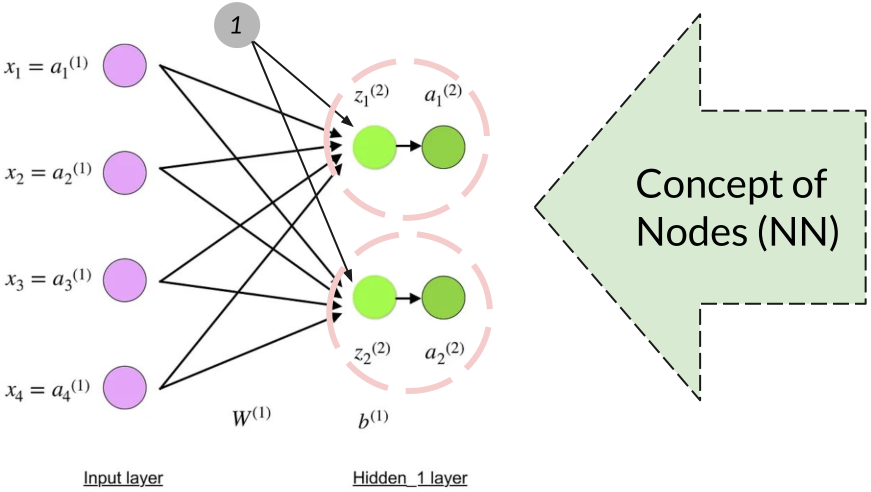
A = X

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

[mainly adapted from]()

Feed-forward (FF) neural networks



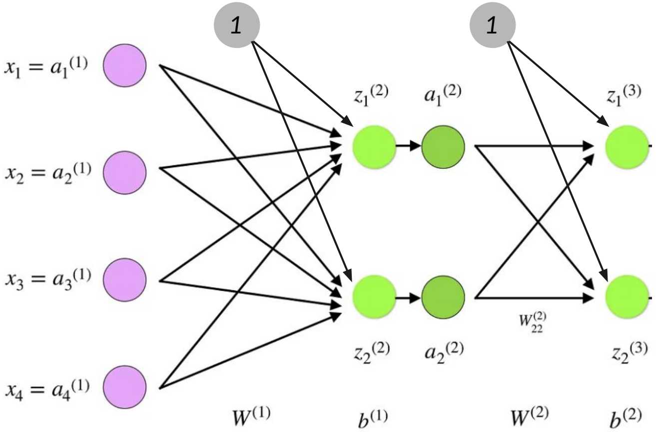
A = X

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

mainly adapted from

Feed-forward (FF) neural networks



Input layer

A = X

Hidden_1 layer

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

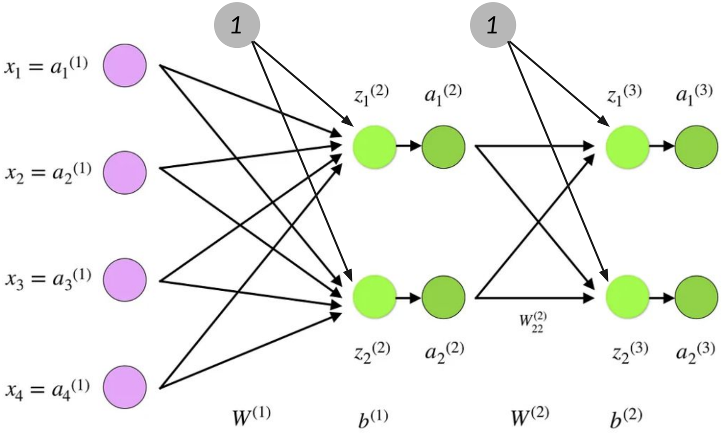
$$a^{(2)} = f(z^{(2)})$$

Hidden_2 layer

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

mainly adapted from

Feed-forward (FF) neural networks



Input layer

A = X

Hidden_1 layer

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

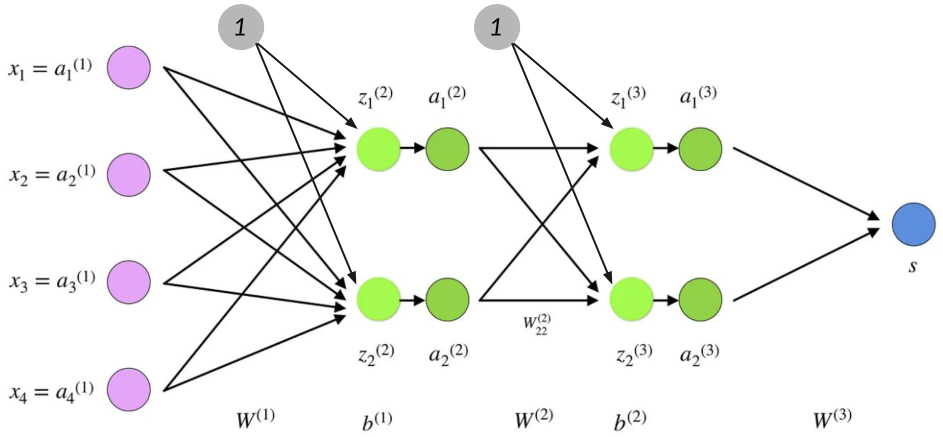
Hidden_2 layer

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

mainly adapted from

Feed-forward (FF) neural networks



Input layer

A = X

Hidden_1 layer

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

Hidden_2 layer

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

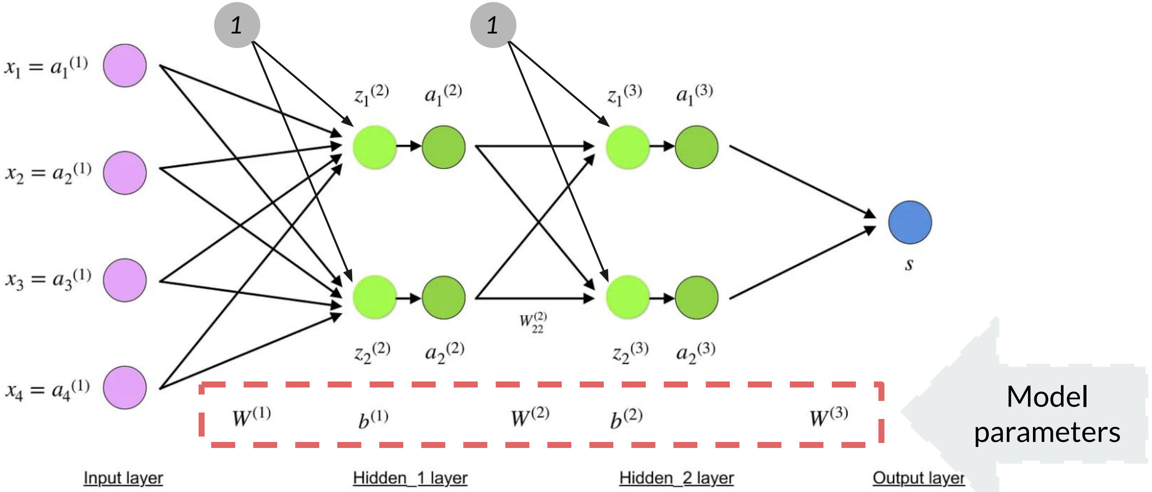
$$a^{(3)} = f(z^{(3)})$$

Output layer

$$s = W^{(3)}a^{(3)}$$

mainly adapted from

Feed-forward (FF) neural networks



A = X

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(3)} = f(z^{(3)})$$

$$s = W^{(3)}a^{(3)}$$

[mainly adapted from](#)

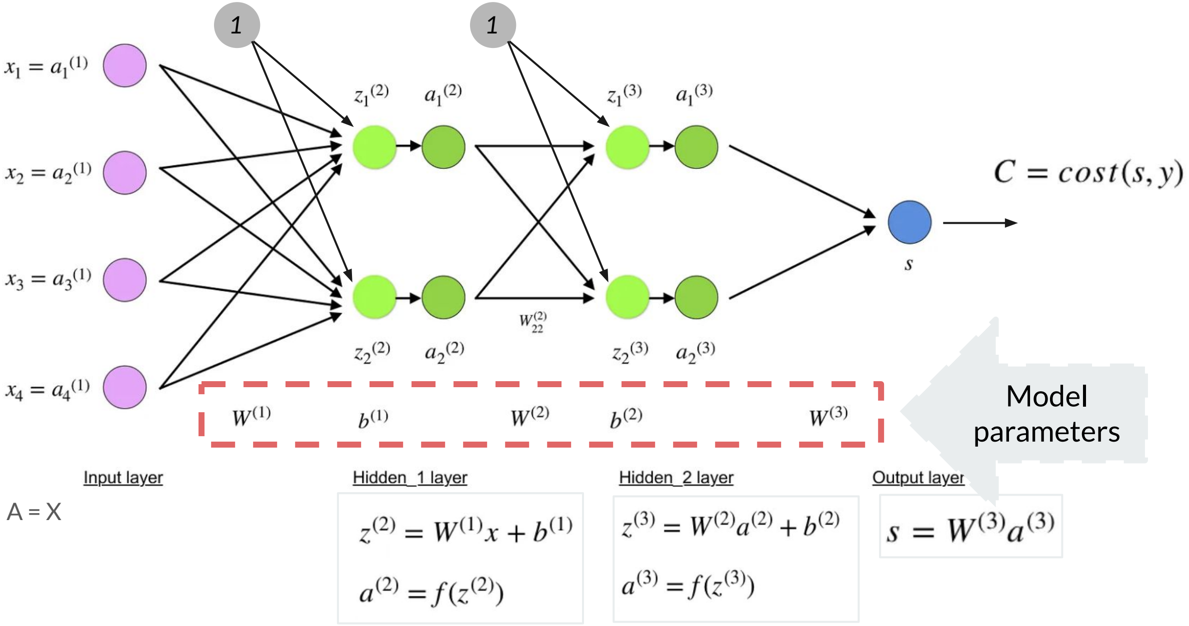
Question?

Q. Draw the diagram of a Feed Forward Neural Network with the properties given below, and estimate the minimum number of parameters your model would have:

1. **Input layer**: 3 nodes (to consume 3 input features, $\{x_1 x_{2,} x_3\}$)
2. **Three (3) Hidden layers** with the following configuration:
   i) Hidden layer one: 3 nodes
   ii) Hidden layer two: 2 nodes
   iii) Hidden layer three: 4 nodes
1. **One bias input node** for each hidden layer in (2)
2. **Output layer**: 1 node (y)

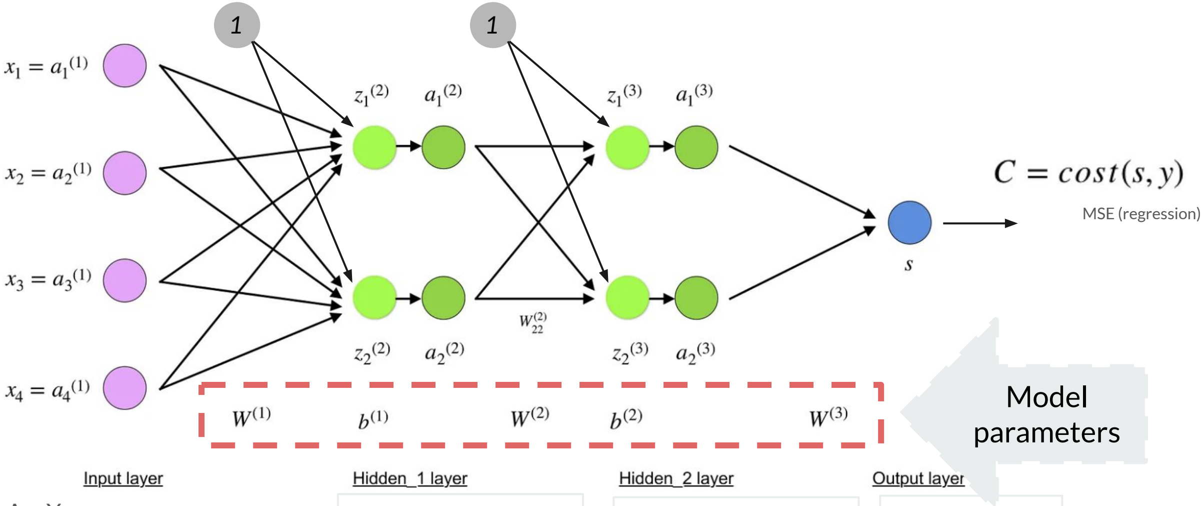
NN Training

Training



[mainly adapted from]

Training



A = X

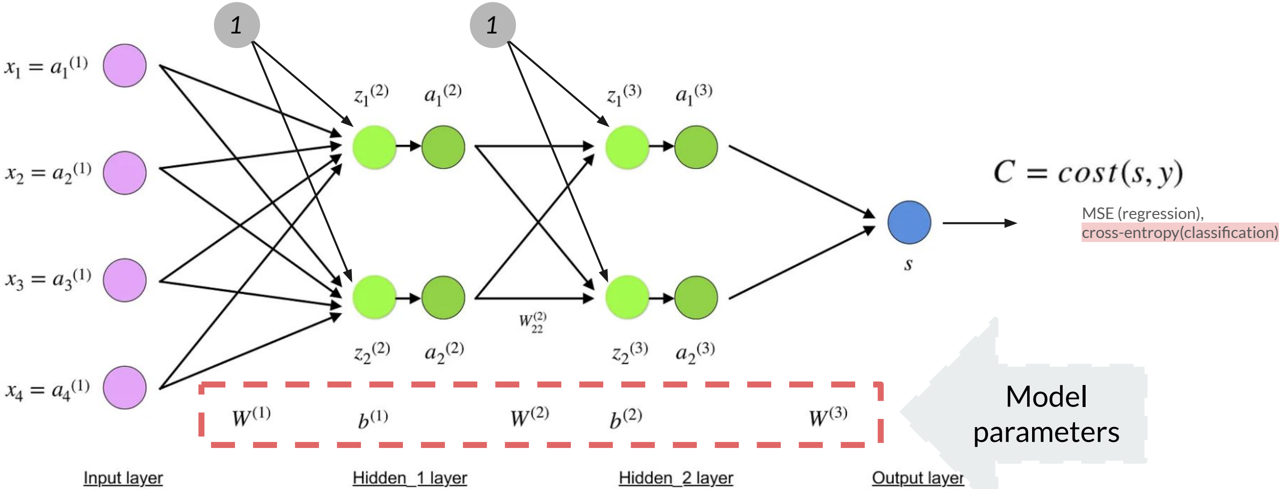
$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(3)} = f(z^{(3)})$$

$$s = W^{(3)}a^{(3)}$$

mainly adapted from

Training



A = X

$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(3)} = f(z^{(3)})$$

$$s = W^{(3)}a^{(3)}$$

[mainly adapted from]()

Gradients

- **x** is your parameter vector/matrix
- Partial derivatives
- Only the last (hidden) layer parameters can have direct derivatives
- Rest (including the input layer) requires to apply a chain rule

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \ldots, \frac{\partial C}{\partial x_m}]$$

mainly adapted from

Gradients

- **x** is your parameter vector
- Partial derivatives
- Only the last (hidden) layer parameters can have direct derivatives
- Rest (including the input layer) requires to apply a chain rule

l: layer index
j: node index in layer l,
k: node index in layer l-1

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} \qquad final\ value$$

mainly adapted from

Gradients

- **x** is your parameter vector
- Partial derivatives
- Only the last (hidden) layer parameters can have direct derivatives
- Rest (including the input layer) requires to apply a chain rule

l: layer index
j: node index in layer l,
k: node index in layer l-1

Hidden layer

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} a_k^{l-1} \qquad final\ value$$

mainly adapted from

Gradients

- **x** is your parameter vector
- Partial derivatives
- Only the last (hidden) layer parameters can have direct derivatives
- Rest (including the input layer) requires to apply a chain rule

l: layer index
j: node index in layer l,
k: node index in layer l-1

Hidden layer

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j}\frac{\partial z^l_j}{\partial w^l_{jk}} \qquad \textit{chain rule}$$

constant

$$z^l_j = \sum_{k=1}^{m} w^l_{jk} a^{l-1}_k + b^l_j \qquad \textit{by definition}$$

$$m \ - \ \textit{number of neurons in } l-1 \textit{ layer}$$

$$\frac{\partial z^l_j}{\partial w^l_{jk}} = a^{l-1}_k \qquad \textit{by differentiation (calculating derivative)}$$

$$\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} a^{l-1}_k \qquad \textit{final value}$$

mainly adapted from

Gradients

- **x** is your parameter vector
- Partial derivatives
- Only the last (hidden) layer parameters can have direct derivatives
- Rest (including the input layer) requires to apply a chain rule

l: layer index
j: node index in layer l,
k: node index in layer l-1

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial b_j^l} \qquad \textit{chain rule}$$

Derivative with respect to bias is 1

$$\frac{\partial z_j^l}{\partial b_j^l} = 1 \qquad \textit{by differentiation (calculating derivative)}$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l}1 \qquad \textit{final value}$$

mainly adapted from

Gradient descent

- Can you recall our gradient descent Linear Regression model training?

$$\mathit{while}\ (\mathit{termination\ condition\ not\ met})$$

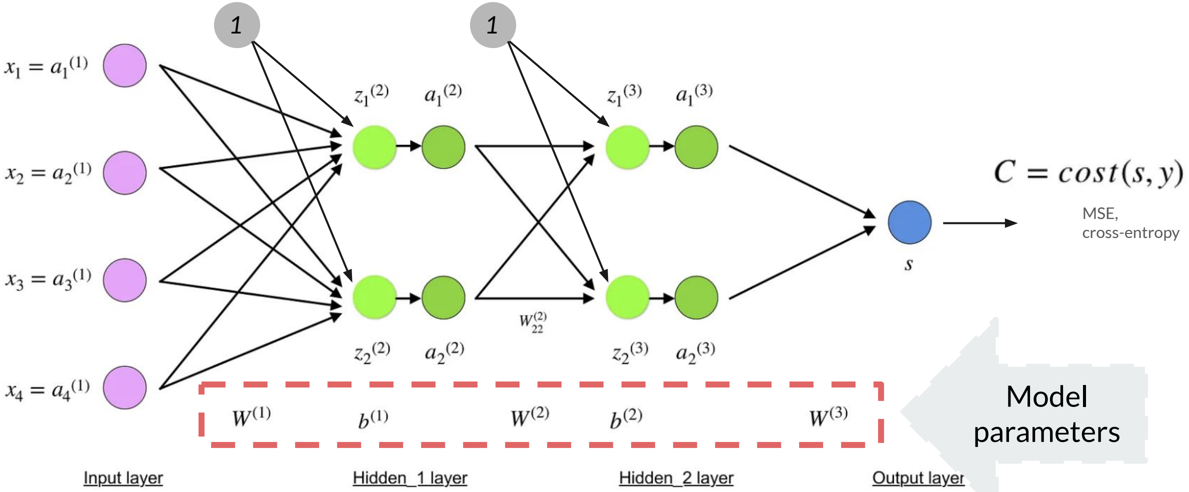
$$w := w - \epsilon \frac{\partial C}{\partial w}$$

$$b := b - \epsilon \frac{\partial C}{\partial b}$$

$$\mathit{end}$$

mainly adapted from

Training



A = X

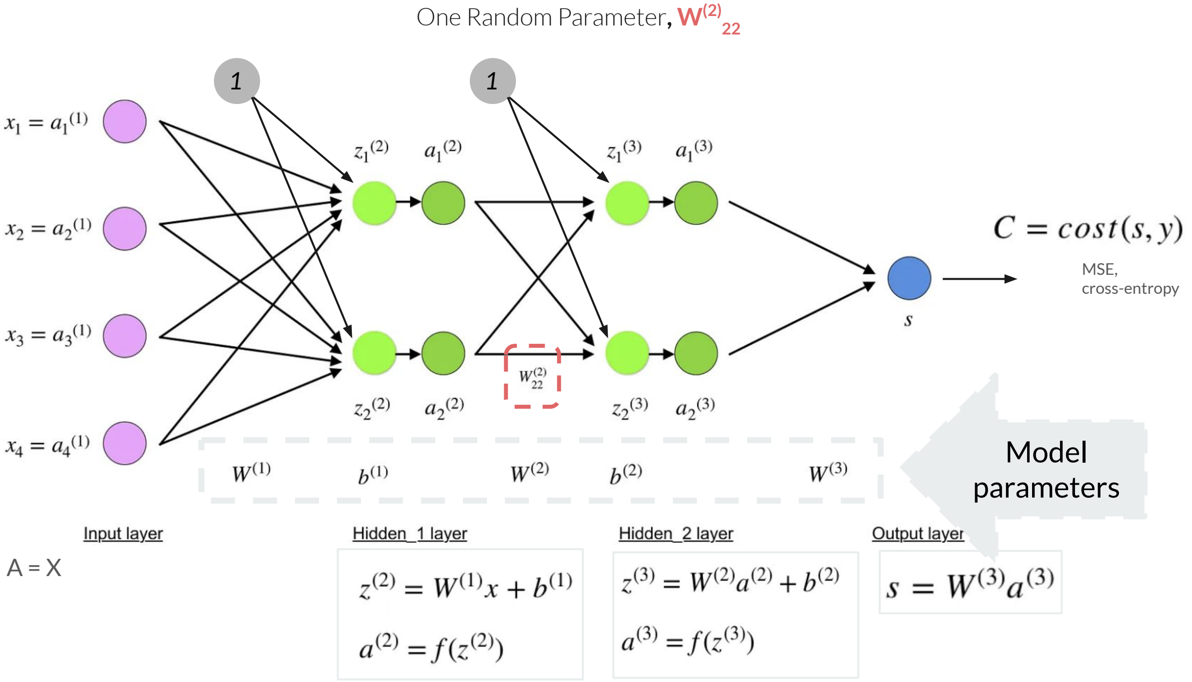
$$z^{(2)} = W^{(1)}x + b^{(1)}$$
$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$
$$a^{(3)} = f(z^{(3)})$$

$$s = W^{(3)}a^{(3)}$$

mainly adapted from

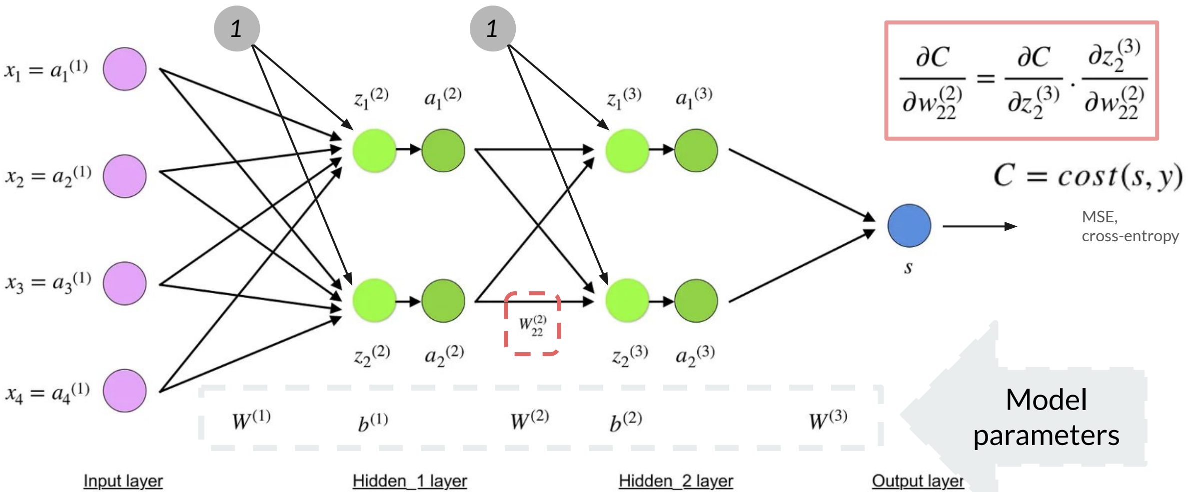
Training



mainly adapted from

Training

One Random Parameter, **$W^{(2)}_{22}$**



A = X

$$z^{(2)} = W^{(1)}x + b^{(1)}$$

$$a^{(2)} = f(z^{(2)})$$

$$z^{(3)} = W^{(2)}a^{(2)} + b^{(2)}$$

$$a^{(3)} = f(z^{(3)})$$

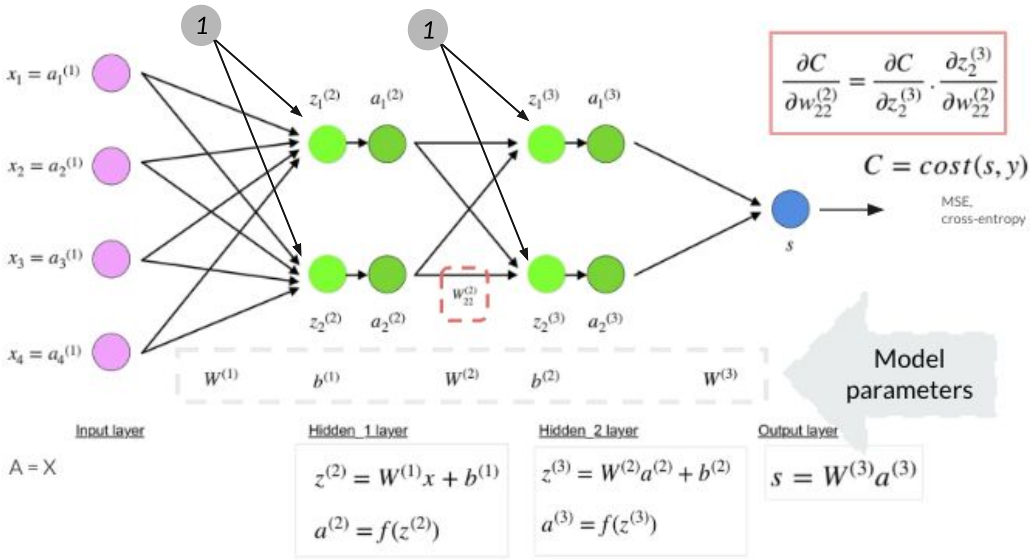
$$s = W^{(3)}a^{(3)}$$

mainly adapted from

Error Backpropagation

One Random Parameter, $W^{(2)}_{22}$

$$\frac{\partial C}{\partial w^{(2)}_{22}} = \frac{\partial C}{\partial z^{(3)}_2} \cdot \frac{\partial z^{(3)}_2}{\partial w^{(2)}_{22}}$$



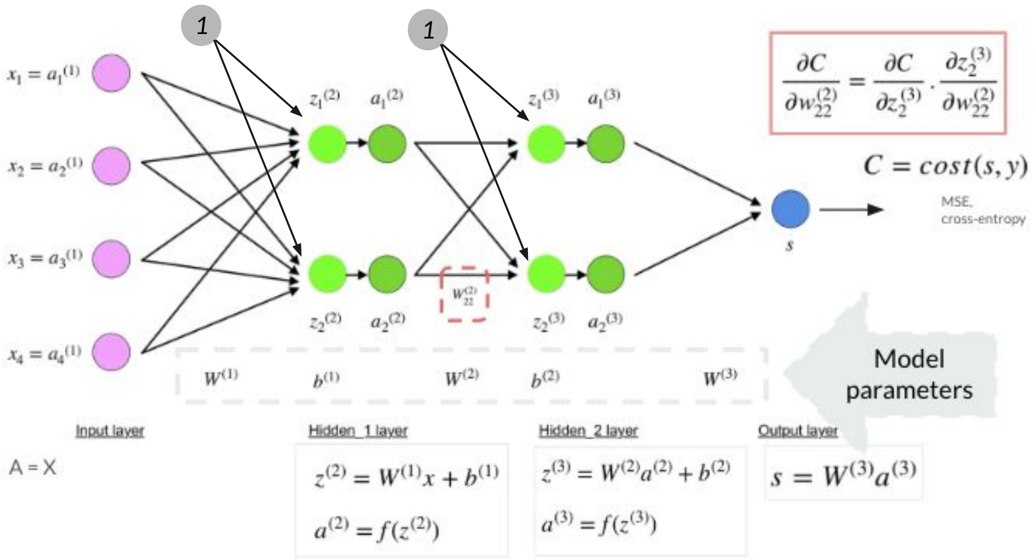
mainly adapted from

Error Backpropagation

One Random Parameter, $W^{(2)}_{22}$

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$$

$$= \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)}$$



mainly adapted from

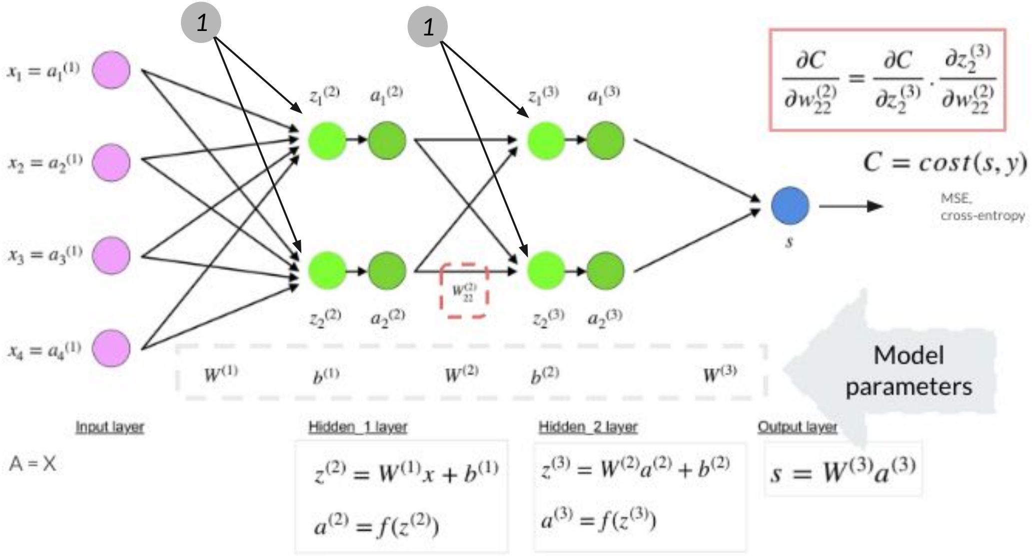
Error Backpropagation

One Random Parameter, $W^{(2)}_{22}$

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$$

$$= \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \, . \, a_2^{(2)}$$

$$= \frac{\partial C}{\partial a_2^{(3)}} \, . f'(z_2^{(3)}) \, . \, a_2^{(2)}$$



mainly adapted from

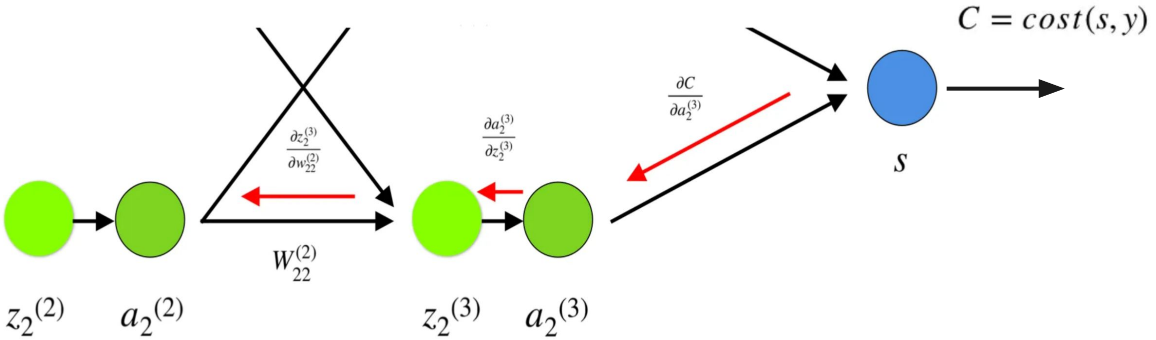
Error Backpropagation

One Random Parameter, $W^{(2)}_{22}$

$$\frac{\partial C}{\partial w_{22}^{(2)}} = \frac{\partial C}{\partial z_2^{(3)}} \cdot \frac{\partial z_2^{(3)}}{\partial w_{22}^{(2)}}$$

$$= \frac{\partial C}{\partial a_2^{(3)}} \cdot \frac{\partial a_2^{(3)}}{\partial z_2^{(3)}} \cdot a_2^{(2)}$$

$$= \frac{\partial C}{\partial a_2^{(3)}} \cdot f'(z_2^{(3)}) \cdot a_2^{(2)}$$



[mainly adapted from]()

QA