Neural network

A gentle introduction: Neural network for supervised learning

Khiem Nguyen

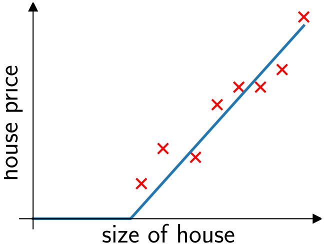
Email	khiem.nguyen@glasgow.ac.uk
MS Teams	khiem.nguyen@glasgow.ac.uk
Whatsapp	+44 7729 532071 (Emergency only)

May 18, 2025

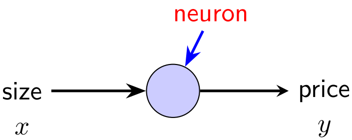
University of Glasgow

Table of Contents

Introduction to neural network: Revisit linear regression



Simplest neural network:



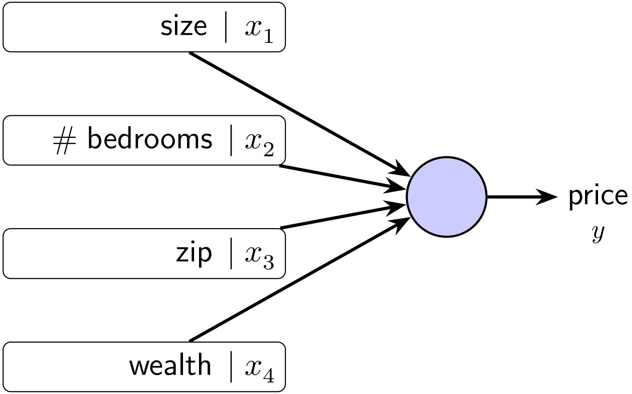
Hypothesis function:

$$h_{w,b} = \mathrm{ReLU}(wx + b),$$

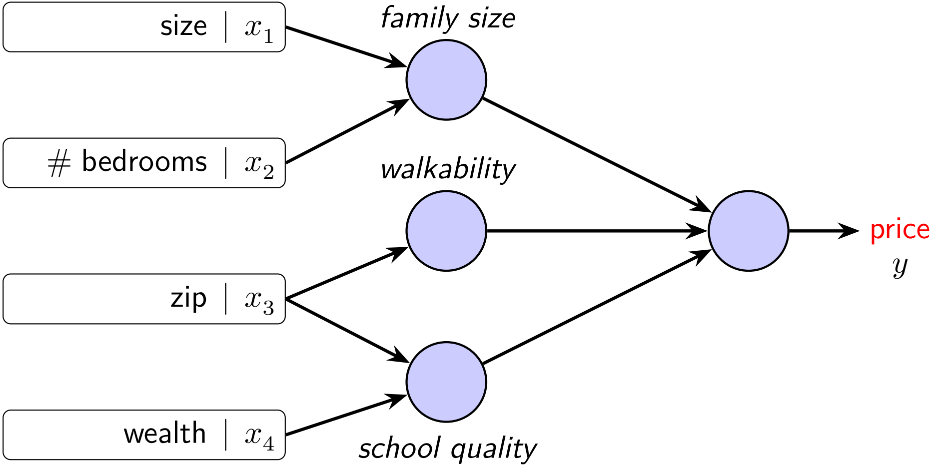
with

$$\mathrm{ReLU}(z) = \begin{cases} z & \forall z \geq 0 \\ 0 & \forall z < 0 \end{cases}$$

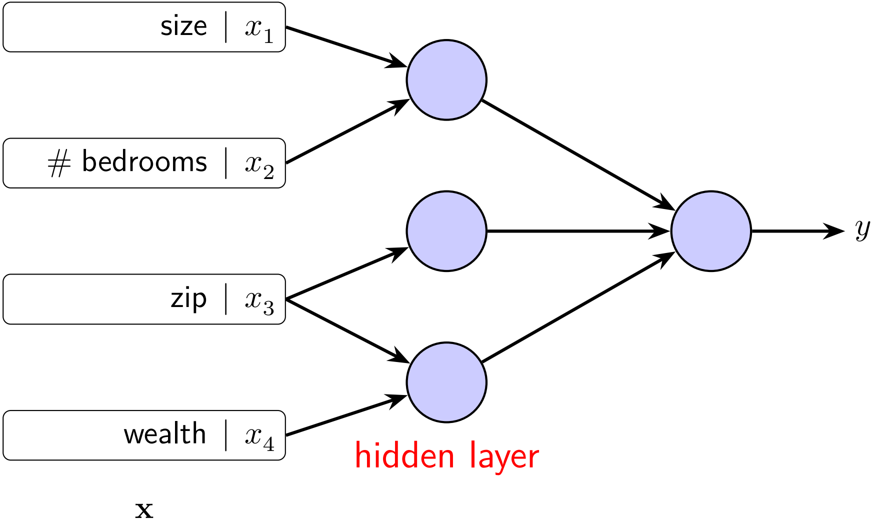
Introduction to neural network: Revisit linear regression



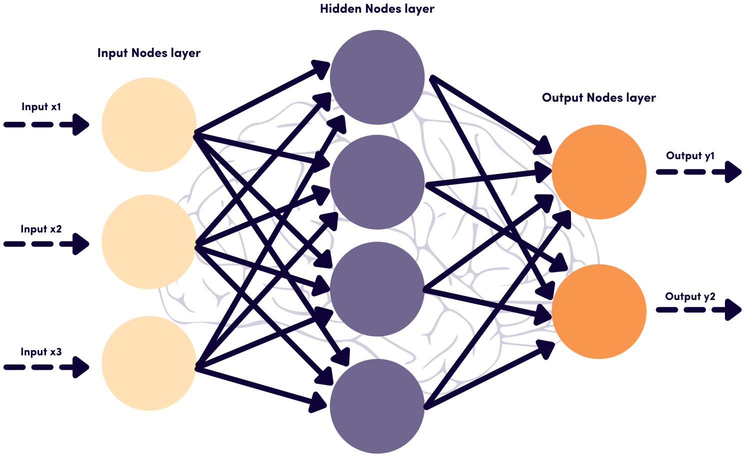
Introduction to neural network: Hidden layer with abstract features



Introduction to neural network: More abstraction



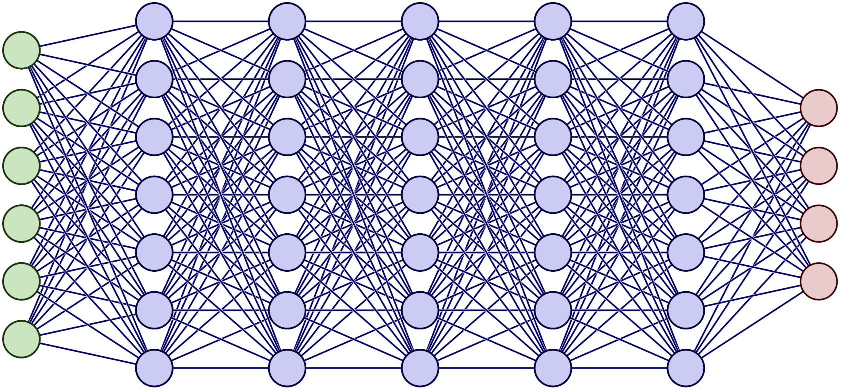
A typical neural network



Supervised learning with Neural Networks

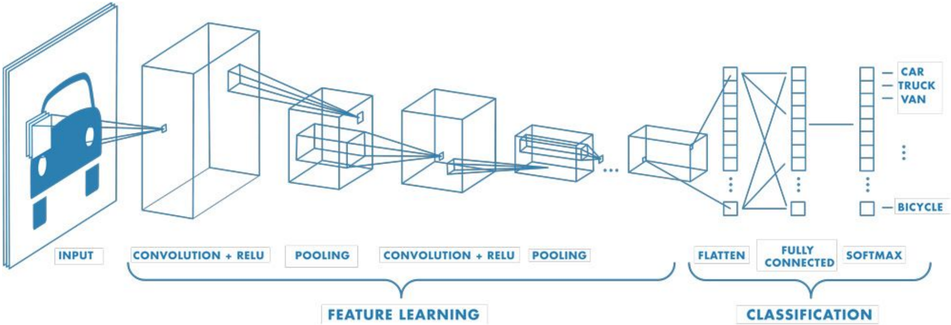
Input (x)	Output (y)	Application
Home features	Price	Real Estate (standard NN)
Ads, user info	Click on ad? $(0/1)$	Online Advertising (standard NN)
Image	Object $(1, \dots, 1000)$	Photo tagging (CNN)
Audio	Text transcript	Speech recognition (RNN)
English	Chinese	Machine translation (RNN)
Image, Radar Info	Position of other cards	Autonomous driving (RNN)

What we will learn



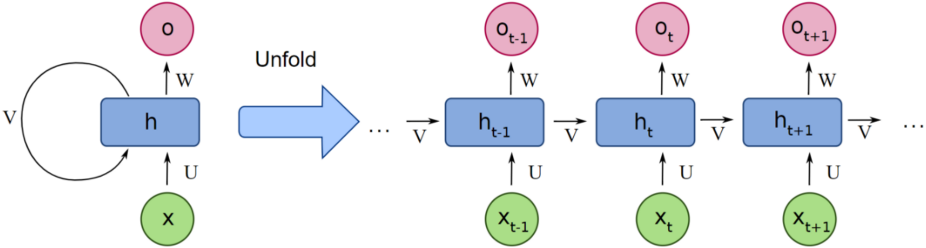
A quite deep Artificial Neural Network (ANN)

What we will learn



A Convolution Neural Network

What we will learn



A Recurrent Neural Network (RNN)

Motivation problem: Handwritten digits

We motivate ourselves by solving the classical problem: Classify handwritten digits

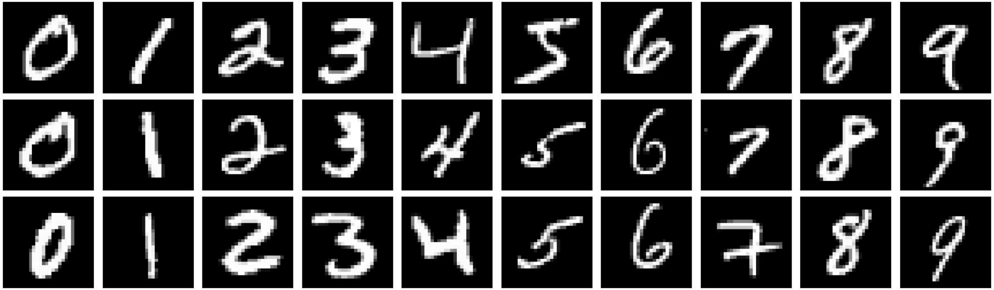


Figure: MNIST digits

- Grayscale image: value of
- Size of one image: 28×28

Motivation problem: Binary classification between 0 and 1

Let us start simple by a binary classification problem: classify handwritten digits 0 and 1.

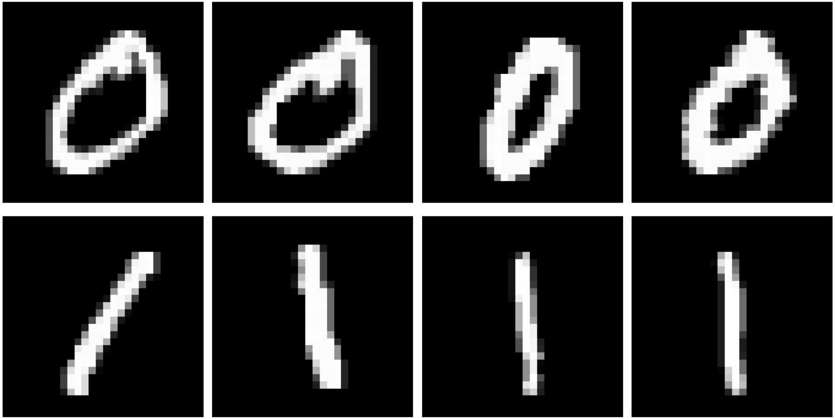


Figure: Classify 0 versus 1

Image representation

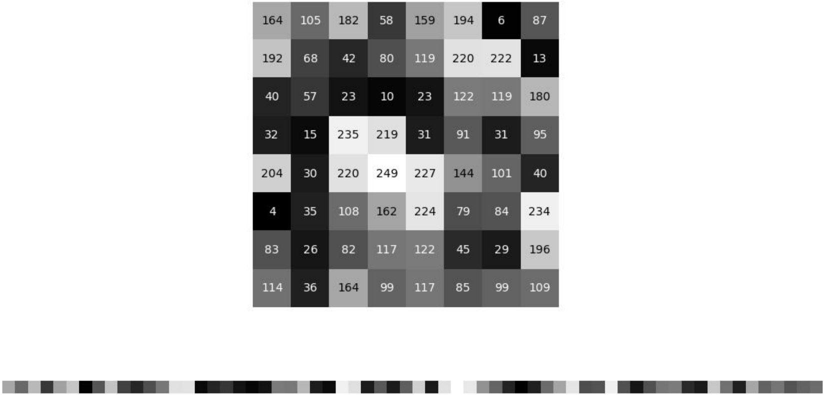


Figure: Pixels with their values – Flatten an image into a vector

In NumPy, we flatten a two-dimensional array into a vector using `A.flatten()`

Notation: Input features and label

- One training example: $(\mathbf{x}, y) \quad \mathbf{x} \in \mathbb{R}^d, \quad y \in \{0, 1\}$
- Data set: m training examples $\quad \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \cdots, (\mathbf{x}^{(m)}, y^{(m)})\}$
- Training set vs test set: $\quad m = m_{\text{train}}, \qquad m_{\text{test}} = \#$ test examples

$$\mathbf{X} = \begin{bmatrix} \text{——} \mathbf{x}^{(1)} \text{——} \\ \text{——} \mathbf{x}^{(2)} \text{——} \\ \vdots \\ \text{——} \mathbf{x}^{(m)} \text{——} \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

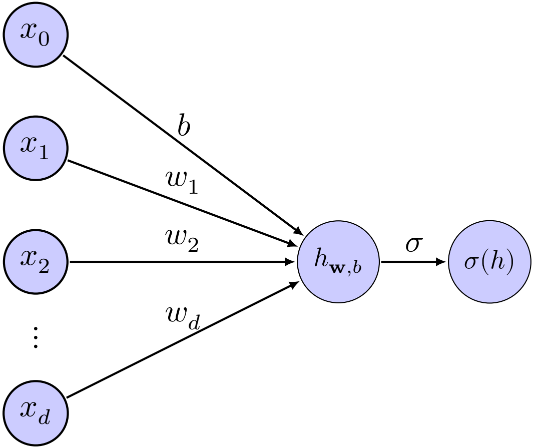
$$\mathbf{X} \in \mathbb{R}^{d \times m}, \quad \mathbf{Y} \in \mathbb{R}^{1 \times m}$$

Remark

- In many textbooks, the column vector is used to denote one training input data $\mathbf{x}$.
- In this course, we try to use the notations that can reflect the PyTorch code as directly as possible.

Table of Contents

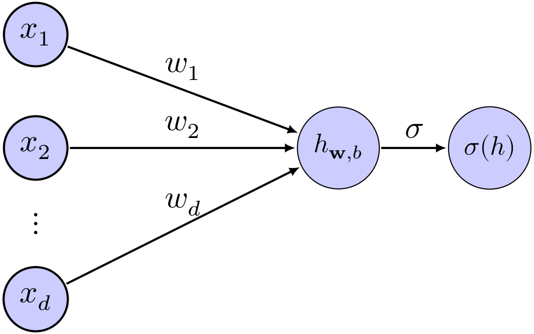
Logistic regression model



Recall that

$\mathbf{x} = (x_1, x_2, \dots x_d)$	input features
$a = \dfrac{1}{1 + \mathrm{e}^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$	output
$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots x_d^{(i)})$	a training example
$y^{(i)} \in \{0, 1\}$	true label

Logistic regression model



➢ The bias b is interpreted and implicitly applied to simplify the drawing.

Recall that

$\mathbf{x} = (x_1, x_2, \dots x_d)$	input features
$a = \dfrac{1}{1 + \mathrm{e}^{-(\mathbf{w}\cdot\mathbf{x}+b)}}$	output
$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots x_d^{(i)})$	a training example
$y^{(i)} \in \{0, 1\}$	true label

Logistic regression model

- Recall:

$$\mathbf{x} = (x_1, x_2, \dots x_d) \qquad \text{input features}$$

$$a = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + \mathrm{e}^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \qquad \text{output}$$

$$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots x_d^{(i)}) \qquad \text{a training example}$$

$$y^{(i)} \in \{0, 1\} \qquad \text{true label}$$

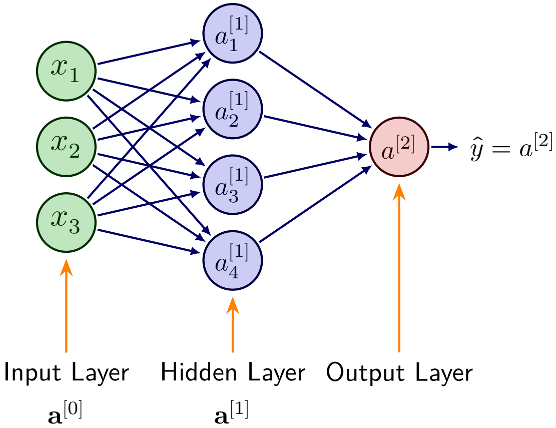
- Minimize the binary cross-entropy loss function L:

$$\min_{\mathbf{w}, b} L(\mathbf{w}, b)$$

$$L = \frac{1}{m} \sum_{i=1}^{m} \left(-y^{(i)} \log[a(\mathbf{x}^{(i)})] - (1 - y^{(i)}) \log[1 - a(\mathbf{x}^{(i)})] \right)$$

- Easily done with `scikit-learn`.

Neural network representation: A simple example

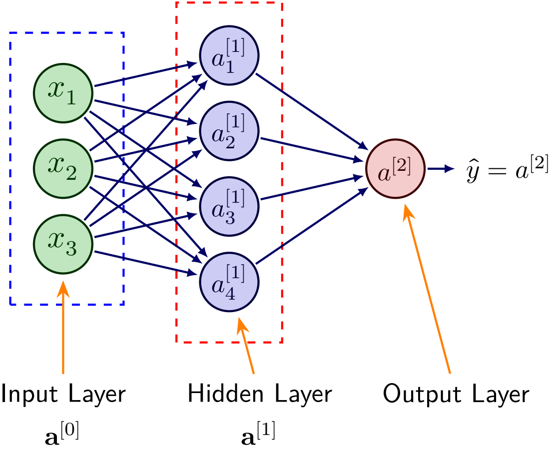


- $a_1^{[1]}$ is a function of (x_1, x_2, x_3)
- $a_2^{[1]}$ is a function of (x_1, x_2, x_3)
- $a_3^{[1]}$ is a function of (x_1, x_2, x_3)
- $a_4^{[1]}$ is a function of (x_1, x_2, x_3)
- $a^{[2]}$ is a function of $(a_1^{[1]}, a_2^{[1]}, a_3^{[1]}, a_4^{[1]})$

➡ $a^{[2]}$ is a function of (x_1, x_2, x_3)

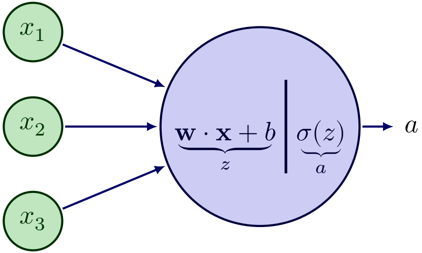
➡ That's the whole point of hypothesis function/model function!

Neural network representation: A simple example



- One layer l comprises all the nodes (neurons) in one vertical column.
- One (circle) node represents a neuron in a layer.
- Superscript $[l]$ in $\mathbf{a}^{[0]}$ and $\mathbf{a}^{[l]}$ denote the layer number.
- Subscript i in $a_i^{[l]}$ denote the index of the neuron in the layer.

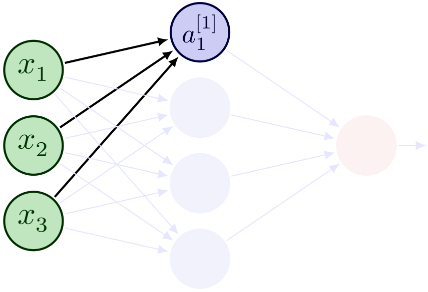
Neural network representation: A simple example

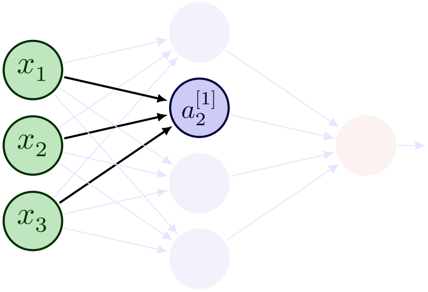


$$z = \mathbf{w}\cdot\mathbf{x} + b$$

$$a = \sigma(z)$$

Neural network representation: A simple example





$$\mathbf{w}_1^{[1]} = (w_{11}^{[1]}, w_{12}^{[1]}, w_{13}^{[1]})$$

$$z_1^{[1]} = \mathbf{w}_1^{[1]} \cdot \mathbf{x} + b_1^{[1]}$$

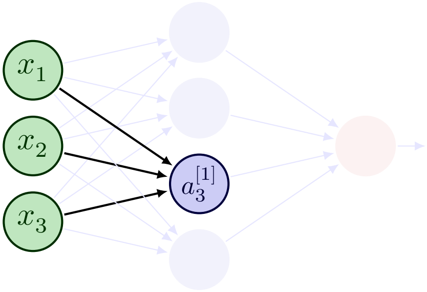
$$a_1^{[1]} = \sigma(z_1^1)$$

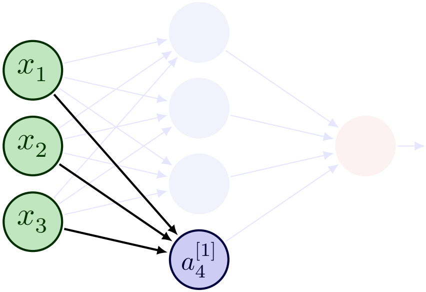
$$\mathbf{w}_2^{[1]} = (w_{21}^{[1]}, w_{22}^{[1]}, w_{23}^{[1]})$$

$$z_2^{[1]} = \mathbf{w}_2^{[1]} \cdot \mathbf{x} + b_2^{[1]}$$

$$a_2^{[1]} = \sigma(z_1^1)$$

Neural network representation: A simple example





$$\mathbf{w}_3^{[1]} = (w_{31}^{[1]}, w_{32}^{[1]}, w_{33}^{[1]})$$

$$z_3^{[1]} = \mathbf{w}_3^{[1]} \cdot \mathbf{x} + b_3^{[1]}$$

$$a_3^{[1]} = \sigma(z_3^1)$$

$$\mathbf{w}_4^{[1]} = (w_{41}^{[1]}, w_{42}^{[1]}, w_{43}^{[1]})$$

$$z_4^{[1]} = \mathbf{w}_4^{[1]} \cdot \mathbf{x} + b_3^{[1]}$$

$$a_4^{[1]} = \sigma(z_1^1)$$

Neural Network Representation: A simple example

By introducing the "activation" in the input layer with

$$\mathbf{a}^{[0]} = \mathbf{x} \quad \Leftrightarrow \quad \mathbf{a}^{[0]} = (a_1^{[0]}, a_2^{[0]}, a_3^{[0]}) = (x_1, x_2, x_3),$$

we can write

$$\begin{aligned}
z_1^{[1]} &= \mathbf{w}_1^{[1]} \cdot \mathbf{a}^{[0]} + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}) \\
z_2^{[1]} &= \mathbf{w}_2^{[1]} \cdot \mathbf{a}^{[0]} + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}) \\
z_3^{[1]} &= \mathbf{w}_3^{[1]} \cdot \mathbf{a}^{[0]} + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}) \\
z_4^{[1]} &= \mathbf{w}_4^{[1]} \cdot \mathbf{a}^{[0]} + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]})
\end{aligned}$$

As $\mathbf{w}_j^{[1]} \in \mathbb{R}^3$, let us denote its components by the second subscript j as follows

$$\mathbf{w}_j^{[1]} = (w_{j1}^{[1]}, w_{j2}^{[1]}, w_{j3}^{[1]}).$$

Neural Network Representation: Mathematical formulation

By introducing

$$\mathbf{a}^{[0]} = \begin{bmatrix} a_1^{[0]} \\ a_2^{[0]} \\ a_3^{[0]} \end{bmatrix}, \quad \mathbf{z}^{[1]} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}, \quad \mathbf{a}^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix},$$

$$\mathbf{W}^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} & w_{13}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} & w_{23}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} & w_{33}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} & w_{43}^{[1]} \end{bmatrix} = \begin{bmatrix} \text{---}\mathbf{w}_1\text{---} \\ \text{---}\mathbf{w}_2\text{---} \\ \text{---}\mathbf{w}_3\text{---} \\ \text{---}\mathbf{w}_4\text{---} \end{bmatrix}, \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

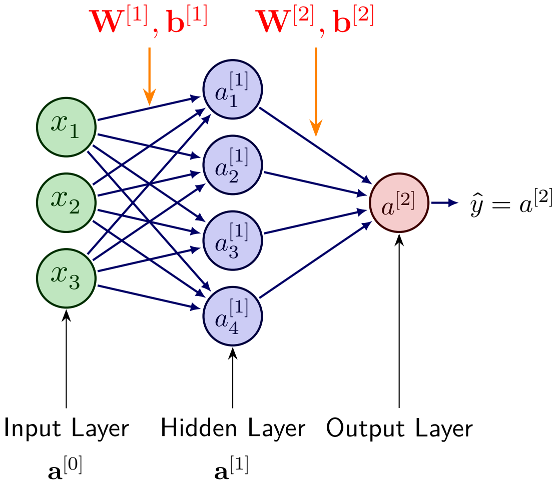
we can write the transformation from the input layer to the first hidden layer as

$$\begin{cases} \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]} \\ \mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]}) \end{cases}$$

Neural Network Representation: Mathematical formulation

$$\begin{cases} z_1^{[1]} = \mathbf{w}_1^{[1]} \cdot \mathbf{a}^{[0]} + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]}) \\ z_2^{[1]} = \mathbf{w}_2^{[1]} \cdot \mathbf{a}^{[0]} + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]}) \\ z_3^{[1]} = \mathbf{w}_3^{[1]} \cdot \mathbf{a}^{[0]} + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]}) \\ z_4^{[1]} = \mathbf{w}_4^{[1]} \cdot \mathbf{a}^{[0]} + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]}) \end{cases} \quad \Leftrightarrow \quad \begin{cases} \mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]} \\ \mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]}) \end{cases}$$

Neural Network Representation: Mathematical formulation



$$\mathbf{W}^{[1]} \in \mathbb{R}^{4\times 3}, \quad \mathbf{b}^{[1]} \in \mathbb{R}^{4\times 1},$$

$$\mathbf{W}^{[2]} \in \mathbb{R}^{1\times 4}, \quad \mathbf{b}^{[2]} \in \mathbb{R}^{1\times 1}$$

Given input $\mathbf{x} = \mathbf{a}^{[0]}$:

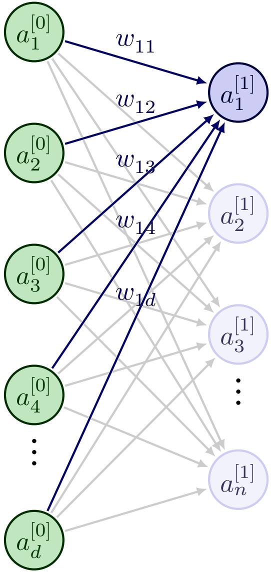
$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{a}^{[0]} + \mathbf{b}^{[1]}$$

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

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

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

Neural network representation: A quick generalization and quick note



$$\begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_m^{[1]} \end{pmatrix} = \sigma \left[\begin{pmatrix} w_{11} & w_{12} & \dots & w_{1d} \\ w_{21} & w_{22} & \dots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{n2} & \dots & w_{nd} \end{pmatrix} \begin{pmatrix} a_1^{[0])} \\ a_2^{[0]} \\ \vdots \\ a_d^{[0]} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_n^{[1]} \end{pmatrix} \right]$$

$$\mathbf{a}^{[1]} = \sigma \left(\mathbf{W}^{[1]} \mathbf{a}^{[0]} + \mathbf{b}^{[1]} \right)$$

- This mathematical presentation makes it easy to write formulation.
- It does not reflect PyTorch (also scikit-learn) implementation.
- This matrix presentation has been widely used in literature.
- One example in PyTorch is represented as a one row vector.

Mathematical formulation consistent with PyTorch

- We consider one example $\mathbf{x}^{(i)}$ and denote it $\mathbf{a}^{[0]}$ without the superscript (i).
- We can write the relationship between $\mathbf{a}^{[0]}$ and $\mathbf{a}^{[1]}$ in row-oriented format for one training example:

$$
\begin{aligned}
\begin{bmatrix} z_1^{[1]} & z_2^{[1]} & z_3^{[1]} & z_4^{[1]} \end{bmatrix} &= \begin{bmatrix} \mathbf{w}_1^{[1]} \cdot \mathbf{a}^{[0]} + b_1^{[1]} & \mathbf{w}_2^{[1]} \cdot \mathbf{a}^{[0]} + b_2^{[2]} & \mathbf{w}_3^{[1]} \cdot \mathbf{a}^{[0]} + b_3^{[1]} & \mathbf{w}_4^{[1]} \cdot \mathbf{a}^{[0]} + b_4^{[1]} \end{bmatrix} \\
&= \begin{bmatrix} a_1^{[0]} & a_2^{[0]} & a_3^{[0]} \end{bmatrix} \begin{bmatrix} w_{11}^{[1]} & w_{21}^{[1]} & w_{31}^{[1]} & w_{41}^{[1]} \\ w_{12}^{[1]} & w_{22}^{[1]} & w_{32}^{[1]} & w_{42}^{[1]} \\ w_{13}^{[1]} & w_{23}^{[1]} & w_{33}^{[1]} & w_{43}^{[1]} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} & b_2^{[1]} & b_3^{[1]} & b_4^{[1]} \end{bmatrix} \\
&= \mathbf{a}^{[0]} \begin{bmatrix} | & | & | & | \\ \mathbf{w}_1^T & \mathbf{w}_2^T & \mathbf{w}_3^T & \mathbf{w}_4^T \\ | & | & | & | \end{bmatrix} + \mathbf{b}^{[1]} = \mathbf{a}^{[0]} (\mathbf{W}^{[1]})^T + \mathbf{b}^{[1]}
\end{aligned}
$$

$$
\begin{bmatrix} a_1^{[1]} & a_2^{[1]} & a_3^{[1]} & a_4^{[1]} \end{bmatrix} = \sigma\left(\begin{bmatrix} z_1^{[1]} & z_2^{[1]} & z_3^{[1]} & z_4^{[1]} \end{bmatrix} \right) = \sigma(\mathbf{z}^{[1]})
$$

Mathematical formulation consistent with PyTorch

- We consider one example $\mathbf{x}^{(i)}$ and denote it $\mathbf{a}^{[0]}$ without the superscript (i).
- We can write the relationship between $\mathbf{a}^{[0]}$ and $\mathbf{a}^{[1]}$ in row-oriented format for one training example:

In row-oriented format for the training examples, the PyTorch consistent formulation reads

$$\begin{cases} \mathbf{z}^{[1]} = \mathbf{a}^{[0]}(\mathbf{W}^{[1]})^T + \mathbf{b}^{[1]} \\ \mathbf{a}^{[1]} = \sigma(\mathbf{z}^{[1]}) \\ \mathbf{z}^{[2]} = \mathbf{a}^{[0]}(\mathbf{W}^{[2]})^T + \mathbf{b}^{[2]} \\ \mathbf{a}^{[2]} = \sigma(\mathbf{z}^{[2]}) \end{cases}$$

This is commonly called '*forward propagation*' in a neural network.

Some activation functions

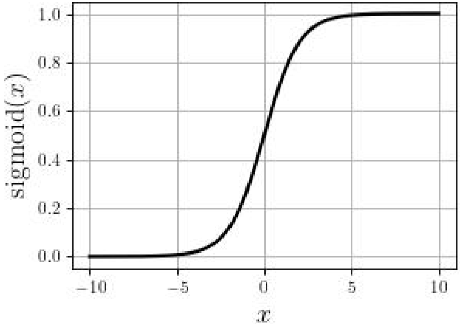


Figure: sigmoid activation

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

Some activation functions

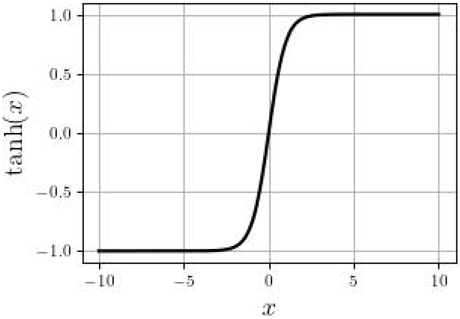


Figure: tanh activation

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

Some activation functions

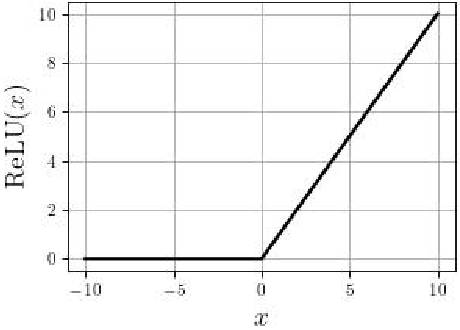


Figure: ReLU (Rectified Linear Unit) activation

$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Some activation functions

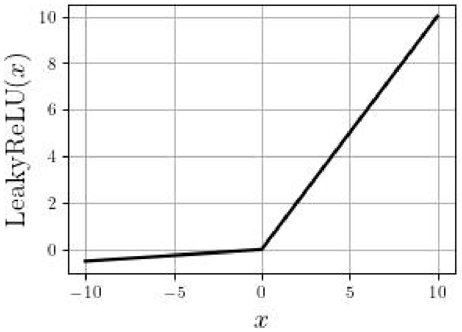


Figure: Leaky ReLU activation

$$\mathrm{ReLU}(x) = \begin{cases} x & x \geq 0 \\ -\alpha x & x < 0 \end{cases}$$

with α being called 'negative' slope ($\alpha > 0$).

Some activation functions