Multilayer Perceptron (MLP)

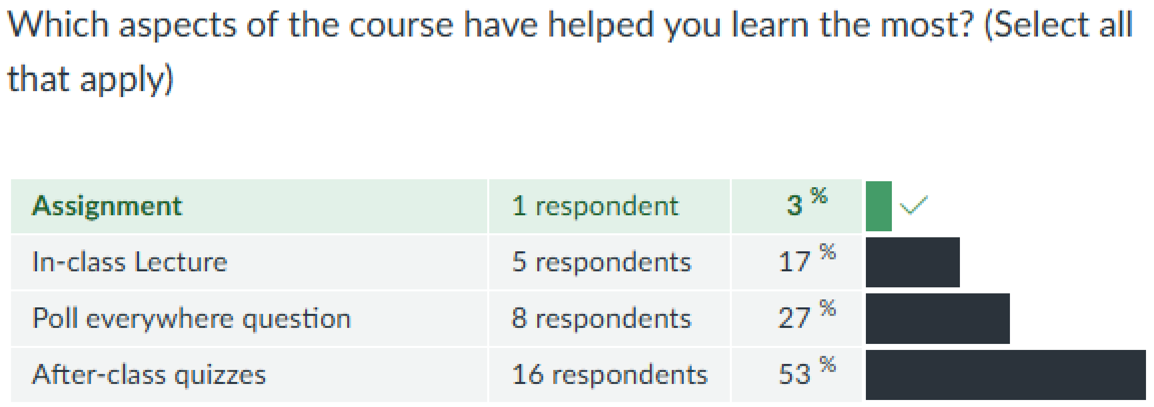
STAT362

Fall 2025

Things to note

- Midterm exam (Mean 86)

Midterm Course survey

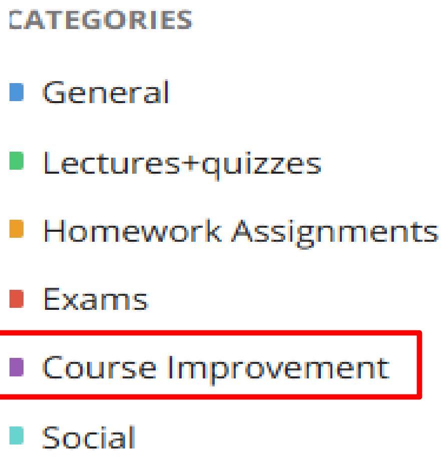


Which aspects of the course have helped you learn the most? (Select all that apply)

Assignment	1 respondent	**3** %	✓
In-class Lecture	5 respondents	17 %	
Poll everywhere question	8 respondents	27 %	
After-class quizzes	16 respondents	53 %	

- Your feedback:
  - ❑ post quiz solutions
  - ❑ shorten the quiz
  - ❑ Show some coding in lecture time

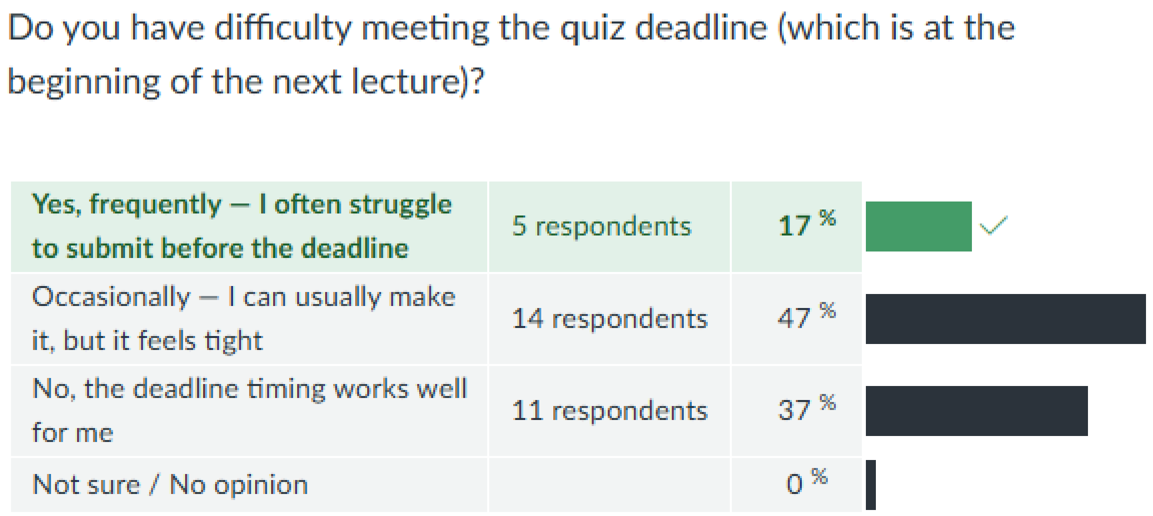
Quiz instruction is unclear?



CATEGORIES

- 🟦 General
- 🟩 Lectures+quizzes
- 🟧 Homework Assignments
- 🟥 Exams
- 🟪 Course Improvement
- 🟦 Social

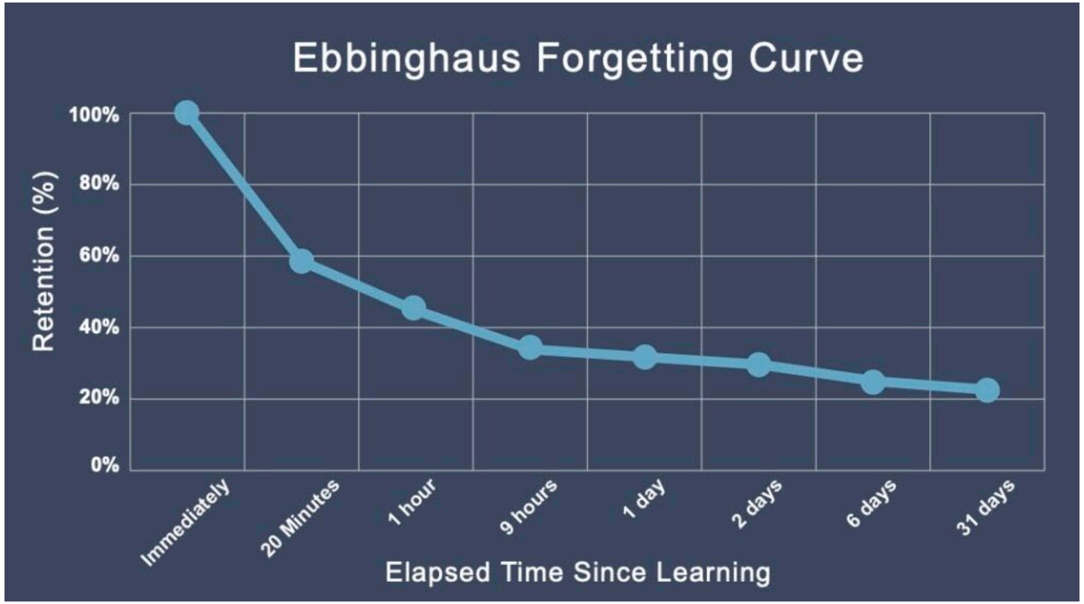
About quiz deadline



Do you have difficulty meeting the quiz deadline (which is at the beginning of the next lecture)?

Yes, frequently — I often struggle to submit before the deadline	5 respondents	17 %	✓
Occasionally — I can usually make it, but it feels tight	14 respondents	47 %	
No, the deadline timing works well for me	11 respondents	37 %	
Not sure / No opinion		0 %	

Why After-Class Quizzes Are Due Before the Next Lecture



- Our memory of newly learned material **decays exponentially** with time
- Reinforce key concepts while **they are still fresh in your mind**
- Would extending the deadline actually reduce its effectiveness?

Come prepared

Shorten the lecture time:

- Review the slides
- Post reading assignments and videos

Your final project

- Final project: proposal deadline (End of Sunday)
- Team formation: up to 4 students, put your team on canvas as well

Agenda Change for Component 3

Today

- Two-Layer
- Building your own neural network step by step

Wednesday:

- L-Layer Neural Networks and activation functions **(HW2 -> Quiz 8)**

Next Monday:

- Initialization, Optimization, and Regularization

Next Wednesday:

- **Softmax and** Keras Tutorials

Following Week:

- **Convolutional Neural Networks (CNNs)** for image processing
- **Recurrent Neural Networks (RNNs)** for sequence processing

Last Monday

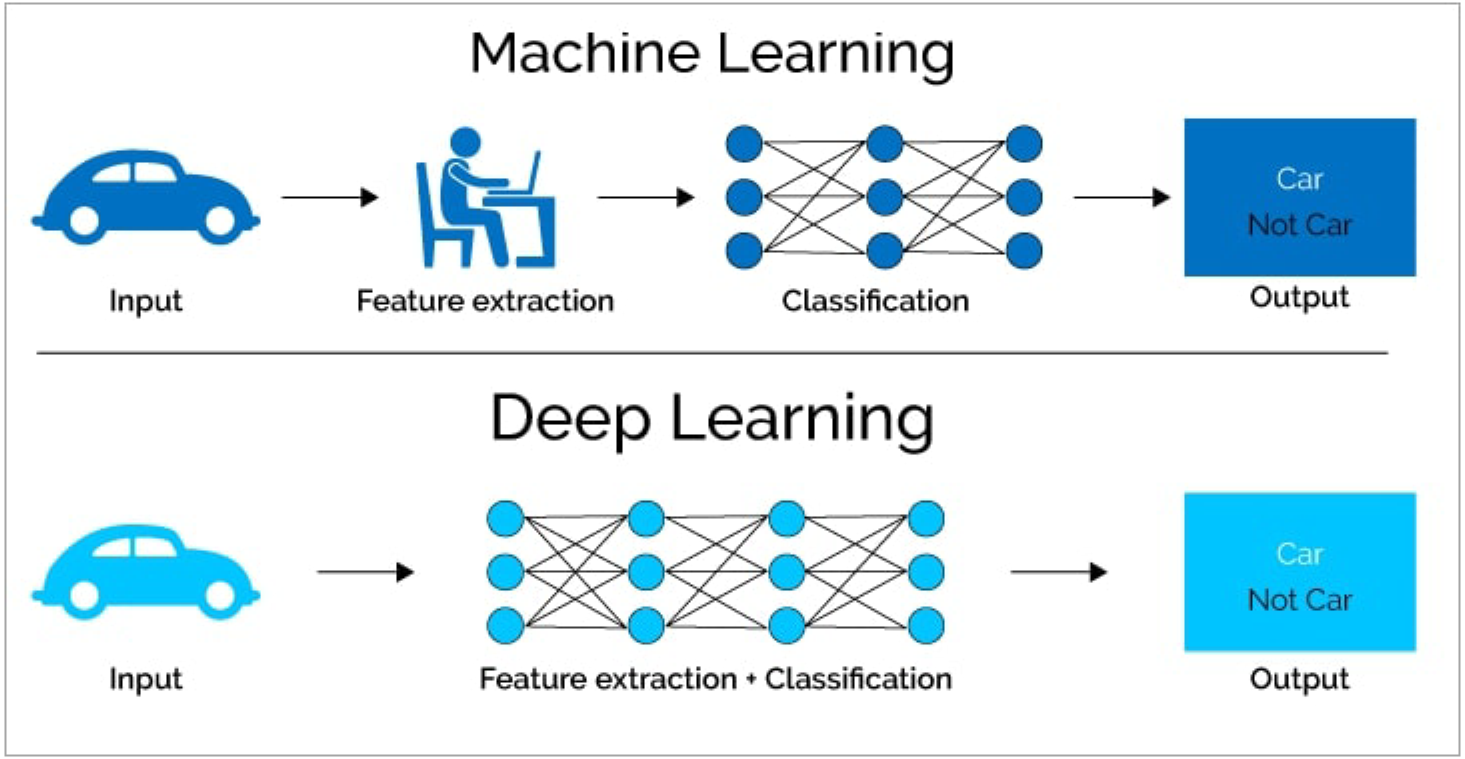
Lecture:

- Introduction to Deep Learning
- Single Neuron Model
  - ➢ **Forward propagation**
  - ➢ **Backpropagation**

After-class assignment

- Implement the forward and the backward propagation of single neural model in raw NumPy.

Last Time: How Deep Learning Differs from Traditional ML?

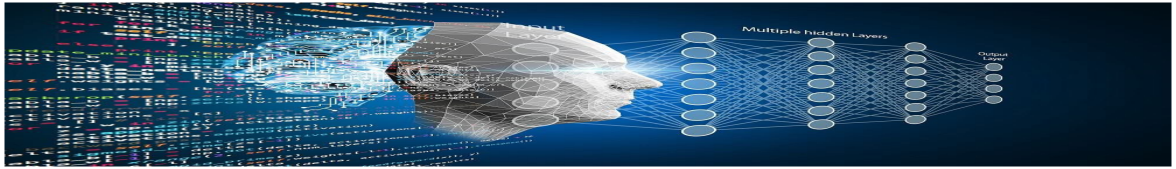


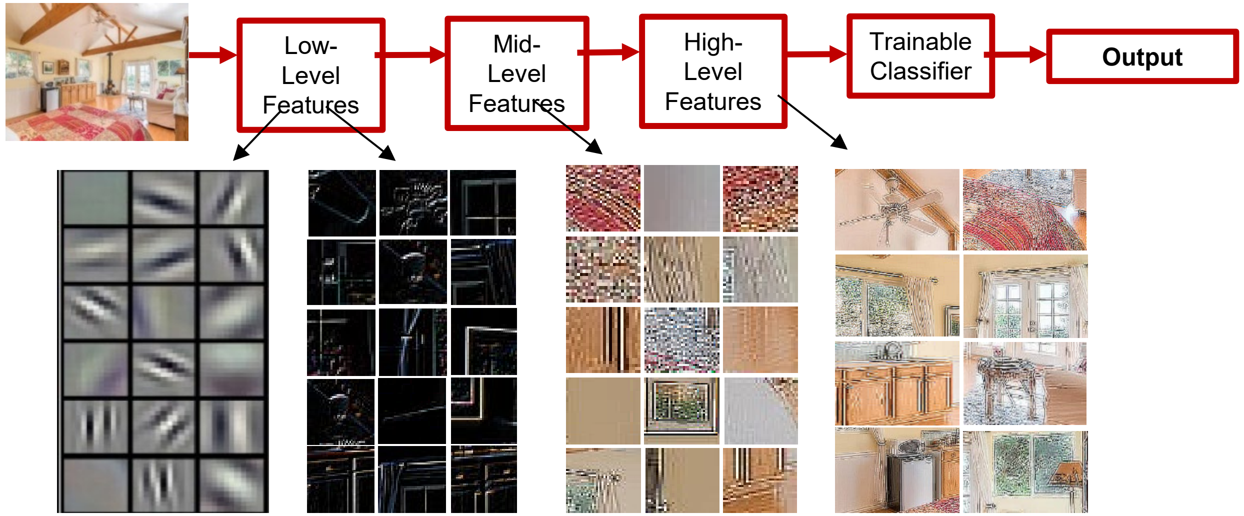
The Deep Learning algorithm doesn't need a software engineer to identify features but is capable of automatic feature engineering through its neural network. (Source: softwaretestinghelp.com)

Last time: Deep Learning for Representation Learning

DL applies a multi-layer process for learning rich **hierarchical features** (i.e., data representations) without the need for handcrafted features or extensive preprocessing.

Input image pixels → Edges → Textures → Parts → Objects





Slide credit: Param Vir Singh – Deep Learning

Last time: Why is Deep Learning Taking Off?

Data:

- The explosion of digital data in recent years, coupled with advancements in data collection and storage technologies, has provided the necessary fuel for training complex neural networks.

Computation Power:

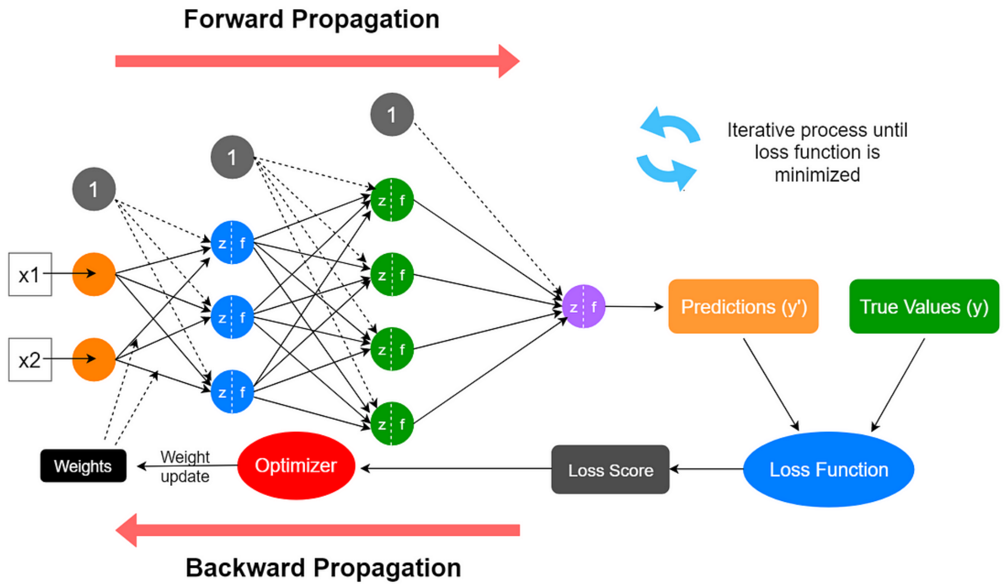
- The availability of powerful hardware, including GPUs (Graphics Processing Units) and specialized AI chips, has enabled researchers and practitioners to train larger and more sophisticated models more efficiently.

Algorithms:

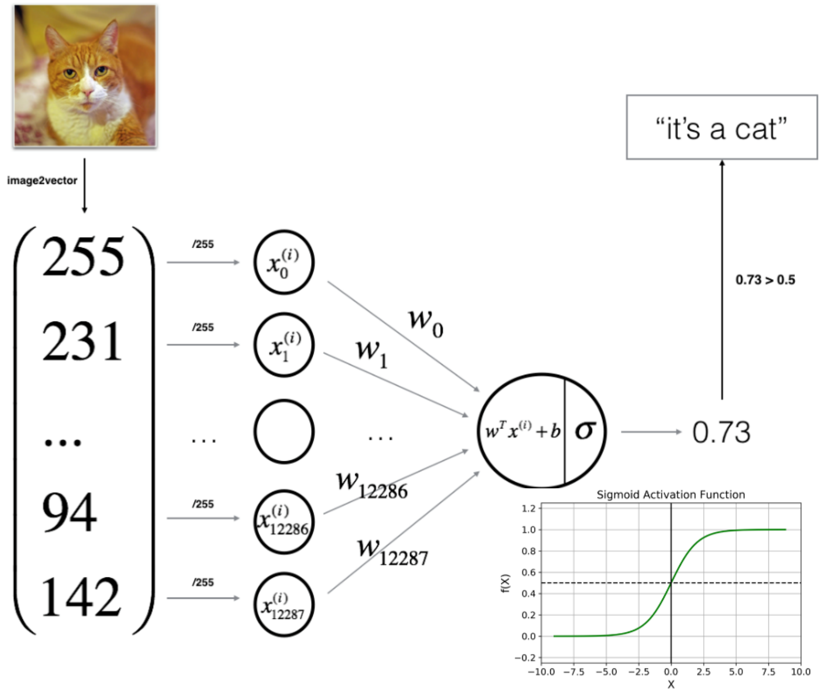
- Deep learning algorithms, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), have undergone significant refinement and optimization. Researchers have developed novel architectures, optimization techniques, and regularization methods to improve model performance and scalability.

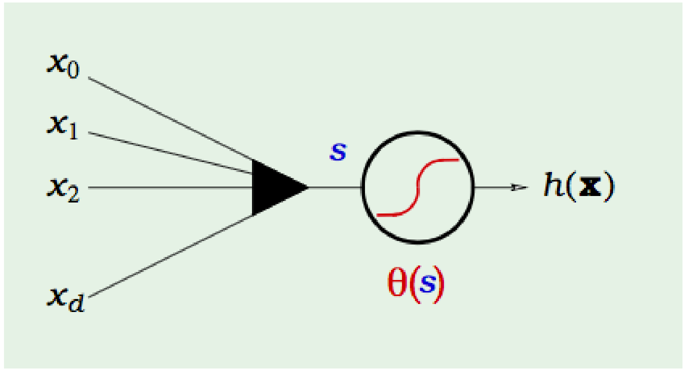
Last time: Anatomy of a deep neural network

- Input data and targets: one or more tensors
- Layers
- Activation Funct
- Loss Function
- Optimizer

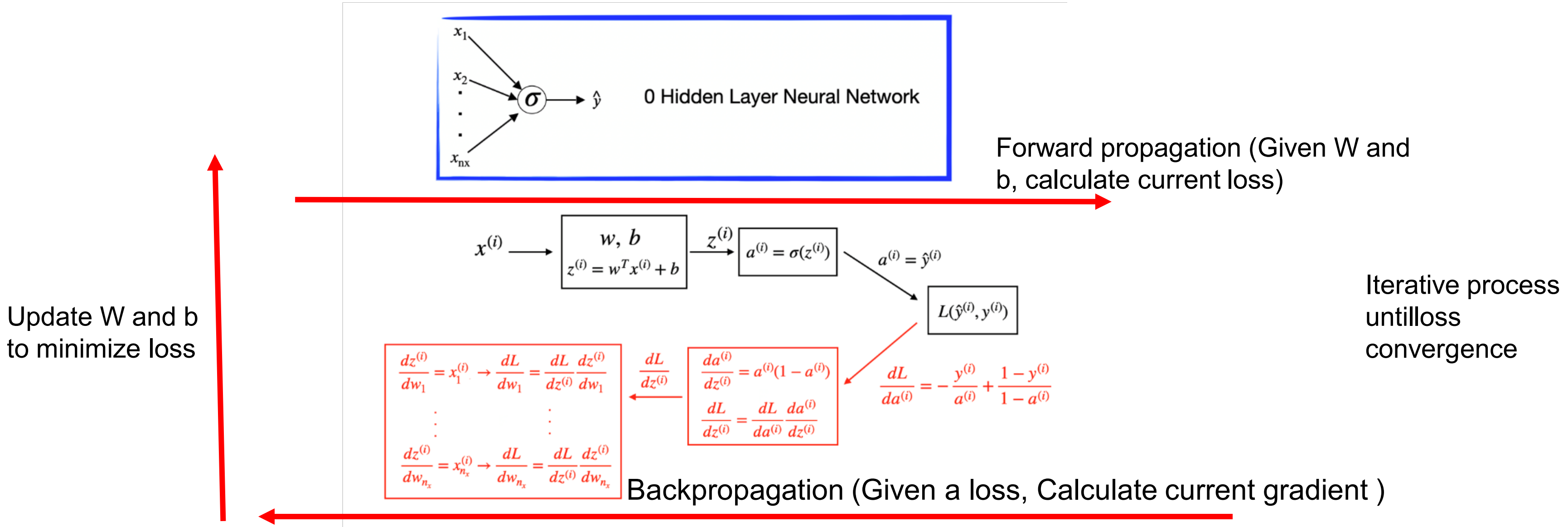


Last time: Single neuron as a linear classifier (Logistic Regression)





Last time: Learning parameters: Gradient Descent


x_1
x_2
x_{nx}
σ
$\hat{y}$
0 Hidden Layer Neural Network
Forward propagation (Given W and b, calculate current loss)
$x^{(i)}$
w, b
$z^{(i)} = w^T x^{(i)} + b$
$z^{(i)}$
$a^{(i)} = \sigma(z^{(i)})$
$a^{(i)} = \hat{y}^{(i)}$
$L(\hat{y}^{(i)}, y^{(i)})$
Update W and b to minimize loss
Iterative process untilloss convergence
$\frac{dL}{da^{(i)}} = -\frac{y^{(i)}}{a^{(i)}} + \frac{1-y^{(i)}}{1-a^{(i)}}$
$\frac{da^{(i)}}{dz^{(i)}} = a^{(i)}(1-a^{(i)})$
$\frac{dL}{dz^{(i)}} = \frac{dL}{da^{(i)}} \frac{da^{(i)}}{dz^{(i)}}$
$\frac{dL}{dz^{(i)}}$
$\frac{dz^{(i)}}{dw_1} = x_1^{(i)} \rightarrow \frac{dL}{dw_1} = \frac{dL}{dz^{(i)}} \frac{dz^{(i)}}{dw_1}$
$\frac{dz^{(i)}}{dw_{n_x}} = x_{n_x}^{(i)} \rightarrow \frac{dL}{dw_{n_x}} = \frac{dL}{dz^{(i)}} \frac{dz^{(i)}}{dw_{n_x}}$
Backpropagation (Given a loss, Calculate current gradient)

Step 1: Step 1: Initialize the model's parameters

Step 2: Loop:

- Calculate current loss (forward propagation)
- Calculate current gradient (backpropagation)
- Update parameters (gradient descent)

Agenda for Today

Lecture: Multilayer Perceptron (MLP)

- Two-layer NN:
  - **Vectorized Notation**
  - **Forward propagation**
  - **Backpropagation**

After-class assignment

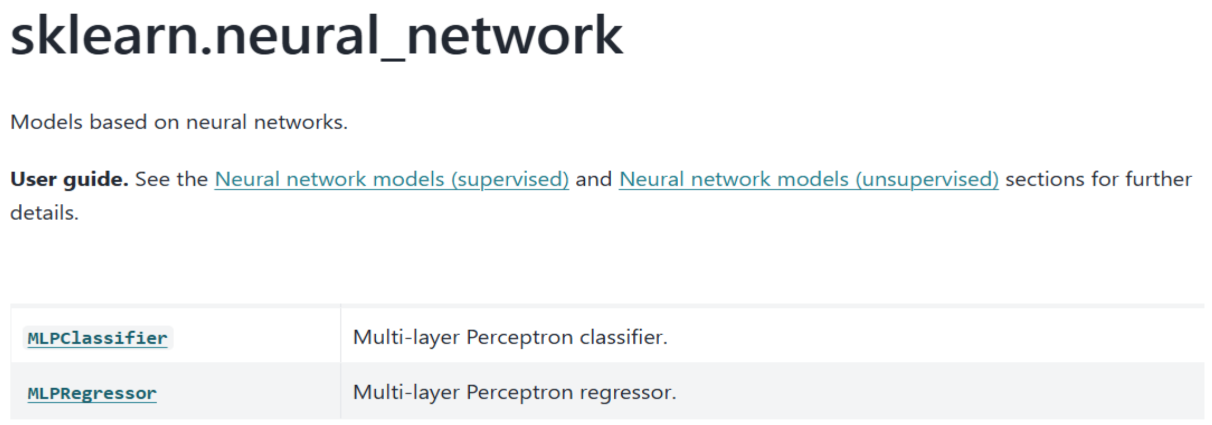
- Implement your two-layer neural network in raw NumPy.

What is Multilayer Perceptron

MLP

MLP:

The **MLP is the earliest practical and general-purpose neural network architecture**, and all modern architectures (CNNs, RNNs, Transformers) are built upon its core principles.



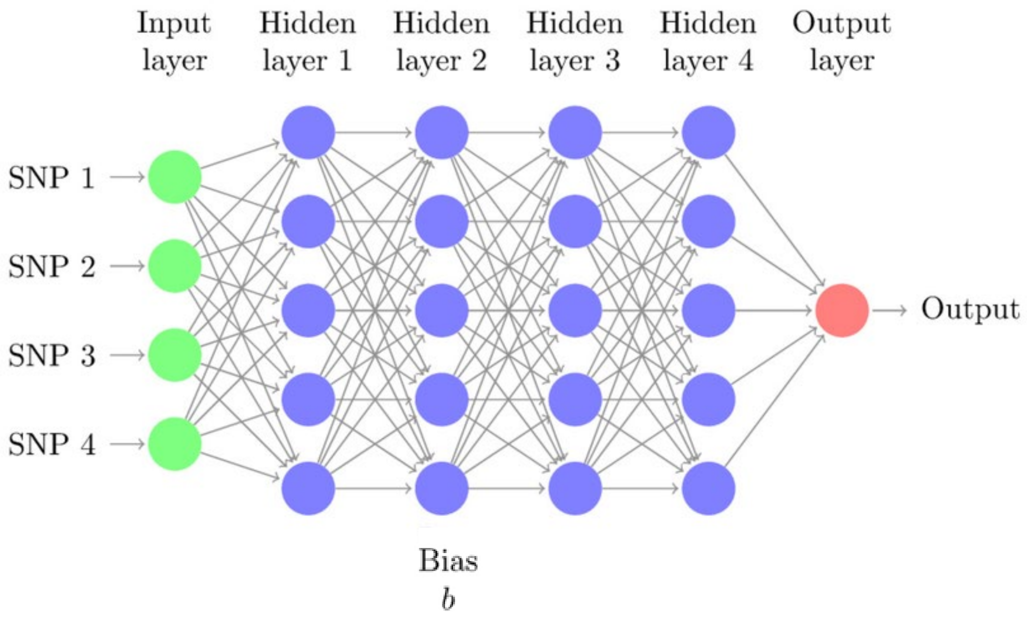
sklearn.neural_network

Models based on neural networks.

User guide. See the Neural network models (supervised) and Neural network models (unsupervised) sections for further details.

`MLPClassifier`	Multi-layer Perceptron classifier.
`MLPRegressor`	Multi-layer Perceptron regressor.

MLP in pytorch



5-Layer Perceptron (MLP)

Feedforward neural network:

- An **input layer**
- One or more **hidden layers**
- An **output layer**
- Each layer is **fully connected** to the next (dense layers)

Each single neuron's computation in MLP

It is split into two parts, which are often denoted as z and a

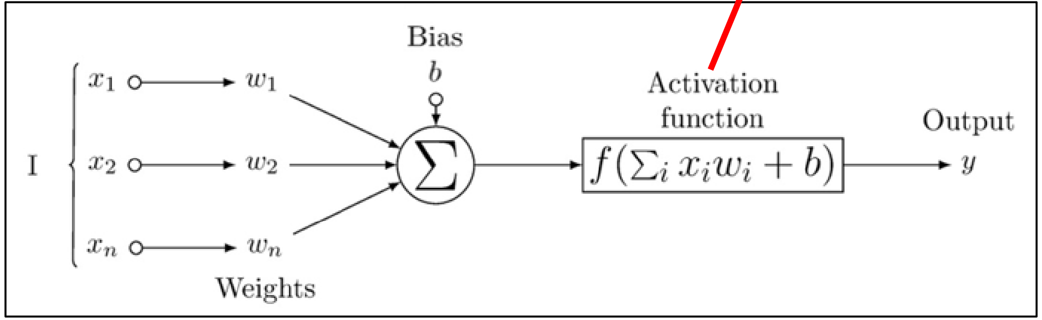
The activation function transforms the output of the linear component; it does not introduce any new parameters.

z: weighted sum

$$z = \sum_i w_i x_i + b$$

a : activation function

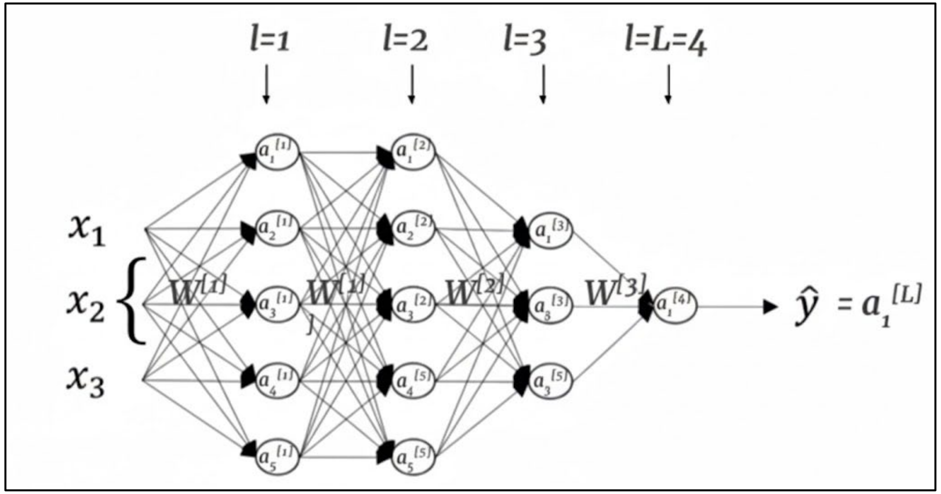
$$a = f(z)$$



$$z = \vec{w}X + b$$

$$\hat{y} = a = \sigma(z)$$ (logistic regression)

Notation in Neural Networks



Superscript $[l]$: Layer index

- Indicates the **layer number**.
- Example:

  $W^{[l]}, b^{[l]}$: weights and bias of the l^{th} layer

Subscript i: Neuron index

- Indicates the **neuron position** within a layer.
- Example:

$$z_i^{[l]} = W_i^{[l]} a^{[l-1]} + b_i^{[l]}$$

$$a_i^{[l]} = f(z_i^{[l]})$$

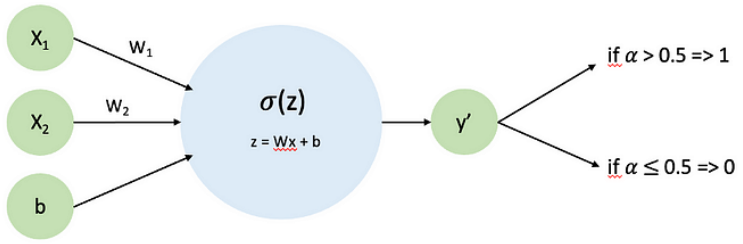
where $z_i^{[l]}$ is the **linear combination (pre-activation)** and $a_i^{[l]}$ is the **activation** of the i^{th} neuron in the l^{th} layer.

Two-layer MLP

One hidden layer

Binary classification

Single Neuron Model

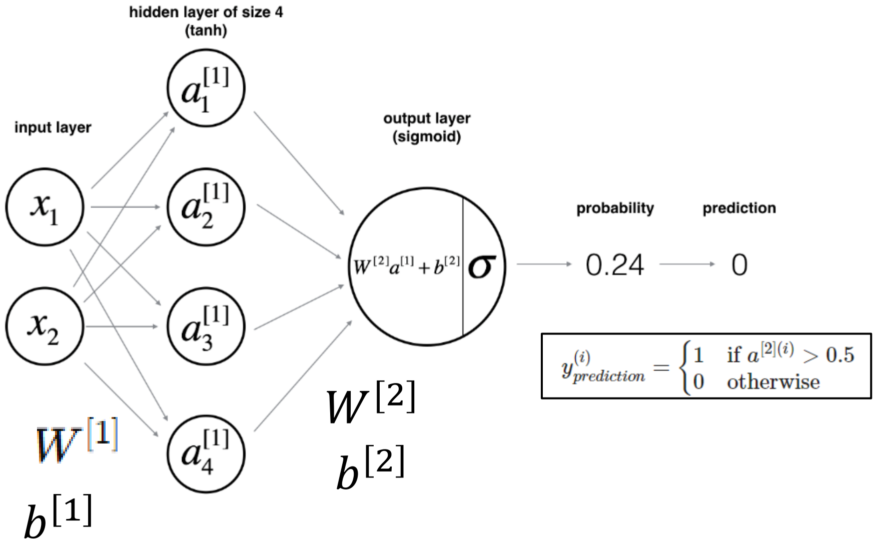


$$z = x_1 w_1 + x_2 w_2 + b = WX + b$$

$$\hat{y} = a = \sigma(z)$$

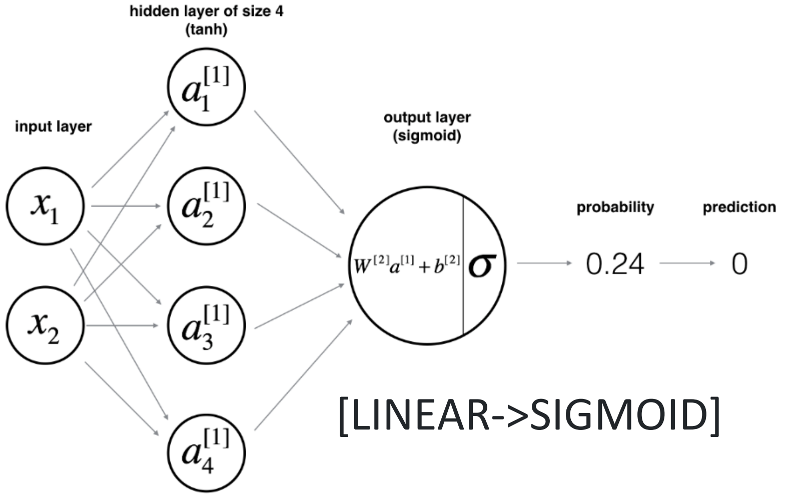
$$\mathcal{L}(a, y) = -(y\log(a) + (1-y)\log(1-a))$$

Two-layer MLP



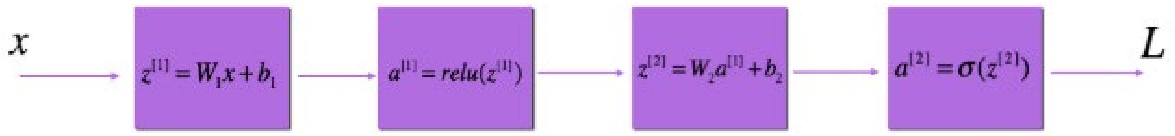
$$y^{(i)}_{prediction} = \begin{cases} 1 & \text{if } a^{[2](i)} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Computational graph in two-layer MLP

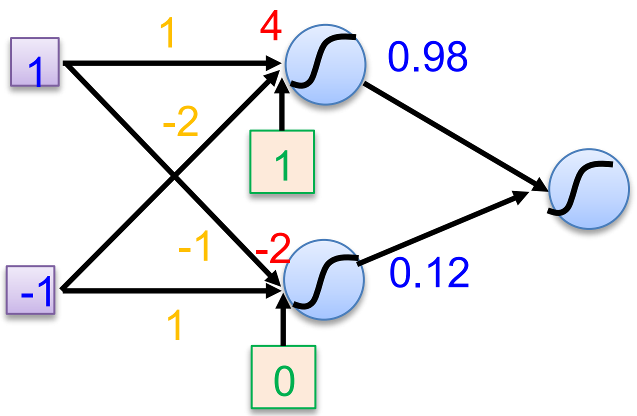


[LINEAR->SIGMOID]

[LINEAR->ACTIVATION]



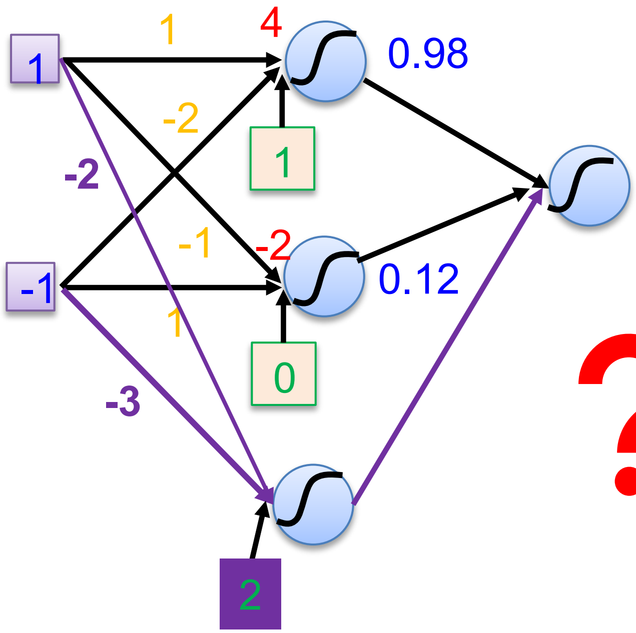
Matrix Operation: [LINEAR->ACTIVATION (sigmoid)]



$$\sigma(\overbrace{\boxed{W}\ \boxed{x} + \boxed{b}}^{z}) = \boxed{a}$$

$$\sigma(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

Matrix Operation: [LINEAR->ACTIVATION (sigmoid)]

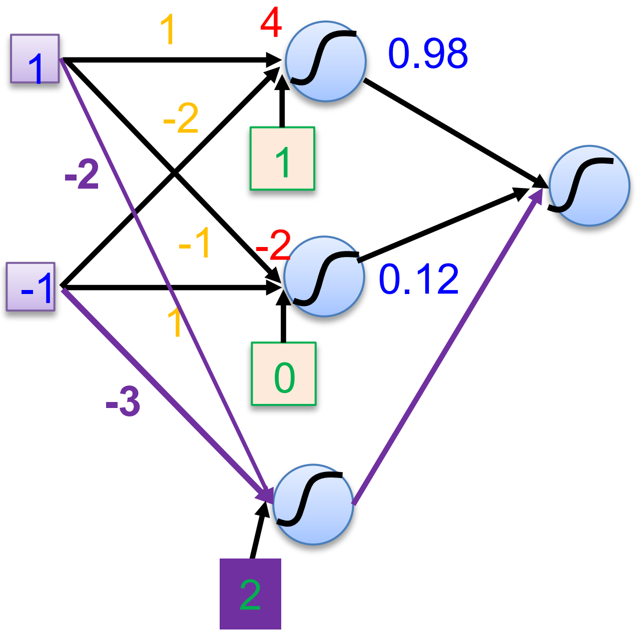


$$\sigma(\boxed{W}\ \boxed{x} + \boxed{b}) = \boxed{a}$$

$$\sigma(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \end{bmatrix}} + \begin{bmatrix} 1 \\ 0 \end{bmatrix}) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

?

Matrix Operation



$$\sigma(\boxed{W}\ \boxed{x} + \boxed{b}) = \boxed{a}$$

$$\sigma(\underbrace{\begin{bmatrix} 1 & -2 \\ -1 & 1 \\ -2 & -3 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 2 \end{bmatrix}}_{\begin{bmatrix} 4 \\ -2 \\ 3 \end{bmatrix}}) = \begin{bmatrix} 0.98 \\ 0.12 \\ 0.95 \end{bmatrix}$$

- What is the shape of $W^{[1]}$?

General notation

The **weight matrix** for layer l is denoted as:

$$W^{[l]}$$

If the layer l has $n^{[l]}$ neurons and the previous layer $l-1$ has $n^{[l-1]}$ neurons, then:

$$W^{[l]} \in \mathbb{R}^{n^{[l]} \times n^{[l-1]}}$$

That is, it's a matrix where:

- **Rows** correspond to **neurons in the current layer (l)**
- **Columns** correspond to **neurons in the previous layer (l–1)**

Expanding the weight matrix: Element notation

An individual entry is written as:

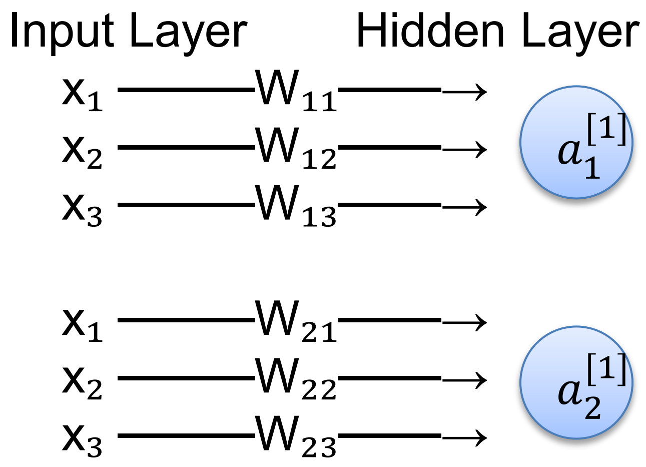
$$W^{[1]} = \begin{bmatrix} 1 & -2 \\ -1 & 1 \\ -2 & -3 \end{bmatrix}$$

$$W_{ij}^{[l]}$$

where:

- i: index of the **neuron in the current layer** l
- j: index of the **neuron in the previous layer** $l-1$

Example: 3 inputs → 2 hidden neurons



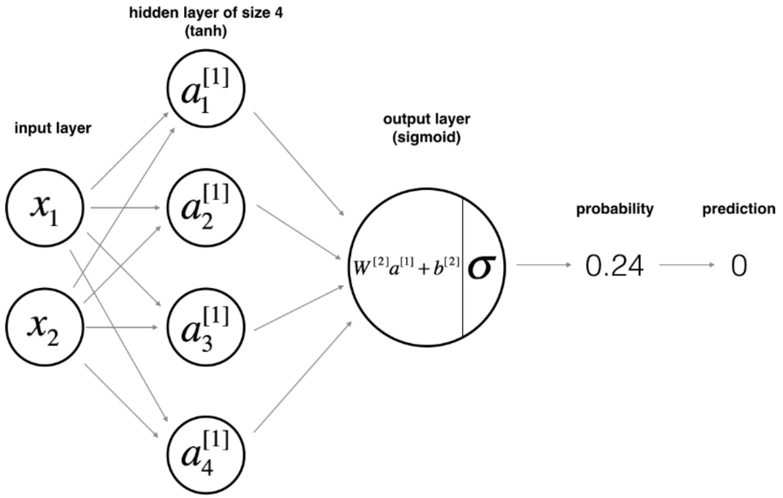
$$W^{[1]} = \begin{bmatrix} W_{11}^{[1]} & W_{12}^{[1]} & W_{13}^{[1]} \\ W_{21}^{[1]} & W_{22}^{[1]} & W_{23}^{[1]} \end{bmatrix} \in \mathbb{R}^{2\times 3}$$

✅ **Explanation:**

- $W^{[1]}$: weight matrix for **layer 1**
- $W_{ij}^{[1]}$: weight connecting the **j^th neuron of the previous layer** to the **i^th neuron of the current layer**
- Shape 2×3: 2 neurons in layer 1, 3 neurons in the previous layer

Matrix Format: 2 inputs → 4 hidden neurons

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$



$$Z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$$A^{[1]} = f(Z^{[1]})$$

$$A^{[1]} = f(Z^{[1]}) = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

Binary classification: Matrix Notation (single instance)

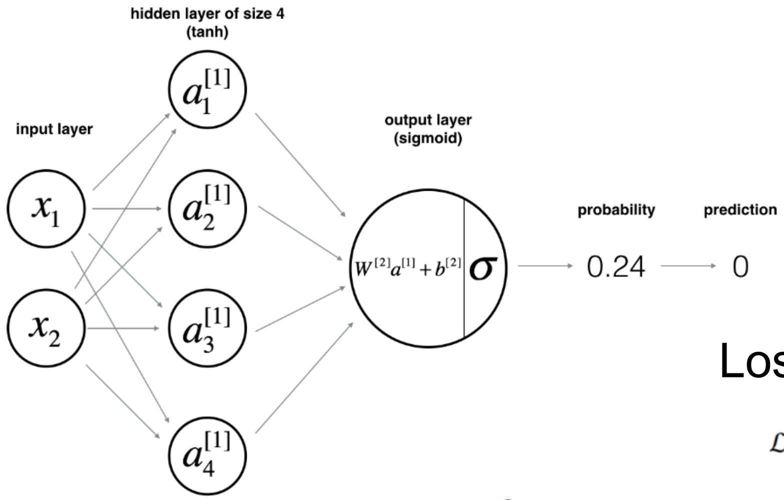
Superscript (i) denotes the *i*th training example (instance)

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1](i)}$$
$$a^{[1](i)} = f\left(z^{[1](i)}\right)$$

$$\rightarrow$$

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

Forward propagation



Prediction rule:

$$y^{(i)}_{prediction} = \begin{cases} 1 & \text{if } a^{[2](i)} > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Loss function:

$$\mathcal{L}(a, y) = -\frac{1}{m}\sum_{i=0}^{m}\left(y^{(i)}\log\left(a^{[2](i)}\right) + (1 - y^{(i)})\log\left(1 - a^{[2](i)}\right)\right)$$

What if we have m observations?

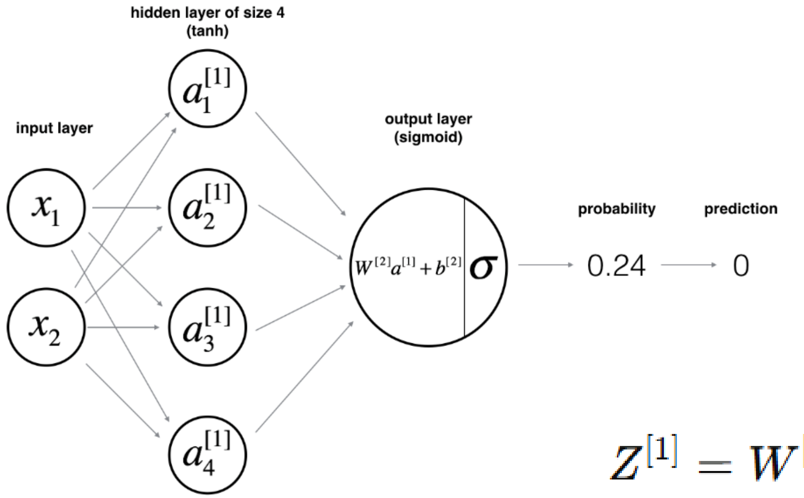
Using m observations

Deep learning frameworks (and Andrew Ng's notation) use **column-wise stacking** for convenience in vectorized propagation:

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times m}$$

- Each **column** represents one training example.
- Each **row** represents one feature.
- This makes the math cleaner for **matrix multiplications** across layers.

Using multiple (4) observations



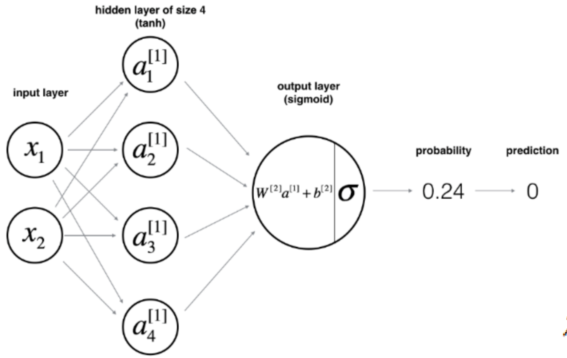
$$W^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix}$$

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$Z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} & b_1^{[1]} & b_1^{[1]} & b_1^{[1]} \\ b_2^{[1]} & b_2^{[1]} & b_2^{[1]} & b_2^{[1]} \\ b_3^{[1]} & b_3^{[1]} & b_3^{[1]} & b_3^{[1]} \\ b_4^{[1]} & b_4^{[1]} & b_4^{[1]} & b_4^{[1]} \end{bmatrix}$$

Expanded form for 4 neurons and 4 observations

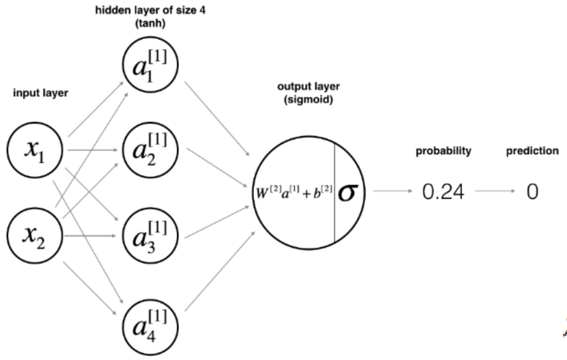


$$Z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} & b_1^{[1]} & b_1^{[1]} & b_1^{[1]} \\ b_2^{[1]} & b_2^{[1]} & b_2^{[1]} & b_2^{[1]} \\ b_3^{[1]} & b_3^{[1]} & b_3^{[1]} & b_3^{[1]} \\ b_4^{[1]} & b_4^{[1]} & b_4^{[1]} & b_4^{[1]} \end{bmatrix}$$

$$z_i^{[1](j)} = w_{i1}^{[1]} x_1^{(j)} + w_{i2}^{[1]} x_2^{(j)} + b_i^{[1]}$$

Expand $Z^{[1]} =$
```
[   ,   ,   ,   ]
[   ,   ,   ,   ]
[   ,   ,   ,   ]
[   ,   ,   ,   ]
```
?

Expanded form for 4 neurons and 4 observations



$$Z^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \\ w_{41}^{[1]} & w_{42}^{[1]} \end{bmatrix} \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & x_1^{(3)} & x_1^{(4)} \\ x_2^{(1)} & x_2^{(2)} & x_2^{(3)} & x_2^{(4)} \end{bmatrix} + \begin{bmatrix} b_1^{[1]} & b_1^{[1]} & b_1^{[1]} & b_1^{[1]} \\ b_2^{[1]} & b_2^{[1]} & b_2^{[1]} & b_2^{[1]} \\ b_3^{[1]} & b_3^{[1]} & b_3^{[1]} & b_3^{[1]} \\ b_4^{[1]} & b_4^{[1]} & b_4^{[1]} & b_4^{[1]} \end{bmatrix}$$

$$z_i^{[1](j)} = w_{i1}^{[1]} x_1^{(j)} + w_{i2}^{[1]} x_2^{(j)} + b_i^{[1]}$$

- **Each column** → corresponds to **one training example** *j*.
- **Each row** → corresponds to **one neuron (unit) in the current layer**.

Forward propagation becomes elegant:

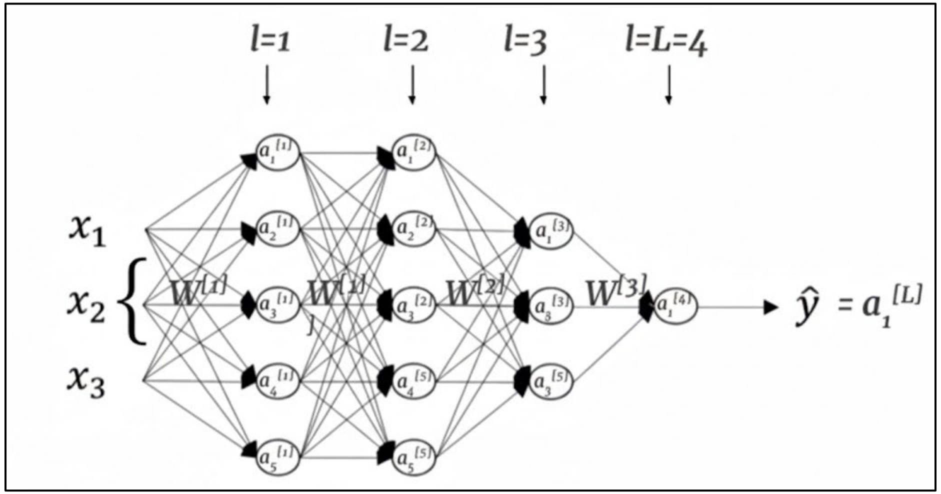
$$A^{[l]} = f(Z^{[l]})$$

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$

where:

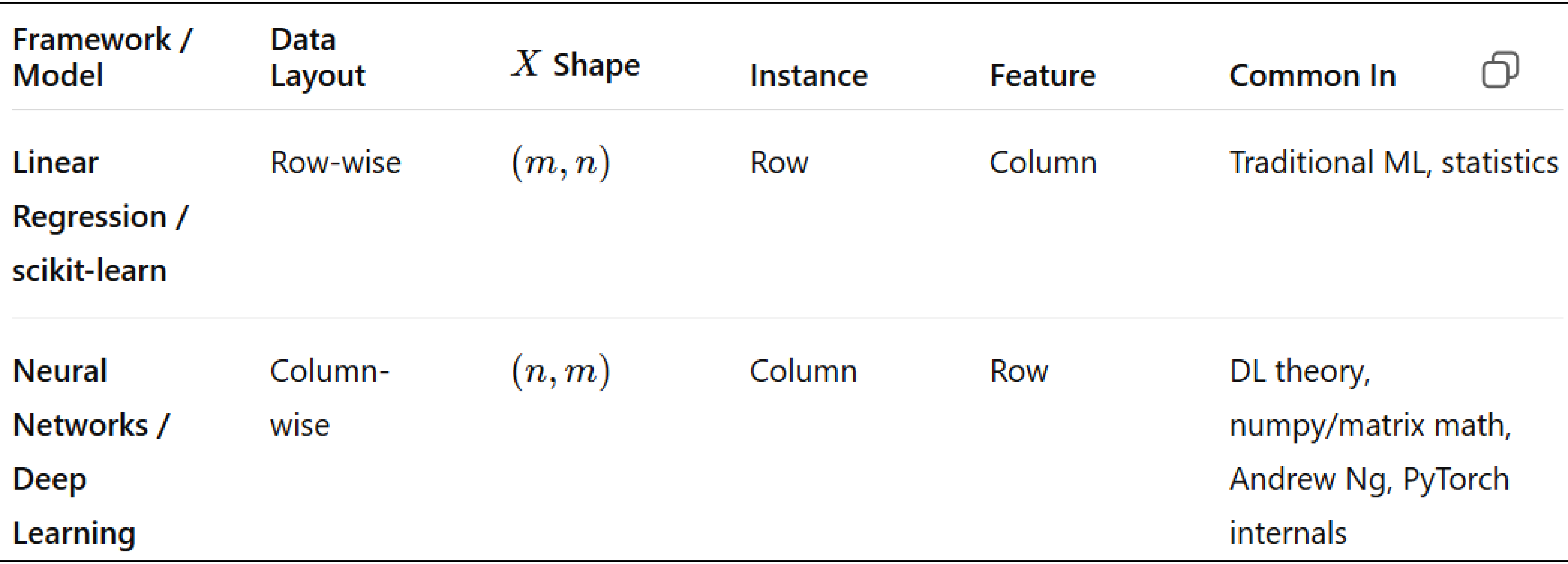
- $W^{[l]} \in \mathbb{R}^{n_l \times n_{l-1}}$
- $A^{[l-1]} \in \mathbb{R}^{n_{l-1} \times m}$
- $b^{[l]} \in \mathbb{R}^{n_l \times 1}$

This yields: $Z^{[l]} \in \mathbb{R}^{n_l \times m}$



➡️ So **each column corresponds to one example** moving through all layers.
➡️ **Each row** corresponds to **one neuron in the current layer**.
➡️ The broadcasting of $b^{[l]}$ across columns (examples) works naturally.

Summary Comparison



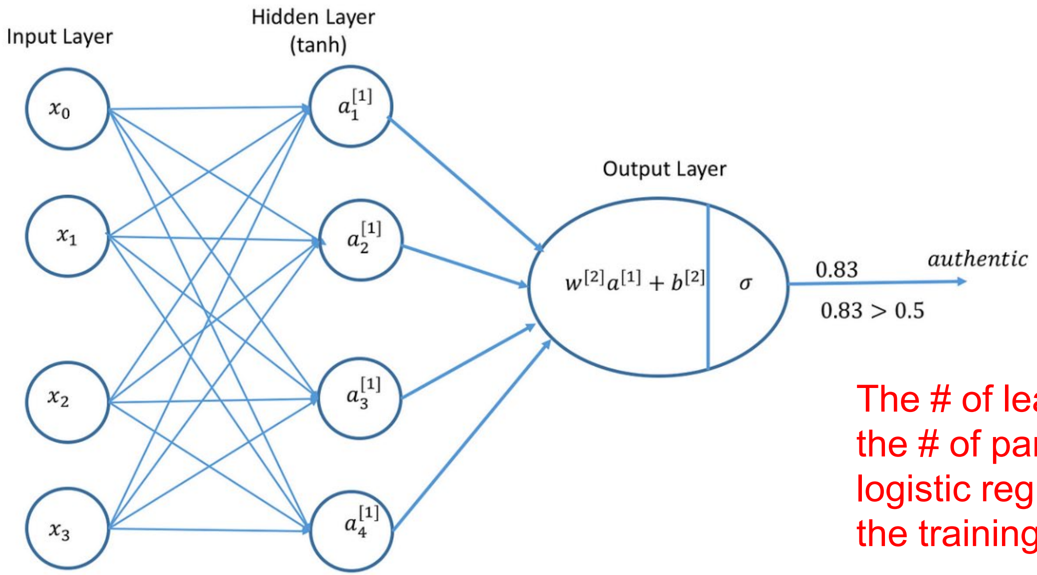
Framework / Model	Data Layout	X Shape	Instance	Feature	Common In
Linear Regression / scikit-learn	Row-wise	(m, n)	Row	Column	Traditional ML, statistics
Neural Networks / Deep Learning	Column-wise	(n, m)	Column	Row	DL theory, numpy/matrix math, Andrew Ng, PyTorch internals

Trainable parameters in NNs

The network ***parameters*** θ include the weight matrices and bias vectors from all layers

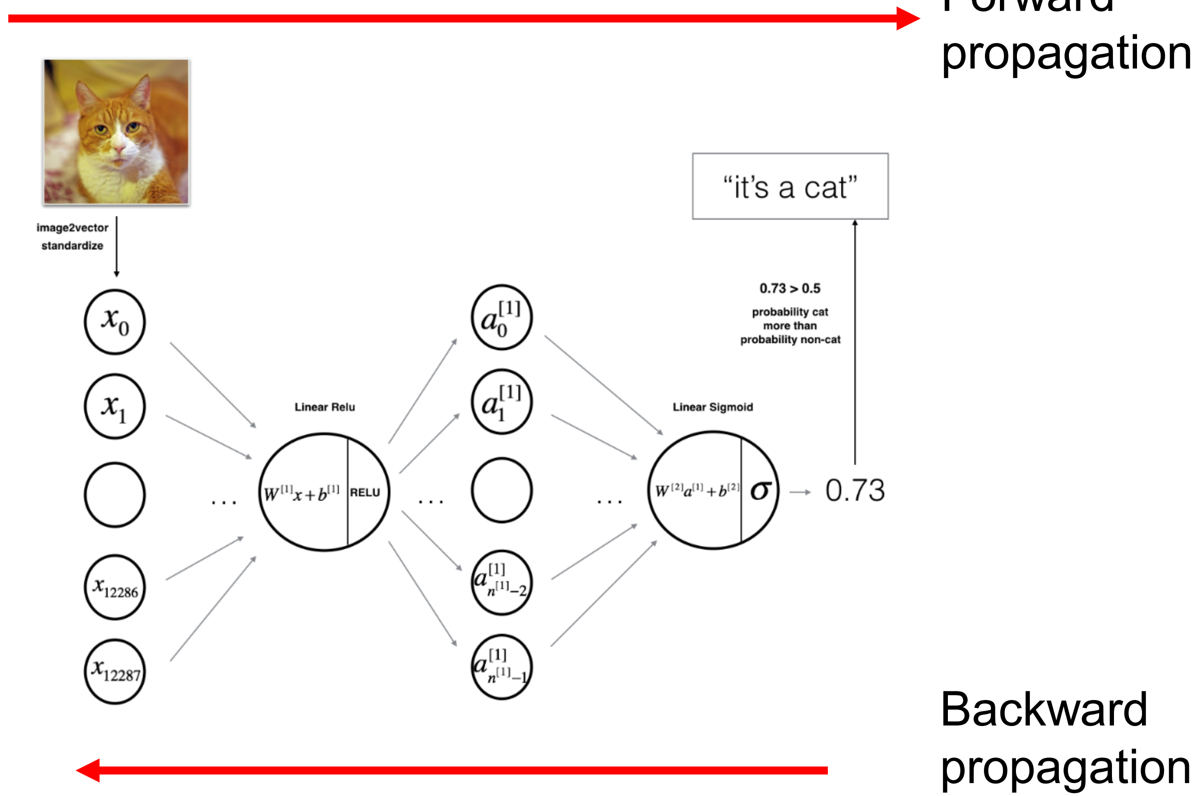
$$\theta = \{W^1, b^1, W^2, b^2\}$$

Training a model to learn a set of parameters θ that are optimal (according to a criterion) is one of the greatest challenges in DL



The # of learnable parameters is far more than the # of parameters in linear regression and logistic regression. It is prone to overfit when the training set is small

Training the two-layer neural nets



Gradient Descent: update parameters

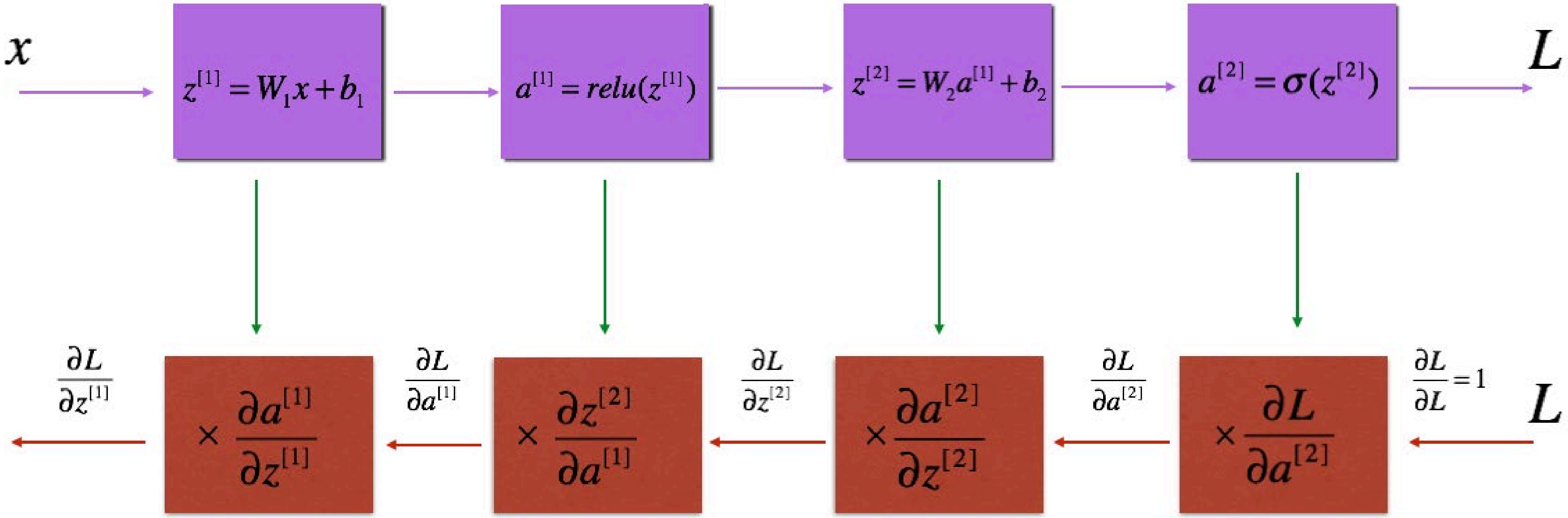
$$W^{[l]} = W^{[l]} - \alpha\, dW^{[l]}$$

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

Forward propagation and backward propagation

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \quad \text{where } A^{[0]} = X$$

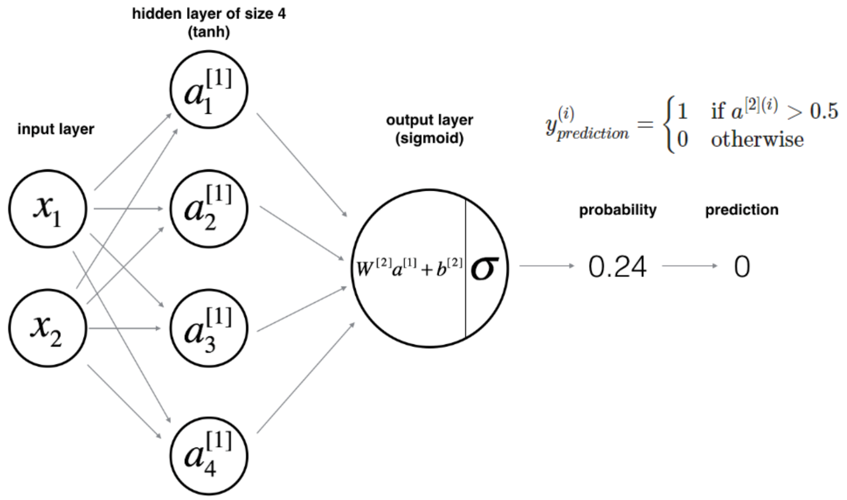
$$A^{[l]} = g\left(Z^{[l]}\right)$$



x
$z^{[1]} = W_1 x + b_1$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W_2 a^{[1]} + b_2$
$a^{[2]} = \sigma(z^{[2]})$
L
$\frac{\partial L}{\partial z^{[1]}}$
$\times \frac{\partial a^{[1]}}{\partial z^{[1]}}$
$\frac{\partial L}{\partial a^{[1]}}$
$\times \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial L}{\partial z^{[2]}}$
$\times \frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial L}{\partial a^{[2]}}$
$\times \frac{\partial L}{\partial a^{[2]}}$
$\frac{\partial L}{\partial L} = 1$
L

$$\frac{d\mathcal{L}(a^{[2]}, y)}{dz^{[1]}} = \frac{d\mathcal{L}(a^{[2]}, y)}{da^{[2]}} \frac{da^{[2]}}{dz^{[2]}} \frac{dz^{[2]}}{da^{[1]}} \frac{da^{[1]}}{dz^{[1]}}$$

Shape of the Parameter matrix:

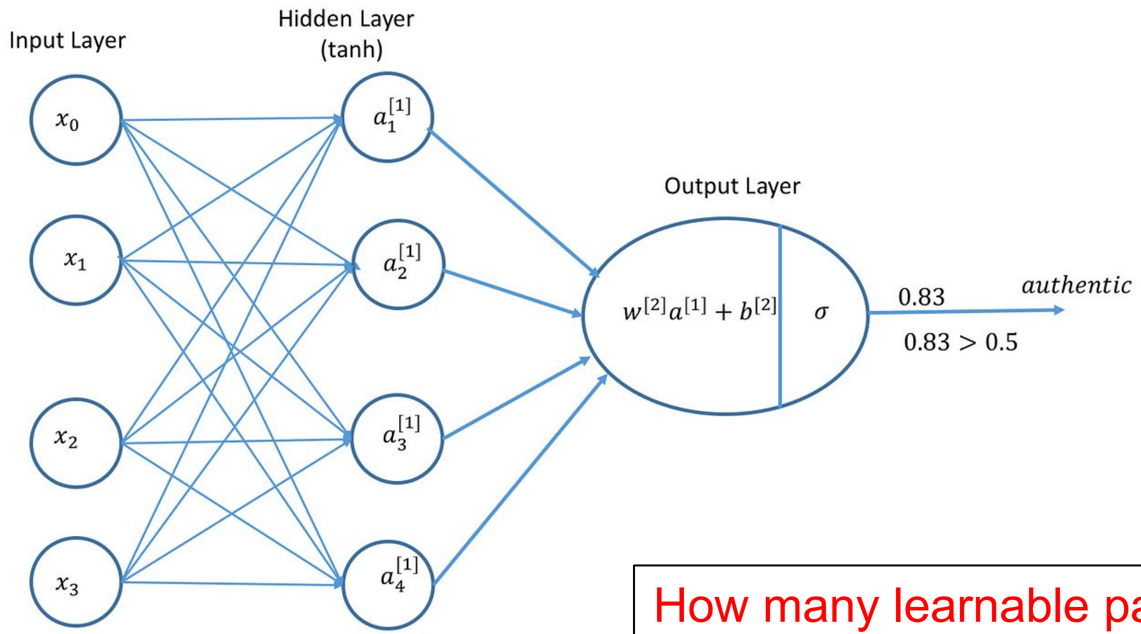


$$A^{[l]} = g(Z^{[l]}) = g(W^{[l]}A^{[l-1]} + b^{[l]})$$

Layer	Parameter	Shape
Layer 1	Weight matrix $W^{(1)}$	$(4, 2)$
	Bias vector $b^{(1)}$	(4)
Layer 2	Weight matrix $W^{(2)}$	$(1, 4)$
	Bias vector $b^{(2)}$	(1)

1. **Weight matrix W**: The shape is determined by the number of neurons in the current layer and the number of neurons in the previous layer.
2. **Bias vector b**: The shape is determined by the number of neurons in the current layer.

Exercise:



What are the shapes of weights and biases?

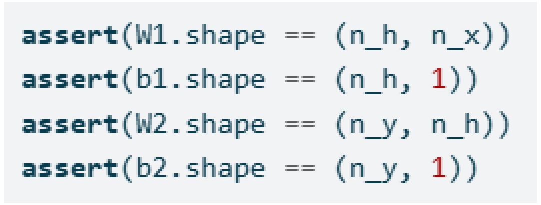
- Shape of weights
- Shape of biases

How many learnable parameters?

- How many weights?
- How many biases?

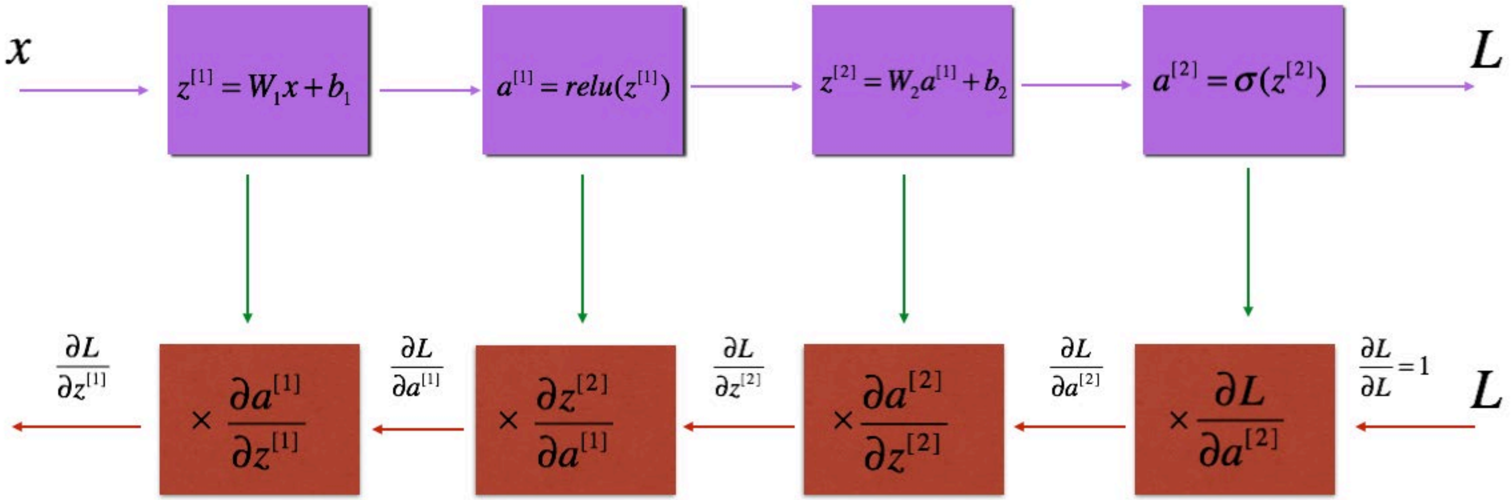
the **assert** statements in the implementation

- Used to verify that the shapes of the weight matrices and bias vectors match the expected dimensions
- This is crucial for ensuring that the matrix multiplications and subsequent operations align properly.



```
assert(W1.shape == (n_h, n_x))
assert(b1.shape == (n_h, 1))
assert(W2.shape == (n_y, n_h))
assert(b2.shape == (n_y, 1))
```

Build your two-layer neural nets (Quiz7)


x
$z^{[1]} = W_1 x + b_1$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W_2 a^{[1]} + b_2$
$a^{[2]} = \sigma(z^{[2]})$
L
$\frac{\partial L}{\partial z^{[1]}}$
$\times \frac{\partial a^{[1]}}{\partial z^{[1]}}$
$\frac{\partial L}{\partial a^{[1]}}$
$\times \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial L}{\partial z^{[2]}}$
$\times \frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial L}{\partial a^{[2]}}$
$\times \frac{\partial L}{\partial a^{[2]}}$
$\frac{\partial L}{\partial L} = 1$
L

Forward Propagation

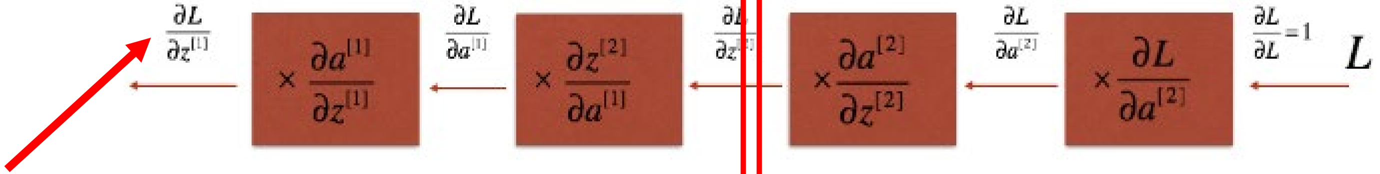
$$\begin{tikzcd} x \arrow[r] & \boxed{z^{[1]} = W_1 x + b_1} \arrow[r] & \boxed{a^{[1]} = relu(z^{[1]})} \arrow[r] & \boxed{z^{[2]} = W_2 a^{[1]} + b_2} \arrow[r] & \boxed{a^{[2]} = \sigma(z^{[2]})} \arrow[r] & L \end{tikzcd}$$

$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]} \qquad \text{where } A^{[0]} = X.$$

$A^{[l]} = g(Z^{[l]}) = g(W^{[l]} A^{[l-1]} + b^{[l]})$ where the activation "g" can be sigmoid() or relu().

sigmoid and *relu* function are provided

Back Propagation


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

$$\frac{dL(a^{[2]}, y)}{dz^{[1]}} = \frac{dL(a^{[2]}, y)}{da^{[2]}} \cdot \frac{da^{[2]}}{dz^{[2]}} \cdot \frac{dz^{[2]}}{da^{[1]}} \cdot \frac{da^{[1]}}{dz^{[1]}}$$

- Layer 1:
  - $\frac{\partial L}{\partial W^{[1]}}$
  - $\frac{\partial L}{\partial b^{[1]}}$

- Layer 2:
  - $\frac{\partial L}{\partial W^{[2]}}$ $\quad \frac{\partial L}{\partial W^{[2]}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}} \cdot (a^{[1]})^T$
  - $\frac{\partial L}{\partial b^{[2]}}$ $\quad \frac{\partial L}{\partial b^{[2]}} = \frac{\partial L}{\partial a^{[2]}} \cdot \frac{\partial a^{[2]}}{\partial z^{[2]}}$

Cache the intermediate output !

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

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

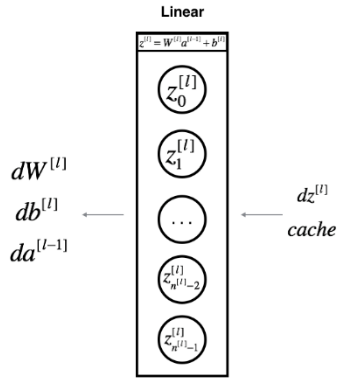
$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

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

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

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

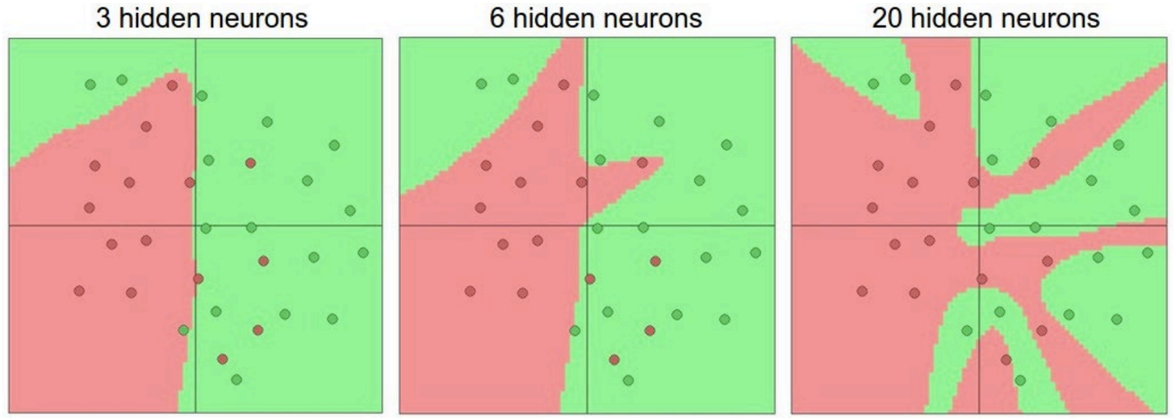
Vectorized Gradients are Provided in the Quiz



For every forward function, there is a corresponding backward function. That is why at every step of your forward module you will be storing some values in a cache. The cached values are useful for computing gradients. In the backpropagation module you will then use the cache to calculate the gradients.

Effect of Hidden Layer Size on Model Complexity

- Neural networks with more neurons (larger hidden layers) can approximate more complex functions.
- More neurons improve representation but may lead to overfitting.



Increasing hidden neurons → more flexible decision boundaries → Increasing variance

Picture from: https://cs231n.github.io/neural-networks-1/

The Universal Approximation Theorem for Neural Nets

A **feedforward neural network** with **one hidden layer** containing a **finite number of neurons** can approximate **any continuous function** on a compact subset of $\mathbb{R}^n$,

if the activation function is:

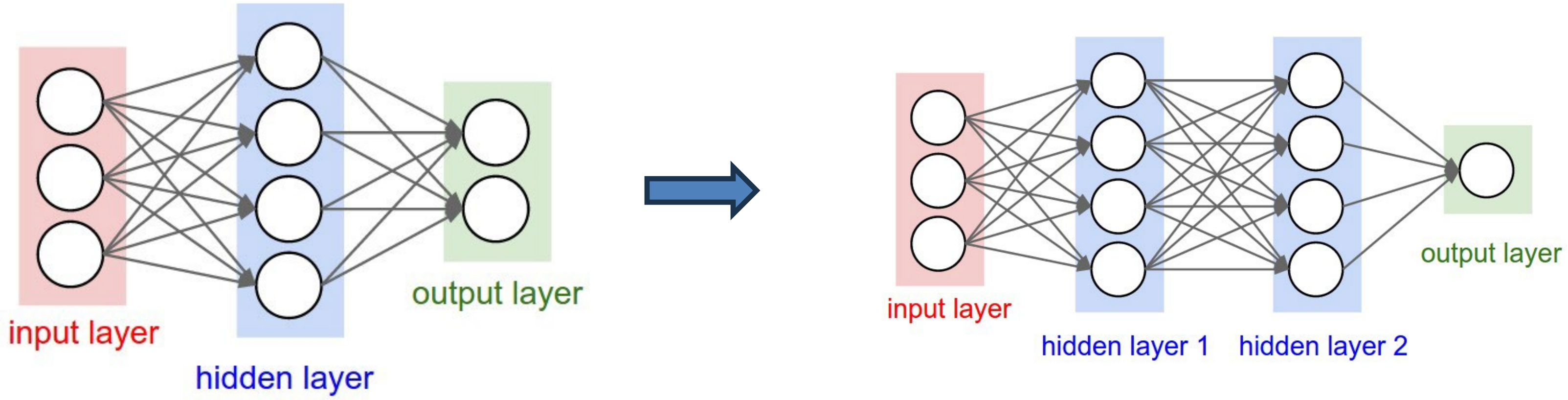
- nonconstant,
- bounded, and
- continuous (e.g., sigmoid, tanh, ReLU, etc.).

Formally:

$$f(x) \approx \sum_{i=1}^{N} a_i \, \sigma(w_i^T x + b_i)$$

for sufficiently large N.

Why Go Deeper?



Later research showed:

- Adding depth (more hidden layers) allows neural networks to approximate the same functions with exponentially fewer neurons.
- Deep networks can capture hierarchical and compositional structures in data more efficiently.

Next Lecture

- L-layer neuron nets
- **Vanishing/exploding gradient problem**
- Activation functions

Learning resource:

- https://d2l.ai/index.html
- https://web.stanford.edu/class/cs224n/