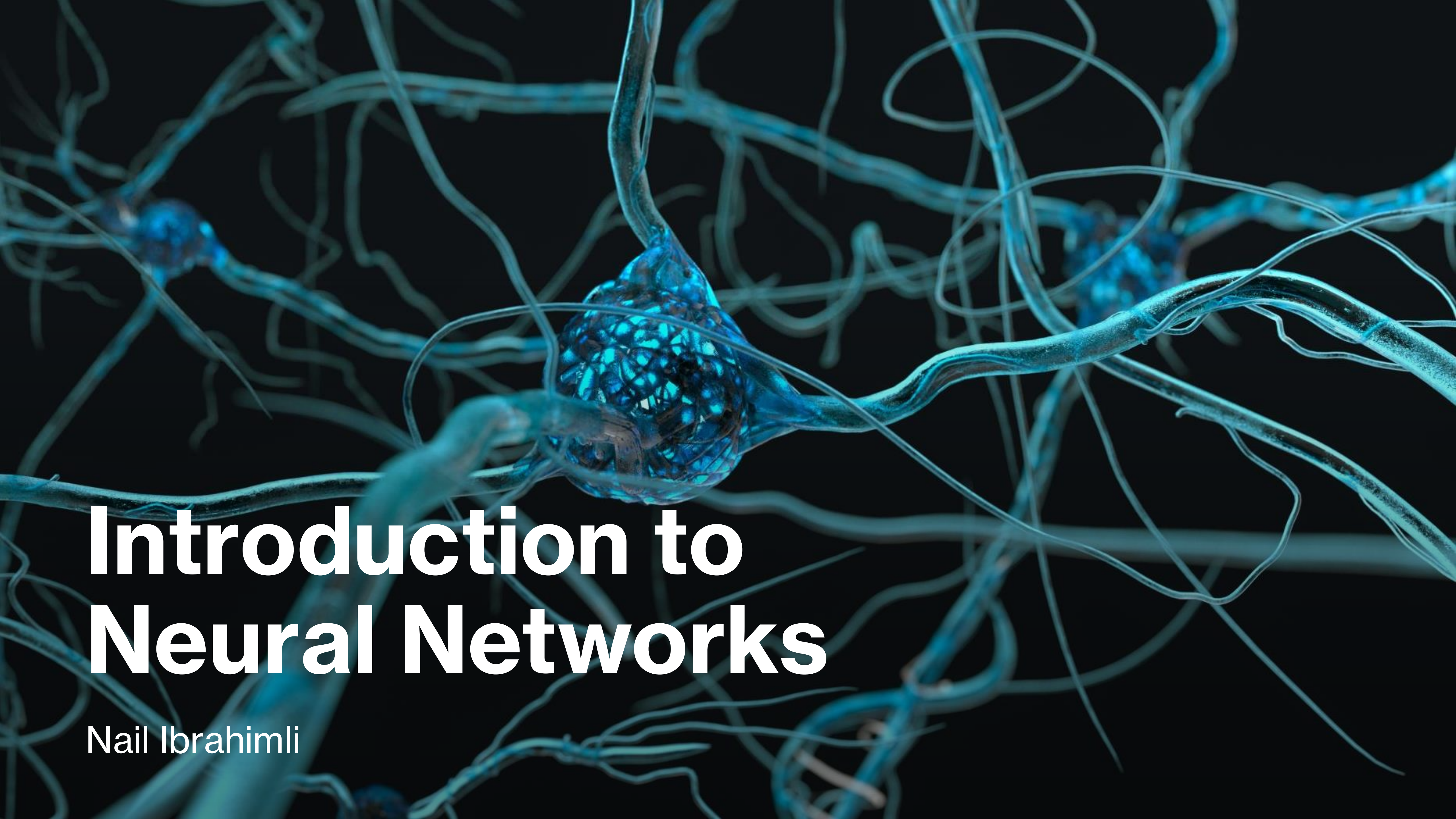
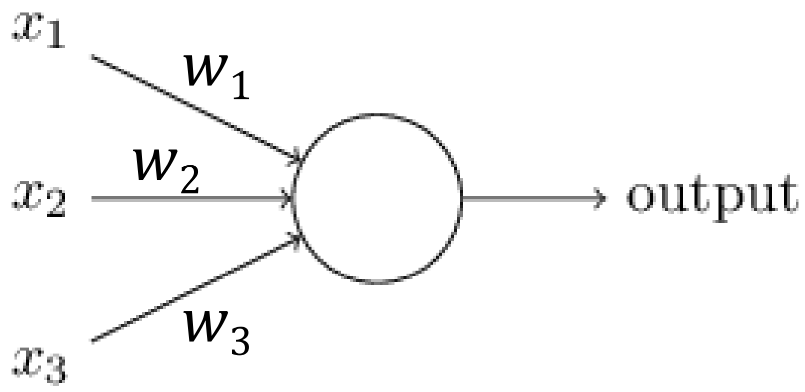


Introduction to Neural Networks

Nail Ibrahimli

Perceptron - a.k.a. single neuron

A perceptron takes multiple inputs (e.g., x_1, x_2, x_3),
computes a **weighted sum**, and produces a **binary output**:

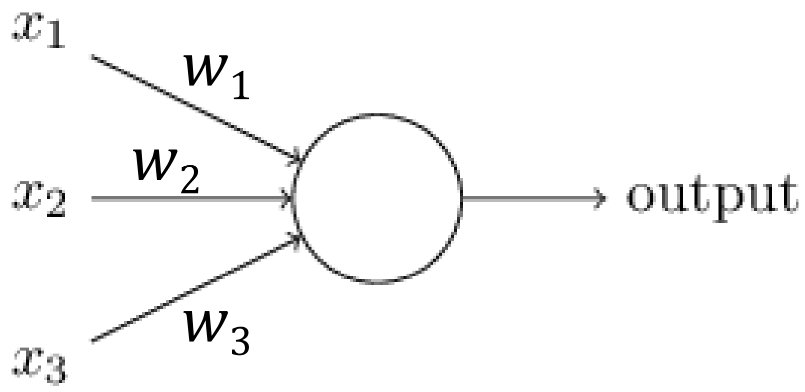


Perceptron - a.k.a. single neuron

A perceptron takes multiple inputs (e.g., $\mathbf{x_1}$, $\mathbf{x_2}$, $\mathbf{x_3}$),
computes a **weighted sum**, and produces a **binary output**:

- Output = 1 if the sum exceeds a **threshold**
- Output = 0 otherwise

$$output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j \le threshold \\ 1 \;\; if \; \sum_j w_j x_j > threshold \end{cases}$$

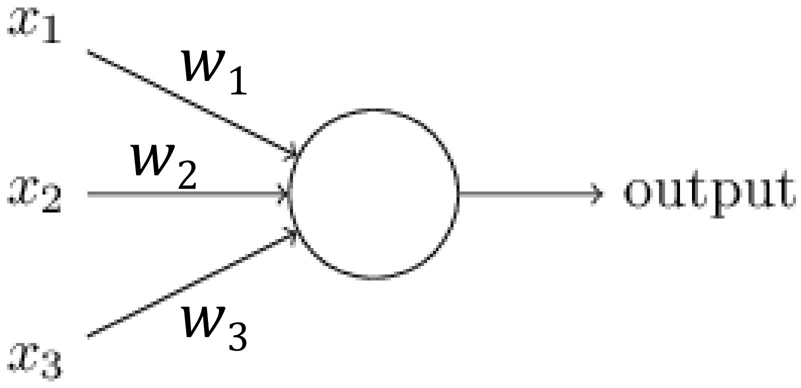


Summary:
The perceptron combines inputs using weights, compares the result to a threshold, and outputs either 0 or 1, a minimal building block of the neural networks.

Perceptron - a.k.a. single neuron

Output: Go to Gouda for the cheese festival on Saturday

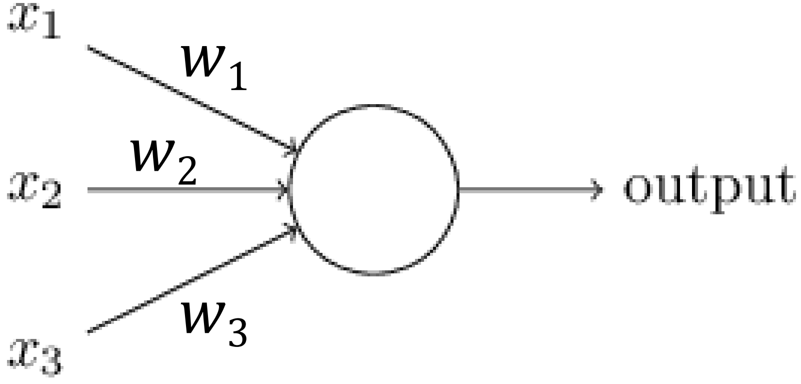
- **Inputs:**
  - $\boldsymbol{x}_1$: Is the weather good?
  - $\boldsymbol{x}_2$: Am I going with a friend?
  - $\boldsymbol{x}_3$: Is the venue easy to commute?



Perceptron - a.k.a. single neuron

Output: Go to Gouda for the cheese festival on Saturday

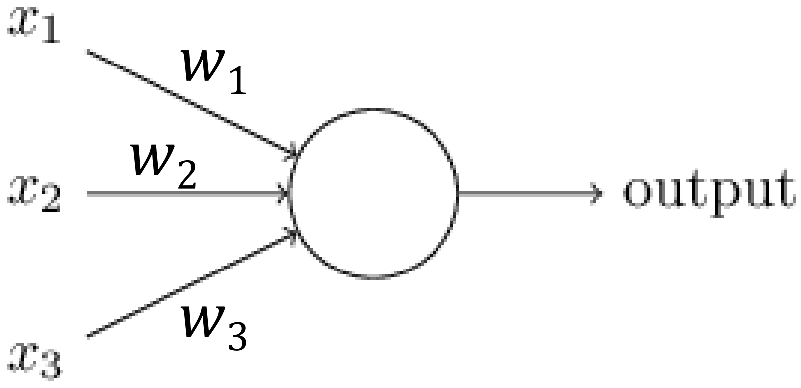
- **Inputs:**
  - $\boldsymbol{x}_1$: Is the weather good?
  - $\boldsymbol{x}_2$: Am I going with a friend?
  - $\boldsymbol{x}_3$: Is the venue easy to commute?
- **Assumptions & Weights:**
  - You dislike bad weather (x_1)
  - You would consider going alone (x_2)
  - You don’t mind a longer commute on weekends (x_3)



Perceptron - a.k.a. single neuron

Output: Go to Gouda for the cheese festival on Saturday

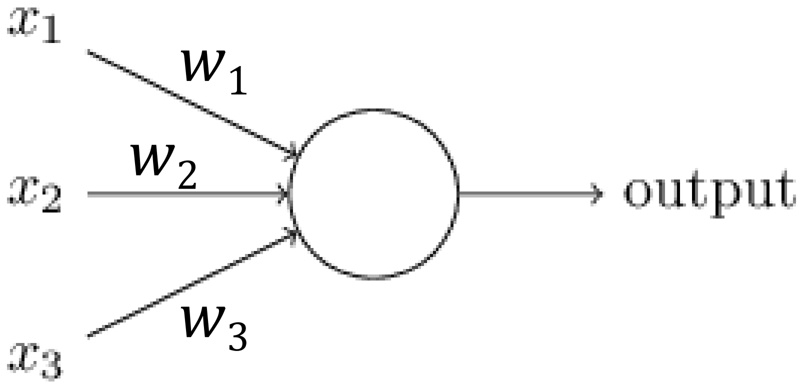
- **Inputs:**
  - $\boldsymbol{x}_1$: Is the weather good?
  - $\boldsymbol{x}_2$: Am I going with a friend?
  - $\boldsymbol{x}_3$: Is the venue easy to commute?
- **Assumptions & Weights:**
  - You dislike bad weather (x_1), so $\boldsymbol{w_1 = 6}$
  - You would consider going alone (x_2), so $\boldsymbol{w}_2 = \mathbf{2}$
  - You don’t mind a longer commute on weekends (x_3), so $\boldsymbol{w_3 = 2}$



Perceptron - a.k.a. single neuron

Output: Go to Gouda for the cheese festival on Saturday

- **Inputs:**
  - $\boldsymbol{x}_1$: Is the weather good?
  - $\boldsymbol{x}_2$: Am I going with a friend?
  - $\boldsymbol{x}_3$: Is the venue easy to commute?
- **Assumptions & Weights:**
  - You dislike bad weather (x_1), so $\boldsymbol{w_1} = \boldsymbol{6}$
  - You would consider going alone (x_2), so $\boldsymbol{w}_2 = \boldsymbol{2}$
  - You don't mind a longer commute on weekends (x_3), so $\boldsymbol{w_3} = \boldsymbol{2}$
- **Questions:**
  - What would happen if threshold is 5?

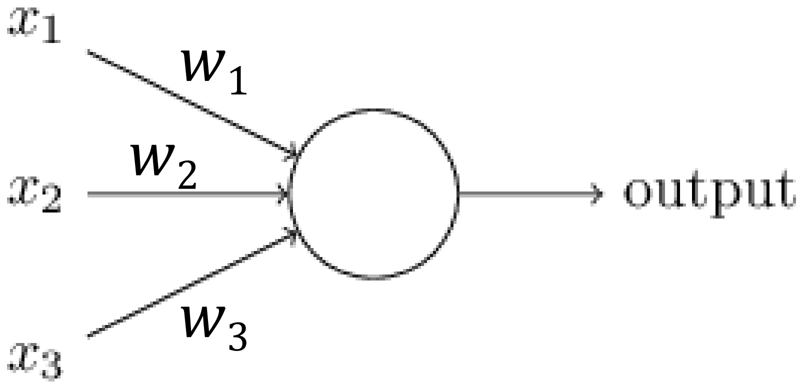


$$output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j \leq threshold \\ 1 \;\; if \; \sum_j w_j x_j > threshold \end{cases}$$

Perceptron - a.k.a. single neuron

Output: Go to Gouda for the cheese festival on Saturday

- **Inputs:**
  - $\boldsymbol{x}_1$: Is the weather good?
  - $\boldsymbol{x}_2$: Am I going with a friend?
  - $\boldsymbol{x}_3$: Is the venue easy to commute?
- **Assumptions & Weights:**
  - You dislike bad weather (x_1), so $\boldsymbol{w_1} = \boldsymbol{6}$
  - You would consider going alone (x_2), so $\boldsymbol{w}_2 = \boldsymbol{2}$
  - You don't mind a longer commute on weekends (x_3), so $\boldsymbol{w_3} = \boldsymbol{2}$



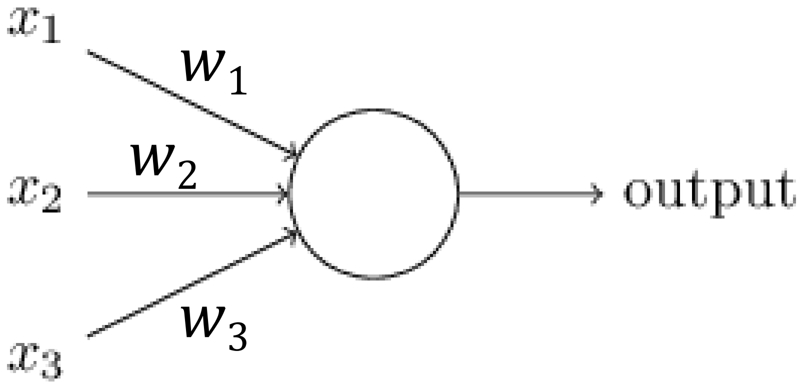
- **Questions:**
  - What would happen if threshold is 5?
  - What would happen if threshold is 3?

$$output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j \; \leq threshold \\ 1 \;\; if \; \sum_j w_j x_j \; > threshold \end{cases}$$

Perceptron - a.k.a. single neuron

- The weighted sum can be expressed as an **inner product** of two vectors:

$$\sum_j w_j x_j = w \cdot x$$

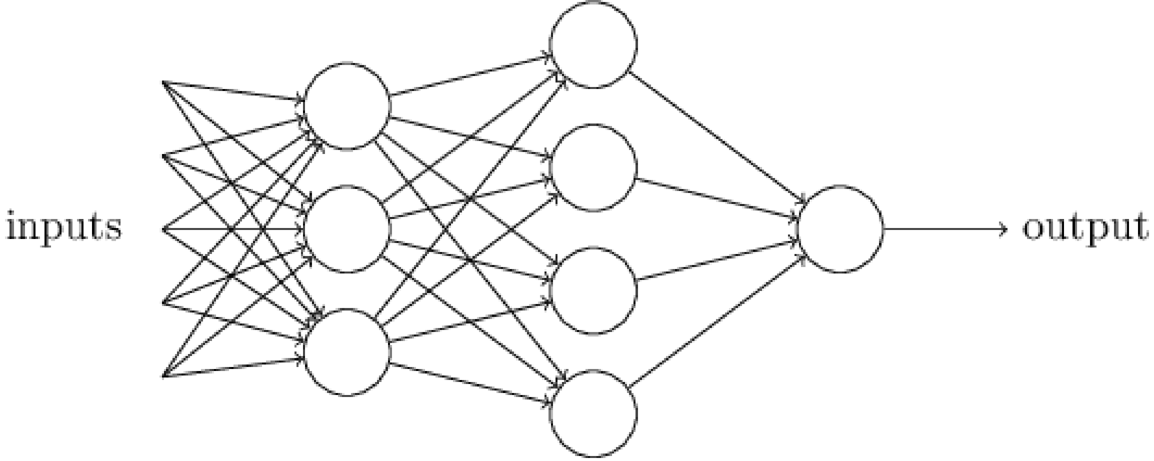


- By setting **b = -threshold**, the formula becomes:

$$output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j \leq threshold \\ 1 \;\; if \; \sum_j w_j x_j > threshold \end{cases} \quad \Longrightarrow \quad output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j + b \leq 0 \\ 1 \;\; if \; \sum_j w_j x_j + b > 0 \end{cases}$$

This reformulation simplifies the computation in neural networks.

Layers of Perceptrons

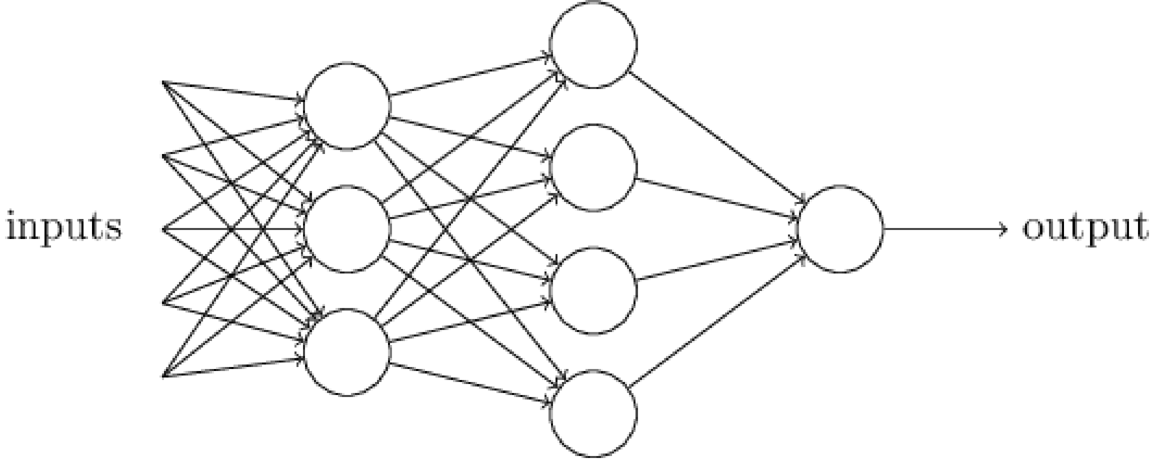


- **First Layer:**
  - Processes raw inputs by making simple decisions (e.g., three basic decisions)
  - Each perceptron weighs specific features differently from the input data

- **Second Layer:**
  - Takes the outputs from the first layer as its inputs
  - Combines these basic decisions to form four more complex decisions
  - Integrates multiple first-layer insights to capture higher-level features

- **Overall Impact:**
  - Stacking layers creates a hierarchical structure
  - Early layers focus on simple, local features, while deeper layers synthesize these into sophisticated, global patterns

Layers of Perceptrons



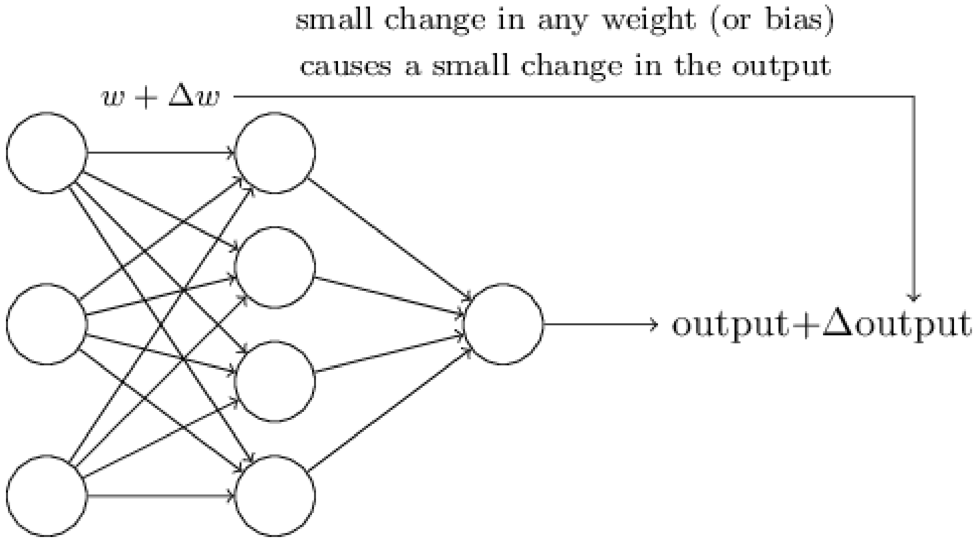
- **First Layer:**
  - Processes raw inputs by making simple decisions (e.g., three basic decisions)
  - Each perceptron weighs specific features differently from the input data

- **Second Layer:**
  - Takes the outputs from the first layer as its inputs
  - Combines these basic decisions to form four more complex decisions
  - Integrates multiple first-layer insights to capture higher-level features

- **Overall Impact:**
  - Stacking layers creates a hierarchical structure
  - Early layers focus on simple, local features, while deeper layers synthesize these into sophisticated, global patterns

  But how we set the **weights (and biases)**?

Neural Networks



Learning Process:

- We observe how small changes in weights affect the network's output.
- Starting from random weights, we iteratively adjust them to move the output closer to the expected value.

Supervised Updates:

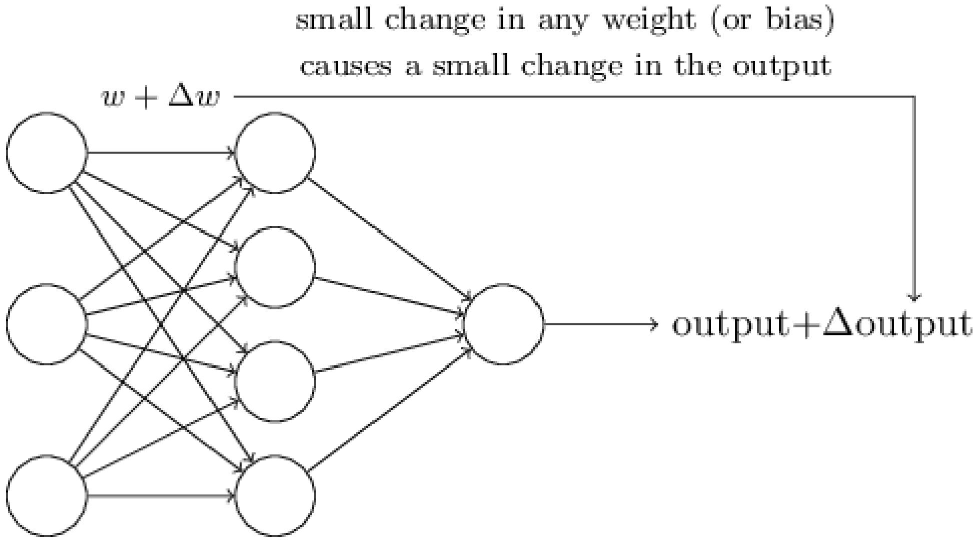
- The weight adjustments are supervised, ensuring the network learns the desired patterns.

Validation:

- The learning process is monitored using separate data not involved in the weight optimization.
- This validation step helps control overfitting and ensures robust generalization.

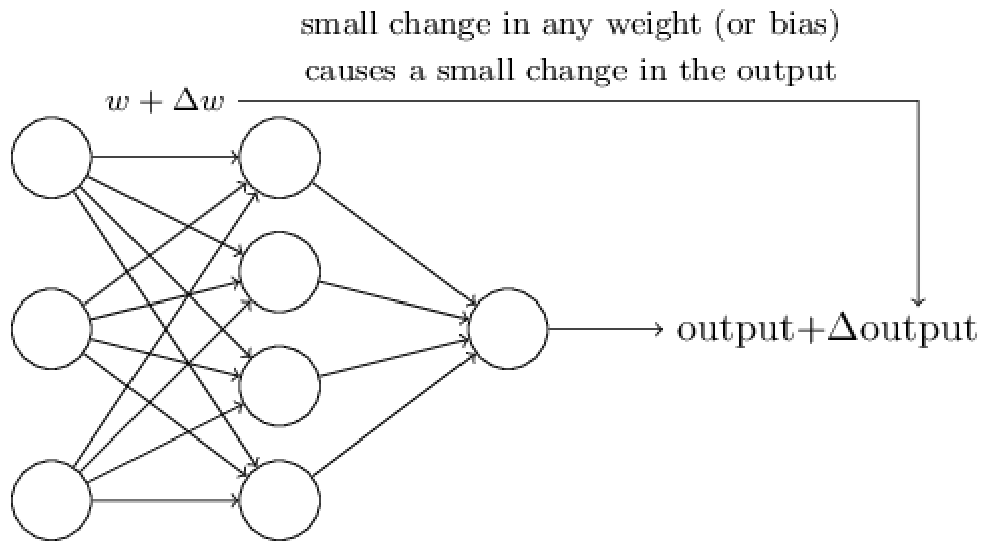
Problem with Perceptron:

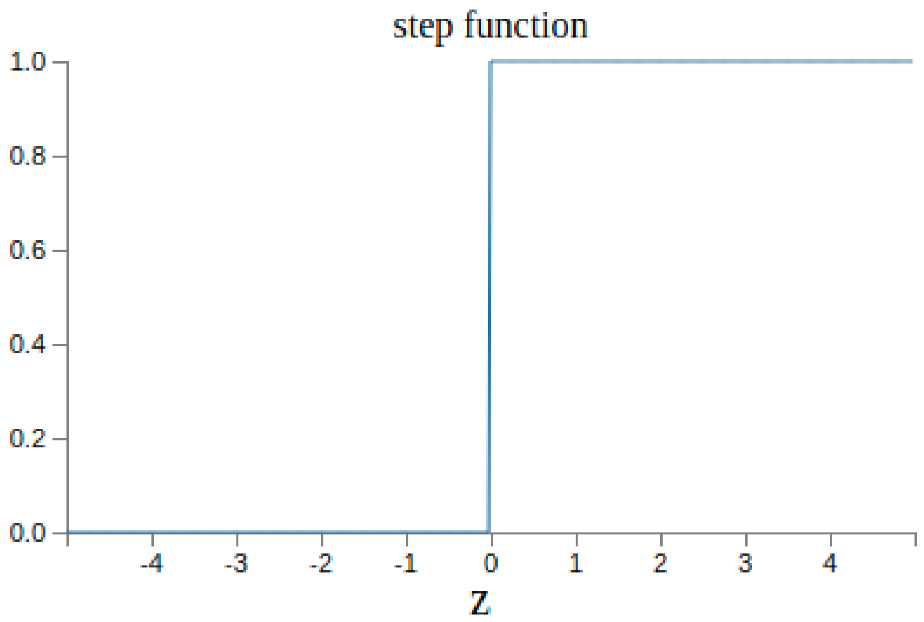
$$output = \begin{cases} 0 \ \ if \ \sum_j w_j x_j + b \leq 0 \\ 1 \ \ if \ \sum_j w_j x_j + b > 0 \end{cases}$$



Problem with Perceptron:

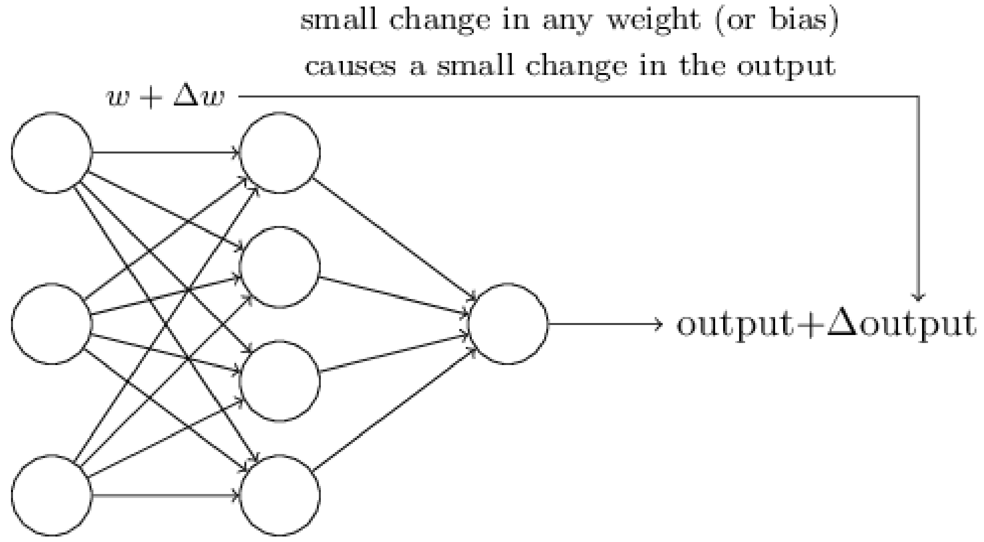
$$output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j + b \leq 0 \\ 1 \;\; if \; \sum_j w_j x_j + b > 0 \end{cases}$$

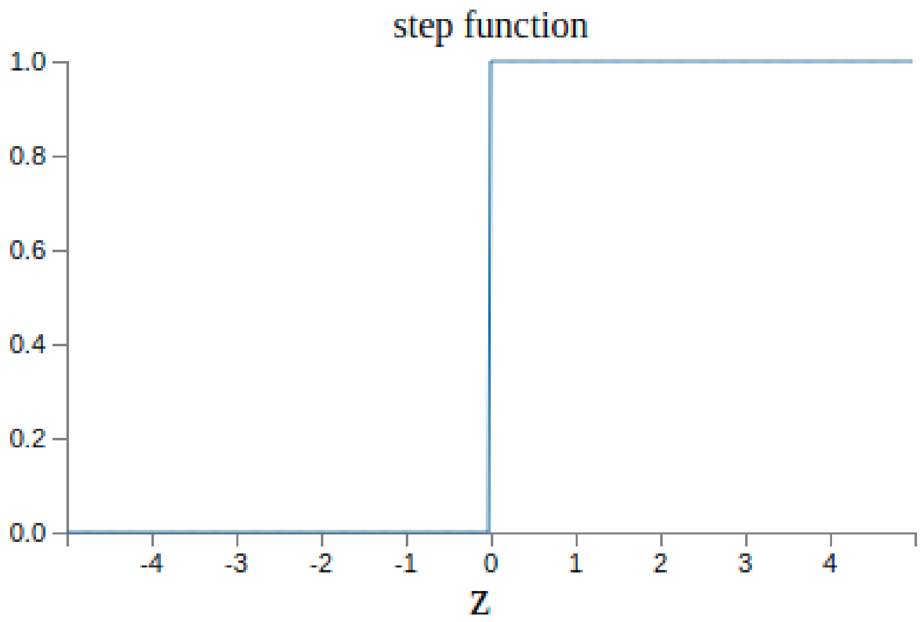


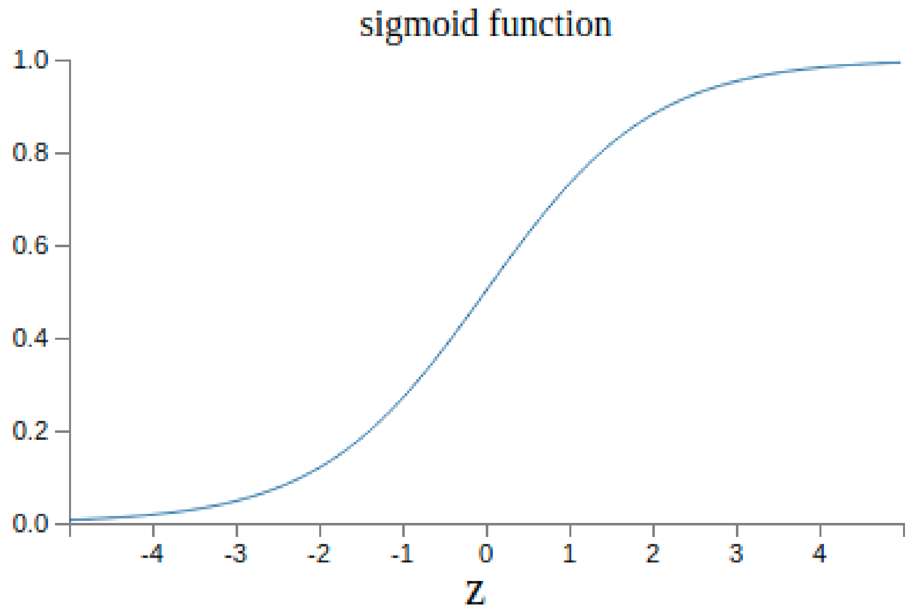


Problem with Perceptron:

$$output = \begin{cases} 0 \;\; if \; \sum_j w_j x_j + b \leq 0 \\ 1 \;\; if \; \sum_j w_j x_j + b > 0 \end{cases}$$







Sigmoid Neuron

A ~~perceptron~~ sigmoid neuron takes multiple inputs
(e.g., x_1, x_2, x_3),
computes a **weighted sum**, and produces a **~~binary~~ single output.**

Input & Operation:

- Takes multiple inputs (e.g., x_1, x_2, x_3)
- Computes a weighted sum of the inputs plus a bias

Activation Function:

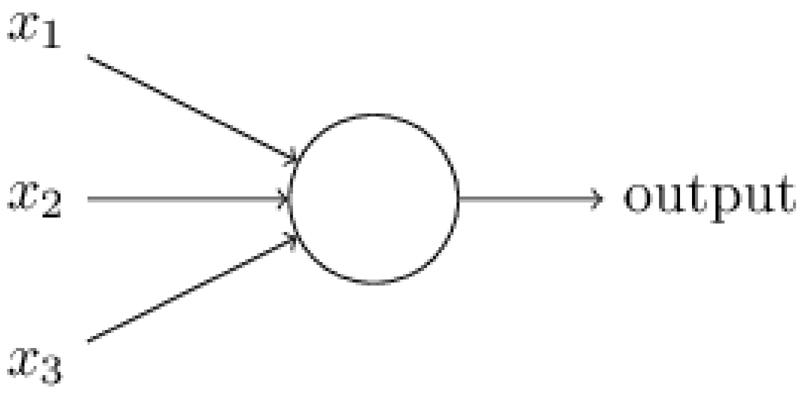
- Uses the **sigmoid function**:

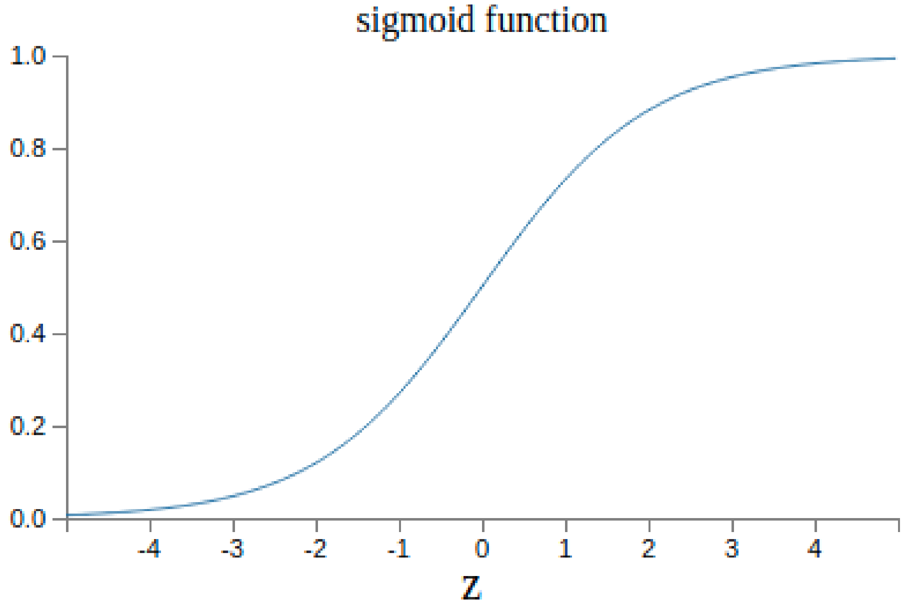
$$\sigma(z) = 1/(1 + e^{-(w \cdot x + b)})$$

- Produces a continuous output between 0 and 1

Key Benefits:

- Allows for smooth transitions in output
- Enables gradient-based learning for fine-tuned weight adjustments





First Order Taylor Approximation (Quick Lookup)

Given: A function f and a known value f(c) at point c

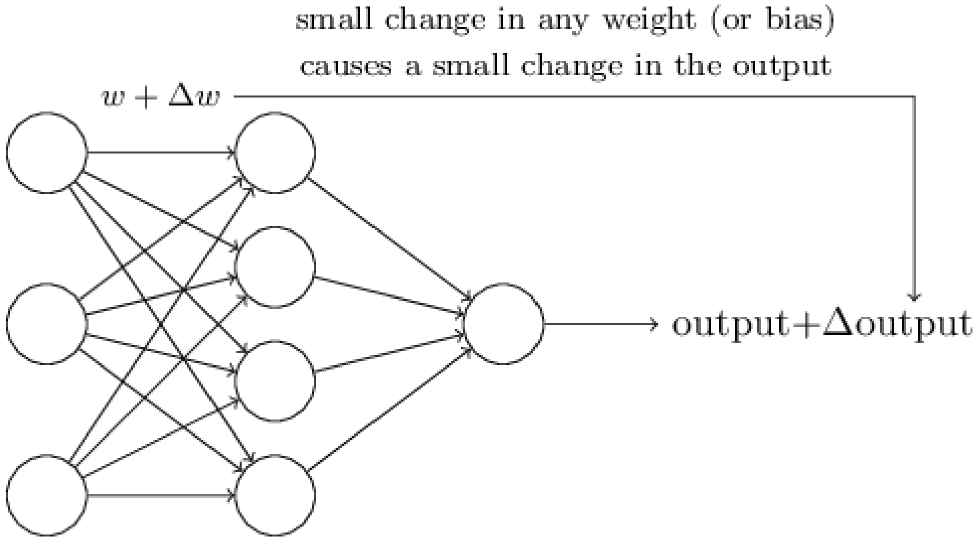
Approximation in the Neighborhood: For x near c, f(x) can be approximated by:

$$f(x) \approx f(c) + \nabla f(c) \cdot (x - c)$$

Reparametrizing it:

$$f(x) - f(c) \approx \nabla f(c) \cdot (x - c)$$
$$\Delta f \approx \nabla f(c) \cdot \Delta x$$

Neural Networks:



Small changes in weights (Δw) and biases (Δb) lead to corresponding changes in the neuron's output (Δf).

In neural networks, you want to understand how to optimize model parameters (weights (Δw) and biases (Δb)) such that $\Delta output$ goes towards the desired groundtruth output.

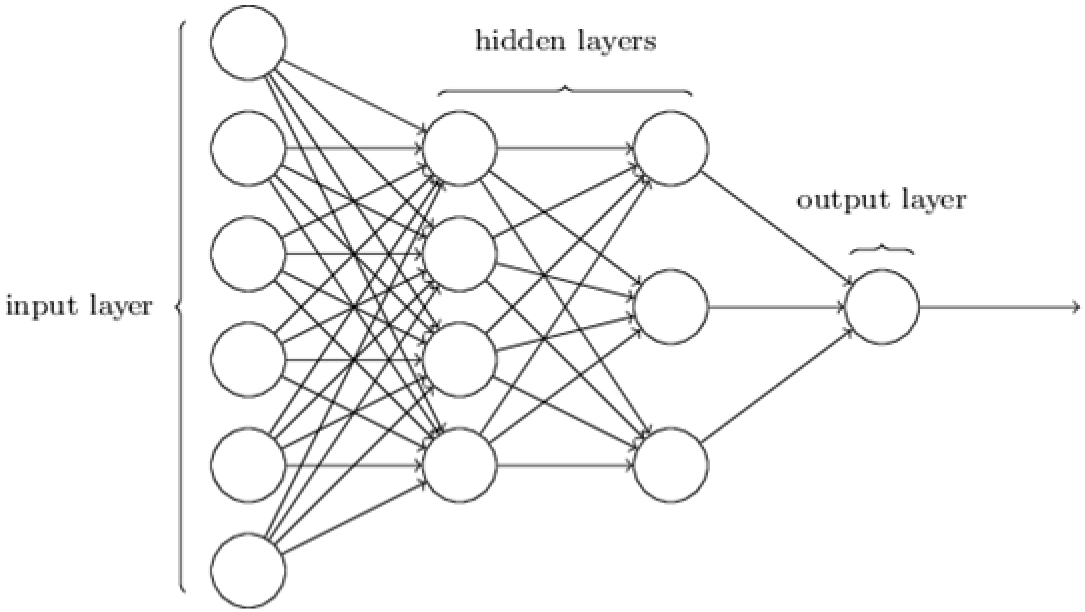
$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial\, \text{output}}{\partial b} \Delta b$$

Feedforward Network architecture

A feedforward network processes data in one direction – from input to output – with no loops.

Layer Types:

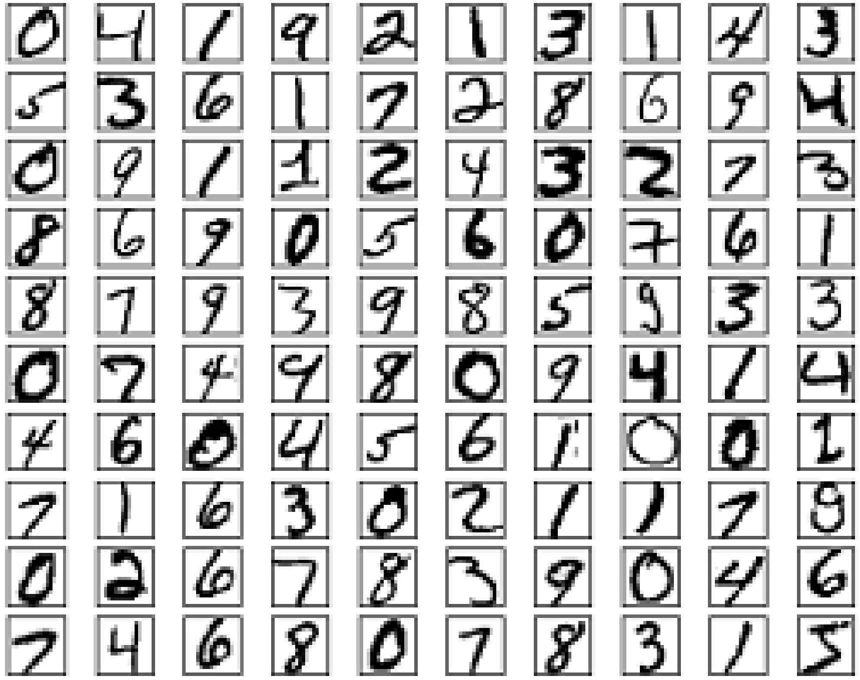
- **Input Layer:**
  - Receives raw data (e.g., pixel values)
- **Hidden Layers:**
  - One or more layers that extract features
  - Utilize activation functions like sigmoid
- **Output Layer:**
  - Produces final predictions (e.g., classification probabilities)



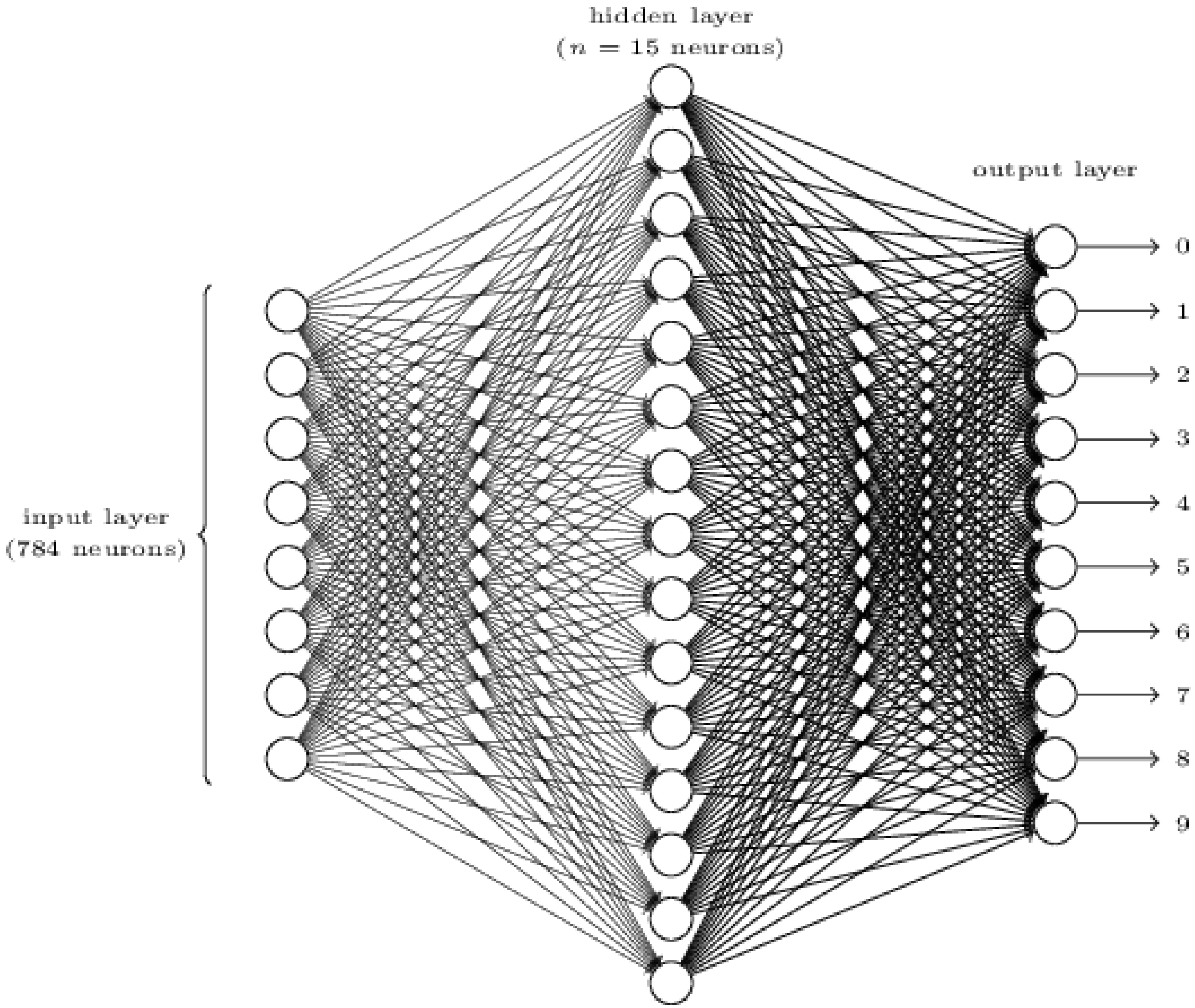
Recognizing Digits with Neural Nets.

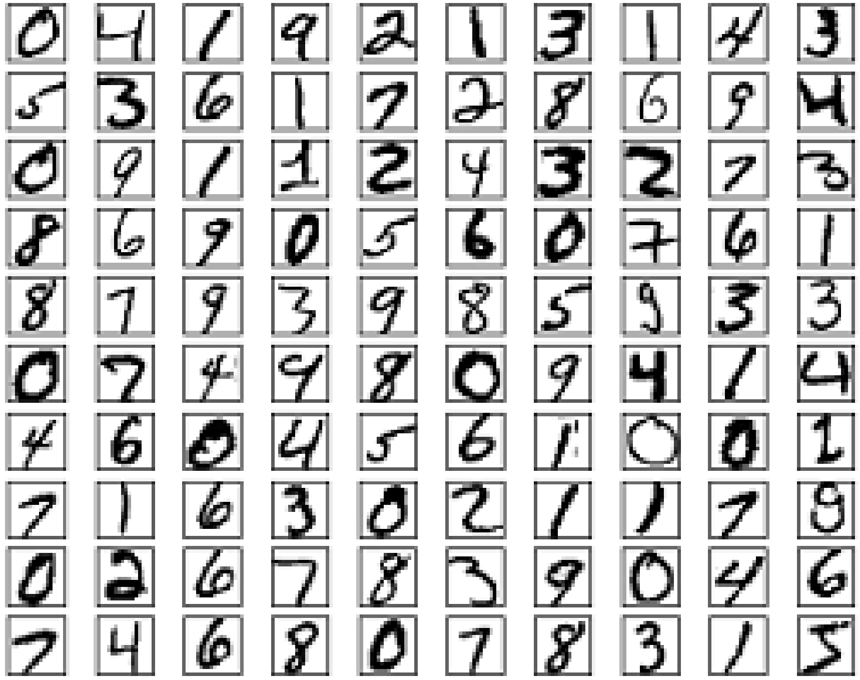
MNIST Dataset:

- **Overview:** A benchmark dataset for handwritten digit recognition.
- **Dataset Details: Images:** 70,000 grayscale images (60,000 for training, 10,000 for testing)
- **Dimensions:** Each image is 28×28 pixels, flattened into a 784-dimensional vector
- **Labels:** Each image corresponds to a digit (0–9)
- **Significance:** Widely used to train and validate neural network models
- Serves as a standard testbed for classification algorithms and deep learning research

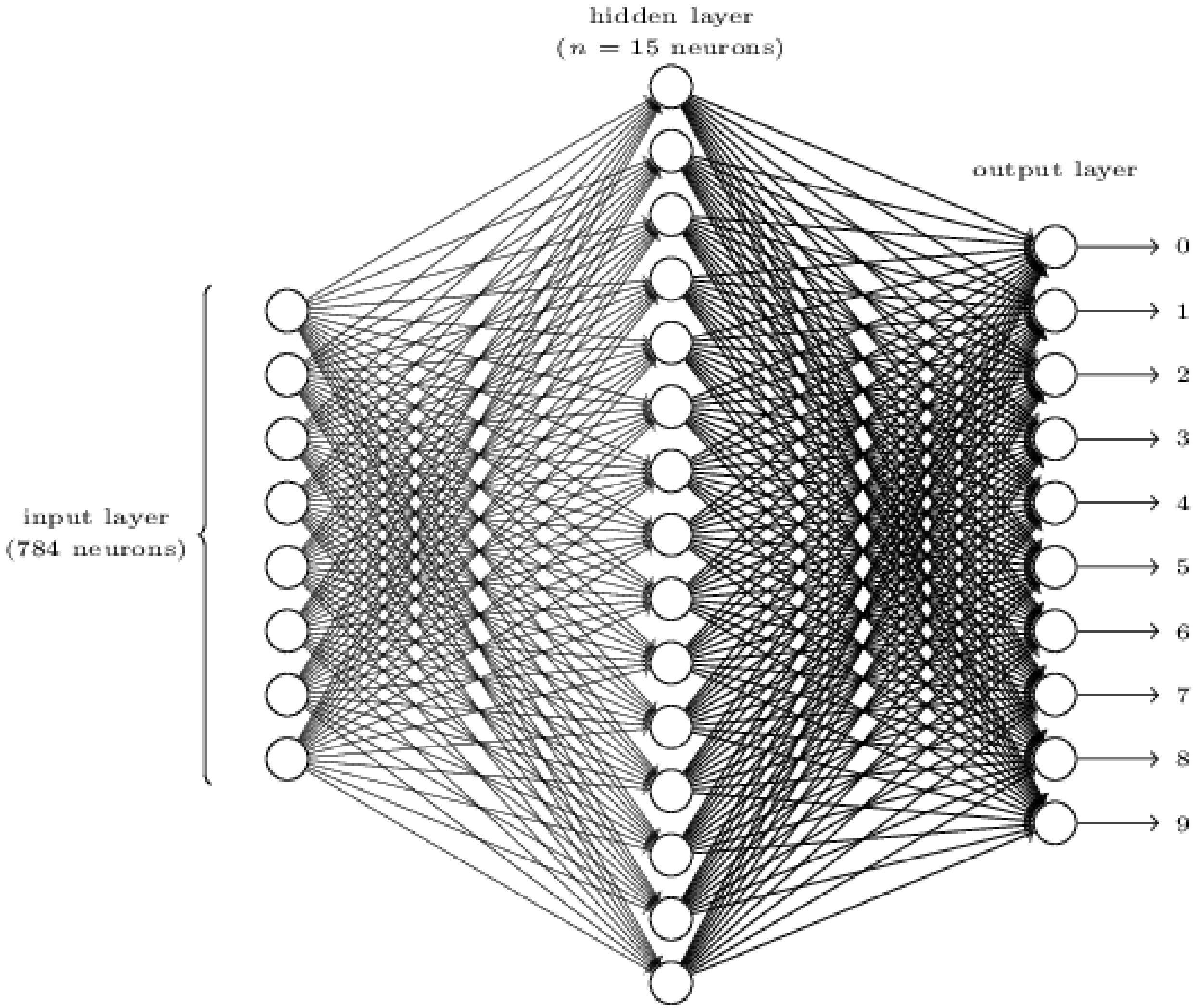


Recognizing Digits with Neural Nets.





Recognizing Digits with Neural Nets.



One-Hot Encoding:
•Represent the digit 6 as a 10-dimensional vector
•**Example:** (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)
•Only the 7th position is 1; all others are 0
Mean Squared Error (MSE) Cost Function:
•**Formula:** $C(w, b) = \frac{1}{2n} \sum_x \| y(x) - a \|^2$

- y(x): Expected output (one-hot encoded label)
- a: Activation/output from the network

•Measures the squared difference between the predicted and true outputs

Learning with gradient descent in single slide

Key Idea:
•A small change in weights (Δw) leads to a change in cost (ΔC)

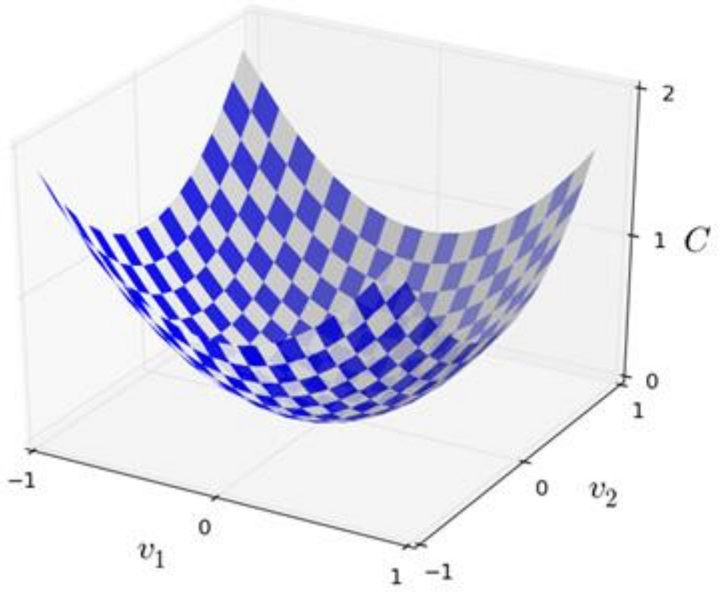
Approximation: $\Delta C \approx \nabla C \cdot \Delta w$
•To minimize cost, choose Δw to **move in the opposite direction** of the gradient:

- $\boldsymbol{\Delta w = -\eta \nabla C}$

Gradient Descent Update Rule:
•**Weights:** $w \rightarrow w - \eta \, \partial C / \partial w$
•**Biases:** $b \rightarrow b - \eta \, \partial C / \partial b$
•η: learning rate (controls step size)

Stochastic Approach:
•Instead of full dataset, use **mini-batches** to estimate gradients
•Faster and scalable for large datasets

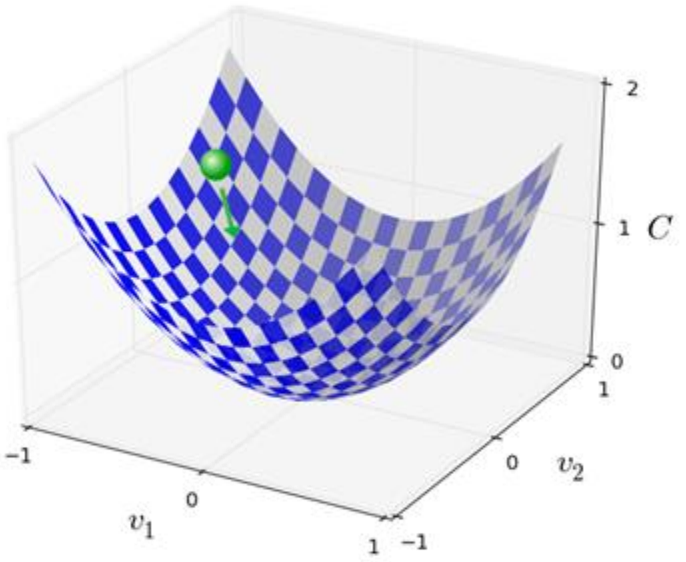
Epoch:
•One full pass over the training set
•Multiple epochs gradually refine weights and biases



Learning with gradient descent

1. Start with Random Initialization:

- Randomly set initial weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$
- Calculate the initial cost $\boldsymbol{C}(\boldsymbol{w}, \boldsymbol{b})$



Learning with gradient descent

1. Start with Random Initialization:

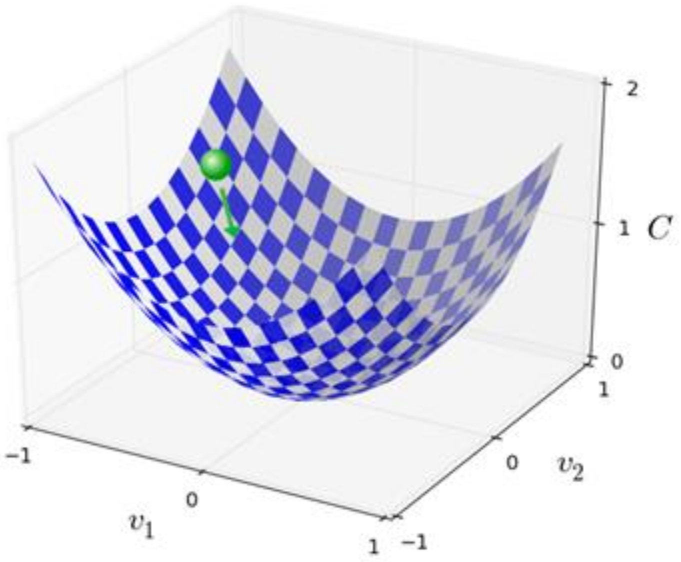
- Randomly set initial weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$
- Calculate the initial cost $\boldsymbol{C}(\boldsymbol{w}, \boldsymbol{b})$

2. Relate Weight Changes to Cost Change:

- Small changes in weights affect the cost:
    - $\Delta C \approx (\partial C/\partial w_1)\Delta w_1 + (\partial C/\partial w_2)\Delta w_2 + (\partial C/\partial w_3)\Delta w_3$

3. Express as Gradient Dot Product:

- The partial derivatives form the **gradient of C**:
    - $\nabla C = (\partial C/\partial w_1, \partial C/\partial w_2, \partial C/\partial w_3)^{\mathrm{T}}$
- Then: $\Delta C \approx \nabla C \cdot \Delta w$



Learning with gradient descent

1. Start with Random Initialization:

- Randomly set initial weights $\boldsymbol{w}$ and biases $\boldsymbol{b}$
- Calculate the initial cost $\boldsymbol{C}(\boldsymbol{w}, \boldsymbol{b})$

2. Relate Weight Changes to Cost Change:

- Small changes in weights affect the cost:
  - $\Delta C \approx (\partial C/\partial w_1)\Delta w_1 + (\partial C/\partial w_2)\Delta w_2 + (\partial C/\partial w_3)\Delta w_3$

3. Express as Gradient Dot Product:

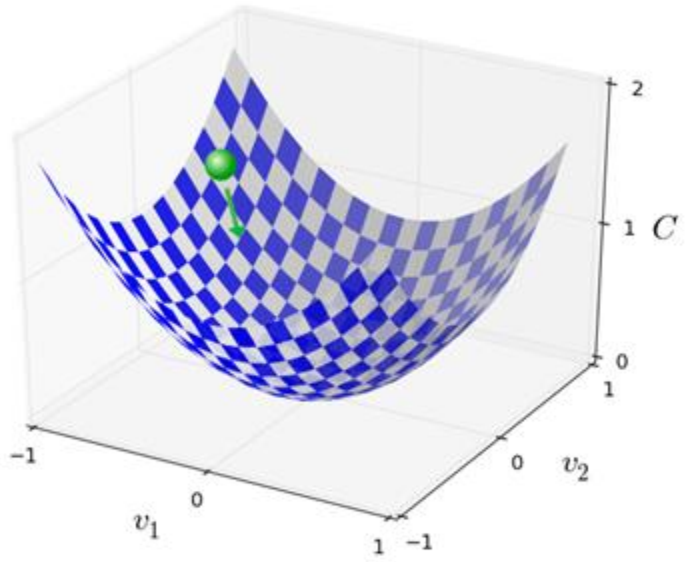
- The partial derivatives form the **gradient of C**:
  - $\nabla C = (\partial C/\partial w_1, \partial C/\partial w_2, \partial C/\partial w_3)^{\mathrm{T}}$
- Then: $\Delta C \approx \nabla C \cdot \Delta w$

4. Choose Δw to Minimize Cost:

- Set $\Delta w = -\eta \, \nabla C$ (move in direction of steepest descent)
- Update rule for weights:
  - $w' = w - \eta \, \nabla C$

5. Update Bias Similarly:

- Bias update: $b' = b - \eta \, \nabla C$



Summary:

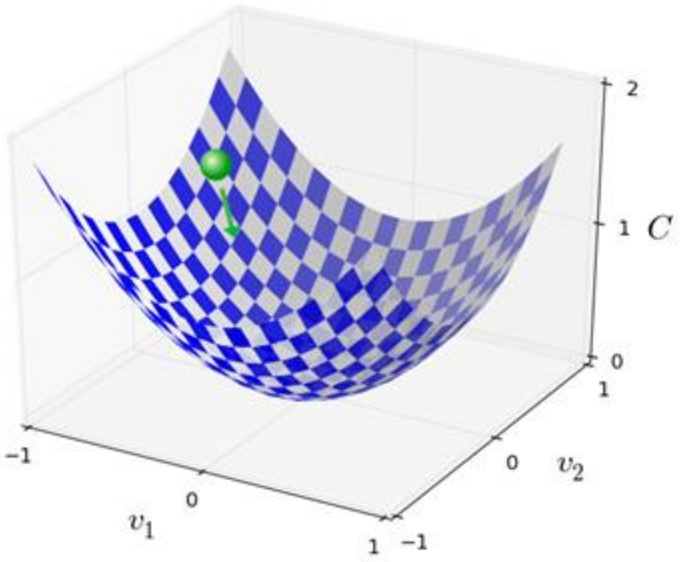
By computing the gradient of the cost, we iteratively update $\boldsymbol{w}$ and $\boldsymbol{b}$ in small steps (scaled by learning rate η) to reduce the cost and improve the network's performance.

Stochastic Gradient Descent

Gradient Over Full Dataset:

•Exact gradient:

- $\nabla C = \frac{1}{n}\sum_x \nabla C(x)$
- Where **n** = total number of training examples



Stochastic Gradient Descent

Gradient Over Full Dataset:

•Exact gradient:

- $\nabla C = \frac{1}{n}\sum_x \nabla C(x)$
- Where **n** = total number of training examples
- Computing this for large datasets is **slow and expensive**

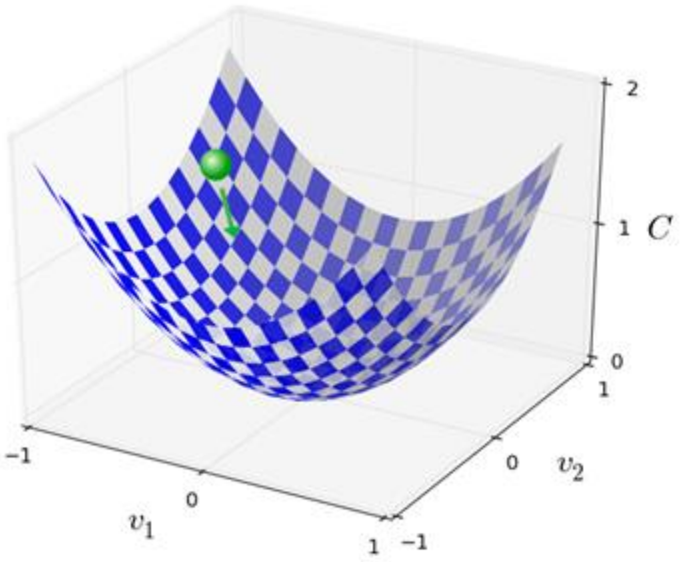
Stochastic Approximation:

•Use a **mini-batch** of m random samples ($m \ll n$):

- $\nabla C \approx \frac{1}{m}\sum_x \nabla C(x)$ over mini-batch

•Average gradient over mini-batch ≈ average gradient over entire dataset

•This approximation is **faster** and **computationally efficient**

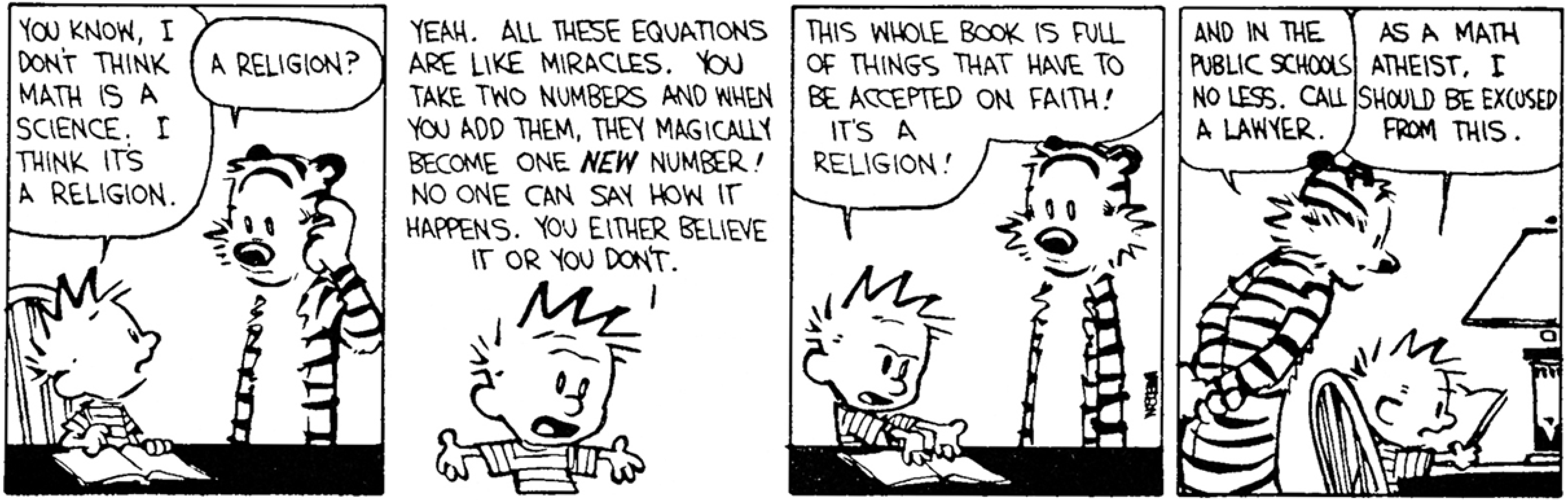


Weight Update Rule Using Mini-Batch:

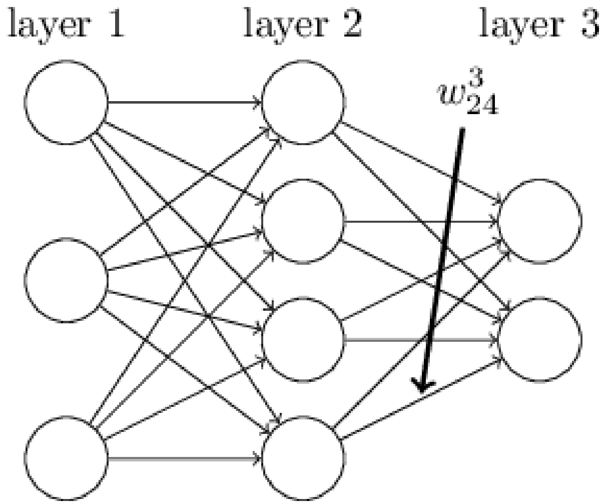
•$w \to w - (\eta/m)\sum_x \partial C(x)/\partial w$

•$b \to b - (\eta/m)\sum_x \partial C(x)/\partial b$

Backpropagation

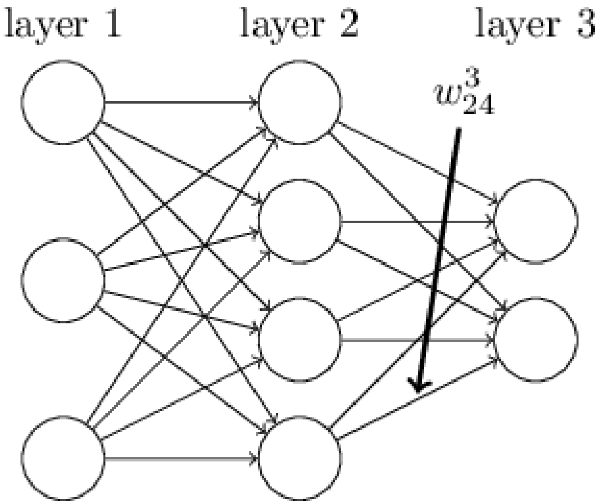


Backpropagation: Notation

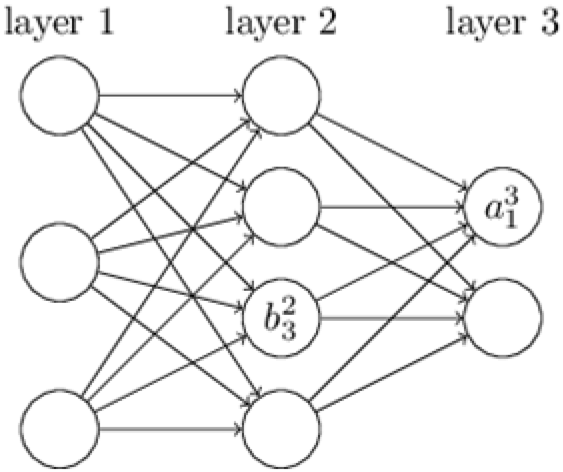


w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

Backpropagation: Notation

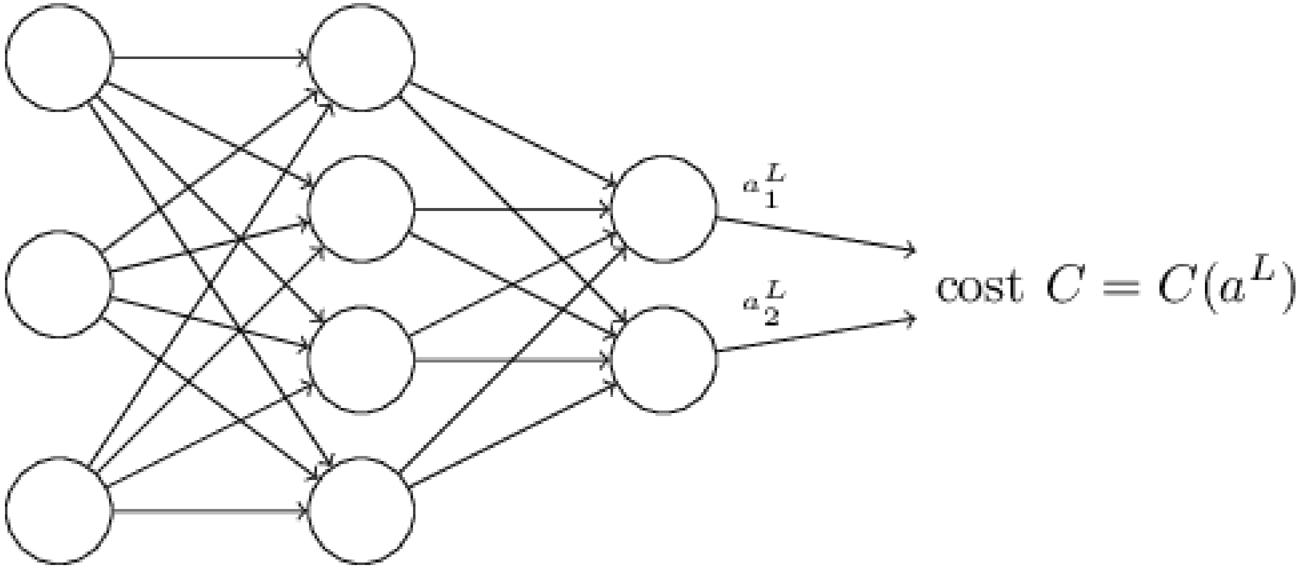


w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

Backpropagation: Cost function

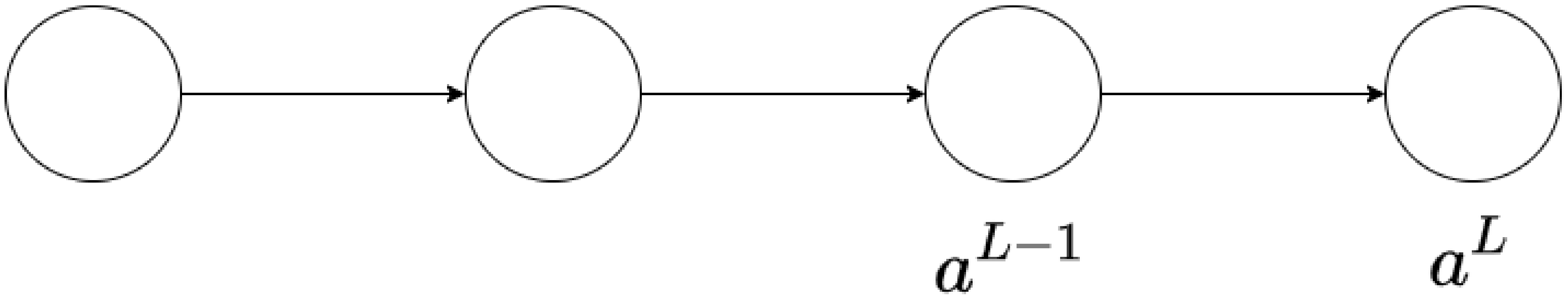


$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

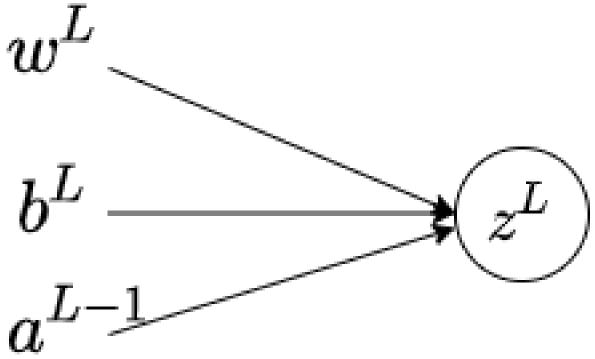
Backpropagation

$$C = \sum (y - a^L)^2$$

$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

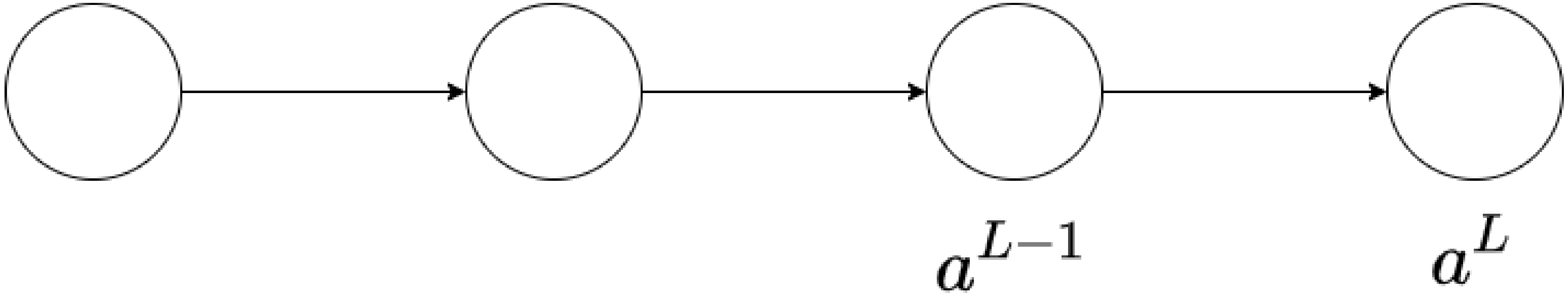


Backpropagation

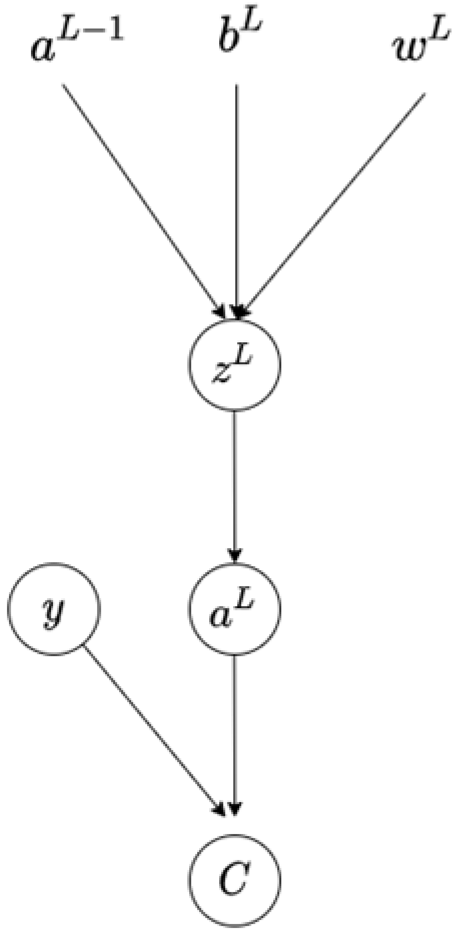


$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

$$C = \sum (y - a^L)^2$$
$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

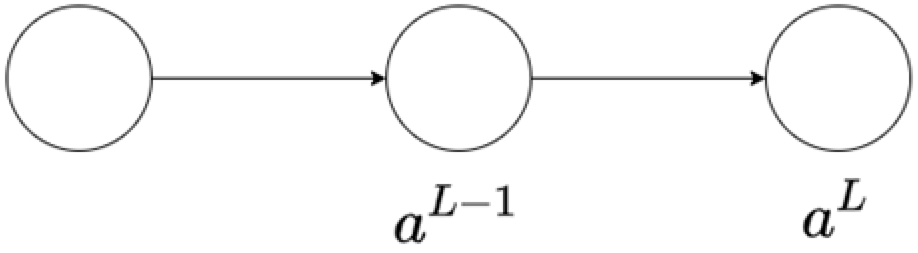


Backpropagation

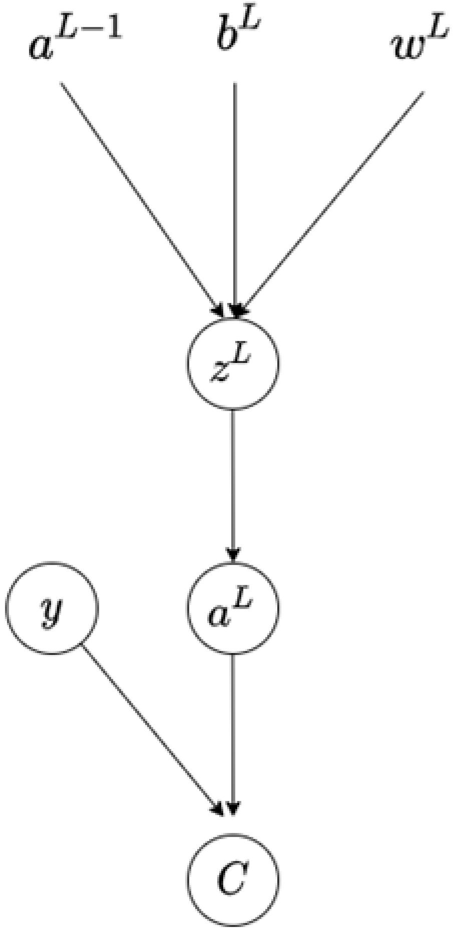


$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

$$C = \sum (y - a^L)^2$$
$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

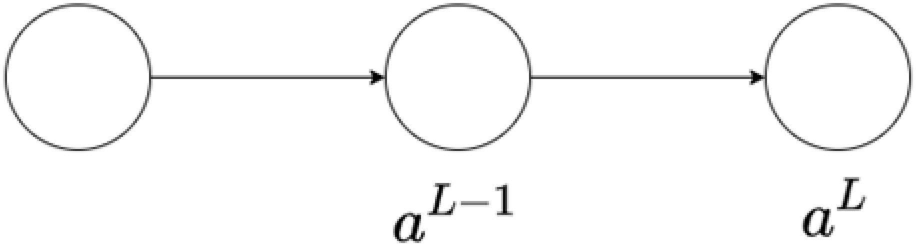


Backpropagation



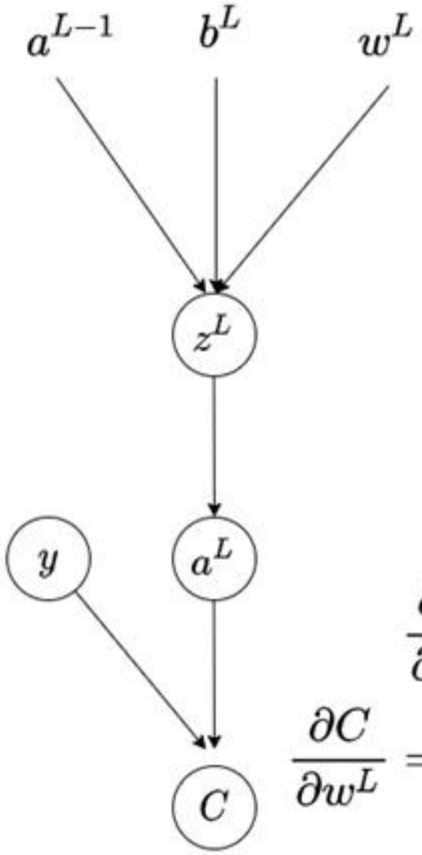
$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

$$C = \sum (y - a^L)^2$$
$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$



$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$
$$\frac{\partial C}{\partial z^L} = \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L} = \sigma'(z^L) \cdot 2(a^L - y)$$

Backpropagation

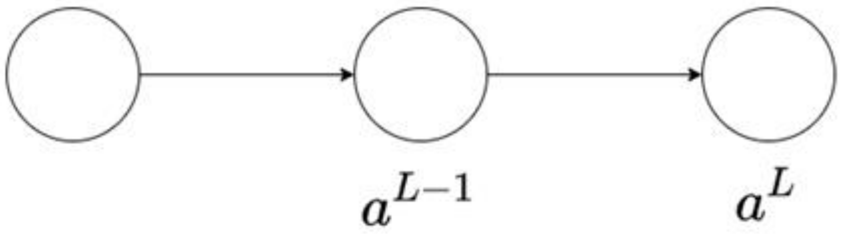


$$z^L = w^L \cdot a^{L-1} + b^L$$

$$a^L = \sigma(z^L)$$

$$C = \sum (y - a^L)^2$$

$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

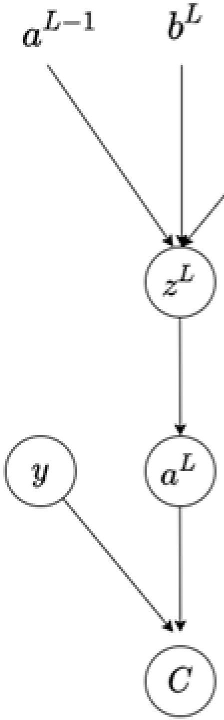


$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

$$\frac{\partial C}{\partial z^L} = \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L} = \sigma'(z^L) \cdot 2(a^L - y)$$

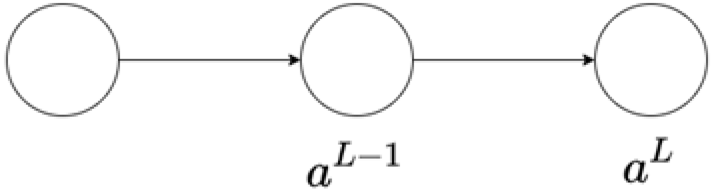
$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L} = a^{(L-1)} \sigma'(z^L) \cdot 2(a^L - y)$$

Backpropagation



$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

$$C = \sum (y - a^L)^2$$
$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$



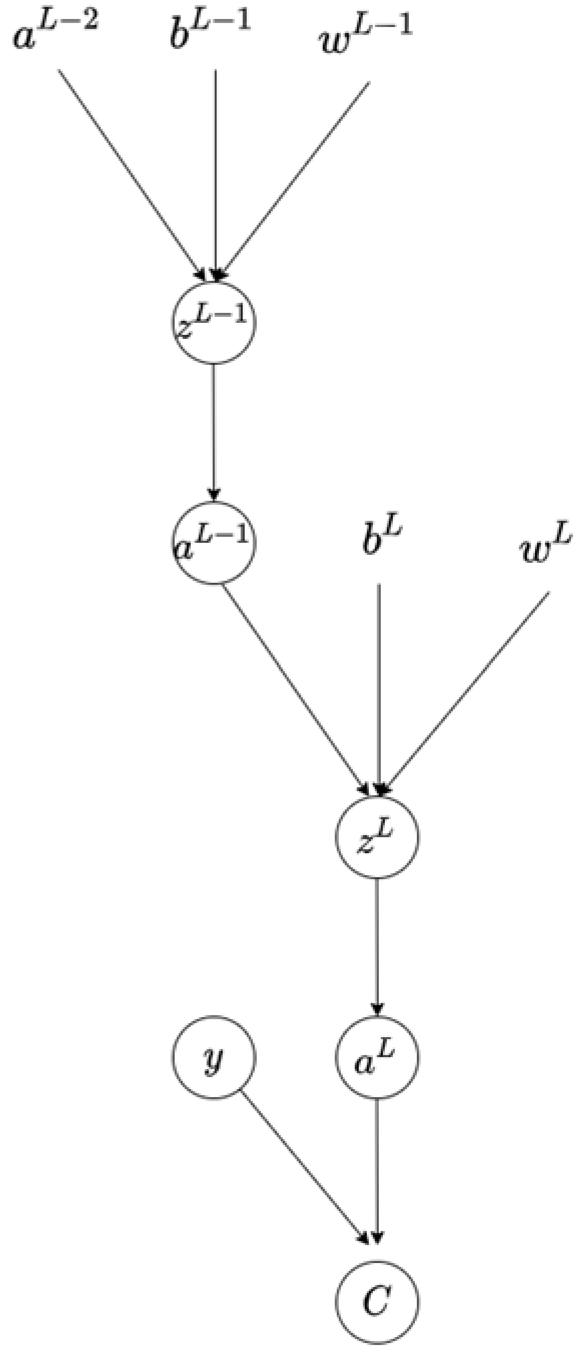
$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

$$\frac{\partial C}{\partial z^L} = \frac{\partial a^L}{\partial z^L}\frac{\partial C}{\partial a^L} = \sigma'(z^L) \cdot 2(a^L - y)$$

$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C}{\partial a^L} = a^{(L-1)}\sigma'(z^L) \cdot 2(a^L - y)$$

$$\frac{\partial C}{\partial b^L} = \frac{\partial z^L}{\partial b^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C}{\partial a^L} = 1 \cdot \sigma'(z^L) \cdot 2(a^L - y)$$

Backpropagation



Backpropagation

$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

$$\frac{\partial C}{\partial z^L} = \frac{\partial a^L}{\partial z^L}\frac{\partial C}{\partial a^L} = \sigma'(z^L)\cdot 2(a^L - y)$$

$$\frac{\partial C}{\partial w^{L-1}} = \frac{\partial z^{L-1}}{\partial w^{L-1}}\cdot\frac{\partial a^{L-1}}{\partial z^{L-1}}\cdot\frac{\partial z^L}{\partial a^{L-1}}\cdot\frac{\partial C}{\partial z^L} = a^{L-2}\cdot\sigma'(z^{L-1})\cdot w^{(L)}\cdot\frac{\partial C}{\partial z^L}$$

$$\frac{\partial C}{\partial b^{L-1}} = \frac{\partial z^{L-1}}{\partial b^{L-1}}\cdot\frac{\partial a^{L-1}}{\partial z^{L-1}}\cdot\frac{\partial z^L}{\partial a^{L-1}}\cdot\frac{\partial C}{\partial z^L} = \sigma'(z^{L-1})\cdot w^{(L)}\cdot\frac{\partial C}{\partial z^L}$$