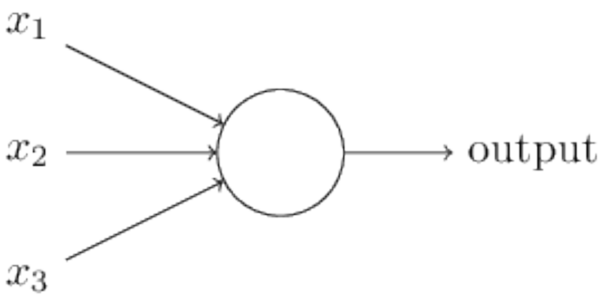
Neural Networks

Nail Ibrahimli

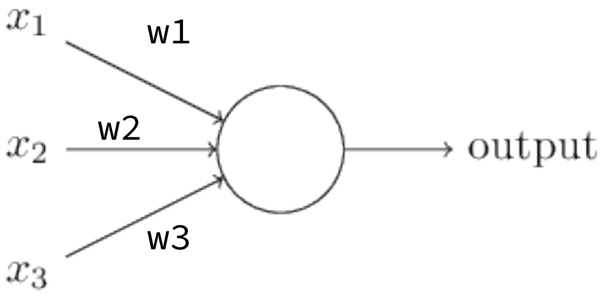
Perceptron - a.k.a. single neuron

A perceptron takes multiple inputs
(e.g.: x1, x2, x3) and produces a single binary output.



Perceptron - a.k.a. single neuron

A perceptron takes multiple inputs
(e.g.: x1, x2, x3) and produces a single binary output.



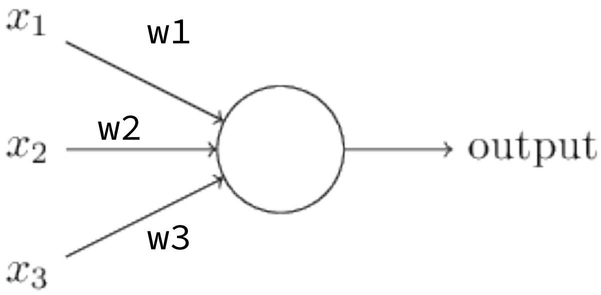
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

That's all there is to how a perceptron works!

Perceptron - a.k.a. single neuron

Output: Going to Gouda for cheese festival on Saturday.

X1: Is the weather good?
X2: Am I going with friend?
X3: Is the venue easy to commute?

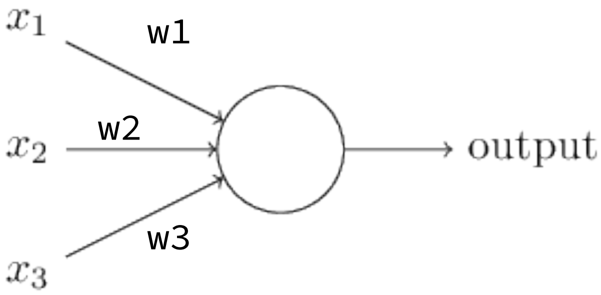


Perceptron - a.k.a. single neuron

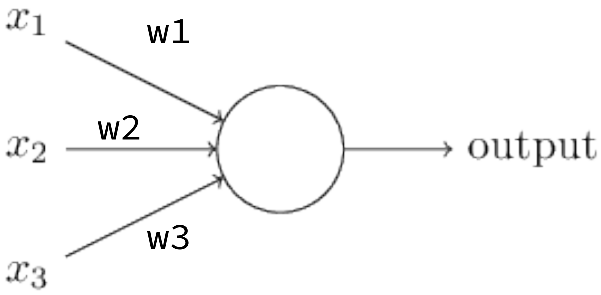
Output: Going to Gouda for cheese festival on Saturday.

X1: Is the weather good?
X2: Am I going with friend?
X3: Is the venue easy to commute?

Let’s say:
You don’t mind that much going alone (X2), and
since it is at the Weekend you don't mind that much to commute for longer(X3).
But you hate bad weather and you would rather stay at home in bad weather(X1).



Perceptron - a.k.a. single neuron



Output: Going to Gouda for cheese festival on Saturday.

X1: Is the weather good?
X2: Am I going with friend?
X3: Is the venue easy to commute?

Let's say:
You don't mind that much going alone (X2), and
since it is at the Weekend you don't mind that much to commute for longer(X3).
But you hate bad weather and you would rather stay at home in bad weather(X1).

w1 = 6, w2 = w3 = 2

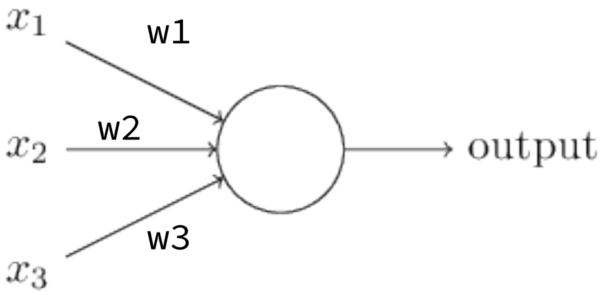
Perceptron - a.k.a. single neuron

Output: Going to Gouda for cheese festival on Saturday.

X1: Is the weather good?
X2: Am I going with friend?
X3: Is the venue easy to commute?

w1 = 6, w2 = w3 = 2

What would happen if threshold = 5?



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

Perceptron - a.k.a. single neuron

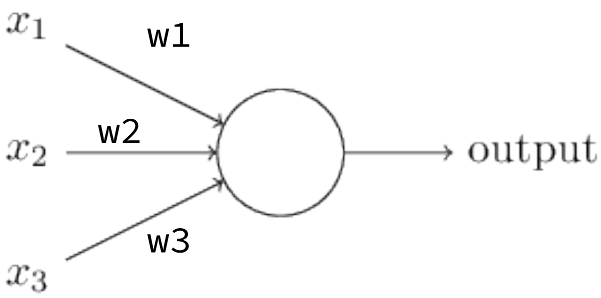
Output: Going to Gouda for cheese festival on Saturday.

X1: Is the weather good?
X2: Am I going with friend?
X3: Is the venue easy to commute?

w1 = 6, w2 = w3 = 2

What would happen if threshold = 5?

What would happen if threshold = 3?



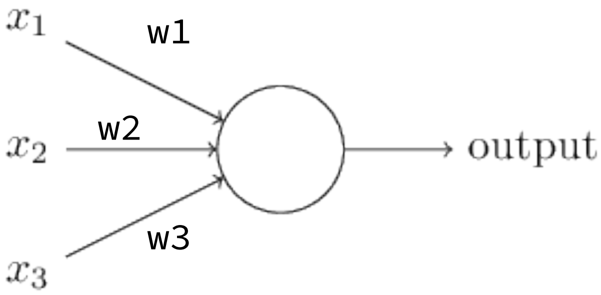
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Perceptron - a.k.a. single neuron

We can rewrite weighted sum as an inner product of two vectors.

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

We can assume that b = -threshold.



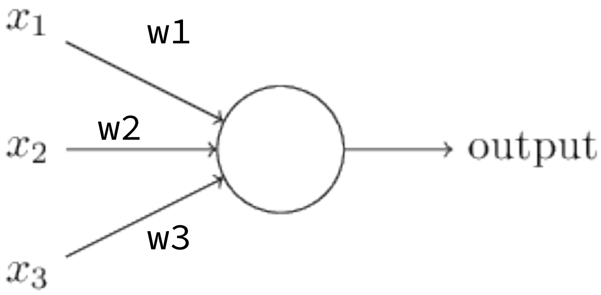
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Perceptron - a.k.a. single neuron

We can rewrite weighted sum as an inner product of two vectors.

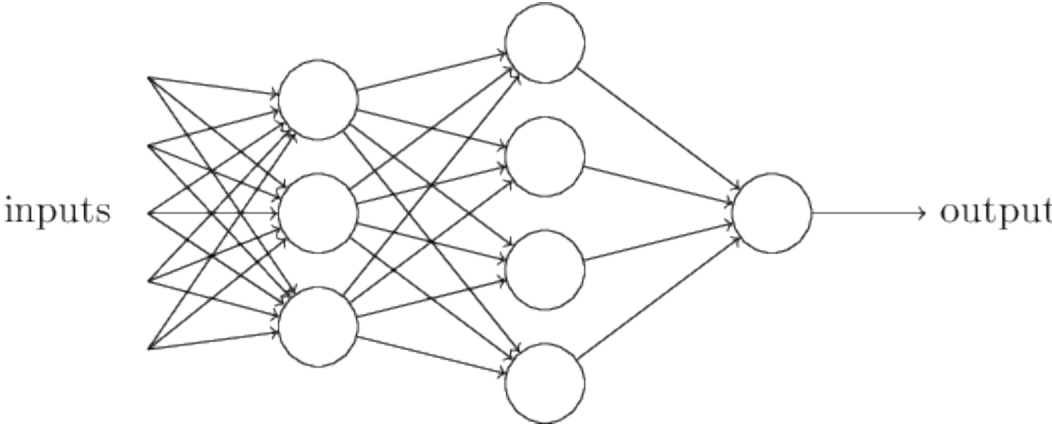
$$\sum_j w_j x_j = w \cdot x$$

We can assume that b = -threshold.



$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Layers of Perceptrons



First layer:

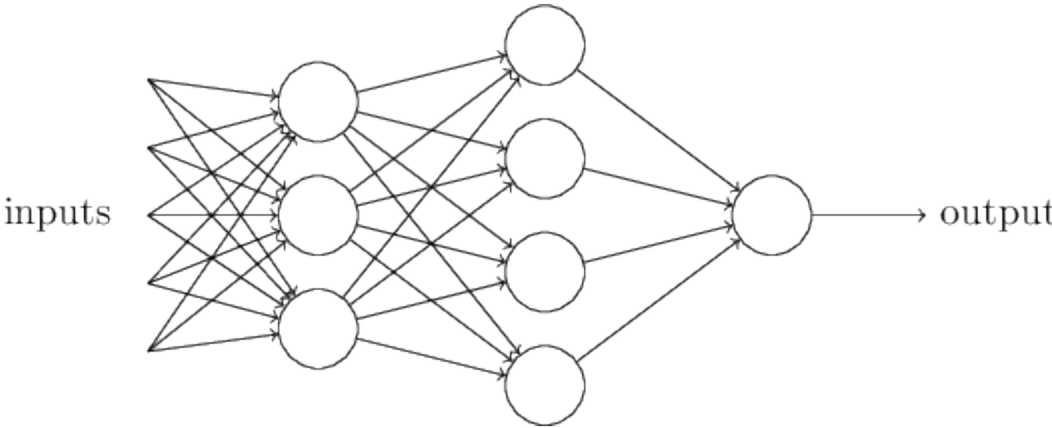
Making simple three decisions
by weighing inputs

Second layer:

Making four decisions by weighing
up the results from first layer decisions making

Using multiple layers of perceptrons, neural networks
can make more sophisticated decisions

Layers of Perceptrons



First layer:
Making simple three decisions
by weighing inputs

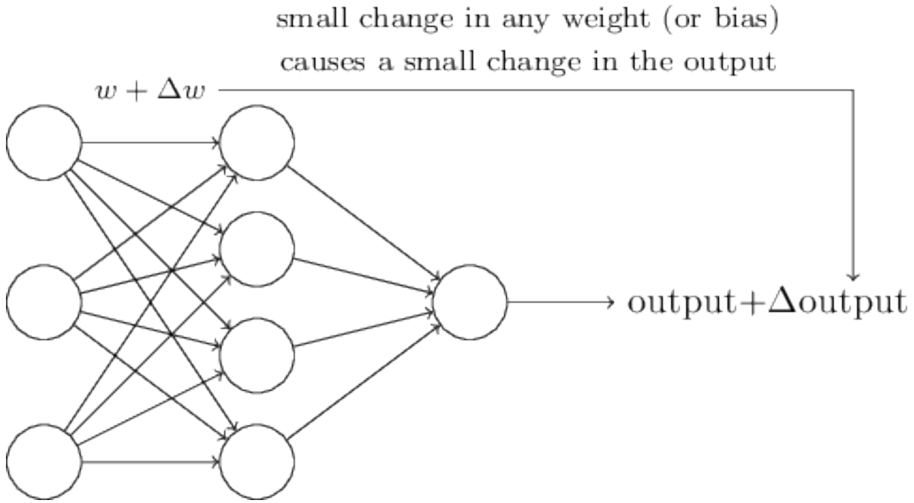
Second layer:
Making four decisions by weighing
up the results from first layer decisions making

Using multiple layers of perceptrons, neural networks
can make more sophisticated decisions.

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

Neural Networks

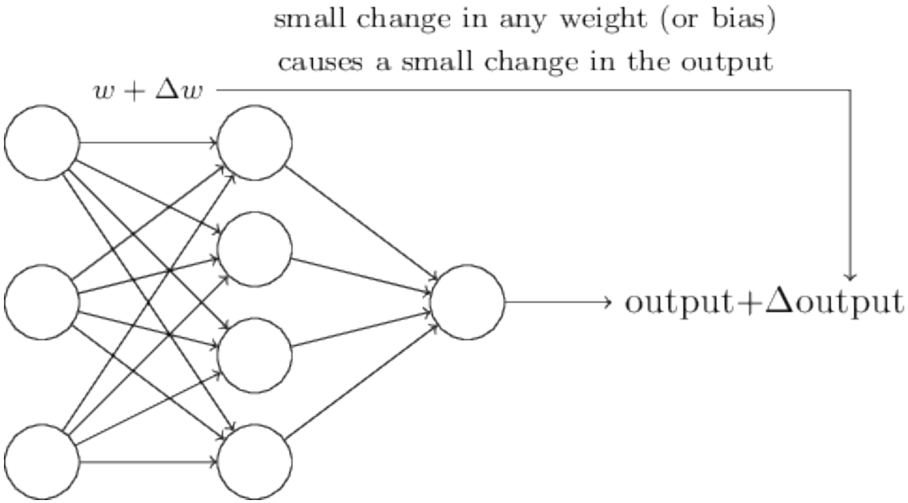
1. Suppose that we know how small change in weights changes the output.



Neural Networks

1. Suppose that we know how small change in weights changes the output.

2. Starting with random initialization we can iteratively update (supervise) the weights with small changes to bring the output to expected value.

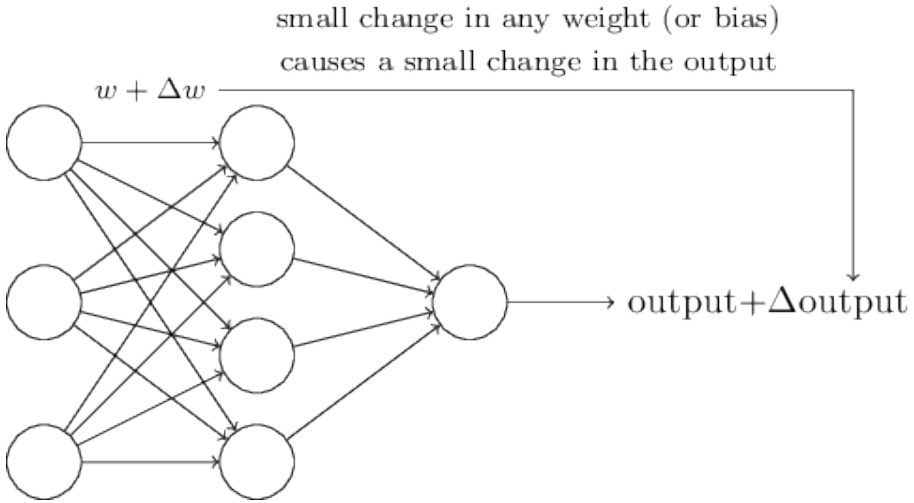


Neural Networks

1. Suppose that we know how small change in weights changes the output.

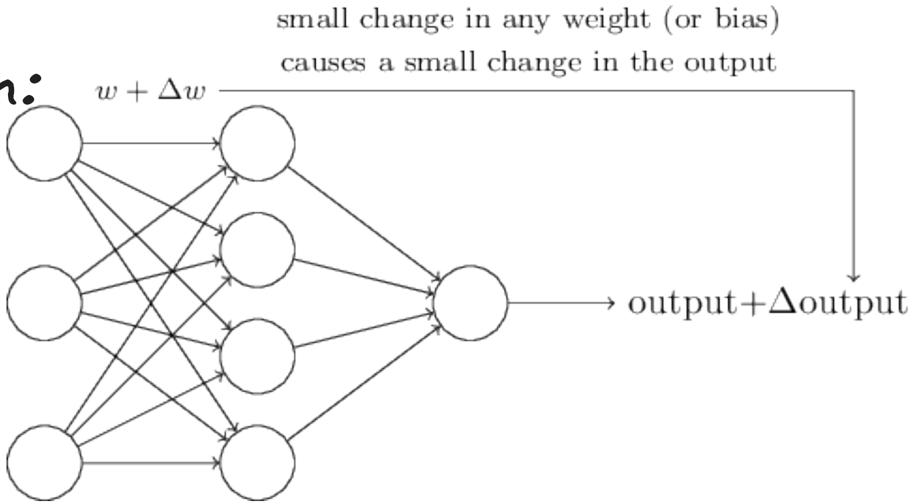
2. Starting with random initialization we can iteratively update (supervise) the weights with small changes to bring the output to expected value.

3. We control the learning process, by testing and validating it with data that were not used while weight optimization.



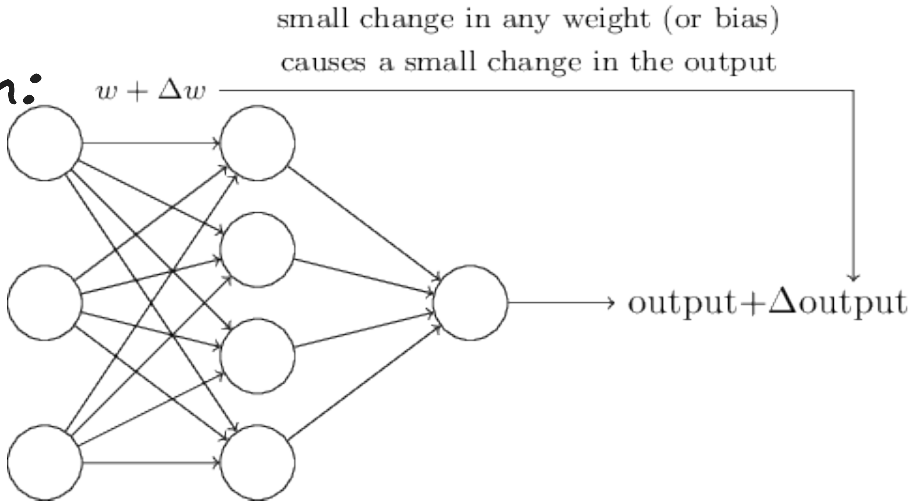
Problem with Perceptron:

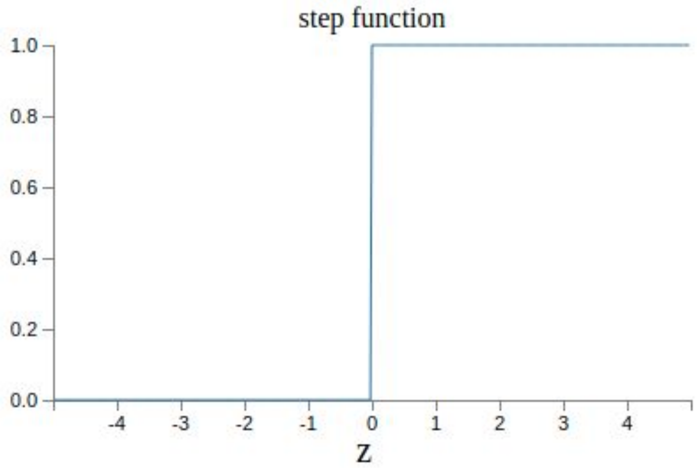
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



Problem with Perceptron:

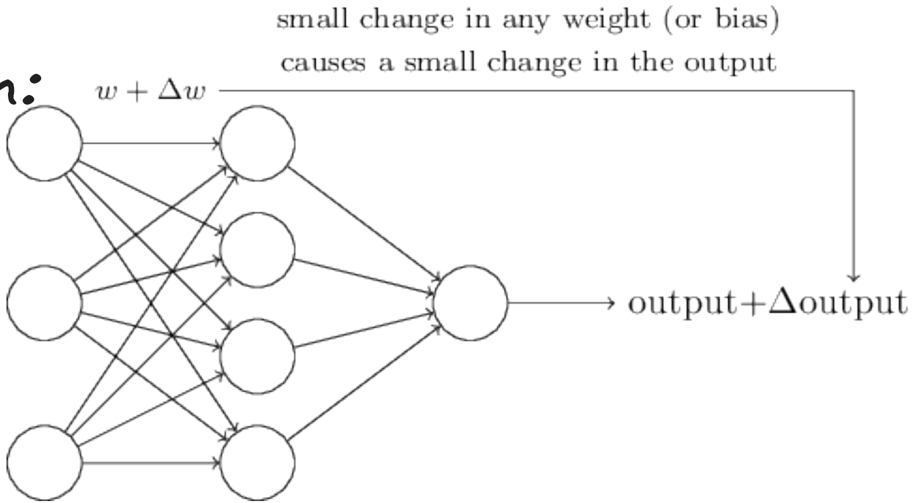
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

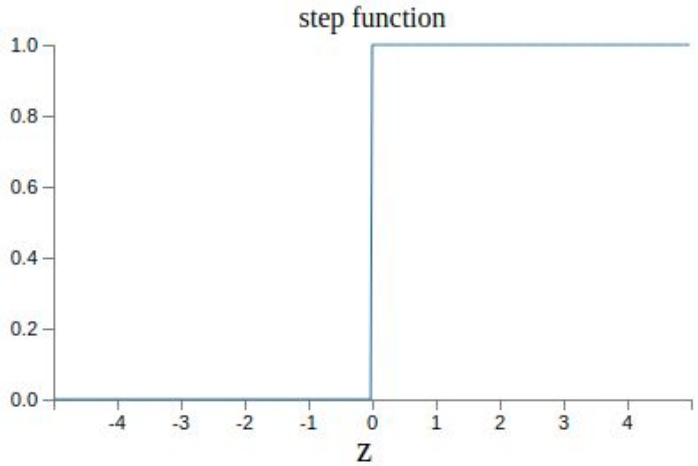


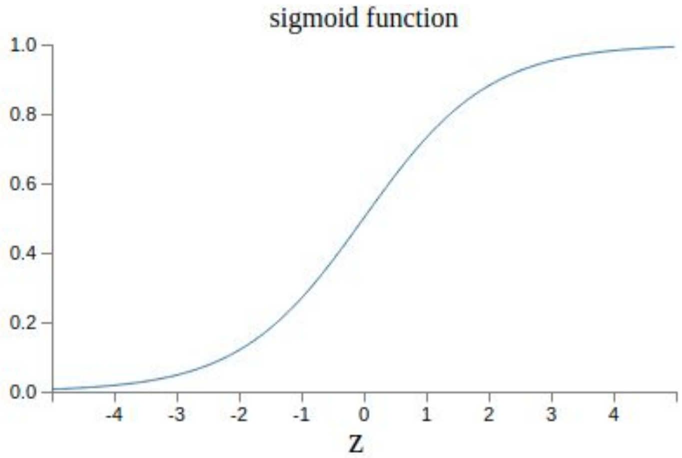


Problem with Perceptron:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \le 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

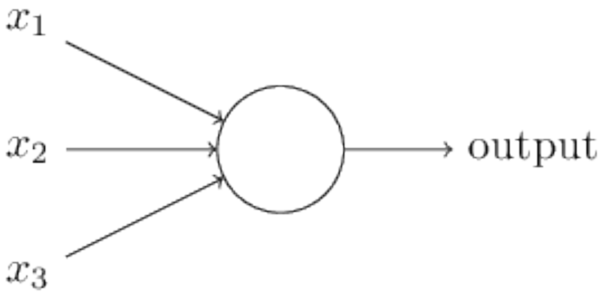






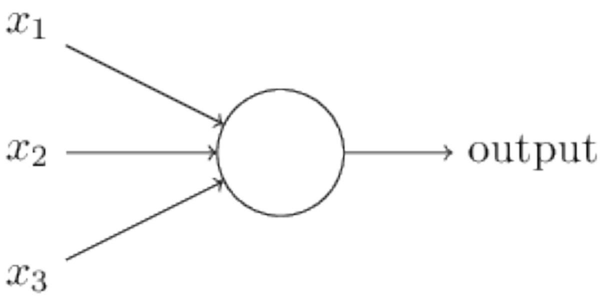
Sigmoid Neuron

A ~~perceptron~~ sigmoid neuron takes multiple inputs (e.g.: x1, x2, x3) and produces a single ~~binary~~ output.



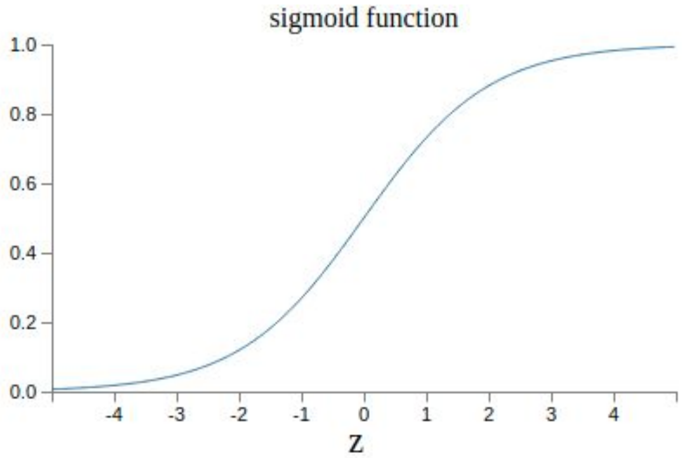
Sigmoid Neuron

A ~~perceptron~~ sigmoid neuron takes multiple inputs (e.g.: x1, x2, x3) and produces a single ~~binary~~ output.



$$z = w \cdot x + b$$

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



First order Taylor approximation (Quick Lookup)

Let's say we have function f, and value c,
for which function output value of f(c) is known.

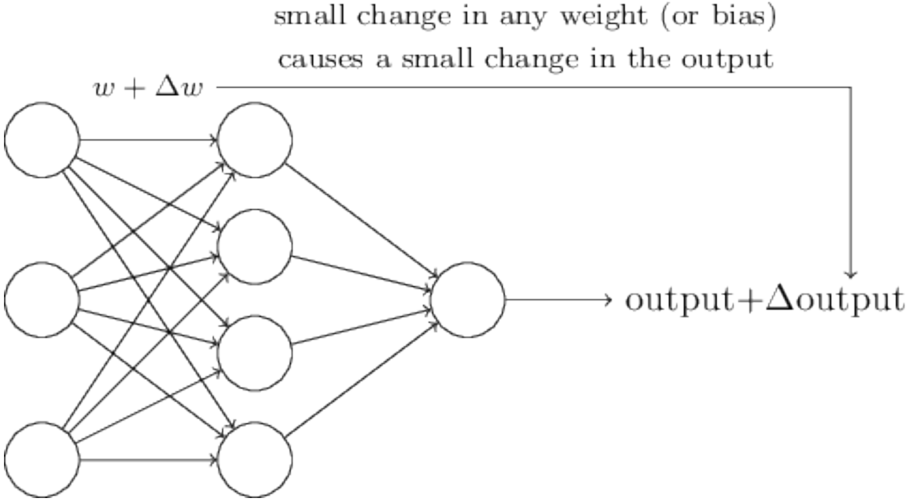
For x in neighborhood of c, output value f(x) can be approximated as:

$$f(x) \approx f'(c)(x-c)+f(c)$$

$$f(x) - f(c) \approx f'(c)(x-c)$$

$$\Delta f \approx f'(c)\ \Delta x$$

Neural Networks:

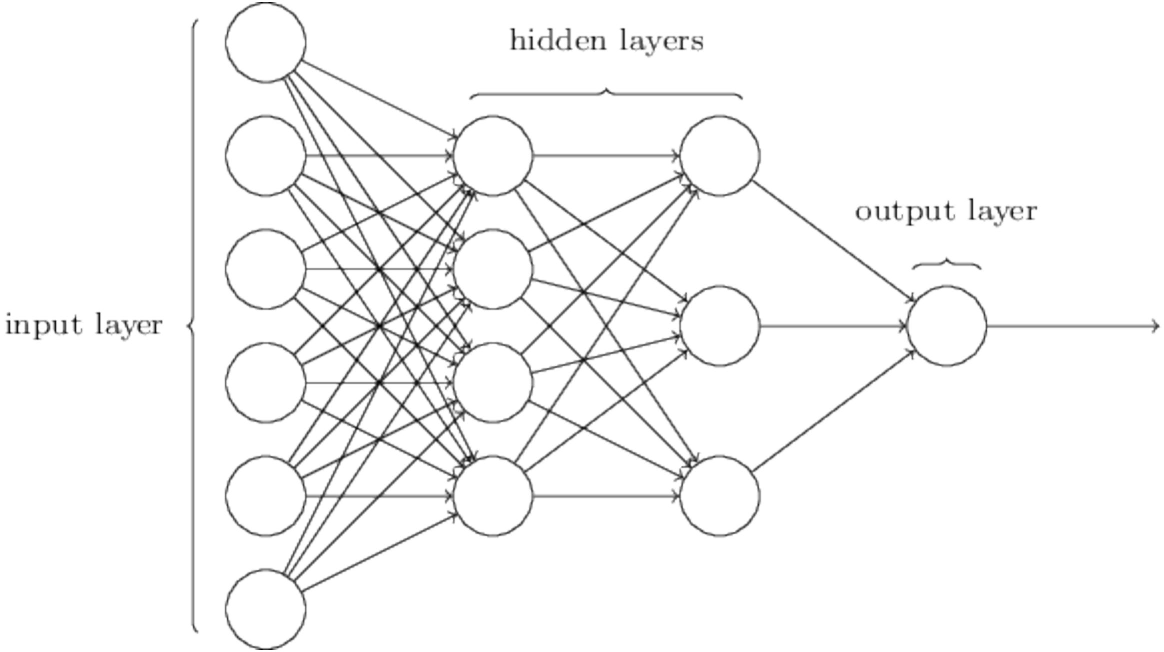


$$\Delta \text{output} \approx \sum_j \frac{\partial\, \text{output}}{\partial w_j} \Delta w_j + \frac{\partial\, \text{output}}{\partial b} \Delta b$$

Feedforward Network architecture

3 layers:

- Input
- Hidden
- Output



Recognizing Digits with Neural Nets.

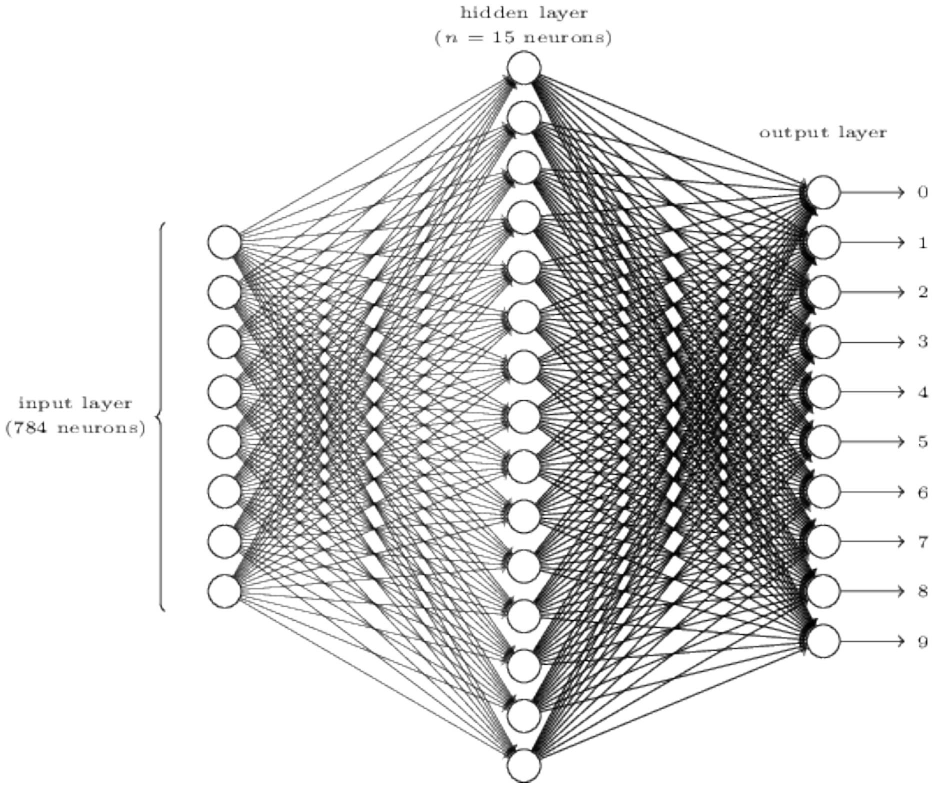
MNIST Data:

    28x28 images

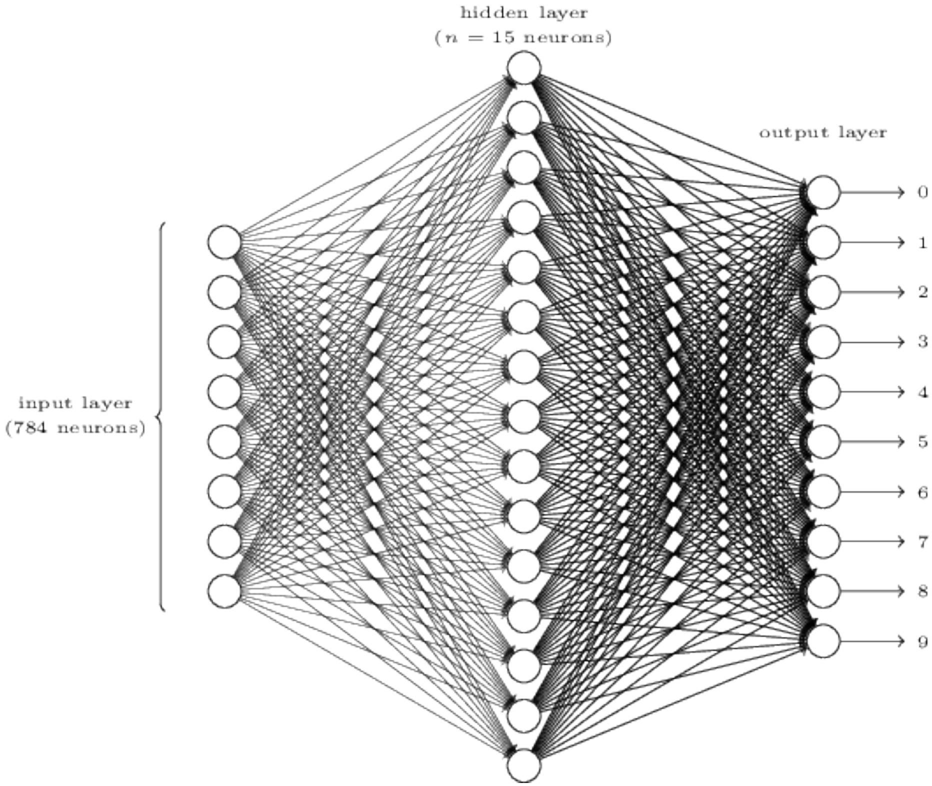
    Greyscale (single channel)

Goal: classifying/recognizing images.

Recognizing Digits with Neural Nets.



Recognizing Digits with Neural Nets.

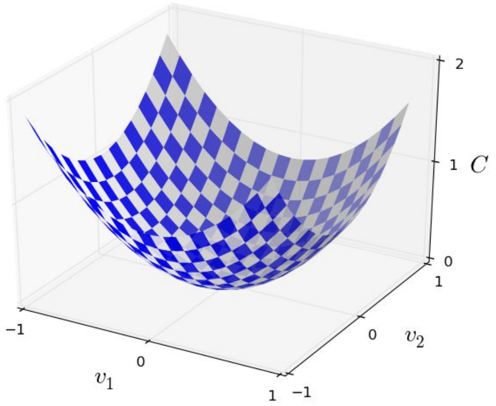


For x representing digit 6:

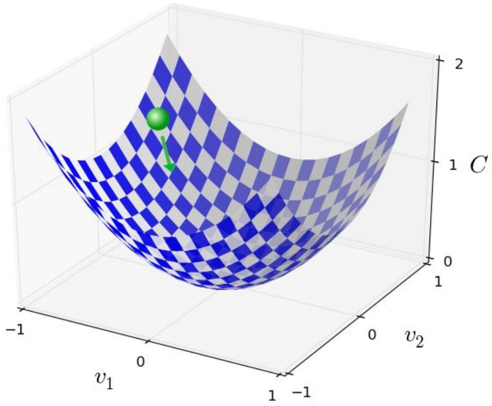
$$y(x) = (0,0,0,0,0,0,1,0,0,0)^T$$

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

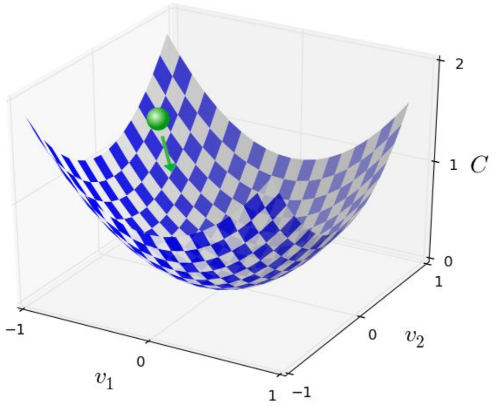
Learning with gradient descent



Learning with gradient descent

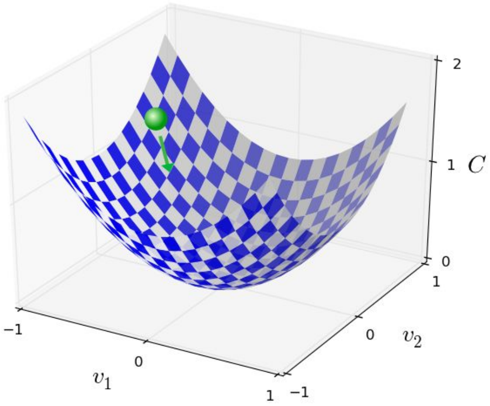


Learning with gradient descent



$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

Learning with gradient descent

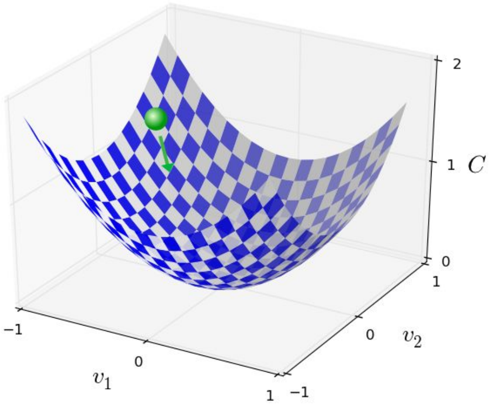


$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

$$\Delta C \approx \nabla C \cdot \Delta v$$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$$

Learning with gradient descent



$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$$

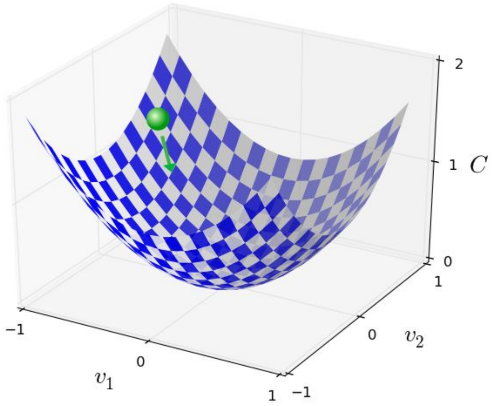
$$\Delta C \approx \nabla C \cdot \Delta v$$

$$\Delta v = -\eta \nabla C$$

$$v \rightarrow v' = v - \eta \nabla C$$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$$

Learning with gradient descent



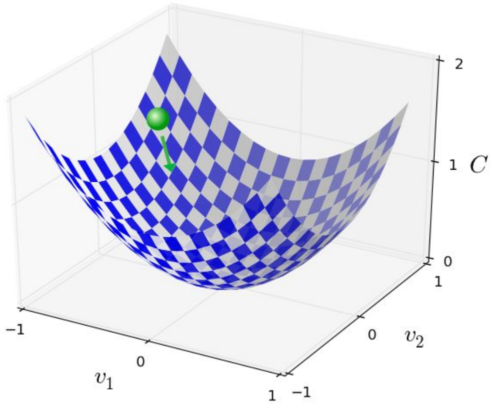
$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T$$

$$\Delta C \approx \nabla C \cdot \Delta v$$

$$\Delta v = -\eta \nabla C$$

$$v \to v' = v - \eta \nabla C$$

Learning with gradient descent



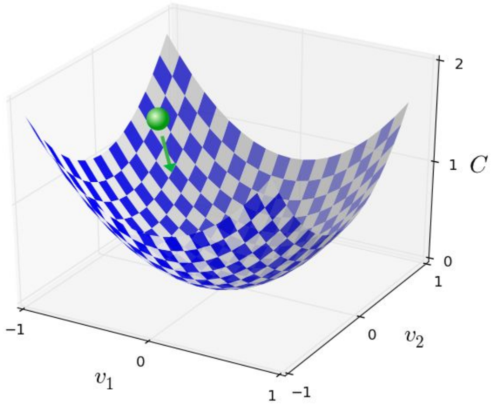
$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T$$

$$\Delta C \approx \nabla C \cdot \Delta v$$

$$\Delta v = -\eta \nabla C$$

$$v \rightarrow v' = v - \eta \nabla C$$

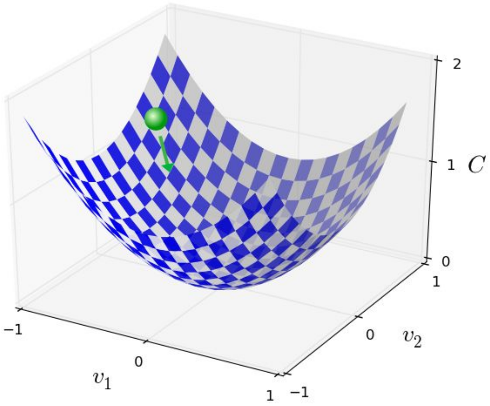
NN with gradient descent



$$w_k \to w'_k = w_k - \eta \frac{\partial C}{\partial w_k}$$

$$b_l \to b'_l = b_l - \eta \frac{\partial C}{\partial b_l}.$$

Stochastic Gradient Descent



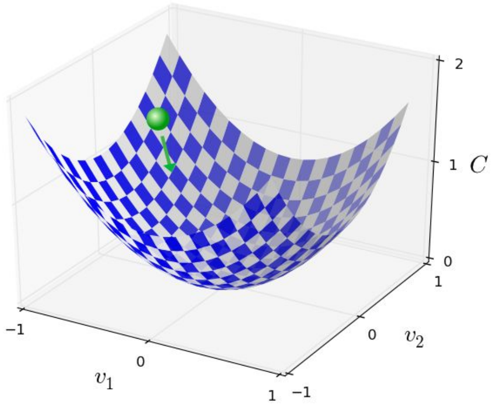
$$C_x \equiv \frac{\|y(x)-a\|^2}{2}$$

$$C = \frac{1}{n}\sum_x C_x$$

$$\nabla C = \frac{1}{n}\sum_x \nabla C_x$$

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

Stochastic Gradient Descent



$$C_x \equiv \frac{\|y(x)-a\|^2}{2}$$

$$C = \frac{1}{n}\sum_x C_x$$

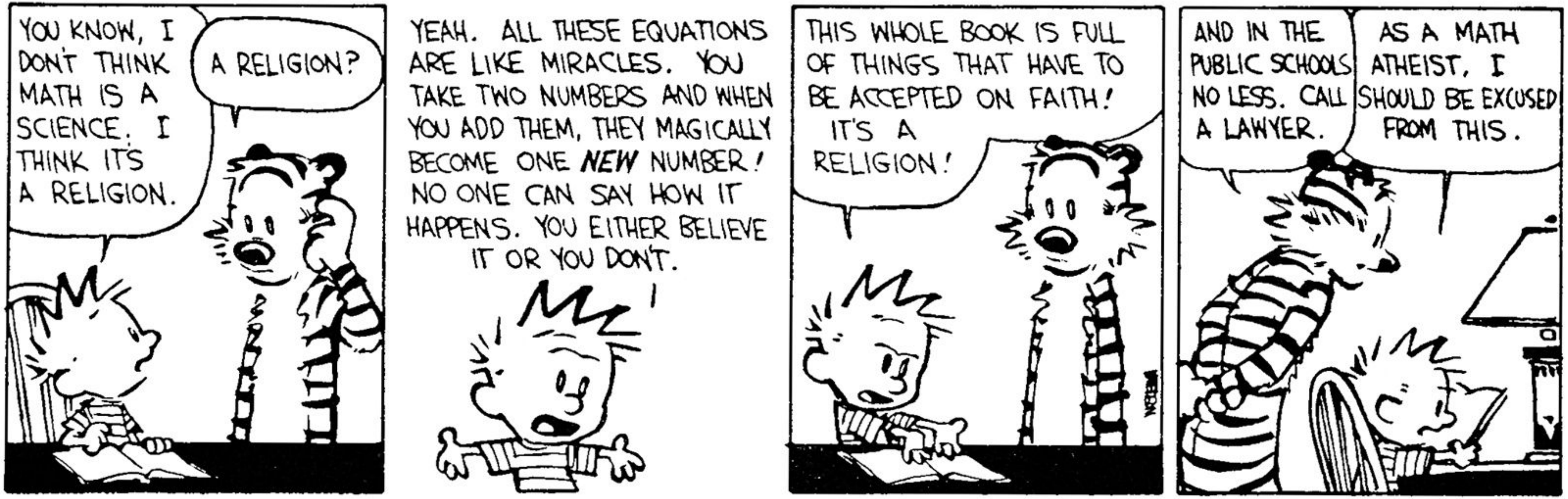
$$\nabla C = \frac{1}{n}\sum_x \nabla C_x$$

$$\frac{\sum_{j=1}^{m} \nabla C_{X_j}}{m} \approx \frac{\sum_x \nabla C_x}{n} = \nabla C$$

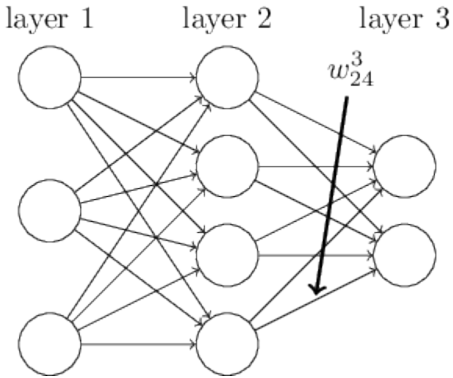
$$w_k \rightarrow w_k' = w_k - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b_l' = b_l - \frac{\eta}{m}\sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

BackPropagation

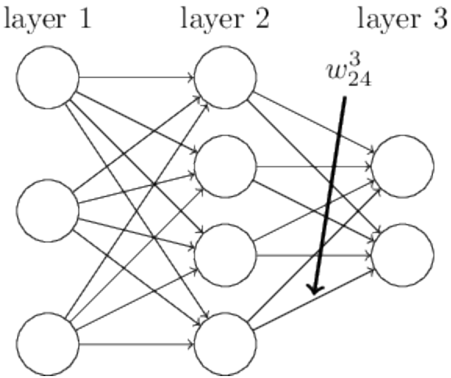


BackPropagation: Notation

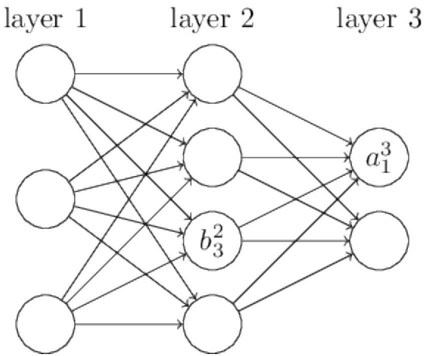


w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

BackPropagation: Notation

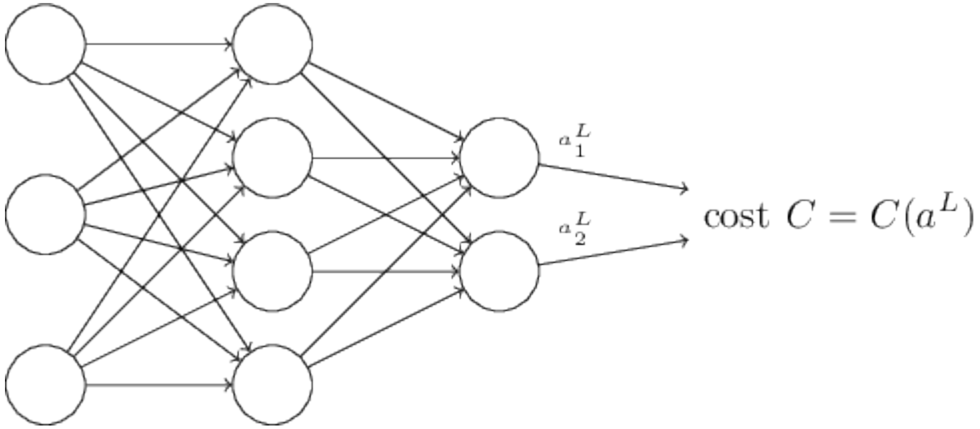


w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer



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

BackPropagation: Cost function

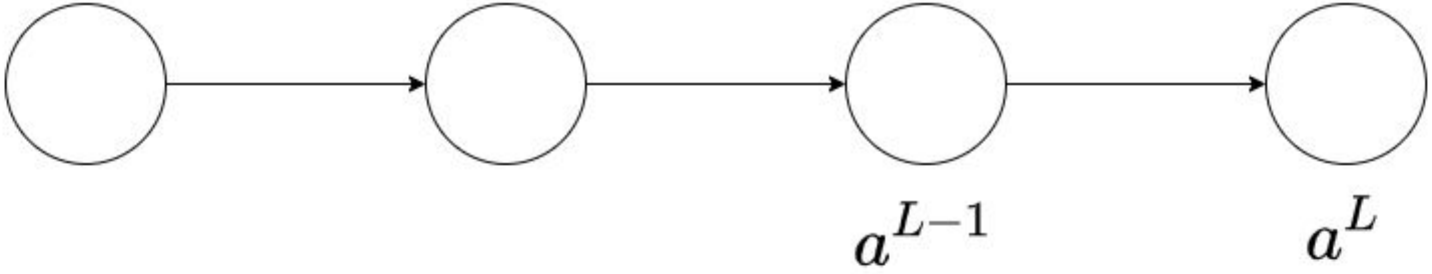


$$C = \frac{1}{2} \|y - a^L\|^2 = \frac{1}{2} \sum_j (y_j - a_j^L)^2$$

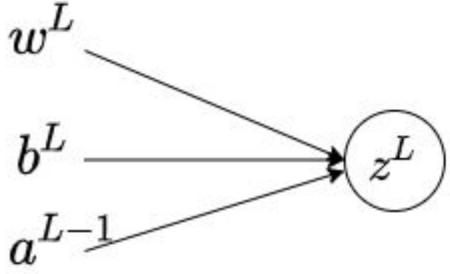
BackPropagation

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

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

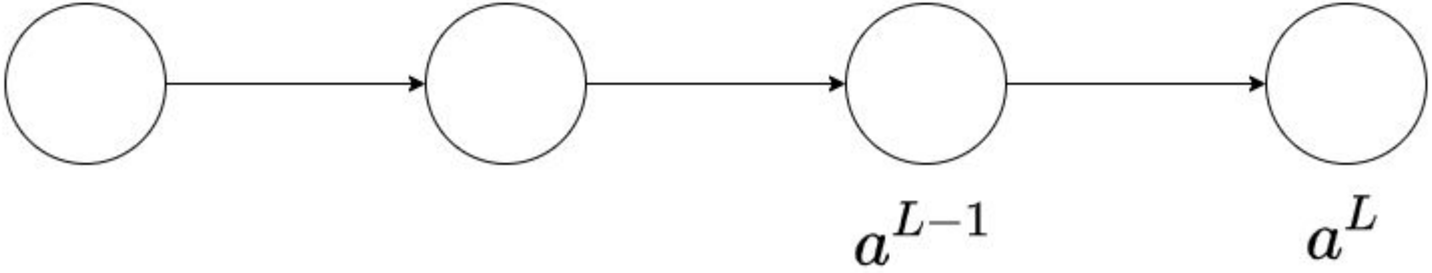


BackPropagation

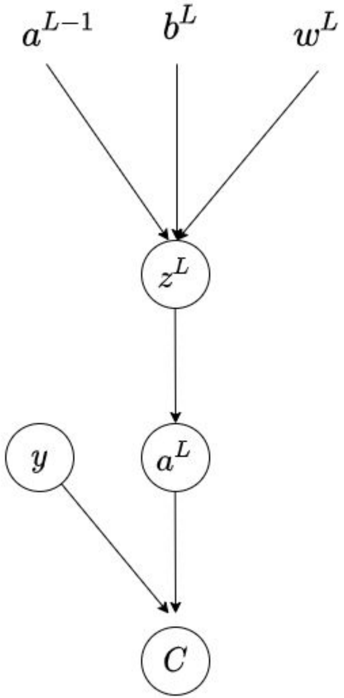


$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

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

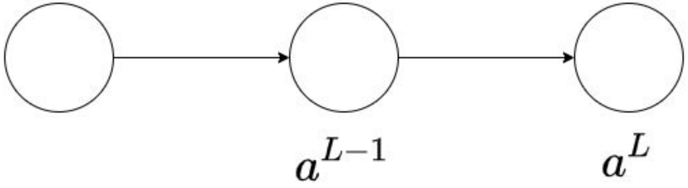


BackPropagation

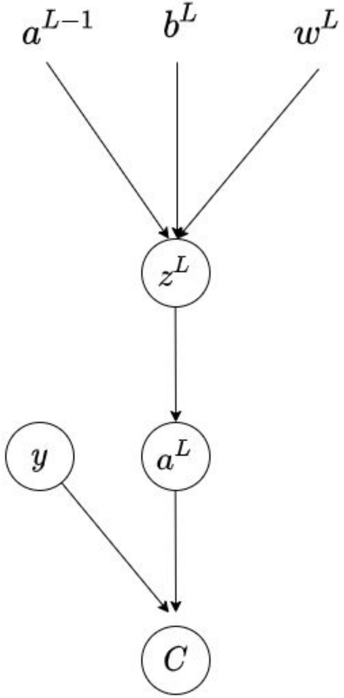


$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

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

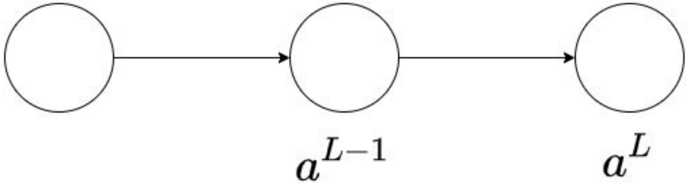


BackPropagation



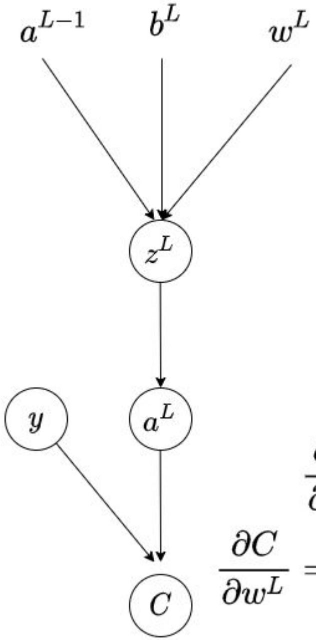
$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

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



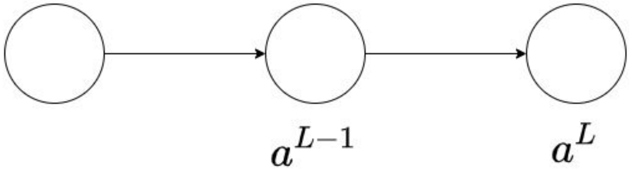
$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$
$$\frac{\partial C}{\partial z^L} = \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L} = \sigma'(z^L) \cdot 2(a^L - y)$$

BackPropagation



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

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

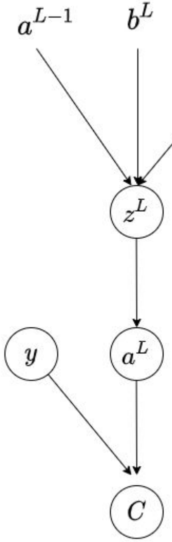


$$\frac{\partial C}{\partial a^L} = 2(a^L - y)$$

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

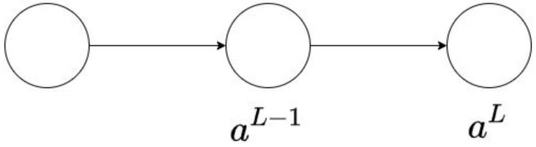
$$\frac{\partial C}{\partial w^L} = \frac{\partial z^L}{\partial w^L} \frac{\partial a^L}{\partial z^L} \frac{\partial C}{\partial a^L} = a^{(L-1)} \sigma'(z^L) \cdot 2(a^L - y)$$

BackPropagation



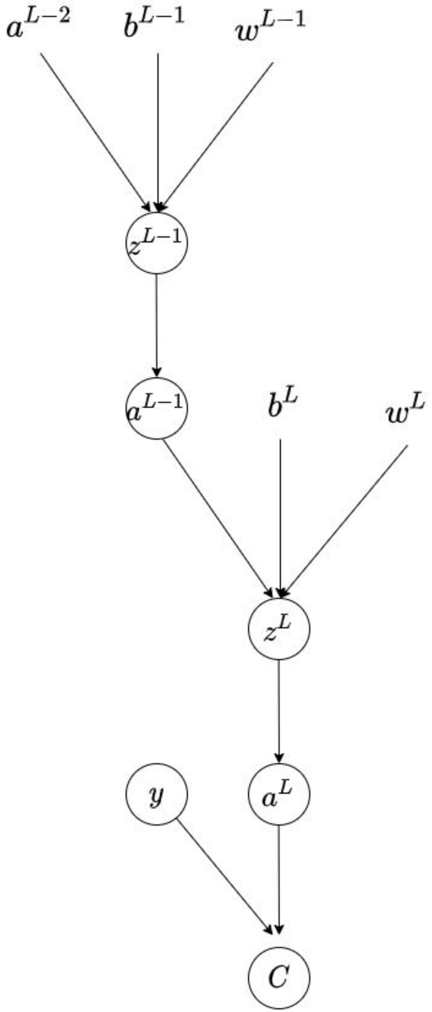
$$z^L = w^L \cdot a^{L-1} + b^L$$
$$a^L = \sigma(z^L)$$

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



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

BackPropagation



BackPropagation

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

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

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

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