Neural Networks

Intelligent Systems and Control

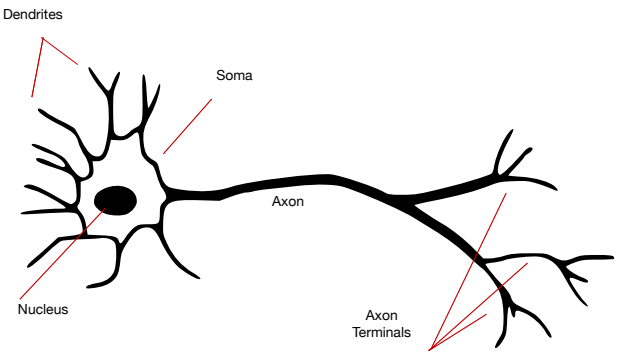
2019

Sepehr Maleki

University of Lincoln
School of Engineering

A Typical Cortical Neuron

- Physical structure:
  - There is one axon that branches.
  - There is a dendritic tree that collects input from other neurons.



- Axons typically contact dendritic trees at synapses.
  - A spike of activity in the axon causes charge to be injected into the post-synaptic neuron.
- Spike generation:
  - There is an axon hillock that generates outgoing spikes whenever enough charge has flowed in at synapses to depolarise the cell membrane.

(S. Maleki 2019)

How Brain Works!

Each neuron receives inputs from other neurons

- A few neurons also connect to receptors.
- Cortical neurons use spikes to communicate.

The effect of each input line on the neuron is controlled by a synaptic weight.

- The weights can be positive or negative.

The synaptic weights adapt so that the whole network learns to perform useful computations.

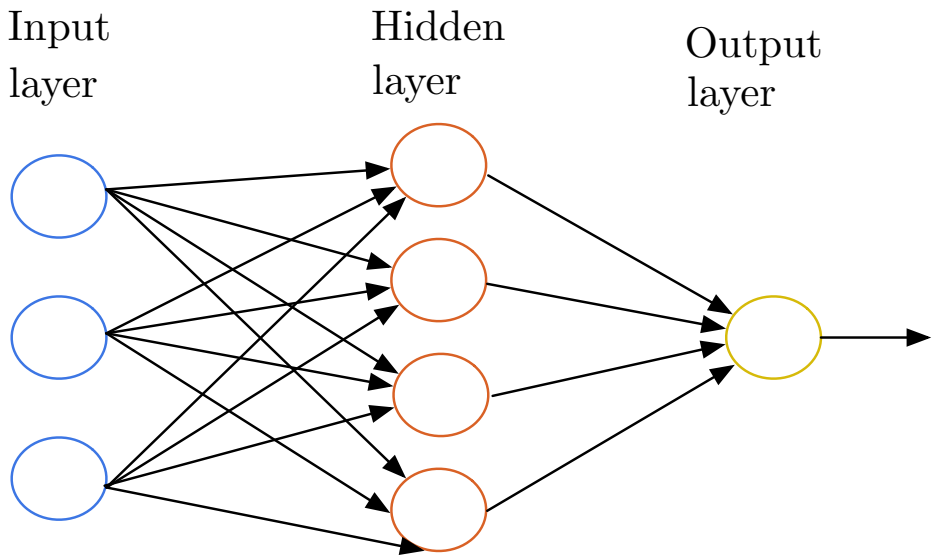
- Recognising objects, understanding language, making plans, controlling the body.

You have about 10^{11} neurons each with about 10^4 weights.

(S. Maleki 2019)

Feed-Forward Neural Networks

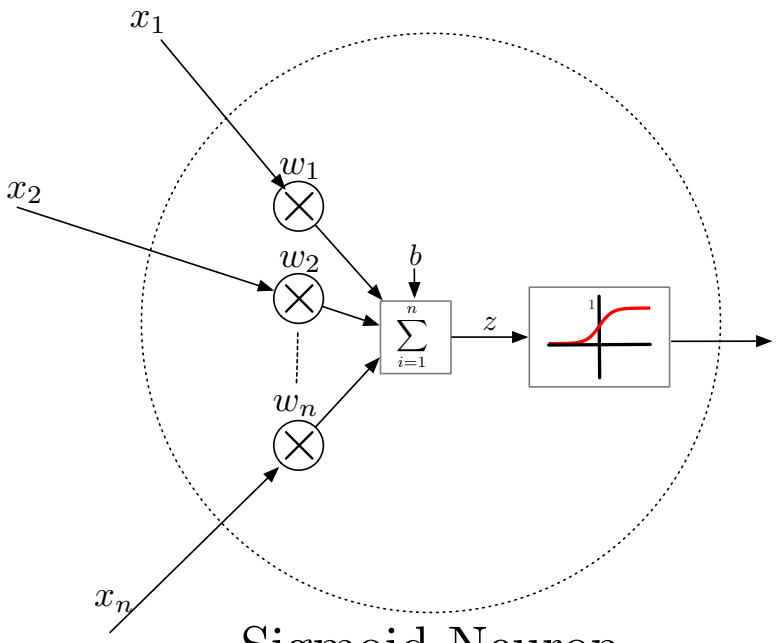
These are the most common type of neural network in practical applications.



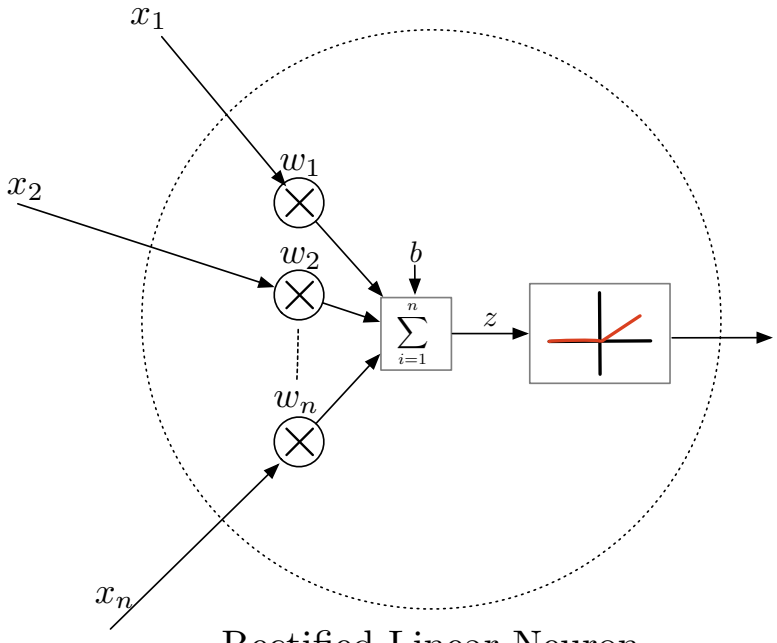
- The first layer is the input and the last is the output.
- If there is more than one hidden layer, we call them “deep” neural networks.

(S. Maleki 2019)

Hidden Units



Sigmoid Neuron



Rectified Linear Neuron

$$a = \frac{1}{1 + e^{-z}}$$

$$a = \begin{cases} z & \text{if} z > 0 \\ 0 & \text{otherwise} \end{cases}$$

(S. Maleki 2019)

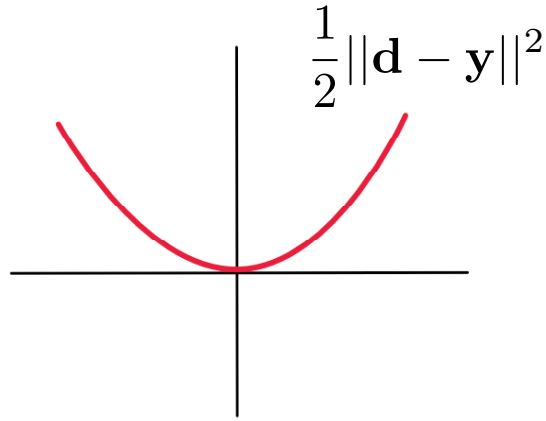
Neural Network is a Function Approximator

Consider a matrix of n training inputs $\mathbf{X} \in \mathbb{R}^{n \times d}$ is processed by some function $g(\mathbf{X})$ and returns a desired output $\mathbf{d}$. That is:

$$\mathbf{d} = g(\mathbf{X})$$

We approximate $g(\mathbf{X})$ by $f(\mathbf{X}, \mathbf{w}, \mathbf{b}) = \mathbf{w}^\top \mathbf{X} + \mathbf{b}$. To evaluate the performance of our approximation we construct the error e:

$$e = \frac{1}{2}||\mathbf{d} - \mathbf{y}||^2 = \frac{1}{2}\sum_{i=1}^{N}(d_i - y_i)^2$$



(S. Maleki 2019)

Fish and Chips Example

Each day you get lunch at a cafeteria that only sells fish, chips and ketchup.

- Every time, you order several portion of each.
- The cashier only tells you the total price of the meal.
- After several days, you want to figure out the price of each portion.

Approach: Start with random guesses for the prices and then adjust them to get a better fit to the observed prices of whole meals.

(S. Maleki 2019)

Solving The Fish and Chips Problem

Each meal price gives a linear constraint on the prices of the portions:

$$price = x_{fish} \times w_{fish} + x_{chips} \times w_{chips} + x_{ketchup} \times w_{ketchup}$$

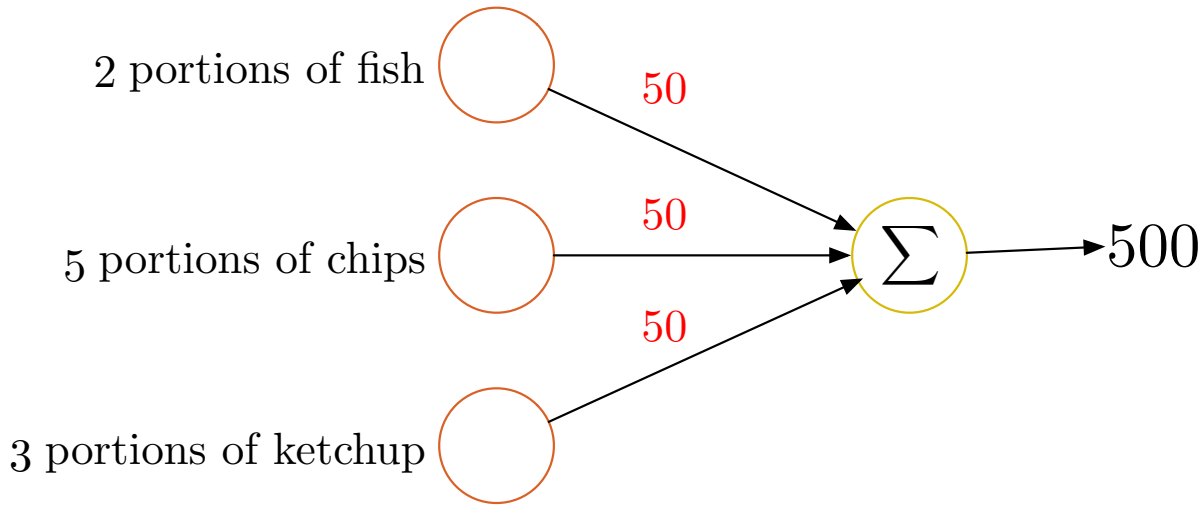
The prices of the portions are like weights of a linear neuron:

$$\mathbf{w} = \begin{bmatrix} w_{fish} \\ w_{chips} \\ w_{ketchup} \end{bmatrix} .$$

We will start with guesses for the weights and then adjust them slightly to give a better fit to the prices given by the cashier.

Solving The Fish and Chips Problem

The cashier tells the price of the meal is 850. We start with random guesses about the individual prices (weights).



Our initial guesses gives us a residual error of $850 - 500 = 350$.

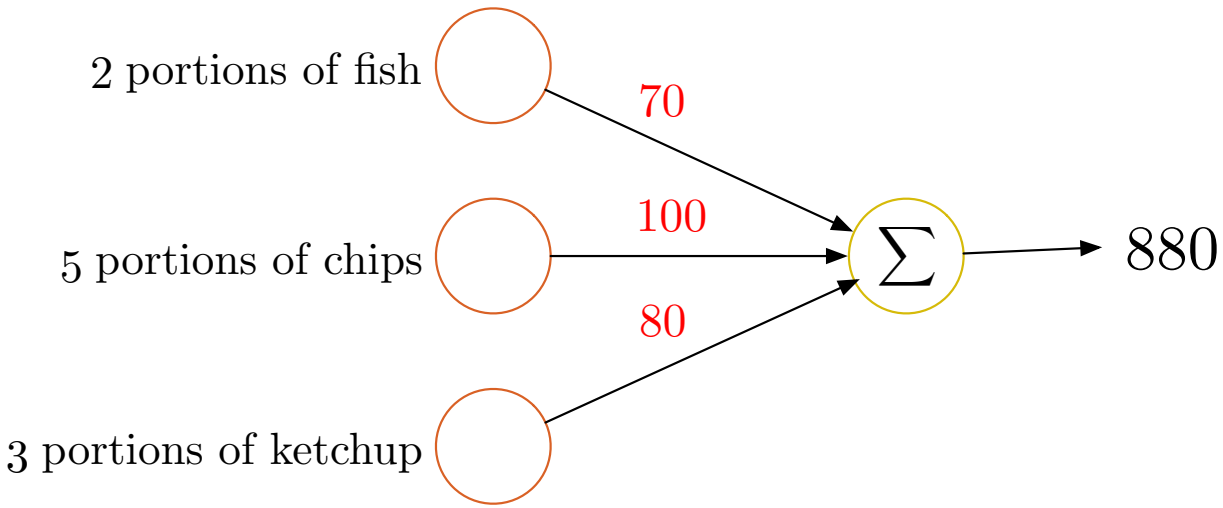
(S. Maleki 2019)

Adjusting The Weights

We use the "delta-rule" to adjust the weights:

$$\Delta w_i = \alpha \, x_i (d - y) \; .$$

Considering a learning rate of $\alpha = \frac{1}{35}$, the weight changes are: $+20, +50, +30$. This results in the new weights of: $70, 100, 80$.



This gives us a residual error of $850 - 880 = -30$.

Deriving The Delta-Rule

We start with the error:

$$e = \frac{1}{2}\sum_{i=1}^{N}(d^{(i)} - y^{(i)})^2 \ .$$

Now differentiate the error to get derivatives for weights:

$$\frac{\partial e}{\partial w_j} = \frac{1}{2}\sum_{i=1}^{N}\frac{\partial y^{(i)}}{\partial w_j}\,\frac{de^{(i)}}{dy^{(i)}} = -\sum_{i=1}^{N} x_j^{(i)}(d^{(i)} - y^{(i)}) \ .$$

Therefore:

$$\Delta w_j = -\alpha\frac{\partial e}{\partial w_j} = \sum_{i=1}^{N}\alpha\, x_j^{(i)}(d^{(i)} - y^{(i)}) \ .$$

Derivatives of a Logistic Neuron

The derivatives of the logit, z with respect to the inputs and weights are very simple:

$$z = b + \sum_{j=1}^{d} x_j w_j$$

$$\frac{\partial z}{\partial w_j} = x_j \ , \qquad \frac{\partial z}{\partial x_j} = w_j \ .$$

The derivative of the output with respect to the logit is also simple:

$$y = \frac{1}{1+e^{-z}} \longrightarrow \frac{dy}{dz} = y(1-y) \ .$$

Derivative of The Logistic Function

The logistic function is given by:

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

Then:

$$\frac{dy}{dz} = \frac{e^{-z}}{(1+e^{-z})^2} = \left(\frac{1}{1+e^{-z}}\right)\left(\frac{e^{-z}}{1+e^{-z}}\right) = y(1-y) \ .$$

Because:

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

(S. Maleki 2019)

Learning The Weights of a Logistic Unit

To learn the weights we need the derivative of the output with respect to each weight:

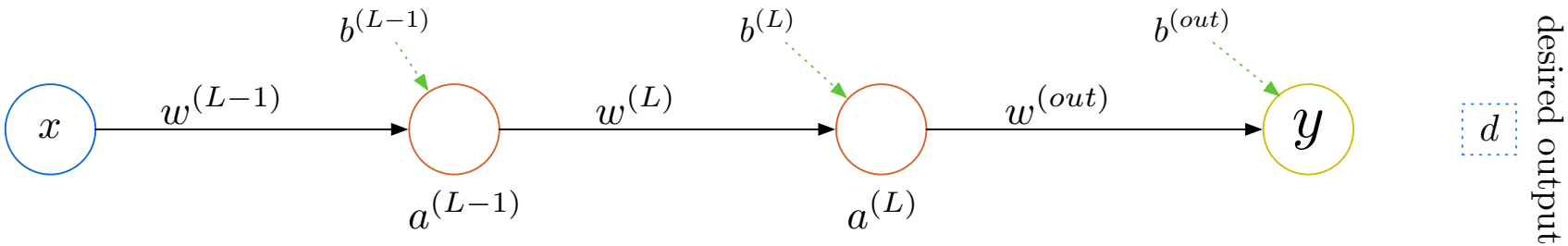
$$\frac{\partial y}{\partial w_j} = \frac{\partial z}{\partial w_j}\,\frac{dy}{dz} = x^{(j)}\; y(1-y)\;.$$

$$\frac{\partial e}{\partial w_j} = \sum_{i=1}^{N} \frac{\partial y_i}{\partial w_j}\,\frac{\partial e}{\partial y_i} = -\sum_{i=1}^{N} x_i^{(j)} y_i(1-y_i)(d_i-y_i)\;.$$

Therefore:

$$\Delta w_j = -\alpha \frac{\partial e}{\partial w_j} = \sum_{i=1}^{N} \alpha\; x_i^{(j)} y_i(1-y_i)(d_i-y_i)\;.$$

Backpropagation



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

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

$$y = \sigma(\underbrace{w^{(out)} a^{(L)} + b^{(out)}}_{z^{(out)}})$$

$$e_1 = \frac{1}{2}(y - d)^2$$

We want to find how sensitive the error (cost function) is, to small changes in $w^{(out)}$. That is:

$$\frac{\partial e_1}{\partial w^{(out)}}$$

(S. Maleki 2019)

Backpropagation

$$\frac{\partial e_1}{\partial w^{(out)}} = \frac{\partial z^{(out)}}{\partial w^{(out)}} \frac{\partial y}{\partial z^{(out)}} \frac{\partial e_1}{\partial y} = a^{(L)} \sigma'(z^{(out)}) (y - d)$$

Since:

$$\frac{\partial e_1}{\partial y} = (y - d)$$
$$\frac{\partial y}{\partial z^{(out)}} = \sigma'(z^{(out)})$$
$$\frac{\partial z^{(out)}}{\partial w^{(out)}} = a^{(L)}$$

Therefore:

$$\frac{\partial e_1}{\partial w^{(out)}} = \frac{\partial z^{(out)}}{\partial w^{(out)}} \frac{\partial y}{\partial z^{(out)}} \frac{\partial e_1}{\partial y} = a^{(L)} \sigma'(z^{(out)}) (y - d)$$

(S. Maleki 2019)

Backpropagation

The sensitivity to the bias term can be obtained analogously:

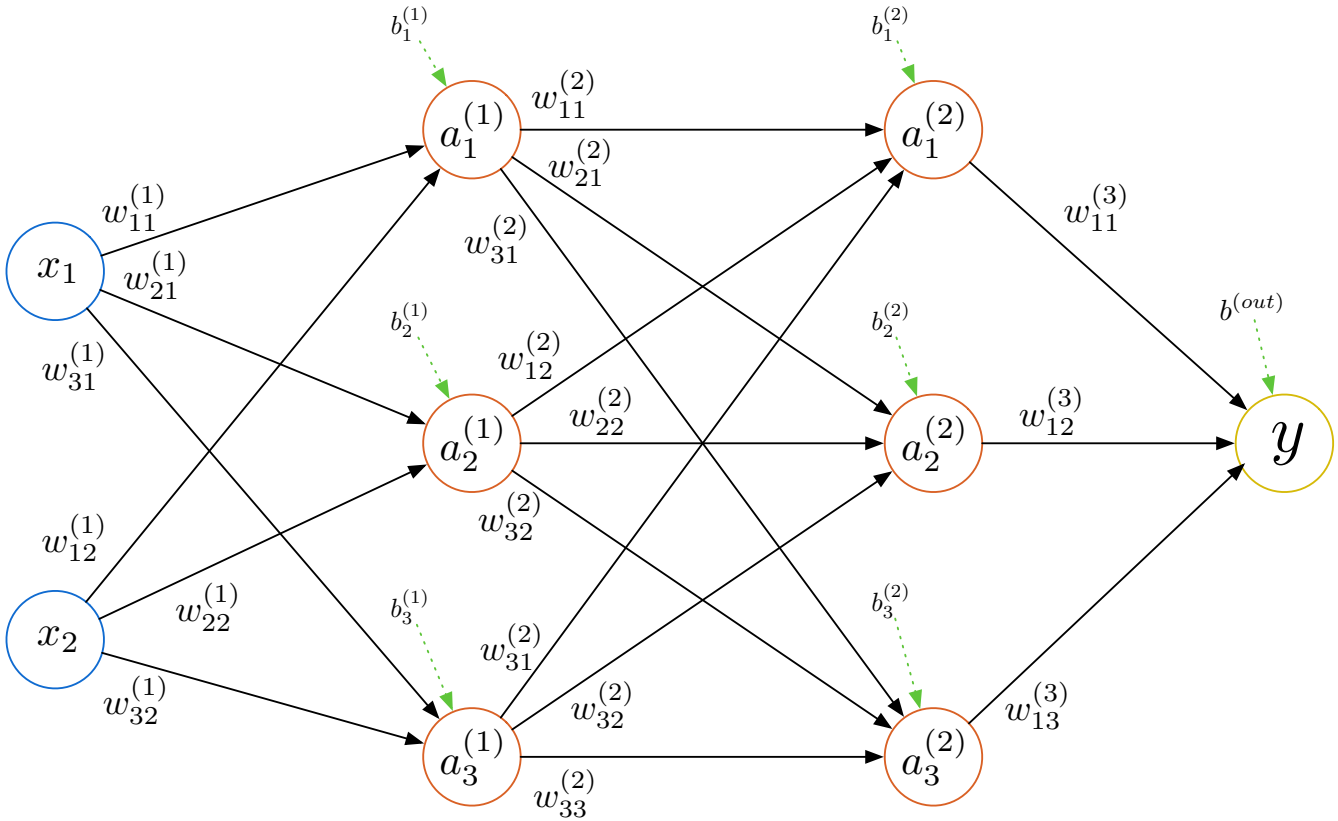
$$\frac{\partial e_1}{\partial b^{(out)}} = \frac{\partial z^{(out)}}{\partial b^{(out)}} \frac{\partial y}{\partial z^{(out)}} \frac{\partial e_1}{\partial y} = \sigma'(z^{(out)}) \, (y - d) \, .$$

If we have more than one training example, then the derivative of the full error (cost function) is given by:

$$\frac{\partial e}{\partial w^{(out)}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial e_i}{\partial w^{(out)}} \, ,$$

$$\frac{\partial e}{\partial b^{(out)}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial e_i}{\partial b^{(out)}} \, .$$

Deep/Wide Neural Networks



$$z_j^{(L)} = \sum_k w_{jk}^{(L)} a_k^{(L-1)} + b_j^{(L)} \ , \qquad a_j^{(L)} = \sigma(z_j^{(L)})$$

(S. Maleki 2019)