Fitting Neural Networks

Cost Function

$$J(\theta) = \mathbb{E}_{x,y} L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} L\left(x^{(i)}, y^{(i)}, \theta\right)$$

J is referred to as the "Cost Function"

L is the single point "Loss Function"

$$J(\theta) = \mathbb{E}_{x,y} L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} -\log\left(p(y|\, x; \theta)\right)$$

Stochastic Gradient Descent Algorithm

Repeat

$$\theta^{n+1} = \theta^{n} - \epsilon \nabla_{\theta} J(\theta)$$

In plain Engligh: take a small step (ϵ, known as the “learning rate”) in the direction that decreases the cost function the most.

$$\nabla_{\theta} J(\theta) = \boldsymbol{g} = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\theta} L(x^{(i)}, y^{(i)}, \theta)$$

Stochastic Gradient Descent Algorithm

$$\boldsymbol{g} = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$$

Since this is an expectation, we only need to operate on "batches"

$$\boldsymbol{g} = \frac{1}{m'} \nabla_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta)$$

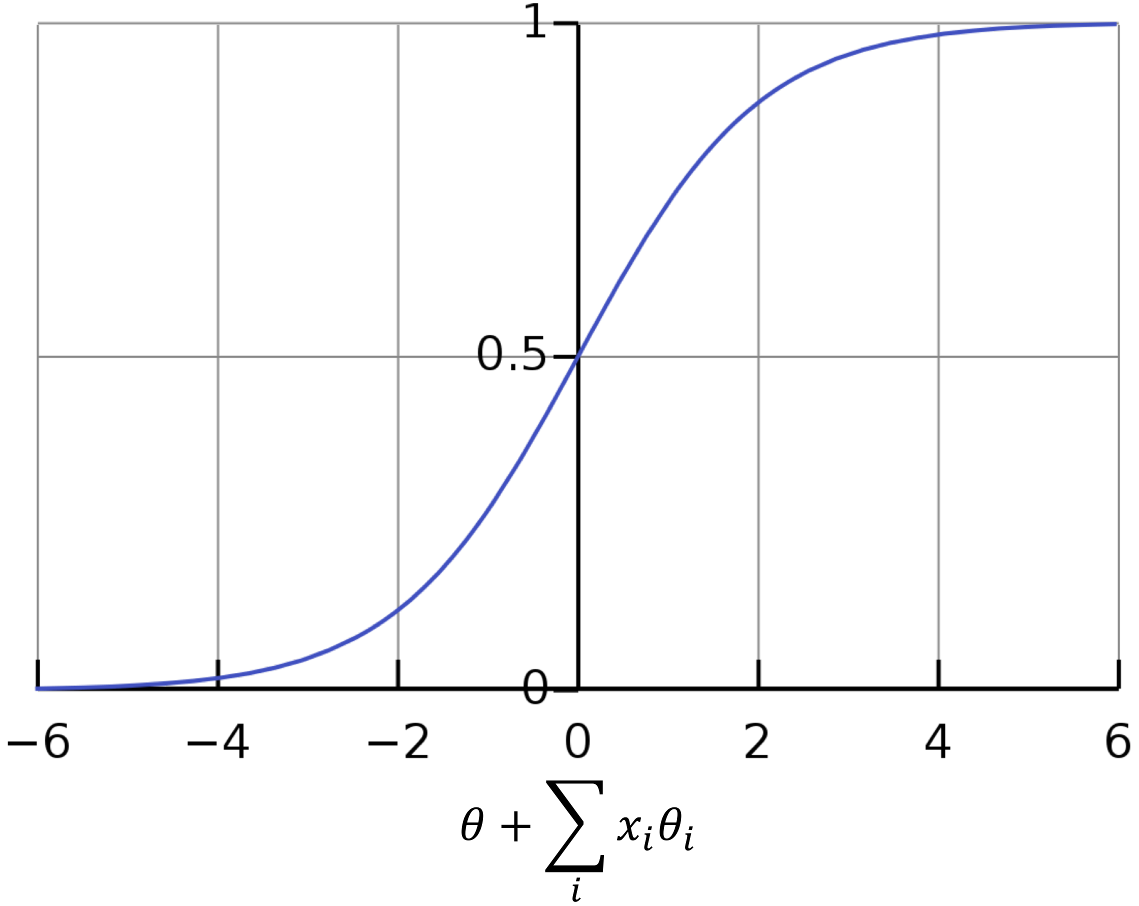
Where m' is a small subset of our training data

Example: Logistic Regression

- Learn $p(y \mid x; \theta)$ as a linear function

$$p(y \mid x; \theta) = \frac{1}{1 + \exp(\theta_0 + \sum_i x_i \theta_i)}$$

- Allows for separation of data into two classes – 0 and 1
- Logistic Regression prediction equates to the likelihood the sample is in class 1
  - Likelihood of class 0 is 1-$p(y = 1|x; \theta)$



Logistic Regression

$$p(y = 1 \mid x; \theta) = \frac{1}{1 + \exp(\theta_0 + \sum_i x_i \theta_i)}$$

$$L(x, y, \theta) = y \log p(y = 0 \mid x; \theta) + (1 - y) \log(p(y = 1 \mid x ; \theta))$$

$$L(x, y, \theta) = y \log \frac{\exp(\theta_0 + \sum_i x_i \theta_i)}{1 + \exp(\theta_0 + \sum_i x_i \theta_i)} + (1 - y) \log\left(\frac{1}{1 + \exp(\theta_0 + \sum_i x_i \theta_i)}\right)$$

Logistic Regression

$$L(x, y, \theta) = y\log\frac{\exp(\theta_0 + \sum_i x_i\theta_i)}{1 + \exp(\theta_0 + \sum_i x_i\theta_i)} + (1 - y)\log\left(\frac{1}{1 + \exp(\theta_0 + \sum_i x_i\theta_i)}\right)$$

$$L(x, y, \theta) \\ = y(\theta_0 + \sum_i x_i\theta_i) - y\log\left(1 + \exp(\theta_0 + \sum_i x_i\theta_i)\right) - \log\left(1 + \exp\left(\theta_0 + \sum_i x_i\theta_i\right)\right) + y\log\left(1 + \exp\left(\theta_0 + \sum_i x_i\theta_i\right)\right)$$

$$L(x, y, \theta) = y(\theta_0 + \sum_i x_i\theta_i) - \log\left(1 + \exp\left(\theta_0 + \sum_i x_i\theta_i\right)\right)$$

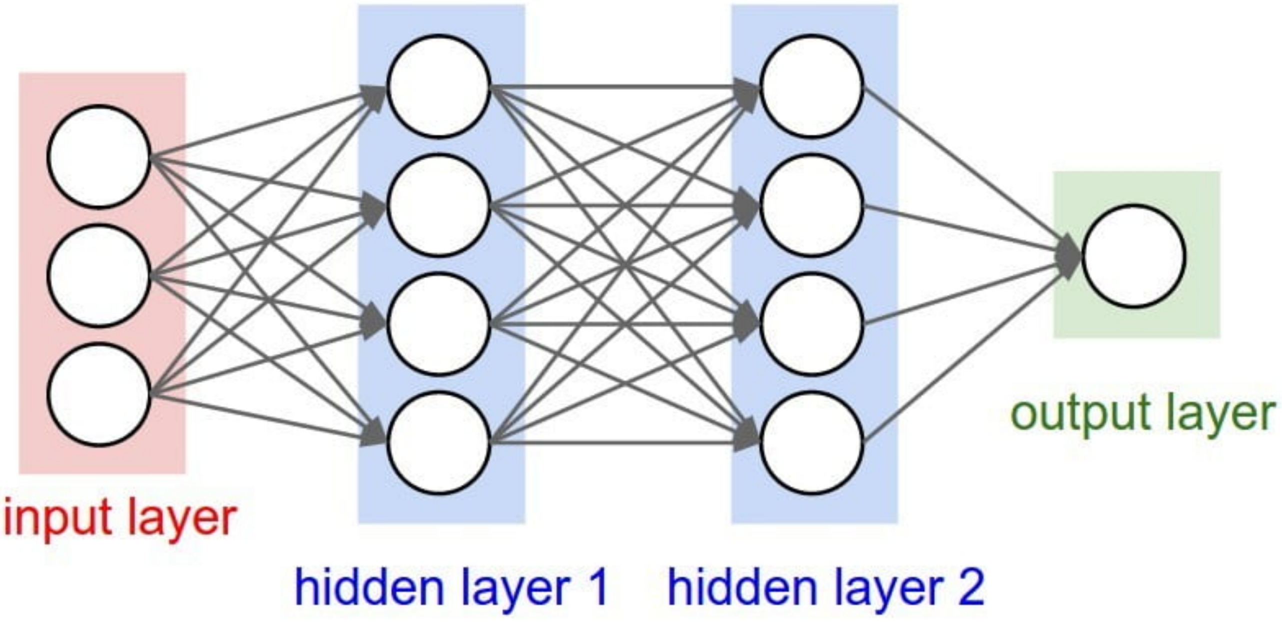
Logistic Regression

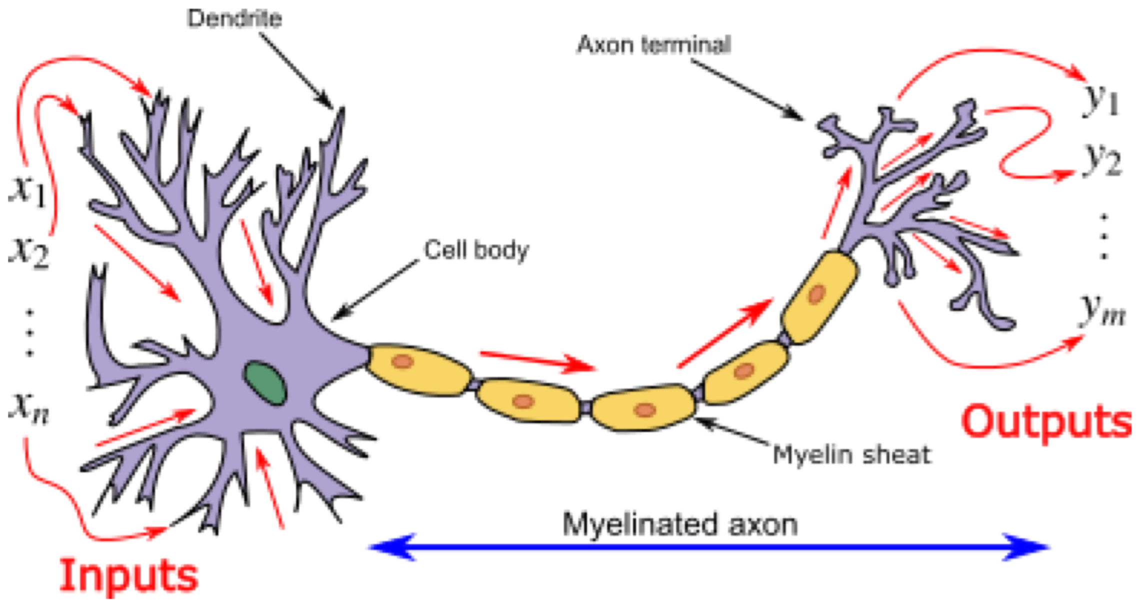
$$L(x,y,\theta) = y(\theta_0 + \sum_i x_i\theta_i) - \log\left(1 + \exp\left(\theta_0 + \sum_i x_i\theta_i\right)\right)$$
$$\nabla_{\theta_0} L(x,y,\theta) = y - p(y = 1 \mid x,\theta)$$
$$\nabla_{\theta_{i=1\ldots n}} L(x,y,\theta) = x_i(y - p(y = 1 \mid x,\theta))$$

Thus the update rules are (reminder: $\theta^{n+1} = \theta^n - \epsilon\nabla_\theta J(\theta)$)

$$\theta_0^{n+1} = \theta_0^n - \epsilon\left(\frac{1}{m'}\sum_j y^j - p(y = 1 \mid x^j, \theta^n)\right)$$
$$\theta_{i=1\ldots n}^{n+1} = \theta_i^n - \epsilon\left(\frac{1}{m'}\sum_j x_i^j (y^j - p(y = 1 \mid x^j, \theta^n))\right)$$

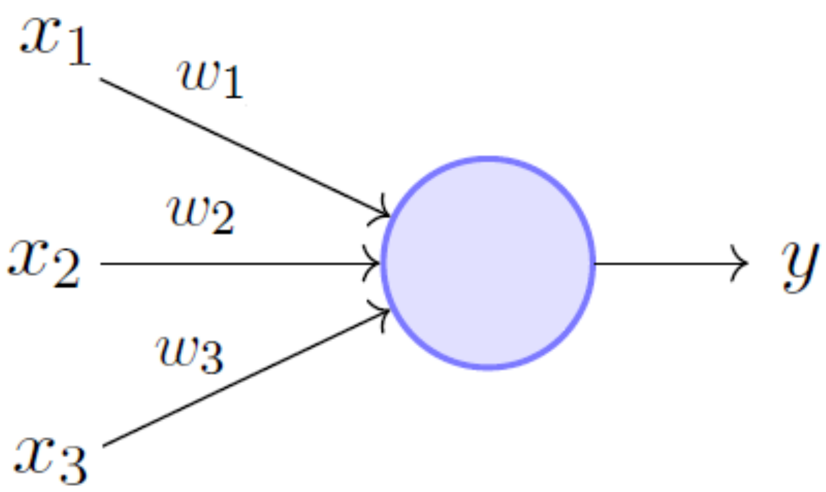
Neural Networks





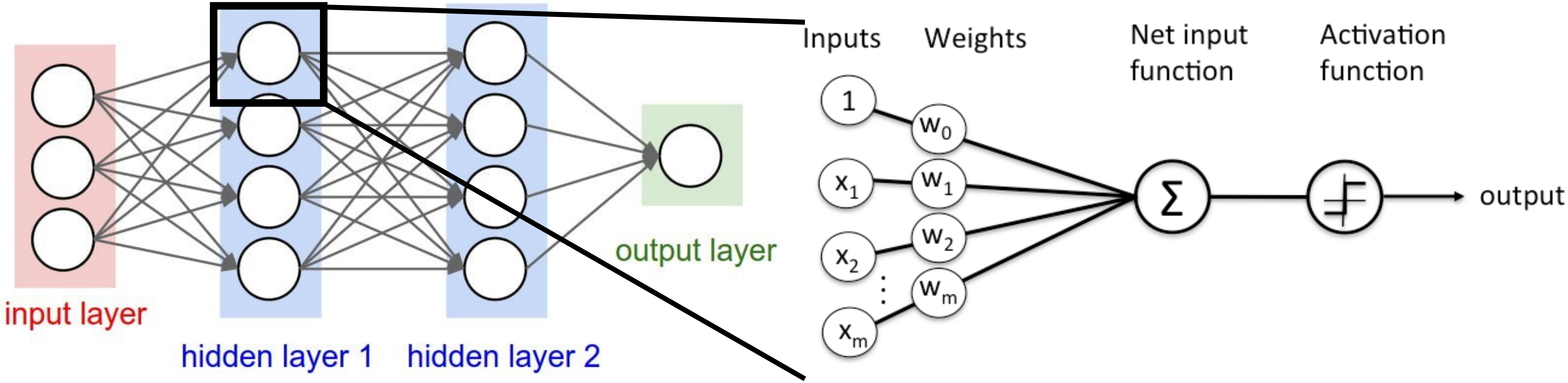
Perceptron

- Similar to logistic regression
- Neural network with only an output layer and no hidden layers



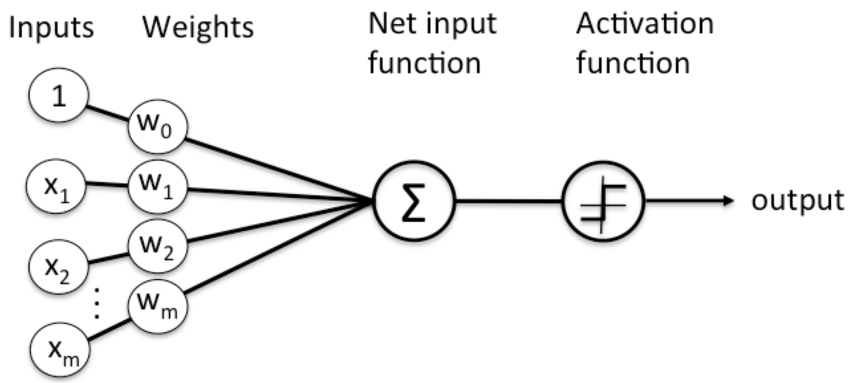
Perceptron Model (Minsky-Papert in 1969)

Neurons and activation functions



Neurons and Activation Functions

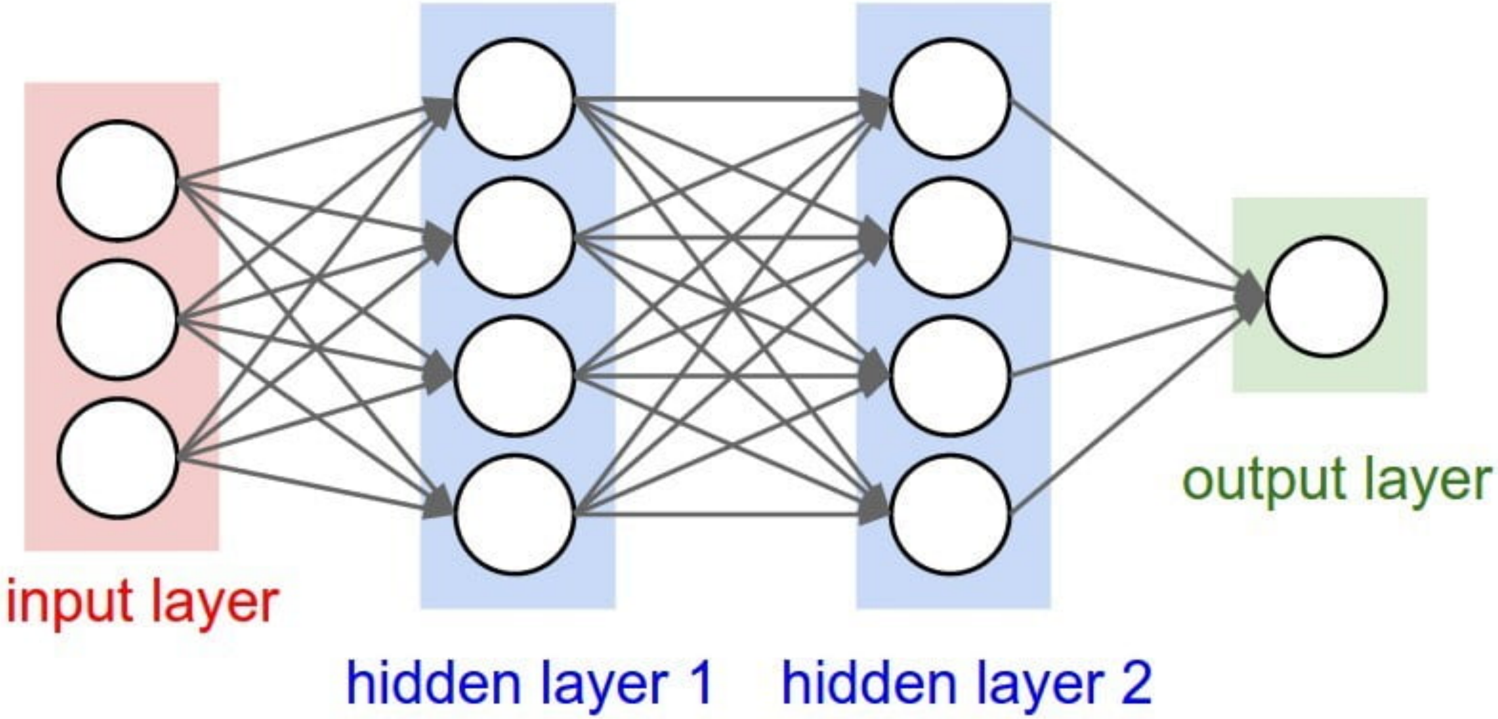
$$\text{output} = f(w_0 + x_1 w_1 + x_2 w_2 + \cdots + x_m w_m)$$



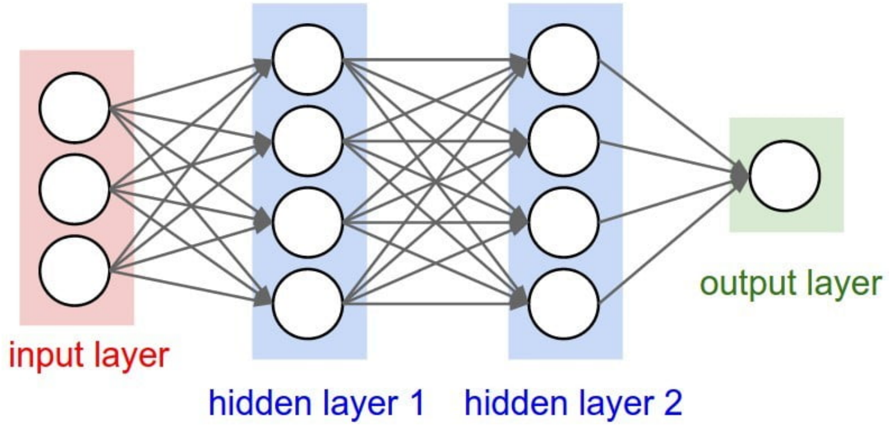
- w_0 is known as the bias
- $f(z)$ is some non-linear function
  - Sigmoid
  - Rectified Linear
  - Tanh
  - etc

Layer of a Network

- Consists of weights and biases, representing a transformation
- Two hidden layers in this example
  - Hidden typically means not shown in any output



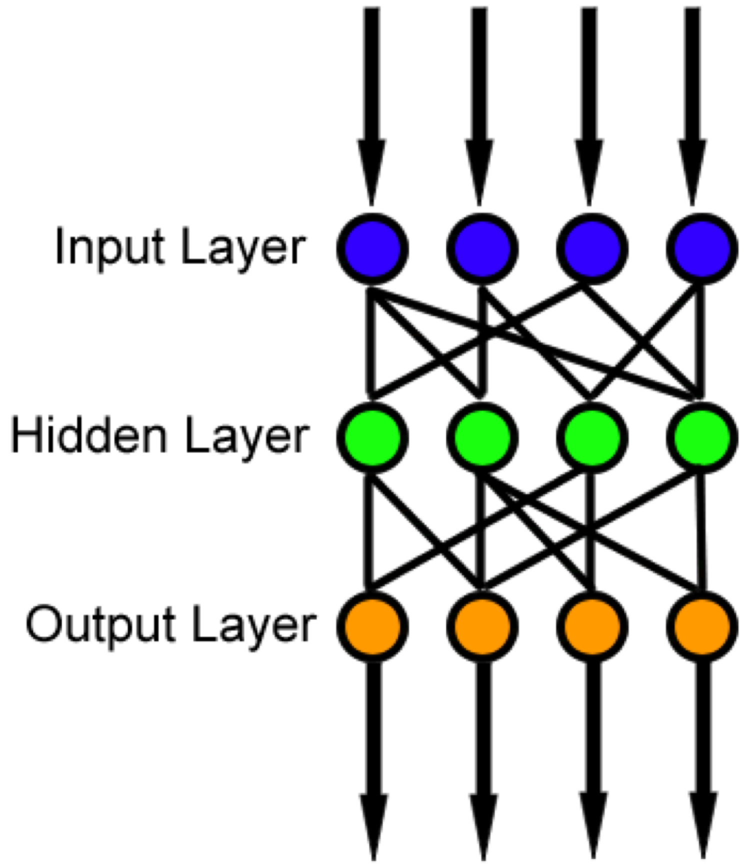
Neural Network Architectures



- Number of layers
  - 2 to hundreds
- Types of activation functions
  - Sigmoid, ReLU, Tanh, PReLU, ELU, etc
- Structure of the layers
  - Fully connected, locally connected, convolutional
- Loss Function
  - Cross-Entropy, Mean Absolute Error, Mean Squared Error

Feedforward Neural Networks

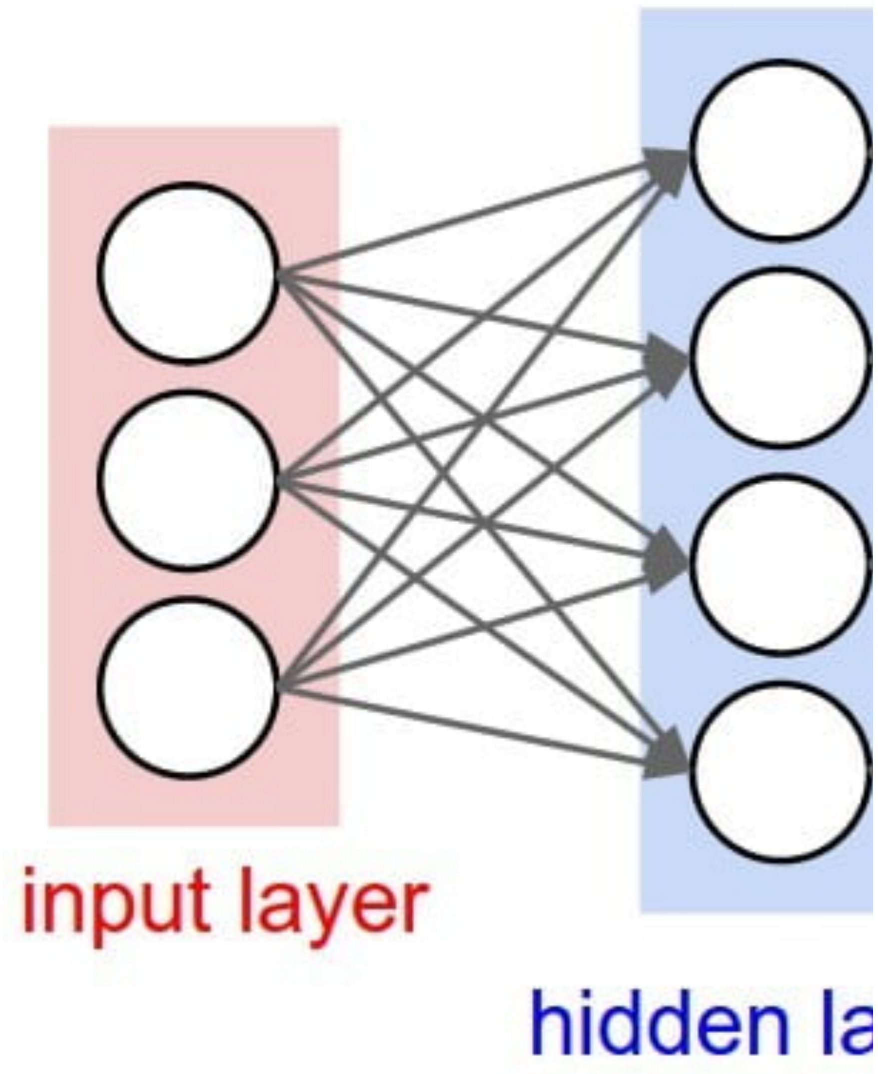
- Inputs are entered into the network, causing the first hidden layer to activate
- Then, the first hidden layer causes the second to activate, then the third, etc
- Finally, the output layer



https://gfycat.com/deadlydeafeningatlanticblackgoby-three-blue-one-brown-machines-learning

Feedforward Step

- Input is still some matrix X where each row is one sample
- Hidden layer is a matrix of size $m \times p$ where m is the number of units in the input layer and p is the number of units in the hidden layer (W_1)
- Additionally, a p-dimensional bias is maintained, corollary to an intercept in linear or logistic regression (W_1)



Feedforward Step

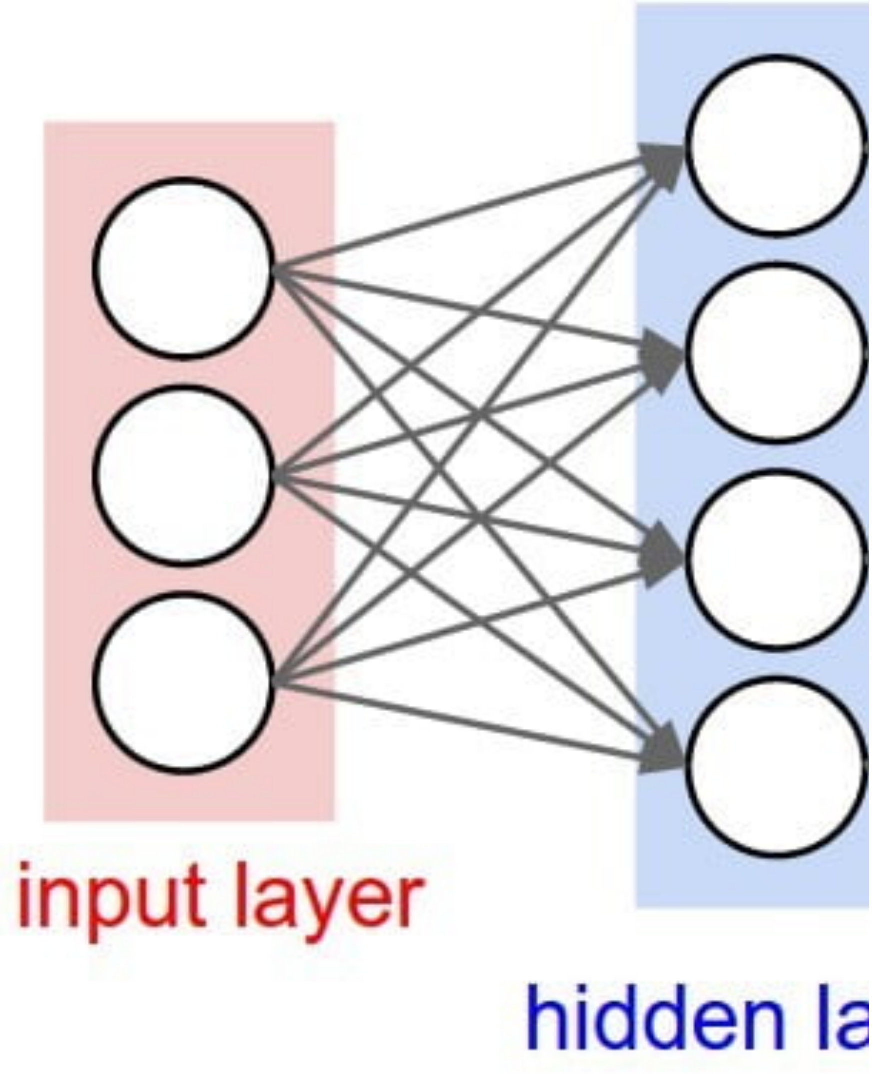
The activation of the first hidden layer can be written as

$$a^{(1)} = f(x \cdot W_1 + b_1)$$

And

$$a_1^{(1)}$$

Is the activation of the top neuron



Backpropogation Algorithm

- Application of the chain rule to training neural networks
- Use stochastic gradient descent to calculate the error with respect to the previous layer, sum up the changes and repeat