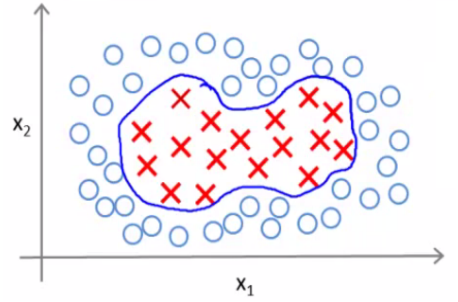
Machine Learning 4: Neural Networks

Consider a face recognition problem

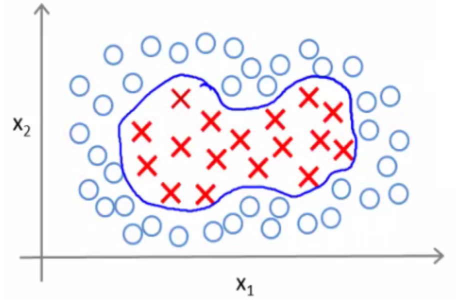
Consider a face recognition problem

For a 50×50 grayscale image, we have 2500 basic features: $x_{1,\ldots,2500} \in \{0, 255\}$

As before, we can map in feature space the 'face' and 'non-face' labels

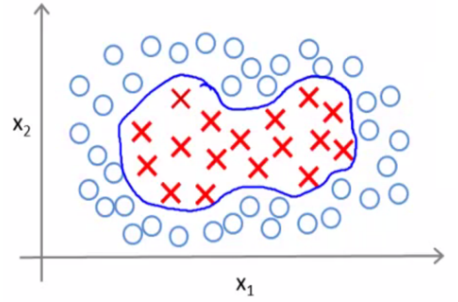


As before, we can map in feature space the 'face' and 'non-face' labels



We definitely require polynomial features to map this decision boundary ...

As before, we can map in feature space the 'face' and 'non-face' labels



We definitely require polynomial features to map this decision boundary ...
Up to quadratic order this is already $\sim 3,000,000$ features ....

For a 2000×2000 colour image, we have 2000^2 basic features ...

For a 2000×2000 colour image, we have 2000^2 basic features …

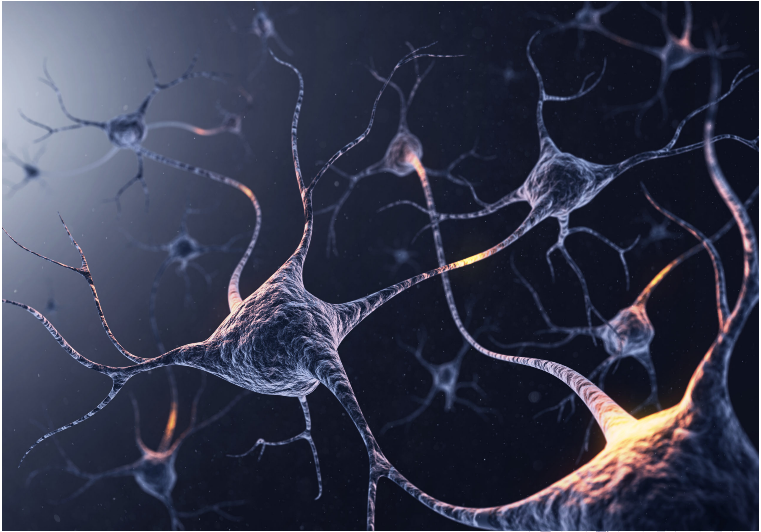
… and we probably want at least up to cubic order for a low bias fit to the data …

For a 2000×2000 colour image, we have 2000^2 basic features ...

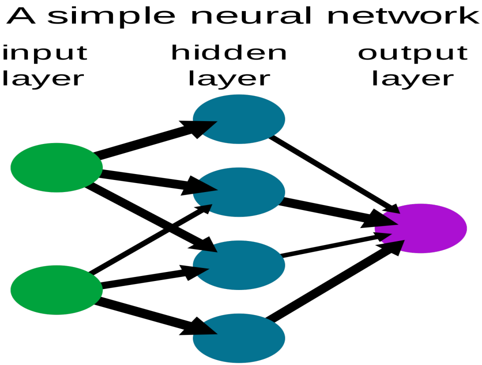
... and we probably want at least up to cubic order for a low bias fit to the data ...

... you get the picture

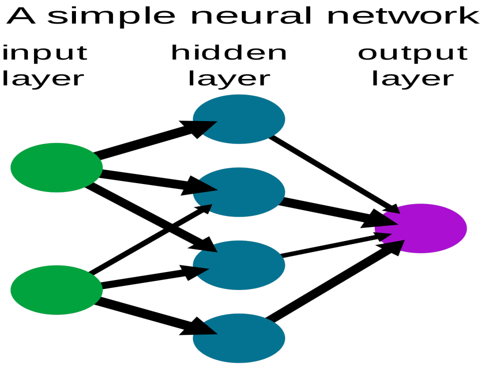
Inspired by nature



Neural Networks

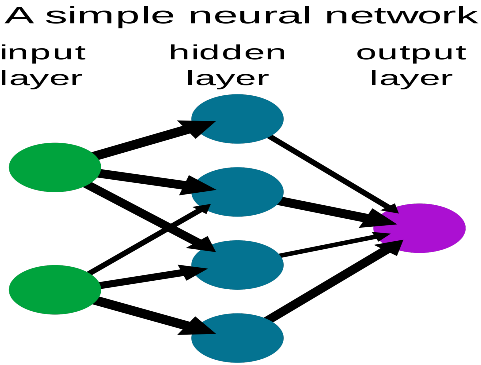


Neural Networks



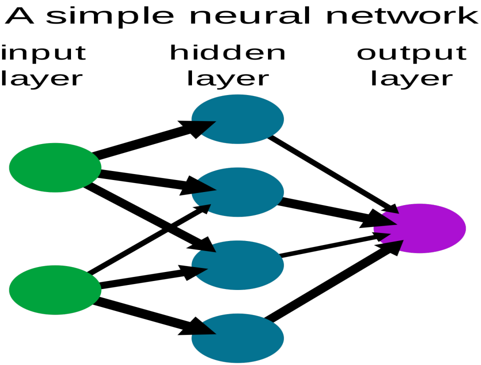
- A basic neural network can be seen as multi-layered, logistic regression based, map, or a string of **logistic units.**

Neural Networks



- A basic neural network can be seen as multi-layered, logistic regression based, map, or a string of **logistic units.**
- Neural networks provide a means to create 'non-linear' features from a set of basic input features. These are called **'activations'**.

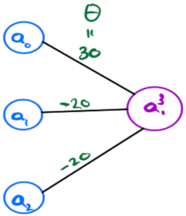
Neural Networks



- A basic neural network can be seen as multi-layered, logistic regression based, map, or a string of **logistic units.**
- Neural networks provide a means to create 'non-linear' features from a set of basic input features. These are called **'activations'**.

Consider the following:

Consider the following:



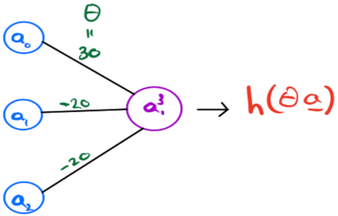
$\rightarrow h(\theta \underline{a})$

a_1	a_2	h = NAND
0	0	1
0	1	1
1	0	1
1	1	0

$$h(\theta \underline{a}) = sig(\theta_0 a_0 + \theta_1 a_1 + \theta_2 a_2)$$

$$= sig(30 - 20a_1 - 20a_2)$$

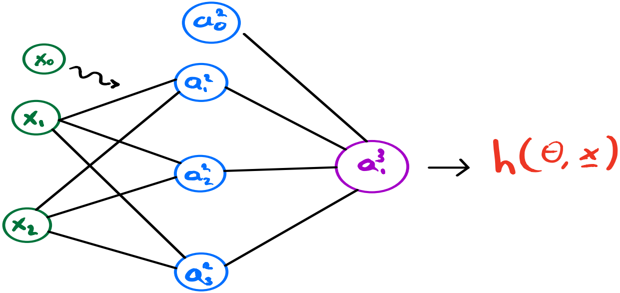
Consider the following:



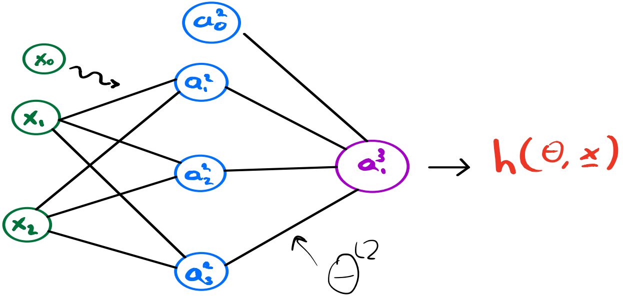
a_1	a_2	h = NAND
0	0	1
0	1	1
1	0	1
1	1	0

$$h(\theta \underline{a}) = sig(\theta_0 a_0 + \theta_1 a_1 + \theta_2 a_2)$$
$$= sig(30 - 20 a_1 - 20 a_2)$$

Let's now consider some binary classification as our goal eg. face or not face.

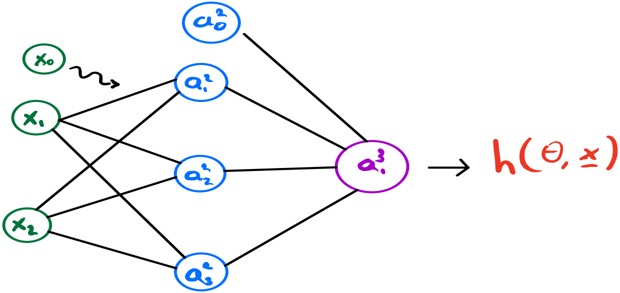


How is the hypothesis defined?



How is the hypothesis defined?

$$h(\Theta, \boldsymbol{x}) = a_1^3 = \mathrm{sig}((\theta^{(2)})^{\mathrm{T}} \boldsymbol{a}),$$
$$(\theta^{(2)})^T \boldsymbol{a} = \theta_{10}^{(2)} a_0^2 + \theta_{11}^{(2)} a_1^2 + \theta_{12}^{(2)} a_2^2 + \theta_{13}^{(2)} a_3^2 \tag{1}$$



How is the hypothesis defined?

$$\begin{aligned} h(\Theta, \boldsymbol{x}) = a_1^3 =& \mathrm{sig}((\theta^{(2)})^{\mathrm{T}} \boldsymbol{a}), \\ (\theta^{(2)})^T \boldsymbol{a} =& \theta_{10}^{(2)} a_0^2 + \theta_{11}^{(2)} a_1^2 + \theta_{12}^{(2)} a_2^2 + \theta_{13}^{(2)} a_3^2 \end{aligned} \tag{1}$$

where

$$\begin{aligned} \tilde{\boldsymbol{a}} =& [a_1^2, a_2^2, a_3^2] = \mathrm{sig}((\theta^{(1)})^{\mathrm{T}} \boldsymbol{x}), \\ a_i^2 =& \mathrm{sig}(\theta_{\mathrm{i0}}^{(1)} \mathrm{x}_0 + \theta_{\mathrm{i1}}^{(1)} \mathrm{x}_1 + \theta_{\mathrm{i2}}^{(1)} \mathrm{x}_2) \end{aligned} \tag{2}$$

Some points

- The bias units $x_0 = a_0^2 = a_0^3 = ... = 1$ allow for an intercept θ_{10} in each input.

Some points

- The bias units $x_0 = a_0^2 = a_0^3 = ... = 1$ allow for an intercept θ_{10} in each input.
- To compute our prediction $h(\Theta, \boldsymbol{x})$, we must compute all the activation units of each layer in a process called **forward propagation**.

Some points

- The bias units $x_0 = a_0^2 = a_0^3 = ... = 1$ allow for an intercept θ_{10} in each input.
- To compute our prediction $h(\Theta, \boldsymbol{x})$, we must compute all the activation units of each layer in a process called **forward propagation**.
- The structure of a neural network, i.e. number of layers and activation units per layer, is called the **architecture**.

Some points

- The bias units $x_0 = a_0^2 = a_0^3 = \ldots = 1$ allow for an intercept θ_{10} in each input.
- To compute our prediction $h(\Theta, \boldsymbol{x})$, we must compute all the activation units of each layer in a process called **forward propagation**.
- The structure of a neural network, i.e. number of layers and activation units per layer, is called the **architecture**.
- We can generalise to **multiclass classification** by introducing more units in the output layer (ex. a_2^3, a_3^3 etc). We then just choose the output unit with the highest probability for our prediction.

Some points

- The bias units $x_0 = a_0^2 = a_0^3 = ... = 1$ allow for an intercept θ_{10} in each input.
- To compute our prediction $h(\Theta, \boldsymbol{x})$, we must compute all the activation units of each layer in a process called **forward propagation**.
- The structure of a neural network, i.e. number of layers and activation units per layer, is called the **architecture**.
- We can generalise to **multiclass classification** by introducing more units in the output layer (ex. a_2^3, a_3^3 etc). We then just choose the output unit with the highest probability for our prediction.
- We want to find Θ which is a collection of weight vectors/matrices θ^j where 'j' is the number of layers. Note that the dimension of θ^i is $s_{j+1} \times (s_j + 1)$ where s_j is the number of units in layer j excluding the bias unit.

Some points

- The bias units $x_0 = a_0^2 = a_0^3 = ... = 1$ allow for an intercept θ_{10} in each input.
- To compute our prediction $h(\Theta, \boldsymbol{x})$, we must compute all the activation units of each layer in a process called **forward propagation**.
- The structure of a neural network, i.e. number of layers and activation units per layer, is called the **architecture**.
- We can generalise to **multiclass classification** by introducing more units in the output layer (ex. a_2^3, a_3^3 etc). We then just choose the output unit with the highest probability for our prediction.
- We want to find Θ which is a collection of weight vectors/matrices θ^j where 'j' is the number of layers. Note that the dimension of θ^i is $s_{j+1} \times (s_j + 1)$ where s_j is the number of units in layer j excluding the bias unit.

So what is our cost function ?

For a neural network with **L** layers, looking to classify into **K** different classes, and a training set of **m** examples, we can extend our old logistic cost function as follows

$$
\begin{aligned}
J(\Theta) = &- \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[y_k^i \log[h_k(x^i)] + (1 - y_k^i) \log[1 - h_k(x^i)] \right] \\
&+ \frac{\lambda}{2m} \sum_{\ell=1}^{L-1} \sum_{i=1}^{s_\ell} \sum_{j=1}^{s_{\ell+1}} (\theta_{ji}^{\ell})^2, \qquad (3)
\end{aligned}
$$

where λ is our regularisation parameter and s_ℓ is the number of units in the ℓ^{th} layer **not** including the bias unit.

For a neural network with **L** layers, looking to classify into **K** different classes, and a training set of **m** examples, we can extend our old logistic cost function as follows

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^i \log[h_k(x^i)] + (1-y_k^i)\log[1-h_k(x^i)]\right] + \frac{\lambda}{2m}\sum_{\ell=1}^{L-1}\sum_{i=1}^{s_\ell}\sum_{j=1}^{s_{\ell+1}}(\theta_{ji}^{\ell})^2, \tag{3}$$

where λ is our regularisation parameter and s_ℓ is the number of units in the ℓ^{th} layer **not** including the bias unit.

We want to minimise the cost by varying the components of Θ. To do this, we can use the **back propagation** algorithm. This is a generalisation of gradient descent which requires the partial derivatives $\frac{\partial J(\Theta)}{\partial \theta_{ij}^{\ell}}$.

Autonomous driving as an example:

https://www.coursera.org/learn/machine-learning/lecture/zYS8T/autonomous-driving

Autonomous driving as an example:

https://www.coursera.org/learn/machine-learning/lecture/zYS8T/autonomous-driving

Take a look at Python tutorial 4