SWINBURNE UNIVERSITY OF TECHNOLOGY

COS30082 Applied Machine Learning

Lecture 4

Support Vector Machine & Artificial Neural Network

SVM Topics

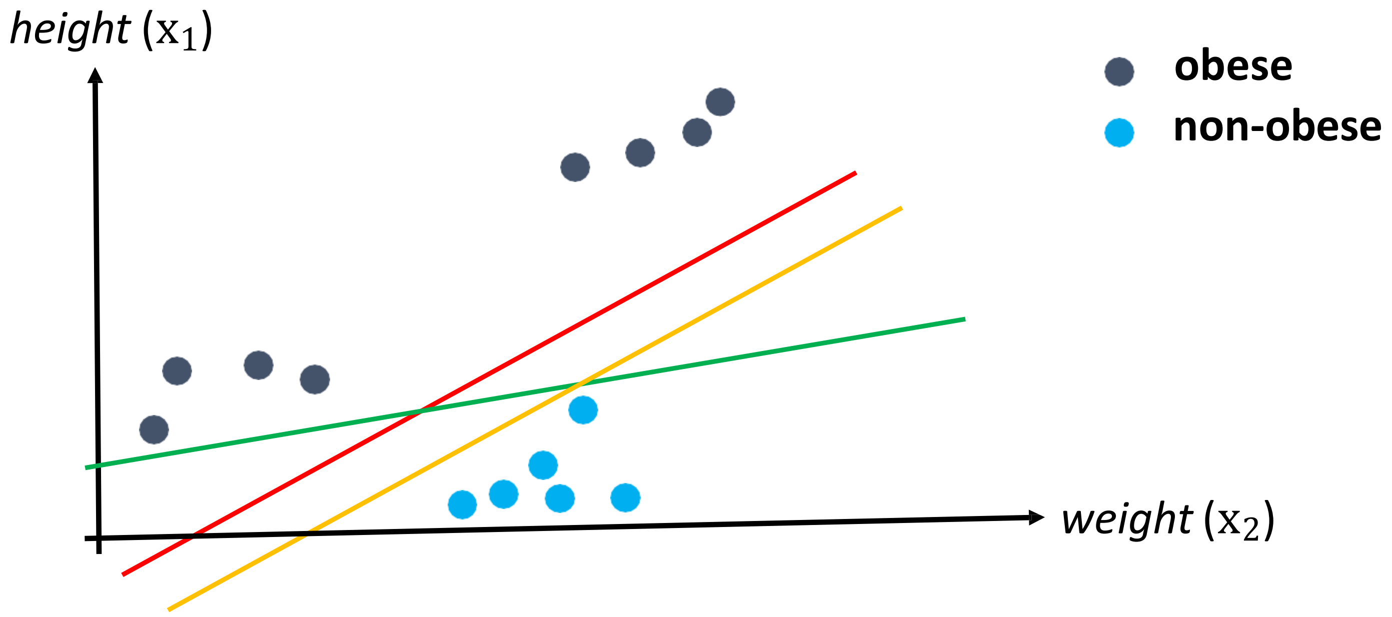
- Understanding of Support Vector Machine (SVM)
- To find out how SVM works
- To learn large-margin optimization techniques
- To explore the differences between SVM and Logistic Regression.
- To solve a machine learning problem using SVM.

What is SVM?

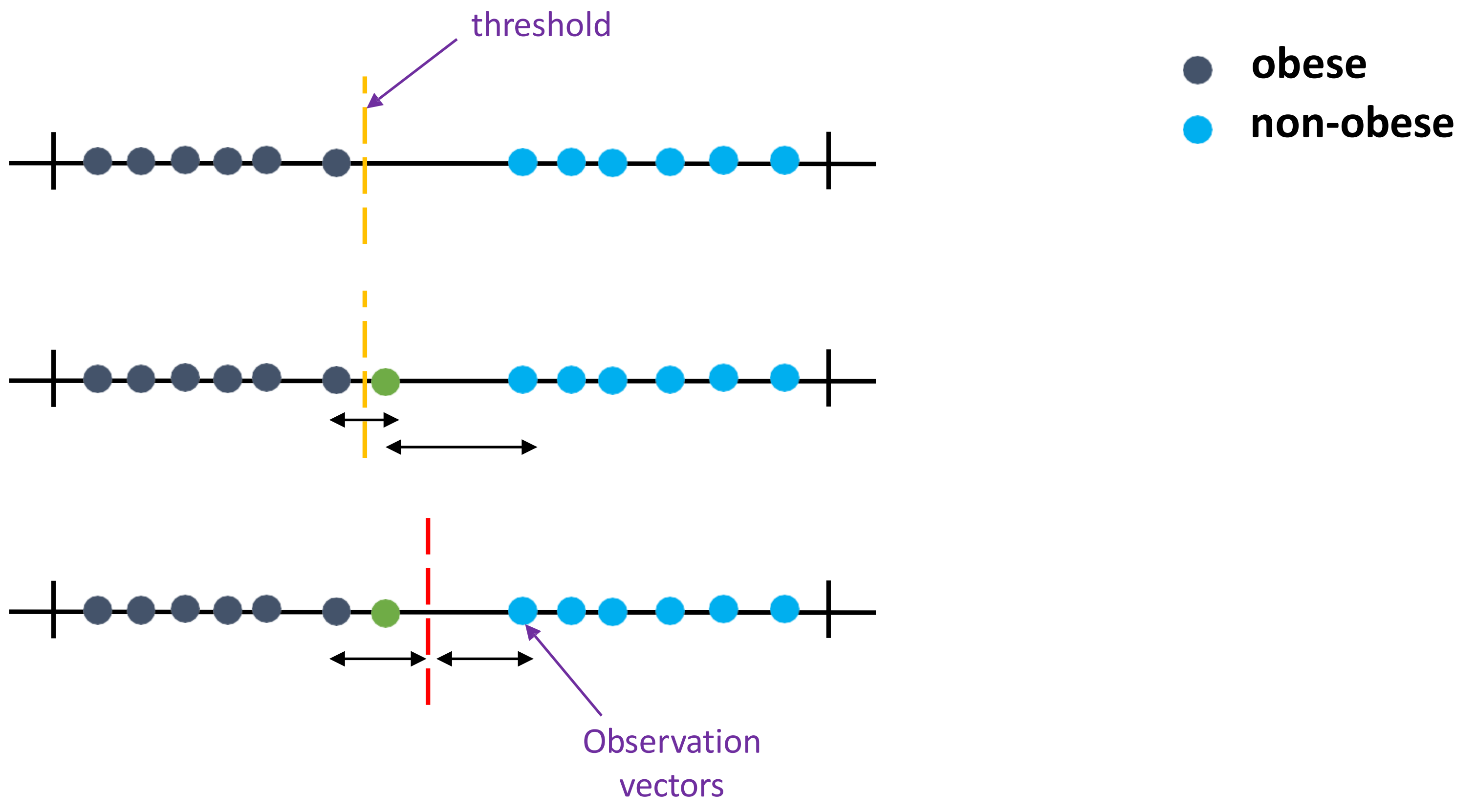
- SVM is a supervised machine learning algorithm which is well-known for classification problem.
- It is also called a **large-margin classifier**

Why large-margin classifier?

- Given *N* number of data points (training set)
- To classify *obese/non-obese* (y) based on the *height* x_1 and *weight* x_2 of the person, which will be the decision boundary chosen by the SVM?



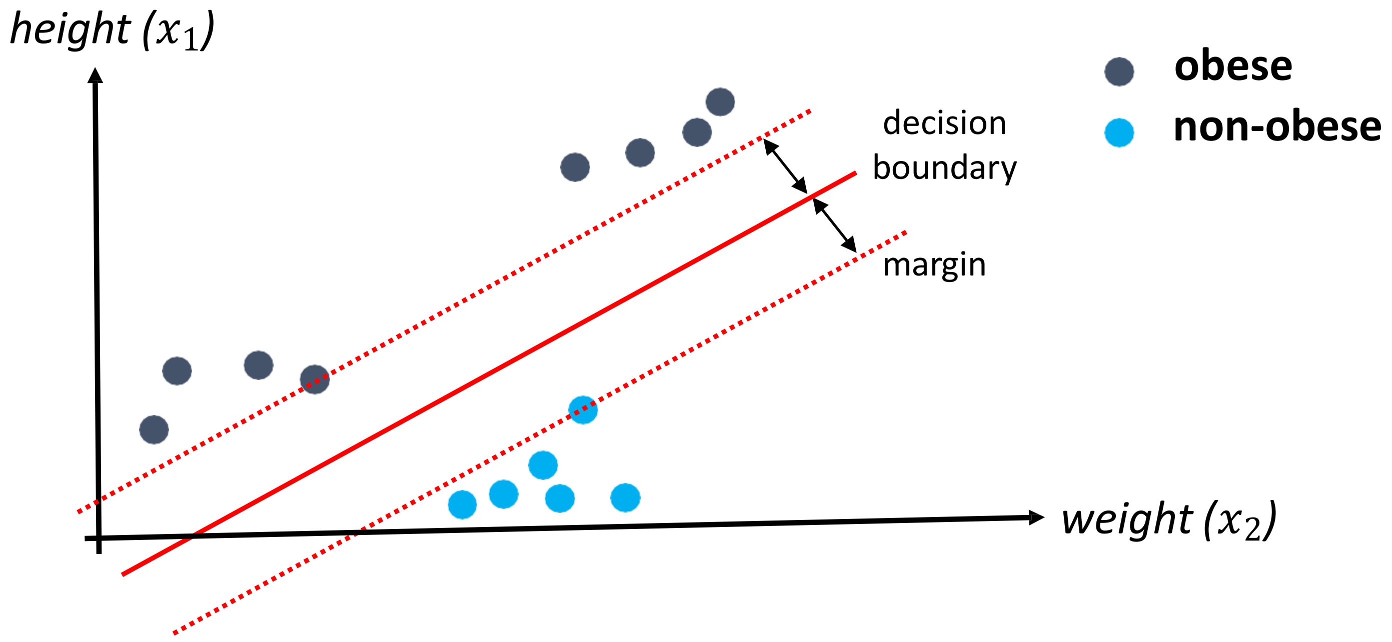
Why large-margin classifier?



- In SVMs, the distance between the observation vectors of the two domains (positive and negative) is equal to the threshold, which together constitute the margin.

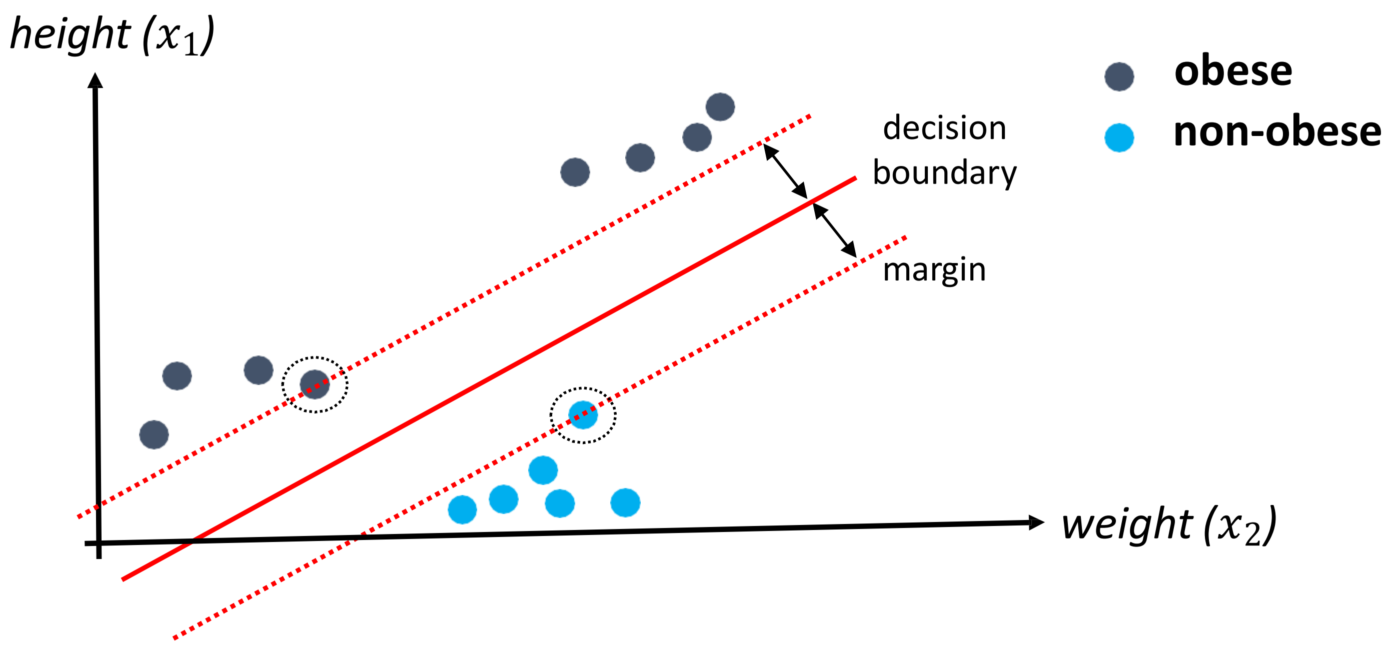
Why large-margin classifier?

- Answer is the red line.
- The objective of the SVM algorithm is to find a decision boundary with the maximum margin, which is the greatest distance from the nearest data point to the decision boundary.



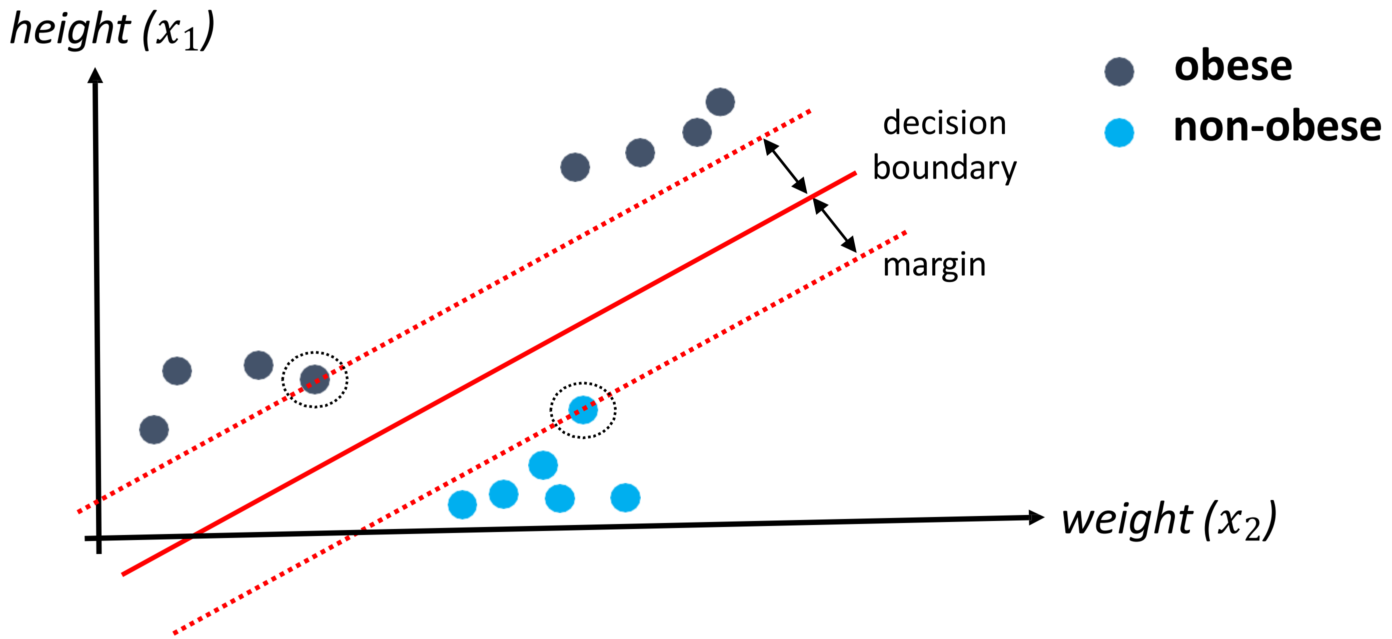
How SVM Works?

- Objective: To find the hyperplane that best separates the classes with the maximum margin.
- Margin: Distance between the hyperplane and the nearest data points from each class (support vectors).



How SVM Works?

- For linearly separable data, SVM finds a direct hyperplane that separates the classes.
- For non-linearly separable data, SVM uses kernel tricks to transform the data into a higher dimension where a hyperplane can be used for separation.



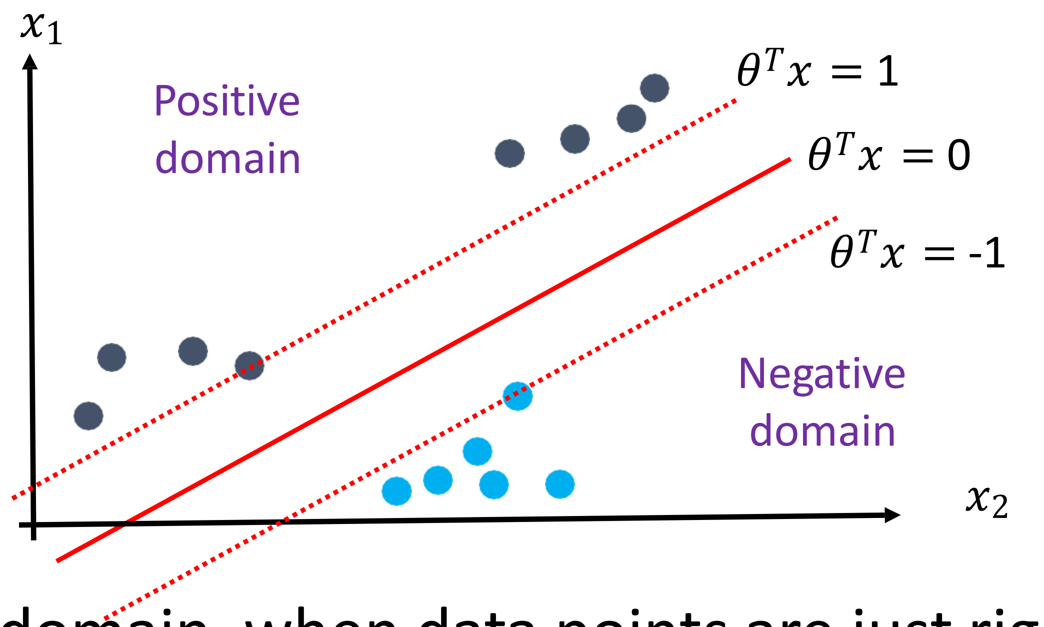
SVM Kernal Trick

- The kernel trick involves transforming data into a higher dimension where it is easier to separate using a linear classifier.
- Common Kernels: Linear, Polynomial, Radial Basis Function (RBF), Sigmoid.
- The choice of kernel affects the decision boundary and the performance of the SVM model.

Math intuition behind SVM

$if \quad y=1 \qquad cost_1(\theta^T x) = \max(0, 1-\theta^T x)$

$if \quad y=0 \qquad cost_0(\theta^T x) = \max(0, 1+\theta^T x)$



Assume 3 hyperplanes
- Decision boundary: $\theta^T x = 0$
- Margin: $\theta^T x = \pm 1$

- For the positive domain, when data points are just right on the margin, $\theta^T x$ = 1, when data points are between decision boundary and margin, 0 < $\theta^T x$ < 1.
- For the negative domain, when data points are just right on the margin, $\theta^T x$ = -1, when data points are between decision boundary and margin, -1 < $\theta^T x$ < 0.

Loss Function

- The loss function is designed to find a decision boundary (a hyperplane in the feature space) that maximizes the margin between different classes while penalizing points that fall on the wrong side of the margin.
- The two primary components of the SVM loss function are:
  - **Hinge Loss**
  - **Regularization Term**

Hinge Loss

- **Hinge Loss:** The hinge loss is used for "maximum-margin". For an individual sample, the hinge loss is defined as:

$$L_i = \max(0, 1 - y_i(w \cdot x_i - b))$$

- where *w* represents the weights of the model, *xi* is the feature vector of the i-th sample, *b* is the bias term, and *yi* is the actual class label for the i-th sample, which should be -1 or 1. The term *w·xi–b* is the raw output of the classifier. If this output is correct and beyond the margin, the loss is 0. Otherwise, the loss increases linearly with the distance from the margin.

Regularization Term

- **Regularization Term:** To prevent the model from overfitting, a regularization term is added to the loss function. The most common form in SVMs is the $L2$ regularization, which is the squared magnitude of the weight vector w.

Loss Function

- complete SVM loss function with the regularization term is given by:

$$J(w) = \frac{\lambda}{2}||w||^2 + \frac{1}{N}\sum_{i=1}^{N} \max(0, 1 - y_i(w \cdot x_i - b))$$

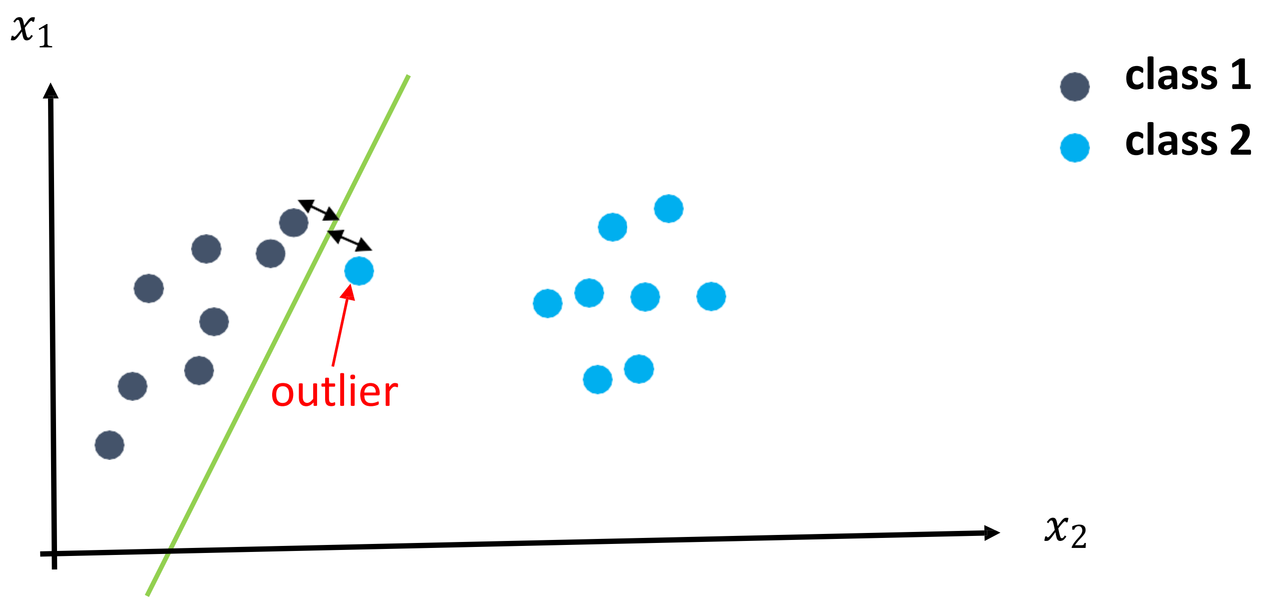
- where *λ* is a regularization parameter that controls the trade-off between increasing the margin size and ensuring that the *xi* lies on the correct side of the margin. *N* is the number of training samples.
- A smaller value for *λ* implies less regularization (allowing more complexity in the model), whereas a larger *λ* promotes simplicity in the model at the risk of not capturing all the nuances in the data.

Loss Function

- In In many descriptions of SVMs, particularly in the context of scikit-learn or other practical machine learning libraries, you might encounter the regularization parameter denoted as C rather than λ. The formulation with C looks like this: :
- $$J(w) = C\sum_{i=1}^{N} \max(0, 1 - y_i(w \cdot x_i - b)) + \frac{1}{2}||w||^2$$
- A large value of *C* corresponds to assigning a higher penalty to misclassified points, essentially prioritizing the correctness of classification over the width of the margin. Conversely, a smaller *C* value emphasizes a larger margin and allows more misclassifications.

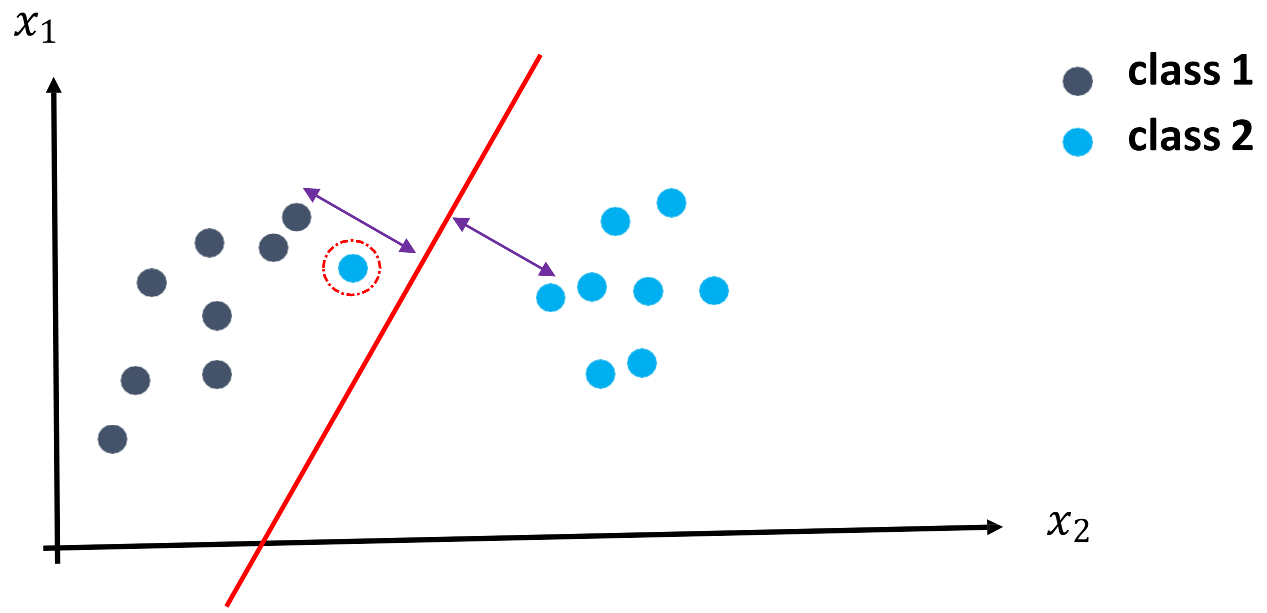
Large C effect

- However, model with large C value is **sensitive to outliers**, similar to no regularization.
- Such a model is subject to overfitting.
  - lower bias, higher variance



Small C effect

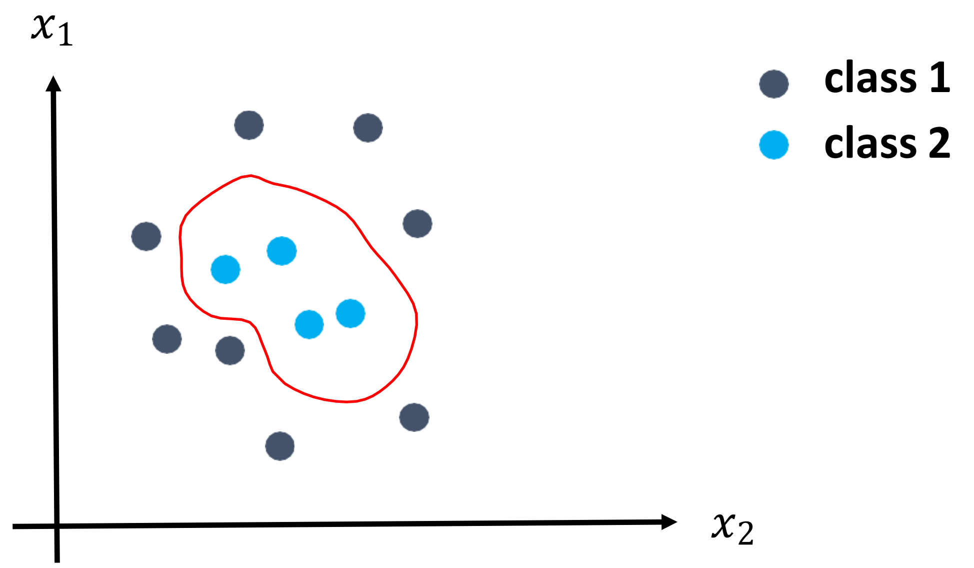
- Smaller C allows a bigger margin. It is useful for non-linearly separable dataset with tolerance of data points who are **misclassified** or **have margin violation**.



- However, a model with a too small C value is subject to underfitting
  - Higher bias, lower variance

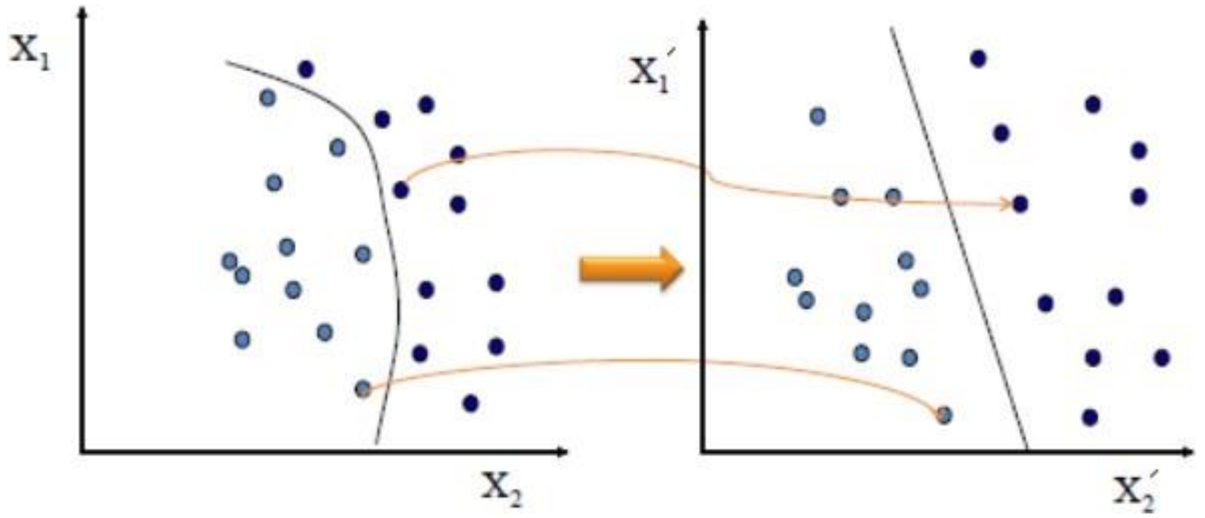
Non-linear SVM

- The simplest way to separate two groups of data is with a straight line (1 dimension), flat plane (2 dimensions) or an N-dimensional hyperplane.
- However, there are situations where a nonlinear region can separate the groups more efficiently.
- Non linear decision boundary is necessary to separate the data points.



What is function f?

- f is a **similarity function** which is called a **kernel function**. It is used to map the data into a different space when a hyperplane (linear) cannot be used to do the separation.
- A non-linear function is learned by a linear learning machine in a high-dimensional feature space while the capacity of the system is controlled by a parameter that does not depend on the dimensionality of the space.
- This is called *kernel trick* which means the kernel function transform the data into a higher dimensional feature space to make it possible to perform the linear separation.



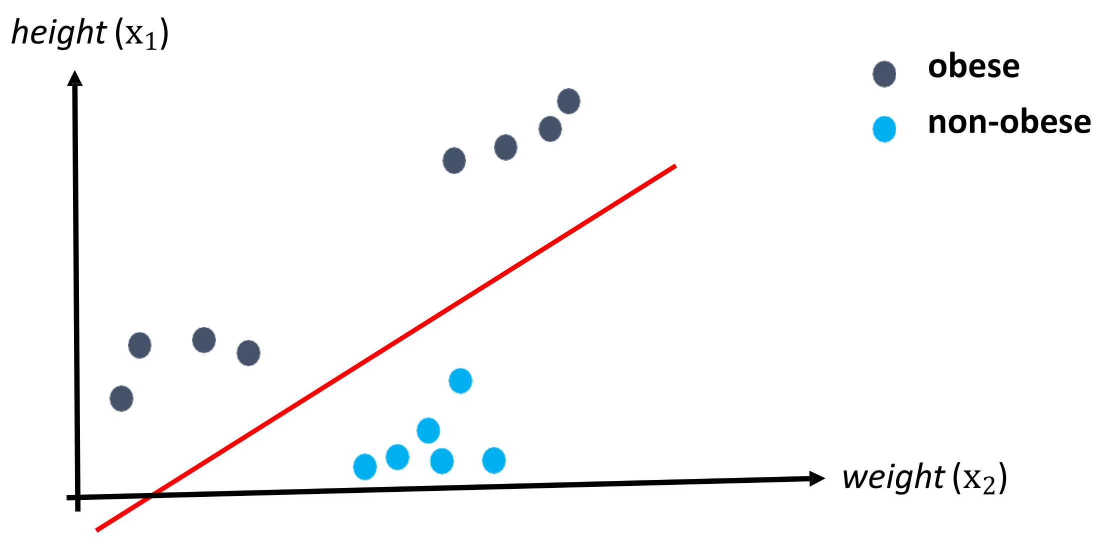
SVM vs logistic regression

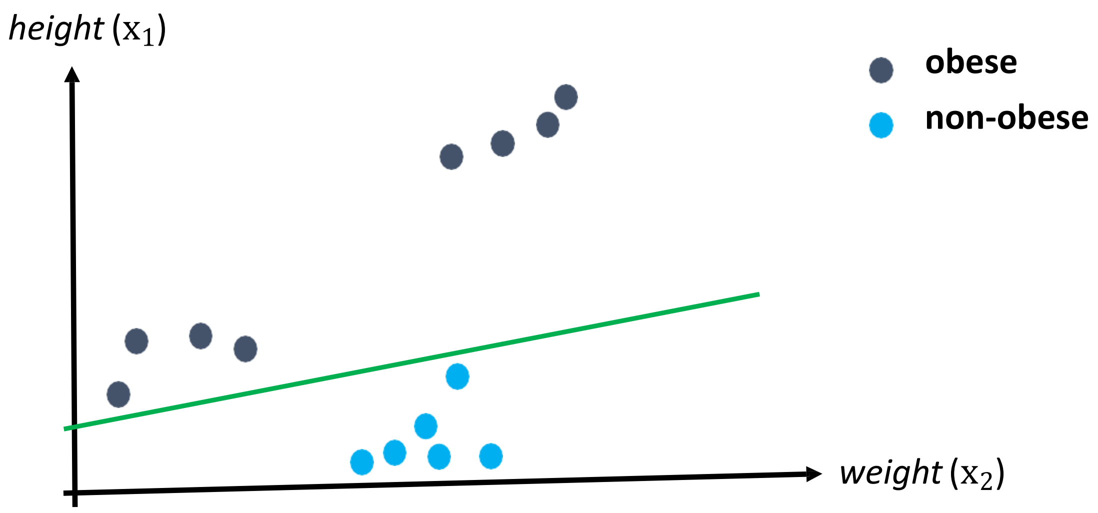
SVM

- It is able to find the maximum margin (distance between the decision boundary and the support vectors) that separates the 2 classes.

Logistic regression (LR)

- It can have different decision boundaries with different parameters θs that are near the optimal solution.





SVM vs logistic regression

- When to use logistic regression and support vector machine?
  - Based on the number of training samples (n) and the number of features (m).

- If m is large and n is small where m >> n: Linear SVM or Logistic Regression might be a choice.
- If m is small and n is intermediate where n > m: SVM with Gaussian kernel might be a choice
- If m is small and n is large where n >> m: Linear SVM or Logistic Regression might be a choice. (preferable add more features)

SVM vs ANN

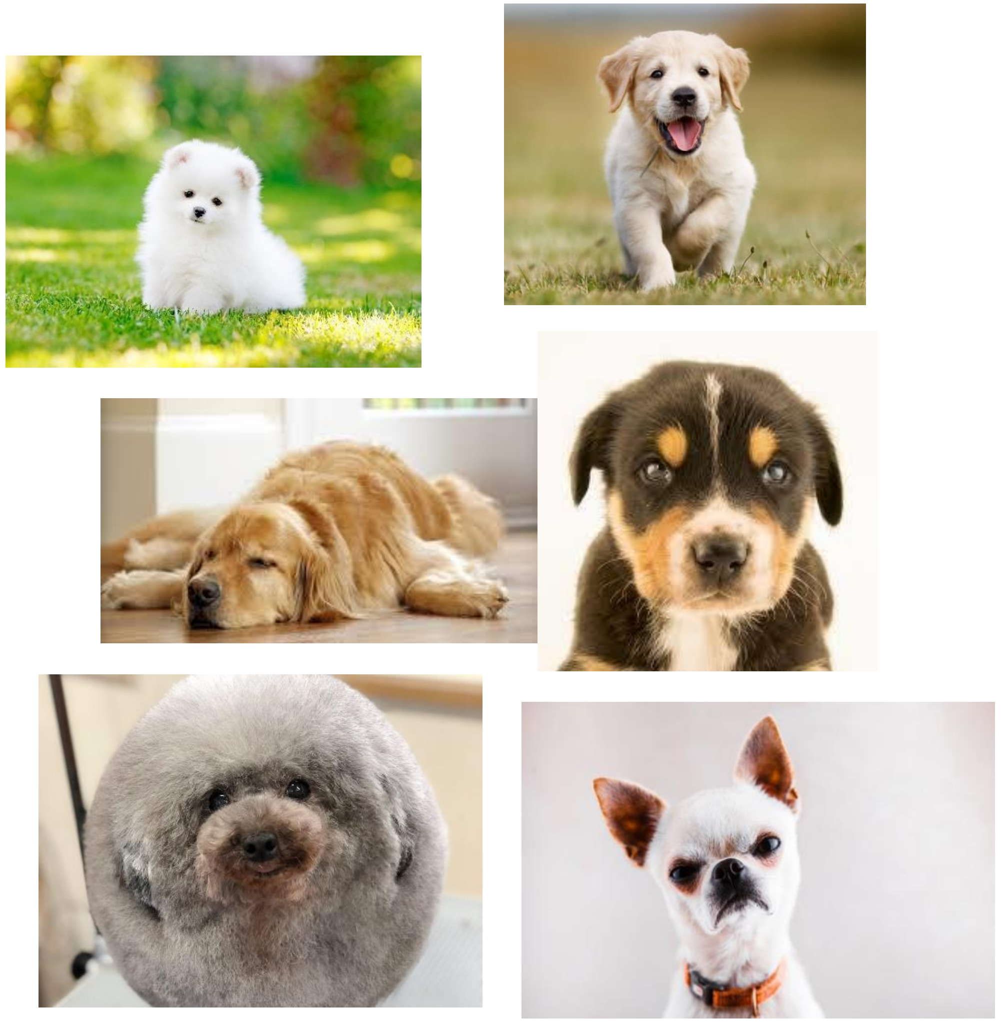
- SVMs transform the classification task into a convex optimization problem that guides towards the global optimum, whereas ANN can suffer from multiple local minima.
- SVMs are inherently binary classifiers, which means they find a hyperplane separating between two classes, whereas ANN can handle many classes by design.
- ANN generally performs better in a very large dataset.
- However, it is not possible to conclude which one performs the best because it depends on the problem and the dataset

ANN Topics

- Understanding of what is Artificial Neural Network (ANN)
- To find out how ANN works
- To explore the ANN learning algorithm and how it differs from previously learned techniques
- To solve a machine learning problem using a ANN algorithm

Why non-linear hypothesis?

Binary classification : Dog or Cat



Why non-linear hypothesis?

Binary classification : Dog or Cat

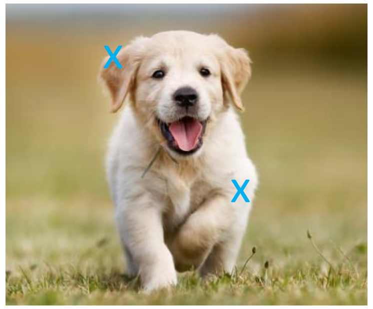
$\rightarrow$ **Features:** colour/pixel intensity $\rightarrow$ **Classification model**

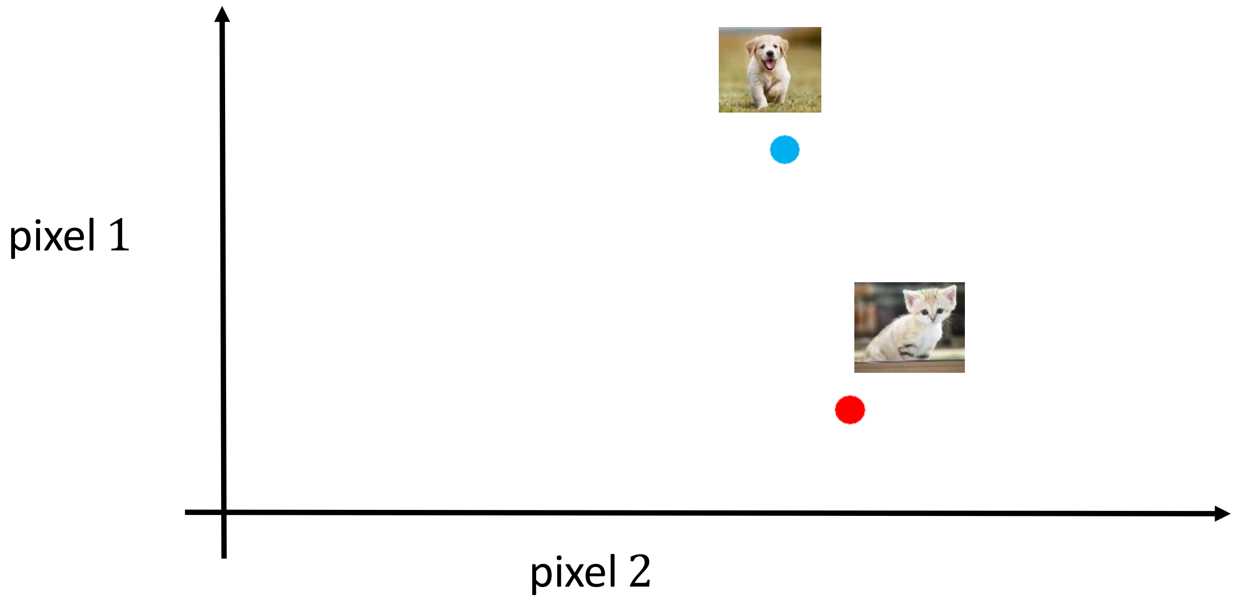
Testing phase

Test image

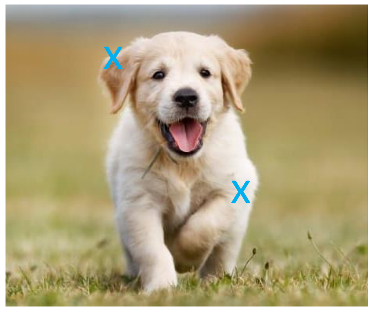
$\rightarrow$ **Features:** colour/pixel intensity $\rightarrow$ **Classification model** $\rightarrow$ Predicted as dog

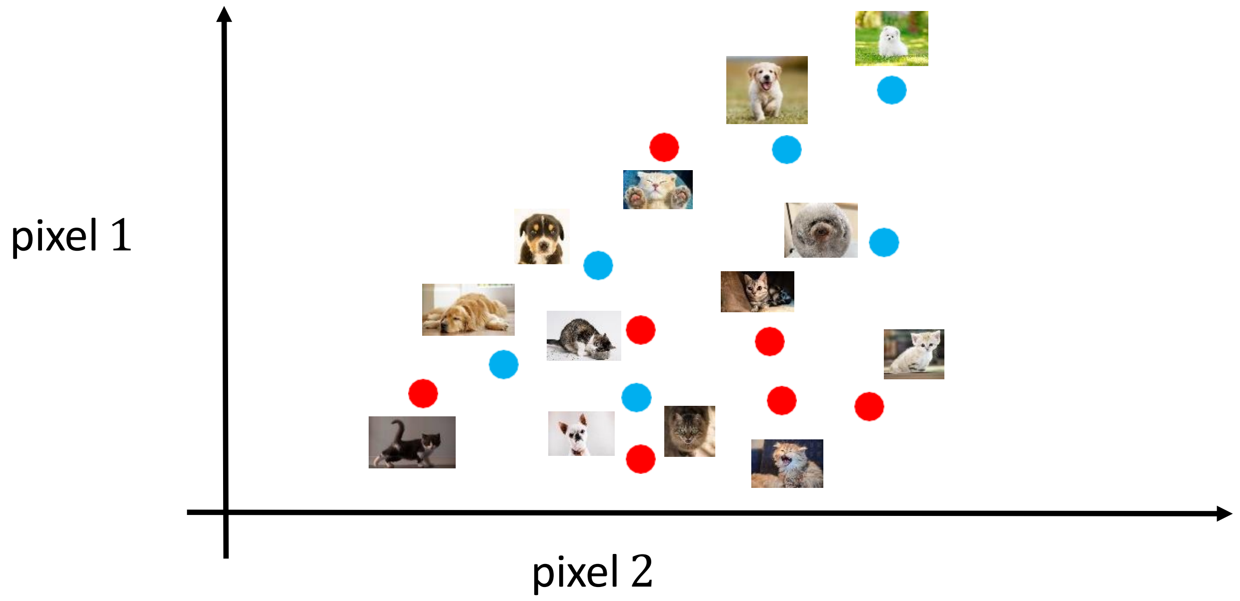
Why non-linear hypothesis?



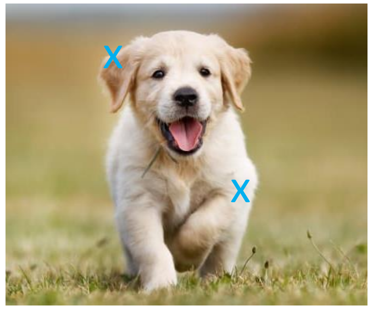


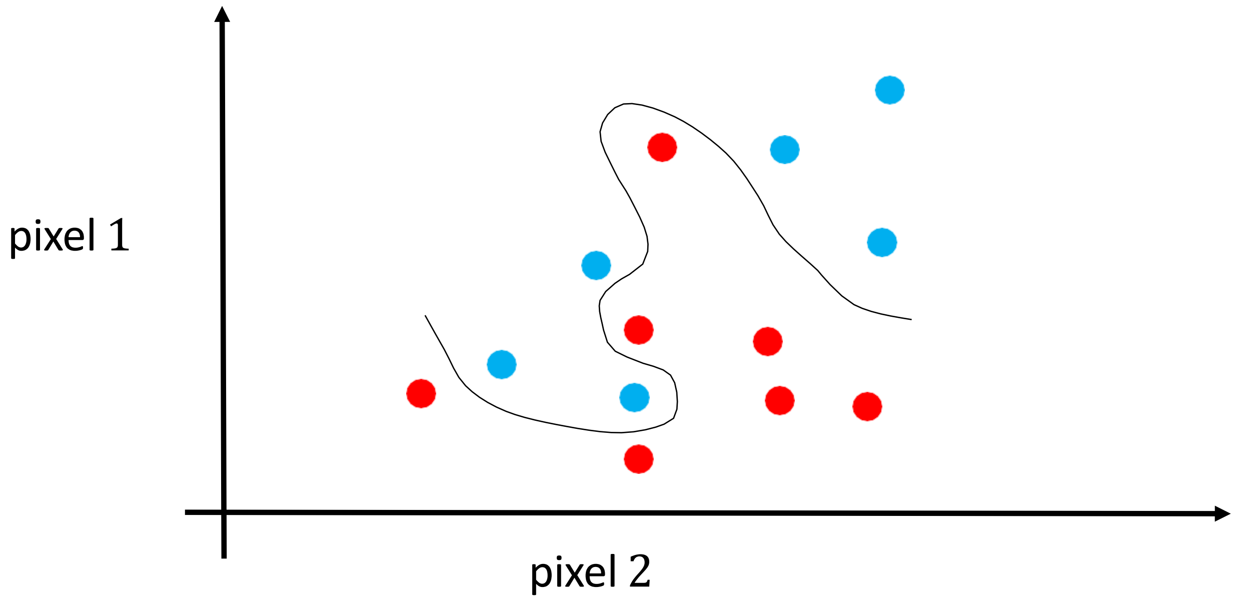
Why non-linear hypothesis?



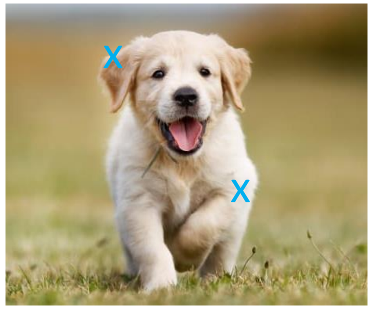


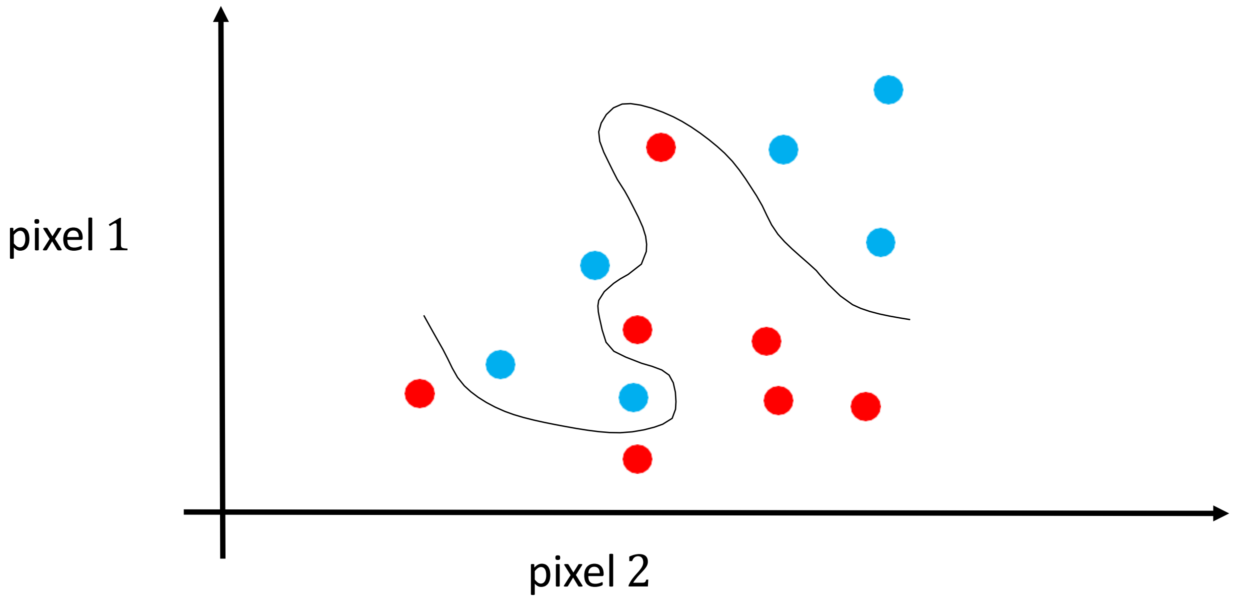
Why non-linear hypothesis?





Why non-linear hypothesis?

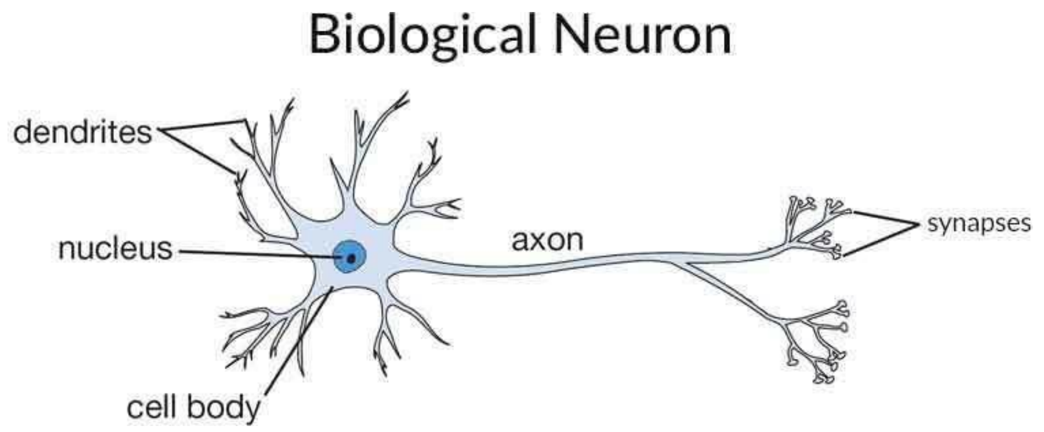




- Imagine, the size of the features is the image size: n = 60 x 60 pixels
- Hence, we have 3600 number of features per image (10800 for RGB images).
- If we want to achieve non linear hypothesis and will to feed in *quadratic features* (x^2), then the $n \approx 58$ million.
- Features that are too large are not a good way of representation and lead to high computational cost.
- Neural network provides a better alternative to learn non-linear model.

Understanding of ANN

- The human brain has hundreds of billions of cells called neurons. Each neuron is made up of a cell body that is responsible for processing information by transporting information to (inputs) and from (outputs) from the brain.

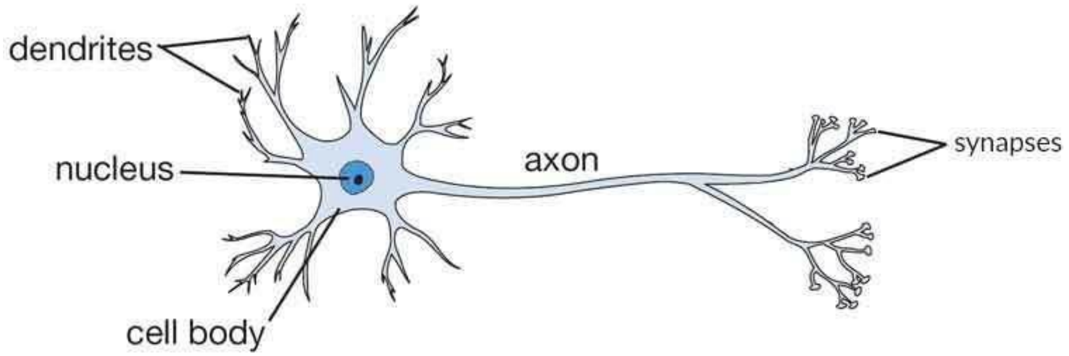


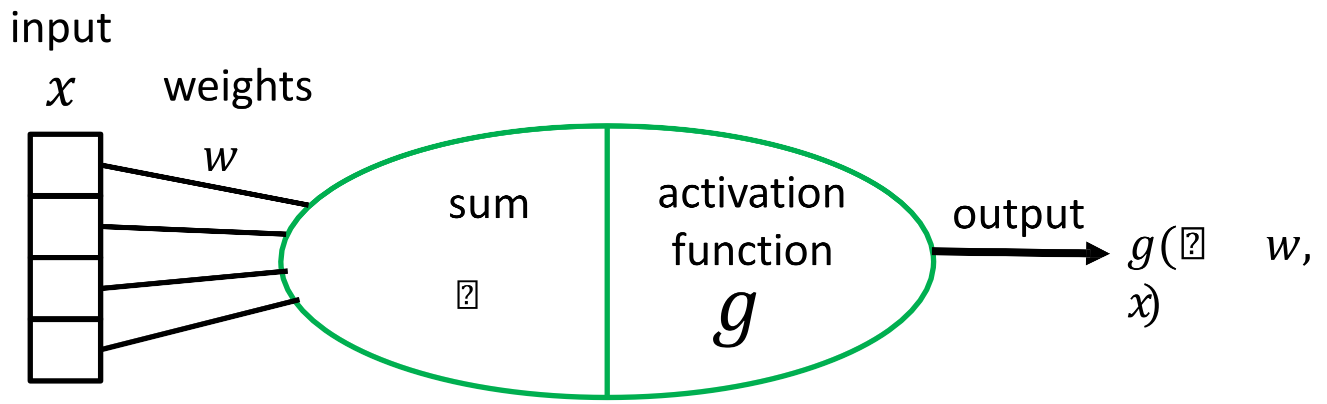
https://www.xenonstack.com/blog/artificial-neural-network-applications/

Understanding of ANN

- Artificial neural networks are built like the human brain. It has hundreds or thousands of artificial neurons which are interconnected. The input neurons receive various forms of data and, based on an internal weighting system, the neural network learns the patterns of the data and matches them to the target output.

Biological Neuron





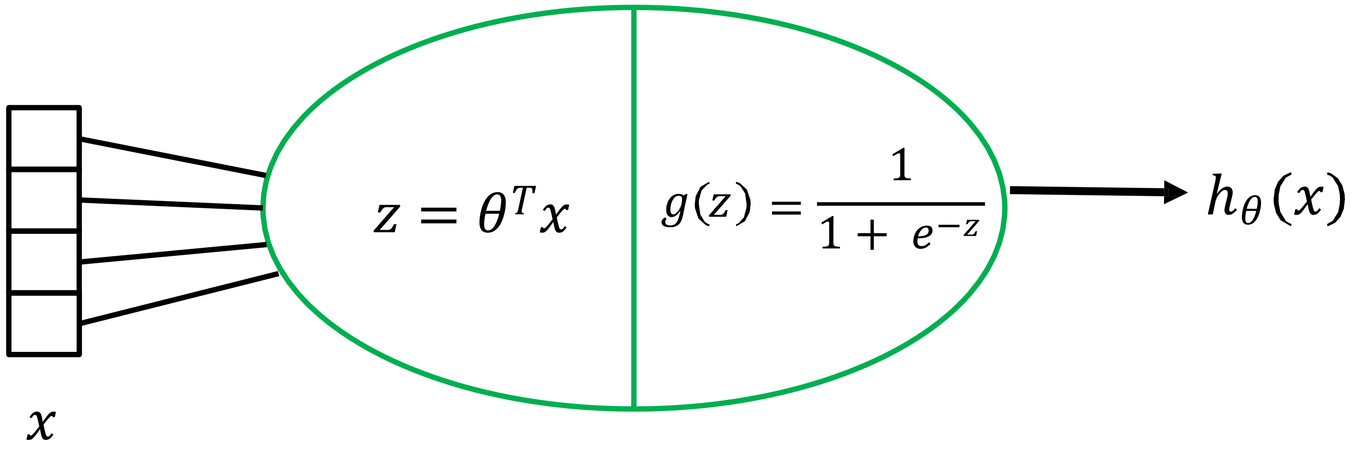
Neuron model for logistic regression

- In binary classification, to ensure predicted value $h_\theta\ (x)$ lies between 0 and 1:

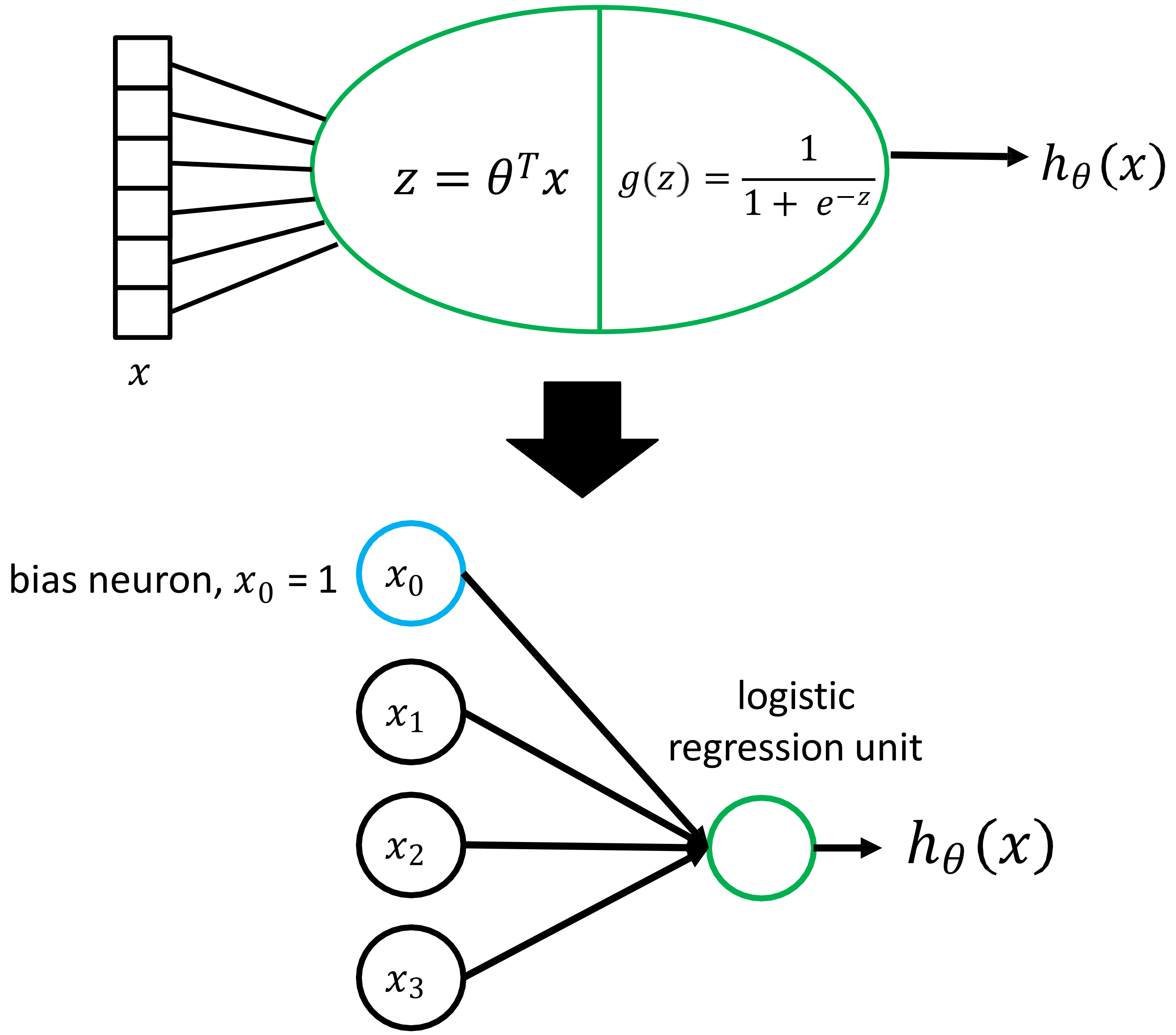
$$1 \leq h_\theta(x) \leq 0$$

$h_\theta(x)$ is represented as $\rightarrow h_\theta\ (x) = g(z)$

where $z = \theta^T x \quad g(z) = \frac{1}{1 + e^{-z}}$



Neuron model for logistic regression

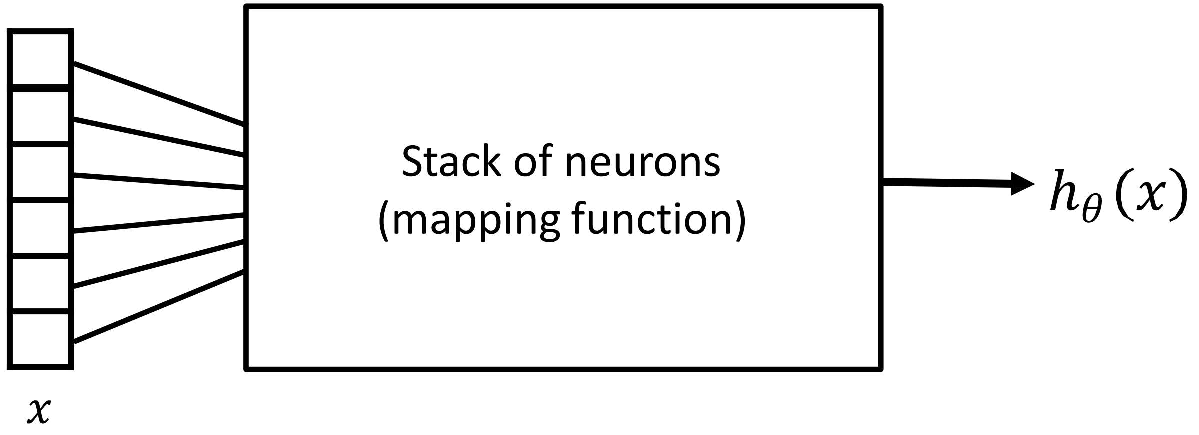


- Says an input x with 3 dimensional features
- The neuron model terminologies for LR:

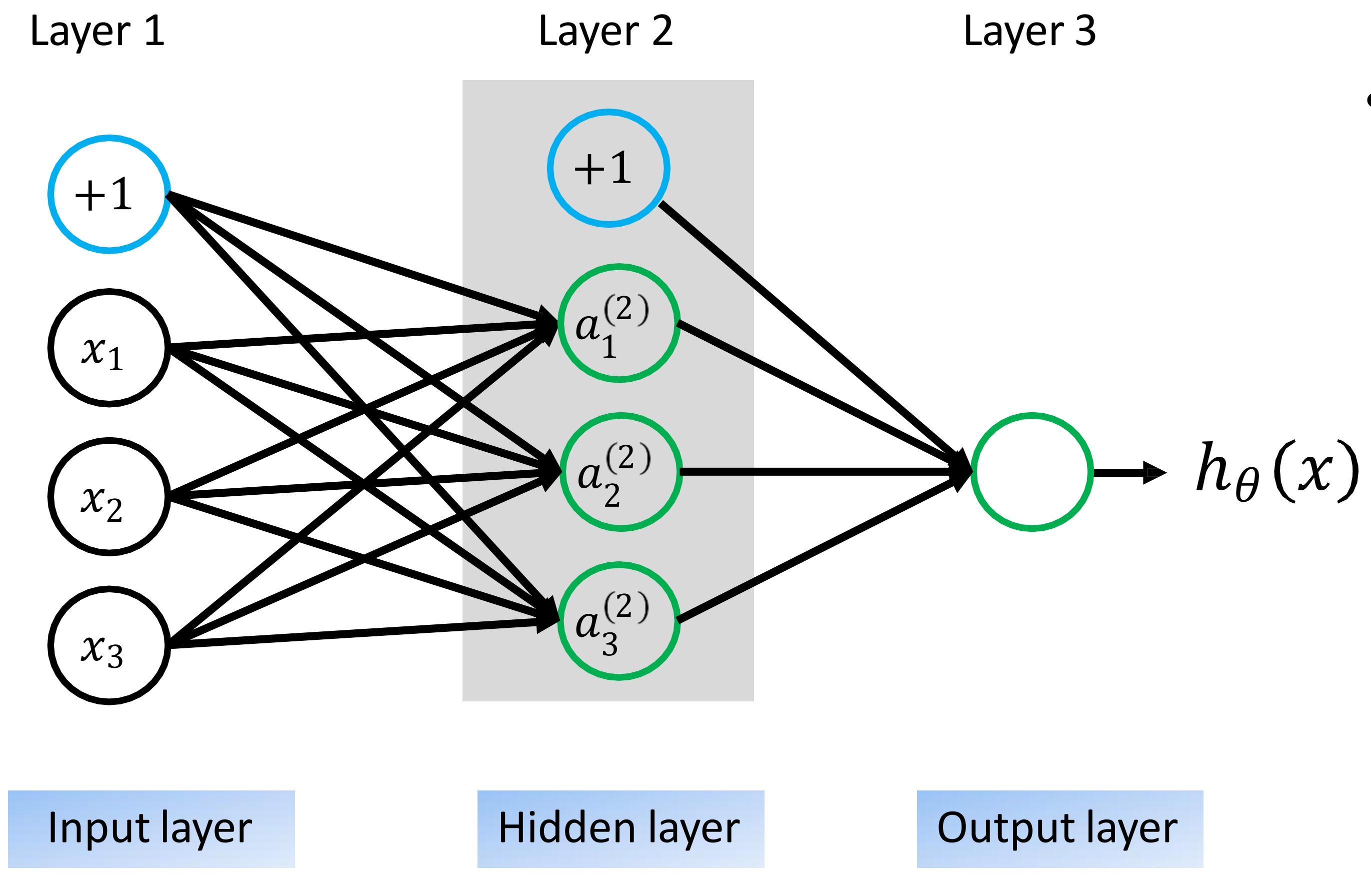
$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$	input
$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$	weights / parameters
$g(z)$	sigmoid (regression) activation function
$h_\theta(x)$	predicted output

Neural network

- Neural network consists of stack of neurons that takes in input x and output predicted value $h_\theta(x)$
- Given enough number of neurons, neural network are incredibly good at mapping input x to the actual output y.



Neural network (single hidden layer)



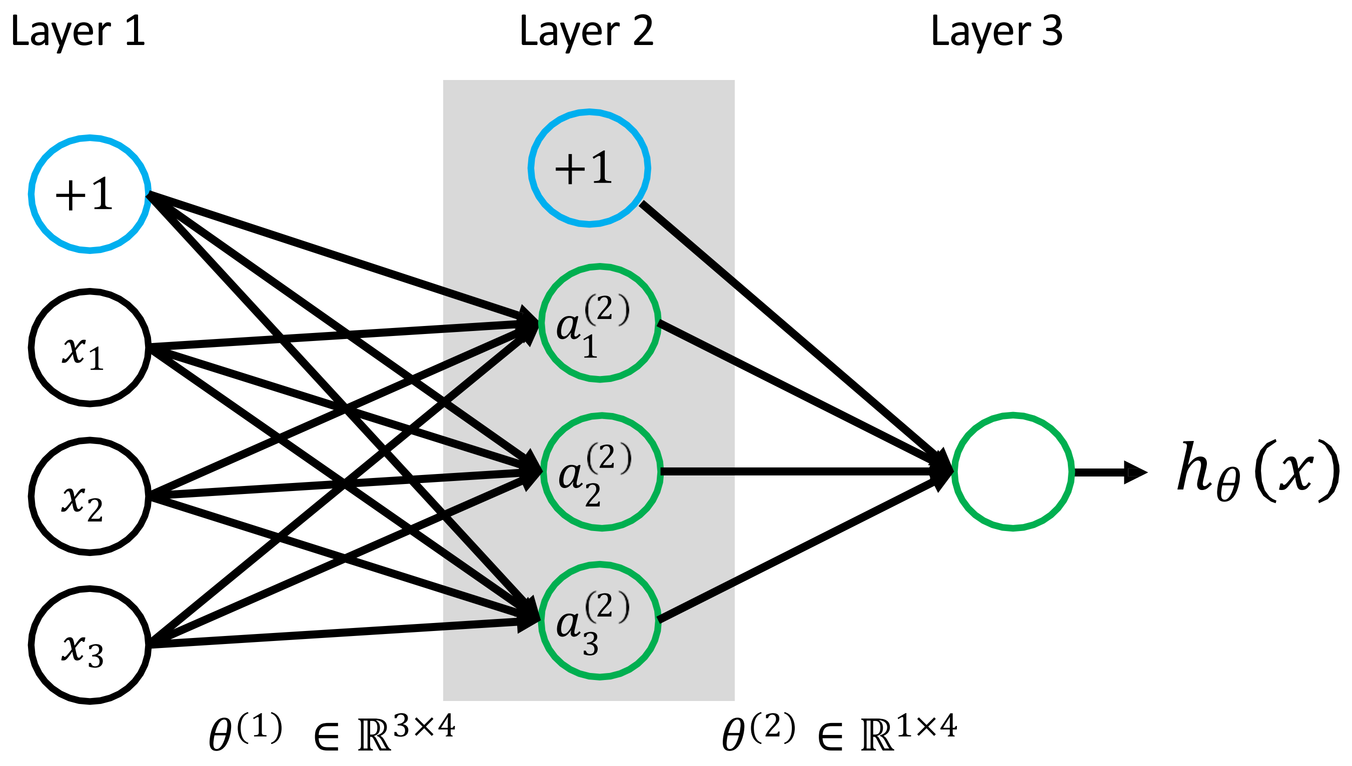
- Notation used in computation:

  $a_i^{(j)}$: activation of neuron i in layer j

  θ^j: matrix of weights make up the mapping function from layer j to $j+1$

  s_j: number of neurons in layer j not including bias

Neural network (single hidden layer)



- Notation used in computation:

  $a_i^{(j)}$: activation of neuron i in layer j

  θ^j: matrix of weights make up the mapping function from layer j to $j+1$

  s_j: number of neurons in layer j not including bias

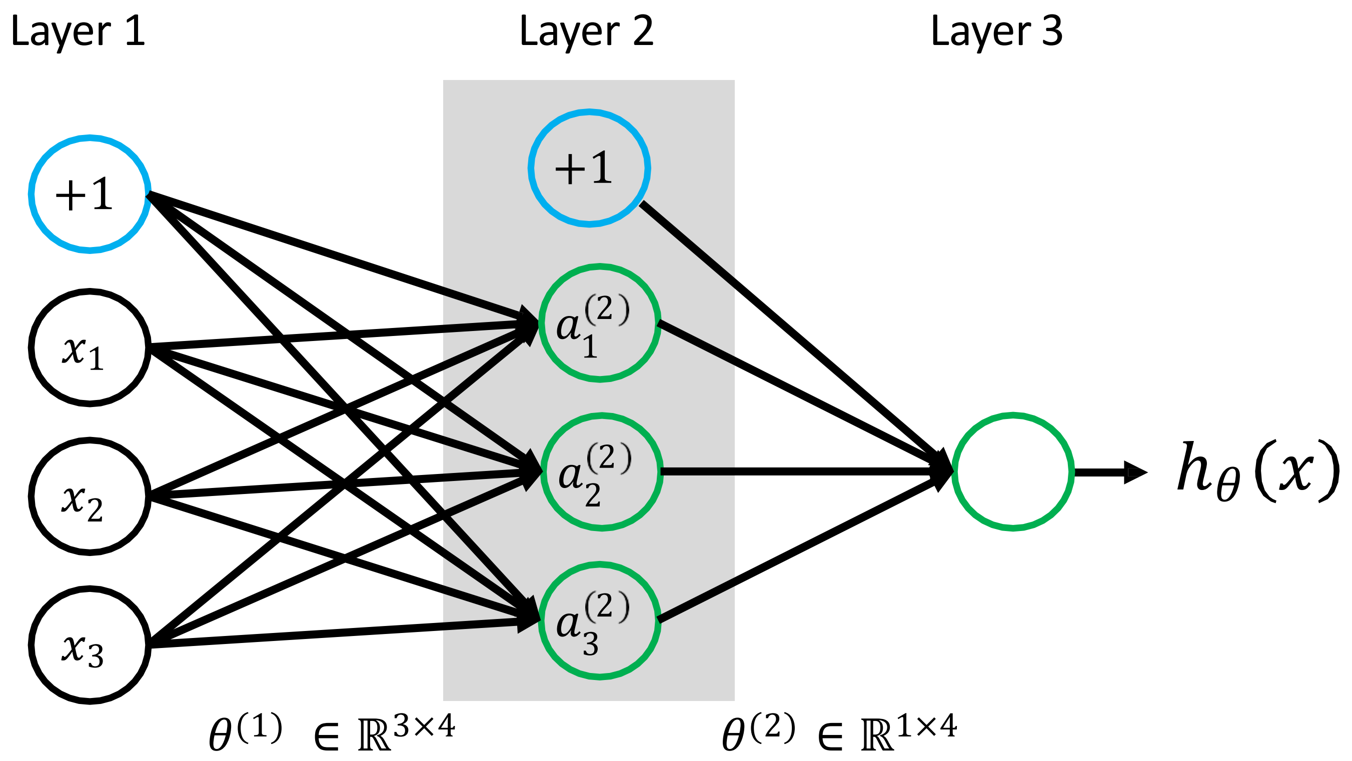
$$a_1^{(2)} = g(\theta_{10}^{(1)} \overset{1}{\cancel{x_o}} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\theta_{20}^{(1)} x_o + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\theta_{30}^{(1)} x_o + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3)$$
$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)} \overset{1}{\cancel{a_0^{(2)}}} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

Matrix format:

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

Neural network (single hidden layer)



$$a_1^{(2)} = g(\theta_{10}^{(1)}\overset{1}{\cancel{x_o}} + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\theta_{20}^{(1)}x_o + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\theta_{30}^{(1)}x_o + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3)$$
$$h_\theta(x) = a_1^{(3)} = g(\theta_{10}^{(2)}\overset{1}{\cancel{a_0^{(2)}}} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)})$$

- Notation used in computation:

  $a_i^{(j)}$: activation of neuron i in layer j

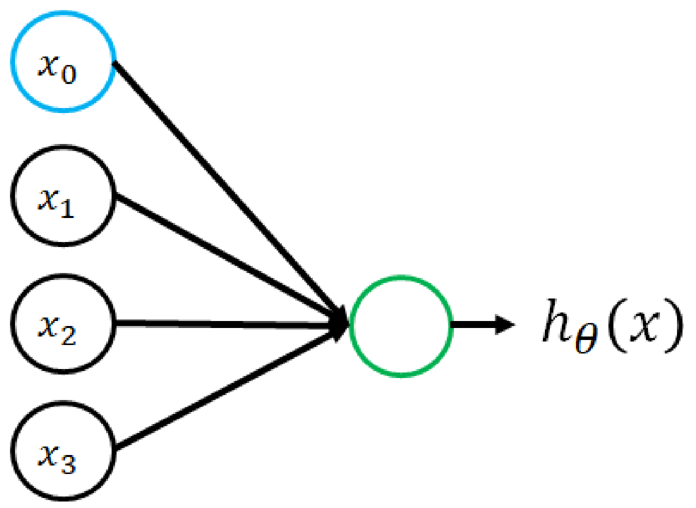
  θ^j: matrix of weights make up the mapping function from layer j to $j+1$

  s_j: number of neurons in layer j not including bias

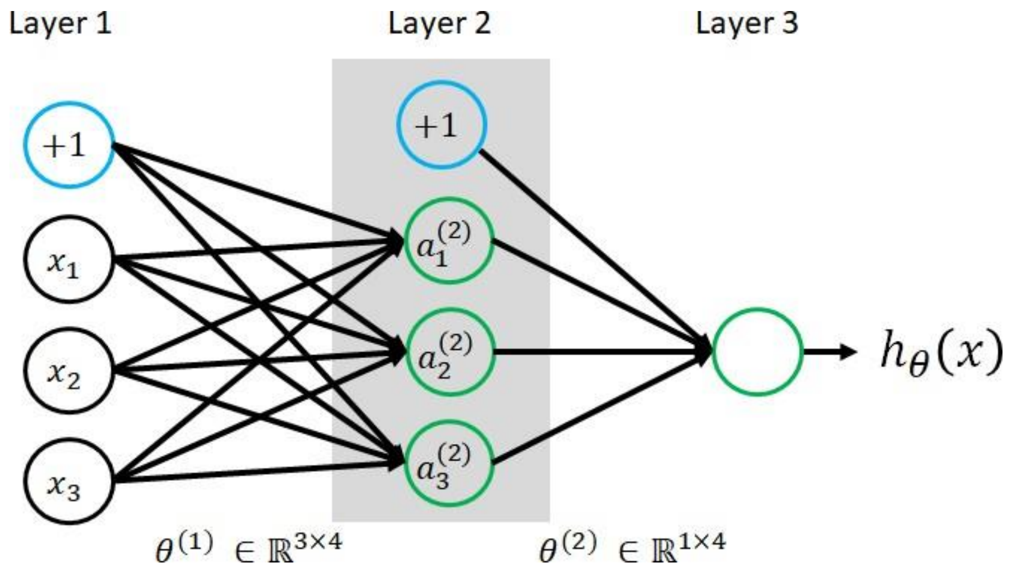
- Note that, if the network has s_j neurons in layer j and s_{j+1} neurons in layer $j+1$, then, the matrix dimension for θ^j is $s_{j+1} \times (s_j+1)$

Feature learning capability

Logistic regression



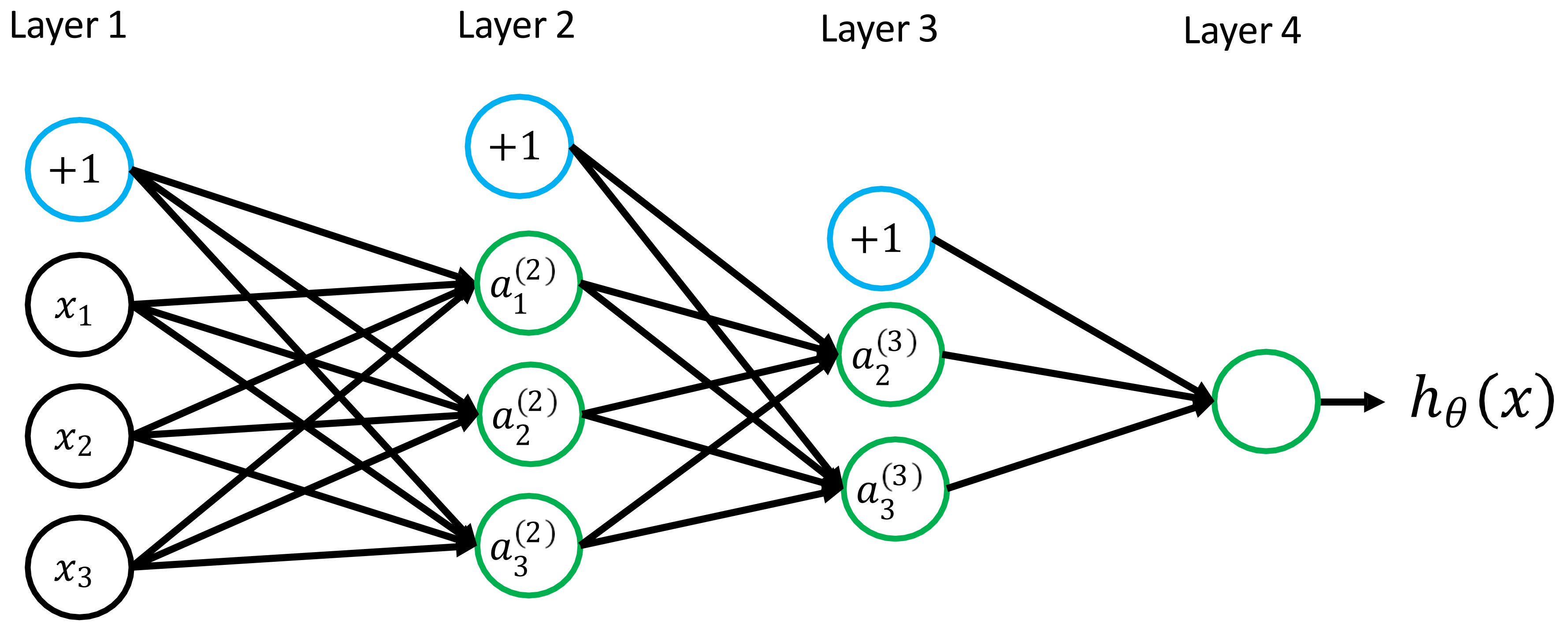
Neural Networks



- Logistic regression can be considered as a neural network without a hidden layer. Its mapping function connects the input x to the output y by a sigmoid function of a linear combination of features.

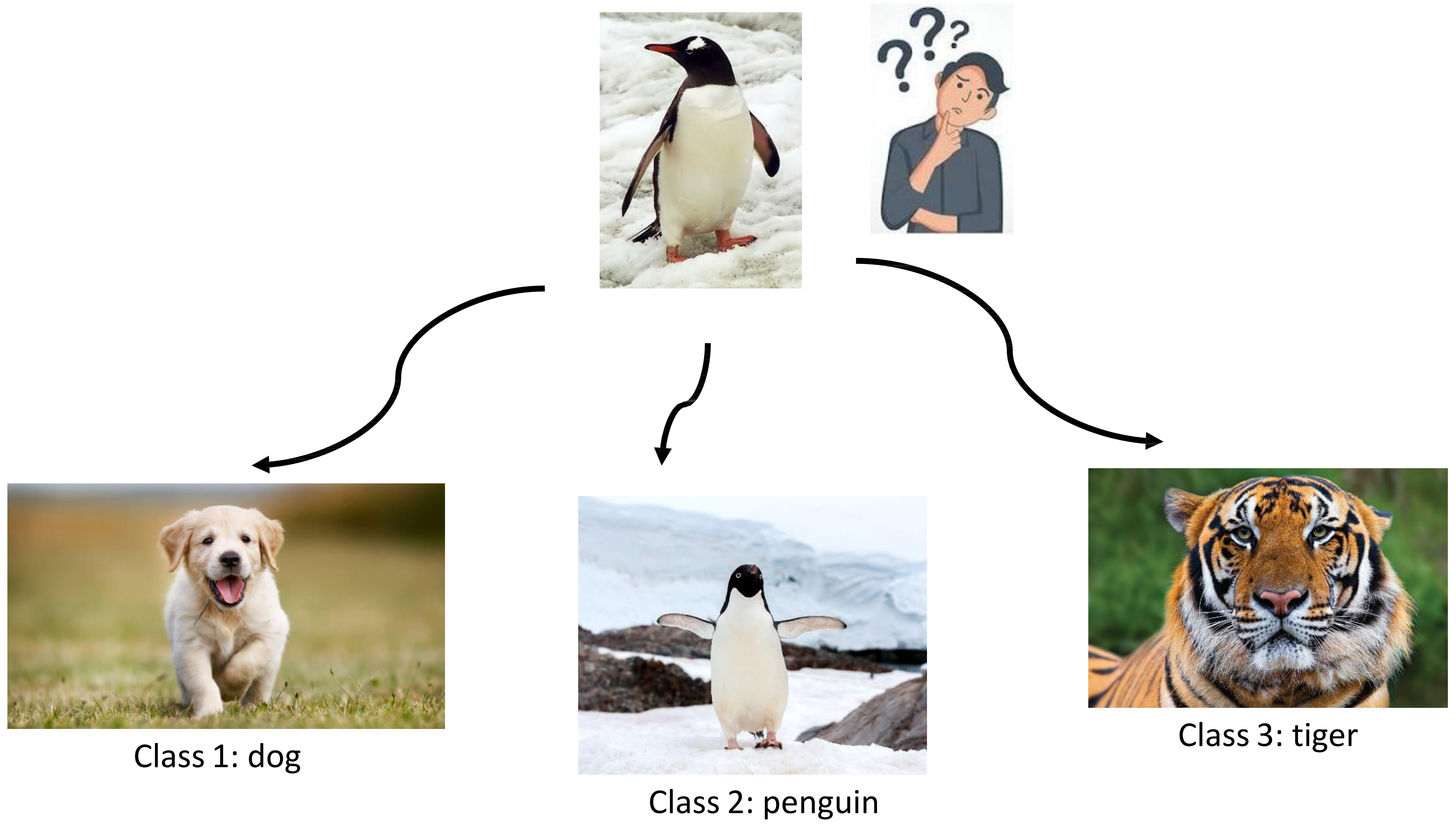
- A neural network is **more complex** than logistic regression. Normally, it has at least one hidden layer, which consists of the new features ($a^{(2)}$) that are learned as a function of $f(\theta^{(1)}, x)$.
  - With such flexibility in feature learning, neural networks can learn more complex features and output a more complex non-linear hypothesis.

Neural network with deeper layers

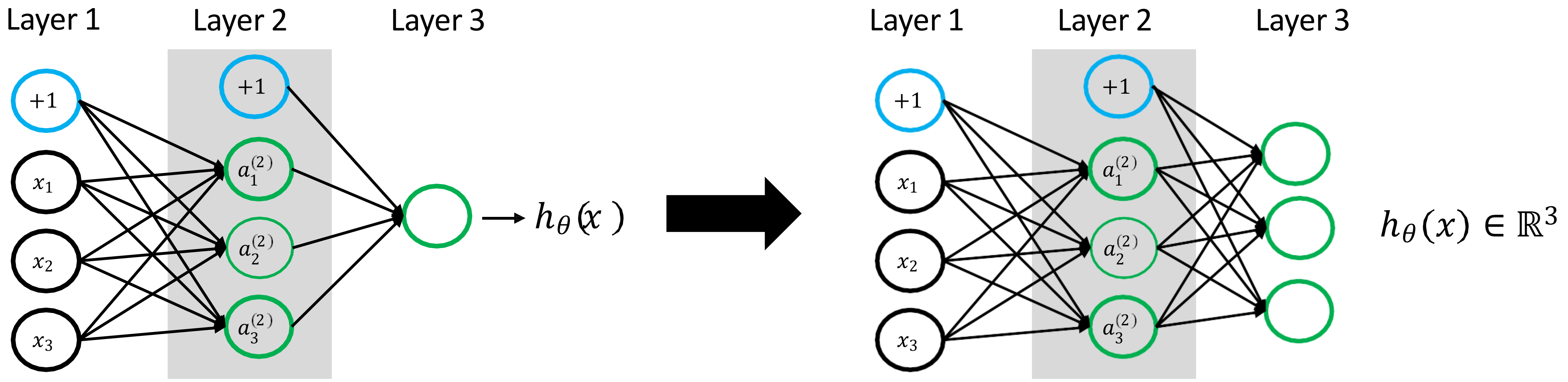


The complexity of the features increases

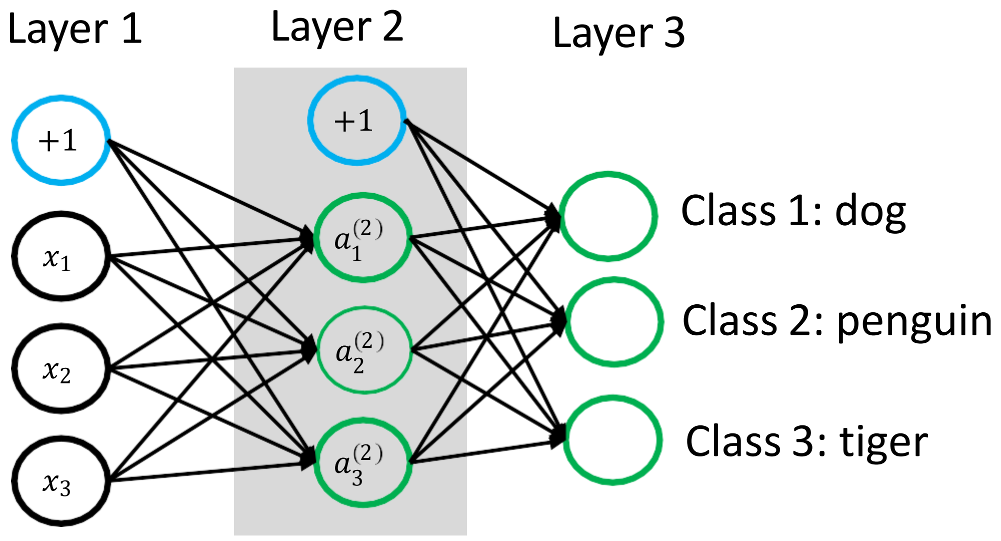
Multiclass classification



Multiple output neurons



Multiple output neurons



- Given *N* number of data points (training set) = $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots, (x^{(N)}, y^{(N)})$
- We no longer label $y^{(n)} \in \{1,2,3\}$ but $y^{(n)} \in \mathbb{R}^3$ as follows:

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Class 1: dog

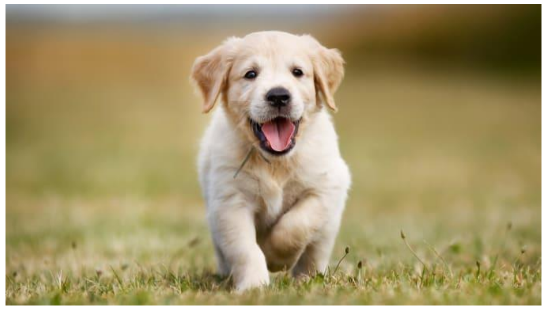
$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Class 2: penguin

$$y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Class 3: tiger

Multiple output neurons



$y = \begin{bmatrix}1\\0\\0\end{bmatrix}$

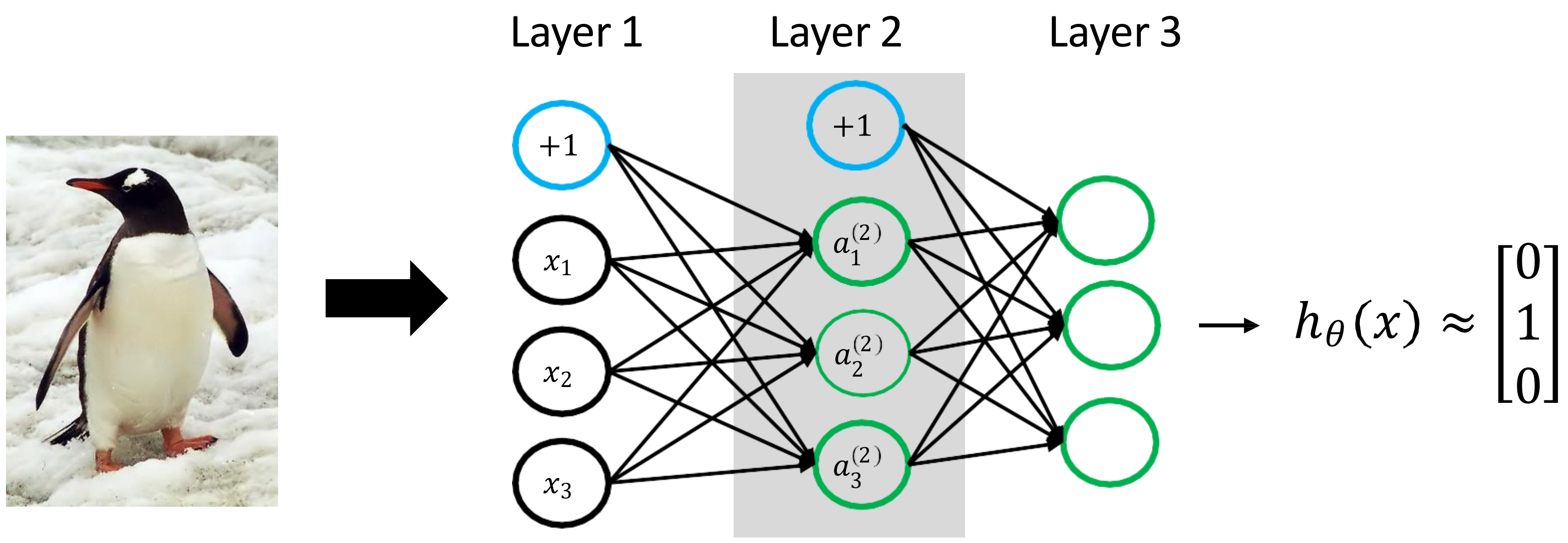
Class 1: dog

$y = \begin{bmatrix}0\\1\\0\end{bmatrix}$

Class 2: penguin

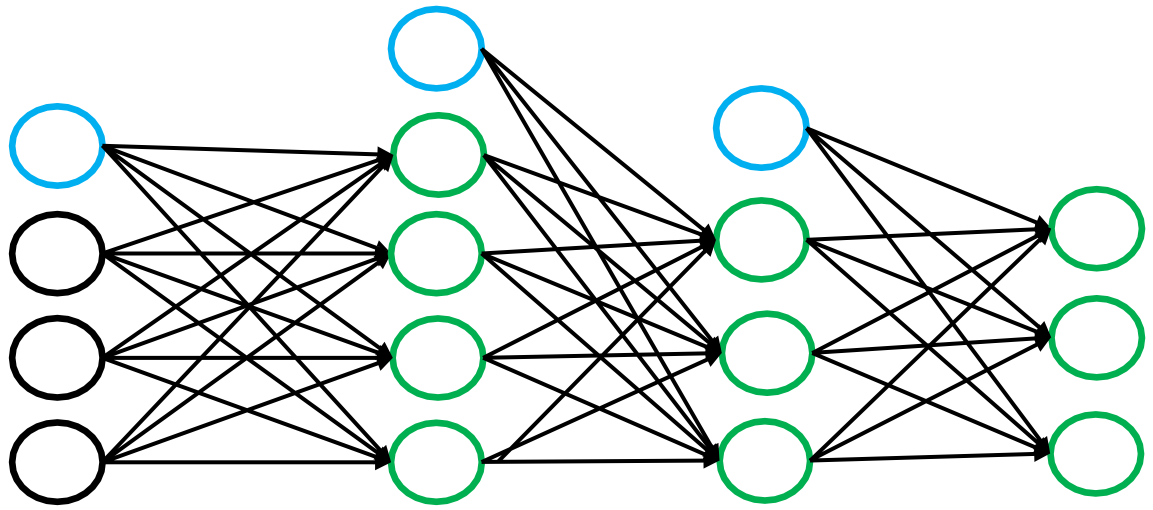
$y = \begin{bmatrix}0\\0\\1\end{bmatrix}$

Class 3: tiger



$\longrightarrow h_\theta(x) \approx \begin{bmatrix}0\\1\\0\end{bmatrix}$

Cost function (Classification)



$K = 3$
$L = 4$
$s_1 = 3 \quad s_2 = 4 \quad s_3 = 3 \quad s_L = 3$
$y \in \mathbb{R}^3$
$h_\theta(x) \in \mathbb{R}^3$

- Terminology used:

K : number of output neurons
L : total number of layers in network
s_l: Number of neurons in layer l (not including bias)

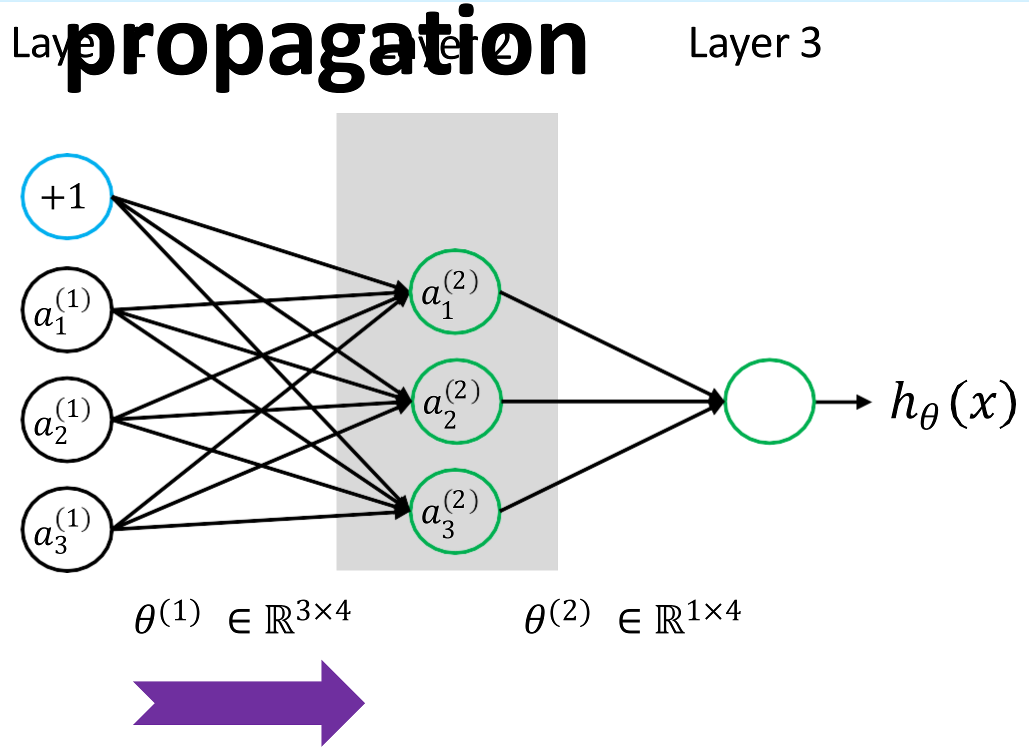
- For multiclass classification (K classes where $K \geq 3$)

$$y \in \mathbb{R}^K$$
$$h_\theta(x) \in \mathbb{R}^K$$

- For binary classification:

$$y \in \{0,1\}$$
$$h_\theta(x) \in \mathbb{R}$$

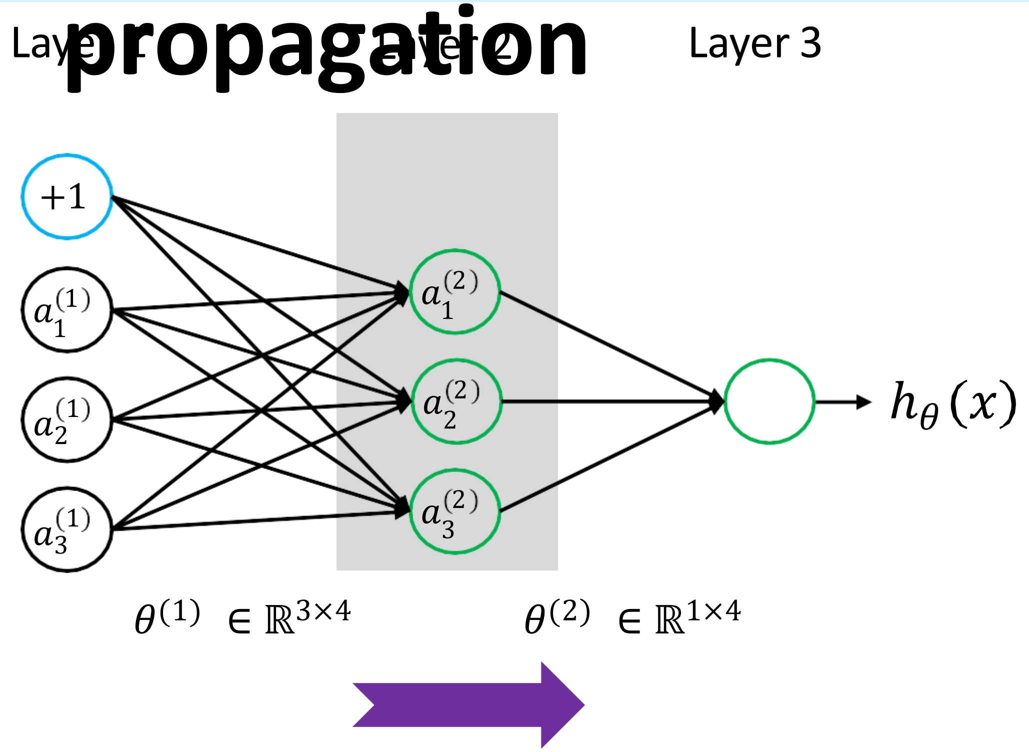
Gradient computation - forward propagation



$$z^{(2)} = \theta^{(1)} a^{(1)}$$

$$a_1^{(2)} = g(\theta_{10}^{(1)} a_0^{(1)} + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)} + \theta_{13}^{(1)} a_3^{(1)})$$
$$a_2^{(2)} = g(\theta_{20}^{(1)} a_0^{(1)} + \theta_{21}^{(1)} a_1^{(1)} + \theta_{22}^{(1)} a_2^{(1)} + \theta_{23}^{(1)} a_3^{(1)})$$
$$a_3^{(2)} = g(\theta_{30}^{(1)} a_0^{(1)} + \theta_{31}^{(1)} a_1^{(1)} + \theta_{32}^{(1)} a_2^{(1)} + \theta_{33}^{(1)} a_3^{(1)})$$

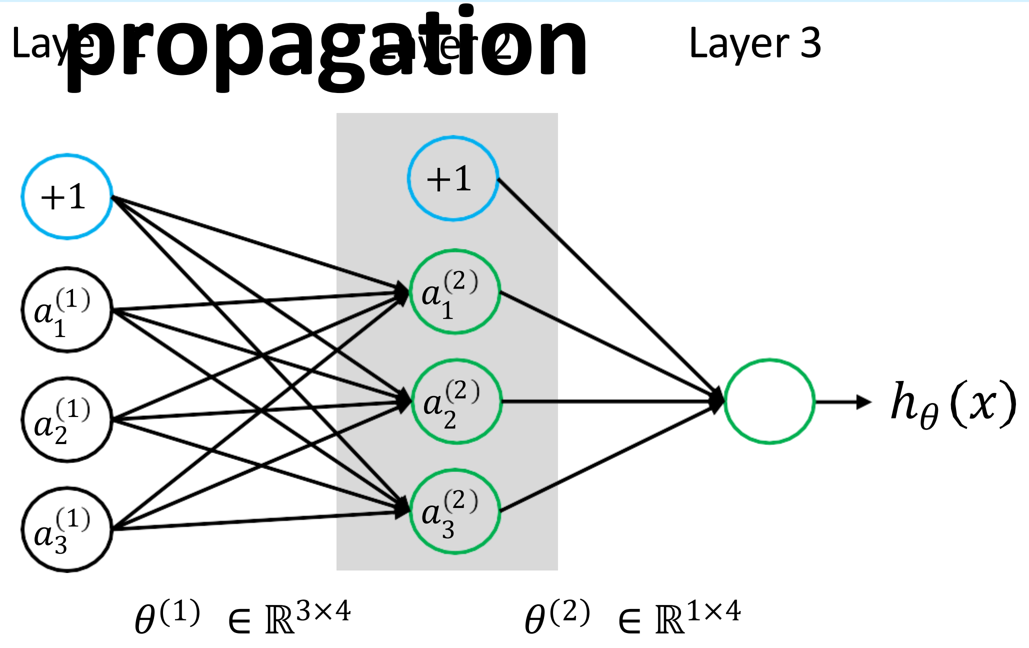
Gradient computation - forward propagation



$$z^{(2)} = \theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$

$$\boxed{a_1^{(2)}} = g(\theta_{10}^{(1)} a_0^{(1)} + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)} + \theta_{13}^{(1)} a_3^{(1)})$$
$$\boxed{a_2^{(2)}} = g(\theta_{20}^{(1)} a_0^{(1)} + \theta_{21}^{(1)} a_1^{(1)} + \theta_{22}^{(1)} a_2^{(1)} + \theta_{23}^{(1)} a_3^{(1)})$$
$$\boxed{a_3^{(2)}} = g(\theta_{30}^{(1)} a_0^{(1)} + \theta_{31}^{(1)} a_1^{(1)} + \theta_{32}^{(1)} a_2^{(1)} + \theta_{33}^{(1)} a_3^{(1)})$$

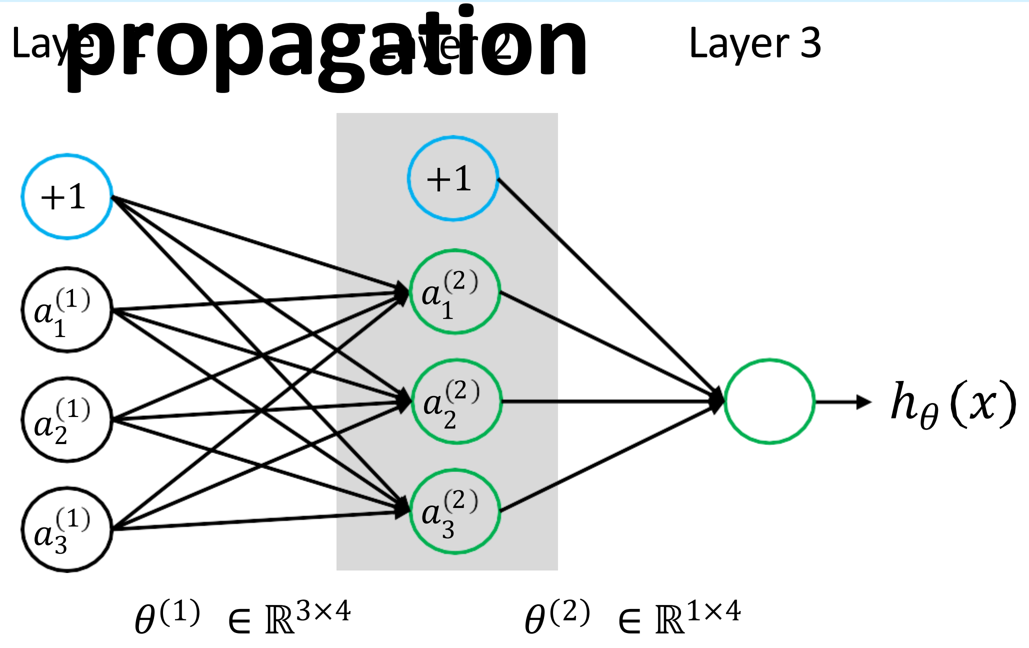
Gradient computation - forward propagation



$$z^{(2)} = \theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)} a^{(2)}$$

$$a_1^{(2)} = g(\theta_{10}^{(1)} a_0^{(1)} + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)} + \theta_{13}^{(1)} a_3^{(1)})$$
$$a_2^{(2)} = g(\theta_{20}^{(1)} a_0^{(1)} + \theta_{21}^{(1)} a_1^{(1)} + \theta_{22}^{(1)} a_2^{(1)} + \theta_{23}^{(1)} a_3^{(1)})$$
$$a_3^{(2)} = g(\theta_{30}^{(1)} a_0^{(1)} + \theta_{31}^{(1)} a_1^{(1)} + \theta_{32}^{(1)} a_2^{(1)} + \theta_{33}^{(1)} a_3^{(1)})$$
$$a_1^{(3)} = g(\boxed{\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)}})$$

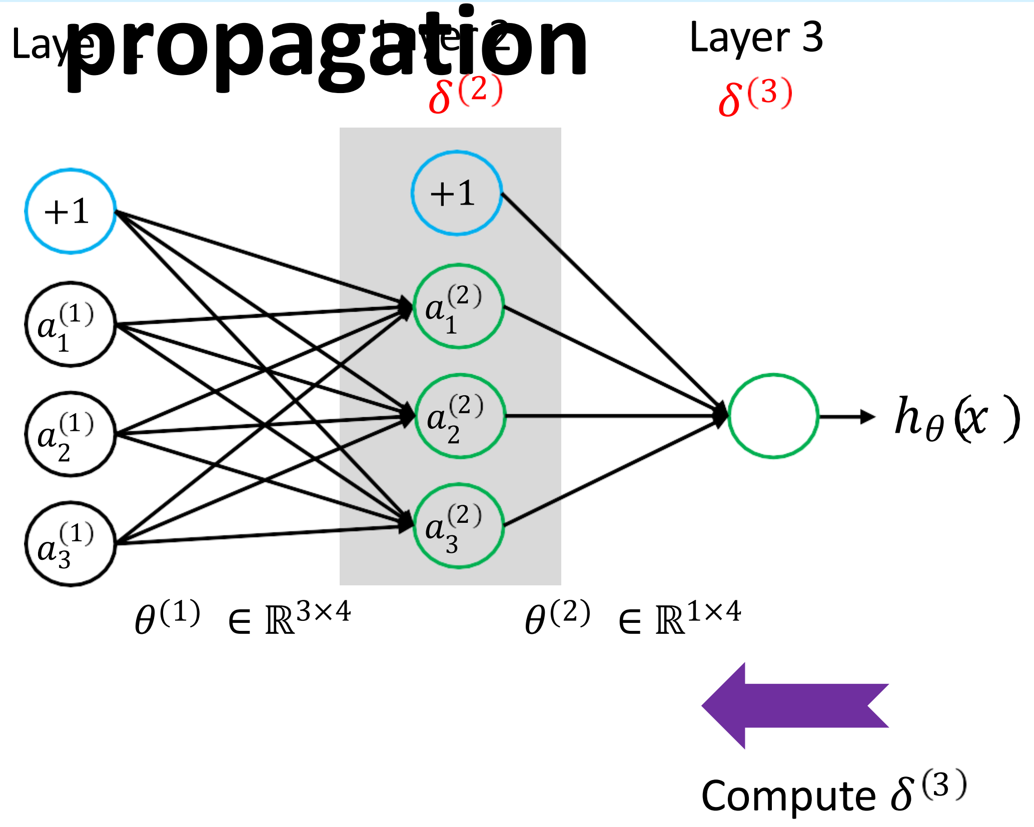
Gradient computation - forward propagation



$$z^{(2)} = \theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$

$$a_1^{(2)} = g(\theta_{10}^{(1)} a_0^{(1)} + \theta_{11}^{(1)} a_1^{(1)} + \theta_{12}^{(1)} a_2^{(1)} + \theta_{13}^{(1)} a_3^{(1)})$$
$$a_2^{(2)} = g(\theta_{20}^{(1)} a_0^{(1)} + \theta_{21}^{(1)} a_1^{(1)} + \theta_{22}^{(1)} a_2^{(1)} + \theta_{23}^{(1)} a_3^{(1)})$$
$$a_3^{(2)} = g(\theta_{30}^{(1)} a_0^{(1)} + \theta_{31}^{(1)} a_1^{(1)} + \theta_{32}^{(1)} a_2^{(1)} + \theta_{33}^{(1)} a_3^{(1)})$$
$$\boxed{a_1^{(3)}} = g(\theta_{10}^{(2)} a_0^{(2)} + \theta_{11}^{(2)} a_1^{(2)} + \theta_{12}^{(2)} a_2^{(2)} + \theta_{13}^{(2)} a_3^{(2)})$$

Gradient computation - backward propagation



$\delta_j^{(\,)}$ = error of j-th neuron in layer l

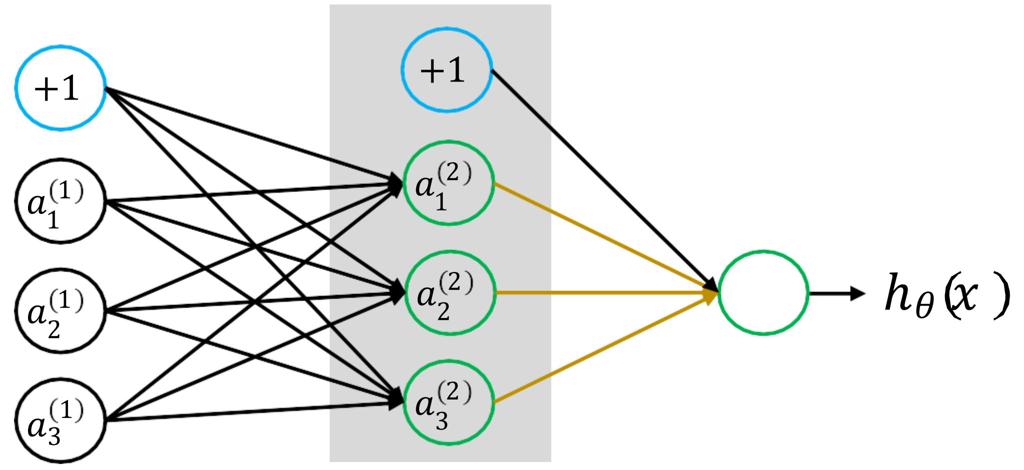
$$z^{(2)} = \theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$h_\theta(x) = a^{(3)}$$

$$\delta^{(3)} = \frac{\delta J(\theta)}{\delta z^{(3)}} = \frac{\delta J(\theta)}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} = (a^{(3)} - y)\, g'(z^{(3)})$$

Gradient computation - backward propagation

Layer 1 Layer 2 Layer 3

$\delta^{(2)}$ $\delta^{(3)}$



$\theta^{(1)} \in \mathbb{R}^{3\times4}$ $\theta^{(2)} \in \mathbb{R}^{1\times4}$

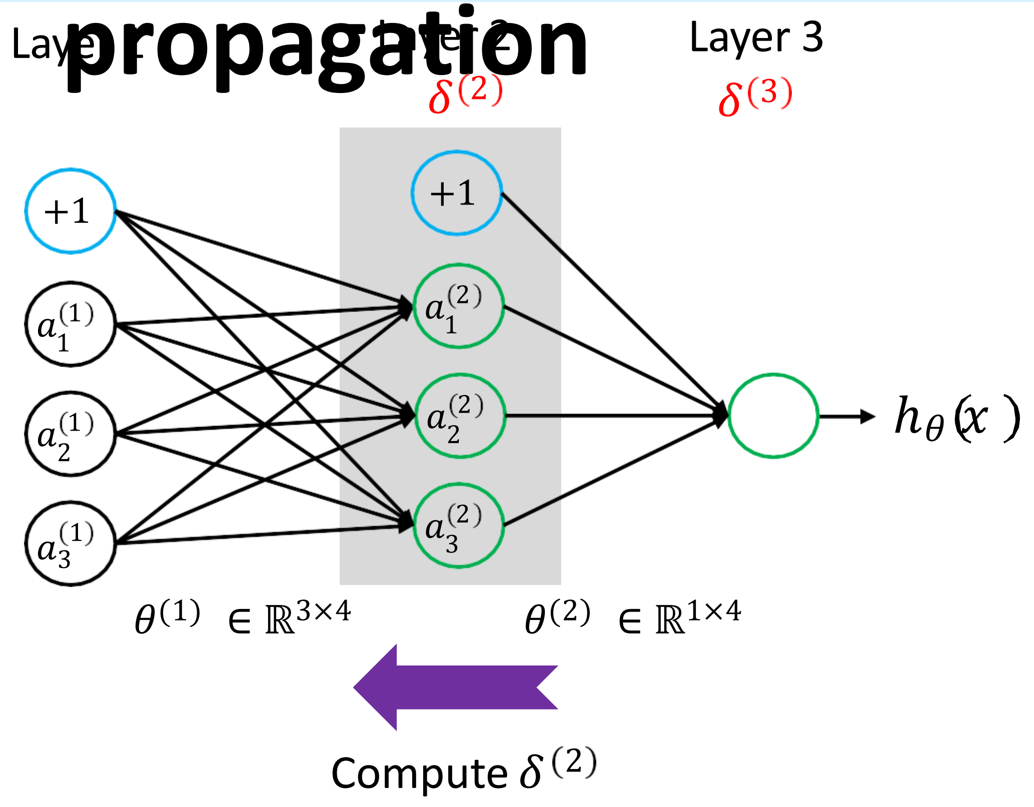
Compute $\frac{\delta J(\theta)}{\delta\theta^{(2)}}$

$\delta_j^{(l)}$= error of j-th neuron in layer l

$$z^{(2)} = \theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)}a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$h_\theta(x) = a^{(3)}$$

$$\delta^{(3)} = \frac{\delta J(\theta)}{\delta z^{(3)}} = \frac{\delta J(\theta)}{\delta a^{(3)}}\frac{\delta a^{(3)}}{\delta z^{(3)}} = (a^{(3)} - y)\, g'(z^{(3)})$$
$$\frac{\delta J(\theta)}{\delta\theta^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}}\frac{\delta z^{(3)}}{\delta\theta^{(2)}} = \delta^{(3)}a^{(2)}$$

Gradient computation - backward propagation



$\delta_j^{(l)}$ = error of j-th neuron in layer l

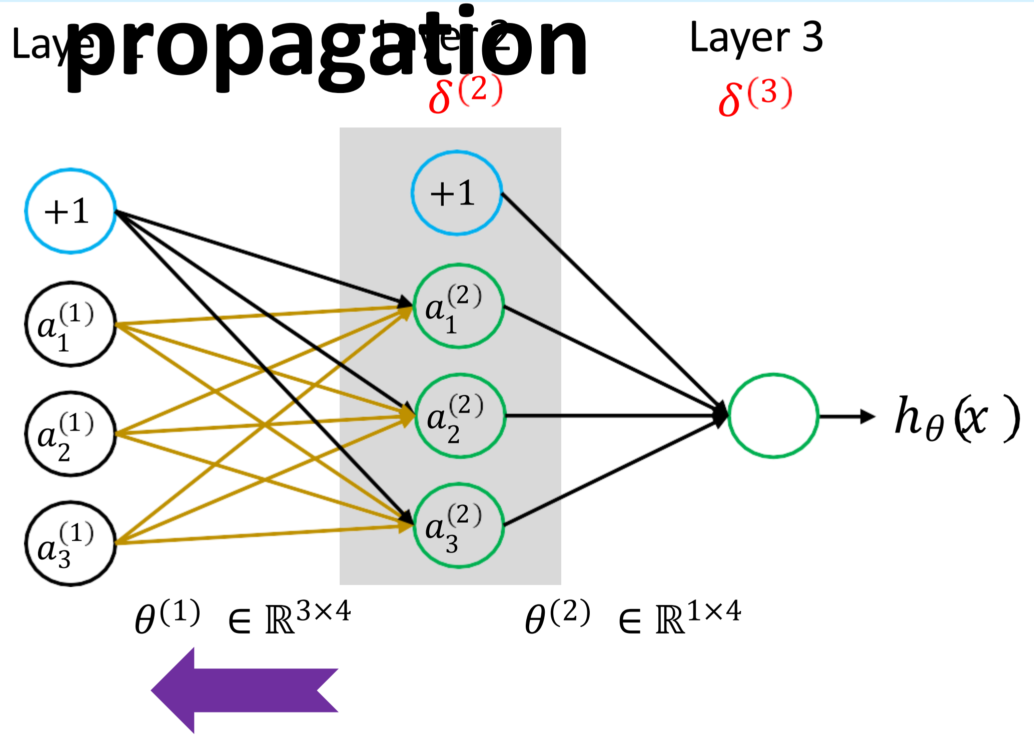
$$z^{(2)} = \theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$h_\theta(x) = a^{(3)}$$

$$\delta^{(3)} = \frac{\delta J(\theta)}{\delta z^{(3)}} = \frac{\delta J(\theta)}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} = (a^{(3)} - y)\, g'(z^{(3)})$$

$$\frac{\delta J(\theta)}{\delta \theta^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta \theta^{(2)}} = \delta^{(3)} a^{(2)}$$

$$\delta^{(2)} = \frac{\delta J(\theta)}{\delta z^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}}$$
$$= \delta^{(3)} (\theta^{(2)})\, g'(z^{(2)})$$
$$= (\theta^{(2)})^{(T)} \delta^{(3)}\, g'(z^{(2)})$$

Gradient computation - backward propagation



Compute $\frac{\delta J(\theta}{\delta\theta^{(1)}}$

$\delta_j^{(\;)}$ = error of j-th neuron in layer l

$$z^{(2)} = \theta^{(1)}a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)}a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$h_\theta(x) = a^{(3)}$$

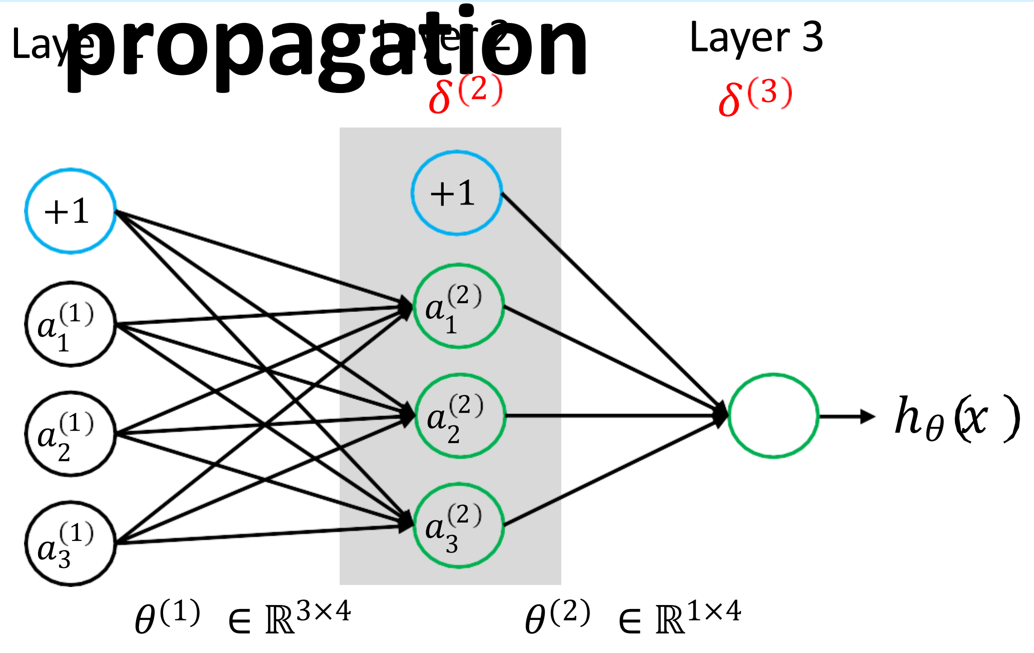
$$\delta^{(3)} = \frac{\delta J(\theta)}{\delta z^{(3)}} = \frac{\delta J(\theta)}{\delta a^{(3)}}\frac{\delta a^{(3)}}{\delta z^{(3)}} = (a^{(3)} - y)\, g'(z^{(3)})$$

$$\frac{\delta J(\theta)}{\delta\theta^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}}\frac{\delta z^{(3)}}{\delta\theta^{(2)}} = \delta^{(3)}a^{(2)}$$

$$\delta^{(2)} = \frac{\delta J(\theta)}{\delta z^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}}\frac{\delta z^{(3)}}{\delta a^{(2)}}\frac{\delta a^{(2)}}{\delta z^{(2)}}$$
$$= \delta^{(3)}(\theta^{(2)})\, g'(z^{(2)})$$
$$= (\theta^{(2)})^{(T)}\,\delta^{(3)} \odot g'(z^{(2)})$$

$$\frac{\delta J(\theta)}{\delta\theta^{(1)}} = \frac{\delta J(\theta)}{\delta z^{(3)}}\frac{\delta z^{(3)}}{\delta a^{(2)}}\frac{\delta a^{(2)}}{\delta z^{(2)}}\frac{\delta z^{(2)}}{\delta\theta^{(1)}} = \delta^{(2)}a^{(1)}$$

Gradient computation - backward propagation



$\delta_j^{(l)}$ = error of j-th neuron in layer l

$$z^{(2)} = \theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$z^{(3)} = \theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)})$$
$$h_\theta(x) = a^{(3)}$$

$$\delta^{(3)} = \frac{\delta J(\theta)}{\delta z^{(3)}} = \frac{\delta J(\theta)}{\delta a^{(3)}} \frac{\delta a^{(3)}}{\delta z^{(3)}} = (a^{(3)} - y)\, g'(z^{(3)})$$

$$\frac{\delta J(\theta)}{\delta \theta^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta \theta^{(2)}} = \delta^{(3)} a^{(2)}$$

$$\delta^{(2)} = \frac{\delta J(\theta)}{\delta z^{(2)}} = \frac{\delta J(\theta)}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}}$$
$$= \delta^{(3)} (\theta^{(2)})\, g'(z^{(2)})$$
$$= (\theta^{(2)})^{(T)}\, \delta^{(3)} \odot g'(z^{(2)})$$

$$\frac{\delta J(\theta)}{\delta \theta^{(1)}} = \frac{\delta J(\theta)}{\delta z^{(3)}} \frac{\delta z^{(3)}}{\delta a^{(2)}} \frac{\delta a^{(2)}}{\delta z^{(2)}} \frac{\delta z^{(2)}}{\delta \theta^{(1)}} = \delta^{(2)} a^{(1)}$$

No $\delta^{(1)}$!

Gradients are computed using chain rule of differentiation

Summary of equation of backpropagation:

$$\delta^{(L)} = \frac{\delta J(\theta)}{\delta a^{(L)}} \frac{\delta a^{(L)}}{\delta z^{(L)}}$$

$$\delta^{(l)} = (\theta^{(l)})^{(T)}\, \delta^{(l+1)} \odot g'(z^{(l)})$$

$$\frac{\delta J(\theta)}{\delta \theta_{jk}^{(l)}} = \delta_j^{(l+1)} a_k^{(l)}$$

$$\frac{\delta J(\theta)}{\delta \theta_{j0}^{(l)}} = \delta_j^{(l+1)}$$

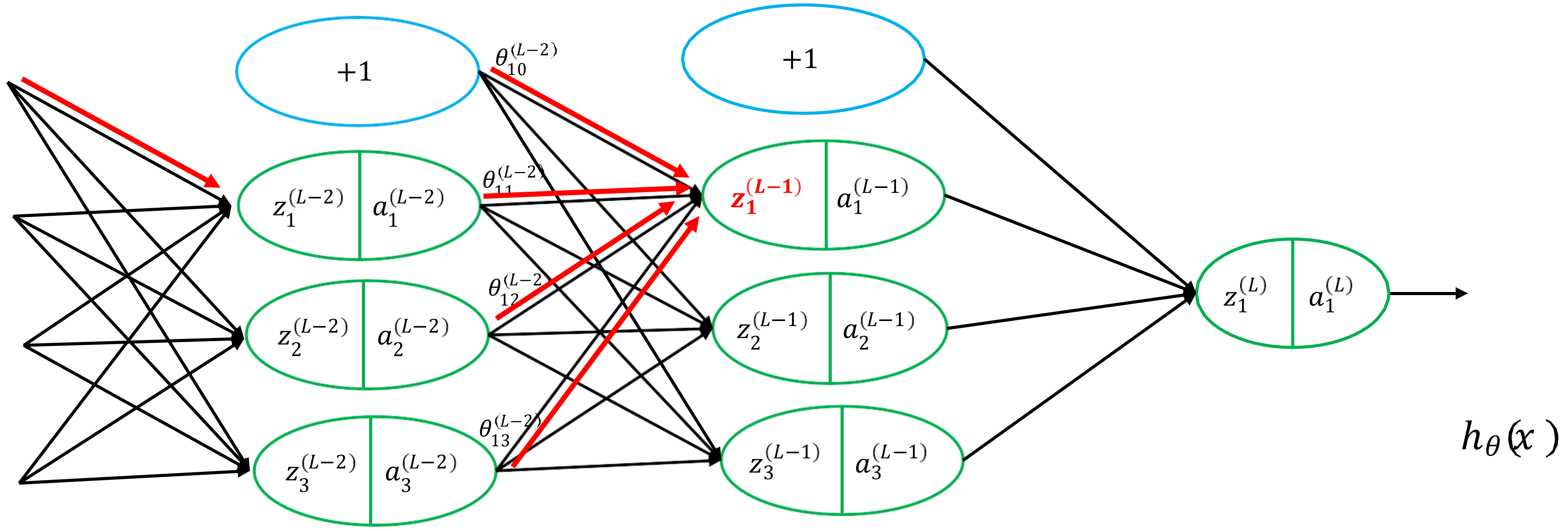
Backpropagation algorithm

1. **Input**: given *N* number of data points (training set) = $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$,……, $(x^{(N)}, y^{(N)})$
2. **Feedforward**: For each $l = 1,2, \ldots, L$, compute $z^{(l+1)} = \theta^{(l)} a^{(l)}$ and $a^{(l+1)} = g(z^{(l+1)})$
3. **Output error**: Compute $\delta^{(L)}$
4. **Backpropagate the error**: For each $l = 2, \ldots, L$, compute $\delta^{(l)}$
5. **Gradient computation**: Compute weights $\frac{\delta J(\theta)}{\delta \theta_{jk}^{(l)}}$ and biases $\frac{\delta J(\theta)}{\delta \theta_{j0}}$
6. **Optimization algorithm**: You can use any gradient descent optimization algorithm to minimize the error function $J(\theta)$.

Backpropagation intuition

- Forward propagation simply computes the weighted sum of $z^{(l)}$ from the left to the right.

E.g. $z^{(L-1)} = \theta_{10}^{(L-2)} a_0^{(L-2)} + \theta_{11}^{(L-2)} a_1^{(L-2)} + \theta_{12}^{(L-2)} a_2^{(L-2)} + \theta_{13}^{(L-2)} a_3^{(L-2)}$

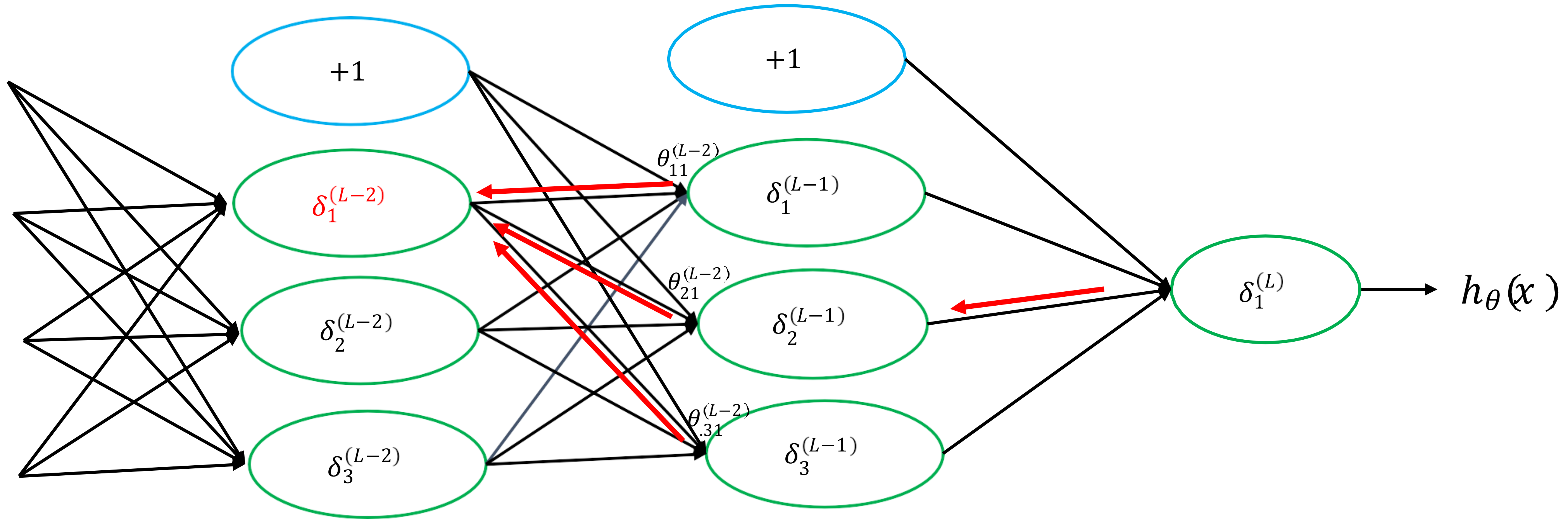


Backpropagation intuition

- Backward propagation computes the error term $\delta^{(l)}$ from the right to the left.

$$\text{E.g. } \delta_1^{(L-2)} = \theta_{11}^{(L-2)}\delta_1^{(L-1)} + \theta_{21}^{(L-2)}\delta_2^{(L-1)} + \theta_{31}^{(L-2)}\delta_3^{(L-1)}$$

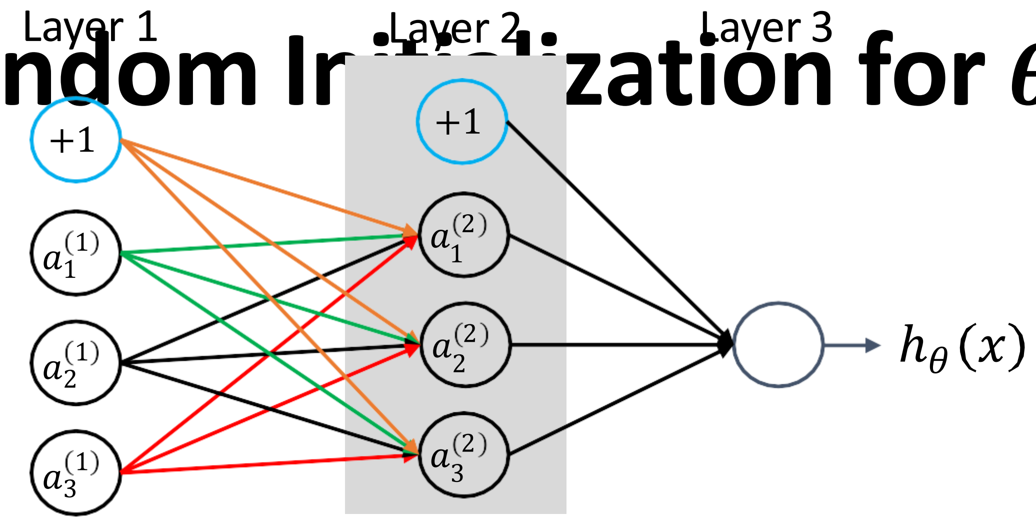
- Measure of how much will we like to modify the weights $\theta_{jk}^{(l)}$ in order to affect this intermediate value of the calculation so as to influence the final result $h_\theta(x)$ and the cost function $J(\theta)$.



$( )$
jk

Random Initialization for θ

l

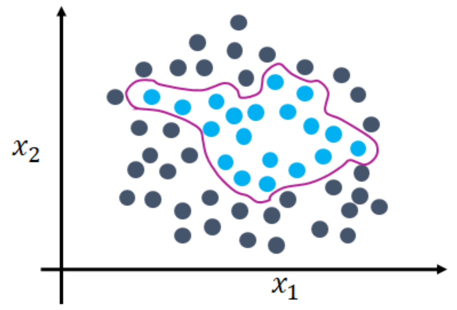


- By initializing the same weights for all layers, all hidden units calculate the same function.
- This is equivalent to learning the same features for each neuron in a hidden layer.
- To break the symmetry, initialize with random number (small value close to zero)

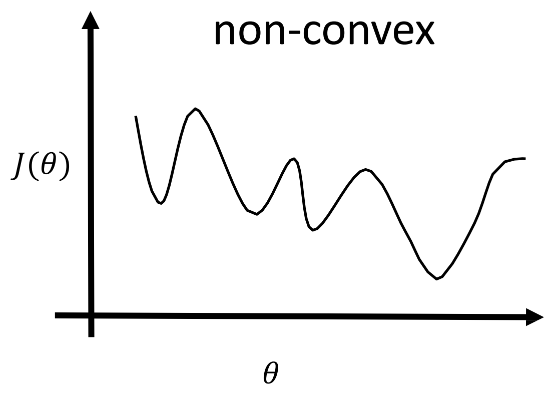
LR vs ANN

Neural Network

- Neural networks are much better for a **complex nonlinear hypothesis.**

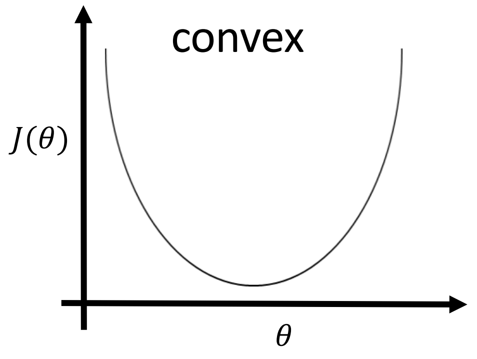


- A neural network is in general **not convex**. It can suffer from multiple local minima.



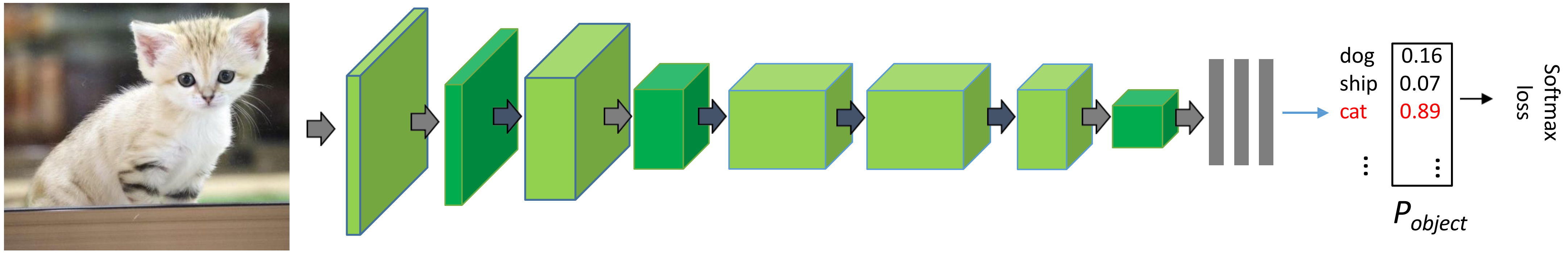
Logistic regression

- LR is useful for dataset where the classes are more or less “linearly separable”
- For “relatively” very **small** dataset.
- It is a great, robust model for **simple** classification tasks
- LR cost function (or max-entropy) is **convex**, and thus we are guaranteed to find the global cost minimum.



Next Lecture

❖ Convolutional neural network



dog
ship
cat
0.16
0.07
0.89
P_{object}
Softmax loss