



Machine Learning

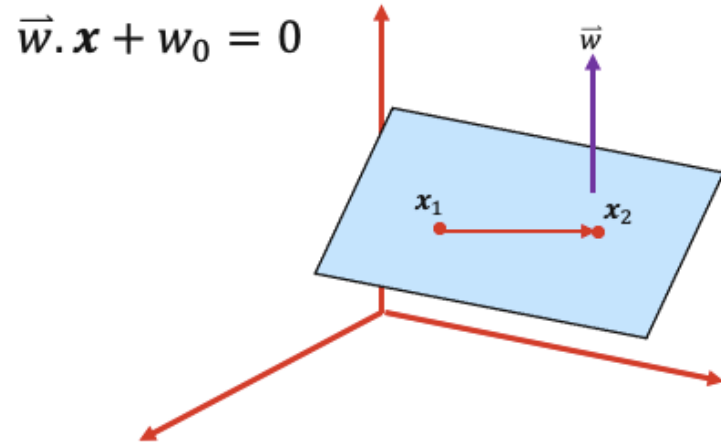
Prof. Adil Khan

Today's Objectives

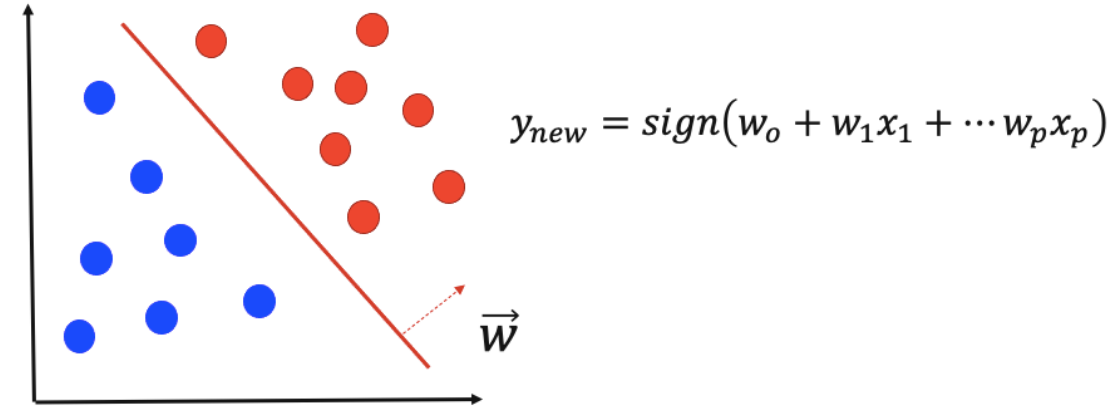
1. A quick recap of last week
2. What are Deep Artificial Neural Networks (DNNs)?
3. Why are they motivated?
 - Feature Learning, Hierarchical Features, Non-linearity
4. Neural Learning
 - What is Backpropagation? How does it work?

Recap (1)

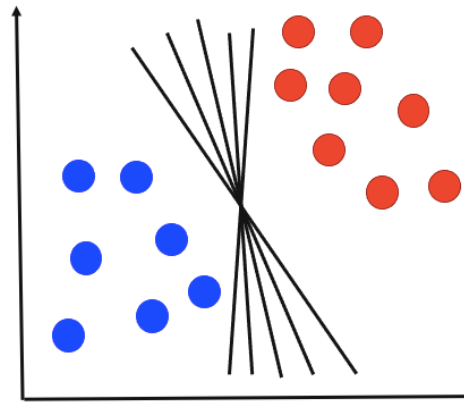
Hyperplane and Normal Vector



Separating Hyperplane



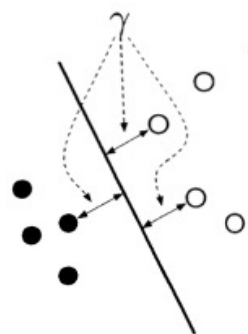
Infinite Many Such Hyperplanes



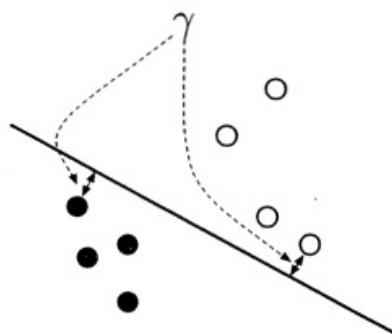
Recap (2)

Optimal Decision Bounday (or Separating Plane)

- One that maximizes the **Margin**



(a) The decision boundary that maximises the margin



(b) A non-optimal decision boundary

$$\gamma = \frac{1}{\|w\|}$$

Maximizing the Margin

- Thus our learning objective becomes

$$\operatorname{argmax}_w \frac{1}{\|w\|}$$

subject to $y_i(w_0 + w_1x_1 + \dots w_px_p) \geq 1$ for all i

- Practically, it is easier to solve the following (equivalent) optimization problem

$$\operatorname{argmin}_w \frac{1}{2} \|w\|^2$$

subject to $y_i(w_0 + w_1x_1 + \dots w_px_p) \geq 1$ for all i

Recap (3)

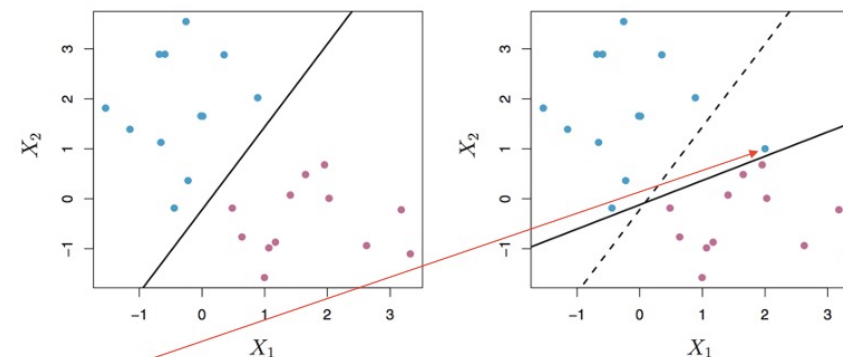
Thus

- In Support Vector Machines
- Given a data set $\mathbb{D} = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$, we solve the following

$$\underset{\mathbf{w}}{\operatorname{argmin}} \quad \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w}^T \mathbf{x}_i + w_0) - 1)$$

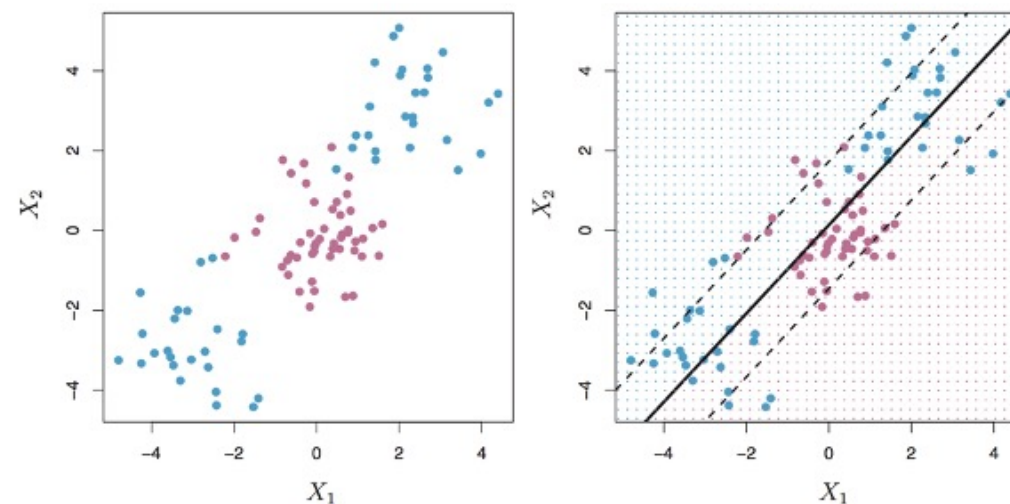
subject to $\alpha_i \geq 0$ for all i

Issue (1): Sensitivity to Noise



An **additional** blue observation has been added, leading to a dramatic shift in the maximal margin hyperplane. This is kind of **overfitting**.

Issue (2): Non-linear Data



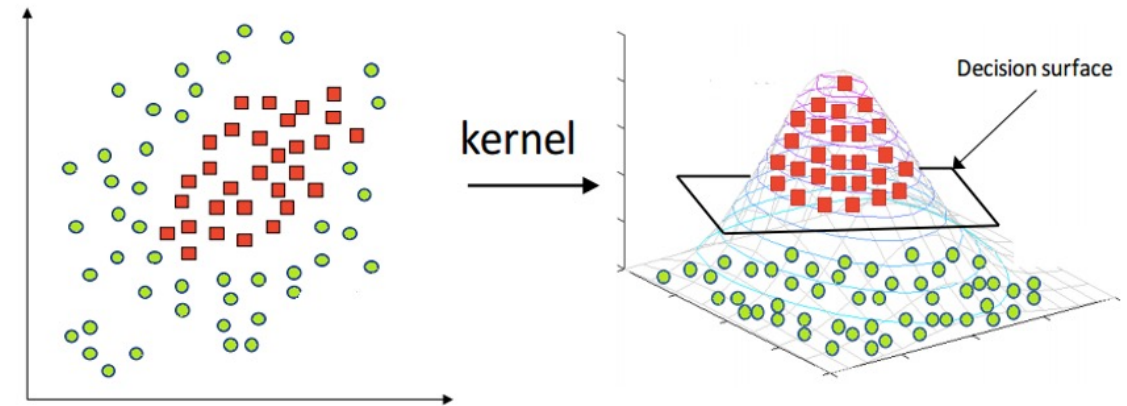
Recap (4)

Soft Margin SVM Objective

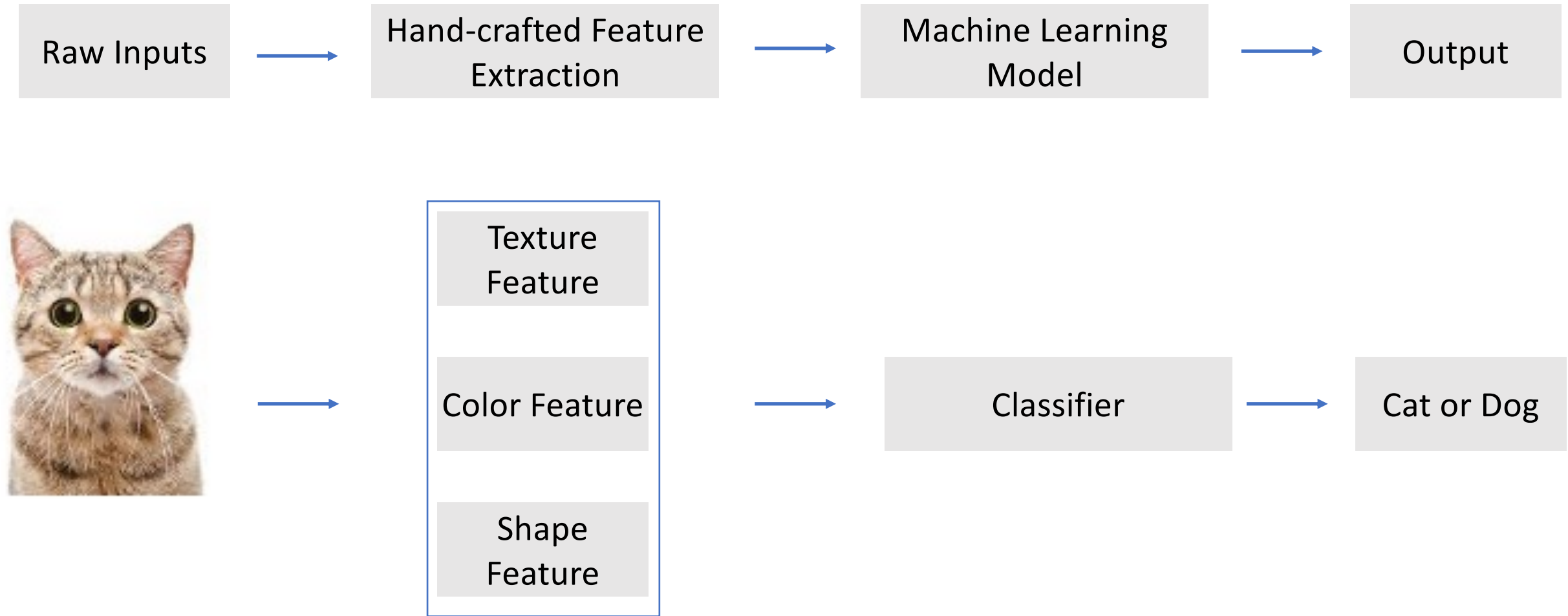
$$\operatorname{argmin}_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{i=1}^n \xi_i$$

subject to $\xi_i \geq 0, y_i(w_0 + w_1 x_1 + \dots + w_p x_p) \geq 1 - \xi_i$ for all i

Kernel Trick



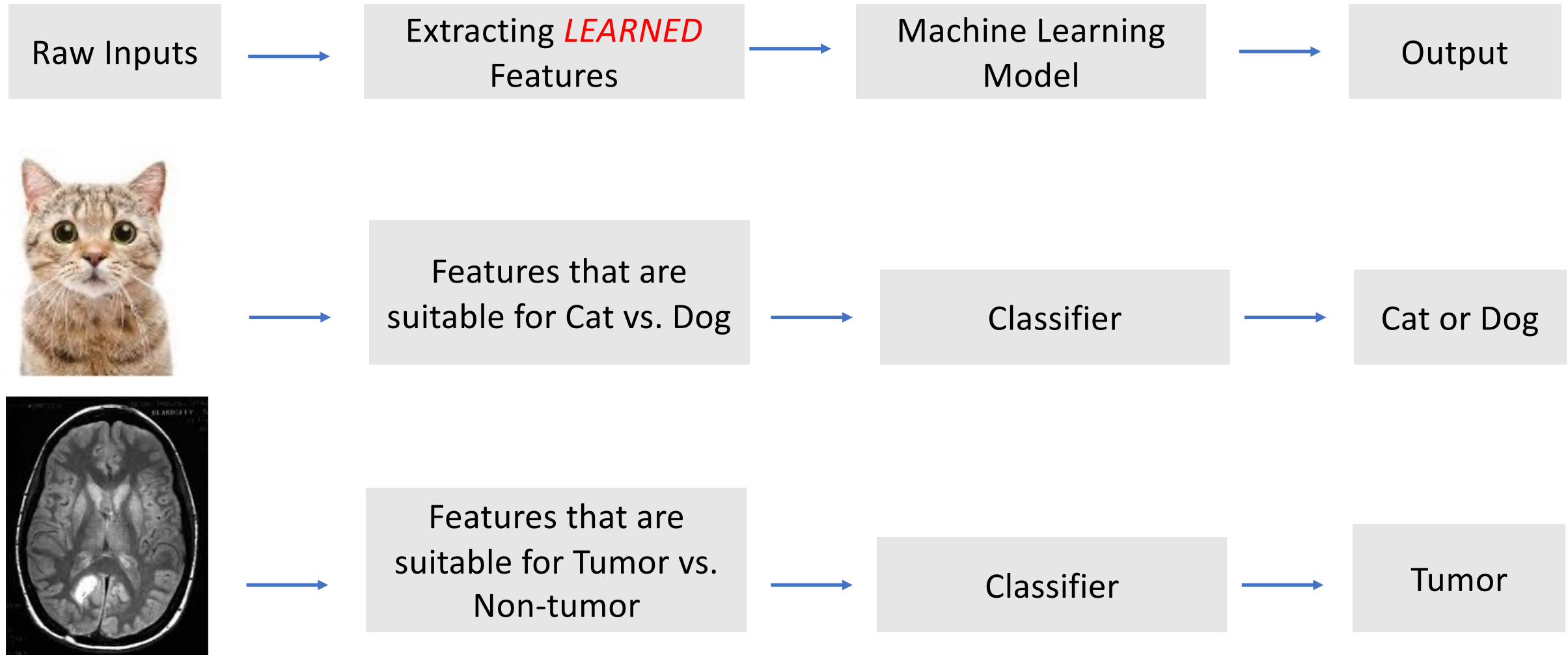
Traditional Machine Learning



Feature Representation Matters

- Which of these cases would be easier for you to solve?
 - Divide 210 by 6
 - Divide CCX by VI
- Same is true in machine learning
 - A feature representation is good if it makes the subsequent learning task easier
 - The choice of the representation thus depends on the learning task

Feature Learning

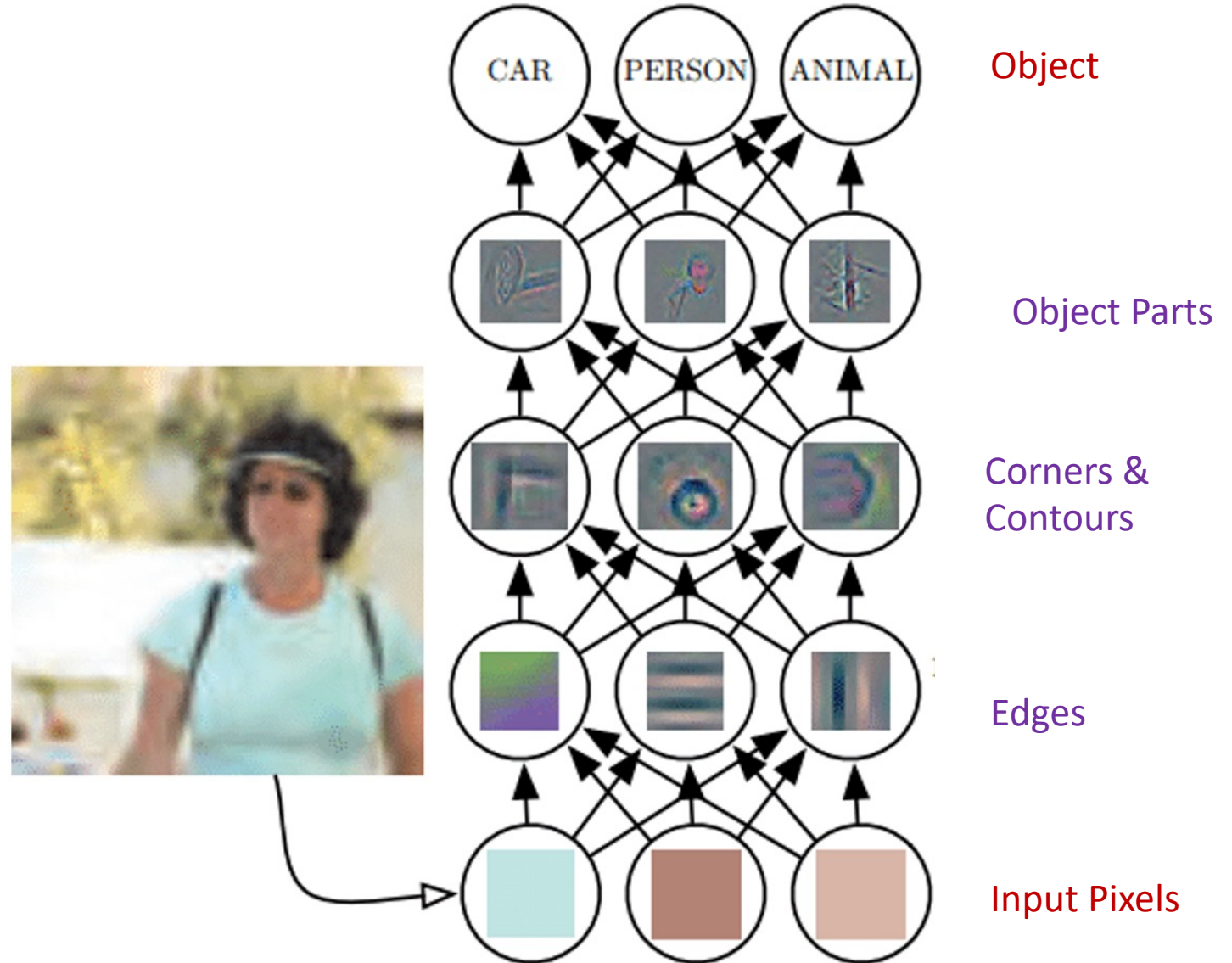


Thus

1. Feature learning is better than using hand-crafted features

Hierarchical Features

- Data consists of a large number of features that are interrelated and have varying degrees of importance.
- Hierarchical features can be used to group these features into higher-level categories that capture broader patterns in the data.



Thus

1. Feature learning is better than using hand-crafted features
2. Feature learning should aim for learning hierarchical features

Non-linear Features

- Effective hierarchical features are often nonlinear, as they capture complex patterns and relationships in the data that cannot be expressed by a simple linear combination of the input features.
- Therefore, **Non-linearity** is important when learning hierarchical features

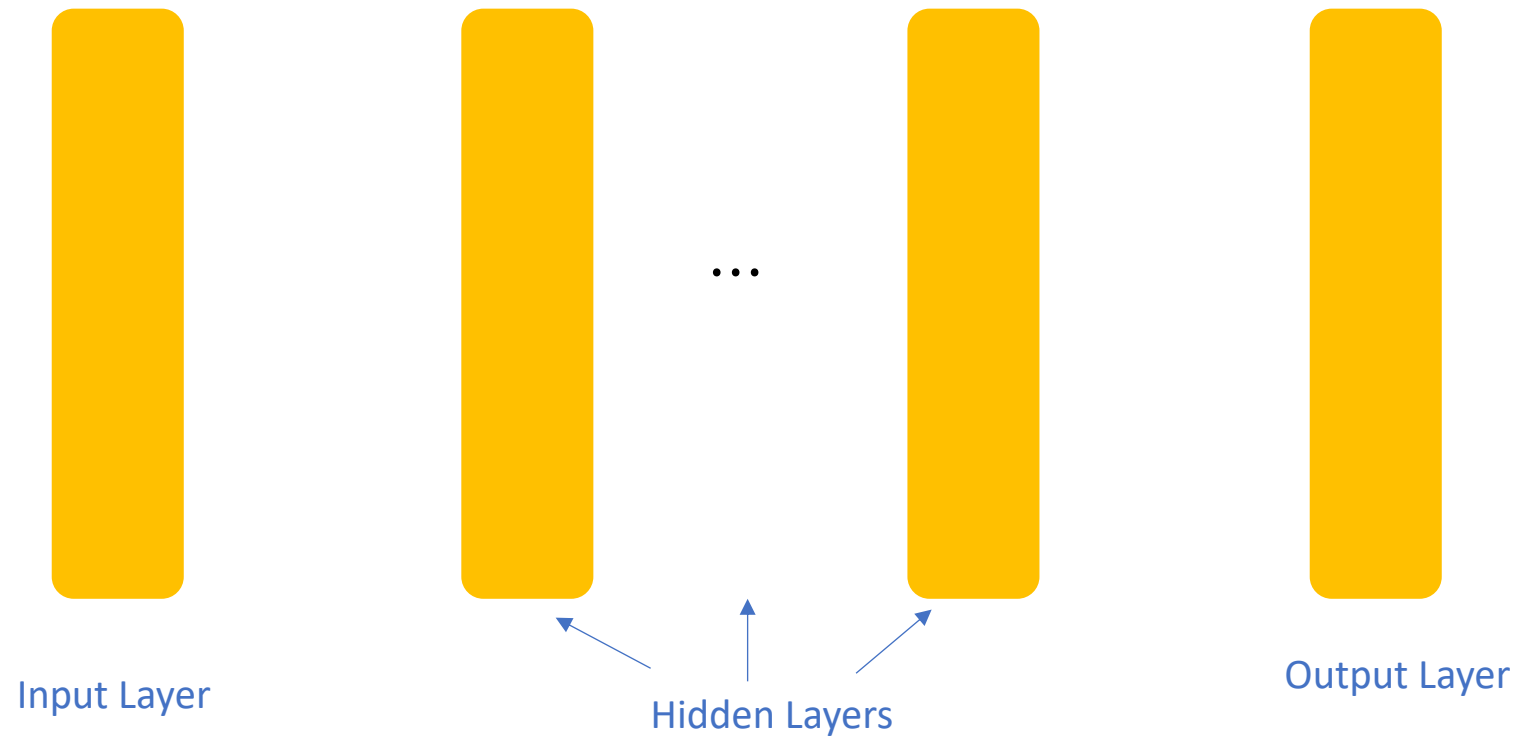
Thus

1. **Feature learning** is better than using hand-crafted features
2. Feature learning should aim for learning **hierarchical features**
3. Higher level features **should not be simple linear combinations** of low-level features

Deep Learning enables us to achieve these goals!

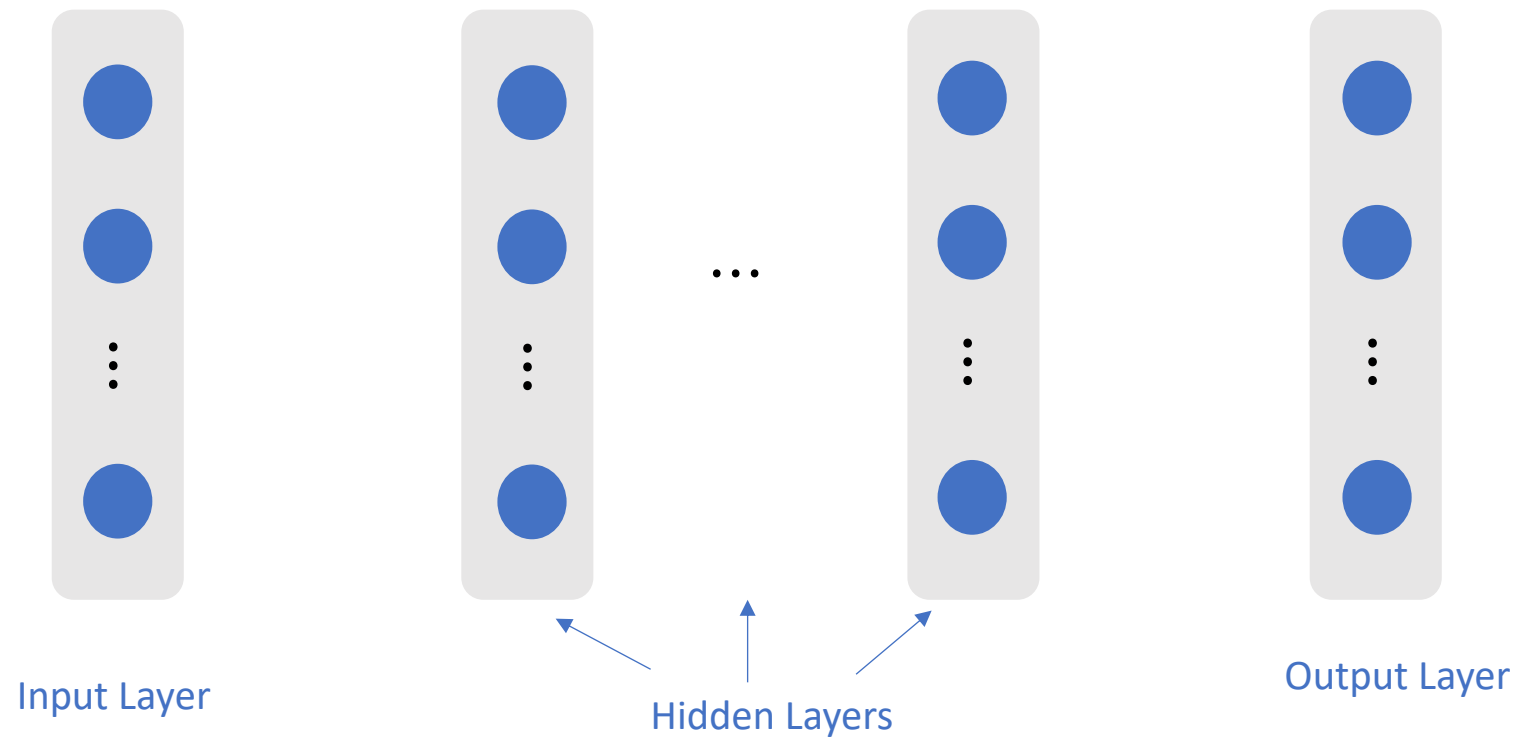
Deep Neural Networks

- Layered Architecture – **enabling Hierarchy**



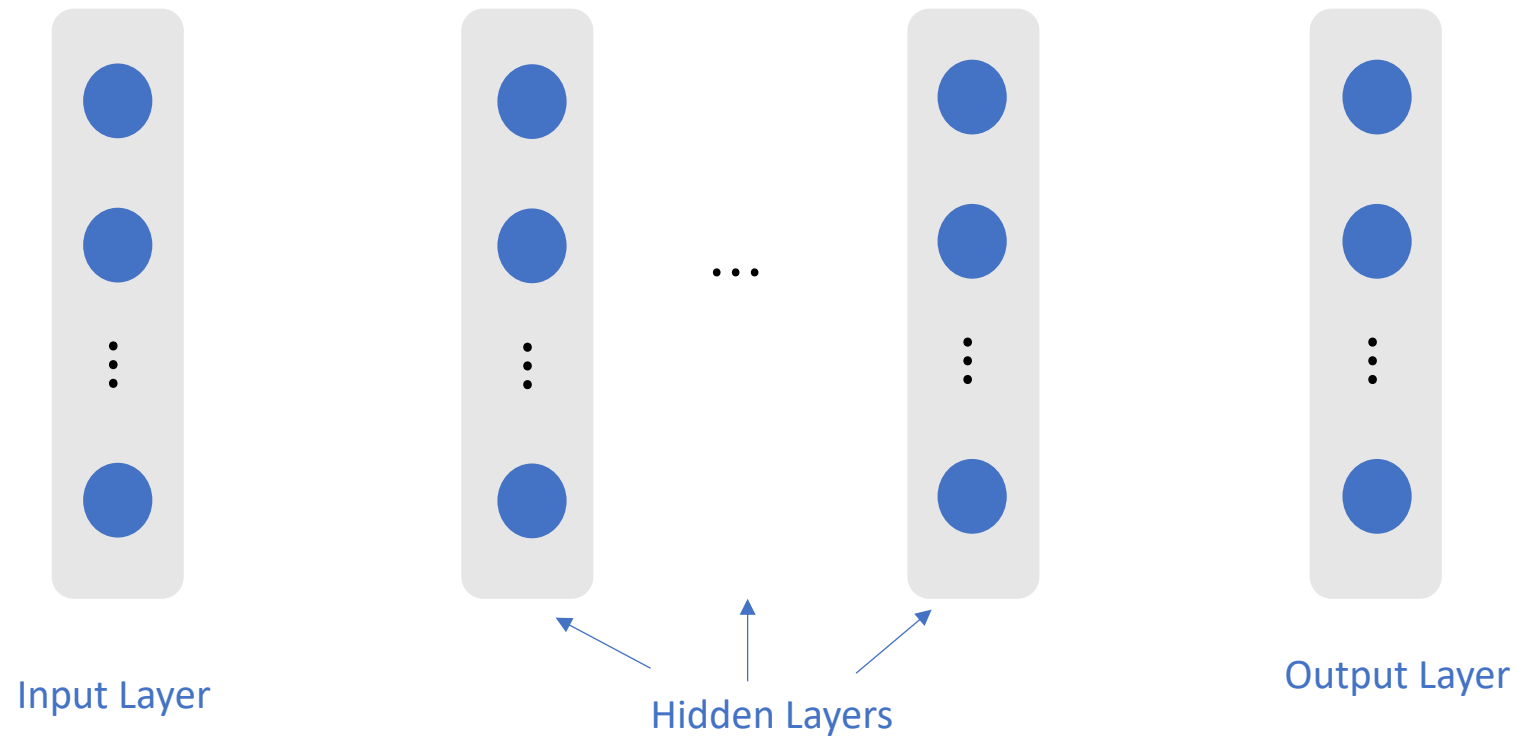
Deep Neural Networks (2)

- Each layers consists of nodes or neurons



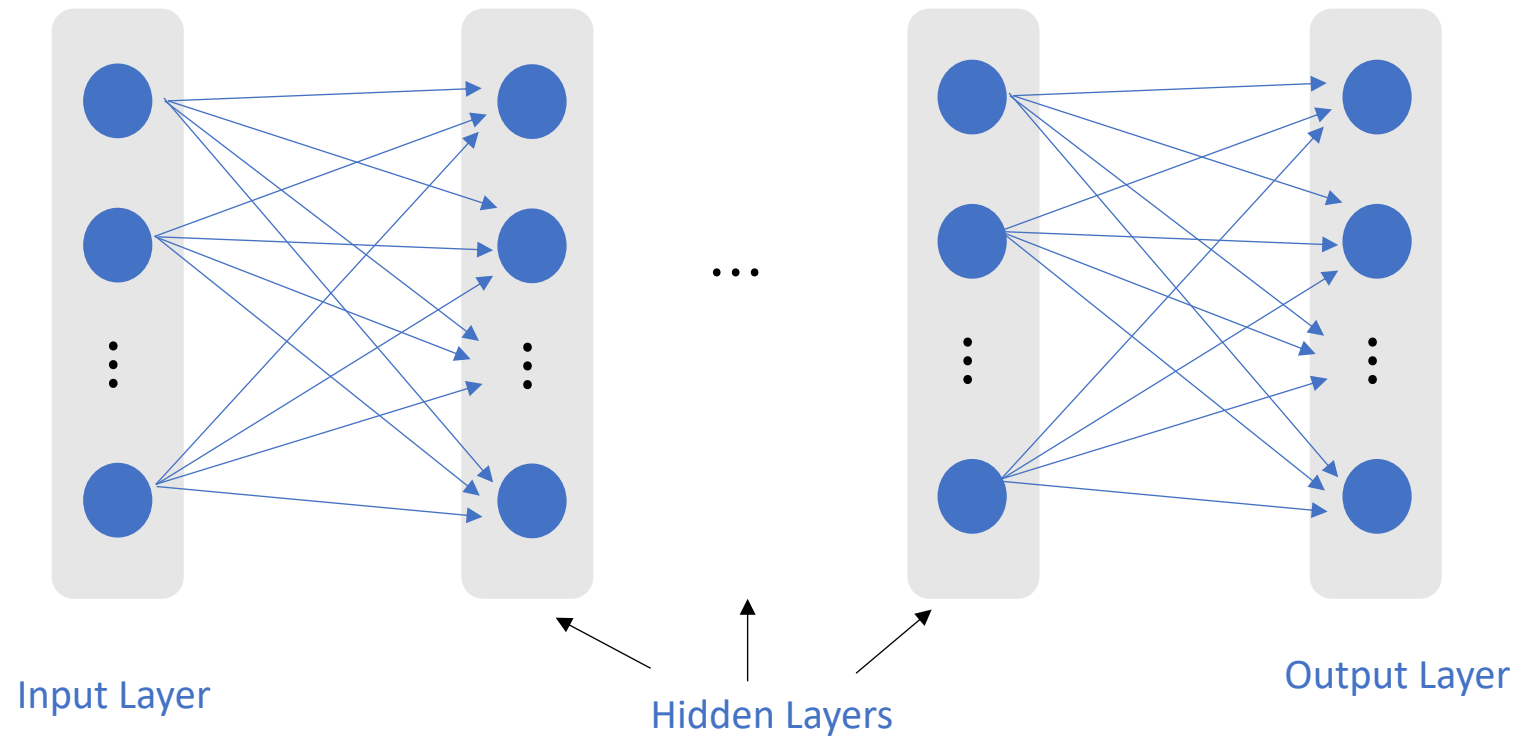
Deep Neural Networks (3)

- Number of Neurons:
 - **Input Layer** (same as the number of inputs), **Output Layer** (same as the number of outputs), **Hidden Layer** (fine-tuned)



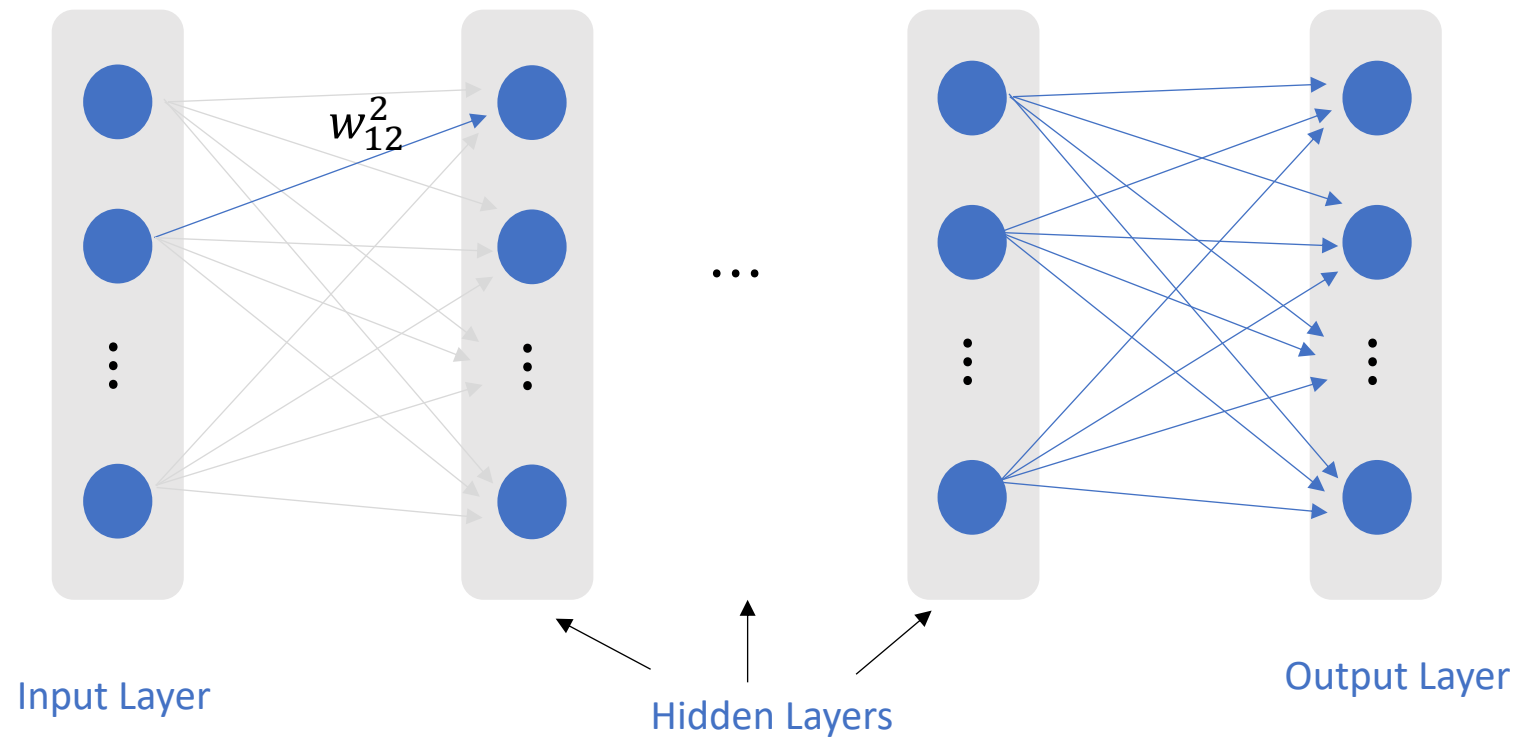
Deep Neural Networks (4)

- Neurons at layer l are connected with the neurons at layer $l - 1$



Deep Neural Networks (5)

- Each connection has an associated weight

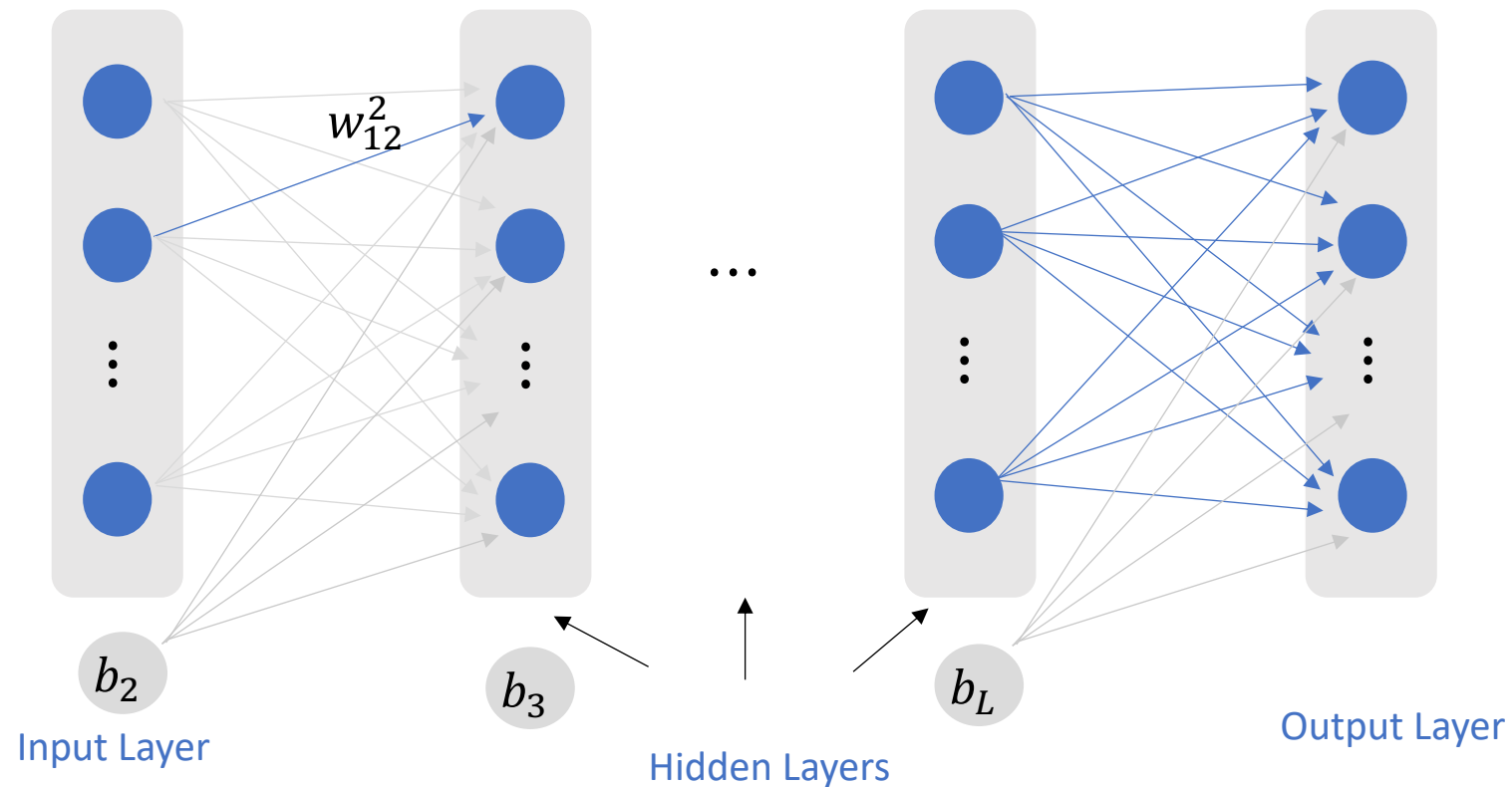


What Happens at a Neuron?

1. WEIGHTED Input is received from the neurons of the previous layer, **plus the bias term**
2. Input is processed
3. Depending on the outcome, the neuron ACTIVATES

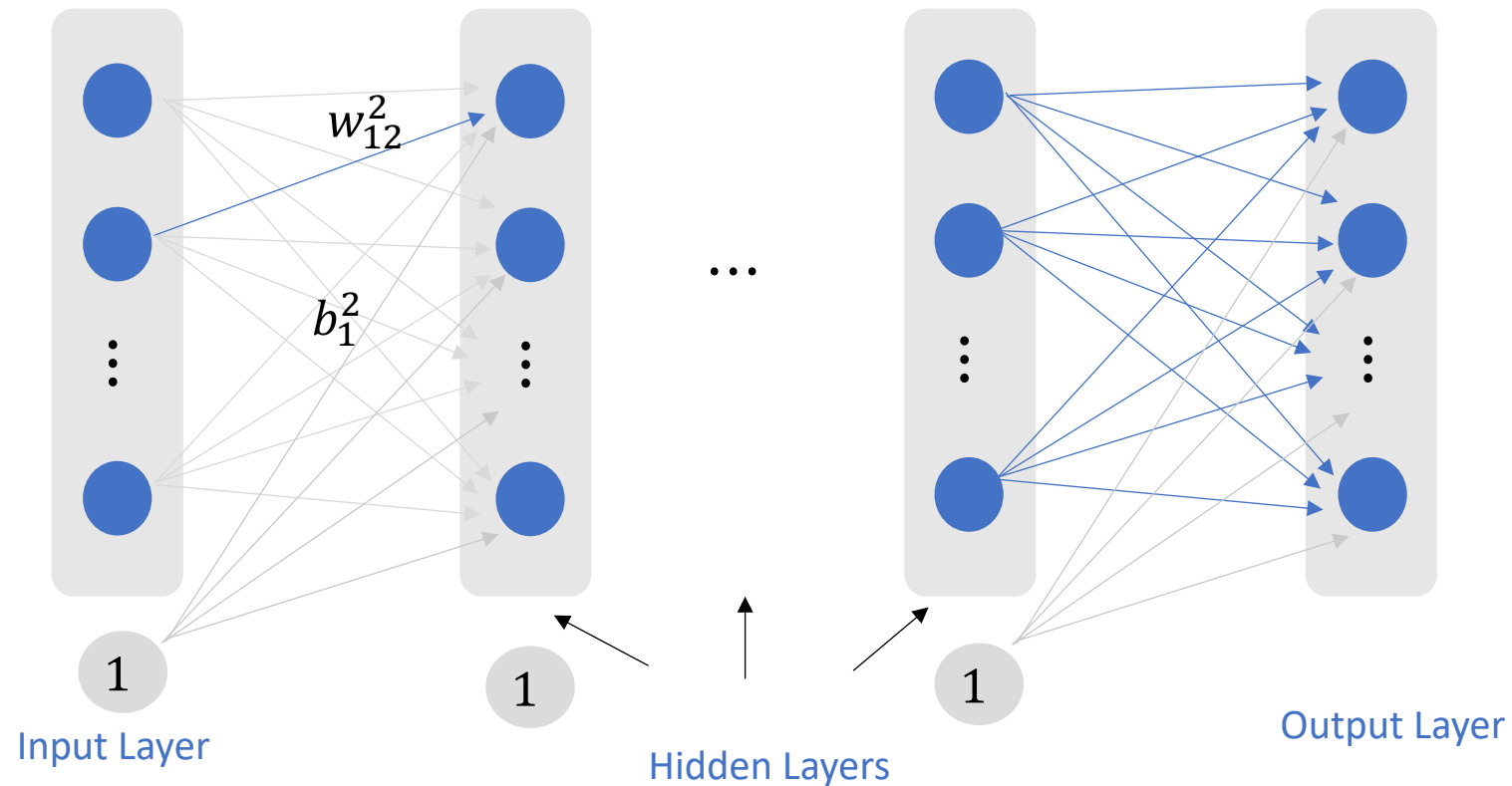
Deep Neural Networks (6)

- Each layer has an associated bias which is fed to the neurons of the next layer



Deep Neural Networks (7)

- Each layer has an associated bias which is fed to the neurons of the next layer



What Happens at a Neuron? (2)

1. WEIGHTED Input is received from the neurons of the previous layer, plus the bias term

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

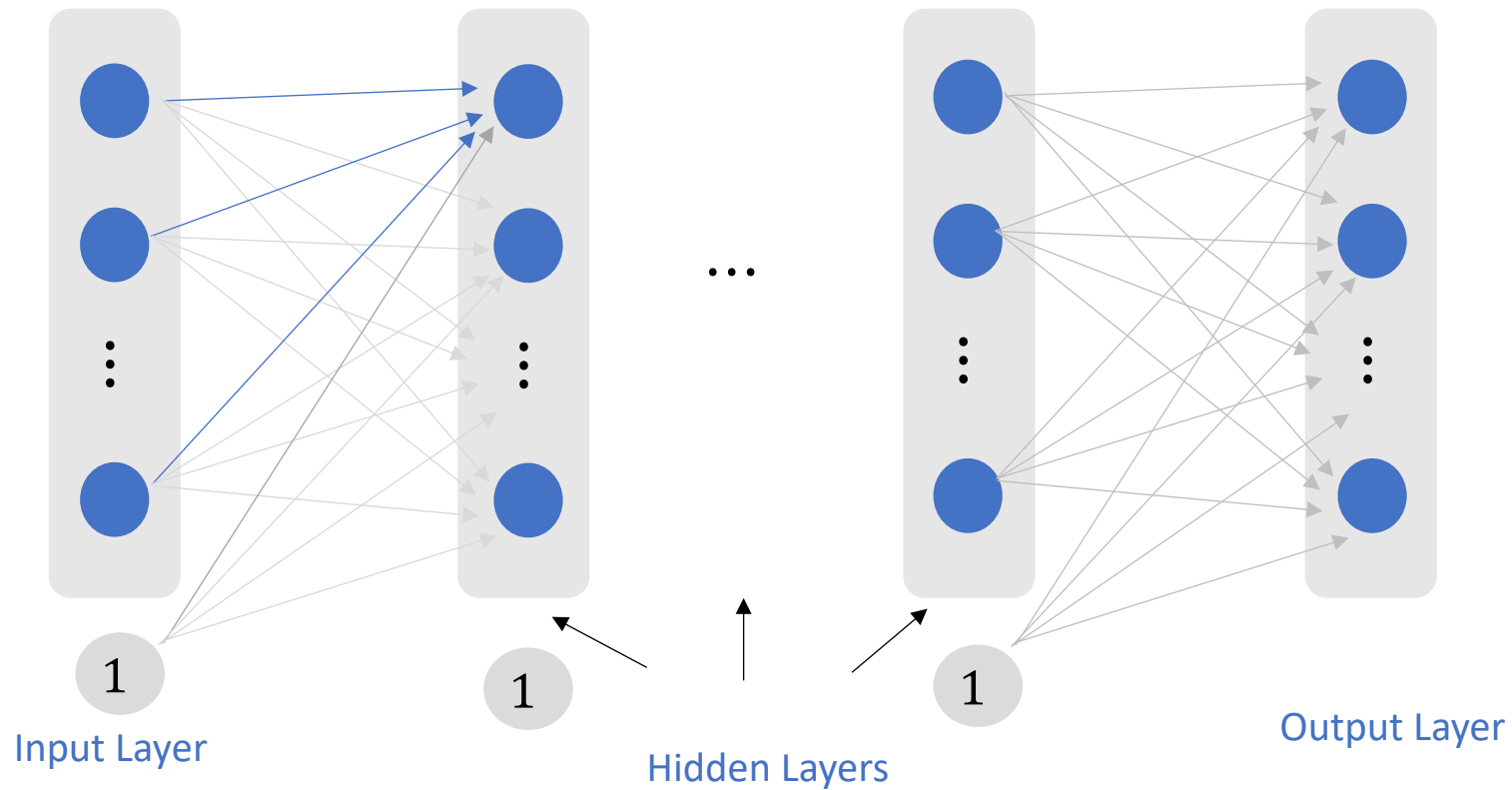
2. Input is processed

- Weighted sum is computed
- Non-linear activation function is applied - Enables non-linearity

3. Depending on the outcome, the neuron ACTIVATES

Deep Neural Networks (8)

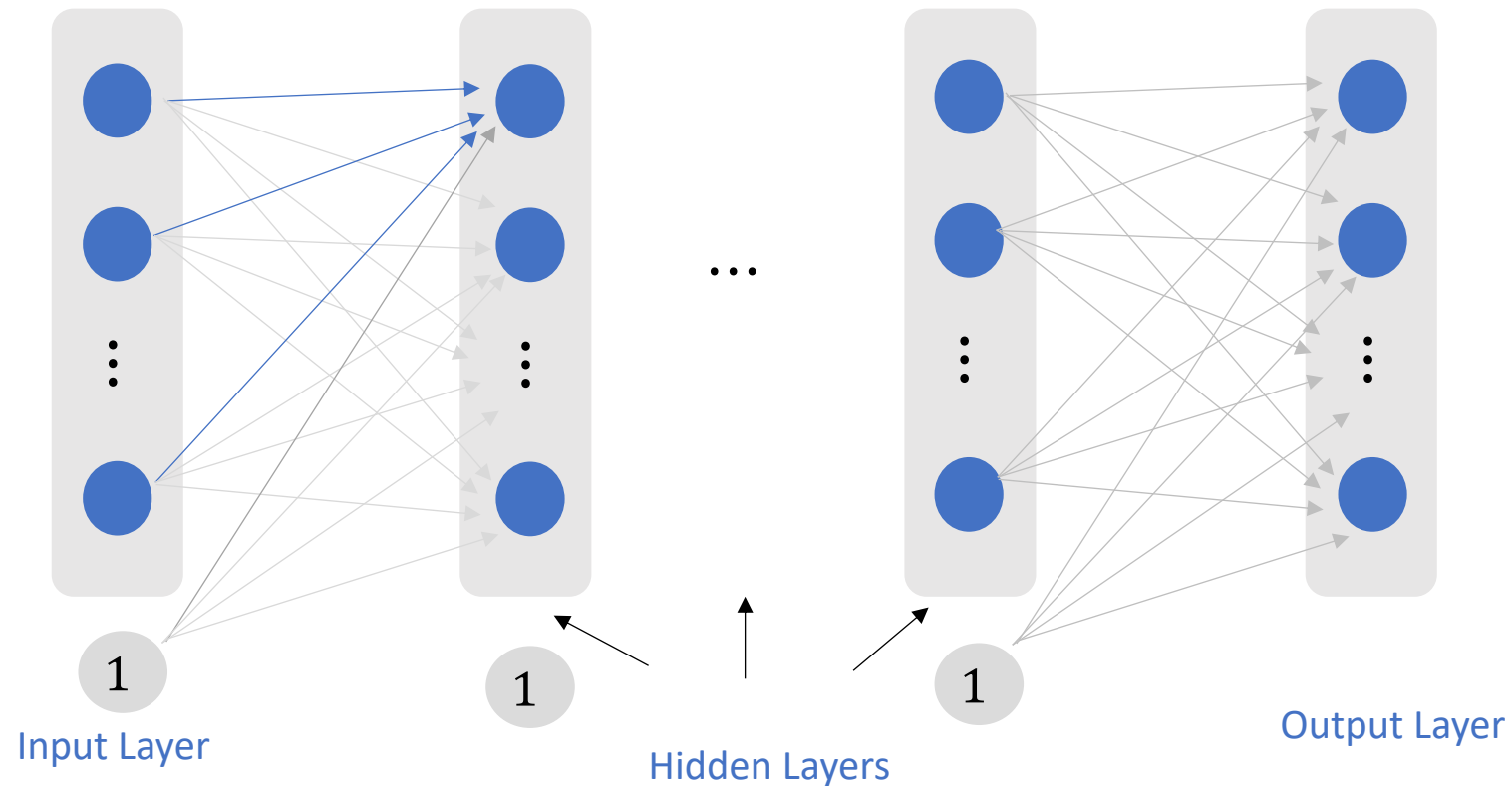
$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$



Summary – Deep Neural Networks

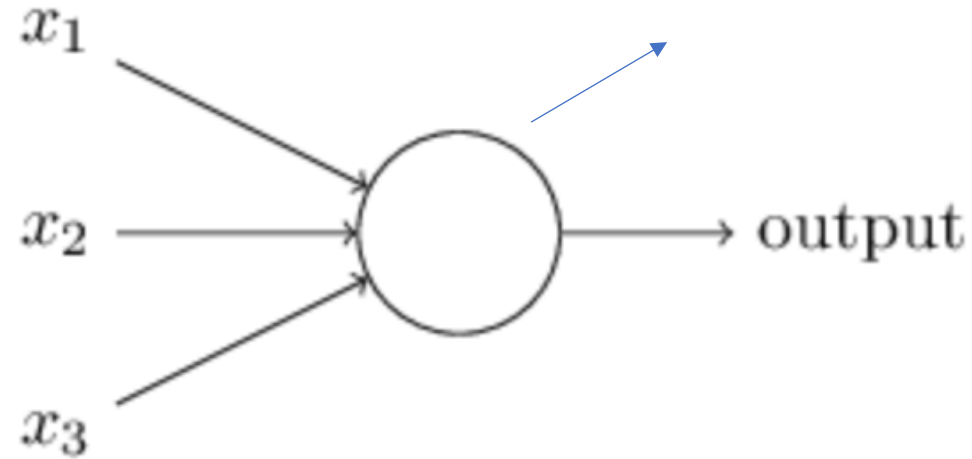
- Also called Feedforward Networks

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

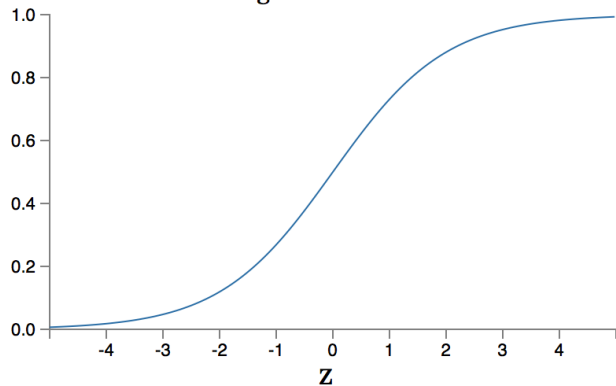


Sigmoid Activation Function

$$\sigma(w \cdot x + b) \longrightarrow \frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$



sigmoid function



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

Smoother response, limited between 0 and 1

Other Options for Activation Functions

1. Hyperbolic Tangent
2. Rectified Linear Unit
3. Leaky ReLU
4. Exponential ReLU

Read about them yourself. Materials easily available online.

Neural Learning

Neural Learning

- Uses Gradient Descent to find the optimum weights and the biases for the DNN that minimize the error or the cost

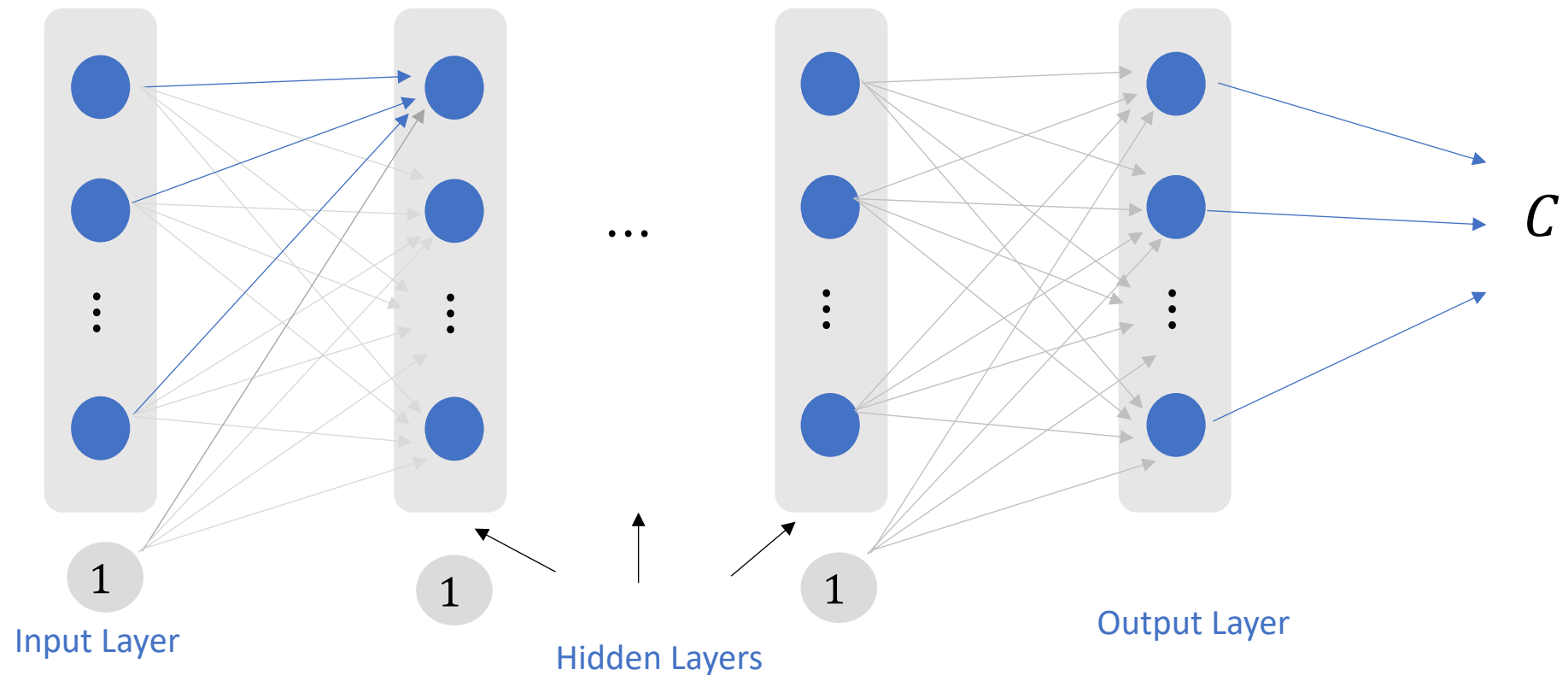
$$w = w - \alpha \frac{\partial \mathcal{C}}{\partial w}$$

But how to do it for each weight in our network?

Important Point!

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

- Cost, Loss or Error is computed at the output layer



Recall

1. $f(x) \rightarrow \frac{df}{dx}$ computed directly
2. $f(g(x)) \rightarrow \frac{df}{dx}$ computed by the chain rule
3. $f(x + y) \rightarrow \frac{df}{dx}$ does not depend on y

Analogy

- $f(x) \rightarrow \frac{df}{dx}$ computed directly

“Imagine you are following a recipe to bake a cake. The function $f(x)$ represents the relationship between the ingredient quantity (x) and the taste of the cake. If you directly adjust the amount of an ingredient (like sugar), you can directly observe and compute the change in the taste of the cake (df/dx). There's no need for intermediary steps; you have a straightforward, direct relationship to observe and measure.”

Analogy

- $f(g(x)) \rightarrow \frac{df}{dx}$ computed by the chain rule $(\frac{df}{dg} \times \frac{dg}{dx})$

“Imagine your room has a thermostat which controls a heater, and you want to adjust the room temperature (f) based on the outside temperature (x). However, the thermostat does not directly sense the outside temperature. Instead, it senses the inside temperature ($g(x)$), which is affected by the outside temperature.

Here, $f(g(x))$ represents the relationship between the outside temperature and the room temperature through the inside temperature. If you want to know how a change in the outside temperature would affect the room temperature, you need to use the chain rule to compute df/dx , taking into account both how the outside temperature affects the inside temperature (dg/dx) and how the inside temperature affects the room temperature (df/dg).”

Analogy

- $f(x + y) \rightarrow \frac{df}{dx}$ does not depend on y

“Imagine you have a plant, and its growth (f) depends on the amount of water (x) and sunlight (y) it receives. If you are **only** looking at how a change in the amount of water affects the plant’s growth, this relationship (df/dx) does not depend on the amount of sunlight the plant is receiving.

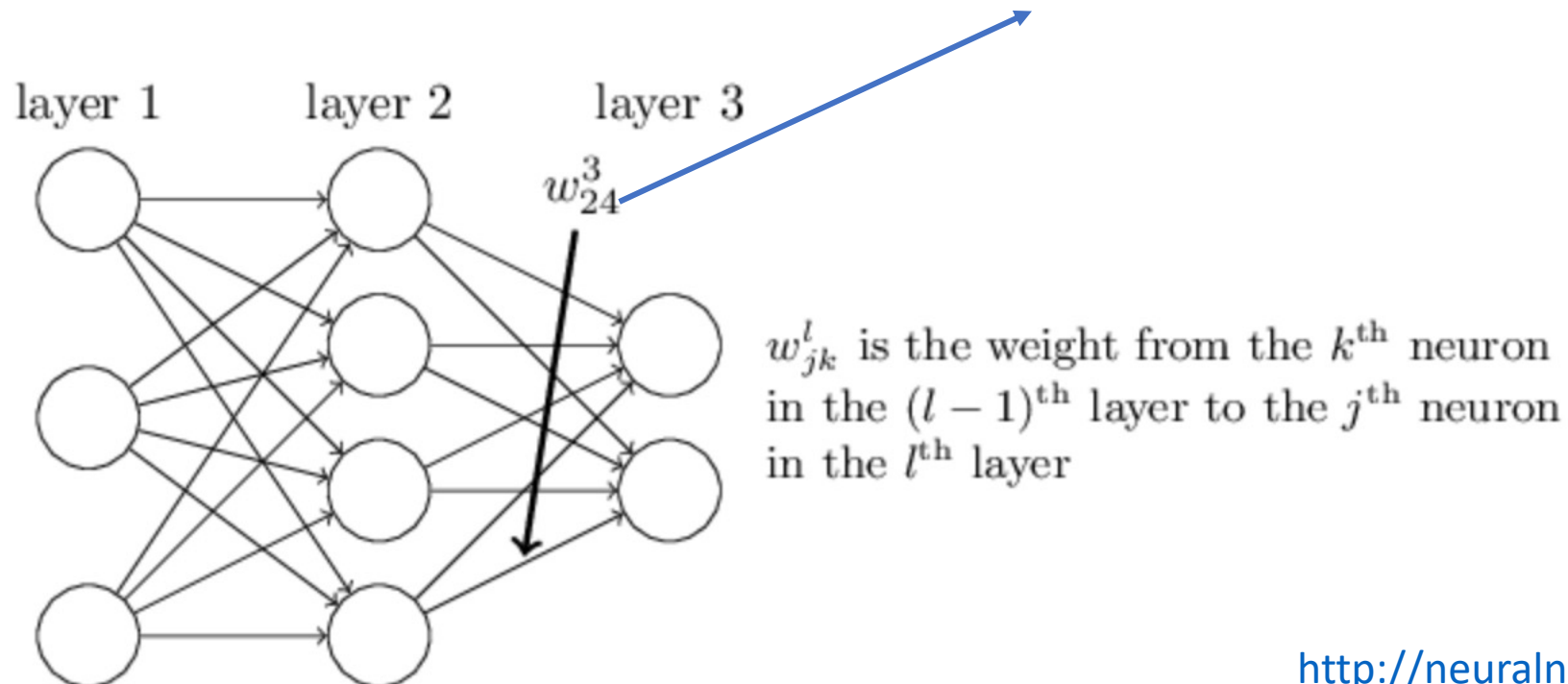
This helps to understand that when differentiating a function with respect to one variable, other variables in the equation can be treated as constants.”

Backpropagation

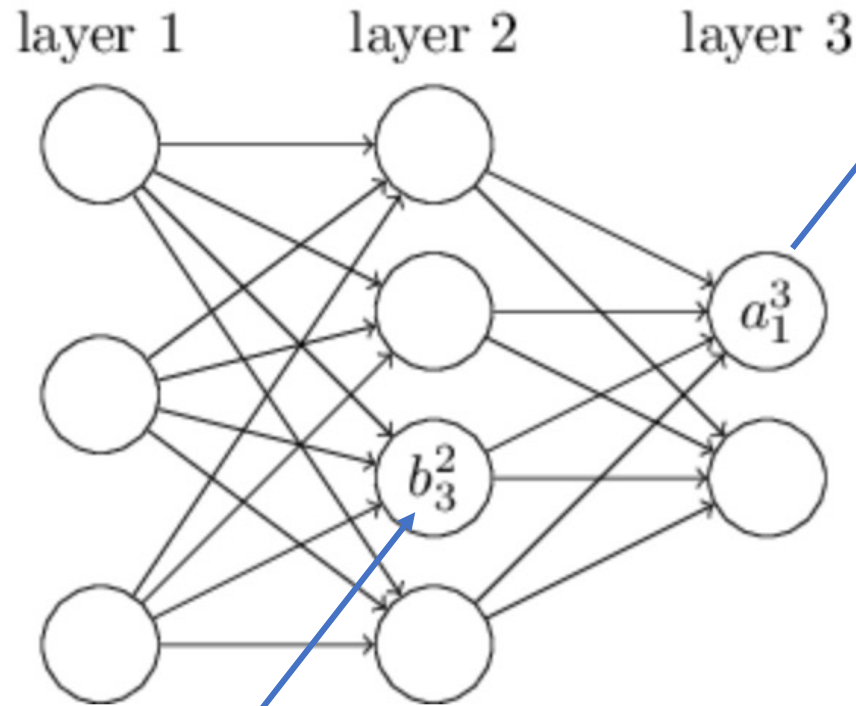
- The workhorse of learning in neural networks.
- At the heart of backpropagation is an expression for the partial derivative of the cost function C with respect to every weight w (or bias b) in the network.

Notations-1

the weight on a connection from the **fourth neuron** in the **second layer** to the **second neuron** in the **third layer** of a network



Notations-2



the activation of the 1st neuron at the 3rd layer

b_j^l for the bias of the j^{th} neuron in the l^{th} layer

a_j^l for the activation of the j^{th} neuron in the l^{th} layer

Bias of the 3rd neuron on the 2nd layer

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

<http://neuralnetworksanddeeplearning.com/>

Vectorized Notations

$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right)$$

$$a^l = \sigma(w^l a^{l-1} + b^l)$$

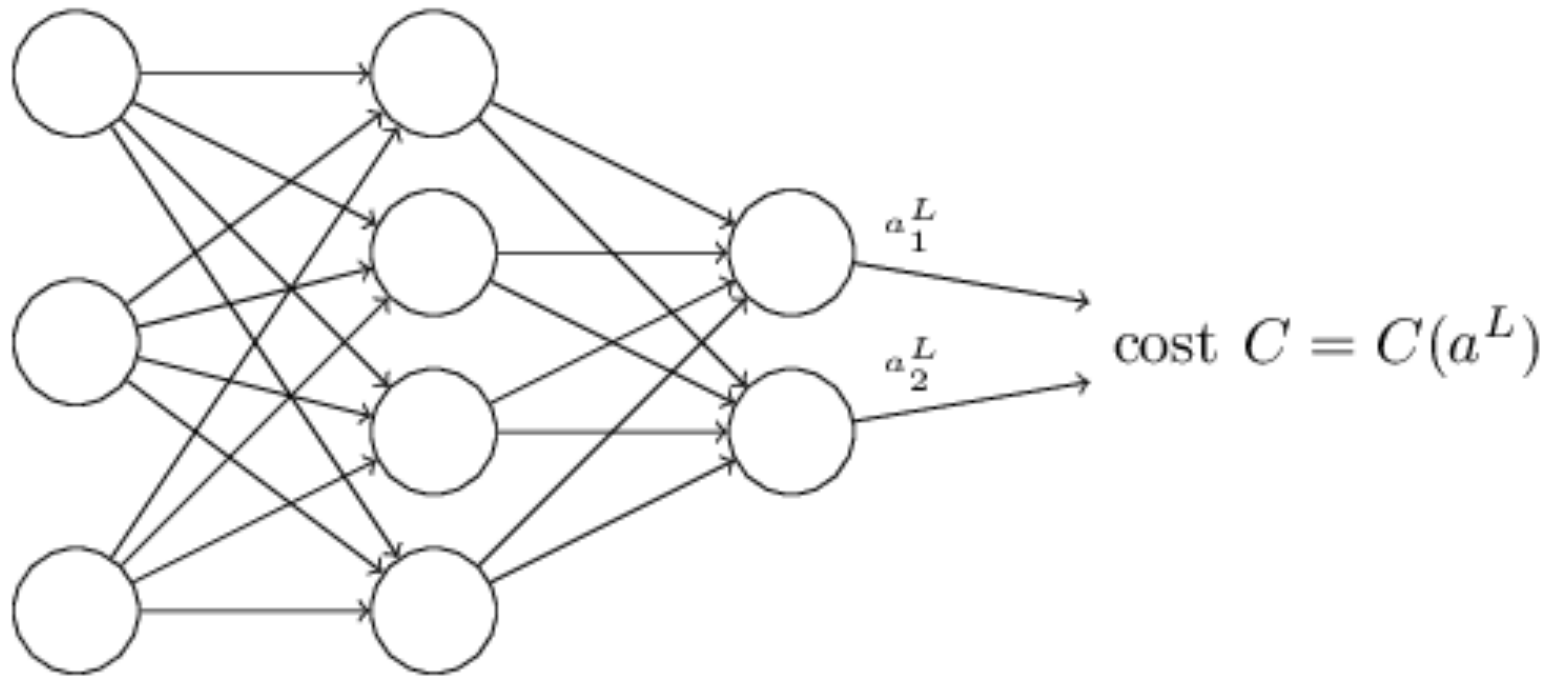
$$z^l = w^l a^{l-1} + b^l$$

Weighted sum in vector form

$$a^l = \sigma(z^l)$$

Important Things to Remember

Cost is a function of activations at the output layer!!!



$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

Important Things to Remember (2)

Cost is a function of activations at the output layer!!!

Activation at any layer is a function of weighted sum

$$a^l = \sigma(z^l)$$

Important Things to Remember

Cost is a function of activations at the output layer!!!

Activation at any layer is a function of weighted sum

Weighted sum is a function of weights and biases

Also Note: Weighted sum is a function of the weighted sum of the previous layer

$$z^l = w^l a^{l-1} + b^l$$

$$z^l = w^l \sigma(z^{l-1}) + b^l$$

Thus

- In order for us to compute

$$\frac{\partial C}{\partial w_{jk}^l} \quad \frac{\partial C}{\partial b_j^l}$$

- We will first need to compute

$$\frac{\partial C}{\partial z_j^l}$$

Backprop in Action


Backpropagation – A1

1. Let's start with the output layer (L)
2. Let's define an intermediate quantity for j – th neuron in layer L as follows

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$

Backpropagation – A2

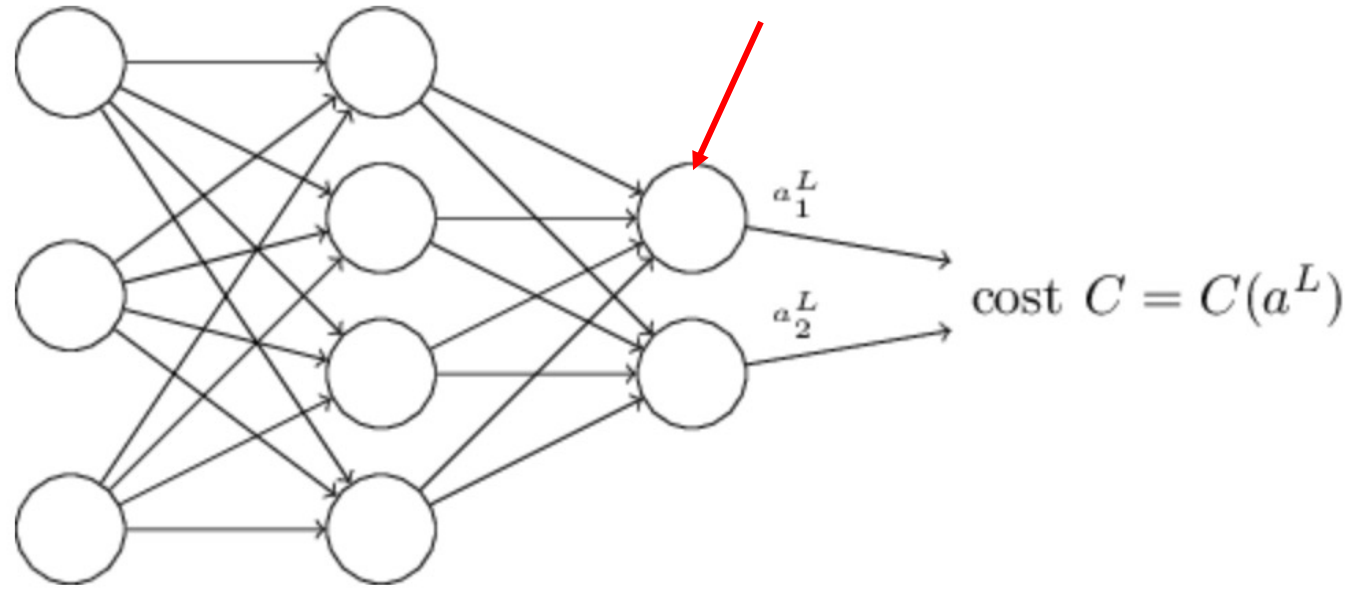
Applying the chain rule

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}$$


where the sum is over all neurons in the output layer.

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$

Does the activation of this neuron depend on the weighted sum of other neurons at this layer?



Backpropogation - A3

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}$$

- Thus the above equation can be written as

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L},$$

Backpropagation – A4

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L},$$

Recalling that $a_j^L = \sigma(z_j^L)$ the second term on the right can be written as $\sigma'(z_j^L)$, and the equation becomes

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L),$$

So we have computed this value for the Last layer, now let's compute it for previous layers.

Note!

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L),$$

- The exact form of $\frac{\partial C}{\partial a_j^L}$ will depend on the cost function

- For example, for the quadratic cost function $C = \frac{1}{2} \sum_j (y_j - a_j^L)^2$

$$\partial C / \partial a_j^L = (a_j^L - y_j)$$

Backpropagation – B1

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}$$

- Computation at j – th neuron in arbitrary layer l , but we don't have C directly available at l

What we will do is to figure out a way to compute this using the computation that we did at the layer $l + 1$

that is, we will **BACKPROPOGATE** the results of layer $l + 1$ to compute the result for layer l

Recall

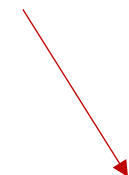
$$z^l = w^l a^{l-1} + b^l$$

$$z^l = w^l \sigma(z^{l-1}) + b^l$$

Backpropagation – B2

- Thus at arbitrary layer l

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \longrightarrow = \sum_k \boxed{\frac{\partial C}{\partial z_k^{l+1}}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$



$$= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \boxed{\delta_k^{l+1}},$$

Note That

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

- Therefore

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l).$$

Backpropagation – B3

- Thus at arbitrary layer l

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l).$$

Recap

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L).$$

- And

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l).$$

Backpropagation – C1

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial C}{\partial z_k^l} \cdot \frac{\partial z_k^l}{\partial b_j^l} \longrightarrow = 1$$

$$z_j^l = w_j^l a^{l-1} + b_j^l \quad \text{Thus}$$

Also, note that inside the sum, both terms will be 0 except for the case where $k = j$,
Therefore

Backpropagation – C2

$$\begin{aligned}\frac{\partial \mathcal{C}}{\partial b_j^l} &= \sum_k \frac{\partial \mathcal{C}}{\partial z_k^l} \cdot \frac{\partial z_k^l}{\partial b_j^l} \\ &= \frac{\partial \mathcal{C}}{\partial z_j^l} \\ &= \delta_j^l\end{aligned}$$

Backpropagation – C3

Thus

$$\frac{\partial \mathcal{C}}{\partial b_j^l} = \delta_j^l$$

Backpropagation – C3

- Finally

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

Prove this yourself!

Recap

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L),$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l).$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l.$$

Backpropagation Algorithm

1. **Input a set of training examples**

2. **For each training example**

x : Set the corresponding input activation $a^{x,1}$, and perform the following steps:

- **Feedforward:** For each $l = 2, 3, \dots, L$ compute

$$z^{x,l} = w^l a^{x,l-1} + b^l \text{ and } a^{x,l} = \sigma(z^{x,l}).$$

- **Output error**

$$\delta^{x,L} : \text{Compute the vector } \delta^{x,L} = \nabla_a C_x \odot \sigma'(z^{x,L}).$$

- **Backpropagate the error:** For each

$l = L - 1, L - 2, \dots, 2$ compute

$$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l}).$$

3. **Gradient descent:** For each $l = L, L - 1, \dots, 2$ update the weights according to the rule $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$, and the biases according to the rule $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$.

Summary

1. Motivation for DNNs

- Feature learning, hierarchical features, non-linearity

2. Feed-forward Neural Networks

3. Neural Learning

- Backpropagation
- Deriving the four main equations of backpropagation

Reflection

1. What are the features in machine learning and why are they important?
2. What is feature or representation learning, and why is it important?
3. How can artificial neural networks (ANNs) help us in feature learning?
4. What enables ANNs to learn hierarchical features?
5. What enables ANNs to learn nonlinear features?
6. What is backpropagation?