



University of
South Australia

Neural networks Week 9

Dr Srećko Joksimović

Intro to neural networks



University of
South Australia

Why do we study neural computation?

To understand how the brain actually works.

- Its very big and very complicated and made of stuff that dies when you poke it around.
- So we need to use computer simulations.

To understand a style of parallel computation inspired by neurons and their adaptive connections.

- Very different style from sequential computation.
 - Should be good for things that brains are good at (e.g. vision)

To solve practical problems by using novel learning algorithms inspired by the brain

- Learning algorithms can be very useful even if they are not how the brain actually works



Understanding the brain: a brief history

- 335 BC (Aristotle) – the brain as a radiator
- 16th century, (Andreas Vesalius) – the first sketch of the nervous system
- 1791 (Luigi Galvani) – electricity fires the nerves
- 1848 Phineas Gage accident
 - 1860-70s by physicians Paul Broca and Carl Wernicke showed that specific parts of the brain were dedicated to different components of speech
- 1900s (Santiago R.y Cajal and Camillo Golgi) neurons are the building blocks of the brain
- 1932 (Sir Charles Sherrington and Edgar Adrian) – concept of synapses
- 1960s the explosion of neuroscience

Old vs New Conceptualisation

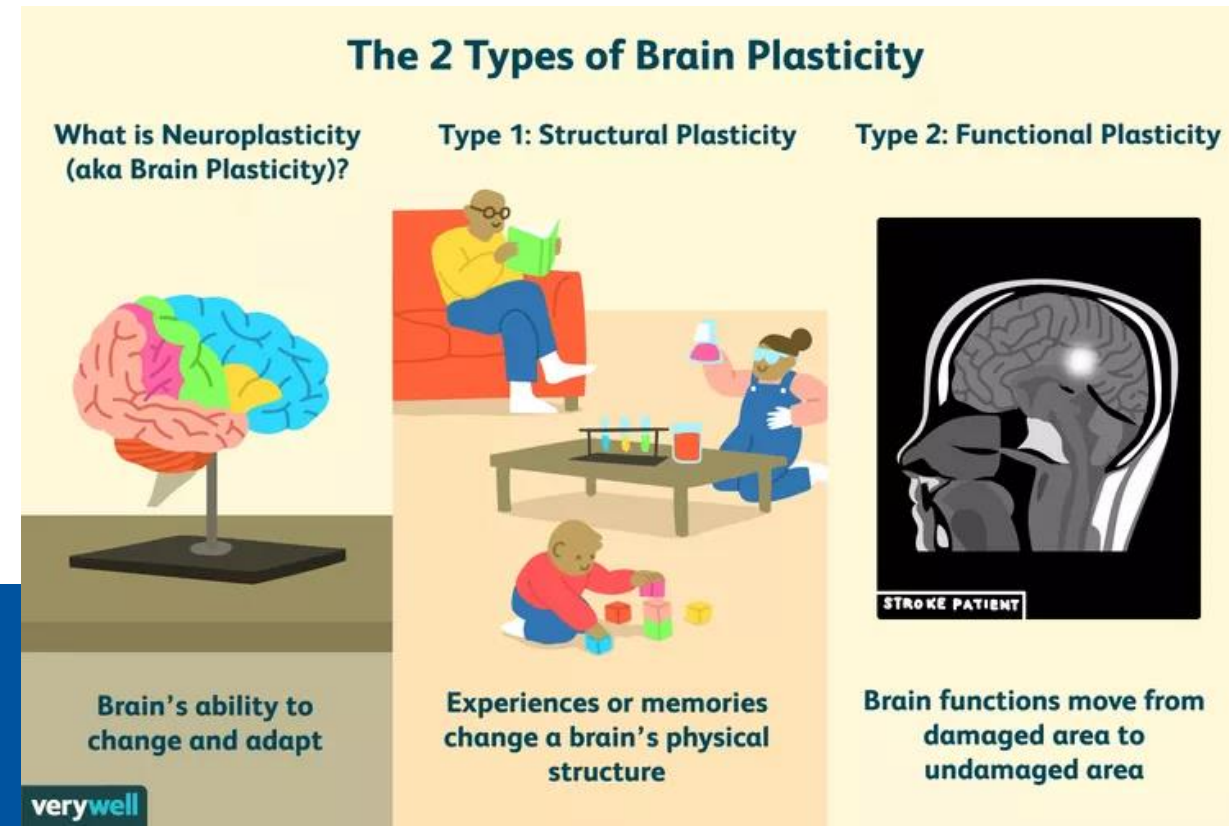
- The static machine (mid-20th century)
 - Fixed and predetermined pathways for neural processing
- Does not account for the complexity and flexibility of neural processing



Neural Plasticity

- Brain adapts in response to experience
 - Growth of neurons
 - The strengthening or weakening of connections between neurons
 - Development of new neural pathways
- Two main types of neuroplasticity
 - Functional plasticity
 - Structural plasticity

Source: <https://www.verywellmind.com/what-is-brain-plasticity-2794886>

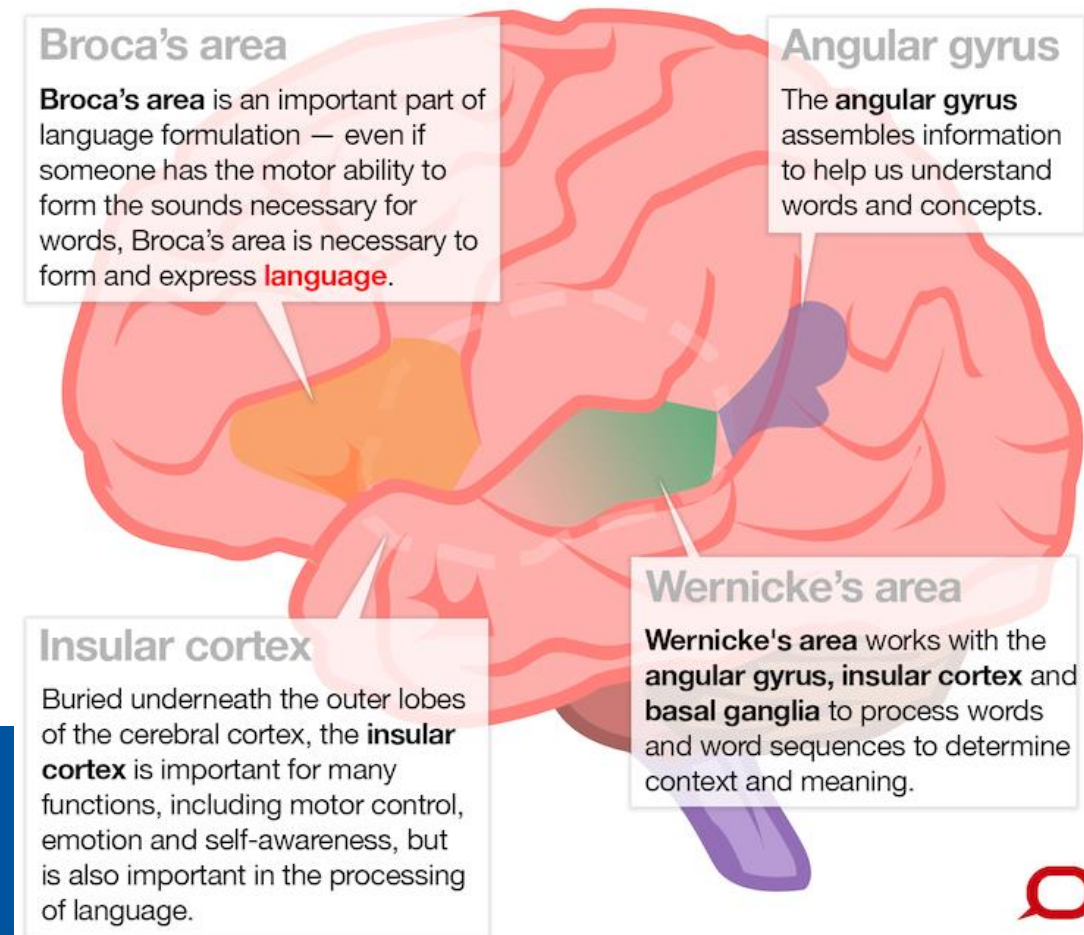


University of
South Australia

Distributed Processing

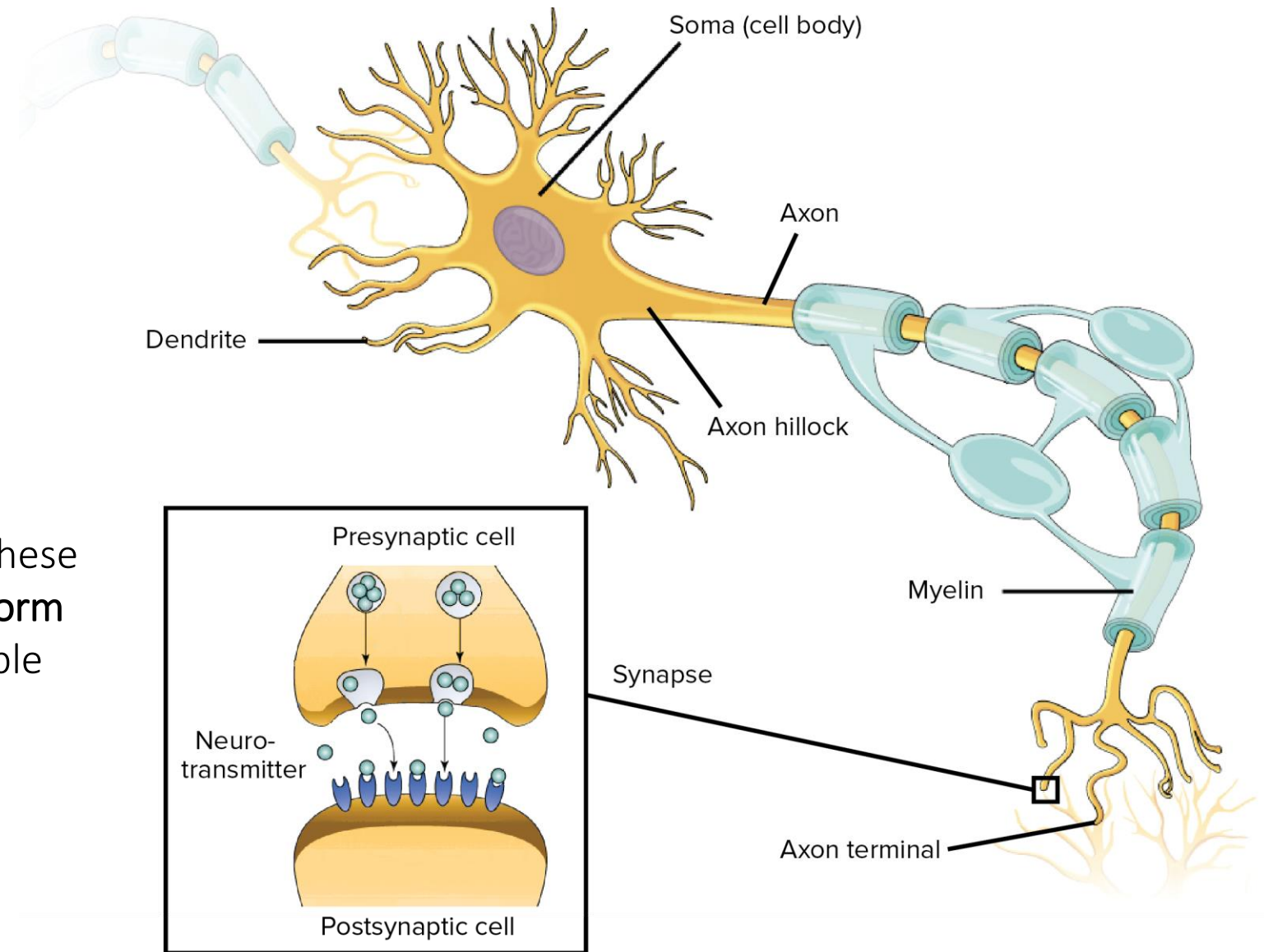
- Different regions work together to perform complex cognitive functions
 - Language processing involves multiple brain regions
- Allows for greater flexibility and adaptability
- Ability to perform complex tasks

Source: <https://theconversation.com/what-brain-regions-control-our-language-and-how-do-we-know-this-63318>



Synaptic Transmission and Neurotransmitters

Synaptic weights adapt and by adapting these weights the **whole network learns to perform different kinds of computation** - for example recognizing objects, or understanding language.



Idealized neurons

To model things we have to idealize them (e.g. atoms)

- Idealization removes complicated details that are not essential for understanding the main principles.
- It allows us to apply mathematics and to make analogies to other, familiar systems.
- Once we understand the basic principles, its easy to add complexity to make the model more faithful.

It is often worth understanding models that are known to be wrong (but we must not forget that they are wrong!)

- e.g. neurons that communicate real values rather than discrete spikes of activity.



Commonly used types of neurons



Linear neurons

These are simple but computationally limited

- If we can make them learn we **may** get insight into more complicated neurons.

$$y = b + \sum_i x_i w_i$$

Diagram illustrating the linear neuron equation $y = b + \sum_i x_i w_i$ with annotations:

- y : output
- b : bias
- i : index over input connections
- x_i : i^{th} input
- w_i : weight on i^{th} input

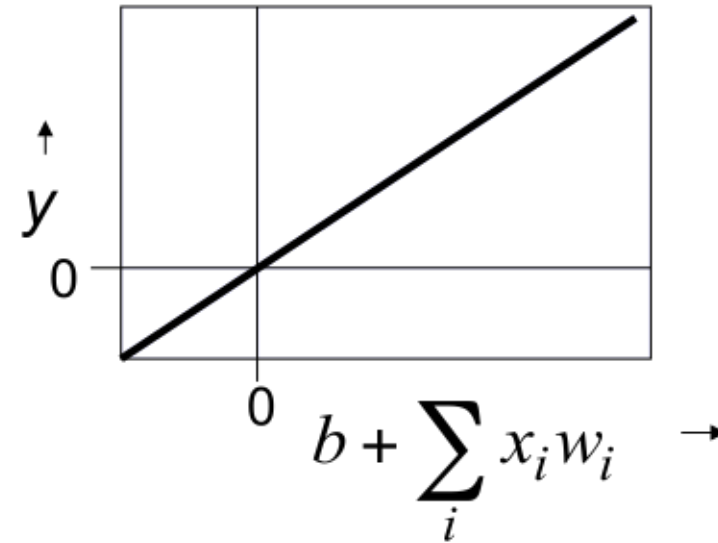


Linear neurons

These are simple but computationally limited

- If we can make them learn we **may** get insight into more complicated neurons.

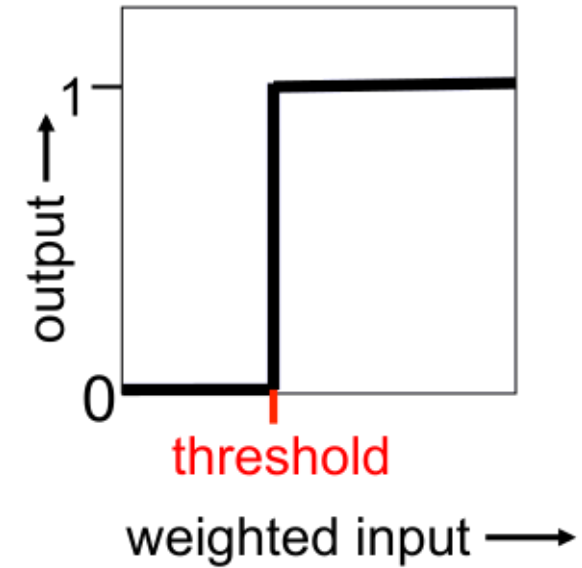
$$y = b + \sum_i x_i w_i$$



Binary threshold neurons

McCulloch-Pitts (1943):

- First compute a weighted sum of the inputs.
- Then send out a fixed size spike of activity if the weighted sum exceeds a threshold.
- McCulloch and Pitts thought that each spike is like the truth value of a proposition and each neuron combines truth values to compute the truth value of another proposition.



Binary threshold neurons

There are two equivalent ways to write the equations for a binary threshold neuron:

$$z = \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

$$\theta = -b$$

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

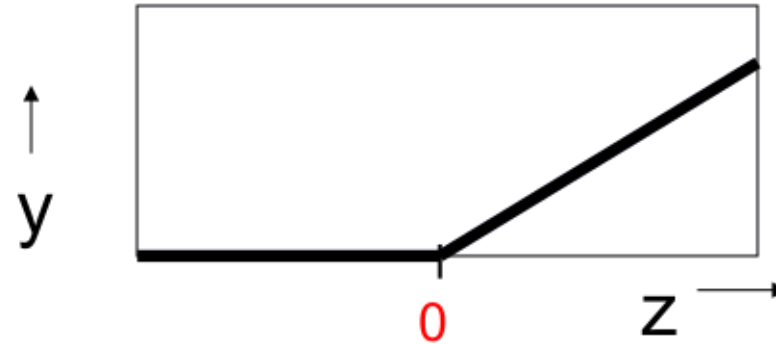


Rectified linear neurons

Also called linear threshold neurons.

They compute a **linear** weighted sum of their inputs.
The output is a **non-linear** function of the total input.

$$z = b + \sum_i x_i w_i$$
$$y = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{otherwise} \end{cases}$$

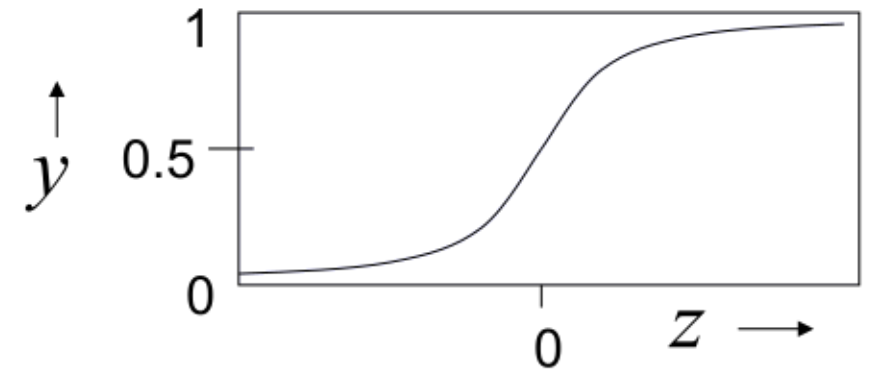


Sigmoid neurons

These give a real-valued output that is a smooth and bounded function of their total input.

- Typically they use the logistic function
- They have nice derivatives which make learning easy.

$$z = b + \sum_i x_i w_i \quad y = \frac{1}{1 + e^{-z}}$$



Stochastic binary neurons

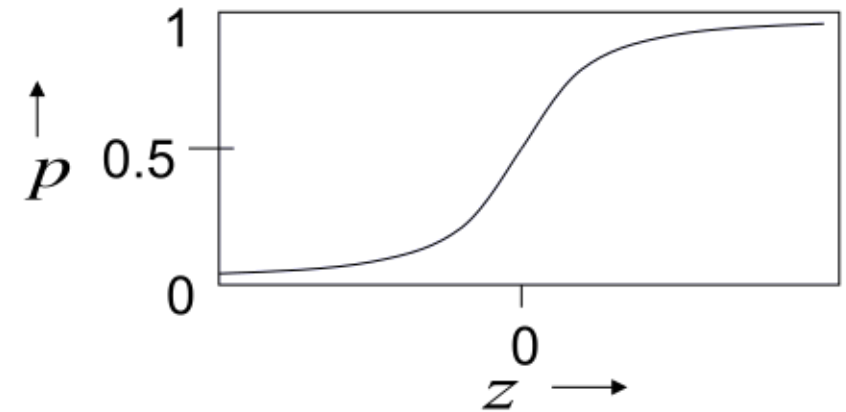
These use the same equations as logistic units.

- But they treat the output of the logistic as the probability of producing a spike in a short time window.

We can do a similar trick for rectified linear units:

- The output is treated as the Poisson rate for spikes.

$$z = b + \sum_i x_i w_i \quad p(s=1) = \frac{1}{1 + e^{-z}}$$



Early Applications



University of
South Australia

Recognizing handwritten shapes

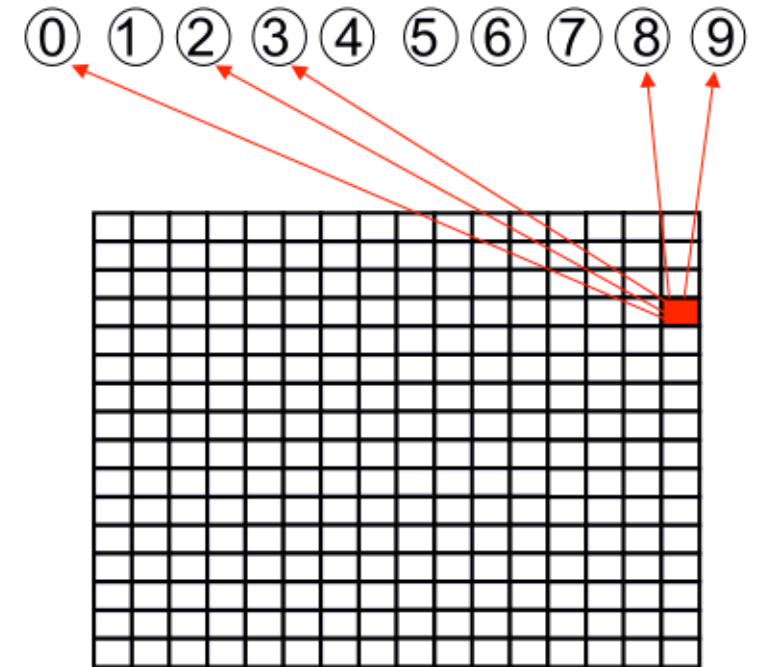
Consider a neural network with two layers of neurons.

- neurons in the top layer represent known shapes.
- neurons in the bottom layer represent pixel intensities.

A pixel gets to vote if it has ink on it.

- Each inked pixel can vote for several different shapes.

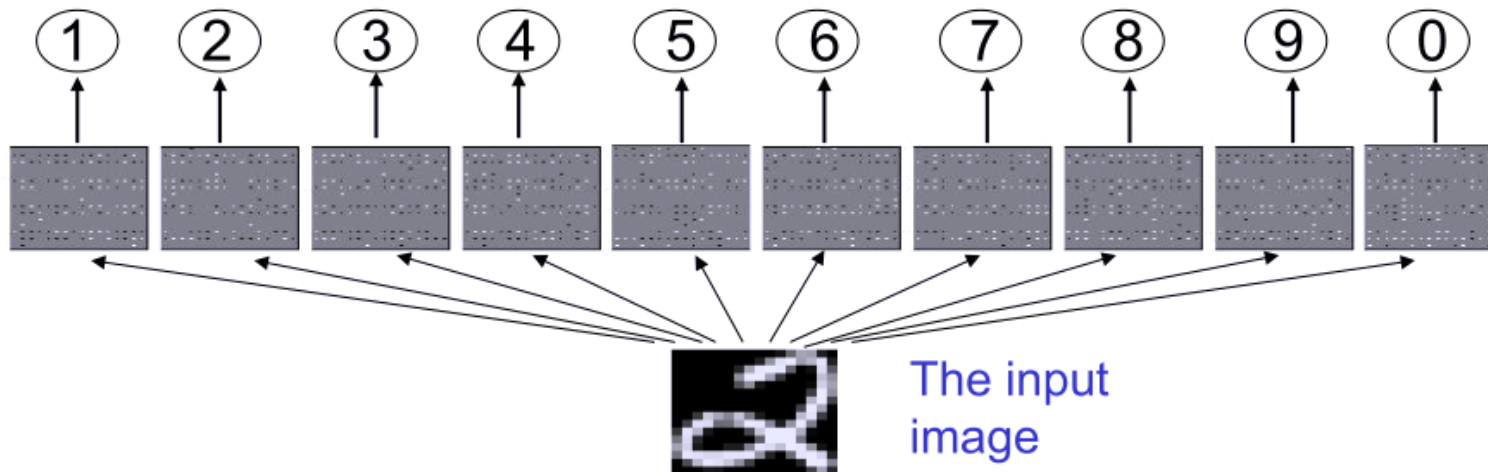
The shape that gets the most votes wins



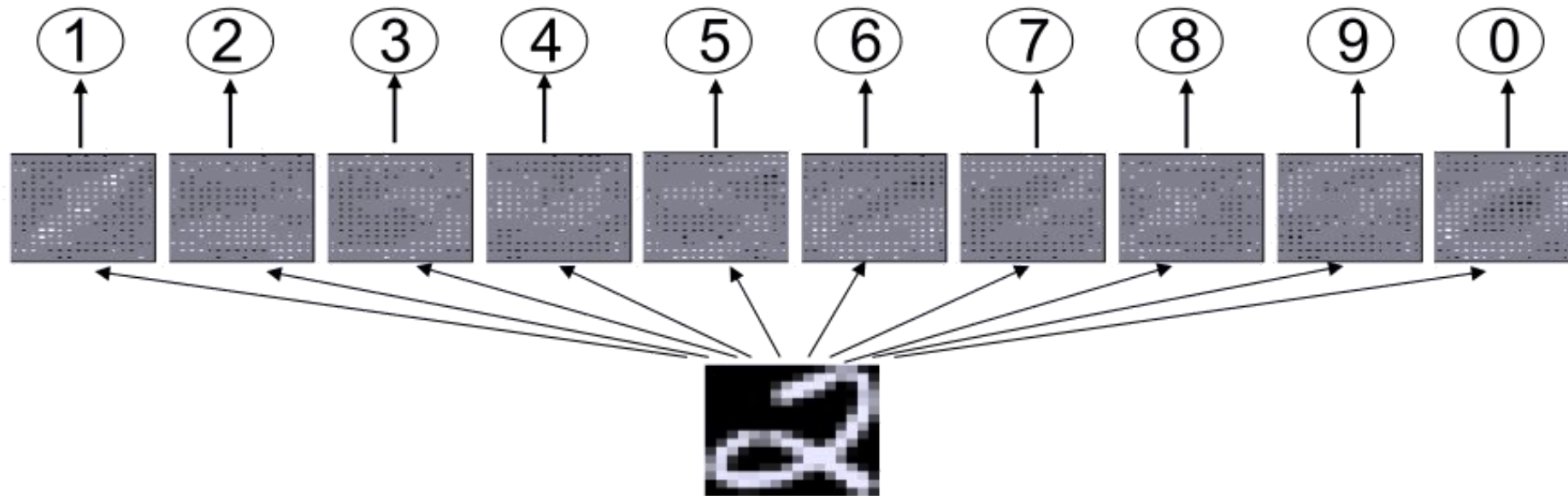
Displaying the weights

Give each output unit its own “map” of the input image and display the weight coming from each pixel in the location of that pixel in the map.

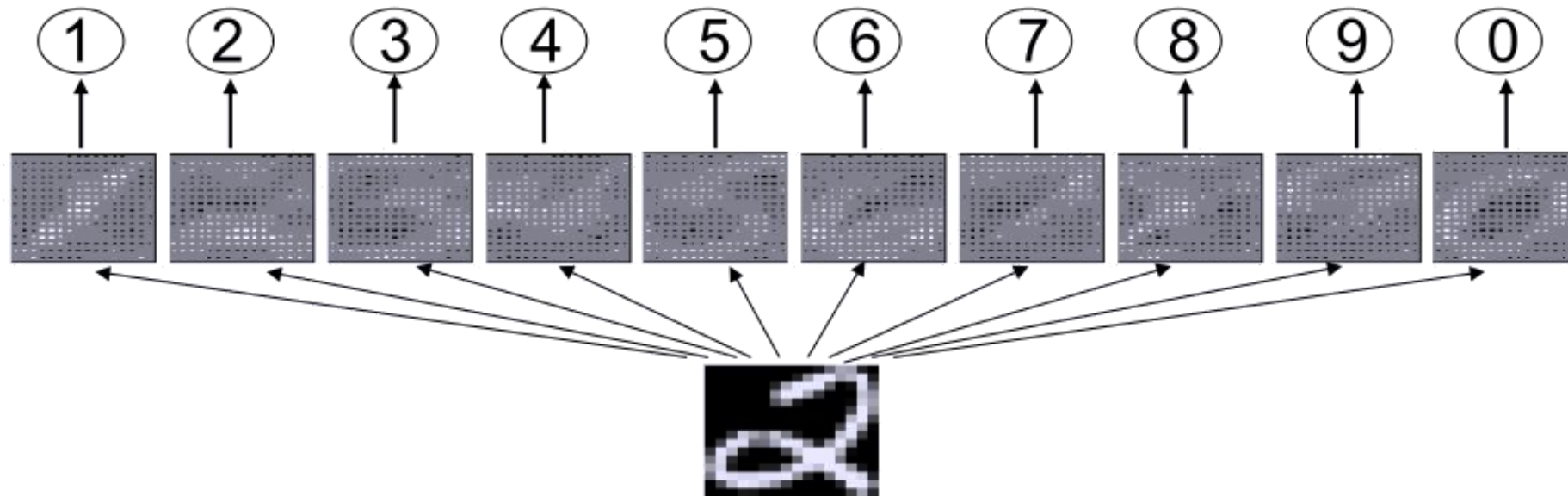
Use a black or white blob with the area representing the magnitude of the weight and the colour representing the sign.



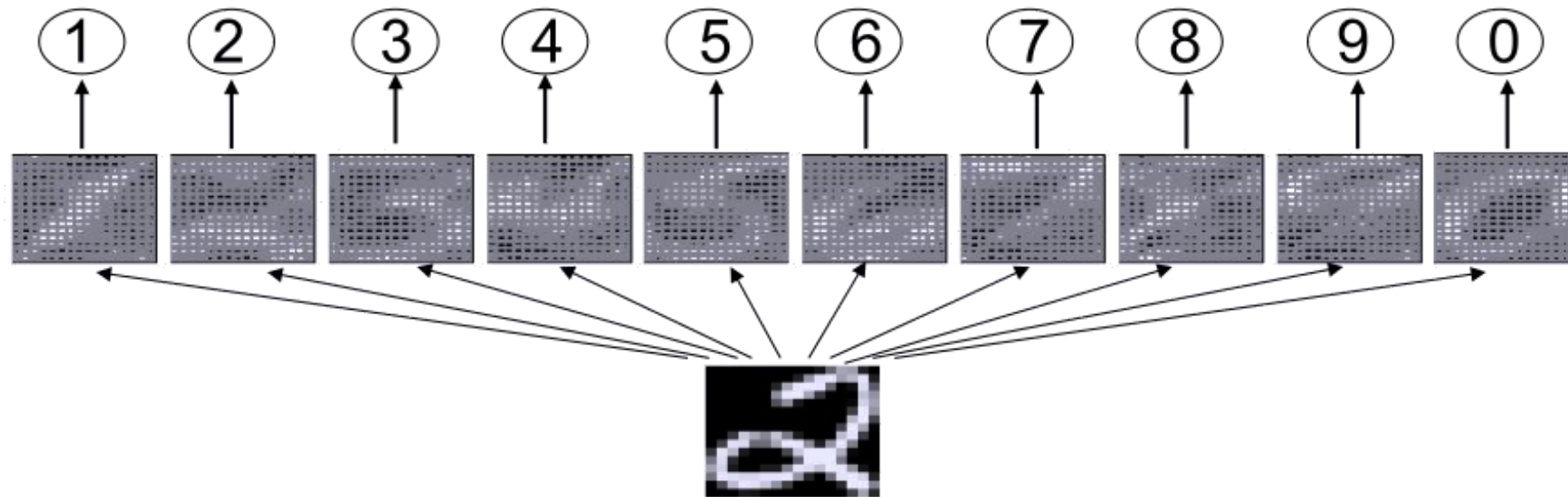
Learning the weights



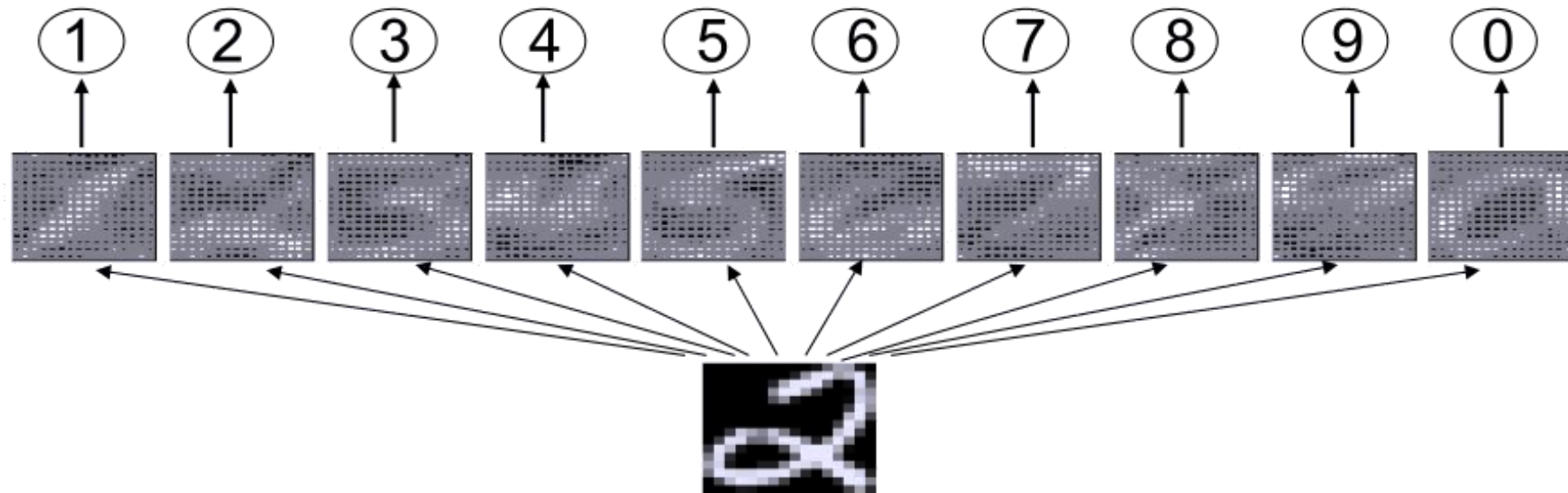
Learning the weights



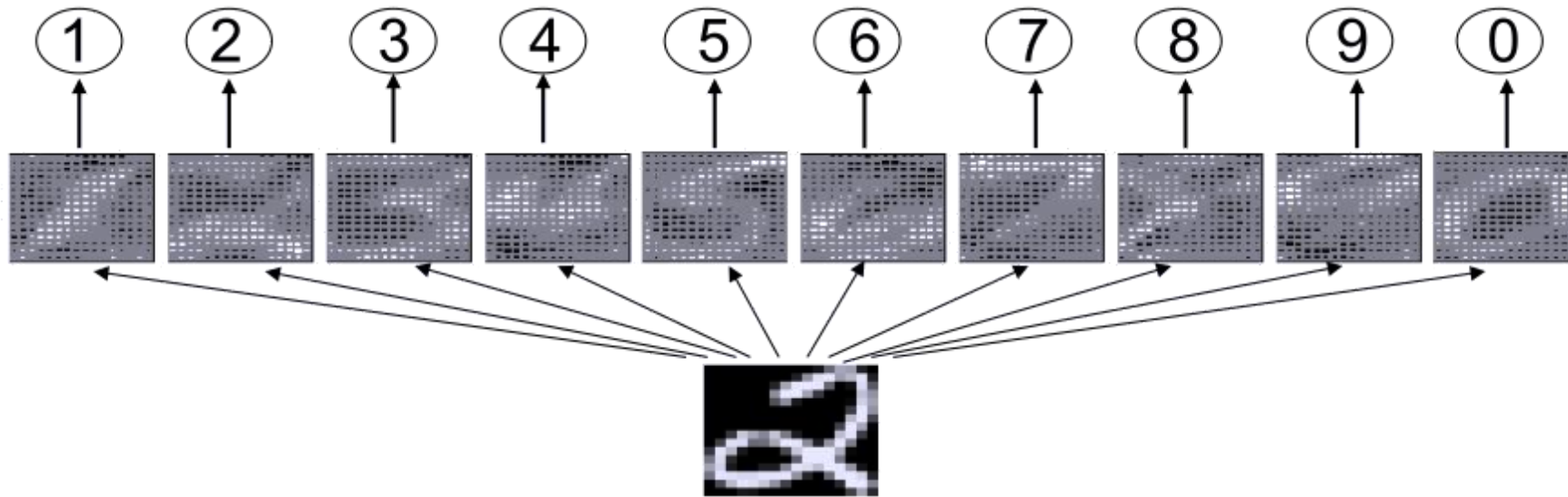
Learning the weights



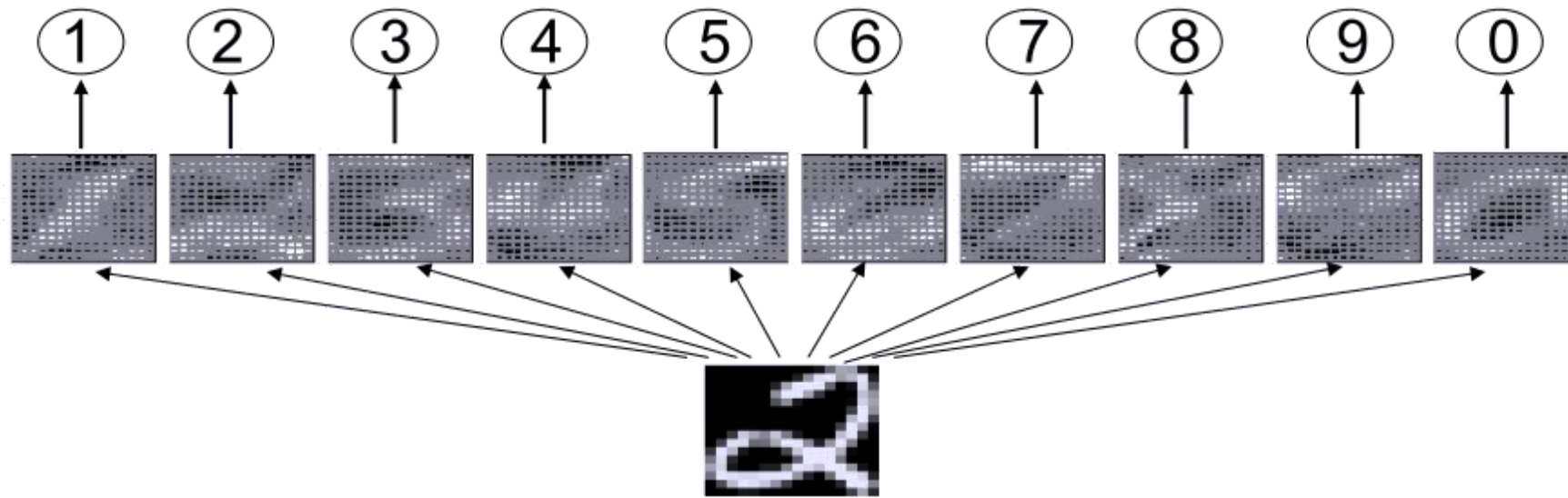
Learning the weights



Learning the weights



Learning the weights



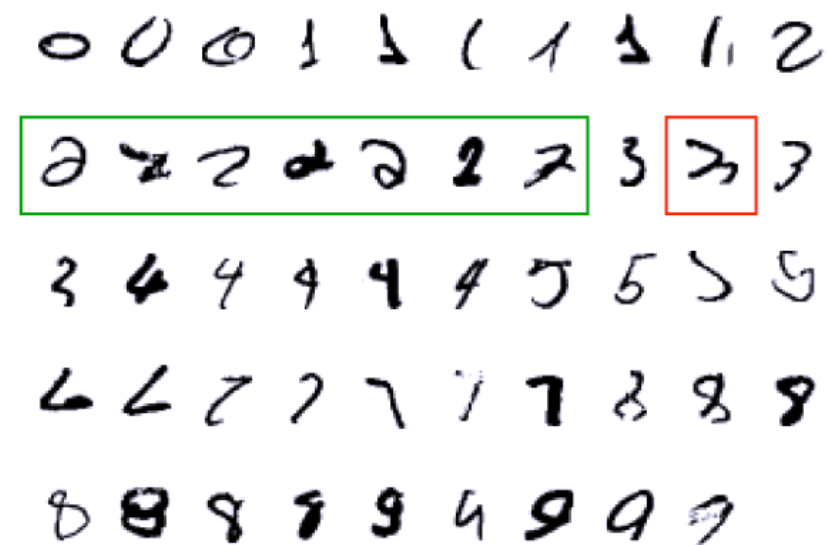
Why the simple learning algorithm is insufficient?

A two layer network with a single winner in the top layer is equivalent to having a rigid template for each shape.

- The winner is the template that has the biggest overlap with the ink.

The ways in which hand-written digits vary are much too complicated to be captured by simple template matches of whole shapes.

- To capture all the allowable variations of a digit we need to learn the features that it is composed of.



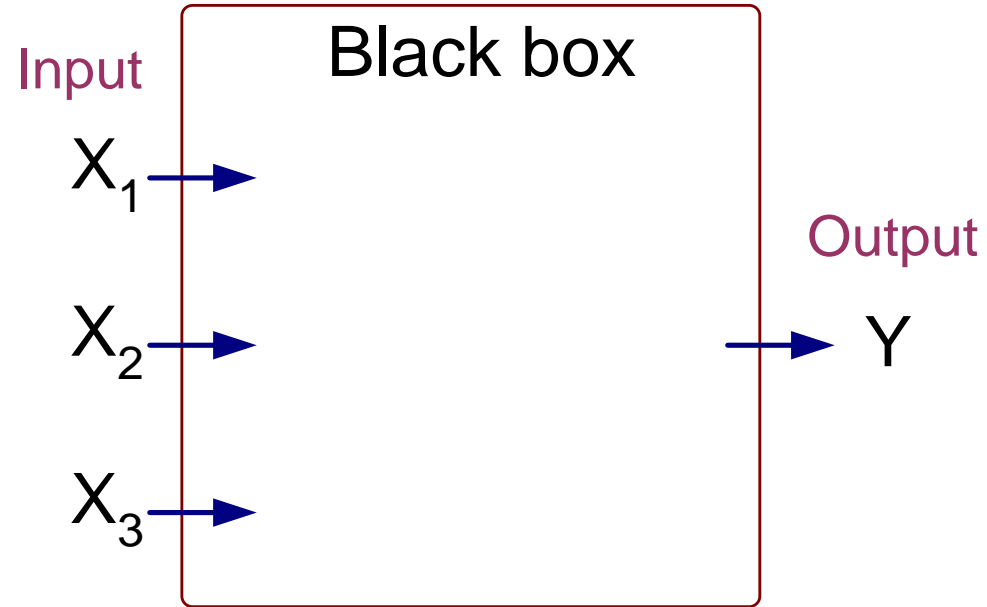
Understanding neural networks



University of
South Australia

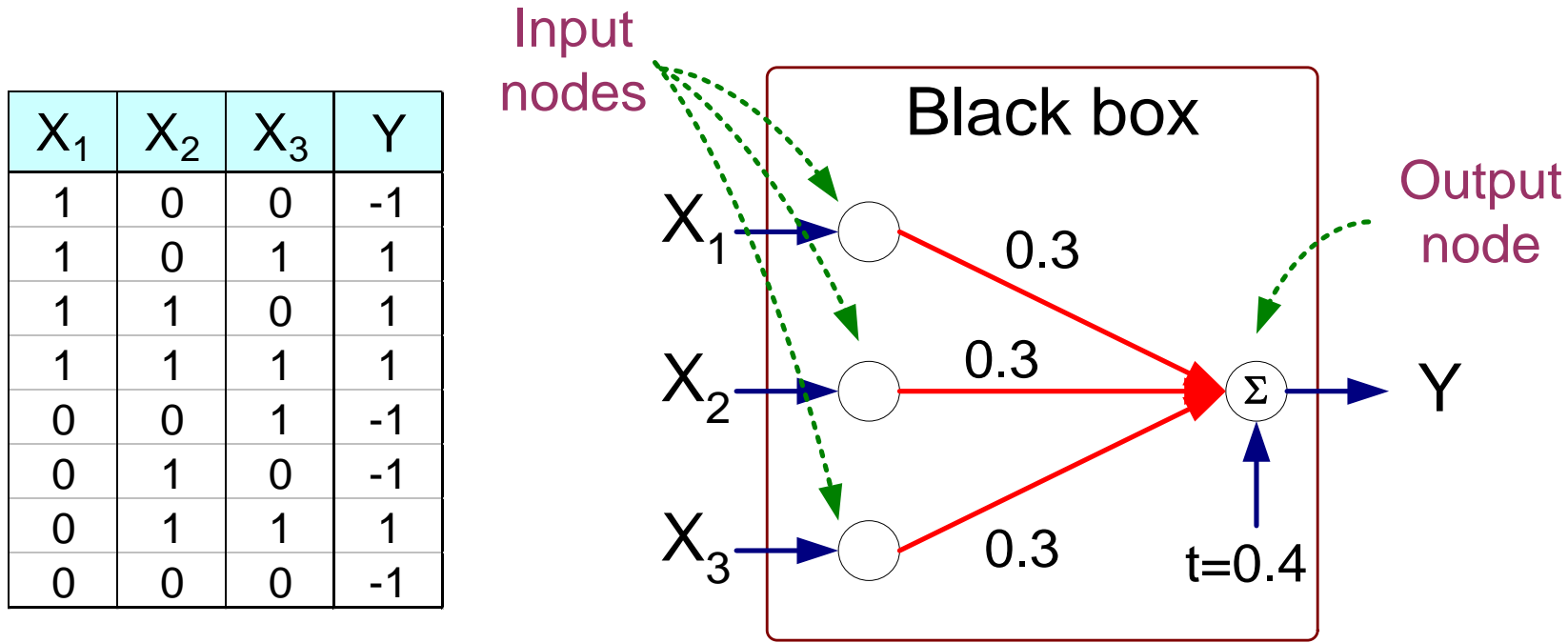
Artificial Neural Networks (ANN)

X_1	X_2	X_3	Y
1	0	0	-1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	-1
0	1	0	-1
0	1	1	1
0	0	0	-1



Output Y is 1 if at least two of the three inputs are equal to 1.

Artificial Neural Networks (ANN)

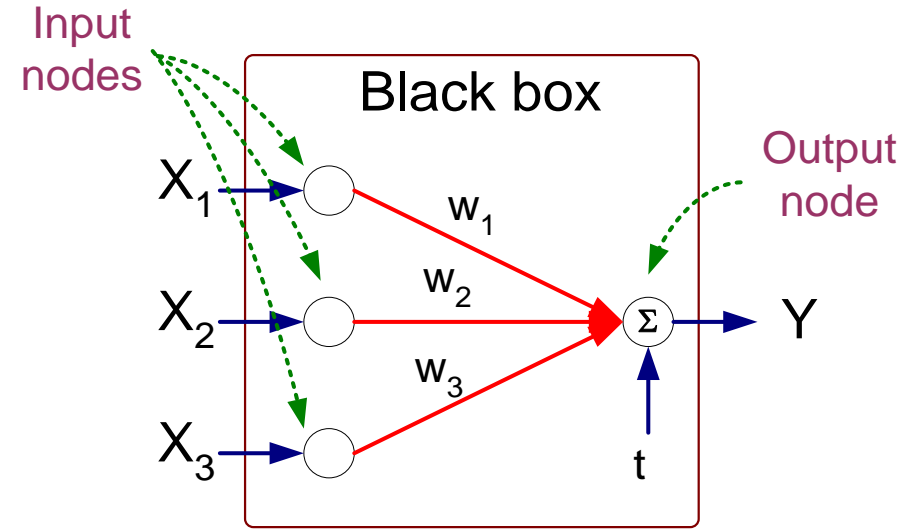


$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Artificial Neural Networks (ANN)

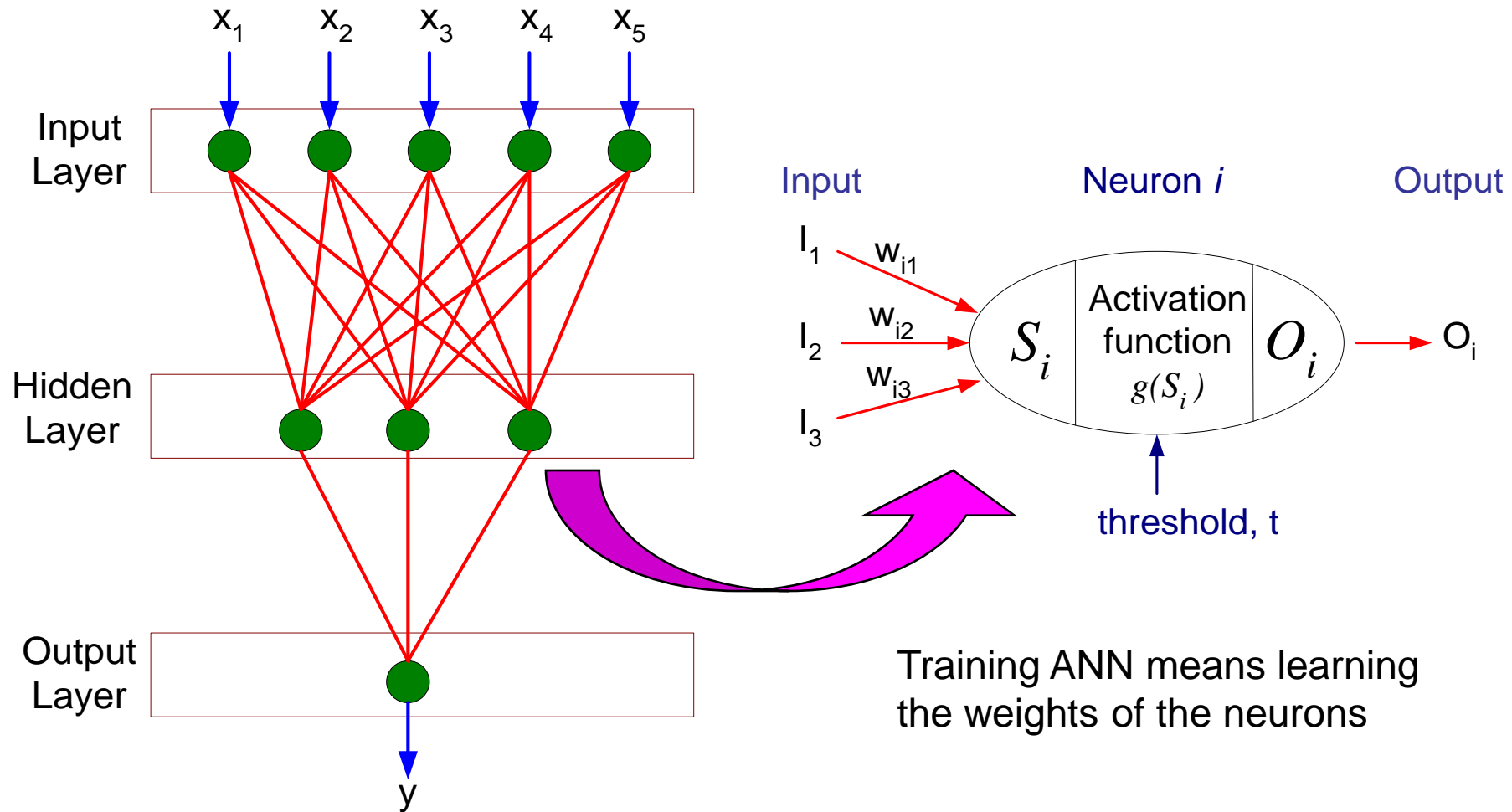
- Model is an assembly of inter-connected nodes and weighted links
- Output node sums up each of its input value according to the weights of its links
- Compare output node against some threshold t



Perceptron Model

$$Y = \text{sign}\left(\sum_{i=1}^d w_i X_i - t\right)$$
$$= \text{sign}\left(\sum_{i=0}^d w_i X_i\right)$$

General Structure of ANN

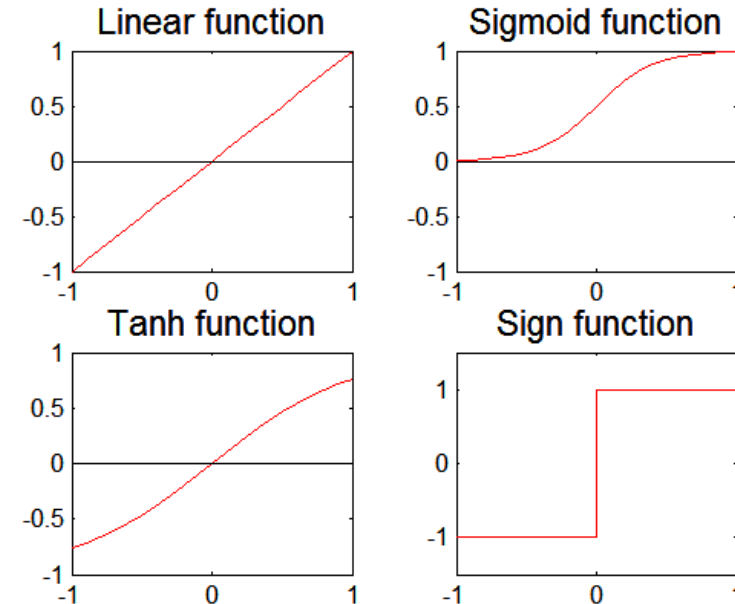


Artificial Neural Networks (ANN)

- Various types of neural network topology
 - single-layered network (perceptron) versus multi-layered network
 - Feed-forward versus recurrent network

- Various types of activation functions (f)

$$Y = f\left(\sum_i w_i X_i\right)$$



Perceptron



University of
South Australia

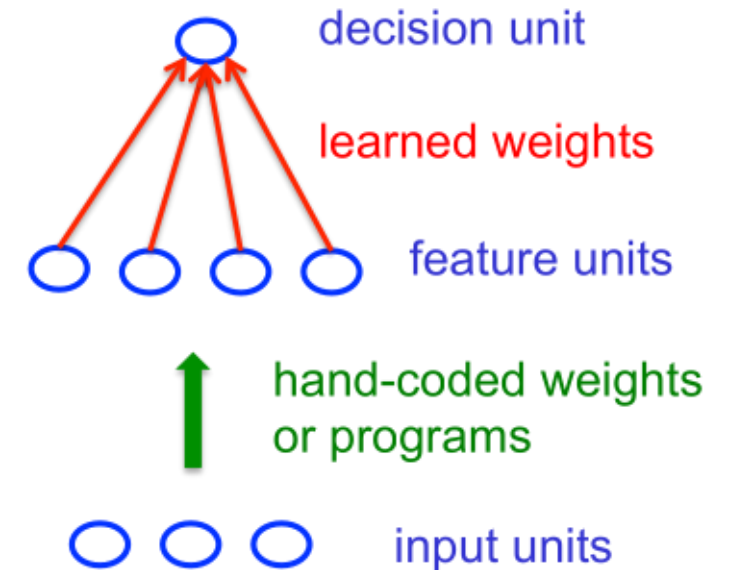
The first generation of neural networks

The standard paradigm for statistical pattern recognition:

Step 1. Convert the raw input vectors into a vector of feature activations

Step 2. Learn how to weight each of the feature activations to get a single scalar quantity.

Step 3. If this quantity is above some threshold, decide that the input vector is a positive example of the target class.



The history of perceptron

They were popularised by Frank Rosenblatt in the early 1960's.

- They appeared to have a very powerful learning algorithm.
- Lots of grand claims were made for what they could learn to do.

In 1969, Minsky and Papert published a book called “Perceptrons” that analysed what they could do and showed their limitations.

- Many people thought these limitations applied to all neural network models.

The perceptron learning procedure is still widely used today for tasks with enormous feature vectors that contain many millions of features.



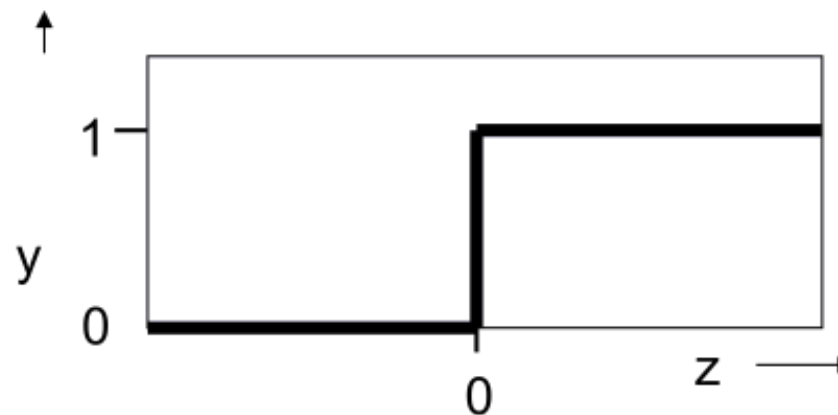
Binary threshold neurons (decision units)

McCulloch-Pitts (1943)

- First compute a weighted sum of the inputs from other neurons (plus a bias).
- Then output a 1 if the weighted sum exceeds zero.

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

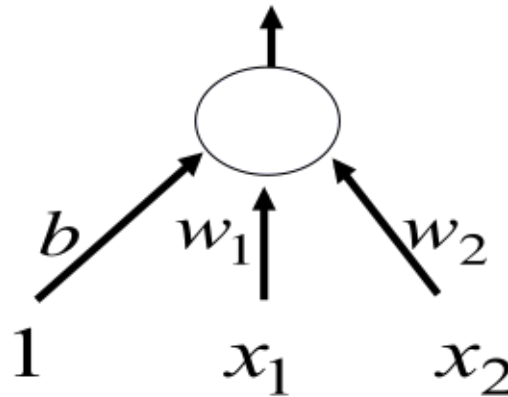


How to learn biases?

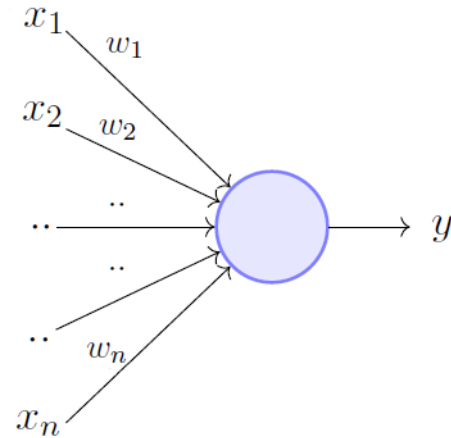
A threshold is equivalent to having a negative bias.

We can avoid having to figure out a separate learning rule for the bias by using a trick:

- A bias is exactly equivalent to a weight on an extra input line that always has an activity of 1.
- We can now learn a bias as if it were a weight.



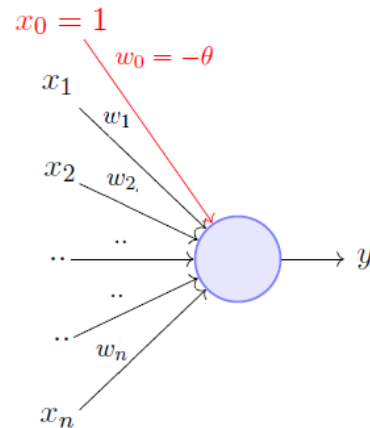
Quick reminder



$$y = 1 \quad \text{if} \quad \sum_{i=1}^n w_i * x_i \geq \theta$$
$$= 0 \quad \text{if} \quad \sum_{i=1}^n w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad \text{if} \quad \sum_{i=1}^n w_i * x_i - \theta \geq 0$$
$$= 0 \quad \text{if} \quad \sum_{i=1}^n w_i * x_i - \theta < 0$$



A more accepted convention,

$$y = 1 \quad \text{if} \quad \sum_{i=0}^n w_i * x_i \geq 0$$
$$= 0 \quad \text{if} \quad \sum_{i=0}^n w_i * x_i < 0$$

where, $x_0 = 1$ and $w_0 = -\theta$

Perceptron

- Single layer network
 - Contains only input and output nodes
- Activation function: $f = \text{sign}(w \bullet x)$
- Applying model is straightforward

$$Y = \text{sign}(0.3X_1 + 0.3X_2 + 0.3X_3 - 0.4)$$

$$\text{where } \text{sign}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

- $X_1 = 1, X_2 = 0, X_3 = 1 \Rightarrow y = \text{sign}(0.2) = 1$

Perceptron Learning Rule

- Initialize the weights (w_0, w_1, \dots, w_d)
- Repeat
 - For each training example (x_i, y_i)
 - ◆ Compute $f(w, x_i)$
 - ◆ Update the weights:

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i$$

- Until stopping condition is met

Perceptron Learning Rule

- Basic idea

$$w_{new} = w_{old} + (learning_{rate} * error * input)$$

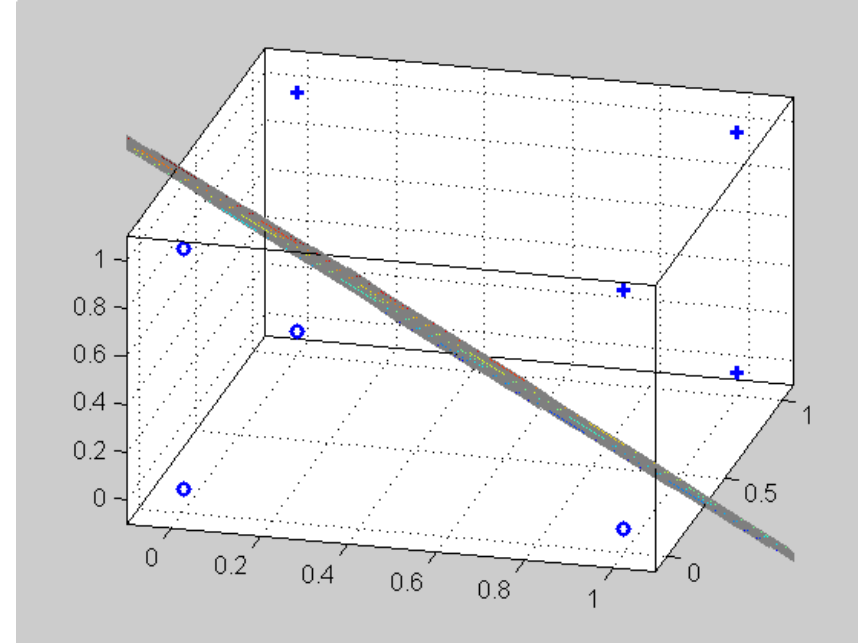
- Weight update formula:

$$w^{(k+1)} = w^{(k)} + \lambda [y_i - f(w^{(k)}, x_i)] x_i ; \lambda: \text{learning rate}$$

- More on this in our Practical session

Perceptron Learning Rule

- Since $f(w, x)$ is a linear combination of input variables, decision boundary is linear



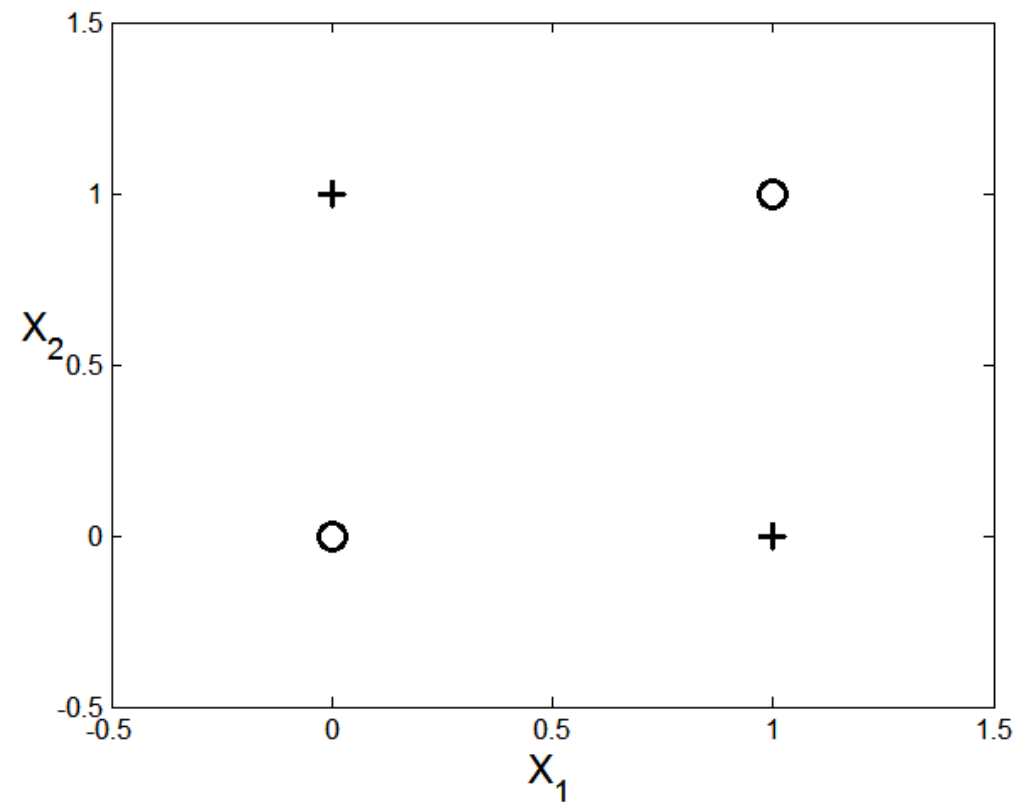
- For nonlinearly separable problems, perceptron learning algorithm will fail because no linear hyperplane can separate the data perfectly

Nonlinearly Separable Data

$$y = x_1 \oplus x_2$$

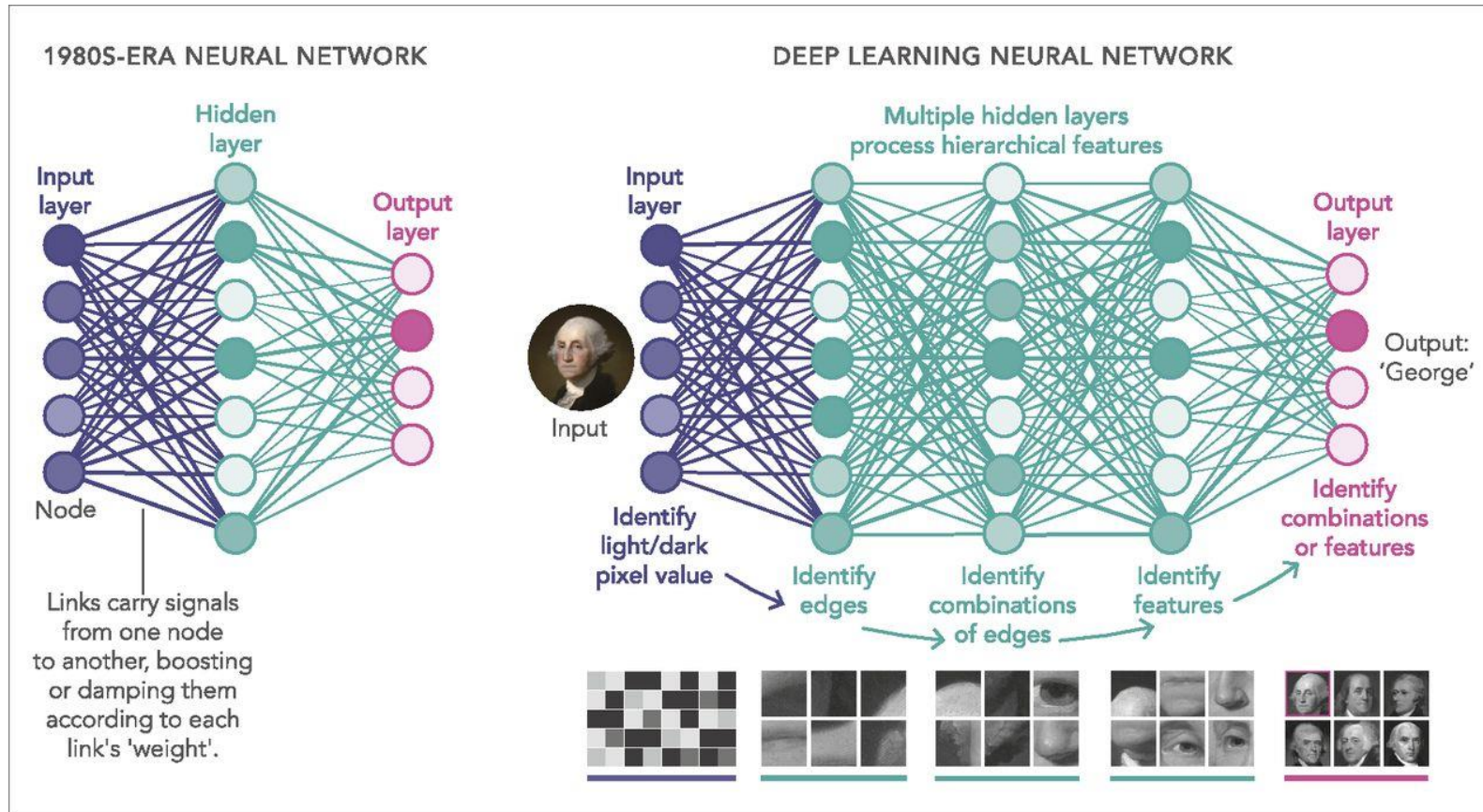
x_1	x_2	y
0	0	-1
1	0	1
0	1	1
1	1	-1

XOR Data



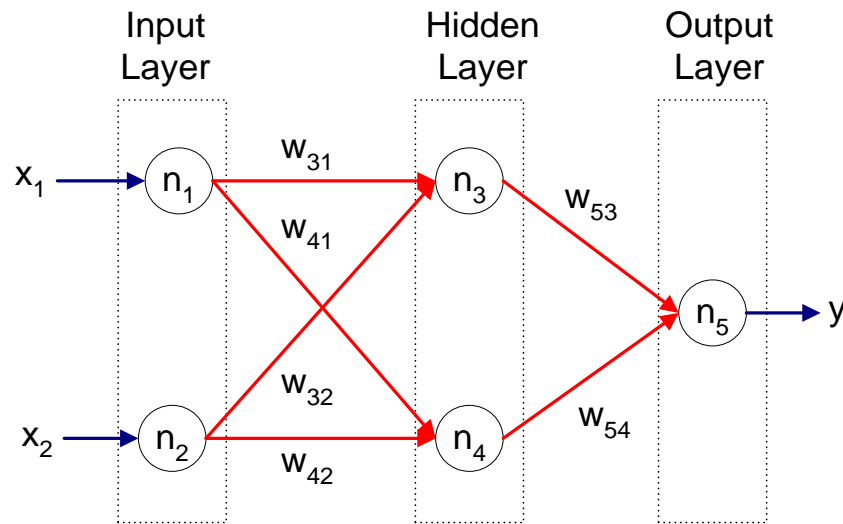
Multilayer Neural Network

- Hidden layers
 - intermediary layers between input & output layers
- More general activation functions (sigmoid or linear, for example)

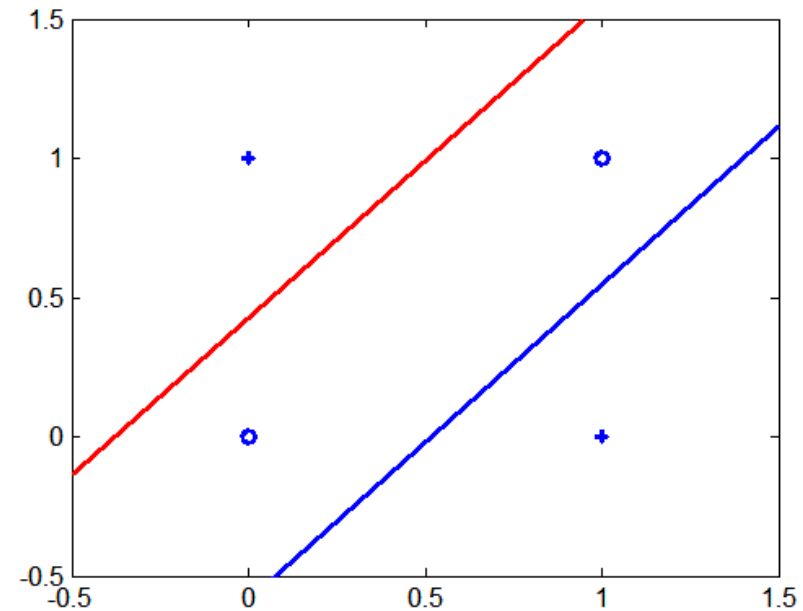


Multi-layer Neural Network

- Multi-layer neural network can solve any type of classification task involving nonlinear decision surfaces



XOR Data



Learning Multi-layer Neural Network

- The weights and bias terms (w , b) are learned during the training process, where we try to minimize the error – the difference between actual and predicted value.
- This is achieved by using a loss function

$$E(w, b) = \sum_{k=1}^n \text{Loss}(y_k, \hat{y}_k)$$

Learning Multi-layer Neural Network

- There are several loss functions that are commonly used

- Squared loss function

$$Loss(y_k, \hat{y}_k) = (y_k - \hat{y}_k)^2$$

- Mean squared error

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

MSE = mean squared error

n = number of data points

Y_i = observed values

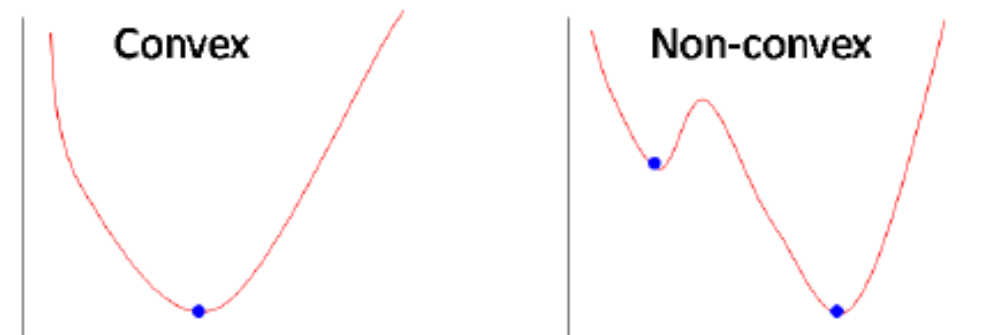
\hat{Y}_i = predicted values

- Binary cross entropy

$$Loss(y_i, \hat{y}_i) = -\frac{1}{n} \sum_{i=1}^n y_i \cdot \log(\hat{y}_i)$$

What is Gradient Descent?

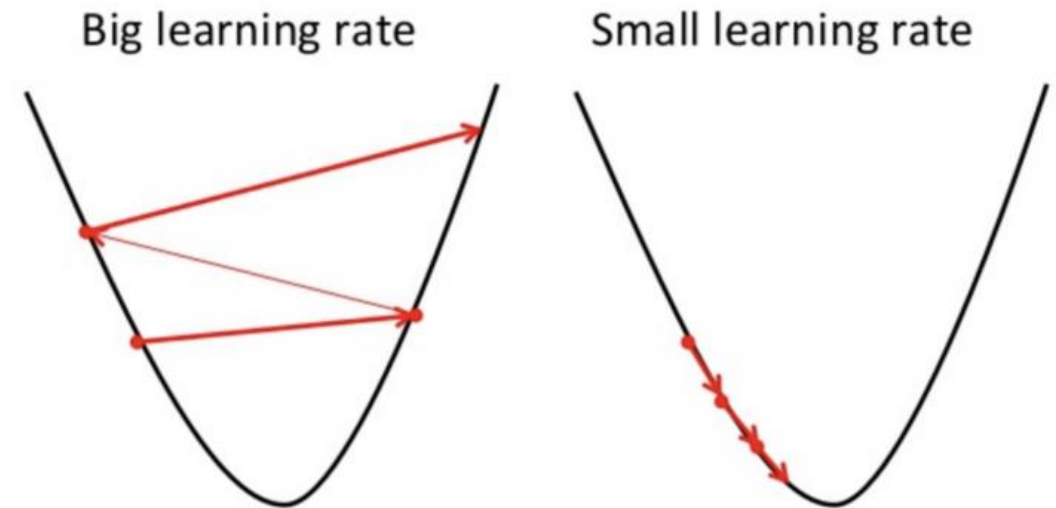
- Gradient descent is by far the most popular optimization strategy used in machine learning, at the moment.
- It is based on a convex function and tweaks its parameters iteratively to minimize a given function to its local minimum.
- We start by defining the initial parameter's values
 - from there gradient descent uses calculus to iteratively adjust the values so they minimize the given cost-function.



Source: <https://builtin.com/data-science/gradient-descent>

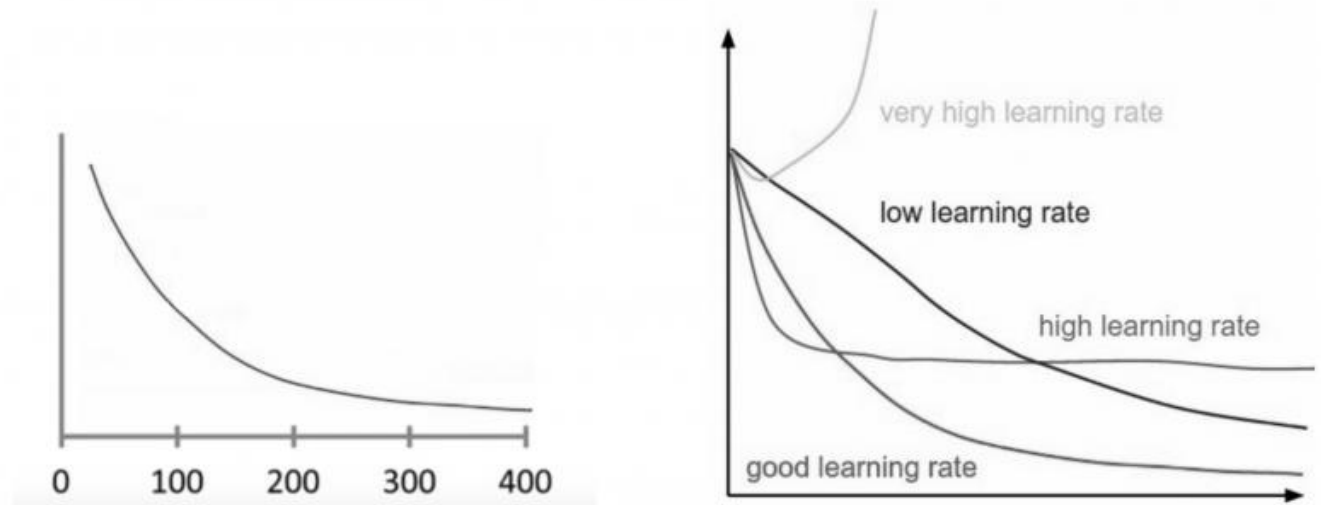
Importance of the learning rate

- Learning rate defines how big the steps are gradient descent takes into the direction of the local minimum, which figures out how fast or slow we will move towards the optimal weights.
- For gradient descent to reach the local minimum we must set the learning rate to an appropriate value, which is neither too low nor too high.
- This is important because if the steps it takes are too big, it may not reach the local minimum because it bounces back and forth between the convex function of gradient descent (see left image below).
- If we set the learning rate to a very small value, gradient descent will eventually reach the local minimum but that may take a while (see the right image).



How to find an optimal learning rate?

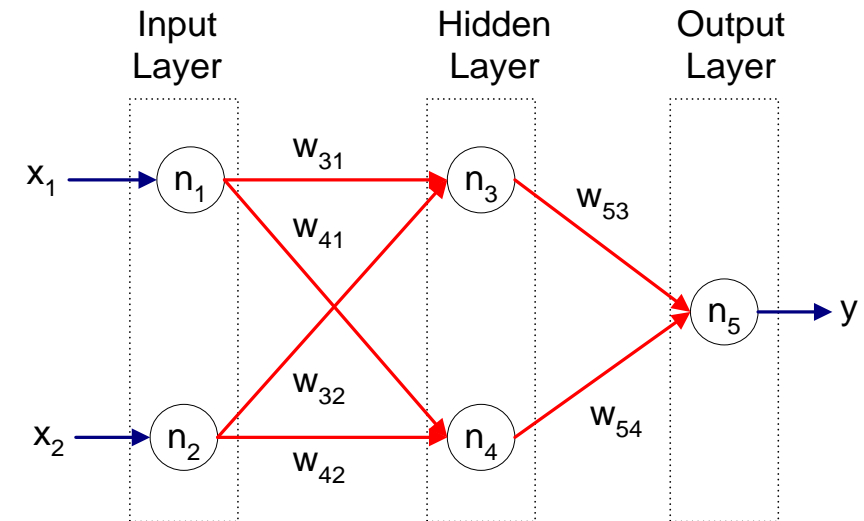
- Grid search
- Learning rate schedules
- Adaptive learning rate methods



- Monitoring: Plot the cost function as the optimization runs
 - number of iterations on the x-axis and the value of the cost-function on the y-axis.
- If gradient descent is working properly, the cost function should decrease after every iteration.
- When gradient descent cannot decrease the cost-function anymore and remains more or less on the same level, it has converged.
- Hard to estimate number of iterations gradient descent needs to converge.

Multi-layer NN learning parameters

- Learning involves adjusting the weights and biases of the connections between neurons to minimize the error between the predicted output and the true output.
- This is usually done through a process called **backpropagation** - “backward propagation of errors”.
- Backpropagation is a two-phase process:
 - Forward propagation
 - Backpropagation



Multi-layer NN learning parameters

□ Forward pass

$$y^l = w^l * x^{l-1} + b^l \quad \text{Linear combination}$$

$$x^l = g(y^l) \quad \text{Activation function}$$

l – the layer number

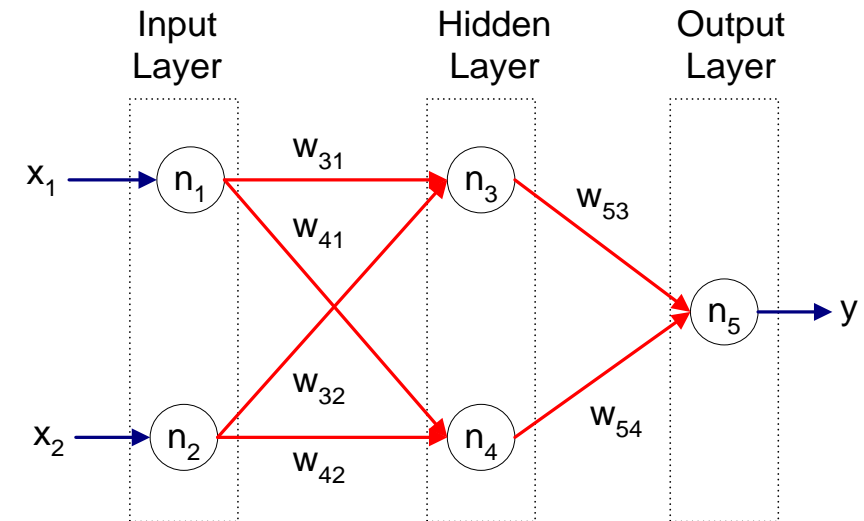
w^l – weights of the l – th layer

b^l – bias of the l – th layer

x^{l-1} – the activation of the previous layer

y^l – the pre – activation of the current layer

□ Backpropagation



Multi-layer NN learning parameters

□ Backpropagation

- Start with the output layer: Calculate the **derivative of the error function with respect to each weight connecting the output layer** neurons to the previous layer (e.g., the input layer or another hidden layer). This derivative tells you **how much the error changes as you change each weight**. You can use the chain rule of calculus to compute this derivative layer by layer, starting from the output layer and working your way backward to the input layer. This is why the algorithm is called "backpropagation."
- Once you have the derivative of the error function with respect to each weight, you can **update the weights using gradient descent**: for each weight w_j , subtract the product of the derivative and the learning rate (λ) from the weight:

$$w_j \leftarrow w_j - \lambda \frac{\partial E(w)}{\partial w_j}$$

- Repeat the process for multiple epochs (passes through the training data), adjusting the weights each time, until the error is minimized or some other stopping criterion is met.

Step-by-step guide

<https://www.jeremyjordan.me/neural-networks-training/>

Types of Neural Networks



University of
South Australia

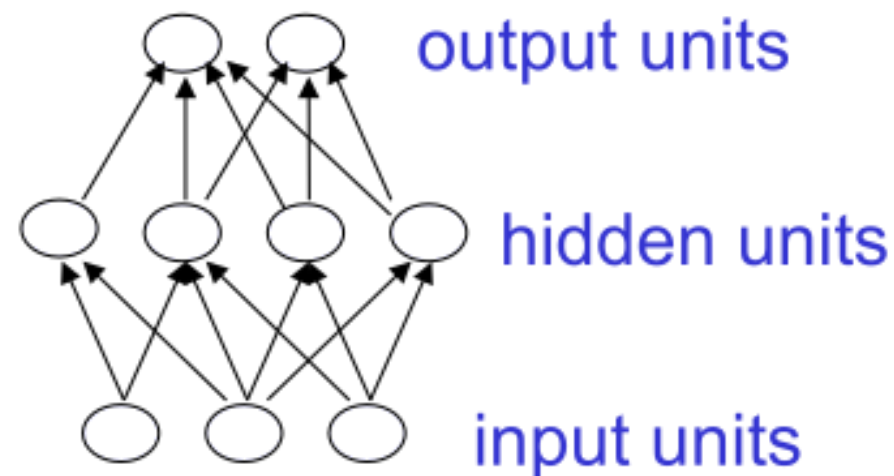
Feed-forward neural networks

These are the commonest type of neural network in practical applications.

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

They compute a series of transformations that change the similarities between cases.

- The activities of the neurons in each layer are a non-linear function of the activities in the layer below.



Recurrent networks

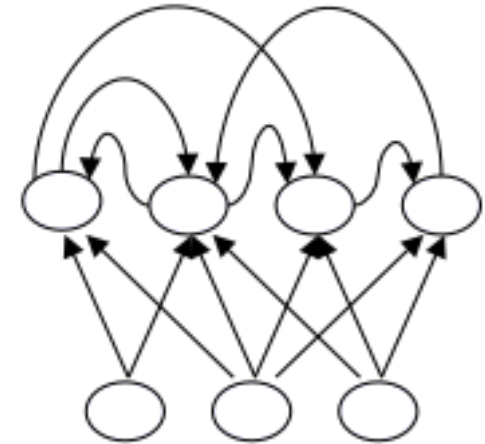
These have directed cycles in their connection graph.

- That means you can sometimes get back to where you started by following the arrows.

They can have complicated dynamics and this can make them very difficult to train.

- There is a lot of interest at present in finding efficient ways of training recurrent nets.

They are more biologically realistic.



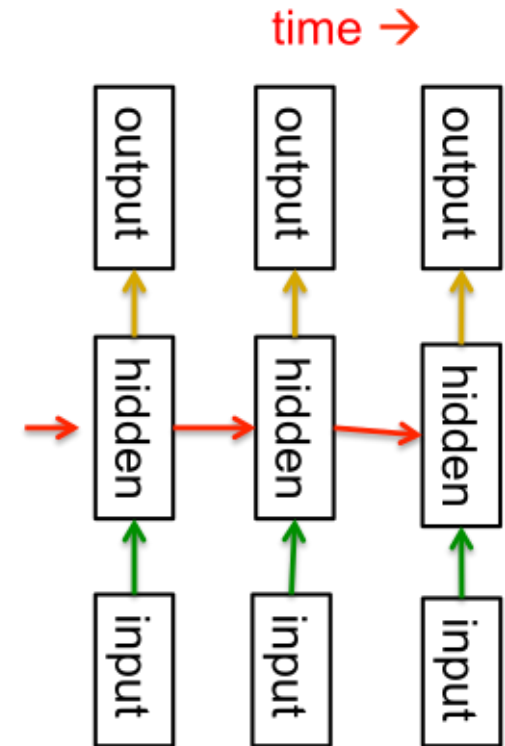
Recurrent neural networks for modeling sequences

Recurrent neural networks are a very natural way to model sequential data:

- They are equivalent to very deep nets with one hidden layer per time slice.
- Except that they use the same weights at every time slice and they get input at every time slice.

They have the ability to remember information in their hidden state for a long time.

- But its very hard to train them to use this potential.



Recurrent neural networks - application

Ilya Sutskever (2011) trained a special type of recurrent neural net to predict the next character in a sequence.

After training for a long time on a string of half a billion characters from English Wikipedia, he got it to generate new text.

- It generates by predicting the probability distribution for the next character and then sampling a character from this distribution.



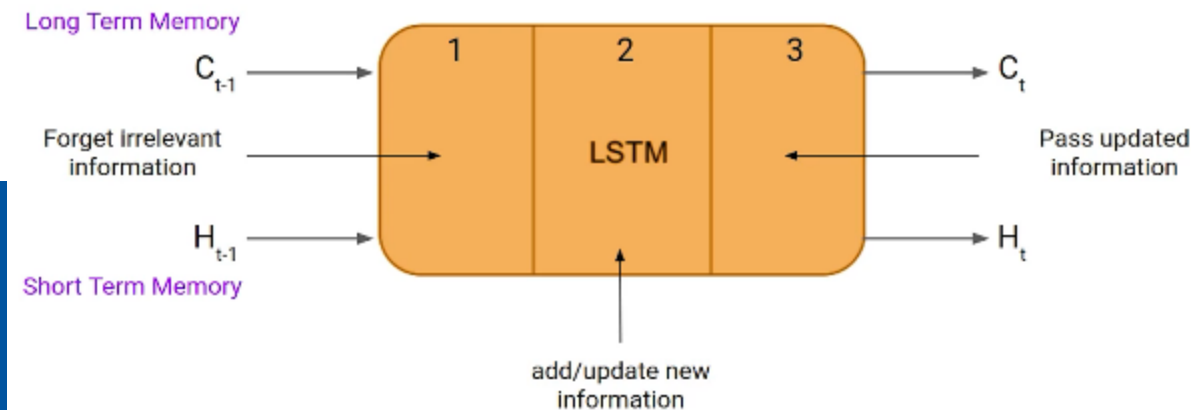
Text generated one character at a time by Ilya Sutskever's recurrent neural network

In 1974 Northern Denver had been overshadowed by CNL, and several Irish intelligence agencies in the Mediterranean region. However, on the Victoria, Kings Hebrew stated that Charles decided to escape during an alliance. The mansion house was completed in 1882, the second in its bridge are omitted, while closing is the proton reticulum composed below it aims, such that it is the blurring of appearing on any well-paid type of box printer.



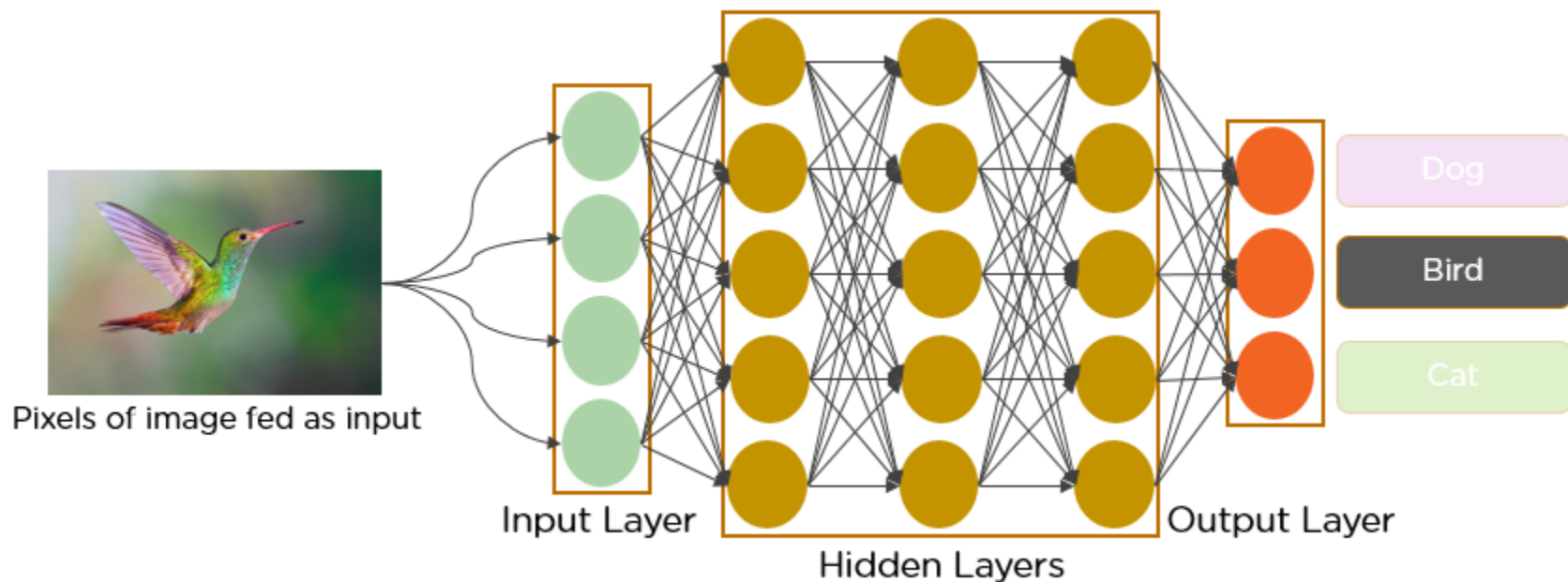
Long Short-Term Memory Networks

- A deep learning, sequential neural network that allows information to persist.
- A special type of Recurrent Neural Network
- Details here -> <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/>



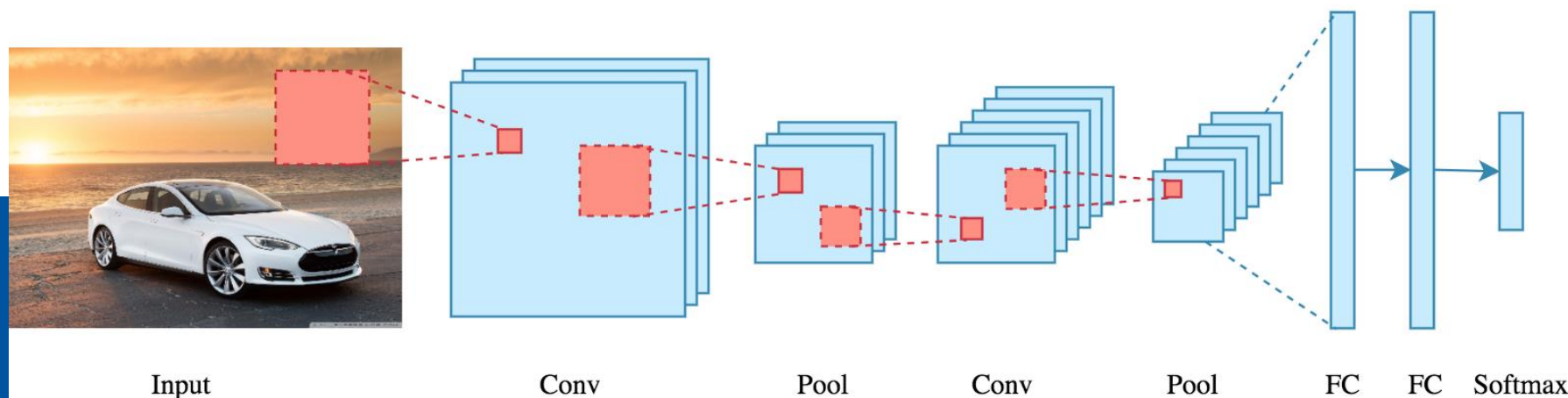
Convolutional networks

- A class of deep neural networks
- Commonly applied in analysing images
- Uses a special technique called Convolution



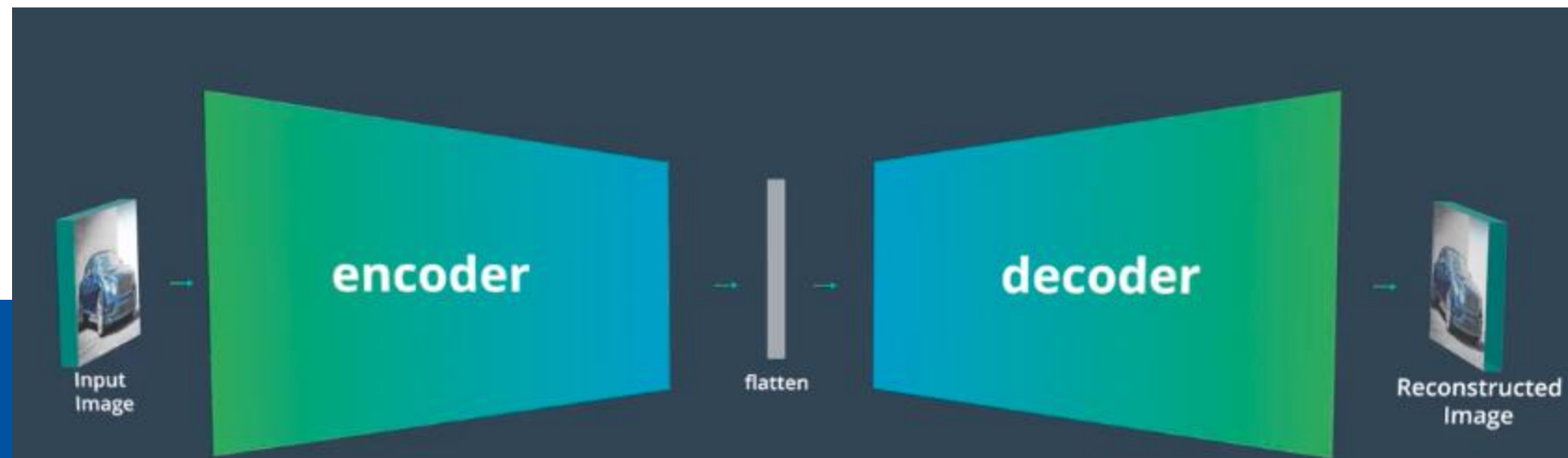
Convolutional networks

- Convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other
- The goal is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.
- Details here -> <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/>



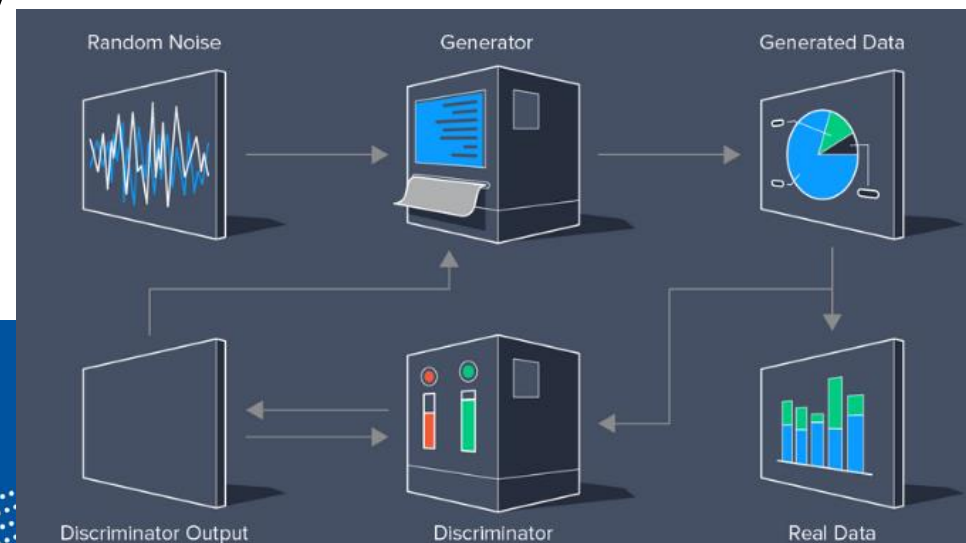
Autoencoders

- Self-supervised models
- Encoders: a compression unit
- Decoder: decompresses the compressed input by reconstructing it
- Details here -> <https://www.analyticsvidhya.com/blog/2021/06/autoencoders-a-gentle-introduction/>



Generative Adversarial Networks (GANs)

- GANs are a type of neural network that consists of two components - a generator and a discriminator - that are trained together in a game-like setting.
- They are often used for tasks such as image and video generation, and have been used to create realistic images of faces, landscapes, and other objects.
- Details here -> <https://www.analyticsvidhya.com/blog/2021/10/an-end-to-end-introduction-to-generative-adversarial-networks-gans/>



Recent Developments



University of
South Australia

Large Language Models

```
Prompt > Gradient descent is a first-order iterative
Prompt > Artificial intelligence (AI), sometimes ca
Prompt > ZDNet is a business technology news websit
Prompt > OpenAI is an artificial intelligence resear
ZDNet > GPT-3 is the next word in AI|
Prompt > Deep learning (also known as deep structur
Prompt > Unsupervised learning is a type of machine
Prompt > Labeled data is a group of samples that ha
Prompt > Conditional probability is a measure of th
```

Human-supplied input: Shadows on the Way Wallace Stevens

GPT-3-generated completion:

I must have shadows on the way If I am to walk I must have
Each step taken slowly and alone To have it ready made And I
must think in lines of grey To have dim thoughts to be my
guide Must look on blue and green And never let my eye
forget That color is my friend And purple must surround me
too The yellow of the sun is no more Intrusive than the bluish
snow That falls on all of us. I must have Grey thoughts and
blue thoughts walk with me If I am to go away at all.

Source: <https://www.zdnet.com/article/what-is-gpt-3-everything-business-needs-to-know-about-openais-breakthrough-ai-language-program/>



University of
South Australia

GPT-3

```
Prompt > Gradient descent is a first-order iterative
Prompt > Artificial intelligence (AI), sometimes ca
Prompt > ZDNet is a business technology news websit
Prompt > OpenAI is an artificial intelligence resear
ZDNet > GPT-3 is the next word in AI|
Prompt > Deep learning (also known as deep structur
Prompt > Unsupervised learning is a type of machine
Prompt > Labeled data is a group of samples that ha
Prompt > Conditional probability is a measure of th
```

Human-supplied input: Shadows on the Way Wallace Stevens

GPT-3-generated completion:

I must have shadows on the way If I am to walk I must have
Each step taken slowly and alone To have it ready made And I
must think in lines of grey To have dim thoughts to be my
guide Must look on blue and green And never let my eye
forget That color is my friend And purple must surround me
too The yellow of the sun is no more Intrusive than the bluish
snow That falls on all of us. I must have Grey thoughts and
blue thoughts walk with me If I am to go away at all.

Source: <https://www.zdnet.com/article/what-is-gpt-3-everything-business-needs-to-know-about-openais-breakthrough-ai-language-program/>



University of
South Australia

DALL·E: Creating Images from Text



Source: <https://openai.com/blog/dall-e/>

TEXT PROMPT	an illustration of <u>a pikachu</u> in a beanie <u>wielding a blue</u> <u>lightsaber</u>				
AI-GENERATED IMAGES					



University of
South Australia

Algorithm

Algorithm 6.4 Learning ANN using backpropagation and gradient descent.

- 1: Let $D.train = \{(\mathbf{x}_k, y_k) \mid k = 1, 2, \dots, n\}$ be the set of training instances.
 - 2: Set counter $c \leftarrow 0$.
 - 3: Initialize the weight and bias terms $(\mathbf{w}^{(0)}, \mathbf{b}^{(0)})$ with random values.
 - 4: **repeat**
 - 5: **for** each training instance $(\mathbf{x}_k, y_k) \in D.train$ **do**
 - 6: Compute the set of activations $(a_i^l)_k$ by making a forward pass using \mathbf{x}_k .
 - 7: Compute the set $(\delta_i^l)_k$ by backpropagation using Equations 6.58 and 6.59.
 - 8: Compute $(\partial \text{Loss} / \partial w_{ij}^l, \partial \text{Loss} / \partial b_i^l)_k$ using Equations 6.56 and 6.57.
 - 9: **end for**
 - 10: Compute $\partial E / \partial w_{ij}^l \leftarrow \sum_{k=1}^n (\partial \text{Loss} / \partial w_{ij}^l)_k$.
 - 11: Compute $\partial E / \partial b_i^l \leftarrow \sum_{k=1}^n (\partial \text{Loss} / \partial b_i^l)_k$.
 - 12: Update $(\mathbf{w}^{(c+1)}, \mathbf{b}^{(c+1)})$ by gradient descent using Equations 6.53 and 6.54.
 - 13: Update $c \leftarrow c + 1$.
 - 14: **until** $(\mathbf{w}^{(c+1)}, \mathbf{b}^{(c+1)})$ and $(\mathbf{w}^{(c)}, \mathbf{b}^{(c)})$ converge to the same value
-

Characteristics of ANNs

- ❑ Multi-layer neural networks with at least one hidden layer are **universal approximators**.
- ❑ ANN provides a natural way to represent a hierarchy of features at multiple levels of abstraction.
- ❑ ANN represents complex high-level features as composition of simple low-level features that are easier to represent.
- ❑ ANN can easily handle irrelevant attributes.
- ❑ Solutions obtained by gradient descent are not guaranteed to be globally optimal.
- ❑ Training of ANN is a time consuming process
- ❑ ANN can learn in the presence of interacting variables



University of
South Australia

INFS 5100 Predictive Analytics

Q&A