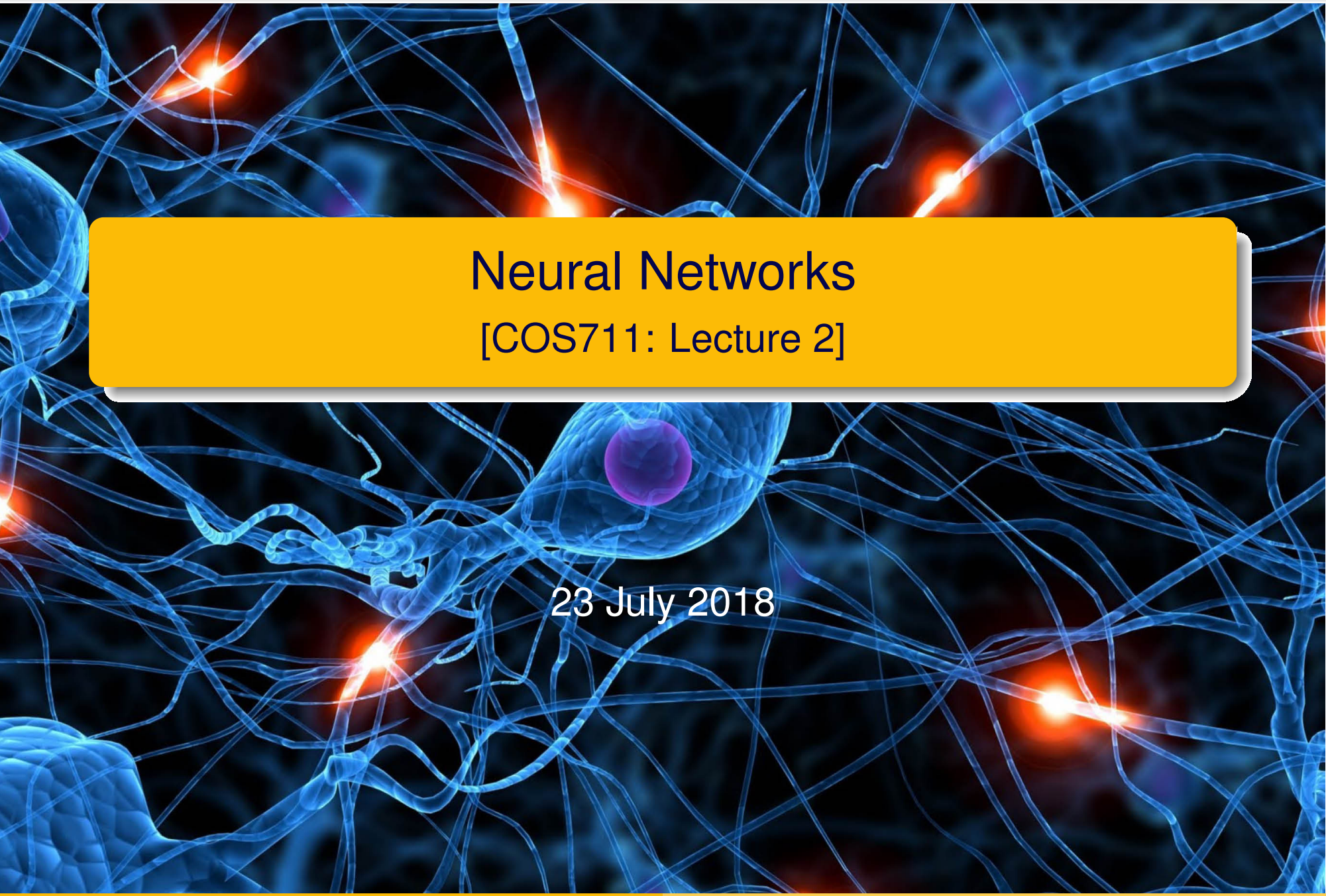


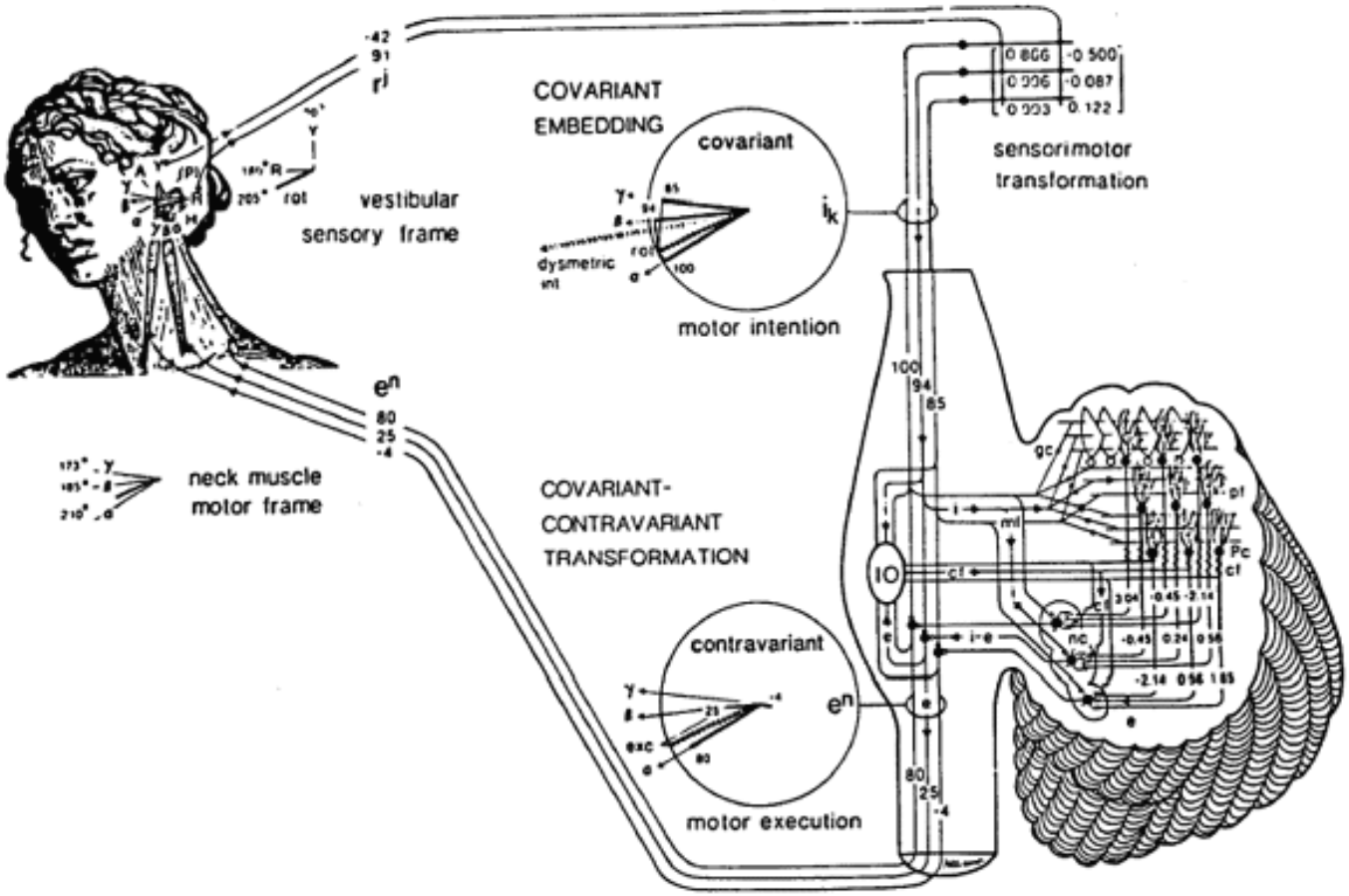
Neural Networks

[COS711: Lecture 2]

23 July 2018

What is a Neural Network?

Your brain is a real-valued circuit



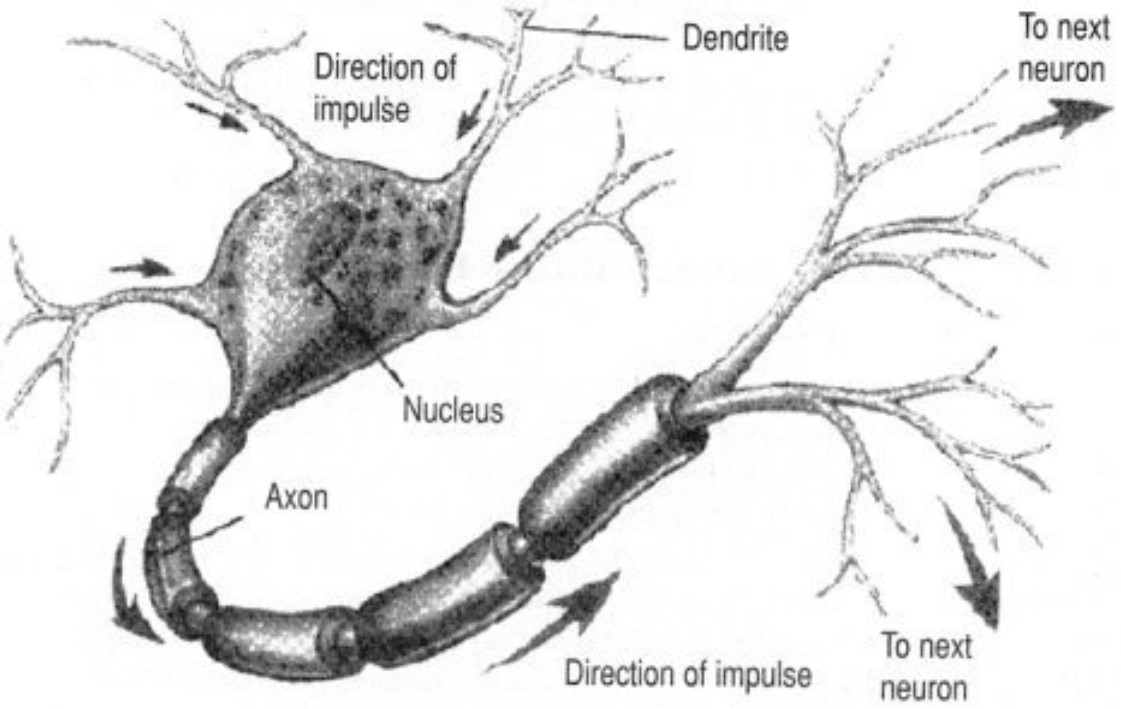
What is cognition?

All cognition is recognition

What is a Neural Network?

Neuron: A single unit of the brain matter

Figure 1: Basic illustration of a neuron



Source: Brown & Benchmark
Introductory Psychology
Electronic Image Bank (1995). Times Mirror Higher Education Group, Inc.

What is a Neural Network?

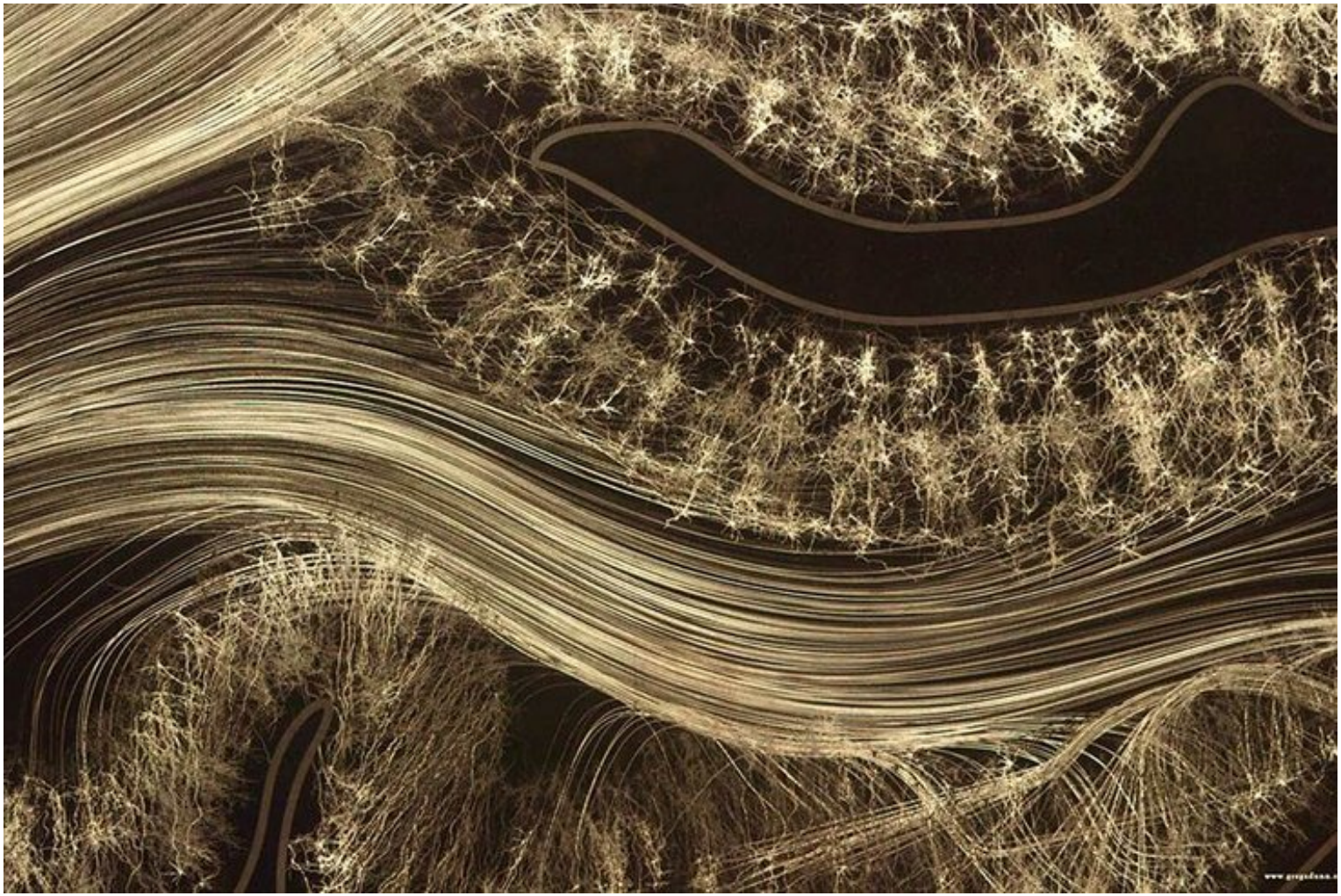
“Self Reflected” by Greg Dunn

What is a Neural Network?

“Self Reflected” by Greg Dunn

What is a Neural Network?

“Self Reflected” by Greg Dunn

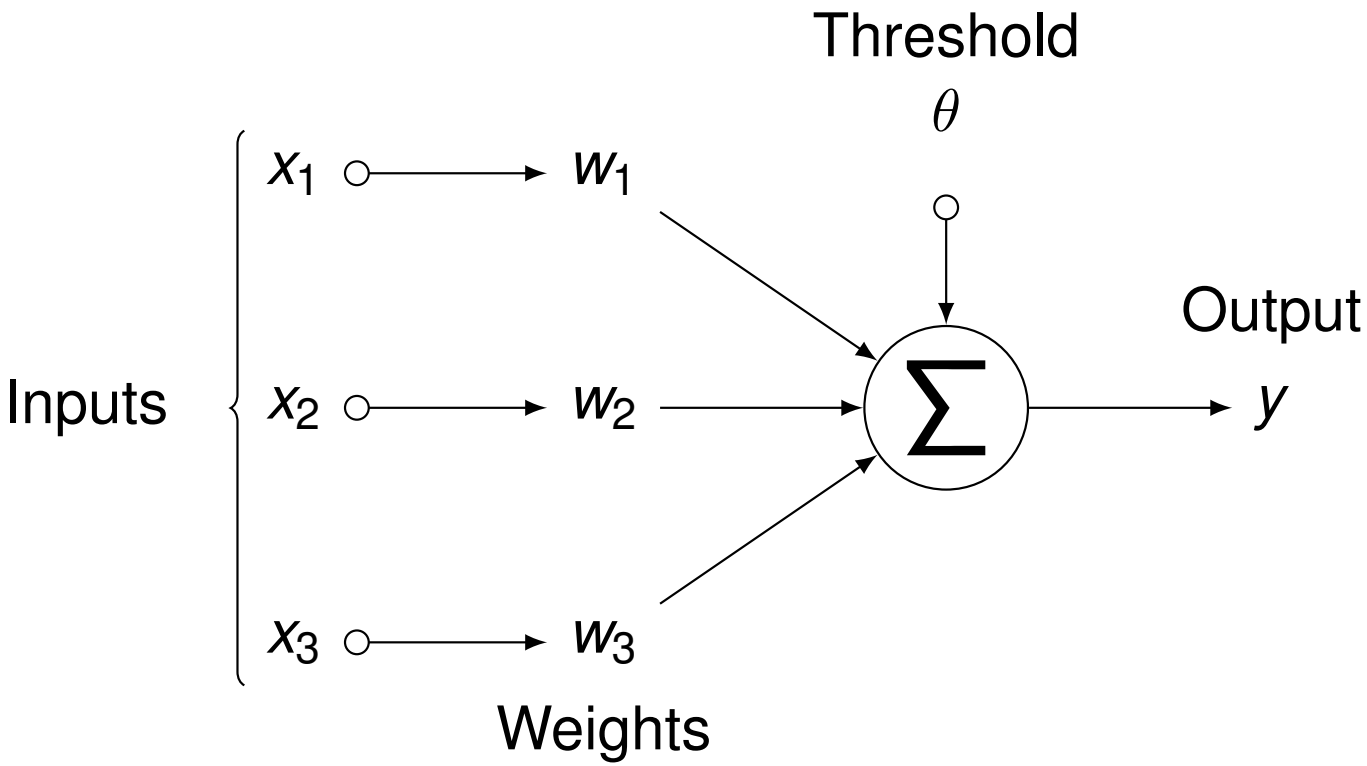


Perceptron: 1957

- First attempt at modelling neural networks was binary: Frank Rosenblatt invented *the perceptron*
- Intention: image recognition
- The New York Times reported the perceptron to be *"the embryo of an electronic computer that ... will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."*

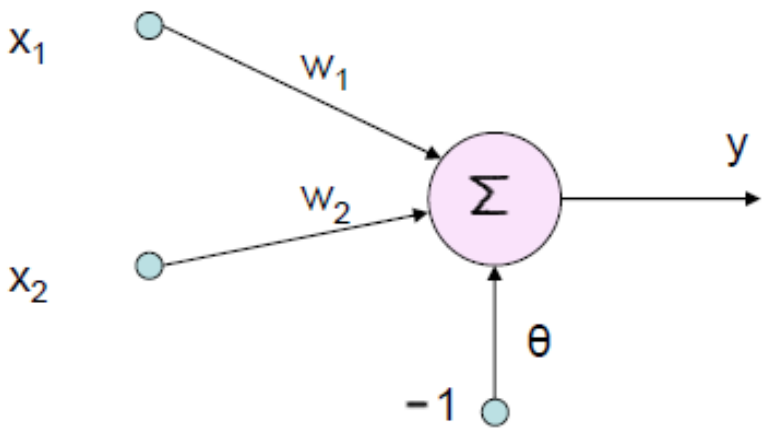
Perceptron: 1957

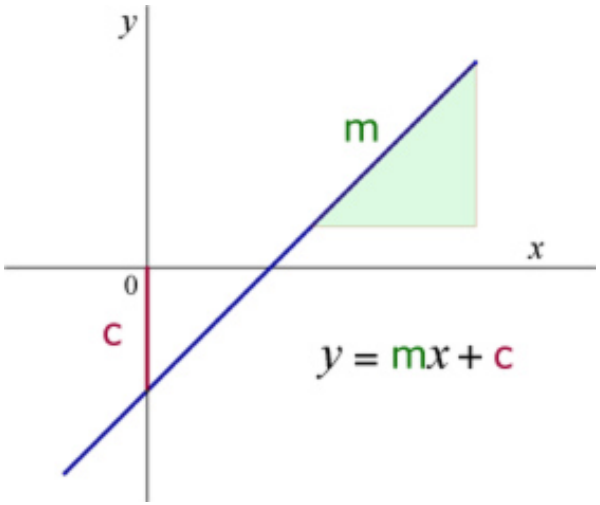
- A perceptron takes several binary inputs, $x1, x2, \ldots$, and produces a single binary output y:
  - $y = 0$ if $\sum x_i w_i \leq \theta$ (Simplify: $\sum x_i w_i - \theta \leq 0$)
  - $y = 1$ if $\sum x_i w_i > \theta$ (Simplify: $\sum x_i w_i - \theta > 0$)



How do perceptrons work?

Straight lines and flat planes

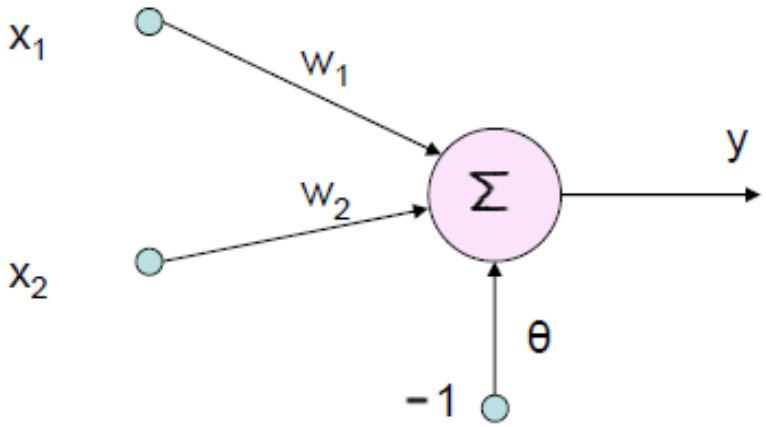




- $net = \sum x_i w_i - \theta$
- Mathematical equation of a line: $y = mx + c$
- Mathematical equation of a hyperplane: $\sum x_i a_i - b = 0$
- $\therefore$ Perceptron models a hyperplane that separates two possible classes
- Output classes must be linearly separable
- What is the purpose of θ?
- Can perceptrons classify into more than 2 classes?

How do you use perceptrons?

Let's model an OR gate!



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

What w_1, w_2, and θ values will work?

$w_1 = 1, w_2 = 1, \theta = 0.5$:

- $(0 * 1) + (0 * 1) - (1 * 0.5) = -0.5 < 0$
- $(1 * 1) + (0 * 1) - (1 * 0.5) = 0.5 > 0$
- $(1 * 1) + (1 * 1) - (1 * 0.5) = 1.5 > 0$
- Is this the only solution?

Training the Perceptron

So a perceptron can divide two classes by a hyperplane...

How can we automate the process of finding the weights?

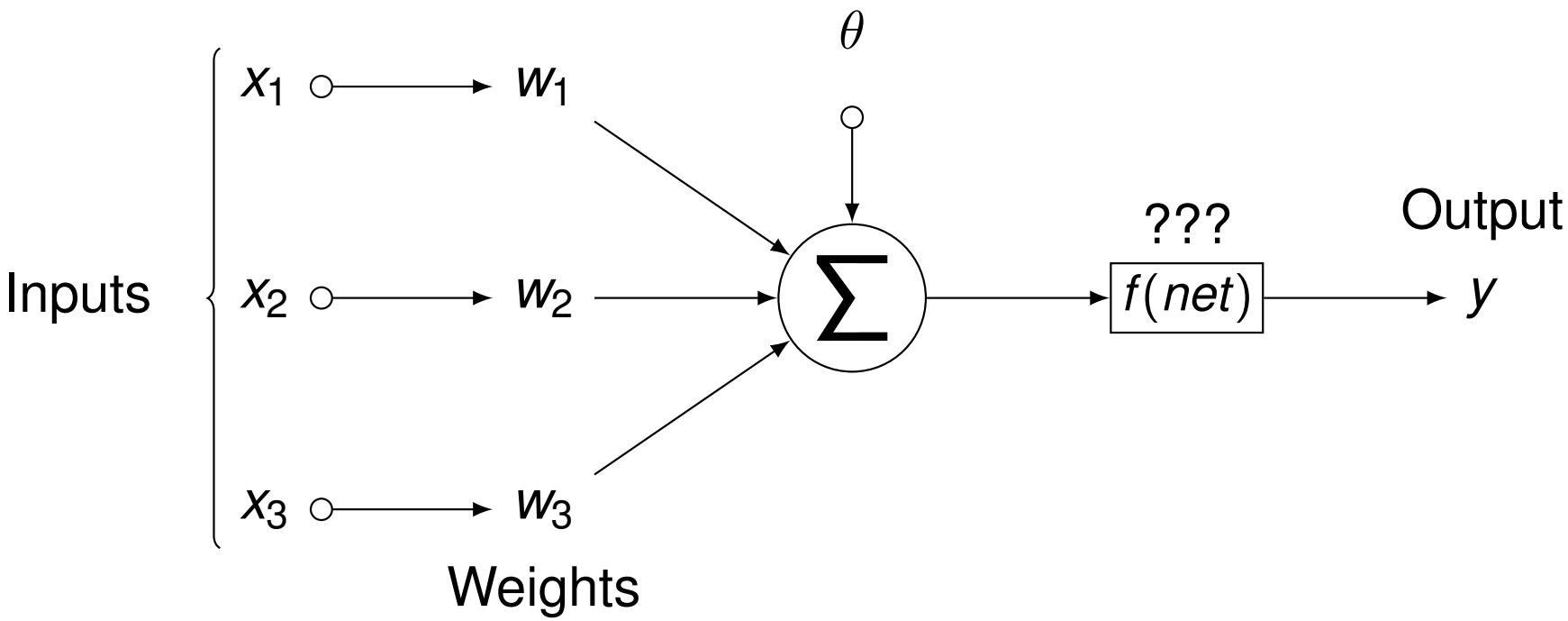
- Start with a random set of weights
- Iteratively move the hyperplane boundary (i.e. modify the weights) such that more points are correctly classified
- Stop when all input vectors are correctly classified

Simplify the equations by augmenting the weight vector

$$\begin{aligned} net &= \sum_{i=1}^{I} x_i w_i - \theta \\ &= \sum_{i=1}^{I} x_i w_i + x_{i+1} w_{i+1} \\ &= \sum_{i=1}^{I+1} x_i w_i \end{aligned}$$

Training the Perceptron

- Let us take another look at the perceptron: what is $f(net)$?

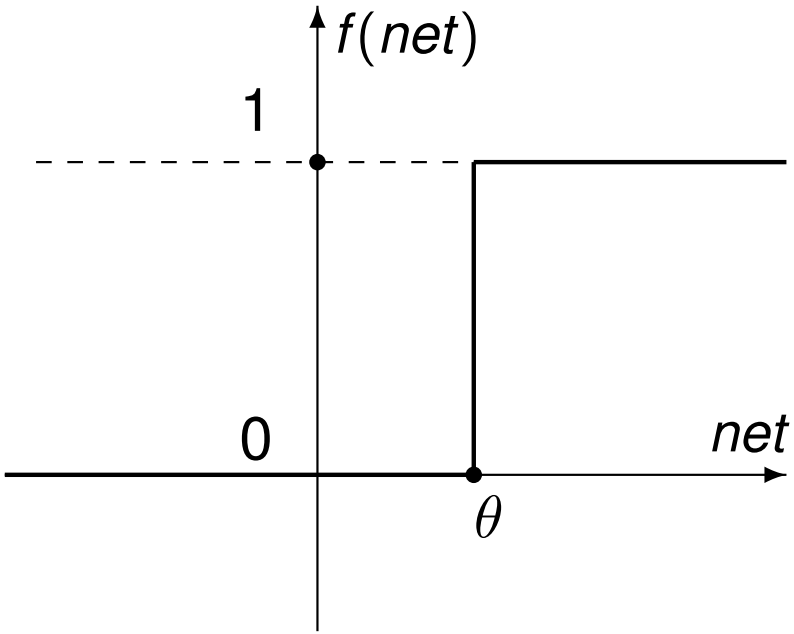


- The purpose of $f(net)$ is to respond to net: output 0 or 1 based on the incoming signals
- $f(net)$ essentially converts net to binary

Activation functions

...and why some are better than others

- $f(net)$ shown on the previous slide is known as the activation function (a.k.a squashing function).
- Perceptron uses the step activation function:

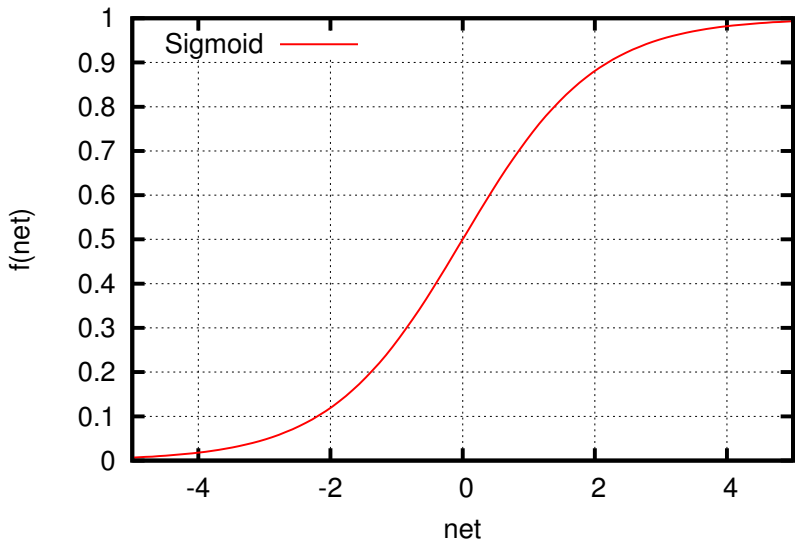


- How will the output of $f(net)$ change if we change the weights?
- Very abruptly!
- Small change in weights will have very little information for training

Activation functions

...and why some are better than others

- We can use a “smooth” version of the step function, the Sigmoid function:

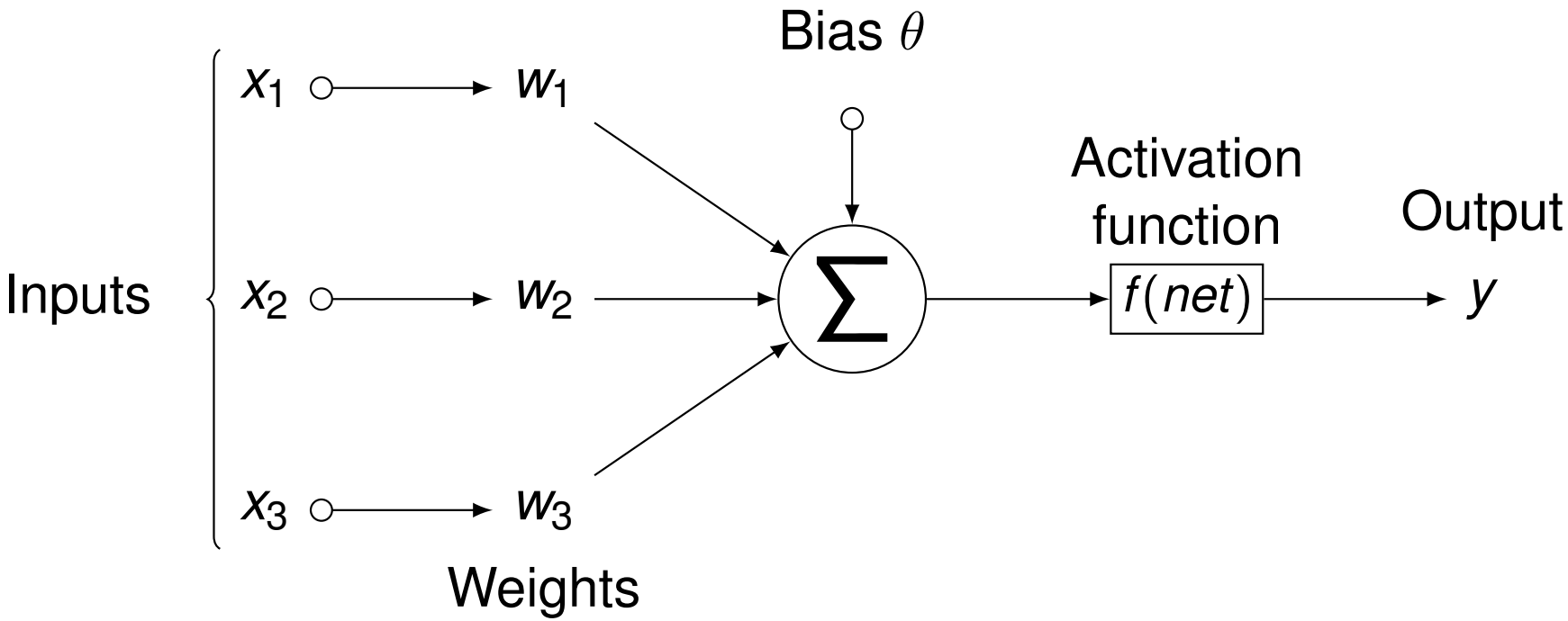


- $f(net) = \frac{1}{1 + e^{-\lambda net}}$
- λ controls the steepness of the slope, and is usually set to 1
- Now, small changes in weights and bias will cause only a small change in the output

Artificial Neuron:
A real-valued perceptron

Artificial neurons are perceptrons that use real values

- Are real values better than binary?
- Are smooth activation functions better than the threshold function?



Training the Artificial Neuron

Supervised Learning

- A data set of inputs and the corresponding targets is given, referred to as the training set
- The training set is iteratively presented to the neuron
- Supervised learning algorithm adjusts the weights at each iteration to minimize the difference between target outputs, t, and actual outputs, $y = f(net)$

$x_{1,1}$	$x_{1,2}$	$\ldots$				$t_{1,1}$	$t_{1,2}$	$\ldots$	$t_{1,m}$
$x_{2,1}$	$x_{2,2}$	$\ldots$				$t_{2,1}$	$t_{2,2}$	$\ldots$	$t_{2,m}$
$x_{3,1}$	$x_{3,2}$	$\ldots$				$t_{3,1}$	$t_{3,2}$	$\ldots$	$t_{3,m}$
$x_{4,1}$	$x_{4,2}$	$\ldots$				$t_{4,1}$	$t_{4,2}$	$\ldots$	$t_{4,m}$
$x_{5,1}$	$x_{5,2}$	$\ldots$				$t_{5,1}$	$t_{5,2}$	$\ldots$	$t_{5,m}$
$\ldots$	$\ldots$					$\ldots$	$\ldots$		$\ldots$
$x_{n,1}$	$x_{n,2}$					$t_{n,1}$	$t_{n,2}$	$\ldots$	$t_{n,m}$

Supervised Learning
Examples

Training set for the OR gate example:

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

Training set for three classes: chair, table, bed

h	*w*	*chair*	*table*	*bed*
0.5	0.3	1	0	0
0.4	0.4	1	0	0
0.9	1.2	0	1	0
0.8	1.5	0	1	0
0.4	2.0	0	0	1
0.6	1.9	0	0	1

Artificial Neuron Learning

Deriving the algorithm

- With supervised learning, we can pass the training inputs through the neuron, and compare the outputs produced by the neuron to the desired outputs
- We want the neuron to classify the inputs correctly
- Essentially, we want to minimize the neuron's error

Sum of squared errors (SSE)

$$E = \sum_{i=1}^{P} (t_i - o_i)^2$$

Where:

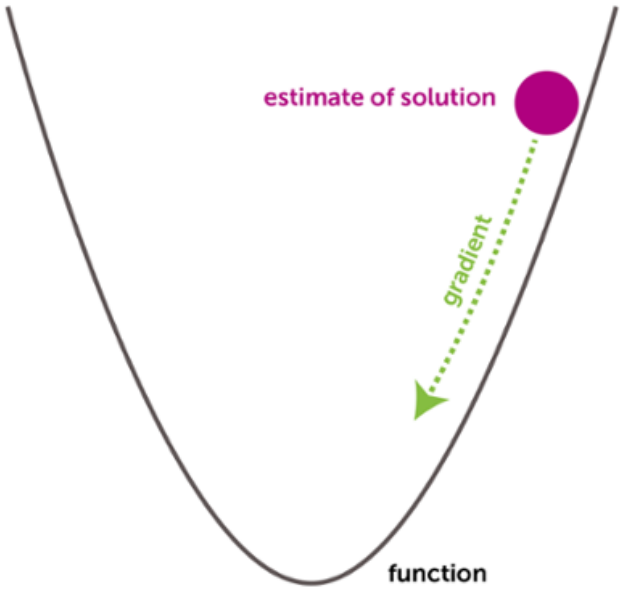
- P is the number of patterns in the training set
- t is the target output of example i
- o is the output obtained by the neuron for example i

Artificial Neuron Learning

Gradient descent

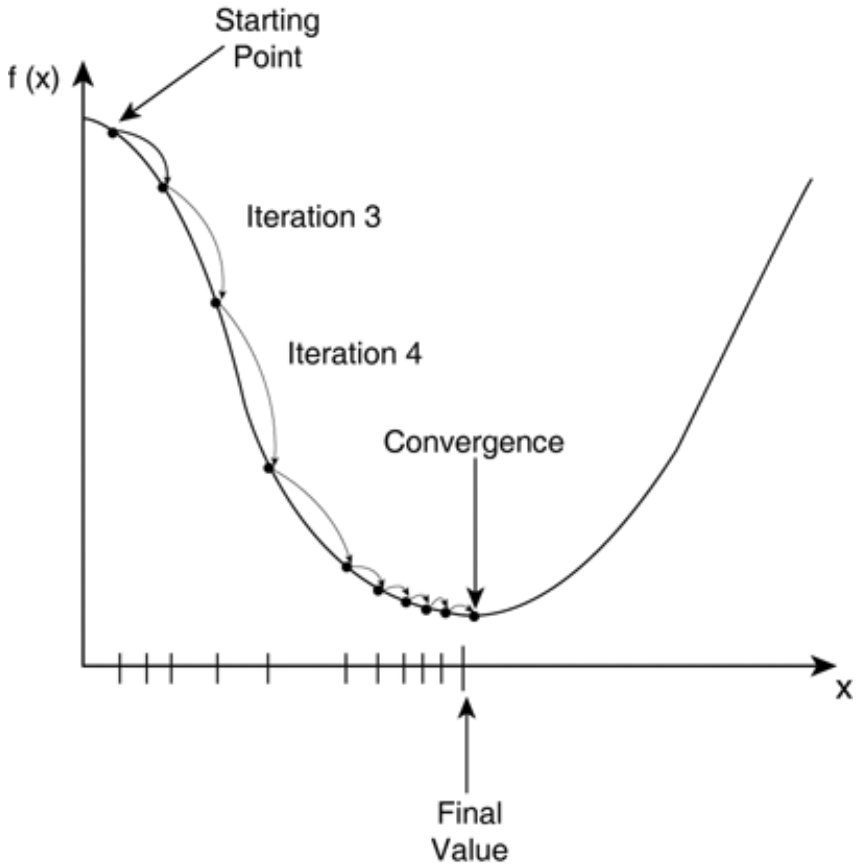
- We want to minimize the neuron's error - in other words, find the minimum of *SSE* with respect to the neuron's weights
- Function minimization algorithm can be applied

- Gradient descent:
  - Calculate the gradient of *SSE* in the weight space
  - Move the weight vector along the negative gradient
  - "Steepest slope descent"



Artificial Neuron Learning

Deriving the algorithm



Gradient Descent Iteration

- $w = w + \Delta w$
- $\Delta w = \eta(-\frac{\delta E}{\delta w})$
- $\frac{\delta E}{\delta w} = -2(t_i - o_i)\frac{\delta f(net)}{\delta net_i}x_j$

Where:

- η is the learning rate
- t is the target of example i
- o is the output of example i

Artificial Neuron Learning

Widrow-Hoff: 1987

- The original perceptron used the step function for $f(net)$
- We can't differentiate discontinuous $f(net)$!
- Assume $f(net) = net$ (linear function). Then:

Widrow-Hoff Learning Rule

- $\frac{\delta f(net)}{\delta net_i} = 1$
- $\frac{\delta E}{\delta w} = -2(t_i - o_i)\frac{\delta f(net)}{\delta net_i}x_j = -2(t_i - o_i)x_j$
- $\therefore w = w + \Delta w = w + \eta(-\frac{\delta E}{\delta w}) = w + 2\eta(t_i - o_i)x_j$
- Can be used to train Rosenblatt's perceptron!

Perceptron's Learning Rule: Error Correction

How exactly did Rosenblatt do it in 1956?

- Inputs, targets and $f(net)$ are binary, weights and learning rate are real-valued

Perceptron Learning Rule

If $t - o \neq 0$ (i.e., when incorrect output is obtained):

$$w_i = w_i + \Delta w_i$$
$$\Delta w_i = \eta(t_i - o_i)x_j$$

- If $t_i = 0$ and $o_i = 1$, then $\Delta w_i < 0$, and w_i decreases
  - We need to make o_i smaller
- If $t_i = 1$ and $o_i = 0$, then $\Delta w_i > 0$, and w_i grows
  - We need to make o_i larger

`https://en.wikipedia.org/wiki/Perceptron#Example`

Artificial Neuron Learning

Generalized Delta Learning Rule

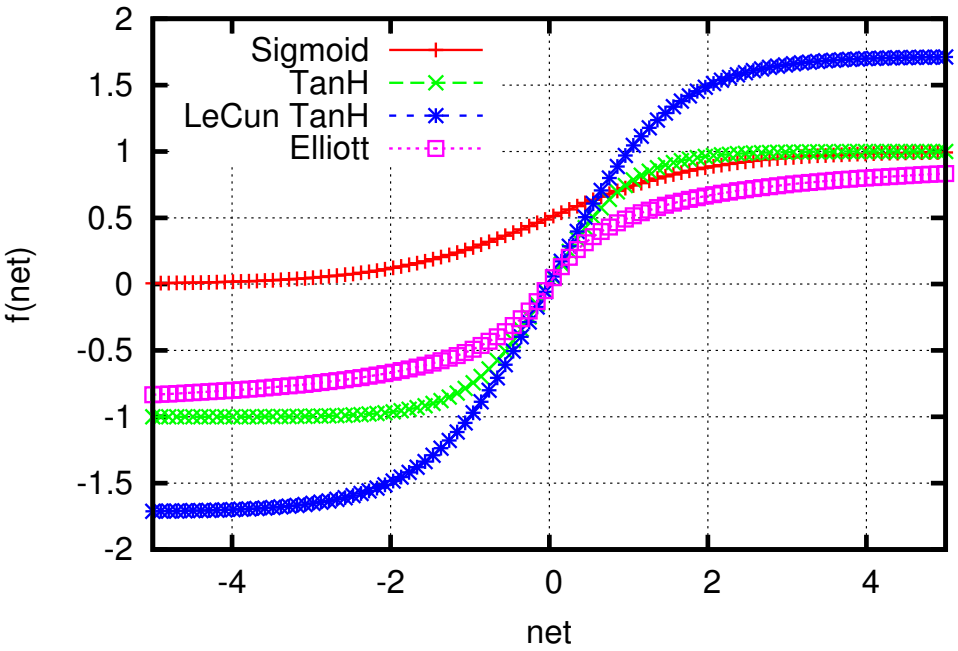
- Generalized version of Widrow-Hoff: instead of assuming $f(net) = net$, assume that $f(net)$ is differentiable
- For Sigmoid $f(net) = \frac{1}{1 + e^{-\lambda net}}$:

Generalized Delta Rule applied to Sigmoid

$$\begin{aligned}
\frac{\delta f(net_i)}{\delta net_i} &= f(net_i)(1 - f(net_i)) \\
&= o_i(1 - o_i) \\
\frac{\delta E}{\delta w} &= -2(t_i - o_i)\frac{\delta f(net_i)}{\delta net_i}x_j \\
&= -2(t_i - o_i)o_i(1 - o_i)x_j \\
\therefore w &= w + \Delta w = w + 2\eta(t_i - o_i)o_i(1 - o_i)x_j
\end{aligned}$$

Artificial Neuron Learning

What other activation functions are there?



- Hyperbolic tangent (tanh):

$$f(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}}$$

- Yann LeCun tanh:

$$f(net) = 1.7159 \tanh\left(\frac{2}{3} net\right)$$

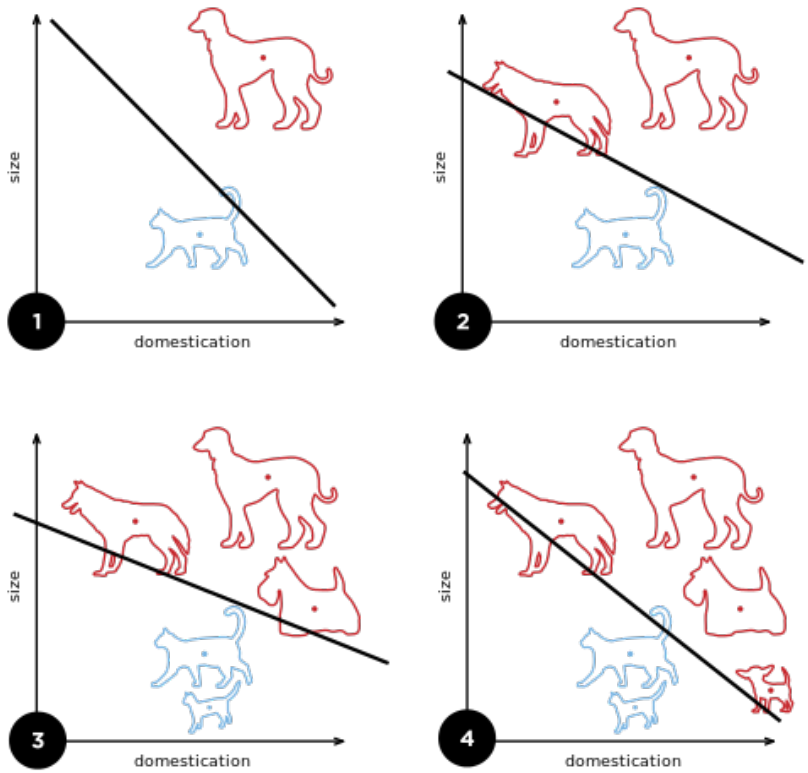
- Elliott:

$$f(net) = \frac{net}{1 + |net|}$$

- Differentiate & substitute into the delta rule!

Artificial Neuron Learning

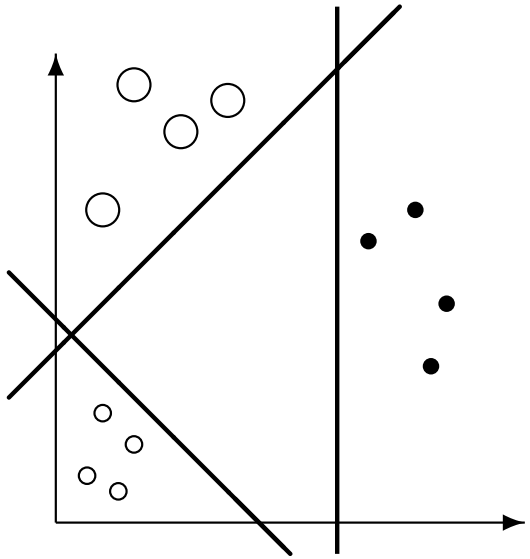
What happens when a neuron learns?

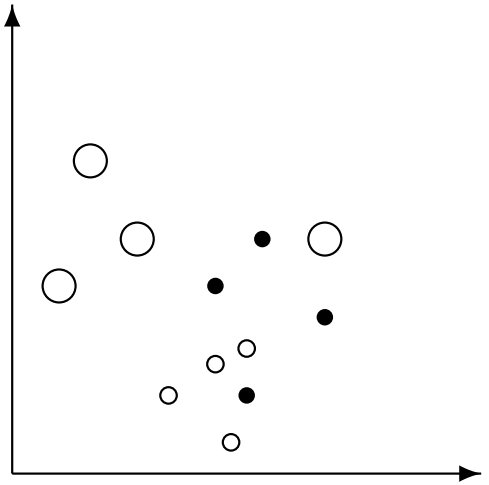


- Change in weights means that the decision boundary represented by the neuron moves
- Every weight update tries to divide the classes more accurately
- It is important to have enough data in the training set

Perceptron limitations

Linear separability





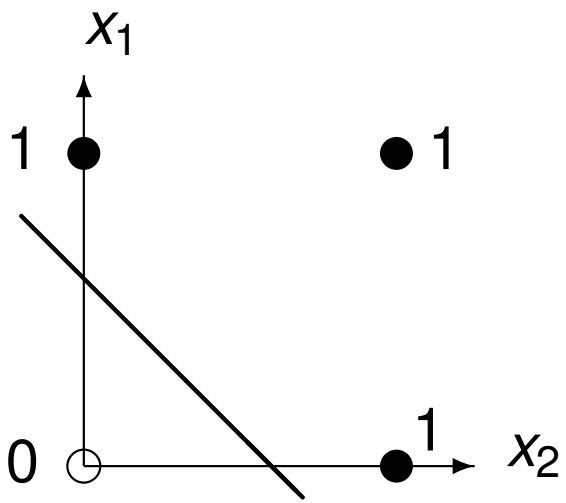
Perceptron Limitations

Can we use straight lines to separate the classes on the right?

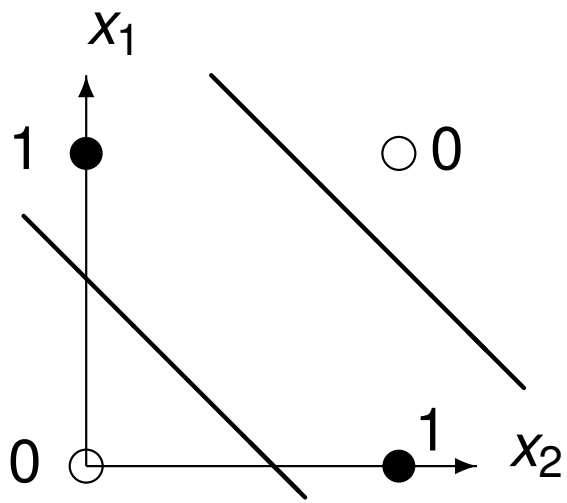
Perceptron Limitations

Linear separability

OR gate



XOR gate

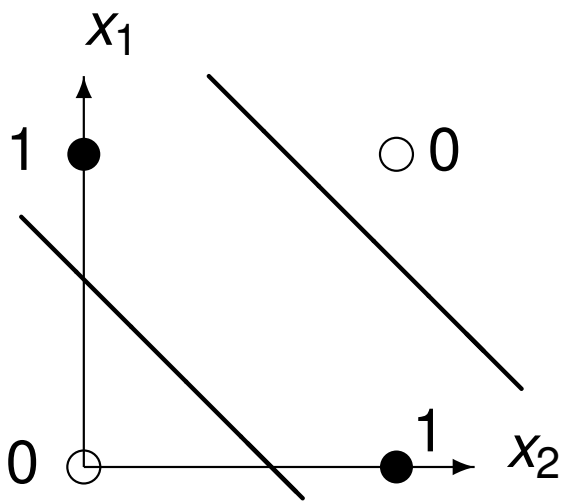


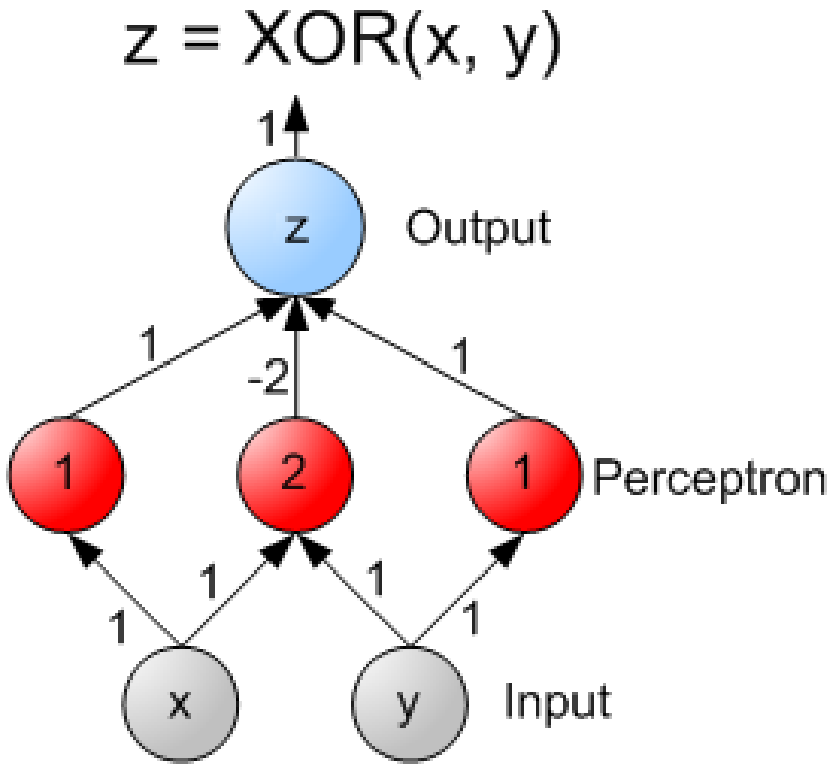
Perceptron Limitations

Perceptron can only model linearly separable functions. This caused a **20 year** hiatus in neural network research.

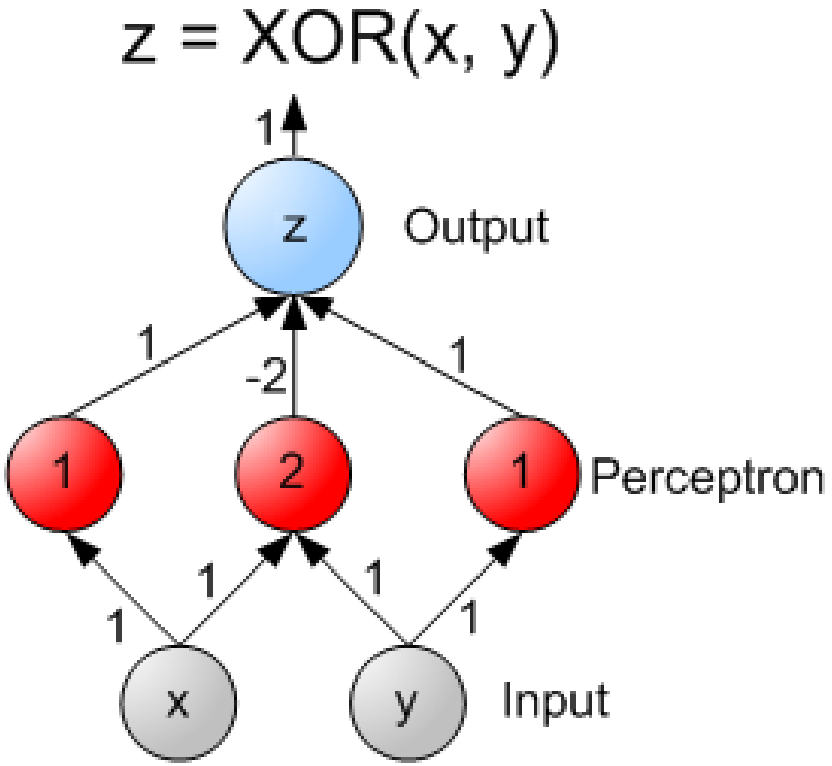
Neural Networks
Universal approximators

So the perceptron failed at modeling the XOR gate...





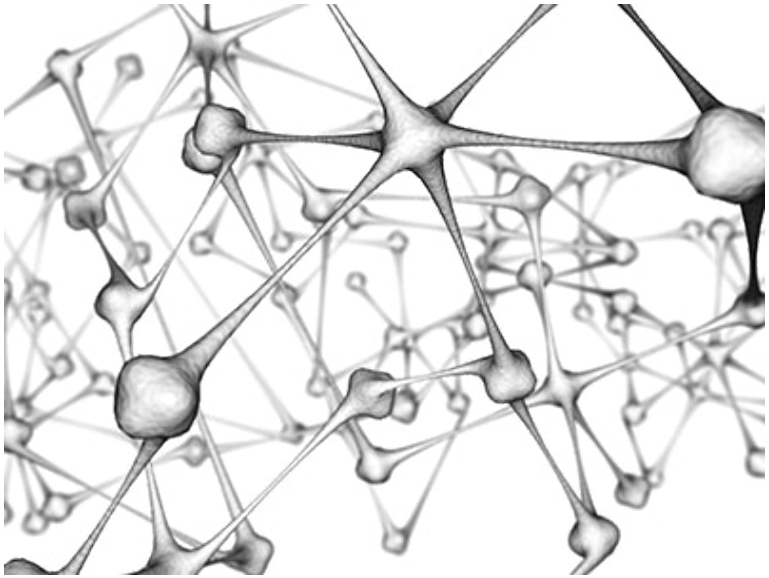
Neural Networks
Universal approximators

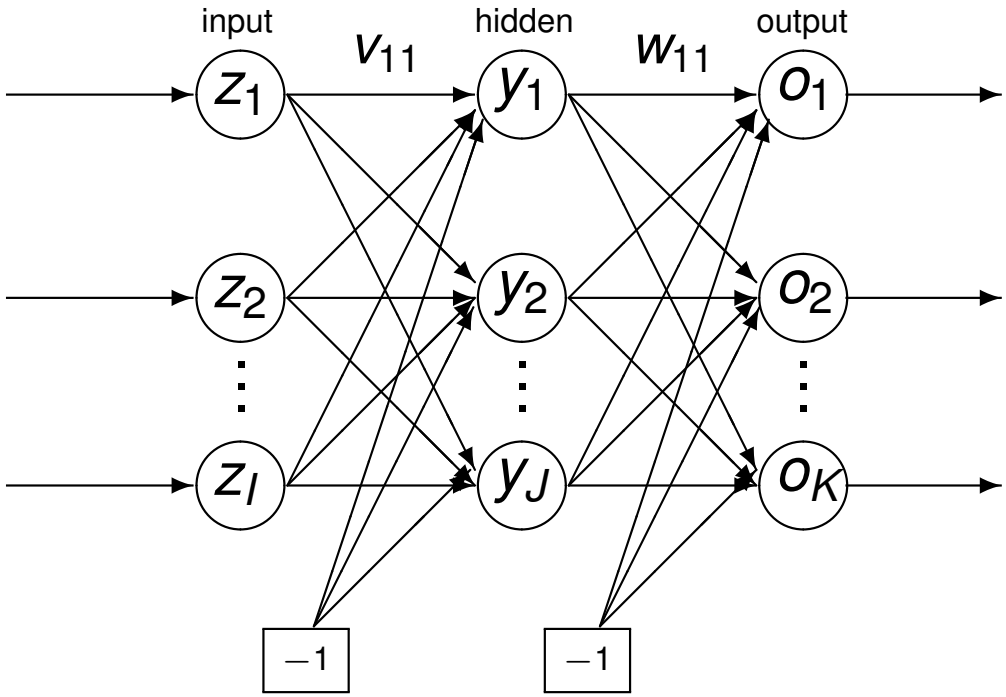


- Linking perceptrons in successive layers introduces non-linearity to the model through the activation functions
- It can be proved mathematically that a NN with enough hidden neurons can approximate **any** non-linear function
- NNs are “universal approximators”
- NB: This statement only holds true if activation functions are **non-linear**

Feed Forward Neural Network

a.k.a. FFNN





A real NN and a FFNN with a single hidden layer

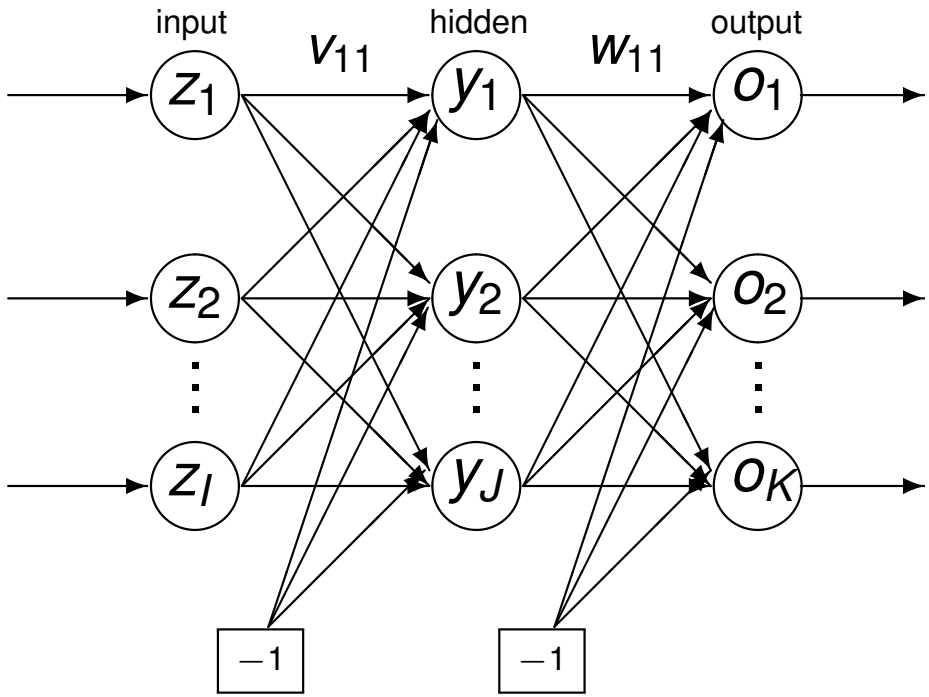
Feed Forward Neural Network

Feed Forward Pass

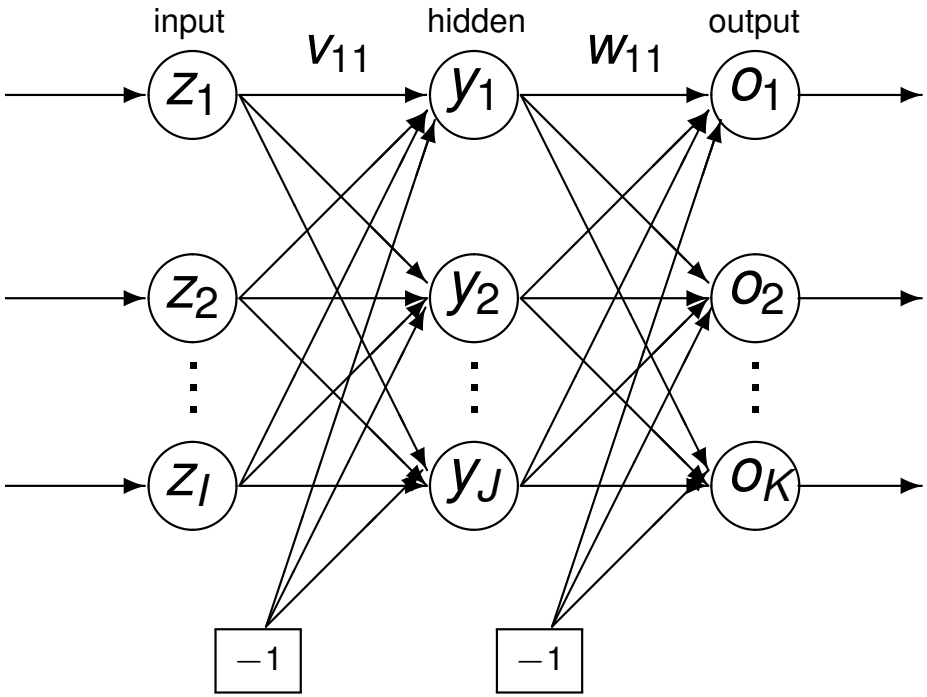
$$o_k = f_{o_k}(net_{o_k})$$
$$= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}(net_{y_j})\right)$$
$$= f_{o_k}\left(\sum_{j=1}^{J+1} w_{kj} f_{y_j}\left(\sum_{i=1}^{I+1} v_{ji} z_i\right)\right)$$

Where:

- f_{o_k}, f_{y_j} are activation functions of the corresponding neurons



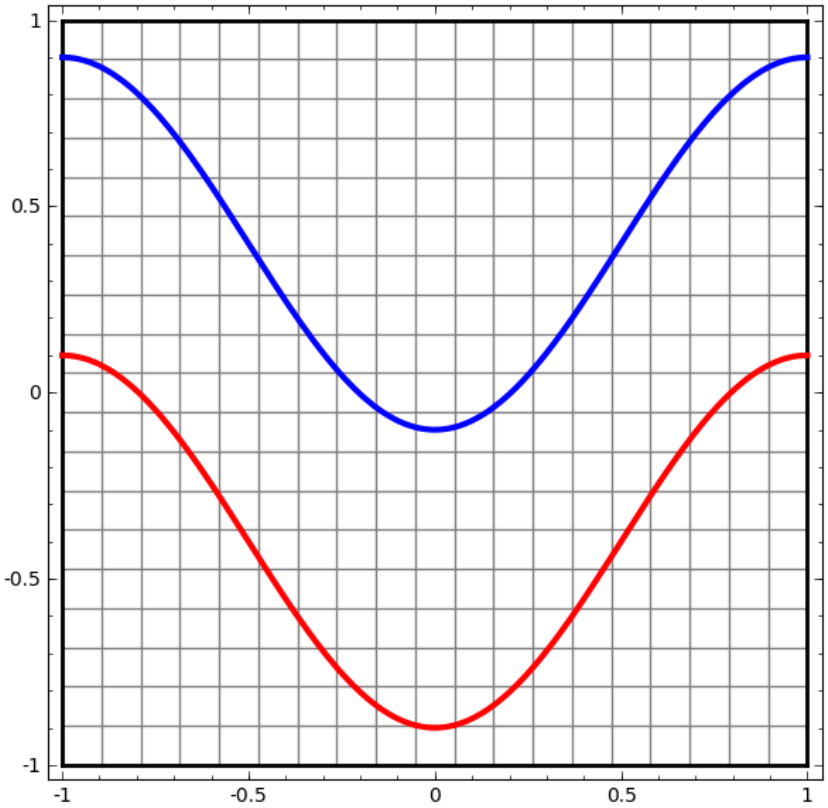
Feed Forward Neural Network



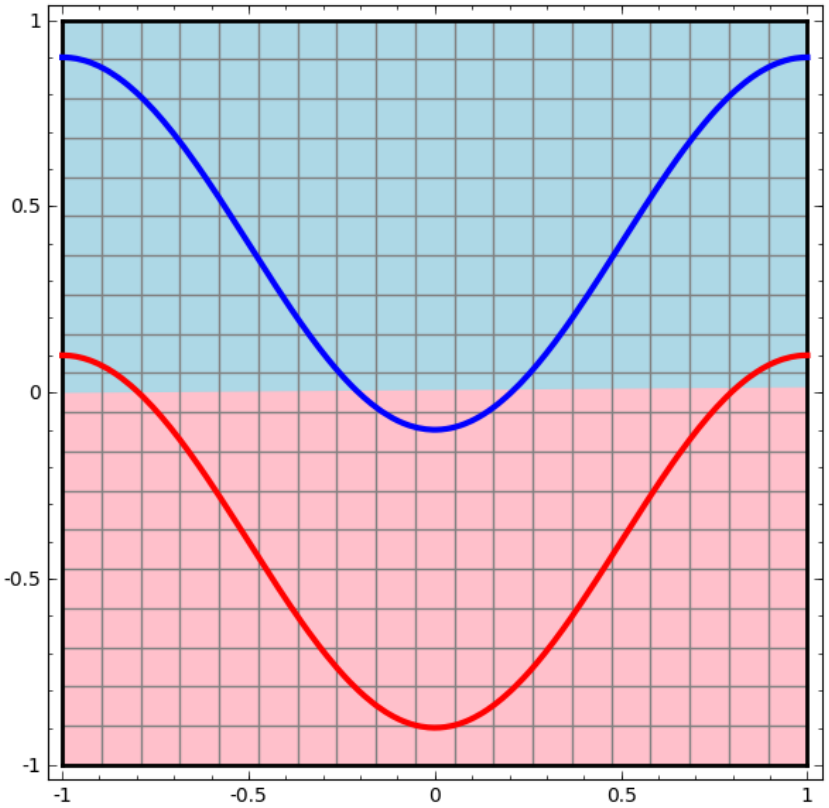
- A neural network with enough hidden neurons can approximate any non-linear function
- How do we choose the number of hidden neurons?

Feed Forward Neural Network

Intuition



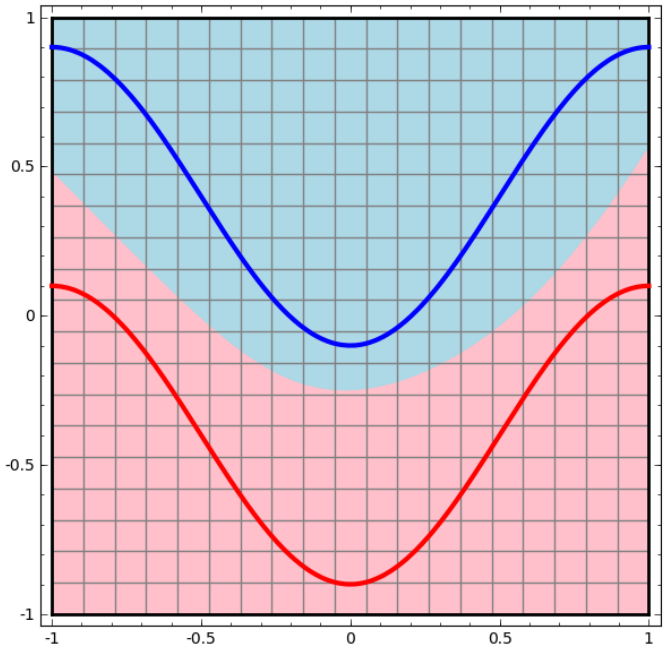
Original Data



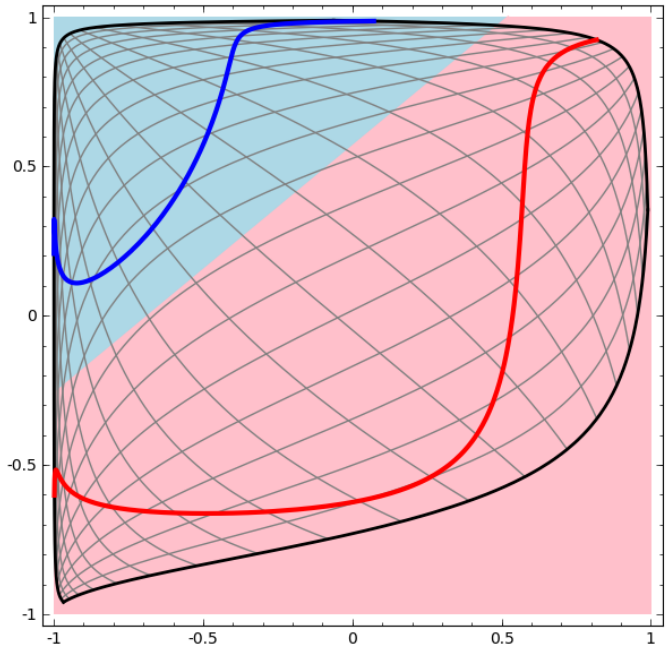
Single Perceptron Attempt

Feed Forward Neural Network

Intuition



Desired Boundary



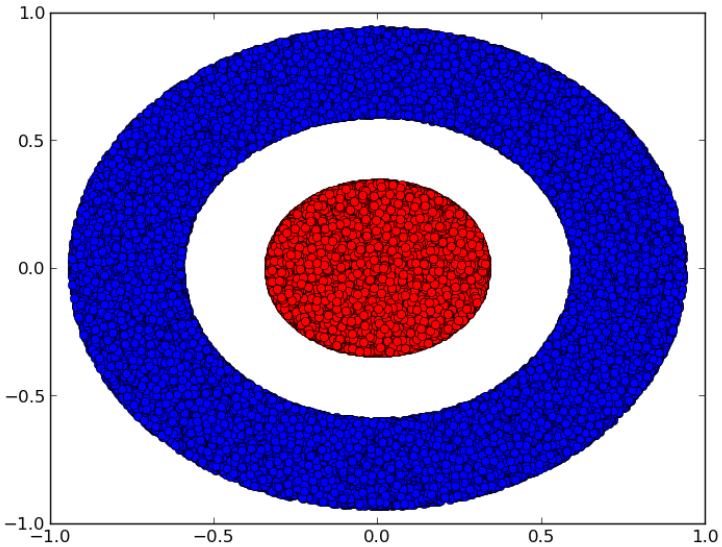
What Actually Happens

Multi-layered perceptrons

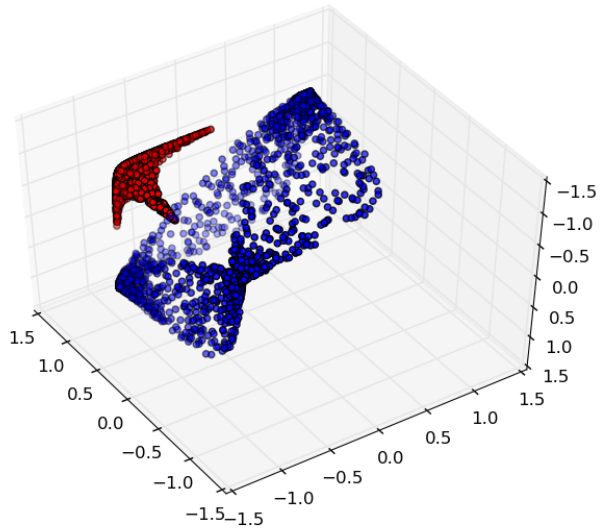
Each hidden neuron (perceptron) warps the space such that the data becomes linearly separable for the successive layers

Feed Forward Neural Network

Intuition



2 neurons: 2 dimensions

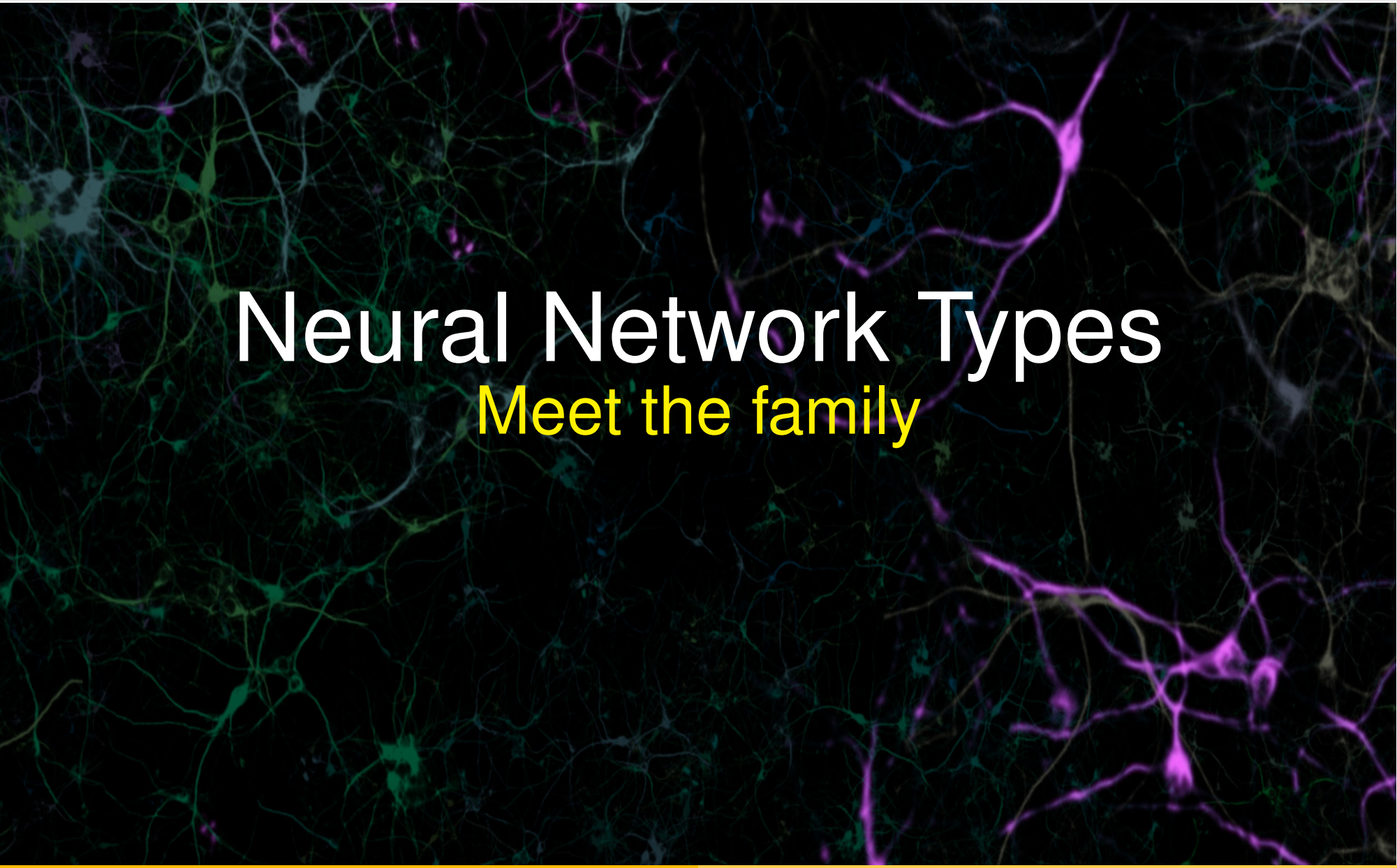


3 neurons: 3 dimensions

Multi-layered perceptrons

Each hidden neuron adds a “dimension” to the warped space, making it easier to transform data to be linearly separable

Neural Networks

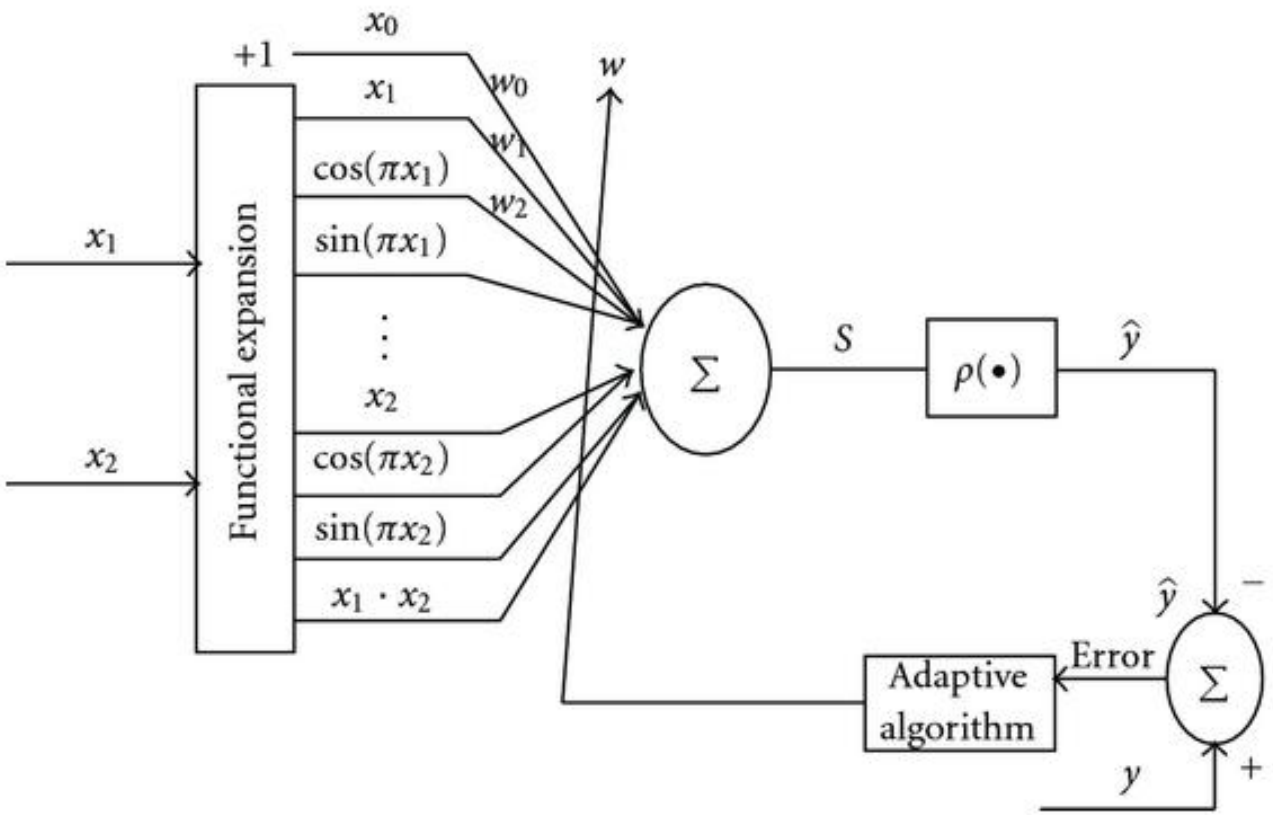


Neural Network Types

Meet the family

Functional Link Neural Network

Increasing representation complexity

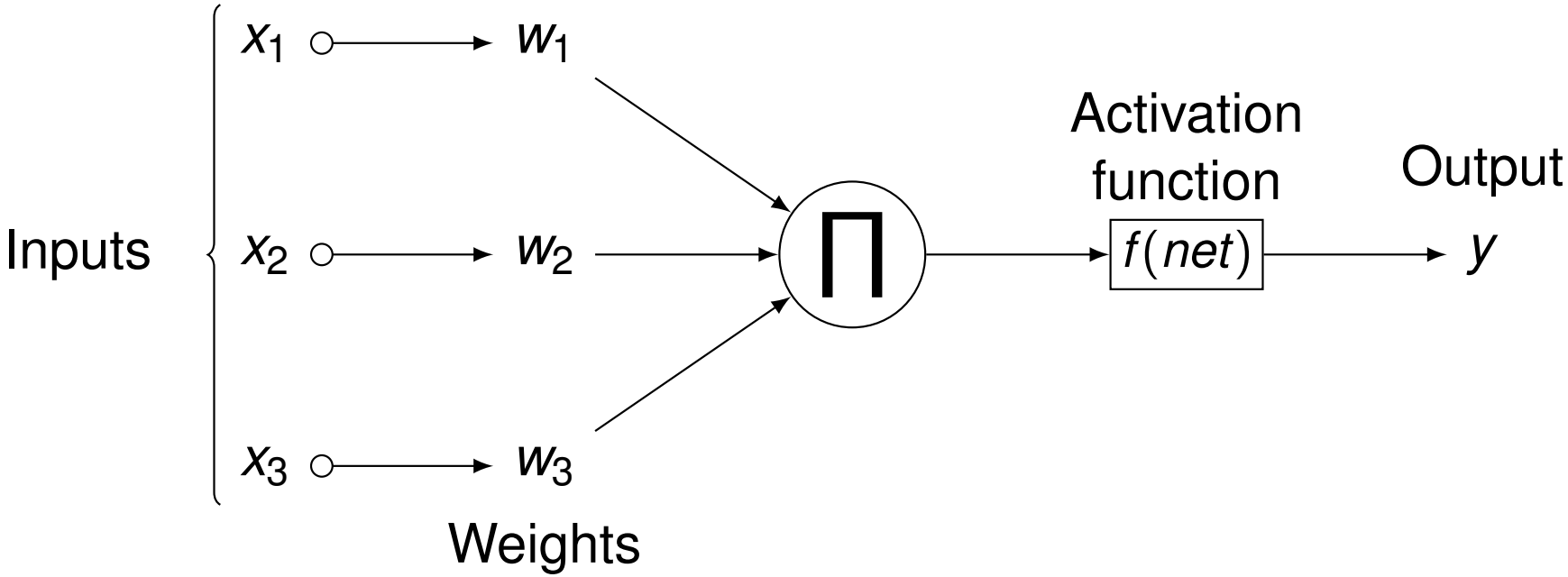


Functional Link Hidden Neuron

In FLNN, input layer is expanded into functional higher-order units, thus adding extra non-linearity to the representation.

Product Unit Neural Networks

Increasing representation complexity

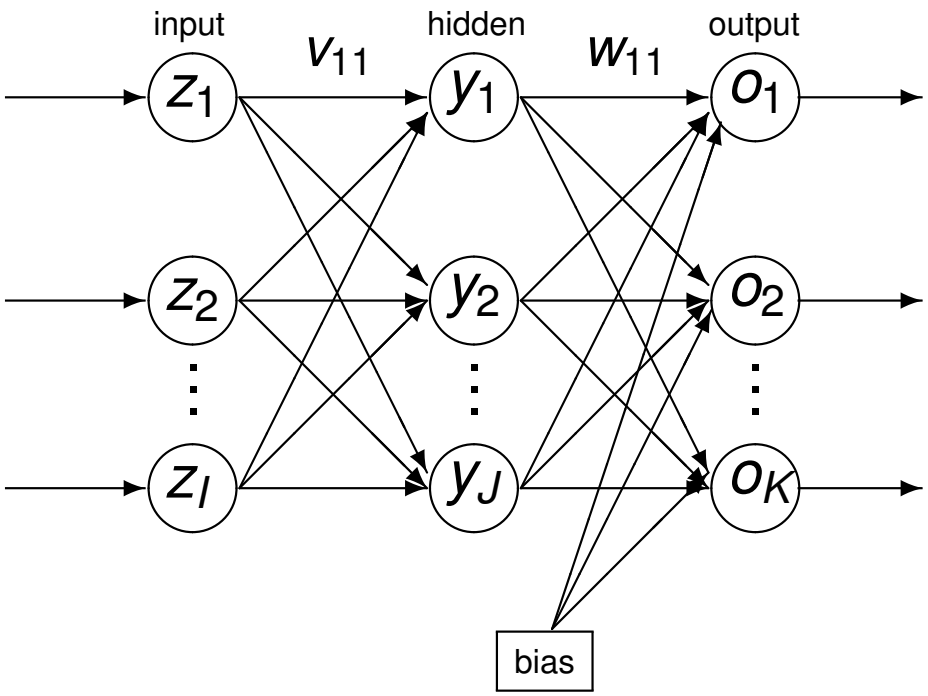


What if we use product instead of summation?

- $net = \prod_{i=1}^{I} x_i^{w_i}$
- Weights are used as powers of the inputs: we are fitting a polynomial
- Why is there no bias/threshold?

Product Unit Neural Network

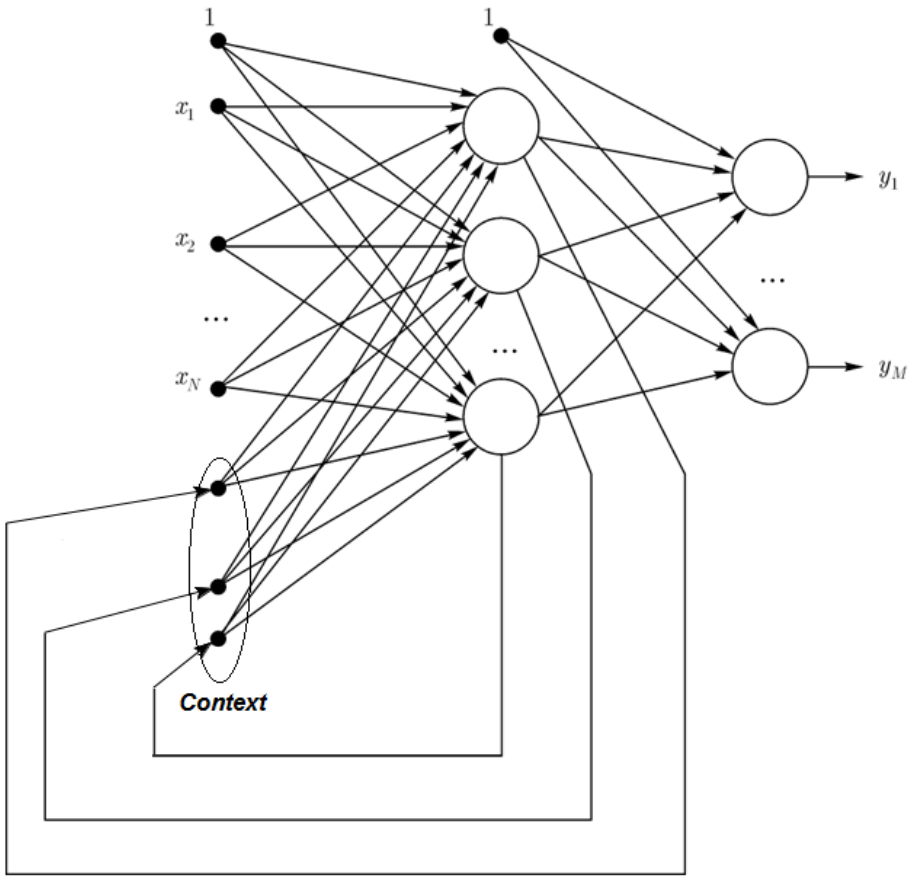
Pi-Sigma NNs



- Linear $f(net)$ is used in the hidden neurons
- Output neurons are normal summation neurons
- Now we can fit $ax^n + c$ with a 1-1-1 architecture!
- However... will this fit always adhere to the Occam's razor principle?

Simple Recurrent Neural Networks

Elman RNN

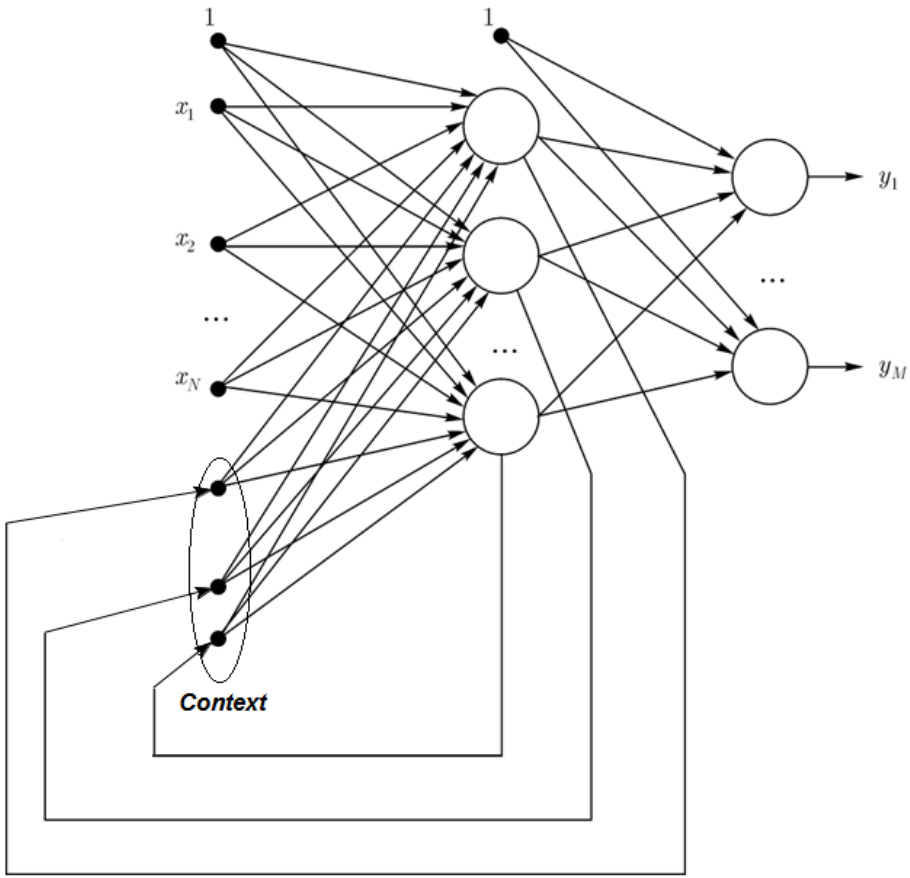


Elman Simple RNN

- In RNNs, feedback connections are added
- Using intermediate/output results as additional inputs allows to learn temporal characteristics of the data
- The current pattern is fed through the RNN with the previous pattern “in mind”
- Previous step may have information about the next step
- Implies sequential data, and an underlying relationship between previous/next step

Simple Recurrent Neural Networks

Elman RNN

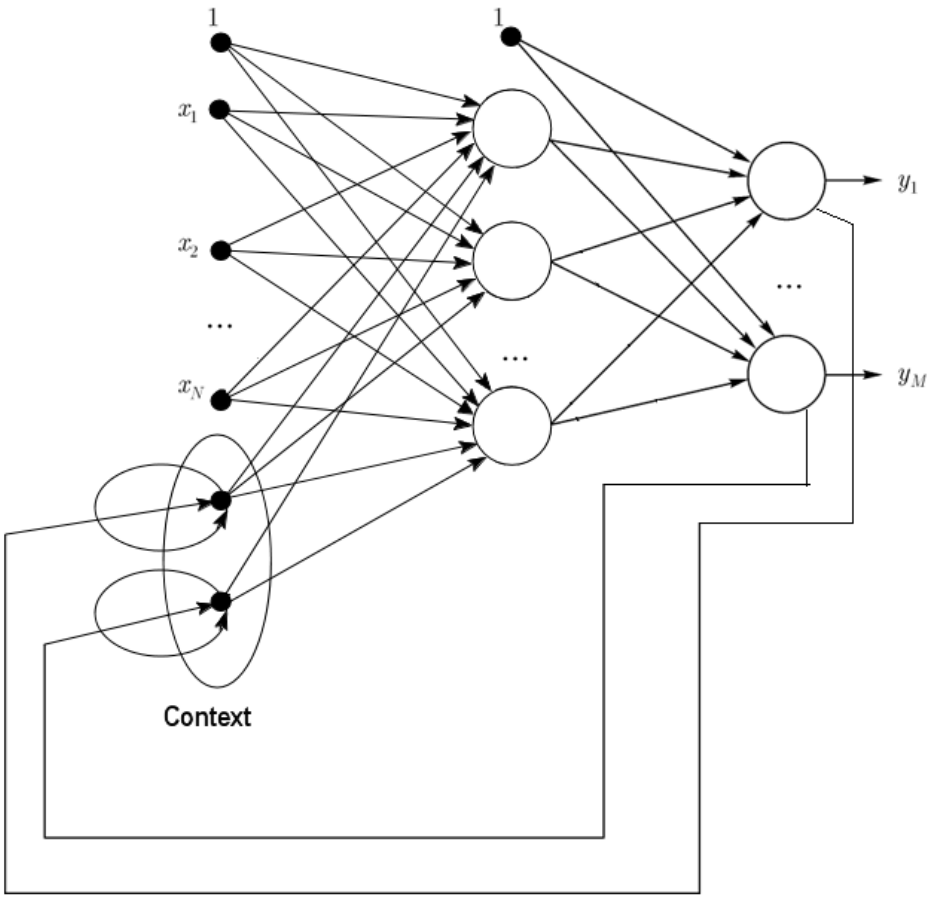


Elman Simple RNN

- Elman RNN creates a "context" layer
- Context layer stores the hidden layer outputs as obtained at previous iteration
- Context inputs are treated as normal inputs
- Output units are oblivious to recurrency
- Weights between context layer/hidden layer may be set to 1, or learned using a training algorithm

Simple Recurrent Neural Networks

Jordan RNN



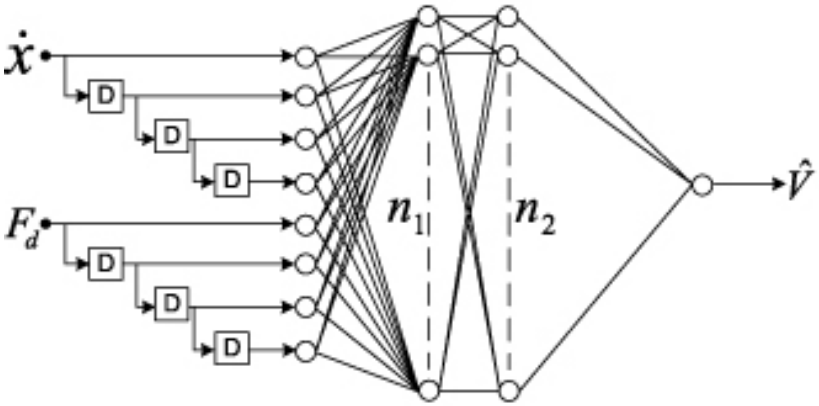
Jordan Simple RNN

- Jordan RNN stores output signals instead of the hidden signals in the context layer
- Context layer is referred to as a “state” layer
- State layer implies a relationship between previous output and next output
- Elman RNN learns the internal relationship between patterns
- Jordan RNN learns the external relationship between patterns

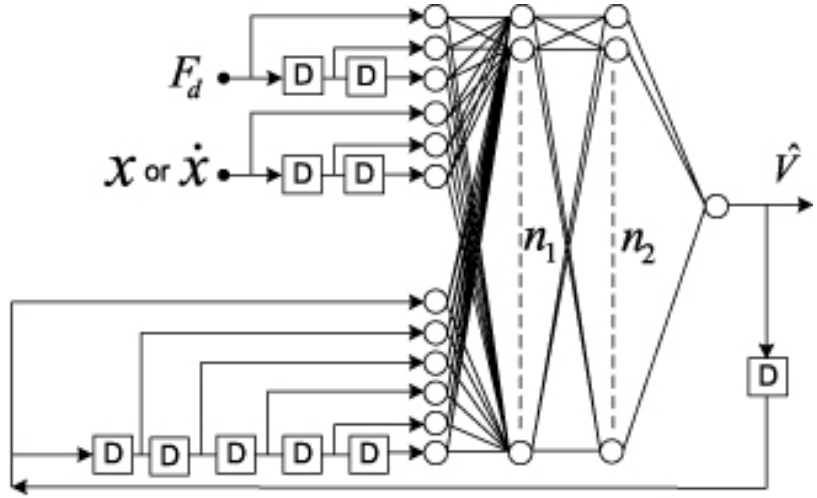
Time Delay Neural Networks

Feedforward and recurrent

- Input layer in time-delayed NNs is constructed from input $x_i(t)$ to $x_i(t + D)$, where D is the chosen delay.
- In other words, D input patterns are fed through the network at the same time
- Applicable to sequential data only (for example, speech recognition)



(a)



(b)

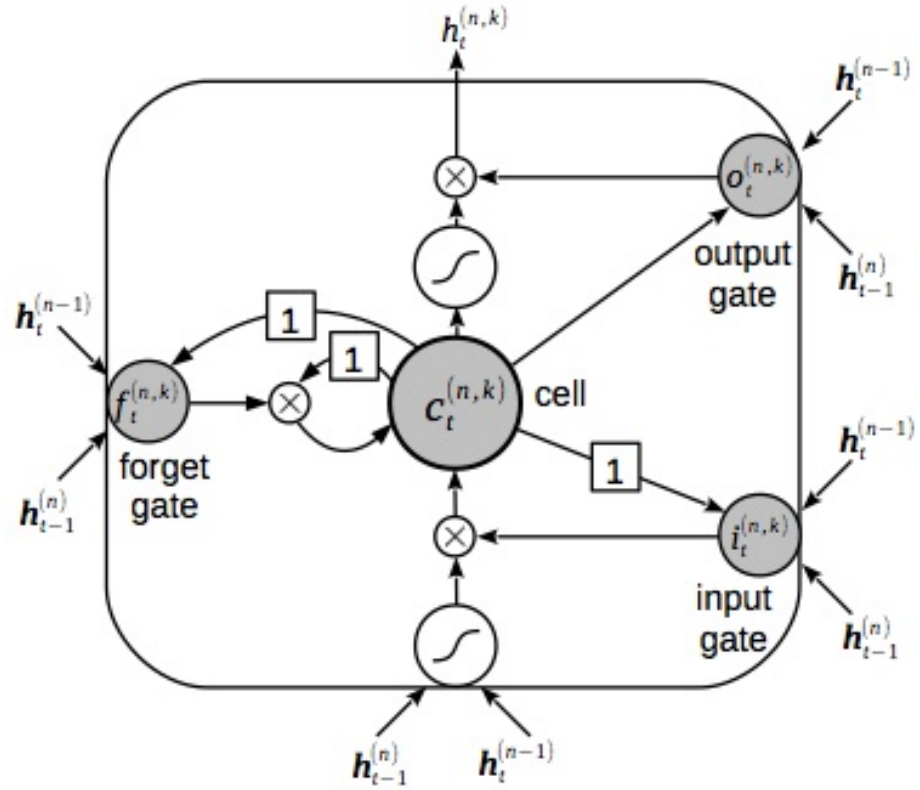
Recurrent Neural Networks

Do we model memory?

- RNNs rely on the existence of sequential relationships in data
- What if this relationship is more complex than we imagine?
- If a relationship is uncovered in the first 100 patterns, will it still be “remembered” by the RNN after another 500 patterns?
- Probably not
- As a result, RNNs fell out of fashion...
- ...until Juergen Schmidhuber came along

Long Short Term Memory (LSTM) NNs

Goal: Preservation of knowledge

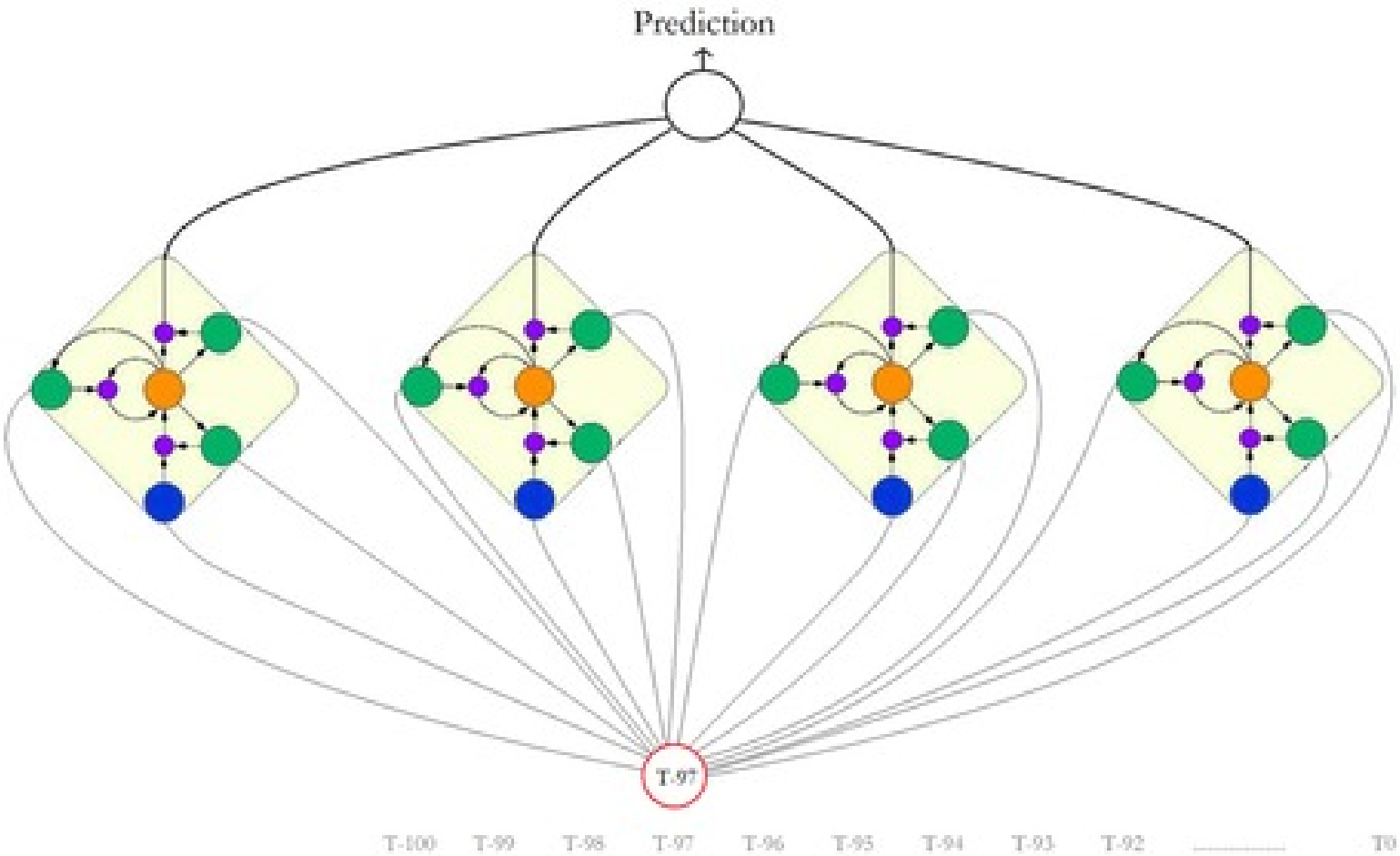


Single Block of Long Short Term Memory

- At the bottom: normal data inflow
- Input gate: when it outputs a value close to zero, it “zeros” (blocks) the input.
- Output gate: determines when the unit should output the value in its memory.
- Forget gate: when it outputs a value close to zero, the LSTM block will forget whatever value it was remembering.
- Self-recurrent connection of the LSTM block keeps the cell’s value for as long as necessary

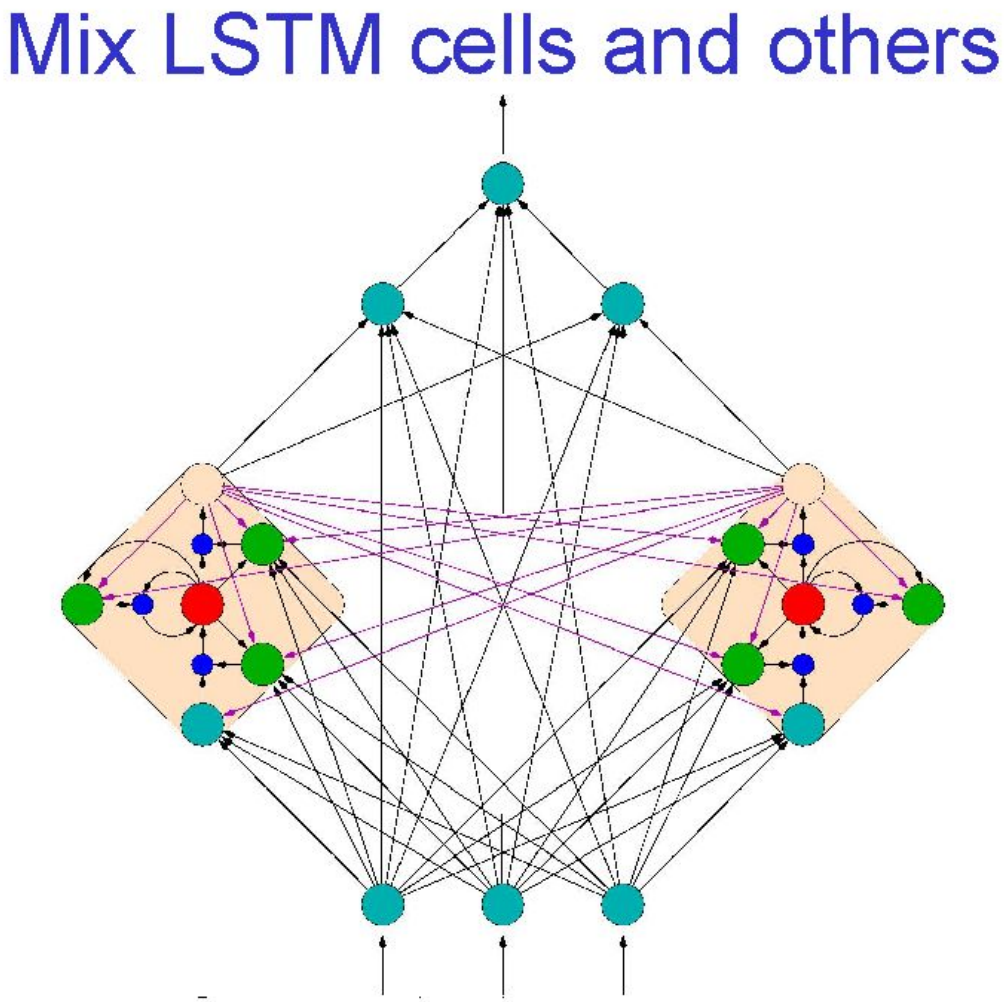
Long Short Term Memory (LSTM) NNs

Goal: Preservation of knowledge



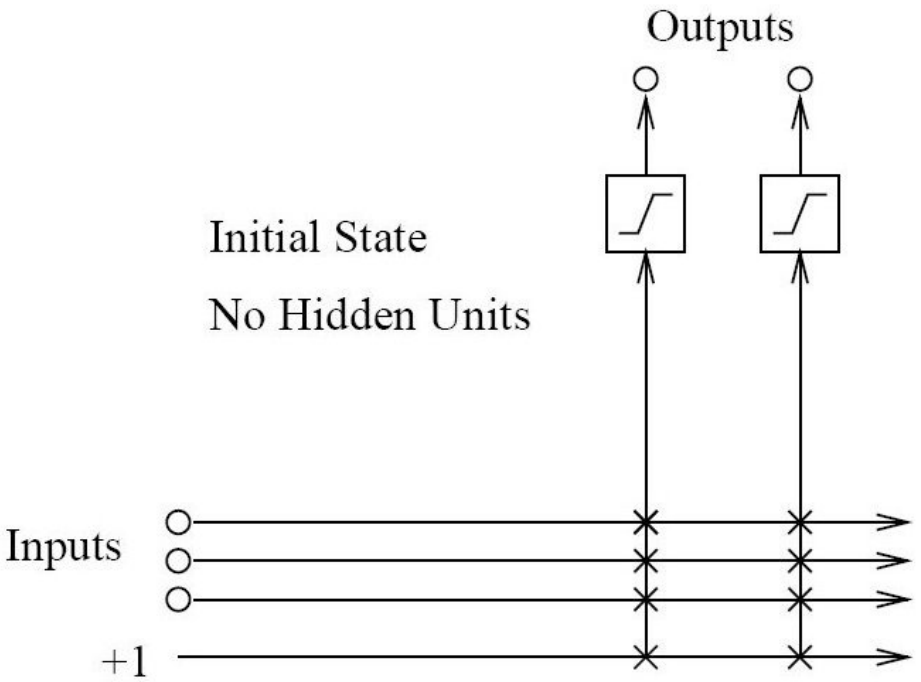
Long Short Term Memory (LSTM) NNs

Goal: Preservation of knowledge



Cascade Correlation NNs

Goal: Minimial effective architecture, easy training

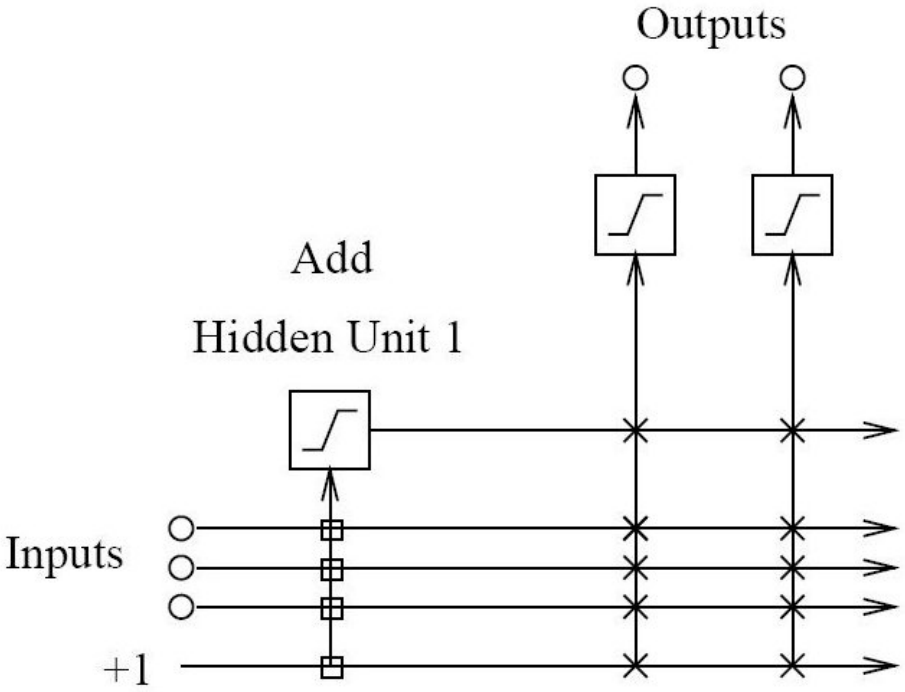


First iteration

- Occam's Razor: if you can do it with a perceptron, you don't need a NN
- Every input is connected to every output
- Training starts with no hidden units
- Train the perceptrons till training stagnates
- If the accuracy after training is unacceptable, a single hidden neuron is added

Cascade Correlation NNs

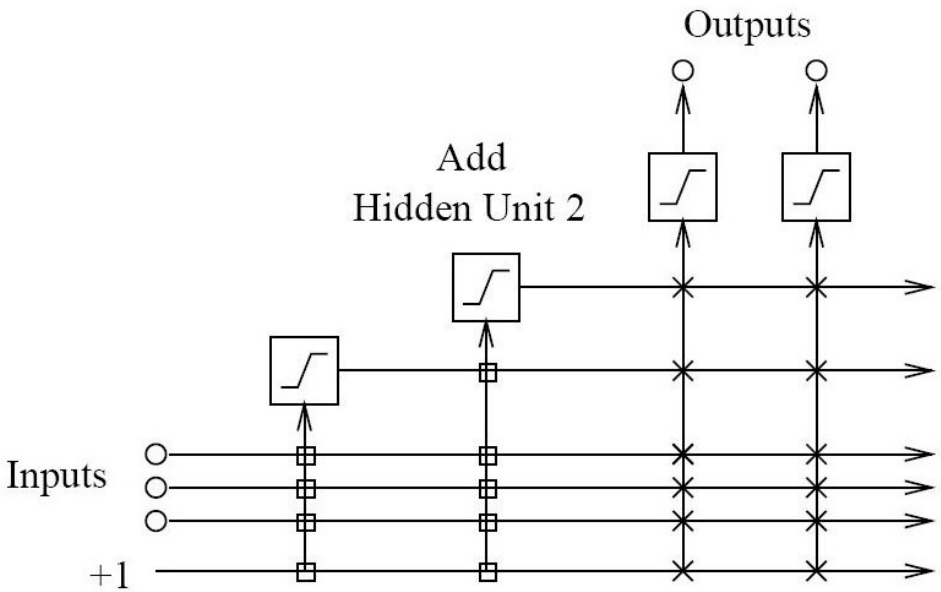
Goal: Minimial effective architecture, easy training



- All previously trained connections remain
- All inputs are connected to the hidden neuron
- Hidden neuron is connected to all outputs
- Hidden neuron's input weights are adjusted to maximize covariance between the new neuron's output and the NN error
- The input weights are then frozen, and only the output weights are trained

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training



- Train input weights first
- Once trained, freeze the input weights
- Apply perceptron learning rules to train the output weights
- In the figure, boxed connections are frozen, x connections are trained iteratively
- We always deal with only a single layer of modifiable weights

The End

- Questions?
- Next lecture: deriving backpropagation
- `http://www.gregadunn.com/self-reflected/`