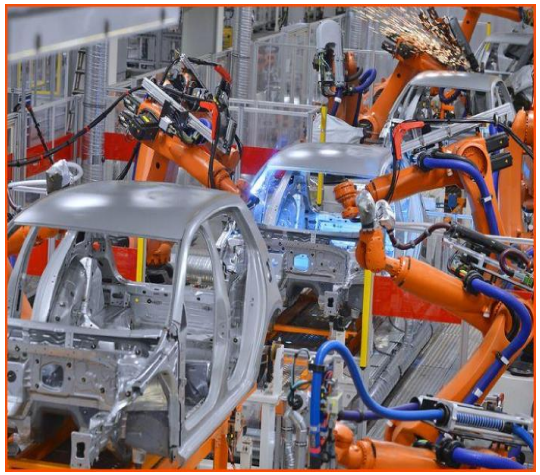
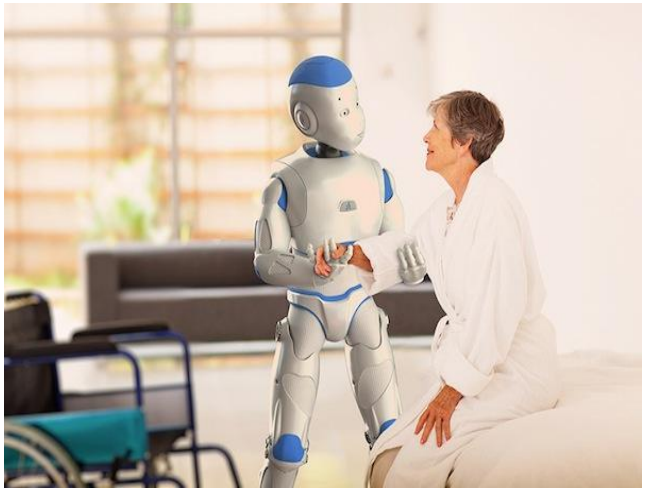
Inspire…Educate…Transform.

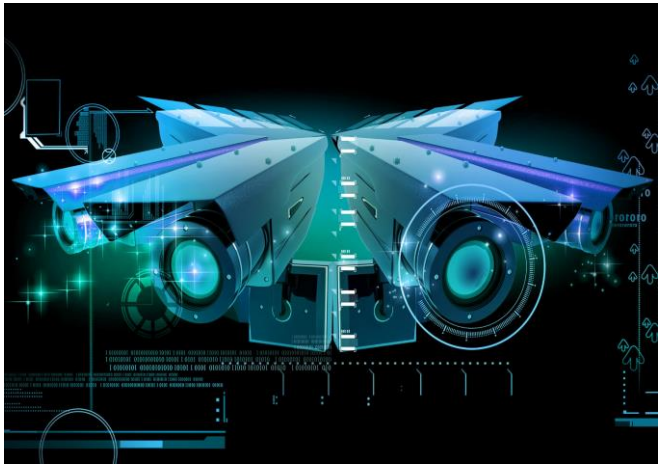
Artificial Neural Networks

Dr. Anand Jayaraman
Mentor, International School of Engineering

Automation is future







Outsider’s point of view

Stephen Hawking: “AI will be 'either best or worst thing' for humanity.”

Bill gates: “I am in the camp that is concerned about super intelligence. First the machines will do a lot of jobs for us and not be super intelligent. That should be positive if we manage it well. A few decades after that though the intelligence is strong enough to be a concern.”

Elon Musk: “I’m increasingly inclined to think that there should be some regulatory oversight, maybe at the national and international level, just to make sure that we don’t do something very foolish.”

What researchers think?

Yann LeCun: “Some people have asked what would prevent a hypothetical super-intelligent autonomous benevolent A.I. to “reprogram” itself and remove its built-in safeguards against getting rid of humans. Most of these people are not themselves A.I. researchers, or even computer scientists.”

Yoshua Bengio: “There is no truth to that perspective if we consider the current A.I. research. Most people do not realize how primitive the systems we build are, and unfortunately, many journalists (and some scientists) propagate a fear of A.I. which is completely out of proportion with reality. We would be baffled if we could build machines that would have the intelligence of a mouse in the near future, but we are far even from that.”

Director - Facebook AI Research

Head - Montreal Institute for Learning Algorithms (MILA)

What I think is that they both are right :)

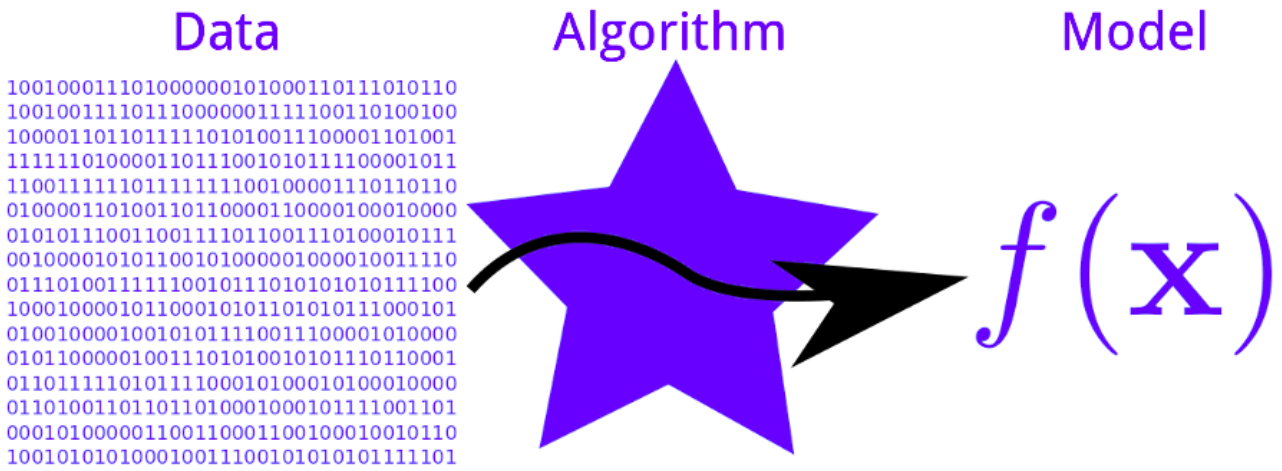
What is learning?

Learning is the act of acquiring new or modifying and reinforcing existing knowledge, behaviors, skills, values, or preferences which may lead to a potential change in synthesizing information, depth of the knowledge, attitude or behavior relative to the type and range of experience

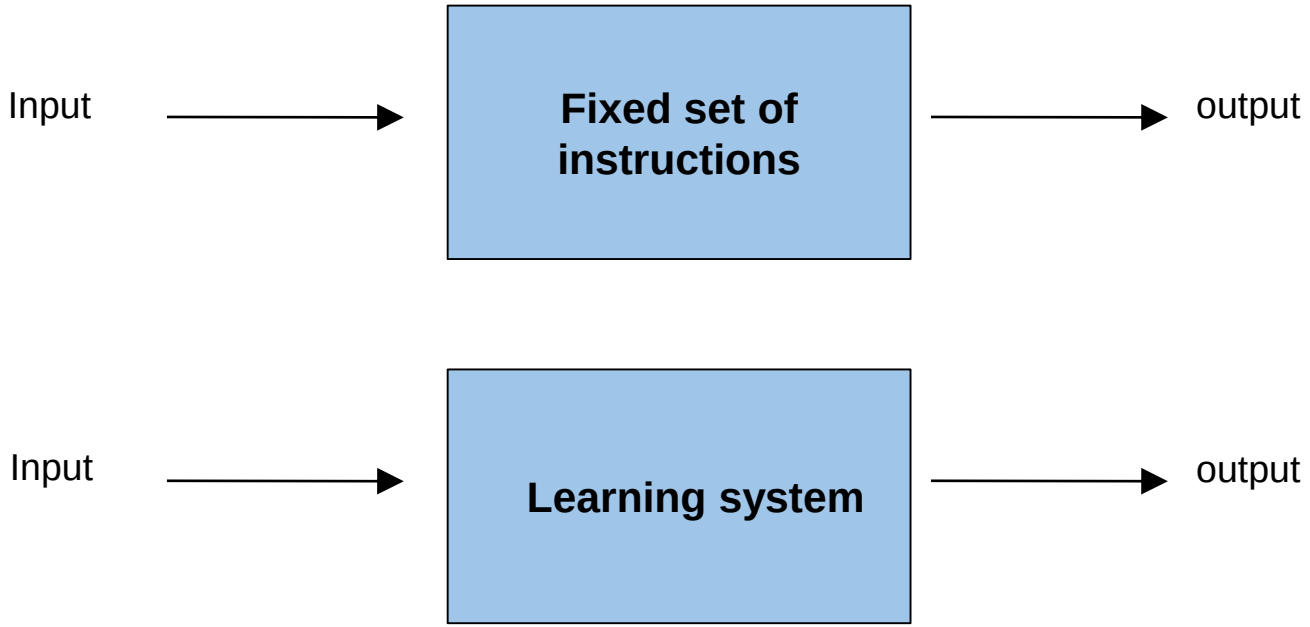
-- Wikipedia.org

Machine Learning

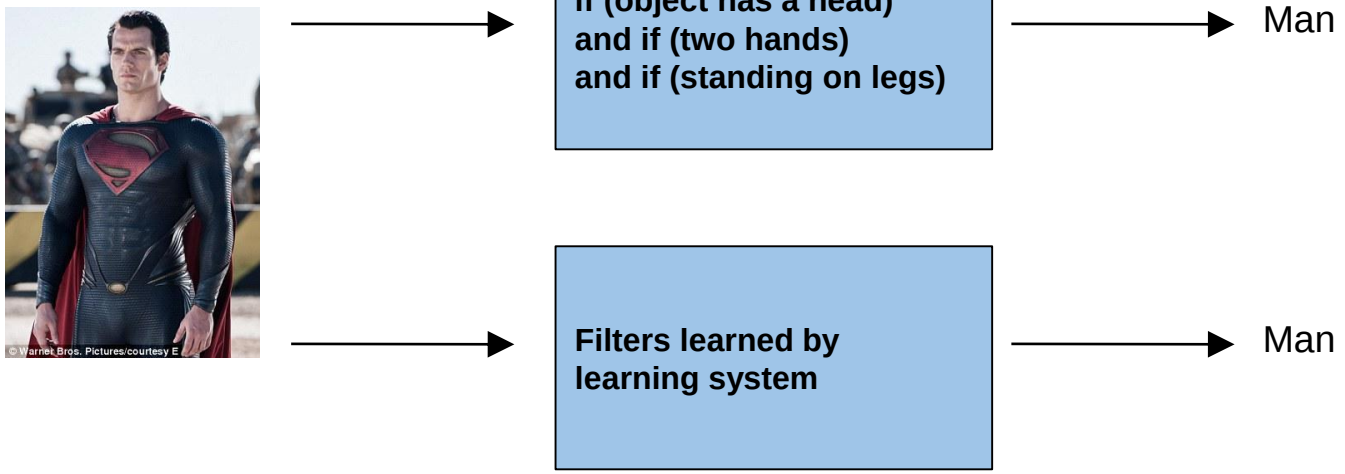
How can we build computer programs that automatically improve their performance through experience?



Two primary ways of doing a task?



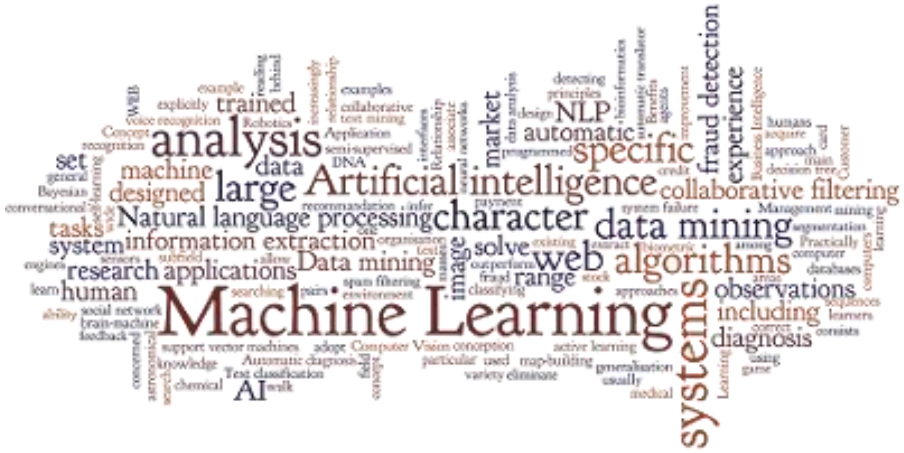
Two primary ways of doing a task?



Machine Learning

Approaches:

- Artificial neural networks
- Bayesian networks
- Clustering
- Representation learning
- Reinforcement learning
- Support Vector Machines
- Logistic and Linear regression
- ...



Applications:

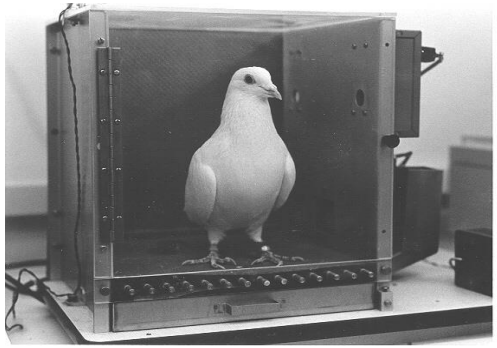
- Structured data problems
- Computer vision
- Natural language processing
- information retrieval
- Robotics
- ...

Best learning system known to us?

Best learning system known to us?

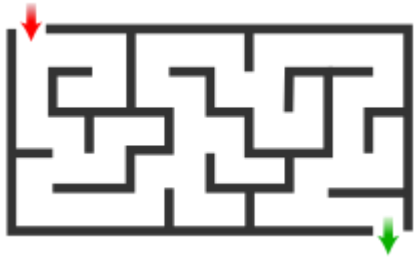
Biological neural system

Thinking is possible even with a “small” brain



Pigeons as art experts – 85% (Watanabe *et al.* 1995)

Source: https://en.wikipedia.org/wiki/Marc_Chagall
https://en.wikipedia.org/wiki/Vincent_van_Gogh



Mice trained to run mazes[1], detect drugs[1][2]

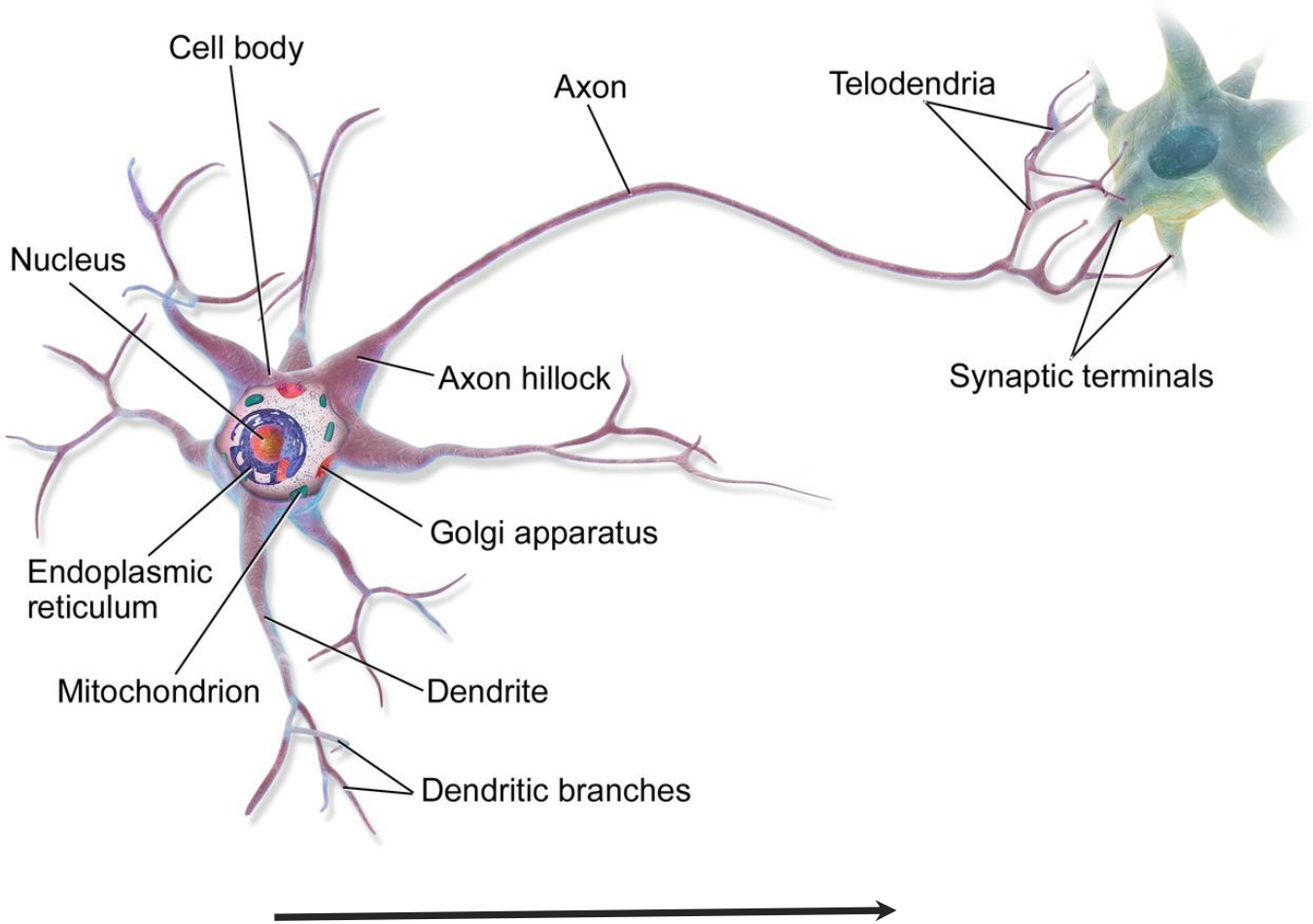
The results

- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)
- Discrimination still 85% successful for previously unseen paintings of the artists
- Mice can memorize mazes, odors of contraband (drugs / chemicals / explosives)

Today's Agenda

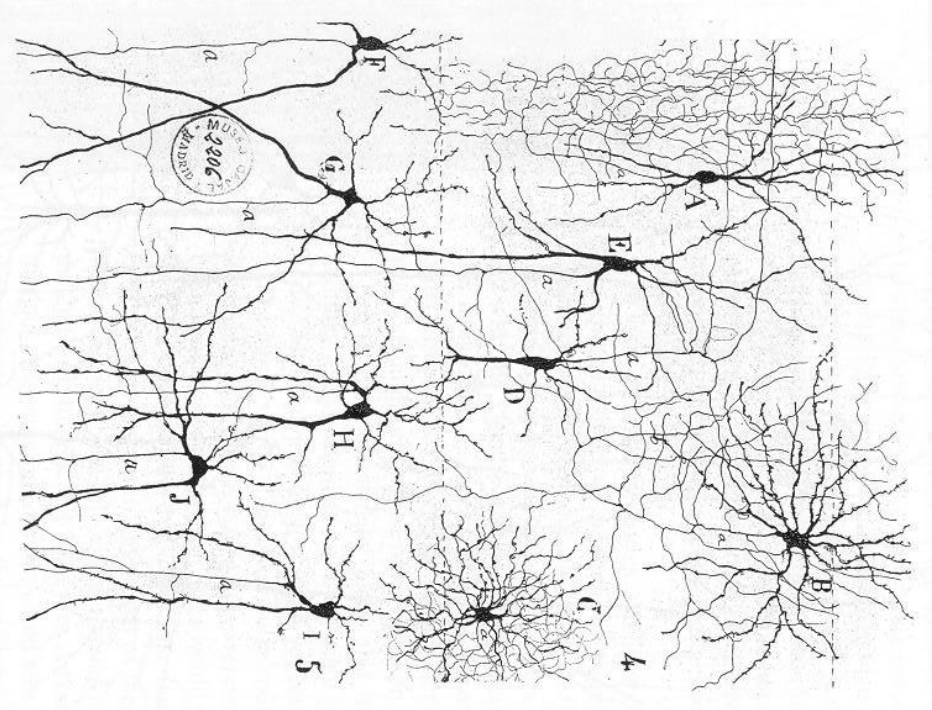
- A simple perceptron model
- How is it similar to linear/logistic regression
- Limitations of a perceptron
- Multi-layer perceptron to over come limitations.
- How to train a neural network?
- Practical tips
- Sample applications
- Future scope

So, how does the brain work?



Direction of signal is along the Axon from Nucleus to synapse

Biological neural networks



- Fundamental units are termed neurons.
- Connections between neurons are synapses.
- Adult human brain consists of 100 billion neurons and 1000 trillion synaptic connections.

Biological inspiration

The spikes travelling along the axon of the pre-synaptic neuron trigger the release of neurotransmitter substances at the synapse.

The neurotransmitters cause excitation or inhibition in the dendrite of the post-synaptic neuron.

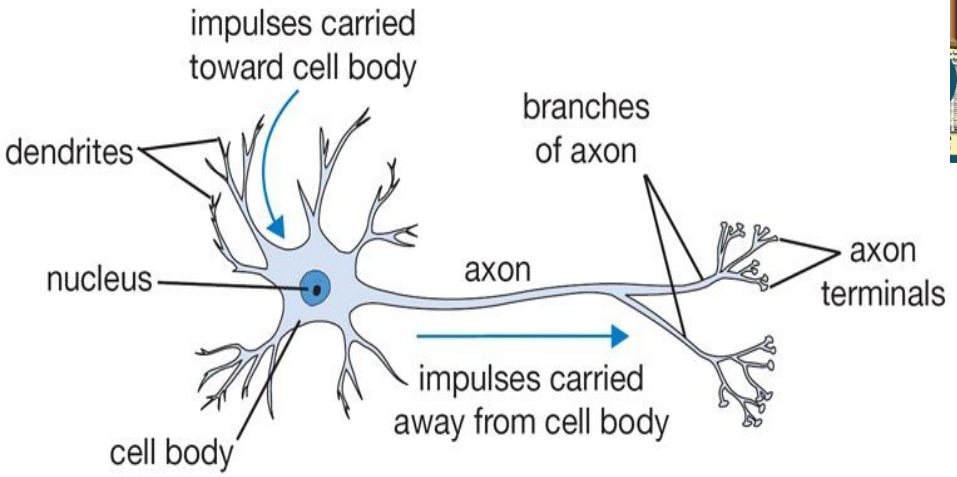
The integration of the excitatory and inhibitory signals may produce spikes in the post-synaptic neuron.

The contribution of the signals depends on the strength of the synaptic connection.

In 1949 Donald Hebb postulated one way for the network to learn. If a synapse is used more, it gets strengthened – releases more Neurotransmitter. This causes that particular path through the network to get stronger, while others, not used, get weaker.

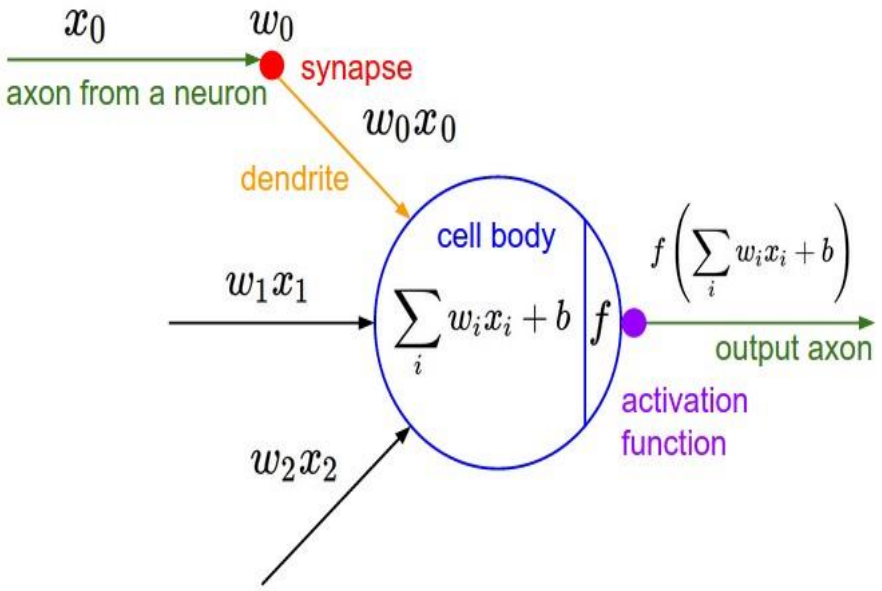
You might say that each connection has a *weight* associated with it – larger weights produce more stimulation and smaller weights produce less.

Computational Neuron

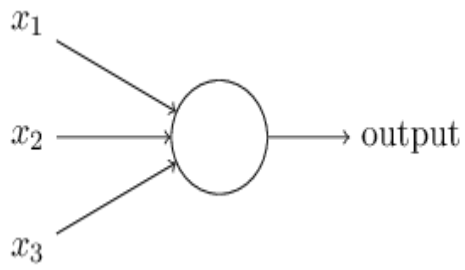


Artificial (Computational) Neuron
- Input, Weights, Sum, Threshold

Learning
- Synaptic strengths : influence of one neuron on another;
  - learnt by feedback / experience / observation
- Dendrites carry signals to cell body where they are summed.
- If (sum > threshold), the neuron fires
  - Input to the next neuron (based on synaptic strength)
- Hebbian Learning : If a synapse is used more, it gets strengthened; This causes that particular path through the network to get stronger, while others, not used, get weaker.
- Machine Learning: Determine synaptic strength (weights) by finding the optimal weights consistent with the given data



Perceptron : The first artificial neuron



!dea
- Neuron structure as a model of decision making
- makes decisions by weighing up evidence.
- Not meant as a complete model of decision making

Captures key characteristics
- Input, Weights, Sum, Threshold

Notation
- Dot Product (Vectors of inputs and weights)
- Neuron Input : Weighted sum of inputs ($z = w \cdot x$)
- Threshold → Bias

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Artificial Neural Networks

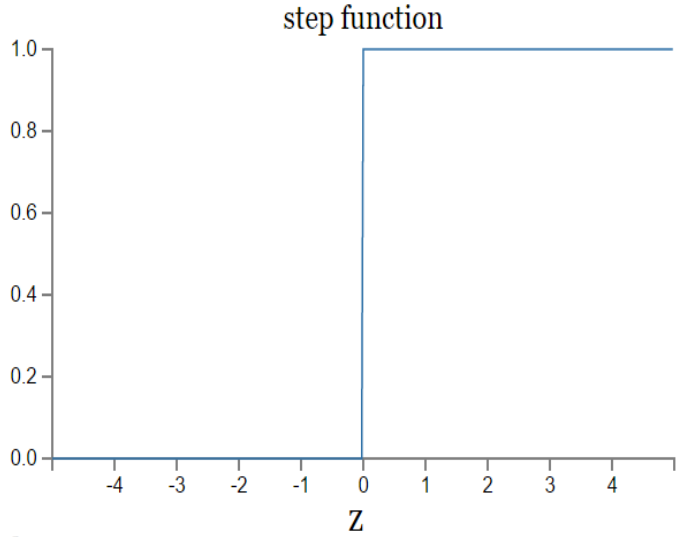
L'Avion III de Clément Ader, 1897(Musée du CNAM, Paris)

Let us draw some inspiration from the best learning system

Warning! Do not copy it exactly!

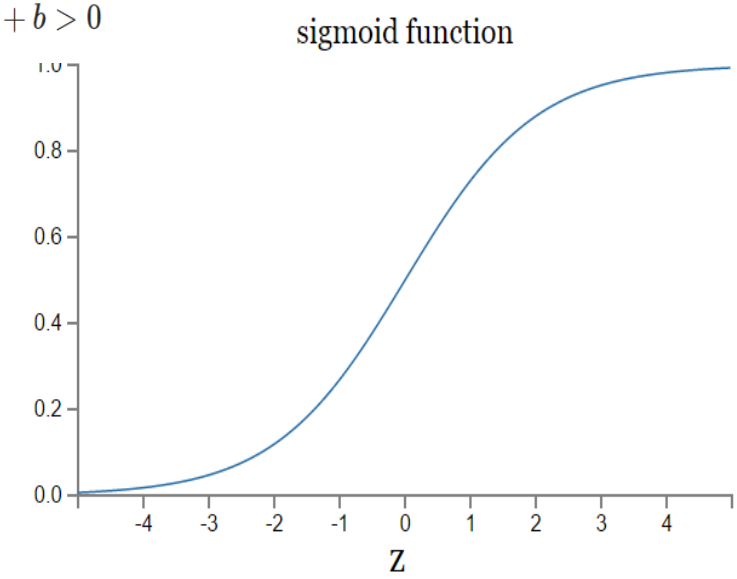
One needs to understand what needs to be borrowed.

From Perceptron to Sigmoid Neurons

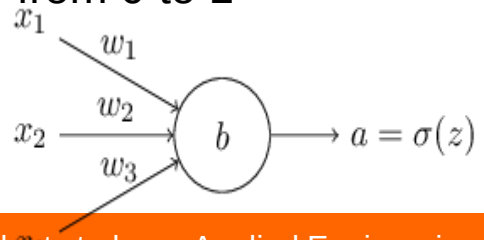


- Perceptrons are brittle
  - Change in output w.r.t. change in input is spiky
- Why does this matter?
  - How to train a perceptron? How to learn from data?
  - Try out different weights which minimize the error
  - Perceptron : As we change weights,
    - most of the time nothing happens
    - At some point, output switches (not smooth)
  - Desirable
    - small change in weight ➔ small change in the output
- Sigmoid Neuron
  - !dea : use a different activation function
  - z is large and positive ➔ output ~ 1
  - z is large and negative ➔ output ~ 0
  - Intermediate z ➔ output changes smoothly from 0 to 1

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$



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



$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

Classification with Logistic

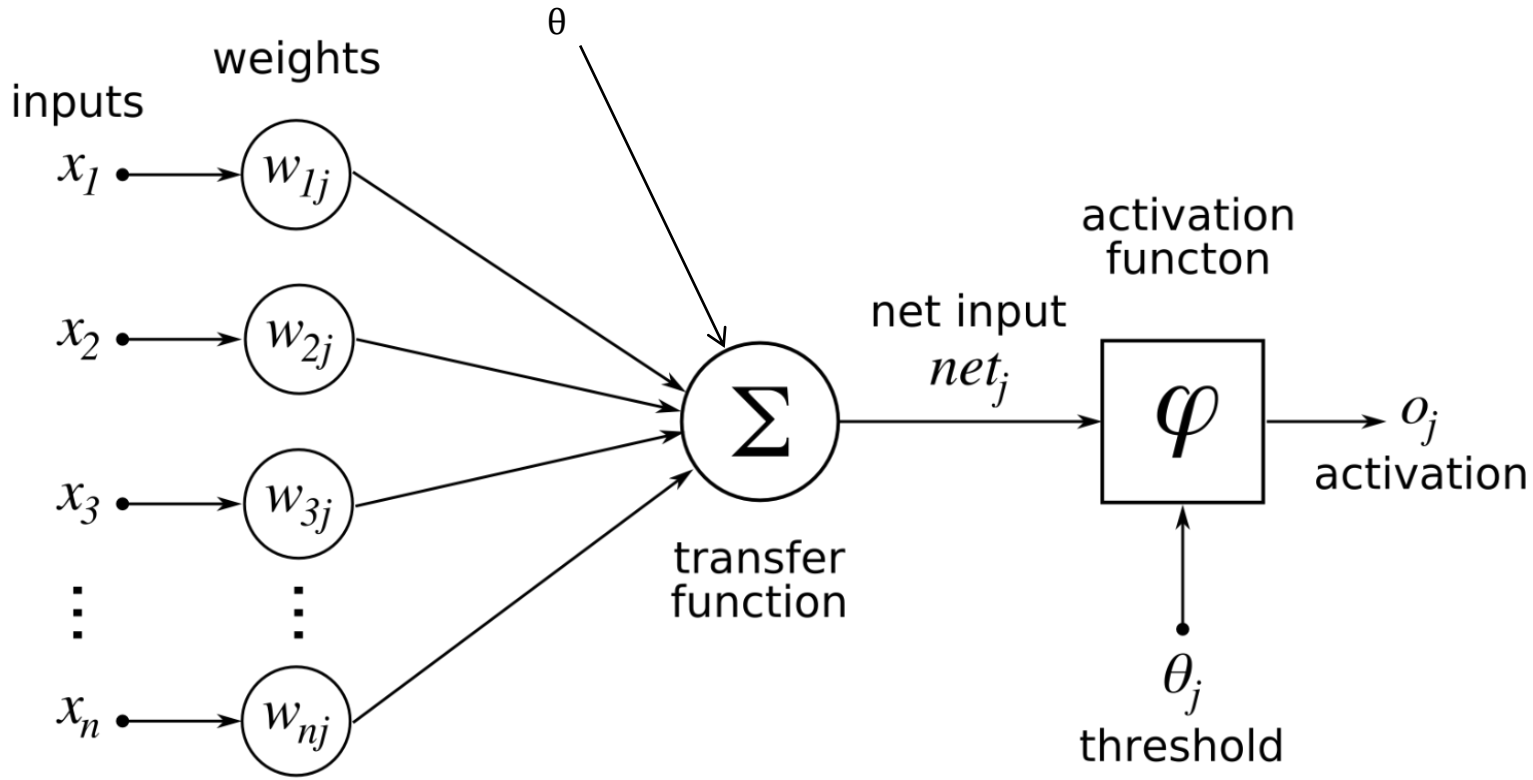
Logistic regression model

Independant variable (x)	Weight coefficients (W)
Salary	w1
Credit rating	w2
Age	w3
Education level	w4
Account balance	w5

Weight coefficients are learned from the data for a classification task (assume binary)

$$Y = \text{sigmoid}(Wx + b)$$

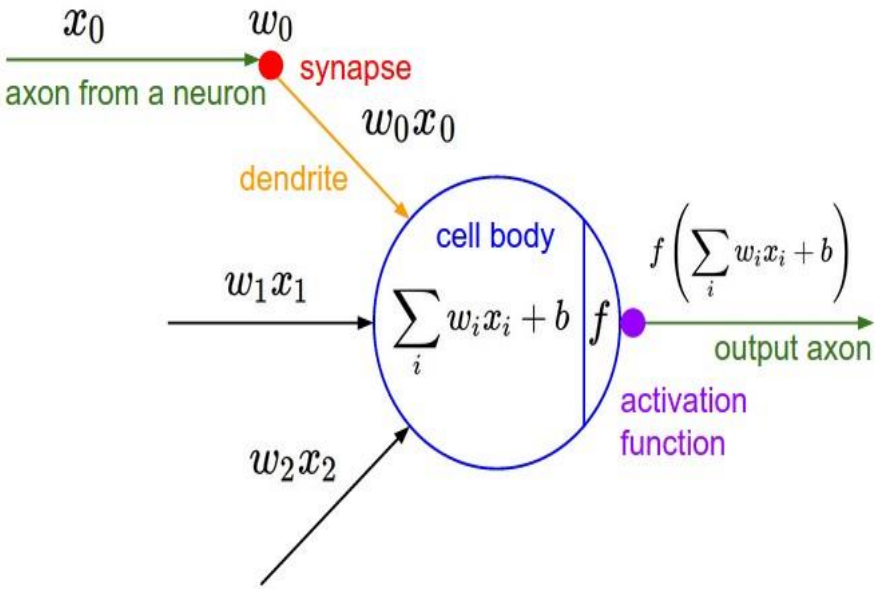
Artificial neural model: Perceptron



$$o = \varphi(wx + \theta)$$

Equation looks similar to some other model?

Neurons : Sigmoid et. al.

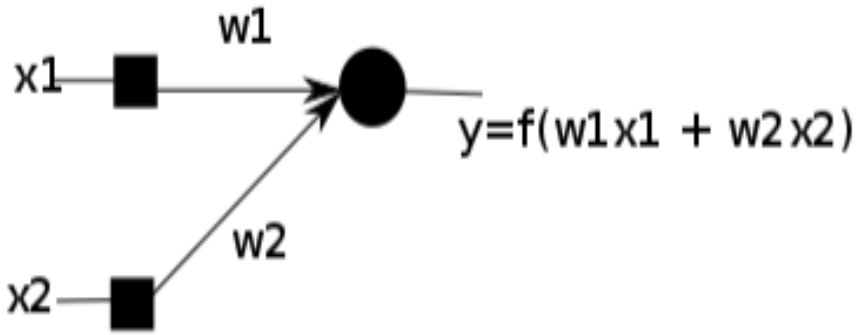


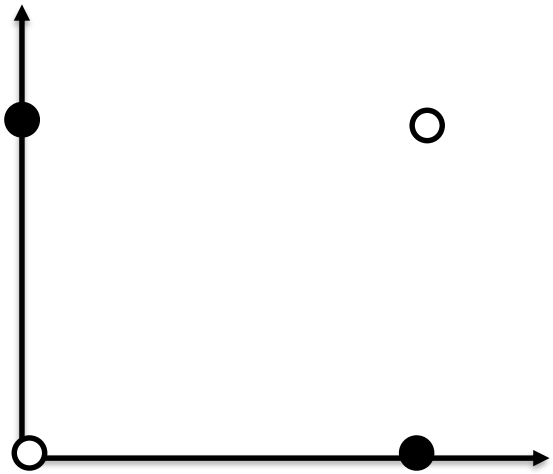
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer NN	

LIMITATIONS OF PERCEPTRON AND HOW TO OVERCOME

Will it work here?

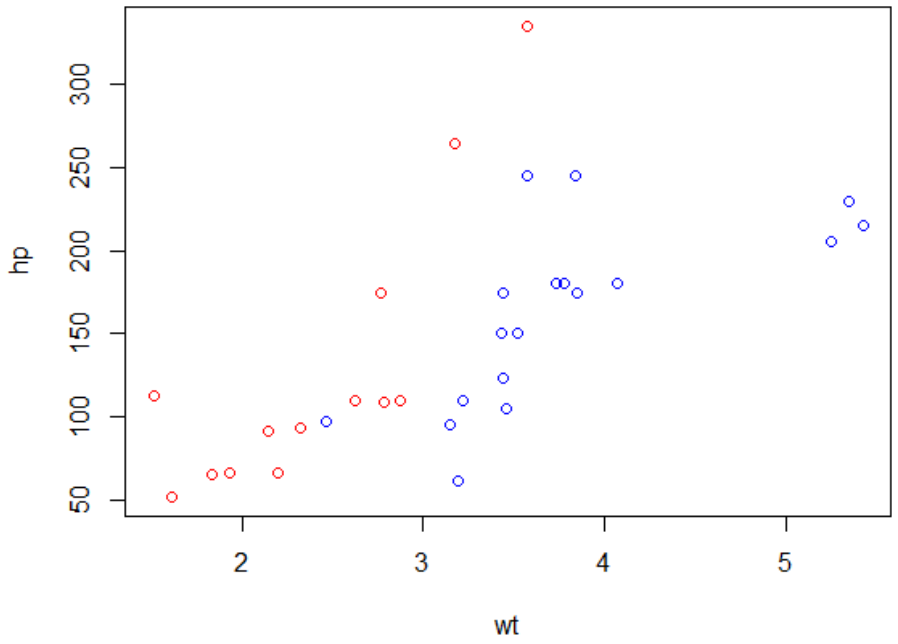
x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0





Remember this example from logistic regression?

Using the MTcars dataset, estimate the probability of a vehicle being fitted with a manual transmission if it has a 120hp engine and weights 2800 lbs.




```
> with(mtcars,plot(wt,hp, col=c("blue","red")[am+1]))
```

Logistic Regression: Class Boundary

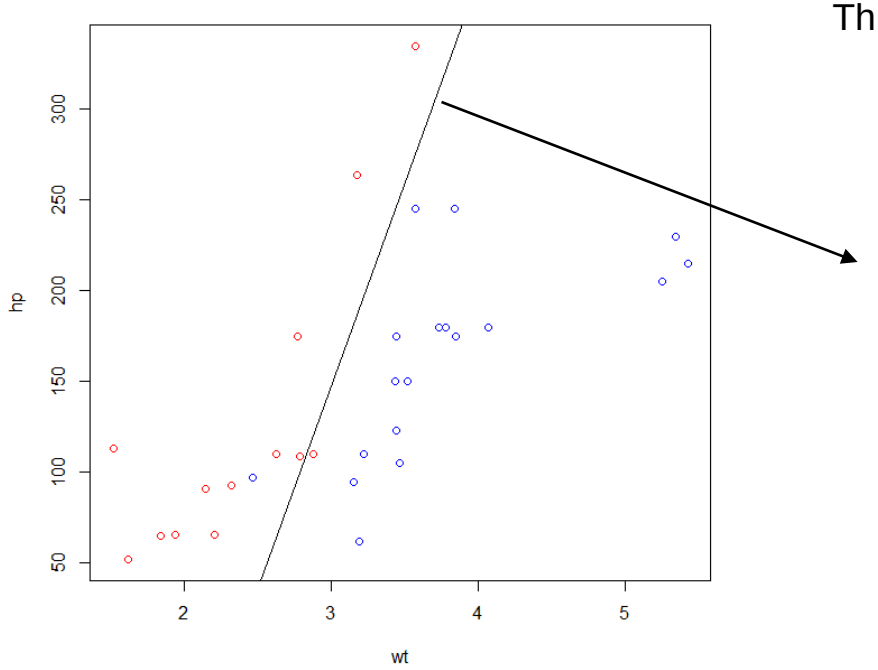
```
Call:
glm(formula = am ~ wt + hp, family = binomial, data = mtcars)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2537  -0.1568  -0.0168   0.1543   1.3449

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 18.86630    7.44356   2.535  0.01126 *
wt          -8.08348    3.06868  -2.634  0.00843 **
hp           0.03626    0.01773   2.044  0.04091 *
---
```

Equation of the fit
Ln(S) = 18.8663 – 8.080348 wt + 0.03636 hp

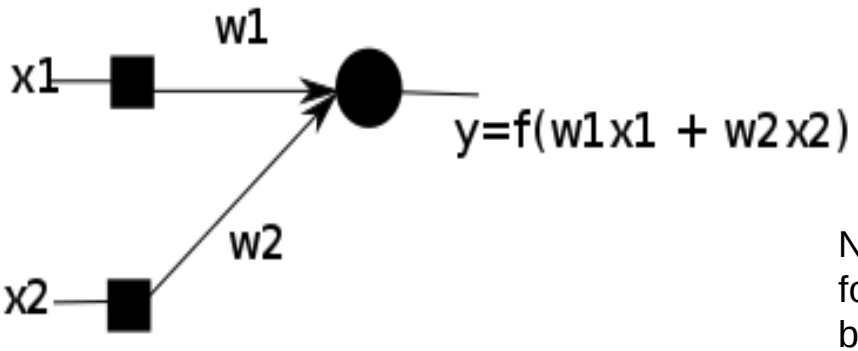
Setting Ln(S) = 0 in the above equation gives
the equation of dividing line between the two classes.
This line marks the set of points for which prob=0.5

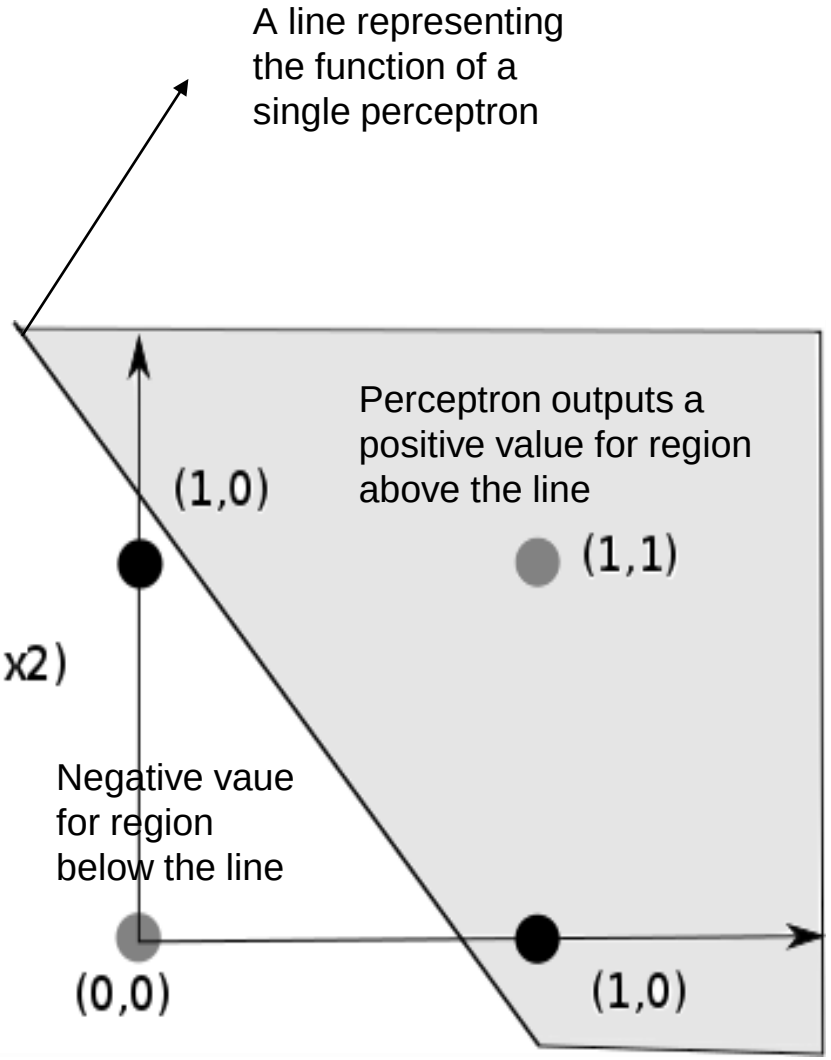


0 = 18.8663 – 8.080348 wt + 0.03636 hp

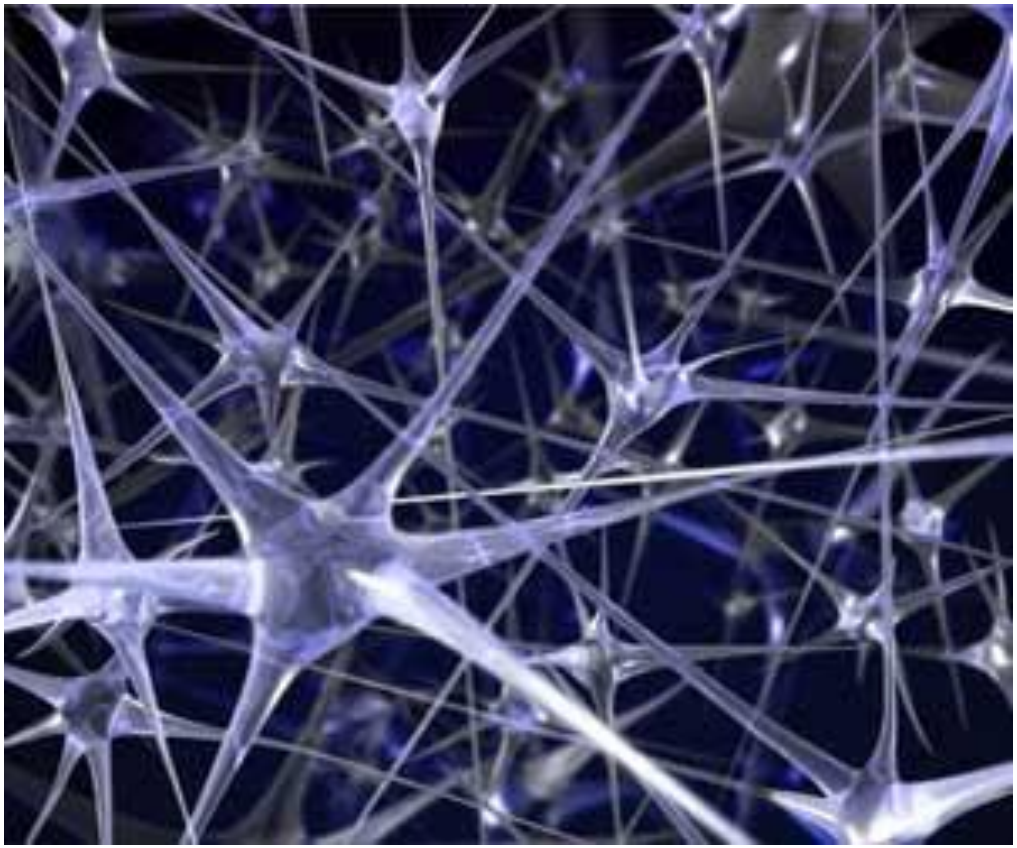
Will it work here?

x1	x2	y
0	0	0
1	0	1
0	1	1
1	1	0





How does brain do non linearity

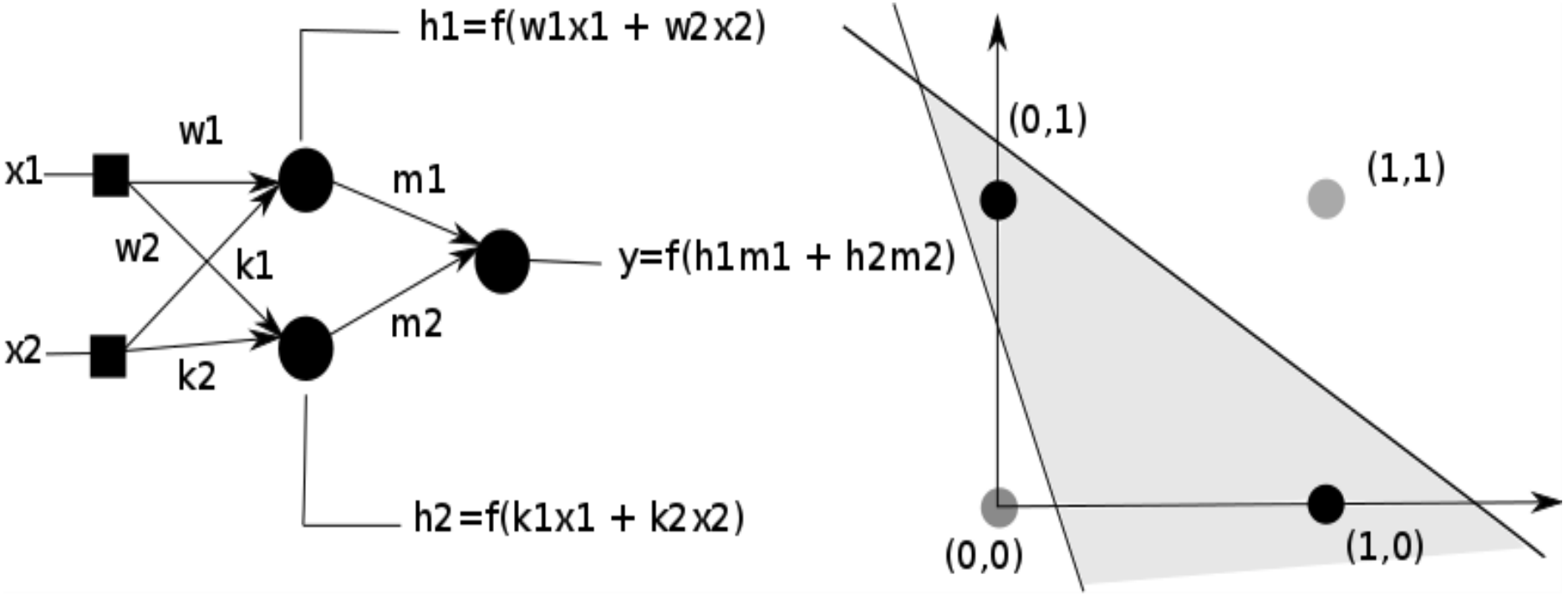


Many neurons connected together

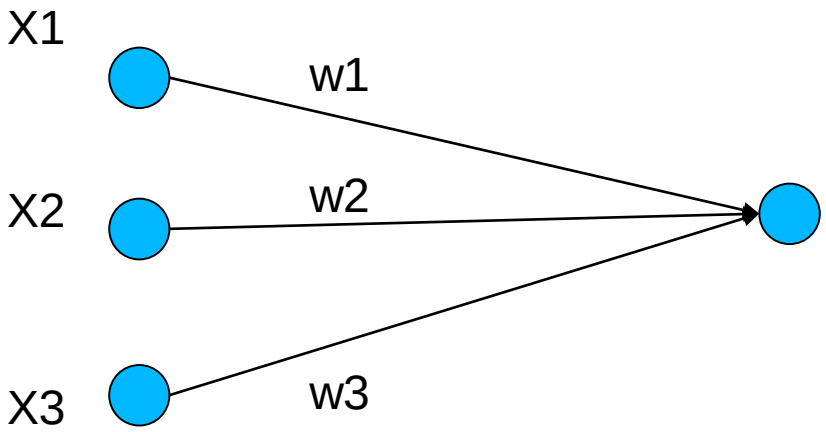
A single perceptron is not able to deal with non-linear data.

What does your intuition tell you to do based on biological inspiration?

A one hidden layer network can learn any function

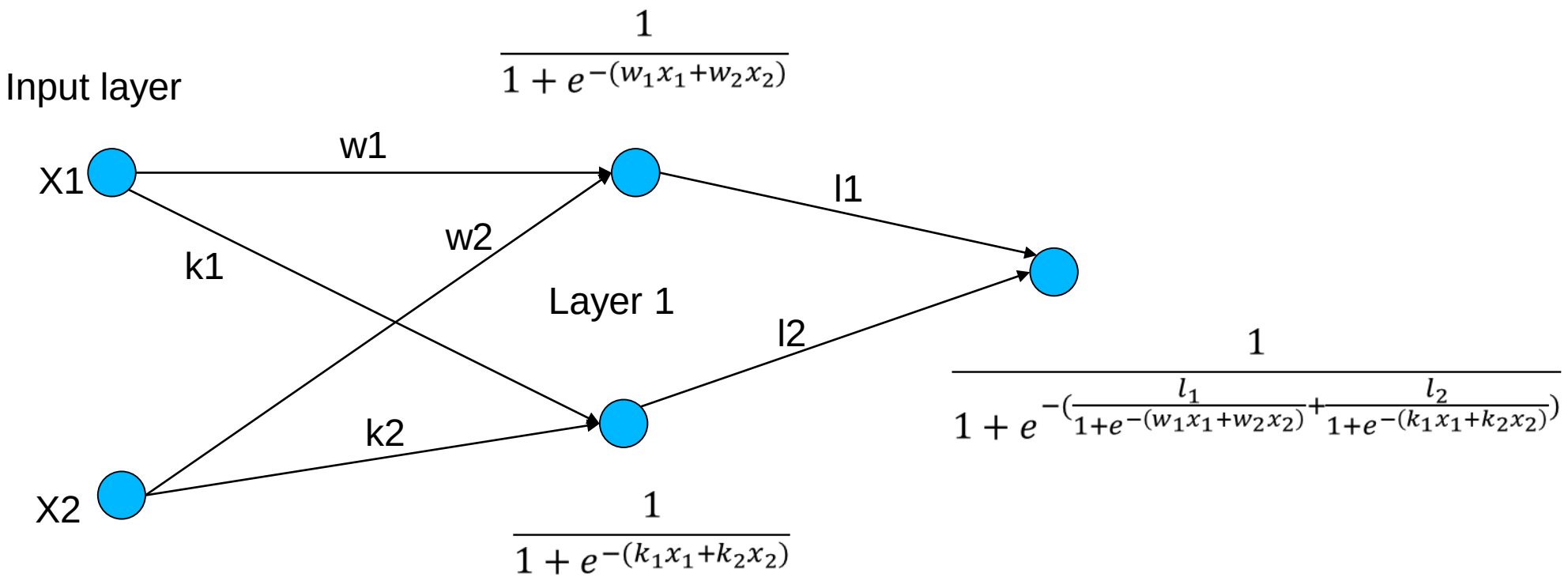


How to describe non-linearity



$$\frac{1}{1+e^{-(w_1x_1+w_2x_2+w_3x_3)}}$$

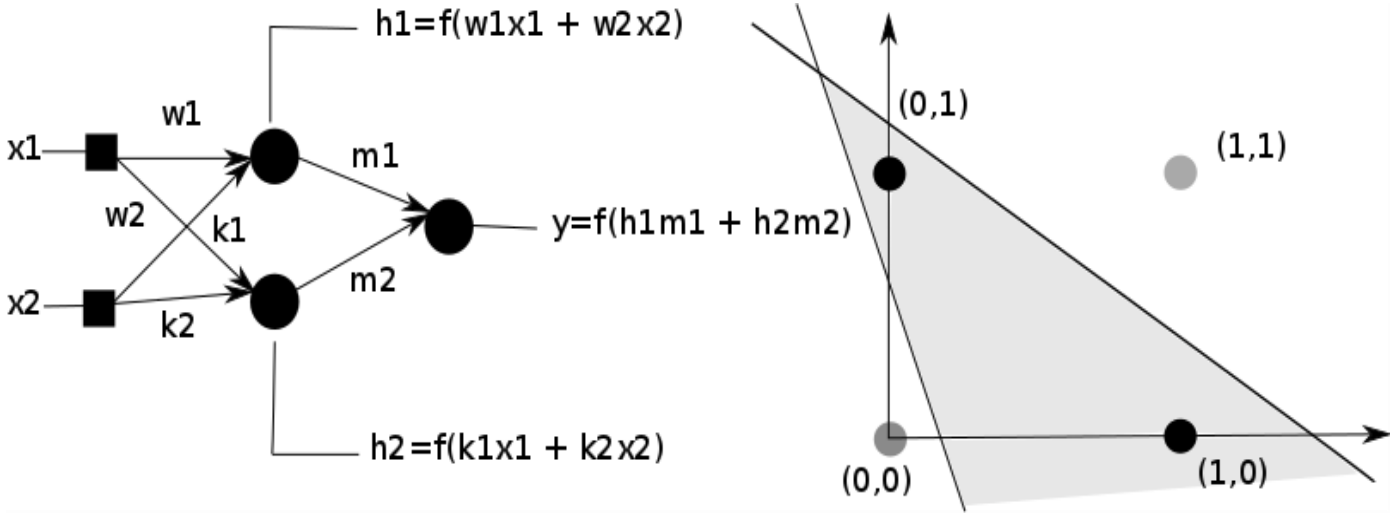
How to describe non-linearity



Transformation matrix between input layer and layer1 is

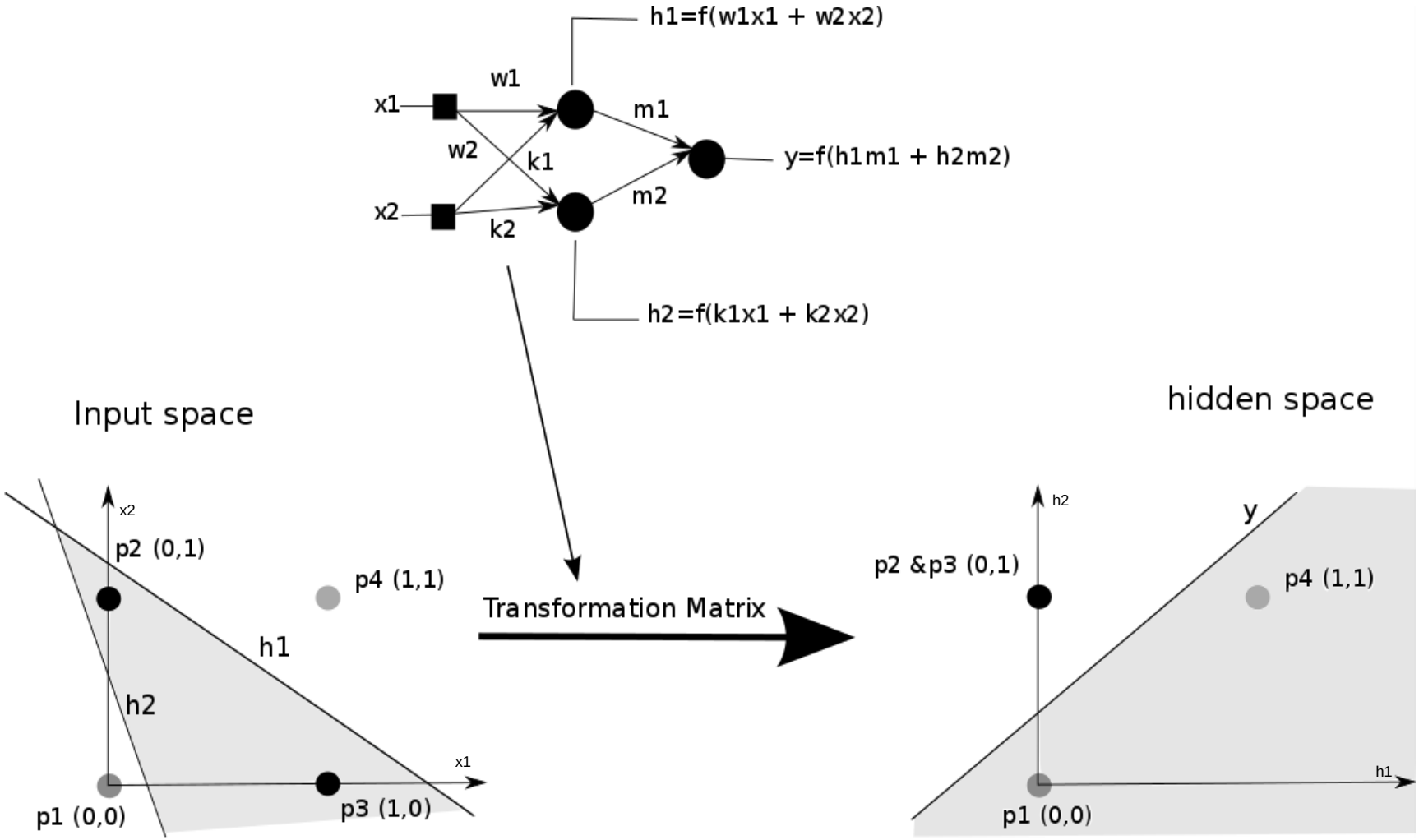
$$W_1 \quad \begin{bmatrix} w1 & k1 \\ w2 & k2 \end{bmatrix}$$

Each layer is a transformation into new space

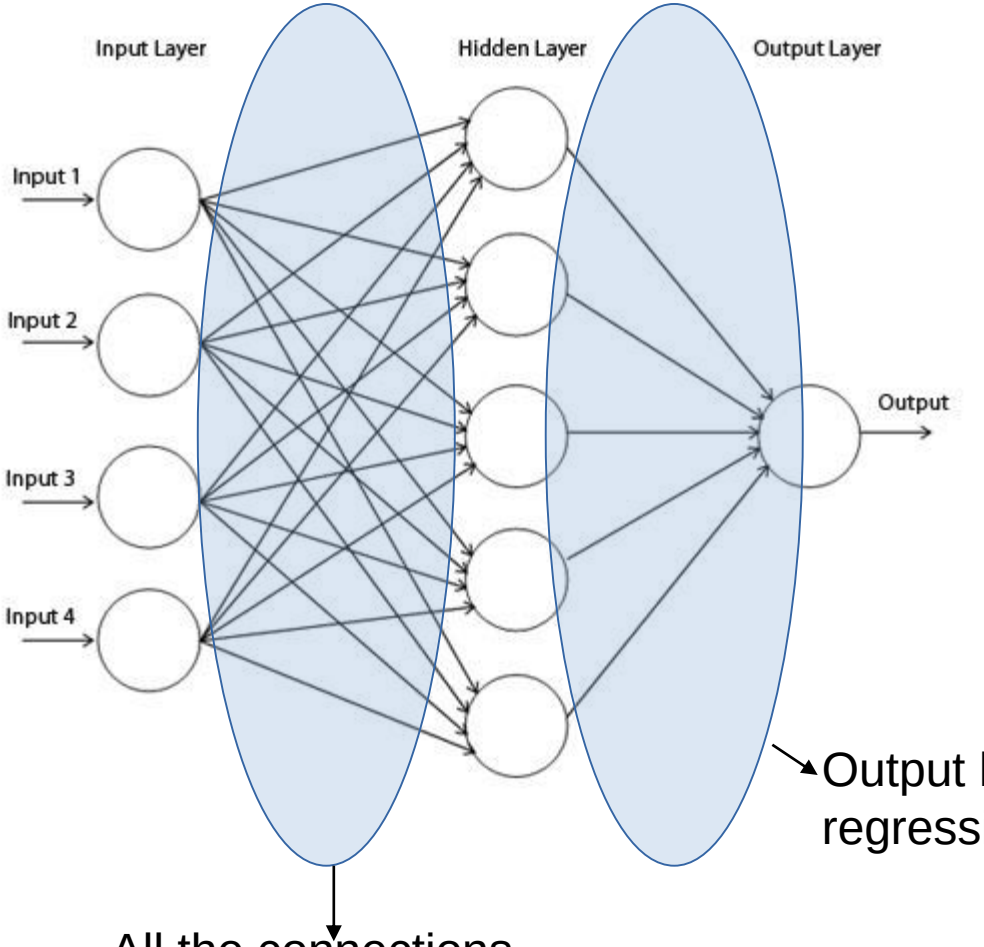


x1	x2	h1	h2	y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Each layer is a transformation into new space



Each layer is a transformation into new space

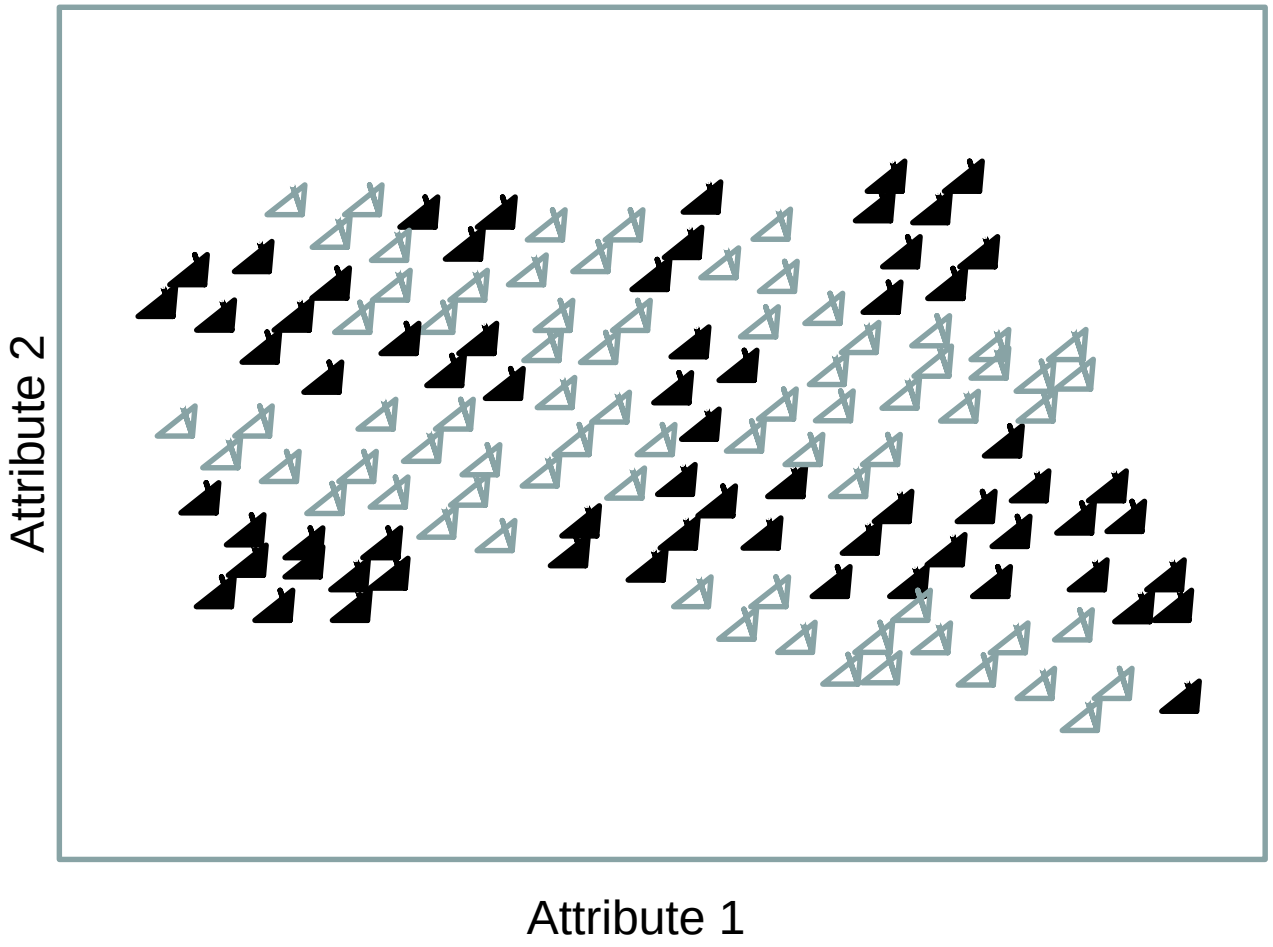


Remember transformation matrix from 2D/3D geometry?

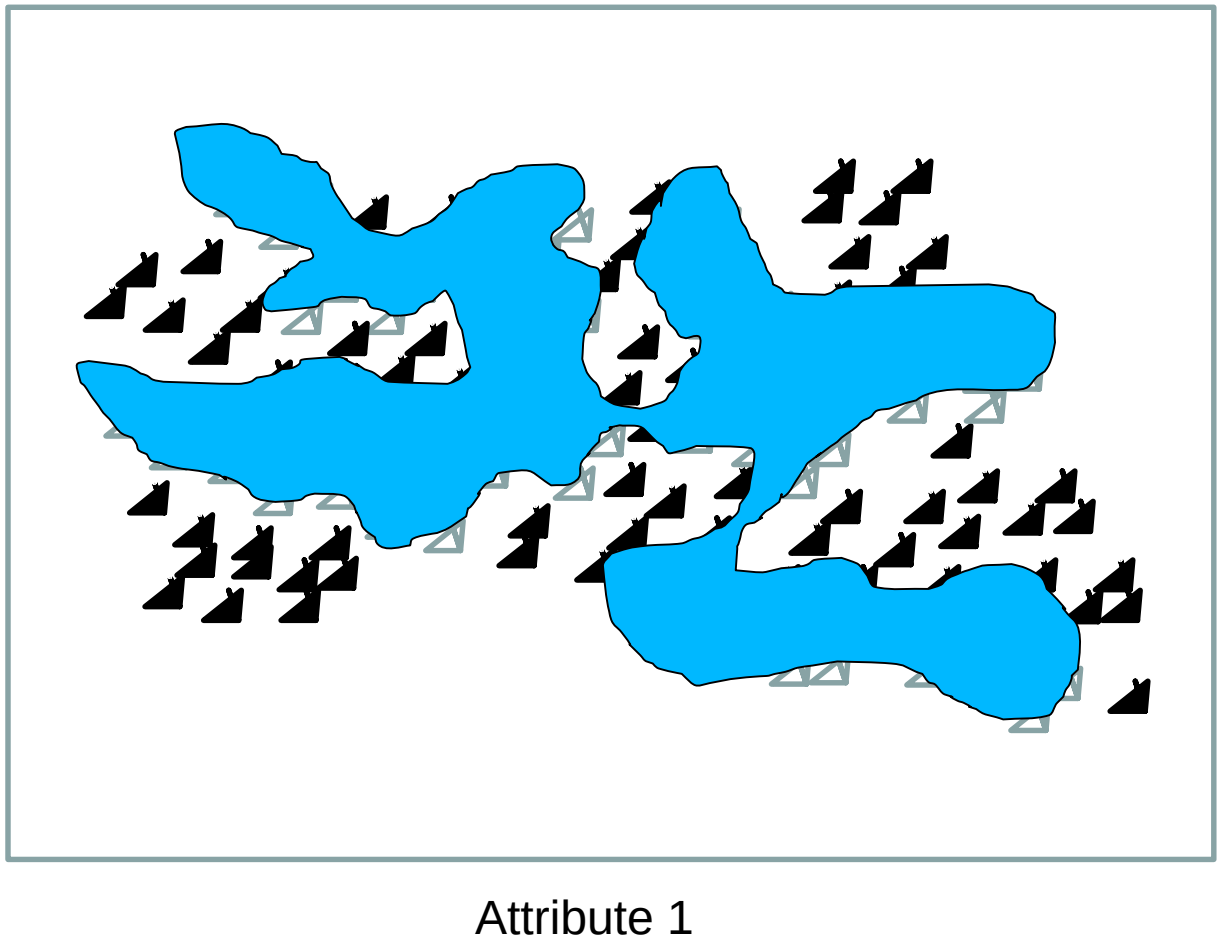
Output layer is a simple regression/classification layer

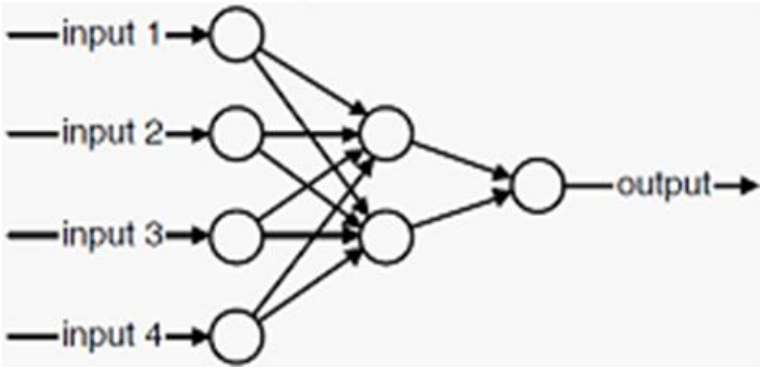
All the connections represent a Matrix W

Non-linear problems

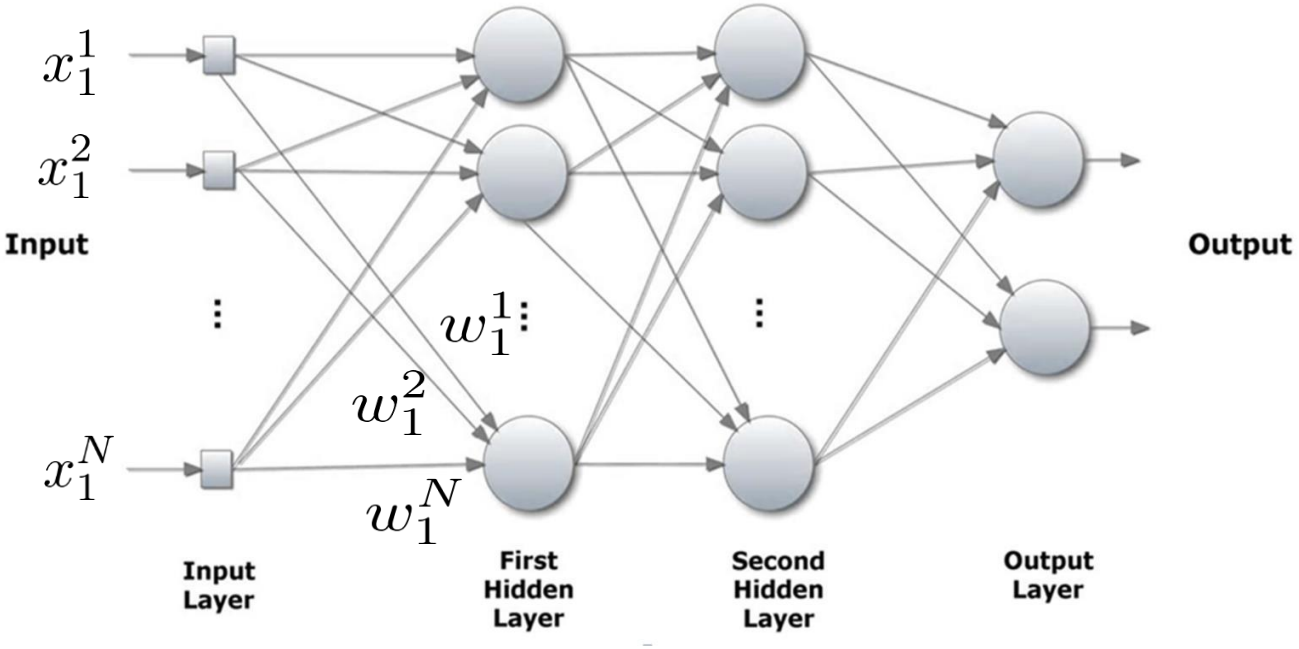


Non-linear problems





Artificial neural networks



A multilayer perceptron

- Feed forward network
- Fully connected

Artificial neural networks

Output of one layer is input to another.

Layer1: $h_1 = f_1(x) = \sigma(w_1 x + b_1)$

Layer2: $h_2 = f_2(h_1) = \sigma(w_2 h_1 + b_2)$

Layer3: $h_3 = f_3(h_2) = \sigma(w_3 h_2 + b_3)$

.

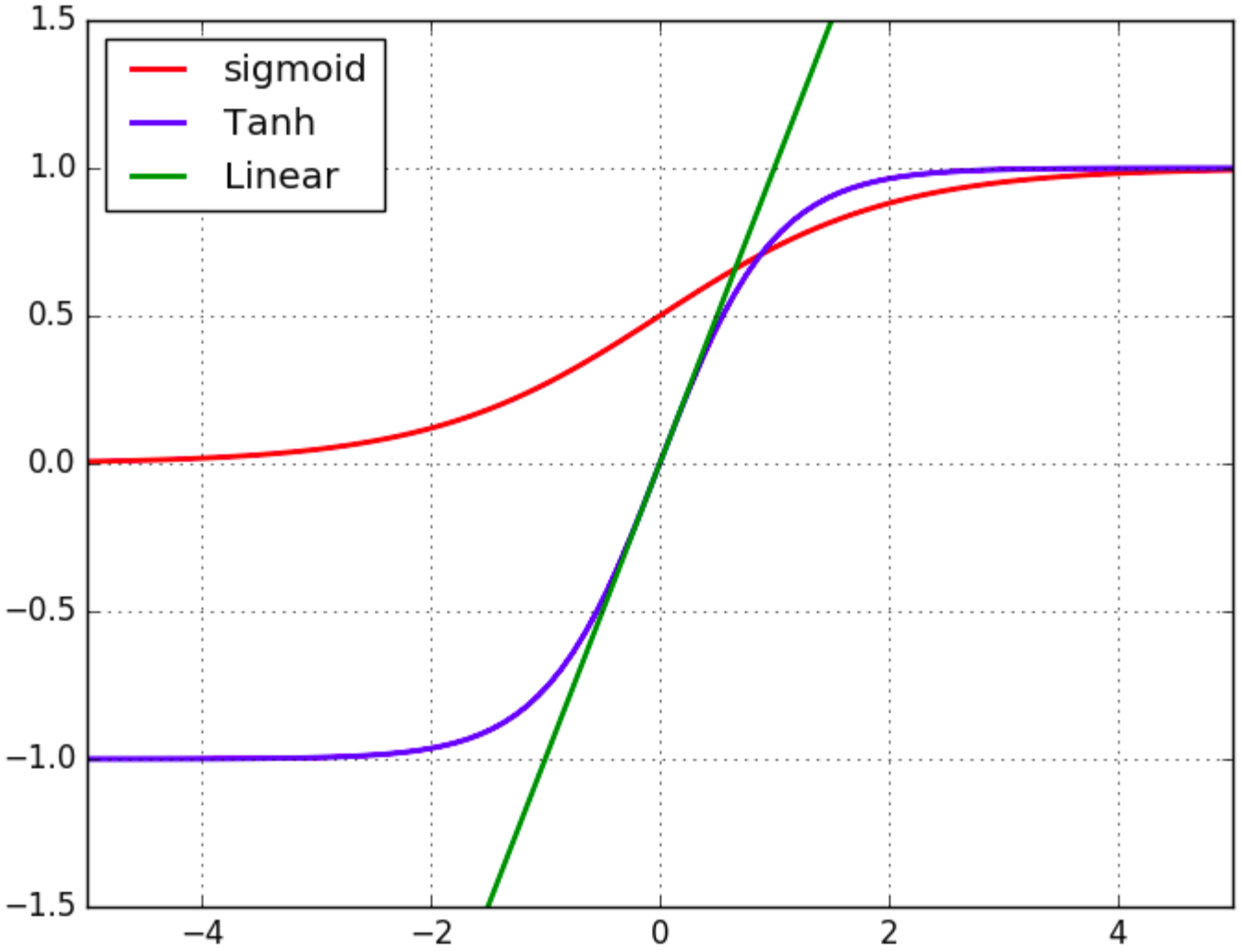
.

.

LayerN: $\text{output} = f_N(h_{N-1}) = \sigma(w_N h_{N-1} + b_N)$

$$\text{output} = f_N(f_{N-1}(f_{N-2}(\ldots(f_1(x)) = f(x)$$

Squashing/Activation/threshold functions



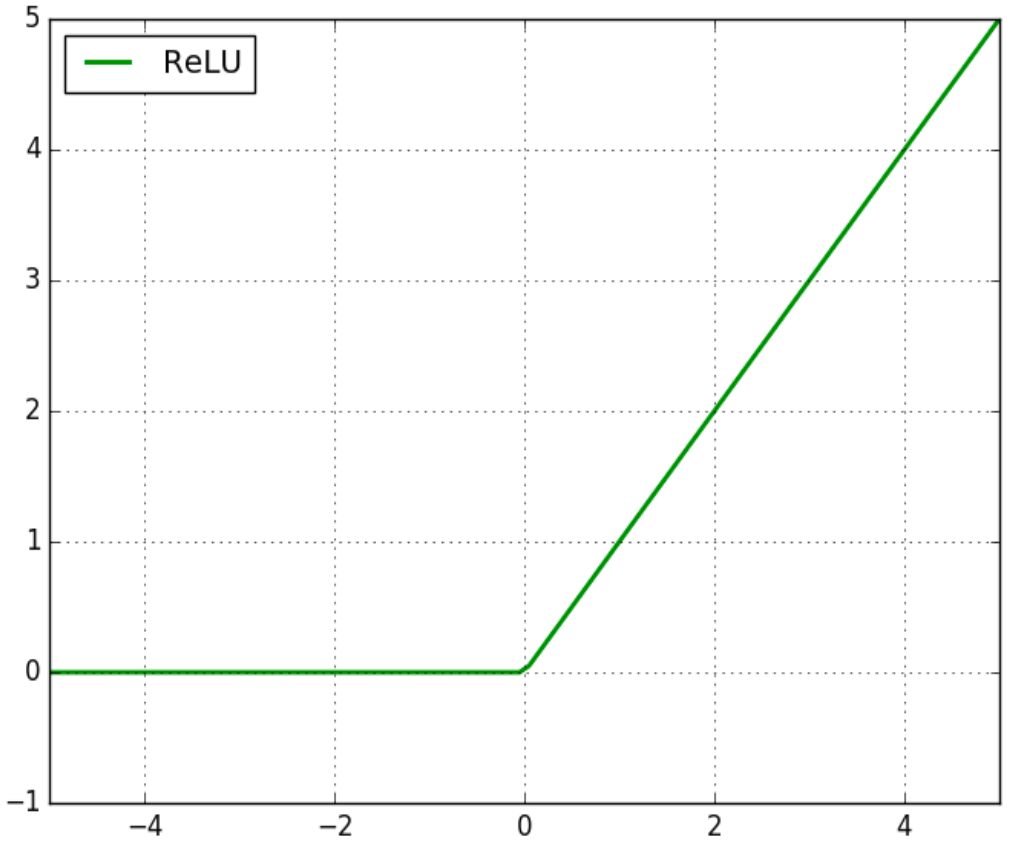
Sigmoid is good for classification

Tanh helps in better learning

Linear for regression

Rectified linear functions

- More biologically plausible
- Computationally faster
- Most popular activation function for deep neural networks
- Efficient gradient propagation: No vanishing gradient problem or exploding effect



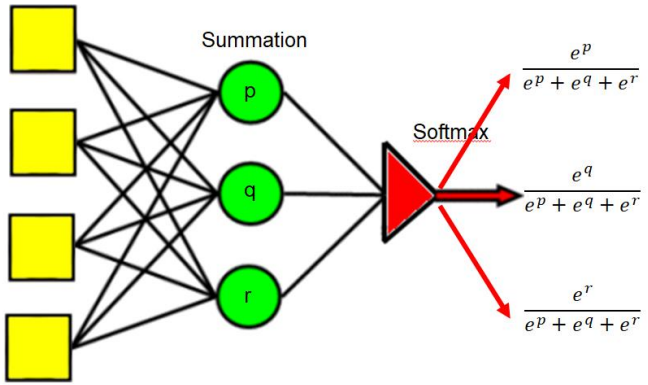
Output Layer

Activation function in output layer

1) Linear : for regression problems
2) Sigmoid/tanh: classification problems

Multi-class classification problems:

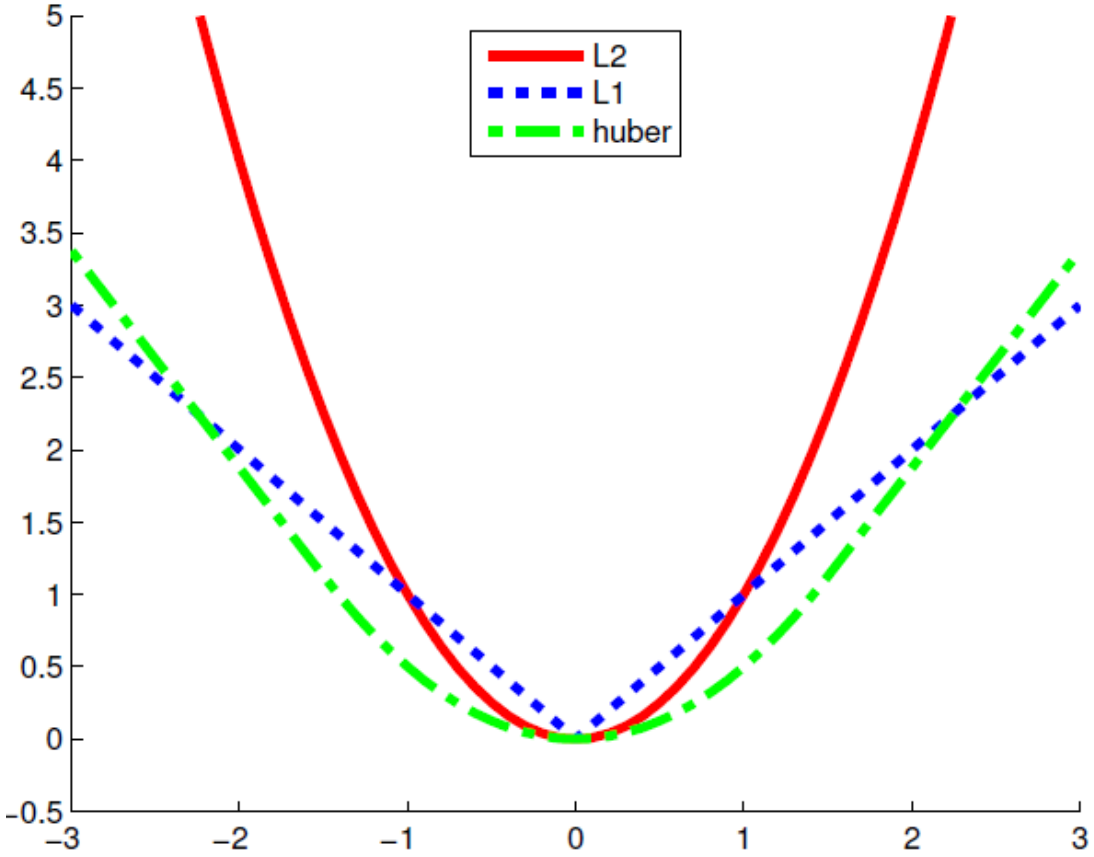
- Multiple output neurons
- Activation function for the last layer: softmax



$$\sigma(x_j) = \frac{e^{x_j}}{\sum_i e^{x_i}}$$

Common loss functions in regression

- Square (l_2) loss $(y - w_i x_i)^2$
  - Properties
    - Convex, Not robust (but OK), Non-Sparse
- Absolute (l_1) loss $|y - w_i x_i|$
  - Properties
    - Convex (not-smooth), Not robust (but OK), Non-Sparse
- Huber loss until some small error, it is l_2 and then l_1
  - More robust and differentiable



Loss functions for Binary {-1,1} Classification

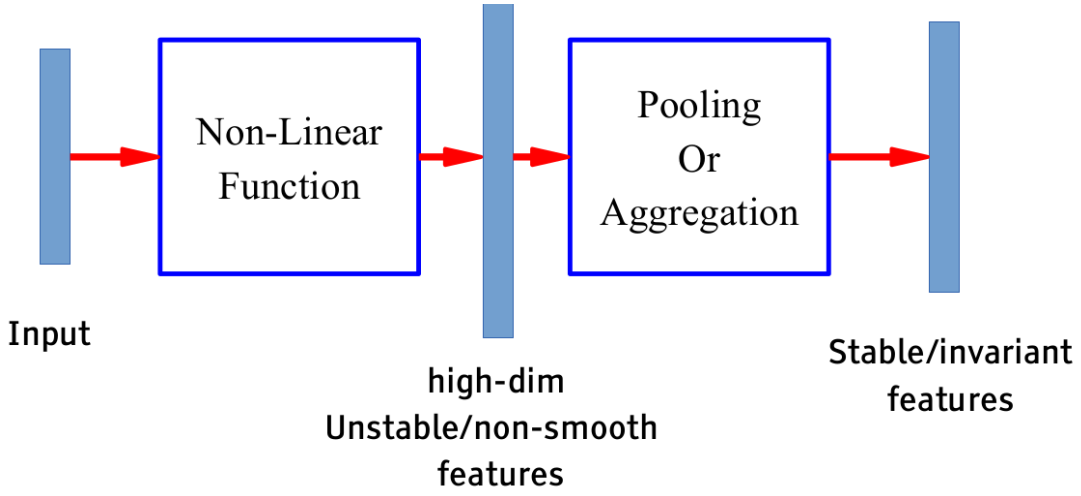
$$L_{logistic}(\Theta) = \frac{1}{\ln 2}\sum_{i=1}^{n}\ln(1+\exp(-y_i\hat{y}_i))$$

$$L_{Bernoulli}(\Theta) = \sum_{i=1}^{n}\ln(1+\exp(-2y_i\hat{y}_i));$$

$$L_{cross-entropy}(\Theta) = -\frac{1}{2}\sum_{i=1}^{n}[(1+y_i)\ln(\hat{y}_i)+(1-y_i)\ln(1-\hat{y}_i)]$$

Basic Idea for Invariant Feature Learning

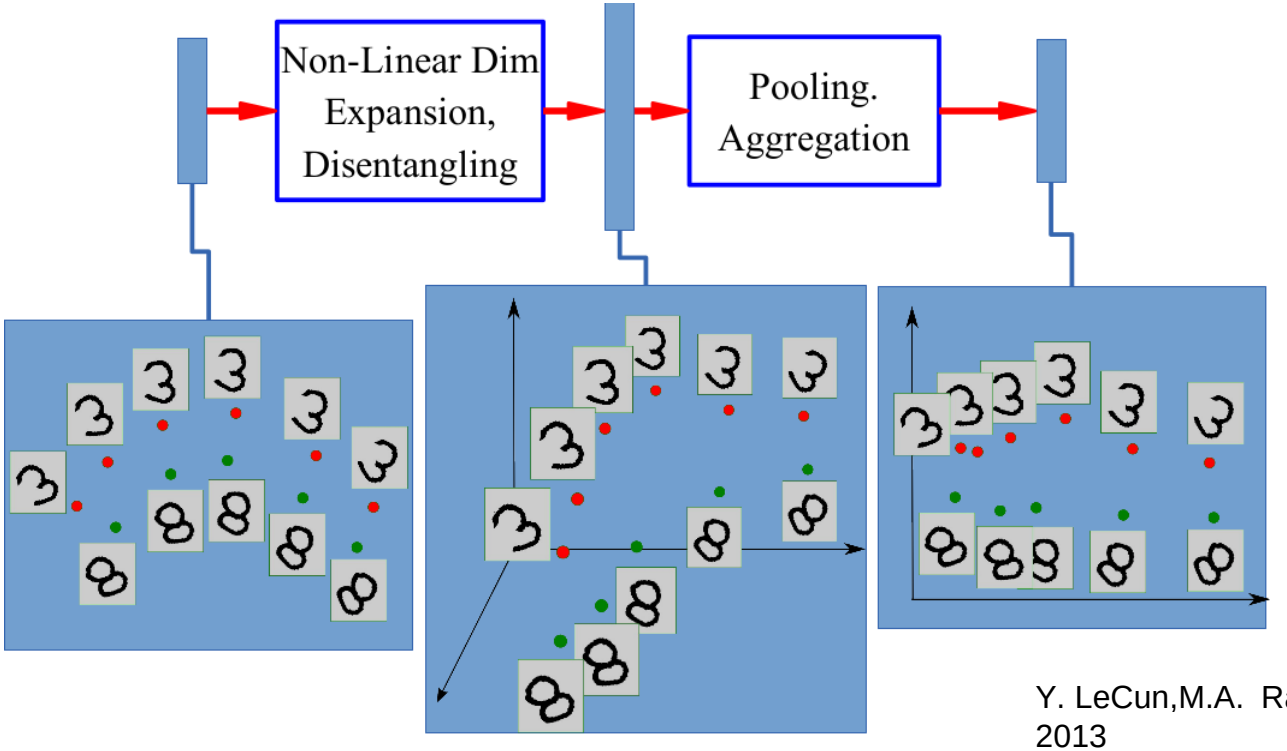
Embed the input non-linearly into a high(er) dimensional space
In the new space, things that were non separable may become separable
Pool regions of the new space together
Bringing together things that are semantically similar. Like pooling.



Y. LeCun,M.A. Ranzato. ICML-2013

Non-Linear Expansion → Pooling

Entangled data manifolds



Y. LeCun,M.A. Ranzato. ICML-2013

Similar to SVM models

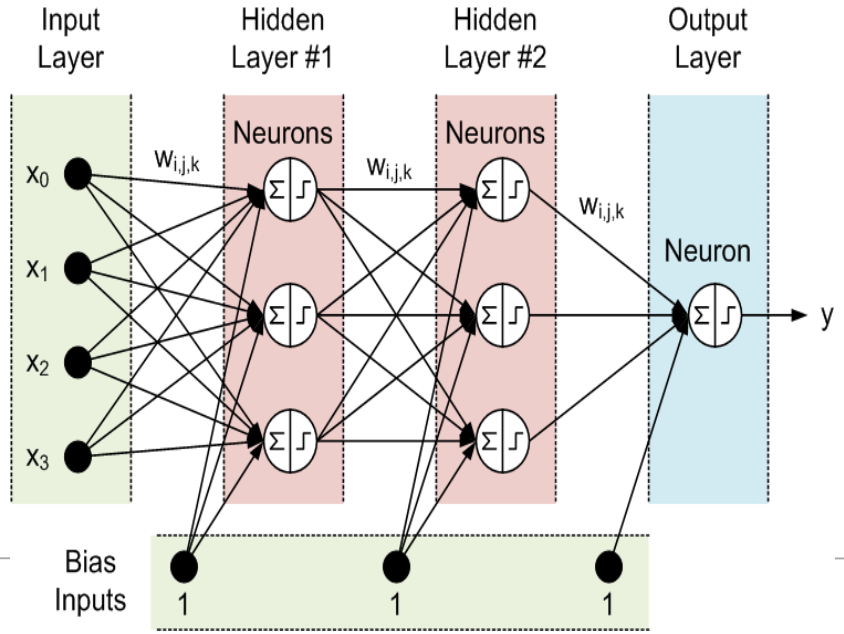
Neural Networks come in various shapes & sizes

Multi Layer perceptrons
- Networks made up of sigmoid or ReLu neurons (NOT perceptrons)

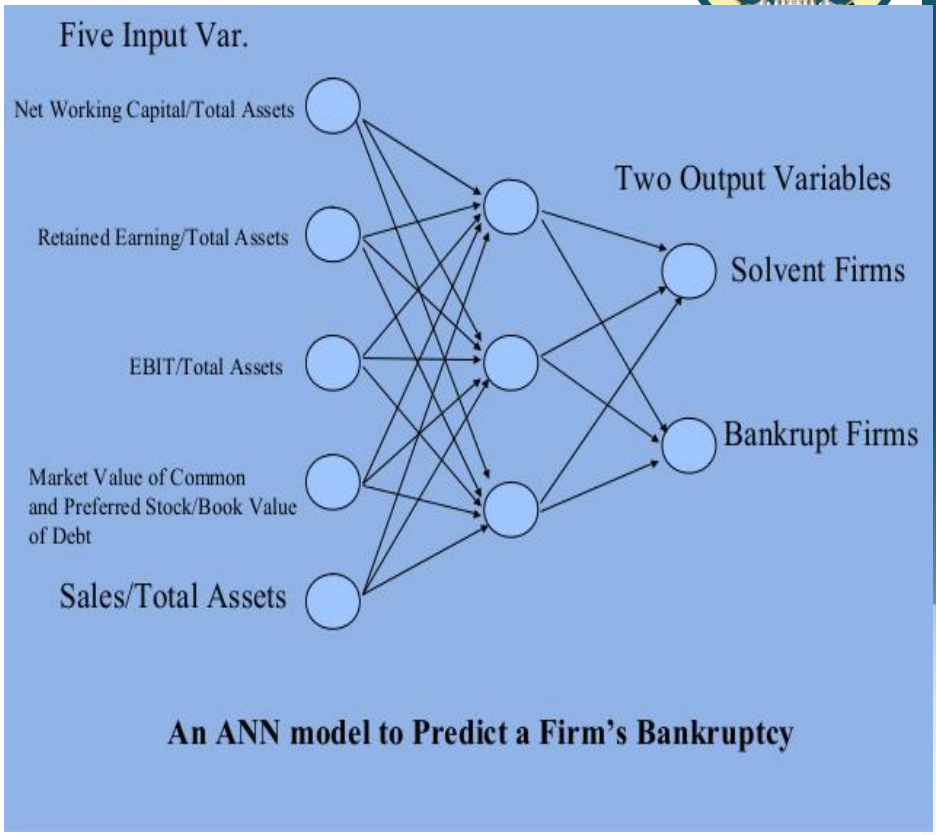
ANNs: characterized by parameters and hyperparameters

Parameters
- Weights & bias
- Learning / Training: Find parameters which minimize cost function

Hyper-parameters
- Network structure
- Activation functions
- Number of hidden layers
- Number of neurons
- Cost function



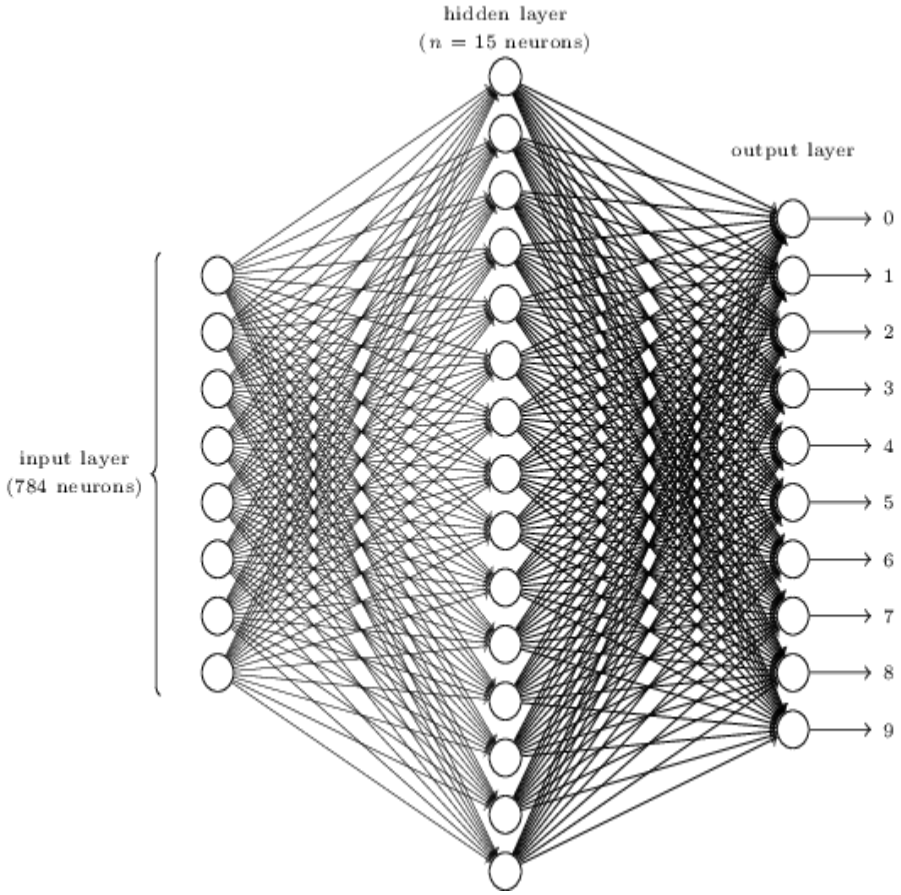
NN e.g. 1



- Hyper-parameters
  - Network structure
  - Activation functions
  - Number of hidden layers
  - Number of neurons in the hidden layer(s)
  - Cost function
- Parameters
  - Weights & bias
  - Learning / Training: Find parameters which minimize cost function

NN e.g. 2

- Digit Recognition
  - Each digit image 28 x 28 (=784) pixels
- Hyper-parameters
  - Network structure
  - Activation functions
  - Number of hidden layers
  - Number of neurons in the hidden layer(s)
  - Cost function
- Parameters
  - Weights & bias
  - Learning / Training: Find parameters which minimize cost function



Regularization functions

Ridge

Simplest is whose sum of squares is the least

$$\min_{w}\{\frac{1}{n}(\hat{X}w - \hat{Y})^2 + \lambda\|w\|_2^2\}$$

Lasso

Simplest is whose absolute sum is the least

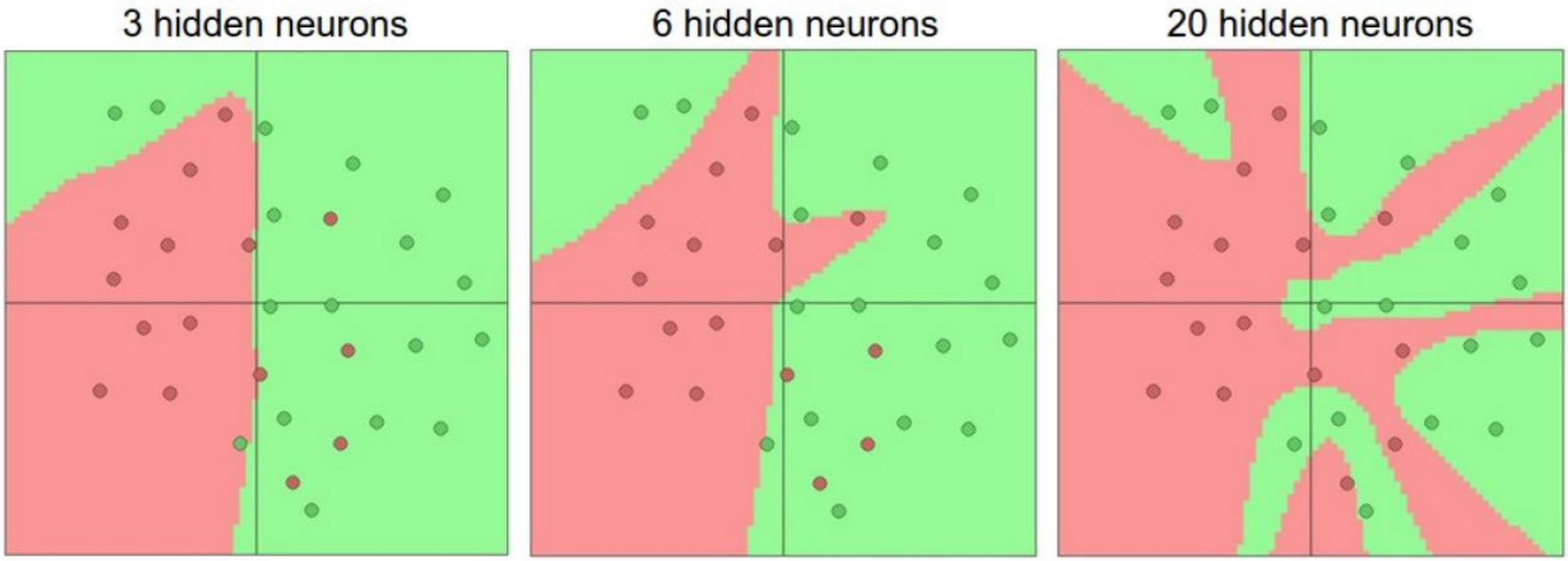
$$\min_{w\in\mathbb{R}^p}\{\frac{1}{n}\|\hat{X}w - \hat{Y}\|^2 + \lambda\|w\|_1\}$$

ElasticNet

$$\min_{w\in\mathbb{R}^p}\{\frac{1}{n}\|\hat{X}w - \hat{Y}\|^2 + \lambda(\alpha\|w\|_1 + (1-\alpha)\|w\|_2^2), \alpha \in [0,1]\}$$

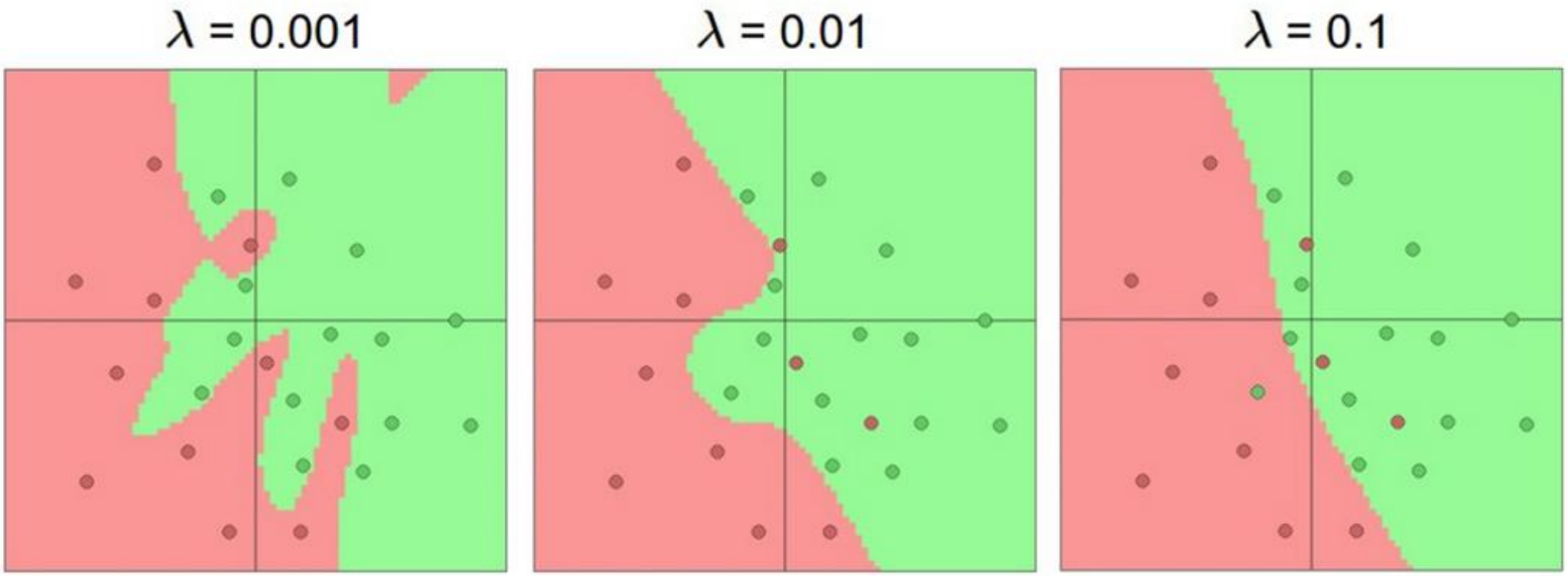
High lambda gives simplest models; At infinity, we have no model

Architecture



http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Regularization is the key



20 node neural net

Lets play with some toy datasets to test the limitations of perceptron and how to over come them with MLP.

http://playground.tensorflow.org

We so far talked of weight coefficients (transformation matrix) at each layer and how they solve the non-linear problem.

Big question is how do we get these weights?

How to train a network?

- Learning is changing weights
- In the very simple cases
  - **Start random**
  - **If** the output is correct **then** do nothing.
  - **If** the output is too high, decrease the weights attached to high inputs
  - **If** the output is too low, increase the weights attached to high inputs

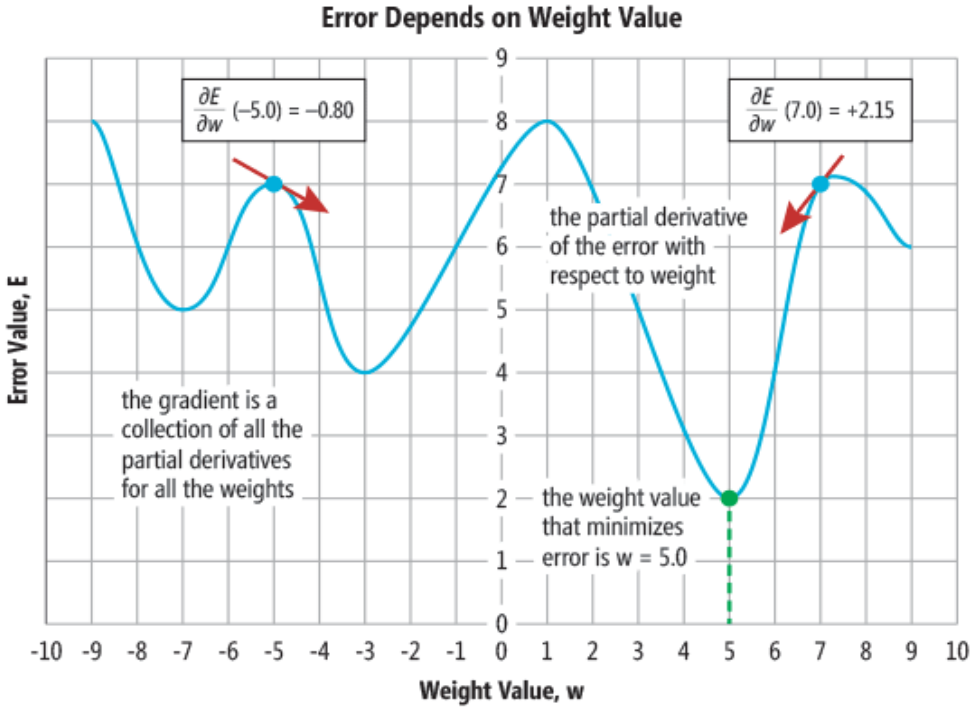
How to train a network?

- Let (x_i, y_i) be input label pairs from the dataset.
- Given an input, $\tilde{y}_i = f(x_i, W)$ is the network output.
  Where W is the set of network parameters.
- The cost function is defined as $E = \sum_i ||y_i - \tilde{y}_i||_2^2$ (SSE)

Our task is to minimize E by updating W. *How can we do that?*

Finding the minima: Gradient Descent

Gradient descent?



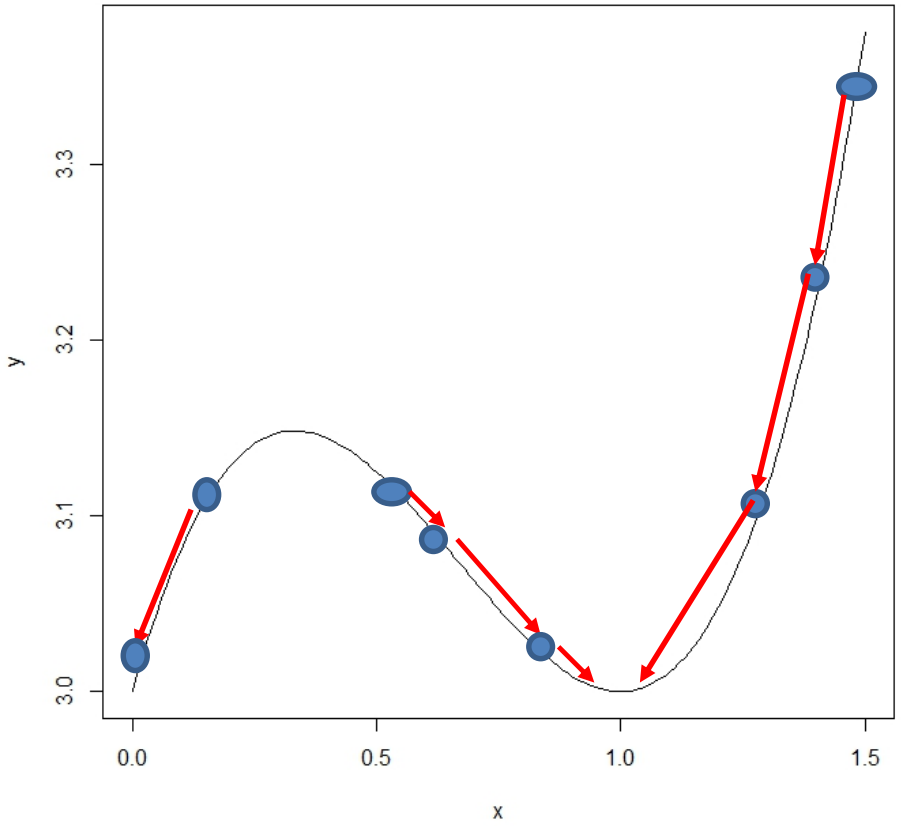
Given function E with parameters W

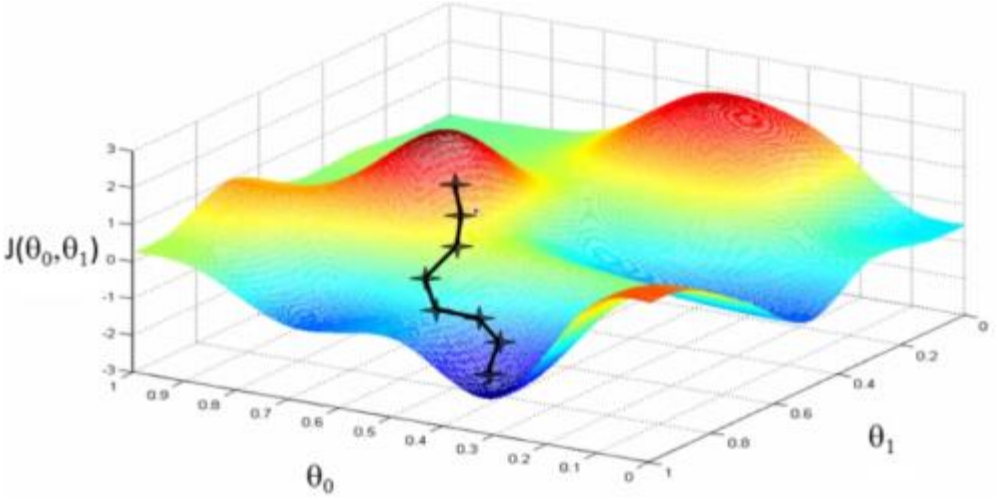
- E can be minimized by moving in direction opposite to gradient of E w.r.t W.

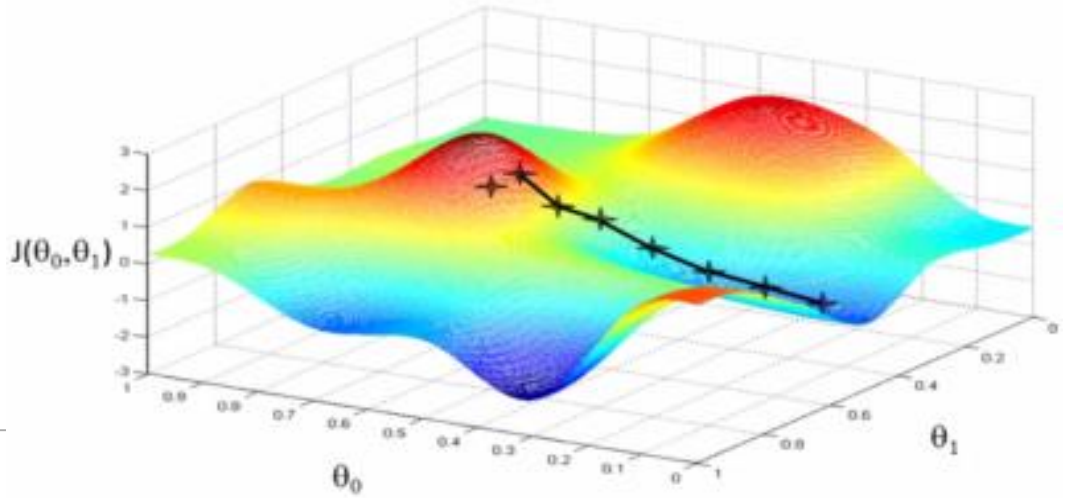
$$\Delta W = -\alpha \nabla E$$

In practice, we scale down the gradient by a parameter, α, called learning rate. Learning rate is between 0 and 1

GD and local minima







MLP: Forward propagation

Moving from input towards the output, layer by layer

Layer1: $$h_1 = f_1(x) = \sigma(w_1 x + b_1)$$

Layer2: $$h_2 = f_2(h_1) = \sigma(w_2 h_1 + b_2)$$

Layer3: $$h_3 = f_3(h_2) = \sigma(w_3 h_2 + b_3)$$

.

.

.

LayerN: $$\text{output} = f_N(h_{N-1}) = \sigma(w_N h_{N-1} + b_N)$$

$$\text{output} = f_N(f_{N-1}(f_{N-2}(\ldots(f_1(x)) = F(x_t; W)$$

Updating the weights by Error Back-Propagation

Error measure:

$$E = \sum_{t=1}^{N} (F(x_t; W) - y_t)^2$$

Rule for changing the synaptic weights:

$$\Delta w_i^j = -\alpha \cdot \frac{\partial E}{\partial w_i^j}(W)$$

$$w_i^{j,new} = w_i^j + \Delta w_i^j$$

α is the learning parameter (usually a constant)

ANN learning

Learning is changing weights

In the very simple cases

Start random

If the output is correct **then** do nothing.

If the output is too high, decrease the weights attached to high inputs

If the output is too low, increase the weights attached to high inputs

$$w_{t+1} = w_t - \alpha \frac{\partial E}{\partial W}$$

The method of computing the sensitivity of the error to the change in weights, $\frac{\partial E}{\partial W}$, is called **Back-propagation**

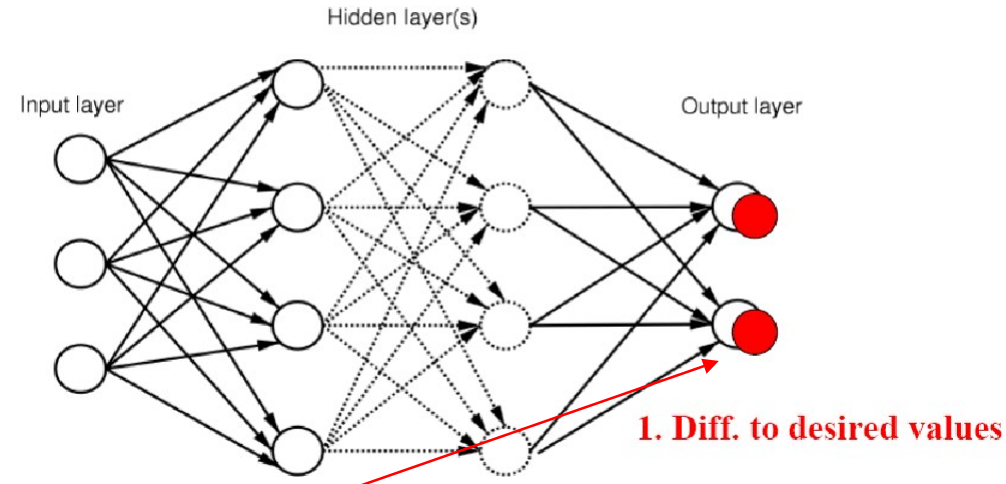
The term is an abbreviation for “Backward propagation of Errors”

Popularized by a paper by Geoffery Hinton

The led to a renaissance in the area of Neural Networks

Visualization of Backpropagation learning

Backpropagation Learning



Backpropagation Learning

$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

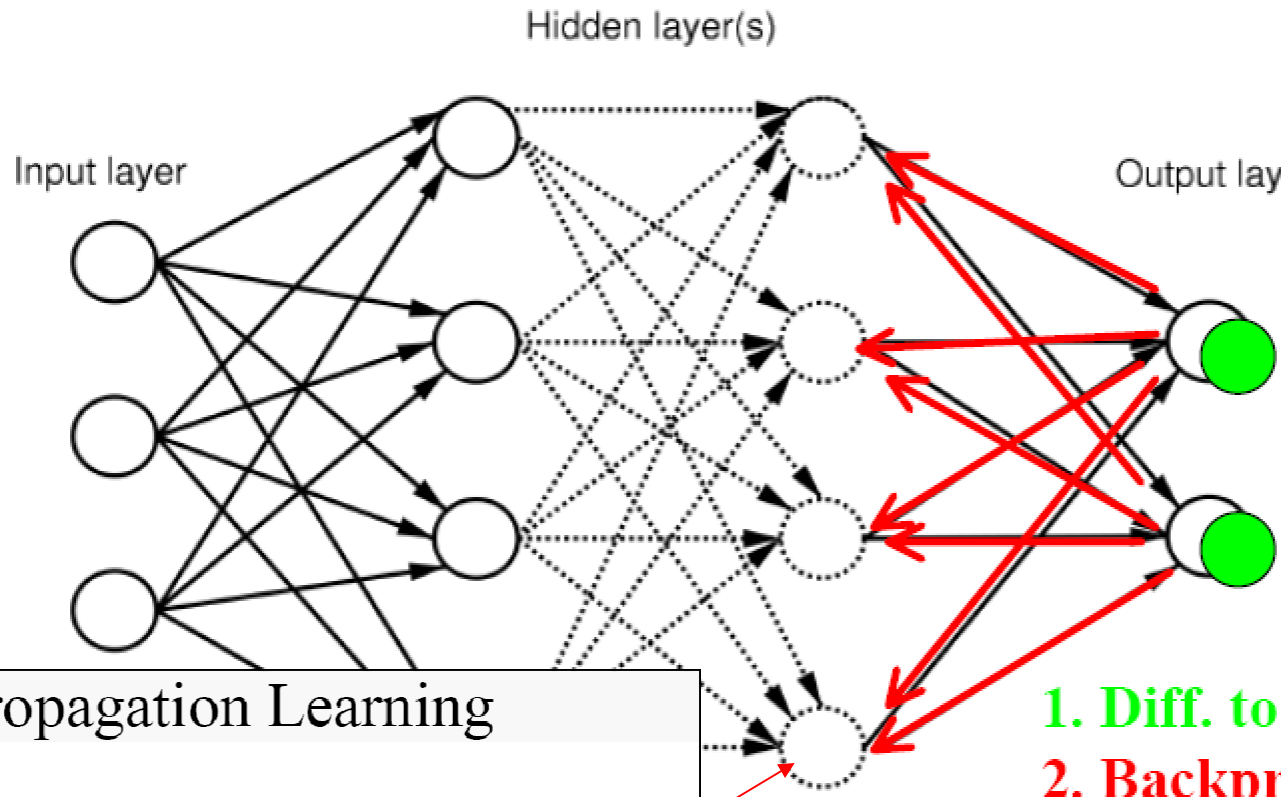
$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

Backprop output layer

From:
https://www.slideshare.net/keepurcalm/backpropagation-in-neural-networks

Backpropagation Learning



1. Diff. to desired values

2. Backprop output layer

Backpropagation Learning

$$E_{out\,i} = d_{out\,i} - out_i$$

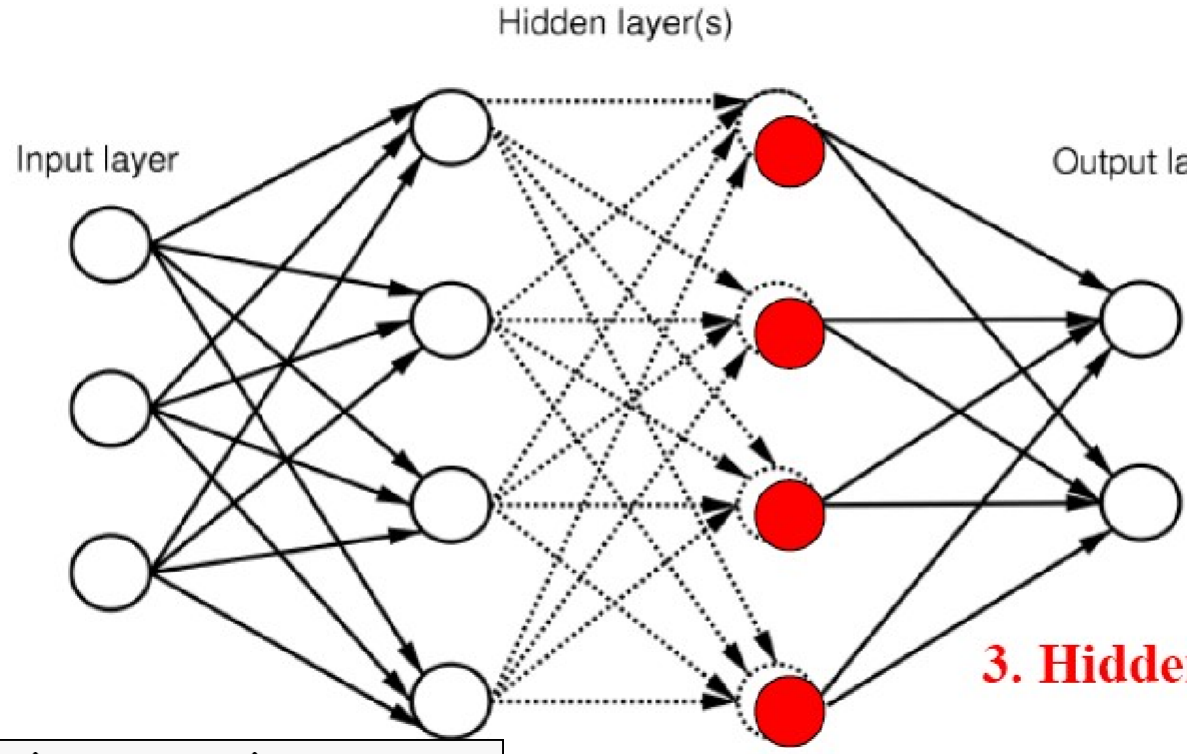
$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\,i}^2$$

$$E_{hid\,i} = \sum_{k=1}^{num(n_{out})} E_{out\,k} \cdot w_{out\,i,k}$$

$$diff_{hid\,i} = E_{hid\,i} \cdot (1 - o(n_{hid\,i})) \cdot o(n_{hid\,i})$$

From:
https://www.slideshare.net/keepurcalm/backpropagation-in-neural-networks

Backpropagation Learning



3. Hidden error values

Backpropagation Learning

$$E_{out\ i} = d_{out\ i} - out_i$$

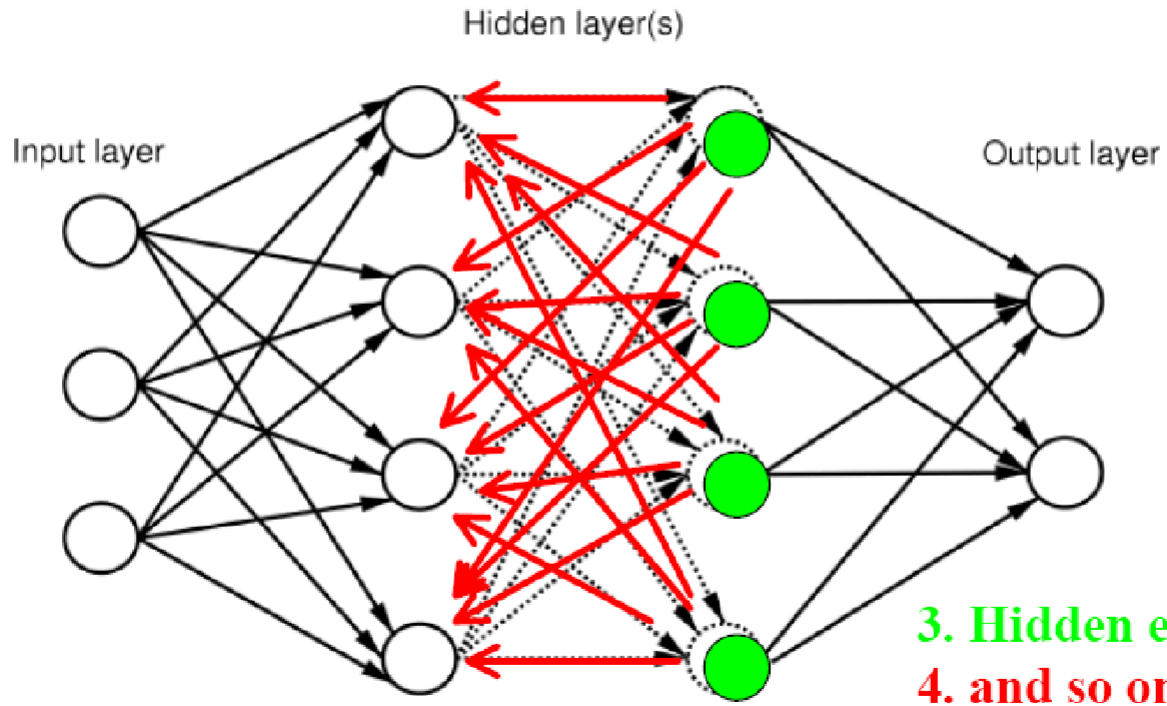
$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

From:
https://www.slideshare.net/keepurcalm/backpropagation-in-neural-networks

Backpropagation Learning



Backpropagation Learning

$$E_{out\ i} = d_{out\ i} - out_i$$

$$E_{total} = \sum_{i=0}^{num(n_{out})} E_{out\ i}^2$$

$$E_{hid\ i} = \sum_{k=1}^{num(n_{out})} E_{out\ k} \cdot w_{out\ i,k}$$

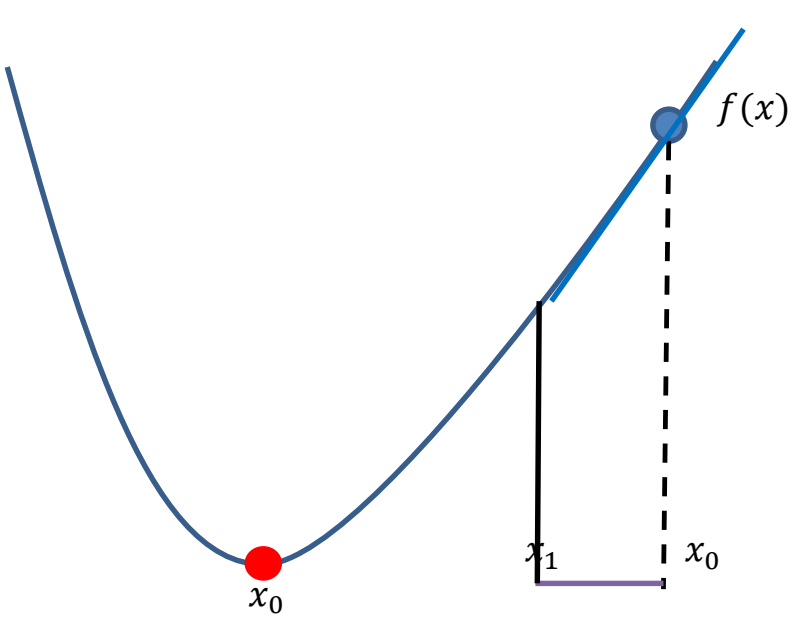
$$diff_{hid\ i} = E_{hid\ i} \cdot (1 - o(n_{hid\ i})) \cdot o(n_{hid\ i})$$

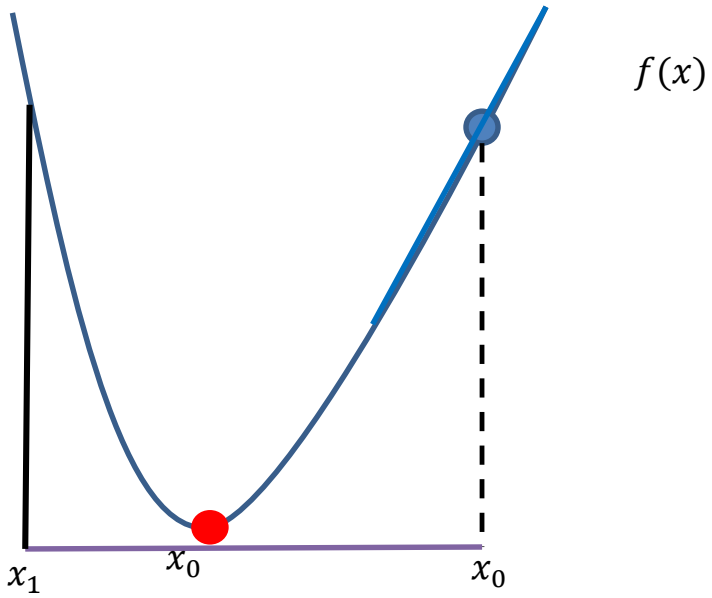
From:
https://www.slideshare.net/keepurcalm/backpropagation-in-neural-networks

Learning rate

Learning rate α, is between 0 and 1

The choice of learning rate can be critical to how soon the algorithm converges.





Excel

Multiple methods of updating weights

Online	Batch	Mini-Batch
Show an input…adjust weights, show another and adjust weights…,once the input is all over, start with row 1 if needed	Show an input, compute the adjustment needed to the weights and store it. Show the second input (with original weights) and continue the process. Update once all inputs are shown	Pick a small random sample of inputs. Perform batch update. Then pick another set of sample inputs randomly…

More on min-batch gradient descent

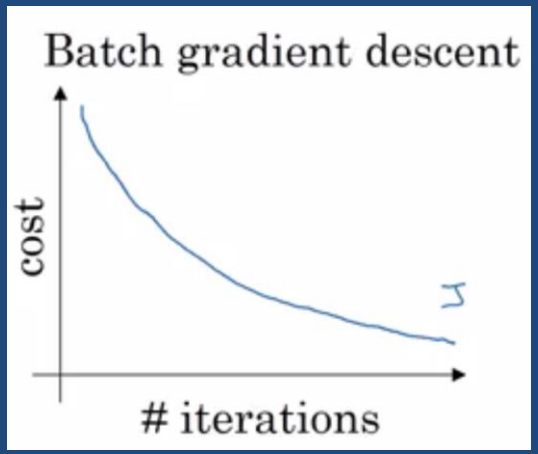
Let us say we have 500K samples and let us say, each mini-batch contains 512 samples. We make ~1000 mini batches

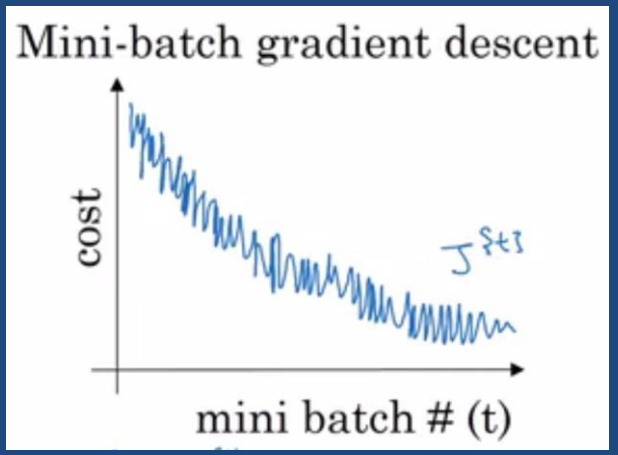
We construct a matrix of 512 samples and forward prop. Make 512 predictions. Then find average cost over 512 samples. Then update the weights once.

An epoch is doing all 1000 mini batches once. We do multiple epochs before convergence

If mini batch is 1, it is stochastic gradient descent. If it is 500K you have gradient descent

For less than 2000 samples, go for batch. For large data sets, the mini-batch is 64-512 (a power of 2) is good.





NN Training: Local versus global

- Gradient descent is greedy
- Multiple initiations and selection of the best is the option
- No way to theoretically guarantee convergence to global optimum.
- But, generally convergence to a local optimum which is very close to global.
- Therefore state-of-the-art results :)

NN Training: Additional Considerations

- Variable selection
  - Input layer will need to have as many nodes as there are inputs
  - Dependence, correlation, dimensionality reduction etc.
  -
- Data pre-processing

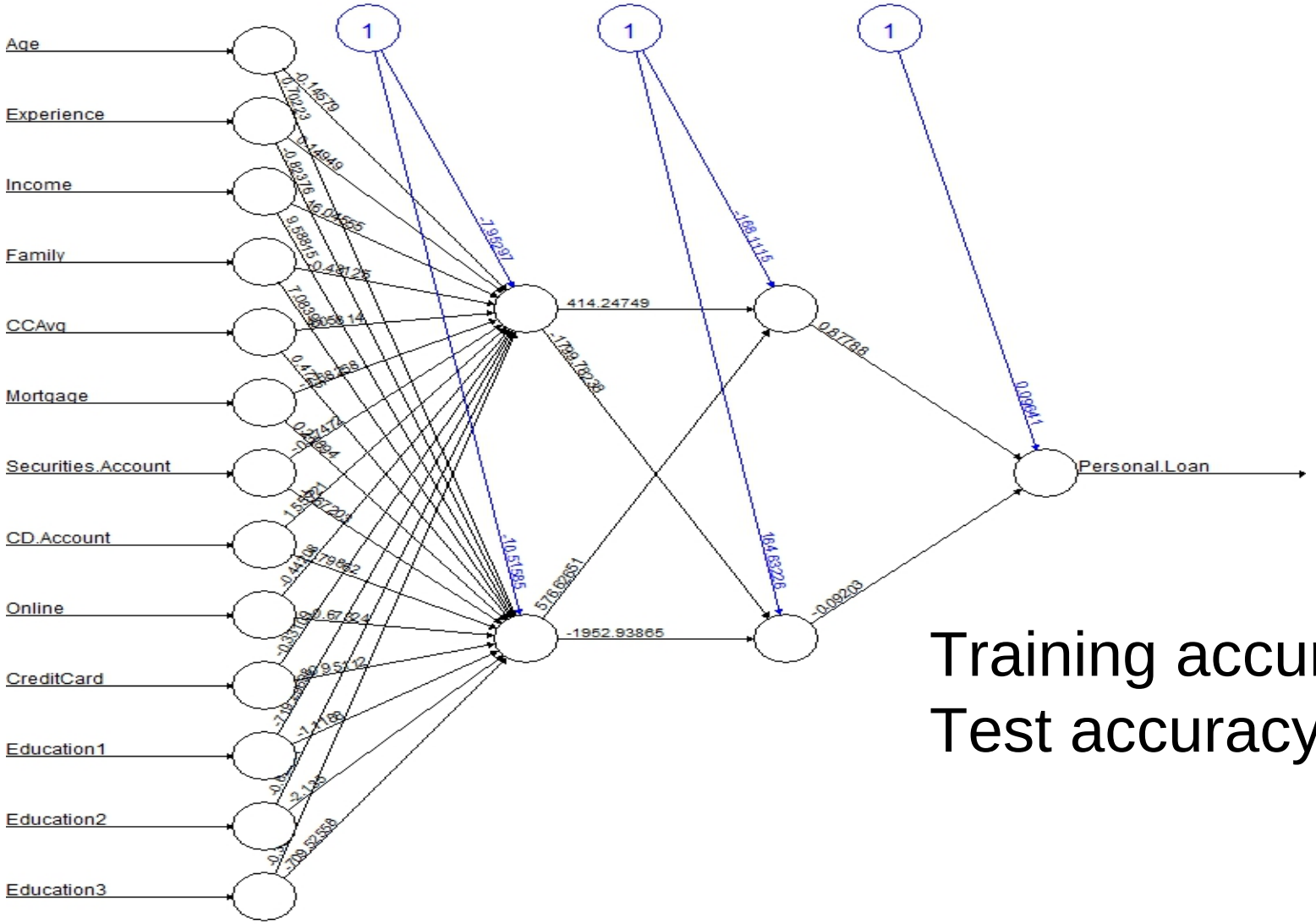
- Data separation into training, validation and test models

Network Topology:

- 90%: One hidden layer
- 10%: Two hidden layers

- Rarely more than that! It might lead to overfitting!

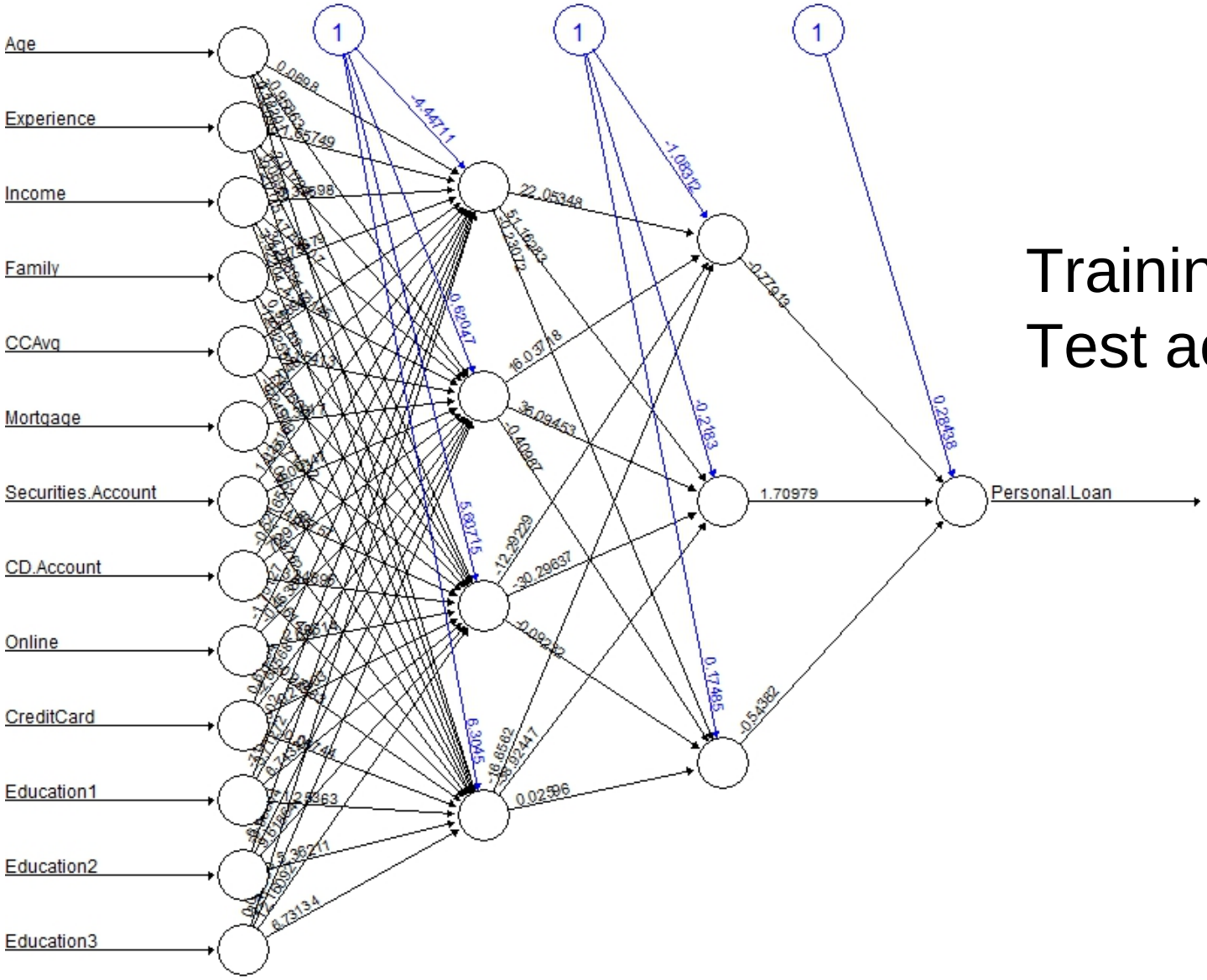
Topology: Example 2x2 ANN



Training accuracy: 97.36%

Test accuracy: 89.38%

Topology: Example 4x3 ANN



Training accuracy: 100%

Test accuracy: 86.24%

Inventory Management: Predicting Backorders

sku - Random ID for the product
national_inv - Current inventory level for the part
lead_time - Transit time for product (if available)
in_transit_qty - Amount of product in transit from source
forecast_3_month - Forecast sales for the next 3 months
forecast_6_month - Forecast sales for the next 6 months
forecast_9_month - Forecast sales for the next 9 months
sales_1_month - Sales quantity for the prior 1 month time period
sales_3_month - Sales quantity for the prior 3 month time period
sales_6_month - Sales quantity for the prior 6 month time period
sales_9_month - Sales quantity for the prior 9 month time period

min_bank - Minimum recommend amount to stock
potential_issue - Source issue for part identified
pieces_past_due - Parts overdue from source
perf_6_month_avg - Source performance for prior 6 month period
perf_12_month_avg - Source performance for prior 12 month period
local_bo_qty - Amount of stock orders overdue
deck_risk - Part risk flag
oe_constraint - Part risk flag
ppap_risk - Part risk flag
stop_auto_buy - Part risk flag
rev_stop - Part risk flag

went_on_backorder - Product actually went on backorder. This is the target value.

	sku	national_inv	lead_time	in_transit_qty	forecast_3_month	forecast_6_month	forecast_9_month	sales_1_month	sales_3_month	sales_6_month	...
0	1888279	117	NaN	0	0	0	0	0	0	15	...
1	1870557	7	2.0	0	0	0	0	0	0	0	...
2	1475481	258	15.0	10	10	77	184	46	132	256	...
3	1758220	46	2.0	0	0	0	0	1	2	6	...
4	1360312	2	2.0	0	4	6	10	2	2	5	...
5	3002608	297	12.0	0	0	0	0	5	6	44	...
6	1707693	3285	8.0	0	0	0	0	18	81	162	...
7	2066128	6	2.0	0	0	3	9	1	5	8	...
8	3174990	99	8.0	0	46	169	225	2	22	176	...
9	3021709	52	2.0	0	0	0	0	0	0	0	...

```
import keras
from keras import regularizers, optimizers
from keras.layers import Dense
from keras.models import Sequential
from keras.callbacks import EarlyStopping, ReduceLROnPlateau

## Defining my optimizer - Its just a refined form of SGD
## ref - https://keras.io/optimizers/
adam = keras.optimizers.Adam()
```

# of input features

First hidden layer with 10 neurons

2nd hidden layer with 18 neurons

```
model_sig = Sequential()
model_sig.add(Dense(10, input_shape=(21,), activation='relu', kernel_initializer='normal'))
model_sig.add(Dense(18, activation='relu', kernel_initializer='normal'))
model_sig.add(Dense(1, activation='sigmoid', kernel_initializer='normal', kernel_regularizer=regularizers.l2(0.04)))
```

```
model_sig.compile(loss='binary_crossentropy', optimizer=adam,metrics=['accuracy'])
model_sig.fit(X_train, y_train, epochs=30, batch_size=100)
```

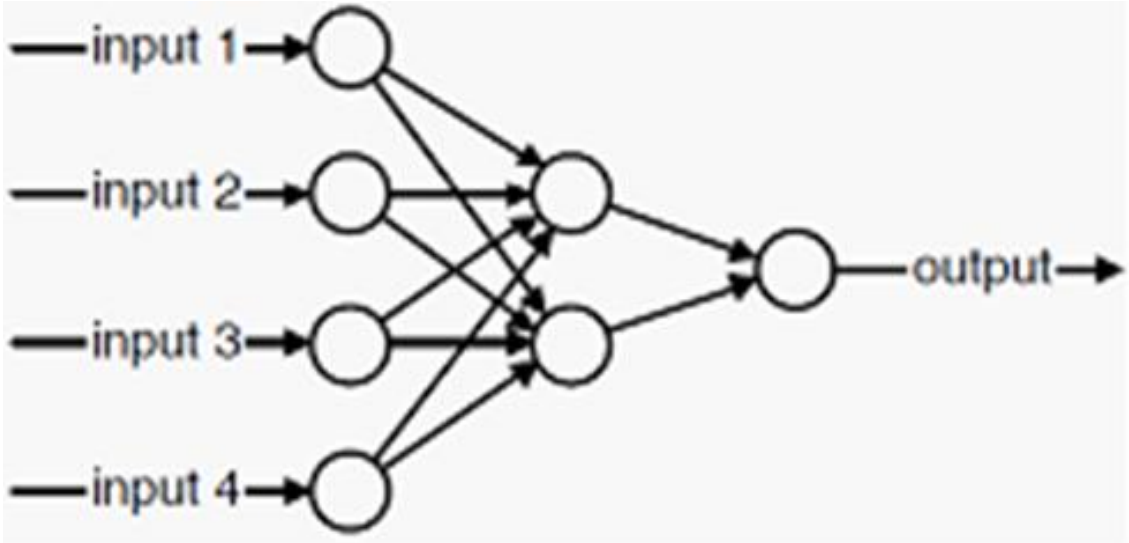
Output layer with 1 output

```
Confusion Matrix
[[13190   723]
 [  879  2383]]
TNR:  0.9480342126069145
TPR:  0.7305334150827713
Accuracy:  0.9067248908296943
```

Topology: How many nodes do we use?

- One Output node for each class
- 0.5 to 3 times the input neurons
- A commonly recommended rule - Geometric pyramid rule: Where input has m nodes and output has n nodes, the hidden layer should have sqrt(mXn)

How many weights are computed?



$$H * (I + O) + H + O$$

5 input features and 10 units in the hidden network and 1 output, then there are 71 weights in the network.

Topology: Other Considerations

- The weights should be 1/100th of the amount of training data set *
- A NN with 4 input, 2 hidden and 2 outputs require how much of data?
  - H * (I + O) + H + O
  = 2*(4+2) + 2 + 2
  = 16

  Approx 1600 samples

* Regularization allows us to relax these guidelines. With regularization, you can even have more weights than training examples

Topology: Baum-Haussler

- Baum-Haussler rule states that

$$N_{hidden} \leq \frac{N_{train} E_{tolerance}}{N_{input} N_{output}}$$

N_{hidden} is the number of hidden nodes,
N_{train} is the number of training patterns,
$E_{tolerance}$ is the error,
N_{input} and N_{output} are the number of input and output nodes respectively.

- For 9 inputs, 4 outputs, if I have 10000 test data and allow only 0.01 error, how many nodes will I need

$$N_{hidden} \leq \frac{N_{train} E_{tolerance}}{N_{input} N_{output}} \qquad \frac{10000 * 0.01}{9 * 4} = 2.78 \geq 2$$

Practical Guidelines

How many hidden layers do we need?

Most structured-data problems can be handled with at-most 2 layers. Rare are problems where higher layers can be justified.

How many neurons/nodes do I use?

The previously mentioned “rules of thumb” are merely that – they are not absolute rules. Experimentation is need needed to figure out the number of neurons.
Number of neurons is a hyper-parameter and like all hyper-parameters best way to determine it, is through cross-validation

See ftp://ftp.sas.com/pub/neural/FAQ3.html#A_hu

Cross-Validation with Neural Networks

```
from sklearn.model_selection import GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
# Function to create model, required for KerasClassifier
def create_model(hidNodes):
    # create model
    model = Sequential()
    model.add(Dense(hidNodes, input_dim=41, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
# define the grid search parameters
batch_size = [64]
hidNodesGrid = [5, 10, 15, 20]
param_grid = dict(batch_size=batch_size, hidNodes=hidNodesGrid, epochs=[100])
model = KerasClassifier(build_fn=create_model, verbose=0)

grid = GridSearchCV(estimator=model, param_grid=param_grid,return_train_score=True)
grid_result = grid.fit(X, Y)
```

```
In [3]: print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
Best: 0.929506 using {'batch_size': 64, 'epochs': 100, 'hidNodes': 10}
```

Comparison to other ML methods

	Neural Networks	Decision Trees	SVM	Regression models
Structured data	Yes	Yes	Yes	Yes
Unstructured data (images, text, audio)	Yes	No	No	No
Feature extractor + classifier	Yes	Classifier	Classifier	Classifer
Number of training samples required	Large	Medium-low	Medium-low	Medium
Hyper-paramter tuning	Heavy	low	low	low

Understanding neural networks

Perceptron

- Provides direct interpretations based on the sign of the weight.
- If weight is positive, y increases with x else it decreases
- Magnitude of the weight indicates the impact of that variable

MLP

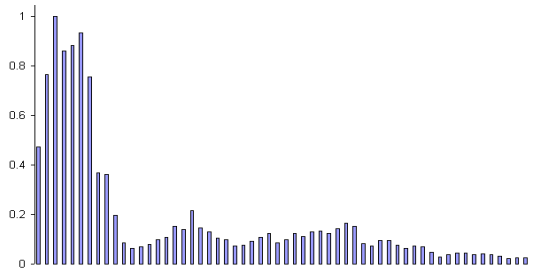
- Due to the high level of nonlinearity, once we have more than a single hidden layer interpreting relative importance of the variables becomes tricky

Example: Voice Recognition

Task: Learn to discriminate between two different voices saying “Hello”

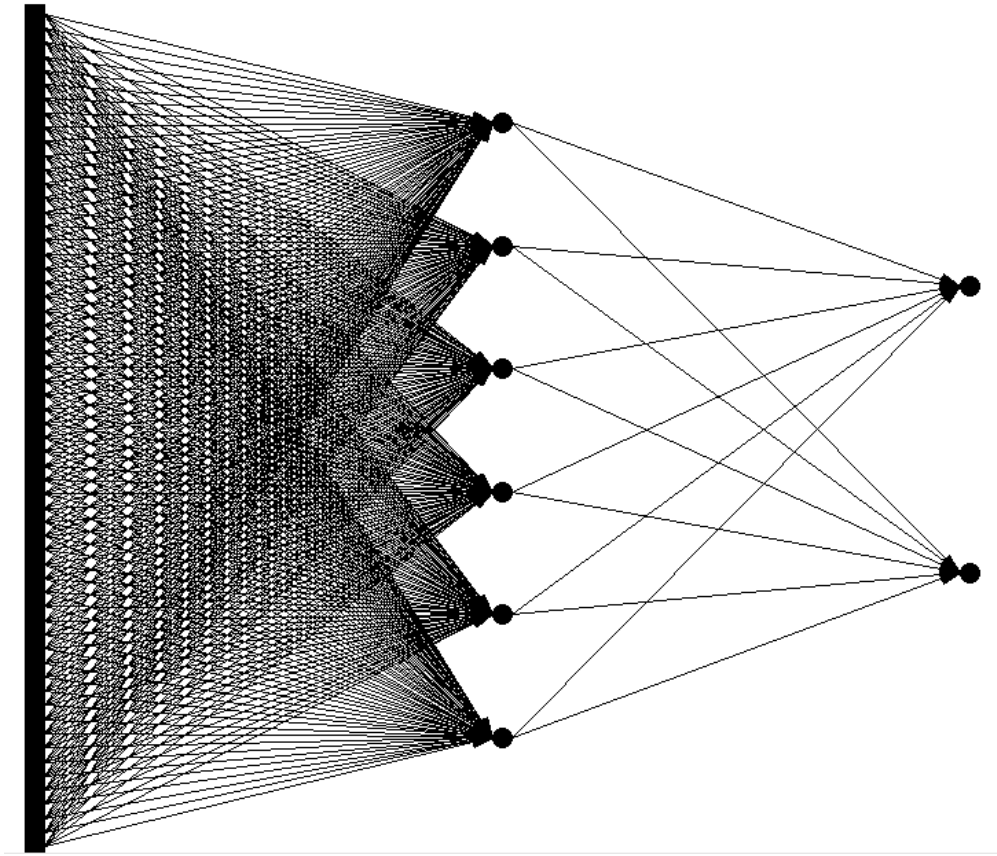
Data

- Sources
  - Steve Simpson
  - David Raubenheimer
- Format
  - Frequency distribution (60 bins)
  - Analogy: cochlea



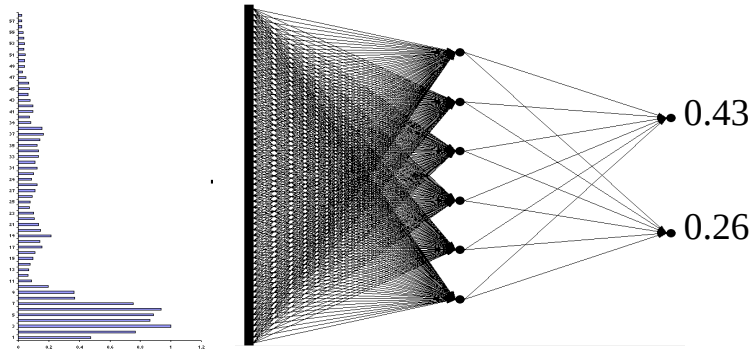
Network architecture

- Feed forward network
  - 60 input (one for each frequency bin)
  - 6 hidden
  - 2 output (0-1 for “Steve”, 1-0 for “David”)

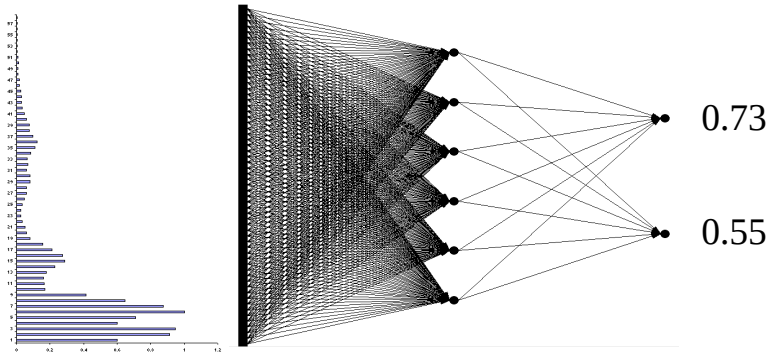


Untrained network

Steve

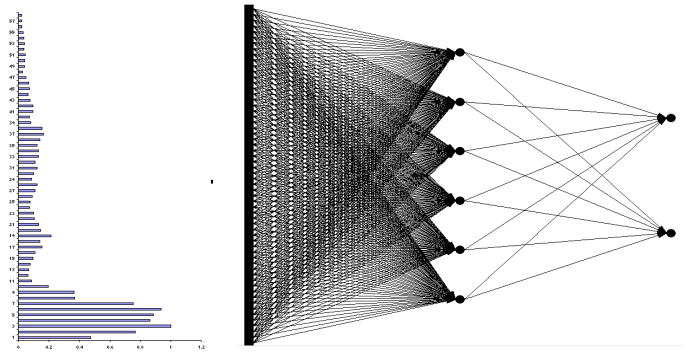


David

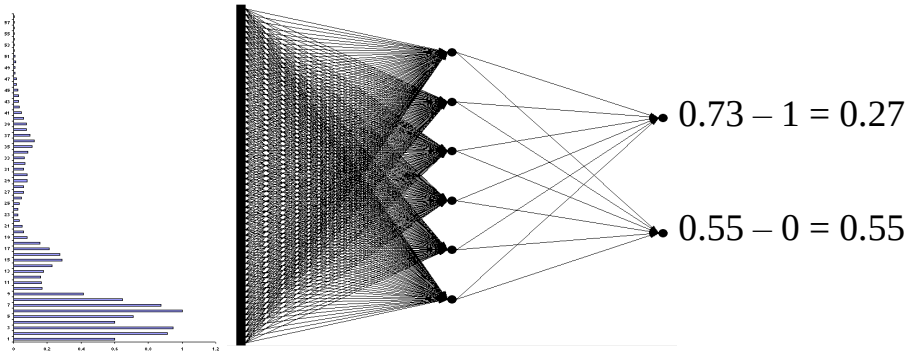


Calculate error

Steve



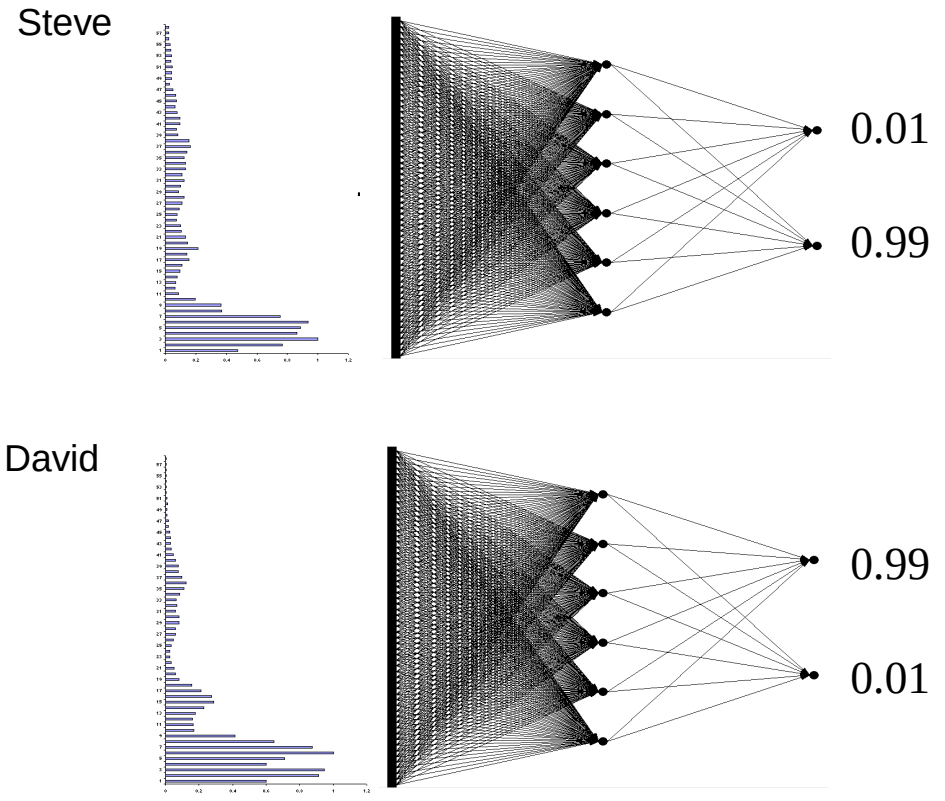
David



Performance of trained network

Discrimination accuracy between known “Hello”s
100%

Discrimination accuracy between new “Hello”s
100%



Beauty of neural networks.

- One model for approaching any machine learning problem.
- Basic functional element is a neuron which simply does *f(Wx+b)*
- A collection of neurons arranged in a specific manner define power ML architectures.
- Today we saw neurons arranged in sequential layer-by-layer architecture termed MLP.
- In future classes we will define new architectures for better addressing unstructured data problems.
- Completely data-driven approach.

History of Artificial Neural Networks

- A computational model for biological neural networks was created in 1943 by Warren McCulloch and Walter Pitts.
- One of the key breakthroughs was the backpropagation algorithm by Werbos in 1975.
- Parallel distributed computing, introduced by David E. Rumelhart and James McClelland.
- The neocognitron is a hierarchical, multilayered artificial neural network proposed by Kunihiko Fukushima in the 1980. (Basis of CNNs)
- Neural networks were overshadowed by the popularity of other ML methods in the 1990s.
- Some people like Yann LeCun, Geoffrey Hinton, Yoshua Bengio and others continued believing in artificial neural networks.

Revival of Artificial Neural Networks

Three main reasons:

- Better training approaches,
  thanks to **scientists**.
- Increased availability of training data,
  thanks to **digital media and internet**.
- computational power
  thanks to revolution in computing hardware: **Graphical processing units (GPU).**

HYDERABAD
2nd Floor, Jyothi Imperial, Vamsiram Builders, Old Mumbai Highway, Gachibowli, Hyderabad - 500 032
+91-9701685511 (Individuals)
+91-9618483483 (Corporates)

BENGALURU
L77, 15th Cross Road, 3rd Main Road, Sector 6, HSR Layout, Bengaluru – 560 102
+91-9502334561 (Individuals)
+91-9502799088 (Corporates)

Social Media

Web:	http://www.insofe.edu.in
Facebook:	https://www.facebook.com/insofe
Twitter:	https://twitter.com/Insofeedu
YouTube:	http://www.youtube.com/InsofeVideos
SlideShare:	http://www.slideshare.net/INSOFE
LinkedIn:	http://www.linkedin.com/company/international-school-of-engineering