GENERAL ASSEMBLY

NEURAL NETWORKS

LEARNING OBJECTIVES

- Understand various types of neural networks
- Applications of neural networks
- Apply a neural network model for regression
- Apply a neural network model for classification
- Understand the concepts behind perceptrons and hidden layers
- Understand back propagation

COURSE

PRE-WORK

PRE-WORK REVIEW

- Understand Logistic Regression and link functions
- Be familiar with training and testing classifiers and regressors

OPENING

ARTIFICIAL NEURAL NETWORKS

OPENING

- Neural networks were first studied in the 1940s (!) as a model of biological neural networks
- Many advances since then have improved the ability to train and apply neural networks
- Good for both classification and regression but difficult to interpret model behaviors
- Deep learning in the past few years has been highly successful for otherwise difficult problems

OPENING

- Today we will focus on types of neural networks and their applications, and skip some of the more technical details

- Specifically we'll skip training neural networks -- there are many methods in various situations and the details can be tedious (but not particularly difficult)

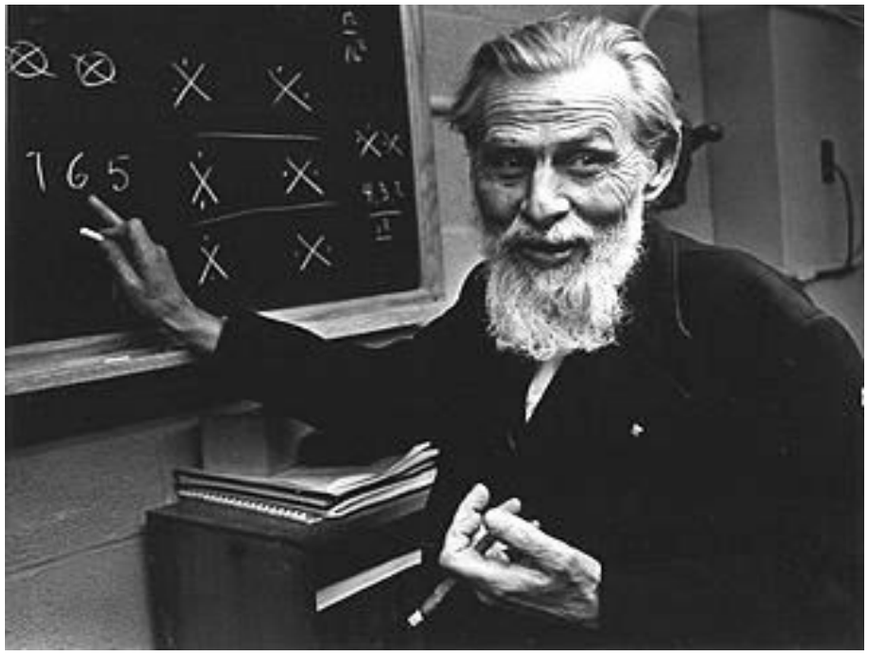
- Methods include backpropagation, gradient descent, and Hessian-free learning

HISTORY

NEURAL NETWORKS

At the age of 12, after reading Principia Mathematica he sent a letter to Bertrand Russell pointing out what he considered serious problems with the first half of the first volume

Walter Pitts



Warren McCulloch

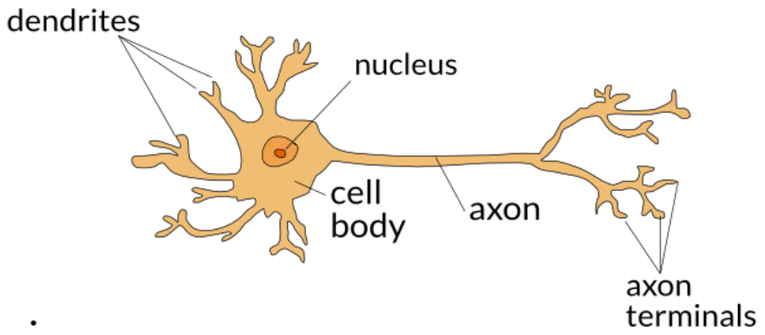
McCulloch was a confident, gray-eyed, wild-bearded, chain-smoking philosopher-poet who lived on whiskey and ice cream and never went to bed before 4 a.m.

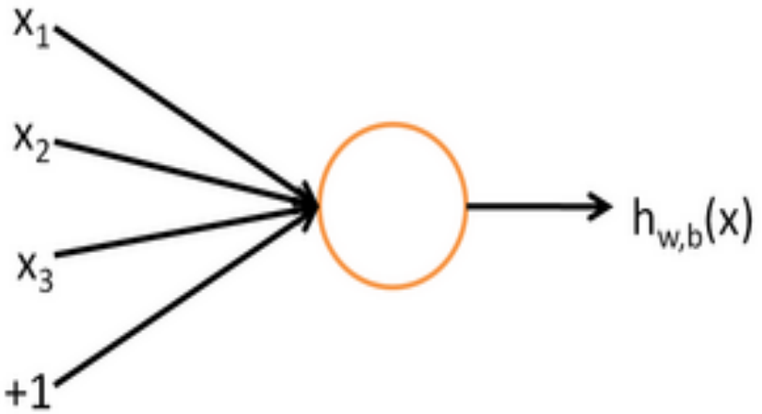
- Proposed that all of mathematics could be built from the ground up using basic logic. Their building block was the proposition – the simplest possible statement, either true or false.
- From there, they employed the fundamental operations of logic, like ‘and’, ‘or’, and ‘not’

http://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic

NEURAL NETWORKS

- Neurons – They act like a ‘true’ or ‘false’?
- They only fire after a minimum threshold has been reached
- A neuron’s signal is a proposition and they work like logic gates where they take in **multiple inputs** and produce a **single output**





http://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic

NEURAL NETWORKS

Example: Where will you go out for lunch?

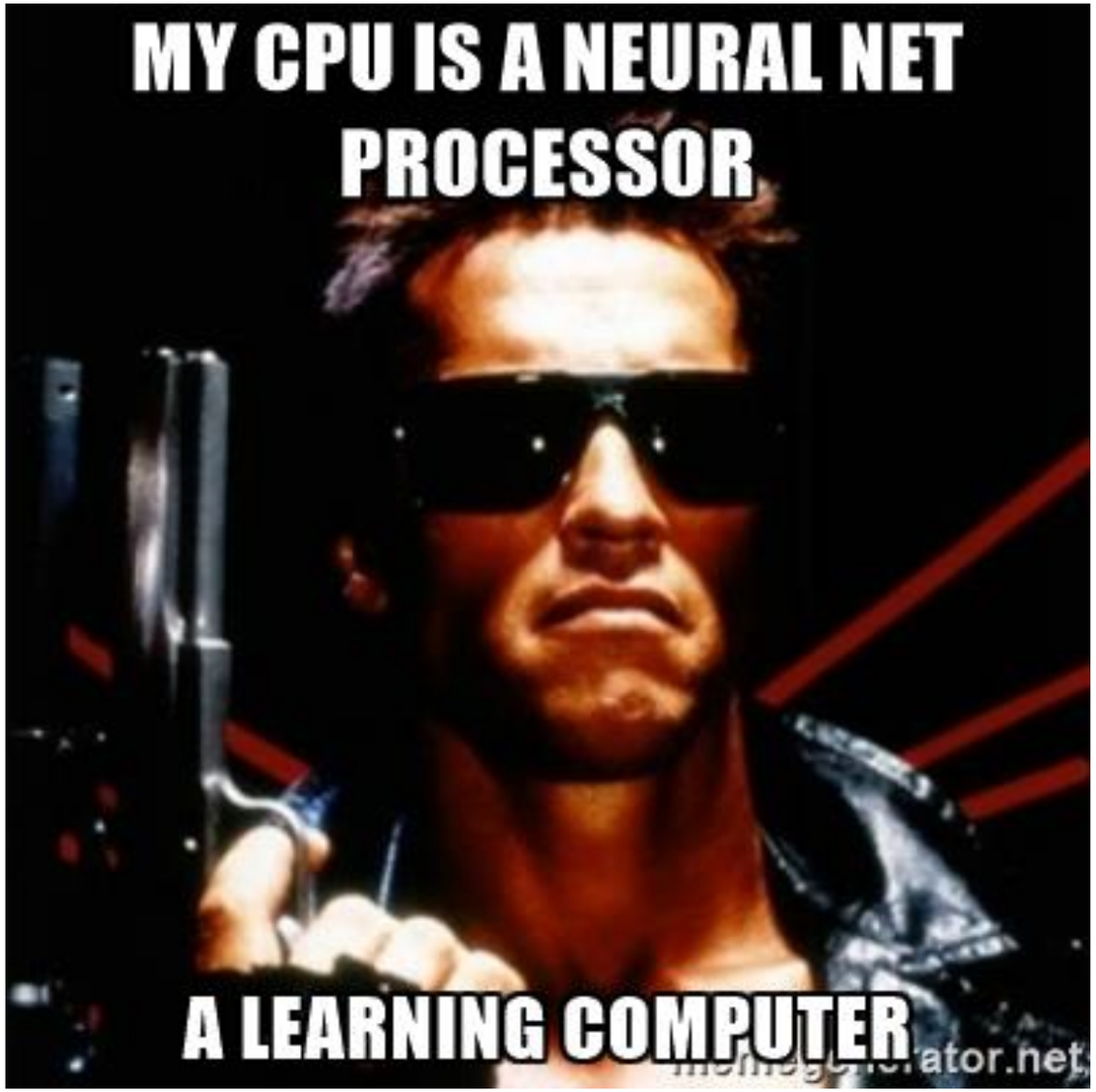
- **What are some of the inputs to making that decision?**
- **You will make a decision if and only if enough criteria are satisfied**

http://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic

NEURAL NETWORKS

- McCulloch became convinced that the brain was just such a machine – one which uses logic encoded in neural networks to compute

- **Neurons could be linked together by the rules of logic to build more complex chains of thought**



http://nautil.us/issue/21/information/the-man-who-tried-to-redeem-the-world-with-logic

NEURAL NETWORKS

NEURAL NETWORKS

- A Neural Network is a highly interconnected network of information-processing elements that mimics the connectivity and functioning of the human brain.

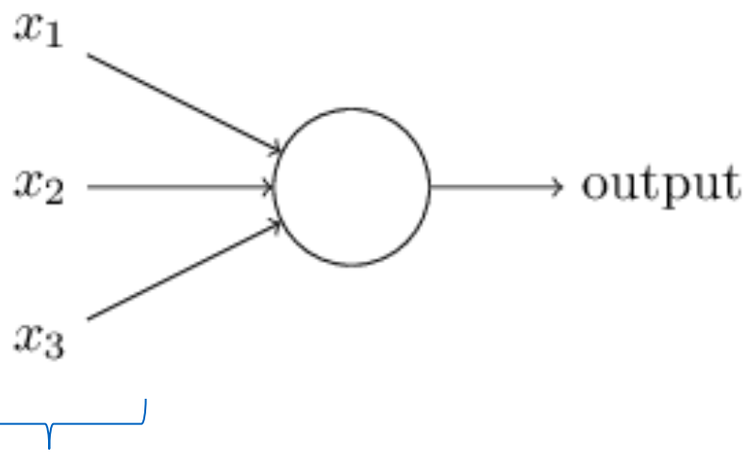
- Used for **supervised** and **unsupervised** learning algorithm

- Advances in **computer processing power** and **storage (database)** are finally allowing neural nets to improve solutions for complex problems such as speech recognition, computer vision, and Natural Language Processing (NLP)

NEURAL NETWORKS

- Neural networks consist of nodes, sometimes called a **perceptron** – they take several inputs (x1, x2, …) and produce a single output

- Think of perceptrons as a method for weighing evidence to make decisions



Input Variables

Activation Function

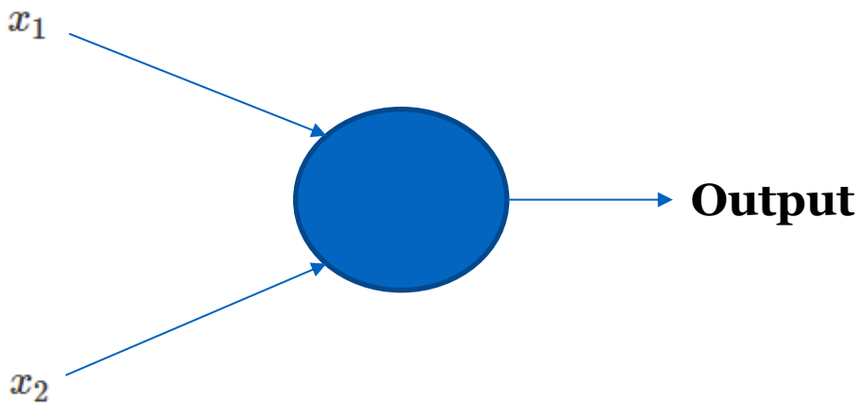
$$\text{output} = \begin{cases} 0 & \text{if } f(x_1, x_2, x_3) \leq threshold \\ 1 & \text{if } f(x_1, x_2, x_3) > threshold \end{cases}$$

http://neuralnetworksanddeeplearning.com/chap1.html

NEURAL NETWORKS

Example: let's say we have two input variables and the output is based on the following function (*also known as an activation function*):

$$\text{output} = \begin{cases} 0 & \text{if} \quad f(x_1, x_2) = -2x_1 - 2x_2 + 3 \geq 0 \\ 1 & \text{if} \quad f(x_1, x_2) = -2x_1 - 2x_2 + 3 < 0 \end{cases}$$



Suppose x1 and x2 are binary. Let's look at the different possible outputs:

- x1 = **1**, x2 = **1**

Output = 1

- x1 = **1**, x2 = **0**

Output = 0

- x1 = **0**, x2 = **0**

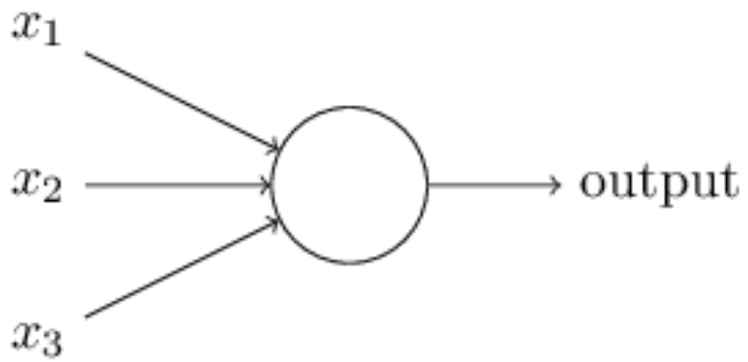
Output = 0

- x1 = **0**, x2 = **1**

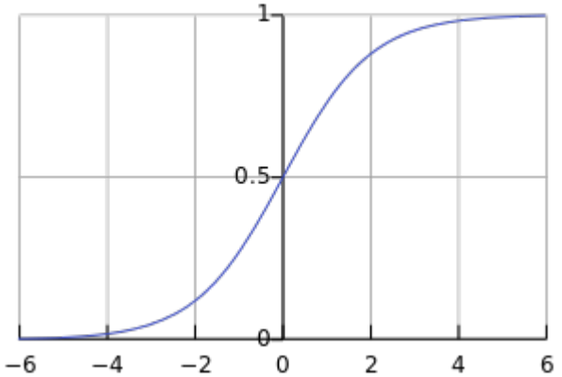
Output = ?

NEURAL NETWORKS

- Another example of a neural network with one perceptron is a **Logistic Regression**
- The decision function is based on the **sigmoid function** where the output is bounded between 0 and 1.
- The linear combination of coefficients with input values gives you **one decision boundary**.

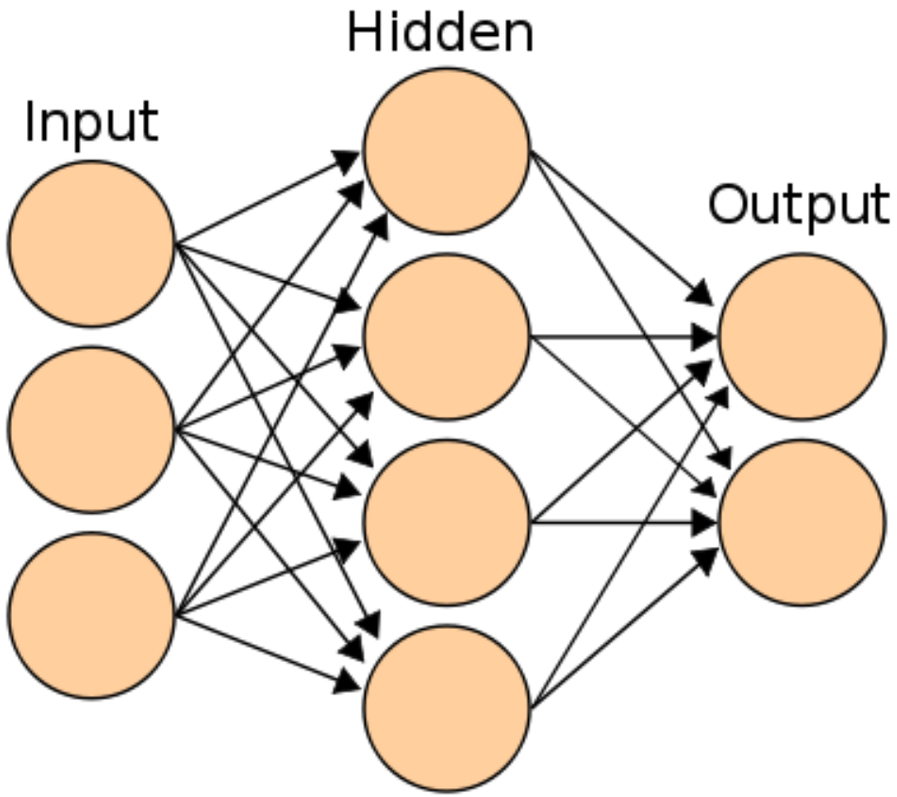


$$\text{output} = \begin{cases} 0 & \text{if } \dfrac{1}{1+e^{-(ax_1+bx_2+cx_3)}} < 0.5 \\ 1 & \text{if } \dfrac{1}{1+e^{-(ax_1+bx_2+cx_3)}} \geq 0.5 \end{cases}$$



NEURAL NETWORKS

- Neural networks are loosely modeled on the brain and have we can have many nodes (thousands to millions) in a neural network.
- There can be many **layers** of perceptrons – called **hidden layers**
- We choose both the # of layers and # of nodes in each layer.



- In this example, the perceptrons in the hidden layer are making four decisions based on the input variables.

- Like input and output, the hidden layers output values based on an **activation function**.

http://neuralnetworksanddeeplearning.com/chap1.html

KNOWLEDGE CHECK

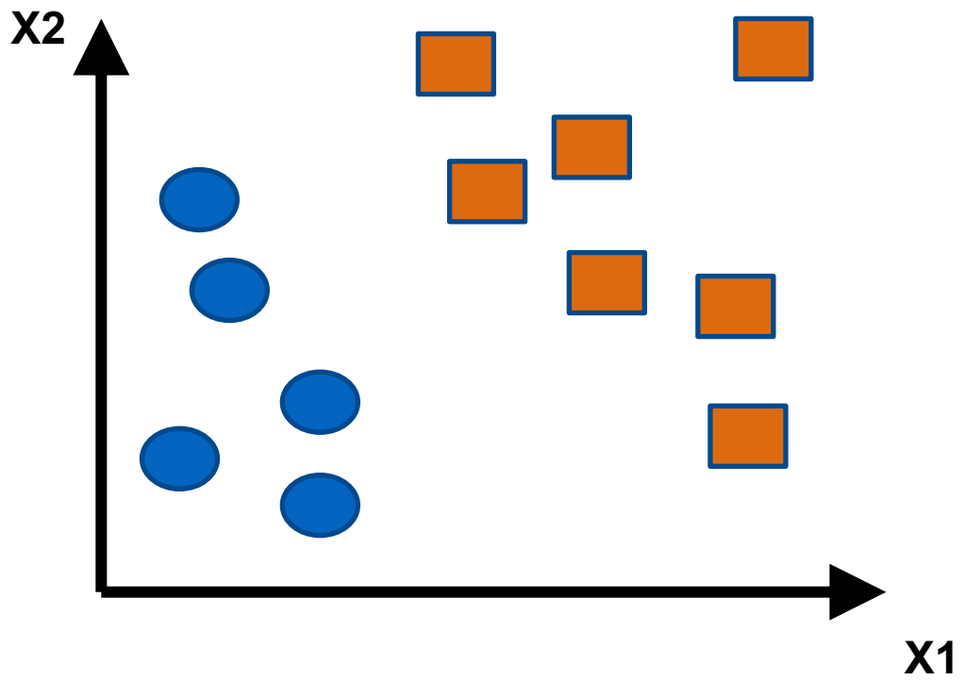
- What is a neural network supposed to mimic?
- What is a perceptron? How does it make decisions?
- What is an example of a neural network with just one perceptron?

EXAMPLE

NEURAL NETWORKS

Example

- Suppose you want to use a neural network to classify whether a point is a circle or square with two features, x1 and x2.

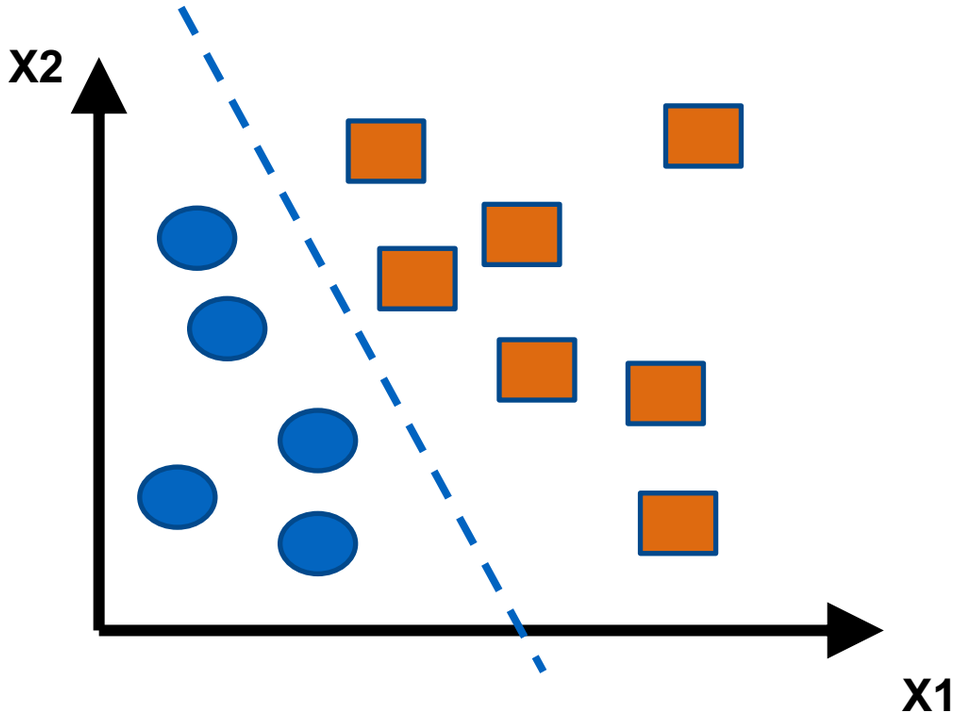


- How would a neural network fit this data?

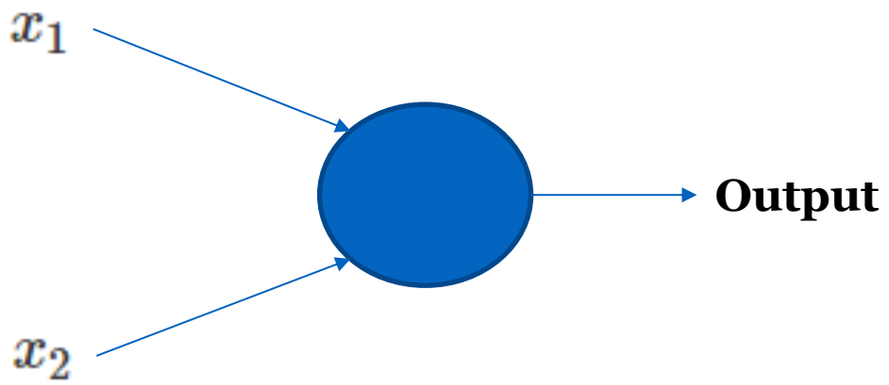
NEURAL NETWORKS

Example

- The points could be separated with one decision boundary AKA one perceptron
- This decision boundary could be derived using Logistic Regression



$$L = ax_1 + bx_2 + c$$

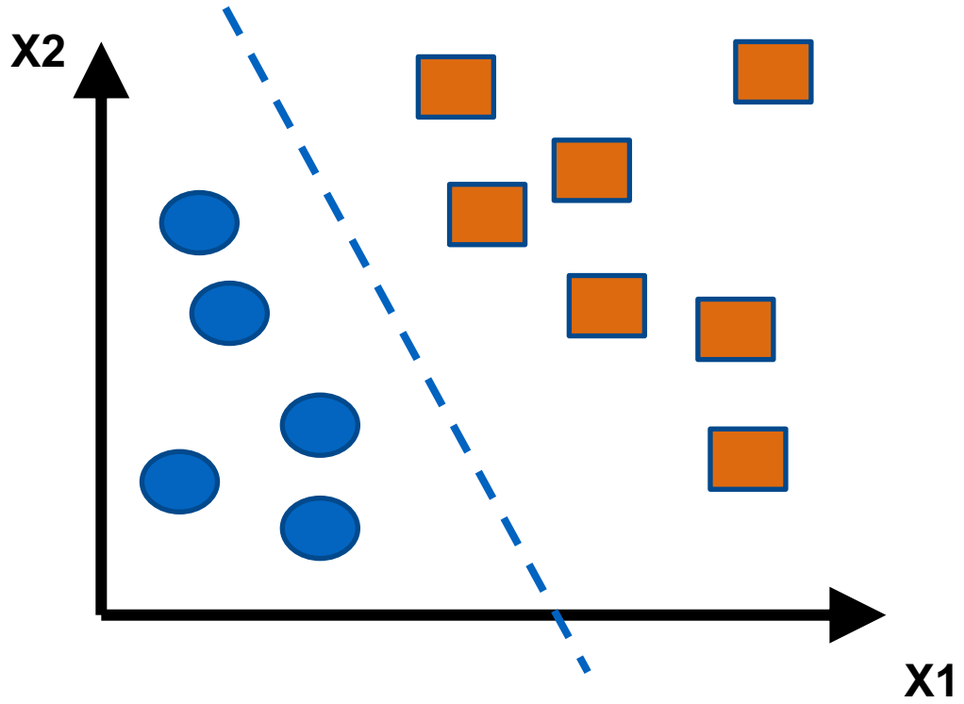


$$\text{Output} = \begin{cases} \text{Square if} & \dfrac{1}{1+e^{-(ax_1+bx_2+c)}} \geq 0.5 \\ \text{Circle if} & \dfrac{1}{1+e^{-(ax_1+bx_2+c)}} < 0.5 \end{cases}$$

- The decision boundary is derived the same way as in Logistic Regression using Gradient Descent

NEURAL NETWORKS

- How does a neural network come up with the decision boundary?



BACK PROPAGATION

BACK PROPAGATION

- **What is it?** The process in which we determine our decision boundaries for classification
- We use the same process for finding the decision boundary in **Logistic Regression**.
- **How does it work?**

1. A set of training examples is given: *{(x, d)*}* where *x* is the input and *d* the target output [supervis ed learning]

2. Examples are presented to the network.

3. For each example, the network gives an output *y*.

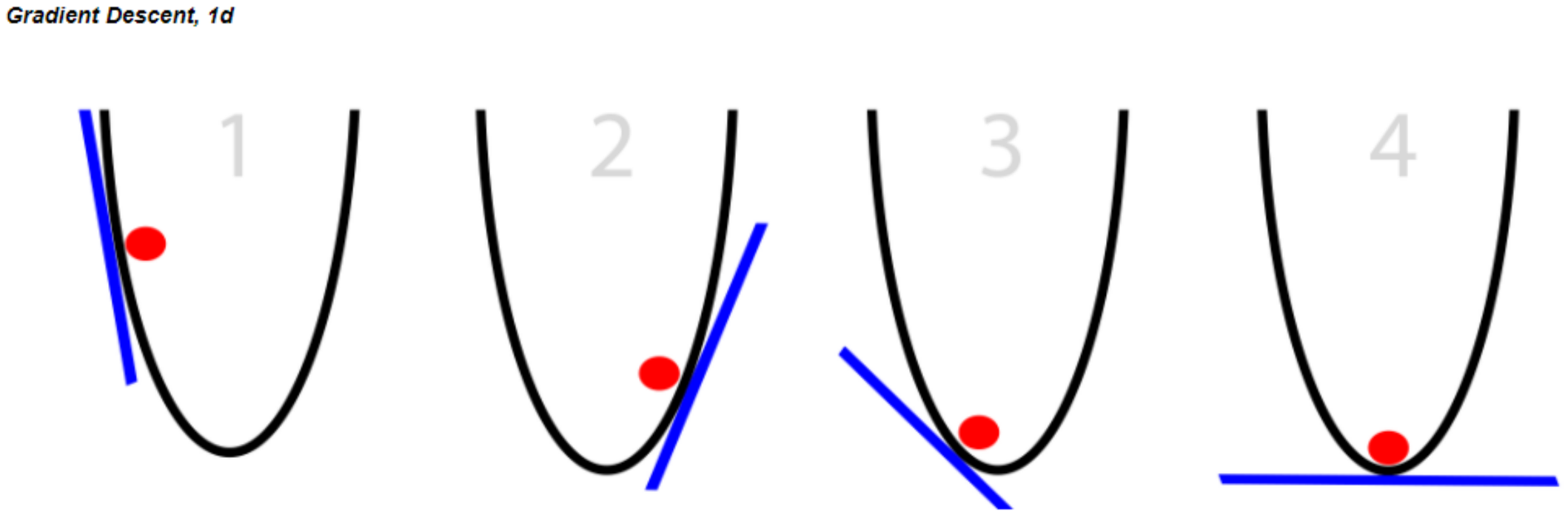
4. If there is an error, the decision boundary/line is moved in order to correct the output error.
   - We adjust the boundary using **Gradient Descent**

5. When the error has reached a minimum threshold, Stop learning.

GRADIENT DESCENT REFRESHER

GRADIENT DESCENT

- Gradient Descent is an **iterative** process that we use to pick our decision boundary
- You can see it as almost a guess and check – do these sets of coefficients minimize the error between my guess and the true point? If no, then keep looking for better coefficients. If yes, then stop!
- The y-axis represents the residuals AKA the **cost function**, and the x-axis represents our parameters for the decision boundary
- Notice how there is only **one** point that minimizes the cost function



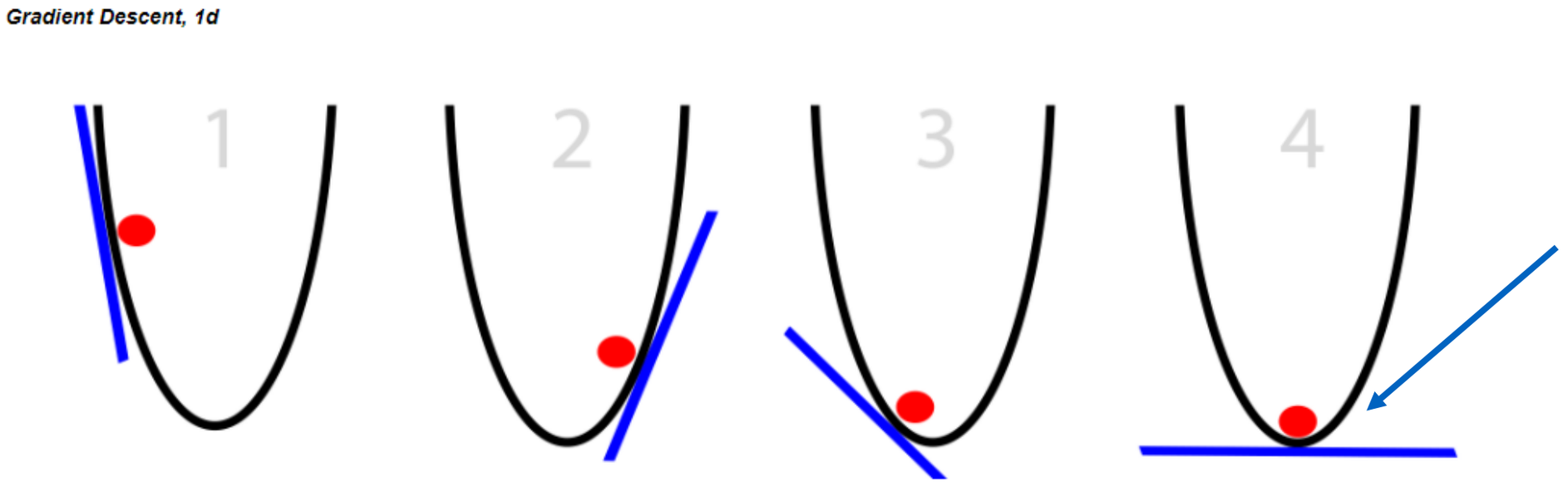
GRADIENT DESCENT

1. A random linear solution is provided as a starting point (usually a "flat" line or solution)

2. The solver then attempts to find a next step: we take a step in any direction and measure each performance.

3. If the solver finds a better solution (optimizing toward a metric such as mean squared error), this is the new starting point.

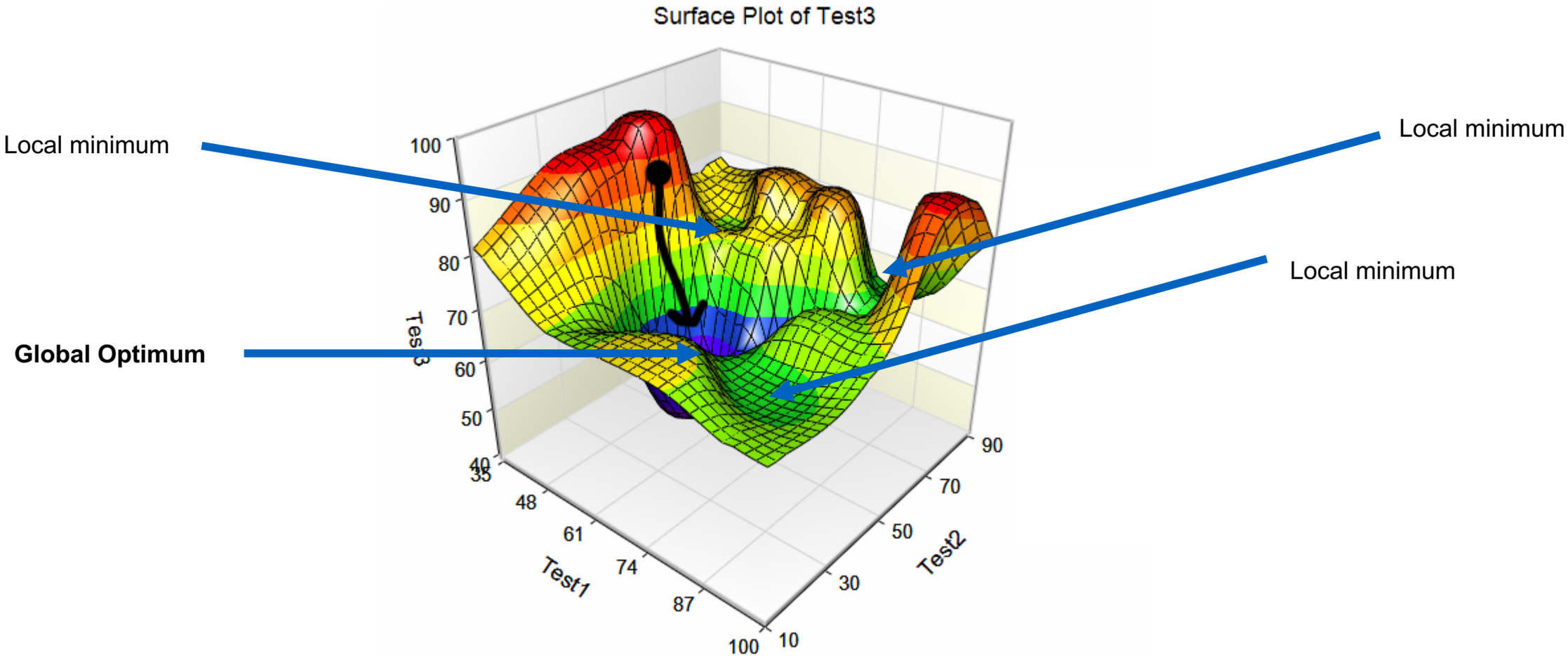
4. Repeat these steps until the performance is optimized and no "next steps" perform better. The size of the steps will shrink over time.



Because we've minimized the residuals at this point, the set of parameters we use to define our decision boundary are here!

GRADIENT DESCENT

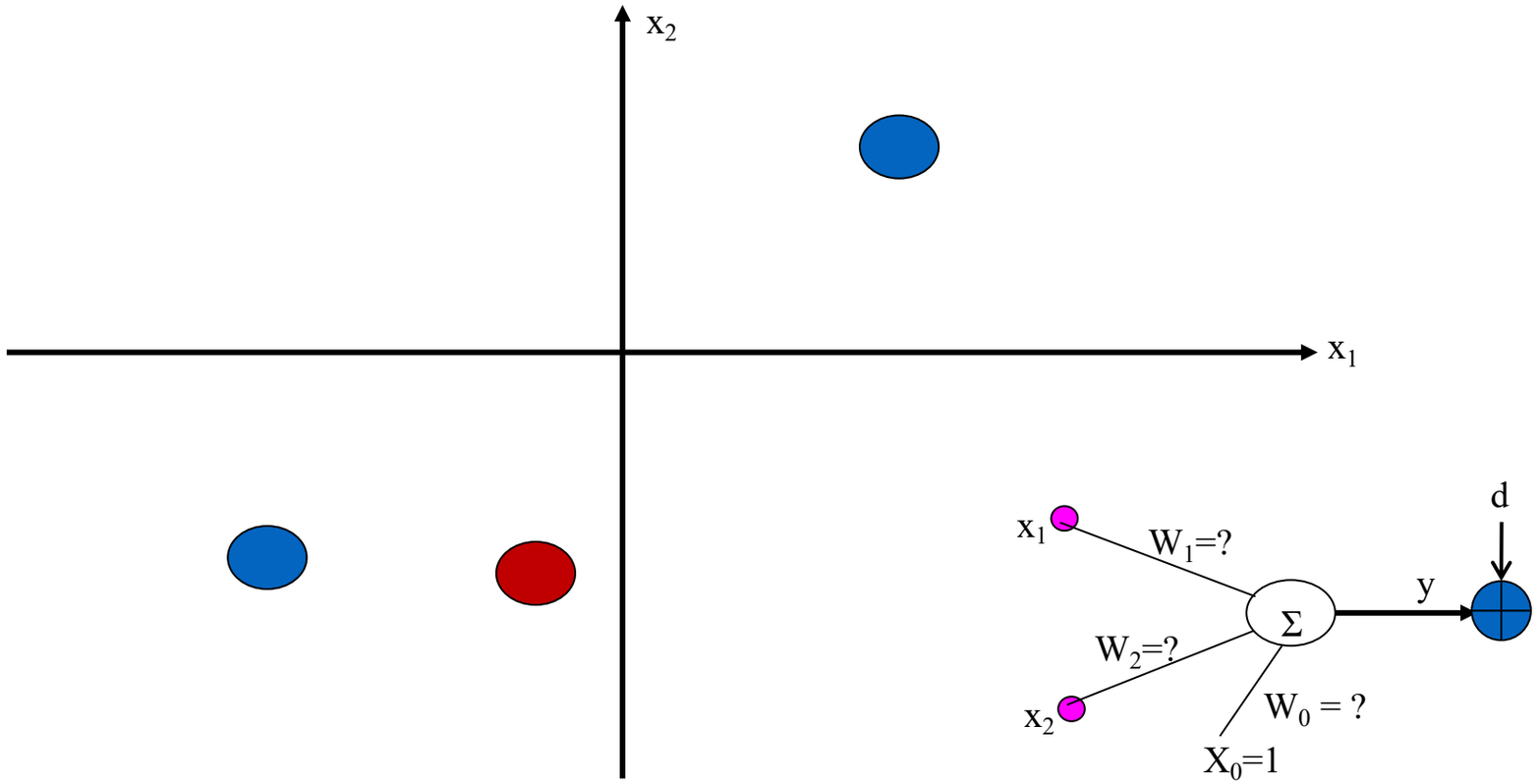
- Neural Nets (with at least one hidden layer) may have **many solutions** for the cost function
- Set your iterations as high as you can (without breaking your computer) to converge to the global minimum



BACK PROPAGATION WALKTHROUGH

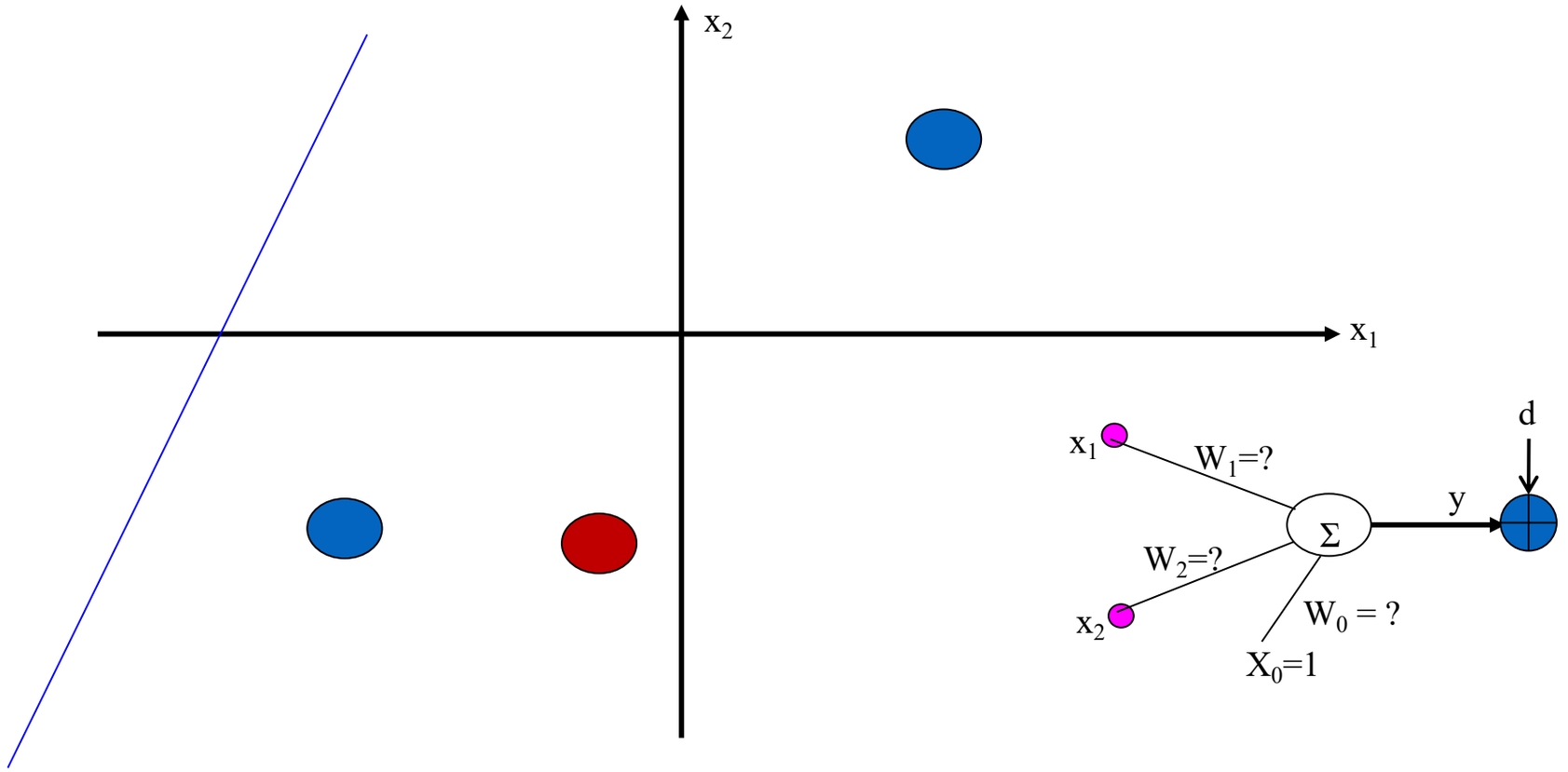
BACK PROPAGATION

- Consider the following example: (two classes: Red and Green)
- How can we find a decision boundary to separate them?



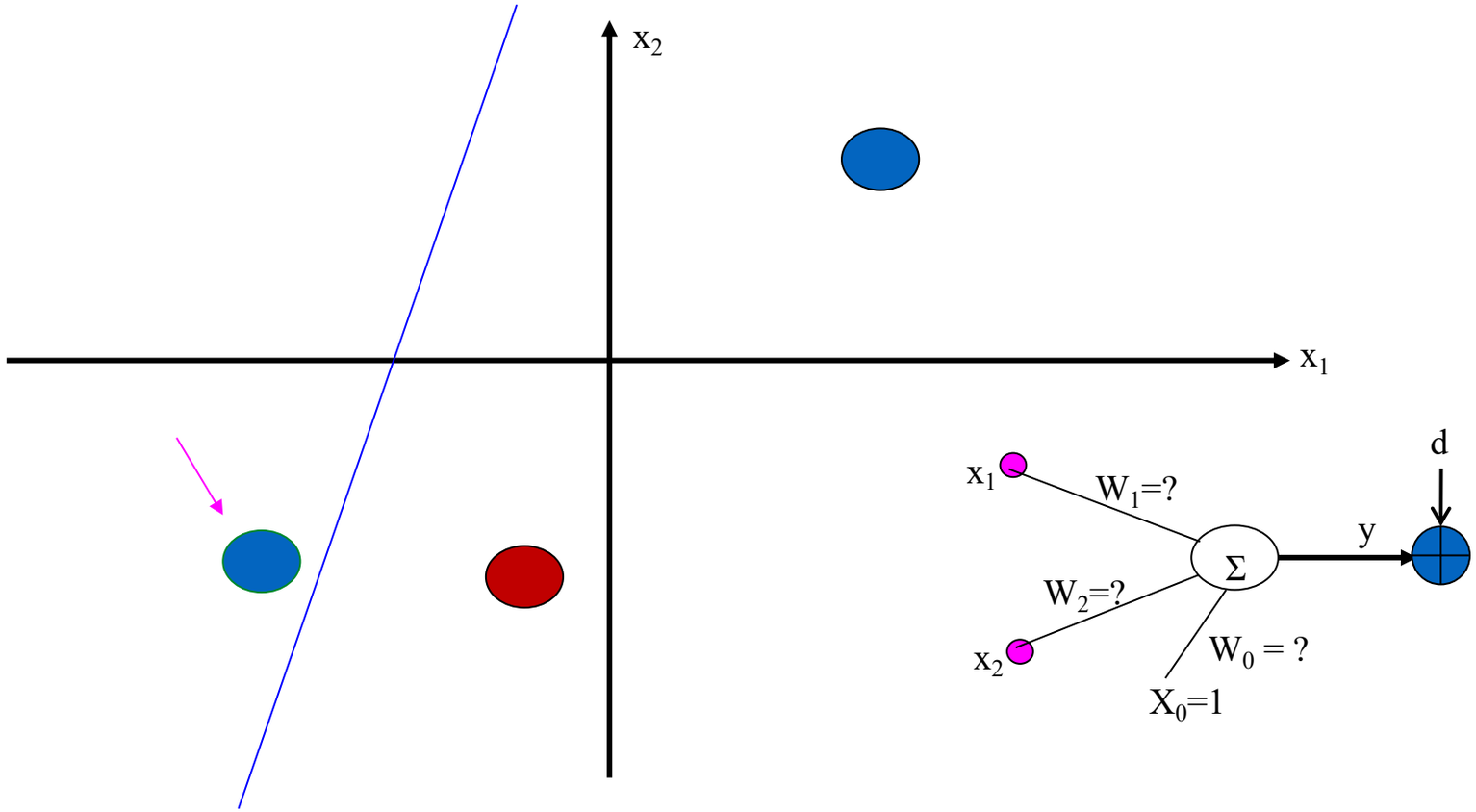
BACK PROPAGATION

- Random Initialization of perceptron weights
- Calculate your error and use gradient descent to figure out how to adjust your line (aka adjust the weights) in order to decrease the error



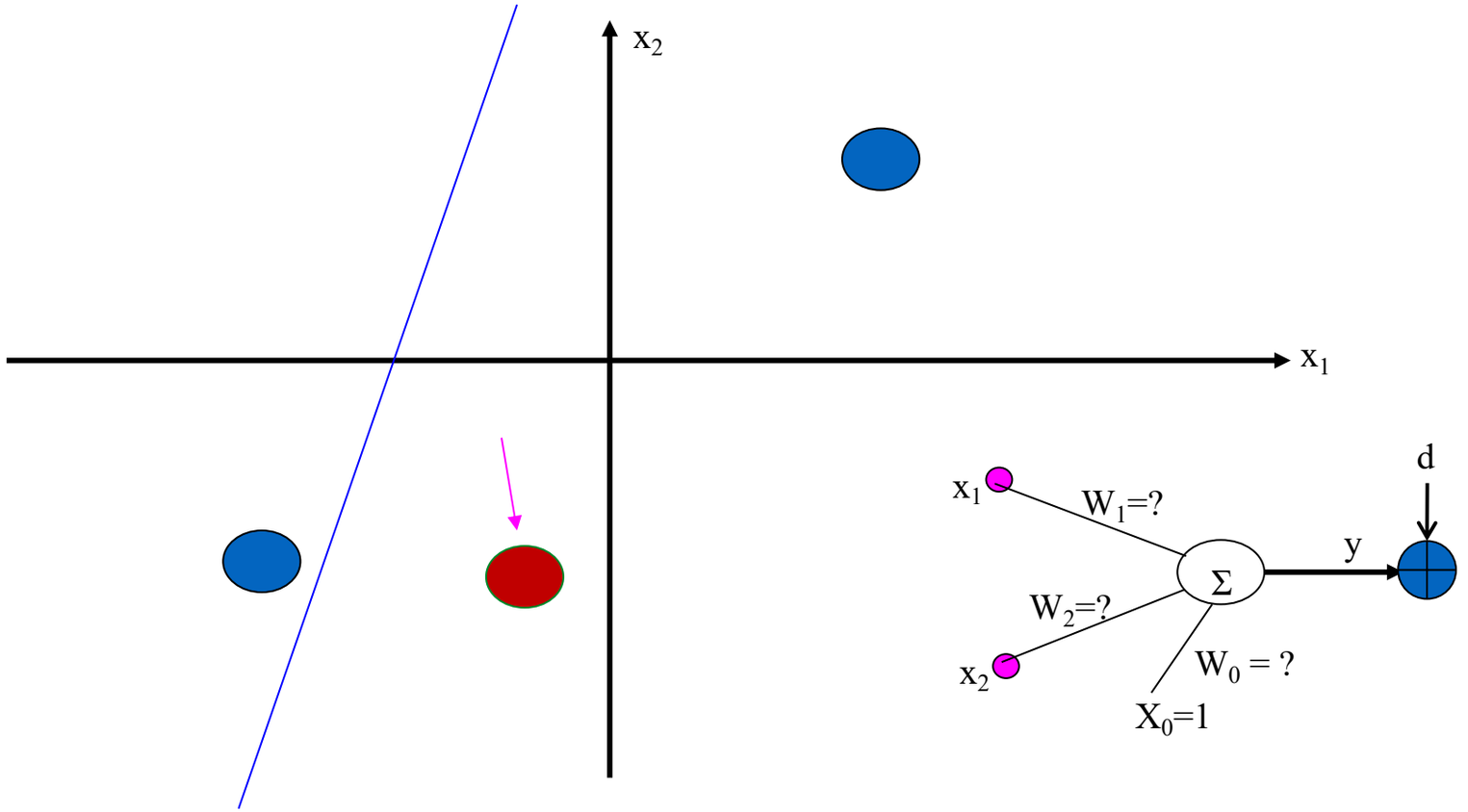
BACK PROPAGATION

- Calculate error of new line
- Using Gradient Descent, adjust the line towards where the **error is decreasing**



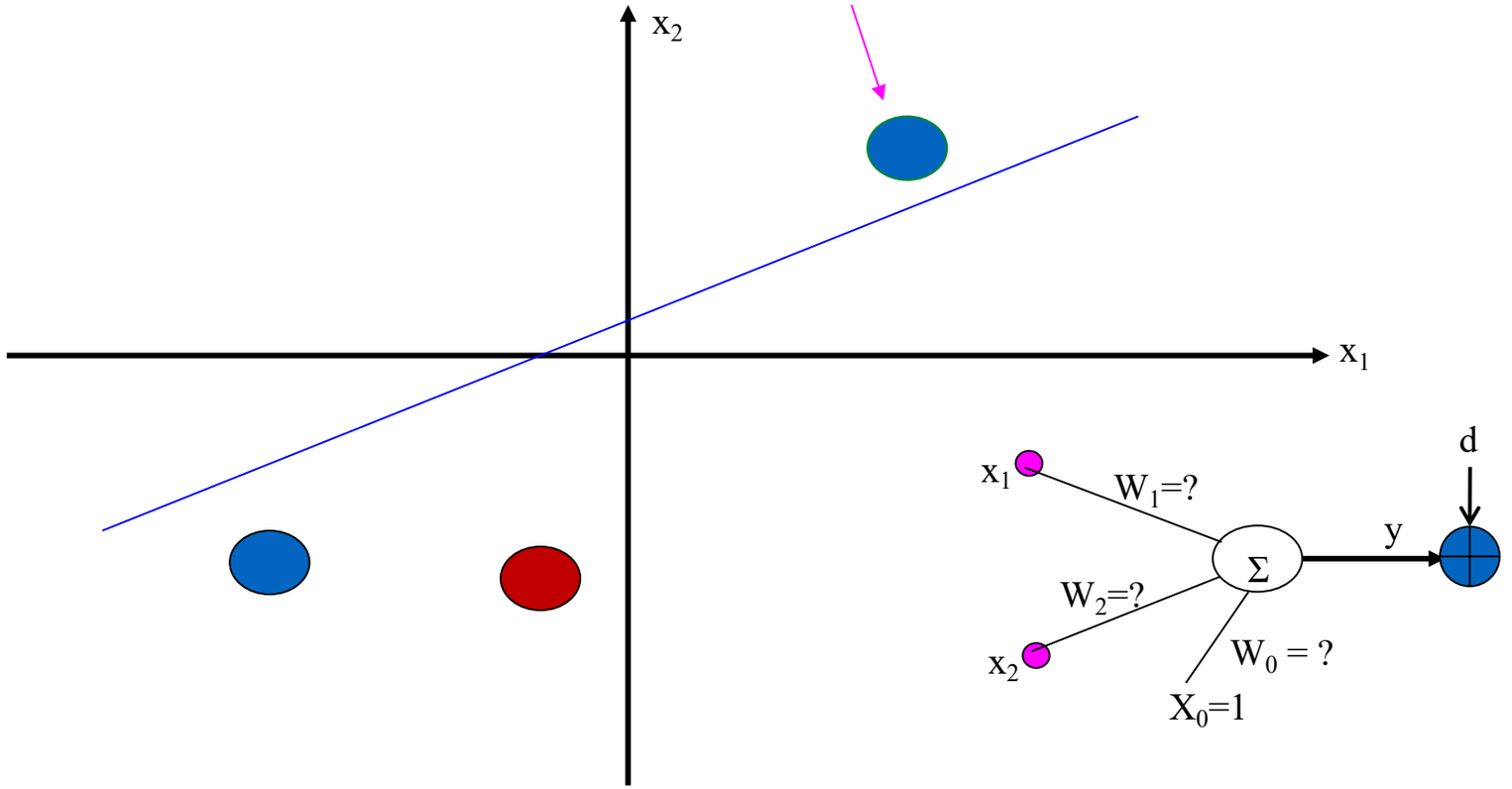
BACK PROPAGATION

Using Gradient Descent, move the line towards where the **error is decreasing**



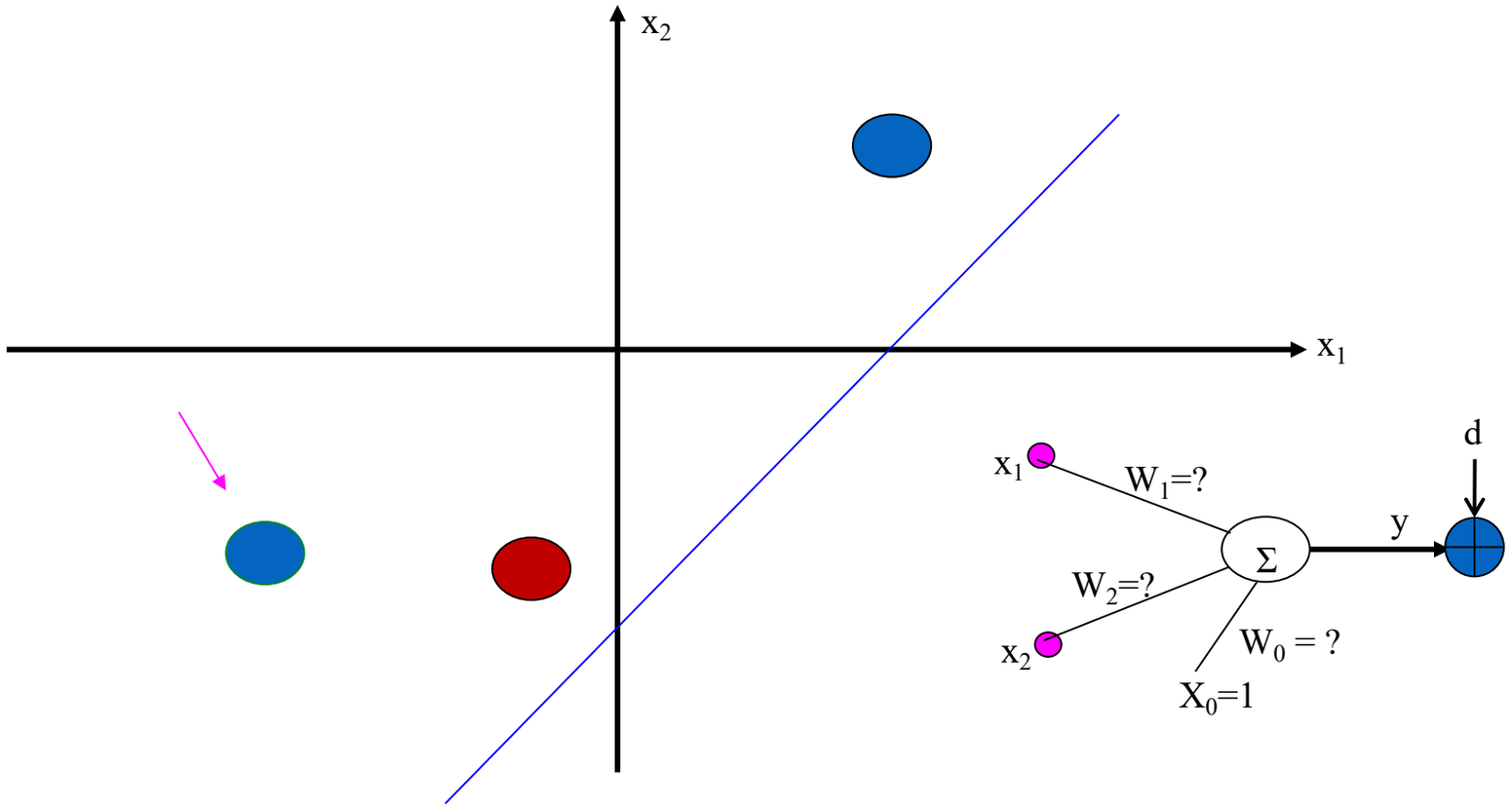
BACK PROPAGATION

Apply Iteratively Perceptron Training Rule on the different examples:



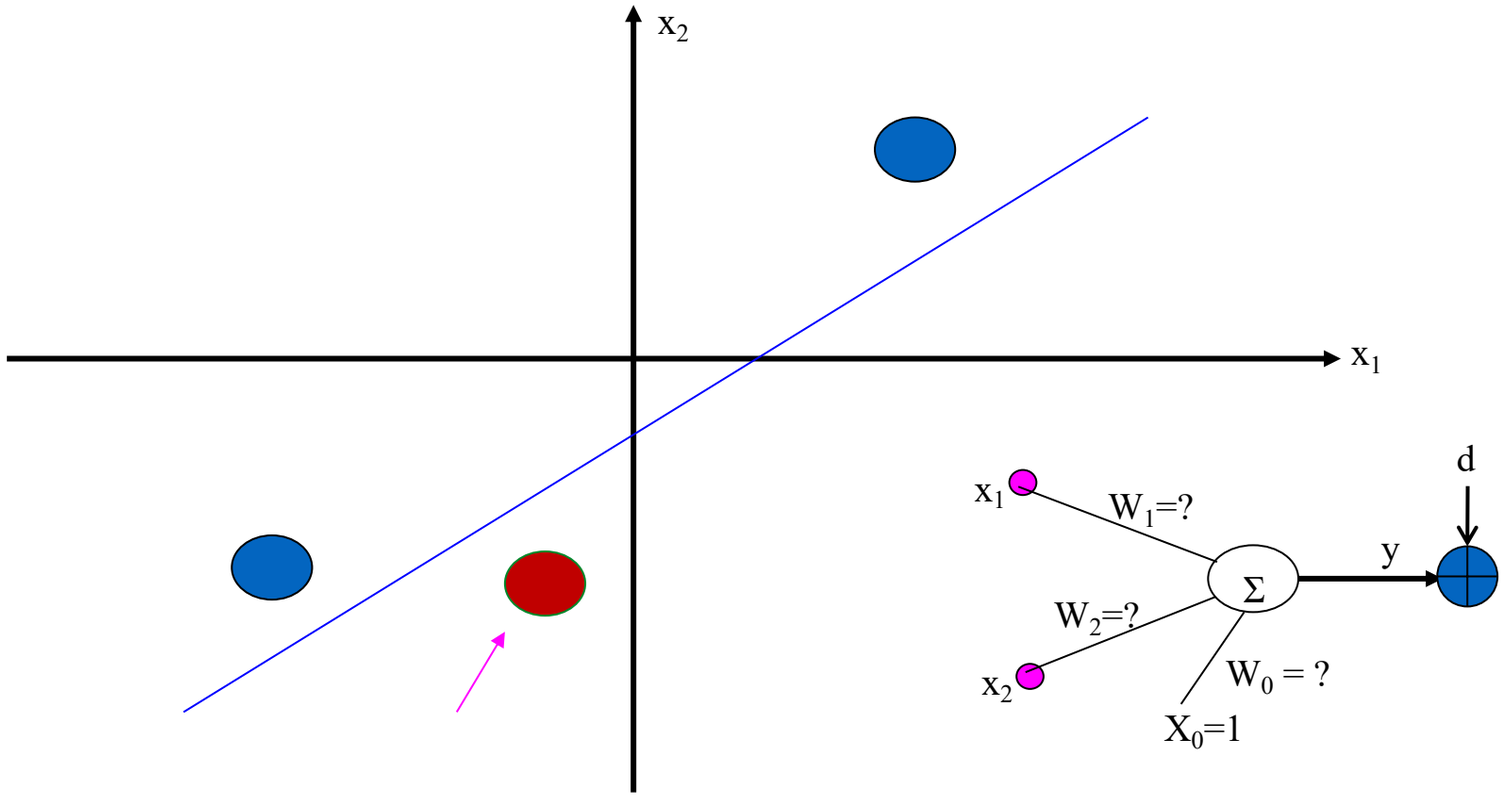
BACK PROPAGATION

Apply Iteratively Perceptron Training Rule on the different examples:



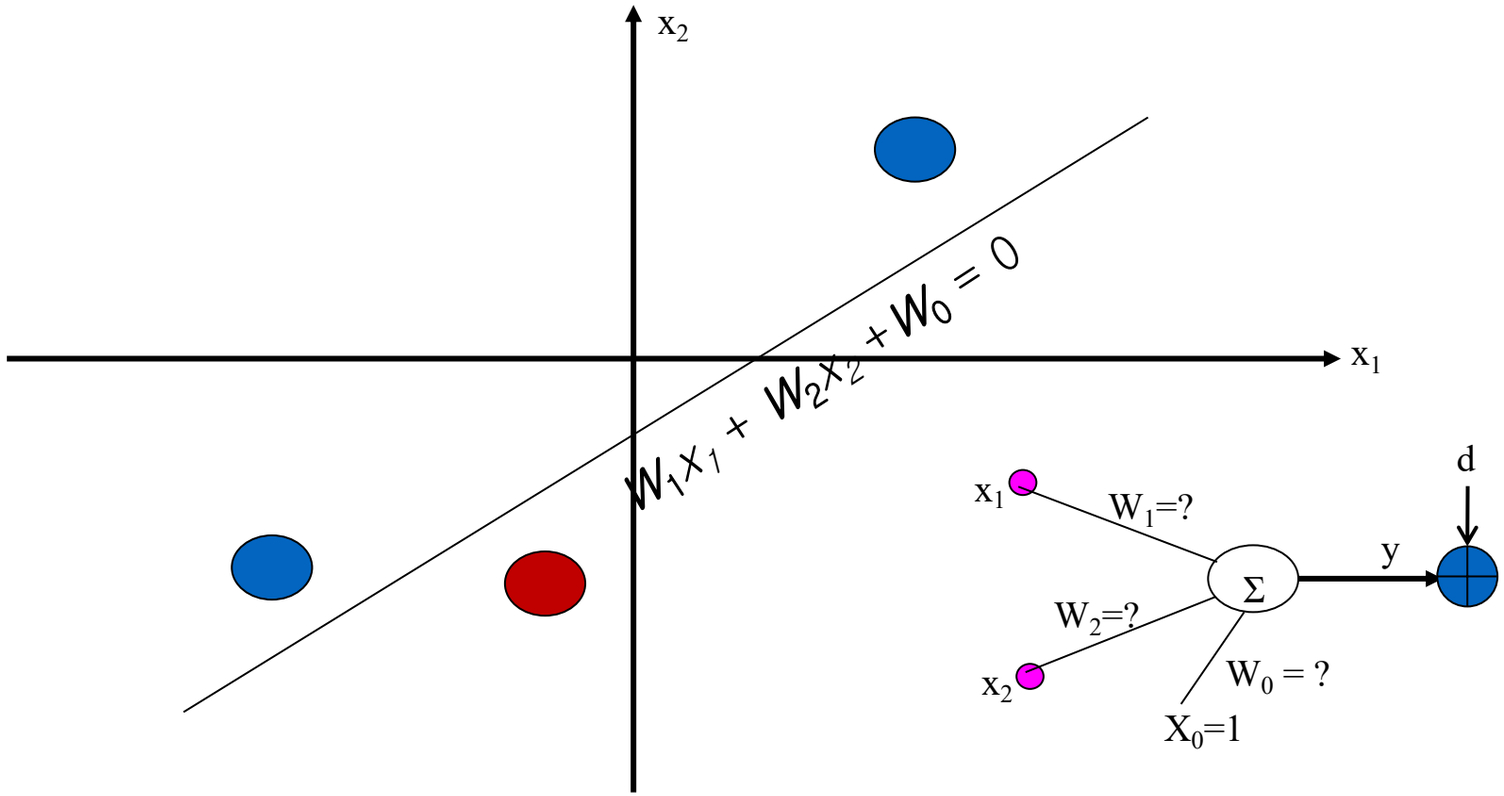
BACK PROPAGATION

All examples are correctly classified … stop Learning



BACK PROPAGATION

The straight line $w_1x + w_2y + w_0 = 0$ separates the two classes

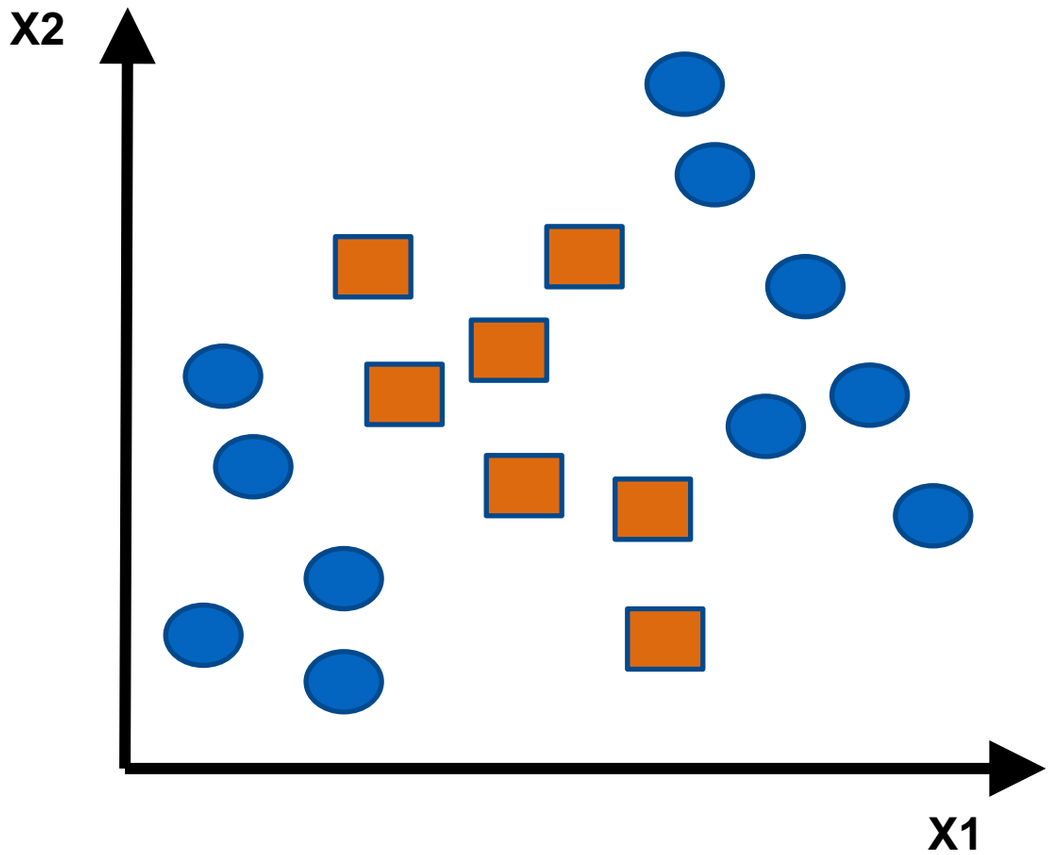


EXAMPLE WITH HIDDEN LAYER

NEURAL NETWORKS

Example

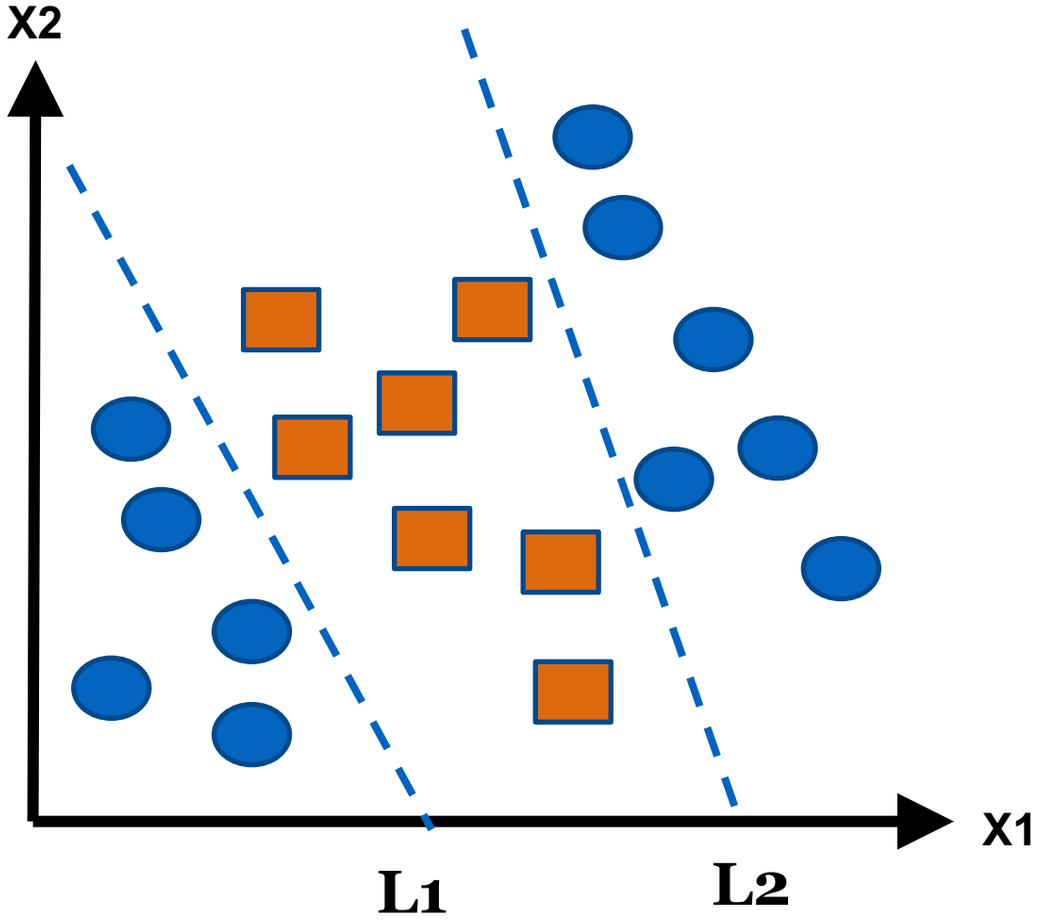
- Now suppose we have this data set. How would a neural network fit this data?



NEURAL NETWORKS

Example

- We could try using **two decision boundaries**

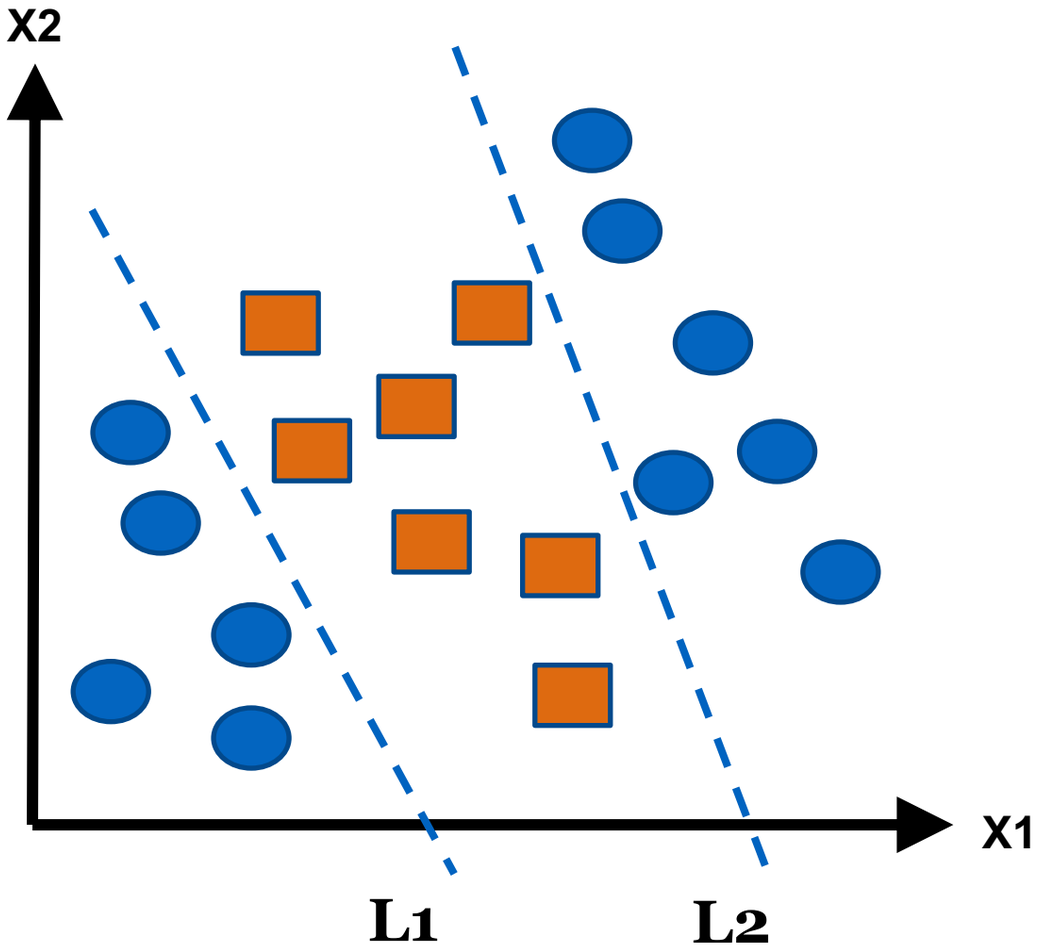


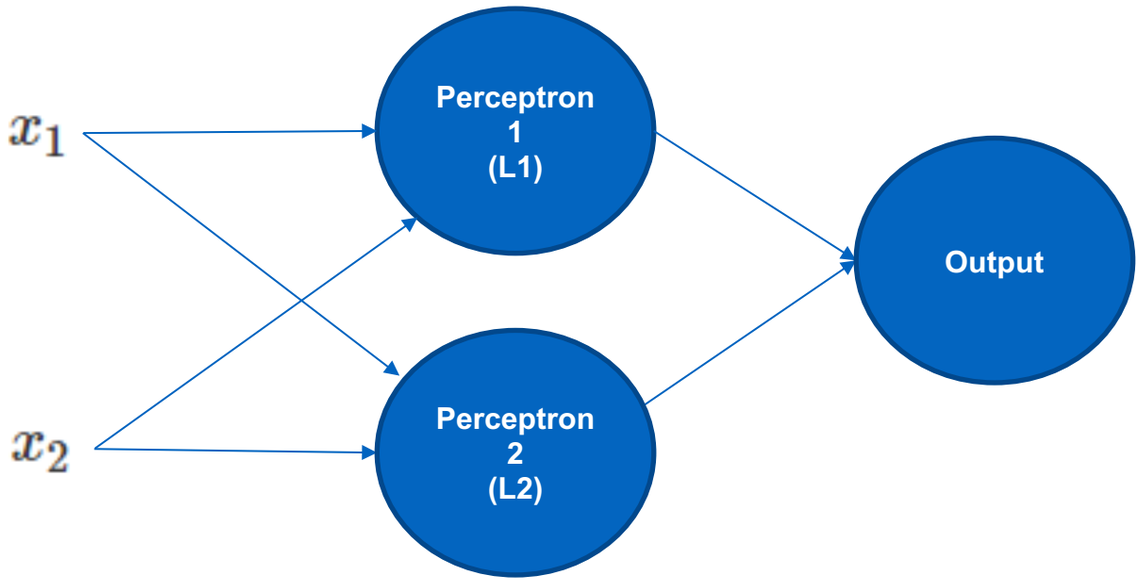
We want a decision function that says…

- "If it's between L1 and L2, then it's a square. Else, it's a circle"
- Or, "If it's greater than L2 OR less than L1, then it's a circle. Else, it's a square"
- Can Neural Nets do this?

NEURAL NETWORKS

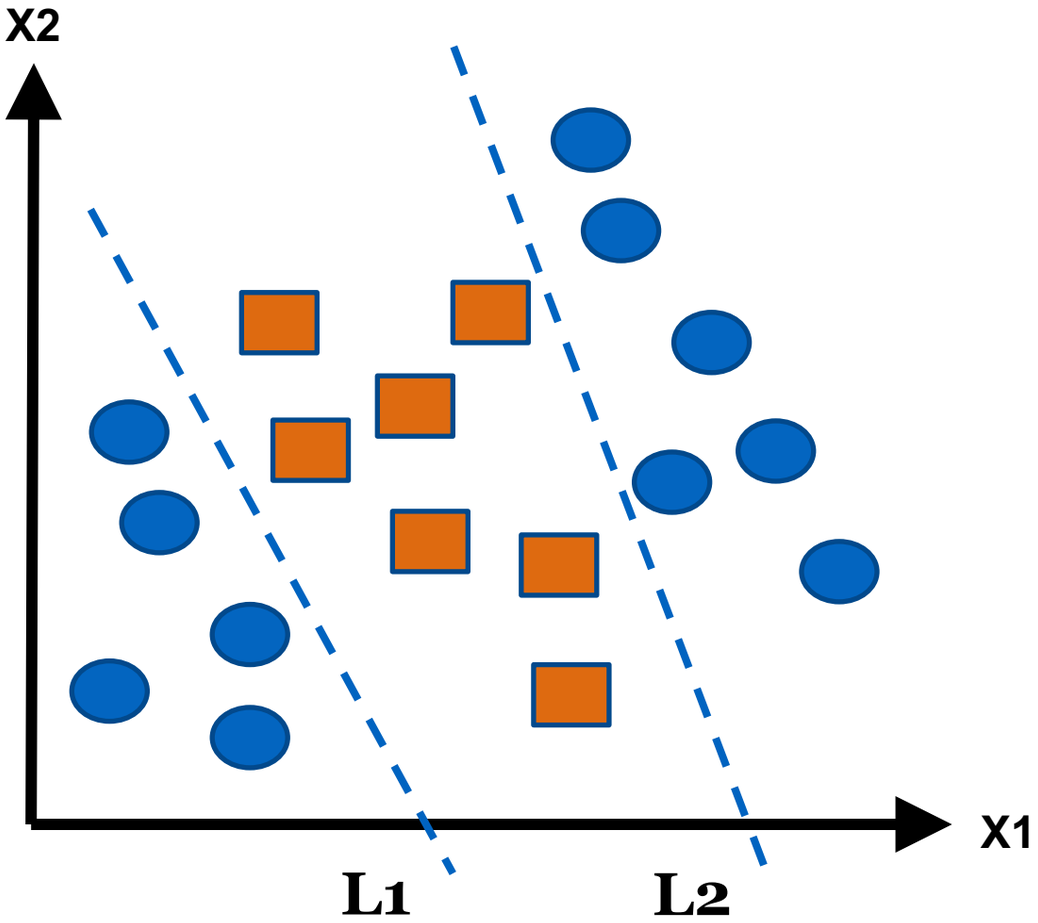
- A Neural Network with two decision boundaries uses two perceptrons: L1 and L2.
- Using back propagation it separates the two classes best with these lines

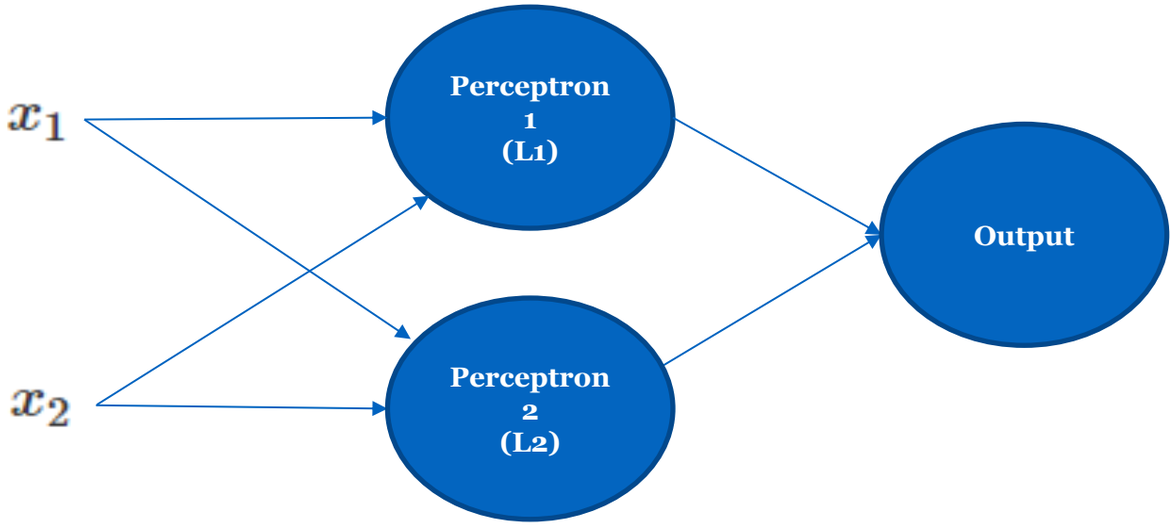




NEURAL NETWORKS

- We can define the final output by looking at the results of each of our decision boundaries

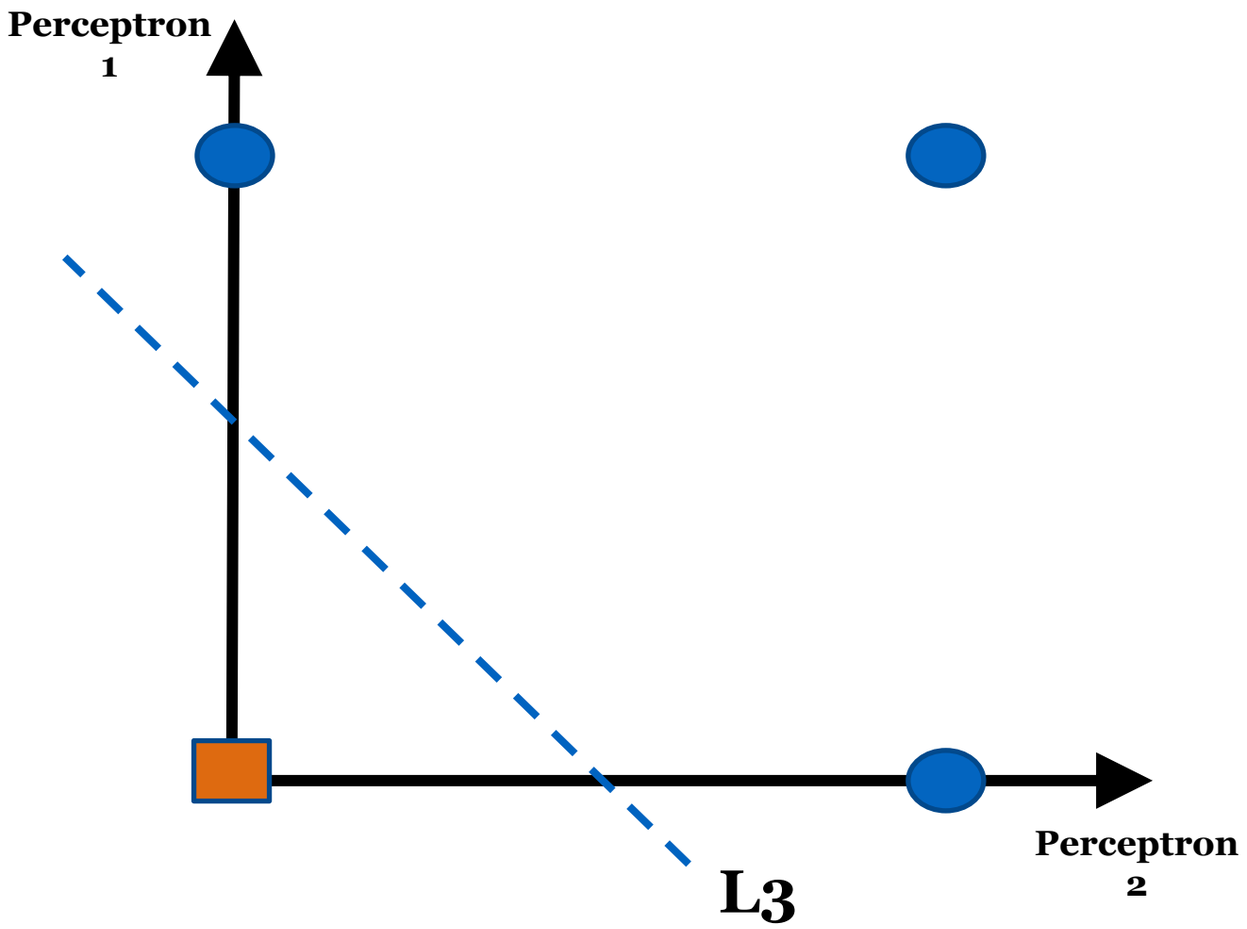




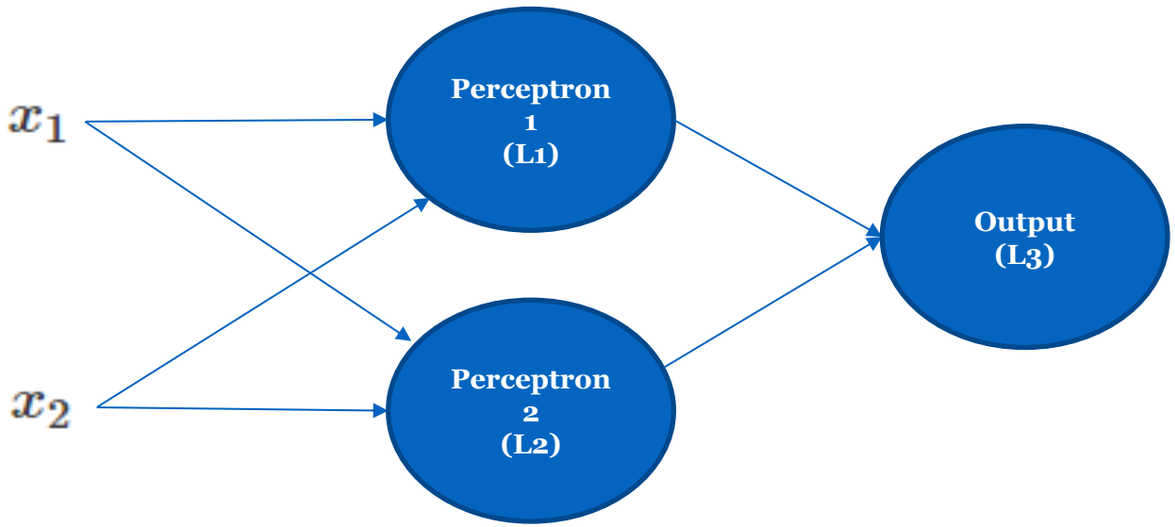
Perceptron 1 value	Perceptron 2 value	Final Output Value
0 = not circle	0 = not circle	0 = Square
0 = not circle	1 = circle	1 = Circle
1 = circle	0 = not circle	1 = Circle
1 = circle	1 = circle	1 = Circle

NEURAL NETWORKS

- Developing a decision boundary in the hidden layer looks something like this



Perceptron 1 value	Perceptron 2 value	Final Output value
0	0	1
0	1	0
1	0	0
1	1	0

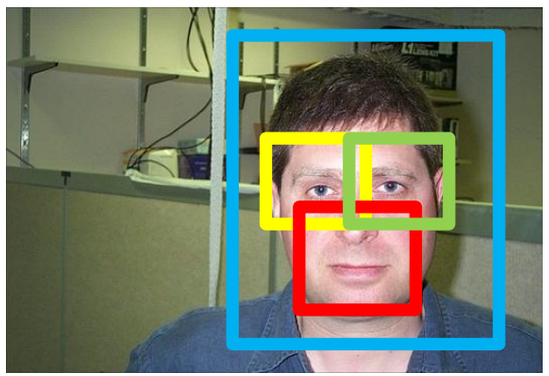


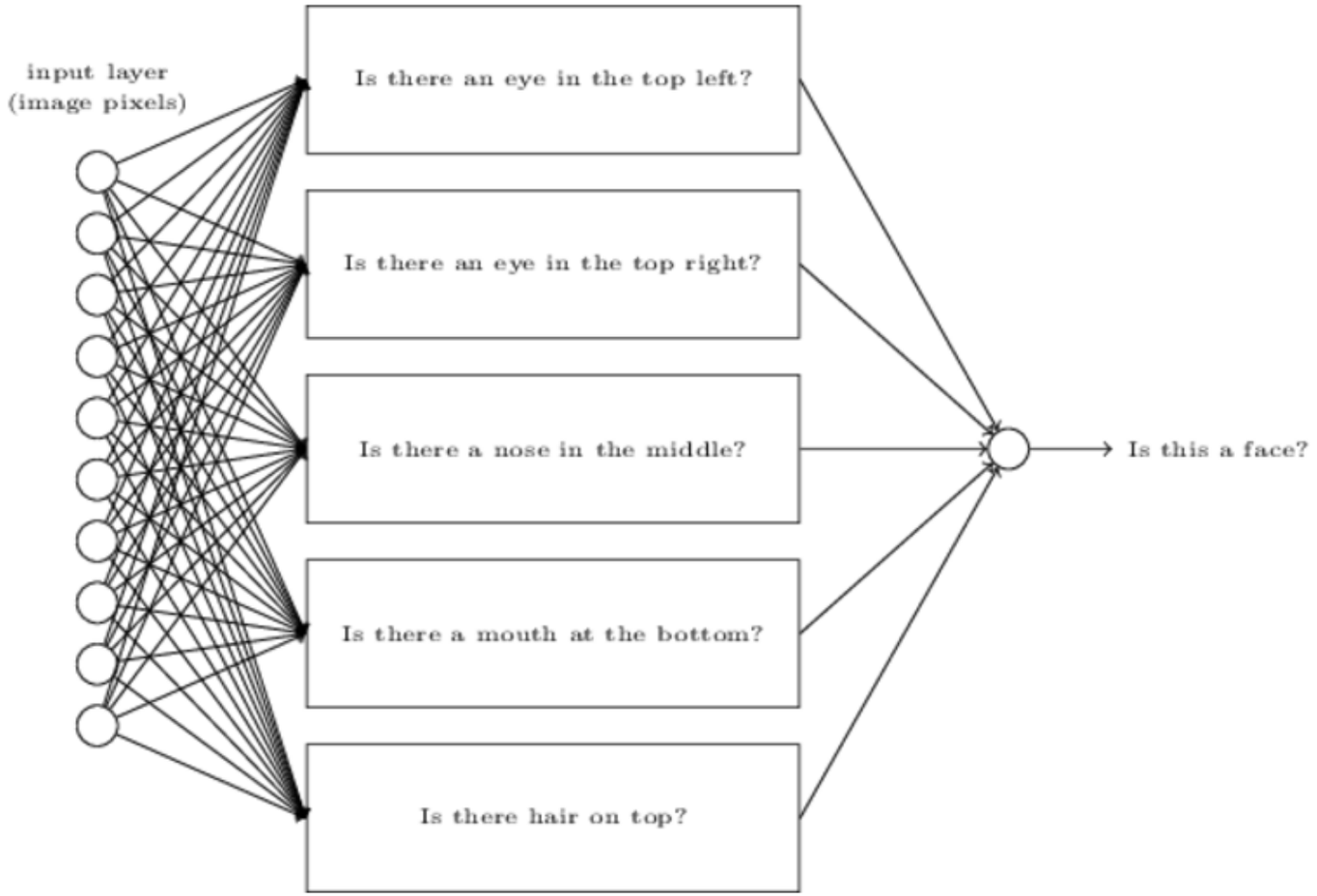
HIDDEN LAYERS: INTUITION

NEURAL NETWORKS

- A **Deep Net** is a neural network that simply has two or more hidden layers
- What are the hidden layers doing? => Breaking down the problem into sub-problems
- End result is a network that breaks down a complex question into very simple questions

Input data



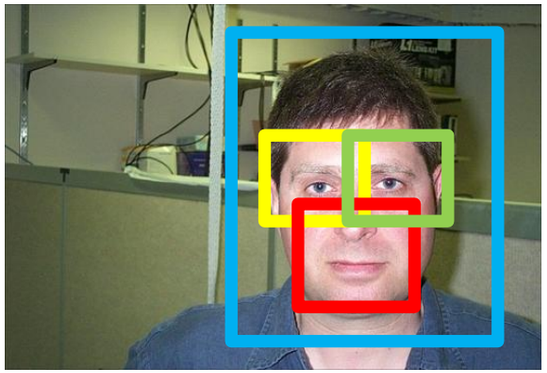


http://neuralnetworksanddeeplearning.com/chap1.html

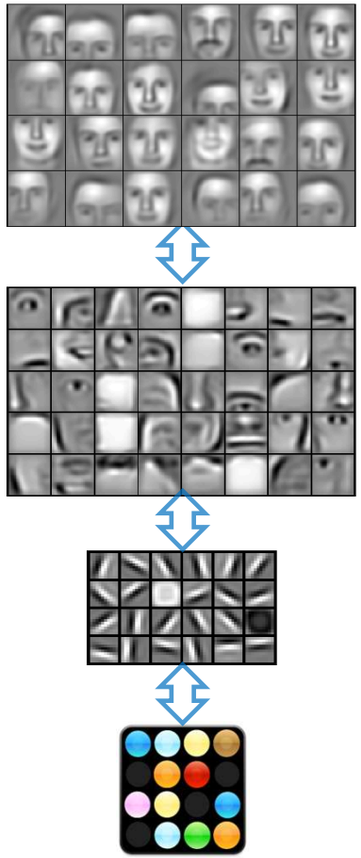
NEURAL NETWORKS

- Hidden layers extract different layers of patterns.
- Each hidden layer gets more detailed and specific about the patterns

Input data



Feature representation



3rd layer
“Objects”

2nd layer
“Object parts”

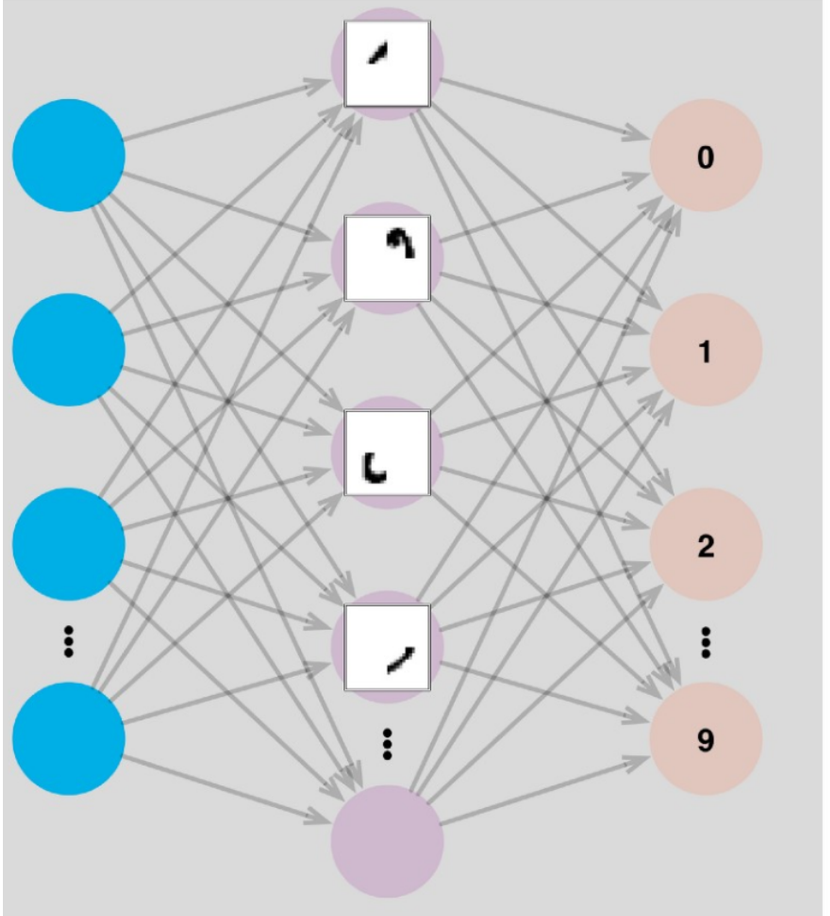
1st layer
“Edges”

Pixels

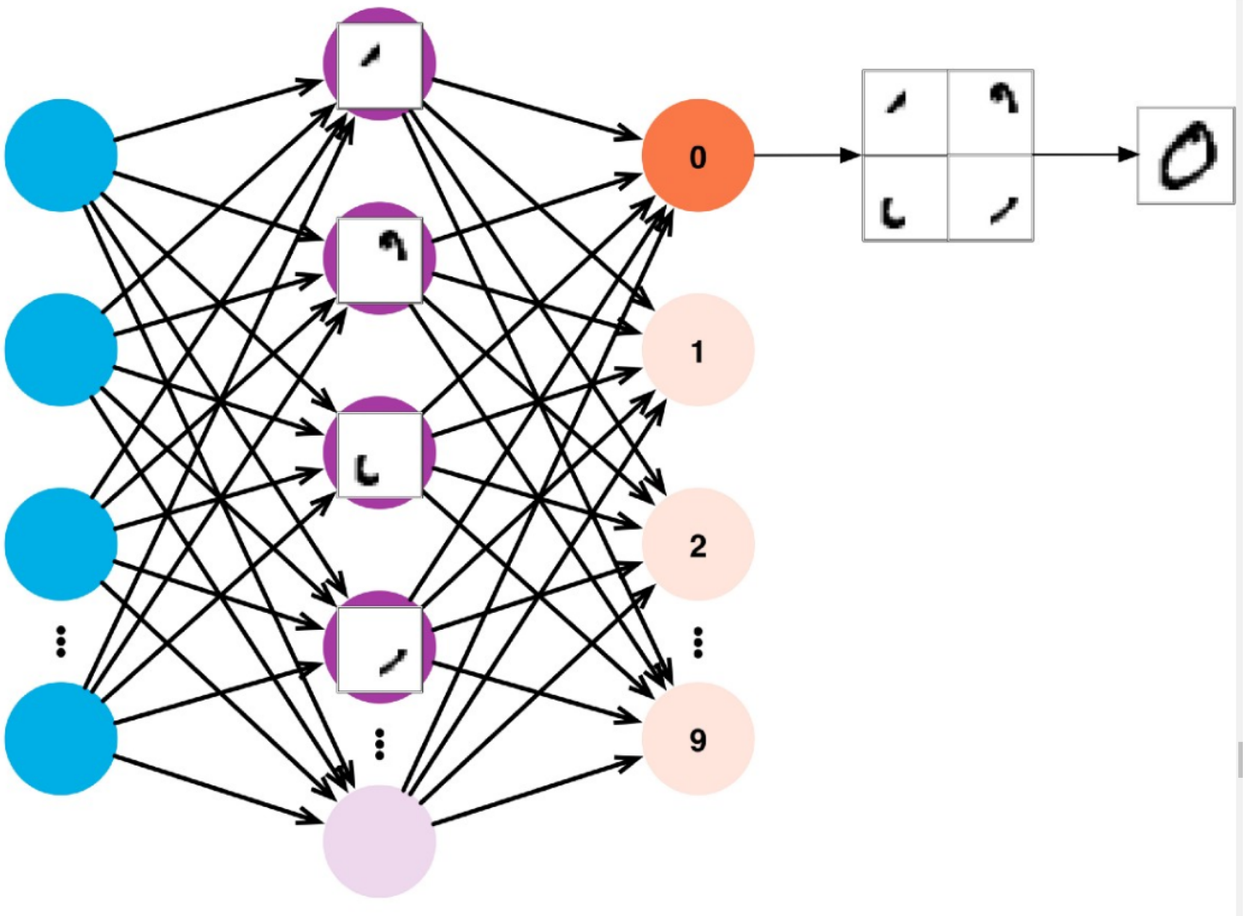
[Lee et al., ICML’09]

NEURAL NETWORKS

- In the case of identifying handwritten digits using a neural network, you can look at the hidden layers recognizing patterns in the digits:



A Trained Neural Network in which each hidden neuron classifies certain pattern

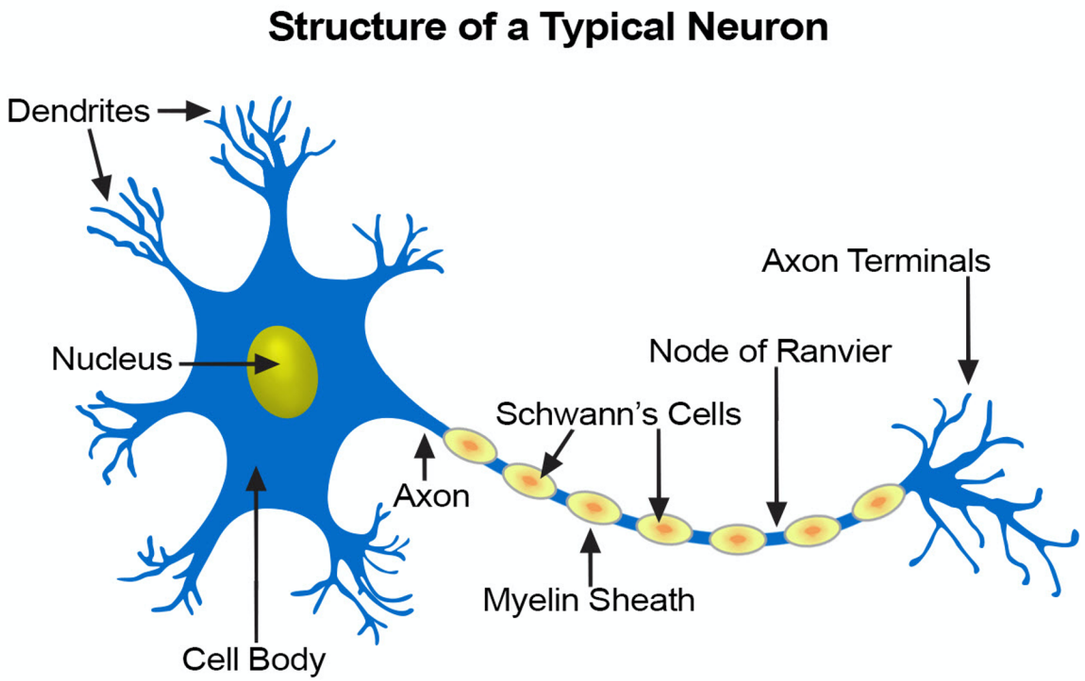


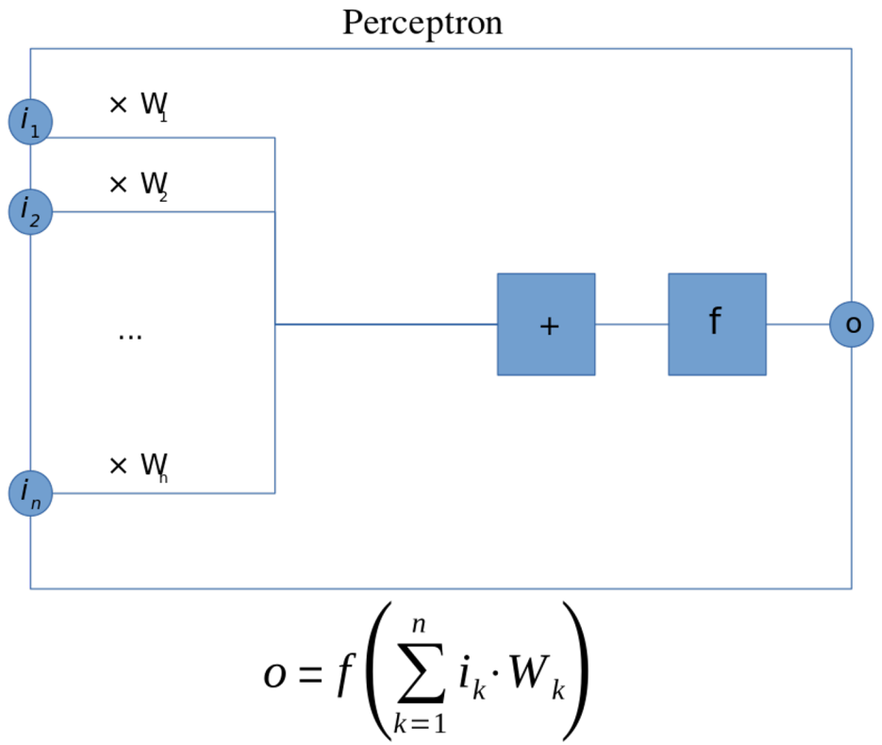
Feeding a handwritten digit of 0 should trigger the 4 hidden layer neurons, and then the first output neuron

PERCEPTRON

PERCEPTRON

- Perceptrons are the simplest example of a neural network
- The idea is to emulate a single neuron

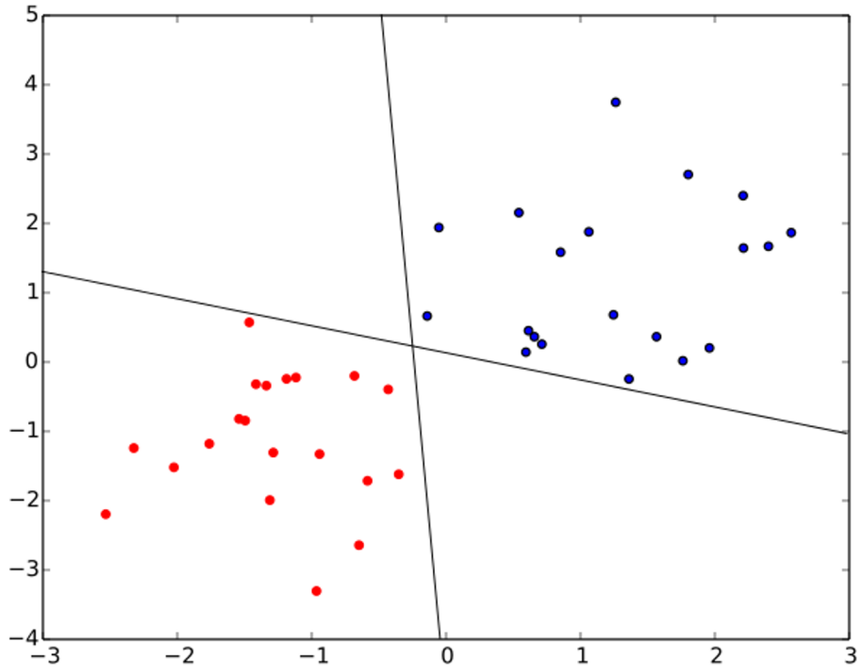


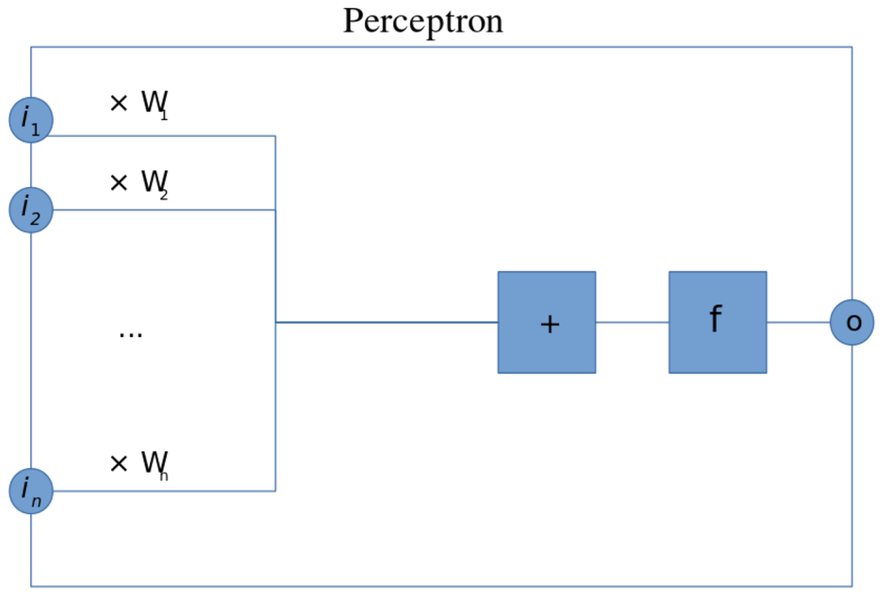


$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

PERCEPTRON

- Perceptrons are the simplest example of a neural network
- Given n inputs and an activation or link function f
- The perceptron computes a linear separating curve

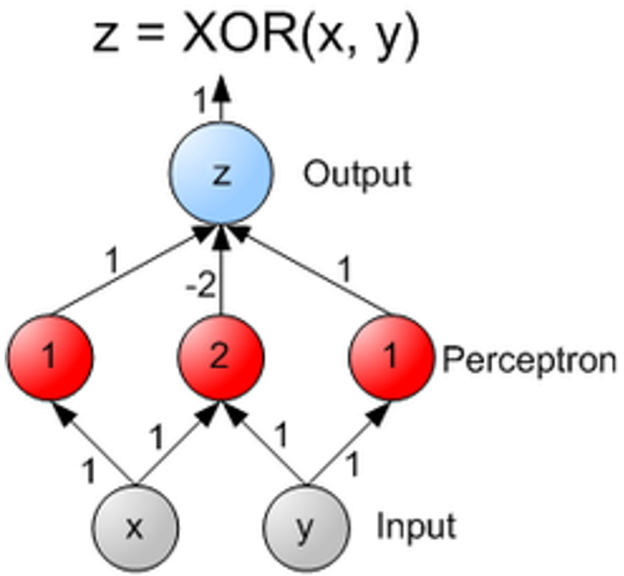




$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

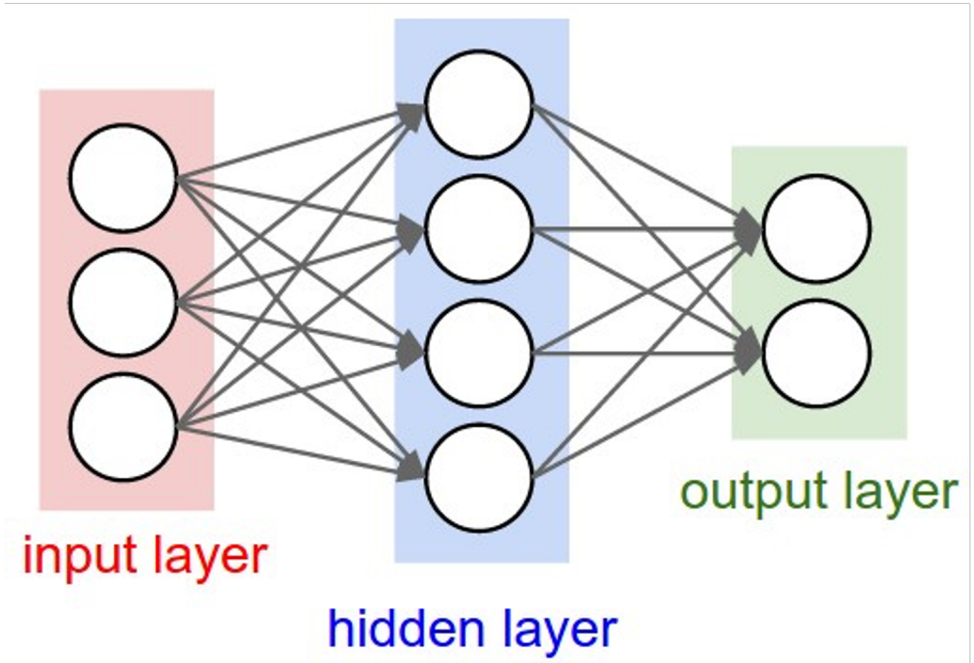
PERCEPTRON

- Common activation functions are linear, logistic, tanh, and softmax
- We'll see shortly that some are better for classification, some for regression
- Perceptrons can be combined into multilayer perceptrons or feed-forward network



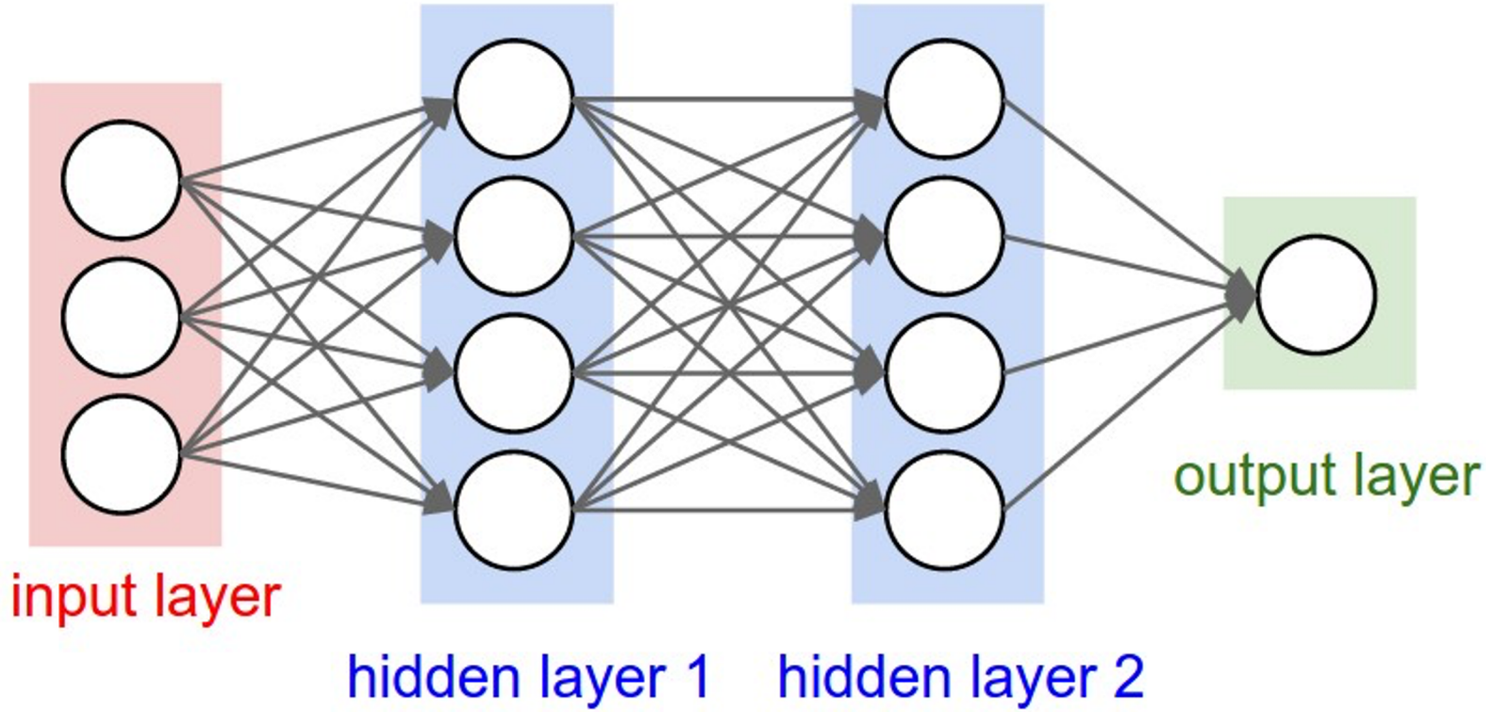
FEED FORWARD NN

- Source



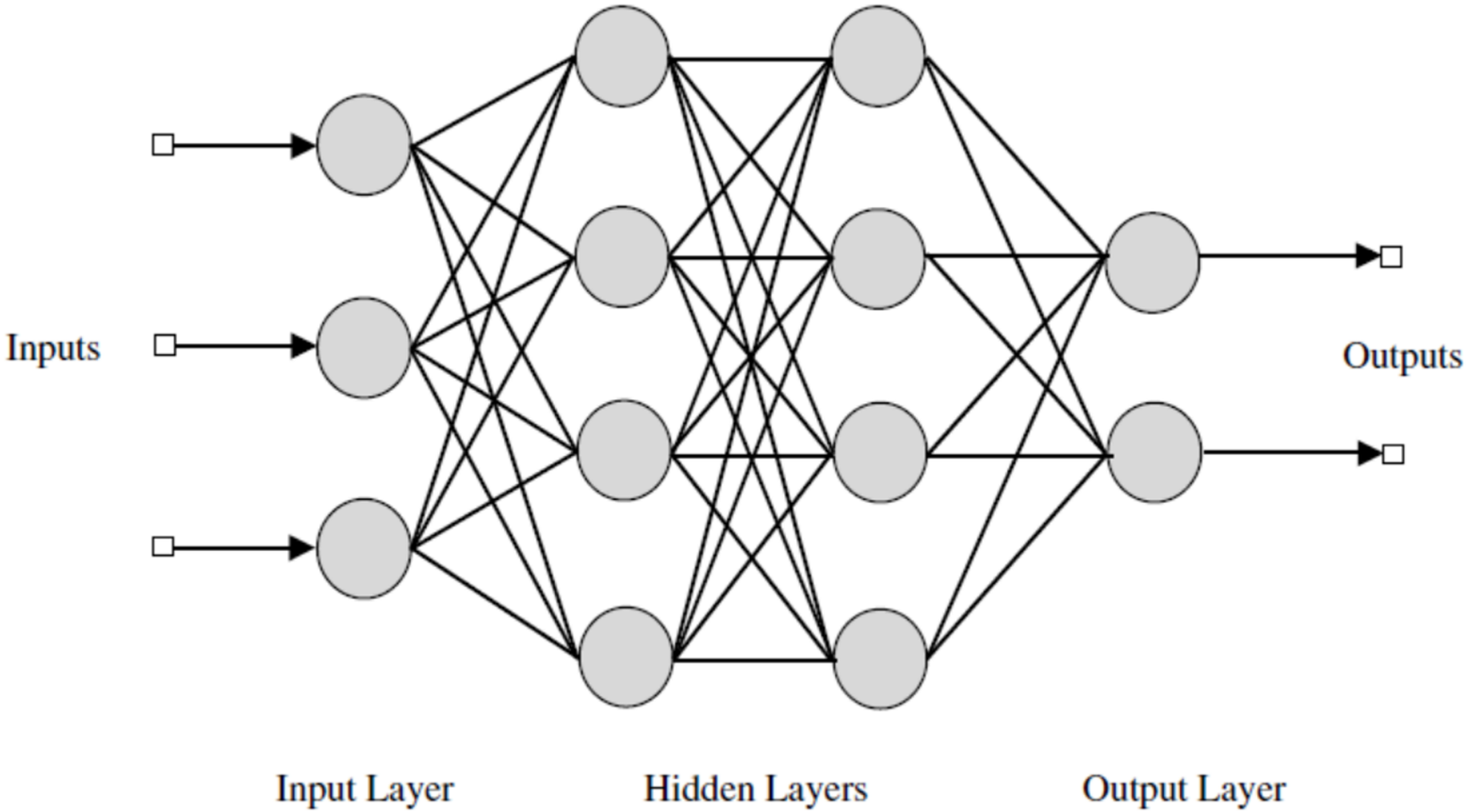
FEED FORWARD NN

- Source



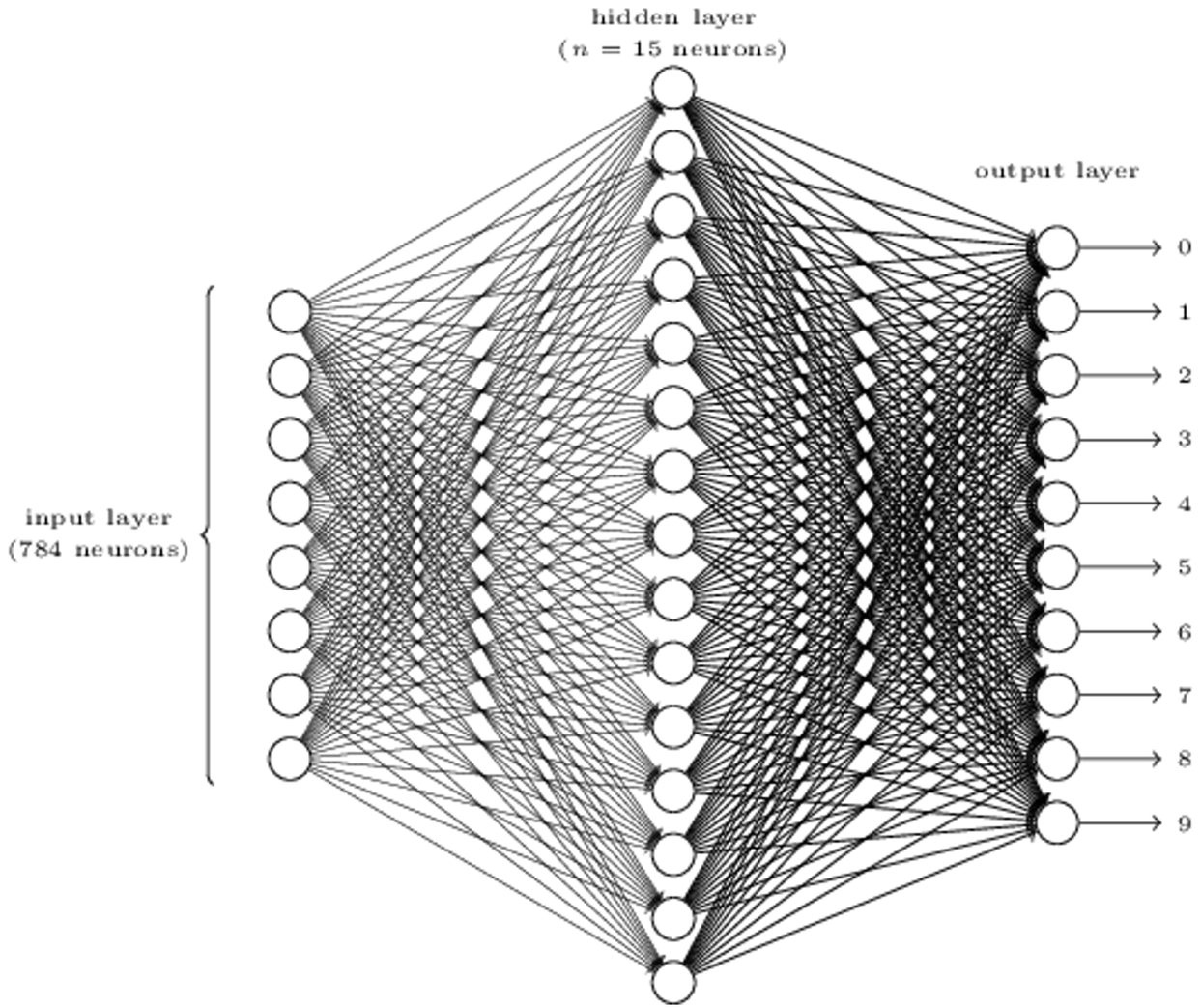
FEED FORWARD NN

- Source



FEED FORWARD NN

- Source



FEED FORWARD NN

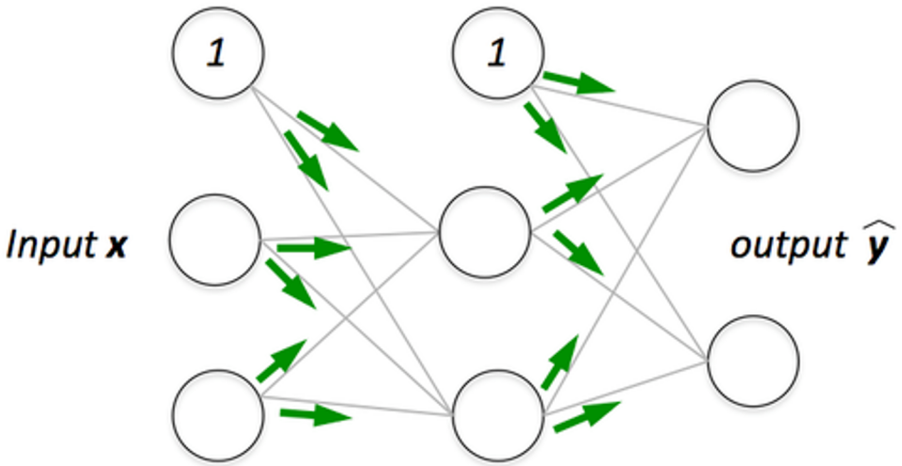
- Typically we use
  - Tanh or logistic layers for input
  - Linear layers for regression output
  - Logistic or Tanh for binary output
  - Softmax for n-class output (yields probabilities)

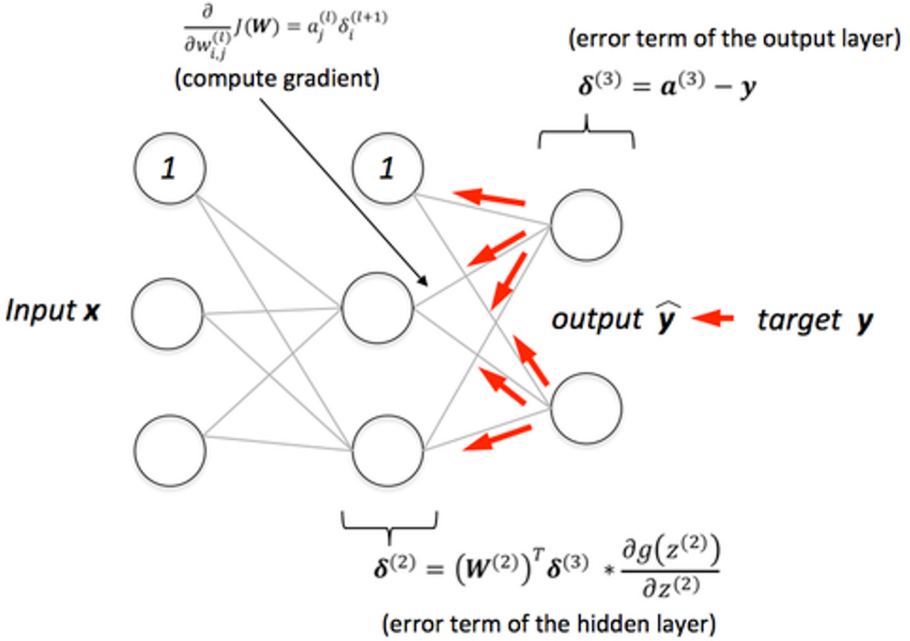
GUIDED PRACTICE

TRAINING

TRAINING

- Feed forward neural networks can be trained with backpropagation
- Source





TRAINING

- Key Parameters
  - Learning Rate (gradient descent for training)
  - Epochs: number of backpropagation passes (over entire dataset)
  - Batch size: how many training points used at a time to update weights

- Model others behaves as usual with
  - model.predict
  - model.predict_classes

TRAINING

- Tips
  - If the error jumps around per epoch, decrease the learning rate
  - Taking too long to train: use higher learning rate or batch_size
  - High error after convergence?
    - More hidden layers / neurons
    - Normalize data or use PCA

UNIVERSAL APPROXIMATION THEORY

UNIVERSAL APPROXIMATION

- One major reason that neural networks are useful is the Universal Approximation Theorem

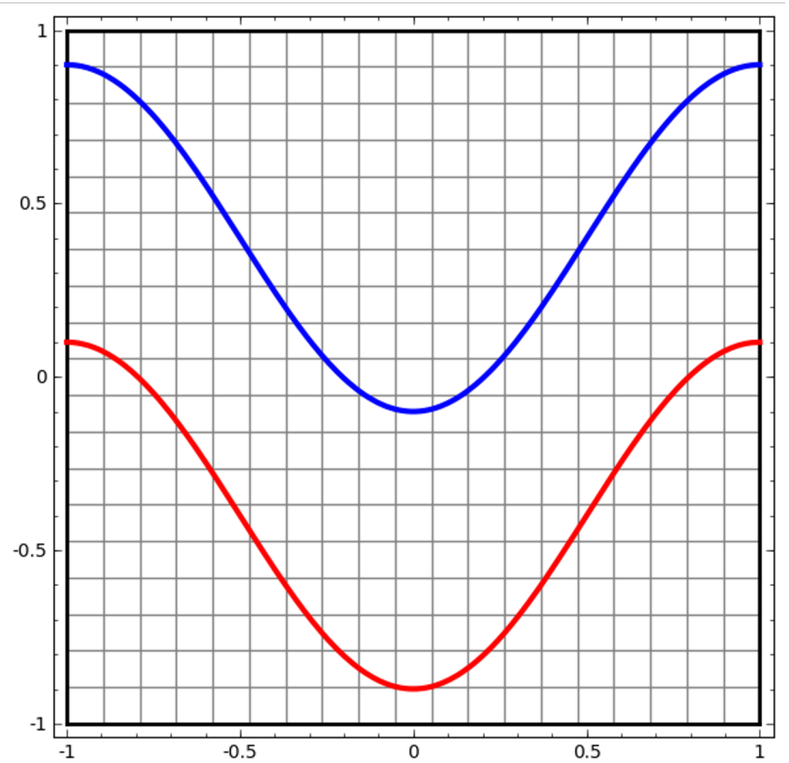
- The result basically says that many real vector-valued functions can be approximated arbitrarily well with *some* feed-forward neural network

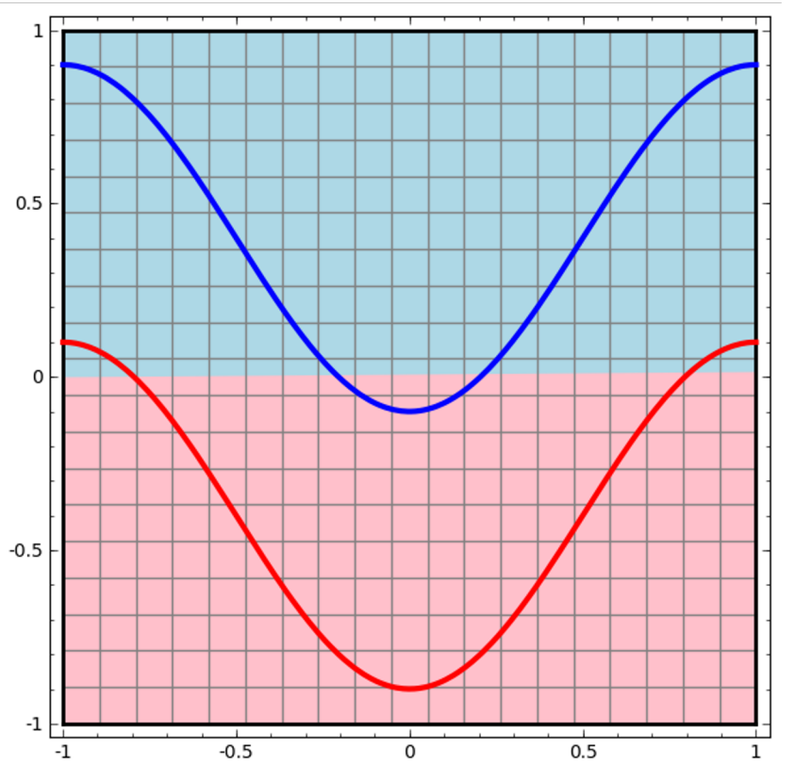
- This is why neural networks are useful for regression -- given enough data and the right network structure they can fit many common data sets

CLASSIFICATION WITH NEURAL NETWORKS

CLASSIFICATION

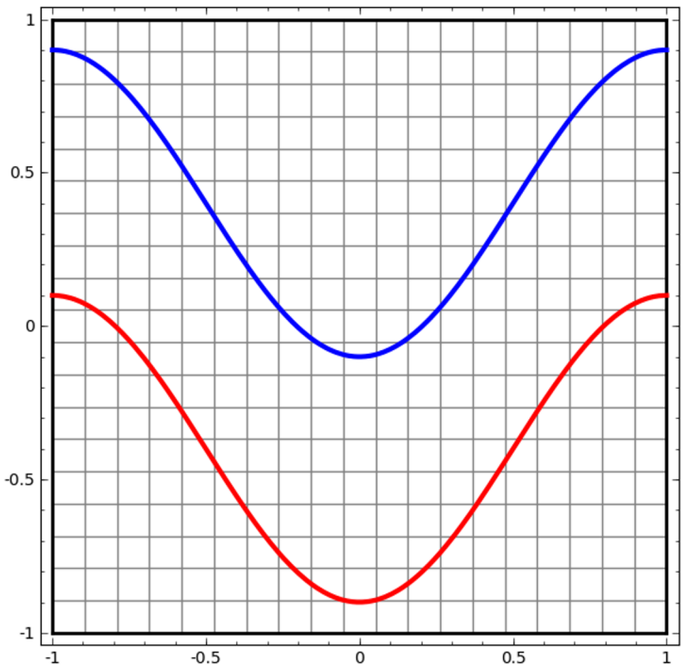
- Neural Networks are also extremely useful for classification (source)
- No hidden layers:

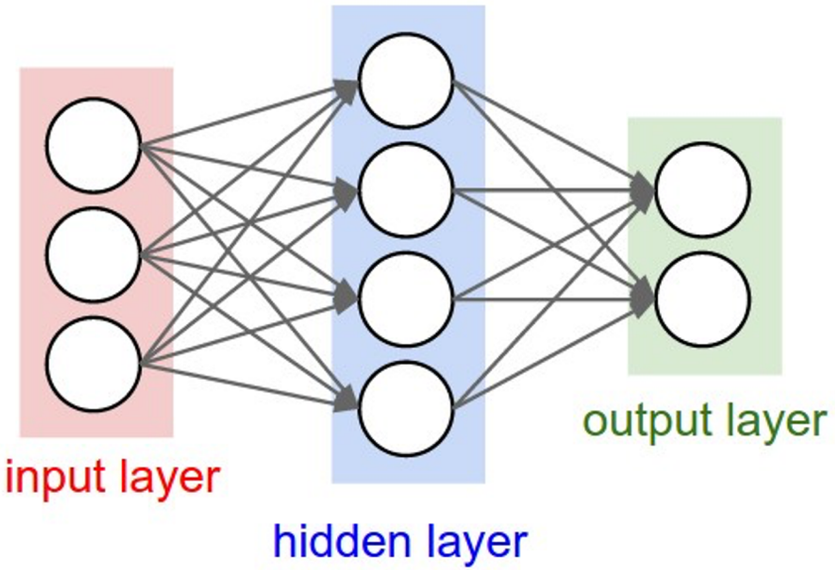


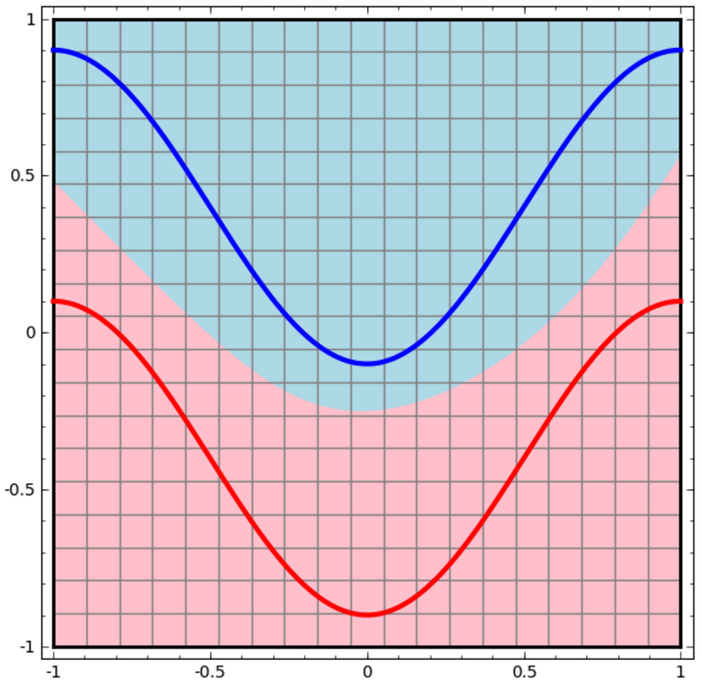


CLASSIFICATION

- Neural Networks are also extremely useful for classification (source)
- One hidden layer:

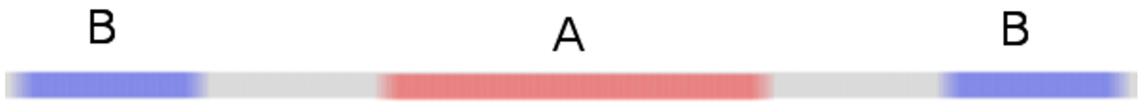


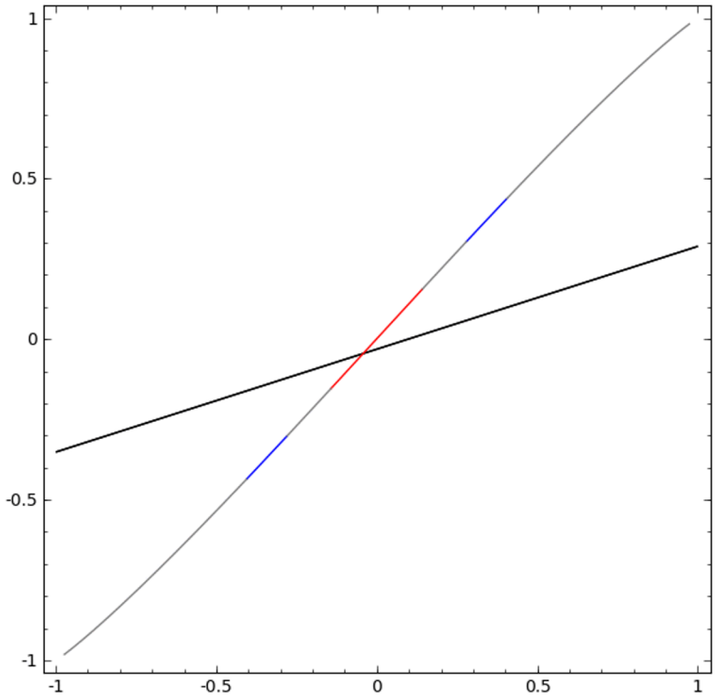




CLASSIFICATION

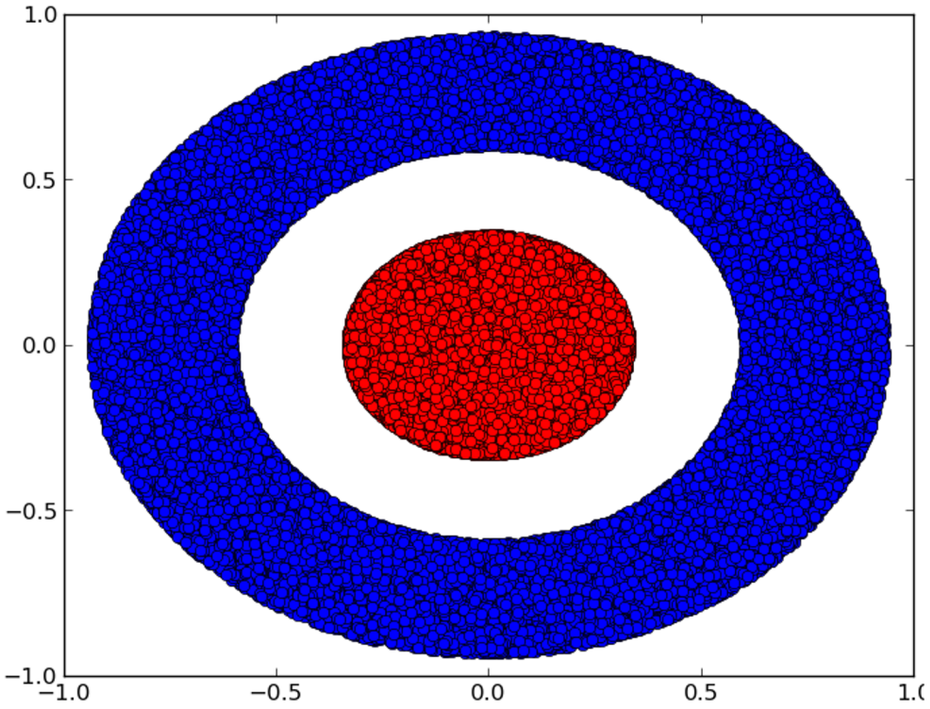
- Neural Networks are also extremely useful for classification (source)

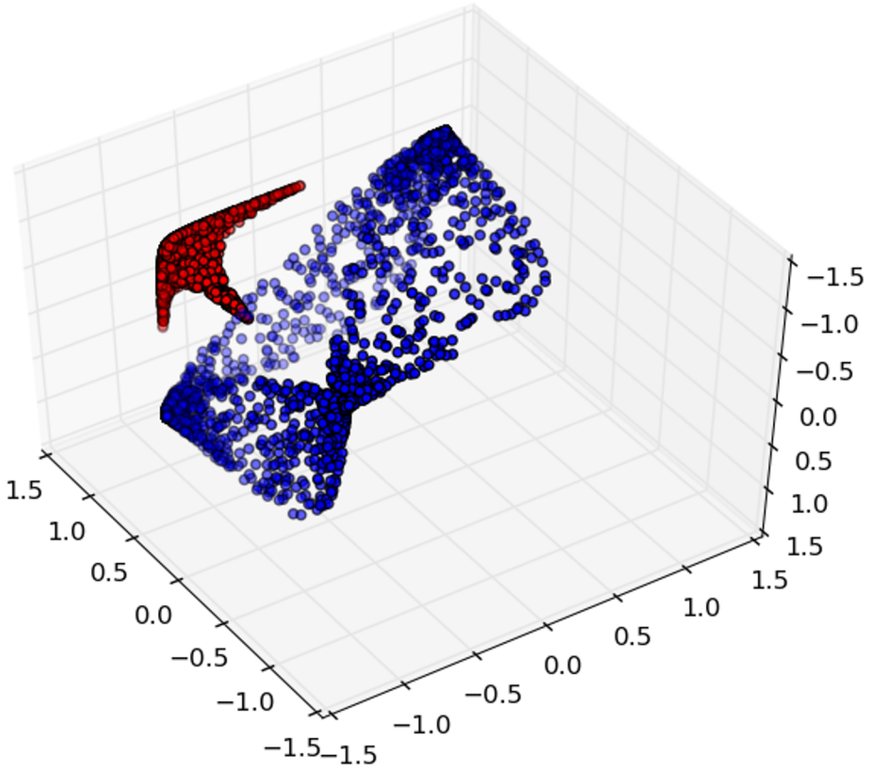




CLASSIFICATION

- Neural Networks are also extremely useful for classification (source)





CLASSIFICATION

- The neural network transforms the data topologically (no tears or breaks) and then separates the data with a hyperplane

- NNs are capable of handling difficult data sets, including:
  - Image processing: recognizing hand-written characters
  - Image compression
  - Financial forecasting
  - Many others

ACTIVITY: KNOWLEDGE CHECK

EXERCISE

ANSWER THE FOLLOWING QUESTIONS

1. Let's practice using neural networks for classification. For each of the four datasets, experiment with the number of layers and neurons to find the best model
2. Also take a look at this visualization

DELIVERABLE

Answers to the above questions

GUIDED PRACTICE

NEURAL NETWORKS IN PYTHON

NN IN PYTHON

- There are many NN libraries for python and other languages
- Python
  - **Theano**
  - **Lasagne**
  - **TensorFlow**
  - **Scikit Learn**
  - **Keras**: A high-level neural networks API, written in Python and capable of running on top of **TensorFlow, CNTK, or Theano**. It was developed with a focus on enabling fast experimentation
- Lua
  - Torch

- Some of these libraries utilize GPUs for (much) faster training

NN IN PYTHON

- Let’s look at some examples in Keras
  - Regression
  - Classification

GUIDED PRACTICE

DESIGNING NEURAL NETWORKS

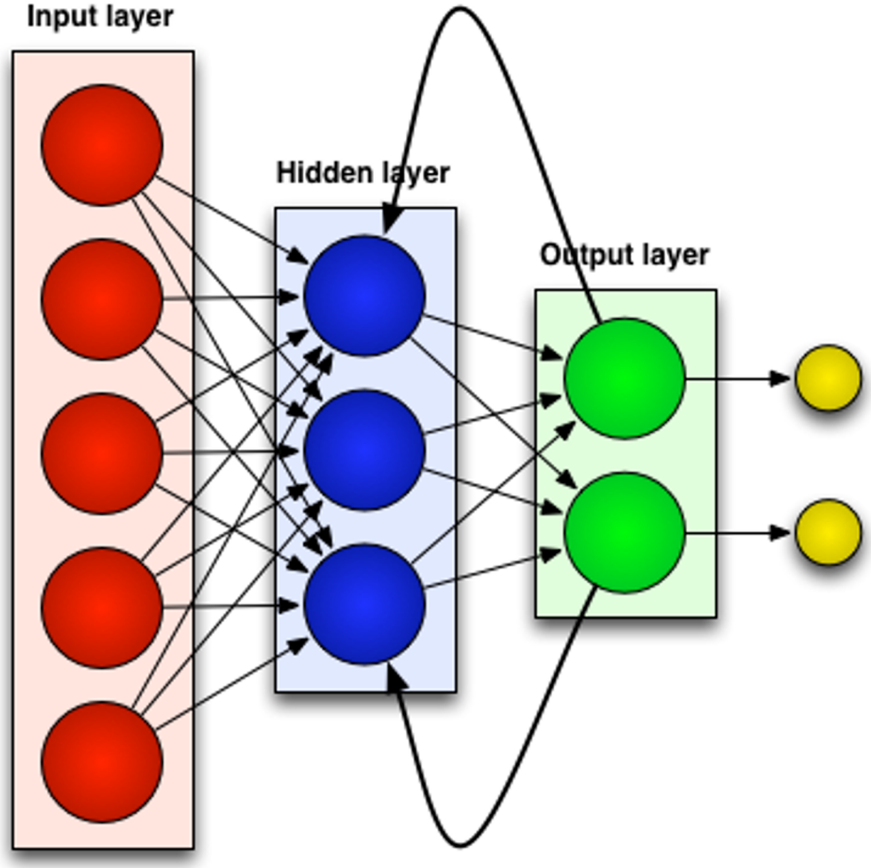
NN IN PYTHON

- Network design is a hard problem
  - Experience helps
  - Evolutionary algorithms are useful for design
  - Nice (free) book available

RECURRENT NEURAL NETWORKS

RECURRENT NEURAL NETWORKS

- Recurrent Neural Networks contain loops (source)



RECURRENT NEURAL NETWORKS

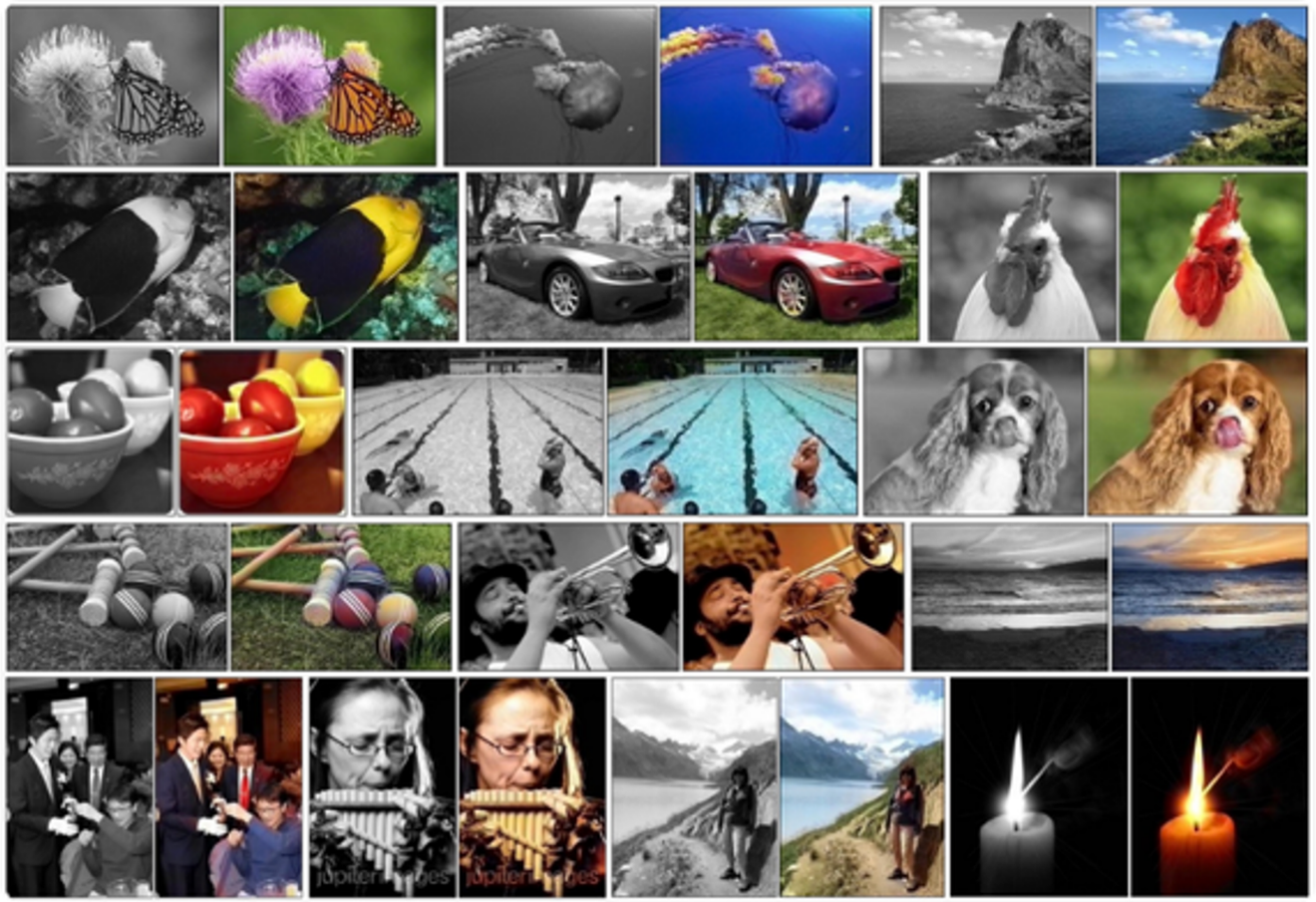
- Recurrent Neural Networks contain loops
- This implements feedback and gives neural networks “memory” or context
- Particularly good for predicting sequences, translating text, recognizing objects in images, speech translation
- Commonly referred to as **deep learning**, involving both feature extraction and modeling
- Nice intro here

RECURRENT NEURAL NETWORKS

- RNN font analysis

RECURRENT NEURAL NETWORKS

- Automatic Colorization with CNN



RECURRENT NEURAL NETWORKS

- RNN font analysis
- Automatic Colorization with CNN
- Automatic translation
- Deep Learning Applications

THINGS TO NOTE

NEURAL NETWORKS: CONCERNS

- **Number of hidden neurons**
  - While the number of inputs and outputs are dictated by the problem, the number of hidden units is not related so explicitly to the application domain
    - The number of hidden units determines the degrees of freedom or expressive power of the model
      - A small number of hidden units may not be sufficient to model complex I/O mappings
      - A large number of hidden units may overfit the training data and prevent the network from generalizing to new examples
  - Despite a number of "rules of thumb" published in the literature, a-priori determination of an appropriate number of hidden units is an unsolved problem
    - The "optimal" number of hidden neurons depends on multiple factors, including number of examples, level of noise in the training set, complexity of the classification problem, number of inputs and outputs, activation functions and training algorithm.
    - In practice, several MLPs are trained and evaluated in order to determine an appropriate number of hidden units
  - A number of adaptive approaches have also been proposed
    - **Constructive** approaches start with a small network and incrementally add hidden neurons (e.g., cascade correlation),
    - **Pruning** approaches, which start with a relatively large network and incrementally remove weights (e.g., optimal brain damage)

NEURAL NETWORKS: CONCERNS

- **Weight decay**
  - To prevent the weights from growing too large (a sign of over-training) it is convenient to add a decay term of the form

$$w(n+1) = (1-\varepsilon)w(n)$$

    - Weights that are not needed eventually decay to zero, whereas necessary weights are continuously updated by back-prop
  - Weight decay is a simple form of regularization, which encourages smoother network mappings [Bishop, 1995]
- **Early stopping**
  - Early stopping can be used to prevent the MLP from over-fitting the training set
  - The stopping point may be determined by monitoring the sum-squared-error of the MLP on a validation set during training

TOPIC REVIEW

CONCLUSION: Neural Networks

Pros:

- Flexible
- Good for a variety of tasks
- Good for many types of data

Cons:

- Can require a lot of data
- Training may be slow
- Many parameters to tune
- Many layer types and activations
- Black Box model

CONCLUSION

- Many more examples for Keras available
- Recommended articles: Convolutional NN,
- Advanced machine learning methods you should explore include Bayesian methods and deep learning

COURSE

BEFORE NEXT CLASS

BEFORE NEXT CLASS

DUE DATE

- Project: Final Project, Part 5!!

LESSON

Q & A

LESSON

EXIT TICKET

DON’T FORGET TO FILL OUT YOUR EXIT TICKET