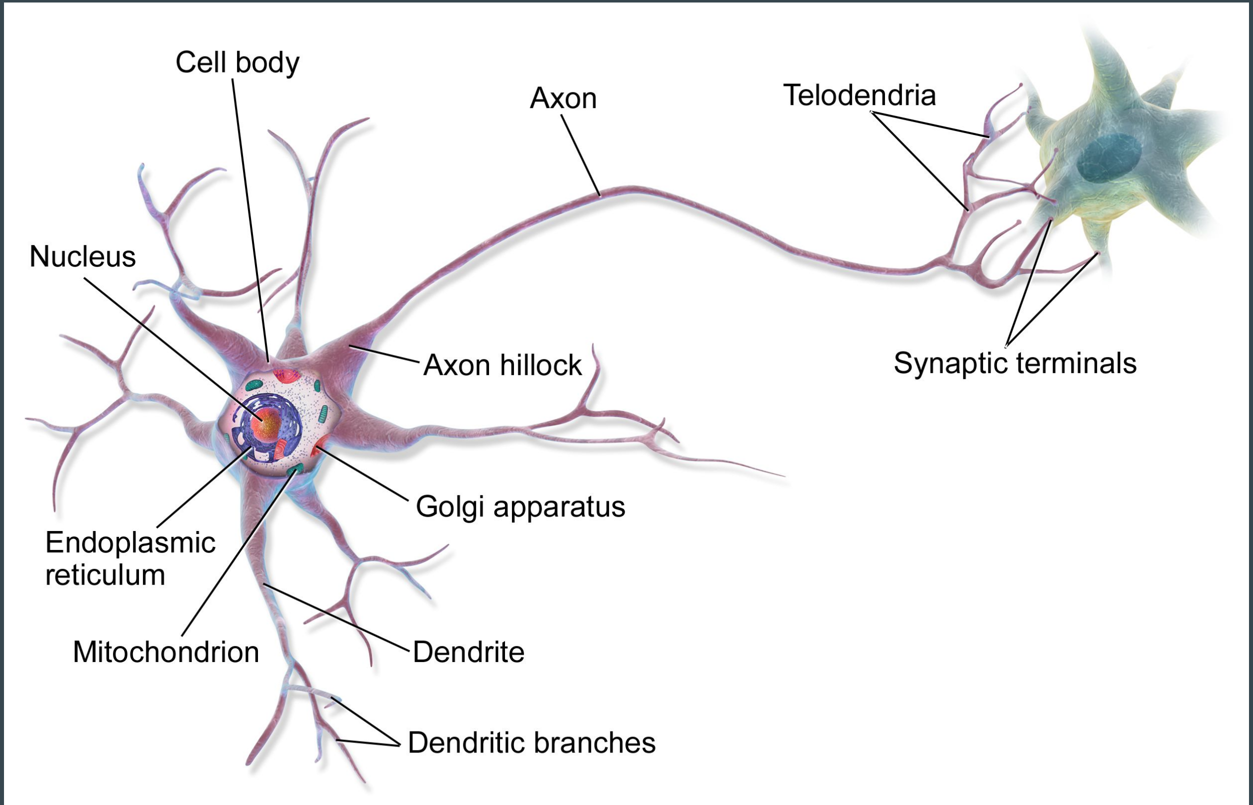
VGP337 - Neural Network & Machine Learning

Instructor: Peter Chan

Biological Neural Networks

- Network of specialized cells called neurons
- Neuron collects signals through fine network of dendrites
- Neuron sends out electrical spikes through axon
- Axon splits into thousands of branches
- At end of each branch, synapse modulates electrical spike for attached neurons

Biological Neuron



Artificial Neural Networks

- Designed to mimic biological neural networks
- Key characteristics:
  - Large number of neurons
  - Highly interconnected
  - Work in unison
  - Solve a common, specific problem
- Usually includes:
  - Learn by example

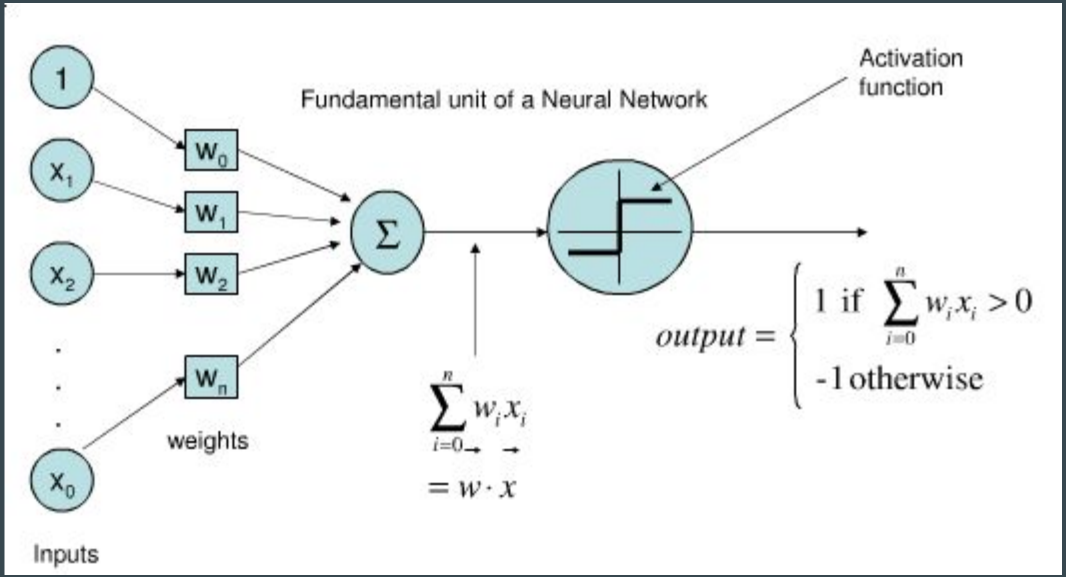
Artificial Neural Networks

- Common uses:
  - Pattern recognition (gesture, text, handwriting, speech, facial, etc)
  - Data classification
  - Game playing (backgammon, chess, etc)
  - Etc

The Neuron

- The basic building block of an ANN
- Consists of a set of weights for its input and an activation function f(x)
- The activation function returns a value based on the sum of the weighted inputs and possibly a threshold T
- When the activation function returns a binary value (0 or 1), the neuron becomes a binary classifier and is commonly referred as the Perceptron

The Perceptron



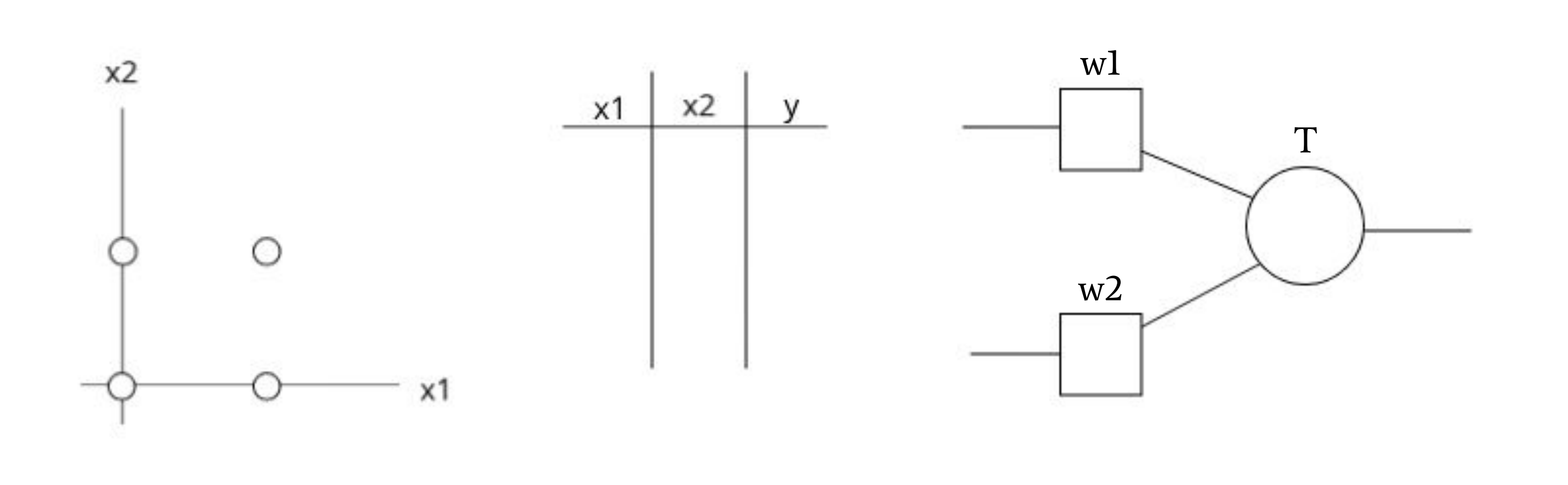
* The threshold T in this case is folded into the summation as w_o

How Powerful is a Perceptron?

- A single Perceptron is essentially a linear function to its inputs
- Therefore, it can be used to classify any data that is linearly separable
- This includes logical operations such as OR, AND, and NOT

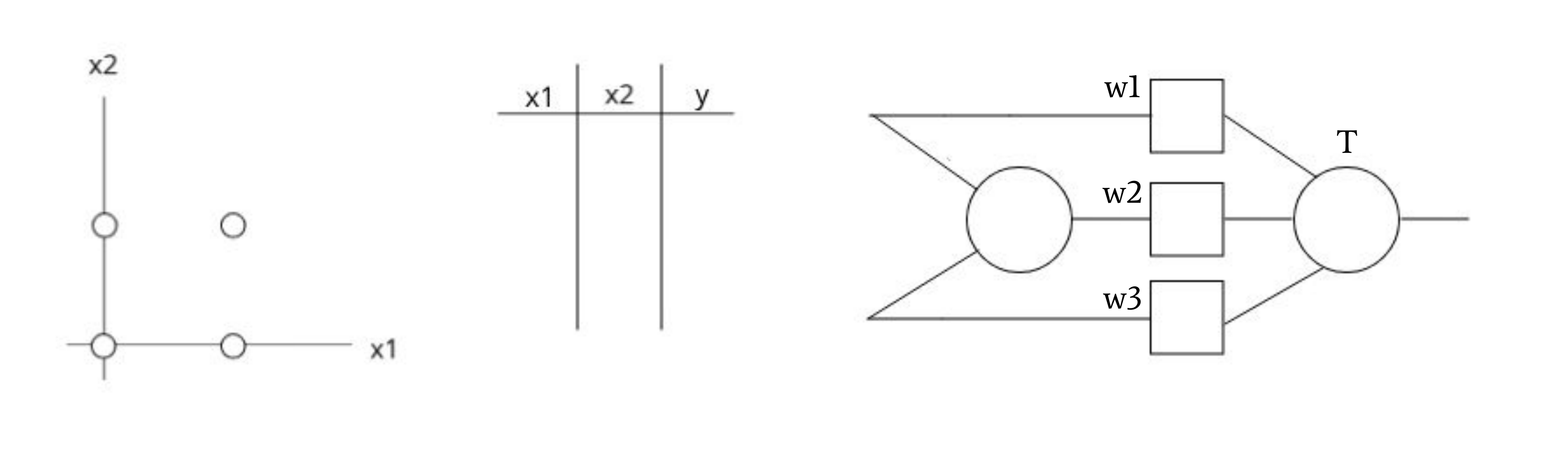
How Powerful is a Perceptron?

- What should you choose for w1, w2, and T?



How Powerful is a Perceptron?

- What about XOR?
- It turns out that XOR is non-linear and cannot be model using a single unit
- Instead, we need to form a network to model non-linear classifiers



Perceptron Training

- In practice, instead of hand picking weights for the perceptron, we want an algorithm that finds weights that map inputs to outputs given a set of examples
- We will look at two rules:
  - Perceptron Rule (Thresholded)
  - Gradient Descent / Delta Rule (Not Thresholded)

Perceptron Rule

- Originally developed by Frank Rosenblatt in the late 1950s
- Training data are presented to a single unit to output an output, then the weights are modified by an amount proportional to the error
- The training set with *s* samples is defined as:

$$D = \{(x_1,d_1), \ldots , (x_s,d_s)\}$$

where

x_i is the n-dimension input vector
d_i is the desired output value for that input

Perceptron Rule

1. Initialize the weights and threshold to small random numbers
2. Present a vector x to the neuron inputs and calculate the output
3. Update the weights according to:

   $$w_j(t + 1) = w_j(t) + L * (d - y_i)x_{ij}$$

   where

   w_j is the j^{th} weight for the perceptron
   t is the iteration number
   L is the learning rate (0.0, 1.0]
   y is the computed output which is either 0 or 1

4. Repeat 2 and 3 until error is less than some threshold or up to iteration count

Perceptron Rule

- Note that the formula only adjusts the weight if an error is made, otherwise the value is unchanged
- The threshold can be folded into the weights by adding a bias value of 1 into the inputs, namely $x_i = \{x_{i1}, x_{i2}, \dots , x_{in}, 1\}$
- As mentioned, the perceptron is a linear classifier. If the training set is indeed linearly separable, then the algorithm is guaranteed to have a finite convergence. Meaning that it will always find an answer given enough iterations.
- However, if the data is not linearly separable, the perceptron rule may run forever!

Delta Rule

- We need a more robust algorithm to approximate the real concept using *gradient descent search*, which is based on Calculus
- The key idea is to minimize the error function:

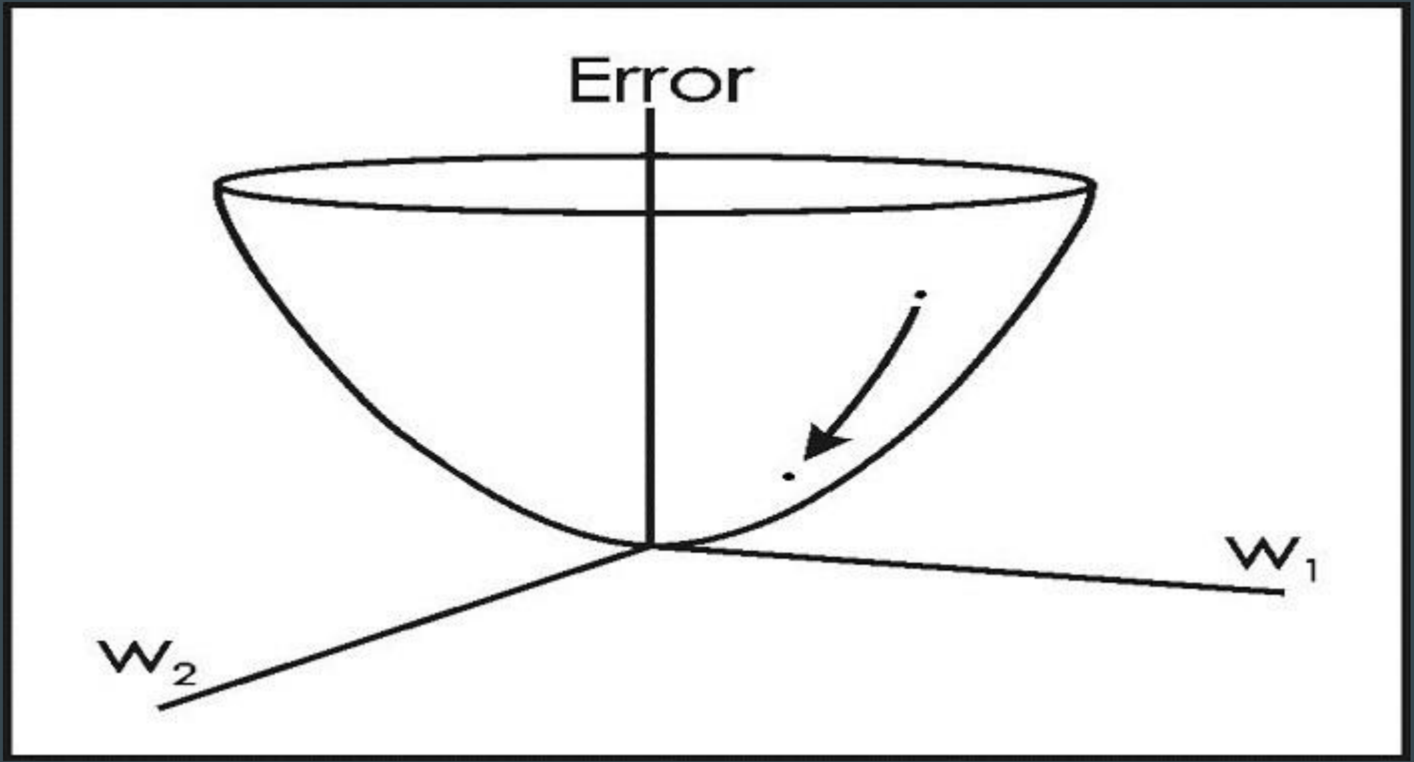
$$E(w) = ½ \Sigma (d – y)^2$$

- After some calculus magic (https://en.wikipedia.org/wiki/Delta_rule), we have a very similar rule for updating the weights:

$$w_j(t + 1) = w_j(t) + L * (d - a_i)x_{ij}$$

where a is the activation: $a = \Sigma (w_j * x_j)$

Delta Rule

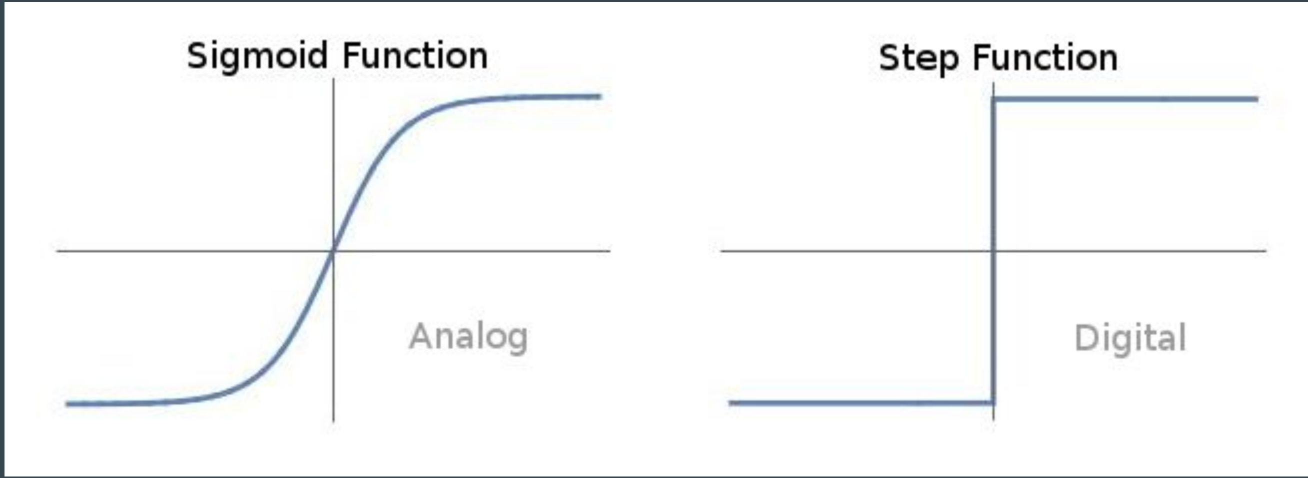


Comparison

- The perceptron rule is thresholded (i.e. based on an output from a step function) whereas the delta rule uses the activation value directly
- The perceptron rule is guaranteed to converge if the data is linearly separable
- The delta rule converges in the limit but is stable for non-linear data
- However, the delta rule relies on the derivative and the step function is not differentiable

Sigmoid Function

- To address this, we can replace the step function with something close to it but is differentiable
- A common choice for this is the Sigmoid Function



Sigmoid Function

- A sigmoid function is a mathematical function having a characteristic “S” shaped curve
- As x goes to ∞, y goes to 1
- As x goes to -∞, y goes to 0

$$y = \frac{1}{1 + e^{-x}}.$$

It has derivative

$$\frac{dy}{dx} = [1 - y(x)]\, y(x)$$

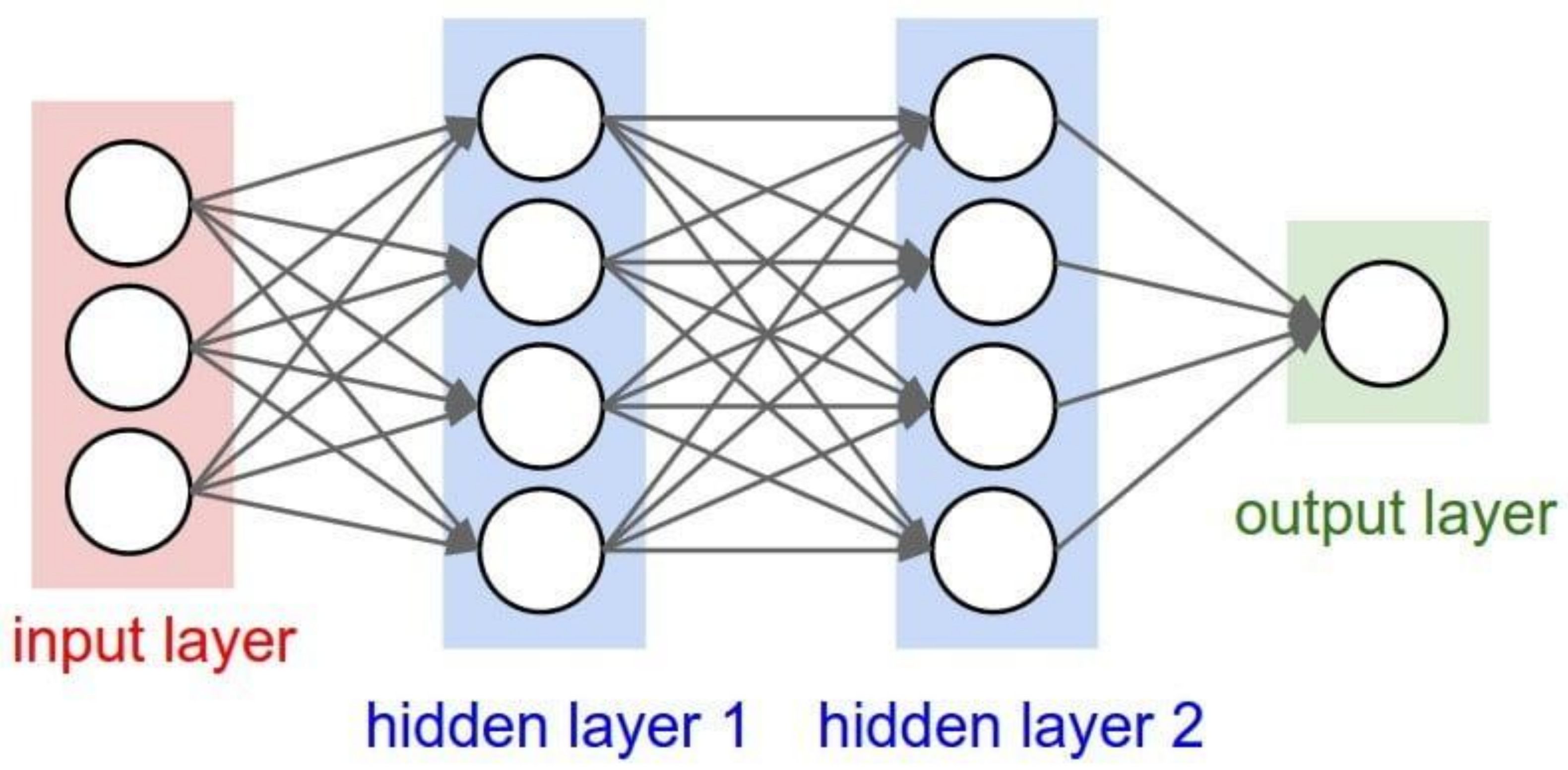
$$= \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$= \frac{e^{x}}{(1 + e^{x})^2}$$

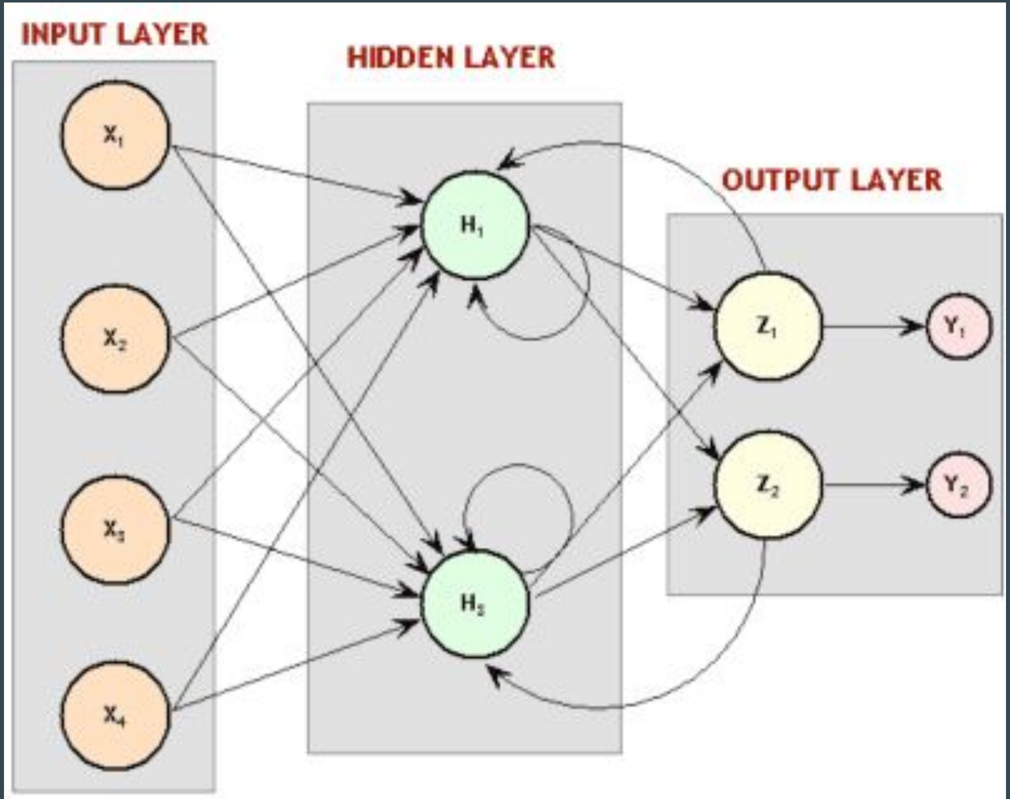
Artificial Neural Networks

- Now that we understand how we can build and train a single perceptron, we have the foundation to build a full network that can be used to model more complex functions.
- Typically, artificial neurons are aggregated into layers. Different layers may perform different kinds of transformations on their input.
- Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing multiple hidden layers.
- A feedforward neural network is an ANN wherein connections do not form a cycle.
- A recurrent neural network on the other hand contains loops (like memory) and can exhibit temporal dynamic behavior

Feedforward Neural Networks



Recurrent Neural Networks



Backpropagation

- Since each neuron in the network can be trained using the delta rule due the a differentiable sigmoid function, the entire network is then also differentiable.
- An extension to the delta rule leads to the idea of backpropagation to calculate a gradient that is needed in the calculation of the weights used in the network, made possible by using the chain rule to iteratively compute gradients for each layer.
- The method requires the derivative of the loss function with respect to the network output to be known.

References

Perceptron

Gradient Descent

Backpropagation

Neural Net in C++ Tutorial

Basic Neural Network Tutorial : C++ Implementation and Source Code

Basic classification: Classify images of clothing