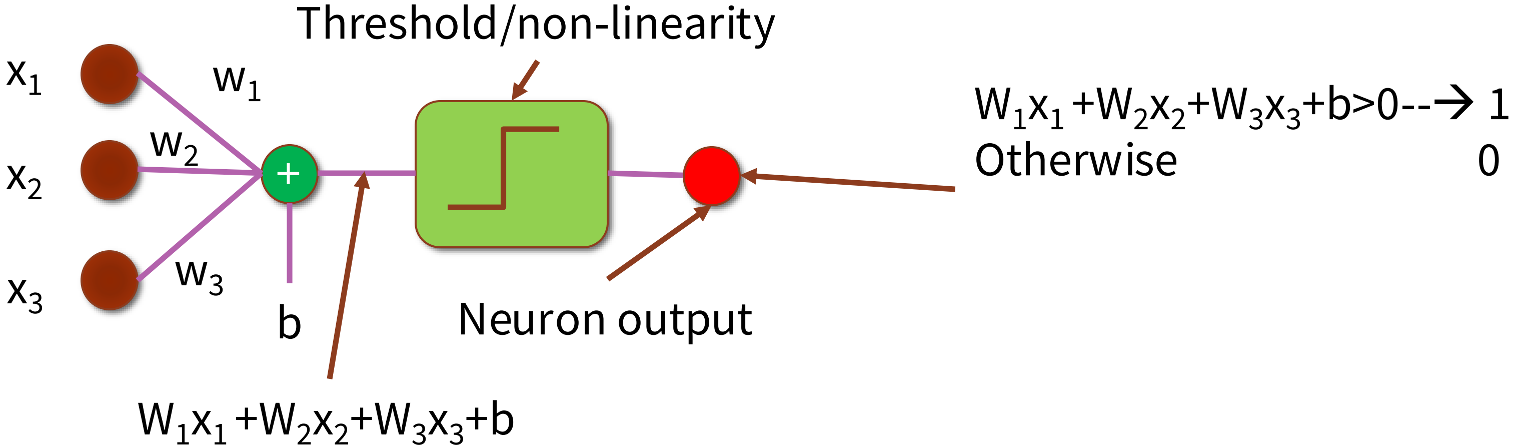
Carnegie Mellon University

Introduction to Deep Learning for Engineers

Spring 2025, Introduction to Deep Learning for Engineers
Jan 16, 2025, Second Session

Amir Barati Farimani
Associate Professor of Mechanical Engineering and Bio-Engineering
Carnegie Mellon University

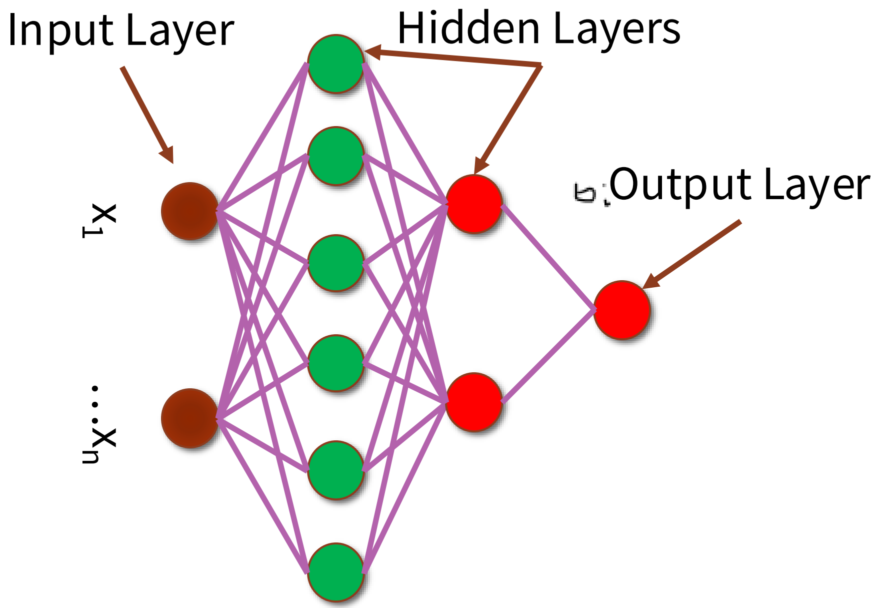
Elements of Perceptron



The bias can also be viewed as the weight of another input component that is always set to 1

If the bias is not explicitly mentioned, we will implicitly be assuming that every perceptron has an additional input that is always fixed at 1

Elements of Neural Network

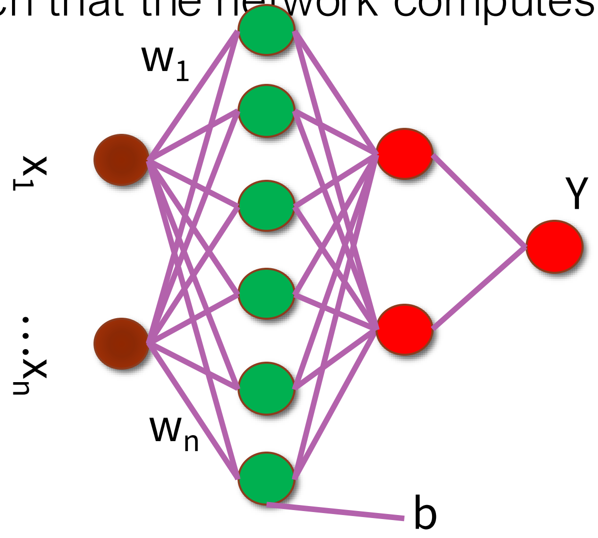


- We will assume a feed-forward network
  - – No loops: Neuron outputs do not feed back to their inputs directly or indirectly
  - – Loopy networks are a future topic
- Part of the design of a network: The architecture

– How many layers/neurons, which neuron connects to which and how,etc

What to learn? Parameters of Network

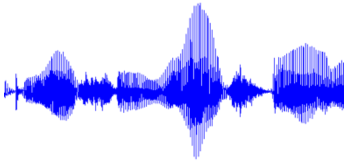
- Given: the architecture of the network
- The parameters of the network: The weights and biases – (pink connections)
- Learning the network : Determining the values of these parameters such that the network computes the desired function



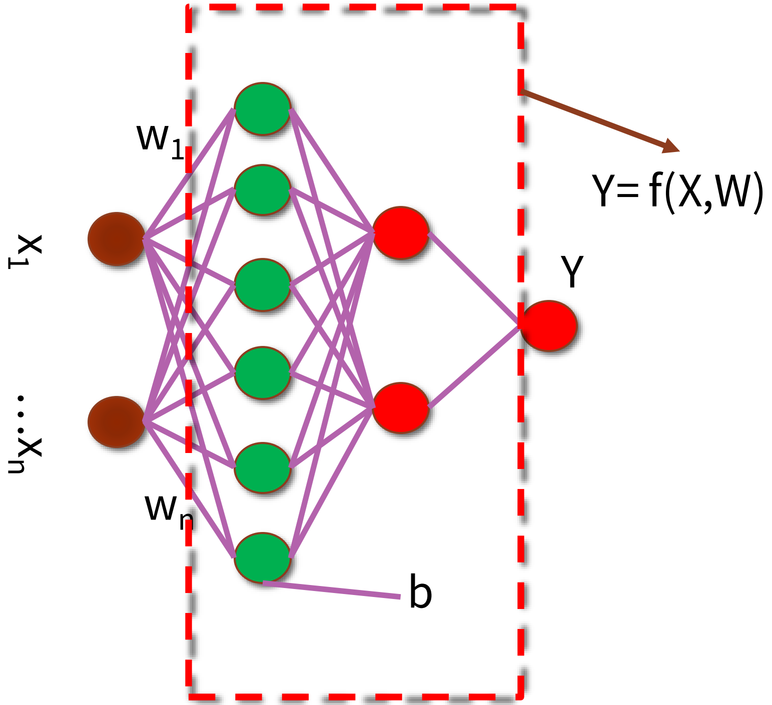
$Y = f(X,W)$

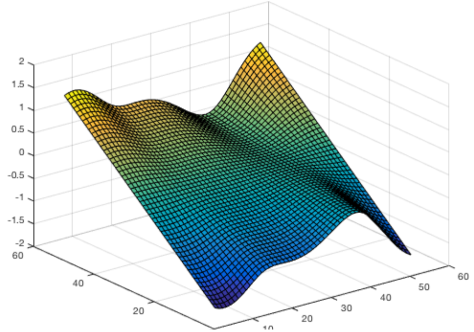
The network is a function f() with parameters W which must be set to the appropriate values to get the desired behavior from the net

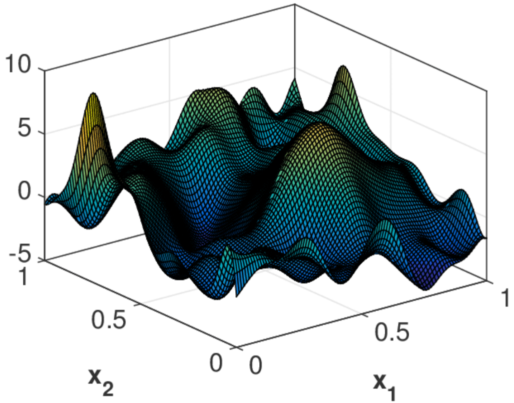
A function mapping the voice to text for Drivethru orders



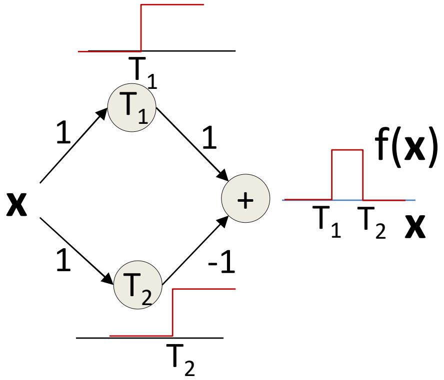
MLP is Universal Function Approximator, but how?

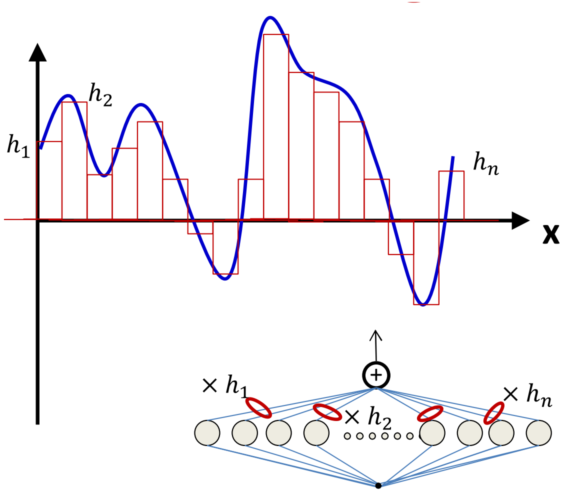






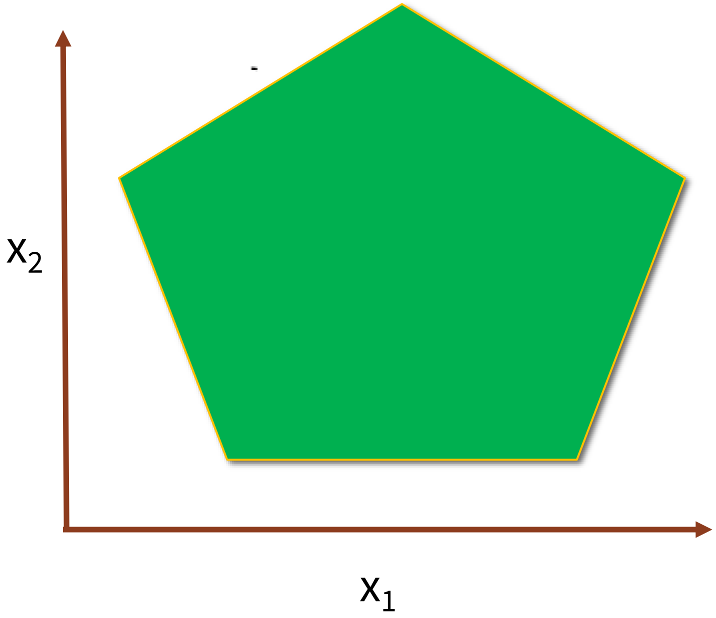
Method 1: build by hand

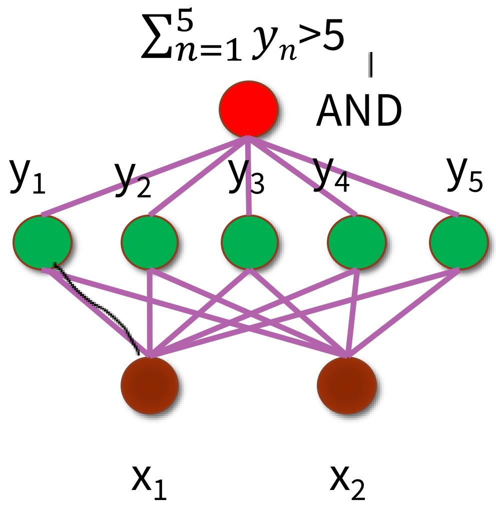




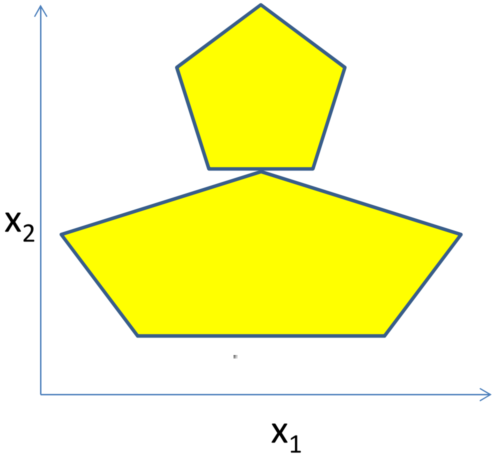
Source: 11785 lecture notes

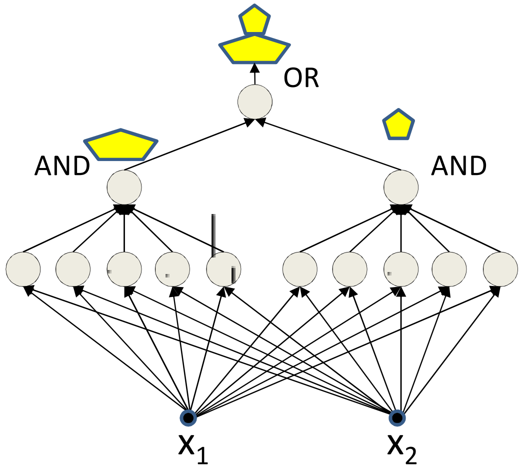
Complex Decision Boundaries





Complex Decision Boundaries

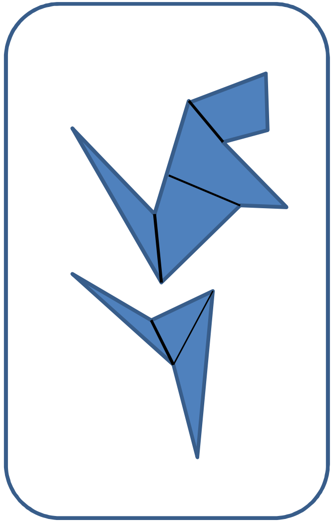


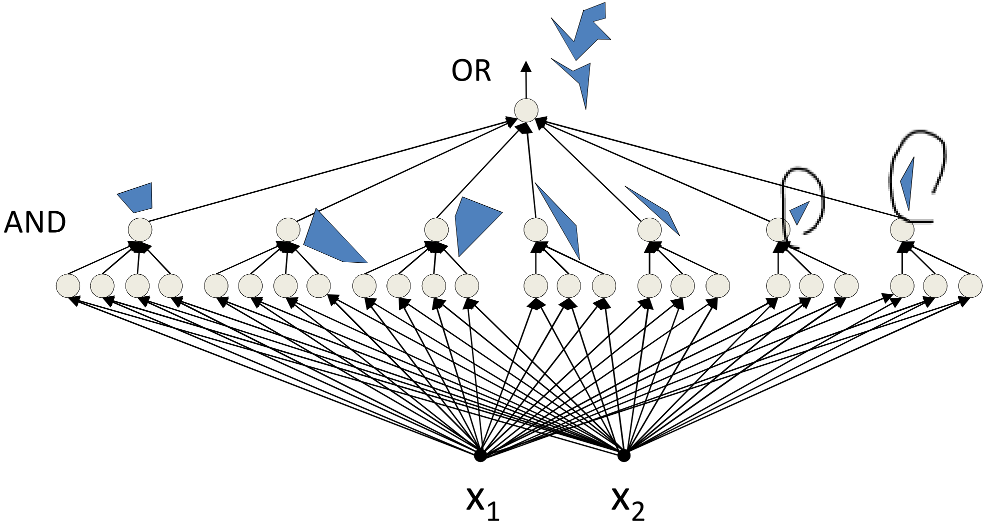


- Network to fire if the input is in the yellow area
  - "OR" two polygons
  - A third layer is required

Source: 11785 lecture notes

Complex Decision Boundaries

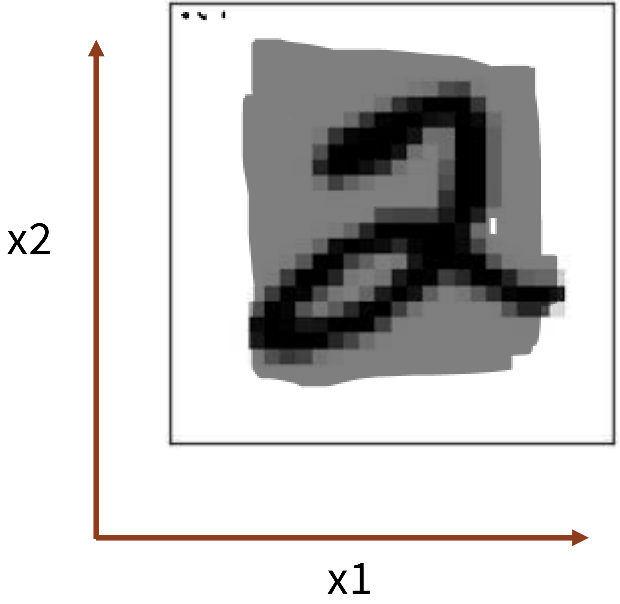


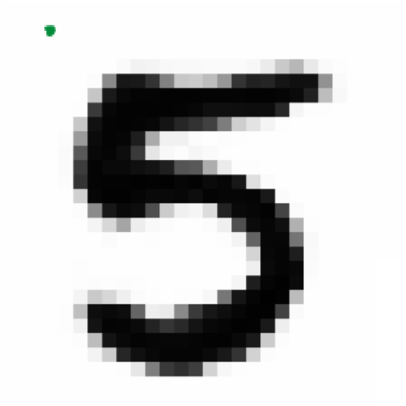


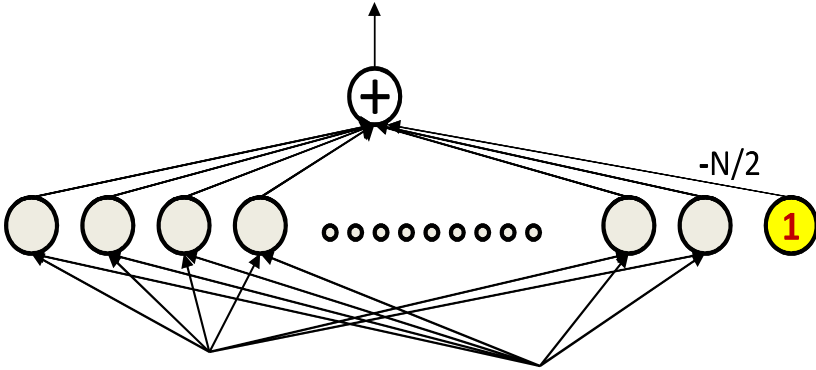
- Can compose *arbitrarily* complex decision boundaries

Source: 11785 lecture notes

Complex Decision Boundaries

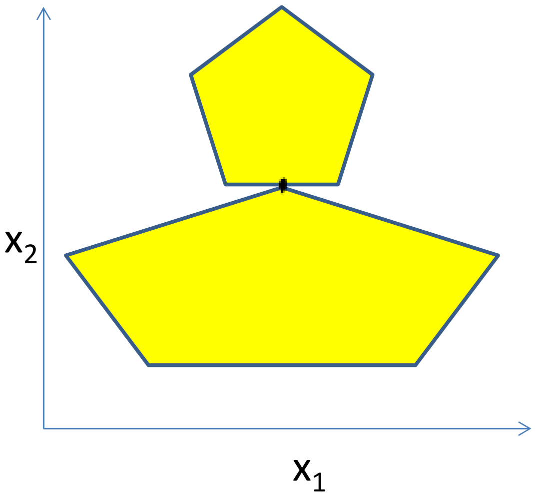


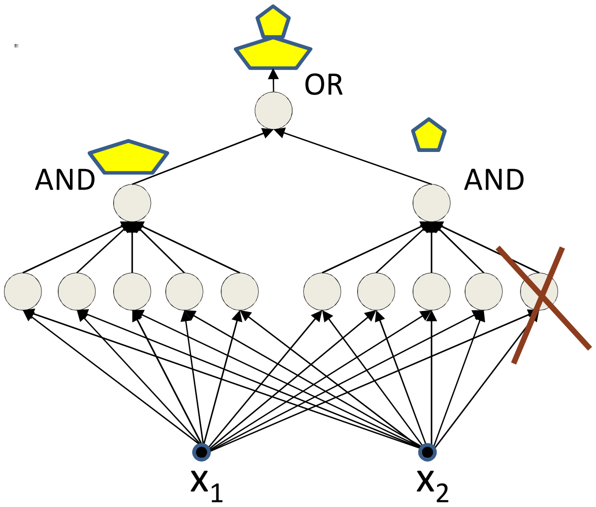




Sufficiency of the Architecture

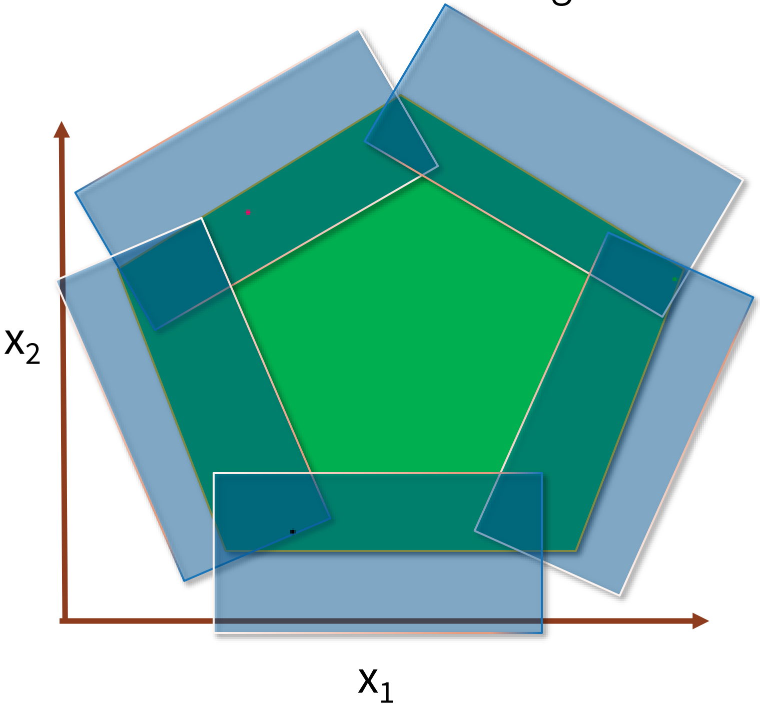
- Number of nodes in a layer (this depends also on the activation function)

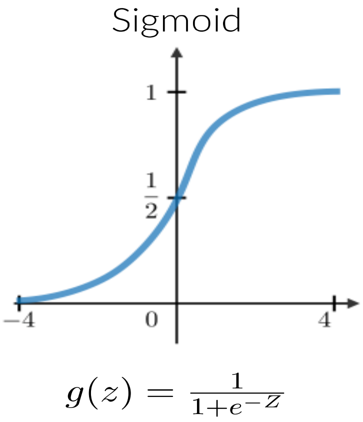


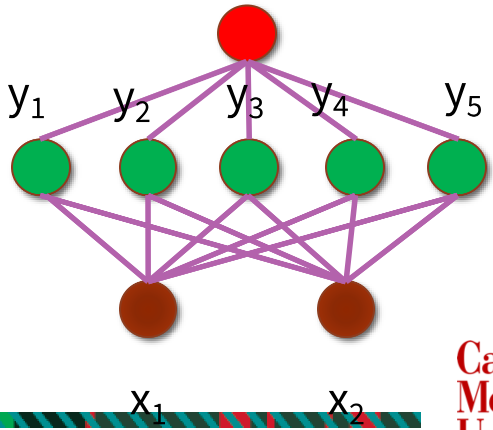


Sufficiency of the Architecture

- This is because of having threshold activation function







Width and Depth vs Activation

- Previous discussion showed that a single-layer MLP is a universal function approximator

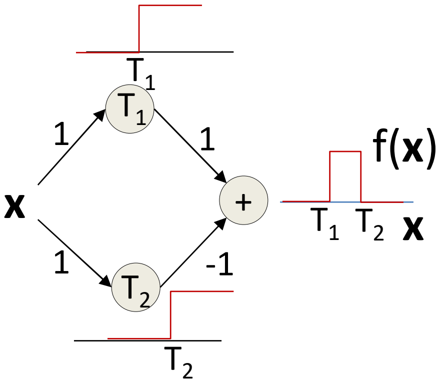
- Can approximate any function to arbitrary precision – But may require infinite neurons in the layer

- More generally, deeper networks will require far fewer neurons for the same approximation error

- Narrow layers can still pass information to subsequent layers if the activation function is sufficiently graded

Carnegie Mellon University

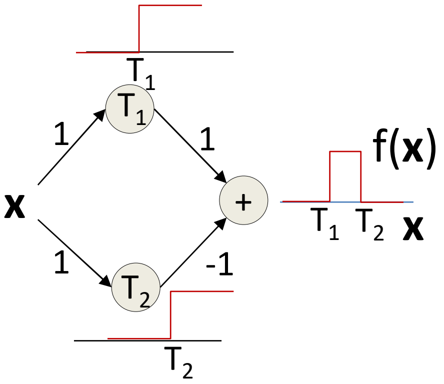
Continuous Output (Regression)

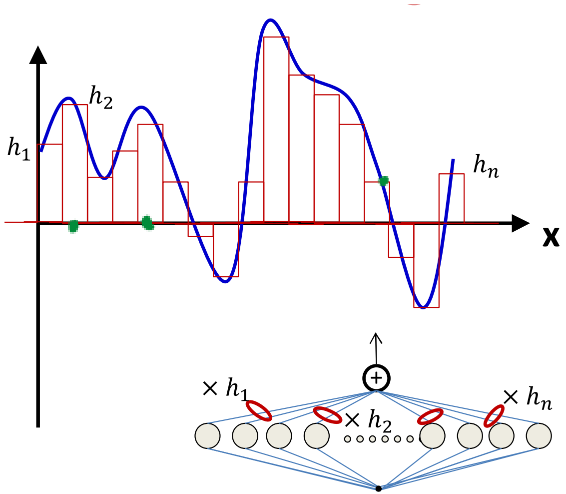


A simple 3-unit MLP with a "summing" output unit can generate a "square pulse" over an input

– Output is 1 only if the input lies between T_1 and T_2 – T_1 and T_2 can be arbitrarily specified

Continuous Output (Regression)



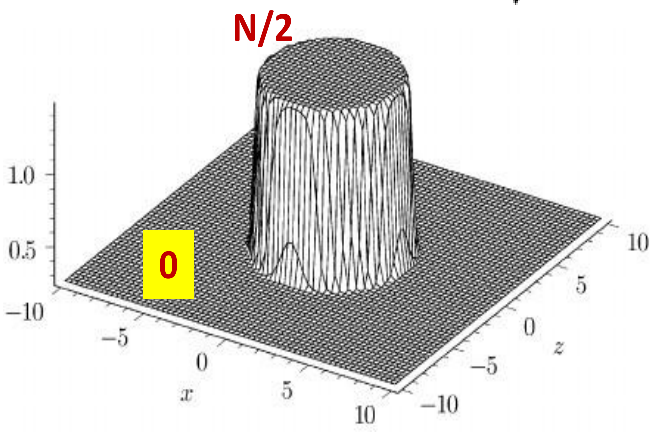


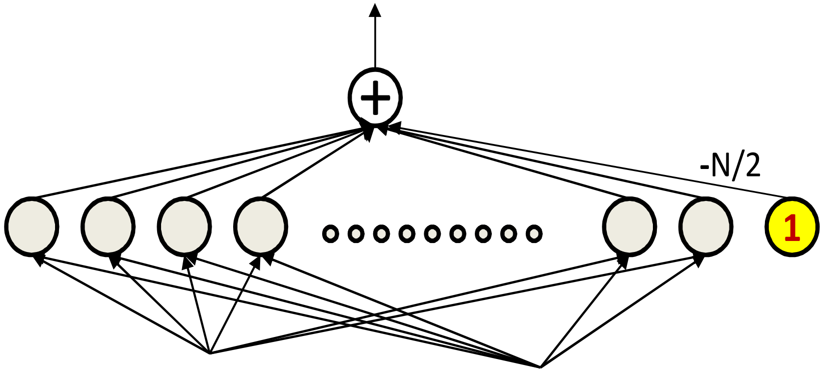
A simple 3-unit MLP with a “summing” output unit can generate a “square pulse” over an input

– Output is 1 only if the input lies between T_1 and T_2 – T_1 and T_2 can be arbitrarily specified

Source: 11785 lecture notes

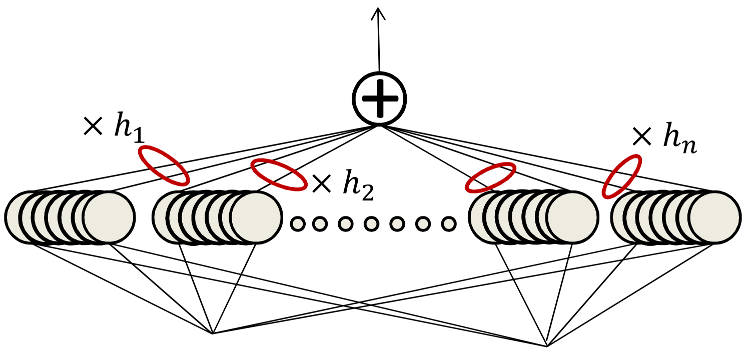
Continuous Output (higher dimension)

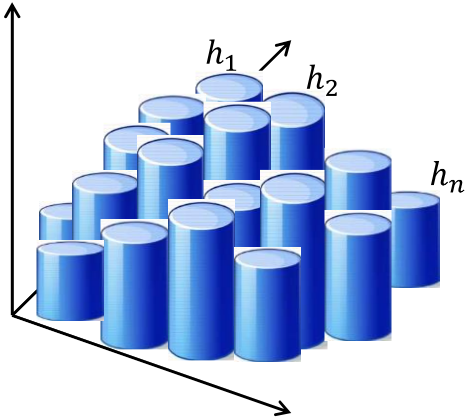




Source: 11785 lecture notes

Continuous Output (higher dimension)



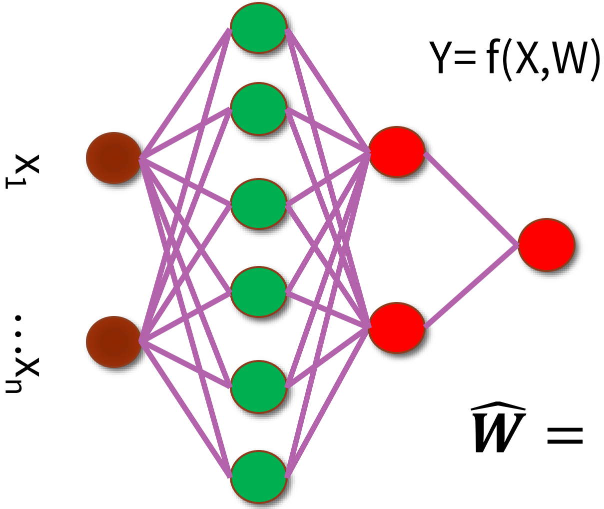


- MLPs can actually compose arbitrary functions in any number of dimensions!
  - Even with only one layer
    - As sums of scaled and shifted cylinders
  - To arbitrary precision
    - By making the cylinders thinner
  - **The MLP is a universal approximator!**

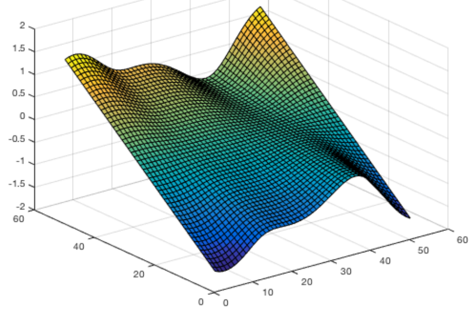
Source: 11785 lecture notes

Method 2: Automatic Parametrization

More generally, given the function to model, we can derive the parameters of the network to model it, through computation

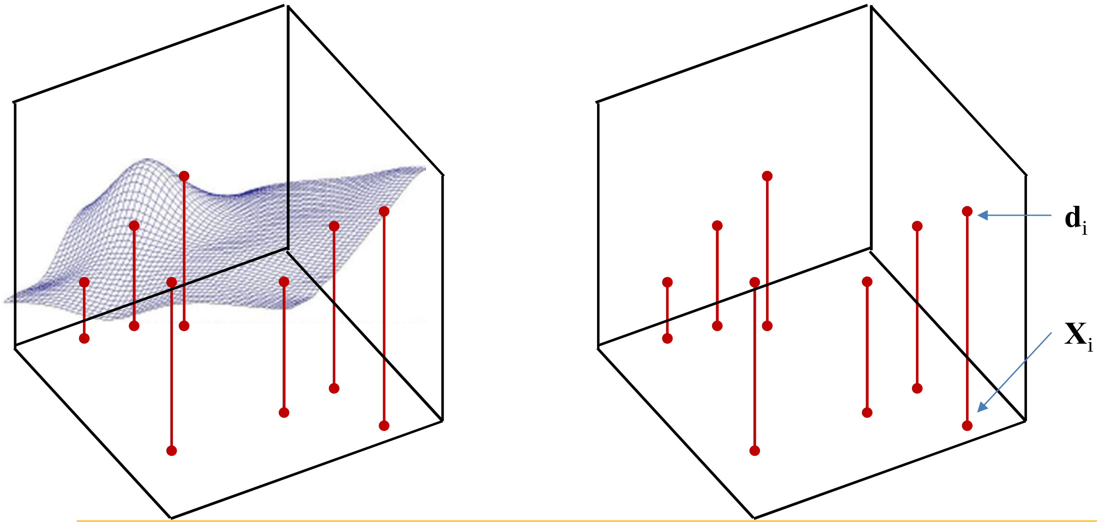


G(X)



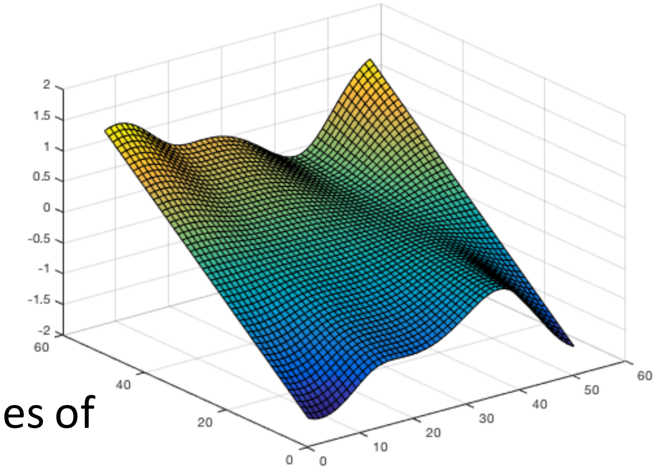
$$\widehat{\boldsymbol{W}} = \underset{W}{\operatorname{argmin}} \int_X div\big(f(X;W), g(X)\big) dX$$

Learning from Samples



- We must ***learn*** the *entire* function from these few examples
  - The "training" samples

g(X) is unknown



Function must be fully specified – Known everywhere, i.e. for every input

- In practice we will not have such specification

23

Sample

– Basically, get input-output pairs for a number of samples of input

– Good sampling: the samples of will be drawn from

- Very easy to do in most problems: just gather training data – E.g. set of images and their class labels

– E.g. speech recordings and their transcription

What we have learnt so far

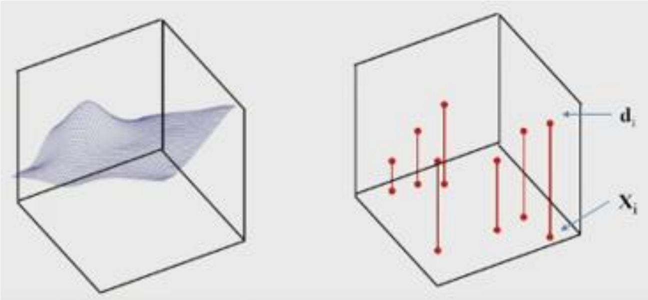
- Previous discussion showed that a single-layer MLP is a universal function approximator

- Can approximate any function to arbitrary precision – But may require infinite neurons in the layer

- More generally, deeper networks will require far fewer neurons for the same approximation error

Carnegie Mellon University

The Empirical risk



- The empirical estimate of the expected error is the average error over the samples

$$E[div(f(X;W),g(X))] \simeq \frac{1}{T}\sum_{!"\ \#}^{\$} div(f(X_{!};W),d_{!})$$

- This approximation is an unbiased estimate of the expected divergence that we actually want to estimate

 -- Assumption: minimizing the empirical loss will minimize the true loss

Problem Statement

- Given a training set of input-output pairs

$(X_{\#}, d_{\#}),\ (X_{\%}, d_{\%}),\ \ldots,\ (X_{\&}, d_{\&}),$

- Minimum the following function

$$Loss(W) = \frac{\#}{\$} \sum_{!} div(f(X_{!}; W), d_{!})$$, w.r.t W

- This is problem of function minimization
  --An instance of optimization

Problem Setup: Things to define

- Given a training set of input-output pairs

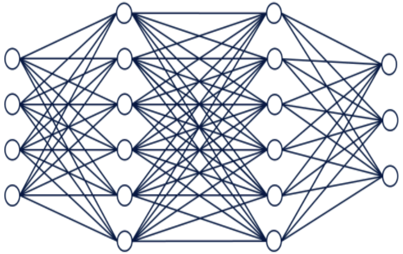
$$(X_{\#}, d_{\#}), (X_{\%}, d_{\%}), \ldots, (X_{\&}, d_{\&}),$$

what are input-output pairs?

$$Loss(W) = \frac{\#}{\$} \sum_{!} div(f(X_{!}; W), d_{!})$$

What is the divergence div()?
What is f() and what are its parameters W?

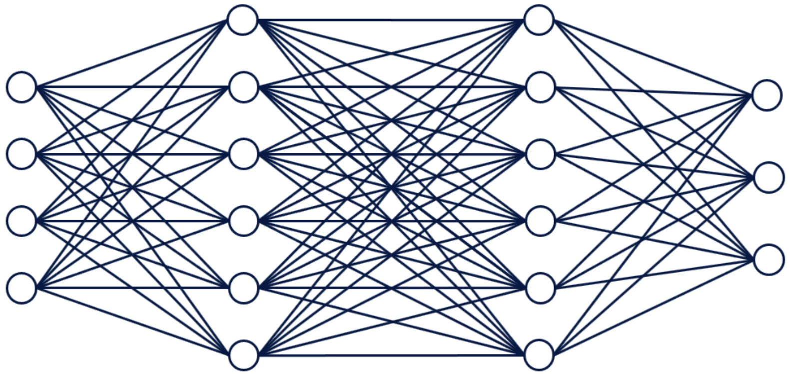
What is f()? Typical network



- Multi-layer perceptron
- A directed network with a set of inputs and outputs
  -- No loops
- Generic terminology
  -- We will refer to the inputs as the input units
    - No neurons here – the "input units" are just the units
  -- We refer to the outputs as the output units
  -- Intermediate units are "hidden" units

Typical network

- We assume a “layered” network for simplicity
  -- We will refer to the inputs as the input layer
    - No neurons here – the “layer” simply refers to inputs
  -- We refer to the outputs as the output layer
  -- Intermediate layer are “hidden” layer



The individual neurons

- Individual neurons operate on a set of inputs and produce a single output assume a “layered” network for simplicity

-- Standard setup: A differentiable activation function applied to an affine combination of the input

$$y = f\left(\sum_i w_i x_i + b\right)$$

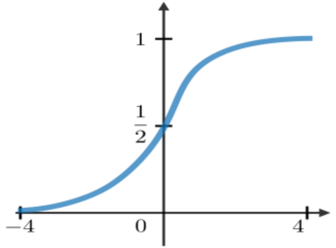
We will assume this unless otherwise specified

Parameters are weights w_i and bias b

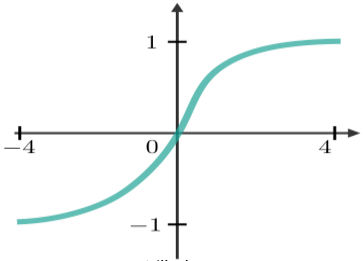
-- More generally: aby differentiable function

$$y = f(x_1, x_2, \ldots, x_N; W)$$

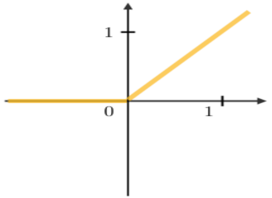
Activation and their derivatives



$$'(N) = \frac{1}{1 + \exp(-N)} \qquad 'B(N) = '(N)\big(1 - '(N)\big)$$



$$'(N) = \tanh(N) \qquad 'B(N) = \big(1 - '\,\%(>)\big)$$



$$'(N) = U\begin{matrix} N & N \geq 0 \\ 0 & N < 0 \end{matrix} \qquad 'B(N) = U\begin{matrix} 1 & N \geq 0 \\ 0 & N < 0 \end{matrix}$$

Some popular activation functions and their derivatives

Carnegie Mellon University

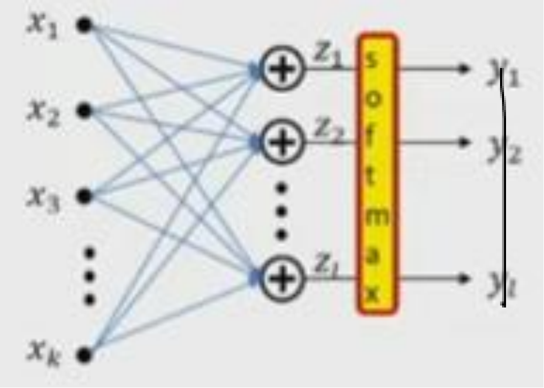
Vector activation example; Softmax

- Example: Softmax vector activation

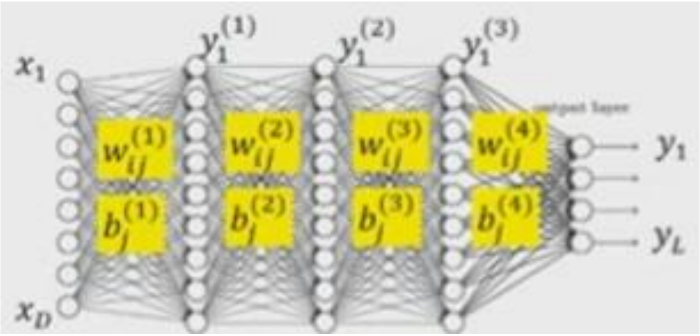
$$z_i = \sum_j w_{ji} x_j + b_i$$

Parameters are weights w_{ji} and bias b_i

$$y = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$



Notation



- The input layer is the layer
- We will represent the output of the i-th perceptron of the X^{56} layer as $J_!^{(-)}$

-- Input to network: $J_!^{(,)} = >_!$

-- Output of network: $J_! = J_!^{(9)}$

- We will represent the weight of the connection between the i-th unit of the k-1th layer and the jth unit of the k-th layer as $K_{!/}^{(-)}$

-- The bias to the jth unit of the k-th layer is $M_/^{(-)}$

Problem Setup: Things to define

- Given a training set of input-output pairs

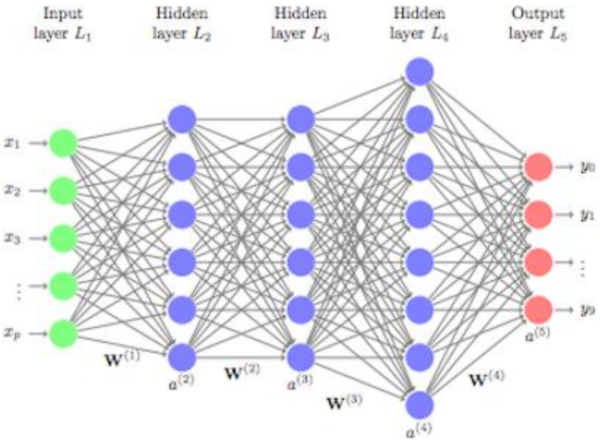
$(X_{\#}, d_{\#}), (X_{\%}, d_{\%}), \ldots, (X_{\&}, d_{\&}),$

- what are input-output pairs?

$$Loss(W) = \frac{\#}{\$} \sum_{!} div(f(X_{!}; W), d_{!})$$

Vector notation

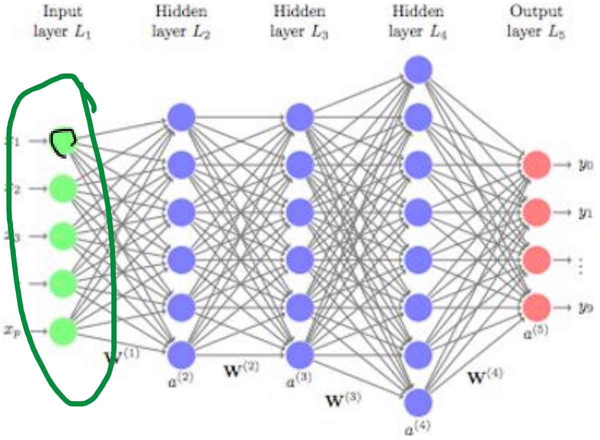
- Given a training set of input-output pairs ((#, ##), ((%, #%), …, ((&, #&)
- () = [>) #, >) %, …, >) :] is the nth input vector
- #) = [#) #, #) %, …, #) ;] is the nth desired output
- Y) = [J) #, J) %, …, J) ;] is the nth vector of actual outputs of the network
- We will sometimes drop the first subscript when referring to a specific instance



Carnegie Mellon University

Representing the Input

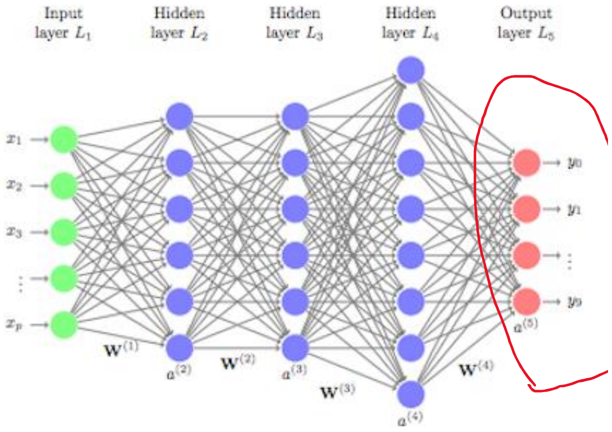
- Vectors of numbers
  -- (or may even be just a scalar, if input layer is of size 1)
  -- e.g. vector of pixel values
  -- e.g. vector of speech features
  -- e.g. real-valued vector representing text
    - We will see hoe this happens later in the course
  -- Other real valued vectors



Carnegie Mellon University

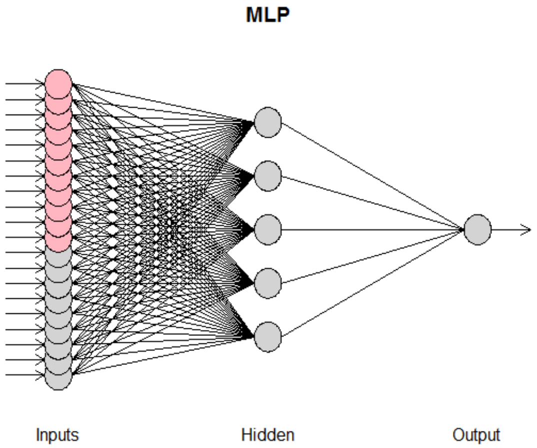
Representing the Output

- If the desired output is real-valued , no special tricks are necessary
  - -- Scalar Output : single output neuron
    - d = scalar (real value)
  - -- Vector Output : as many output neurons as the dimension of the desired output
    - $\# = [\#_{\#}, \#_{\%}, \#_{\triangleleft}]$ (vector of real values)



Representing the Output

- If the desired output is binary (is this a cat or not),use a simple 1/0 representation of the desired output
  -- 1 = Yes, it’s a cat
  -- 0 = No, it’s not a cat

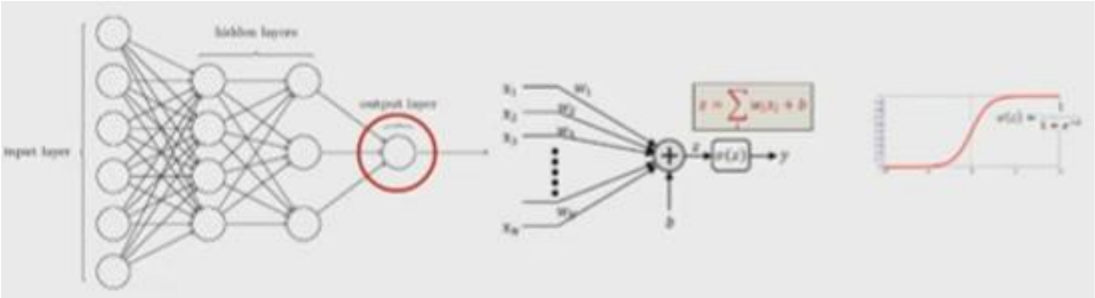


Carnegie Mellon University

Representing the Output

- If the desired output is binary (is this a cat or not), use a simple 1/0 representation of the desired output
- Output activation: Typically, a sigmoid

  -- Viewed as the probability P(Y = 1/X) of class value 1
  - Indicating the fact that for actual data, in general a feature value X may occur for both classes, but with different probabilities
  - Is differentiable



Carnegie Mellon University

Multi-class output: One-hot representations

- Consider a network that must distinguish if an input is a cat, a dog, a camel, a hat, or a flower
- We can represent this set as the following vector:

  [Z8[#4, Z8: ₩ℏ8[']4K₩θ]$^{\$}$

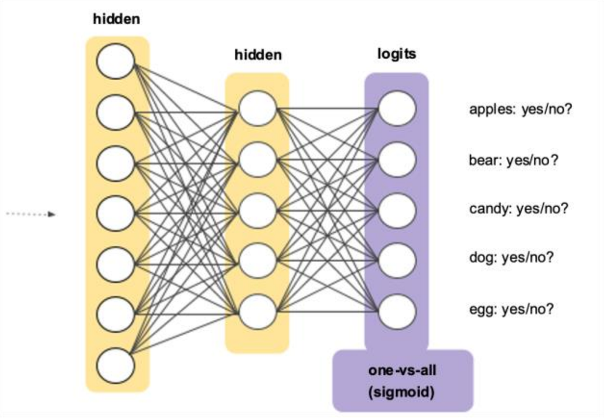
- For inputs of each of the five classes the desired output is:

  cat: $[1\ 0\ 0\ 0\ 0]^{\$}$
  dog: $[0\ 1\ 0\ 0\ 0]^{\$}$
  camel: $[0\ 0\ 1\ 0\ 0]^{\$}$
  hat: $[0\ 0\ 0\ 1\ 0]^{\$}$
  flower: $[0\ 0\ 0\ 0\ 1]^{\$}$

- For an input of any class, we will have a five-dimensional vector output with four zeros and a single 1 at the position of that class
- This is a one hot vector

Carnegie Mellon University

Multi-class network



- For a multi-class classifier with N classes, the one-hot representation will have N binary outputs
  -- An N-dimensional binary vector
- The neural network's output too must ideally be binary (N-1 zeros and a single 1 in the right place)
- More realistically, it will be a probability vector
  -- N probability values that sum to 1

Carnegie Mellon University