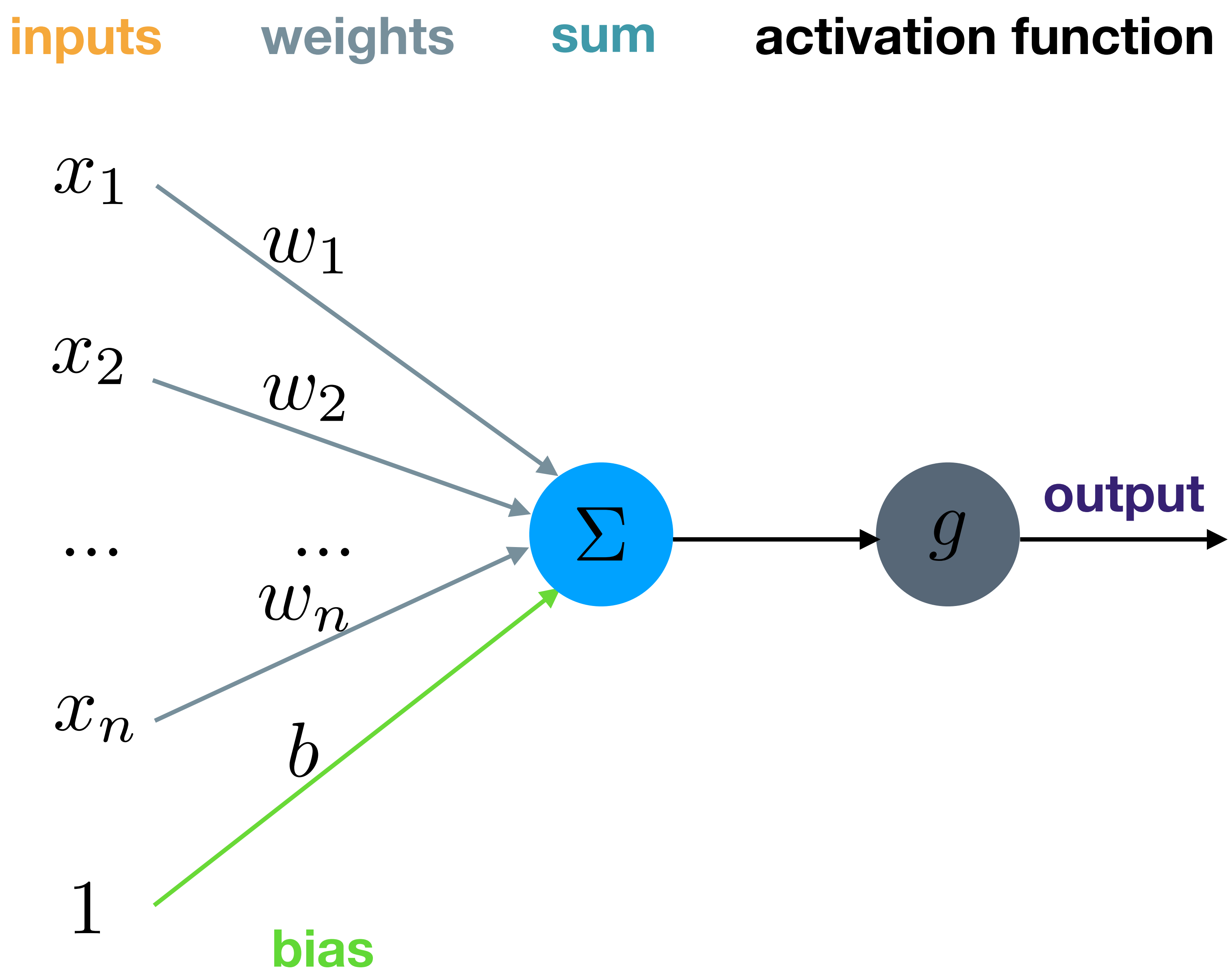
Deep Feedforward Networks

Han Shao, Hou Pong Chan, and Hongyi Zhang

Deep Feedforward Networks

- Goal: approximate some function f^*
  - e.g., a classifier, $y = f^*(x)$ maps input x to a class y
- Defines a mapping $y = f(x; \theta)$ and learns the value θ that results in the best approximation

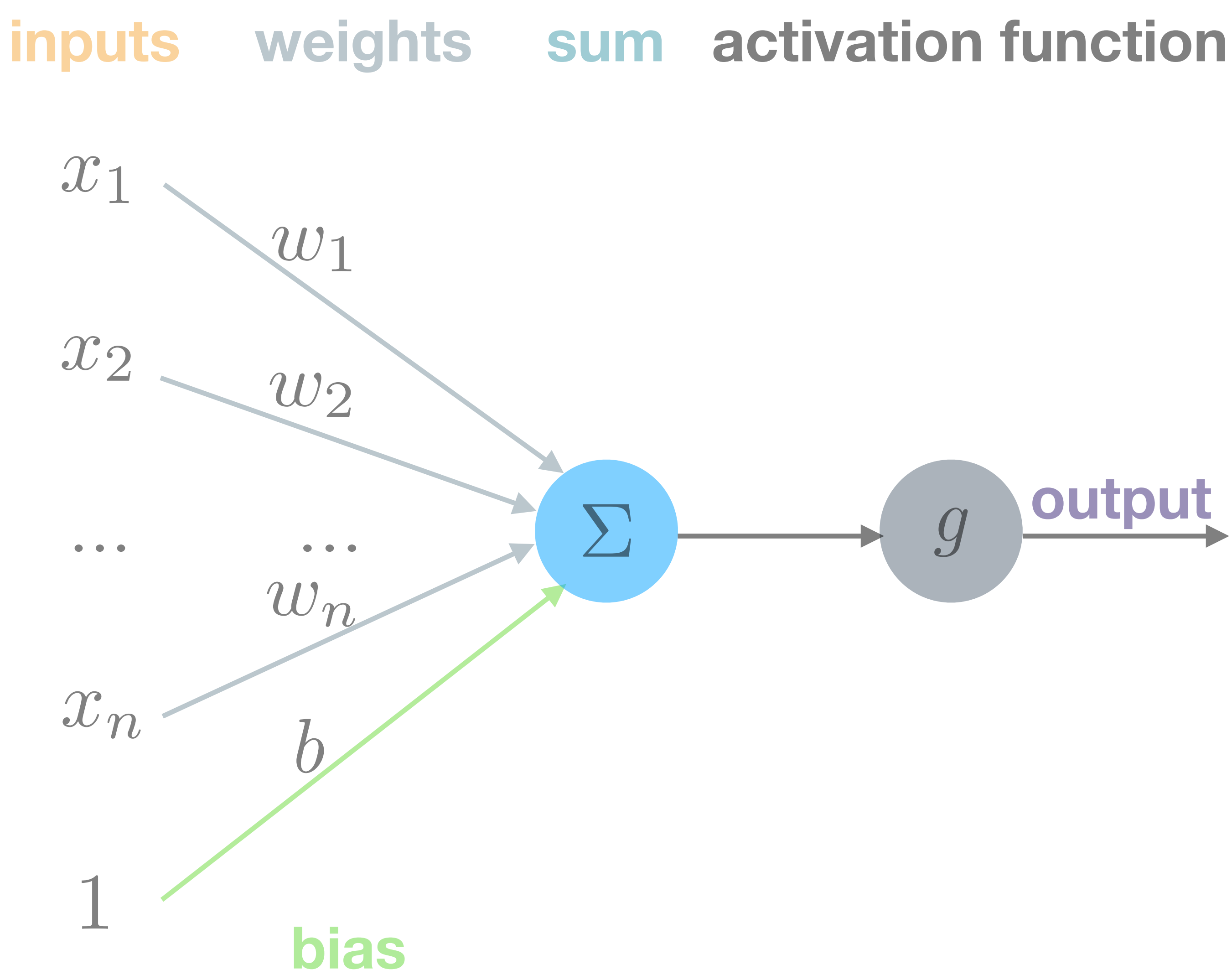
Neuron



- Takes n inputs and produce a single output

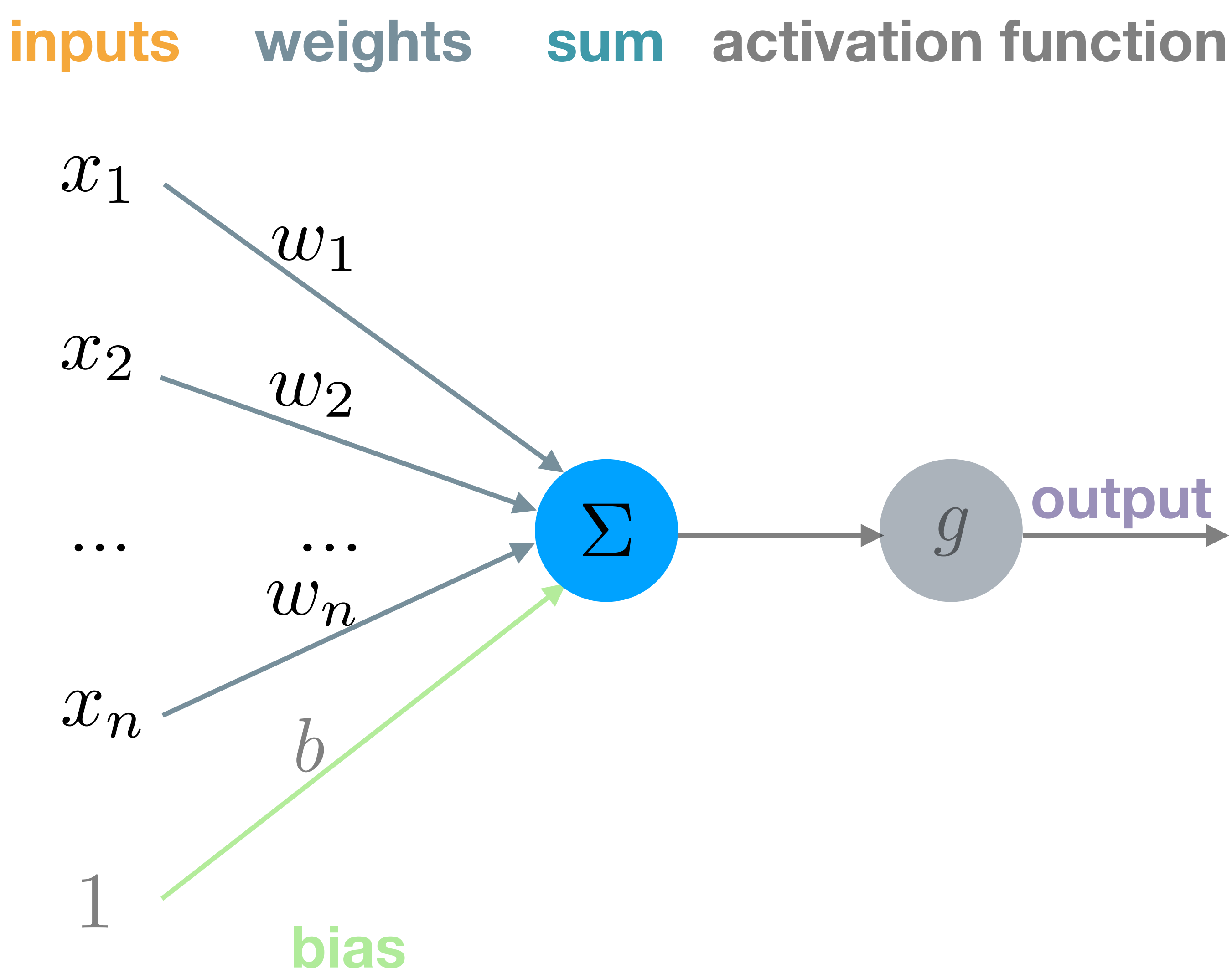
Neuron

$$\text{output} =$$



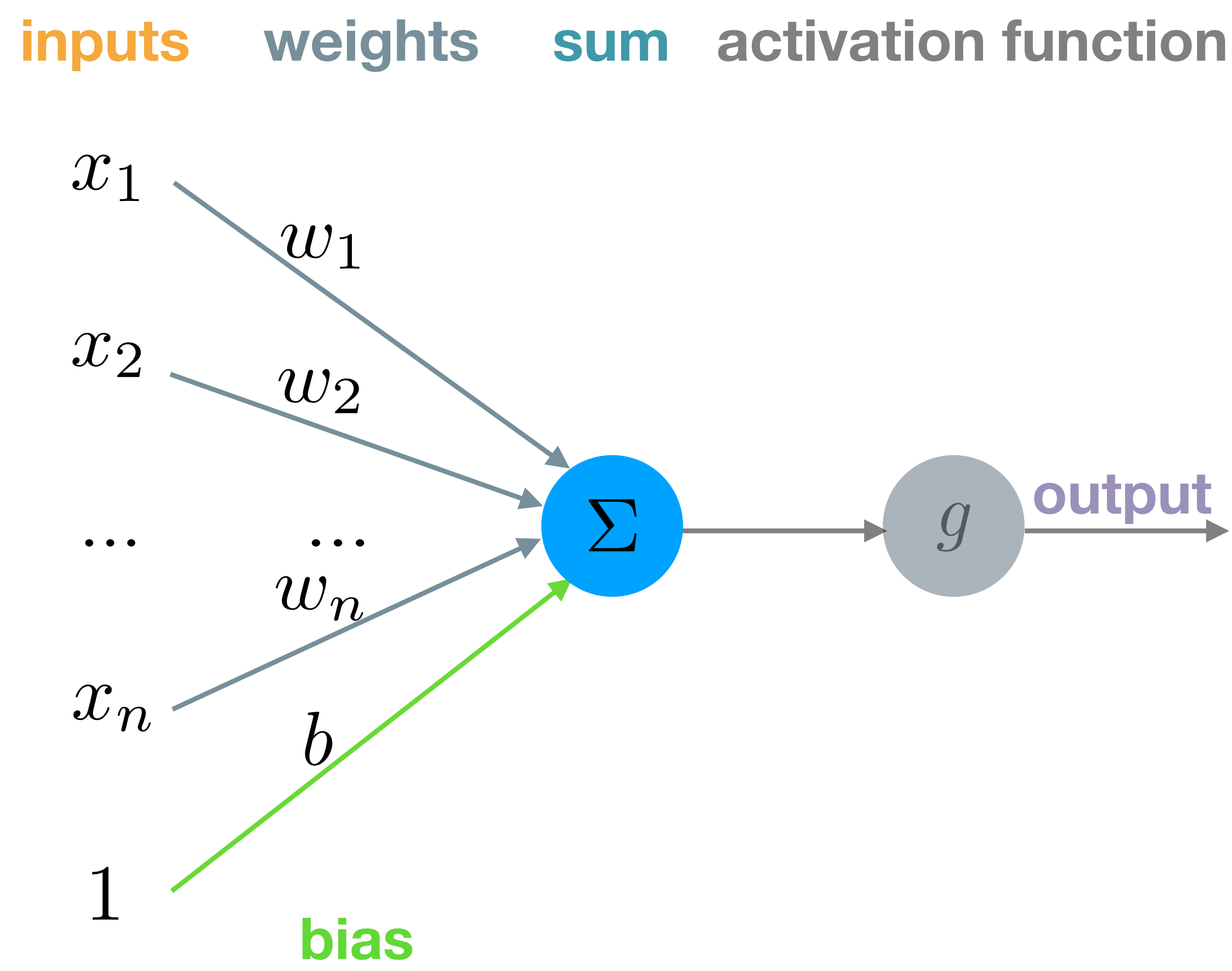
Neuron

$$\text{output} = \sum_{i=1}^{n} x_i w_i$$



Neuron

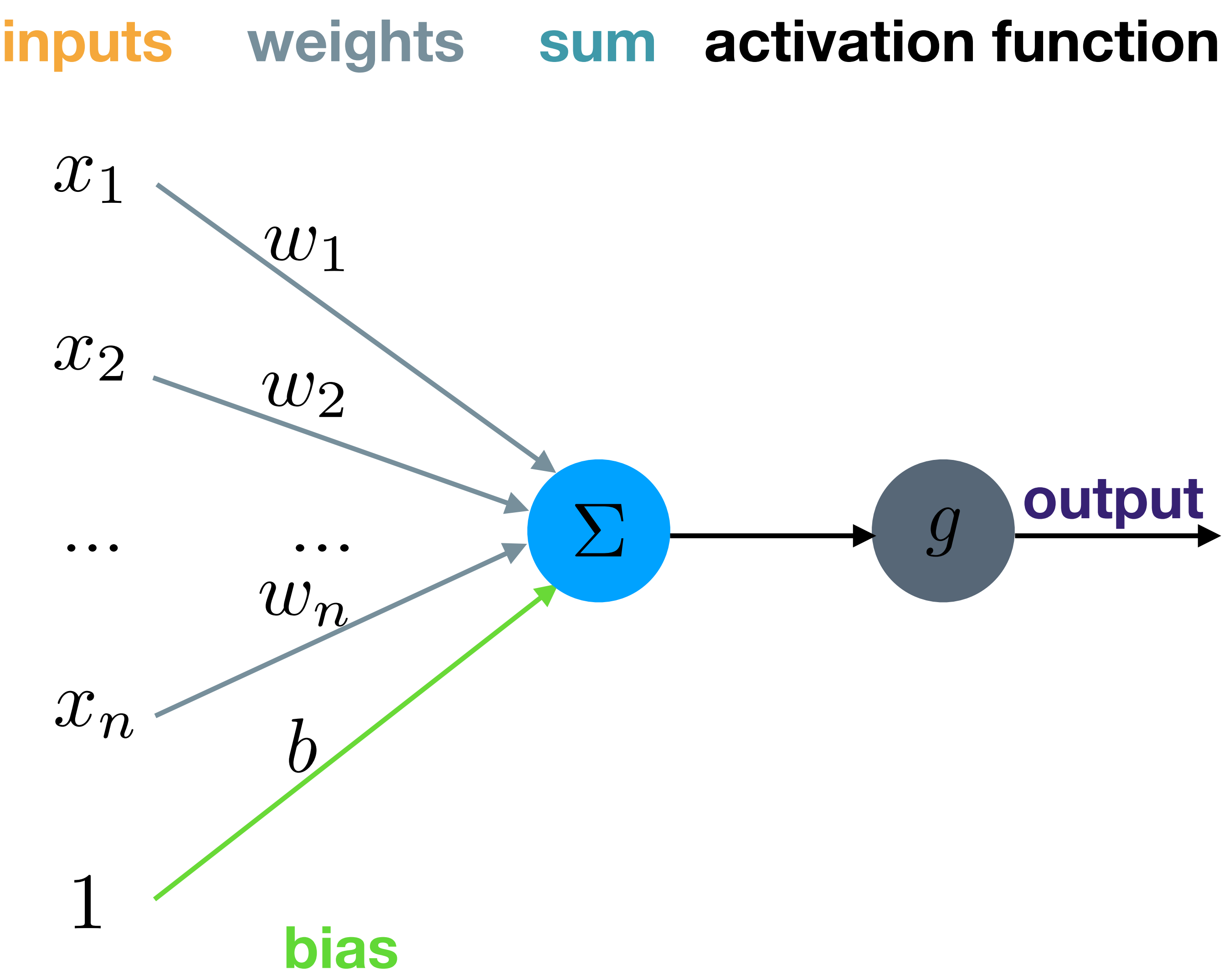
$$\text{output} = \sum_{i=1}^{n} x_i w_i + b$$



Neuron

activation function

$$\text{output} = g(\sum_{i=1}^{n} x_i w_i + b)$$

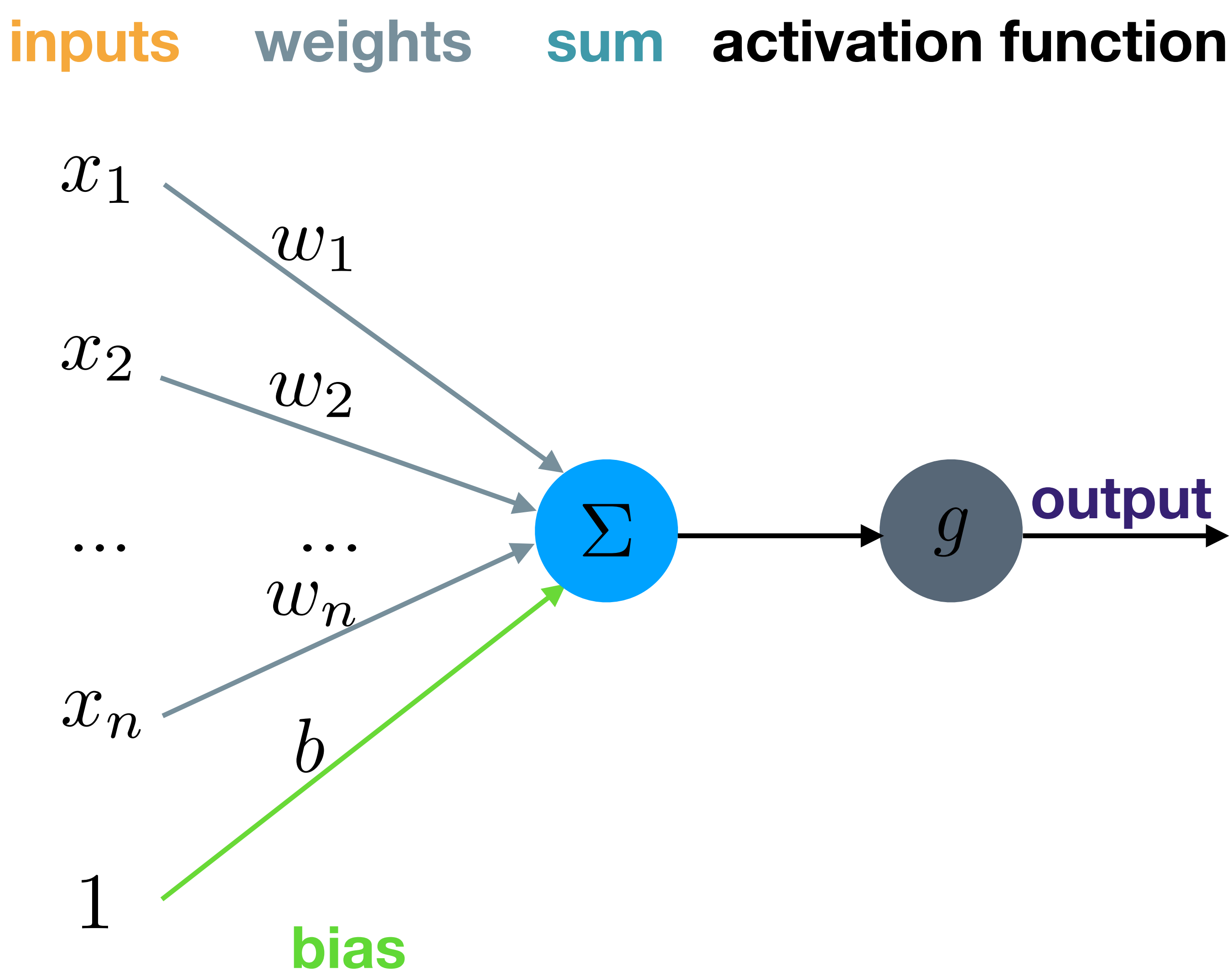


Neuron

activation function

$$\text{output} = g(w^T x + b)$$

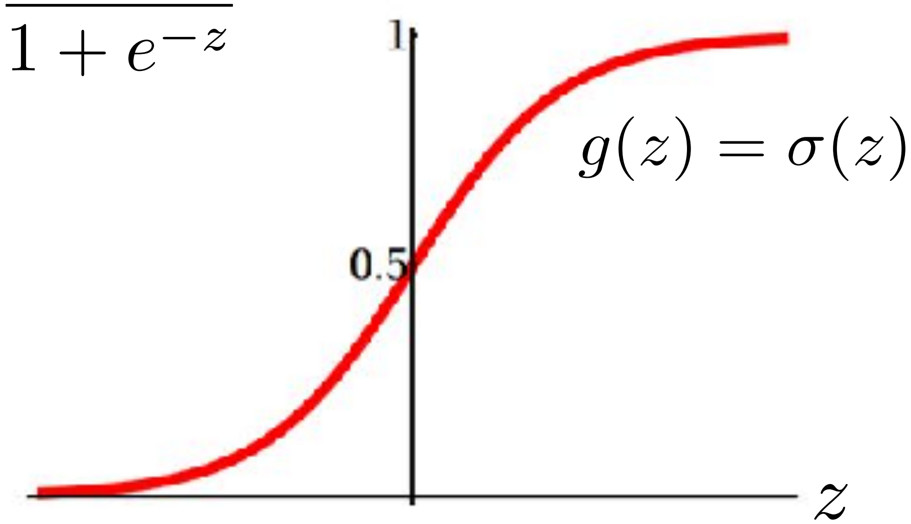
$$x = [x_1, ..., x_n]^T$$
$$w = [w_1, ..., w_n]^T$$



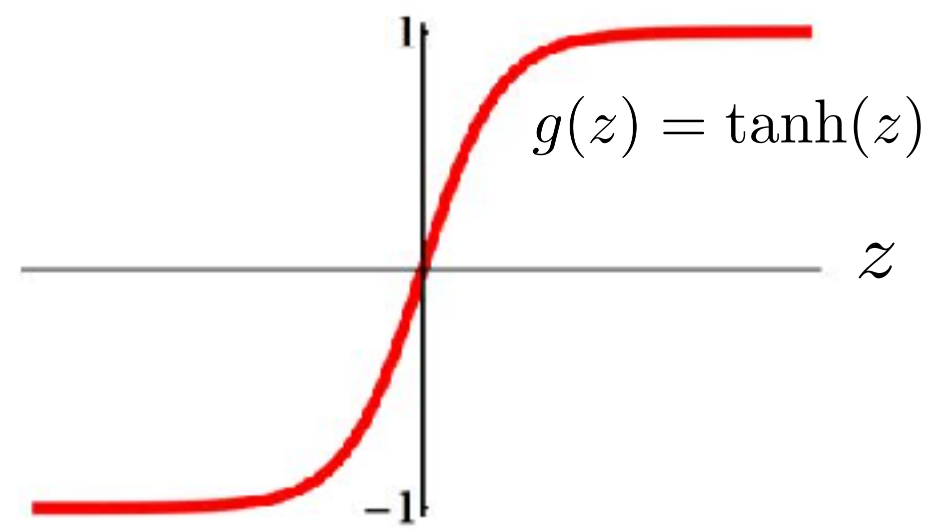
Common Activation Functions

Sigmoid

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



Hyperbolic



Rectified-Linear (ReLu)

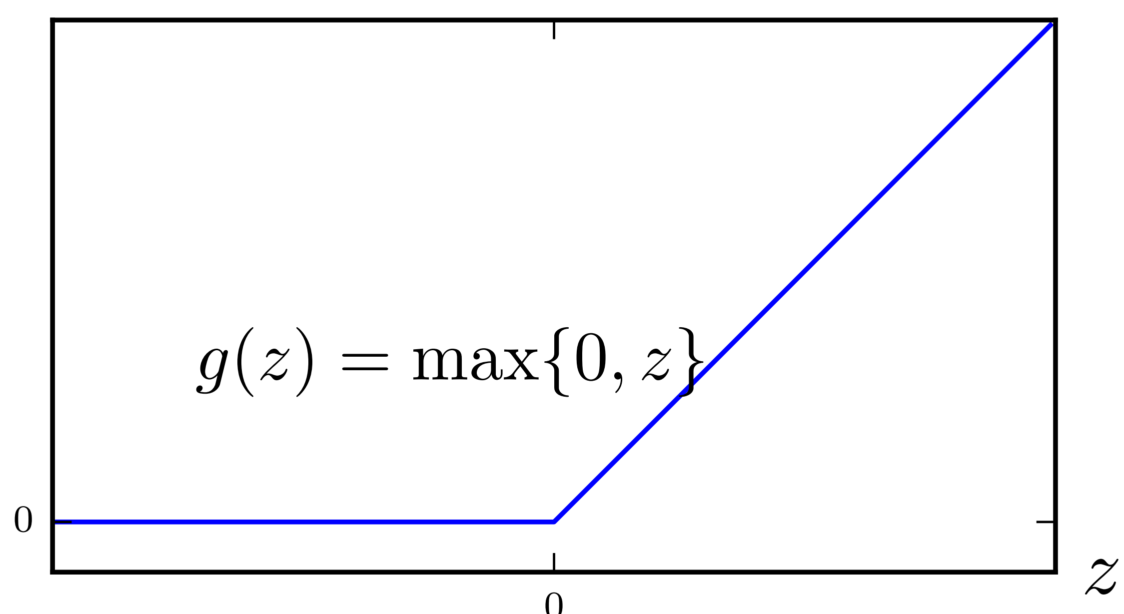
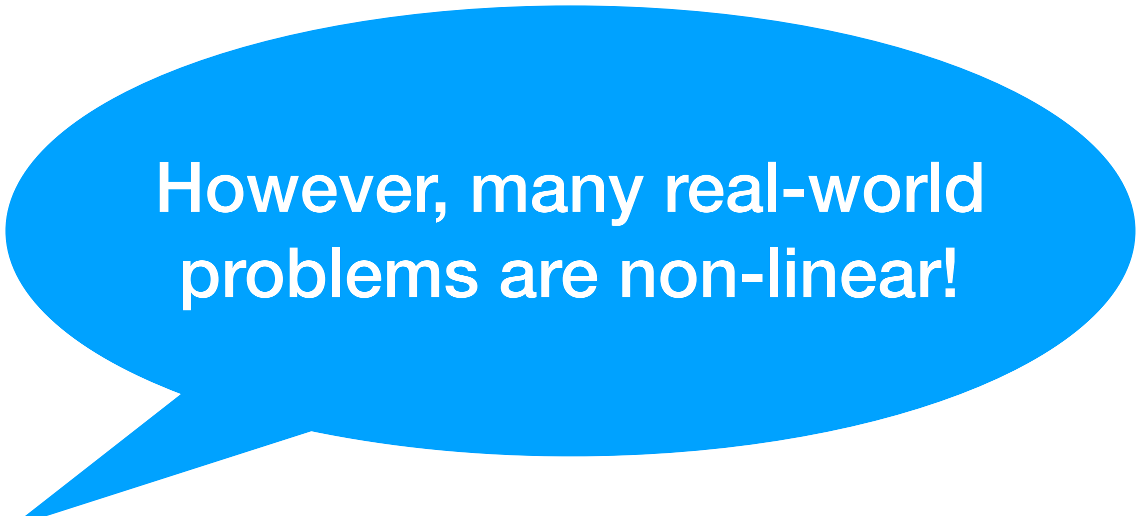
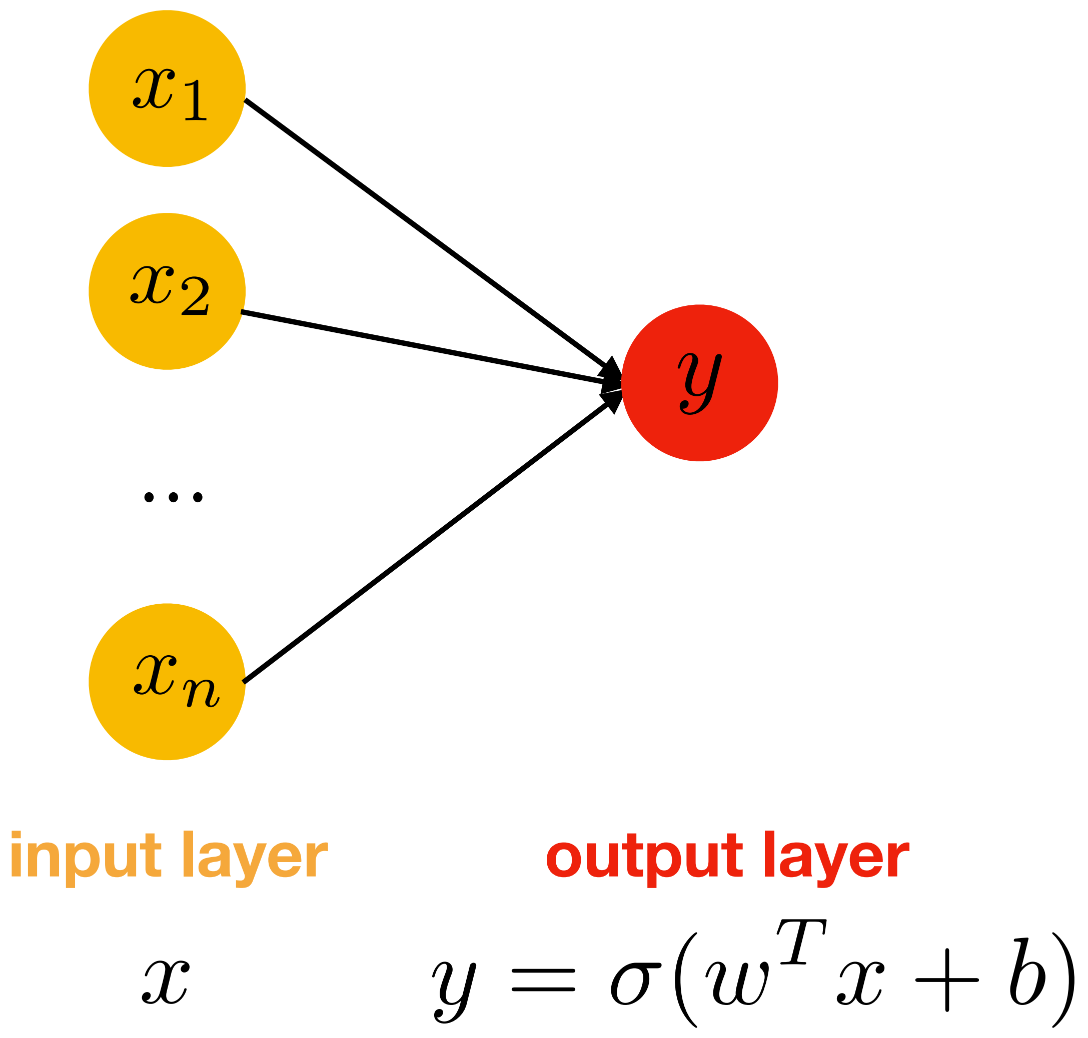


Figure from Deep Learning Book

Two-layer Neural Networks

- Two-layer neural networks model linear classifiers
- e.g., logistic regression





Example: Learning the XOR

- XOR function:
  - Operation on two binary values, x_1 and x_2
  - If exactly one of them is 1, returns 1
  - Else, returns 0
- Goal: Learn a function that correctly performs on

$$\mathbb{X} = \{[0,0]^T, [0,1]^T, [1,0]^T, [1,1]^T\}$$

Example: Learning the XOR

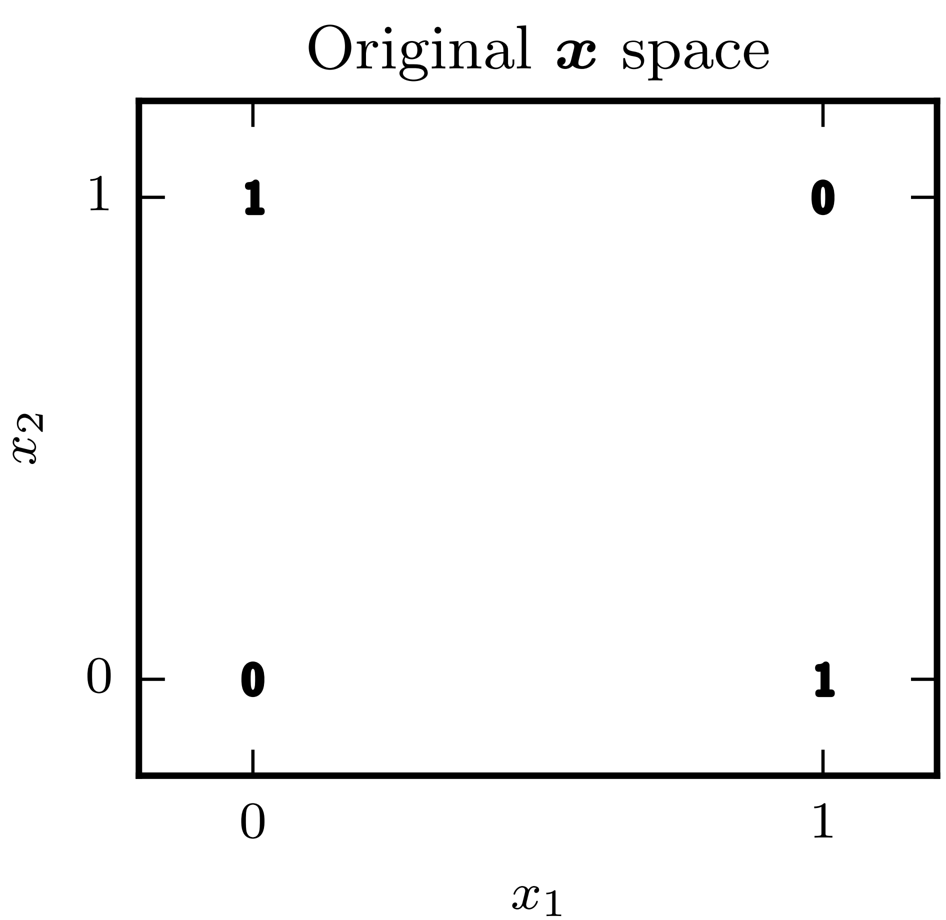
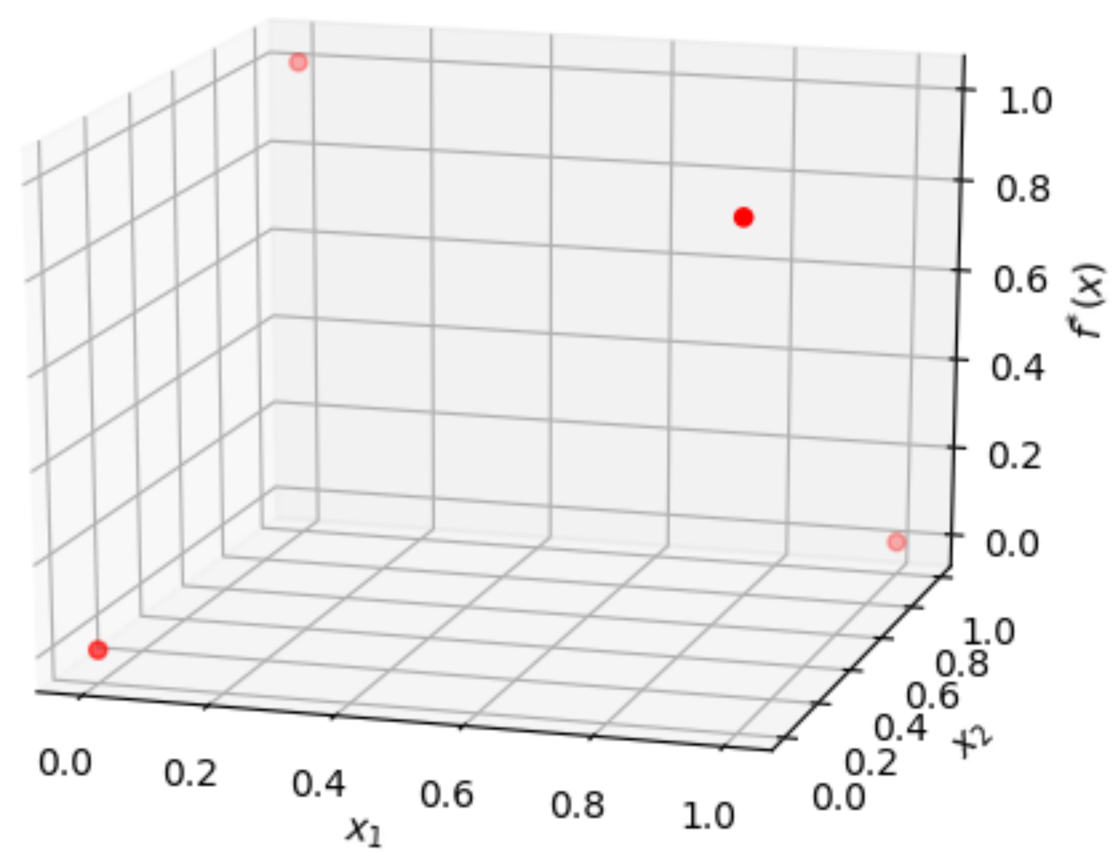


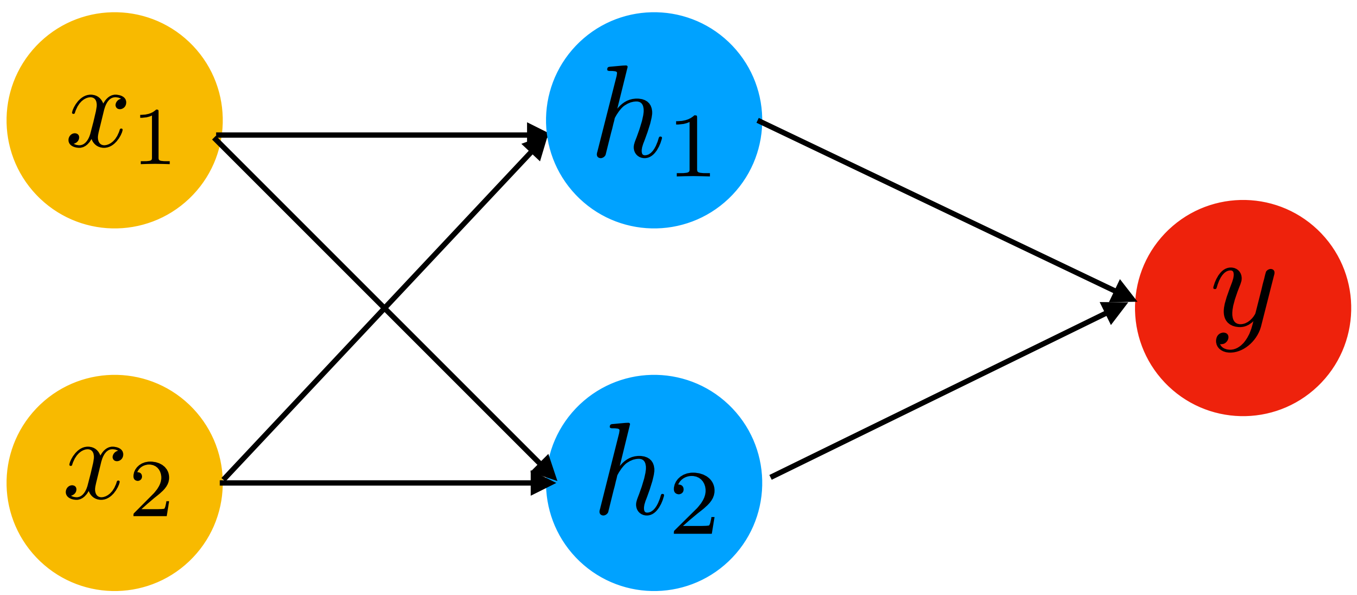
Figure from Deep Learning Book



- Cannot use a linear model to fit the data
- Need a three-layer neural network

Example: Learning the XOR

- Define a three-layer neural network (one hidden layer)



input layer | hidden layer | output layer

x $\quad h = g(U^T x + c) \quad y = w^T h + b$

Use ReLu

$$g(z) = \max\{0, z\}$$

Perform linear regression on the learned space

Example: Learning the XOR

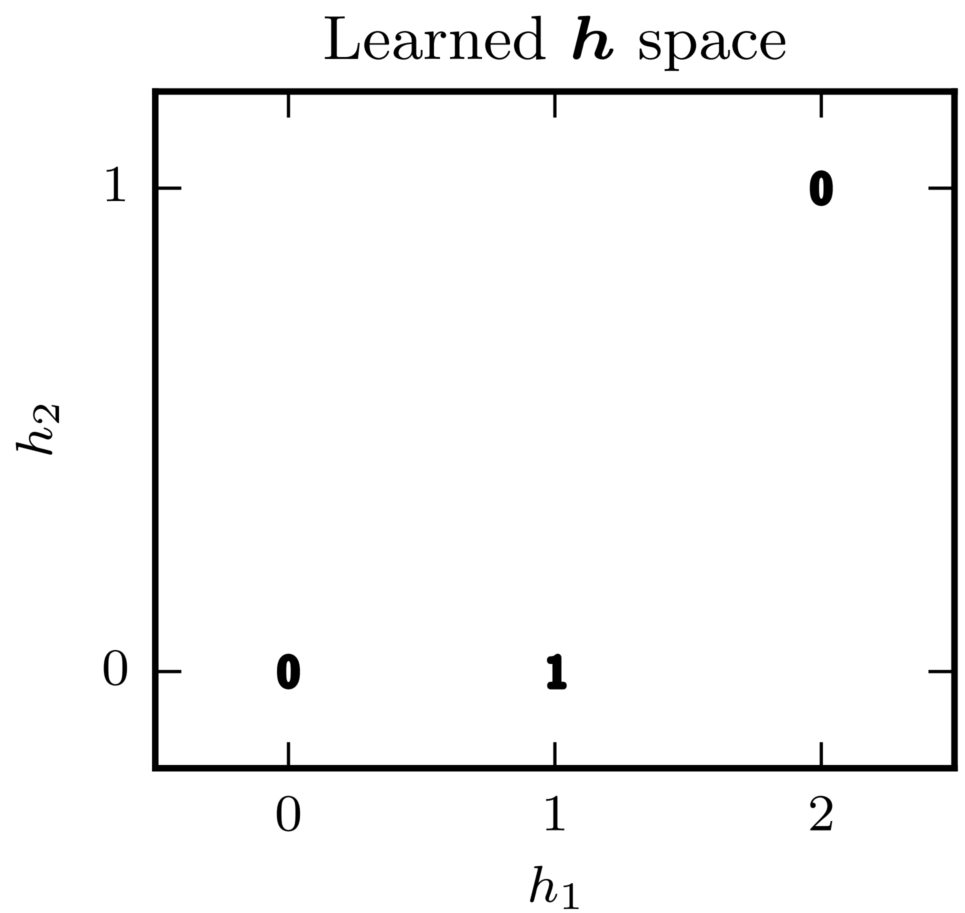
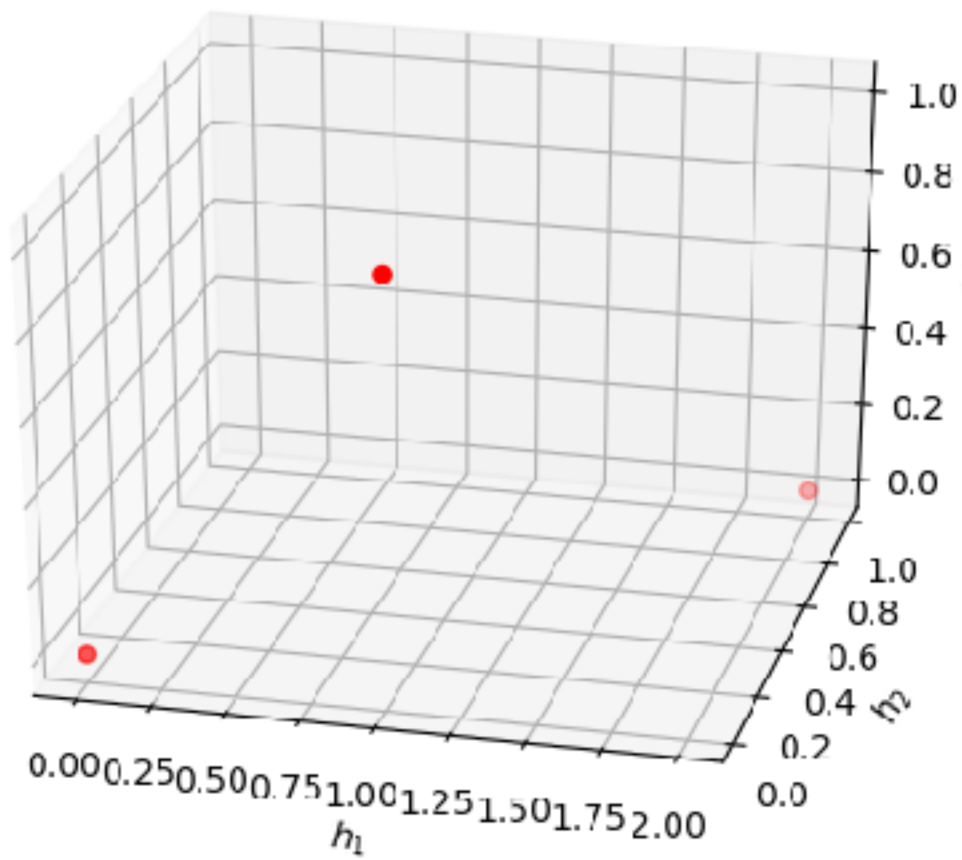


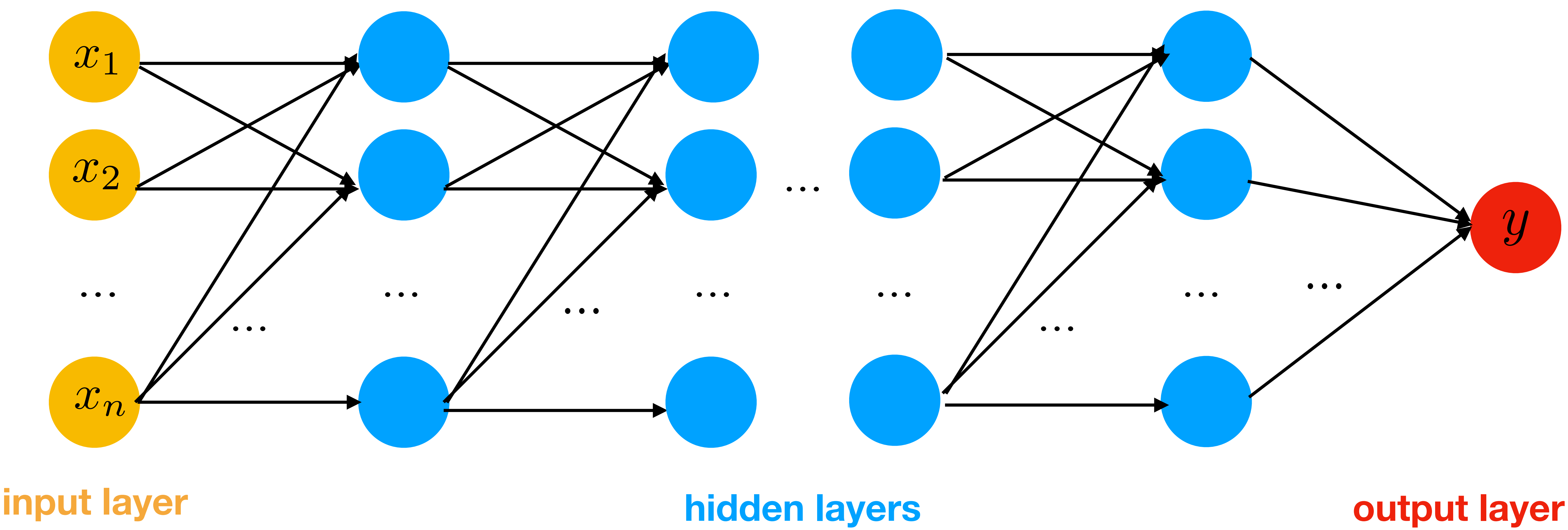
Figure from Deep Learning Book



- Can use a linear model to fit the data in the learned space

Deep Feedforward Network

- Add more hidden layers to build a deep architecture
- The word “deep” means many layers
- Why going “deep”?



Shallow Architecture

- A feedforward network with a single hidden layer can approximate any function

- But the number of hidden units required can be very large

    - $O(N)$ parameters are needed to represent N regions

    - e.g., represent the following k-NN classifier

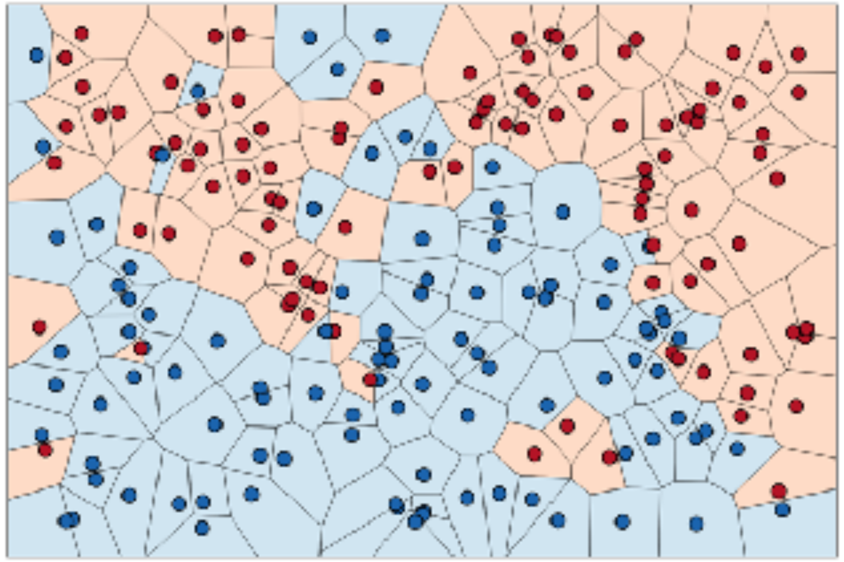


Figure from kevinzakka.github.io

Deep Architecture

- Greater expressive power
  - A feedforward network with piece-wise linear activation functions (e.g., ReLu) can represent functions with a number of regions that is exponential in the depth of the network [Montufar et al. 2014]
- Better generalization
  - Empirically results show that greater depth results in better generalization for a wide variety of tasks

Better Generalization with Greater Depth

- Transcribe multi-digit numbers from photographs of addresses [Goodfellow et al. 2014d]

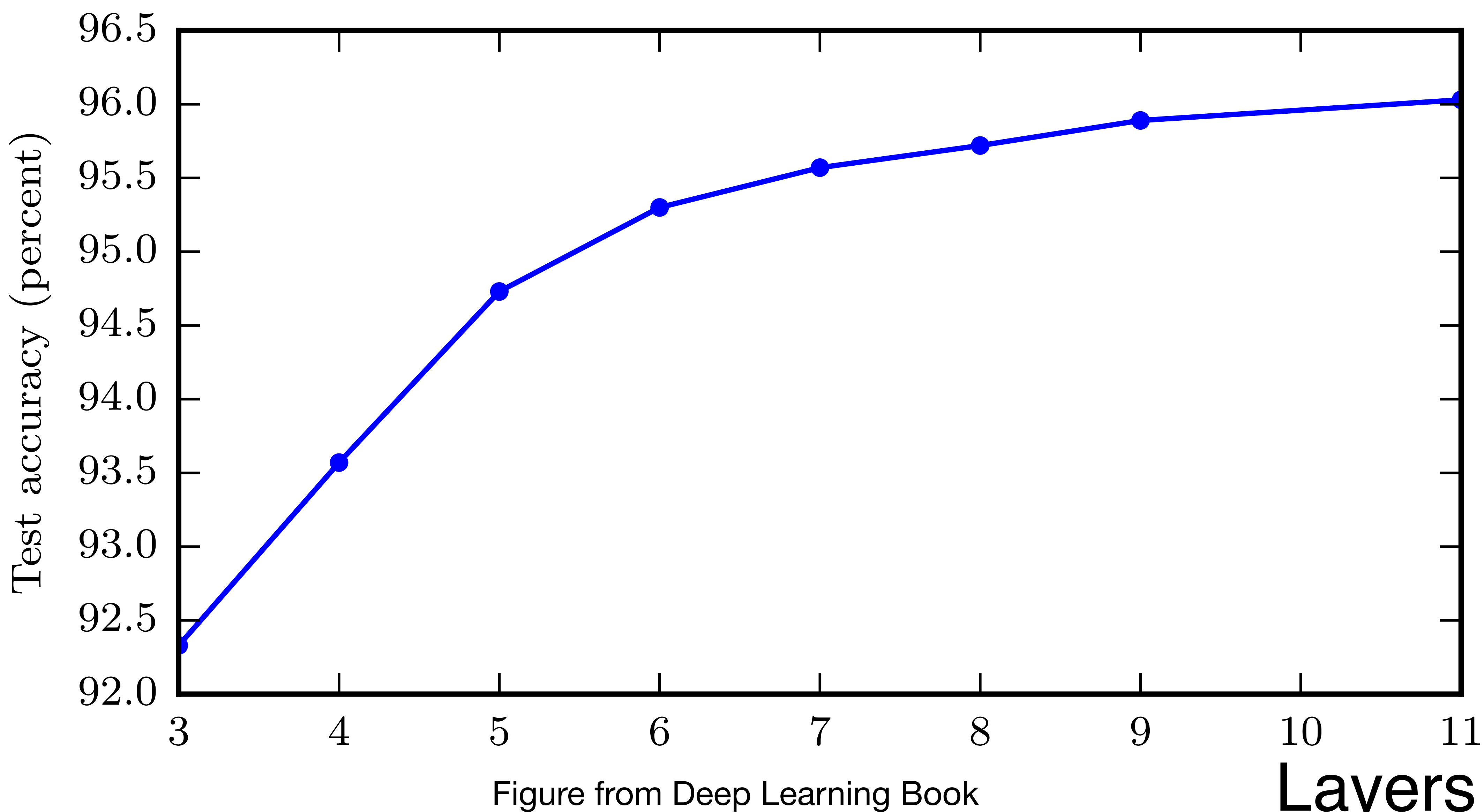


Figure from Deep Learning Book

Large Shadow models over fit more

- Transcribe multi-digit numbers from photographs of addresses [Goodfellow et al. 2014d]

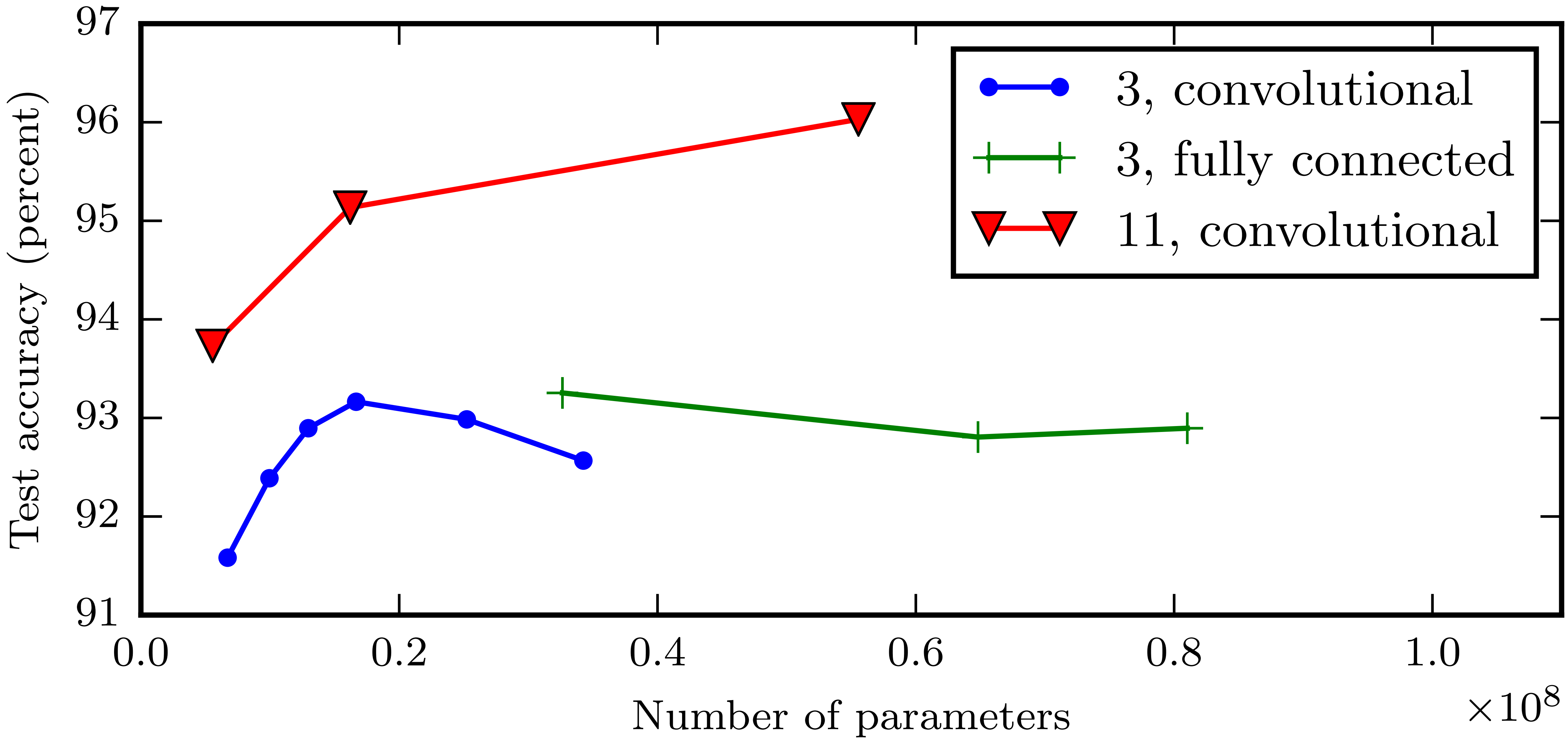


Figure from Deep Learning Book

Training

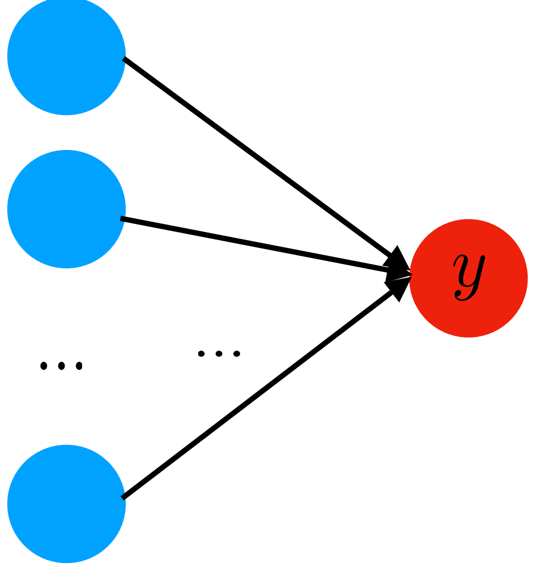
- Commonly used loss functions:
  - Squared loss: $l(\theta) = \frac{1}{2}\mathbb{E}_{x,y\sim\hat{P}_{data}}||x - f(x;\theta)||^2$

    Empirical distribution

  - Cross-entropy loss: $l(\theta) = -\mathbb{E}_{x,y\sim\hat{P}_{data}} \log f(x;\theta)$

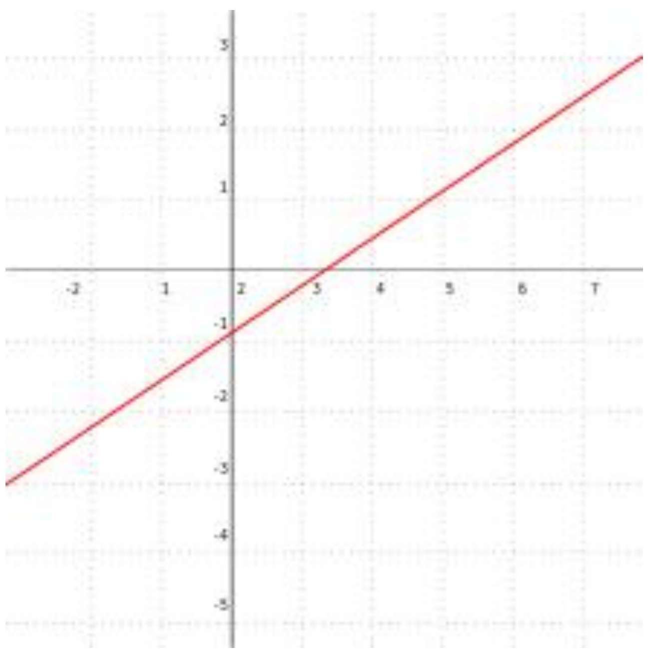
Use it when the output is a probability distribution

- Use gradient-based optimization algorithms to learn the parameters



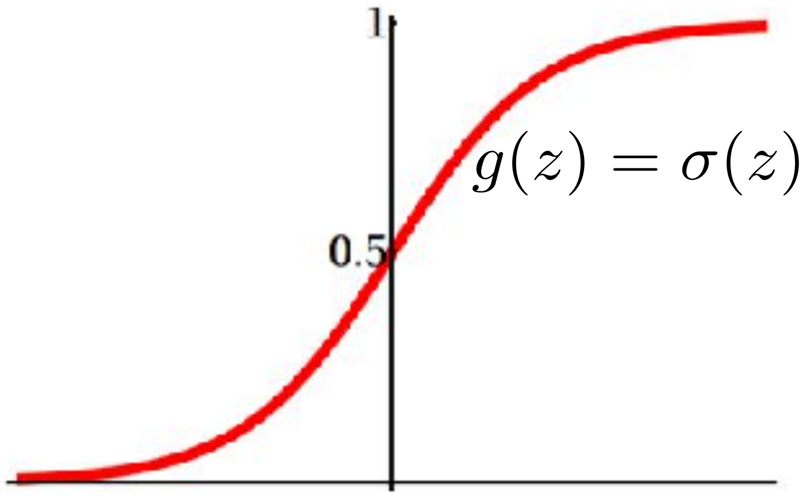
Output Units

- Suppose the network provides us hidden features h
- Linear Units:



  - $y = w^T h + b$
  - No activation function
  - Usually used to produce the mean of a conditional Gaussian
  - Do not saturated, good for gradient based algorithm

Output Units

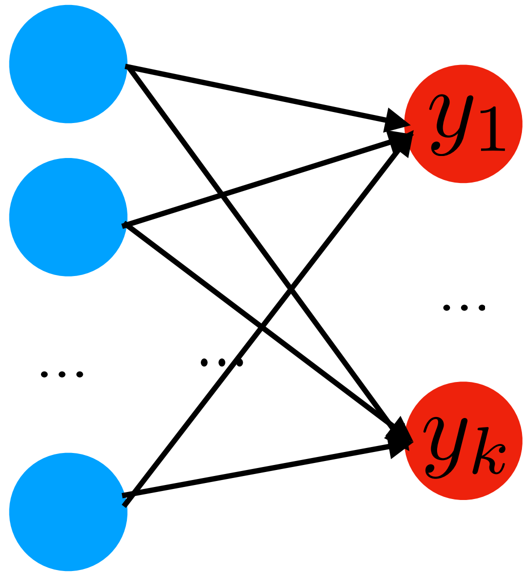


- Sigmoid Units
  - $y = \sigma(w^T h + b)$
  - Usually used to predict a Bernoulli distribution
    - e.g., binary classification, output $P(\text{class} = 1|x)$
  - Saturated when y is close to 1 or 0 because it is exponentiated
    - Should use cross-entropy loss as training loss

$$l(\theta) = -\mathbb{E}_{x,y\sim\hat{P}_{data}} \log f(x;\theta)$$

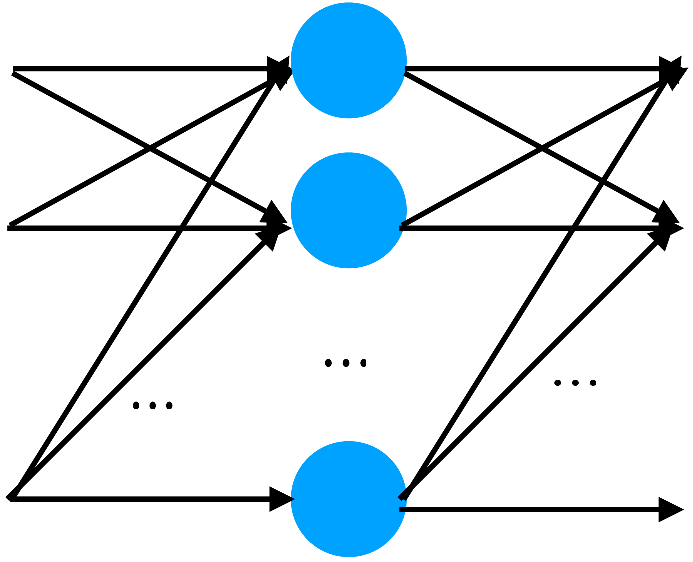
Undergoes the exp in the sigmoid

Output Units



- Softmax Units
  - $y = \text{softmax}(W^T h + b), y \in \mathbb{R}^k, W \in \mathbb{R}^{d \times k}$
  - Output a probability distribution over a discrete variable with k possible values
  - $\text{softmax}(z)_i = \dfrac{\exp(z_i)}{\sum_j \exp(z_i)}$
  - Softmax is a generalisation of sigmoid
    - Squashes the values of a k-dimensional vector
  - Suffers from saturation, should use cross-entropy loss

Hidden Units



- Rectified-Linear Units
  - $h = g(U^T x + c)$
  - $g(z) = \max\{0, z\}$
  - Excellent default choices
  - The derivative remains 1 whenever the unit is active
  - Easy to optimise by gradient-based algorithms
  - Drawback: cannot take gradient when activation is 0

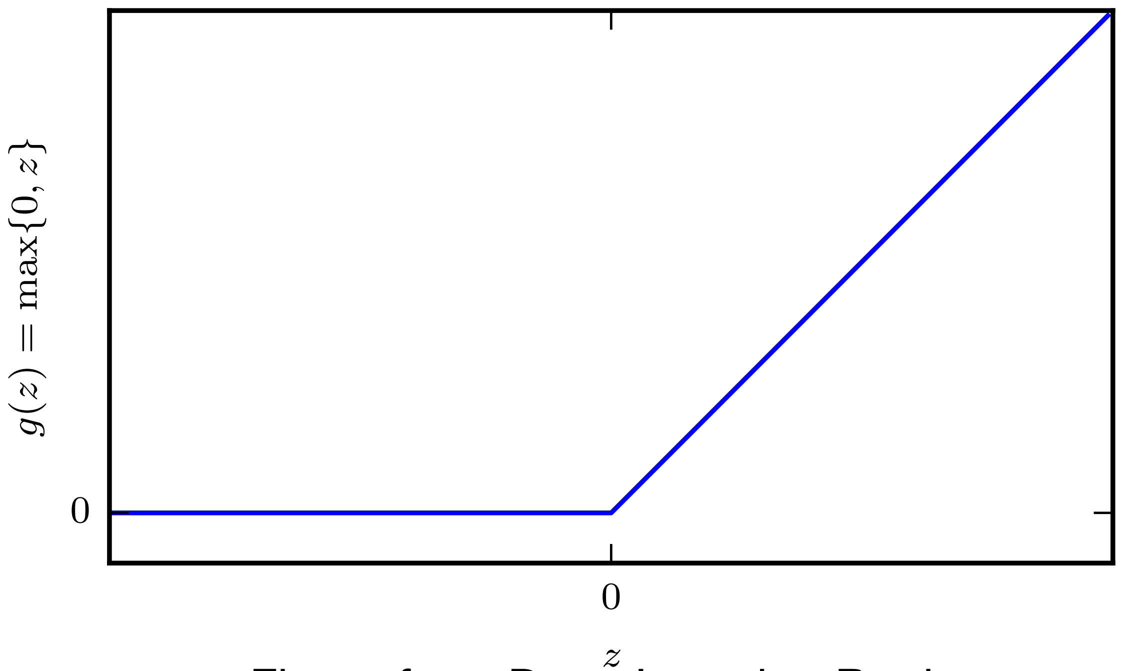
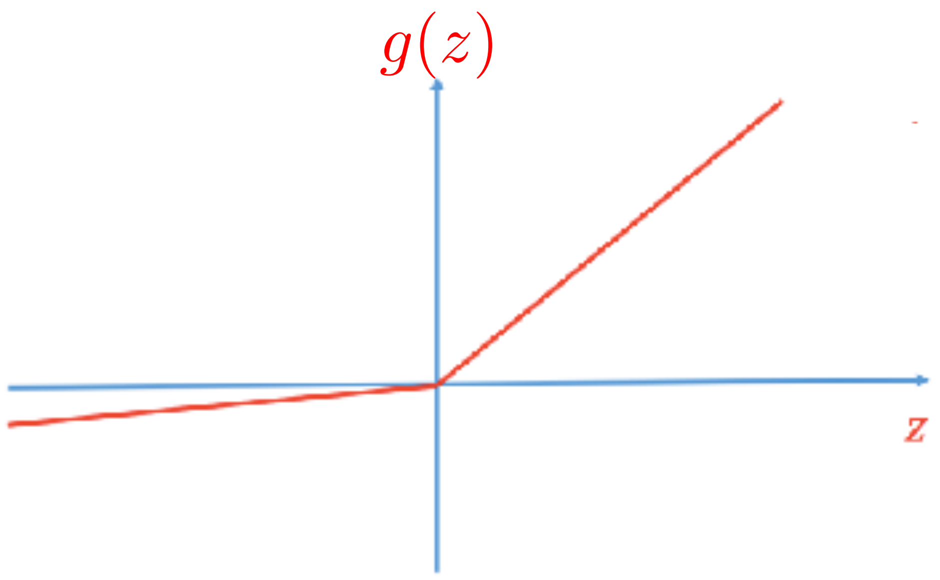


Figure from Deep Learning Book

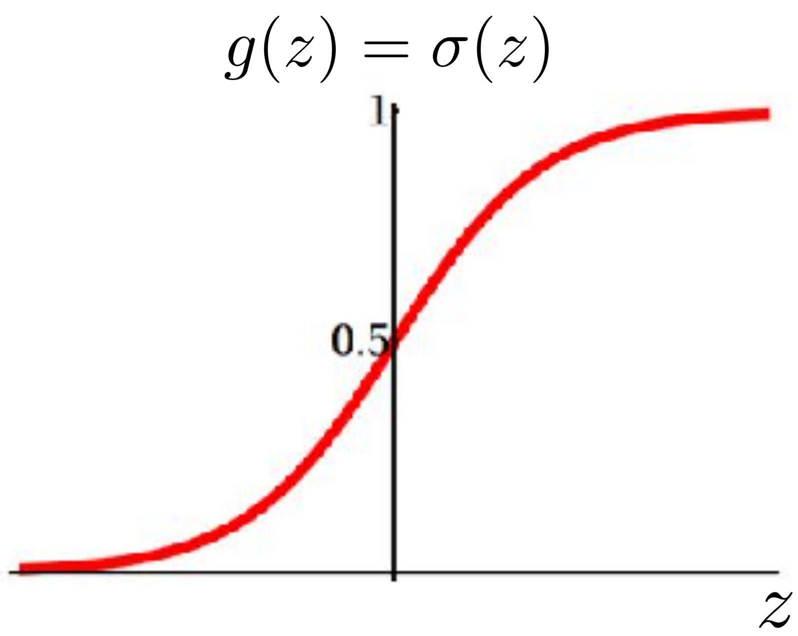
Hidden Units

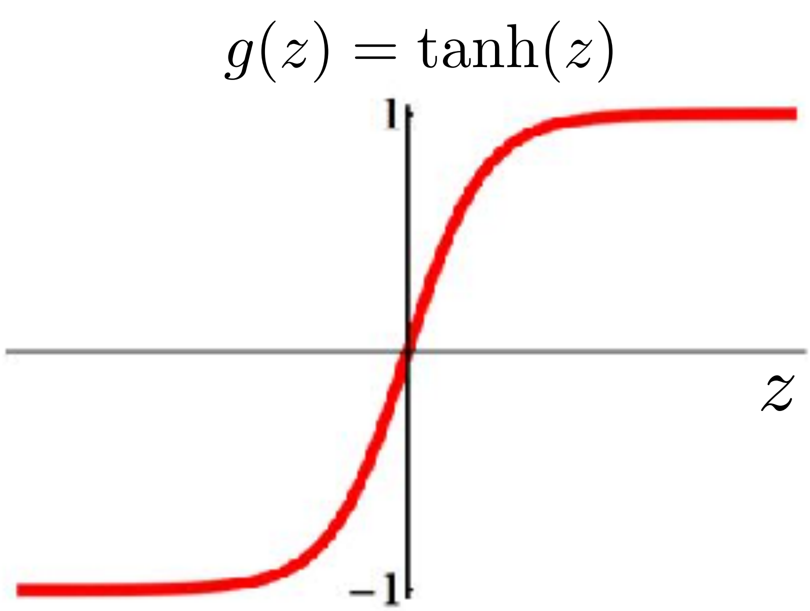
- Generalization of ReLU
  - $g(\alpha, z) = \max(0, z) + \alpha \min(z, 0)$
  - Leaky ReLu [Maas et al. 2013]
    - Fixes $\alpha = 0.01, \; g(z) = \max(0, z) + 0.01 \min(z, 0)$
  - Parametric ReLu [He et al. 2015]
    - Treat α as a learnable parameter
  - Occasionally performs better than ReLu



Hidden Units

- Sigmoid Units
  - $y = \sigma(U^T x + c)$
- Hyperbolic Tangent Units
  - $y = \tanh(U^T x + c)$
- Both of them have widespread saturation
- Use them as hidden units in feedforward network are discouraged





Demo

- Task - digit recognition (a classification task)
- Dataset - notMNIST
- Setup
  - Training set - 200000 pics
  - Validation set - 10000 pics
  - Test set - 18724 pics
- Measurement - accuracy