COSI 165B
Deep Learning

Chuxu Zhang
Computer Science Department
Brandeis University

2/8/2021

More of Last Lecture

❑ Validation Set

Split the training data into two disjoint subsets: training set, validation set

One for model parameter optimization, the other for hyperparameter selection.

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} \qquad \mathrm{MSE}_{\mathrm{test}} = \frac{1}{m} \sum_i (\hat{\boldsymbol{y}}^{(\mathrm{test})} - \boldsymbol{y}^{(\mathrm{test})})_i^2. \qquad J(\boldsymbol{w}) = \mathrm{MSE}_{\mathrm{train}} + \lambda \boldsymbol{w}^\top \boldsymbol{w},$$

❑ K-fold Cross Validation

Split data into k non-overlapping subsets.

On trial i, the i-th subset of the data is used as the validation/test set and the rest of the data is used as the training set.

The validation/test error may then be estimated by taking the average validation/test error across k trials.

More of Last Lecture

❑ Why Gradient Descent

$$\frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2 = J(\theta)$$

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\
&= \frac{1}{2}\nabla_\theta \left((X\theta)^T X\theta - (X\theta)^T\vec{y} - \vec{y}^T(X\theta) + \vec{y}^T\vec{y}\right) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T(X^TX)\theta - \vec{y}^T(X\theta) - \vec{y}^T(X\theta)\right) \\
&= \frac{1}{2}\nabla_\theta \left(\theta^T(X^TX)\theta - 2(X^T\vec{y})^T\theta\right) \\
&= \frac{1}{2}\left(2X^TX\theta - 2X^T\vec{y}\right) \\
&= X^TX\theta - X^T\vec{y}
\end{aligned}$$

$$\theta = (X^TX)^{-1}X^T\vec{y}.$$

Does not exist for most cases

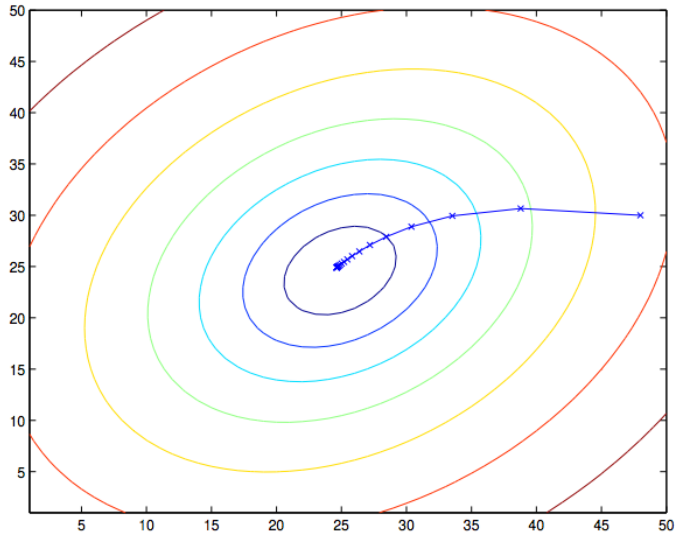
Loop {

for $i = 1$ to n, {

$$\theta_j := \theta_j + \alpha\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}, \quad \text{(for every } j\text{)}$$

}

}



Find local/global minimum step by step

Unsupervised Learning Algorithms

- Extract information from a distribution that do not require human labor to annotate examples.
- Find the “best” representation of the data.
- Looking for a representation that preserves as much information about data as possible while keep the representation simpler than data itself.
- Low-dimensional representations: compress as much information about x as possible in a smaller representation.
- Sparse representations: representation whose entries are mostly zeroes for most inputs.
- Independent representations: dimensions of the representation are statistically independent.

k-means clustering

In the clustering problem, we are given a training set $\{x^{(1)}, \ldots, x^{(n)}\}$, and want to group the data into a few cohesive "clusters." Here, $x^{(i)} \in \mathbb{R}^d$ as usual; but no labels $y^{(i)}$ are given. So, this is an unsupervised learning problem.

The k-means clustering algorithm is as follows:

1. Initialize **cluster centroids** $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^d$ randomly.
2. Repeat until convergence: {

   For every i, set

   $$c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2.$$

   For each j, set

   $$\mu_j := \frac{\sum_{i=1}^{n} 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^{n} 1\{c^{(i)} = j\}}.$$

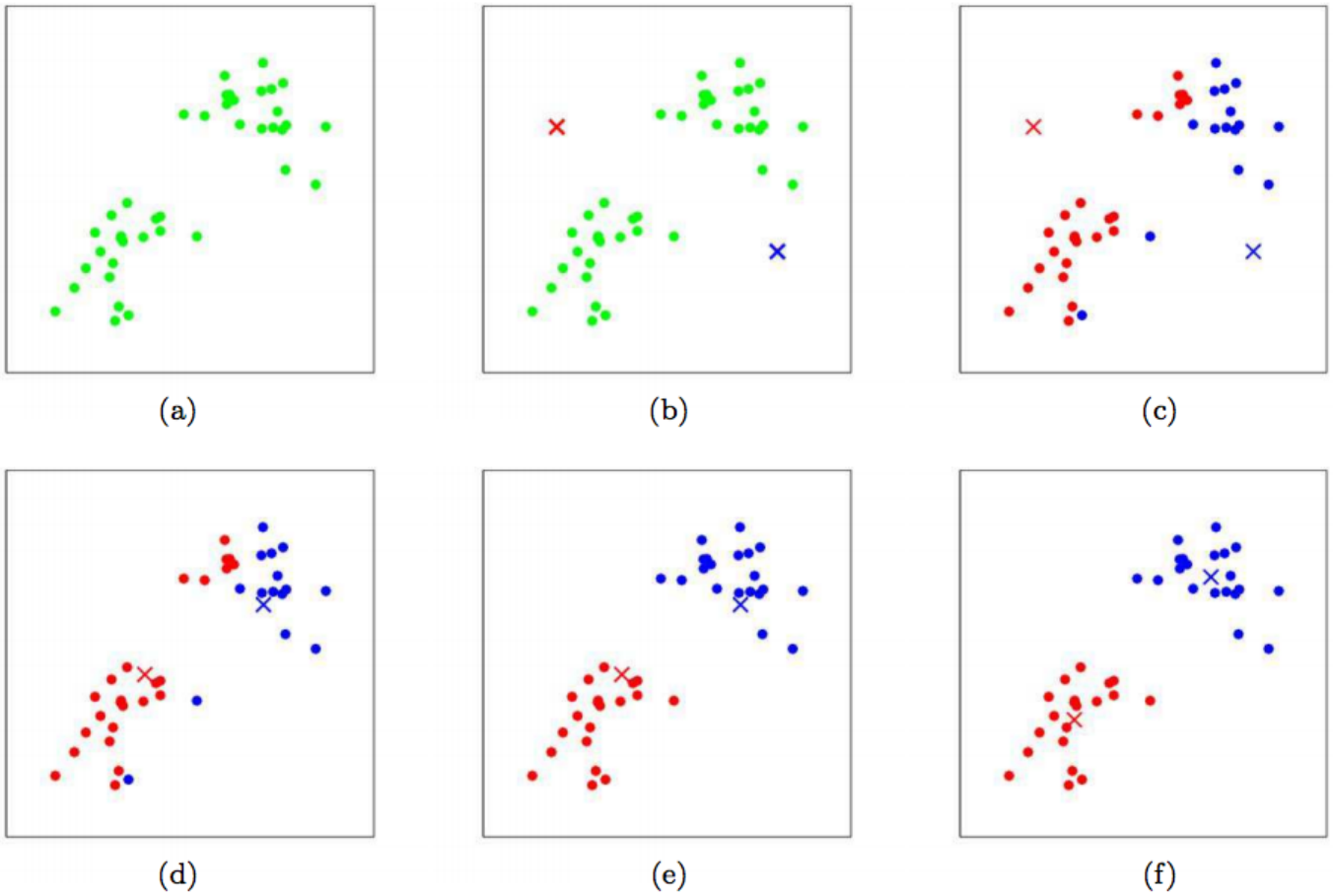
   }

distortion function

$$J(c, \mu) = \sum_{i=1}^{n} ||x^{(i)} - \mu_{c^{(i)}}||^2$$

The inner-loop of k-means repeatedly minimizes J with respect to c while holding μ fixed, and then minimizes J with respect to μ while holding c fixed. Thus, J must monotonically decrease, and the value of J must converge.

k-means clustering



Random initial cluster centroids

assign each training example to the closest cluster centroid

move each cluster centroid to the mean of the points assigned to it

Unsupervised Learning Algorithms

- ❑ Principal Components Analysis
- ❑ EM Algorithm
- ❑ Gaussian mixture model

Machine Learning | Andrew Ng: YouTube, Coursera

Build Machine Learning Algorithms

- Dataset and Task

  e.g., housing price prediction

- Model (e.g., linear regression)

$$h(x) = \sum_{i=0}^{d} \theta_i x_i = \theta^T x,$$

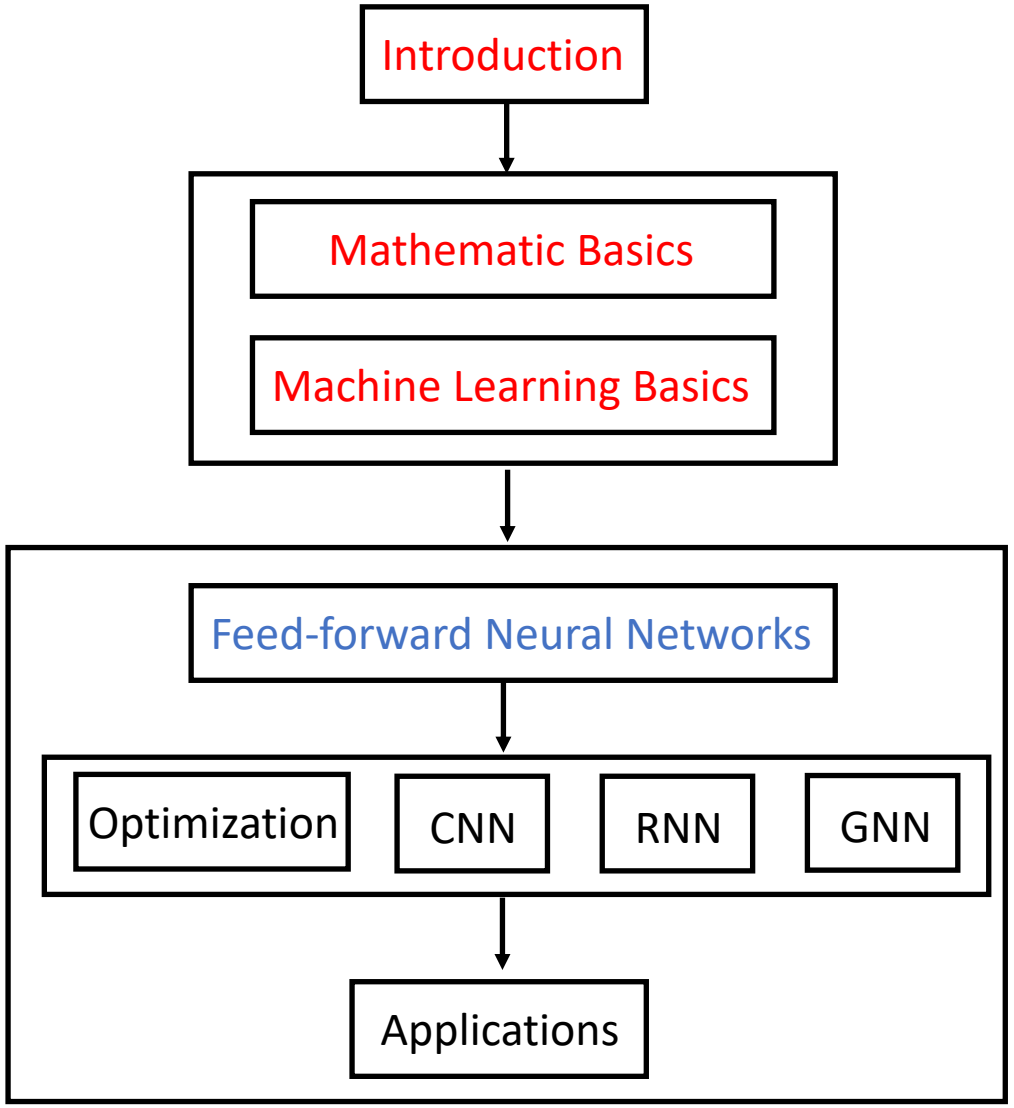
- Objective Function (e.g., mean square error)

$$\frac{1}{2} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$$

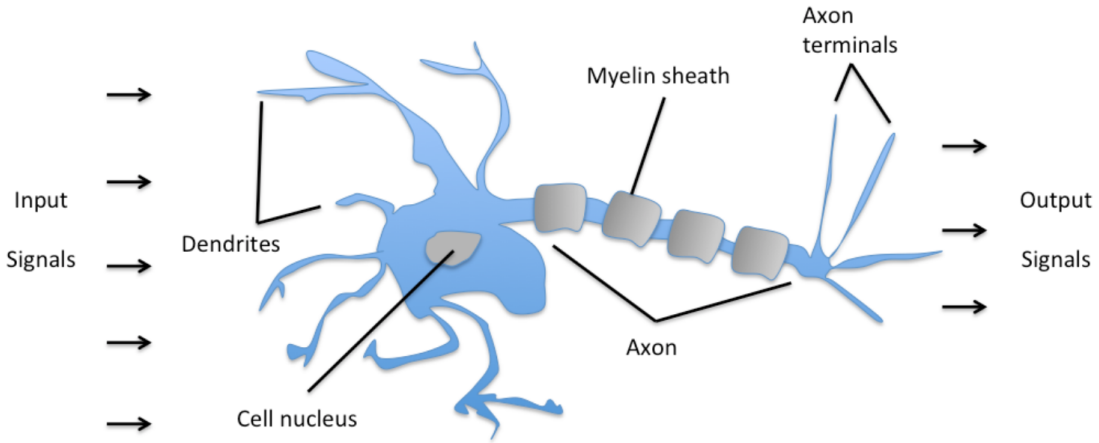
- Optimization Strategy (e.g., gradient descent)

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}, \quad \text{(for every } j\text{)}$$

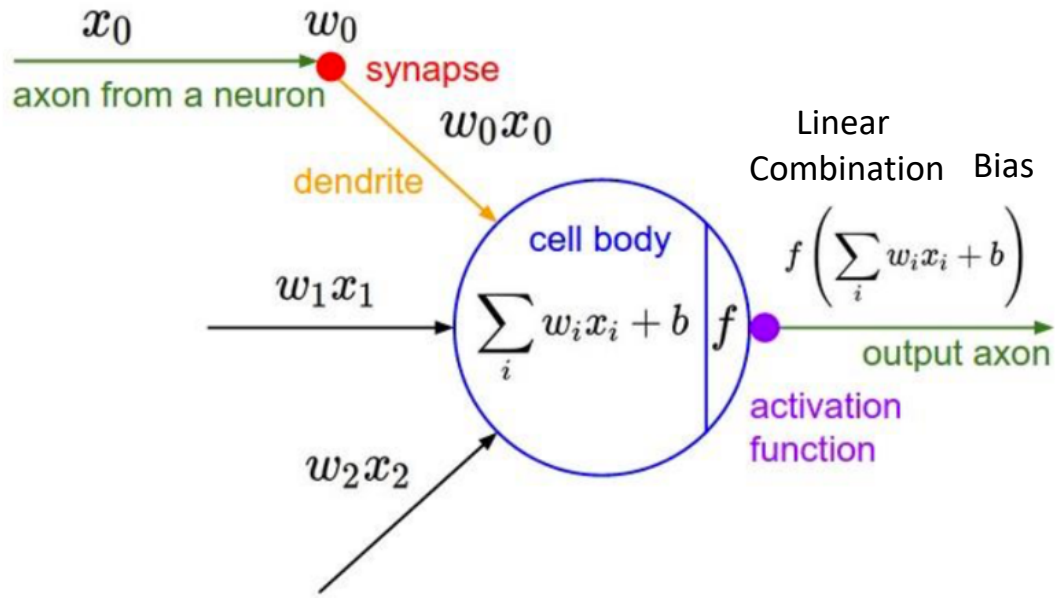
Structure of This Course



❑ Perceptron: Single Neuron



Inspiration from biological network

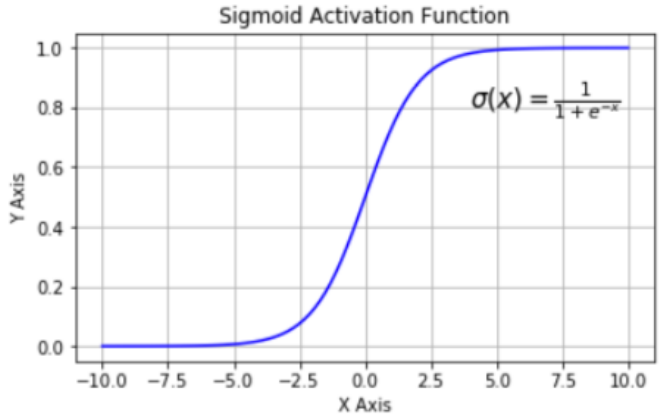


Artificial neuron

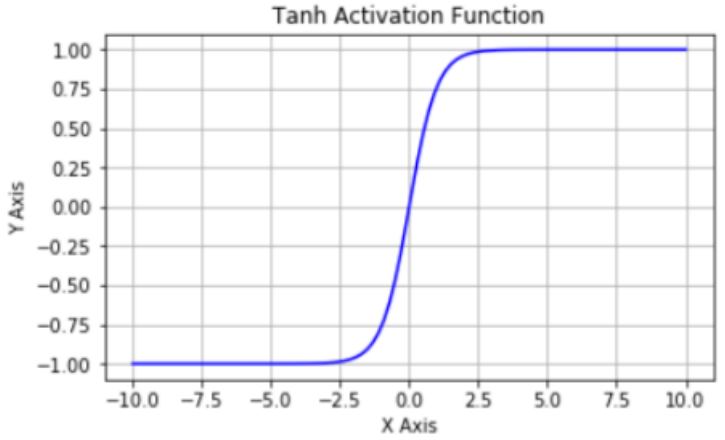
Feed-forward Neural Networks

❑ Activation Function

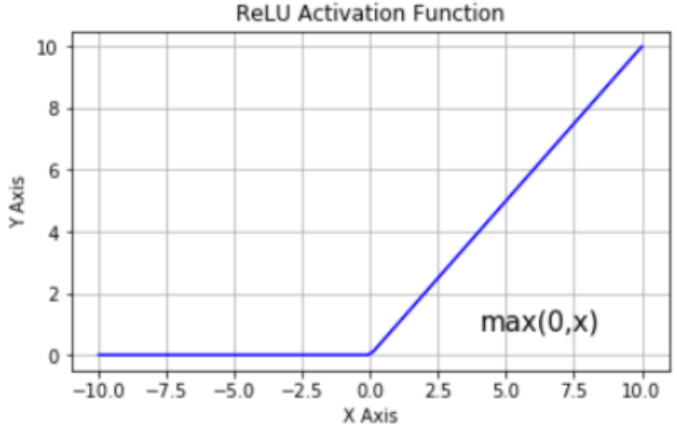
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



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



$$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max\{0, x\} = x\mathbf{1}_{x>0}$$

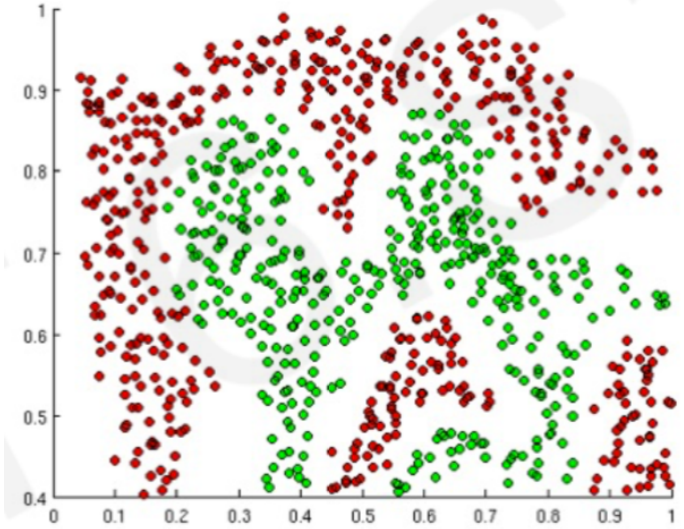


❑ Others

Softplus, Hard tanh, and so on

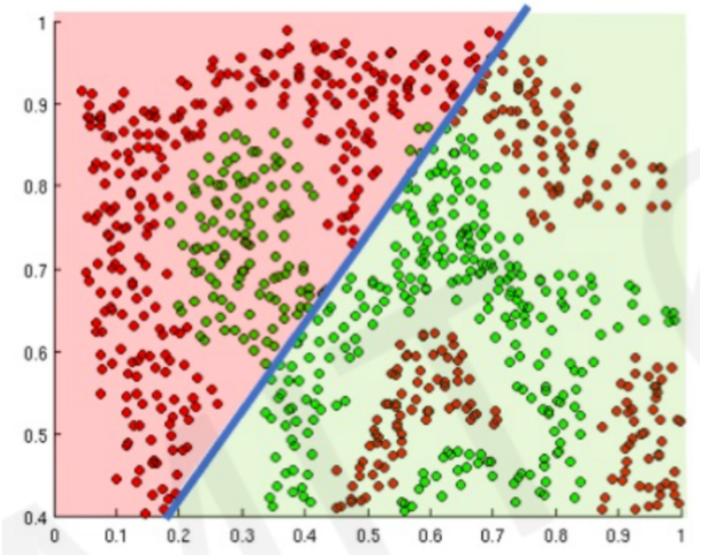
Feed-forward Neural Networks

❑ Why Non-linear Activation

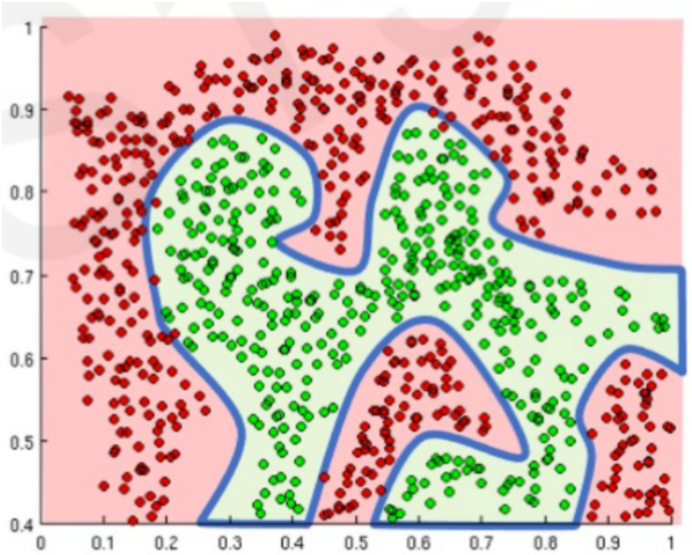


Classify green and red points

(Pictures from MIT 6S191)



Linear Action

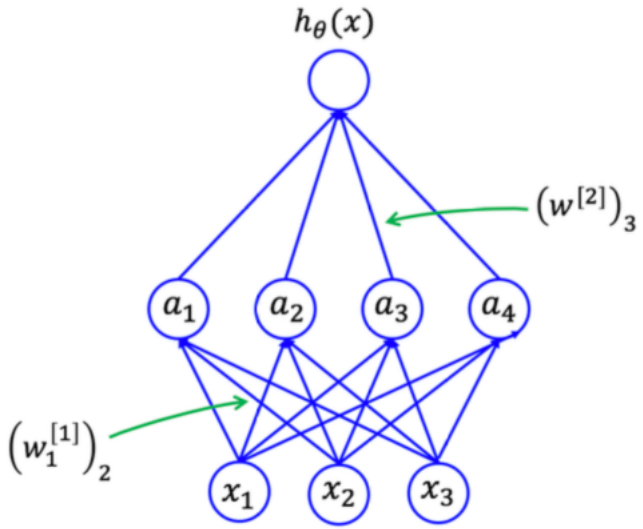


Non-linear Action

Non-linear activations allow us to approximate arbitrary complex function

Feed-forward Neural Networks

❑ Fully Connected Neural Network (Multi-layer Perceptron)



Two Layer

Connection: parameters/weights (subscript or superscript of pars indicate layer number and dependence)

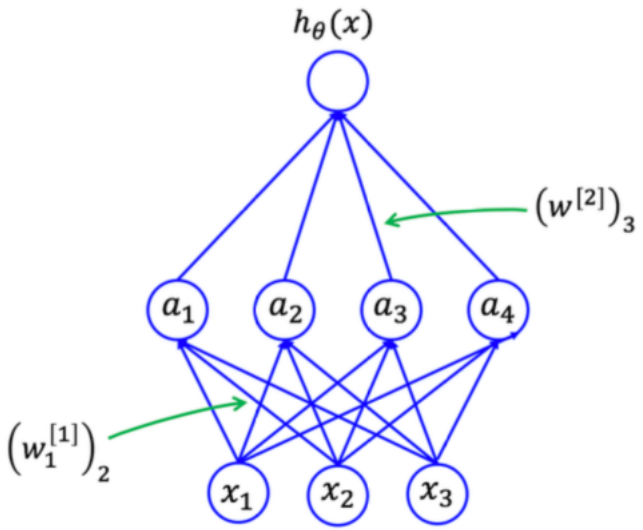
$$\forall j \in [1, ..., m], \quad z_j = {w_j^{[1]}}^\top x + b_j^{[1]} \text{ where } w_j^{[1]} \in \mathbb{R}^d, b_j^{[1]} \in \mathbb{R}$$

$$a_j = \text{ReLU}(z_j),$$

$$a = [a_1, \ldots, a_m]^\top \in \mathbb{R}^m$$

$$h_\theta(x) = {w^{[2]}}^\top a + b^{[2]} \text{ where } w^{[2]} \in \mathbb{R}^m, b^{[2]} \in \mathbb{R}$$

❑ Fully Connected Neural Network (Vectorization)



$$W^{[1]} = \begin{bmatrix} — & w_1^{[1]\top} & — \\ — & w_2^{[1]\top} & — \\ & \vdots & \\ — & w_m^{[1]\top} & — \end{bmatrix} \in \mathbb{R}^{m \times d} \qquad \underbrace{\begin{bmatrix} z_1 \\ \vdots \\ z_m \end{bmatrix}}_{z \in \mathbb{R}^{m \times 1}} = \underbrace{\begin{bmatrix} — & w_1^{[1]\top} & — \\ — & w_2^{[1]\top} & — \\ & \vdots & \\ — & w_m^{[1]\top} & — \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m \times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x \in \mathbb{R}^{d \times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_m^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{m \times 1}}$$

$$z = W^{[1]}x + b^{[1]} \qquad a = \mathrm{ReLU}(z)$$

$$W^{[2]} = [w^{[2]\top}] \in \mathbb{R}^{1 \times m}$$

$$a = \mathrm{ReLU}(W^{[1]}x + b^{[1]})$$
$$h_\theta(x) = W^{[2]}a + b^{[2]}$$

Feed-forward Neural Networks

❑ Multi-layer Fully Connected Neural Network

$$
\begin{aligned}
a^{[1]} &= \mathrm{ReLU}(W^{[1]}x + b^{[1]}) \\
a^{[2]} &= \mathrm{ReLU}(W^{[2]}a^{[1]} + b^{[2]}) \\
&\cdots \\
a^{[r-1]} &= \mathrm{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]}) \\
h_\theta(x) &= W^{[r]}a^{[r-1]} + b^{[r]}
\end{aligned}
$$

If $a^{[k]}$ has dimension m_k,

then the weight matrix $W^{[k]}$ should be of dimension $m_k \times m_{k-1}$, and the bias $b^{[k]} \in \mathbb{R}^{m_k}$. Moreover, $W^{[1]} \in \mathbb{R}^{m_1 \times d}$ and $W^{[r]} \in \mathbb{R}^{1 \times m_{r-1}}$.

The total number of neurons in the network is $m_1 + \cdots + m_r$, and the total number of parameters in this network is $(d+1)m_1 + (m_1+1)m_2 + \cdots + (m_{r-1}+1)m_r$.

❑ Architecture Design

The overall structure of the network: how many units it should have and how these units should be connected to each other.

Organized into groups of units called layers, arrange these layers in a chain structure, with each layer being a function of the layer that preceded it.

$$\boldsymbol{h}^{(1)} = g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right)$$

$$\boldsymbol{h}^{(2)} = g^{(2)}\left(\boldsymbol{W}^{(2)\top}\boldsymbol{h}^{(1)} + \boldsymbol{b}^{(2)}\right)$$

Choose the depth of the network and the width of each layer

Deeper networks often are able to use far fewer units per layer and far fewer parameters and often generalize to the test set, but are also often harder to optimize.

❑ Universal Approximation Theorem

A feedforward network with a linear output layer and at least one hidden layer with non-linear activation function (e.g., sigmoid activation function) can approximate any measurable function from one finite-dimensional space to another with any desired non-zero amount of error, provided that the network is given enough hidden units.

Fact: A large MLP will be able to represent any function.

Fact: We are not guaranteed that the training algorithm will be able to learn that function. 1. The optimization algorithm used for training may not be able to find the optimal parameters that corresponds to the desired function.
2. The training algorithm might choose the wrong function due to overfitting.

❑ Universal Approximation Theorem

Fact-1: A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.

Fact-2: Using deeper models can reduce the number of units required to represent the desired function and can reduce the amount of generalization error.

Fact-3: There exist families of functions which can be approximated efficiently by an architecture with depth greater than some value d, but which require a much larger model if depth is restricted to be less than or equal to d.

Q & A