Lecture 8: Neural Network & Deep Learning

Fall 2022

Kai-Wei Chang

CS @ UCLA

kw+cm146@kwchang.net

The instructor gratefully acknowledges Dan Roth, Vivek Srikuar, Sriram Sankararaman, Fei Sha, Ameet Talwalkar, Eric Eaton, and Jessica Wu whose slides are heavily used, and the many others who made their course material freely available online.

Announcements

- Quiz 2 is due today
- Hw 1 is due next Tue
  - The definition of F1 score will be covered today
- Midterm postpones to 11/1?
  - The practice exam will be posted

What you will learn today

- Optimization
  - Gradient descent
  - Stochastic gradient descent (SGD)
- Evaluation Metrics
- Neural network / Deep learning
  - Non-linear classifier
  - Feed-forward neural network
  - Deep learning architecture

Gradient Descent

Example $\min f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

- Use the following iterative procedure for gradient descent

$$\nabla f(\theta) = \begin{bmatrix} 2(\theta_1^2 - \theta_2)\theta_1 + \theta_1 - 1 \\ -(\theta_1^2 - \theta_2) \end{bmatrix}$$

1. Initialize $\theta_1^{(0)}$ and $\theta_2^{(0)}$, and $t = 0$
2. do

Type equation here.

$$\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \eta \left[2({\theta_1^{(t)}}^2 - \theta_2^{(t)})\theta_1^{(t)} + \theta_1^{(t)} - 1 \right]$$

$$\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \eta \left[-({\theta_1^{(t)}}^2 - \theta_2^{(t)}) \right]$$

$$t \leftarrow t + 1$$

3. until $f(\boldsymbol{\theta}^{(t)})$ *does not change much*

Remarks

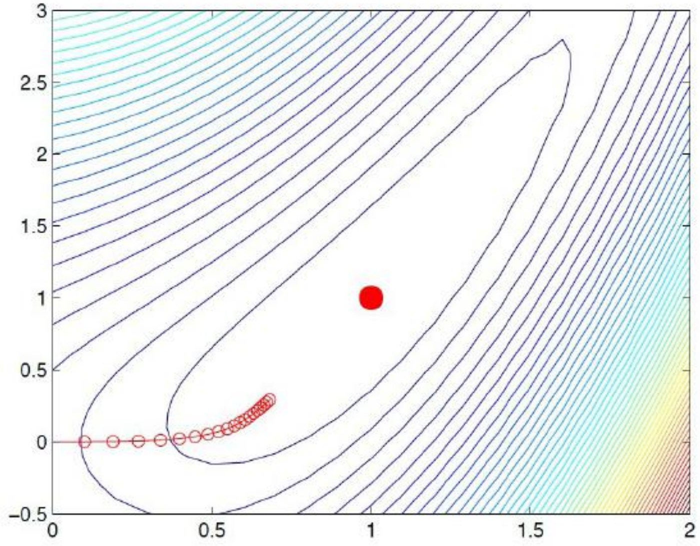
- η is often called step size or learning rate -- how far our update will go along the the direction of the negative gradient
- With a suitable choice of η, the iterative procedure converges to a stationary point where

$$\frac{\partial f}{\partial \boldsymbol{\theta}} = 0$$

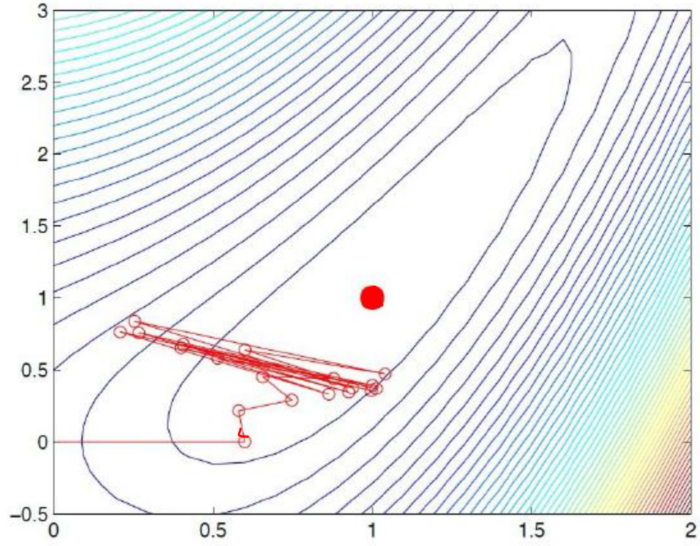
- A stationary point is only necessary for being the minimum

Choosing the right learning rate (η) is important

small η is too slow?



large η is too unstable?



Recap: Logistic Regression

- Training data: $S = \{(x_i, y_i)\}$, m examples
- Hypothesis space:

$$H = \{\ \mathrm{h} \mid h : X \rightarrow P(Y \mid X), h(x) = \sigma\,(\,w^T x + b)\,\}$$

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

i.e., model $P(Y|X)$ by $\sigma(w^T x + b)$

- How to find the best $h \in H$: maximum log-likelihood

$$\arg\max - \sum_{i=1}^{m} \log(1 + \exp(-y_i(w^T x_i + b)))$$

Gradient Descent for Logistic Regression

- Maximum log-likelihood

$$\arg\max -\sum_{i=1}^{m} \log(1 + \exp(-y_i(w^T x_i + b)))$$

- Equivalent to the following minimization problem

$$\arg\min \underbrace{\sum_{i=1}^{m} \log(1 + \exp(-y_i(w^T x_i + b)))}_{L(w,b)}$$

- Gradient of $L(w, b)$

$$\nabla L(w, b) = \sum_{i=1}^{m} \nabla \log(1 + \exp(-y_i(w^T x_i + b)))$$

Recap: Gradient

- Let z to be a n-dimensional vector of variables, $f(z)$ is a function of z

$$\nabla f(z) = \begin{bmatrix} \partial f(z)/\partial z_1 \\ \partial f(z)/\partial z_2 \\ \vdots \\ \partial f(z)/\partial z_{n-1} \\ \partial f(z)/\partial z_n \end{bmatrix}$$

Exercise

- Let $z = [z_1, z_2, z_3]^T$ to be a 3-dimensional vector of variables, $a = [3, 2, 4]^T$

  $f(z) = \log(a^T z)$

  $= \log(3z_1 + 2z_2 + 4z_3)$

$$\nabla f(z) = \begin{bmatrix} \partial f(z)/\partial z_1 \\ \partial f(z)/\partial z_2 \\ \vdots \\ \partial f(z)/\partial z_{n-1} \\ \partial f(z)/\partial z_n \end{bmatrix}$$

- $\nabla f(z) = \begin{bmatrix} \partial f(z)/\partial z_1 \\ \partial f(z)/\partial z_2 \\ \partial f(z)/\partial z_3 \end{bmatrix} = ?$

Exercise

- Let $z = [z_1, z_2, z_3]^T$ to be a 3-dimensional vector of variables, $a = [3, 2, 4]^T$
  $f(z) = \log(a^T z)$
  $= \log(3z_1 + 2z_2 + 4z_3)$

$$\nabla f(z) = \begin{bmatrix} \partial f(z)/\partial z_1 \\ \partial f(z)/\partial z_2 \\ \vdots \\ \partial f(z)/\partial z_{n-1} \\ \partial f(z)/\partial z_n \end{bmatrix}$$

- $\nabla f(z) = \begin{bmatrix} \partial f(z)/\partial z_1 \\ \partial f(z)/\partial z_2 \\ \partial f(z)/\partial z_3 \end{bmatrix} = \begin{bmatrix} \frac{3}{3z_1+2z_2+4z_3} \\ \frac{2}{3z_1+2z_2+4z_3} \\ \frac{4}{3z_1+2z_2+4z_3} \end{bmatrix}$

$$= \frac{1}{3z_1+2z_2+4z_3} \begin{bmatrix} 3 \\ 2 \\ 4 \end{bmatrix} = \frac{1}{a^T z} a$$

Gradient of $L(w, b)$

$$\nabla L(w,b) = \Sigma_{i=1}^{m} \ \nabla \underbrace{\log(1 + \exp(-y_i(w^T x_i + b)))}_{\nabla \log \frac{1}{\sigma(y_i(w^T x_i + b))}}$$

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

- $\nabla \log \frac{1}{\sigma(z)} = \nabla \log\big(1 + \exp(-z)\big) = -\frac{\exp(-z)}{1+\exp(-z)}$

$$= -\frac{1 + \exp(-z) - 1}{1 + \exp(-z)} = -1 + \frac{1}{1 + \exp(-z)}$$

$$= \sigma(z) - 1$$

Gradient of $L(w, b)$

$$\nabla L(w,b) = \Sigma_{i=1}^{m} \nabla \underbrace{\log(1 + \exp(-y_i(w^T x_i + b)))}_{\nabla \log \frac{1}{\sigma(y_i(w^T x_i + b))}}$$

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

- Using $\nabla \log \frac{1}{\sigma(z)} = \sigma(z) - 1$

Partial gradient w.r.t w

$$\nabla_w L(w,b) = \Sigma_{i=1}^{m} \nabla_{\mathrm{w}} \log \frac{1}{\sigma(y_i(w^T x_i + b))}$$

$$= \Sigma_{i=1}^{m} (\sigma(y_i(w^T x_i + b)) - 1) y_i x_i$$

$$\nabla_b L(w,b) = \Sigma_{i=1}^{m} (\sigma(y_i(w^T x_i + b)) - 1) y_i$$

Gradient descent for logistic regression

Given a training data set $S = \{(x_i, y_i)\}, i = 1 \ldots m$

1. Initialize w (e.g., $w \leftarrow 0 \in R^n$)
2. For epoch $1 \ldots T$:
3. Compute $\nabla_w L(w, b)$ and $\nabla_b L(w, b)$

Loop over instance to compute the summation

$$\nabla_w L(w,b) = \Sigma_{i=1}^{m} (\sigma(y_i(w^T x_i + b)) - 1) y_i x_i$$
$$\nabla_b L(w,b) = \Sigma_{i=1}^{m} (\sigma(y_i(w^T x_i + b)) - 1) y_i$$

4. Update w and b

$$w \leftarrow w - \eta \nabla_w L(w, b)$$
$$\mathrm{b} \leftarrow \mathrm{b} - \eta \nabla_b\, L(w, b)$$

5. Return w and b

Remark

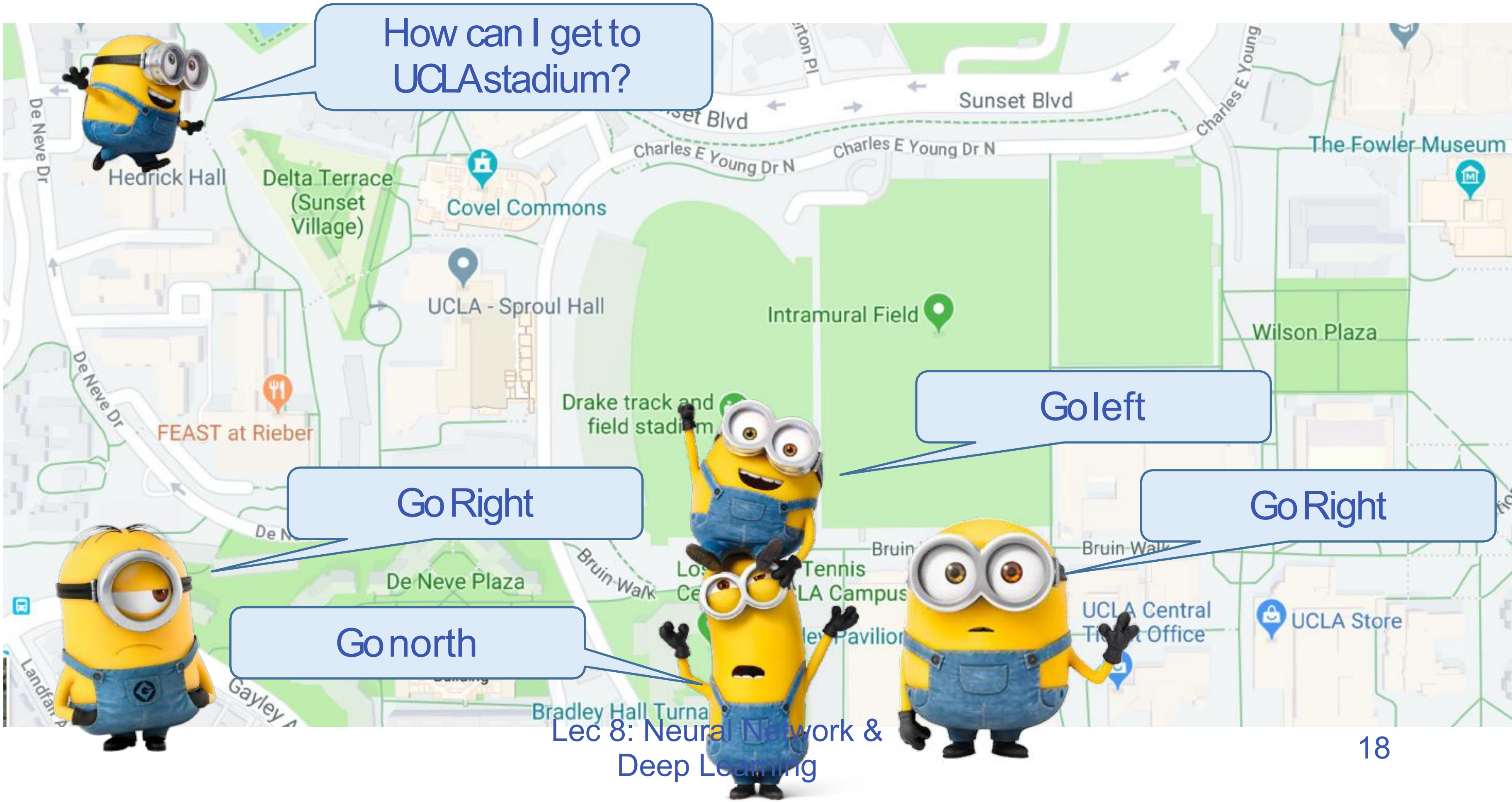
$$\nabla_w L(w,b) = \Sigma_{i=1}^{m}(\sigma(y_i(w^T x_i + b)) - 1)y_i x_i$$
$$\nabla_b L(w,b) = \Sigma_{i=1}^{m}(\sigma(y_i(w^T x_i + b)) - 1)y_i$$

- Need to compute $(\sigma(y_i(w^T x_i + b)) - 1)$ for every data point (x_i, y_i)
- Gradient descent usually needs many iterations to converge
- When size of data (m) is large, computing $\nabla L(w,b)$ is expensive

Stochastic Gradient Descent

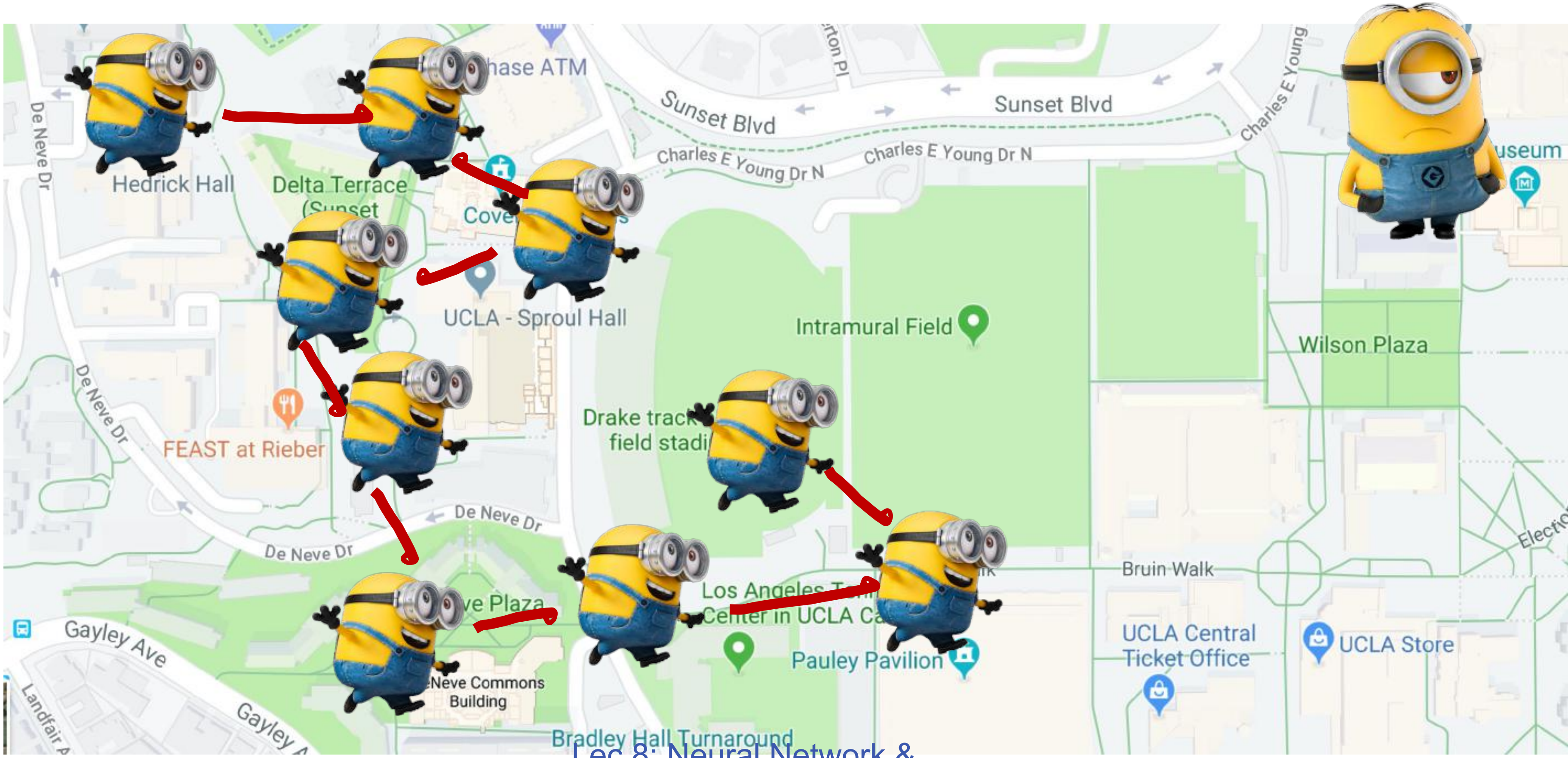
Intuition

Asking direction. Gradient descent: compute gradient of all instances.



Intuition

Asking direction. Stochastic Gradient descent:
compute approximate gradient by one instance



Incremental/Stochastic gradient descent

Repeat for each example ($\mathbf{x}_i$, y_i)

- Use this example to calculate approximate the gradient and update the model

Contrast with *batch gradient descent* which makes one update to the weight vector for every pass over the data

Recap: Gradient Descent

- $\nabla_w L(w,b) = \sum_{i=1}^{m} \underbrace{(\sigma(y_i(w^T x_i + b)) - 1) y_i x_i}_{\nabla L_i(w,b)}$
- Gradient descent update:
  $w \leftarrow w - \eta \sum_{i=1}^{m} \nabla_{\mathrm{w}} L_i(w,b)$
- Alternative way of gradient update
  For i = 1 … m
  $$w \leftarrow w - \eta \, \nabla_{\mathrm{w}} L_i(w,b)$$

Stochastic Gradient Descent

- $\nabla_w L(w,b) = \sum_{i=1}^{m} \nabla_w L_i(w,b) = m \frac{\sum_{i=1}^{m} \nabla L_i(w,b)}{m} = m\, \underbrace{\text{avg}\left(\nabla_{\mathrm{w}} L_i(w,b)\right)}_{\text{Average } L_i(w,b) \text{ over instance}}$
- $\text{avg}\left(\nabla_{\mathrm{w}} L_i(w,b)\right) = E_{(x_i,y_i)\sim S}\left[\nabla_{\mathrm{w}} L_i(w,b)\right]$
- Gradient descent update:
  $w \leftarrow w - \eta \sum_{i=1}^{m} \nabla_{\mathrm{w}} L_i(w,b)$
- Stochastic gradient descent
  - Repeat until converge
    Sample a data point (x_i, y_i) from S
    $w \leftarrow w - \eta' \nabla_{\mathrm{w}} L_i(w,b)$

Expectation of gradient $L_i(w,b)$ over dataset S

Stochastic Gradient descent for logistic regression

Given a training data set $S = \{(x_i, y_i)\}, i = 1 \ldots m$

1. Initialize w (e.g., $w \leftarrow 0 \in R^n$)
2. For epoch $1 \ldots T$:
3. Sample a data point (x_i, y_i) from S
4. Compute $\nabla_w L_i(w, b)$ and $\nabla_b L_i(w, b)$

$$\nabla_w L_i(w, b) = (\sigma(y_i(w^T x_i + b)) - 1) y_i x_i$$

$$\nabla_b L_i(w, b) = (\sigma(y_i(w^T x_i + b)) - 1) y_i$$

5. Update w and b

$$w \leftarrow w - \eta \nabla_w L_i(w, b)$$

$$\mathrm{b} \leftarrow \mathrm{b} - \eta \nabla_b L_i(w, b)$$

6. Return w and b

The Perceptron Algorithm [Rosenblatt 1958]

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For epoch $1 \ldots T$:
3. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
4. if $y(\boldsymbol{w}^\top \boldsymbol{x}) < 0$
5. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y \boldsymbol{x}$
6. Return $\boldsymbol{w}$

Prediction: $y^{\text{test}} \leftarrow sgn(\boldsymbol{w}^\top \boldsymbol{x}^{\text{test}})$

The Perceptron Algorithm [Rosenblatt 1958]

Given a training set $\mathcal{D} = \{(\boldsymbol{x}, y)\}$

1. Initialize $\boldsymbol{w} \leftarrow \mathbf{0} \in \mathbb{R}^n$
2. For epoch $1 \ldots T$:
3. For $(\boldsymbol{x}, y)$ in $\mathcal{D}$:
4. if $y(\boldsymbol{w}^\top \boldsymbol{x}) < \mathbf{0}$
5. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta y \boldsymbol{x}$
6. Return $\boldsymbol{w}$

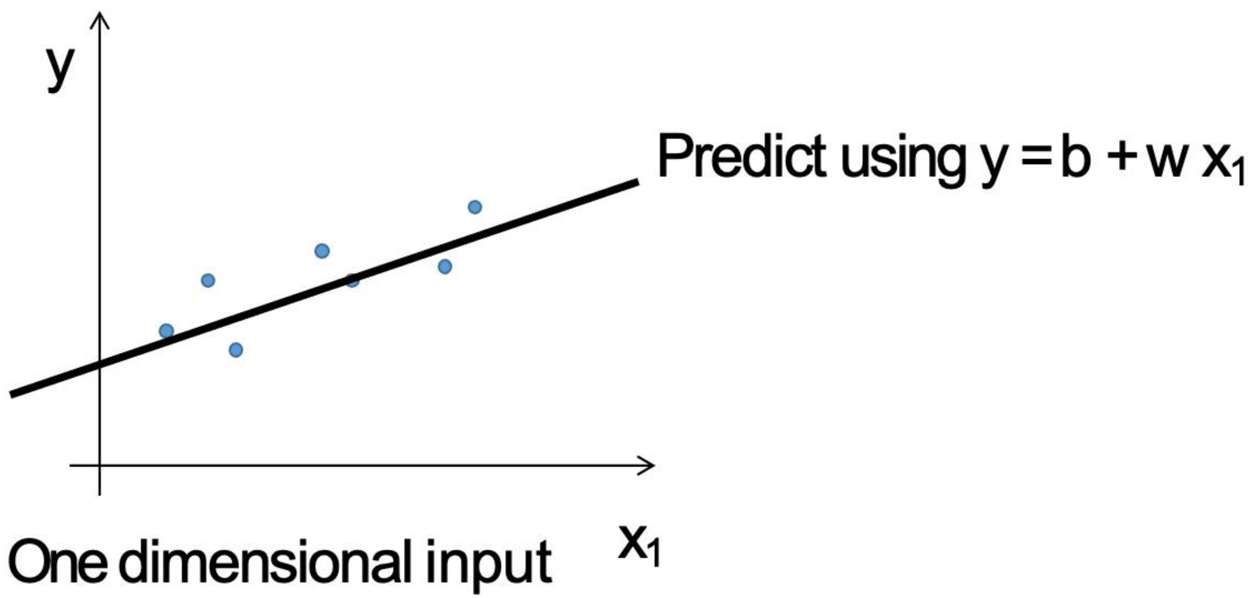
Prediction: y^{test}

Perceptron effectively minimizing:

$$\sum_i \max(0,\ 1 - \ y_i(\mathbf{w}^T \mathbf{x}_i))$$

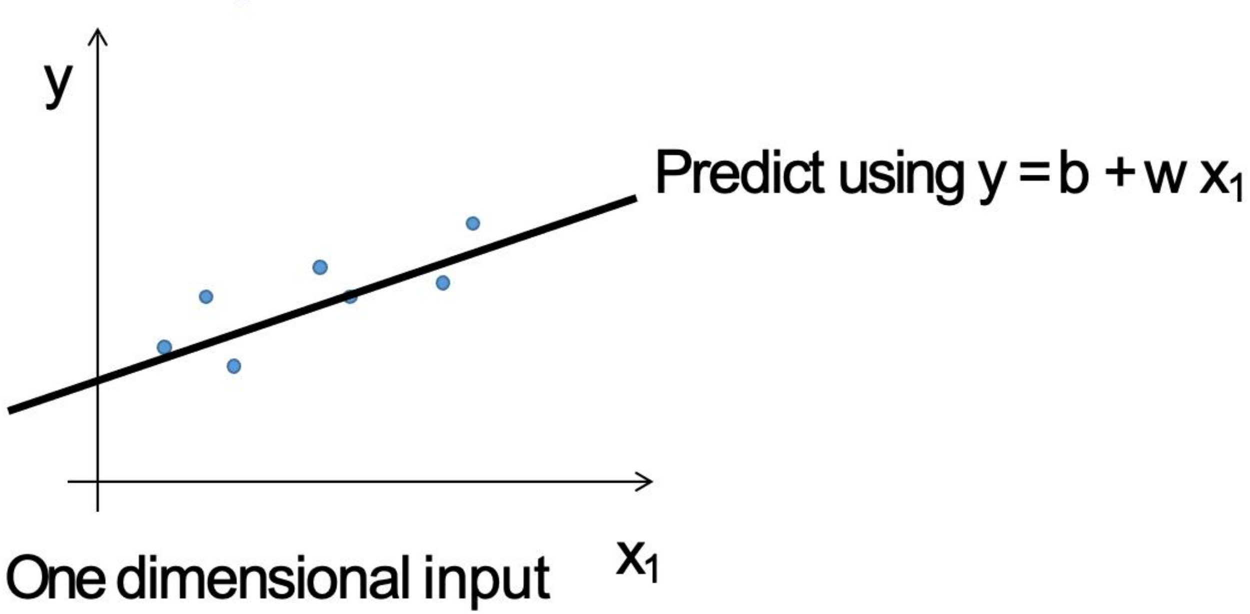
Linear Regression

- Find a line $w^T x + b$ to approximate real-value output y based on input x
  e.g., predict house price next year



Linear Regression

- Find a line to approximate real-value output
  e.g., predict house price next year

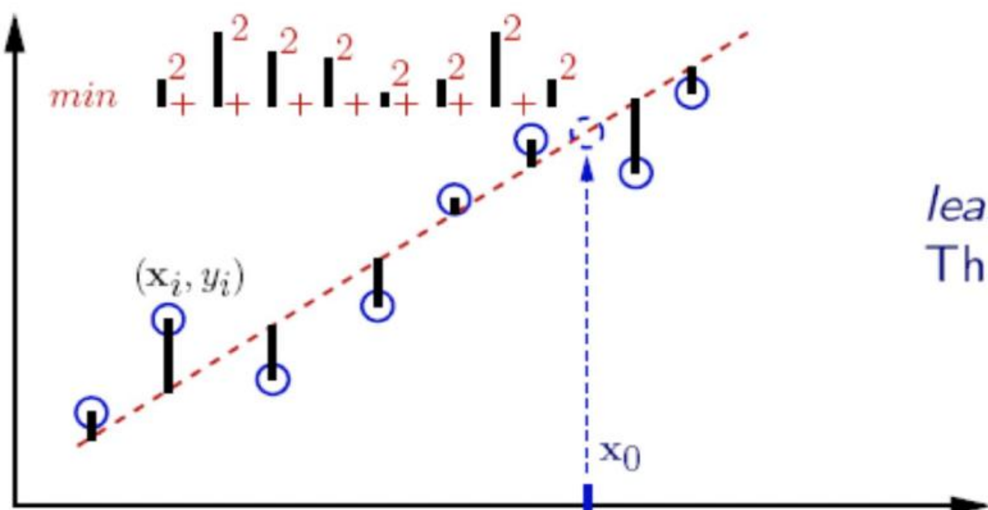


Least Mean Squares (LMS) Regression

Given a dataset $S = \{(x_i, y_i)\}_{i=1..m}, x_i \in \mathbb{R}^n, y \in \mathbb{R}$

$$\arg\min_{w,b} \frac{1}{2}\sum_{i}^{m}(y_i - (w^T x_i + b))^2$$

Learning: minimizing mean squared error



least squares (LSQ)
The fitted line is used as a predictor

Exercise

- Derive the stochastic gradient descent algorithm for solving LMS regression

$$\arg\min_{w,b} \frac{1}{2} \sum_{i}^{m} (y_i - (w^T x_i + b))^2$$

What you will learn today

- Optimization
  - Gradient descent
  - Stochastic gradient descent (SGD)
- Evaluation Metrics
- Neural network / Deep learning
  - Non-linear classifier
  - Feed-forward neural network
  - Deep learning architecture

Accuracy and Error Rate

True label	-	-	-	-	+	-	-	-	-	-	+	-	-	+	-	-
Predicted label	-	-	-	-	+	-	+	-	-	-	-	+	-	+	-	-

Error rate = 3/16

Accuracy = 13/16

Accuracy and Error Rate

True label	-	-	-	-	+	-	-	-	-	-	+	-	-	+	-	-
Predicted label	-	-	-	-	+	-	+	-	-	-	-	+	-	+	-	-

Error rate = 3/16 = 19%

Accuracy = 13/16 = 81%

When data is unbalanced, check the performance of majority baseline

True label	-	-	-	-	+	-	-	-	-	-	+	-	-	+	-	-
Predicted label	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Confusion Matrix

True label	-	-	-	-	+	-	-	-	-	-	+	-	-	+	-	-
Predicted label	-	-	-	-	+	-	+	-	-	-	-	+	-	+	-	-

	True Label Positive	True Label Negative
Predicted Label Positive	2 True Positive (TP)	2 False Positive (FP)
Predicted Label Negative	1 False Negative (FN)	16 True Negative (TN)

Accuracy = (TP+TN)/(TP+TN+FN+FP)

Precision, Recall

True label	-	-	-	-	+	-	-	-	-	-	+	-	-	+	-	-
Predicted label	-	-	-	-	+	-	+	-	-	-	-	+	-	+	-	-

	True Label Positive	True Label Negative
Predicted Label Positive	2 True Positive (TP)	2 False Positive (FP)
Predicted Label Negative	1 False Negative (FN)	16 True Negative (TN)

Accuracy = (TP+TN)/(TP+TN+FN+FP)

Precision = (TP)/(TP+FP)

Recall = (TP)/(TP+FN)

F1 Score

- Harmonic mean of precision and recall:

$$\frac{1}{F_1} = \left(\frac{1}{P} + \frac{1}{R}\right)/2$$

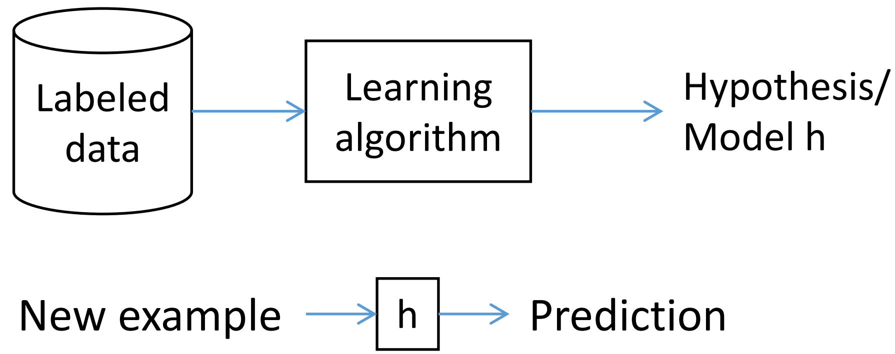
$$F_1 = \frac{2TP}{2TP + FP + FN}$$

What you will learn today

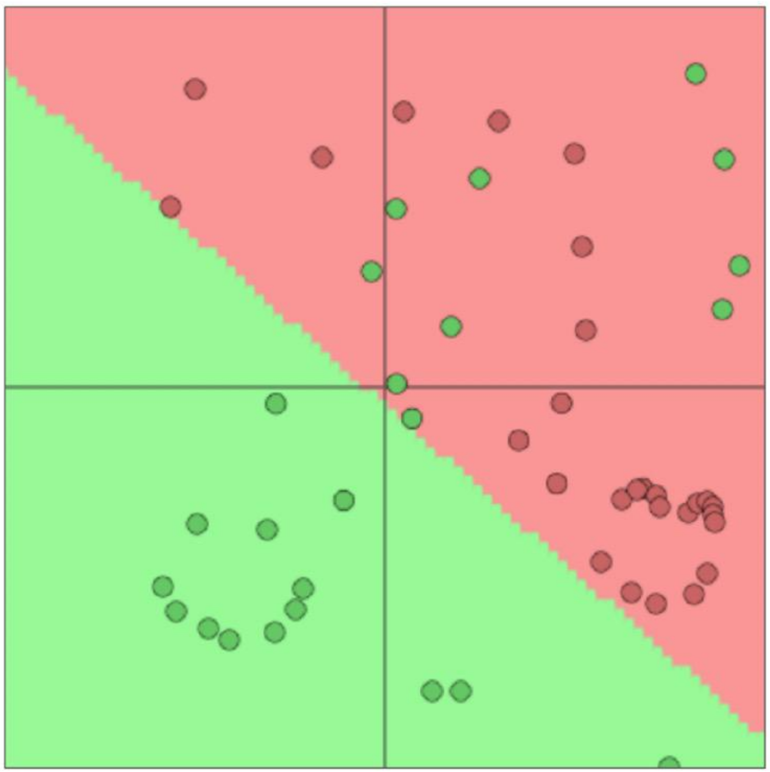
- Optimization
  - Gradient descent
  - Stochastic gradient descent (SGD)
- Evaluation Metrics
- Neural network / Deep learning
  - Non-linear classifier
  - Feed-forward neural network
  - Deep learning architecture

Checkpoint: The bigger picture

- Supervised learning: instances, concepts, and hypotheses
- Specific learners
  - Decision trees
  - K-NN
  - Perceptron
  - Logistic regression
- General ML ideas
  - Feature vectors
  - Overfitting
  - Probabilistic model



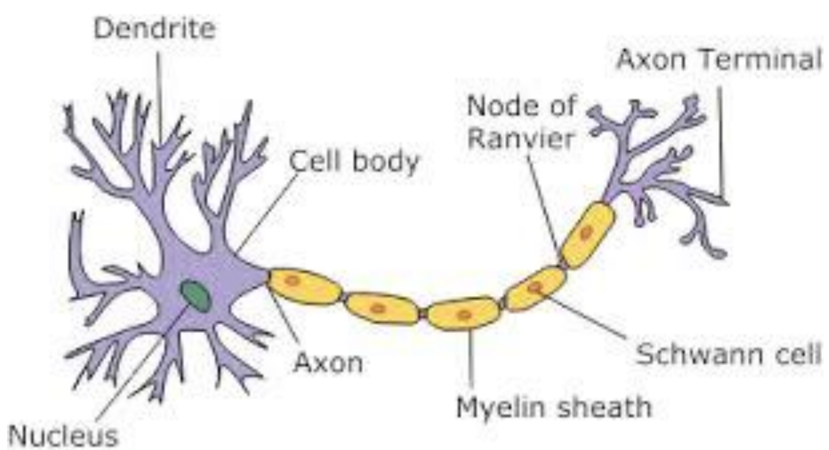
Non-Linear Decision Boundary

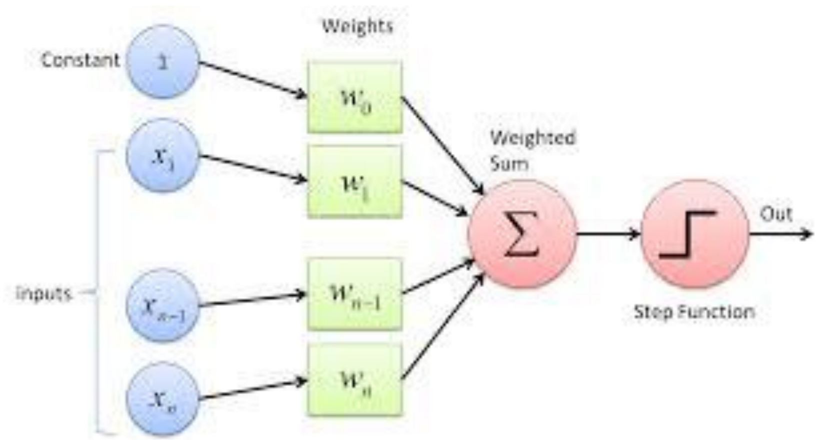


Neural Networks

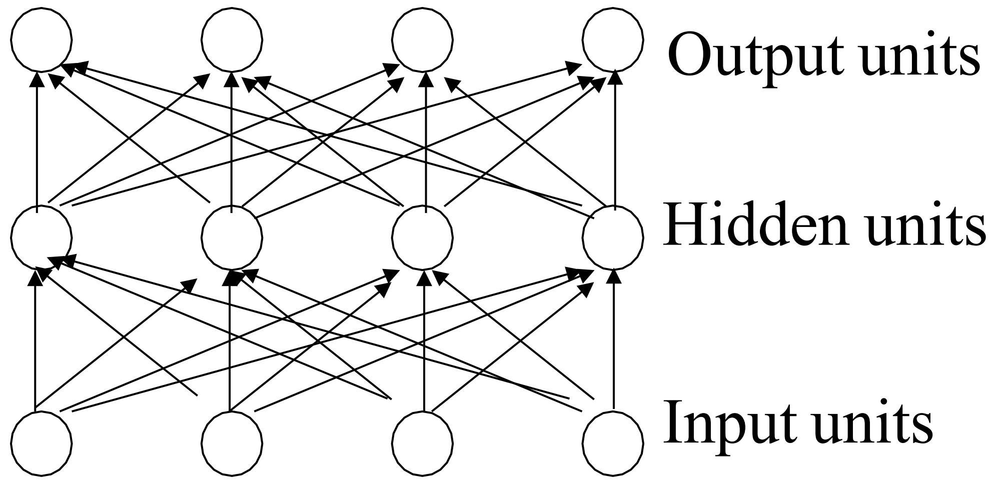
- Design to mimic the brain.

- Artificial neural networks are not nearly as complex or intricate as the actual brain structure





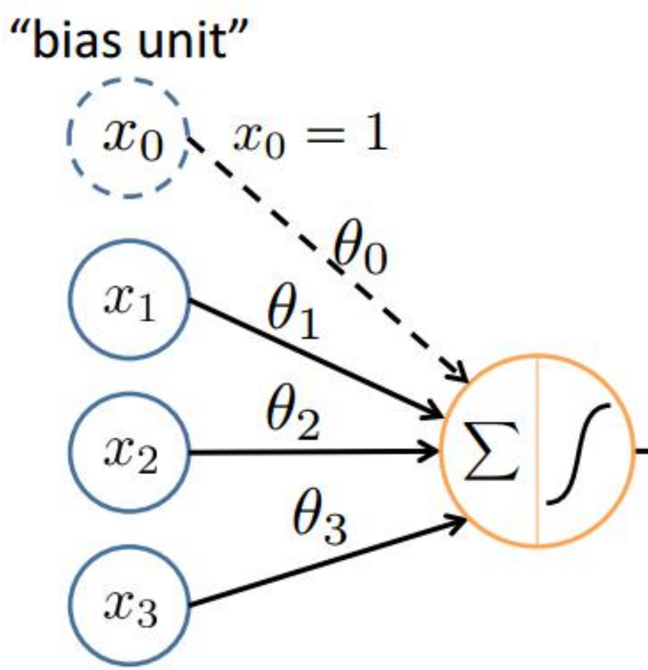
Feed-forward neural network



Layered feed-forward network

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**

Neuron Model Example: Logistic Unit

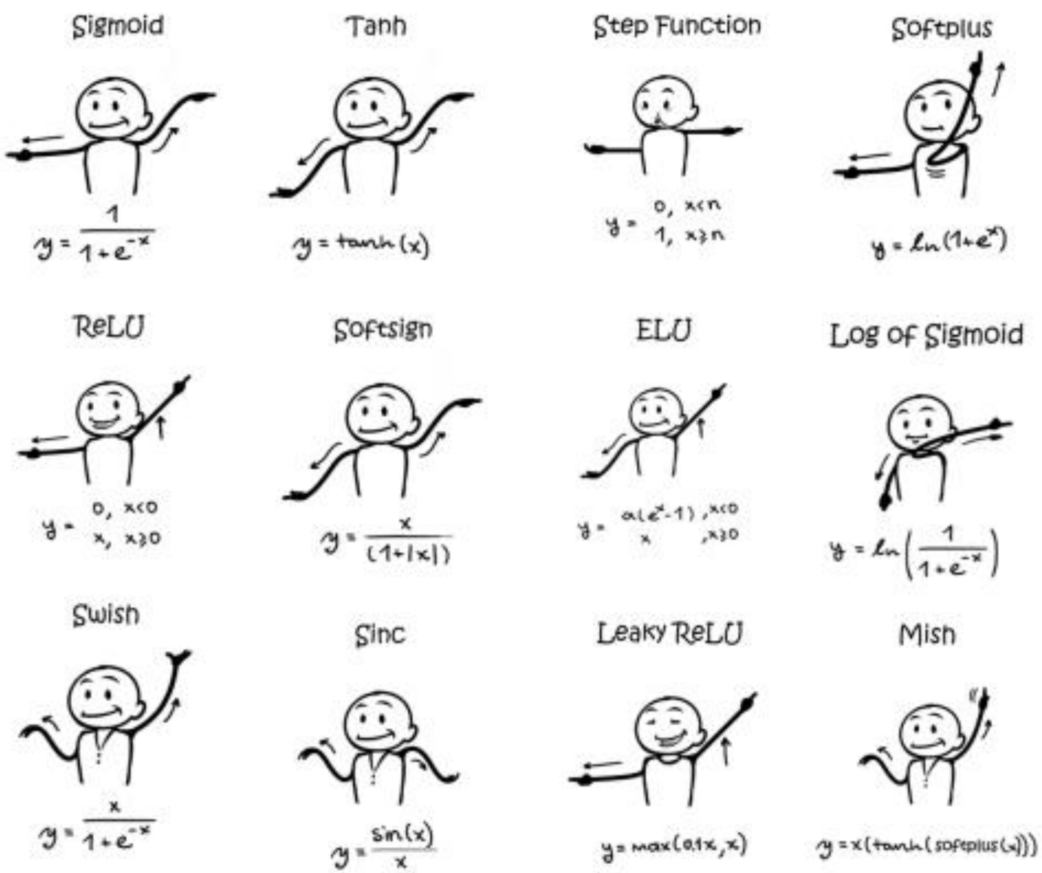


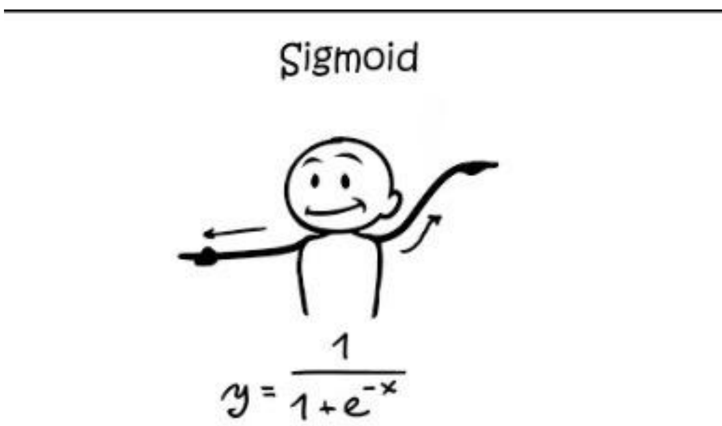
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = g\left(\boldsymbol{\theta}^\intercal \mathbf{x}\right) = \frac{1}{1 + e^{-\boldsymbol{\theta}^\intercal \mathbf{x}}}$$

Sigmoid (logistic) activation function: $g(z) = \dfrac{1}{1 + e^{-z}}$

Activation function

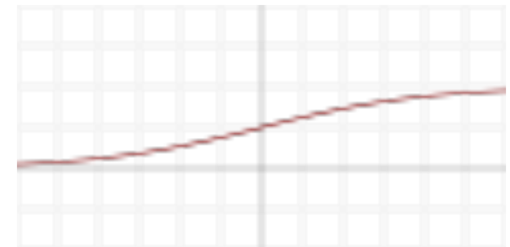




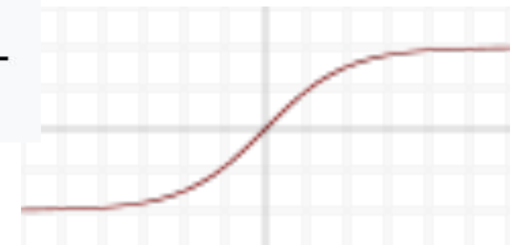
https://sefiks.com/2020/02/02/dance-moves-of-deep-learning-activation-functions/
by Sefik Ilkin Serengil

Activation functions

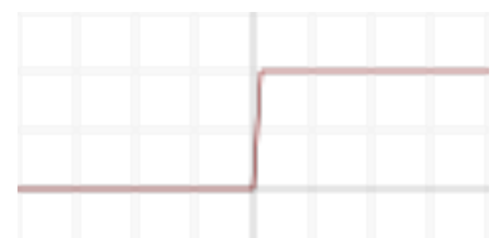
- sigmoid function $$f(x) = \frac{1}{1+e^{-x}}$$



- hyperbolic tangent $$\tanh(x) \doteq \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

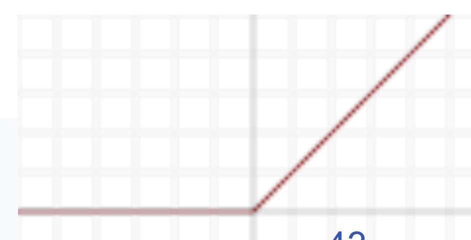


- step function $$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

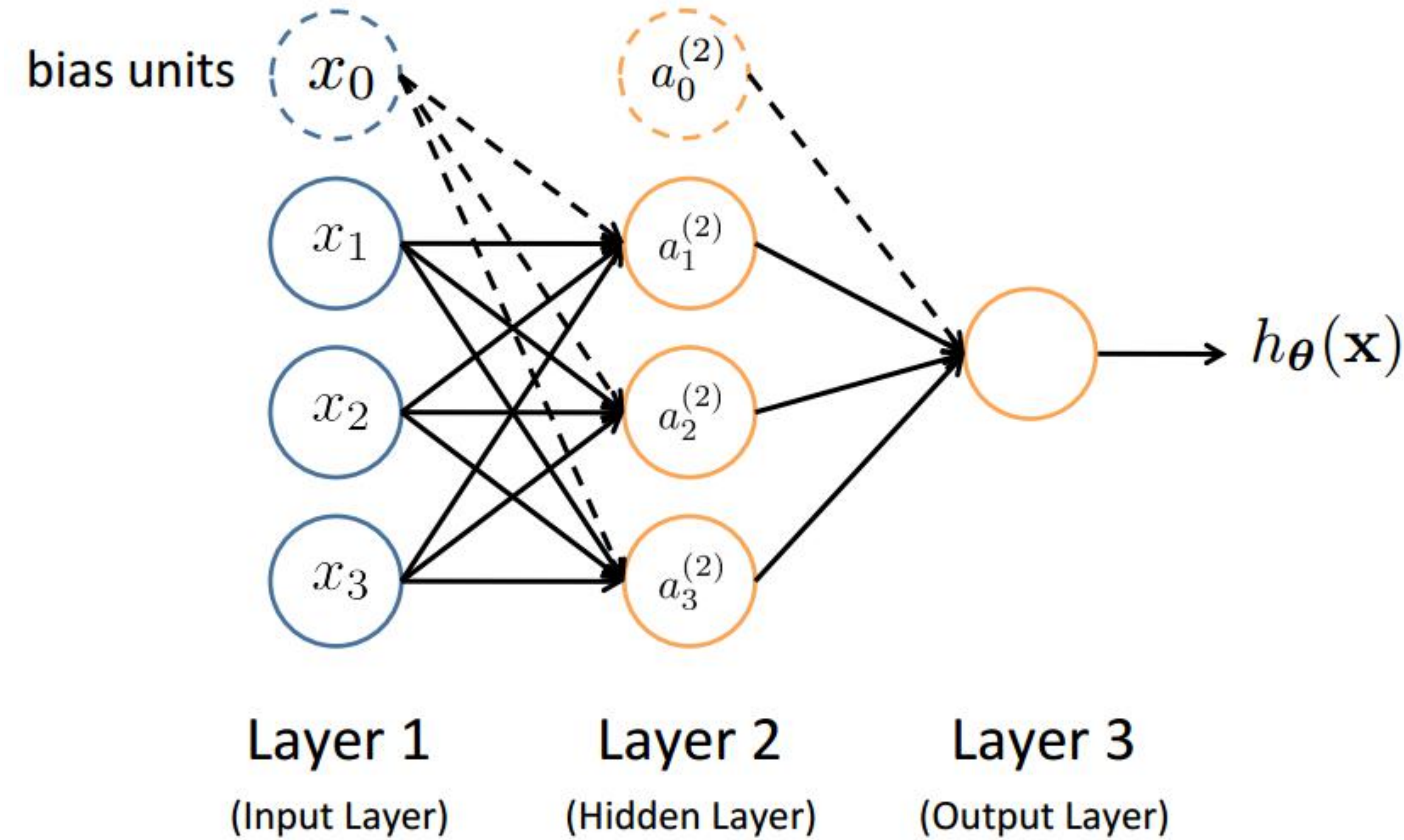


- Rectified linear unit (ReLU)

$$(x)^{+} \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$



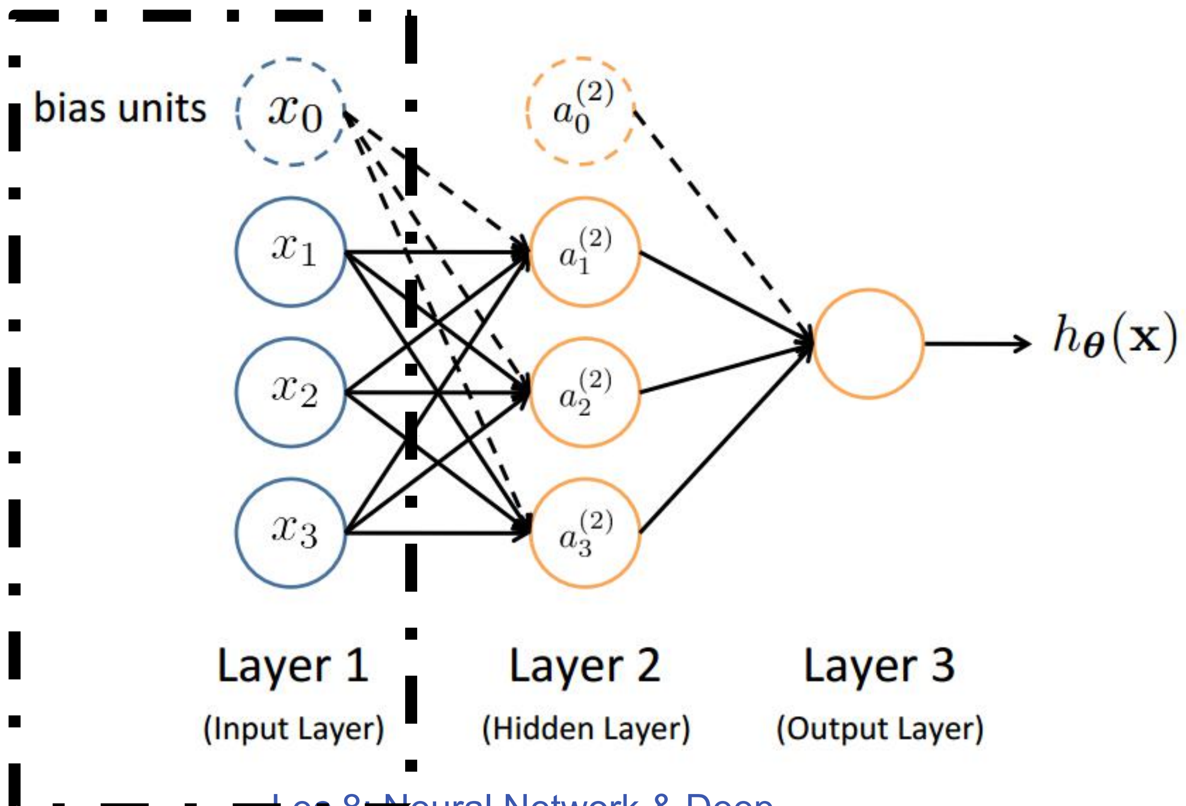
Neural Network



Slide by Andrew Ng

Feed-Forward Process

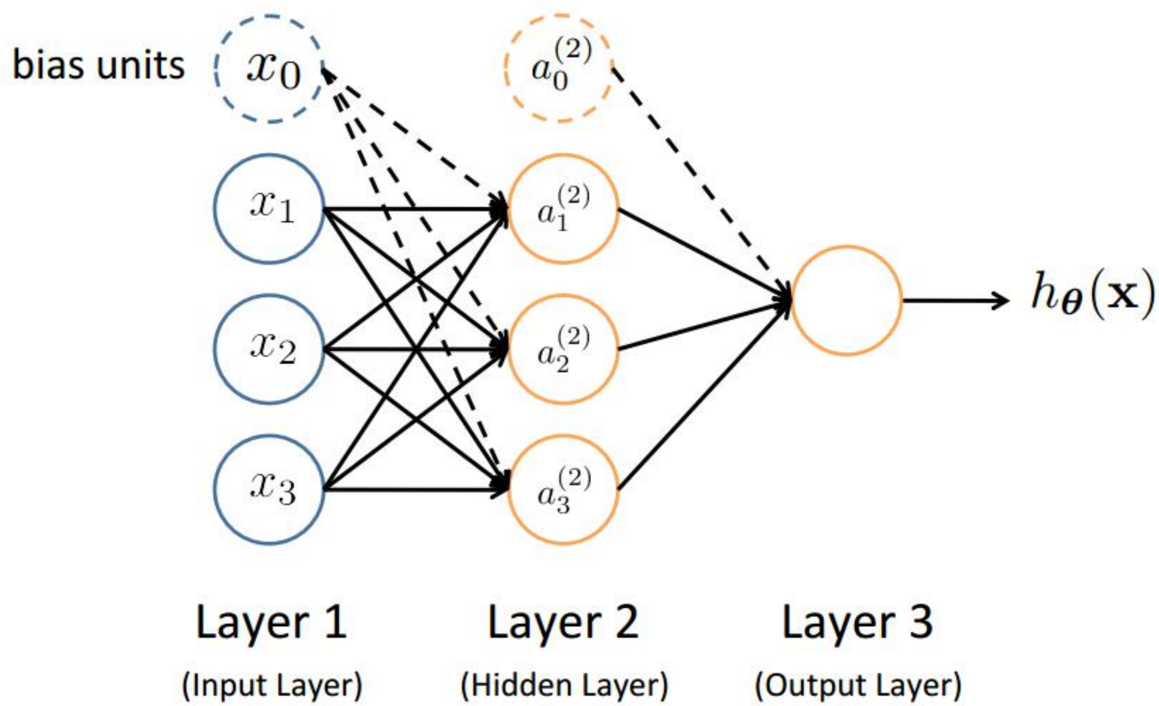
- Input layer units are set by some exterior function (think of these as **sensors**), which causes their output links to be **activated** at the specified level



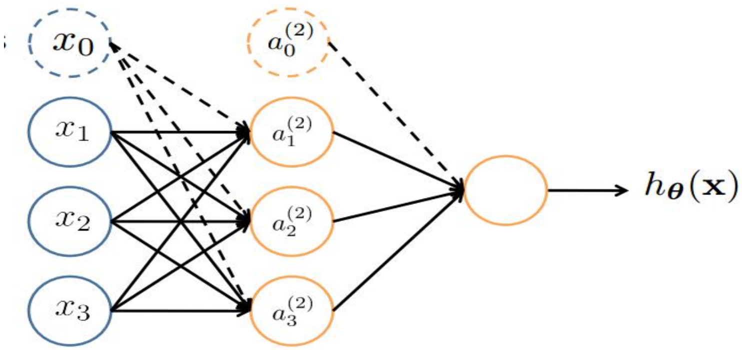
Based on slide by T. Finin, M. desJardins, L Getoor, R. Par

Feed-Forward Process

- Working forward through the network, the **input function** of each unit is applied to compute the input value
- The **activation function** transforms this input function into a final value



Neural Network



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = weight matrix controlling function mapping from layer j to layer j + 1

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

If network has s_j units in layer j *and* s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ has dimension $s_{j+1} \times (s_j+1)$.

$$\Theta^{(1)} \in \mathbb{R}^{3\times 4} \qquad \Theta^{(2)} \in \mathbb{R}^{1\times 4}$$

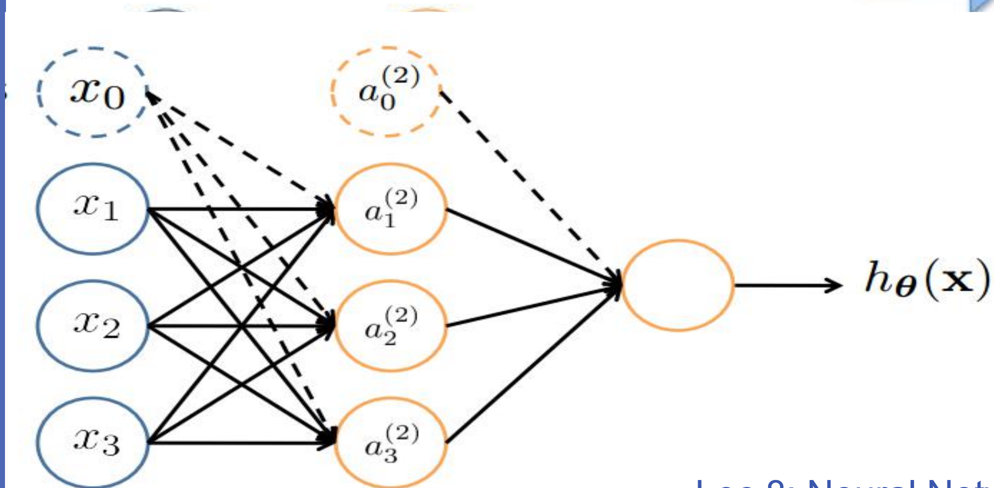
Vectorization

$$a_1^{(2)} = g\left(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3\right) = g\left(z_1^{(2)}\right)$$

$$a_2^{(2)} = g\left(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3\right) = g\left(z_2^{(2)}\right)$$

$$a_3^{(2)} = g\left(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3\right) = g\left(z_3^{(2)}\right)$$

$$h_\Theta(\mathbf{x}) = g\left(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)}\right) = g\left(z_1^{(3)}\right)$$



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Exercise

- Why do we need non-linear activation functions?
  - What happen if $g(z) = z$

Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

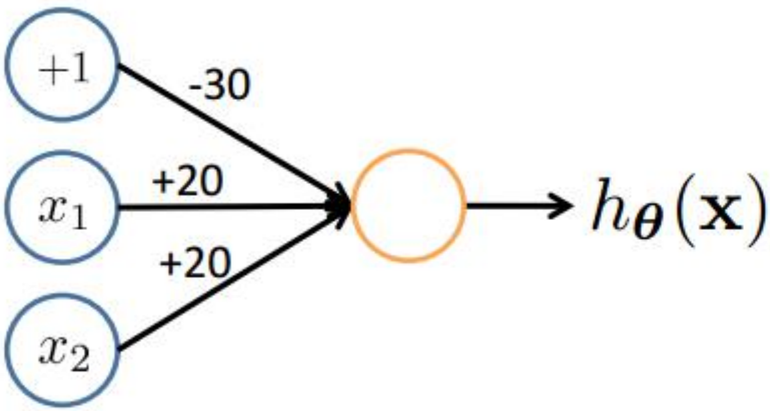
$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

Non-Linear Representations

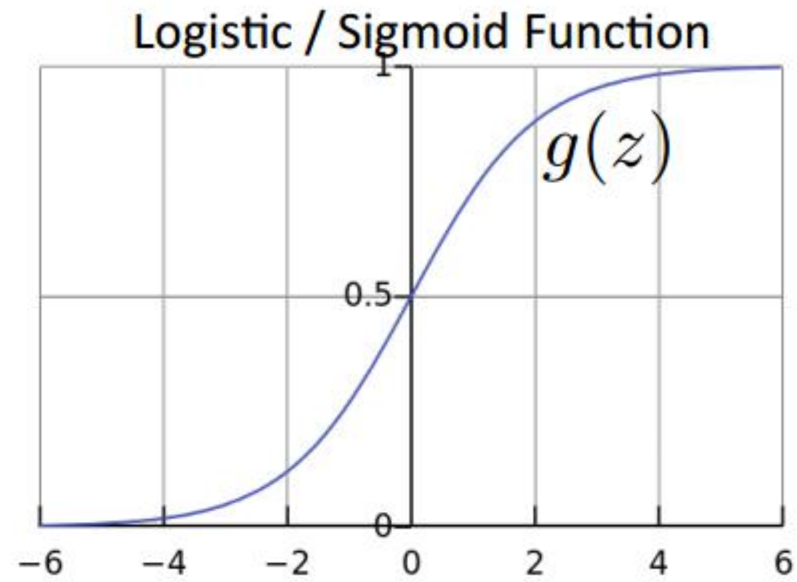
Simple example: AND

$x_1, x_2 \in \{0, 1\}$

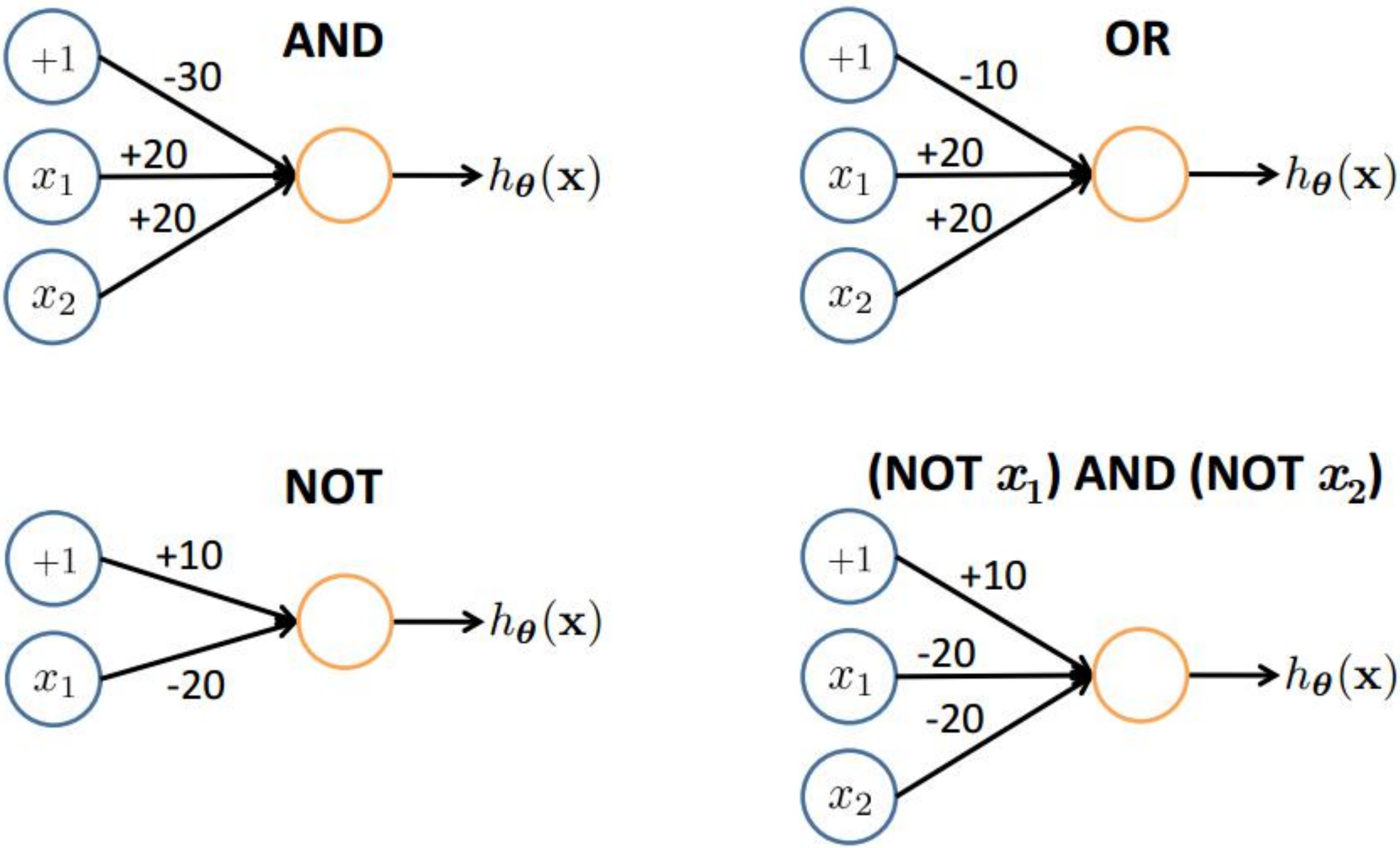
$y = x_1 \text{ AND } x_2$

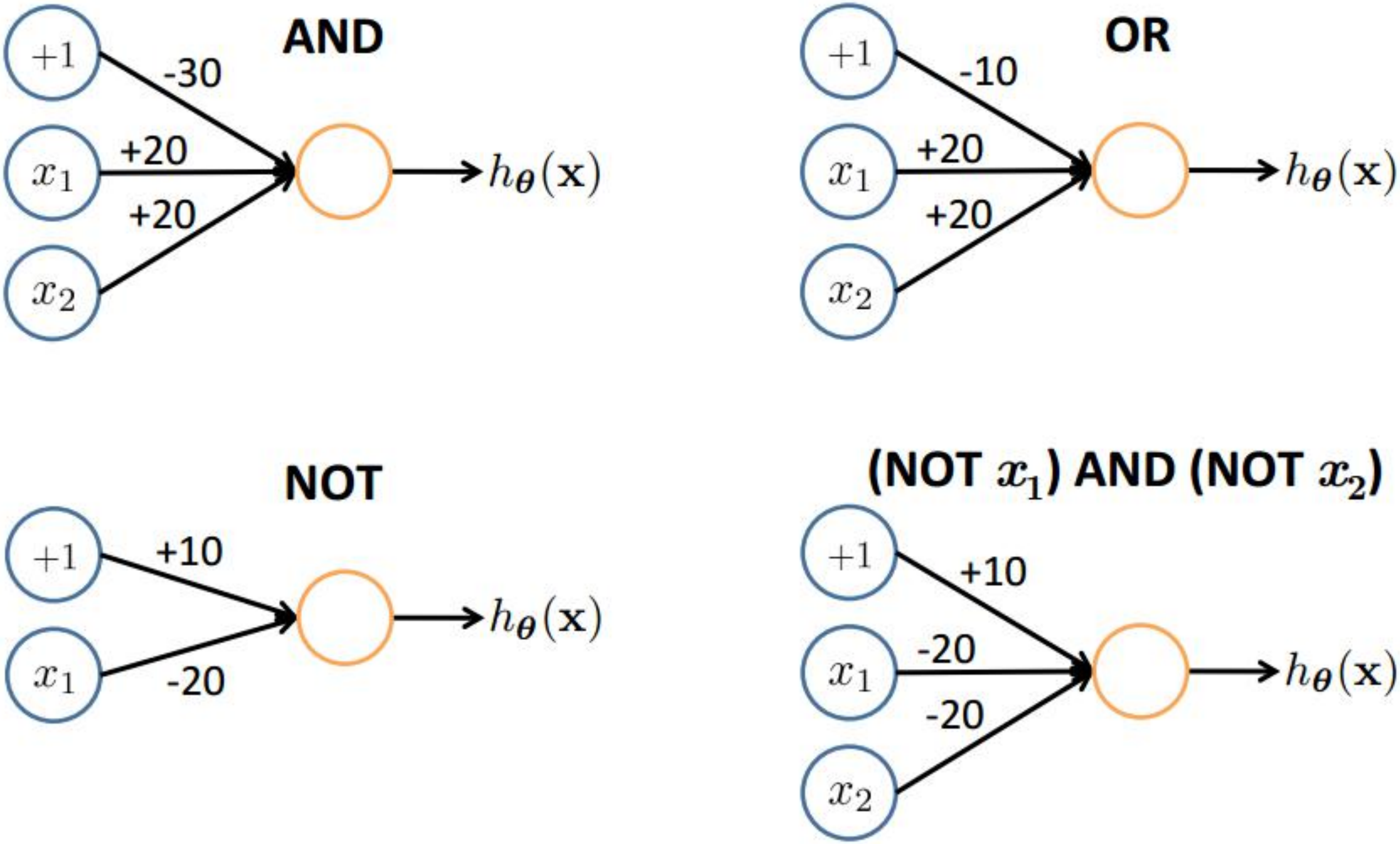


$$h_{\Theta}(\mathbf{x}) = g(-30 + 20x_1 + 20x_2)$$



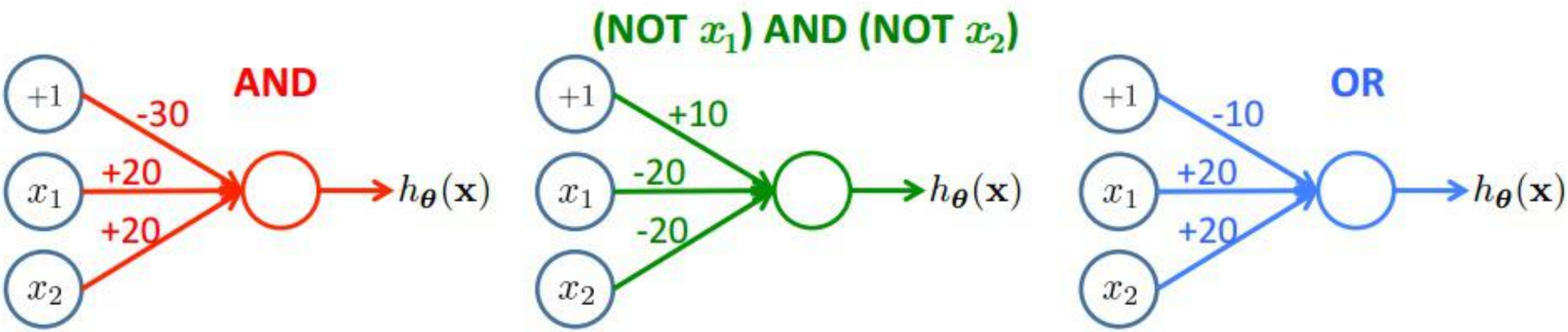
x_1	x_2	$h_{\Theta}(\mathbf{x})$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$


AND
$+1$
x_1
x_2
-30
+20
+20
$h_\theta(\mathbf{x})$
OR
$+1$
x_1
x_2
-10
+20
+20
$h_\theta(\mathbf{x})$
NOT
$+1$
x_1
+10
-20
$h_\theta(\mathbf{x})$
(NOT x_1) AND (NOT x_2)
$+1$
x_1
x_2
+10
-20
-20
$h_\theta(\mathbf{x})$

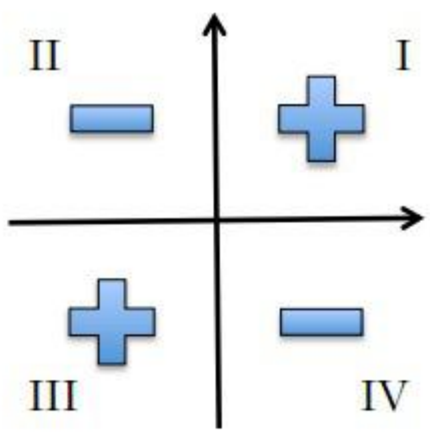


AND
+1
-30
x_1
+20
+20
x_2
$h_\theta(\mathbf{x})$
OR
+1
-10
x_1
+20
+20
x_2
$h_\theta(\mathbf{x})$
NOT
+1
+10
x_1
-20
$h_\theta(\mathbf{x})$
(NOT x_1) AND (NOT x_2)
+1
+10
x_1
-20
-20
x_2
$h_\theta(\mathbf{x})$

Combining Representations to Create Non-Linear Functions

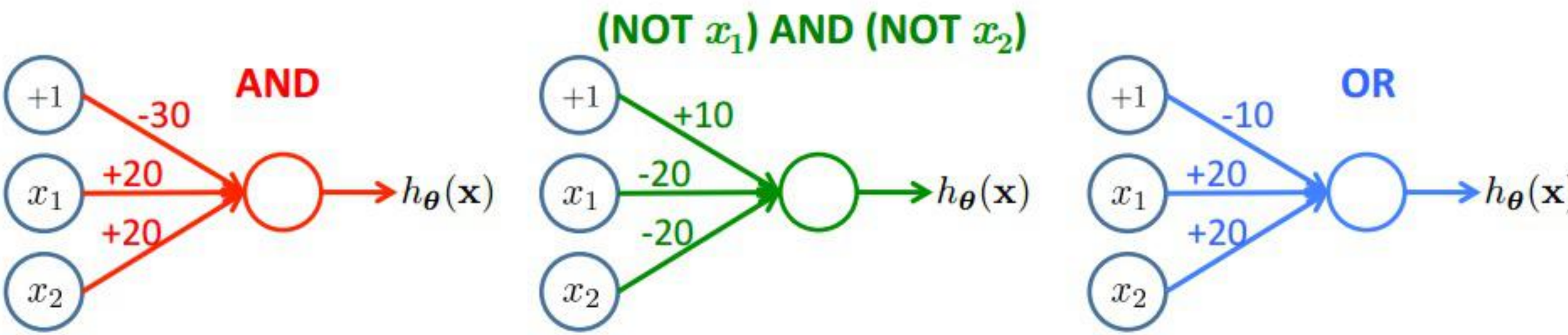


XNOR

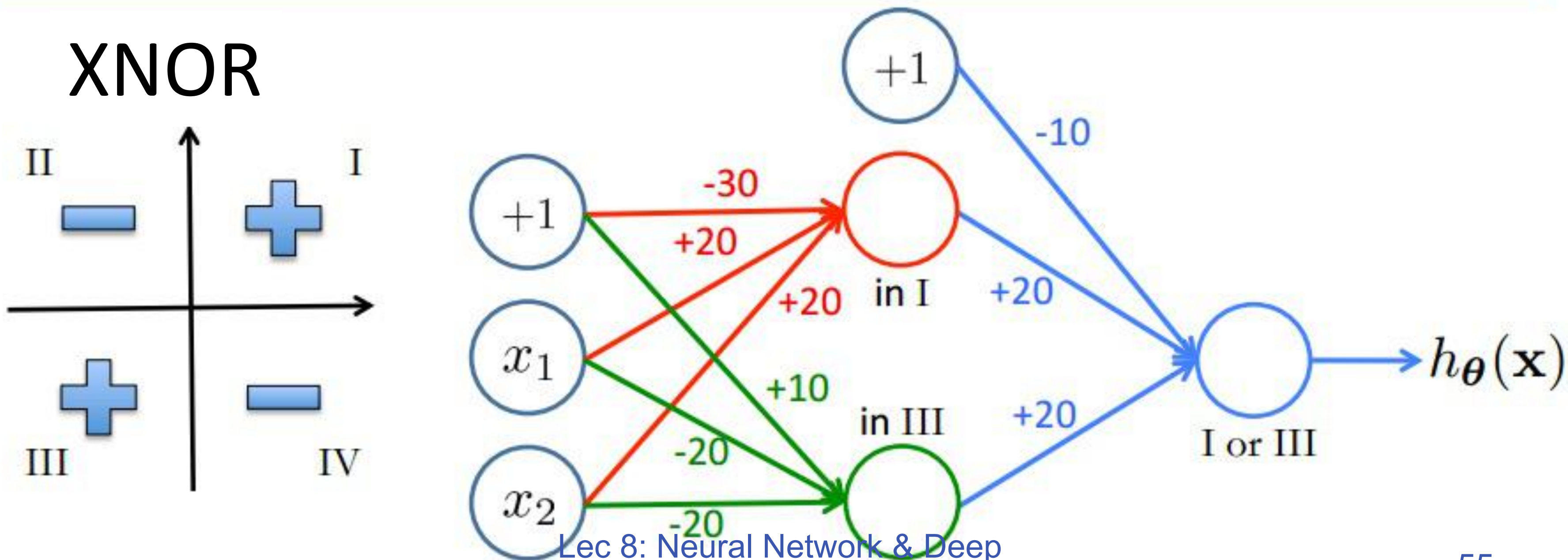


XNOR

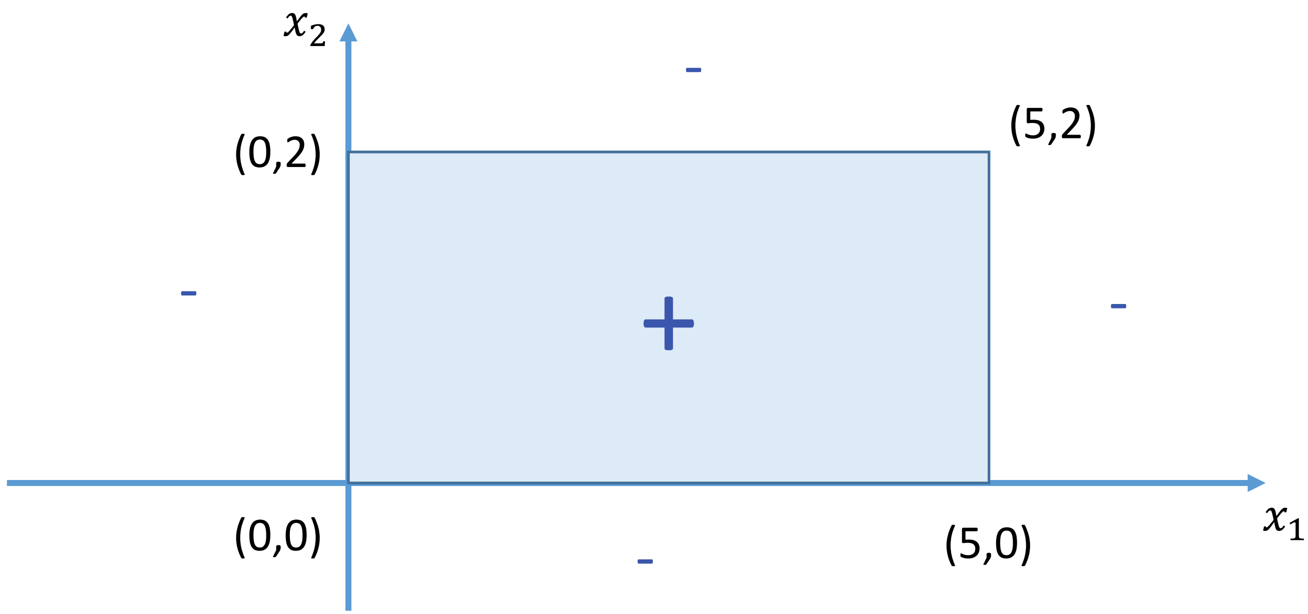
Combining Representations to Create Non-Linear Functions



XNOR



Exercise



Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

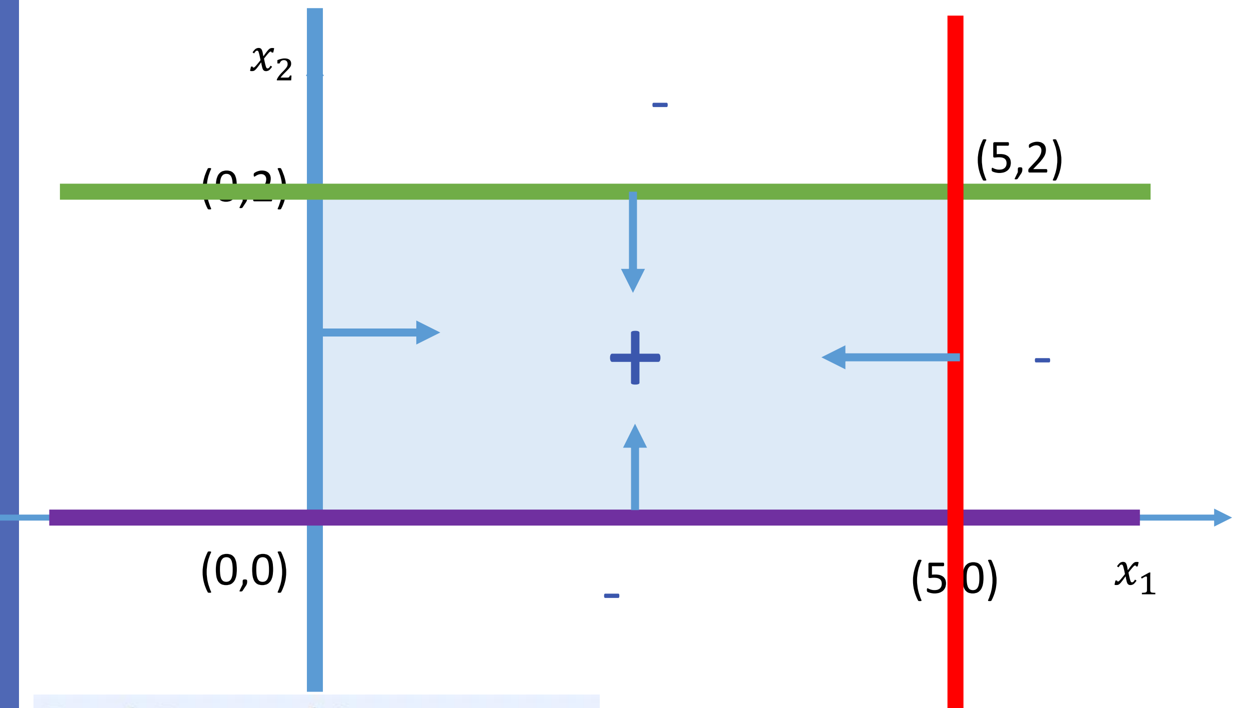
If all the samples inside the rectangle are positive; otherwise are negative

Show a feedforward NN can classify all the samples correctly

For simplicity, we assume g(z) is a step function.

What are $\Theta^{(1)}$ and $\Theta^{(2)}$

Exercise



$$\begin{cases} x_1 > 0 \\ 5 - x_1 > 0 \\ x_2 > 0 \\ 2 - x_2 > 0 \end{cases}$$

$$\Rightarrow \Theta^{(1)} = \begin{array}{c} \begin{bmatrix} 0 & 1 & 0 \\ 5 & -1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & -1 \end{bmatrix} \\ \Theta_0^{(1)} \ \Theta_1^{(1)} \ \Theta_2^{(1)} \end{array}$$

$$-3.5 + a_1 + a_2 + a_3 + a_4 > 0$$

$$\Theta^{(2)} = \begin{array}{c} [-3.5 \quad 1 \quad 1 \quad 1 \quad 1] \\ \Theta_0^{(2)} \ \Theta_1^{(2)} \ \Theta_2^{(2)} \Theta_3^{(2)} \ \Theta_4^{(2)} \end{array}$$

Feed-Forward Steps:

$$\mathbf{z}^{(2)} = \Theta^{(1)}\mathbf{x}$$

$$\mathbf{a}^{(2)} = g(\mathbf{z}^{(2)})$$

$$\text{Add } a_0^{(2)} = 1$$

$$\mathbf{z}^{(3)} = \Theta^{(2)}\mathbf{a}^{(2)}$$

$$h_\Theta(\mathbf{x}) = \mathbf{a}^{(3)} = g(\mathbf{z}^{(3)})$$

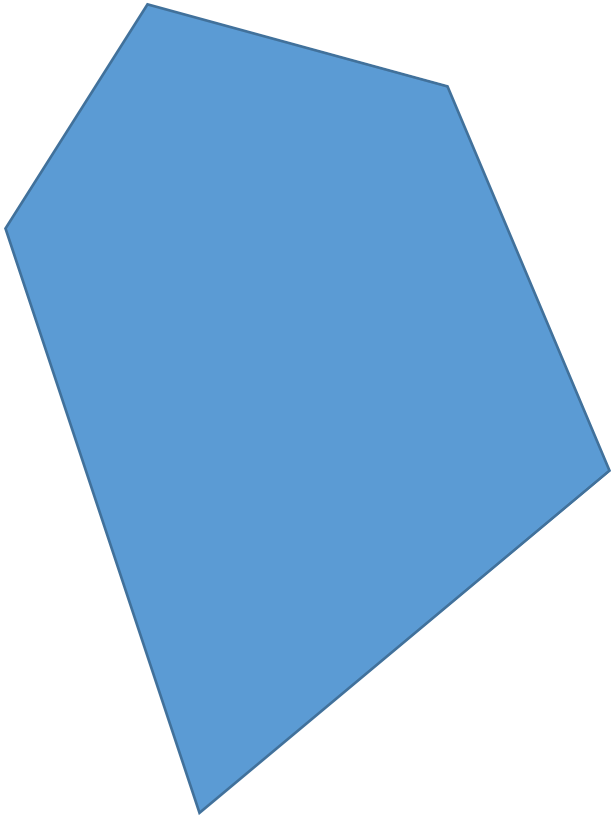
If all the samples inside the rectangle are positive;
otherwise are negative

Show a feedforward NN can classify all the samples correctly

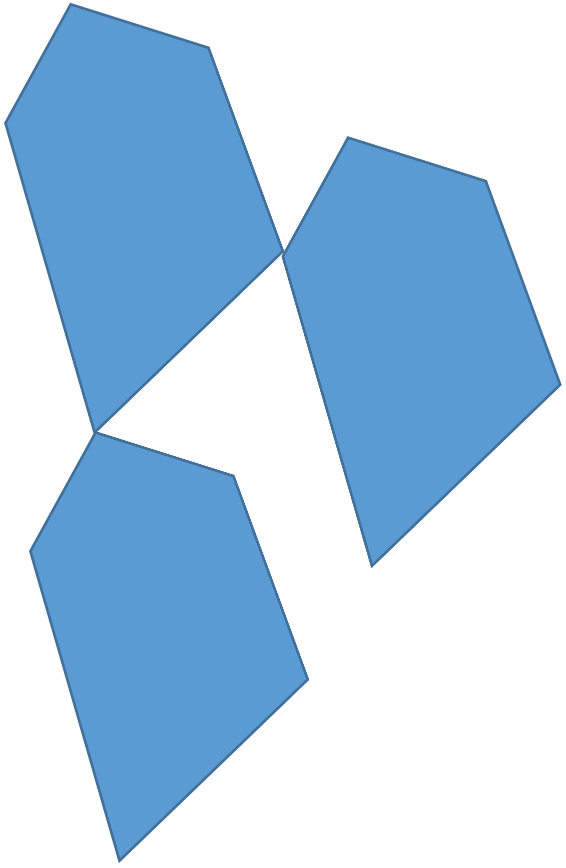
For simplicity, we assume g(z) is a step function.

What are $\Theta^{(1)}$ and $\Theta^{(2)}$

Arbitrary Decision Boundary



Arbitrary Decision Boundary



Neural Network Training Animation

- https://playground.tensorflow.org/