CS 466/566
Introduction to Deep Learning

Lecture 2

Introduction to Machine Learning - Part 2

Composed from various sources such as Andrew Ng, Quoc Le, etc.

Logistic Regression (Binary Classification)

- In linear regression we tried to predict the value of $y^{(i)}$ for the ith example $x^{(i)}$ using a linear function $y = h_{\theta,b}(x) = \theta^T x + b$
- This is clearly not a great solution for predicting binary-valued labels such as $y^{(i)} \in \{0, 1\}$.
- In logistic regression we use a different hypothesis class to try to predict the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class.

Logistic Regression

- Specifically, we will try to learn a function of the form:

$$P(y=1|x) = h_{\theta,b}(x) = \frac{1}{1+\exp(-\theta^T x - b)} \equiv \sigma(\theta^T x + b)$$

$$P(y=0|x) = 1 - P(y=1|x) = 1 - h_{\theta,b}(x)$$

- $\sigma(z) = \frac{1}{1+e^{-z}}$ is called the “sigmoid” or “logistic” function.
- it is an S-shaped function that “squashes” the value of $\theta^T x + b$ into the range [0, 1] so that we may interpret $h_{\theta,b}(x)$ as a probability.

Logistic Regression

- Our goal is to search for a value of θ so that the probability $P(y = 1|x) = h_{\theta,b}(x)$ is large when **x** belongs to the “1” class and small when x belongs to the “0” class.
- For a set of training examples with binary labels $\{x^{(i)}, y^{(i)}: i = 1, \dots, m\}$ the following cost function measures how well a given $h_{\theta,b}$ does this:

$$J(\theta, b) = -\sum_i \Big(y^{(i)} \log\Big(h_{\theta,b}(\, x^{(i)})\Big) + (1 - y^{(i)}) \log\Big(1 - h_{\theta,b}(\, x^{(i)})\Big)\Big)$$

Logistic Regression

$$J(\theta, b) = -\sum_i \left[y^{(i)} \log\left(h_{\theta,b}(\, x^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - h_{\theta,b}(\, x^{(i)})\right)\right]$$

- only one of the two terms in the summation is non-zero for each training example.
- we now have a cost function that measures how well a given hypothesis h_θ fits our training data.
- we can learn to classify our training data by minimizing $J(\theta, b)$ to find the best choice of θ, b.
- we can classify a new test point as “1” or “0” by checking which of these two class labels is most probable.

A case study of movie recommendations

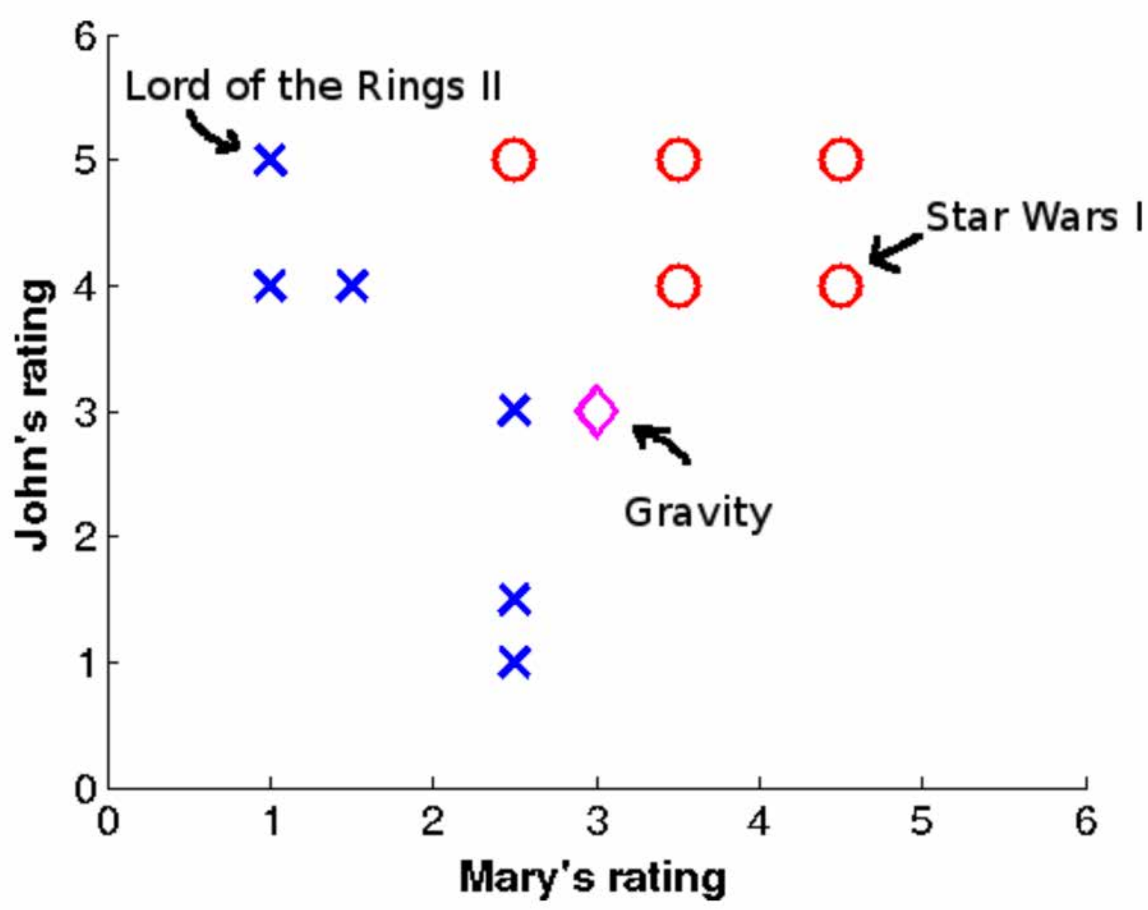
- Should I watch Gravity or not?
- We ask our close friends, let’s say, Mary and John. They watched it.
- In a scale of 1 to 5, they both rate it as a 3.
- 3 meaning that “not outstanding but worth watching”.
- How can I decide if I should go to Gravity or not?
- I need more data!

More movie data from friends

Movie Name	Mary’s Rating	John’s Rating	Do I like it?
Lord of the Rings II	1	5	No
…	…	…	…
Star Wars I	4.5	4	Yes
Gravity	3	3	?

- What am I still missing?
- I need to somehow bind these ratings to my taste.
- Therefore, I am going to label this data as I like it or not.

Visualized movie data



The question is:
"Am I going to like Gravity?"

Write a computer program for predicting it

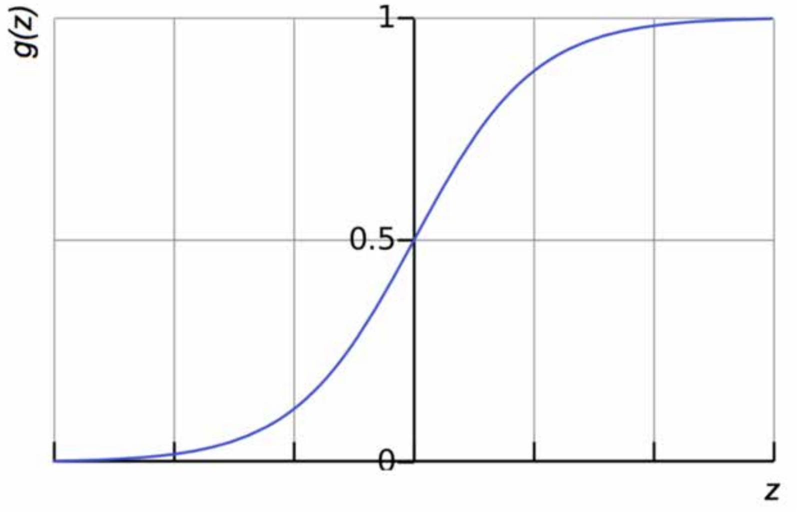
- Labels: "I like it" -> 1 "I don't like it" -> 0
- Inputs: Mary's rating, John's rating
- A decision function can be as simple as weighted linear combination of my friends:

$$h_{\theta,b} = \theta_1 x_1 + \theta_2 x_2 + b$$

$$h_{\theta,b} = \theta^T x + b$$

- This function has a problem. Its values are unbounded. We want its output to be in the range of 0 and 1.

Bound its values between 0 and 1



- Below function is unbounded:

$$h_{\theta,b} = \theta^T x + b$$

- We are going to bound its output:

$$h_{\theta,b} = g(\theta^T x + b),$$

where $g(z)$ is sigmoid function.

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

Using past data to learn the decision function

- We will use the past data to learn θ, b to approximate y. In particular, we want to obtain θ, b such that:

$h_{\theta,b}\left(x^{(1)}\right) \approx y^{(1)}$ where $x^{(1)}$ is my friend's ratings for 1st movie.

$h_{\theta,b}\left(x^{(2)}\right) \approx y^{(2)}$ where $x^{(2)}$ is my friend's ratings for 2nd movie.

…

$h_{\theta,b}\left(x^{(m)}\right) \approx y^{(m)}$ where $x^{(m)}$ is my friend's ratings for mth movie.

Using past data to learn the decision function

To find values of θ and b we can minimize the following ***cost function***:

$$J(\theta, b) = \left(h_{\theta,b}\left(x^{(1)}\right) - y^{(1)}\right)^2 + \left(h_{\theta,b}\left(x^{(2)}\right) - y^{(2)}\right)^2 + \ \ldots + \left(h_{\theta,b}\left(x^{(m)}\right) - y^{(m)}\right)^2$$

$$J(\theta, b) = \sum_{i=1}^{m} \left(h_{\theta,b}\left(x^{(i)}\right) - y^{(i)}\right)^2$$

Use Stochastic Gradient Descent (SGD):

$$\theta_1 = \theta_1 - \alpha \Delta \theta_1$$

$$\theta_2 = \theta_2 - \alpha \Delta \theta_2$$

$$b = b - \alpha \Delta b$$

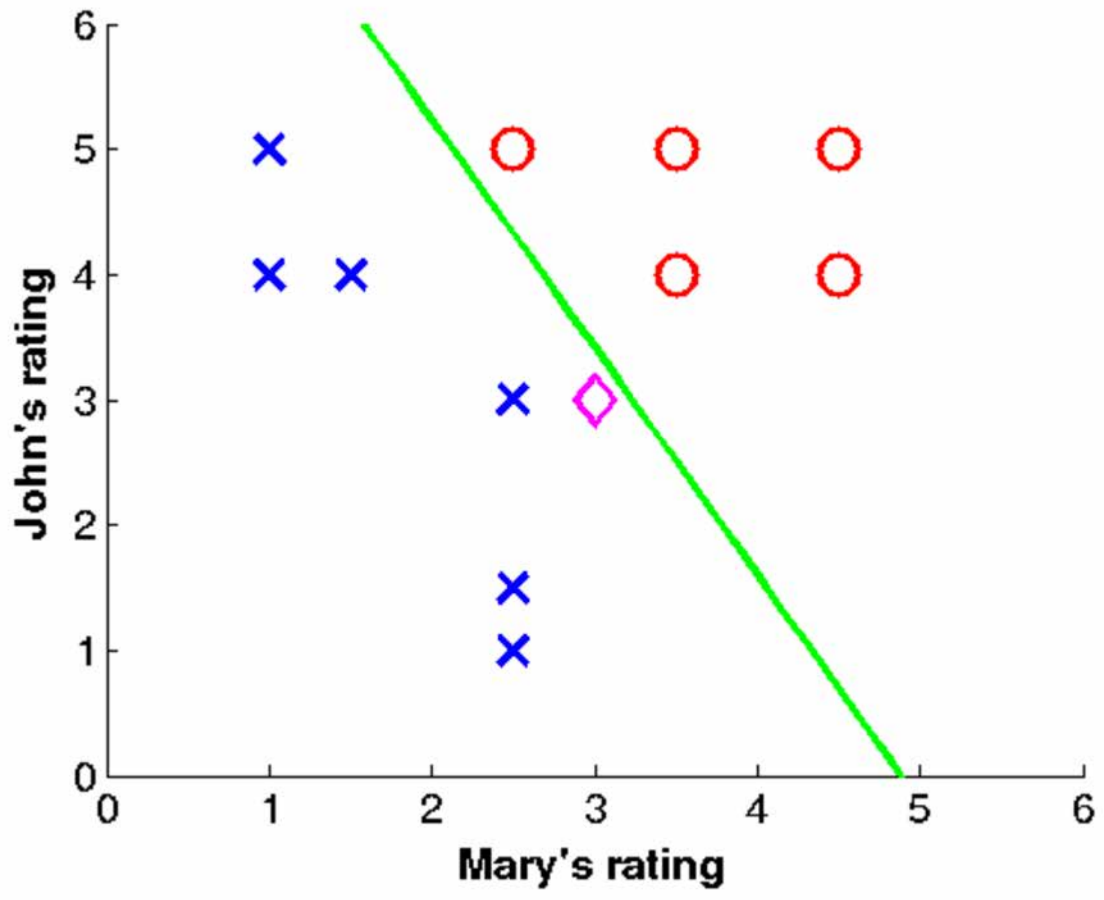
Apply our magic Stochastic Gradient Descent

1. Initialize the parameters θ and b at random
2. Pick a random example $\{x^{(i)}, y^{(i)}\}$
3. Compute the partial derivatives of θ_1, θ_2, b
4. Update parameters using:

$$\theta_1 = \theta_1 - \alpha\Delta\theta_1$$
$$\theta_2 = \theta_2 - \alpha\Delta\theta_2$$
$$b = b - \alpha\Delta b$$

Stop it when parameters don't change much, or after a certain number of iterations.

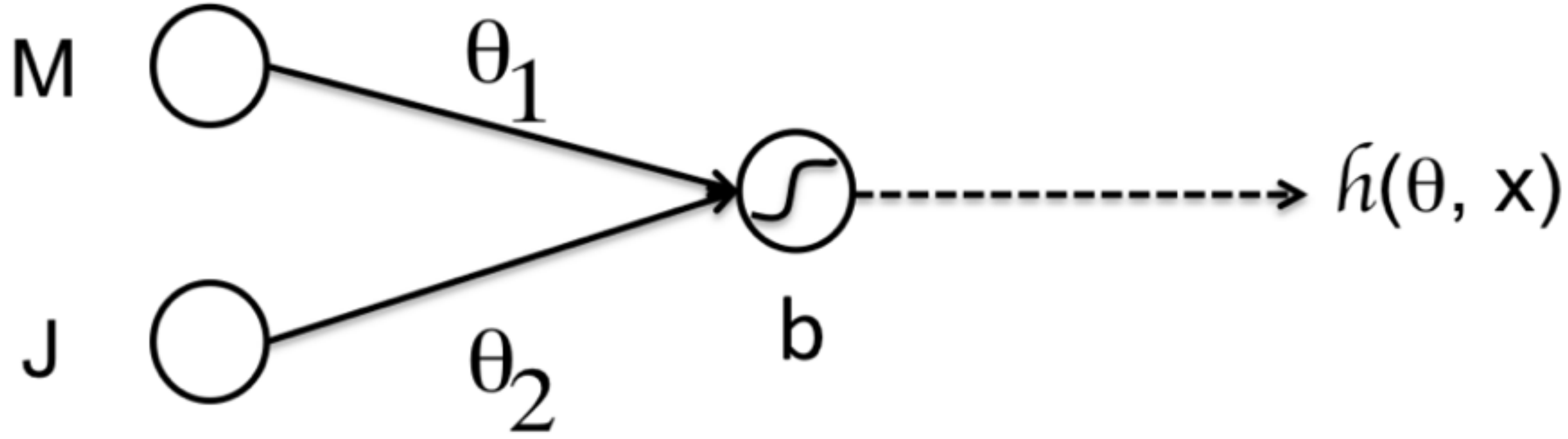
Here is my decision boundary



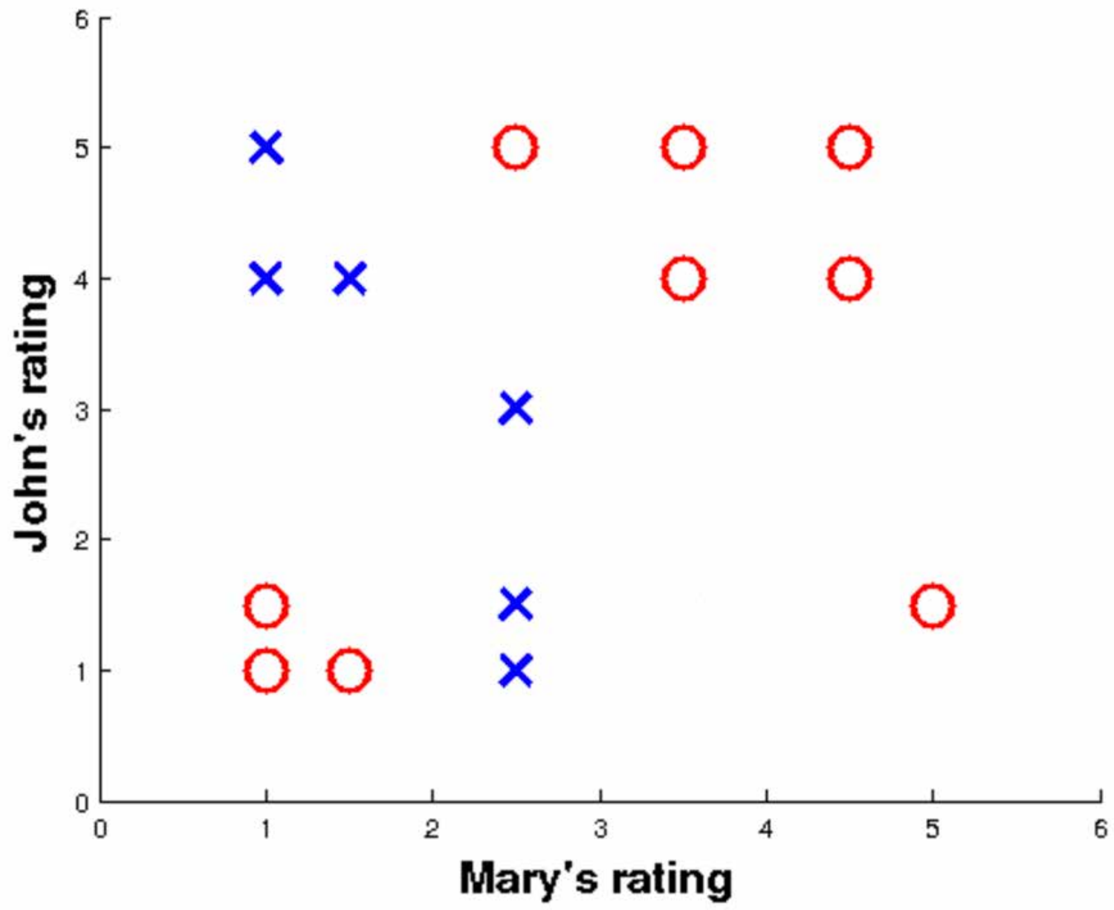
Gravity movie is slightly on the "don't watch" side.

With this data set, it seems like "not watching it" makes more sense.

Another way of representing our model



Were we lucky? Let’s plot Susan’s movies.

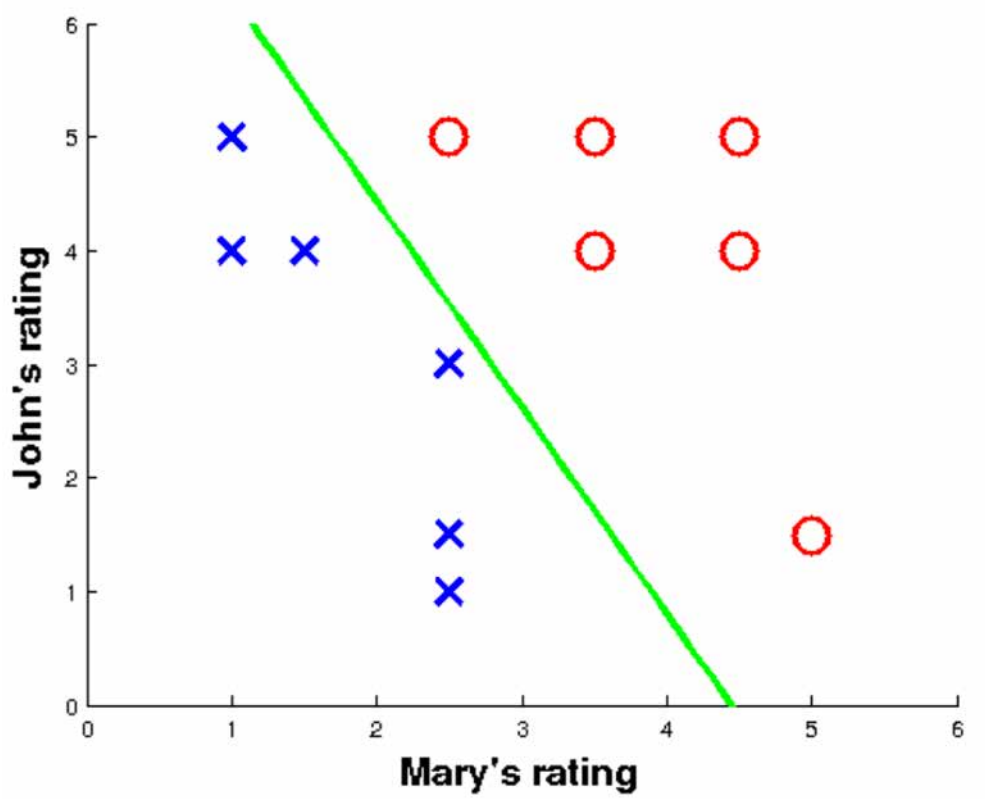


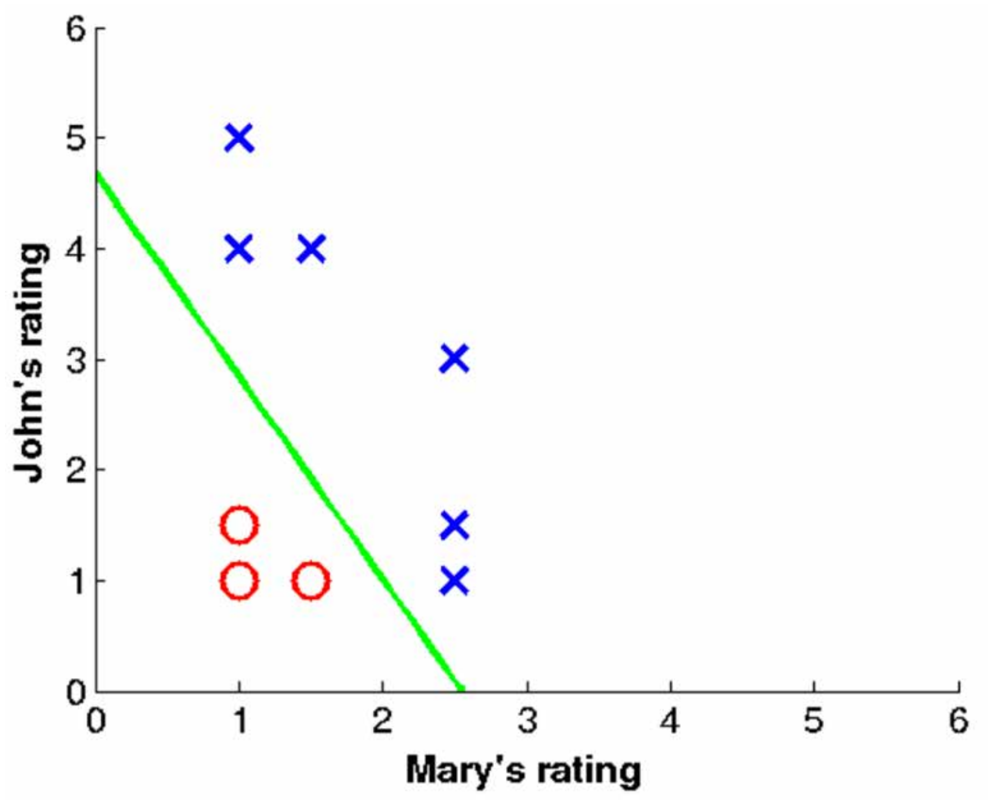
Susan likes some of the movies both Mary and John rated poorly.

How can I have a linear decision boundary separate these?

Maybe we should split the problem into two.

Divide and conquer: We have 2 decision functions.

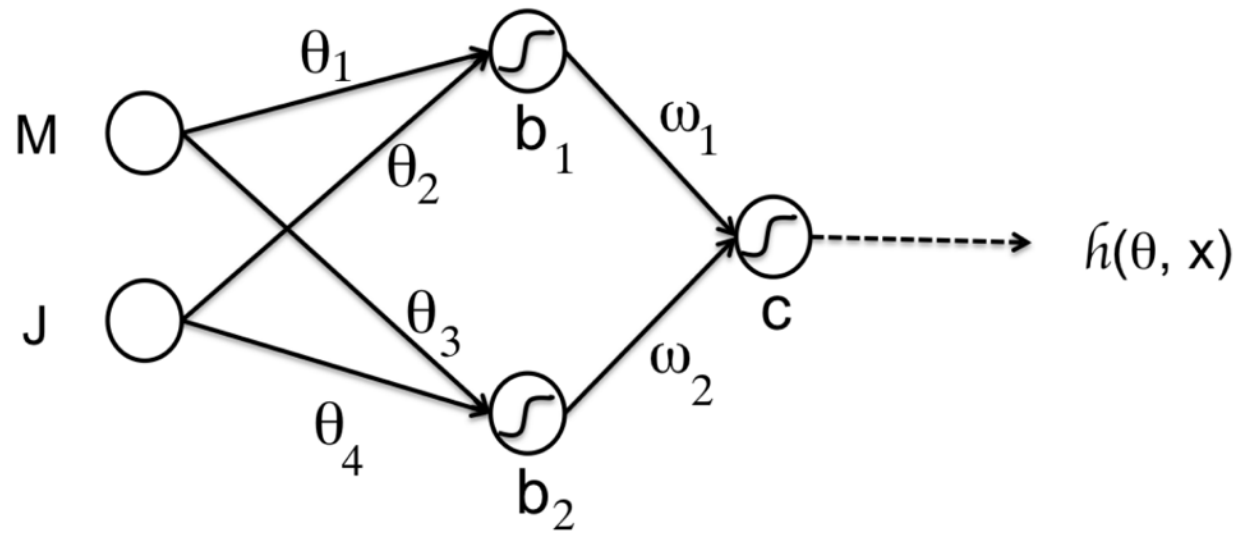




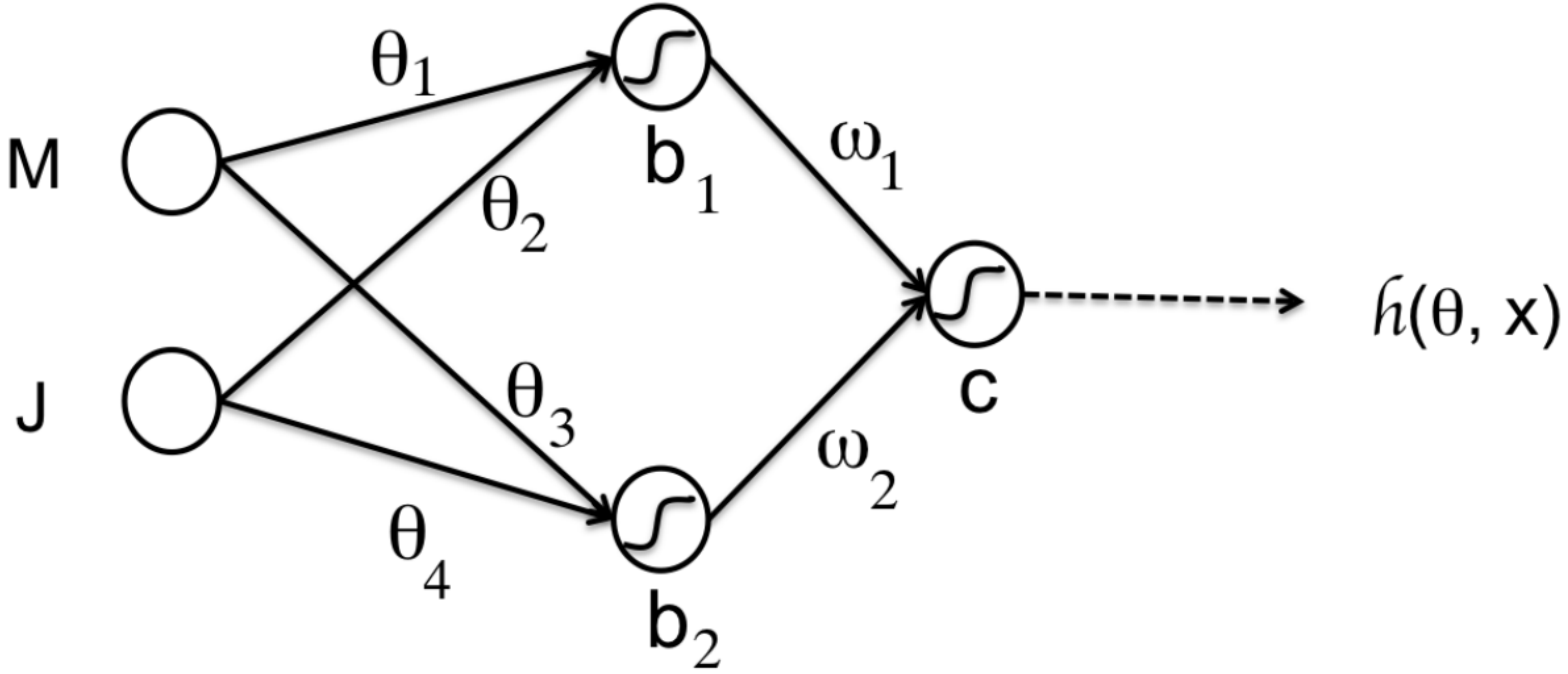
A decision function of decision functions

Movie Name	Output by decision function h_1	Output by decision function h_2	Does Susan like it?
Lord of the Rings II	$h_1(x^{(1)})$	$h_2(x^{(2)})$	No
…	…	…	…
Star Wars I	$h_1(x^{(n)})$	$h_2(x^{(n)})$	Yes
Gravity	$h_1(x^{(n+1)})$	$h_2(x^{(n+1)})$	?

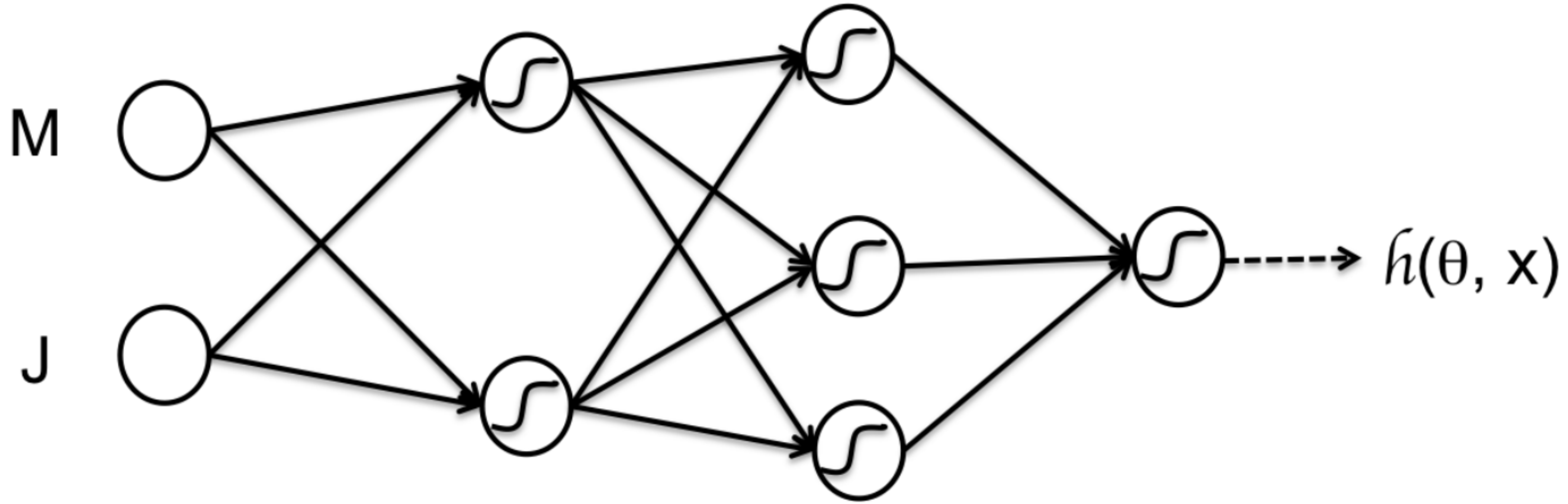
This problem is the same problem as Mary and John case.
Just consider Output by decision function h_1 and h_2 as two new friends.



A neural network it is!

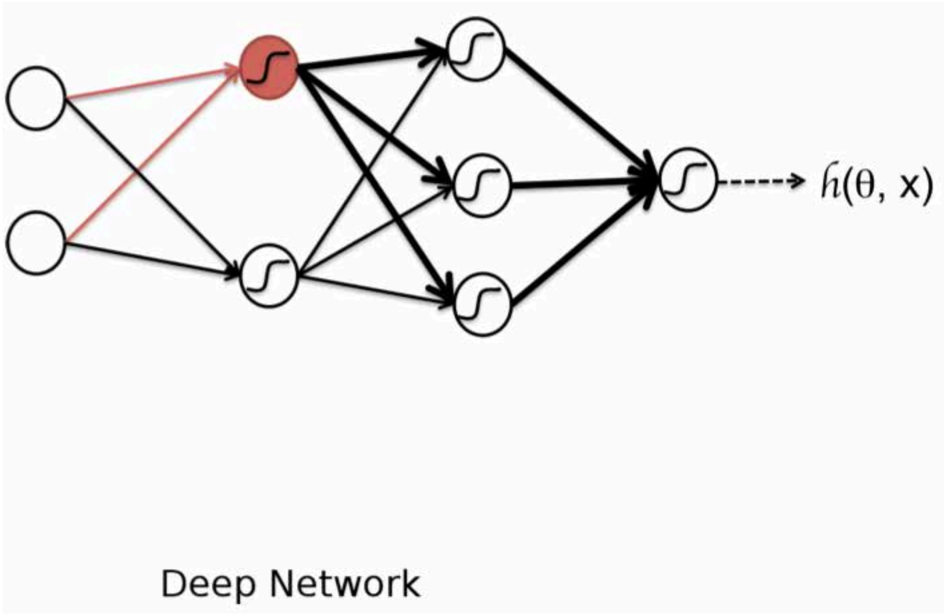


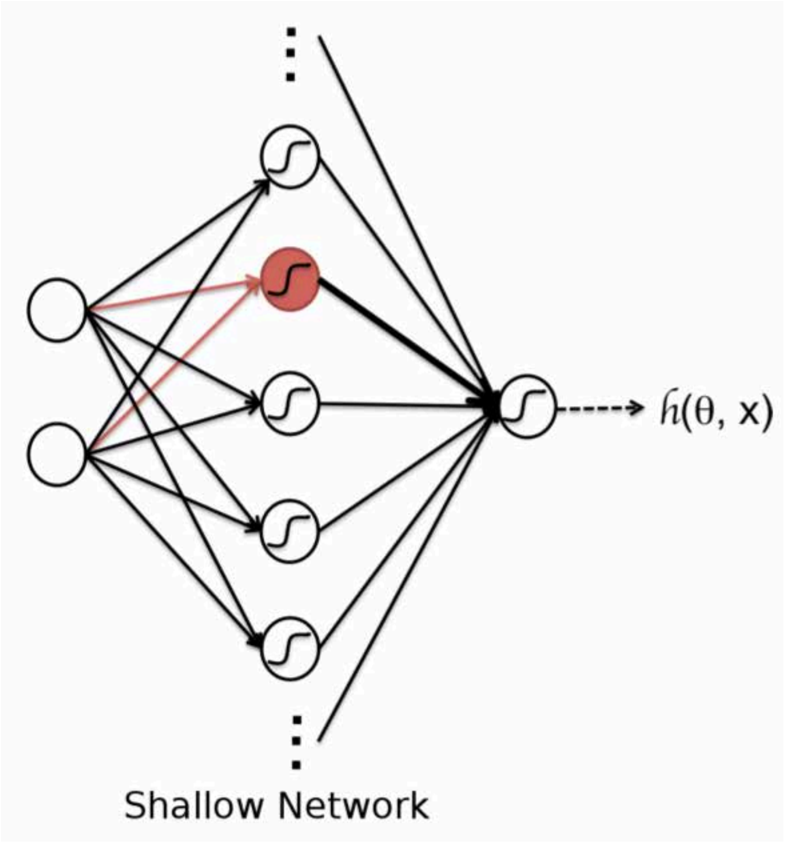
A deeper neural network



Deep vs Shallow Networks

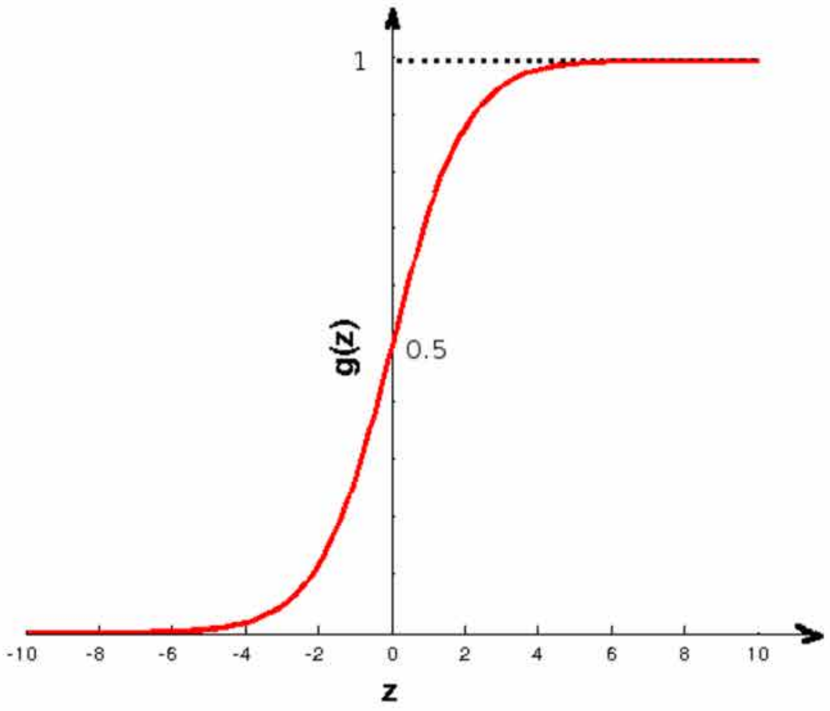
- When the problem does exhibit nonlinear properties, deep networks seem computationally more attractive than shallow networks.
- Bolded edges mean computation paths that need the red neuron to produce the final output.



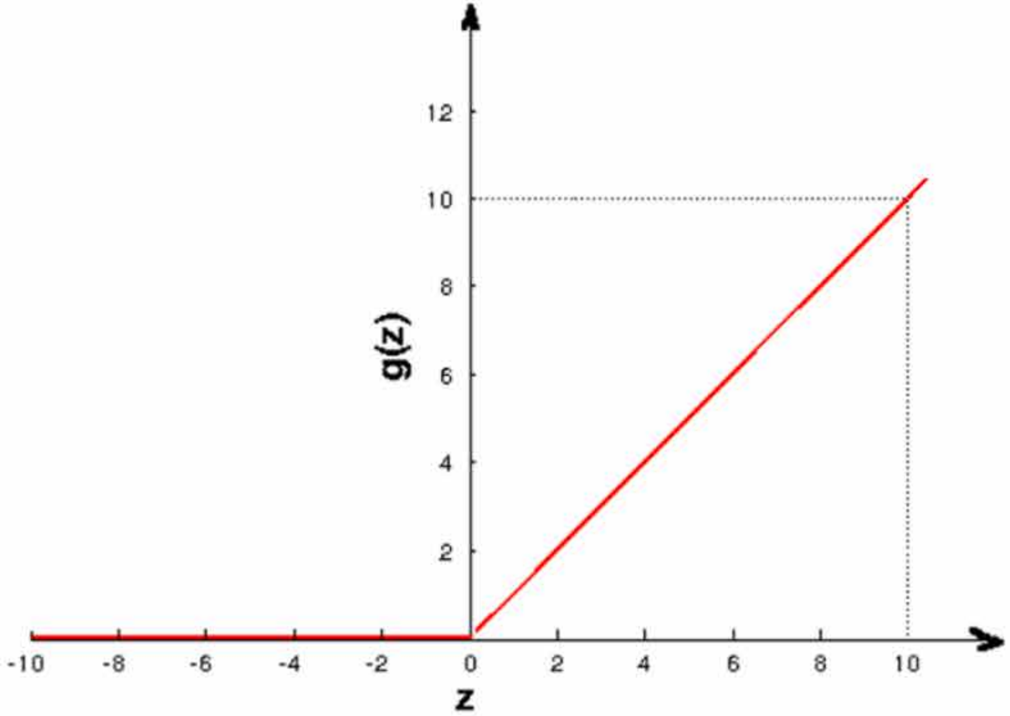


A better activation function: Rectified Linear Unit (ReLU)

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

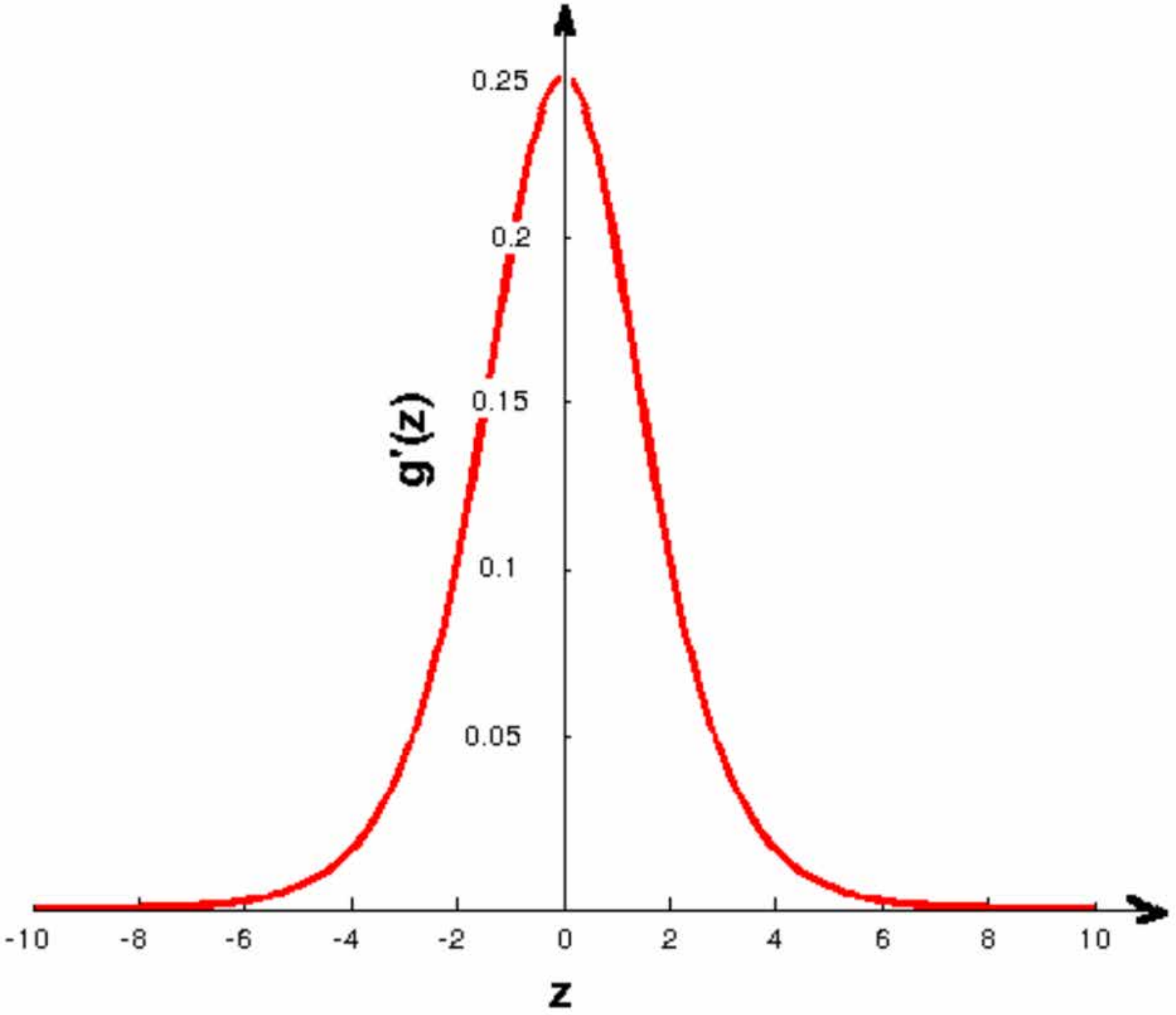


The sigmoid activation function

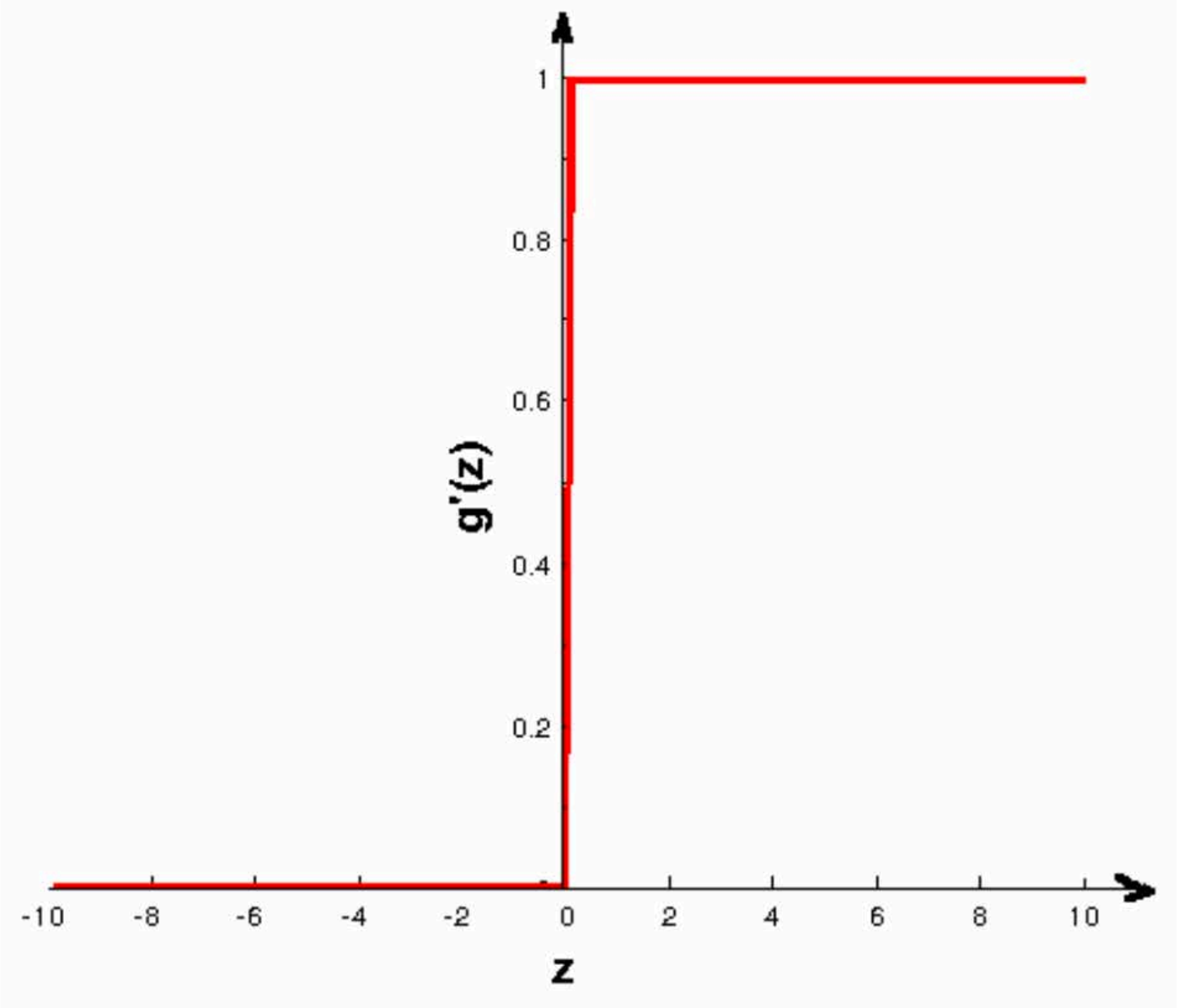


The rectified linear activation function

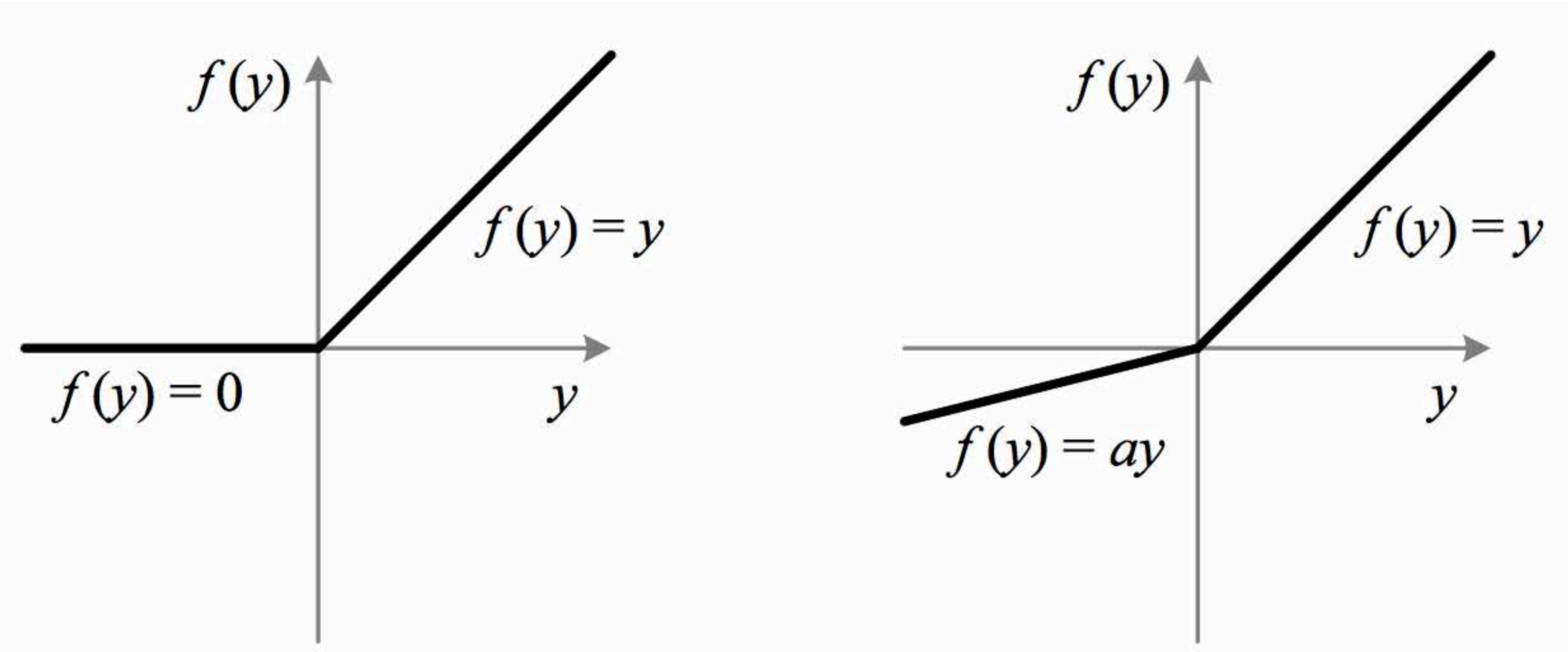
Why is ReLU better?
Consider derivative of sigmoid.



Derivative of ReLU



An even better activation function? Parametric Rectified Linear Unit (PReLU)



ReLU vs PReLU

Softmax Regression

- Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes.
- In logistic regression we assumed that the labels were binary:

$$y^{(i)} \in \{0, 1\}$$

- Softmax regression allows us to handle

$$y^{(i)} \in \{1, \dots, K\}$$

- where K is the number of classes. K

Softmax Regression

- Recall that we have a training set $\{(x^{(i)}, y^{(i)}), \ldots, (x^{(m)}, y^{(m)})\}$ of m labeled examples.
- Input features are $x^{(i)} \in \mathbb{R}^n$
- Output features are $y^{(i)} \in \{1, \ldots, K\}$
- For example, in the M-NIST digit recognition task, we would have K=10 different classes.
- Given a test input x, we want our hypothesis to estimate the probability that $P(y = k|x)$ for each value of k=1,…, K.

Softmax Regression

- Given a test input **x**, we want our hypothesis to estimate the probability that $P(y=k|x)$ for each value of k=1,..., K.
- we want to estimate the probability of the class label taking on each of the K different possible values.
- Thus, our hypothesis will output a K-dimensional vector (whose elements sum to 1) giving us our K estimated probabilities.

$$h_{\theta,b}(x) = \begin{bmatrix} P(y=1|x;\theta,b) \\ P(y=2|x;\theta,b) \\ \vdots \\ P(y=K|x;\theta,b) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)T}x + b^{(j)})} \begin{bmatrix} \exp\left(\theta^{(1)T}x + b^{(1)}\right) \\ \exp\left(\theta^{(2)T}x + b^{(2)}\right) \\ \vdots \\ \exp\left(\theta^{(K)T}x + b^{(K)}\right) \end{bmatrix}$$

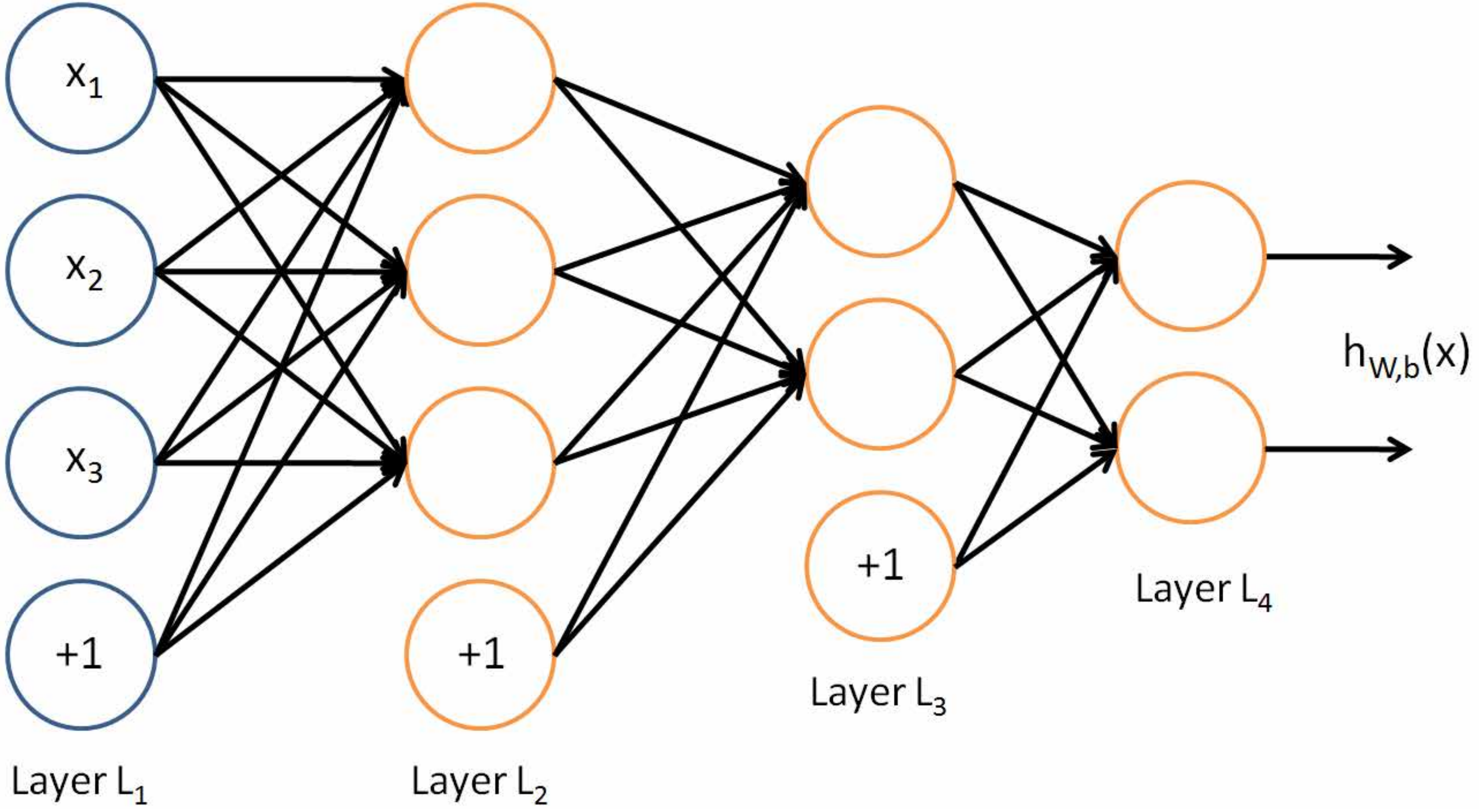
Softmax Regression

$$h_{\theta,b}(x) = \begin{bmatrix} P(y=1|x;\theta,b) \\ P(y=2|x;\theta,b) \\ \vdots \\ P(y=K|x;\theta,b) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\theta^{(j)T}x + b^{(j)})} \begin{bmatrix} \exp\left(\theta^{(1)T}x + b^{(1)}\right) \\ \exp\left(\theta^{(2)T}x + b^{(2)}\right) \\ \vdots \\ \exp\left(\theta^{(K)T}x + b^{(K)}\right) \end{bmatrix}$$

- Notice that $\frac{1}{\sum_{j=1}^{K} \exp\left(\theta^{(j)T}x\right)}$ is a normalization constant.

- Distribution sums up to 1. This makes it a probability distribution.

A better neural network model visualization



Softmax Layer Visualized