From Logistic Regression to Feed-Forward Neural Networks

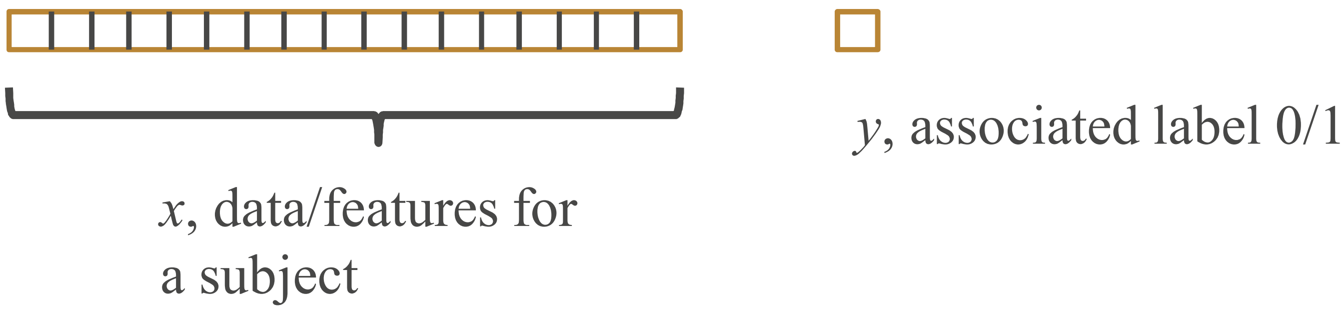
Vahid Tarokh
ECE685D, Fall 2025

Introduction

- We will next discuss logistic regression and the construction of neural Neural Networks.
- Important Note: Source of some of my slides (with great appreciation and acknowledgements)
    - Professor David Carlson Slides
    - Professor Alex Smola's slides (available online)
    - Professor Ruslan Salakhutdinov's slides (available online)
    - Professor Hugo Larochelle's class on Neural Networks

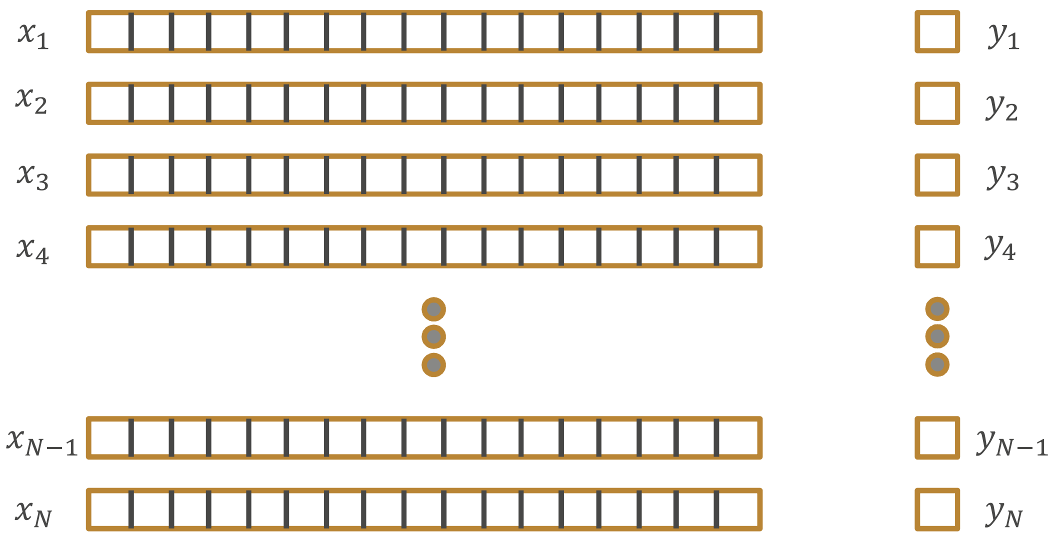
Logistic Regression

Learning a Predictive Model Based on Labeled Data



End goal: *predict* y from x

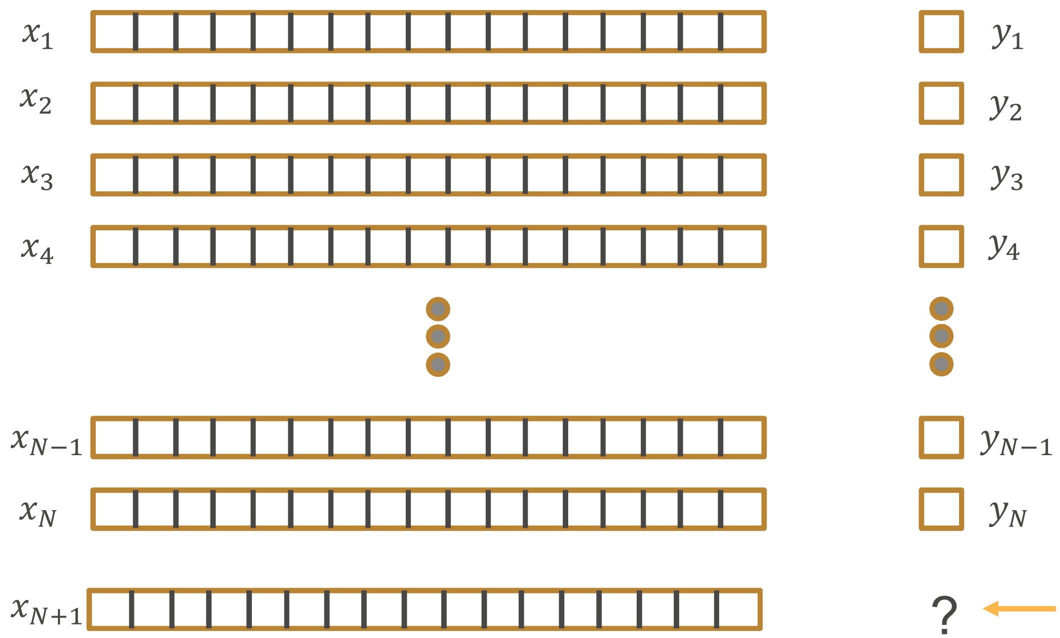
Training Set (Historical Data)



Start by limiting y to a binary outcome:

- False/True
- 0/1

Making Predictions



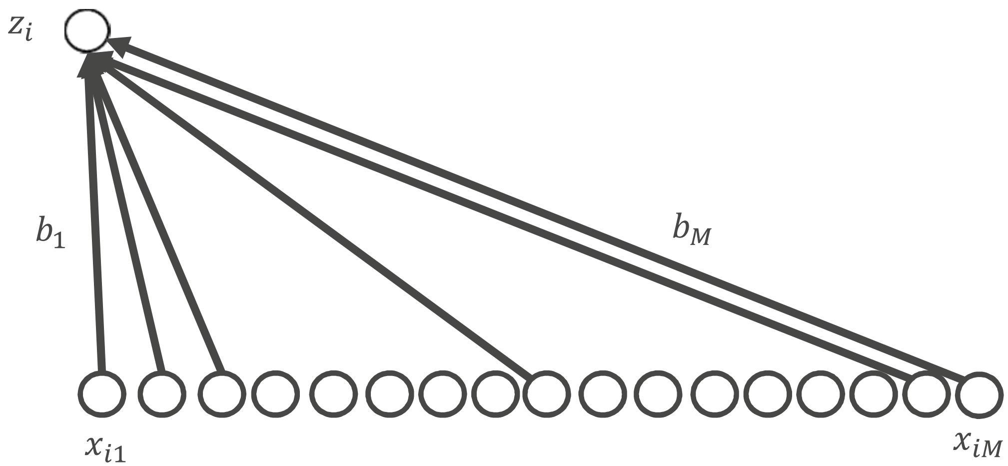
New Data

Start by limiting y to a binary outcome:

- False/True
- 0/1

Want to learn how to *predict* outcome

Linear Predictive Model



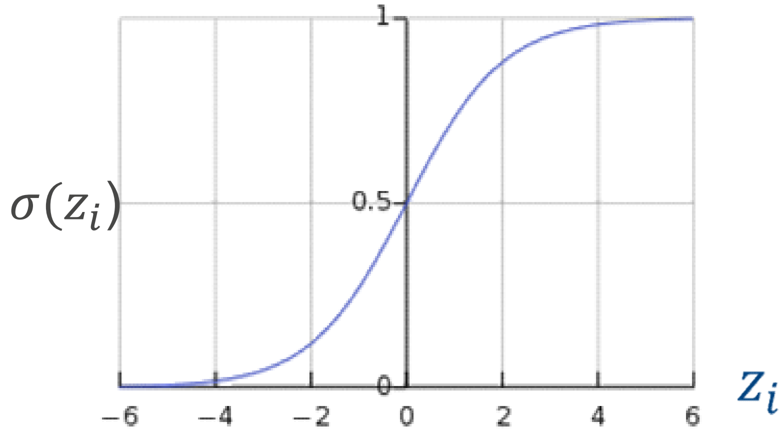
$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

Convert to a Probability

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Extra Constant

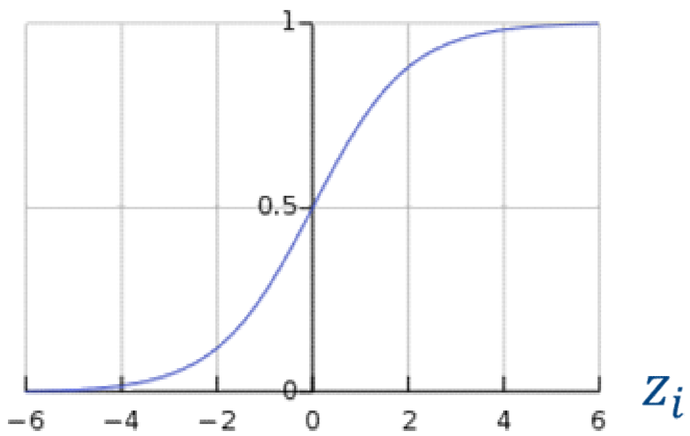
$$p(y_i = 1 | x_i) = \sigma(z_i)$$



Convert to a Probability

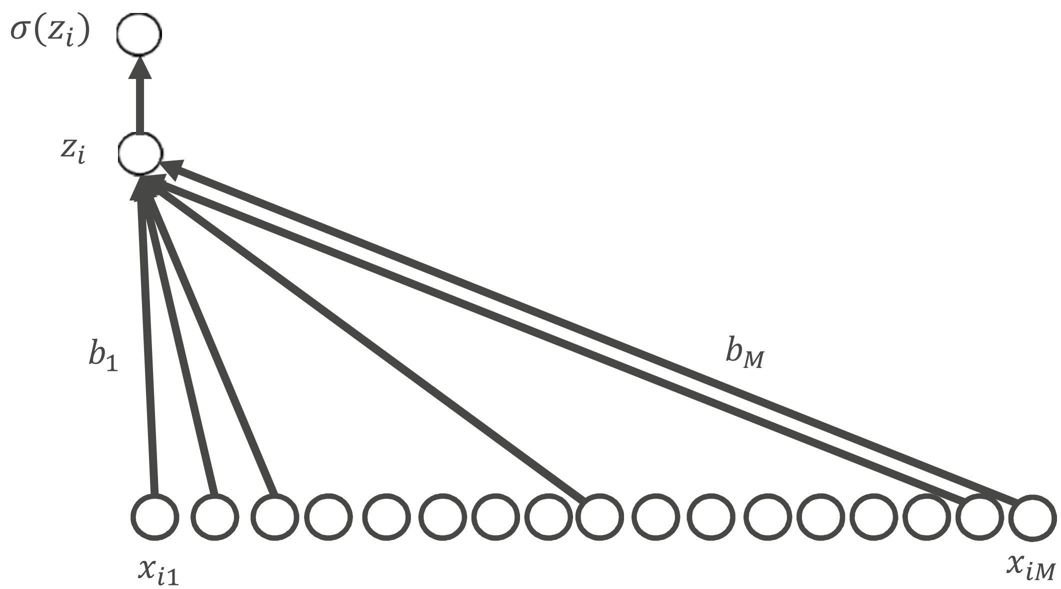
$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

$$p(y_i = 1|x_i) = \sigma(z_i) = \frac{\exp(z_i)}{1+\exp(z_i)} = \frac{1}{1+\exp(-z_i)}$$

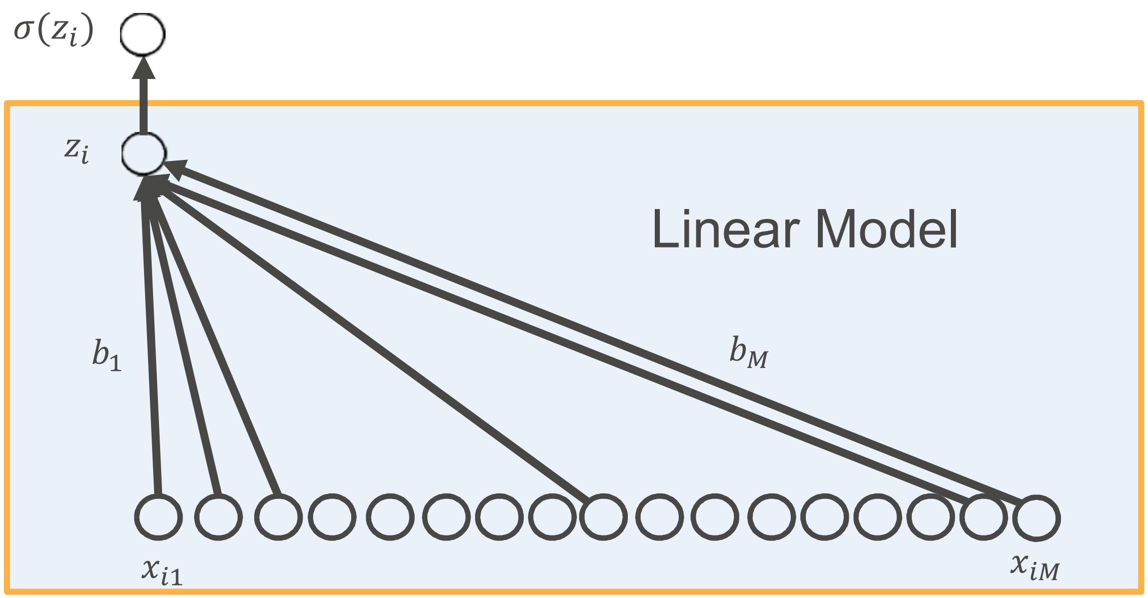


- Large and positive z_i indicates that event $y_i = 1$ is likely
- Large and negative z_i indicates that event $y_i = 0$ is likely

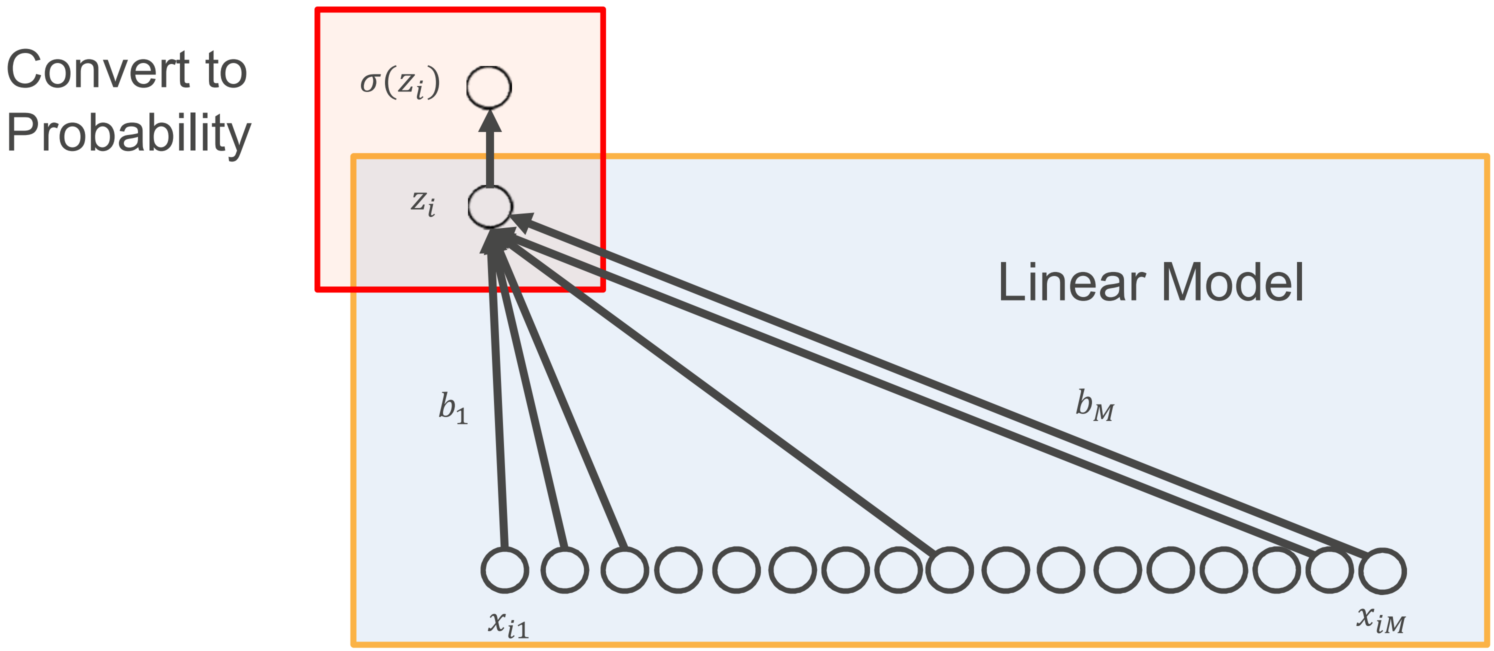
Logistic Regression



Logistic Regression



Logistic Regression



What do the parameters and model mean?

AN EXAMPLE

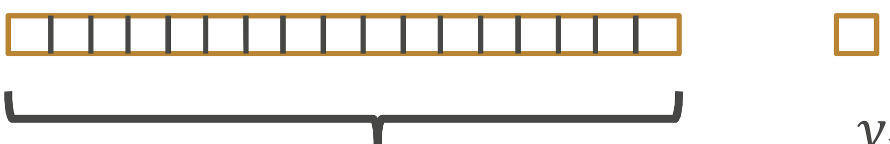
Example

Outcome:

- $y_i = 1$, it rains on day i;
- $y_i = 0$, it does not rain on day i

Features:

- On day i what is the $\{cloud\ cover, humidity, temperature, air\ pressure, \ldots\}$



Example

Outcome: $y_i = 1$, it rains on day i; $y_i = 0$, it does not rain on day i

Features: On day i what is the $\{1\text{: } cloud\ cover, 2\text{: } humidity, 3\text{: } temperature, \ldots\}$

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Cloud Cover Humidity

- If cloud cover is positively related to rainfall, b_1 should be positive

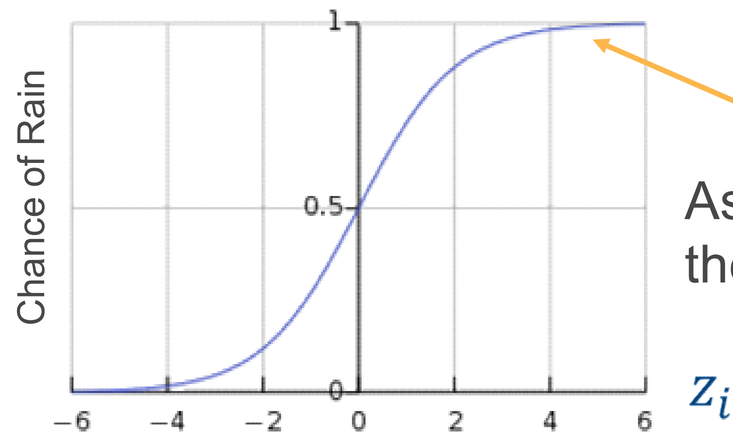
Impact on the Sigmoid Function

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM}) + b_0$$

Humidity

Cloud Cover

$$p(y_i = 1|x_i) = \sigma(z_i)$$



As the value z_i increases, the chance of rain increases

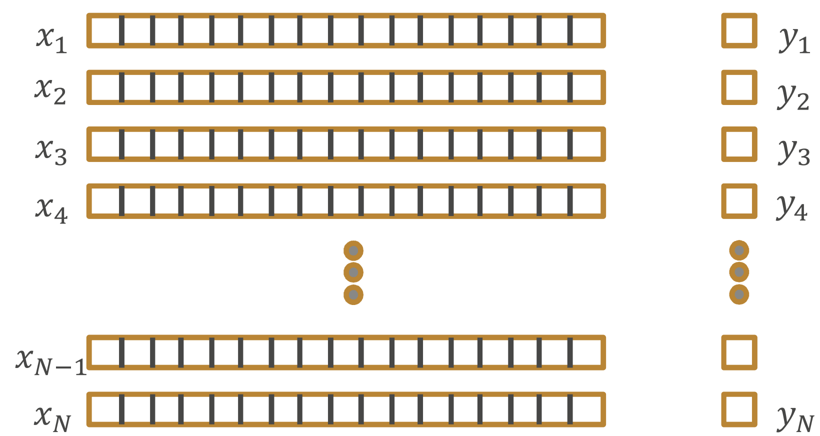
Building the Training Set

Need to learn the parameters

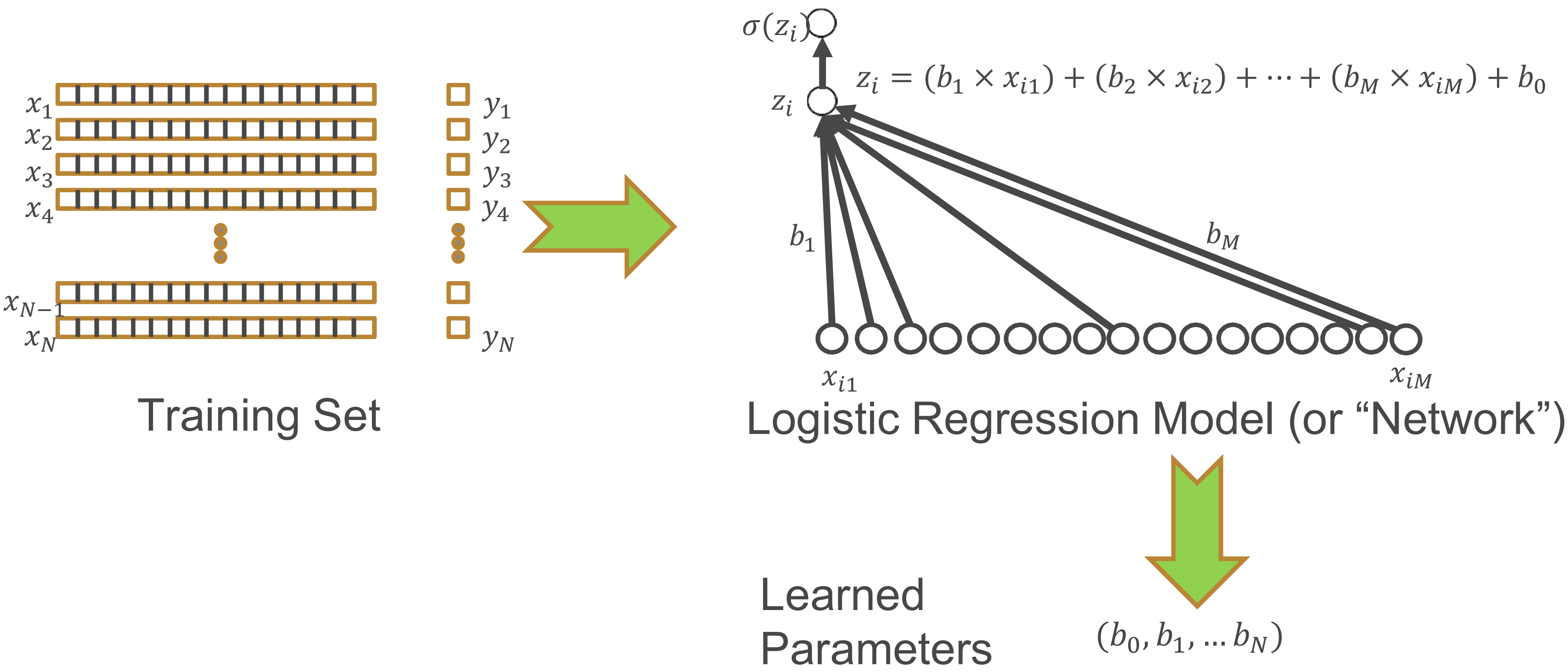
Requires *training data*

Record data from N days

- Capture features: $\{cloud\ cover, humidity, temperature, \ldots\}$
- Did it rain?

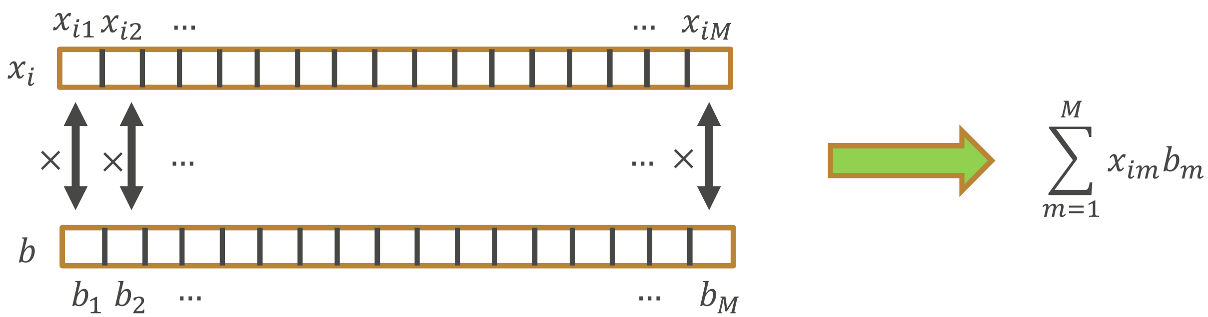


Learning Model Parameters



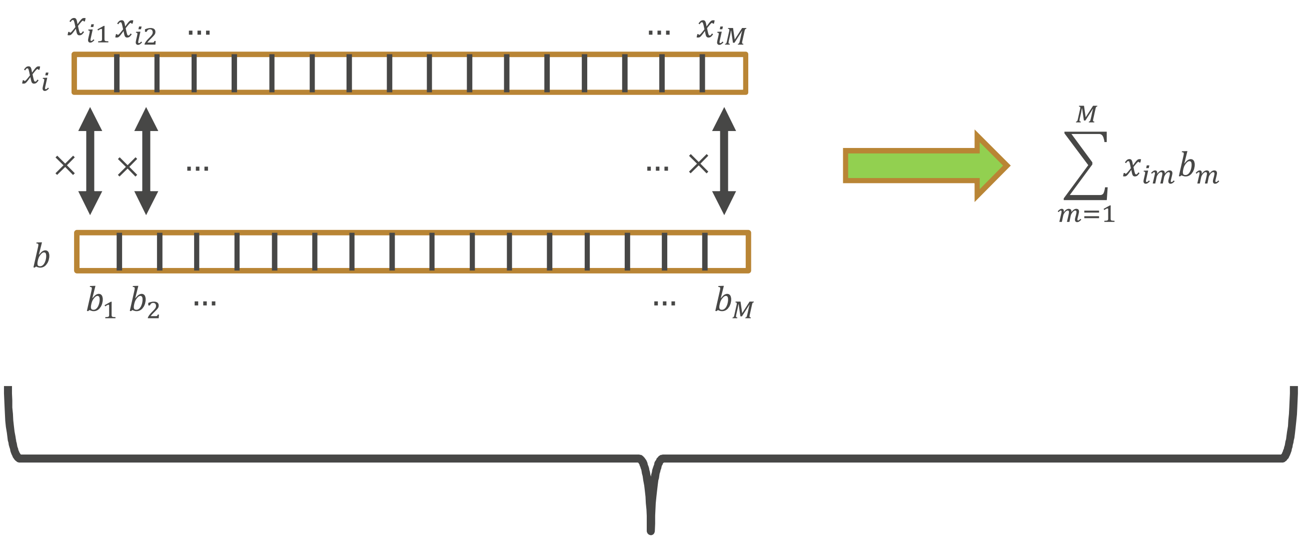
Interpretation of Logistic Regression

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$



Interpretation of Logistic Regression

$$z_i = (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

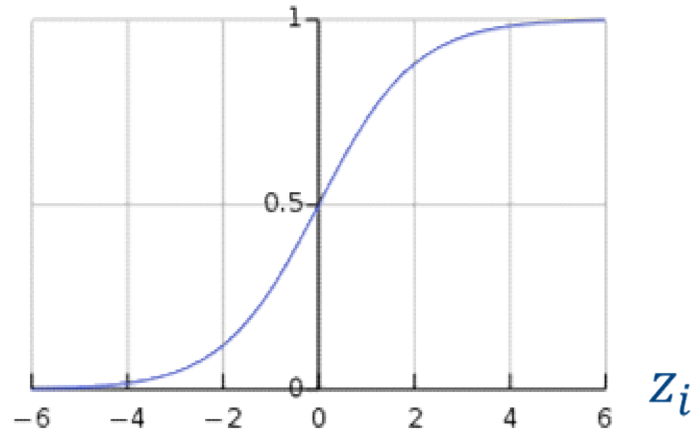


Compact Notation: $x_i \cdot b$ (or "inner product")

Interpretation of Logistic Regression

$$z_i = b_0 + (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

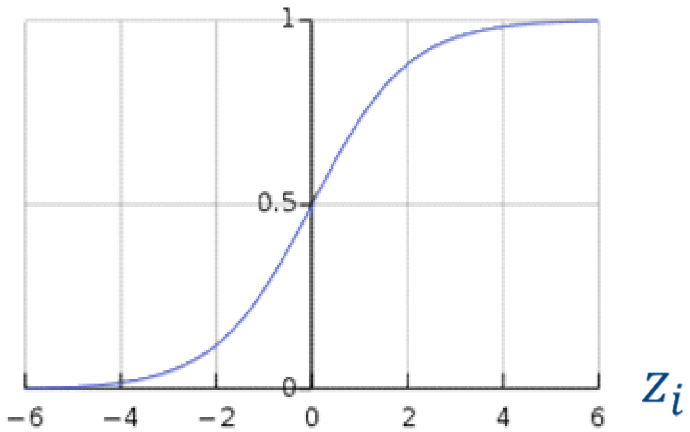
$$p(y_i = 1|x_i) = \sigma(z_i)$$



Interpretation of Logistic Regression

$$z_i = b_0 + (b_1 \times x_{i1}) + (b_2 \times x_{i2}) + \cdots + (b_M \times x_{iM})$$

$$p(y_i = 1|x_i) = \sigma(z_i)$$



- May think of vector b as a template or filter (will visualize to make clear)
- If x_i is aligned/matched with b, then the sum will be larger
- The parameter b_0 is a bias to correct for class prevalences

Artificial Neurons

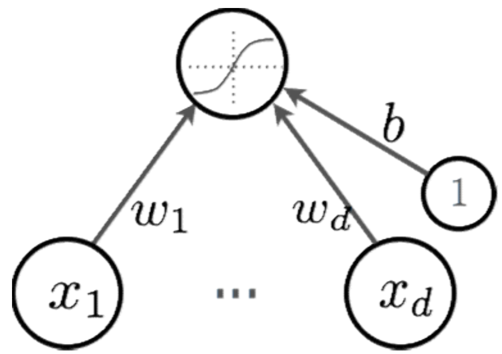
Artificial Neuron

- Neuron pre-activation (or input activation):

$$a(\mathbf{x}) = b + \sum_i w_i x_i = b + \mathbf{w}^\top \mathbf{x}$$

- Neuron output activation:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$



where

$\mathbf{w}$ are the weights (parameters)

b is the bias term

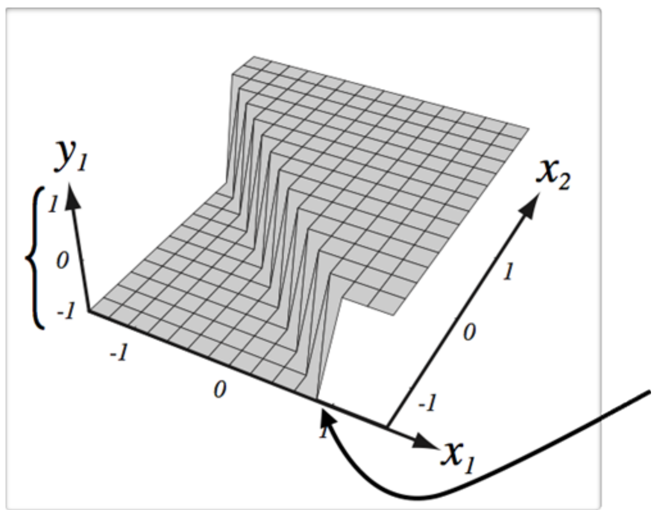
$g(\cdot)$ is called the activation function

Artificial Neuron

- Output activation of the neuron:

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g(b + \sum_i w_i x_i)$$

Range is determined by $g(\cdot)$



(from Pascal Vincent's slides)

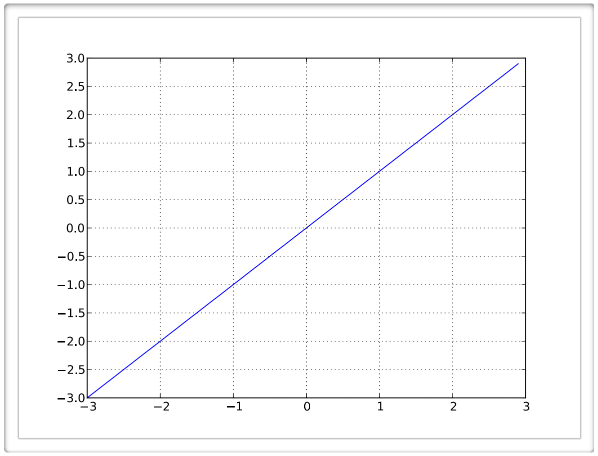
Bias only changes the position of the riff

Activation Function

- Linear activation function:

$$g(a) = a$$

   - ➢ No nonlinear transformation
   - ➢ No input squashing

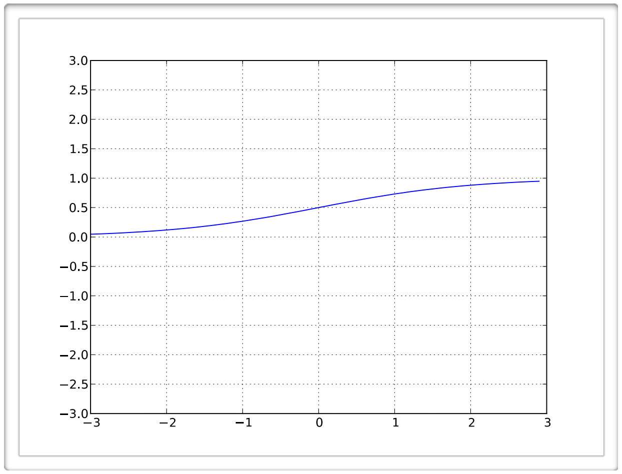


Activation Function

- Sigmoid activation function:

$$g(a) = \mathrm{sigm}(a) = \frac{1}{1+\exp(-a)}$$

- ➢ Squashes the neuron's output between 0 and 1
- ➢ Always positive
- ➢ Bounded
- ➢ Strictly Increasing



Does this ring a bell?

Activation Function

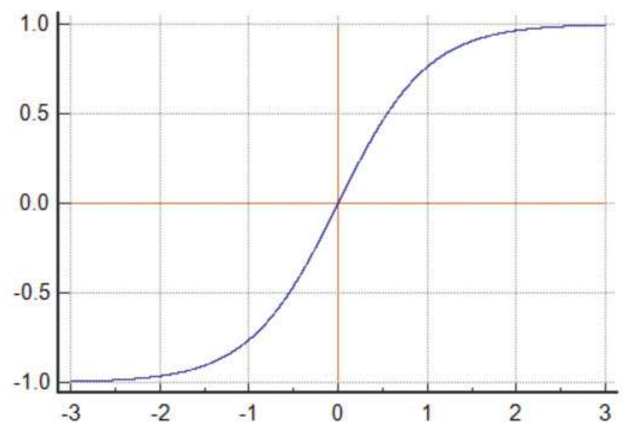
- Hyperbolic tangent ("tanh") activation function:

$$g(a) = \tanh(a) =$$

$$= \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} = \frac{\exp(2a) - 1}{\exp(2a) + 1}$$

- Squashes the neuron's activation between -1 and 1
- Can be positive or negative
- Bounded
- Strictly increasing

(wrong plot)

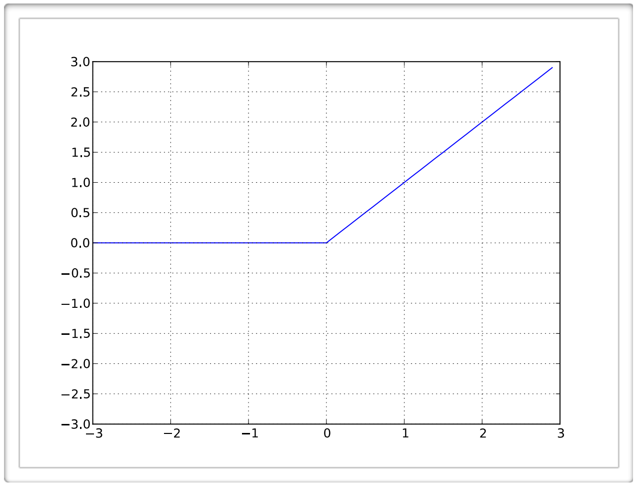


Activation Function

- Rectified linear (ReLU) activation function:

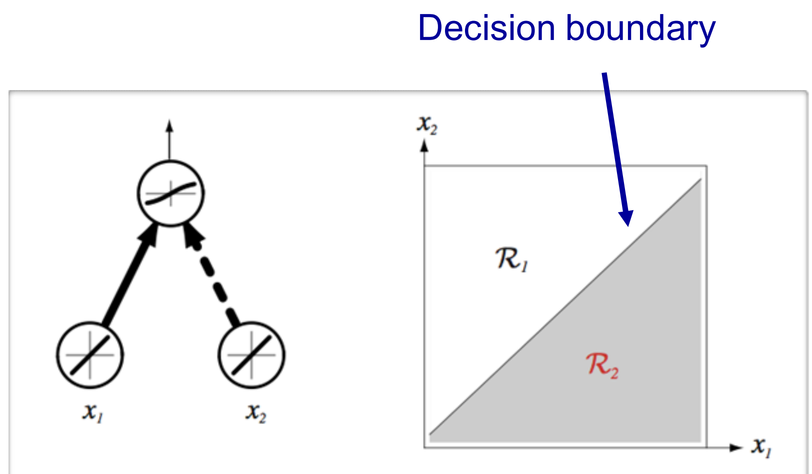
$$g(a) = \mathrm{reclin}(a) = \max(0, a)$$

- ➢ Bounded below by 0 (always non-negative)
- ➢ Tends to produce units with sparse activities
- ➢ Not upper bounded
- ➢ Strictly increasing



Decision Boundary of a Neuron

- Binary classification:
  - With sigmoid, one can interpret neuron as estimating $p(y = 1|\mathbf{x})$
  - Interpret as a logistic classifier
  - If activation is greater than 0.5, predict 1
  - Otherwise predict 0

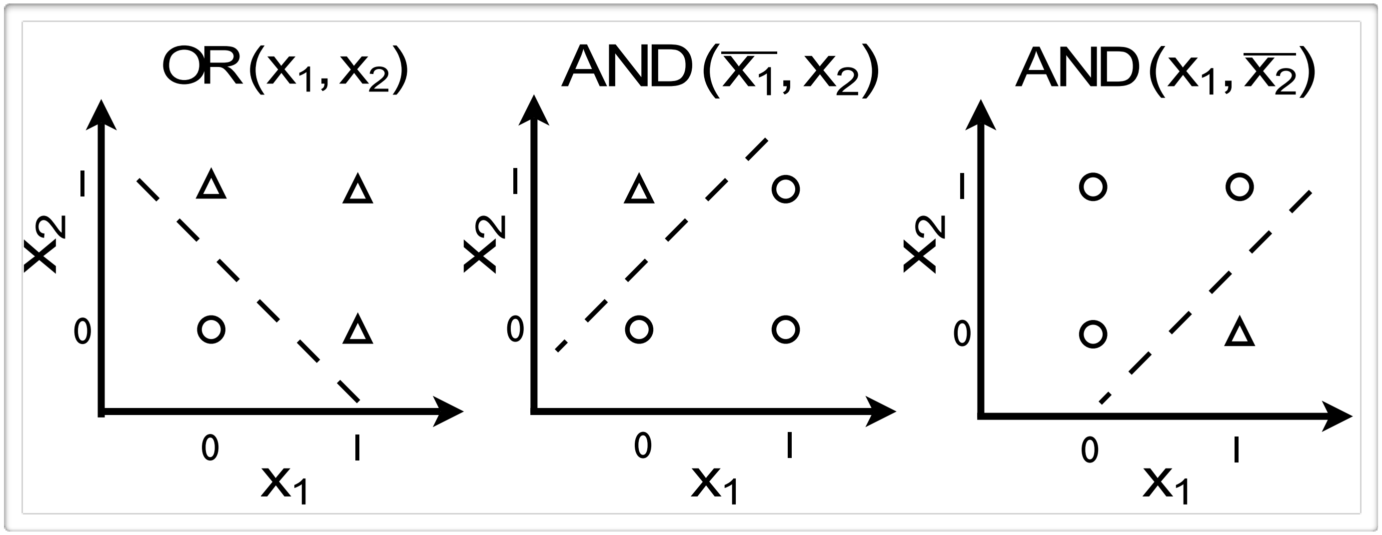


(from Pascal Vincent's slides)

Same idea can be applied to a tanh( ) activation

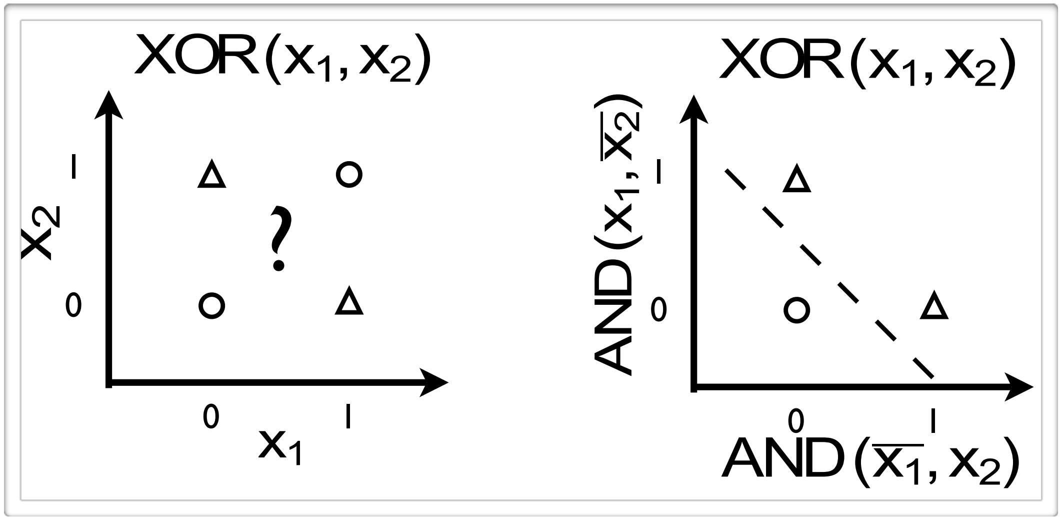
Capacity of a Single Neuron

- Can solve linearly separable problems.



Capacity of a Single Neuron

- Can not solve non-linearly separable problems.

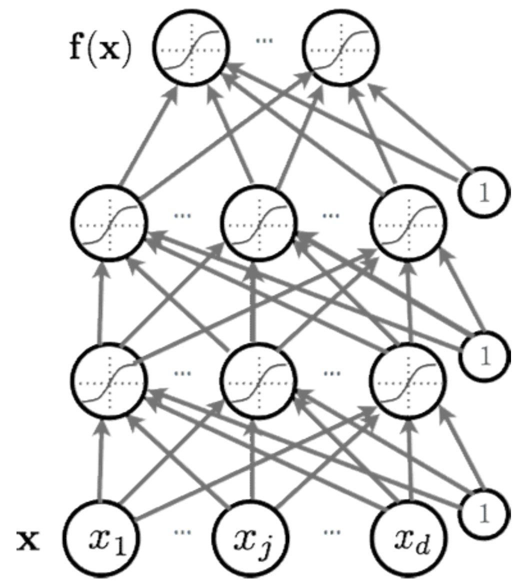


- Need to transform the input into a better representation.
- Remember basis functions!

Feed-Forward Neural Nets

Feedforward Neural Networks

- How neural networks predict f(x) given an input x:
  - Forward propagation
  - Types of units
  - Capacity of neural networks
- How to train neural nets:
  - Loss function
  - Back-propagation with gradient descent
- More recent techniques:
  - Dropout
  - Batch normalization
  - Unsupervised Pre-training



Single Hidden Layer Neural Net

- Hidden layer pre-activation:

$$\mathbf{a}(\mathbf{x}) = \mathbf{b}^{(1)} + \mathbf{W}^{(1)}\mathbf{x}$$

$$\left(a(\mathbf{x})_i = b_i^{(1)} + \sum_j W_{i,j}^{(1)} x_j\right)$$

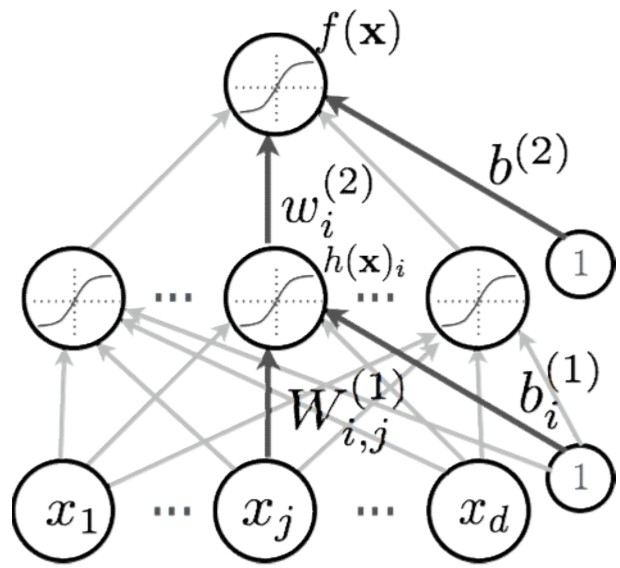
- Hidden layer activation:

$$\mathbf{h}(\mathbf{x}) = \mathbf{g}(\mathbf{a}(\mathbf{x}))$$

- Output layer activation:

$$f(\mathbf{x}) = o\left(b^{(2)} + \mathbf{w}^{(2)\top}\mathbf{h}^{(1)}\mathbf{x}\right)$$

Output activation function



Softmax Activation Function

- Remember multi-way classification:
  - We need multiple outputs (1 output per class)
  - We need to estimate conditional probability: $p(y = c|\mathbf{x})$
  - Discriminative Learning

- Softmax activation function at the output

$$\mathbf{o}(\mathbf{a}) = \mathrm{softmax}(\mathbf{a}) = \left[\frac{\exp(a_1)}{\sum_c \exp(a_c)} \cdots \frac{\exp(a_C)}{\sum_c \exp(a_c)}\right]^\top$$

  - strictly positive
  - sums to one

- Predict class with the highest estimated class conditional probability.

Multilayer Neural Net

- Consider a network with L hidden layers.

- layer pre-activation for k>0

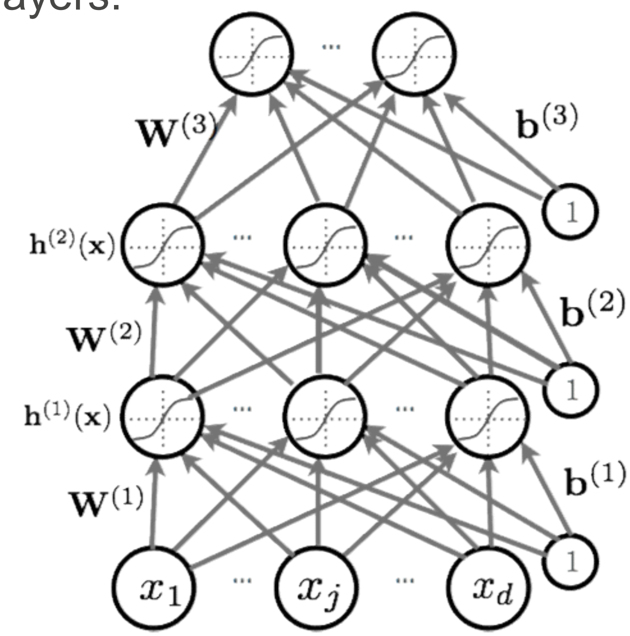
$$\mathbf{a}^{(k)}(\mathbf{x}) = \mathbf{b}^{(k)} + \mathbf{W}^{(k)}\mathbf{h}^{(k-1)}(\mathbf{x})$$

- hidden layer activation from 1 to L:

$$\mathbf{h}^{(k)}(\mathbf{x}) = \mathbf{g}(\mathbf{a}^{(k)}(\mathbf{x}))$$

- output layer activation (k=L+1):

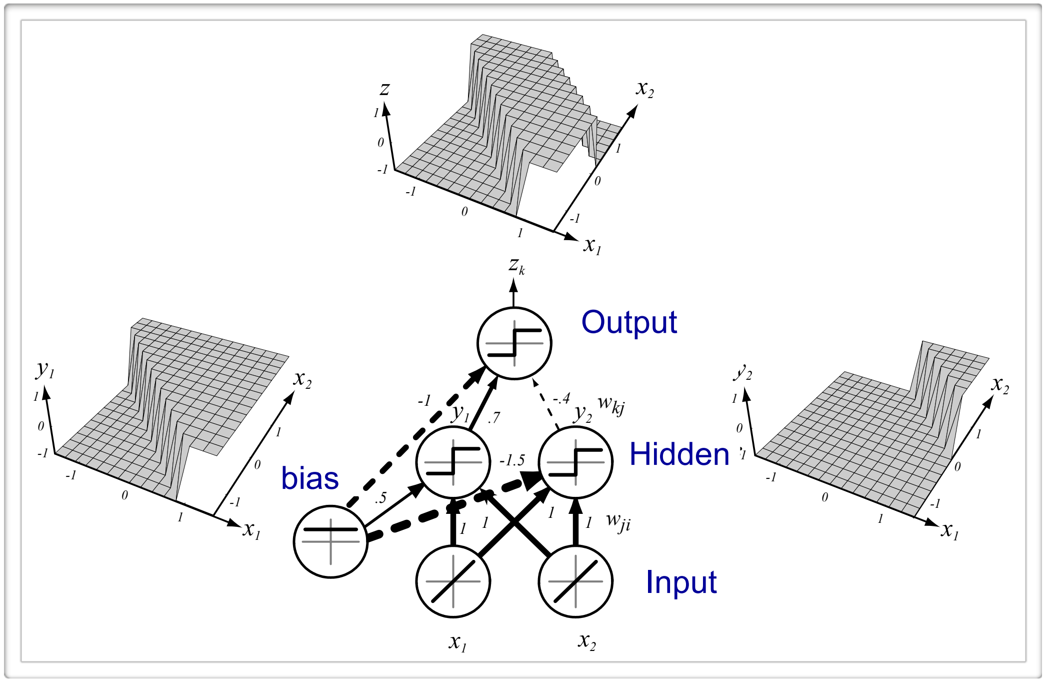
$$\mathbf{h}^{(L+1)}(\mathbf{x}) = \mathbf{o}(\mathbf{a}^{(L+1)}(\mathbf{x})) = \mathbf{f}(\mathbf{x})$$



$$(\mathbf{h}^{(0)}(\mathbf{x}) = \mathbf{x})$$

Capacity of Neural Nets

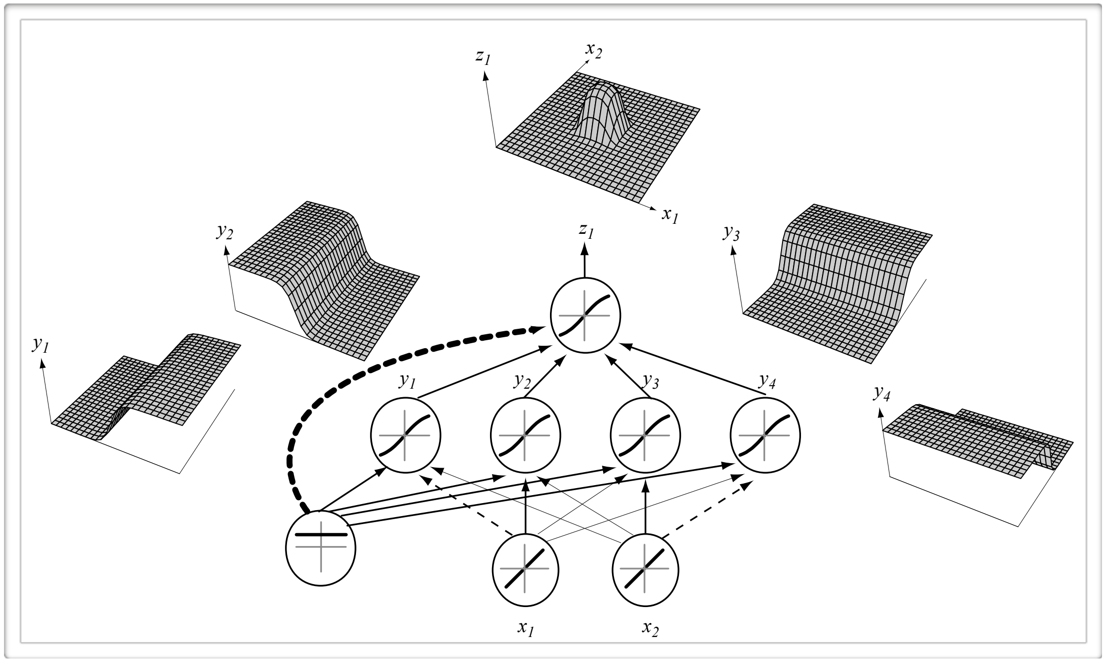
- Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

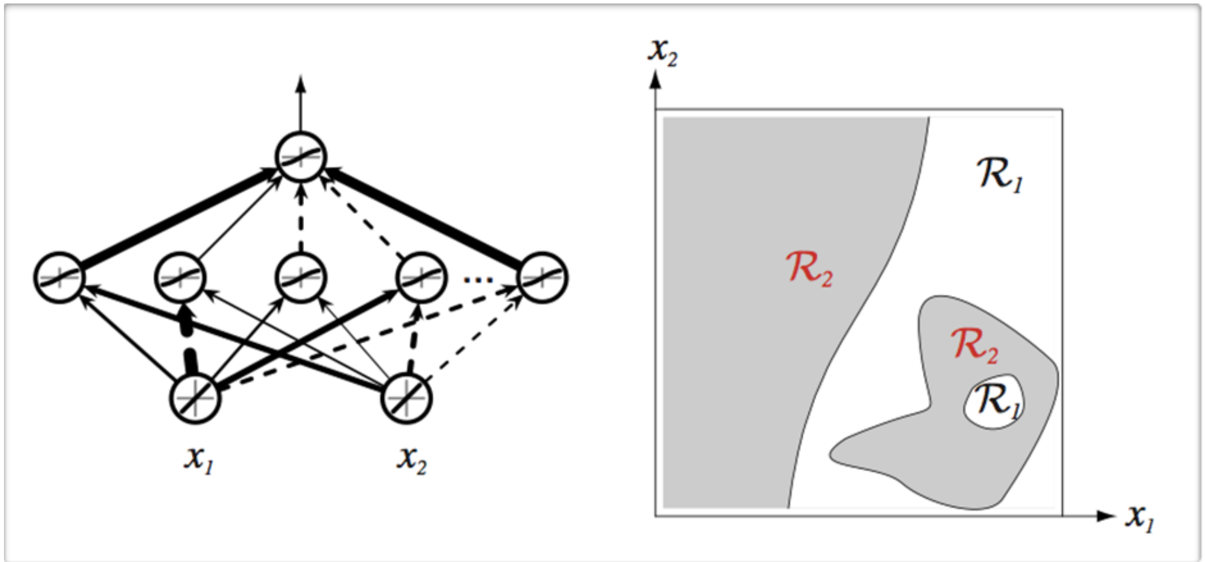
- Consider a single layer neural network



(from Pascal Vincent's slides)

Capacity of Neural Nets

- Consider a single layer neural network



(from Pascal Vincent's slides)

Universal Approximation

The key result is due to great mathematicians Kolmogorov and Arnold (very difficult to prove) established in 1956.

Any continuous function of m inputs can be represented **exactly** by a small (polynomial sized) two-layer network.

$$f(x_1, \ldots, x_m) = \sum_{i=1}^{2m+1} g_i \left(\sum_{j=1}^{m} h_{i,j}(x_j) \right)$$

Where g_i and $h_{i,j}$ are continuous scalar-to-scalar functions.

Universal Approximation

A much more trivial result to prove is:

For any (possibly discontinuous) $f : [0,1]^m \to \mathbb{R}$ we have

$$f(x_1, \ldots, x_m) = g\left(\sum_i h_i(x_i)\right)$$

for (discontinuous) scalar-to-scalar functions g and h_i.

Proof: Any single real number contains an infinite amount of information.

Select h_i to spread out the digits of its argument so that $\sum_i h_i(x_i)$ contains all the digits of all the x_i.

Universal Approximation

Another relatively straightforward result is due to Cybenko (1989):

Any continuous function can be approximated arbitrarily well by a two layer perceptron.

For any continuous $f : [0,1]^m \to \mathbb{R}$ and any $\varepsilon > 0$, there exists

$$F(x) = \alpha \cdot \sigma(Wx + \beta)$$

$$= \sum_i \alpha_i \sigma\left(\sum_j W_{i,j}\, x_j + \beta_i\right)$$

such that for all x in $[0,1]^m$ we have $|F(x) - f(x)| < \varepsilon$.

Universal Approximation

• Universal Approximation Theorem (Hornik, 1991):

- “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units’’

• This applies for sigmoid, tanh and many other activation functions.

• However, this does not mean that there is learning algorithm that can find the necessary parameter values.