School of Computer Science
Carnegie Mellon

10-701 Introduction to Machine Learning

Perceptron & Neural Networks

Readings:
Bishop Ch. 4.1.7, Ch. 5
Murphy Ch. 16.5, Ch. 28
Mitchell Ch. 4

Matt Gormley
Lecture 12
October 17, 2016

Reminders

- Homework 3:
  - due 10/24/16

Outline

- Discriminative vs. Generative
- Perceptron
- Neural Networks
- Backpropagation

DISCRIMINATIVE AND GENERATIVE CLASSIFIERS

Generative vs. Discriminative

- **Generative Classifiers:**
  - Example: Naïve Bayes
  - Define a joint model of the observations **x** and the labels y: $p(\boldsymbol{x}, y)$
  - Learning maximizes (joint) likelihood
  - Use Bayes' Rule to classify based on the posterior:

$$p(y|\mathbf{x}) = p(\mathbf{x}|y)p(y)/p(\mathbf{x})$$

- **Discriminative Classifiers:**
  - Example: Logistic Regression
  - Directly model the conditional: $p(y|\mathbf{x})$
  - Learning maximizes conditional likelihood

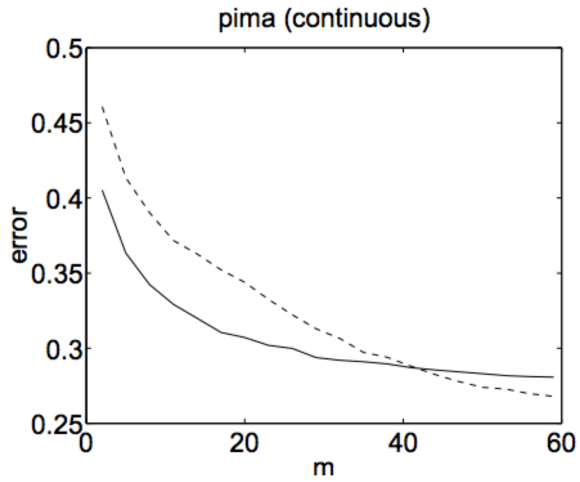
Generative vs. Discriminative

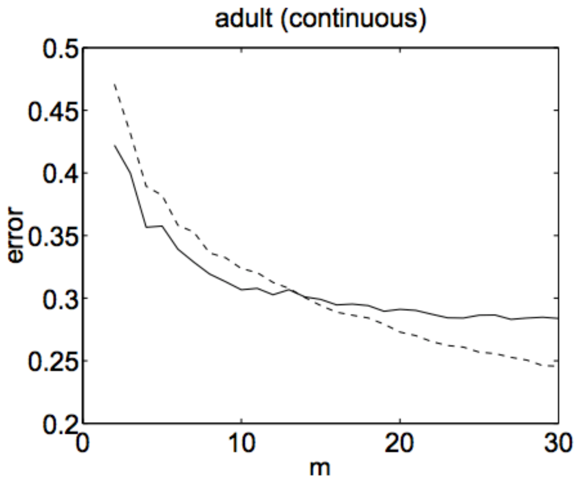
Finite Sample Analysis (Ng & Jordan, 2002)

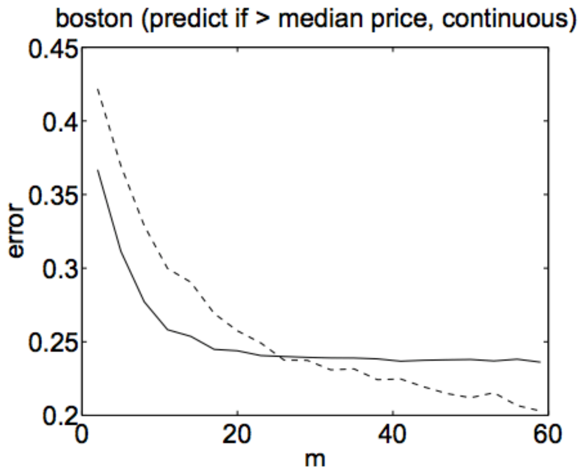
[Assume that we are learning from a finite training dataset]

If model assumptions are correct: Naive Bayes is a more efficient learner (requires fewer samples) than Logistic Regression

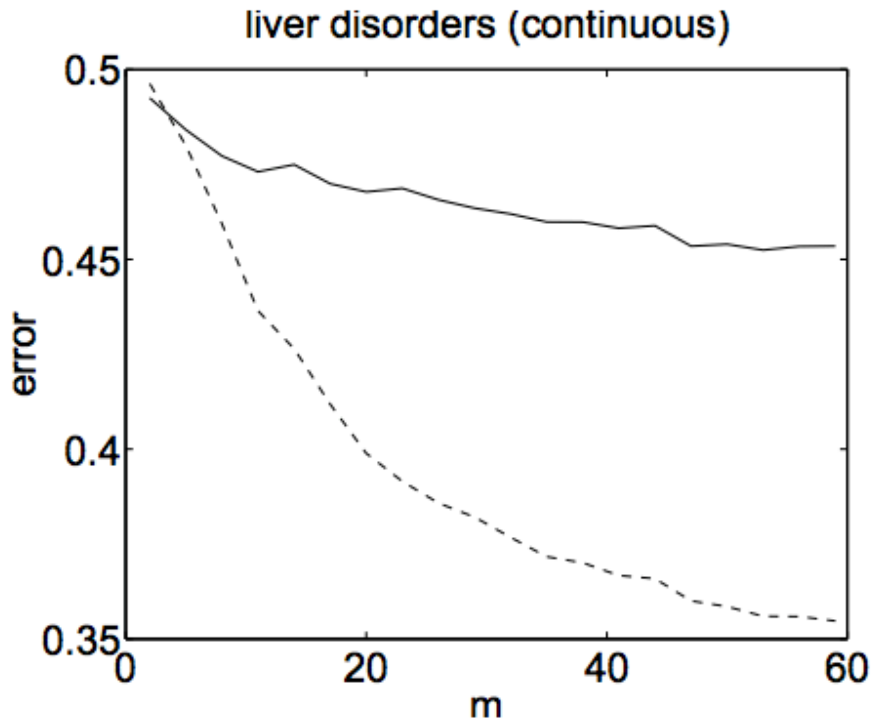
If model assumptions are incorrect: Logistic Regression has lower asymtotic error, and does better than Naïve Bayes

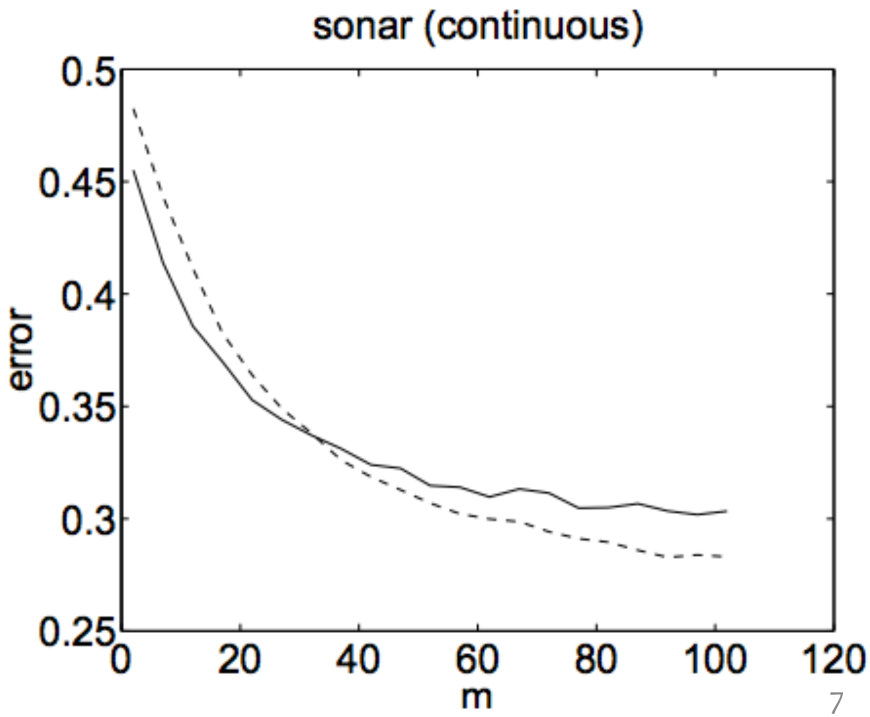






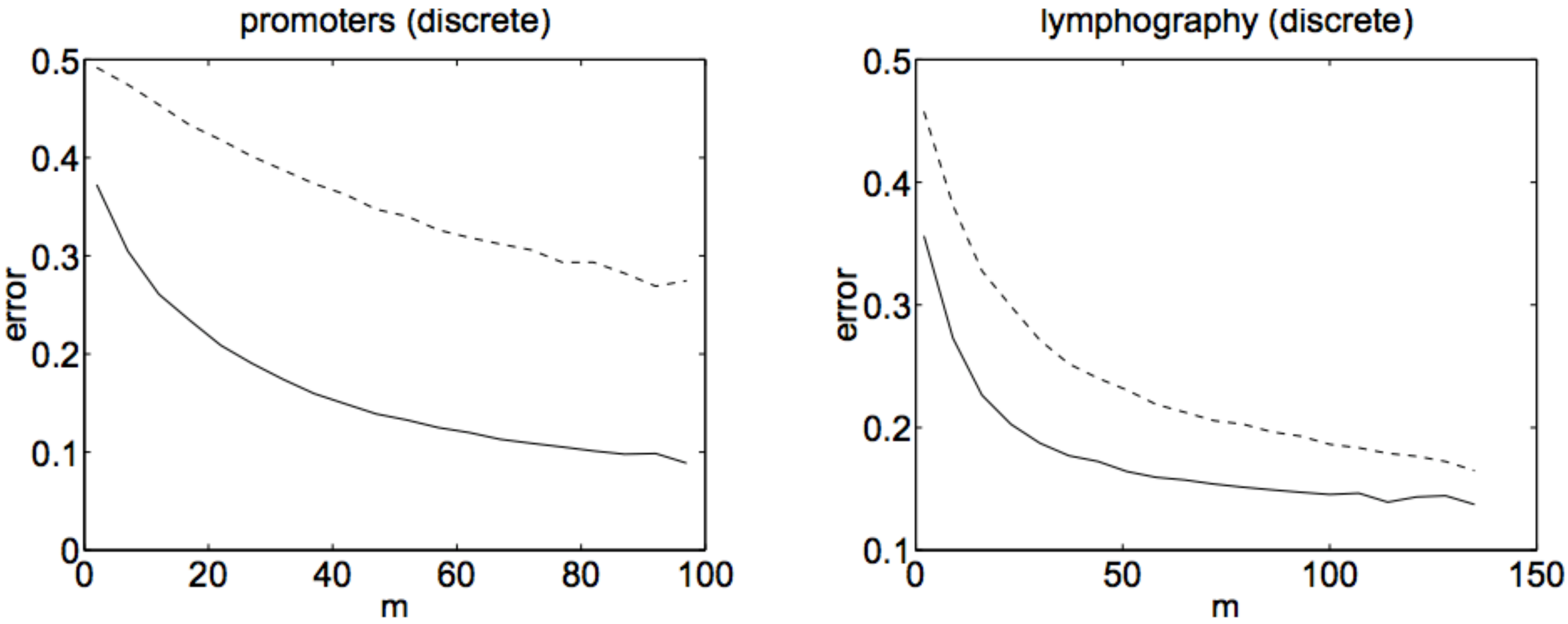
solid: NB dashed: LR





Slide courtesy of William Cohen

solid: NB dashed: LR



Naïve Bayes makes stronger assumptions about the data but needs fewer examples to estimate the parameters

“On Discriminative vs Generative Classifiers: ….” Andrew Ng and Michael Jordan, NIPS 2001.

Slide courtesy of William Cohen

Generative vs. Discriminative

Learning (Parameter Estimation)

Naïve Bayes:
Parameters are decoupled → Closed form solution for MLE

Logistic Regression:
Parameters are coupled → No closed form solution – must use iterative optimization techniques instead

Naïve Bayes vs. Logistic Reg.

Learning (MAP Estimation of Parameters)

Bernoulli Naïve Bayes:
Parameters are probabilities → Beta prior (usually) pushes probabilities away from zero / one extremes

Logistic Regression:
Parameters are not probabilities → Gaussian prior encourages parameters to be close to zero

(effectively pushes the probabilities away from zero / one extremes)

Naïve Bayes vs. Logistic Reg.

Features

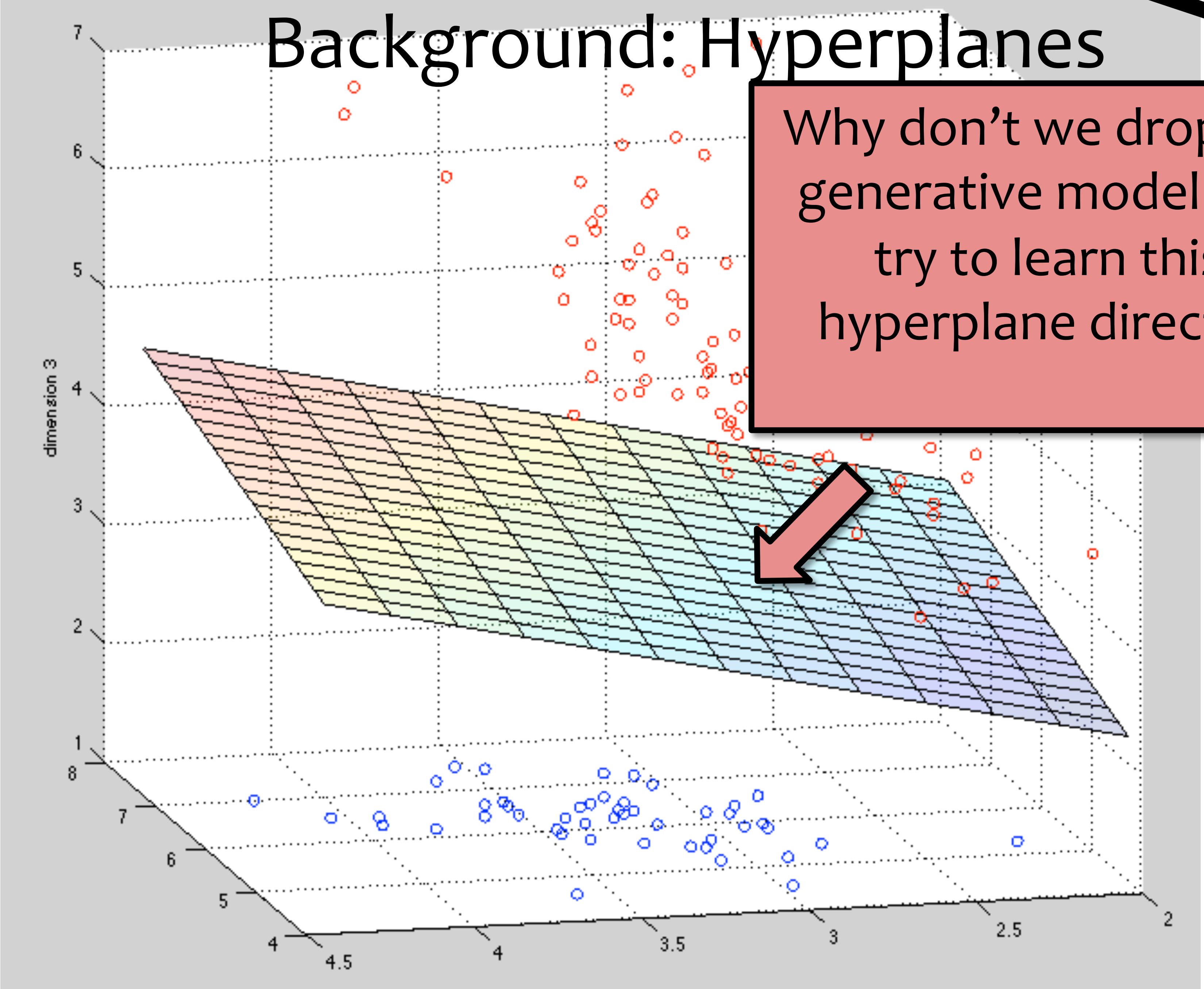
Naïve Bayes:
Features x are assumed to be conditionally independent given y. (i.e. Naïve Bayes Assumption)

Logistic Regression:
No assumptions are made about the form of the features x. They can be dependent and correlated in any fashion.

THE PERCEPTRON ALGORITHM

Background: Hyperplanes

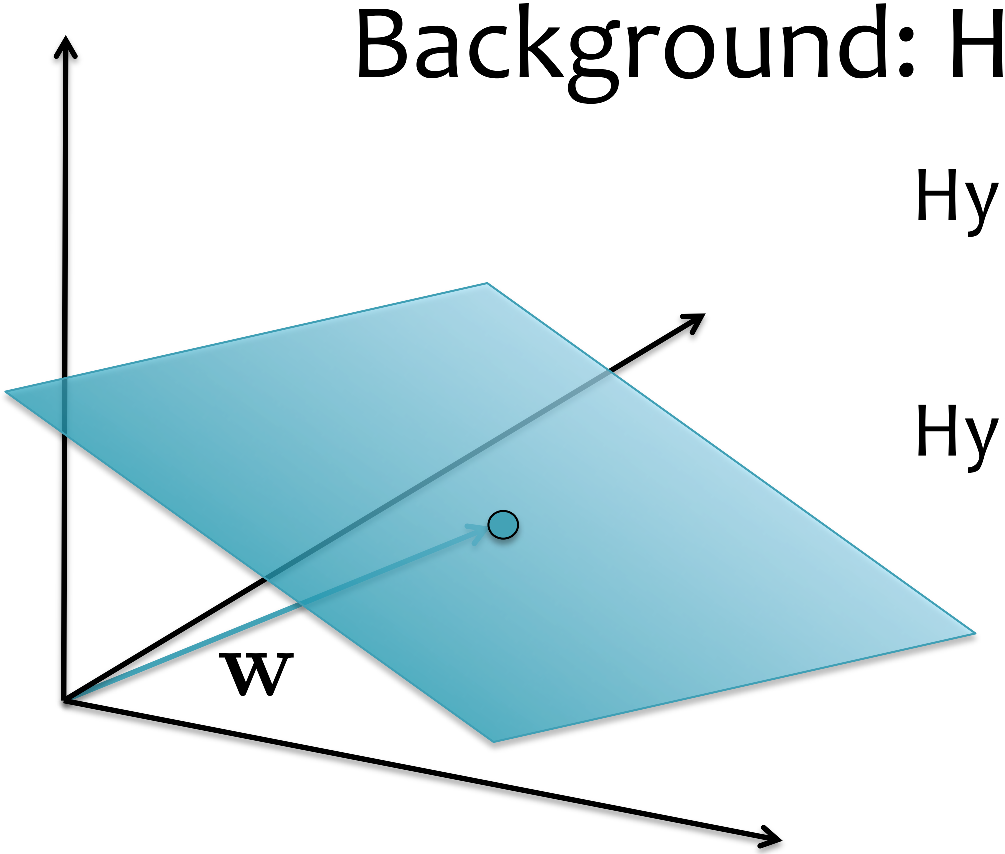
Recall…



Why don't we drop the generative model and try to learn this hyperplane directly?

Recall...

Background: Hyperplanes



Hyperplane (Definition 1):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = b\}$$

Hyperplane (Definition 2):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0$$

$$\text{and } x_1 = 1\}$$

Half-spaces:

$$\mathcal{H}^+ = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} > 0 \text{ and } x_1 = 1\}$$

$$\mathcal{H}^- = \{\mathbf{x} : \mathbf{w}^T\mathbf{x} < 0 \text{ and } x_1 = 1\}$$

Recall…

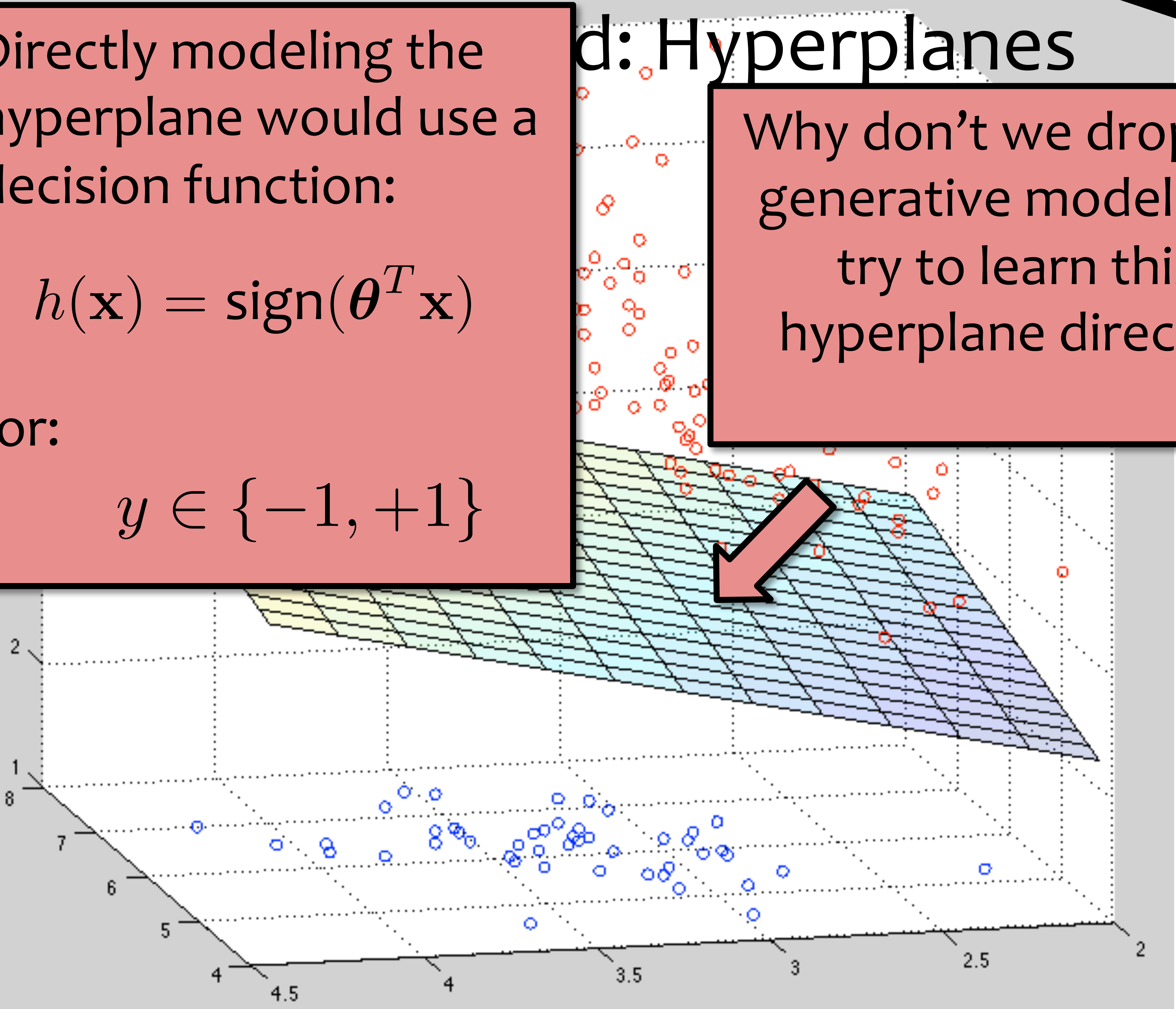
d: Hyperplanes

Directly modeling the hyperplane would use a decision function:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

for:

$$y \in \{-1, +1\}$$



Why don't we drop the generative model and try to learn this hyperplane directly?

Online Learning Model

Setup:

- We receive an example (**x**, y)
- Make a prediction h(**x**)
- Check for correctness h(**x**) = y?

Goal:

- **Minimize** the number of **mistakes**

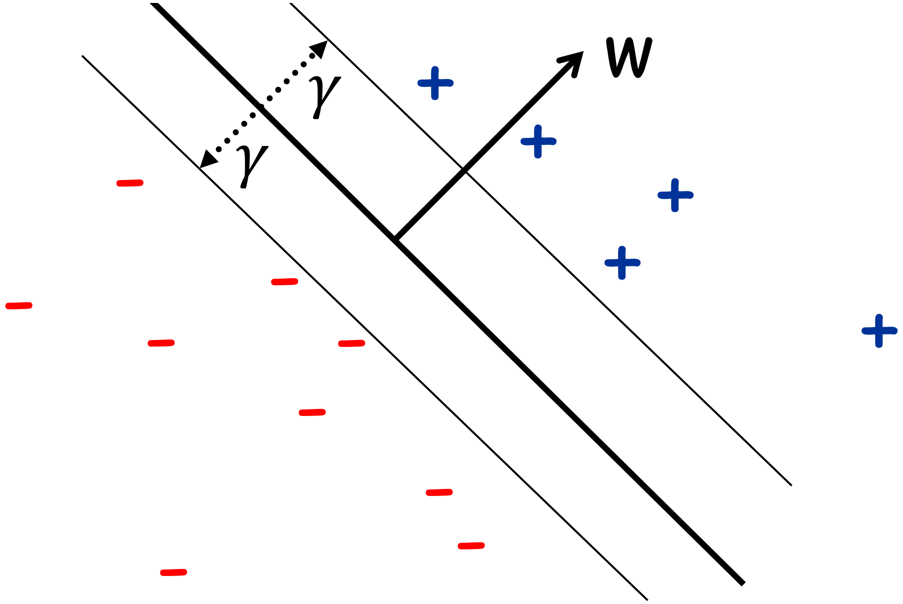
Recall...

Margins

Definition: The margin of example x w.r.t. a linear sep. w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S wrt a linear separator w is the smallest margin over points $x \in S$.

Definition: The margin γ of a set of examples S is the maximum γ_w over all linear separators w.



Slide from Nina Balcan

Perceptron Algorithm

Data: Inputs are continuous vectors of length K. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{+1, -1\}$$

Prediction: Output determined by hyperplane.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Learning: Iterative procedure:

- **while** not converged
  - **receive** next example (**x**, y)
  - **predict** y' = h(**x**)
  - **if** positive mistake: **add x** to parameters
  - **if** negative mistake: **subtract x** from parameters

Perceptron Algorithm

Learning:

Algorithm 1 Perceptron Learning Algorithm (Batch)

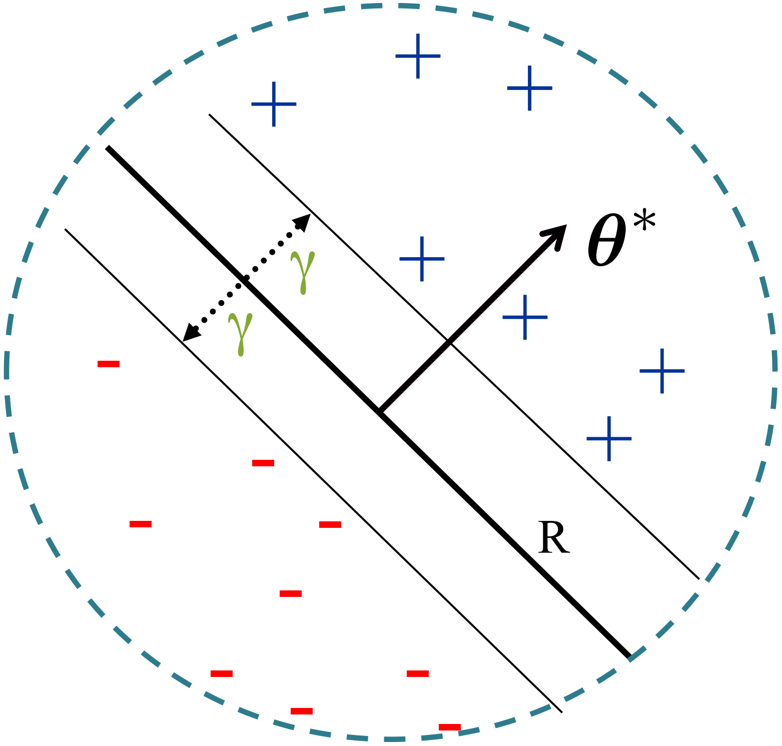
```
procedure PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$)
    $\boldsymbol{\theta} \leftarrow \mathbf{0}$                                  ▷ Initialize parameters
    while not converged do
        for $i \in \{1, 2, \ldots, N\}$ do                    ▷ For each example
            $\hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$                    ▷ Predict
            if $\hat{y} \neq y^{(i)}$ then                          ▷ If mistake
                $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$          ▷ Update parameters
    return $\boldsymbol{\theta}$
```

Analysis: Perceptron

Perceptron Mistake Bound

Guarantee: If data has margin γ and all points inside a ball of radius R, then Perceptron makes $\leq (R/\gamma)^2$ mistakes.

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes; algo is invariant to scaling.)



Slide from Nina Balcan

Analysis: Perceptron

Perceptron Mistake Bound

Theorem 0.1 (Block (1962), Novikoff (1962)).

Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.

Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$

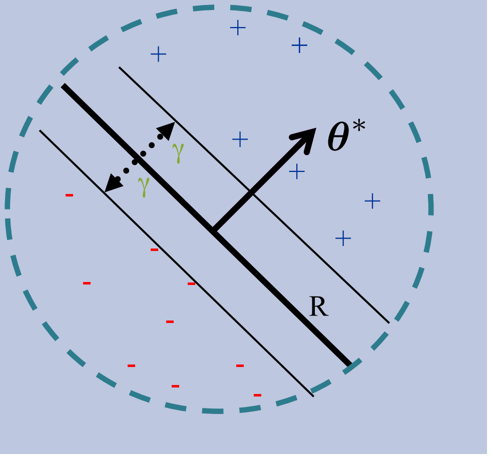


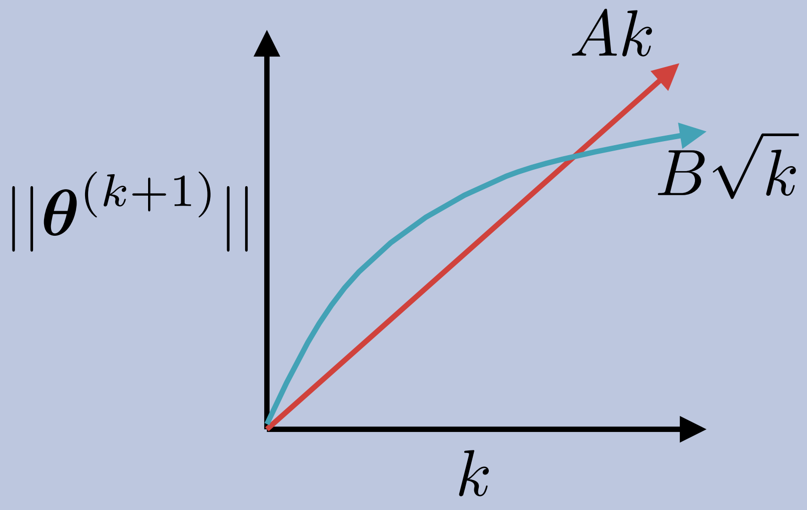
Figure from Nina Balcan

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

We will show that there exist constants A and B s.t.

$$Ak \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$$



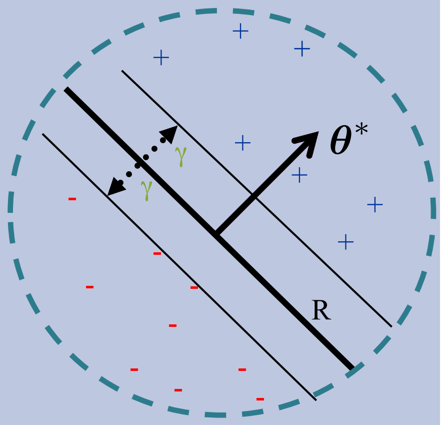
Analysis: Perceptron

Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ s.t. $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$



Algorithm 1 Perceptron Learning Algorithm (Online)

```
procedure PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots\}$)
    $\boldsymbol{\theta} \leftarrow \mathbf{0}$, $k = 1$                                  ▷ Initialize parameters
    for $i \in \{1, 2, \ldots\}$ do                                   ▷ For each example
        if $y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$ then                           ▷ If mistake
            $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}$                 ▷ Update parameters
            $k \leftarrow k + 1$
    return $\boldsymbol{\theta}$
```

Analysis: Perceptron

Whiteboard:
Proof of Perceptron Mistake Bound

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 1: for some A, $Ak \leq ||\boldsymbol{\theta}^{(k+1)}||$

$$\boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^{*} = (\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)})\boldsymbol{\theta}^{*}$$

by Perceptron algorithm update

$$= \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^{*} + y^{(i)}(\boldsymbol{\theta}^{*} \cdot \mathbf{x}^{(i)})$$

$$\geq \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^{*} + \gamma$$

by assumption

$$\Rightarrow \boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^{*} \geq k\gamma$$

by induction on k since $\theta^{(1)} = \mathbf{0}$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \geq k\gamma$$

since $\underbrace{||\mathbf{w}|| \times ||\mathbf{u}|| \geq \mathbf{w} \cdot \mathbf{u}}_{\text{Cauchy-Schwartz inequality}}$ and $||\theta^{*}|| = 1$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 2: for some B, $||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$

$$||\boldsymbol{\theta}^{(k+1)}||^2 = ||\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}||^2$$

by Perceptron algorithm update

$$= ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2 + 2y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)})$$

$$\leq ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2$$

since kth mistake $\Rightarrow y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$

$$= ||\boldsymbol{\theta}^{(k)}||^2 + R^2$$

since $(y^{(i)})^2||\mathbf{x}^{(i)}||^2 = ||\mathbf{x}^{(i)}||^2 = R^2$ by assumption and $(y^{(i)})^2 = 1$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}||^2 \leq kR^2$$

by induction on k since $(\theta^{(1)})^2 = 0$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 3: Combining the bounds finishes the proof.

$$k\gamma \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

$$\Rightarrow k \leq \underbrace{(R/\gamma)^2}_{\text{The total number of mistakes must be less than this}}$$

Extensions of Perceptron

- **Kernel Perceptron**
  - Choose a kernel K(x’, x)
  - Apply the **kernel trick** to Perceptron
  - Resulting algorithm is **still very simple**
- **Structured Perceptron**
  - Basic idea can also be applied when **y** ranges over an exponentially large set
  - Mistake bound **does not** depend on the size of that set

Summary: Perceptron

- Perceptron is a simple **linear classifier**
- **Simple learning algorithm**: when a mistake is made, add / subtract the features
- For linearly separable and inseparable data, we can **bound the number of mistakes** (geometric argument)
- Extensions support nonlinear separators and structured prediction

RECALL: LOGISTIC REGRESSION

Using gradient ascent for linea classifiers

Recall…

Key idea behind today's lecture:

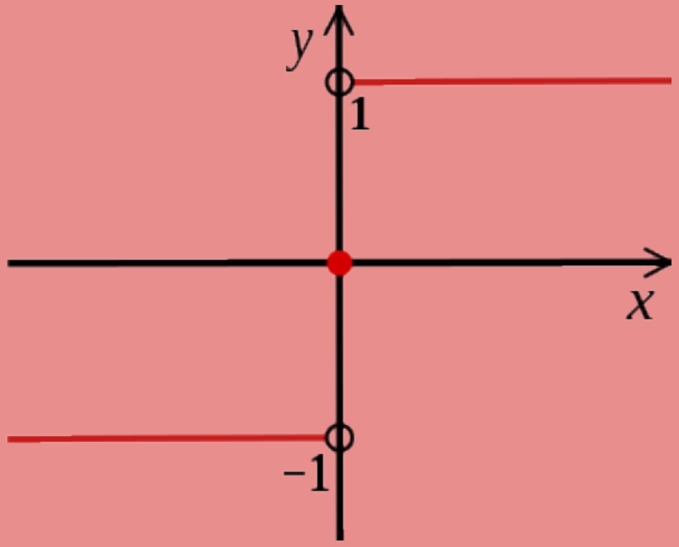
1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Using gradient ascent for linea classifiers

Recall...

This decision function isn't differentiable:

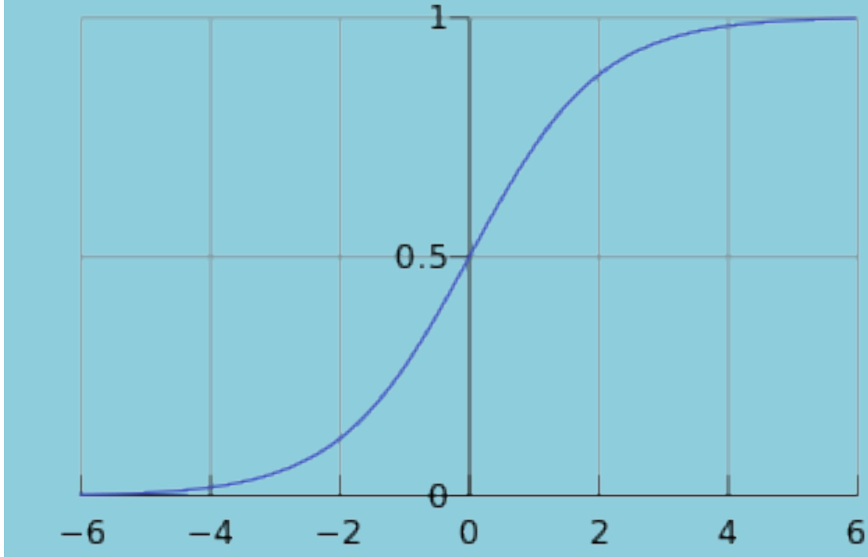
$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$



sign(x)

Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y=1|\mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^T \mathbf{x})}$$



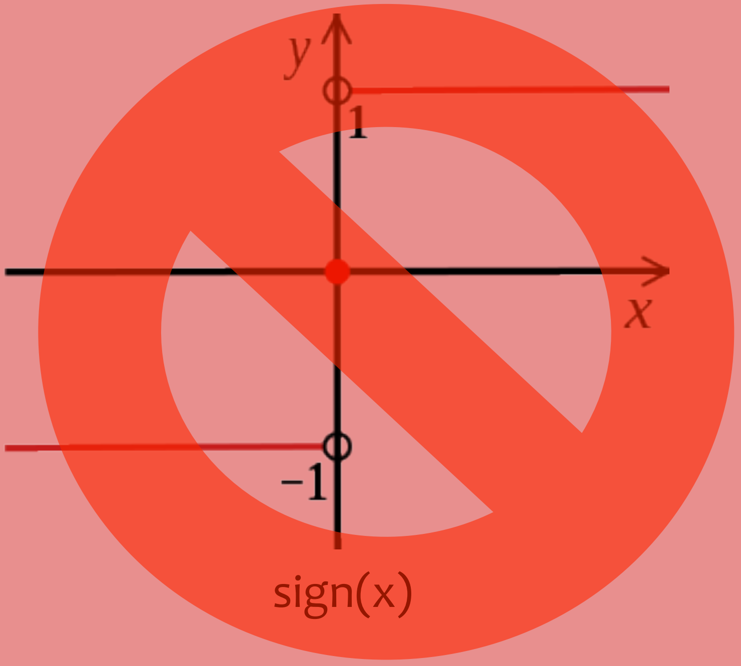
$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

Using gradient ascent for linear classifiers

Recall…

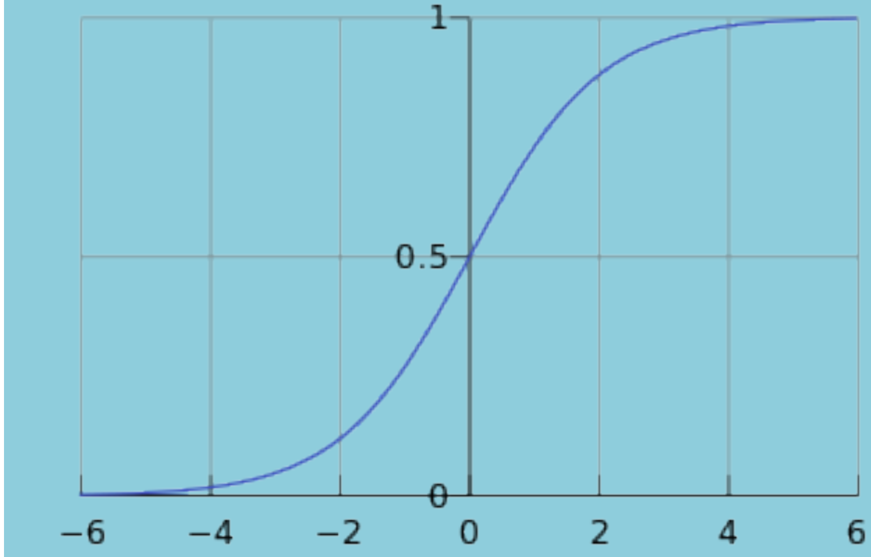
This decision function isn't differentiable:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T\mathbf{x})$$



Use a differentiable function instead:

$$p_{\boldsymbol{\theta}}(y=1|\mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^T\mathbf{x})}$$



$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

Recall…

Logistic Regression

Data: Inputs are continuous vectors of length K. Outputs are discrete.

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N} \text{ where } \mathbf{x} \in \mathbb{R}^K \text{ and } y \in \{0, 1\}$$

Model: Logistic function applied to dot product of parameters with input vector.

$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

Learning: finds the parameters that minimize some objective function. $\boldsymbol{\theta}^* = \operatorname*{argmin}_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

Prediction: Output is the most probable class.

$$\hat{y} = \operatorname*{argmax}_{y \in \{0,1\}} p_{\boldsymbol{\theta}}(y|\mathbf{x})$$

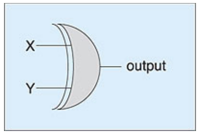
NEURAL NETWORKS

Learning highly non-linear functions

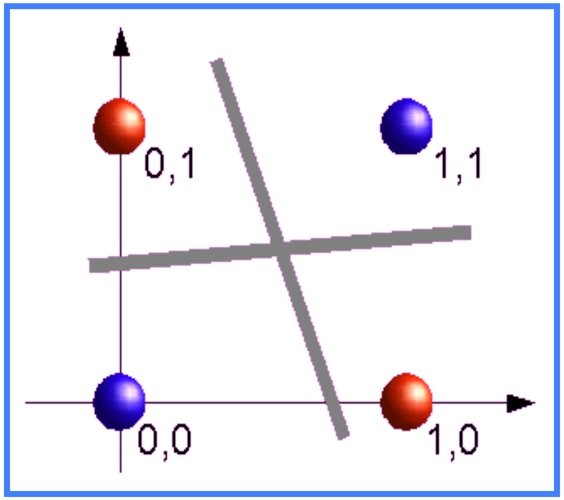
f: X → Y

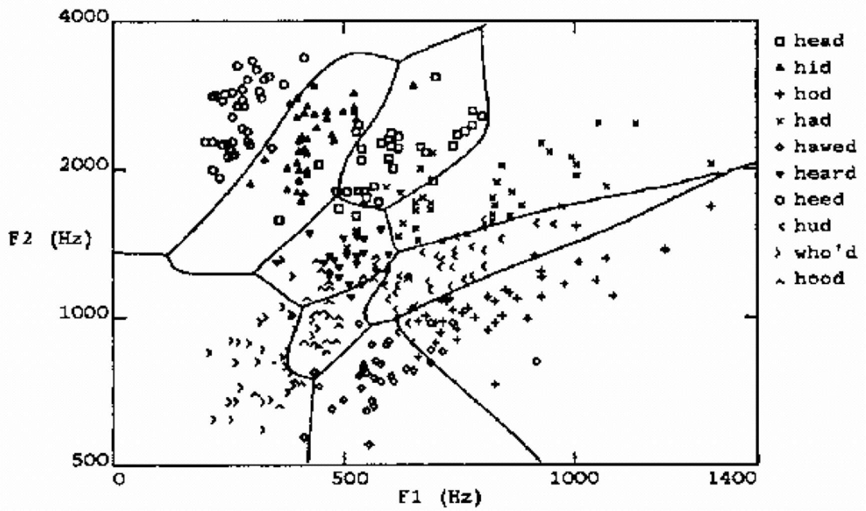
- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

The XOR gate



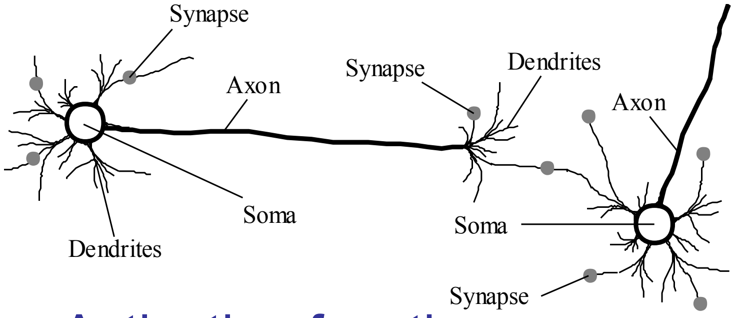
Speech recognition

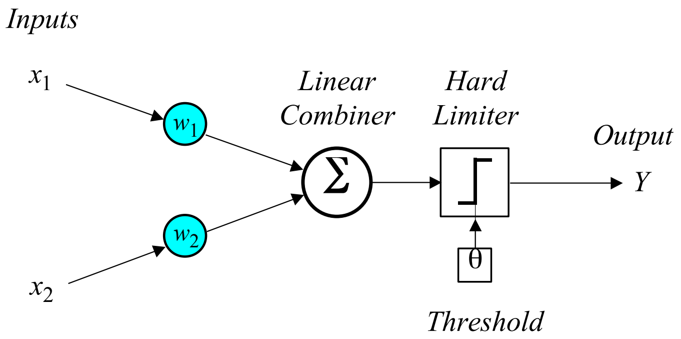




Perceptron and Neural Nets

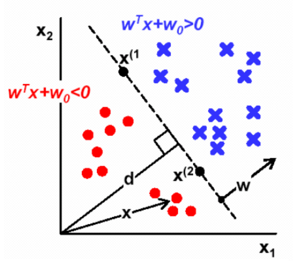
- From biological neuron to artificial neuron (perceptron)



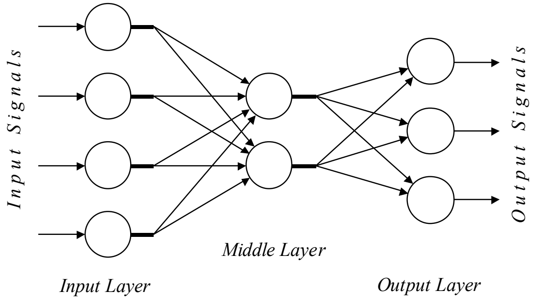


- Activation function

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \omega_0 \\ -1, & \text{if } X < \omega_0 \end{cases}$$

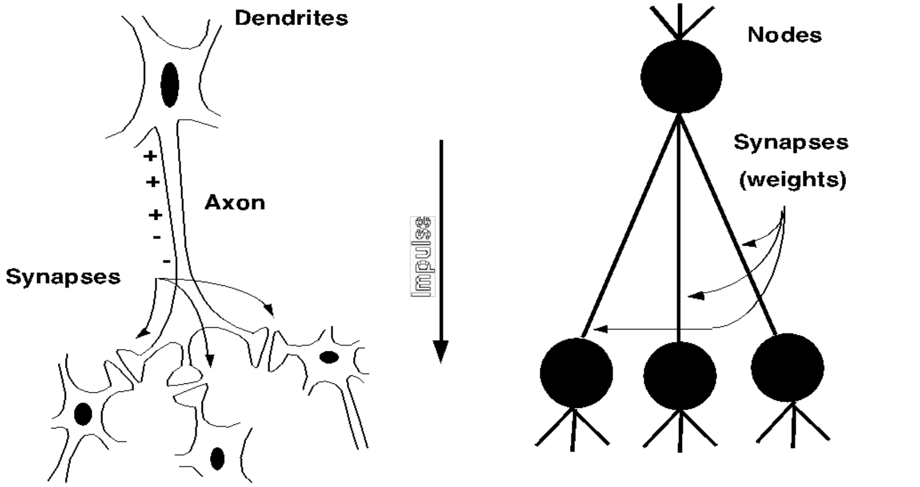


- Artificial neuron networks
  - supervised learning
  - gradient descent



Connectionist Models

- Consider humans:
  - Neuron switching time
    ~ 0.001 second
  - Number of neurons
    ~ 10^{10}
  - Connections per neuron
    ~ 10^{4-5}
  - Scene recognition time
    ~ 0.1 second
  - 100 inference steps doesn't seem like enough
    → much parallel computation
- Properties of artificial neural nets (ANN)
  - Many neuron-like threshold switching units
  - Many weighted interconnections among units
  - Highly parallel, distributed processes

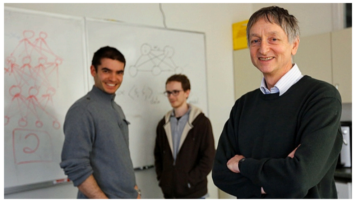


Motivation

Why is everyone talking about Deep Learning?

- Because a lot of money is invested in it…
  - DeepMind: Acquired by Google for **$400 million**
  - DNNResearch: **Three person startup** (including Geoff Hinton) acquired by Google for unknown price tag
  - Enlitic, Ersatz, MetaMind, Nervana, Skylab: Deep Learning startups commanding **millions of VC dollars**
- Because it made the **front page** of the New York Times

G$$gle™

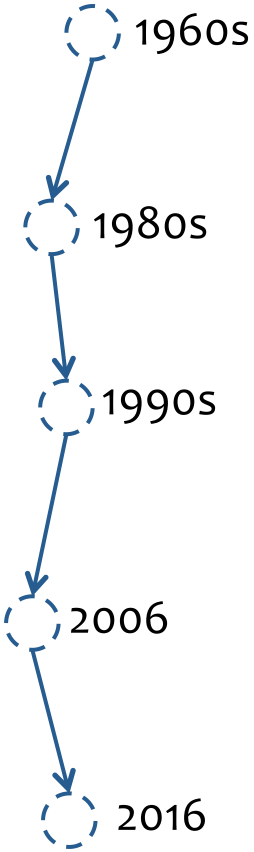


MetaMind

The New York Times

Motivation

Why is everyone talking about Deep Learning?



Deep learning:

- Has won numerous pattern recognition competitions
- Does so with minimal feature engineering

This wasn't always the case!

Since 1980s: Form of models hasn't changed much, but lots of new tricks…

- More hidden units
- Better (online) optimization
- New nonlinear functions (ReLUs)
- Faster computers (CPUs and GPUs)

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

Face

Face

Not a face

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of t

– Decision functio

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

A Recipe for

$\boldsymbol{y}_i)$

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

opp site the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Goals for Today's Lecture

1. Explore a **new class of decision functions** (Neural Networks)
2. Consider **variants of this recipe** for training

B

A Recipe for

1.

$y_i)$

2. Choose each of these:

– Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

4. Train with SGD:

(take small steps opposite the gradient)

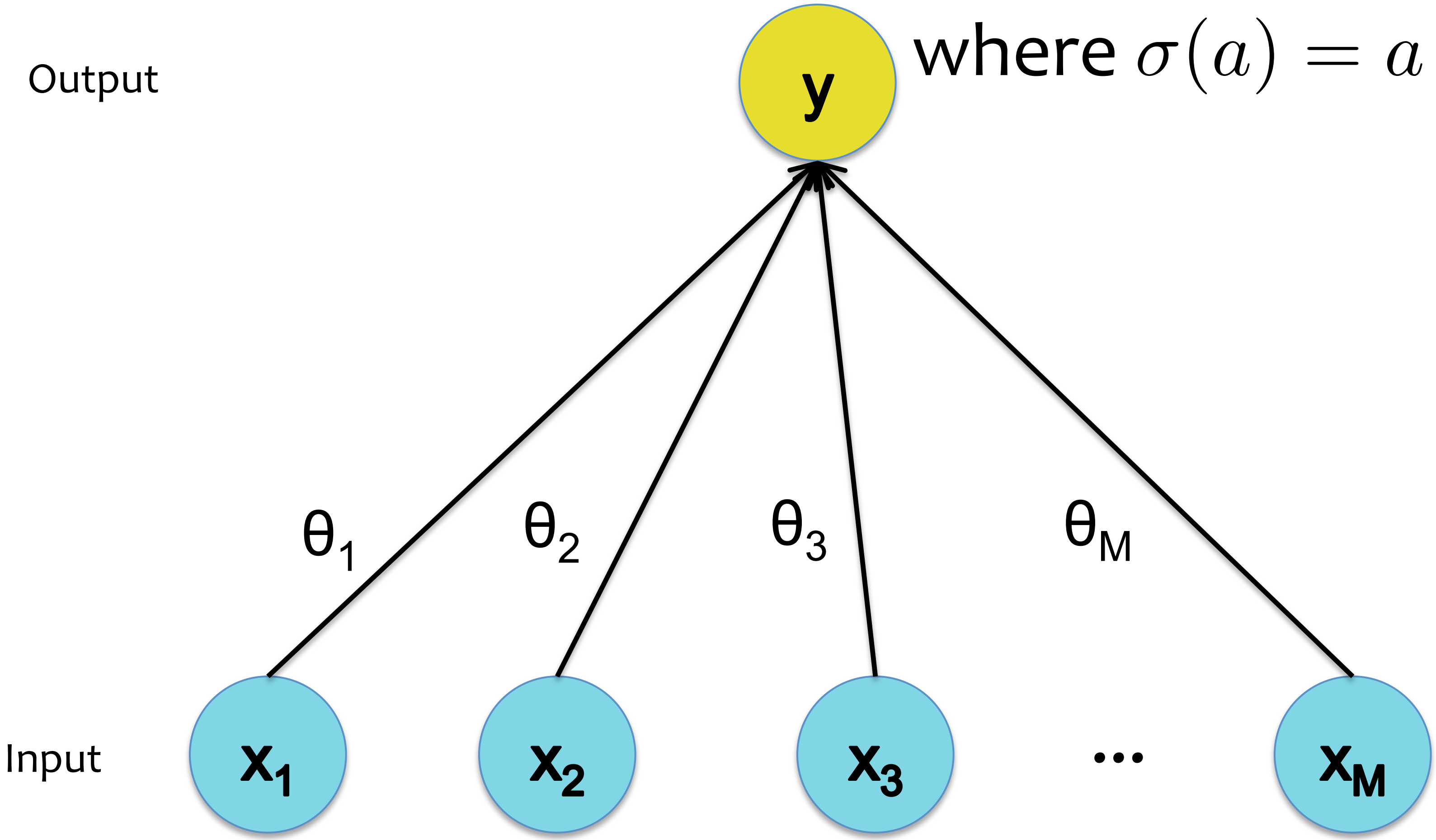
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Decision Functions

Linear Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

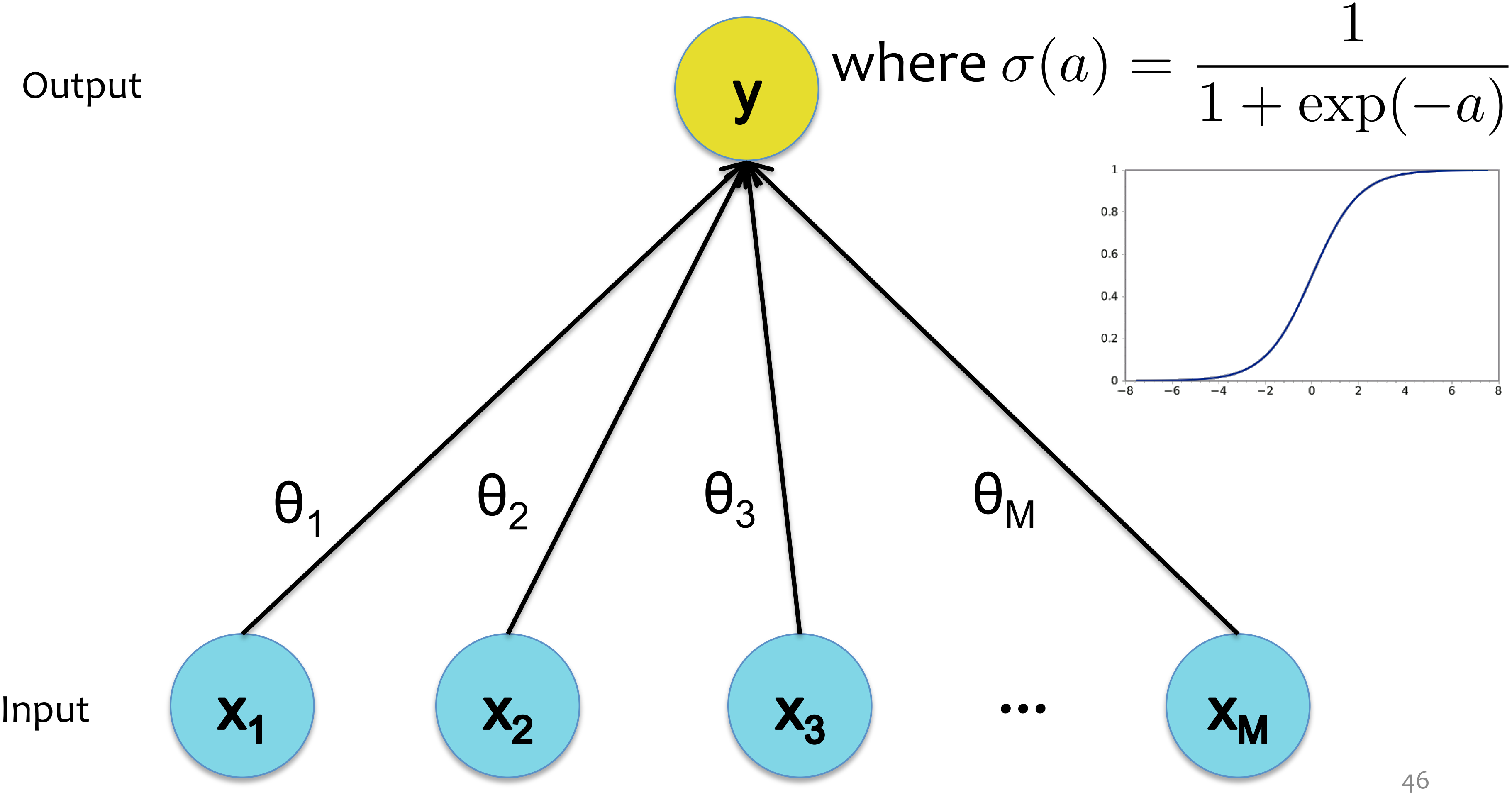
where $\sigma(a) = a$



Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

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

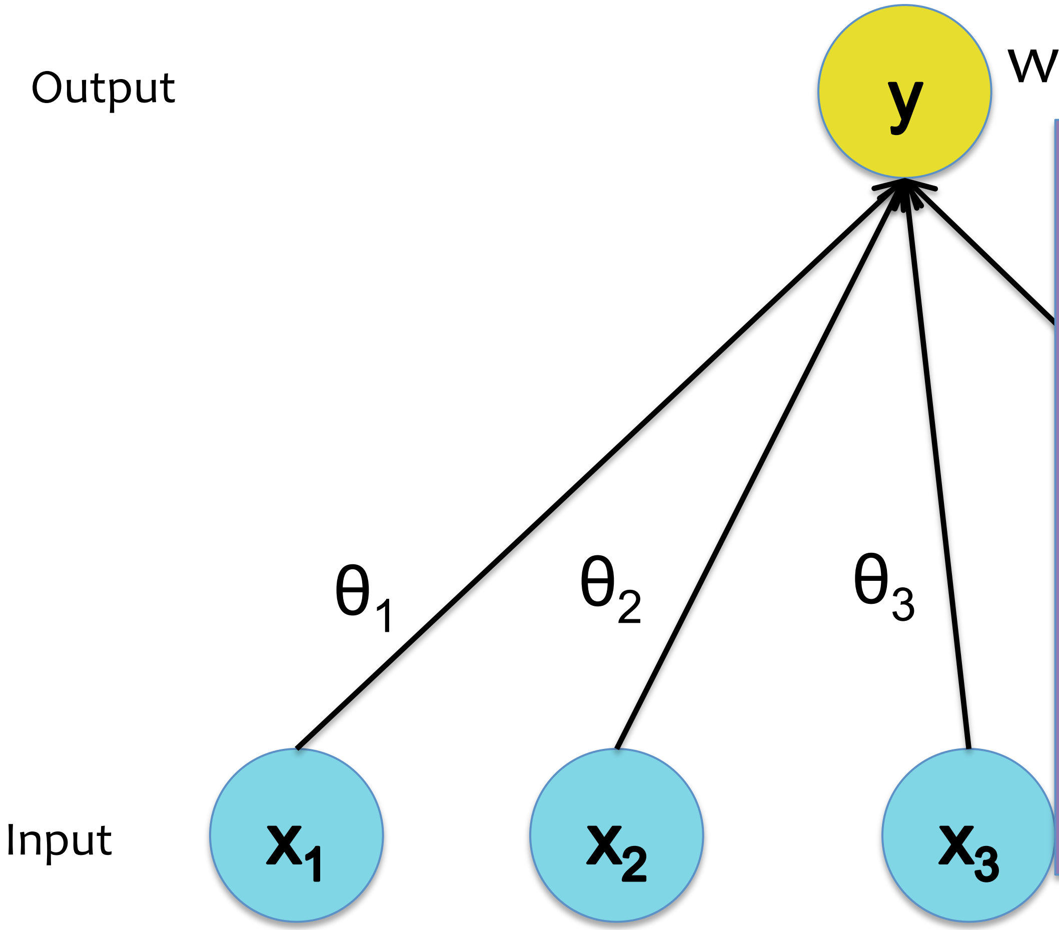


Decision Functions

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

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



Face

Face

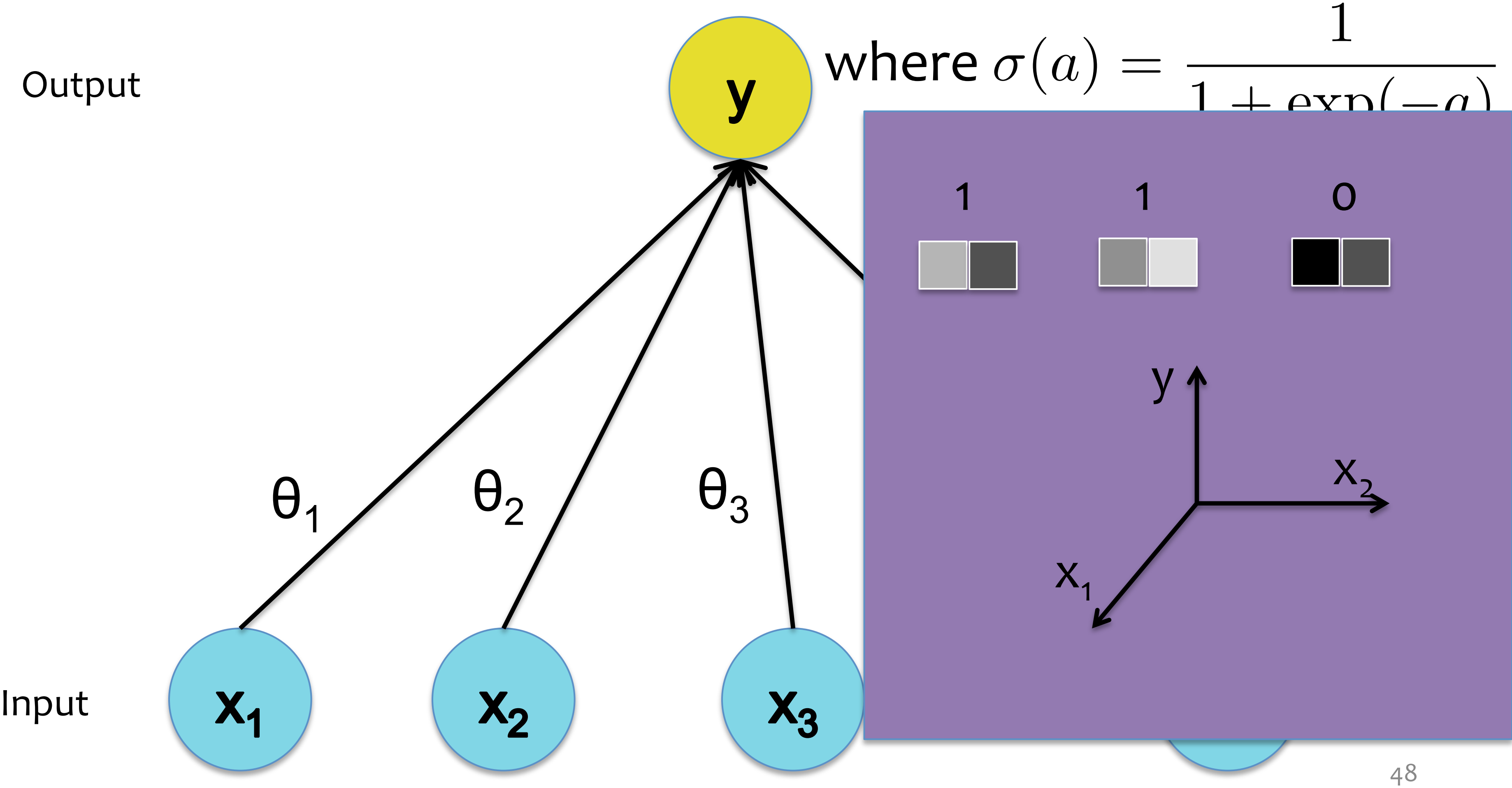
Not a face

Decision Functions

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$

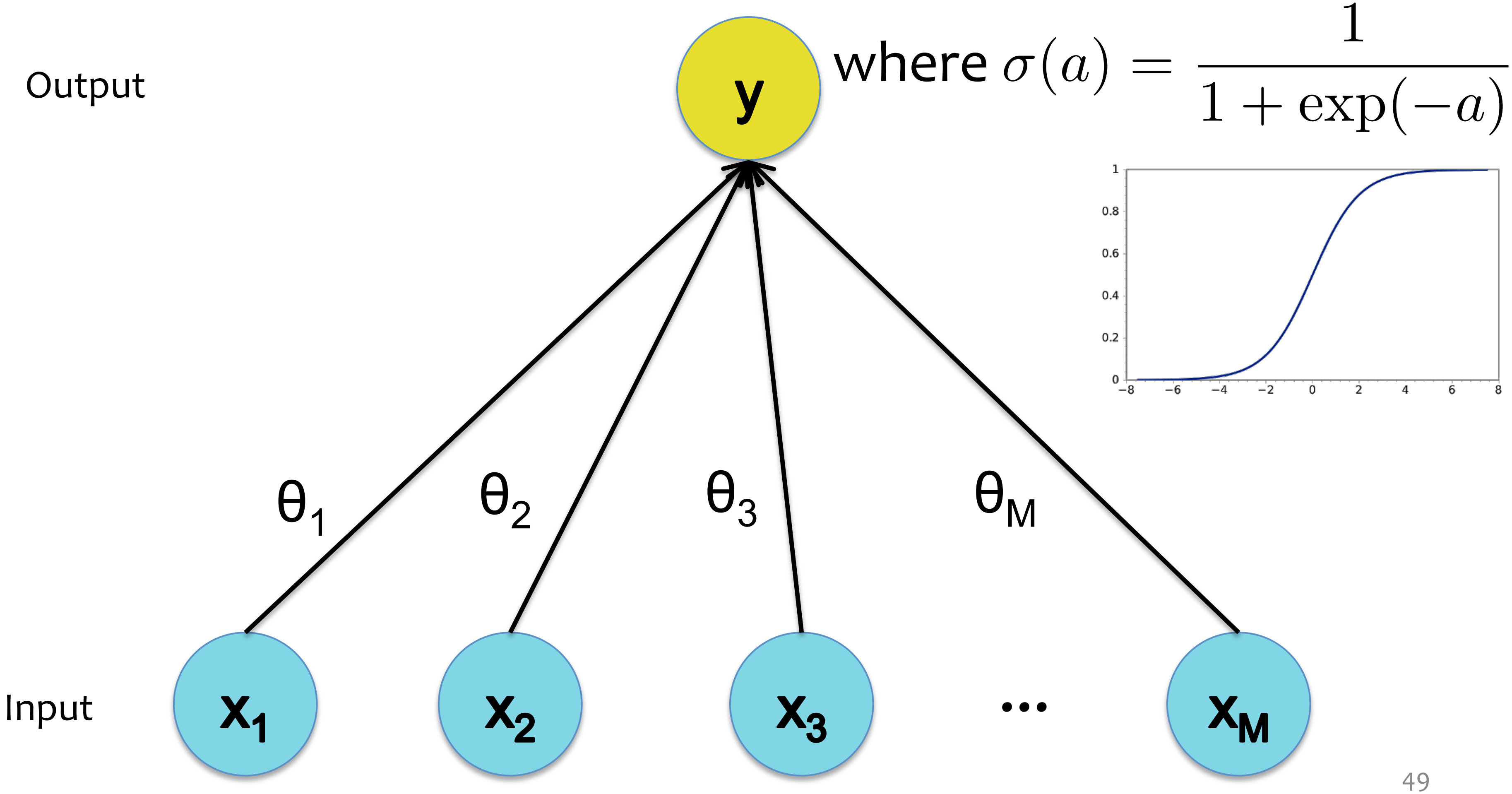


Decision Functions

Logistic Regression

$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



Neural Network Model

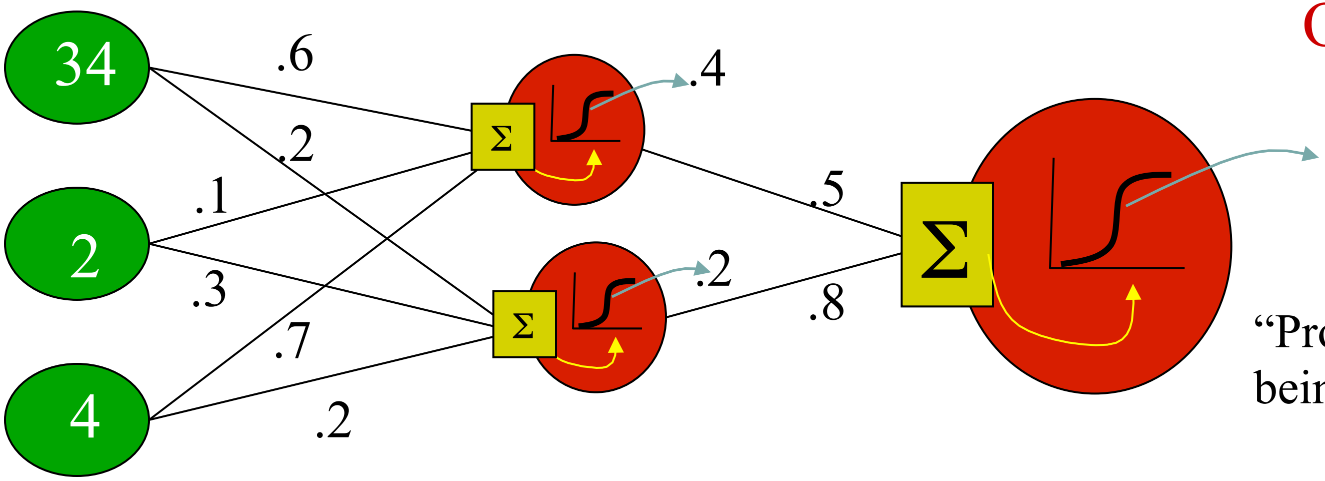
Inputs

Output

Age

Gender

Stage



0.6

"Probability of beingAlive"

Independent variables

Weights

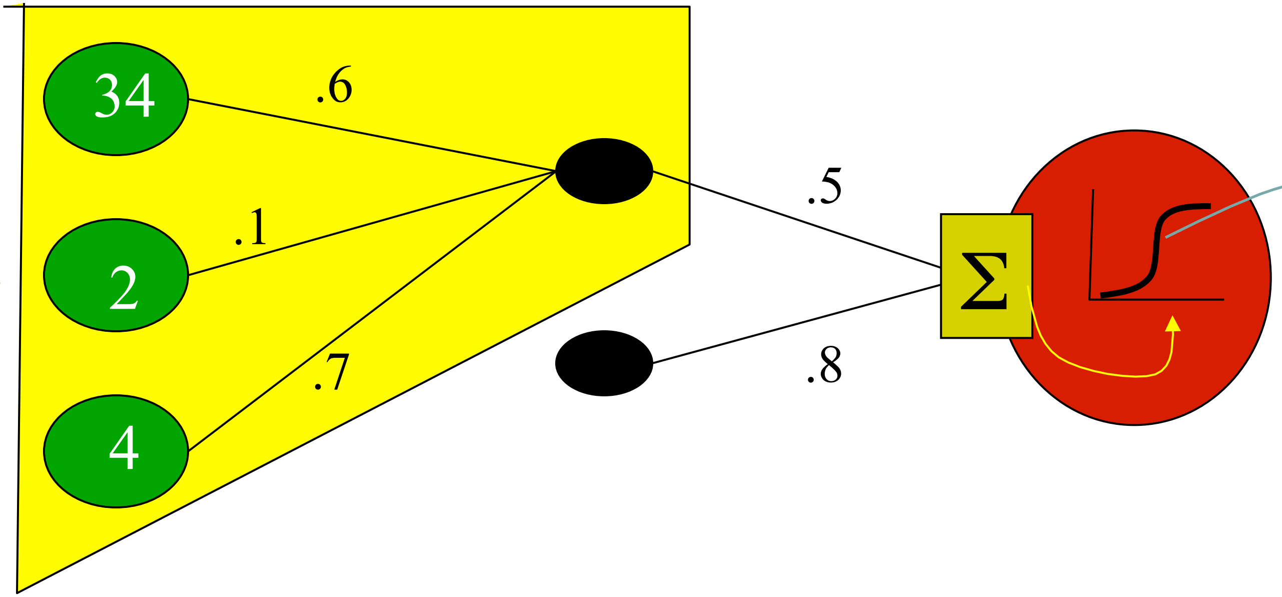
HiddenLayer

Weights

Dependent variable

Prediction

“Combined logistic models”



Inputs

Age

34

.6

Gender

2

.1

Stage

4

.7

.5

.8

Σ

Output

0.6

“Probability of beingAlive”

Independent variables

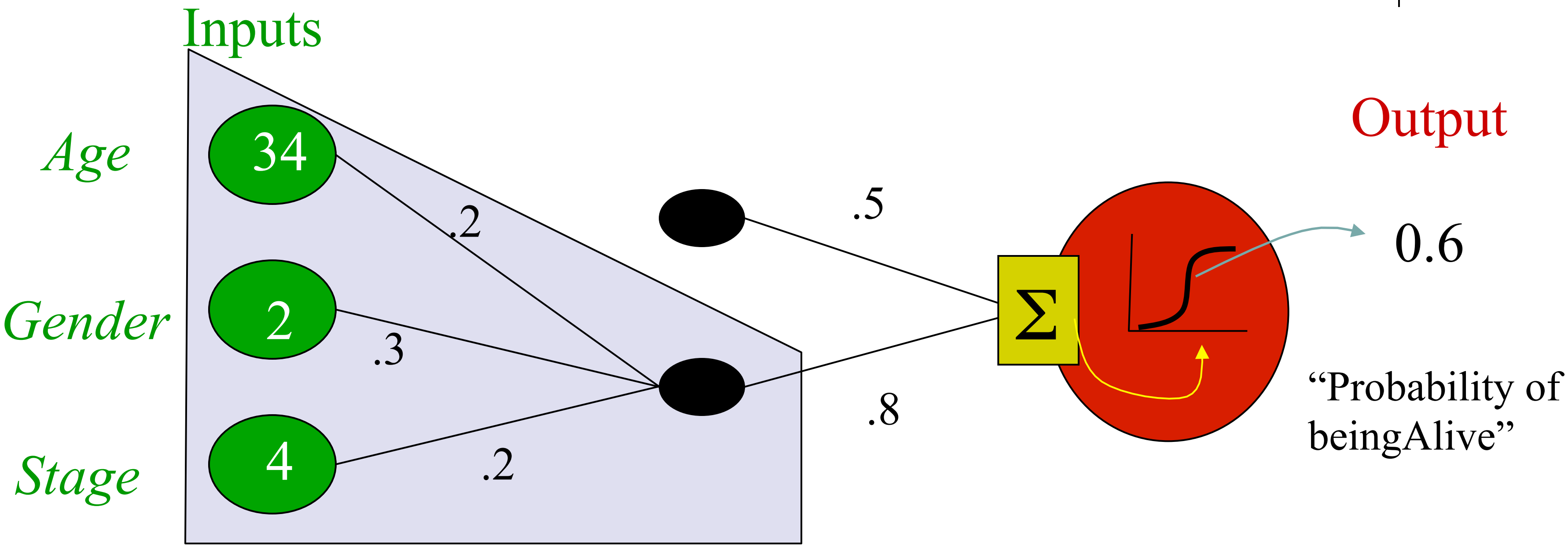
Weights

HiddenLayer

Weights

Dependent variable

Prediction



Inputs

Age

34

Gender

2

Stage

4

.2

.3

.2

.5

.8

Σ

Output

0.6

“Probability of beingAlive”

Independent variables

Weights

HiddenLayer

Weights

Dependent variable

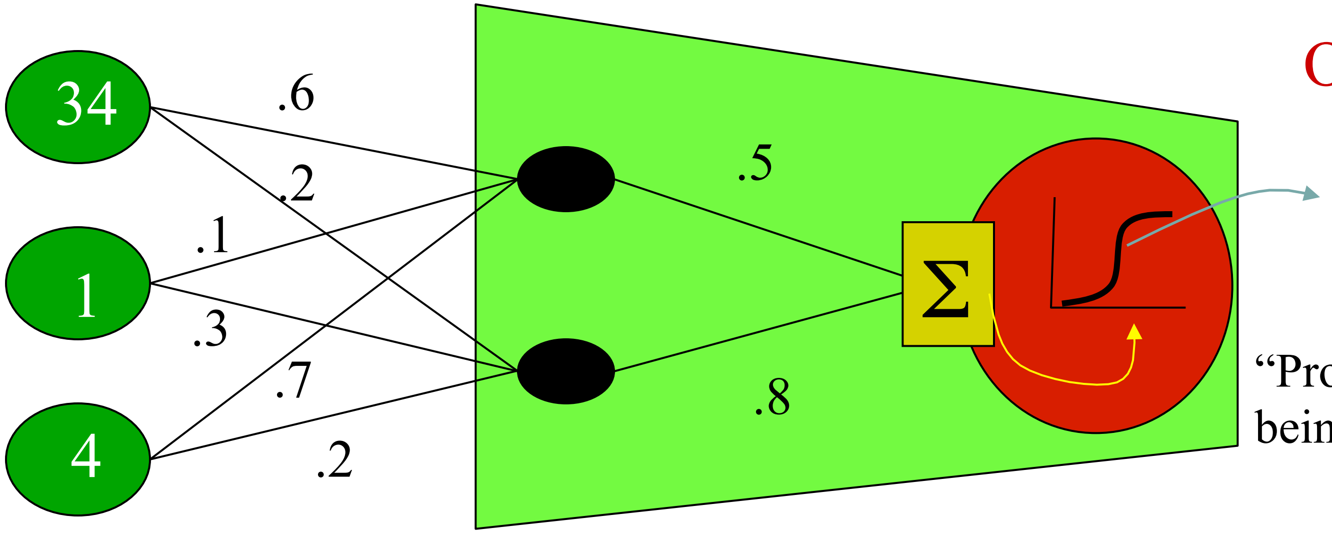
Prediction

Inputs

Age

Gender

Stage



Output

0.6

“Probability of beingAlive”

Independent variables

Weights

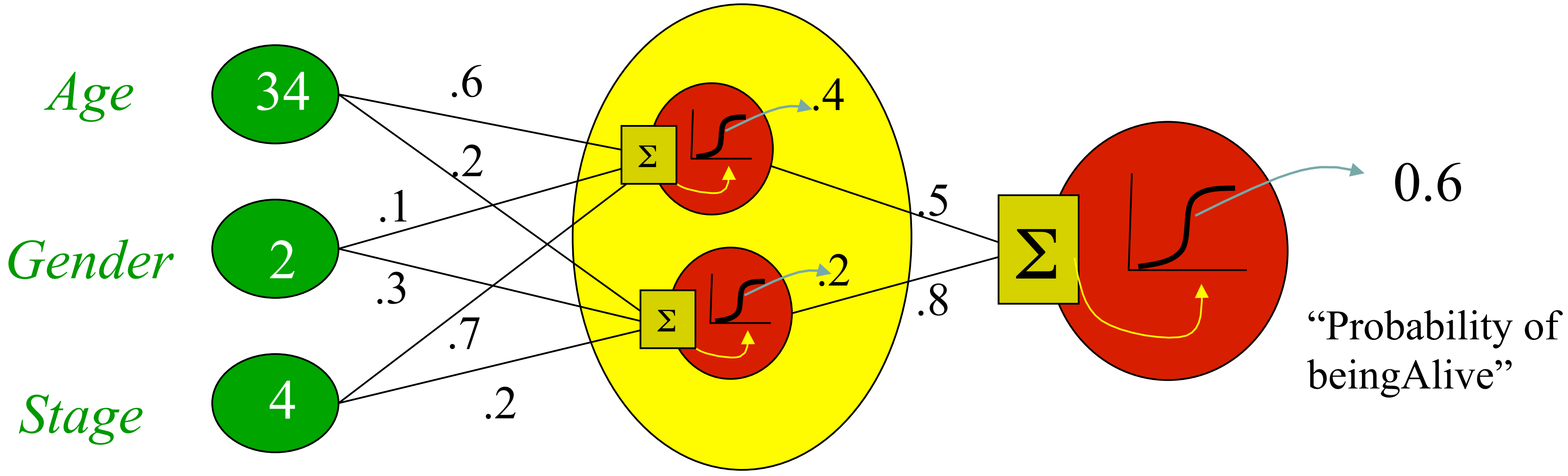
HiddenLayer

Weights

Dependent variable

Prediction

Not really,
no target for hidden units...



Independent variables

Weights

HiddenLayer

Weights

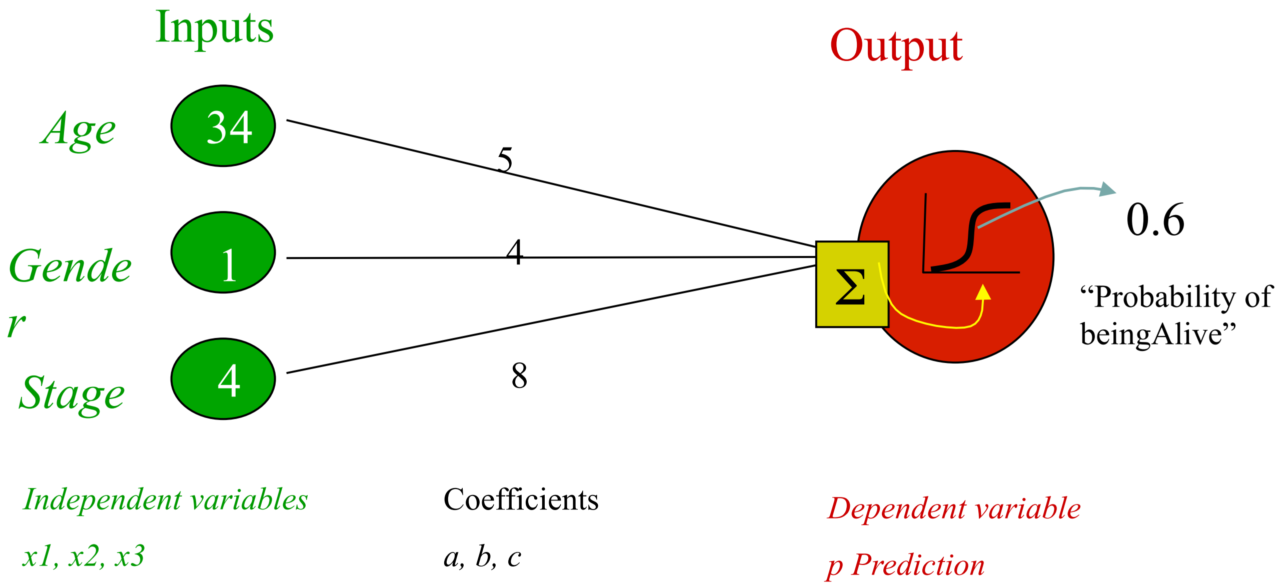
Dependent variable

Prediction

Jargon Pseudo-Correspondence

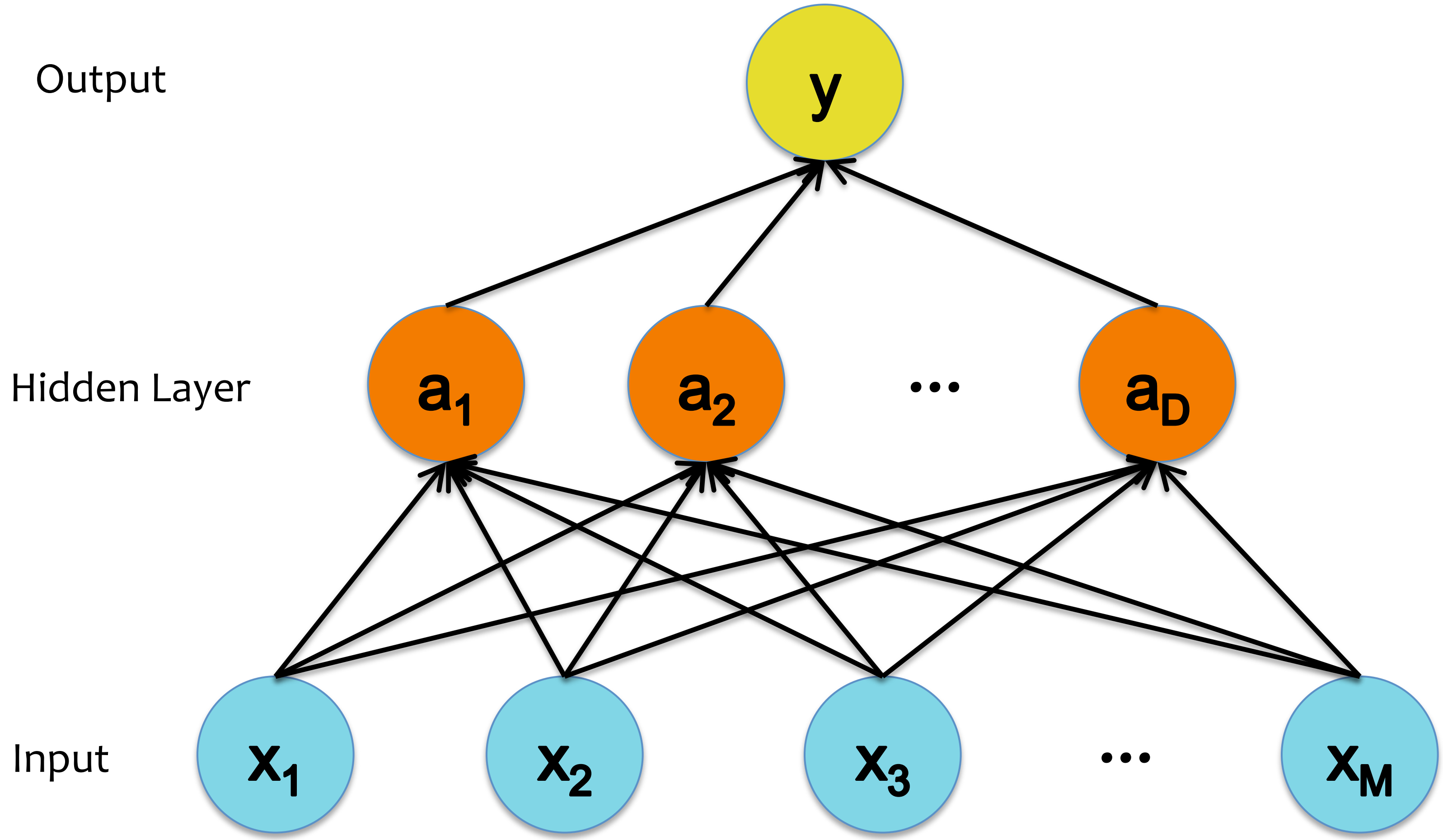
- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = "weights"
- Estimates = "targets"

Logistic Regression Model (the sigmoid unit)



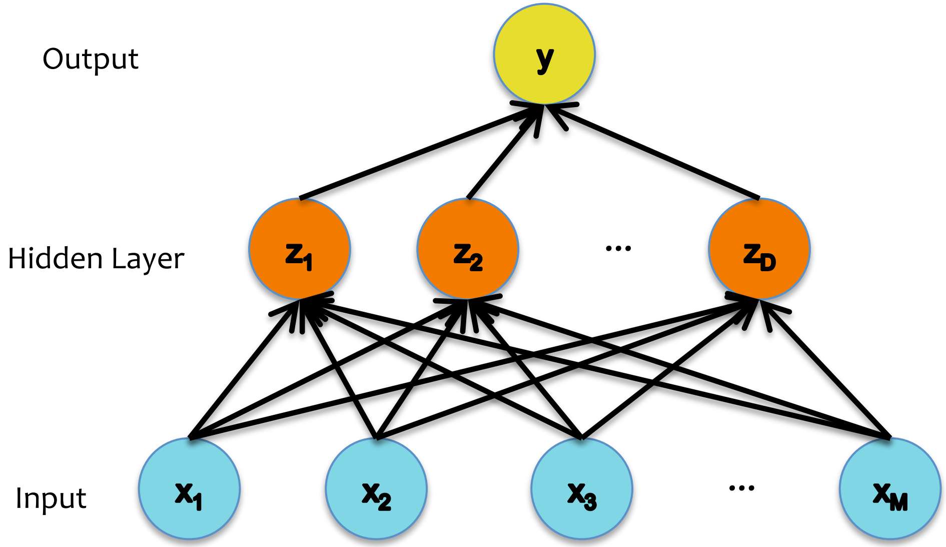
Decision Functions

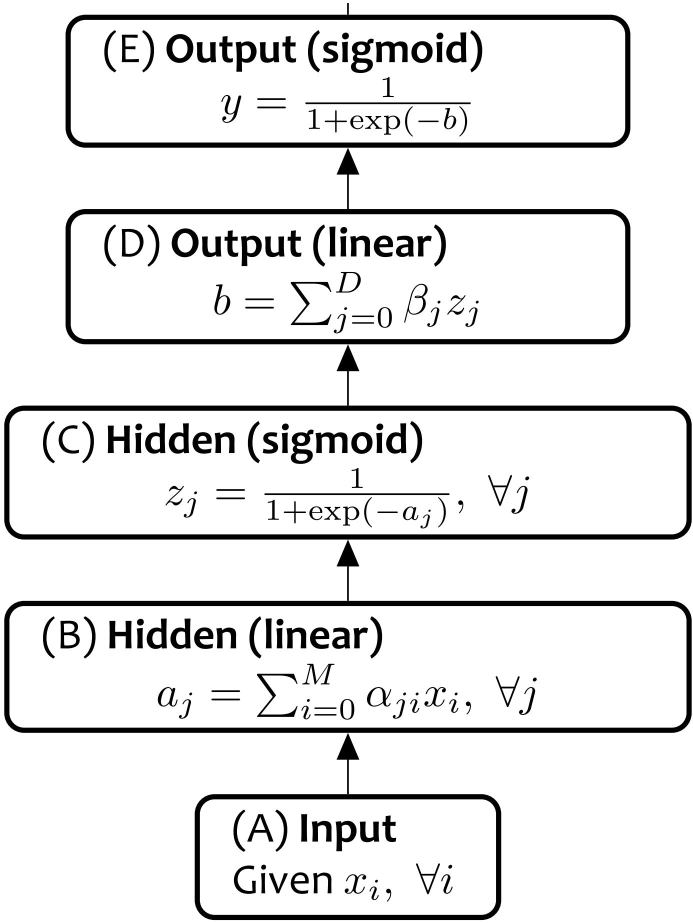
Neural Network



Decision Functions

Neural Network





(E) **Output (sigmoid)**
$$y = \frac{1}{1+\exp(-b)}$$

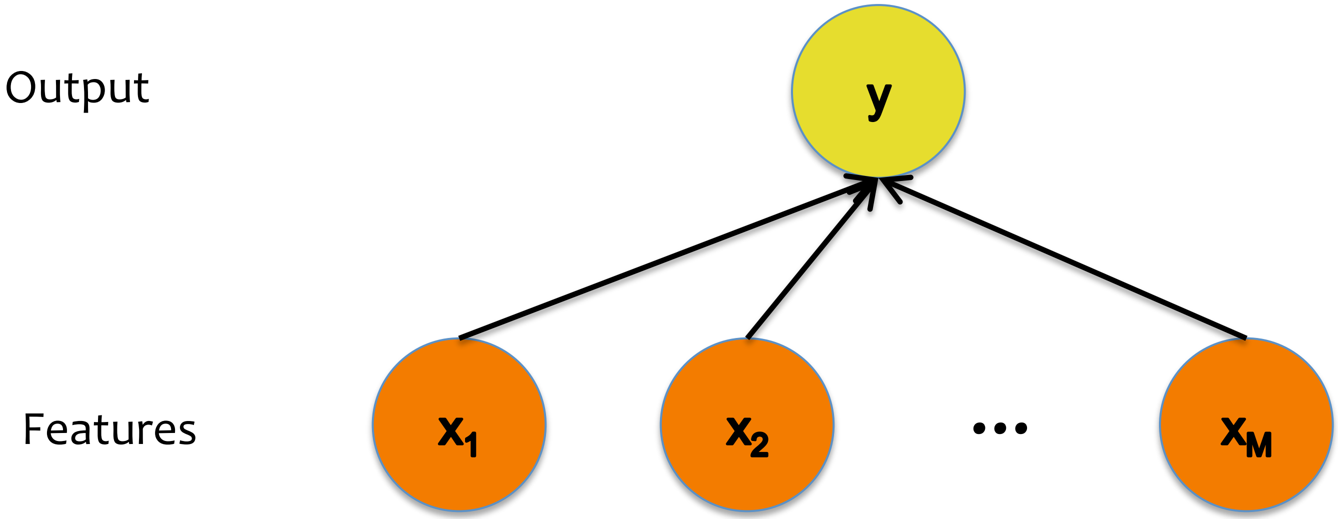
(D) **Output (linear)**
$$b = \sum_{j=0}^{D} \beta_j z_j$$

(C) **Hidden (sigmoid)**
$$z_j = \frac{1}{1+\exp(-a_j)}, \;\; \forall j$$

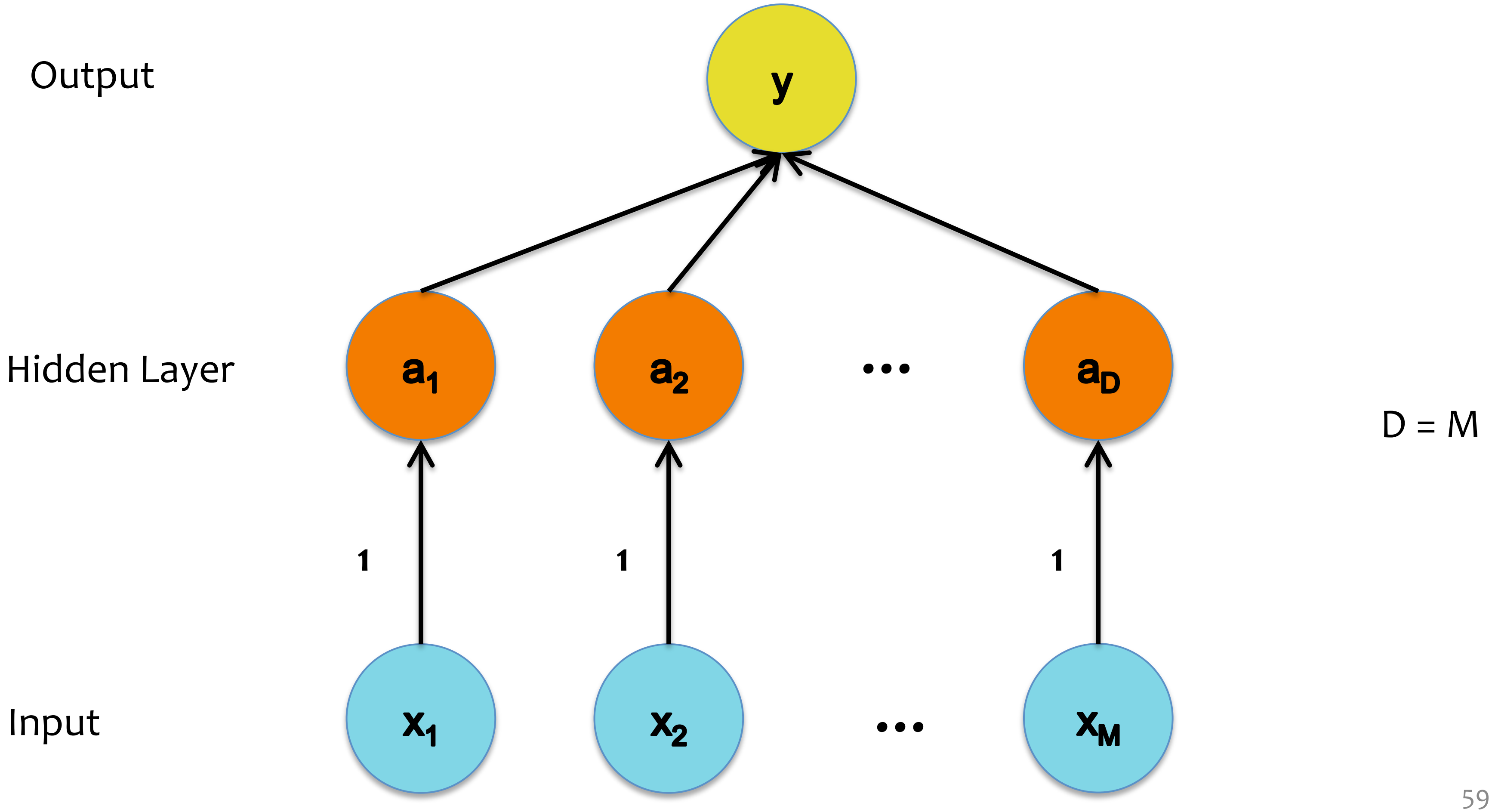
(B) **Hidden (linear)**
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \;\; \forall j$$

(A) **Input**
Given $x_i, \;\; \forall i$

Building a Neural Net

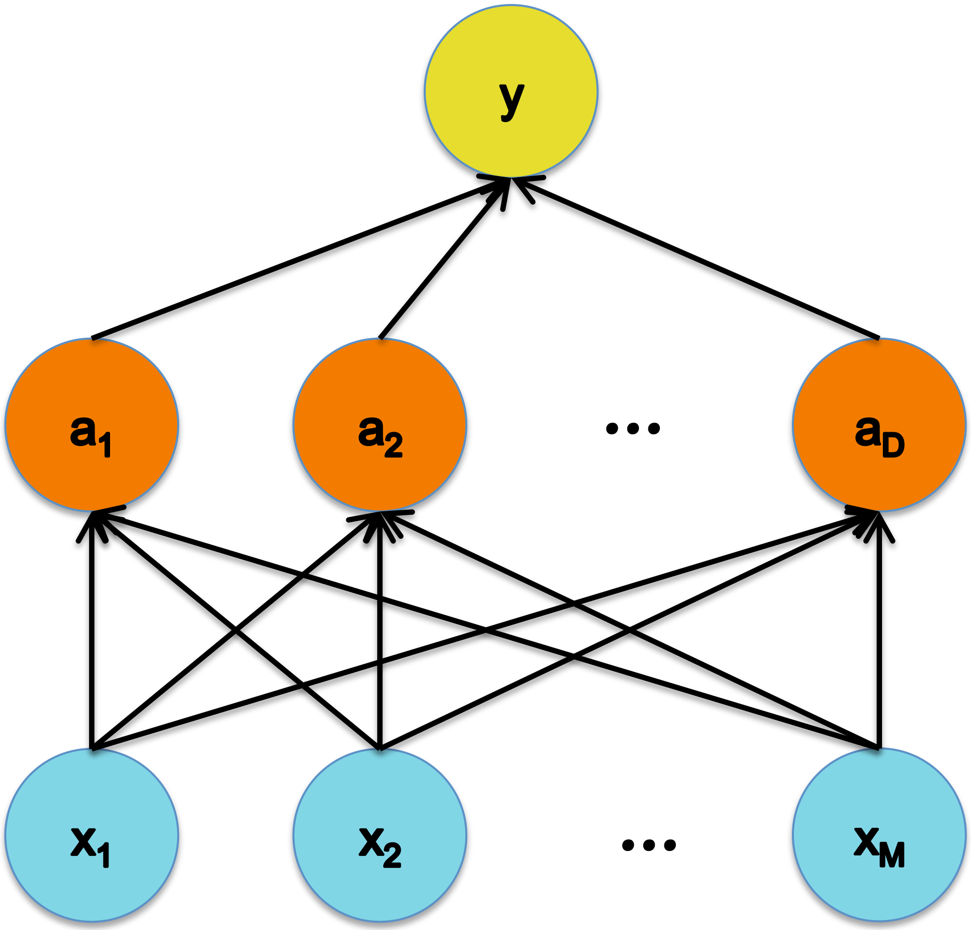


Building a Neural Net



D = M

Building a Neural Net



Output

Hidden Layer

Input

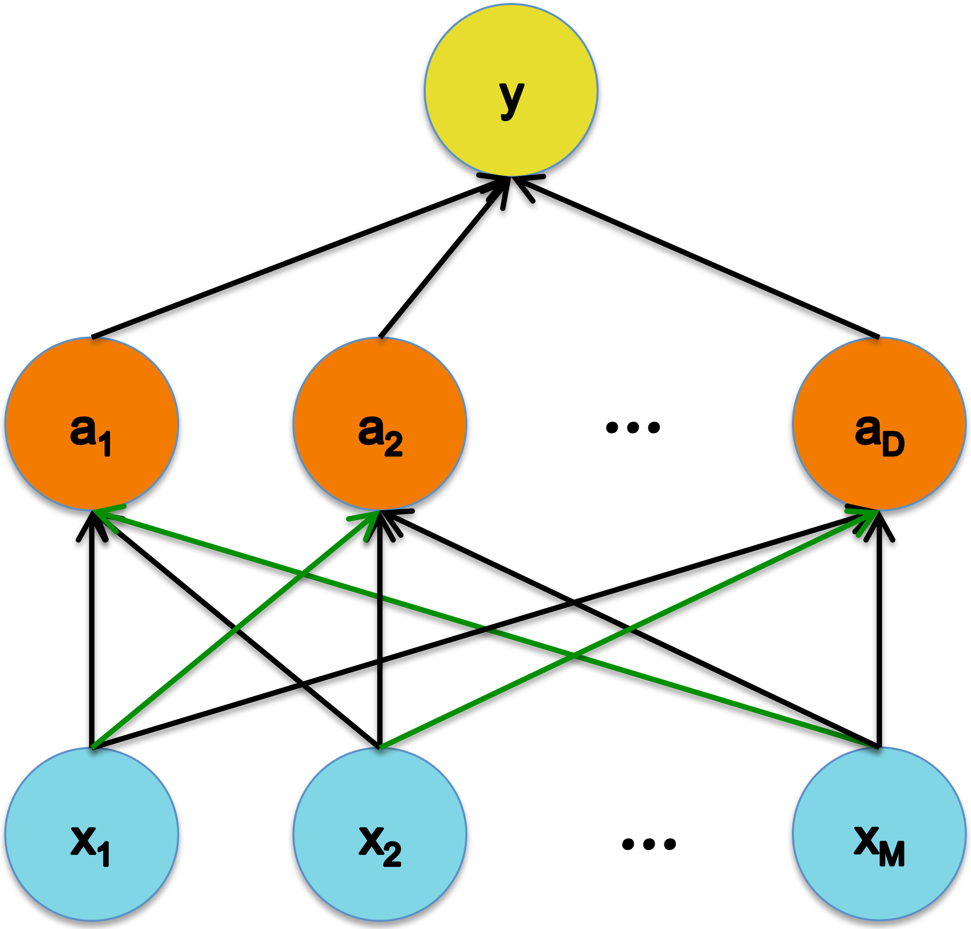
D = M

Building a Neural Net

Output

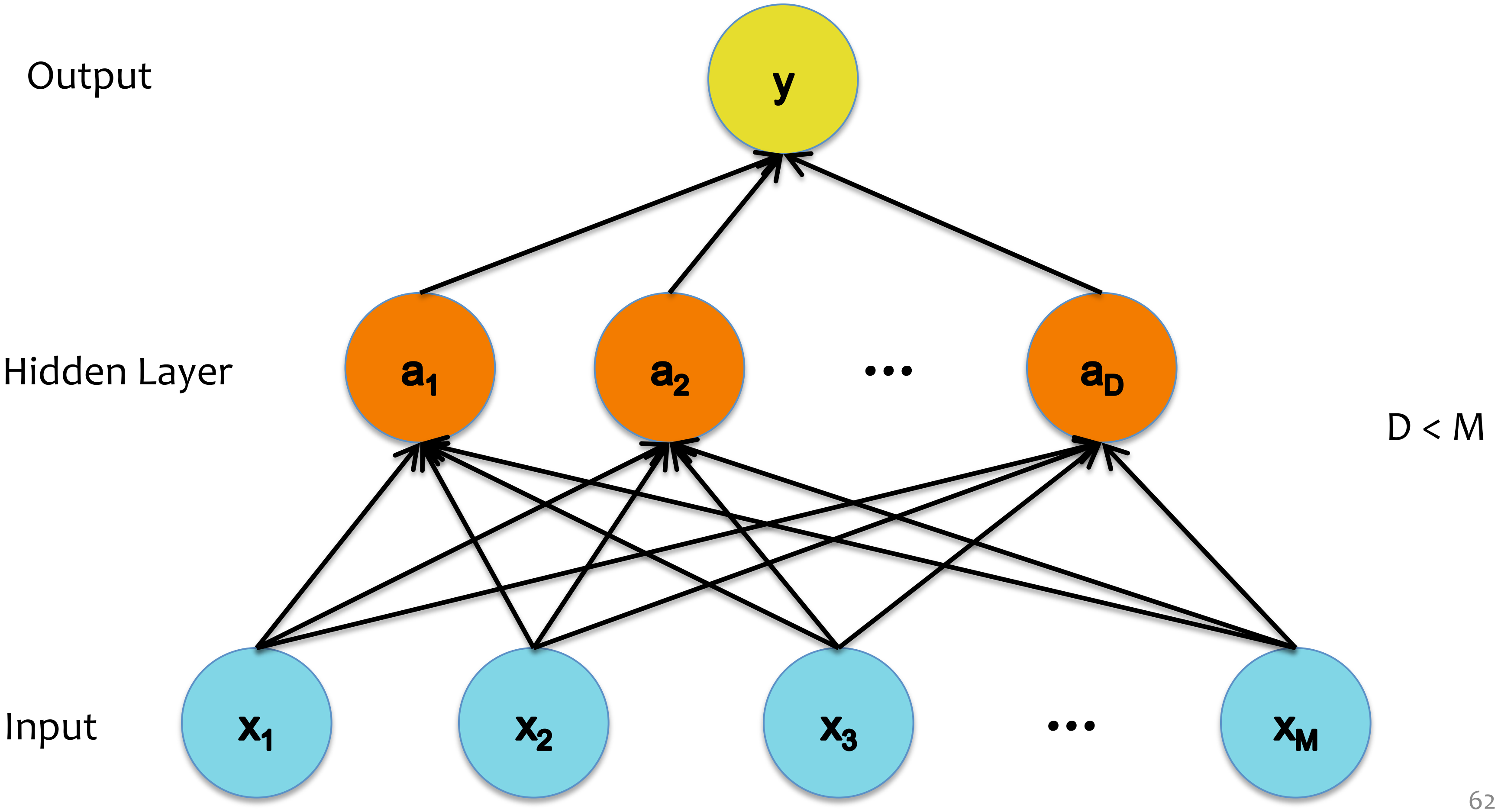
Hidden Layer

Input



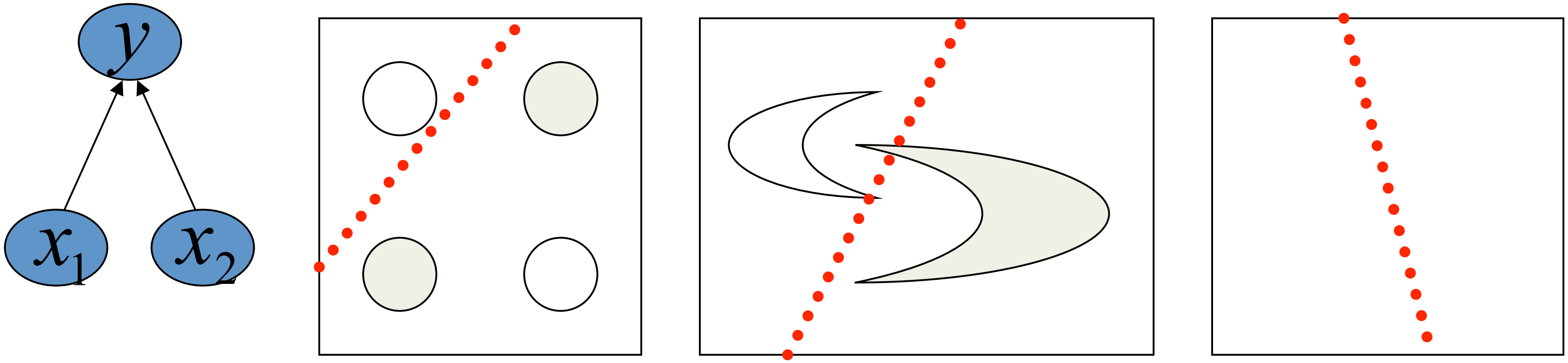
D = M

Building a Neural Net



Decision Boundary

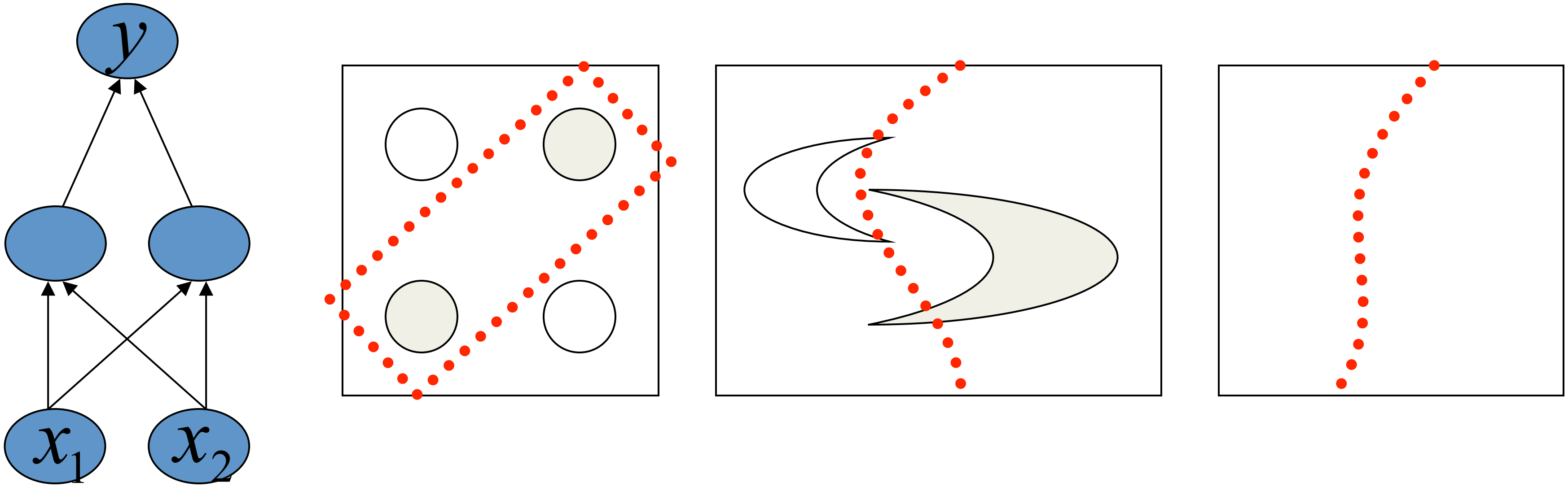
- 0 hidden layers: linear classifier
  - Hyperplanes



Example from to Eric Postma via Jason Eisner

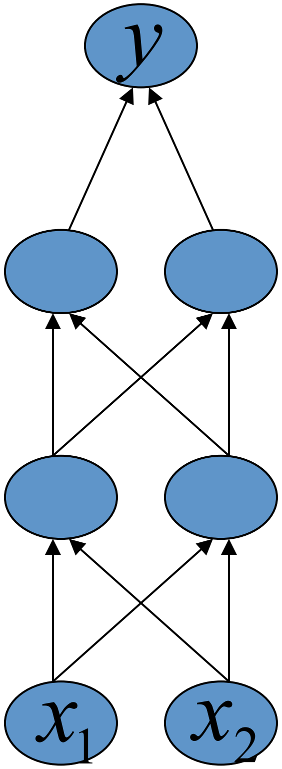
Decision Boundary

- 1 hidden layer
  - – Boundary of convex region (open or closed)

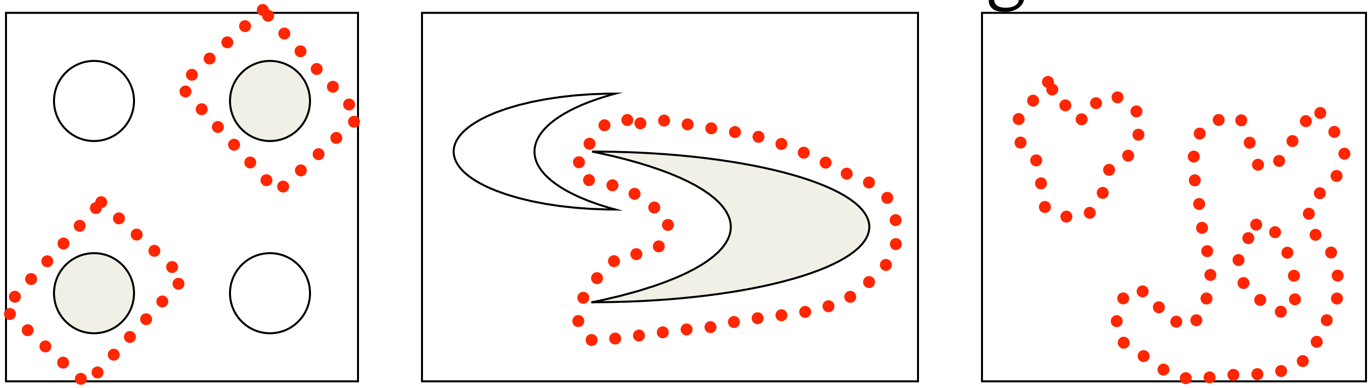


Example from to Eric Postma via Jason Eisner

Decision Boundary



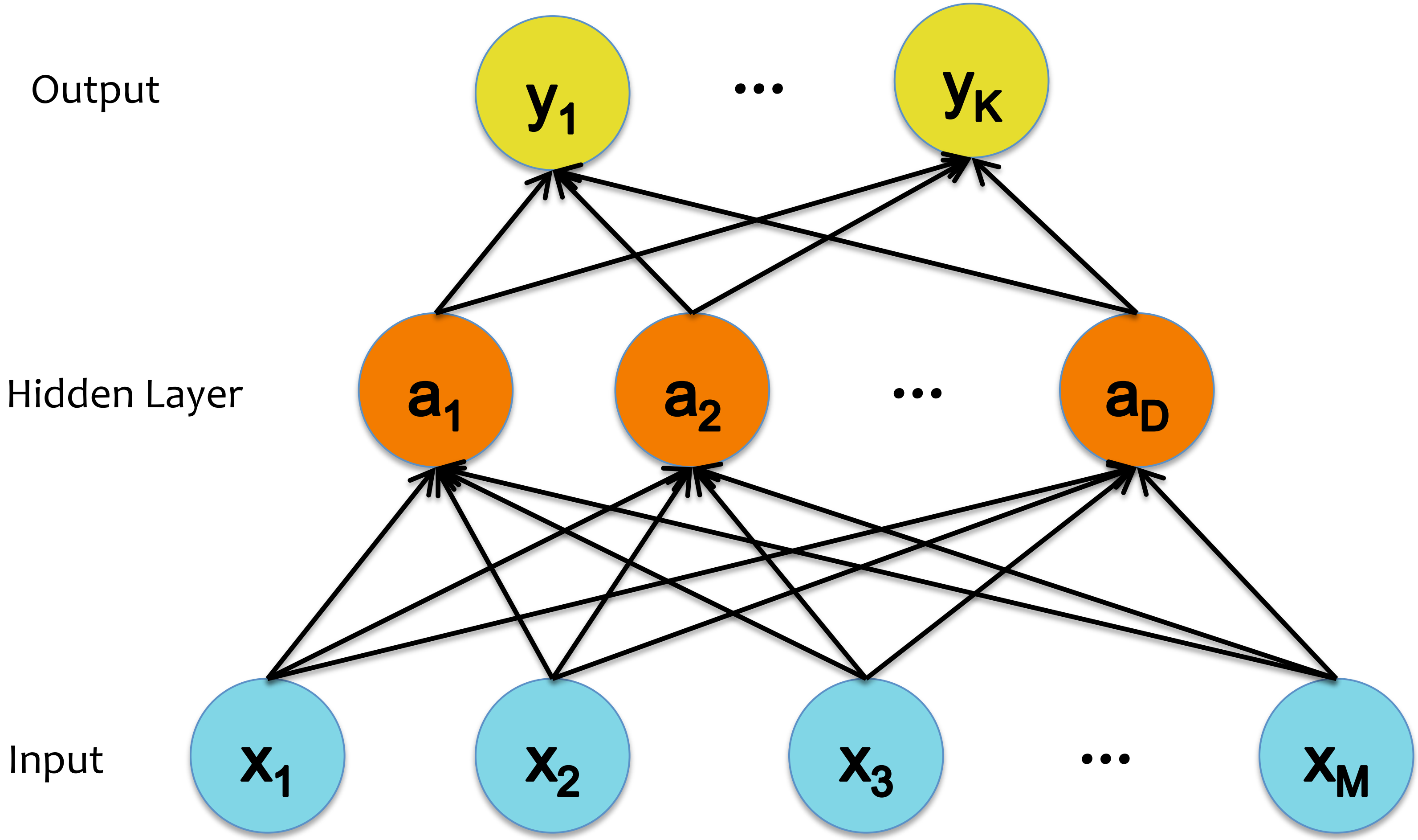
- 2 hidden layers
  - Combinations of convex regions



Example from to Eric Postma via Jason Eisner

Decision Functions

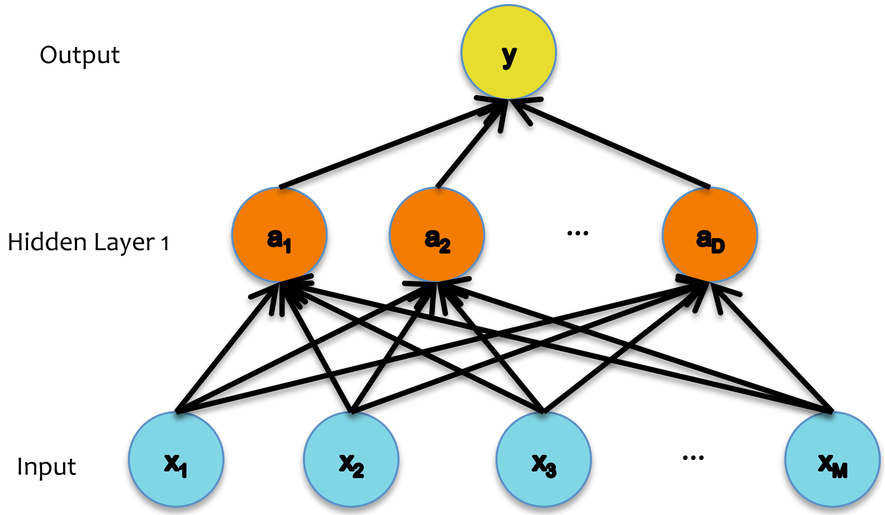
Multi-Class Output



Decision Functions

Deeper Networks

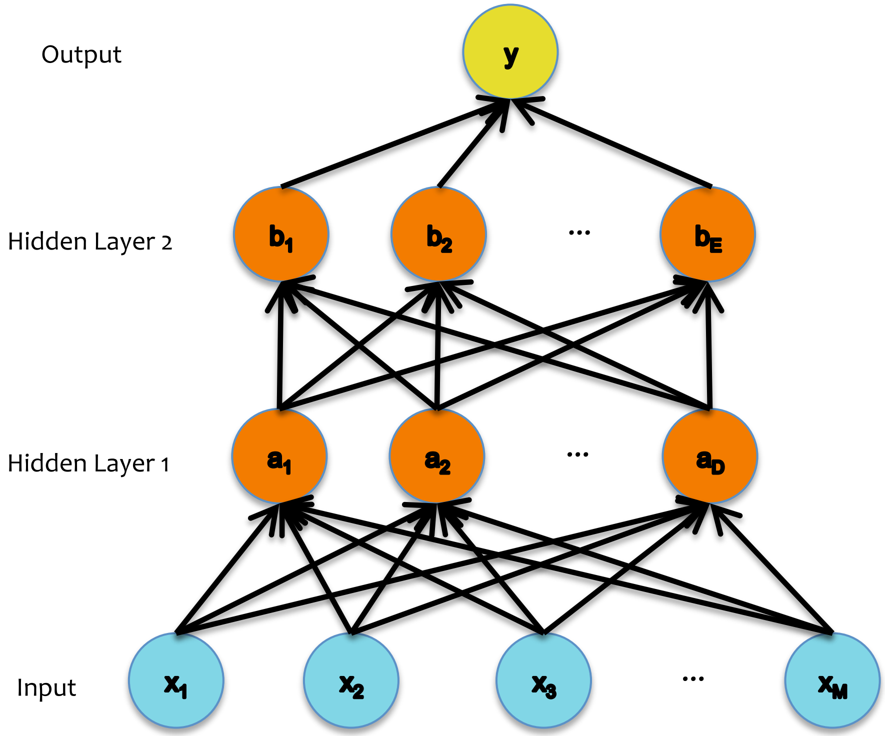
Next lecture:



Decision Functions

Deeper Networks

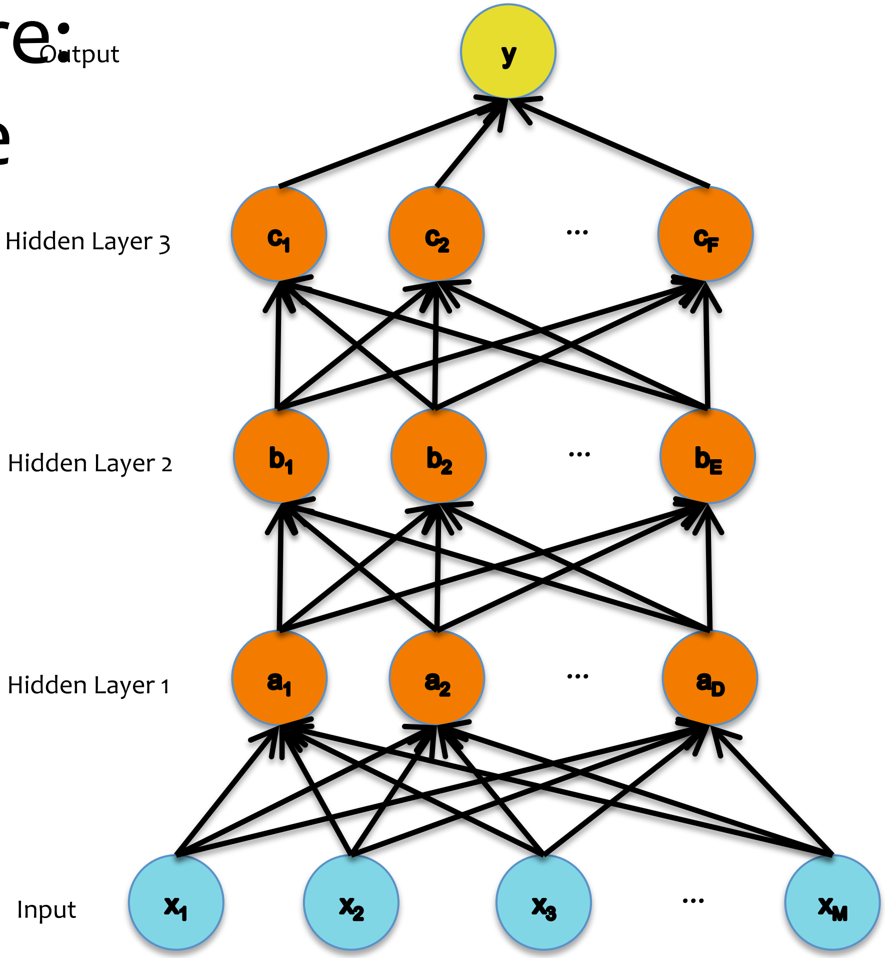
Next lecture:



Decision Functions

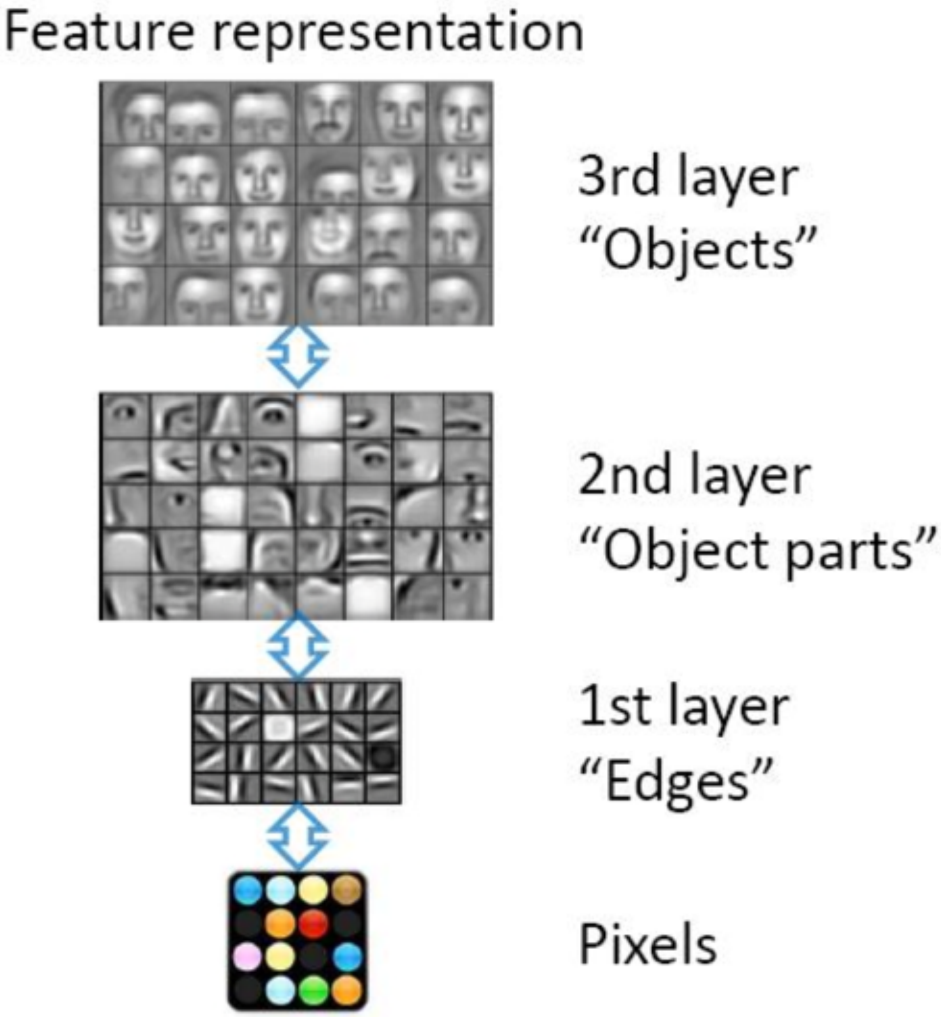
Deeper Networks

Next lecture: Making the neural networks deeper



Different Levels of Abstraction

- We don't know the "right" levels of abstraction
- So let the model figure it out!

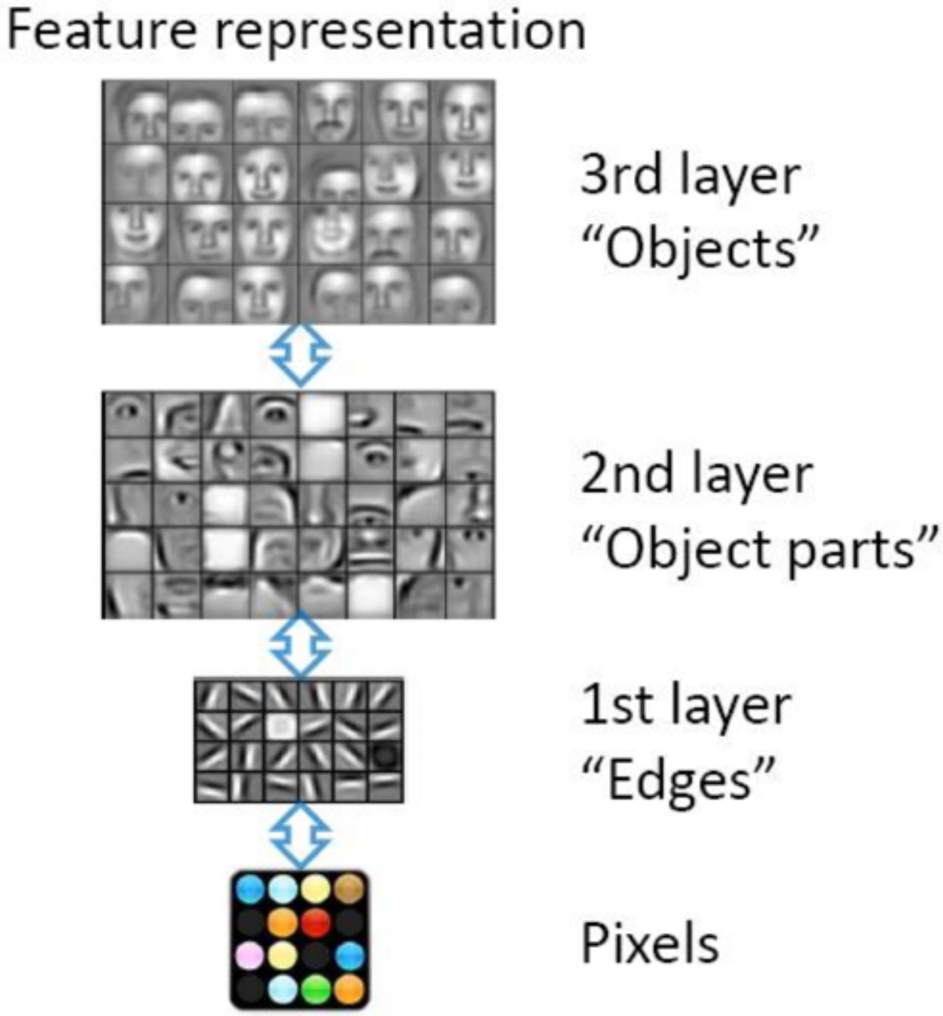


Example from Honglak Lee (NIPS 2010)

Different Levels of Abstraction

Face Recognition:

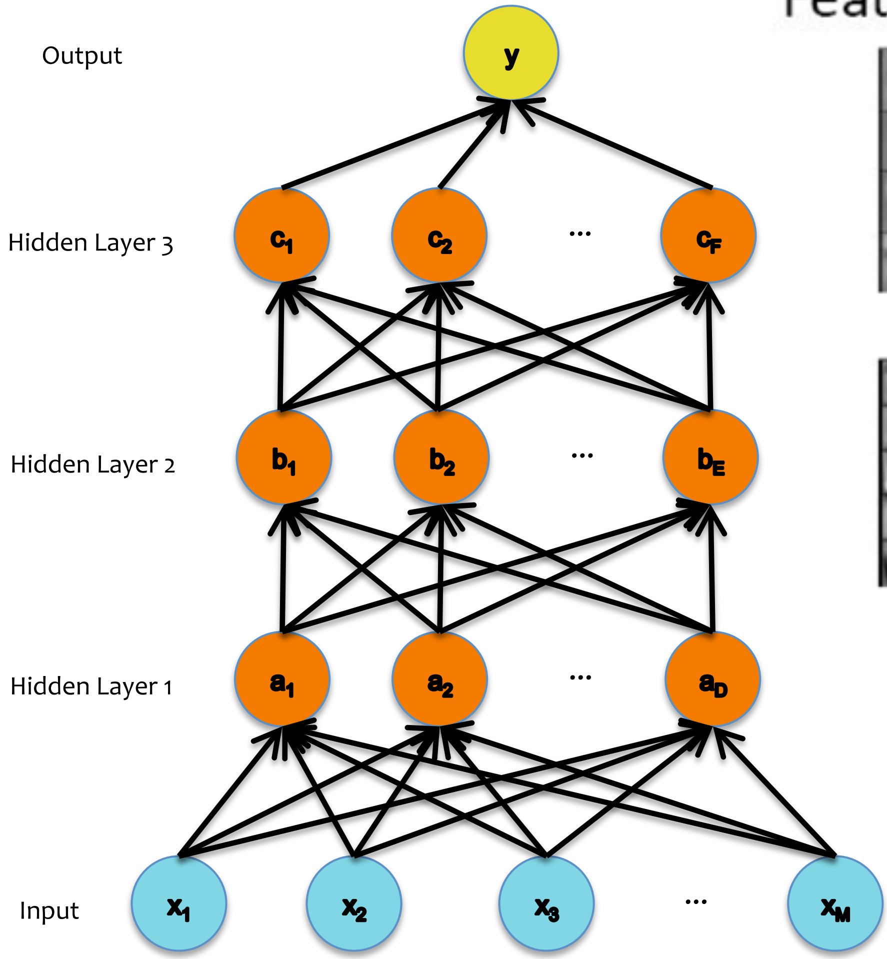
- Deep Network can build up increasingly higher levels of abstraction
- Lines, parts, regions



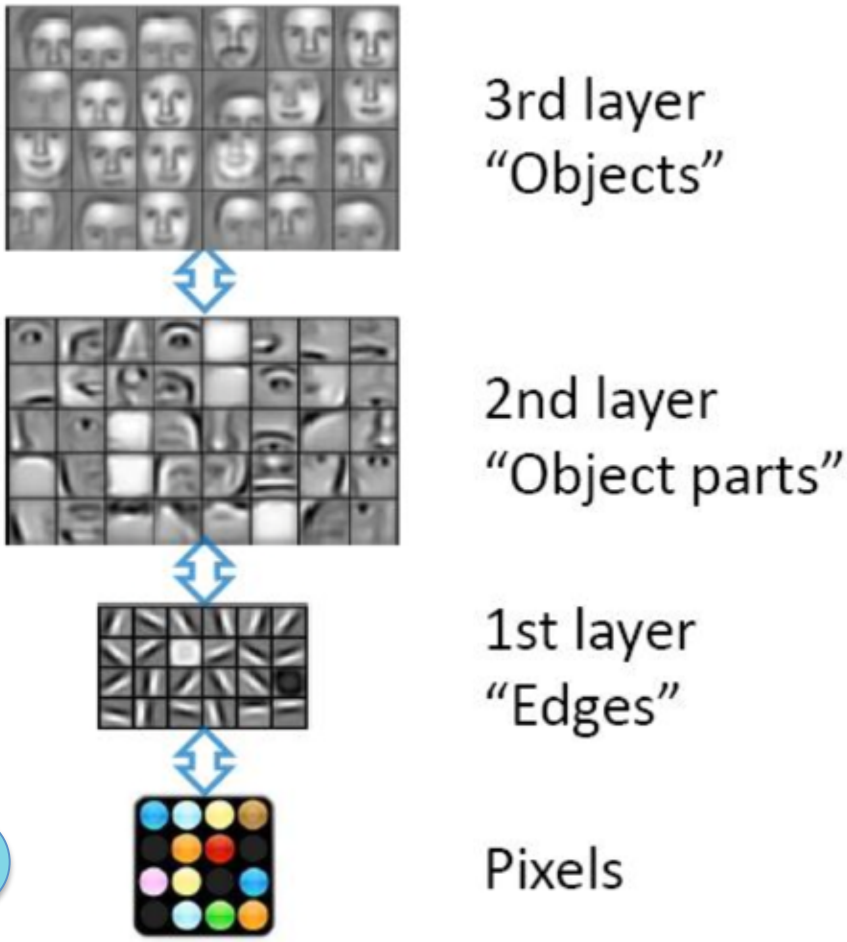
Example from Honglak Lee (NIPS 2010)

Decision Functions

Different Levels of Abstraction



Feature representation



Example from Honglak Lee (NIPS 2010)

ARCHITECTURES

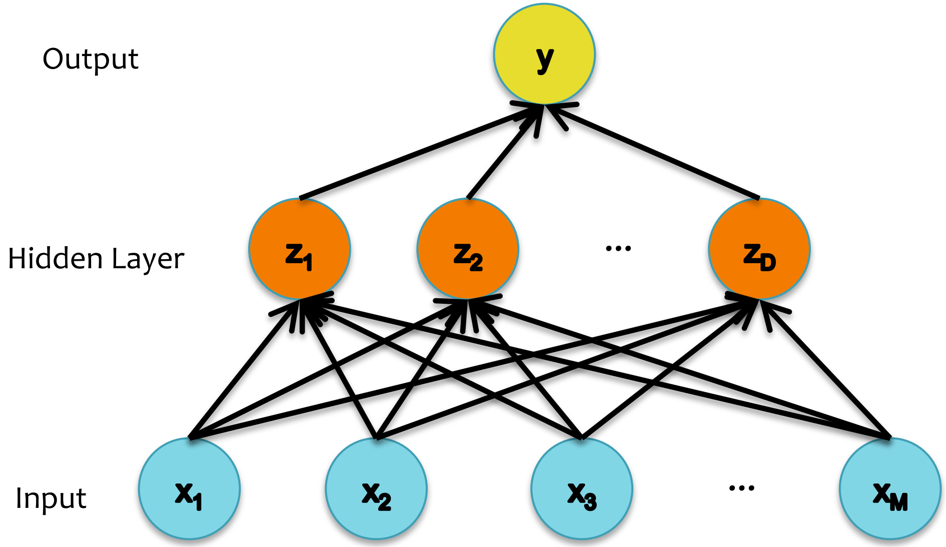
Neural Network Architectures

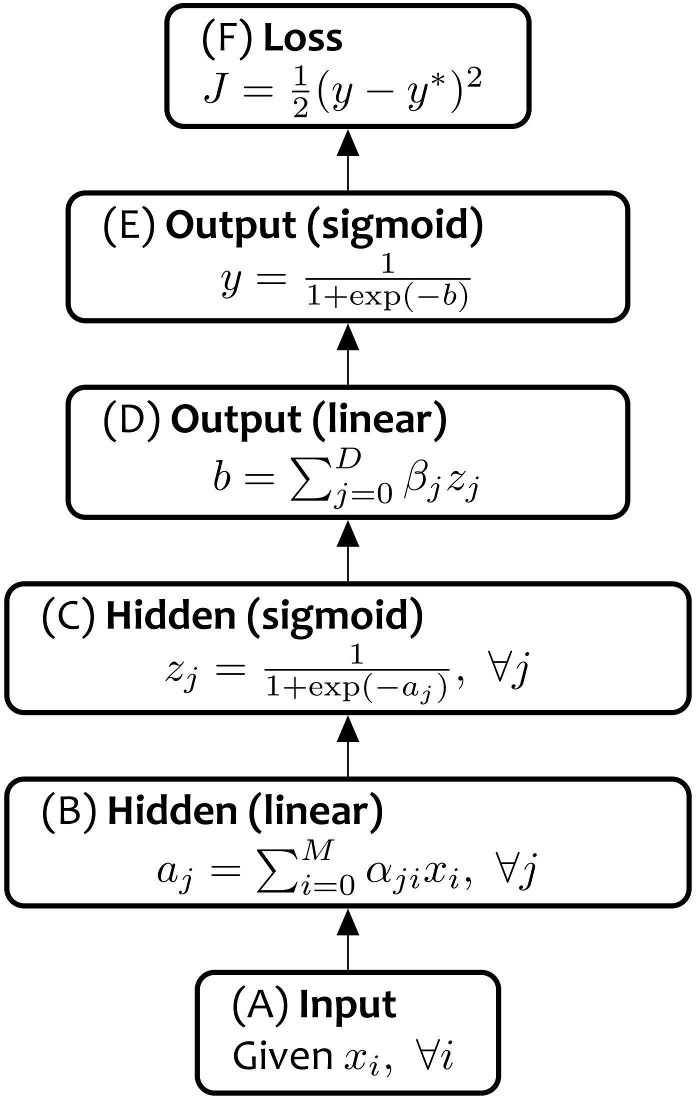
Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function

Activation Functions

Neural Network with sigmoid activation functions





(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

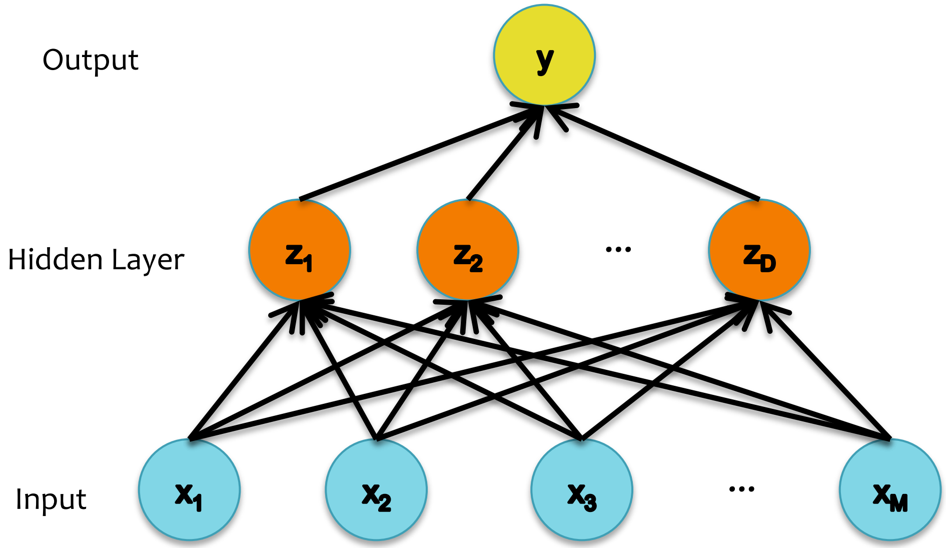
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

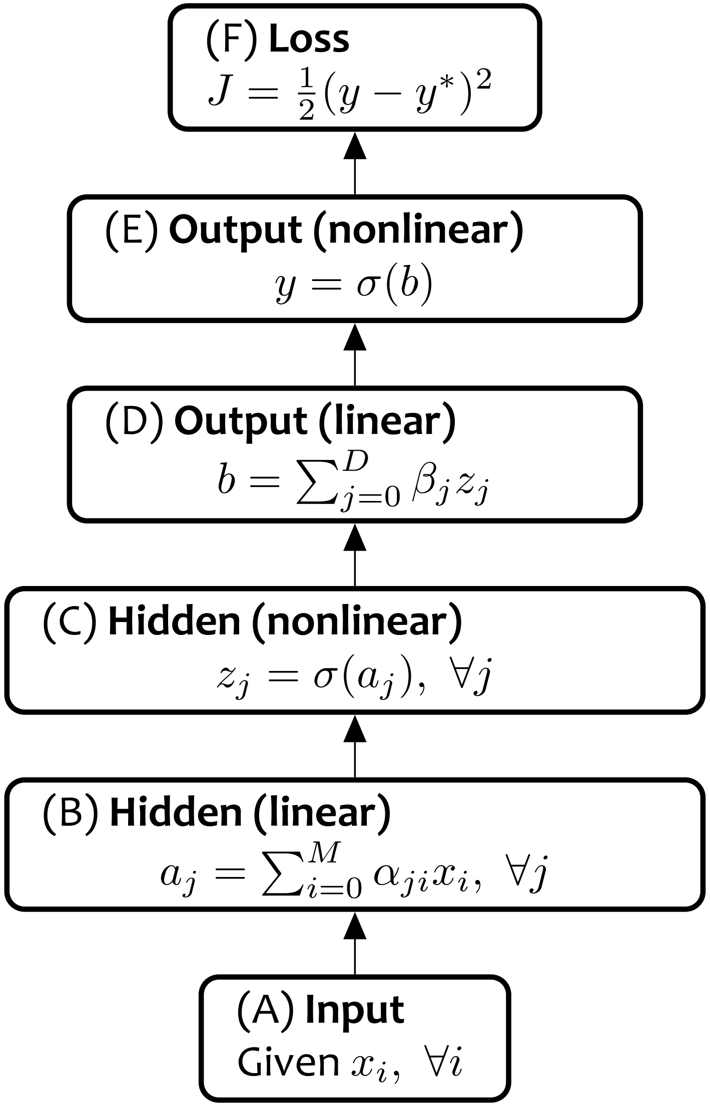
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

Activation Functions

Neural Network with arbitrary nonlinear activation functions





(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (nonlinear)**
$y = \sigma(b)$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

(C) **Hidden (nonlinear)**
$z_j = \sigma(a_j), \ \forall j$

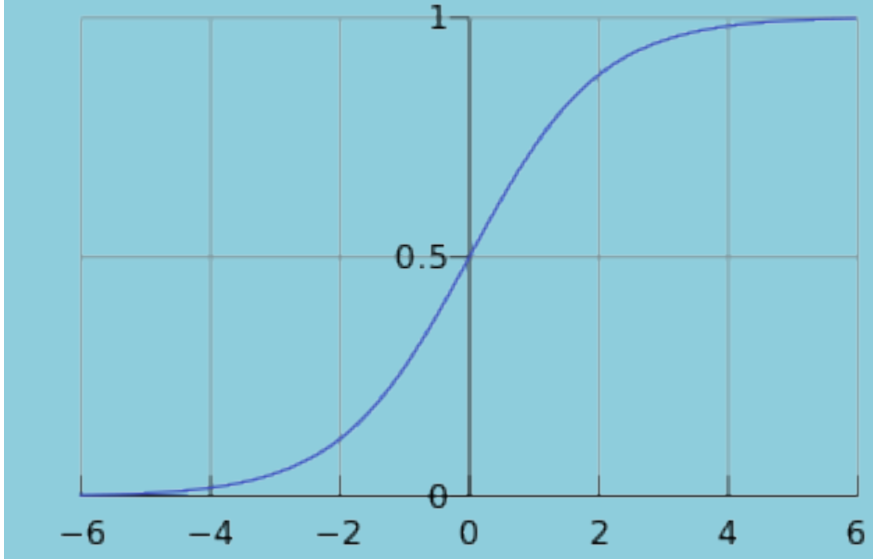
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

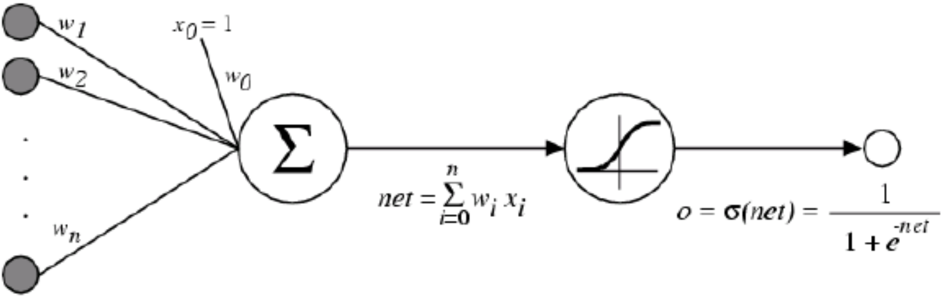
Activation Functions

Sigmoid / Logistic Function

$$\text{logistic}(u) \equiv \frac{1}{1+e^{-u}}$$

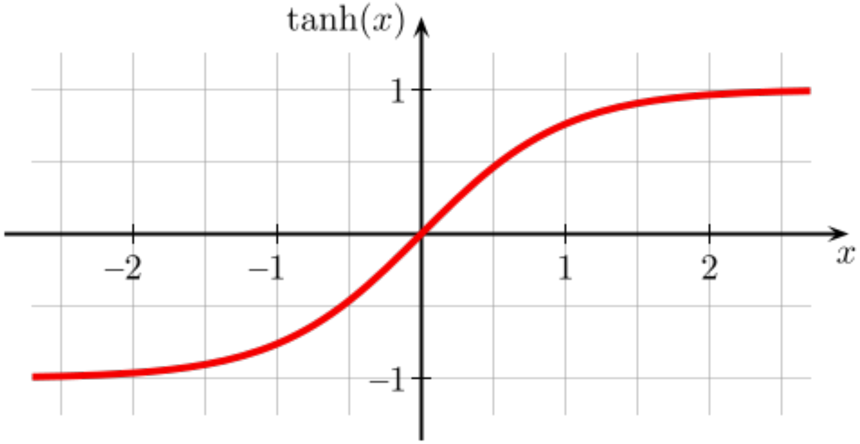


So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…


w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$

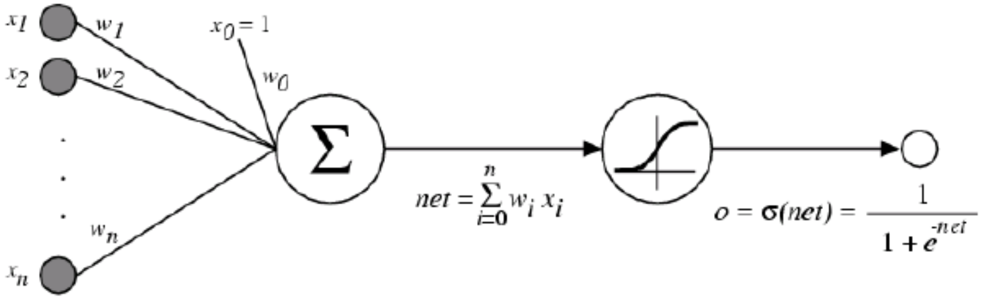
Activation Functions

- A new change: modifying the nonlinearity
  - The logistic is not widely used in modern ANNs



Alternate 1: tanh

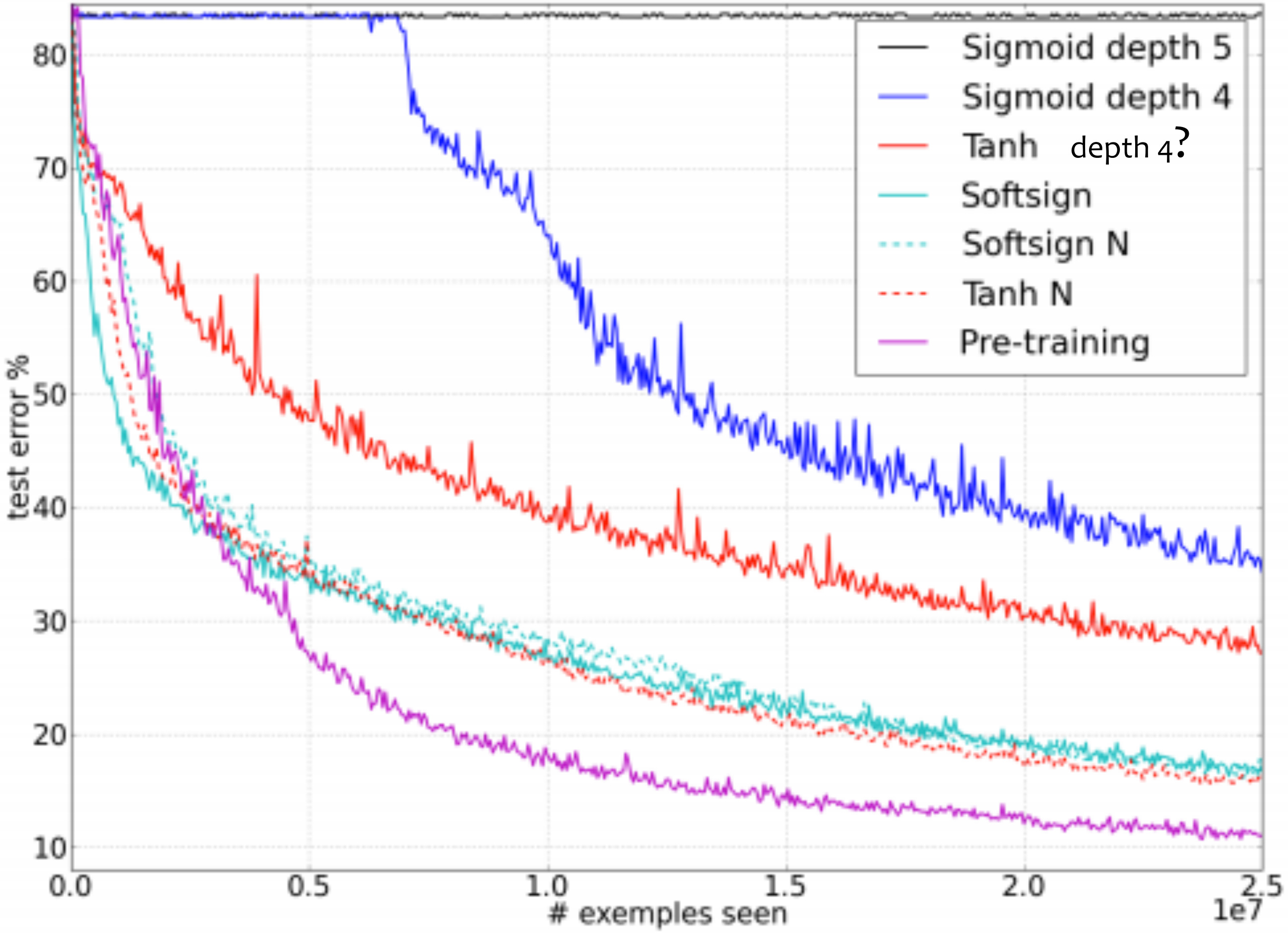
Like logistic function but shifted to range [-1, +1]


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

Slide from William Cohen

Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

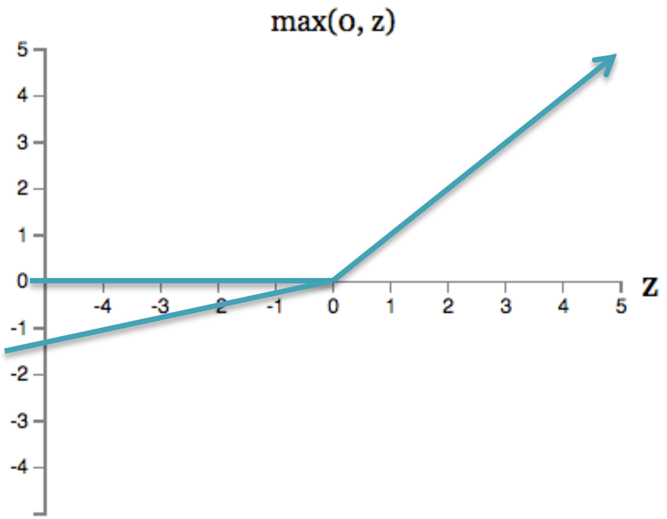


sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

Activation Functions

- A new change: modifying the nonlinearity
  - reLU often used in vision tasks

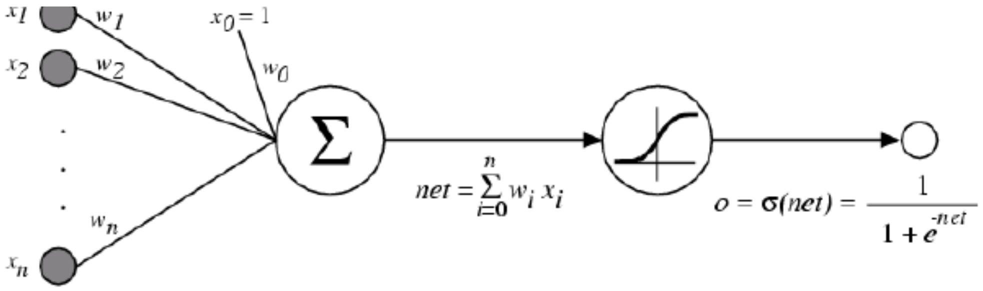


Alternate 2: rectified linear unit

Linear with a cutoff at zero

(Implementation: clip the gradient when you pass zero)

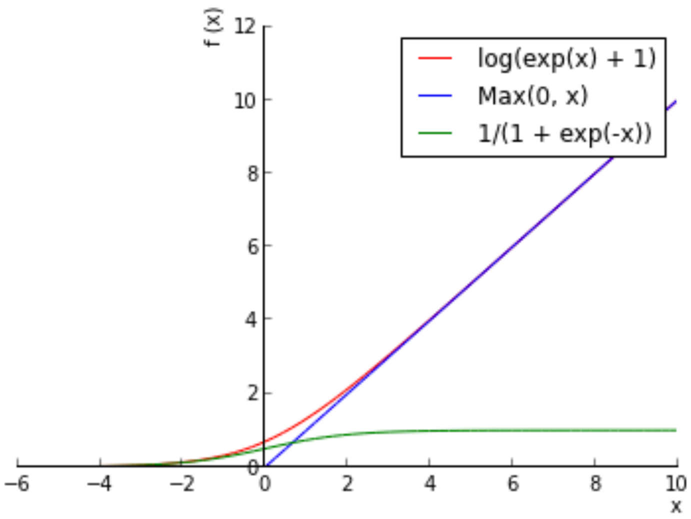
$$\max(0, w \cdot x + b).$$


x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

Slide from William Cohen

Activation Functions

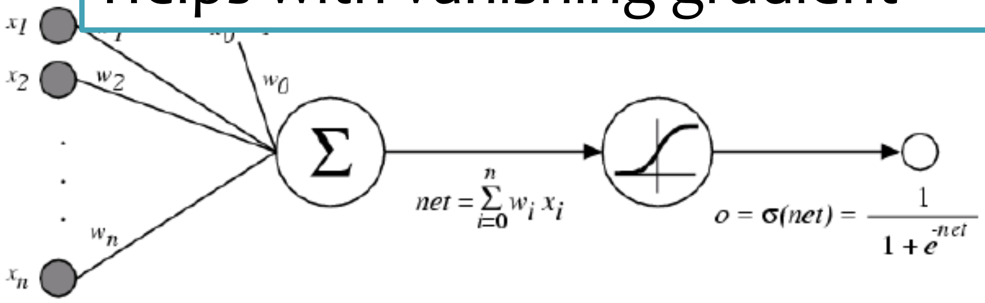
- A new change: modifying the nonlinearity
  - reLU often used in vision tasks



Alternate 2: rectified linear unit

Soft version: log(exp(x)+1)

Doesn't saturate (at one end)
Sparsifies outputs
Helps with vanishing gradient


x_1
x_2
x_n
w_1
$x_0 = 1$
w_2
w_0
w_n
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

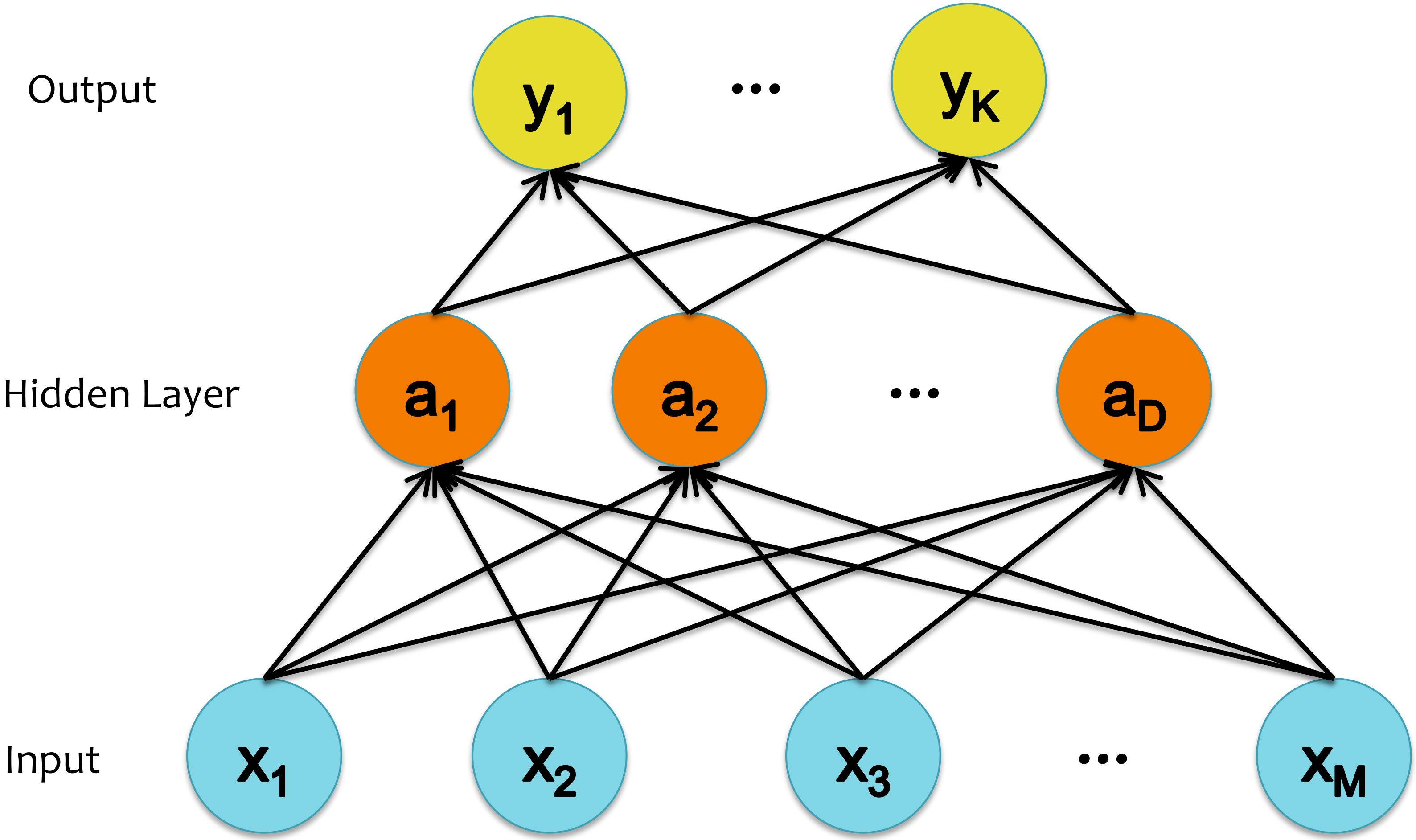
Slide from William Cohen

Objective Functions for NNs

- Regression:
  - Use the same objective as Linear Regression
  - Quadratic loss (i.e. mean squared error)
- Classification:
  - Use the same objective as Logistic Regression
  - Cross-entropy (i.e. negative log likelihood)
  - This requires probabilities, so we add an additional “softmax” layer at the end of our network

	Forward	Backward
Quadratic	$J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$
Cross Entropy	$J = y^* \log(y) + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = y^* \frac{1}{y} + (1 - y^*) \frac{1}{y - 1}$

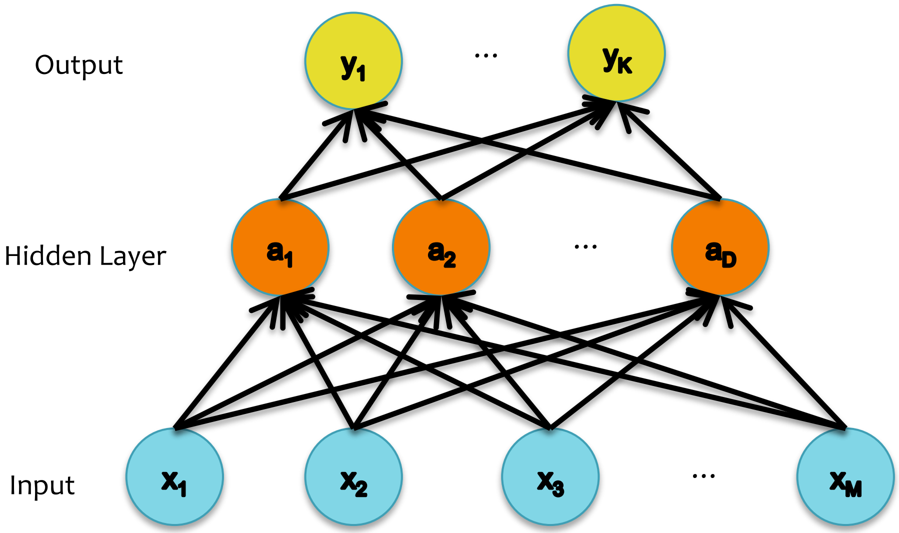
Multi-Class Output

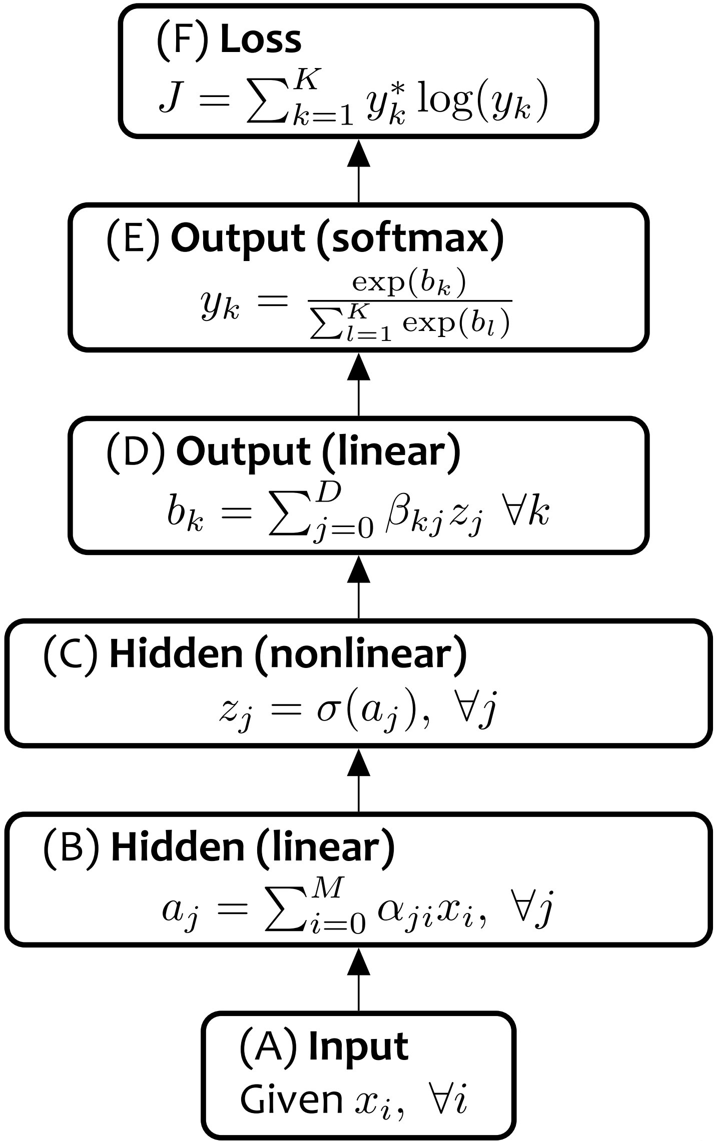


Multi-Class Output

Softmax:

$$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$$





(F) Loss
$J = \sum_{k=1}^{K} y_k^* \log(y_k)$

(E) Output (softmax)
$y_k = \frac{\exp(b_k)}{\sum_{l=1}^{K} \exp(b_l)}$

(D) Output (linear)
$b_k = \sum_{j=0}^{D} \beta_{kj} z_j \;\; \forall k$

(C) Hidden (nonlinear)
$z_j = \sigma(a_j), \;\; \forall j$

(B) Hidden (linear)
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \;\; \forall j$

(A) Input
Given $x_i, \;\; \forall i$

Cross-entropy vs. Quadratic loss

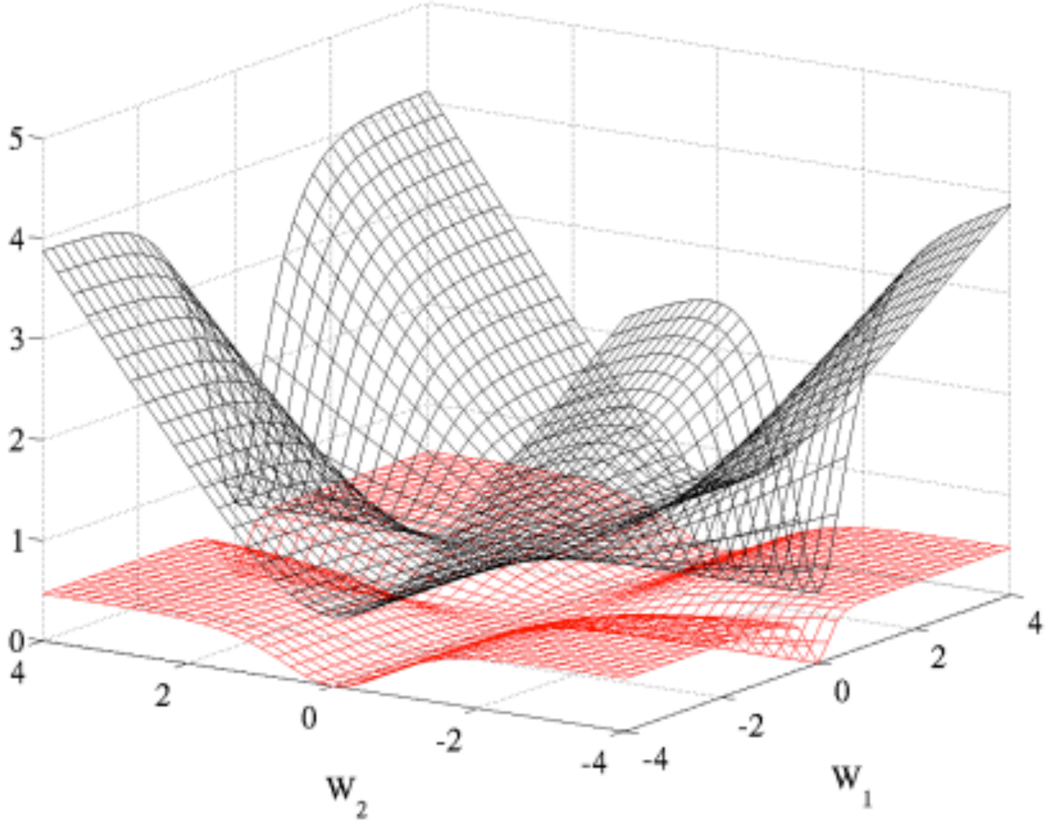


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Figure from Glorot & Bentio (2010)

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Objective Functions

Matching Quiz: Suppose you are given a neural net with a single output, y, and one hidden layer.

1) Minimizing sum of squared errors…

2) Minimizing sum of squared errors plus squared Euclidean norm of weights…

3) Minimizing cross-entropy…

4) Minimizing hinge loss…

…gives…

5) … MLE estimates of weights assuming target follows a Bernoulli with parameter given by the output value

6) … MAP estimates of weights assuming weight priors are zero mean Gaussian

7) … estimates with a large margin on the training data

8) … MLE estimates of weights assuming zero mean Gaussian noise on the output value

A. 1=5, 2=7, 3=6, 4=8

B. 1=5, 2=7, 3=8, 4=6

C. 1=7, 2=5, 3=5, 4=7

D. 1=7, 2=5, 3=6, 4=8

E. 1=8, 2=6, 3=5, 4=7

F. 1=8, 2=6, 3=8, 4=6

BACKPROPAGATION

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:

– Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

– Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Training

Backpropagation

- **Question 1:**
  When can we compute the gradients of the parameters of an arbitrary neural network?

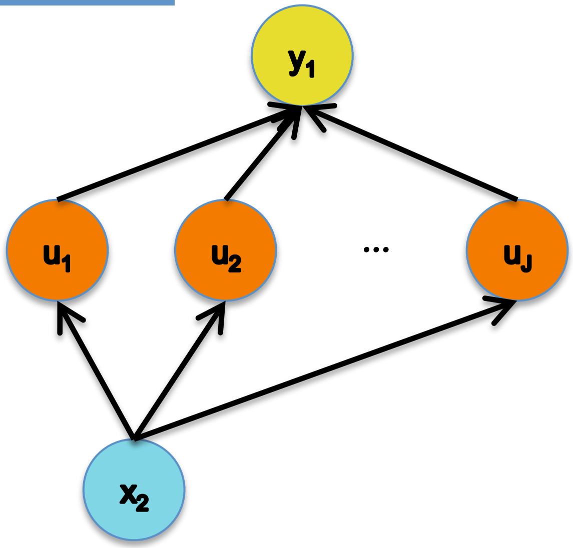
- **Question 2:**
  When can we make the gradient computation efficient?

Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Training

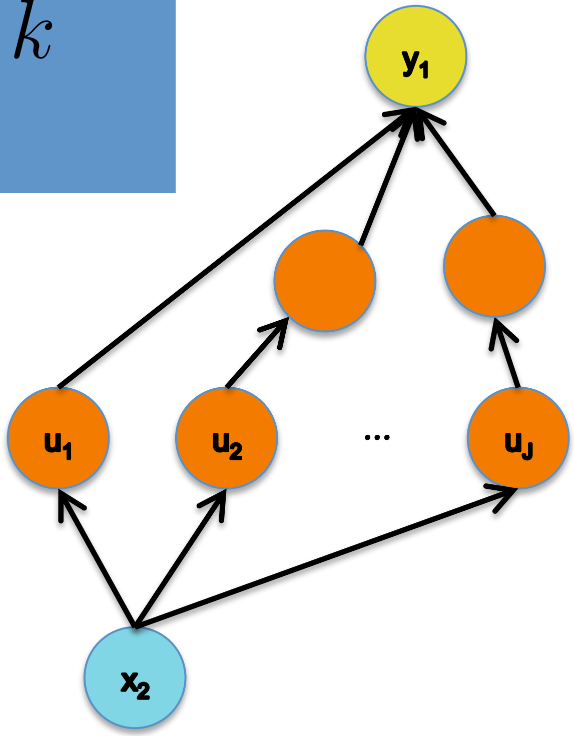
Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k$$

Backpropagation is just repeated application of the **chain rule** from Calculus 101.

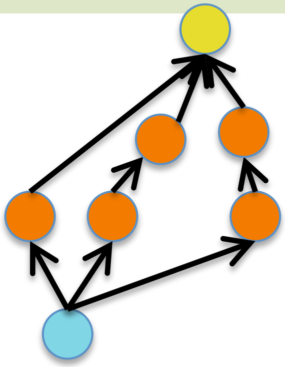


Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation:

1. **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node's intermediate quantity.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called **automatic differentiation in the reverse-mode**

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

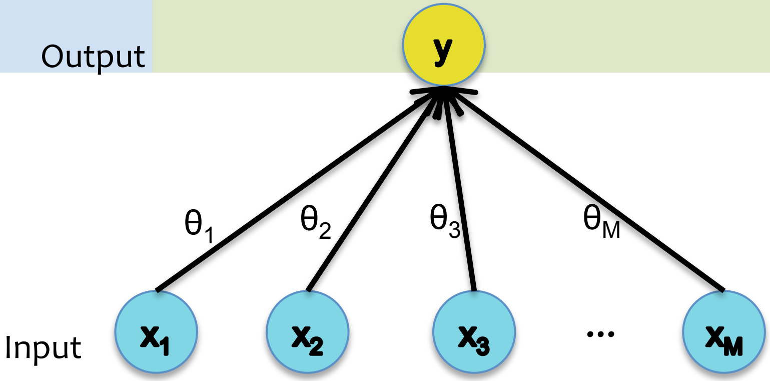
$t = x^2$

Backpropagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward	Backward
$J = cos(u)$	$\frac{dJ}{du} \mathrel{+}= -sin(u)$
$u = u_1 + u_2$	$\frac{dJ}{du_1} \mathrel{+}= \frac{dJ}{du}\frac{du}{du_1}, \quad \frac{du}{du_1} = 1 \qquad \frac{dJ}{du_2} \mathrel{+}= \frac{dJ}{du}\frac{du}{du_2}, \quad \frac{du}{du_2} = 1$
$u_1 = sin(t)$	$\frac{dJ}{dt} \mathrel{+}= \frac{dJ}{du_1}\frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$
$u_2 = 3t$	$\frac{dJ}{dt} \mathrel{+}= \frac{dJ}{du_2}\frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$
$t = x^2$	$\frac{dJ}{dx} \mathrel{+}= \frac{dJ}{dt}\frac{dt}{dx}, \quad \frac{dt}{dx} = 2x$

Training Backpropagation



Case 1:
Logistic Regression

Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

$$a = \sum_{j=0}^{D} \theta_j x_j$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

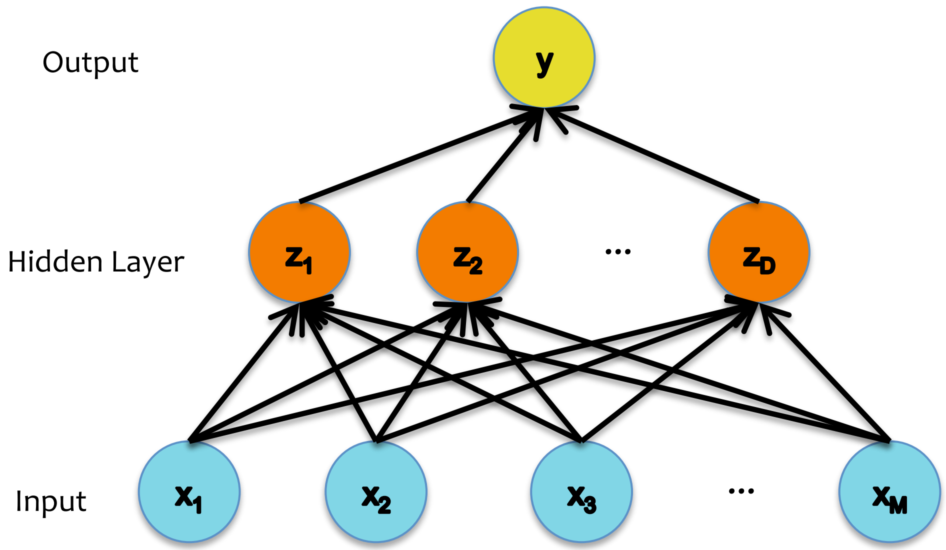
$$\frac{dJ}{da} = \frac{dJ}{dy}\frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

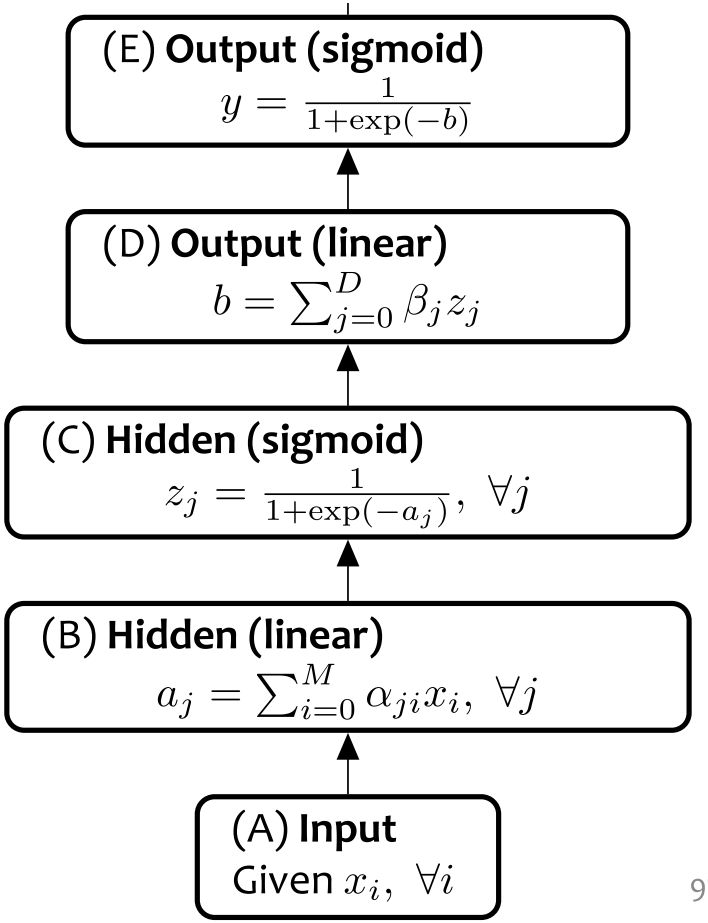
$$\frac{dJ}{d\theta_j} = \frac{dJ}{da}\frac{da}{d\theta_j}, \quad \frac{da}{d\theta_j} = x_j$$

$$\frac{dJ}{dx_j} = \frac{dJ}{da}\frac{da}{dx_j}, \quad \frac{da}{dx_j} = \theta_j$$

Training

Backpropagation





(E) **Output (sigmoid)**
$$y = \frac{1}{1+\exp(-b)}$$

(D) **Output (linear)**
$$b = \sum_{j=0}^{D} \beta_j z_j$$

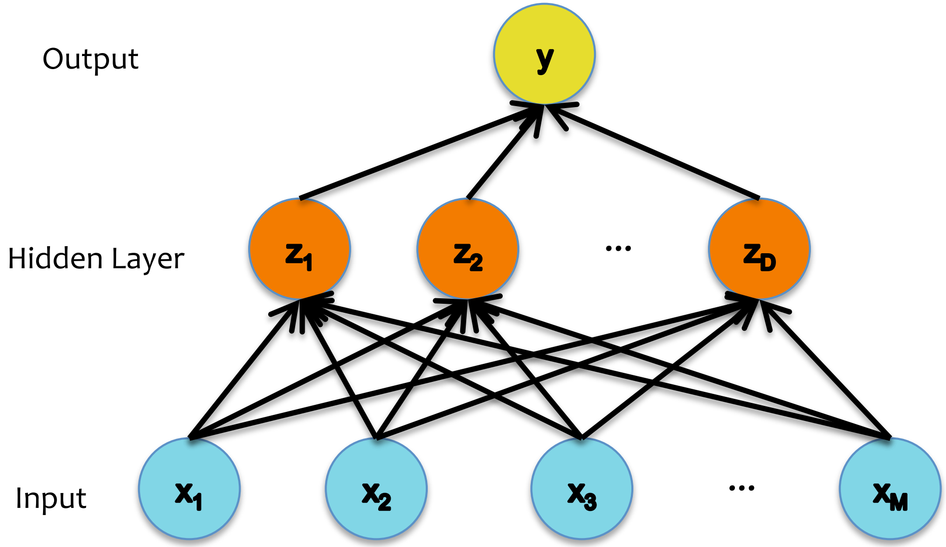
(C) **Hidden (sigmoid)**
$$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$$

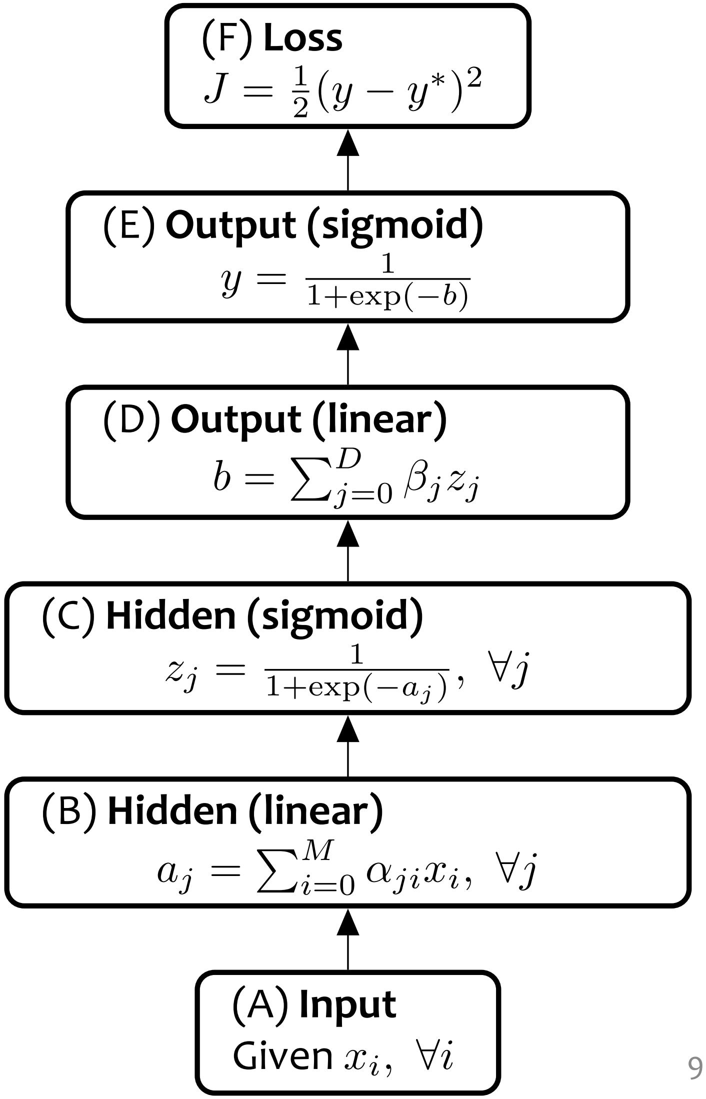
(B) **Hidden (linear)**
$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$$

(A) **Input**
Given $x_i, \ \forall i$

Training

Backpropagation





(F) **Loss**
$J = \frac{1}{2}(y - y^*)^2$

(E) **Output (sigmoid)**
$y = \frac{1}{1+\exp(-b)}$

(D) **Output (linear)**
$b = \sum_{j=0}^{D} \beta_j z_j$

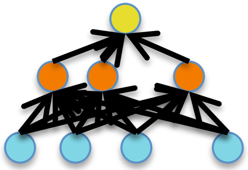
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

(A) **Input**
Given $x_i, \ \forall i$

Training Backpropagation

Case 2: Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

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

$$b = \sum_{j=0}^{D} \beta_j z_j$$

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

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$$

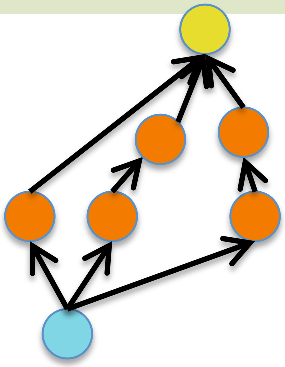
Training

Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation:

1. **Instantiate the computation as a directed acyclic graph**, where each intermediate quantity is a node
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivative** of the goal with respect to that node's intermediate quantity.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order**. At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called **automatic differentiation in the reverse-mode**

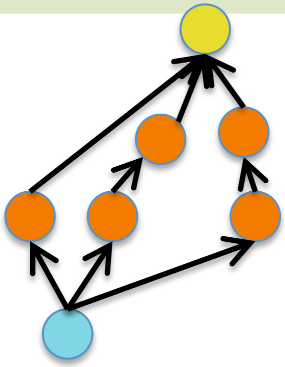
Training

Chain Rule

Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j} \frac{du_j}{dx_k}, \quad \forall i, k$$



Backpropagation:

1. **Instantiate the computation as a directed acyclic graph,** where each node represents a Tensor.
2. At each node, store (a) the quantity computed in the forward pass and (b) the **partial derivatives** of the goal with respect to that node's Tensor.
3. **Initialize** all partial derivatives to 0.
4. Visit each node in **reverse topological order.** At each node, add its contribution to the partial derivatives of its parents

This algorithm is also called **automatic differentiation in the reverse-mode**

Training

Backpropagation

Case 2:

	Forward	Backward
Module 5	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Module 4	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Module 3	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$
Module 2	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Module 1	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \sum_{j=0}^{D} \alpha_{ji}$

Background

1. Given training dat

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of t

- Decision functior

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

- Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

A Recipe for

Gradients

Backpropagation can compute this gradient!

And it's a **special case of a more general algorithm** called reverse-mode automatic differentiation that can compute the gradient of any differentiable function efficiently!

$\boldsymbol{y}_i)$

opp site the gradient)

$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Summary

1. **Neural Networks**…
   - provide a way of learning features
   - are highly nonlinear prediction functions
   - (can be) a highly parallel network of logistic regression classifiers
   - discover useful hidden representations of the input
2. **Backpropagation**…
   - provides an efficient way to compute gradients
   - is a special case of reverse-mode automatic differentiation