

10-601B Introduction to Machine Learning

Neural Networks

Presentation by:
Amirhossein Abaskohi
ACM Summer School 2021 University of Tehran

Matt Gormley
Lecture 15
October 19, 2016

Reminders

Outline

- Logistic Regression (Recap)
- Neural Networks
- Backpropagation

RECALL: LOGISTIC REGRESSION

Using gradient ascent for linear classifiers

Recall...

Key idea behind today's lecture:

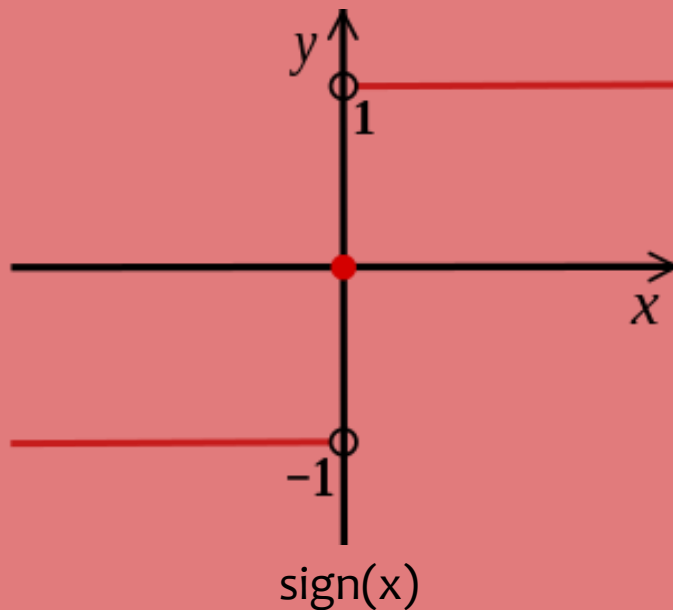
1. Define a linear classifier (logistic regression)
2. Define an objective function (likelihood)
3. Optimize it with gradient descent to learn parameters
4. Predict the class with highest probability under the model

Using gradient ascent for linear classifiers

Recall...

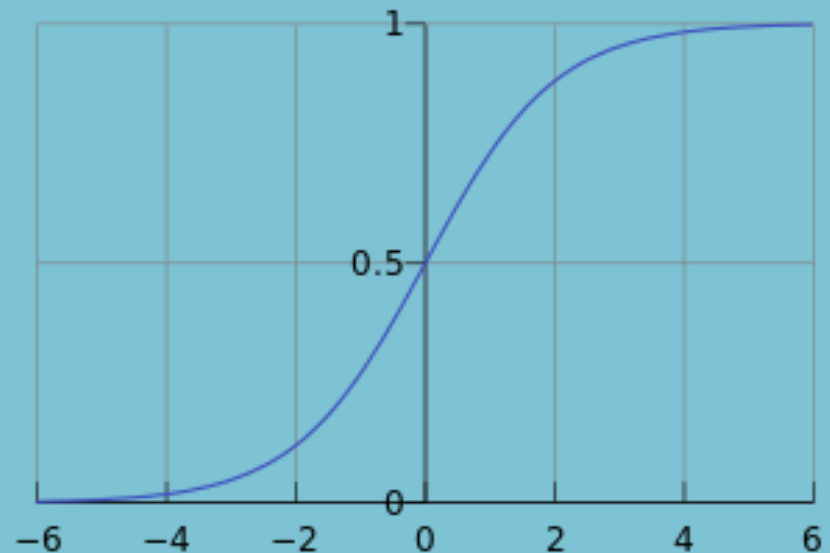
This decision function isn't differentiable:

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



Use a differentiable function instead:

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



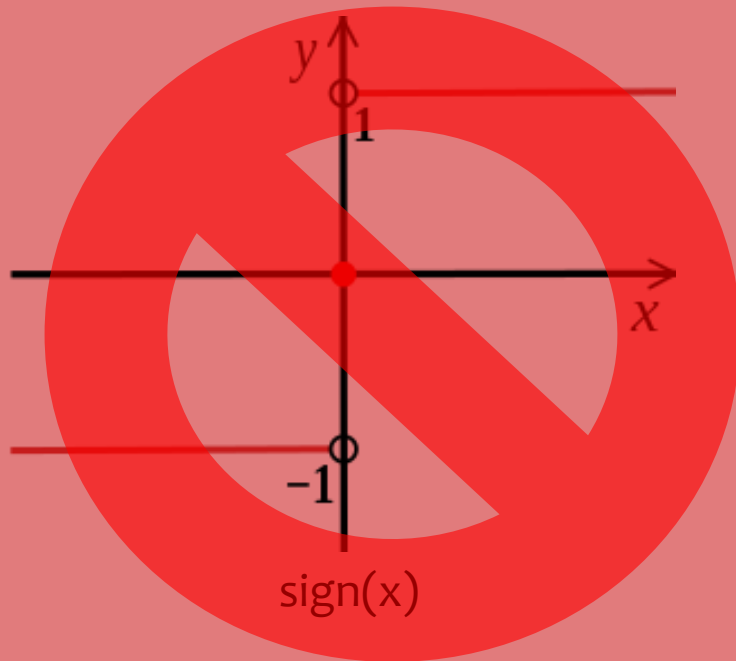
$$\text{logistic}(u) \circ \frac{1}{1 + e^{-u}}$$

Using gradient ascent for linear classifiers

Recall...

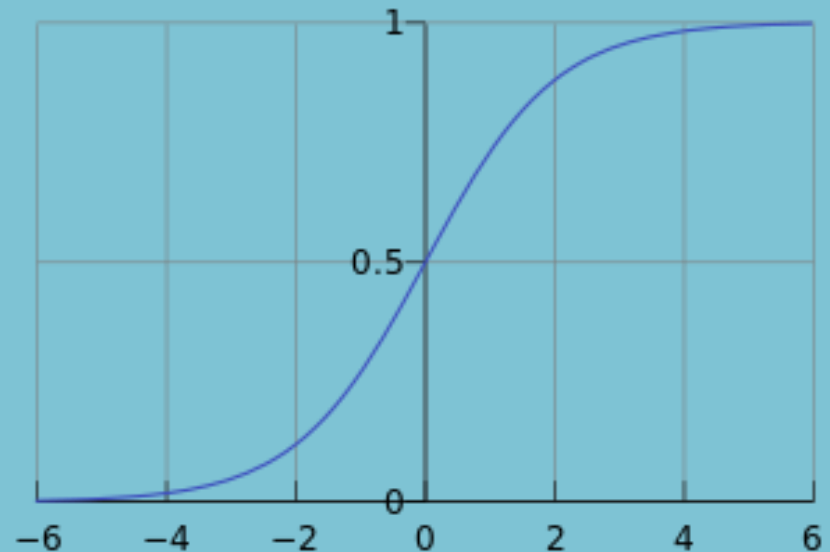
This decision function isn't differentiable:

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



Use a differentiable function instead:

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



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

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.

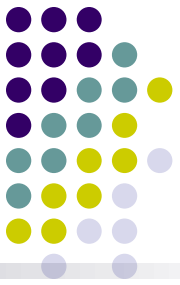
$$p_{\boldsymbol{\theta}}(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T \mathbf{x})}$$

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

Prediction: Output is the most probable class.

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

NEURAL NETWORKS

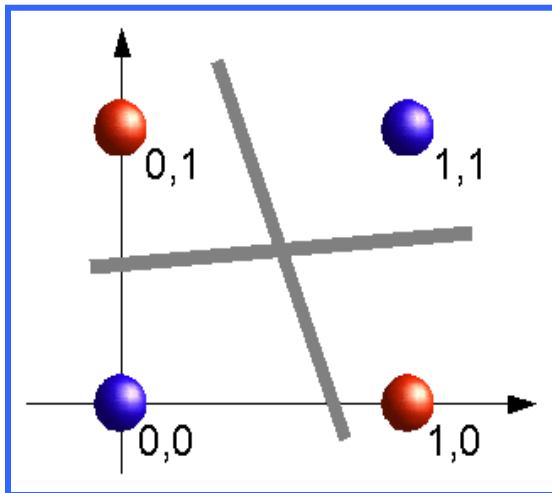
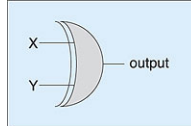


Learning highly non-linear functions

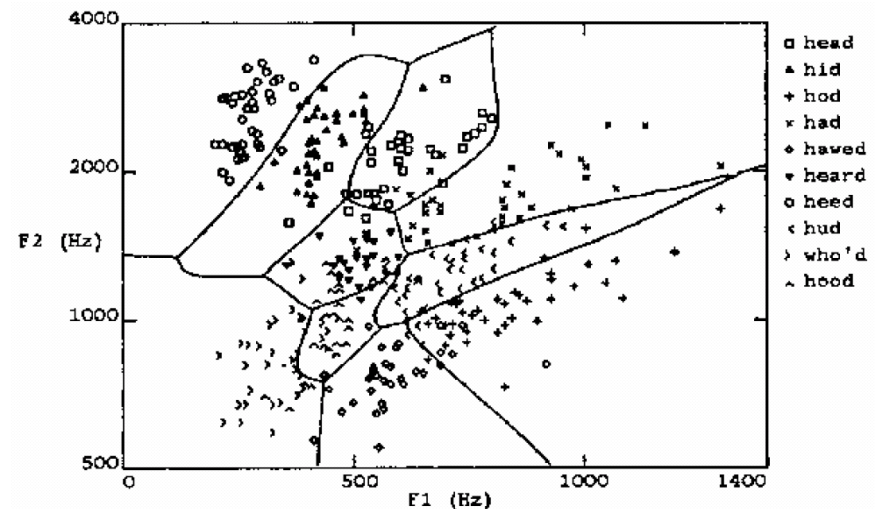
$f: X \rightarrow Y$

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

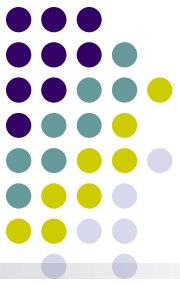
The XOR gate



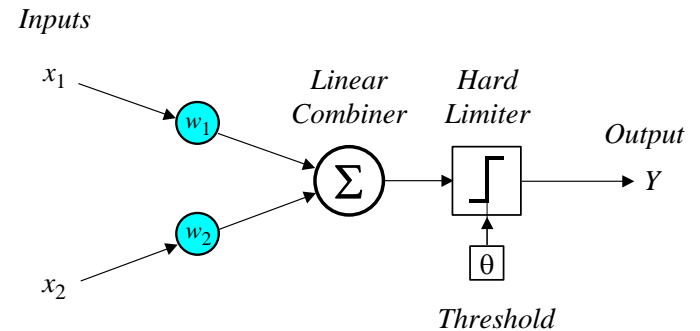
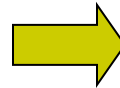
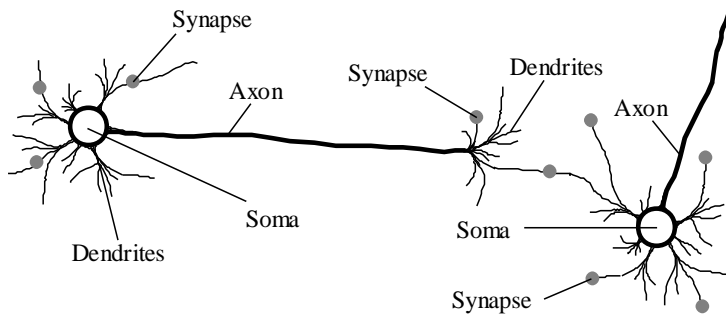
Speech recognition



Perceptron and Neural Nets

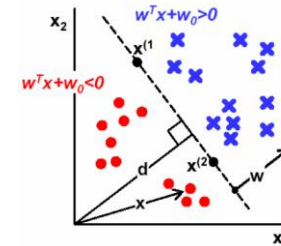


- From biological neuron to artificial neuron (perceptron)



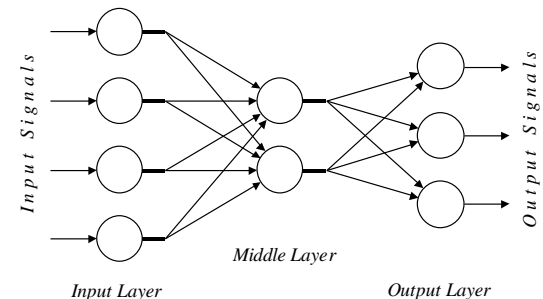
- Activation function

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



- Artificial neuron networks

- supervised learning
- gradient descent

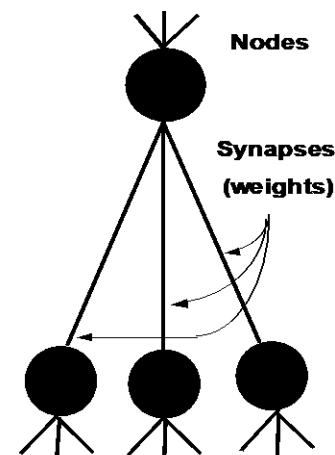
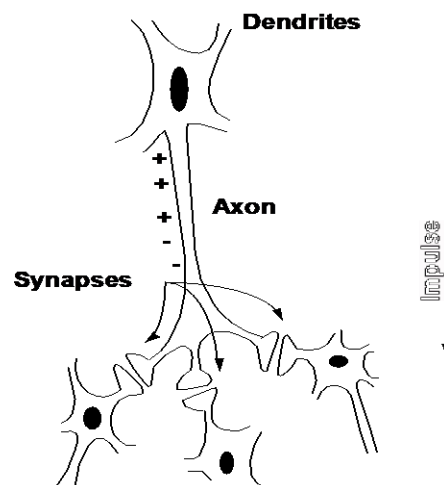


Connectionist Models



- 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

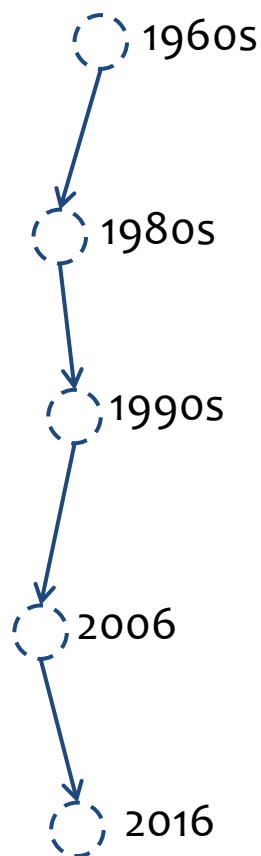
Why is everyone talking about Deep Learning?

- 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



The New York Times

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:

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

2. Choose each of these:

- Decision function

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

- Loss function

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

Face



Face



Not a face



Examples: Linear regression,
Logistic regression, Neural Network

Examples: Mean-squared error,
Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of these:

- Decision function

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

- Loss function

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

3. Define goal:

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

4. Train with SGD:

(take small steps
opposite the gradient)

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

Background

A Recipe for Gradients

1. Given training data

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

2. Choose each of the

- Decision function

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


- Loss function

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

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!

opposite the gradient)


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

Goals for Today's Lecture

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

2. Choose each of these:

– Decision function

$$\hat{y} = f_{\theta}(x_i)$$

– Loss function

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

4. Train with SGD:

– Take small steps
opposite the gradient)

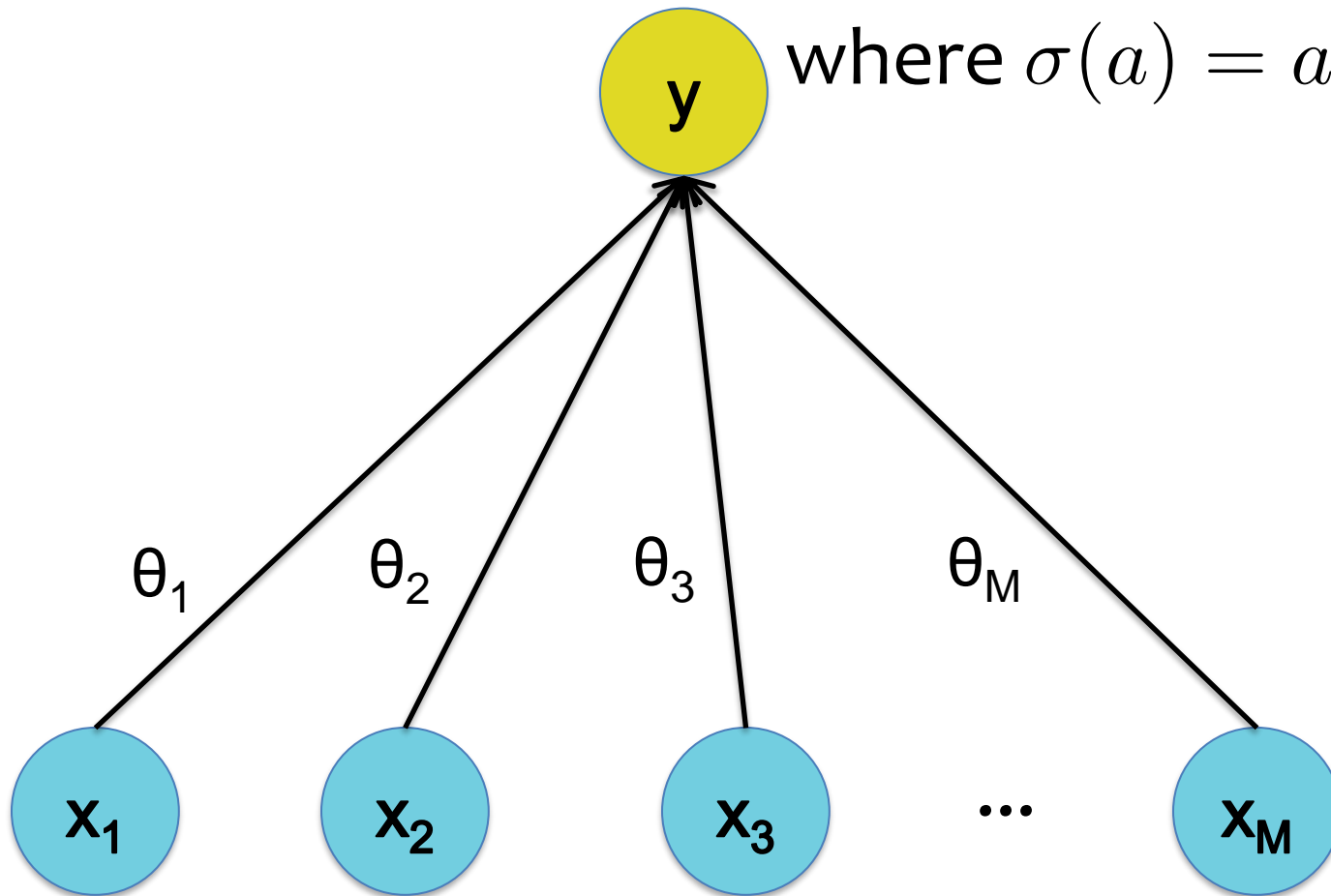
$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i)$$

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

where $\sigma(a) = a$

Output

Input

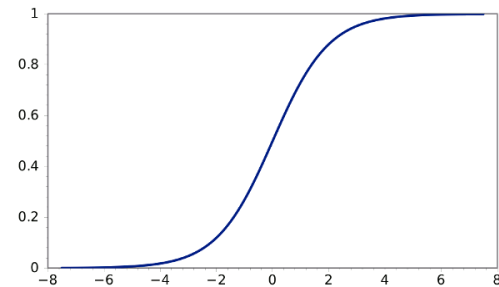


Logistic Regression

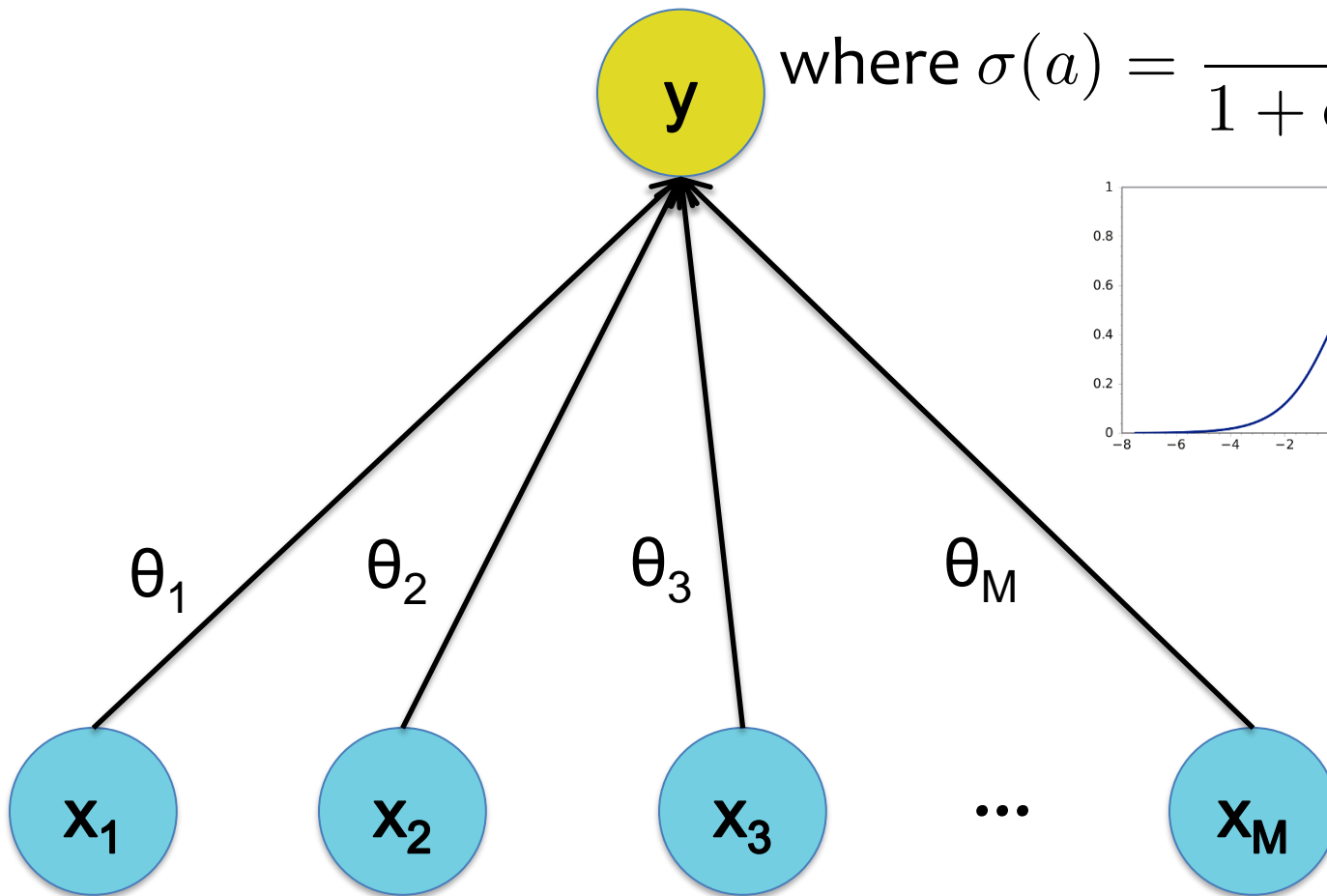
$$y = h_{\theta}(\mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Output

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



Input



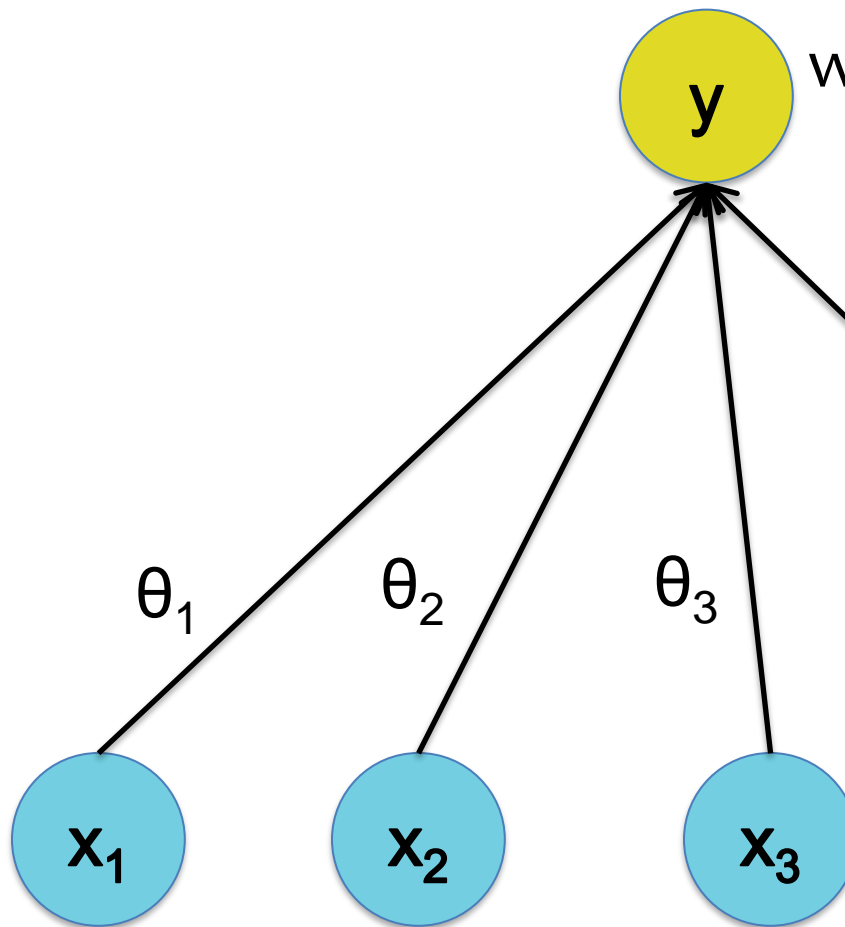
Logistic Regression

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

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

Output

Input

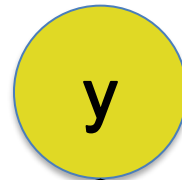


Logistic Regression

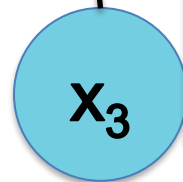
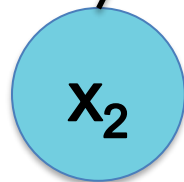
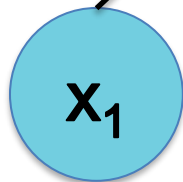
$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

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

Output



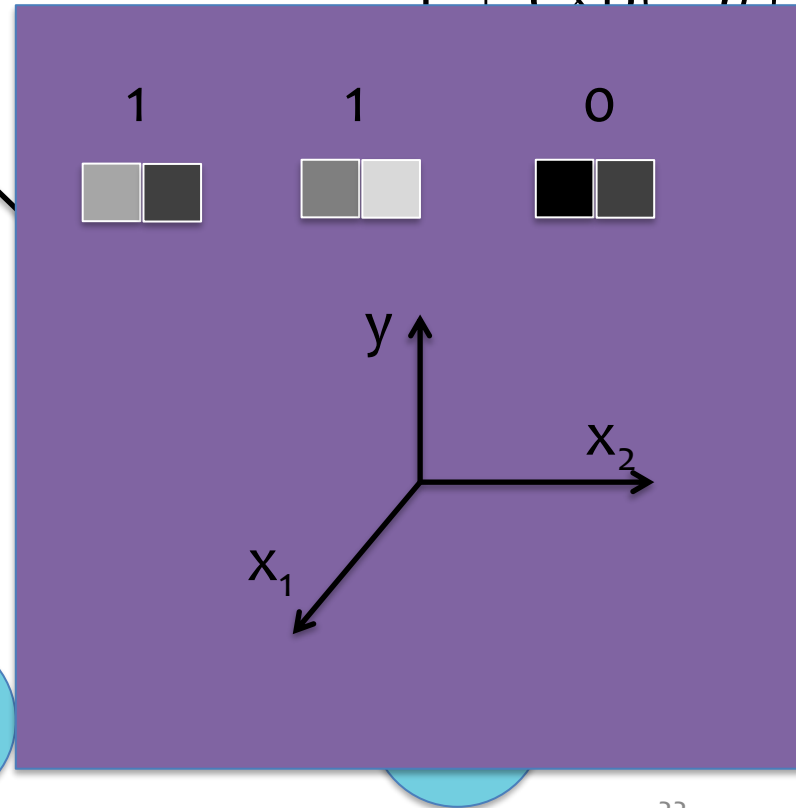
Input



θ_1

θ_2

θ_3

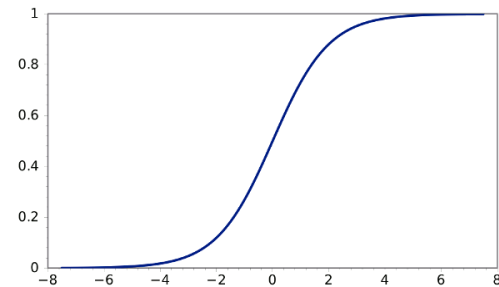


Logistic Regression

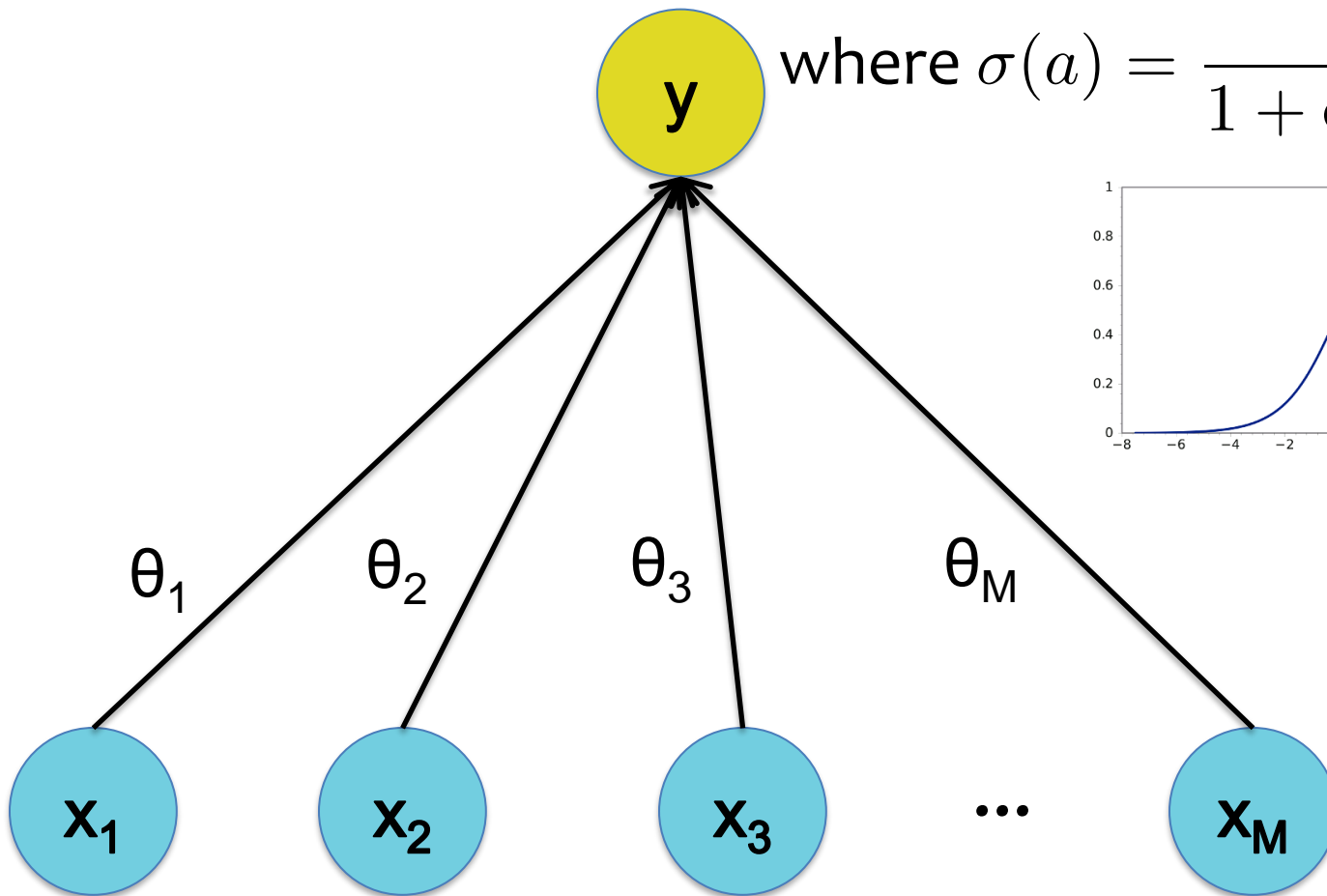
$$y = h_{\theta}(\mathbf{x}) = \sigma(\theta^T \mathbf{x})$$

Output

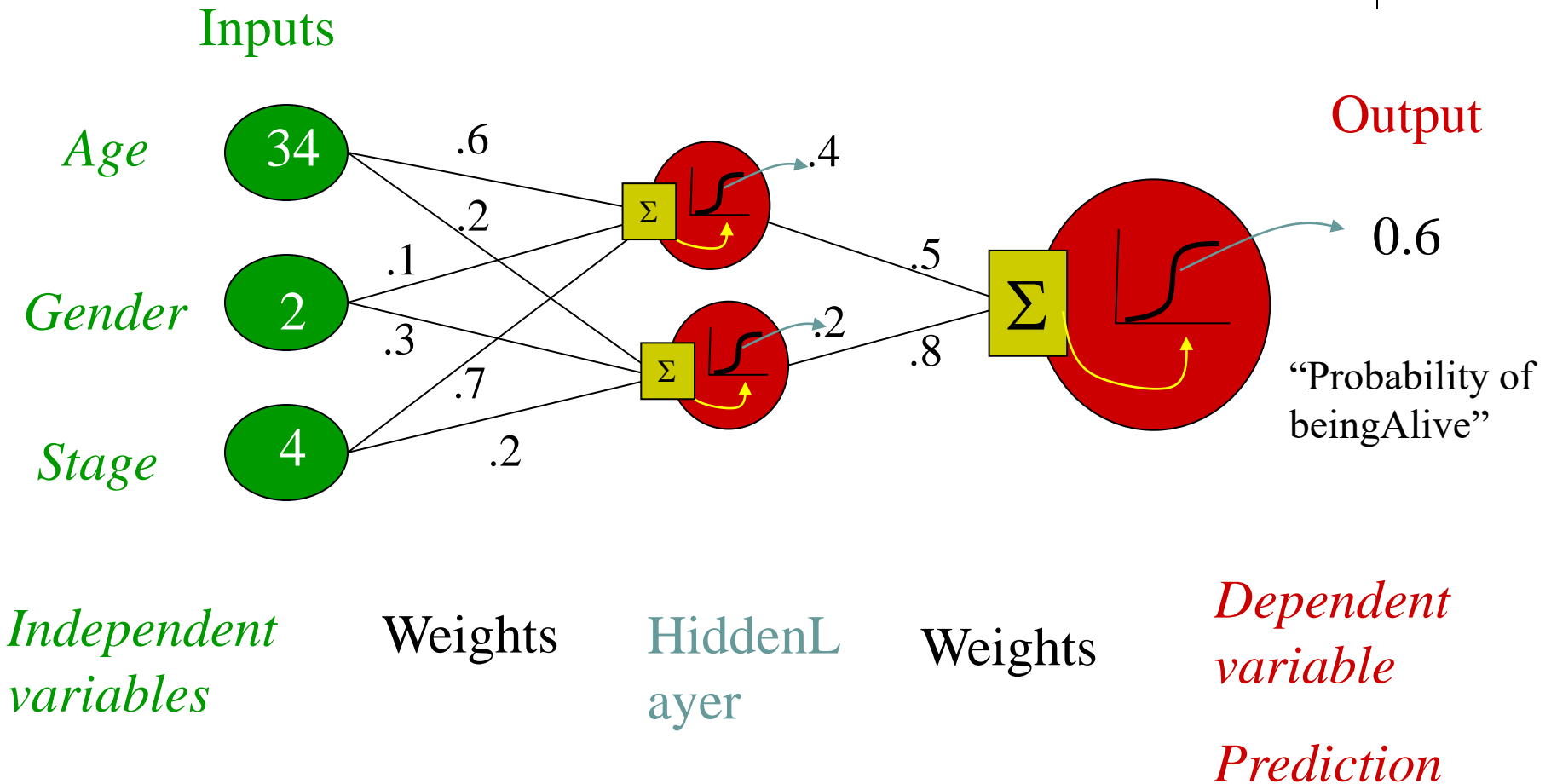
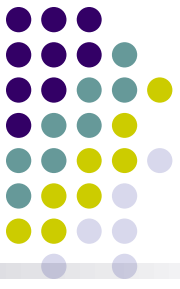
where $\sigma(a) = \frac{1}{1 + \exp(-a)}$



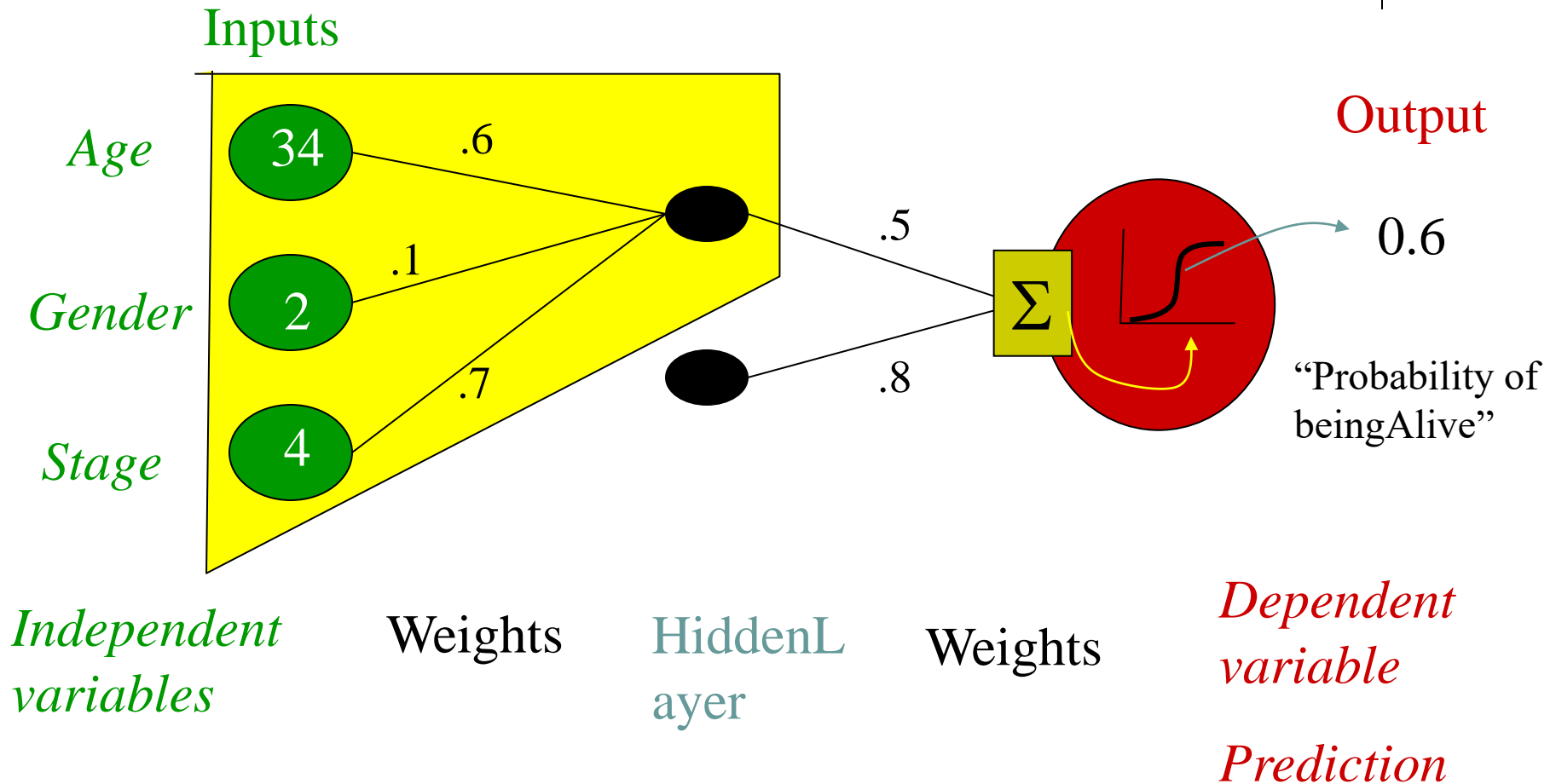
Input

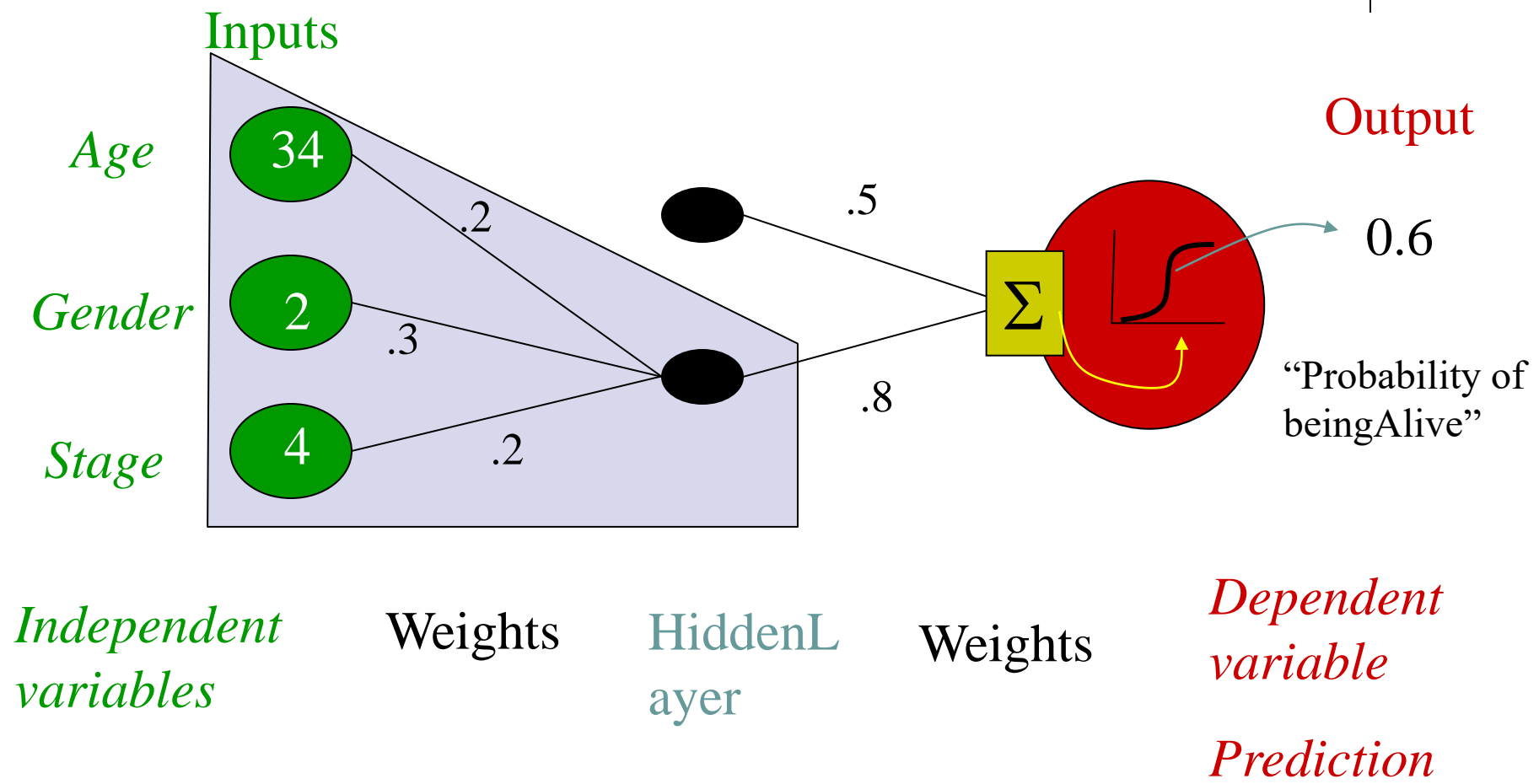


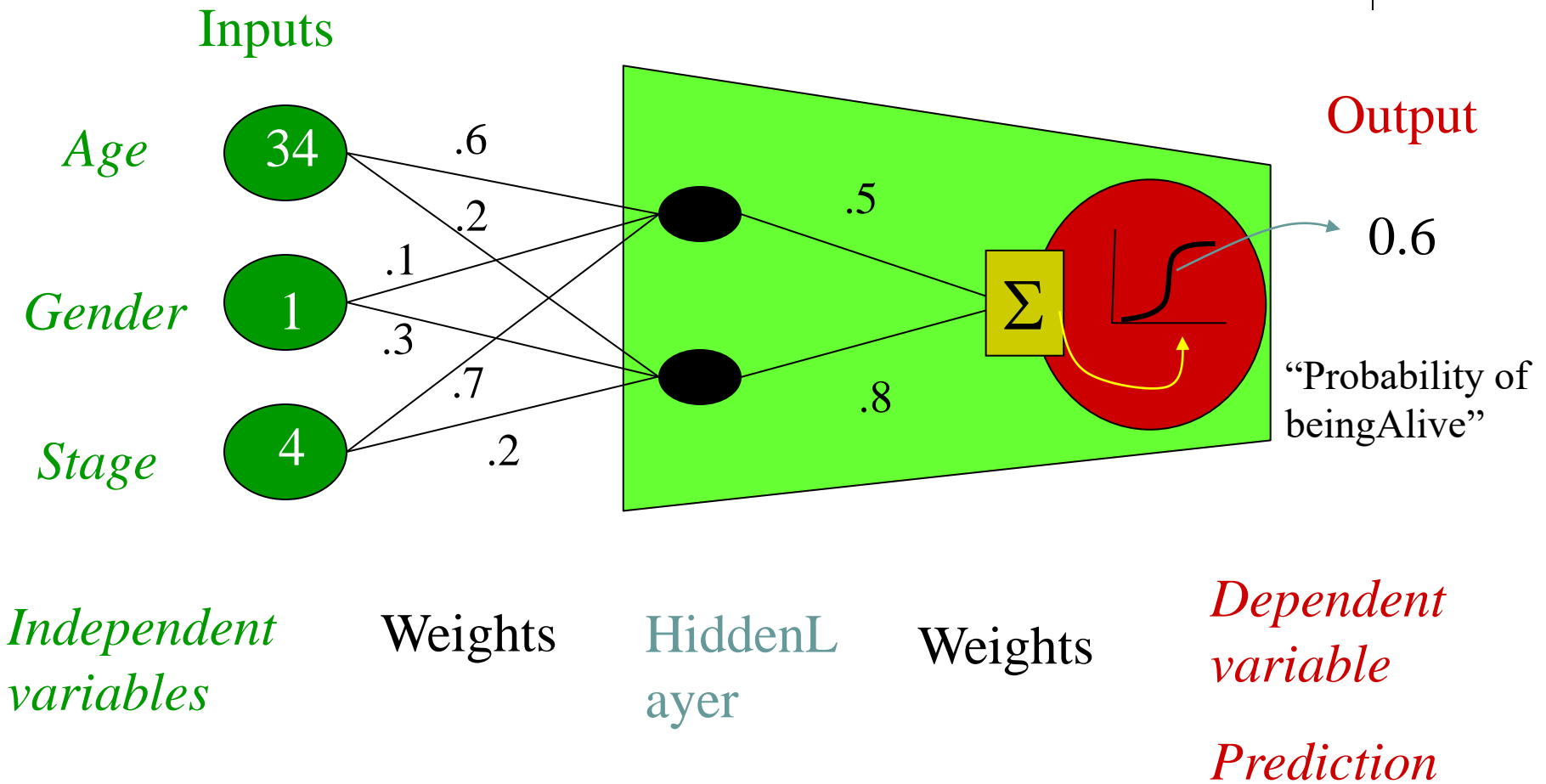
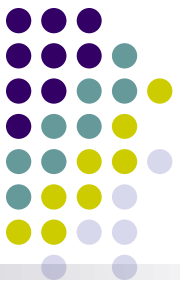
Neural Network Model



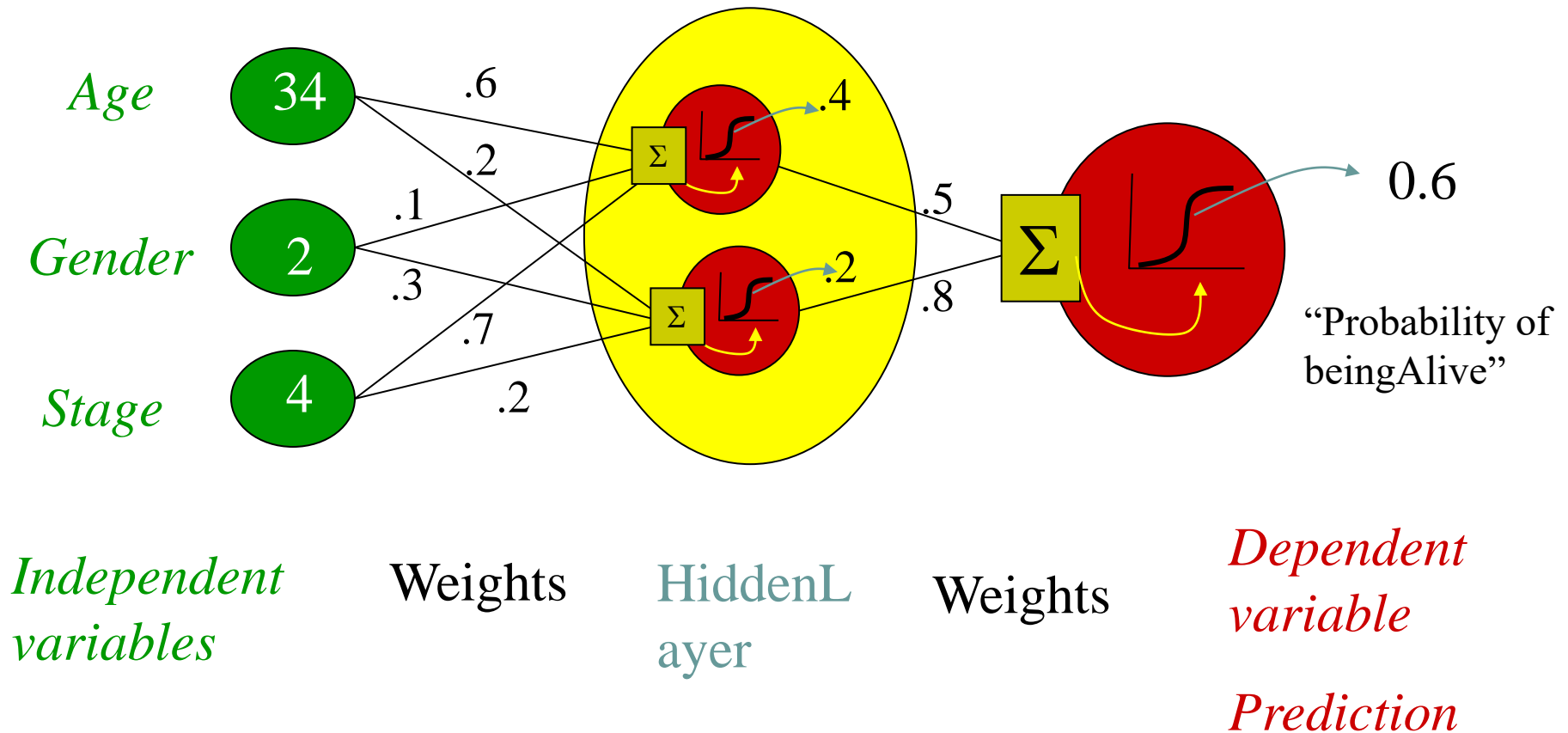
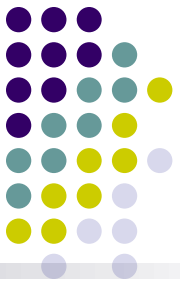
“Combined logistic models”



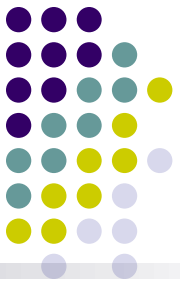




Not really, no target for hidden units...

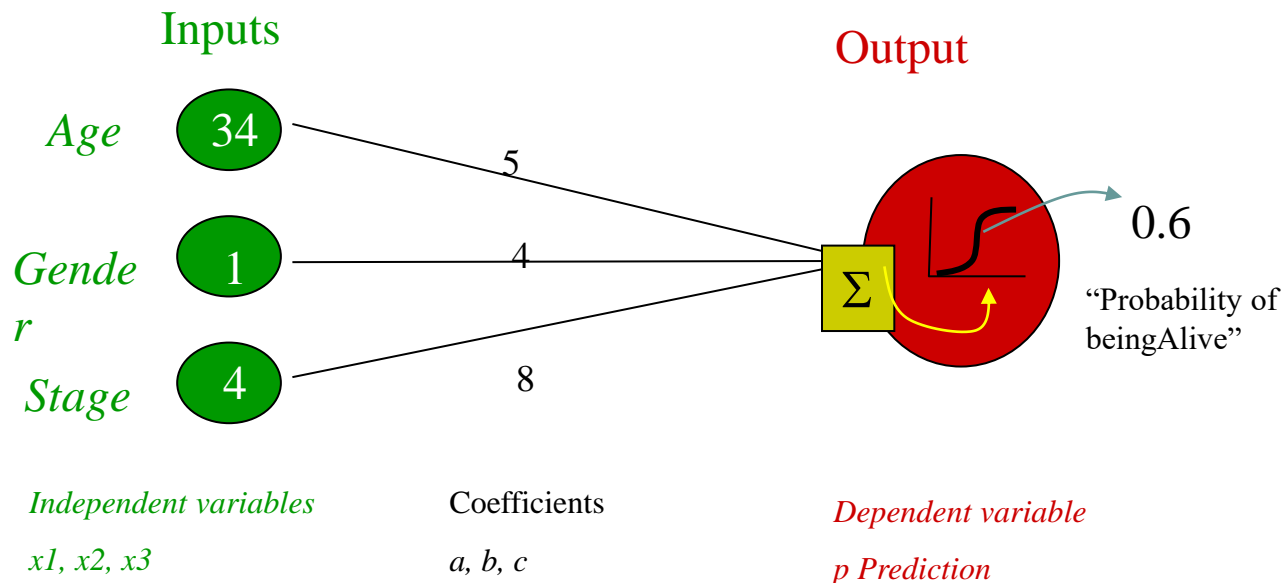


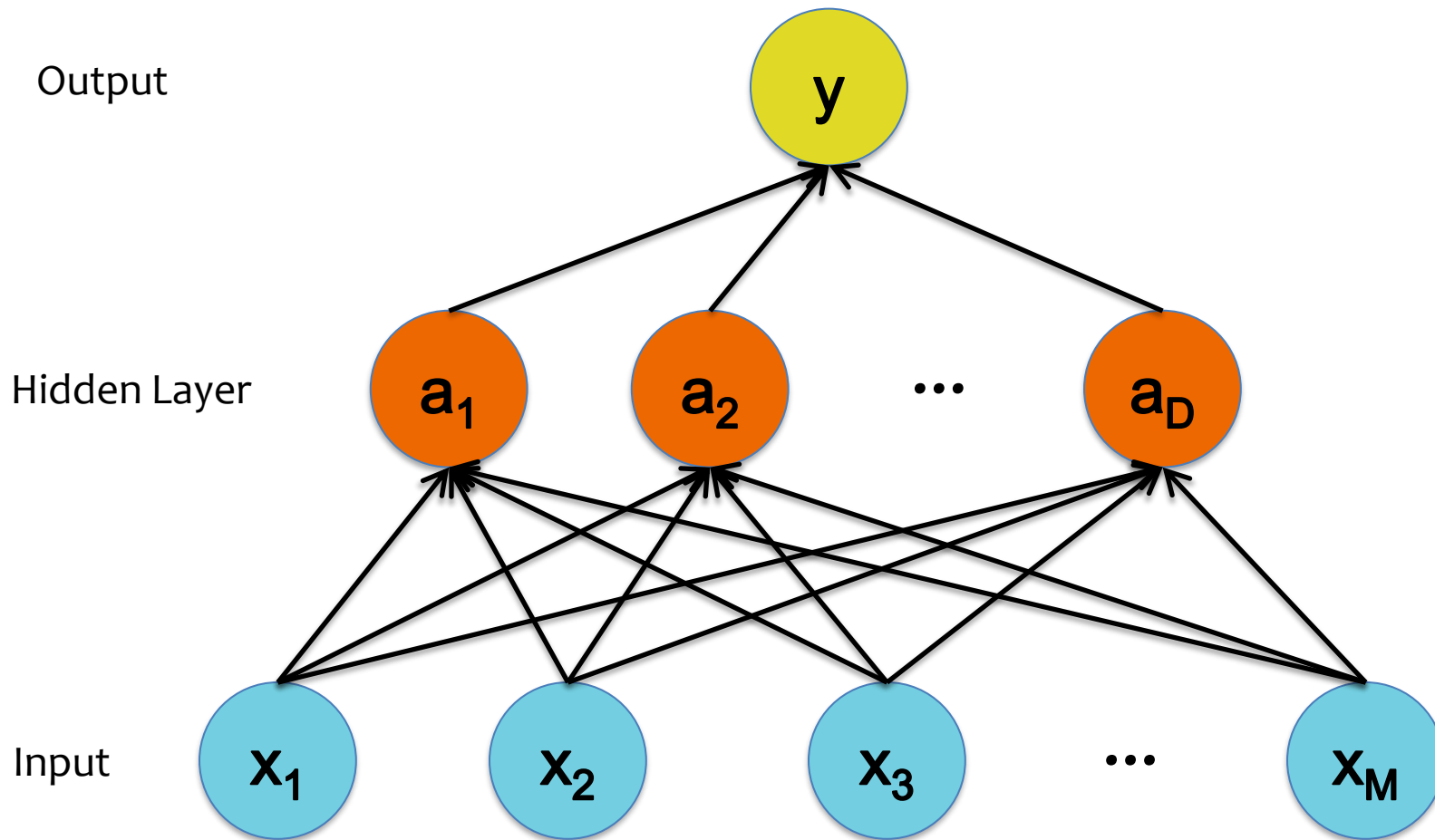
Jargon Pseudo-Correspondence



- Independent variable = input variable
- Dependent variable = output variable
- Coefficients = “weights”
- Estimates = “targets”

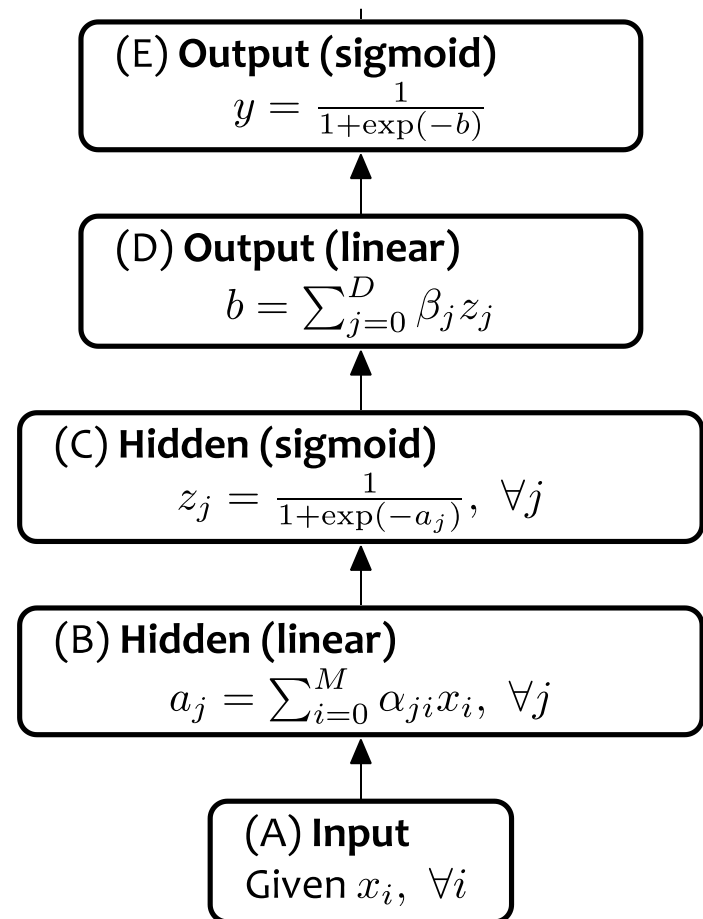
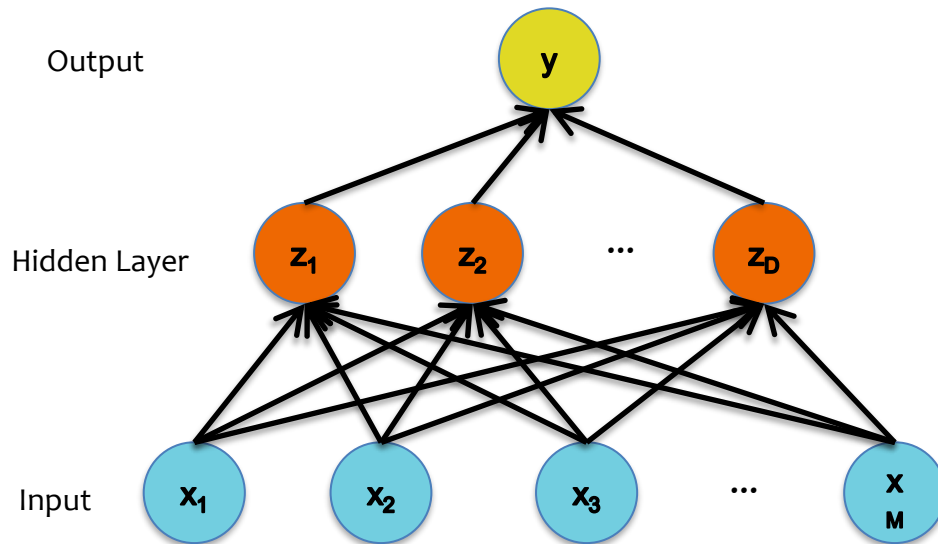
Logistic Regression Model (the sigmoid unit)





Decision Functions

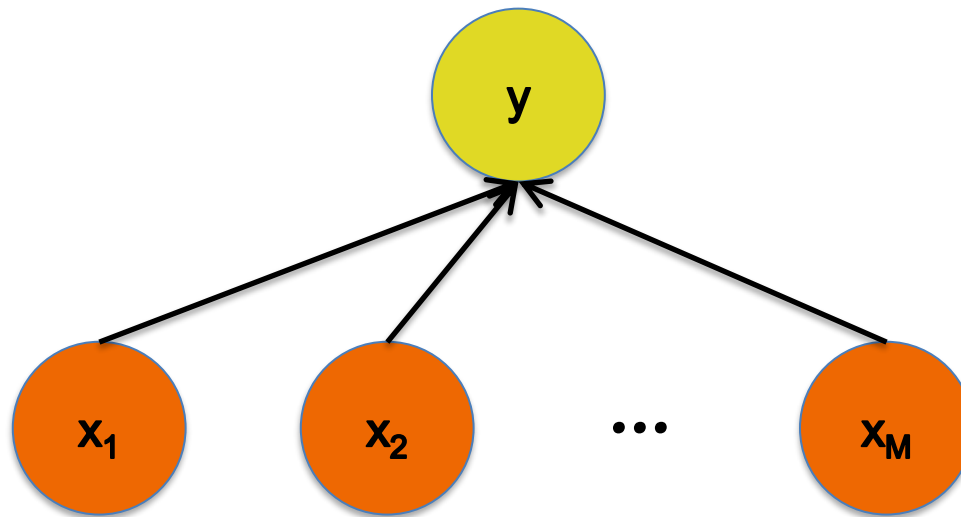
Neural Network



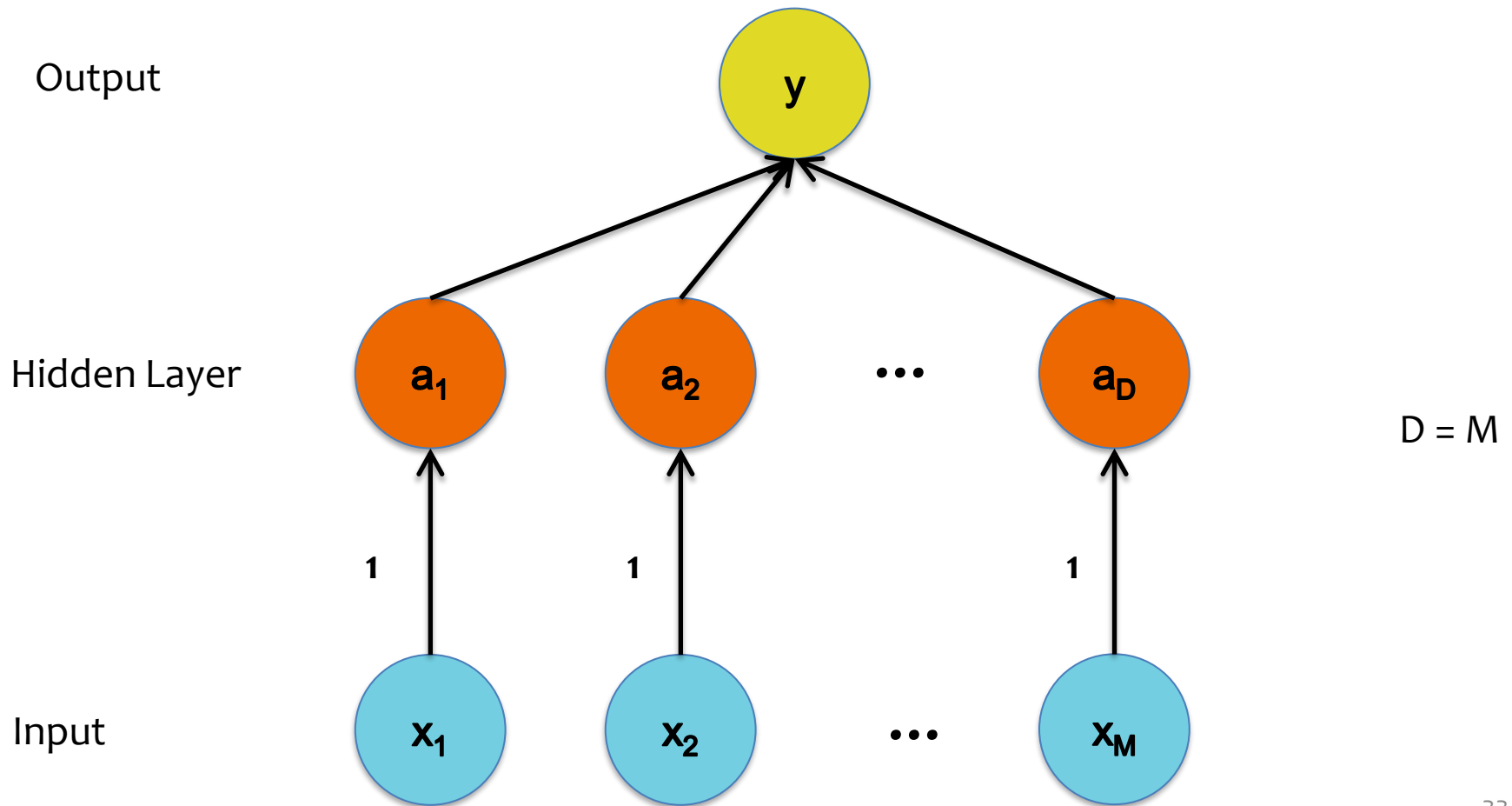
Building a Neural Net

Output

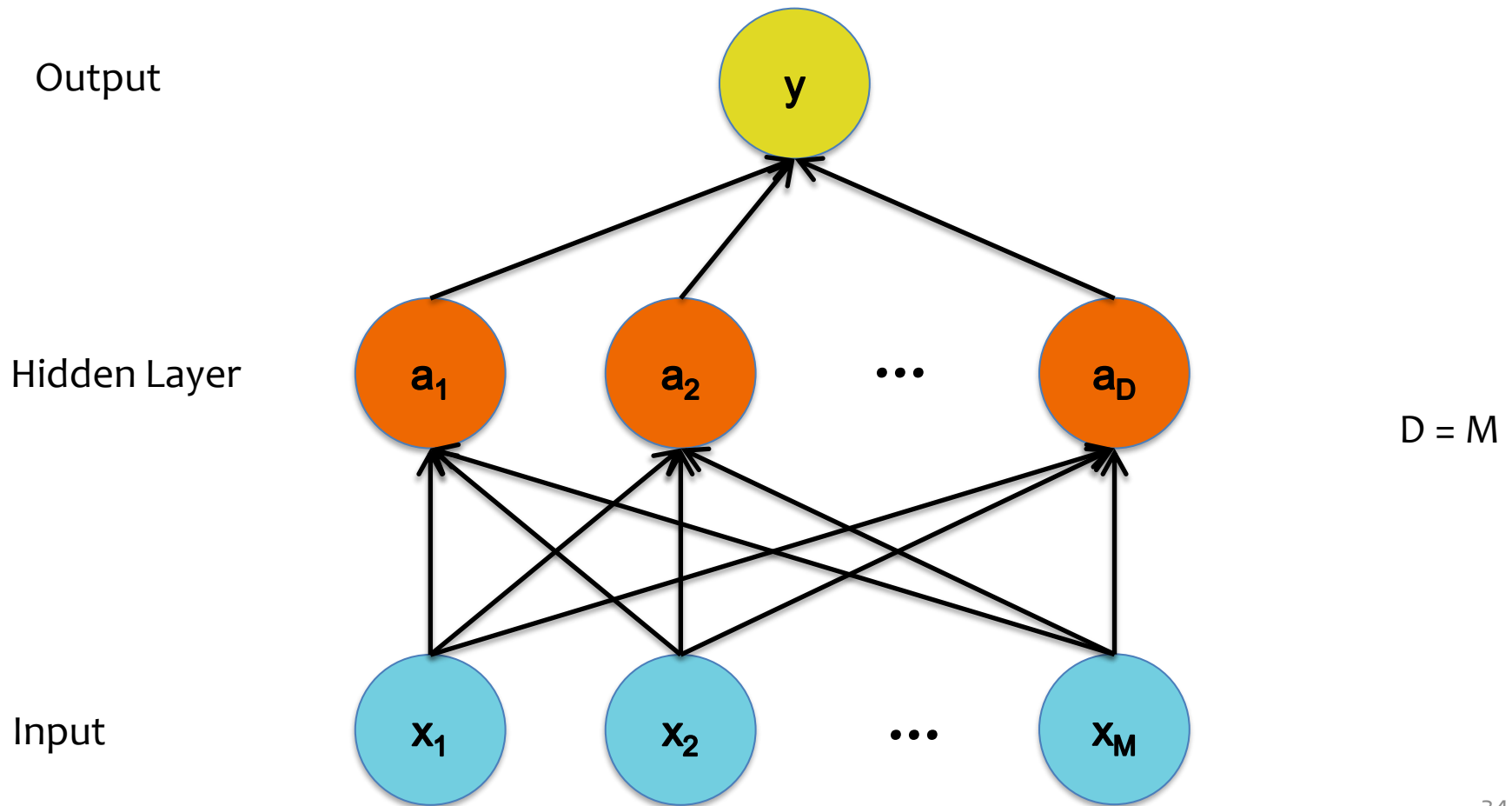
Features



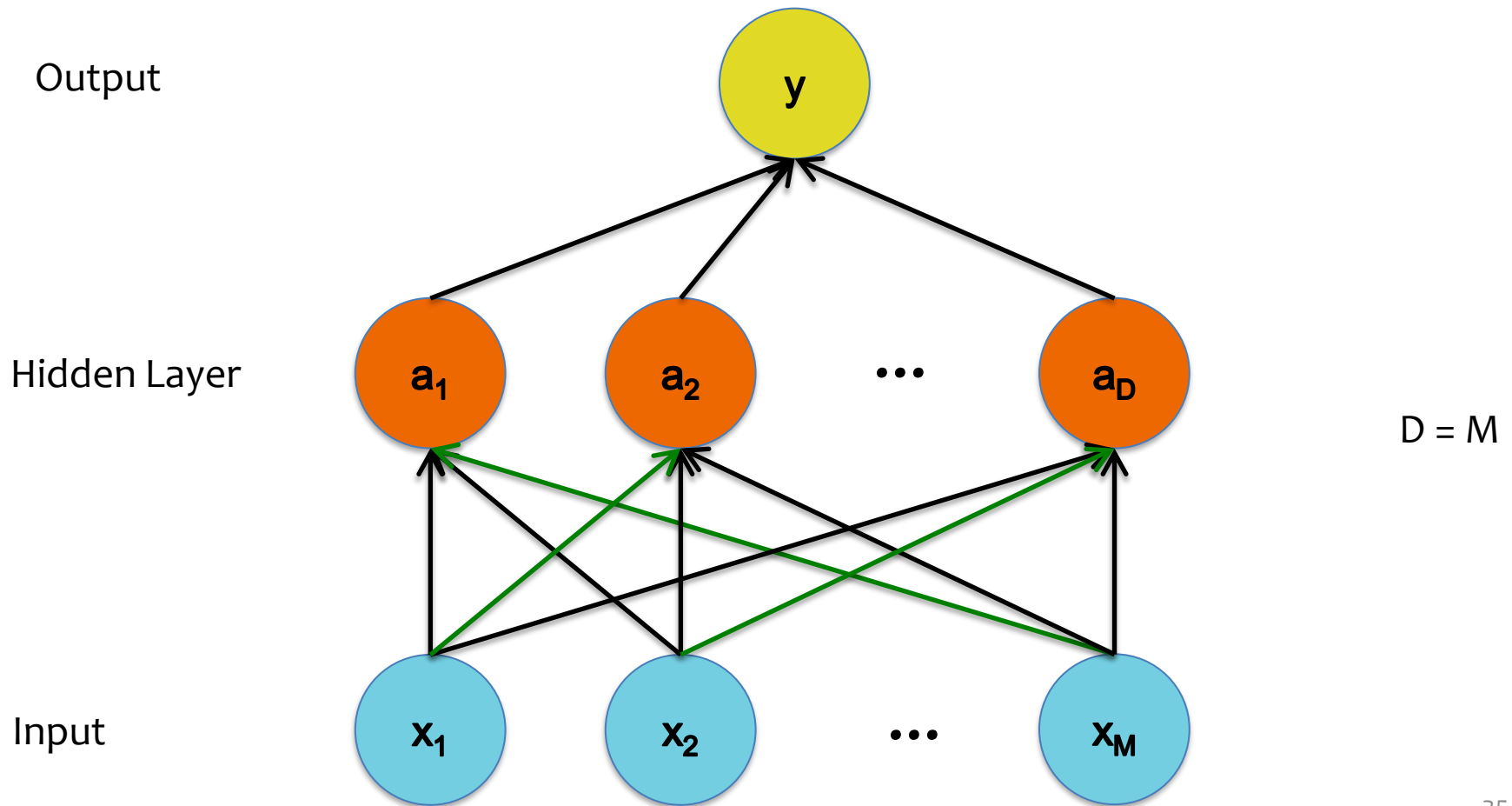
Building a Neural Net



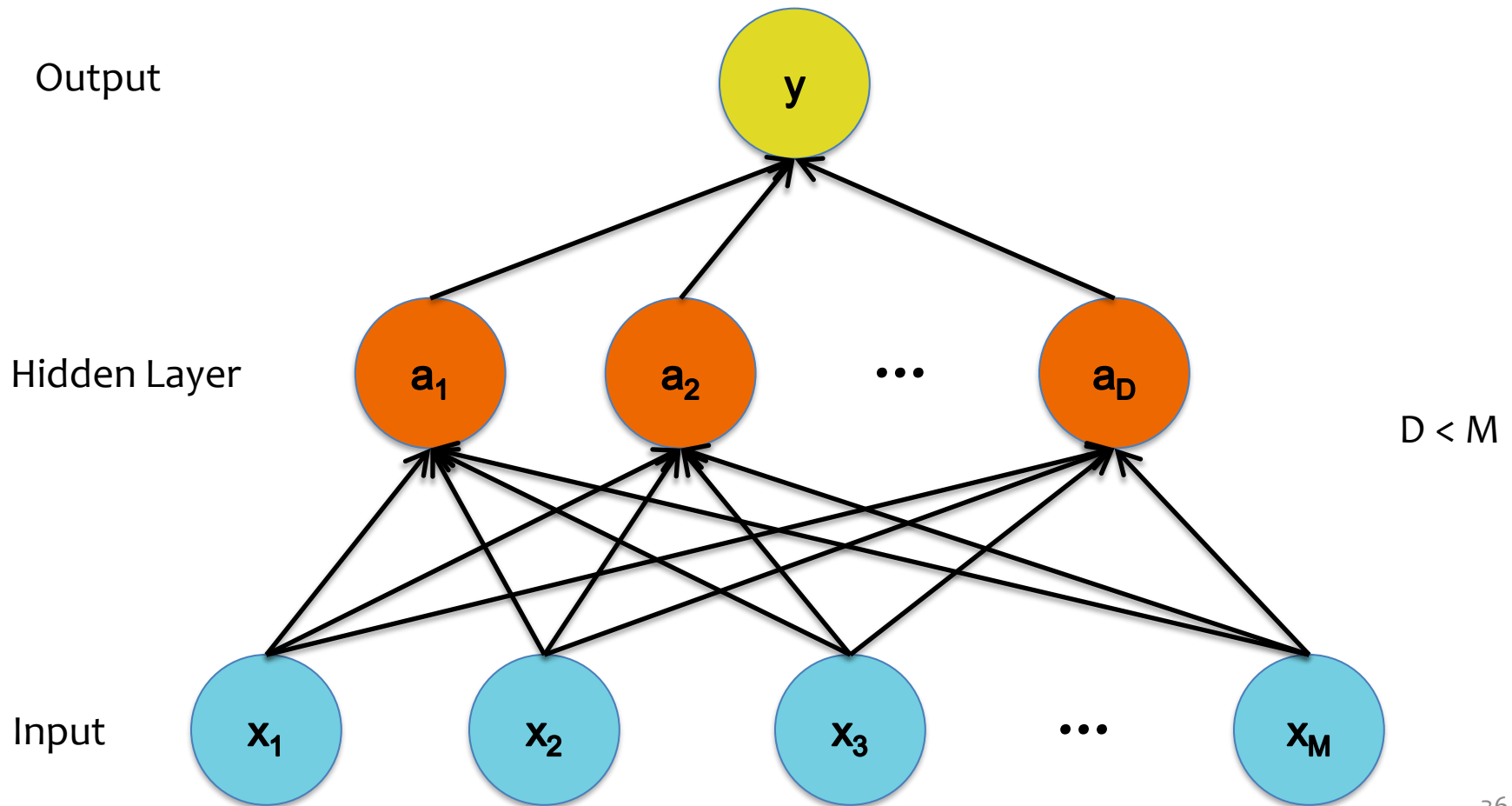
Building a Neural Net



Building a Neural Net

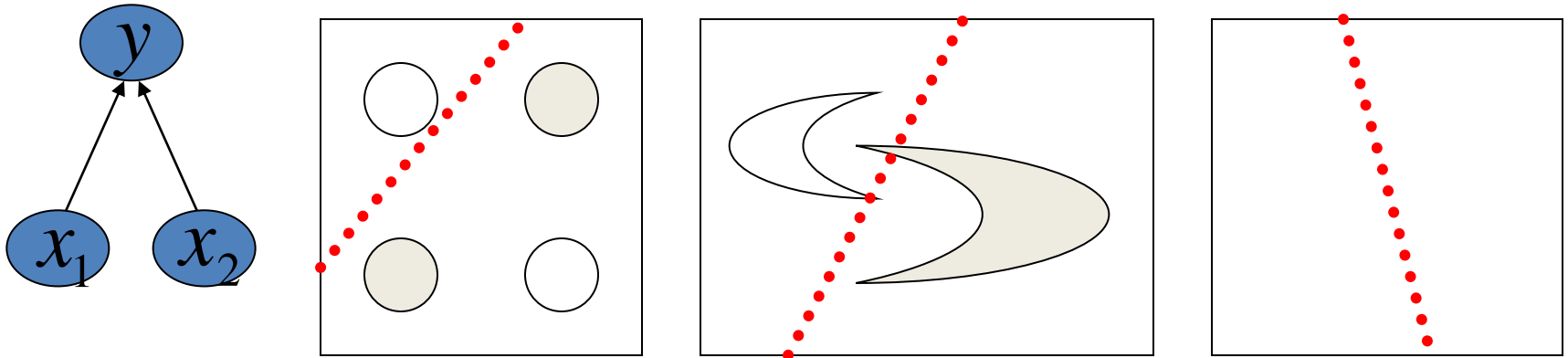


Building a Neural Net



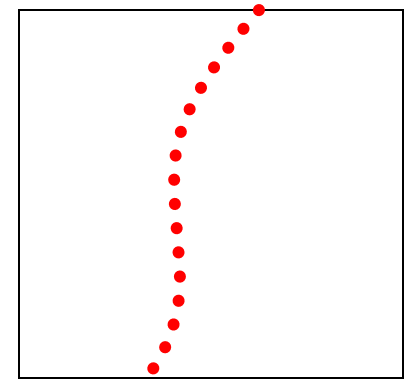
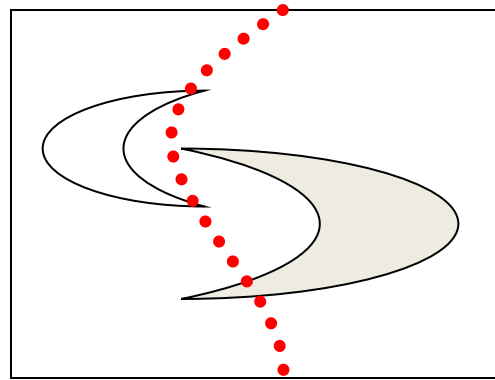
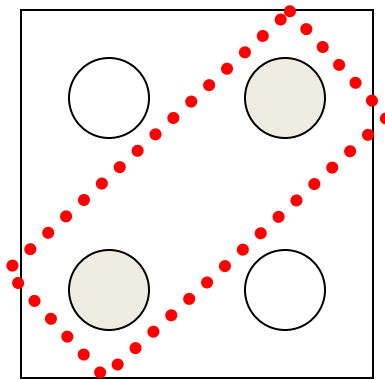
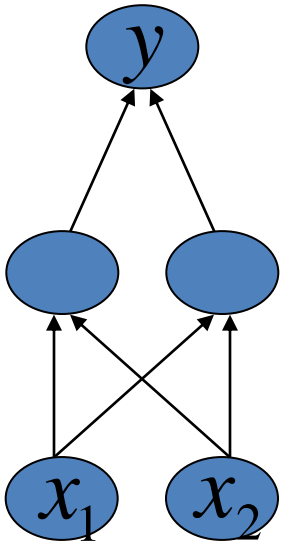
Decision Boundary

- 0 hidden layers: linear classifier
 - Hyperplanes

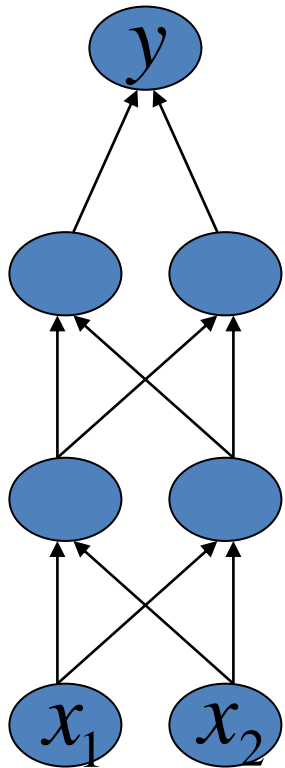


Decision Boundary

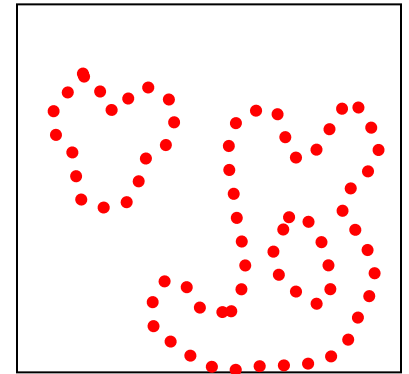
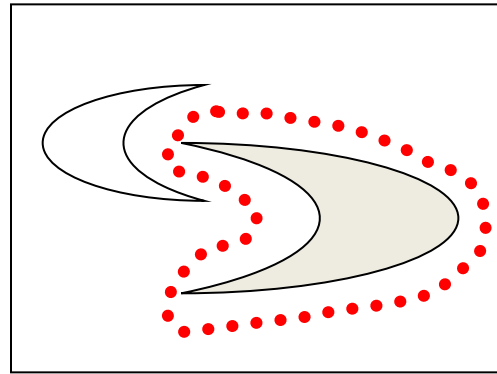
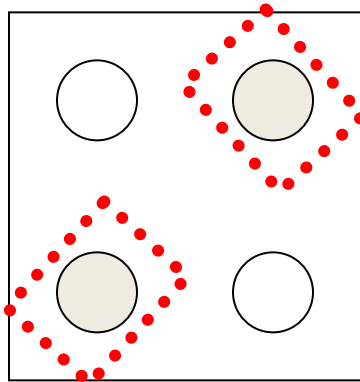
- 1 hidden layer
 - Boundary of convex region (open or closed)

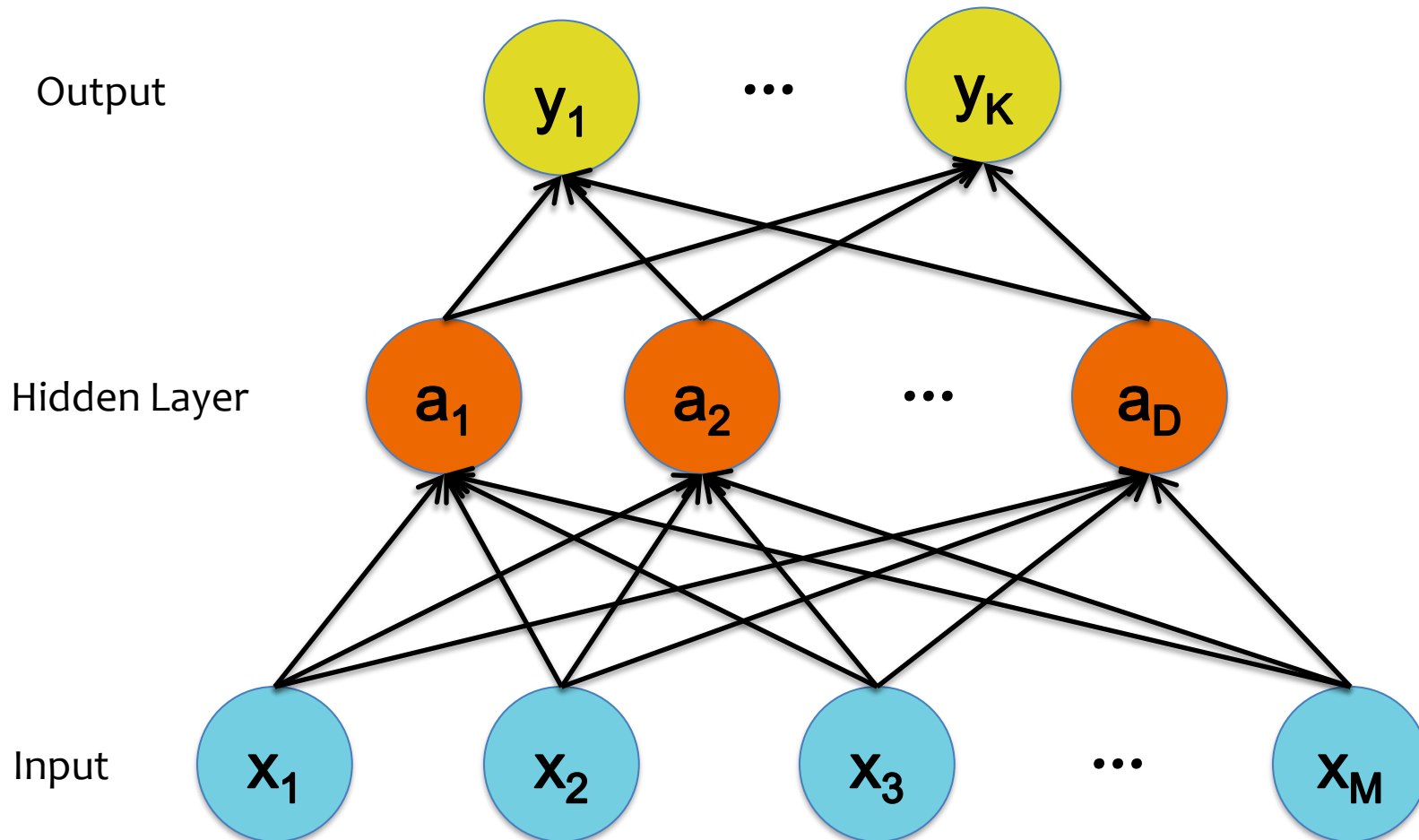


Decision Boundary

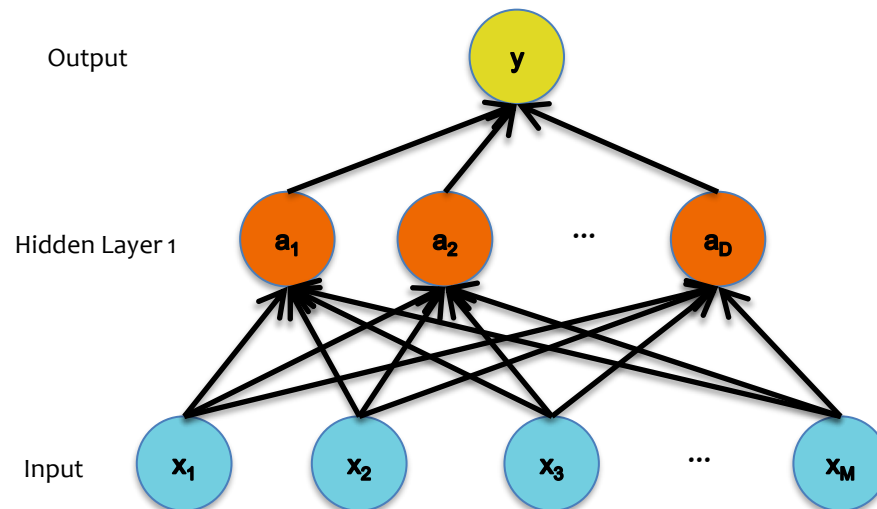


- 2 hidden layers
 - Combinations of convex regions

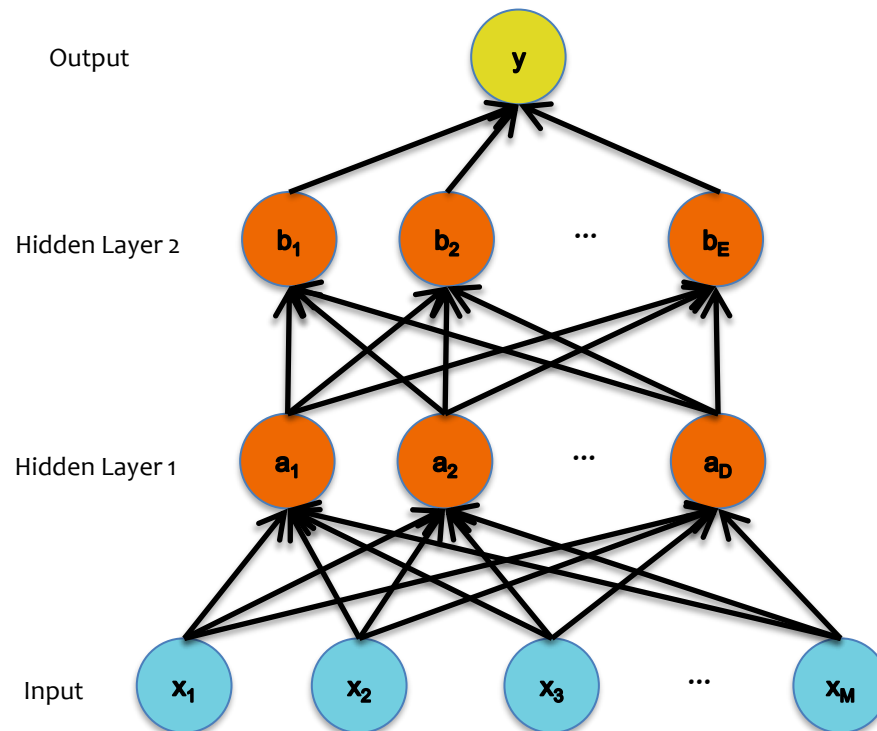




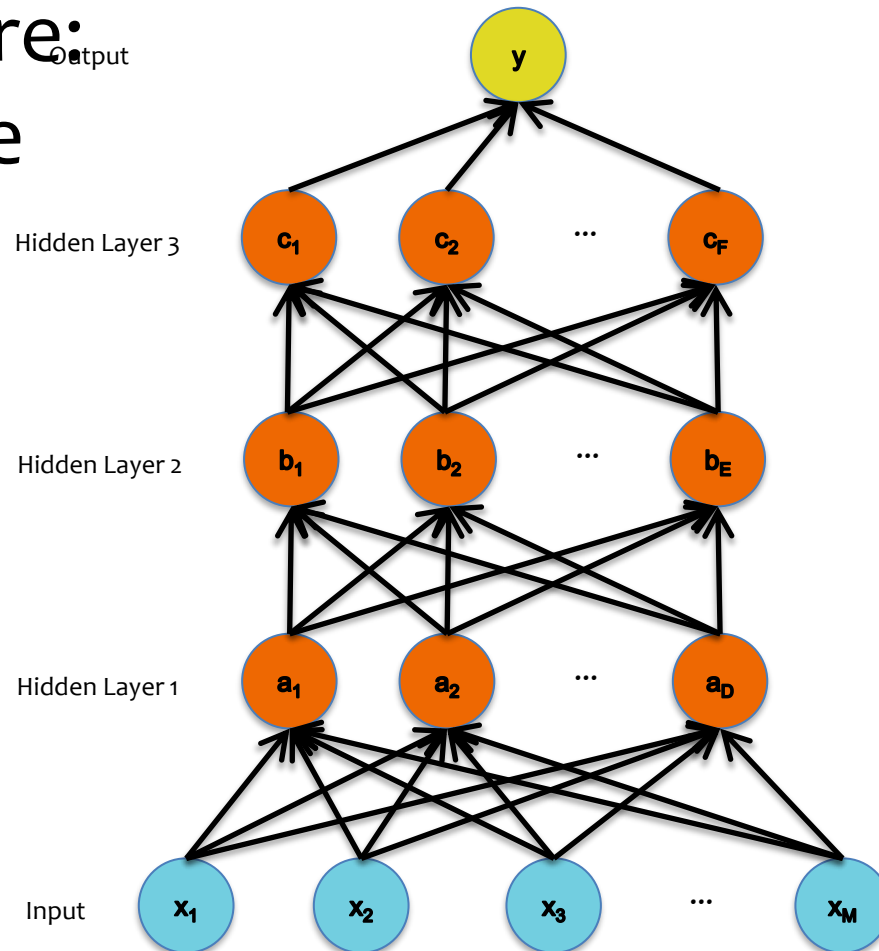
Next lecture:



Next lecture:



Next lecture:
Making the
neural
networks
deeper

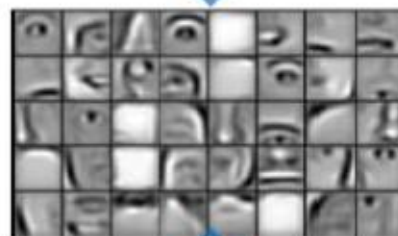


- We don't know the “right” levels of abstraction
- So let the model figure it out!

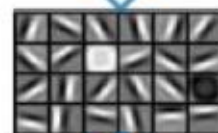
Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”



1st layer
“Edges”



Pixels

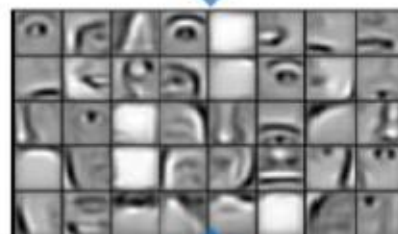
Face Recognition:

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

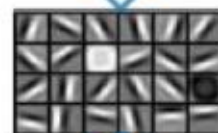
Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”



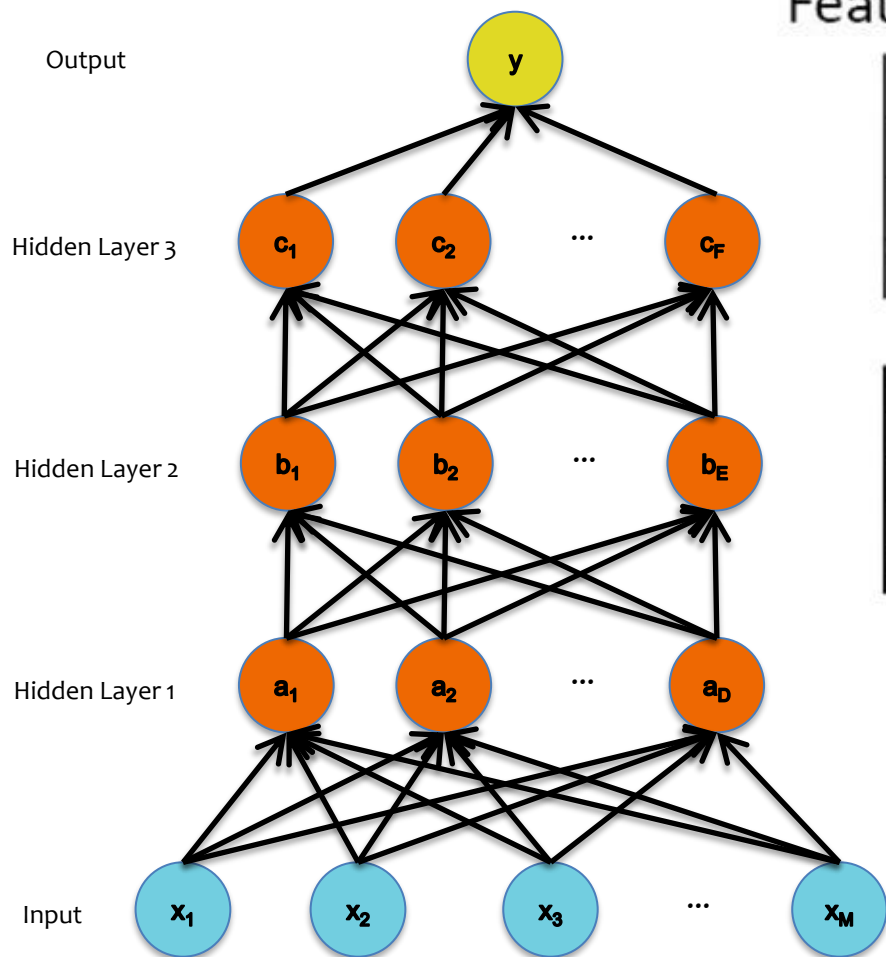
1st layer
“Edges”



Pixels

Decision Functions

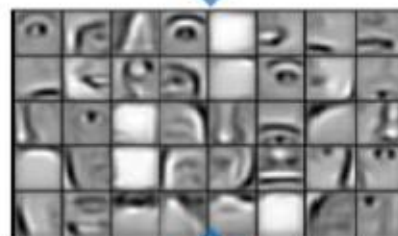
Different Levels of Abstraction



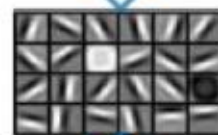
Feature representation



3rd layer
"Objects"



2nd layer
"Object parts"



1st layer
"Edges"



Pixels

ARCHITECTURES

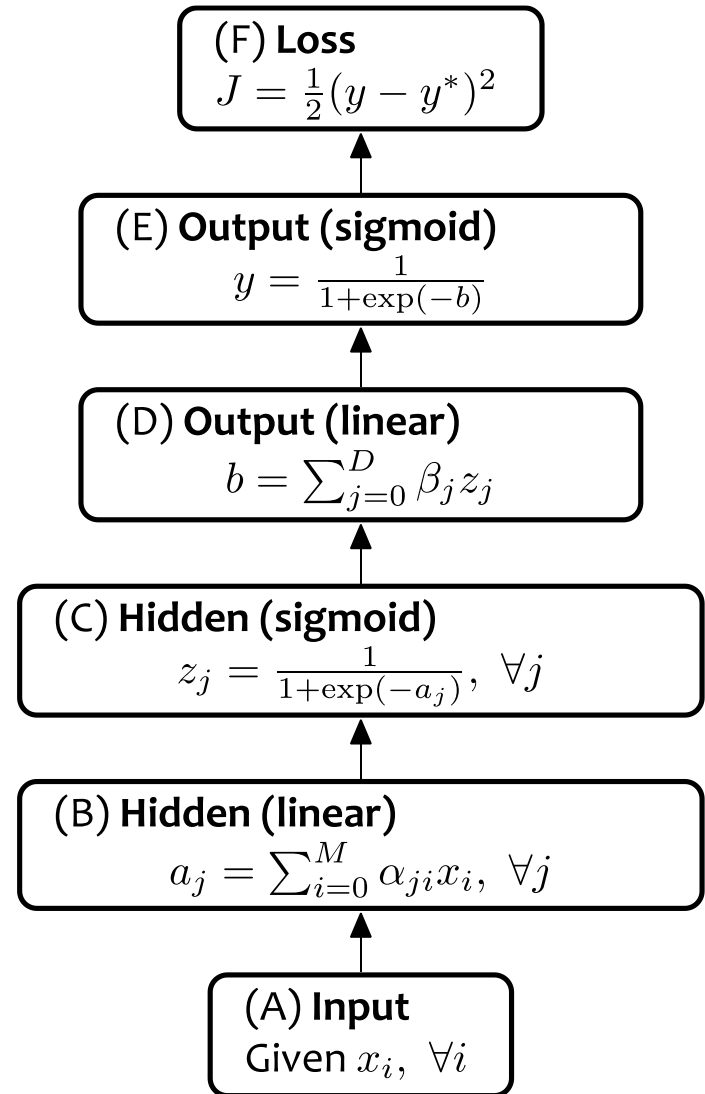
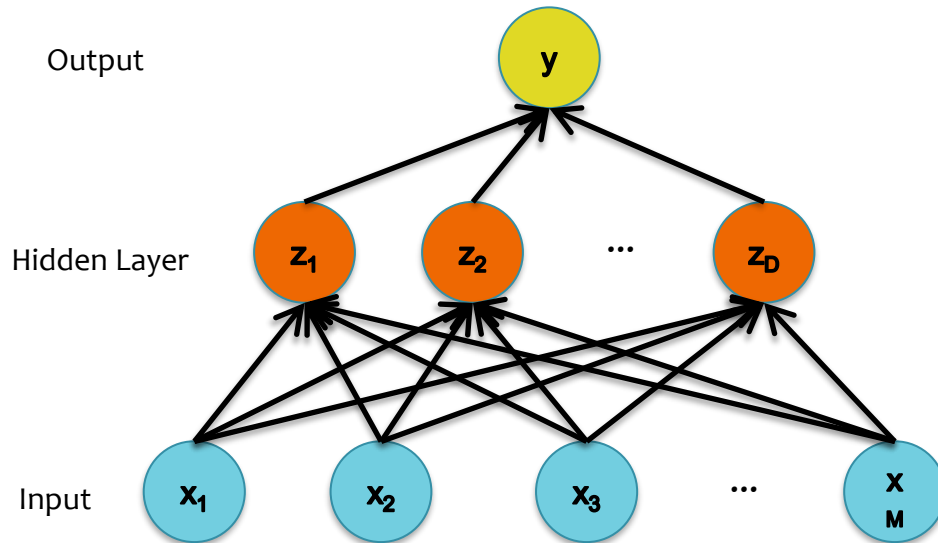
Neural Network Architectures

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

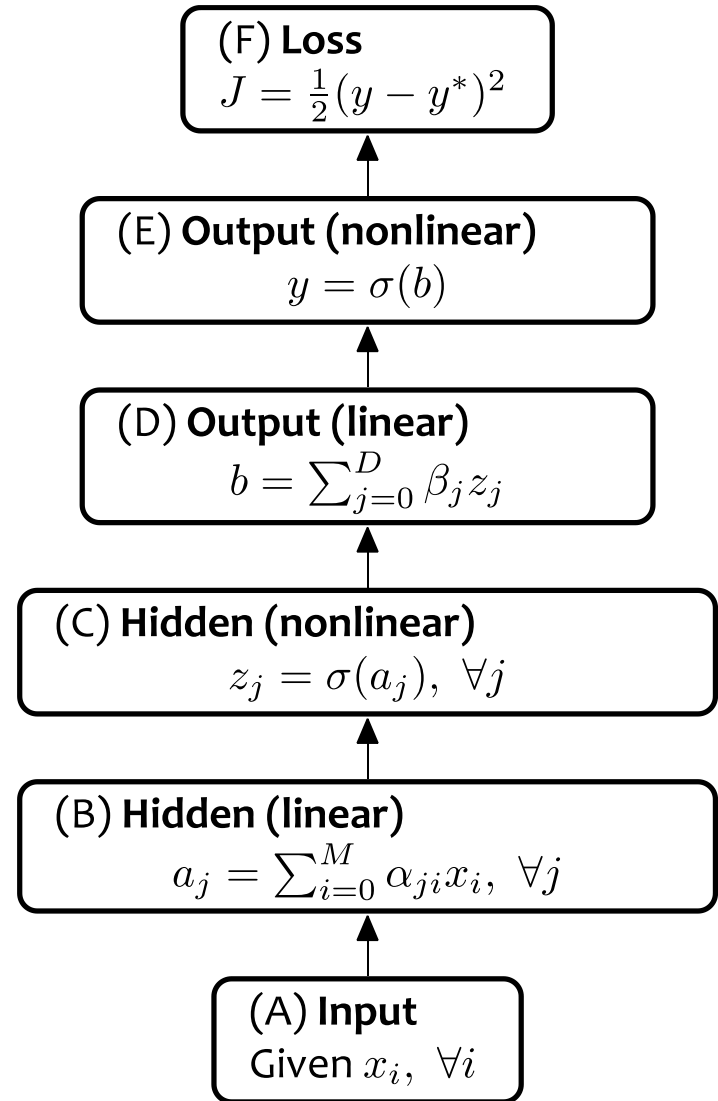
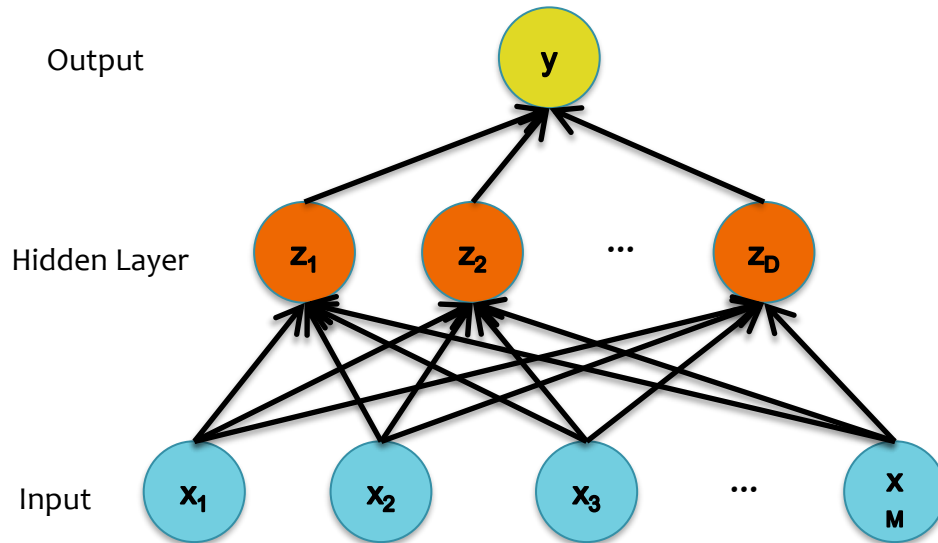
Activation Functions

Neural Network with sigmoid
activation functions



Activation Functions

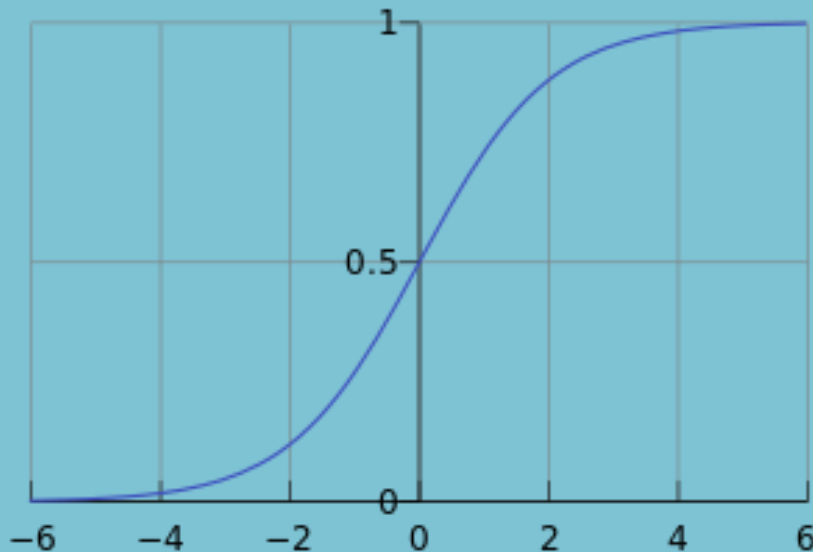
Neural Network with arbitrary nonlinear activation functions



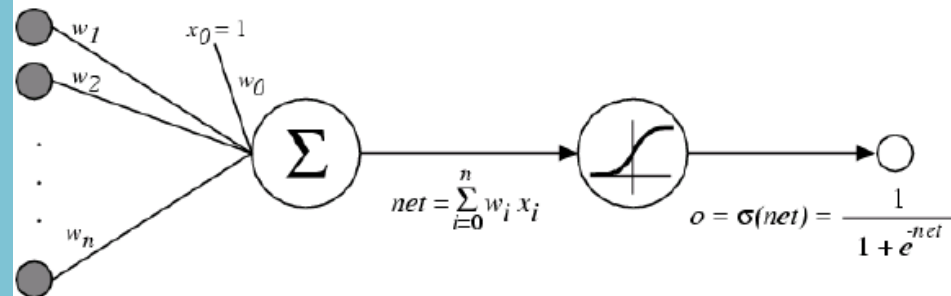
Activation Functions

Sigmoid / Logistic Function

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

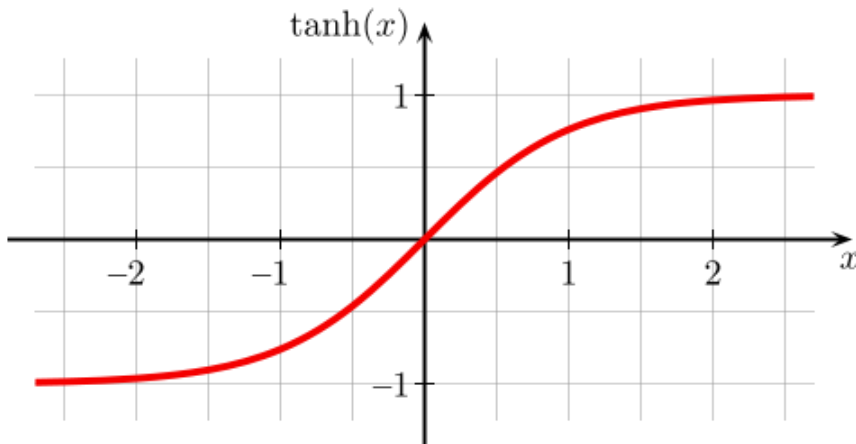


So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function...



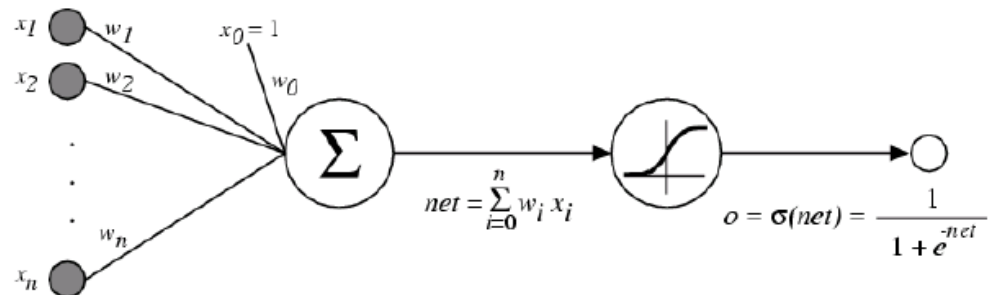
Activation Functions

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



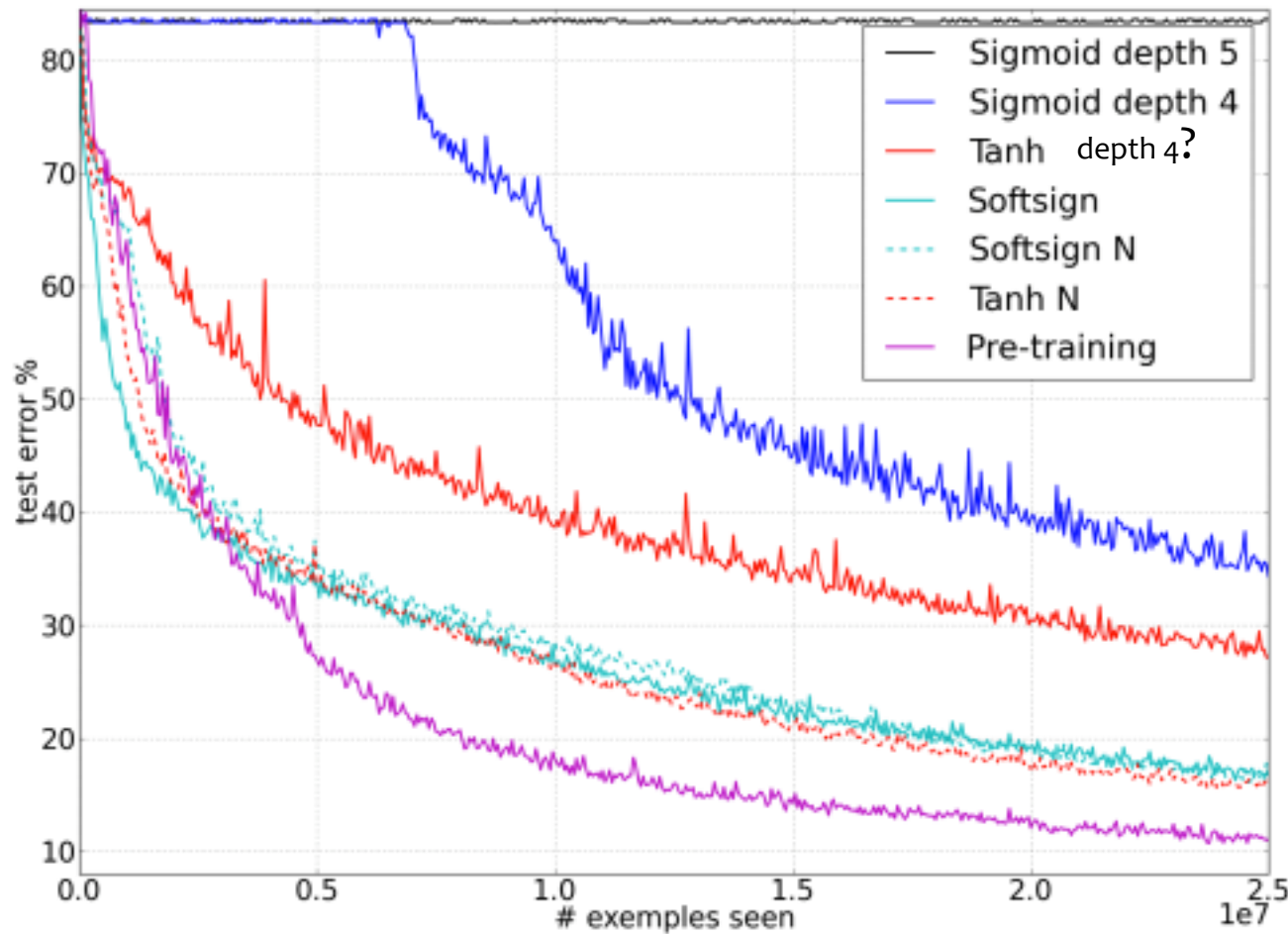
Alternate 1:
tanh

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



Understanding the difficulty of training deep feedforward neural networks

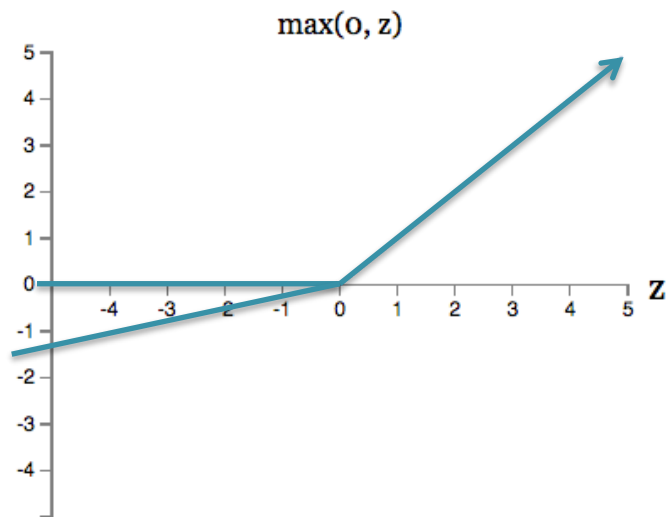
AI Stats 2010



} sigmoid
vs.
tanh

Activation Functions

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

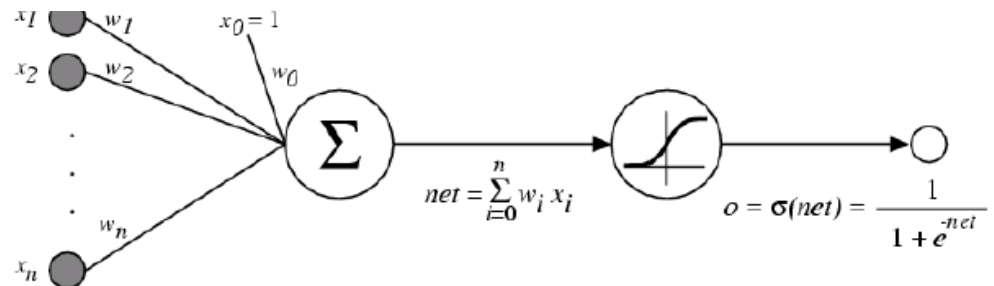


$$\max(0, w \cdot x + b).$$

Alternate 2: rectified linear unit

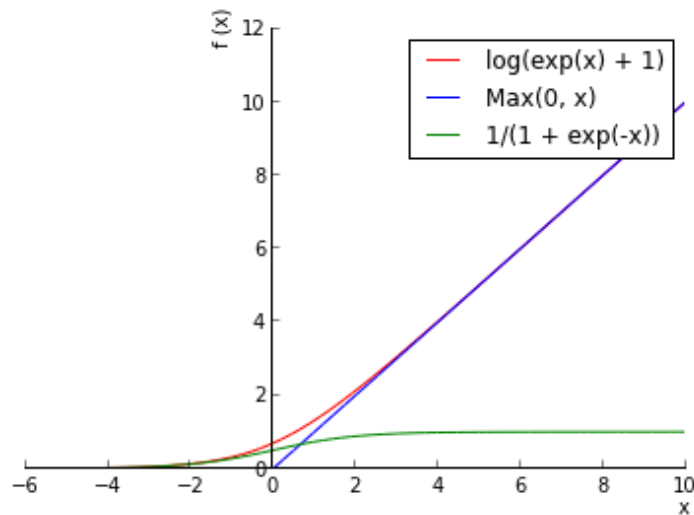
Linear with a cutoff at zero

(Implementation: clip the gradient when you pass zero)



Activation Functions

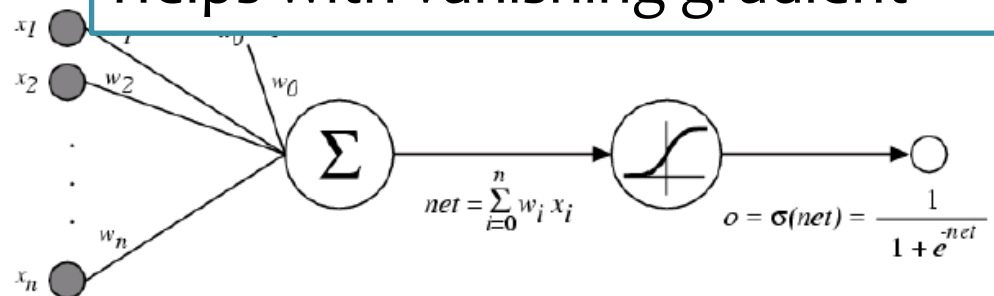
- A new change: modifying the nonlinearity
 - reLU often used in vision tasks



Alternate 2: rectified linear unit

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

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

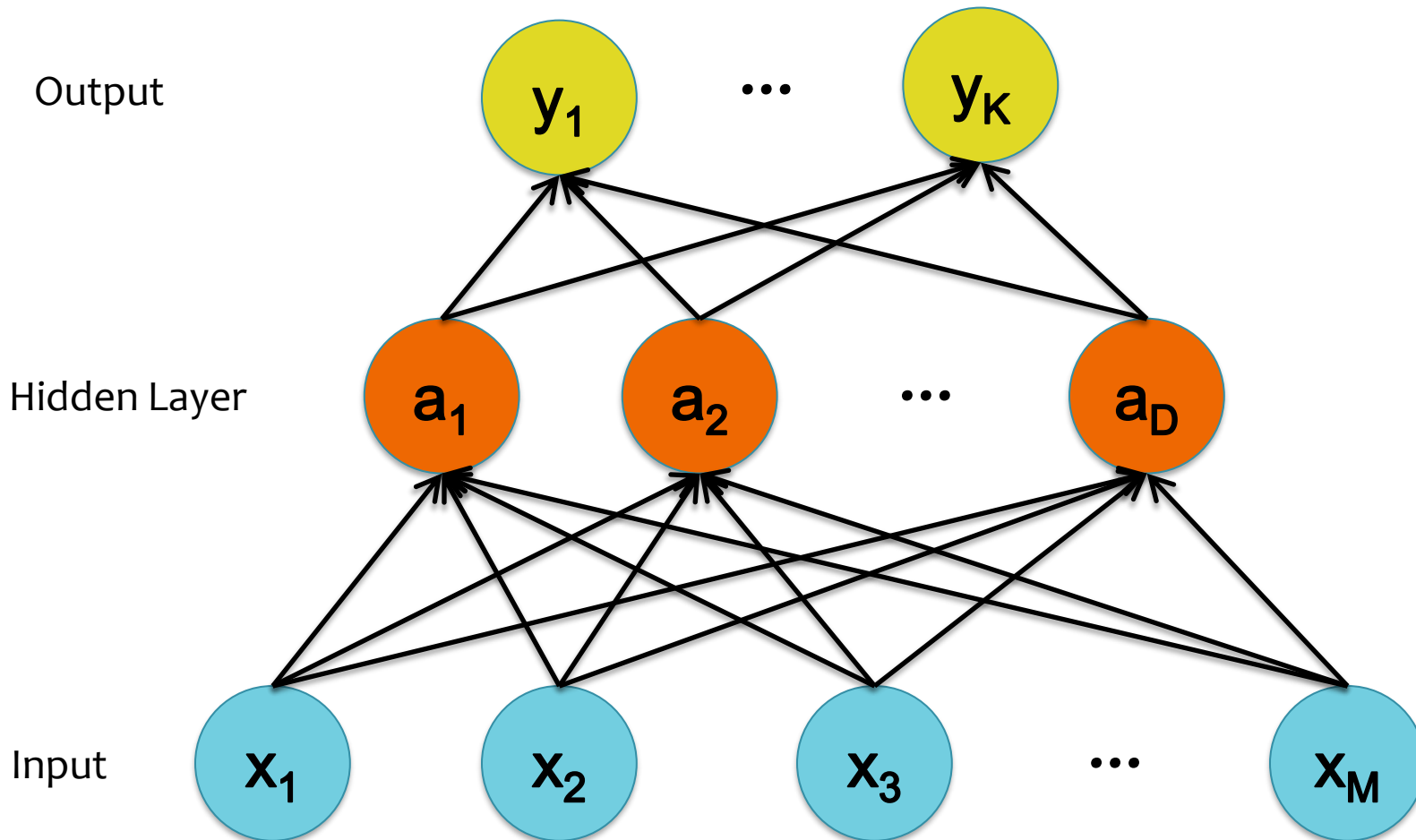


Objective Functions for NNs

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

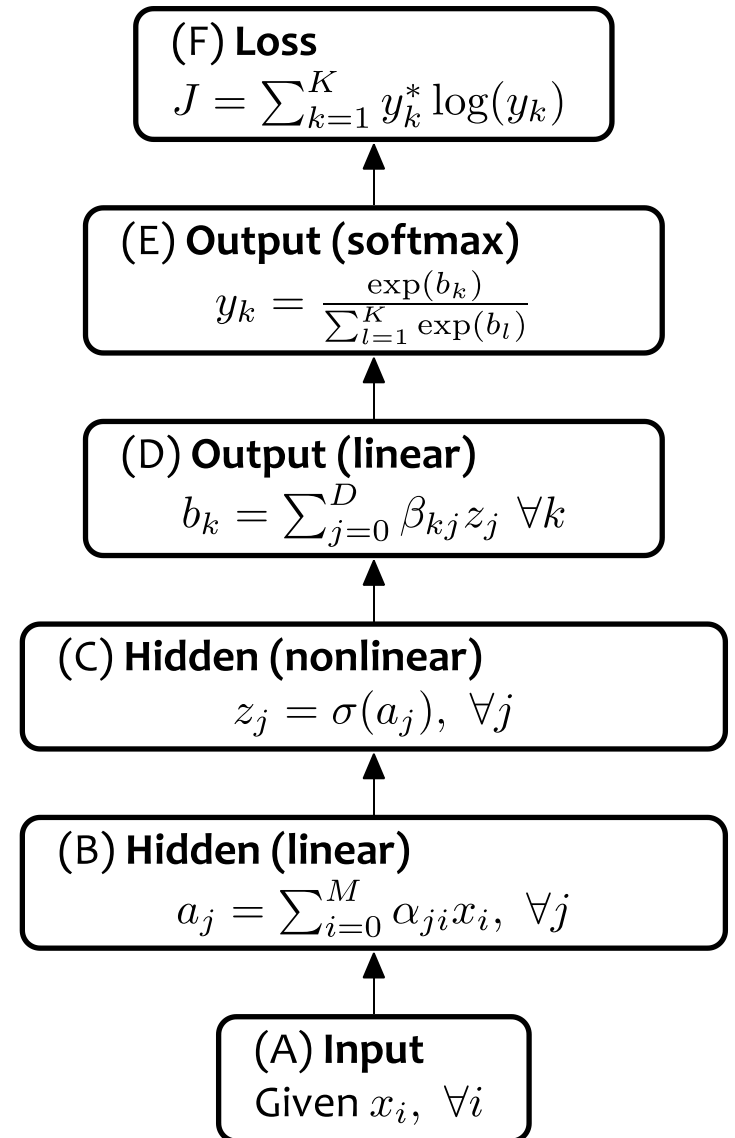
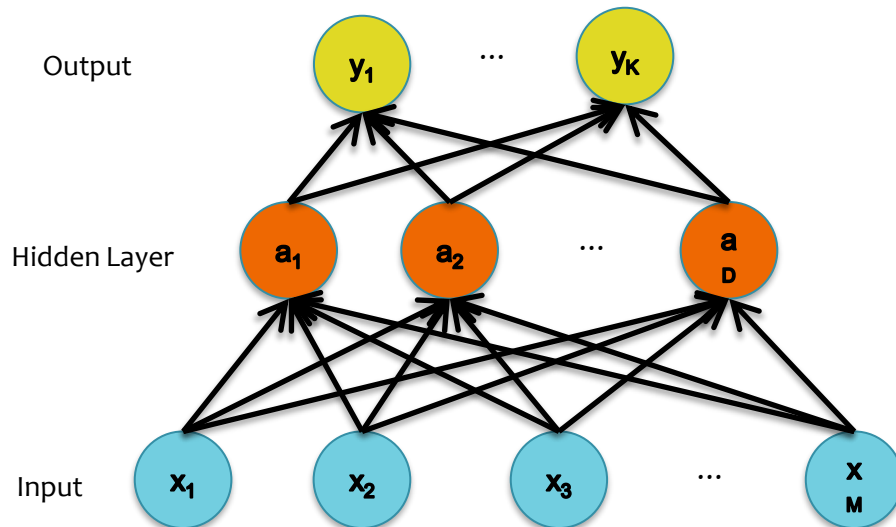
Multi-Class Output



Multi-Class Output

Softmax:

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



Cross-entropy vs. Quadratic loss

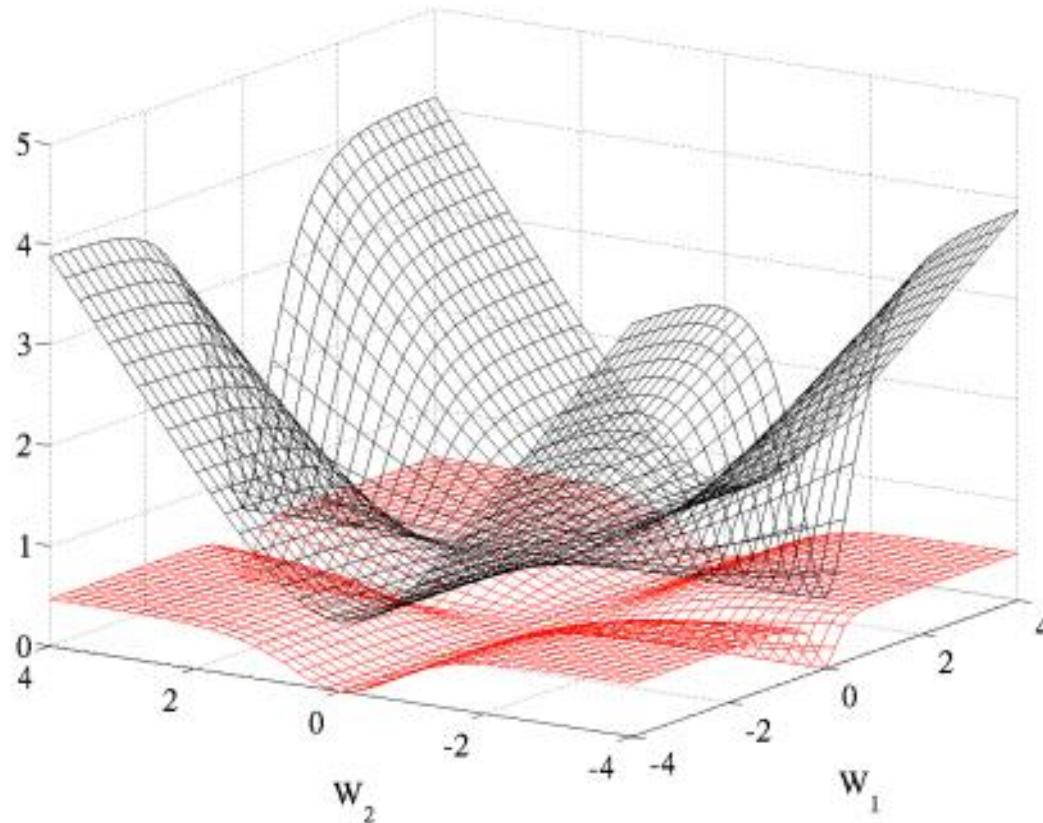


Figure 5: Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.

Background

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of these:

– Decision function

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

– Loss function

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

3. Define goal:

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

4. Train with SGD:

(take small steps
opposite the gradient)

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

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:

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

2. Choose each of these:

– Decision function

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

– Loss function

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

3. Define goal:

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

4. Train with SGD:

(take small steps
opposite the gradient)

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

- **Question 1:**

When can we compute the gradients of the parameters of an arbitrary neural network?

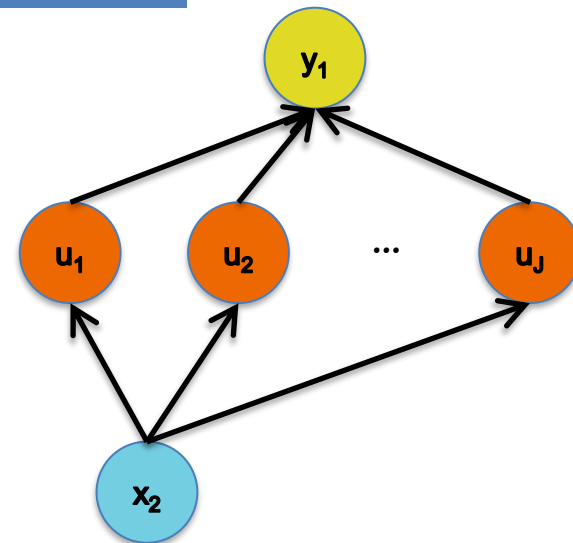
- **Question 2:**

When can we make the gradient computation efficient?

Given: $y = g(u)$ and $u = h(x)$.

Chain Rule:

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

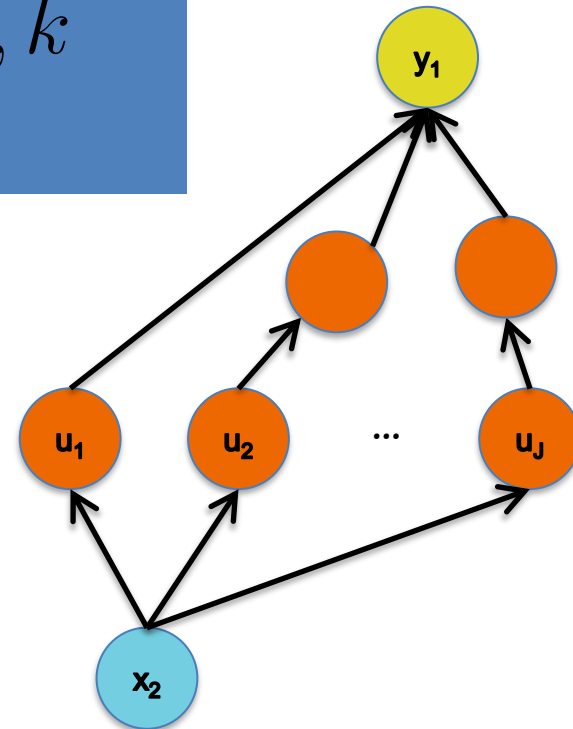


Given: $y = g(u)$ and $u = h(x)$.

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.



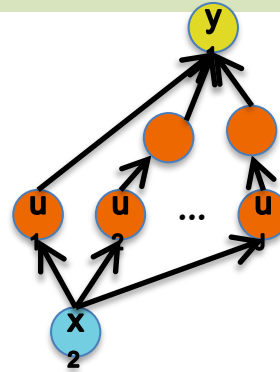
Training

Chain Rule

Given: $y = g(u)$ and $u = h(x)$.

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**

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

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

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

$$J = \cos(u)$$

$$u = u_1 + u_2$$

$$u_1 = \sin(t)$$

$$u_2 = 3t$$

$$t = x^2$$

Backward

$$\frac{dJ}{du} += -\sin(u)$$

$$\frac{dJ}{du_1} += \frac{dJ}{du} \frac{du}{du_1}, \quad \frac{du}{du_1} = 1 \qquad \frac{dJ}{du_2} += \frac{dJ}{du} \frac{du}{du_2}, \quad \frac{du}{du_2} = 1$$

$$\frac{dJ}{dt} += \frac{dJ}{du_1} \frac{du_1}{dt}, \quad \frac{du_1}{dt} = \cos(t)$$

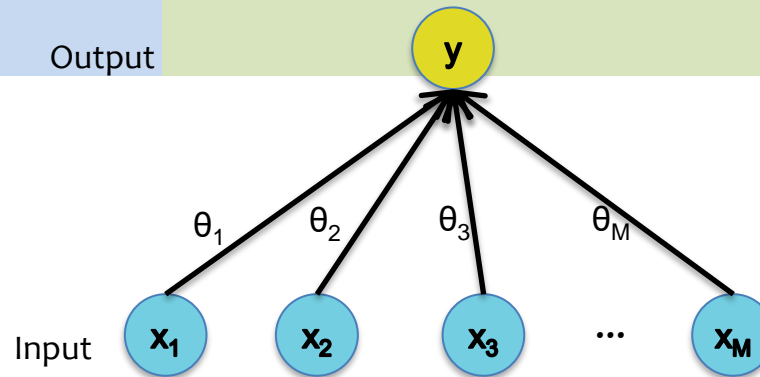
$$\frac{dJ}{dt} += \frac{dJ}{du_2} \frac{du_2}{dt}, \quad \frac{du_2}{dt} = 3$$

$$\frac{dJ}{dx} += \frac{dJ}{dt} \frac{dt}{dx}, \quad \frac{dt}{dx} = 2x$$

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

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

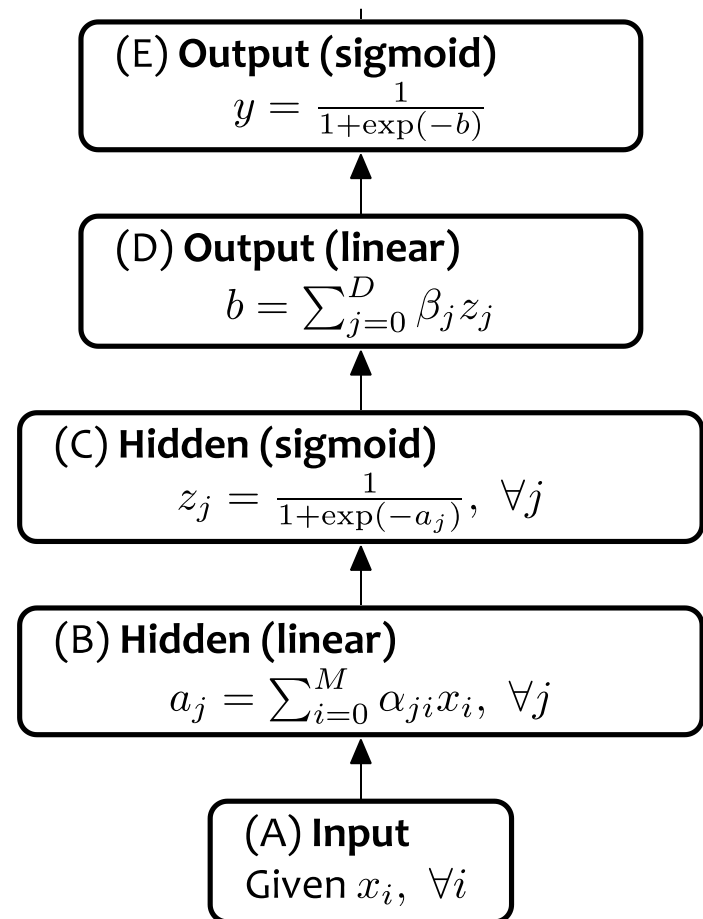
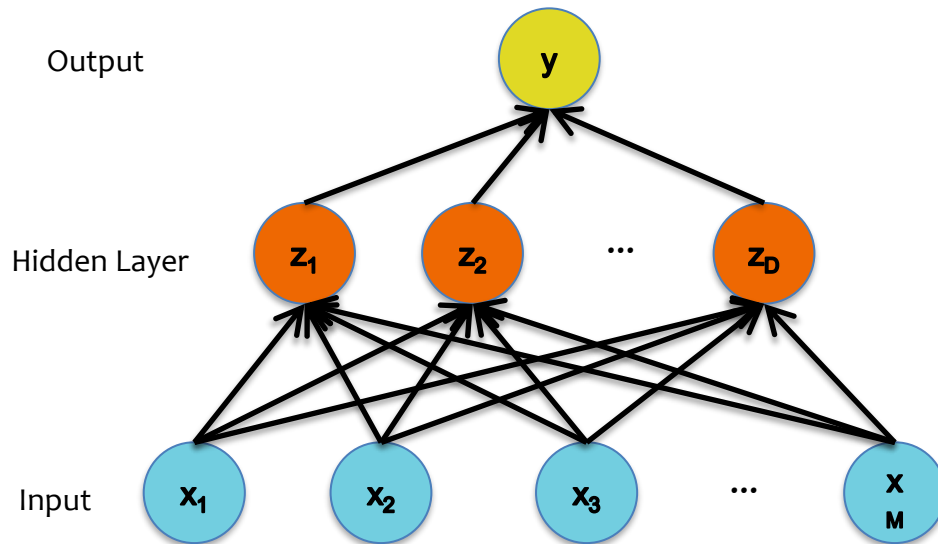
$$\frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da}, \quad \frac{dy}{da} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$$

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

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

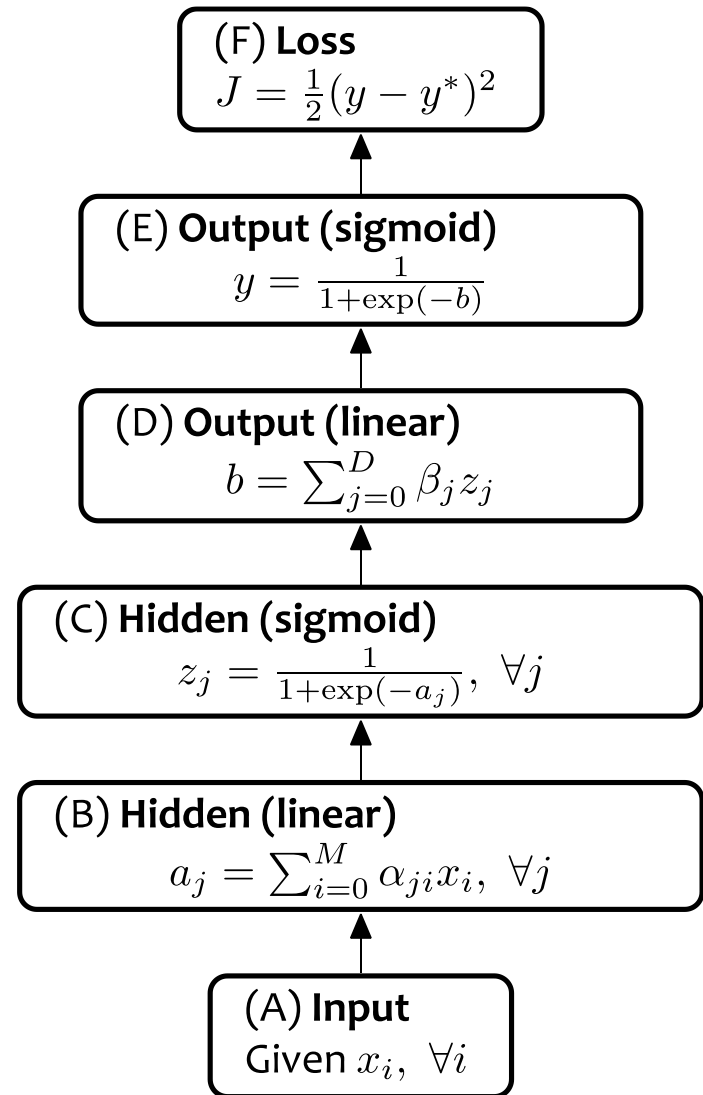
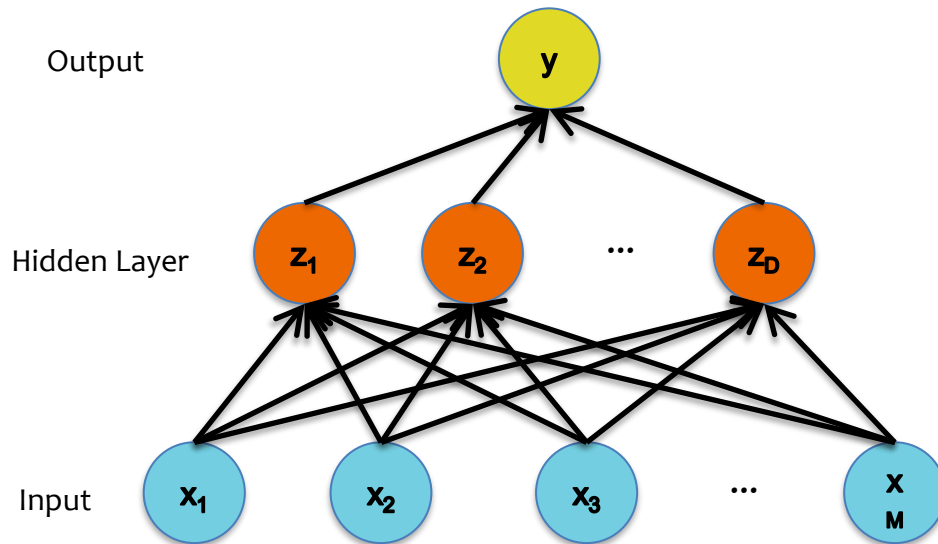
Training

Backpropagation



Training

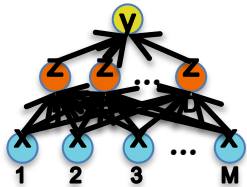
Backpropagation



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

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

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

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

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

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

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

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

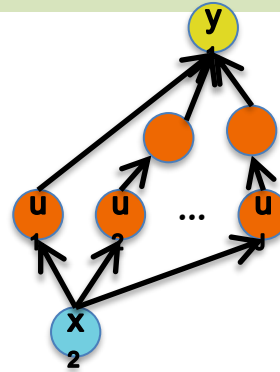
Training

Chain Rule

Given: $y = g(u)$ and $u = h(x)$.

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**

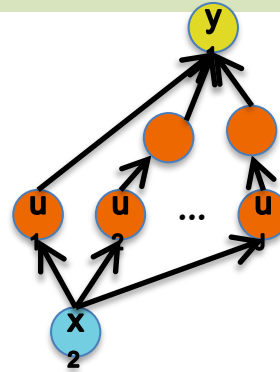
Training

Chain Rule

Given: $y = g(u)$ and $u = h(x)$.

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

A Recipe for Gradients

1. Given training data

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

2. Choose each of the

- Decision function

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


- Loss function

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

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!

opposite the gradient)


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

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