



# 10-301/10-601 Introduction to Machine Learning

Machine Learning Department  
School of Computer Science  
Carnegie Mellon University

## Neural Networks

Hoda Heidari, Henry Chai, Matt Gormley

Lecture 11

Feb. 21, 2024

# Reminders

- **Homework 4: Logistic Regression**
  - Out: Mon, Feb 19
  - Due: Wed, Feb 28 at 11:59pm
- **Piazza discussion: Society, Ethics, & ML**
  - participation credit

# **A RECIPE FOR ML**

# 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



# 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

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

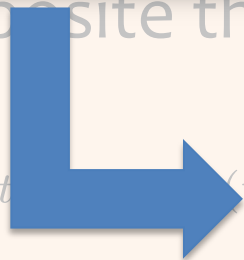
- Loss function

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

**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

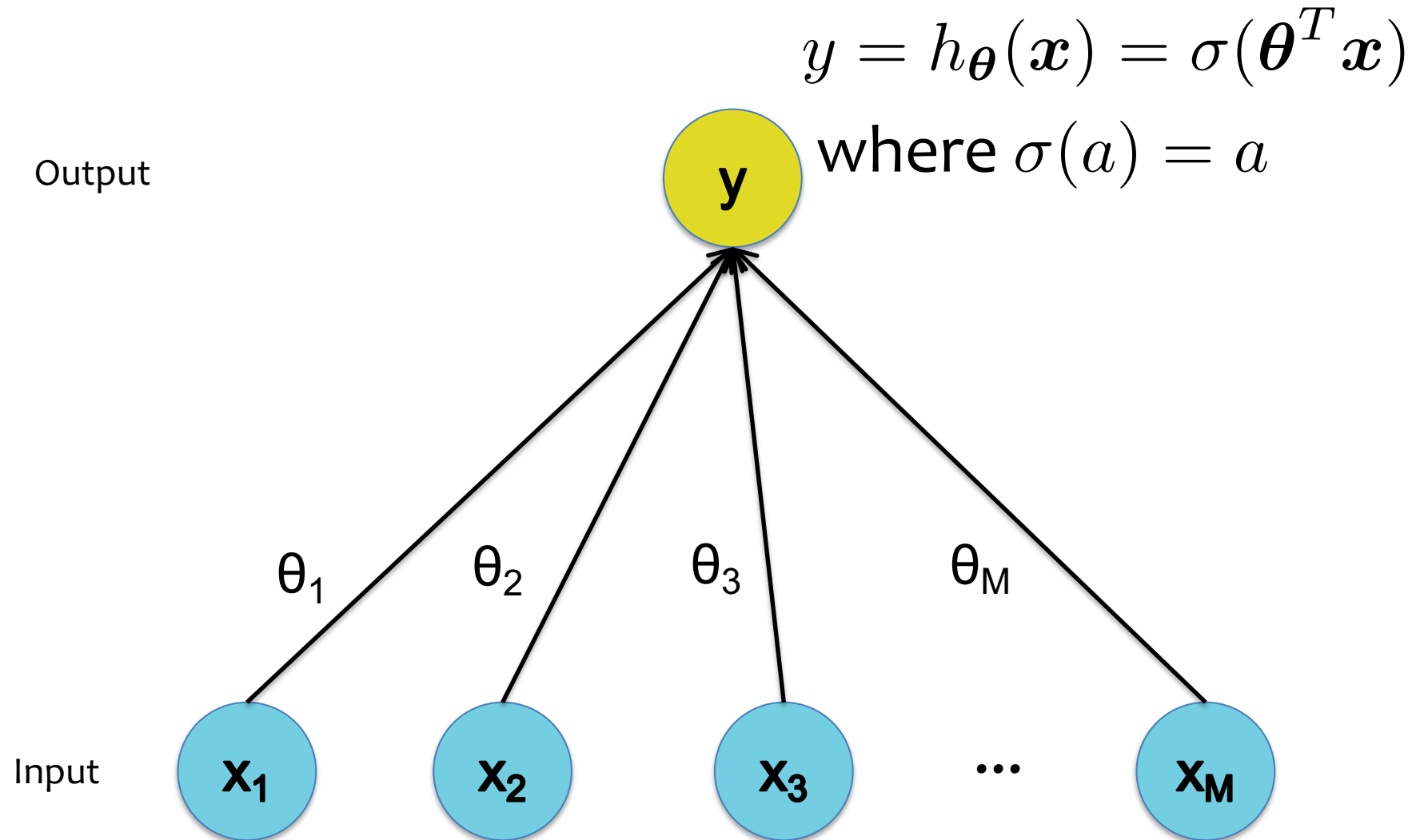
$$\ell(\hat{y}, y_i) \in \mathbb{R}$$

4. Train with SGD:

(Take small steps opposite the gradient)

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

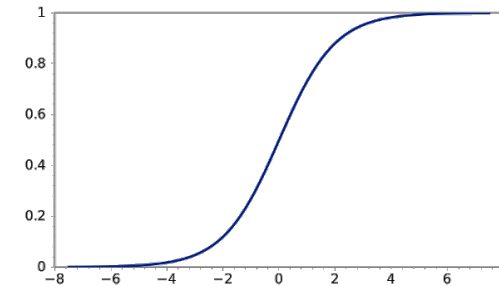
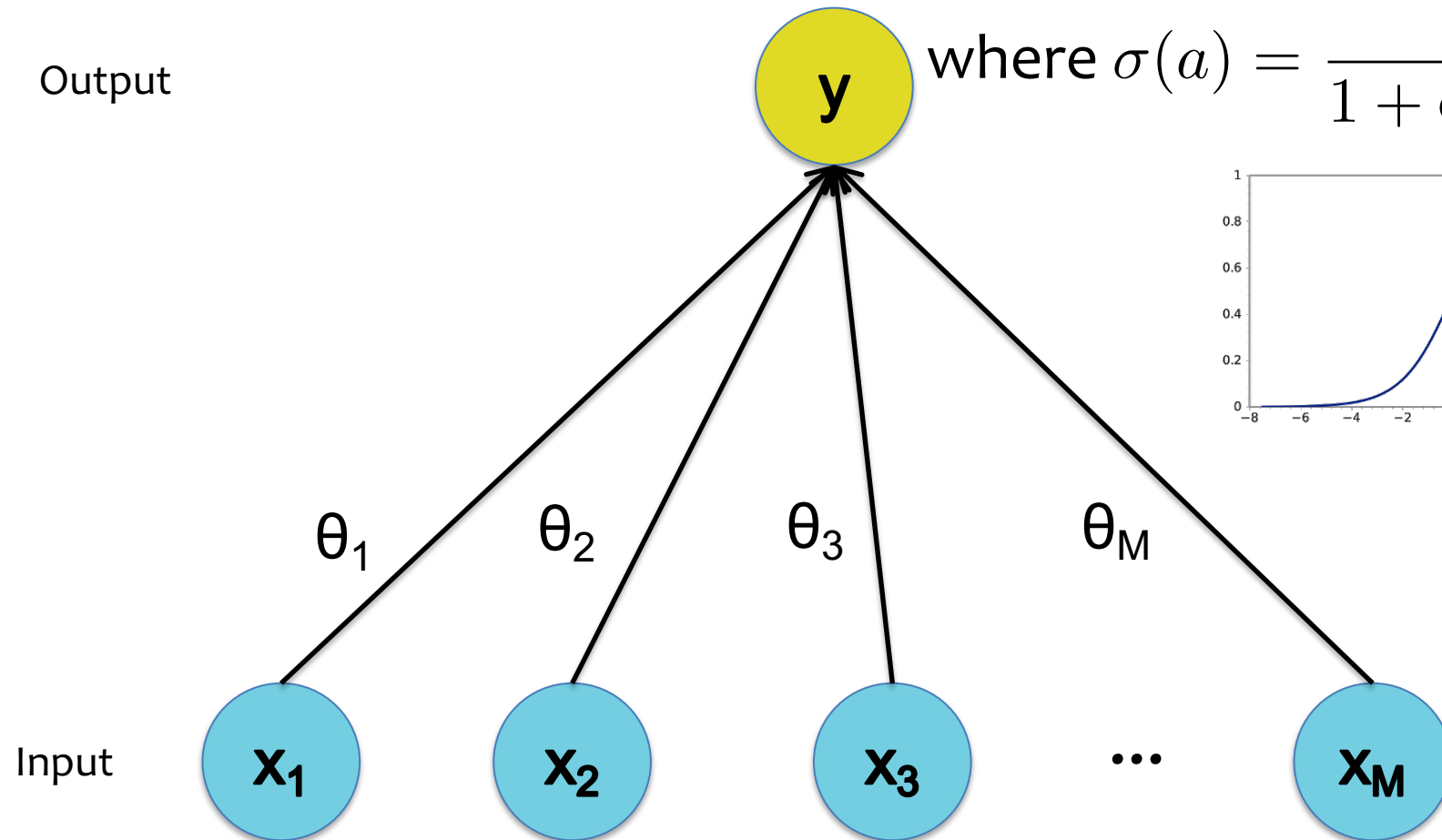




$$y = h_{\theta}(x) = \sigma(\theta^T x)$$

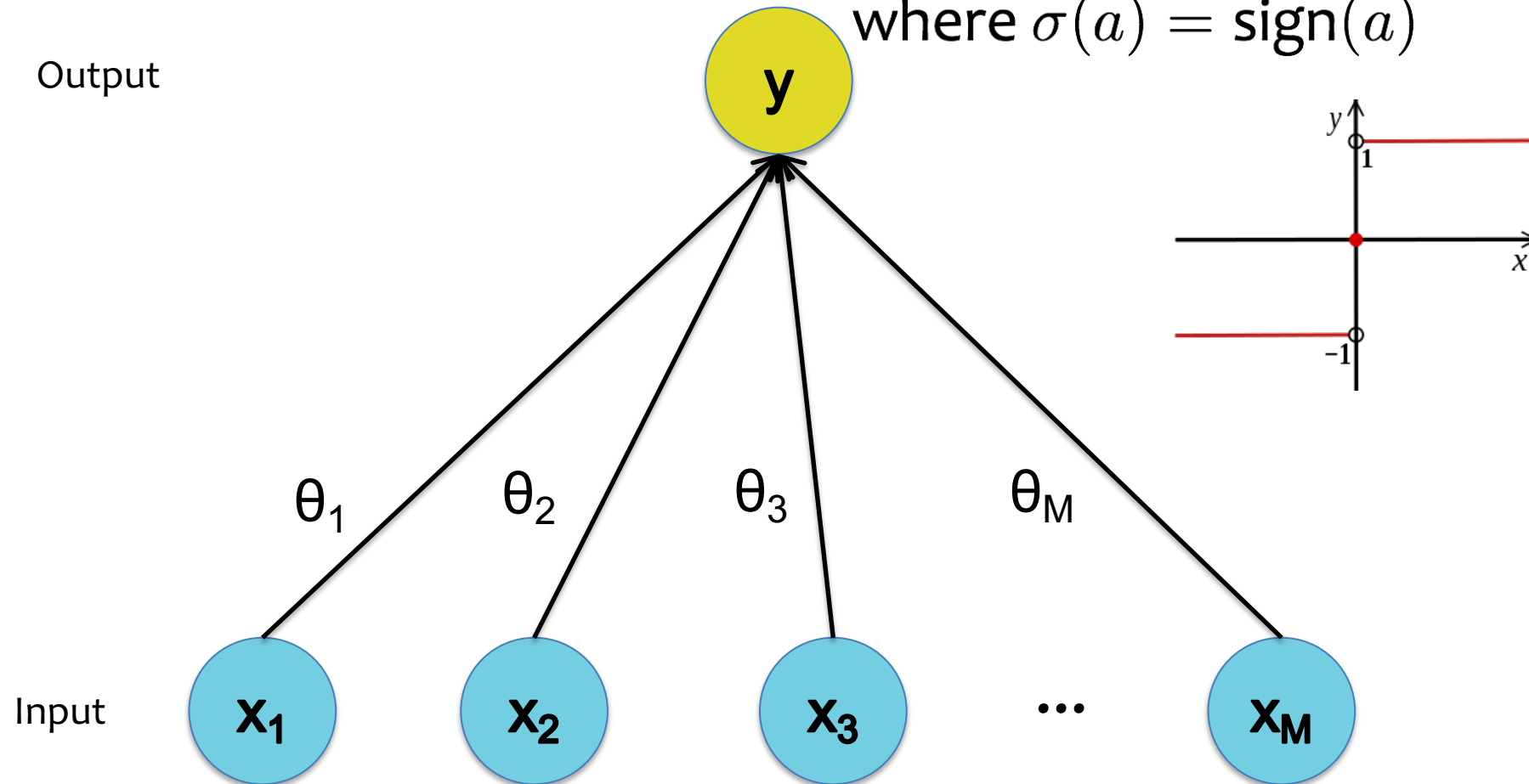
$$\text{where } \sigma(a) = \frac{1}{1 + \exp(-a)}$$

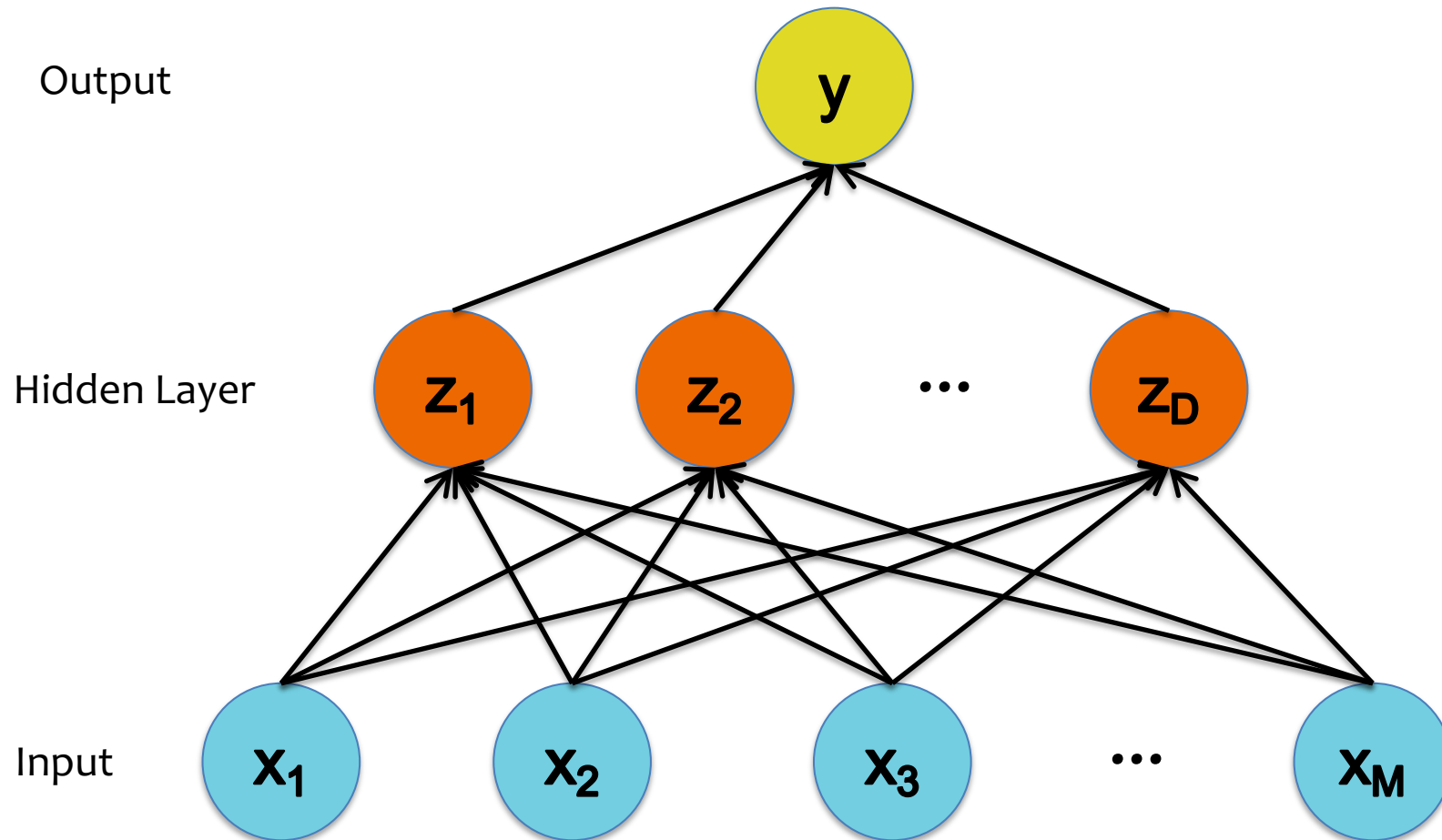
Output



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

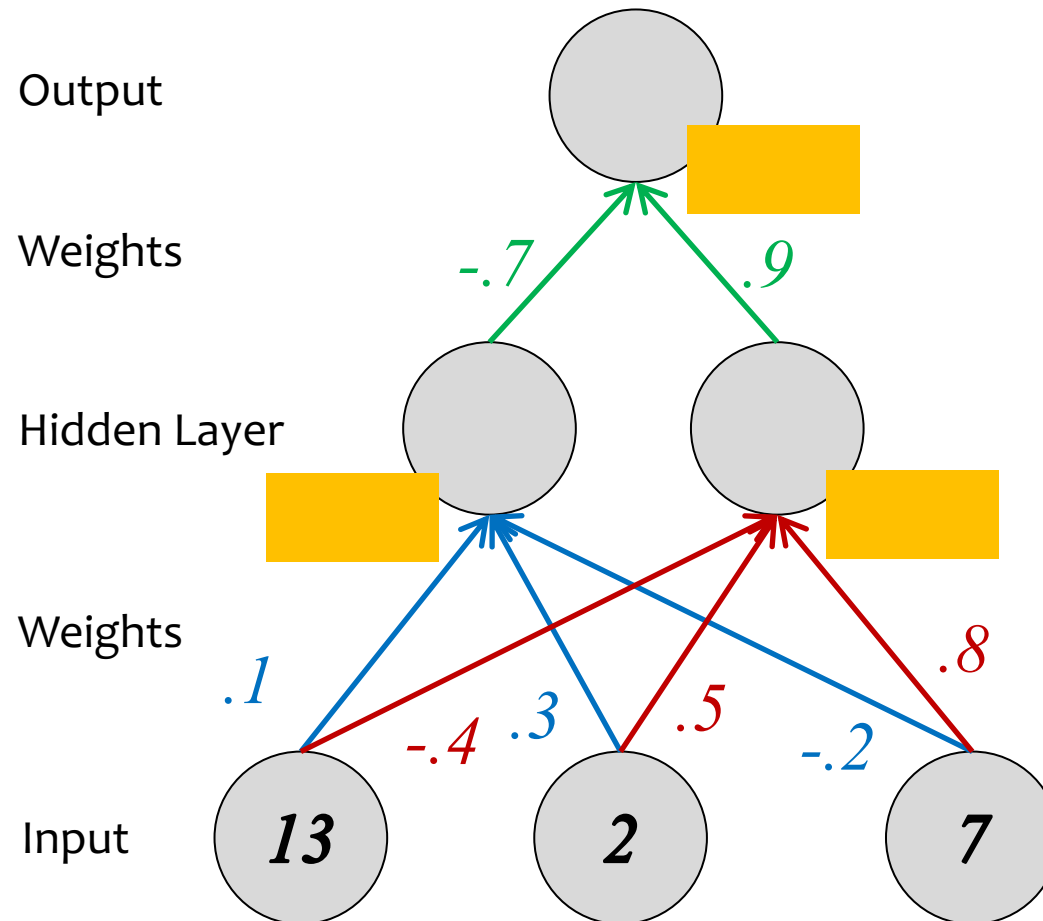
where  $\sigma(a) = \text{sign}(a)$





# **COMPONENTS OF A NEURAL NETWORK**



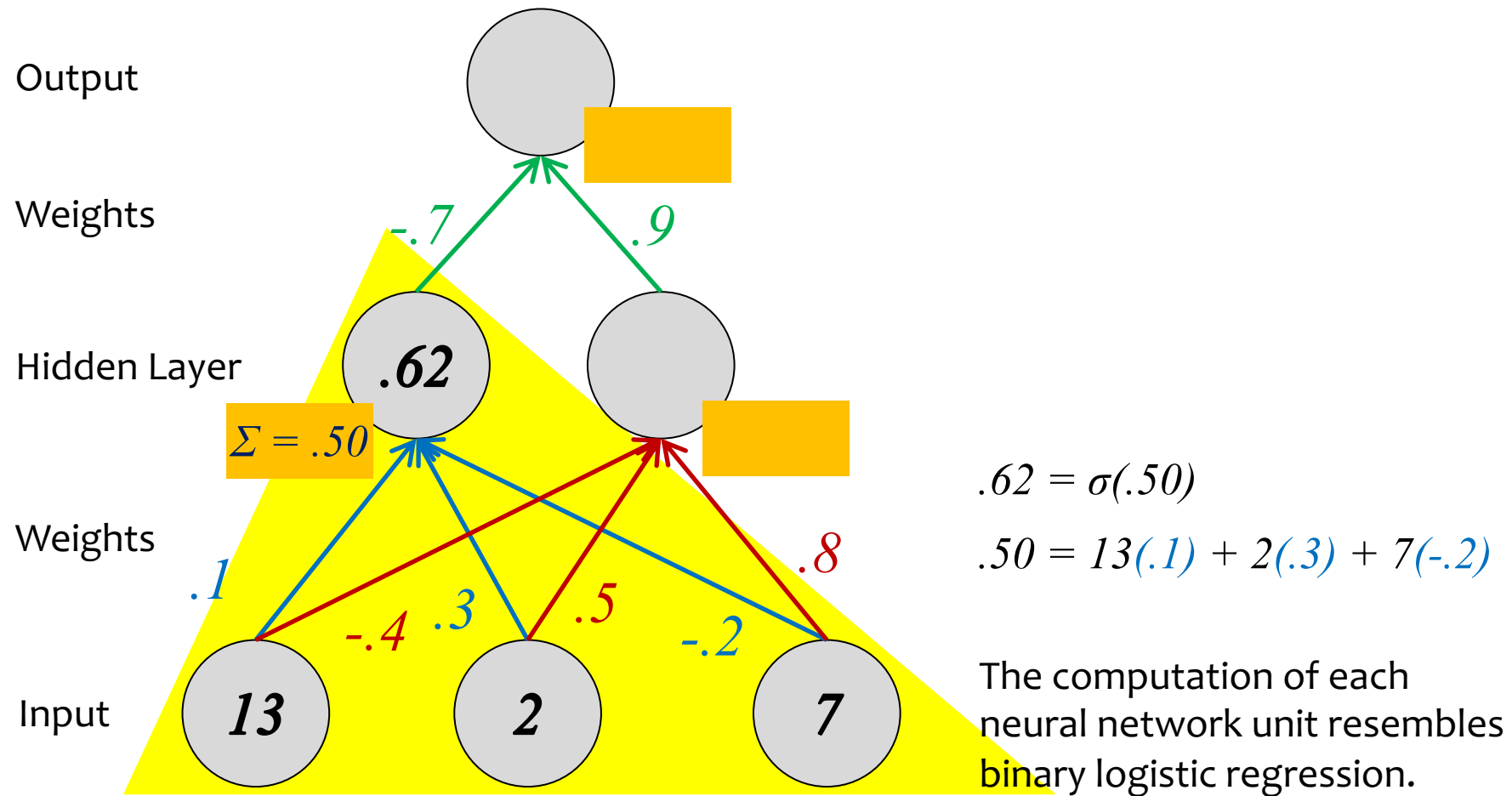


Suppose we already learned the weights of the neural network.

To make a new prediction, we take in some new features (aka. the input layer) and perform the feed-forward computation.

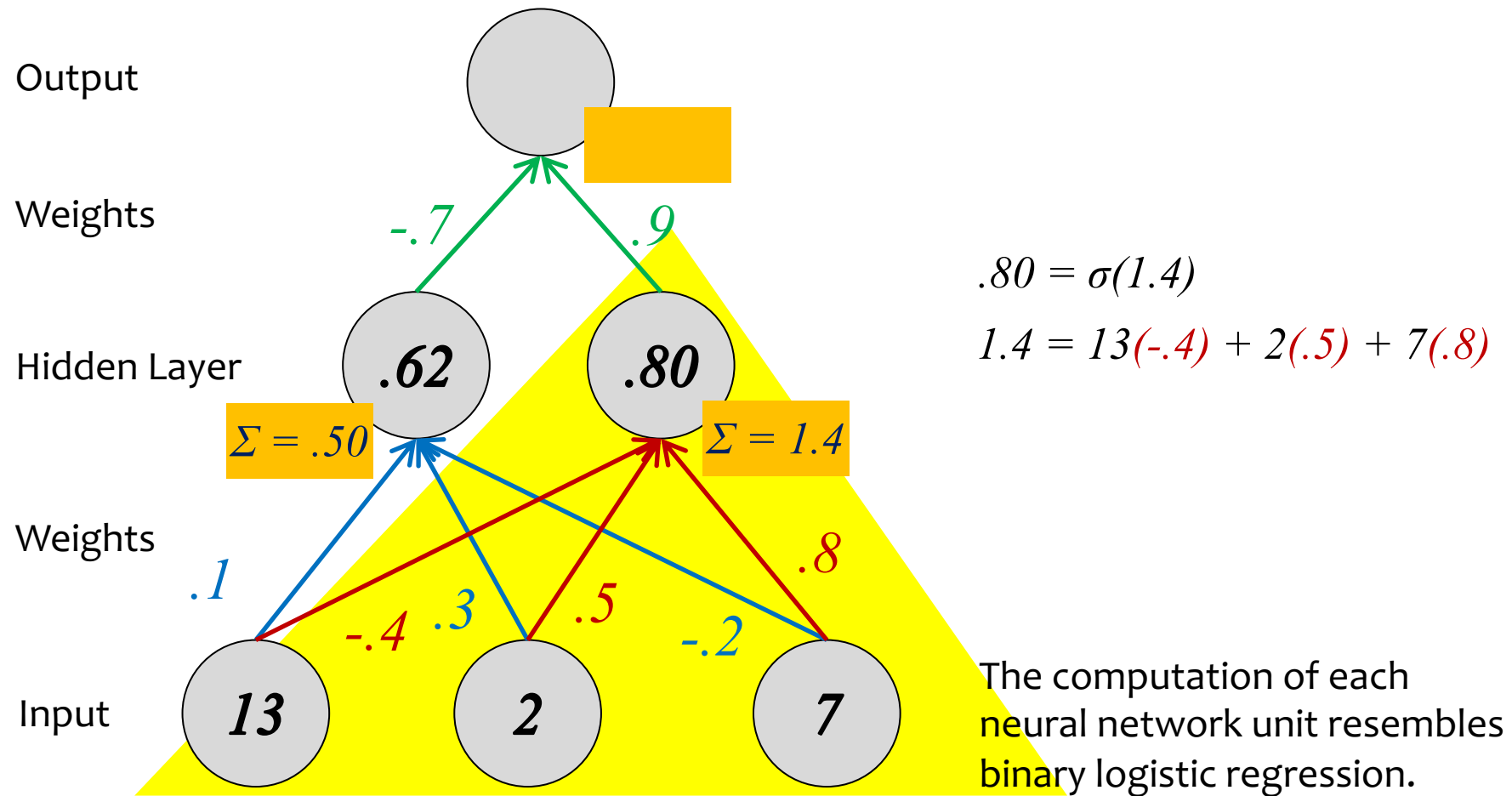
# Decision Functions

# Neural Network



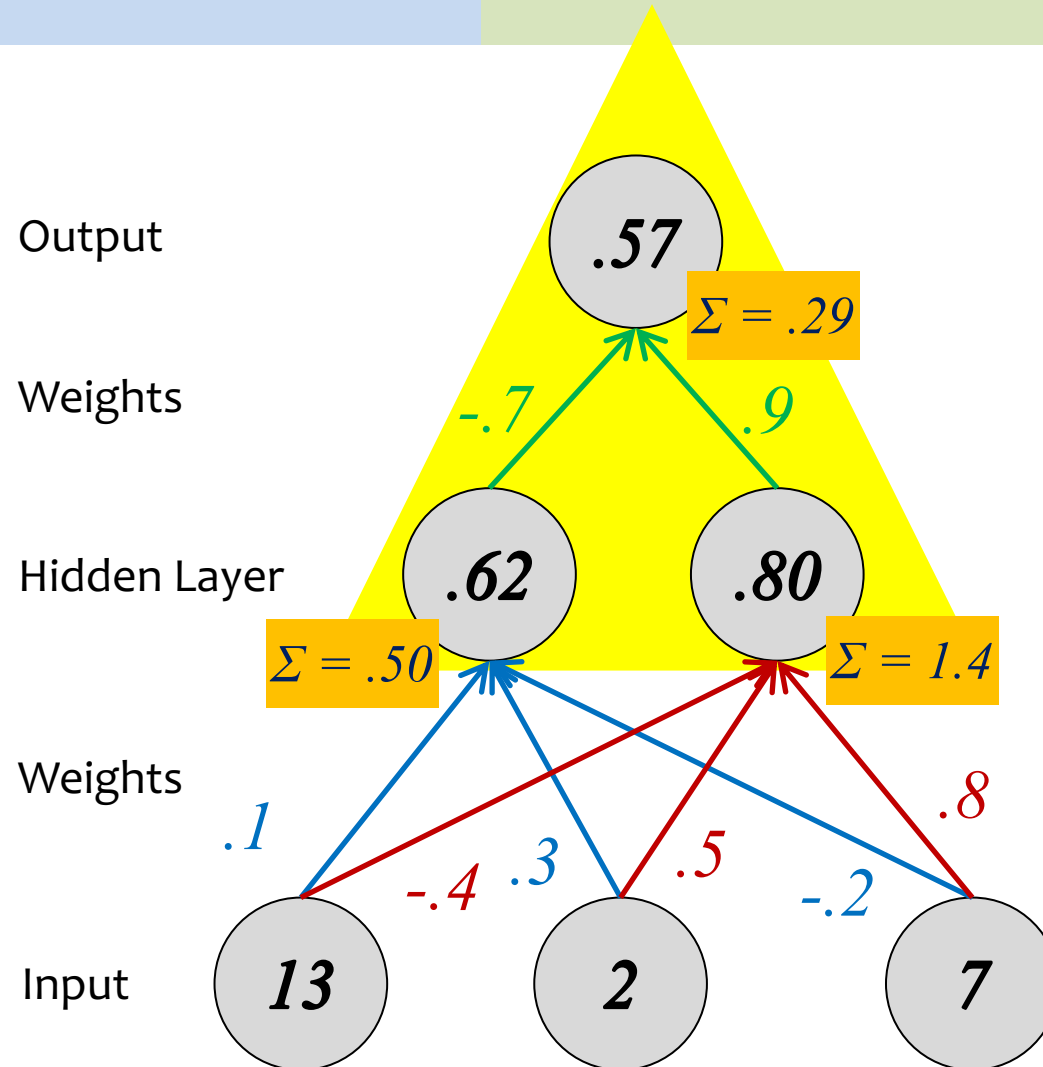
# Decision Functions

# Neural Network



## Decision Functions

## Neural Network



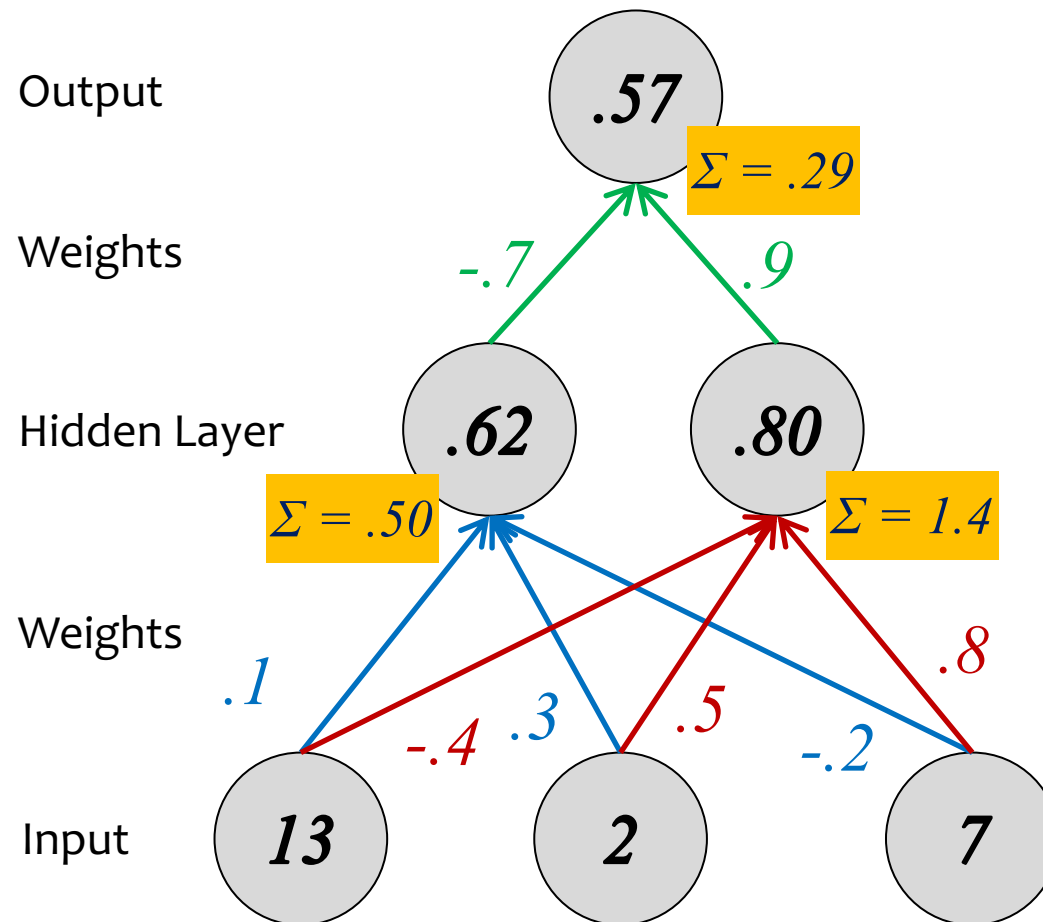
$$.57 = \sigma(.29)$$

$$.29 = .62(-.7) + .80(.9)$$

The computation of each neural network unit resembles binary logistic regression.

## Decision Functions

# Neural Network



$$.57 = \sigma(.29)$$

$$.29 = .62(-.7) + .80(.9)$$

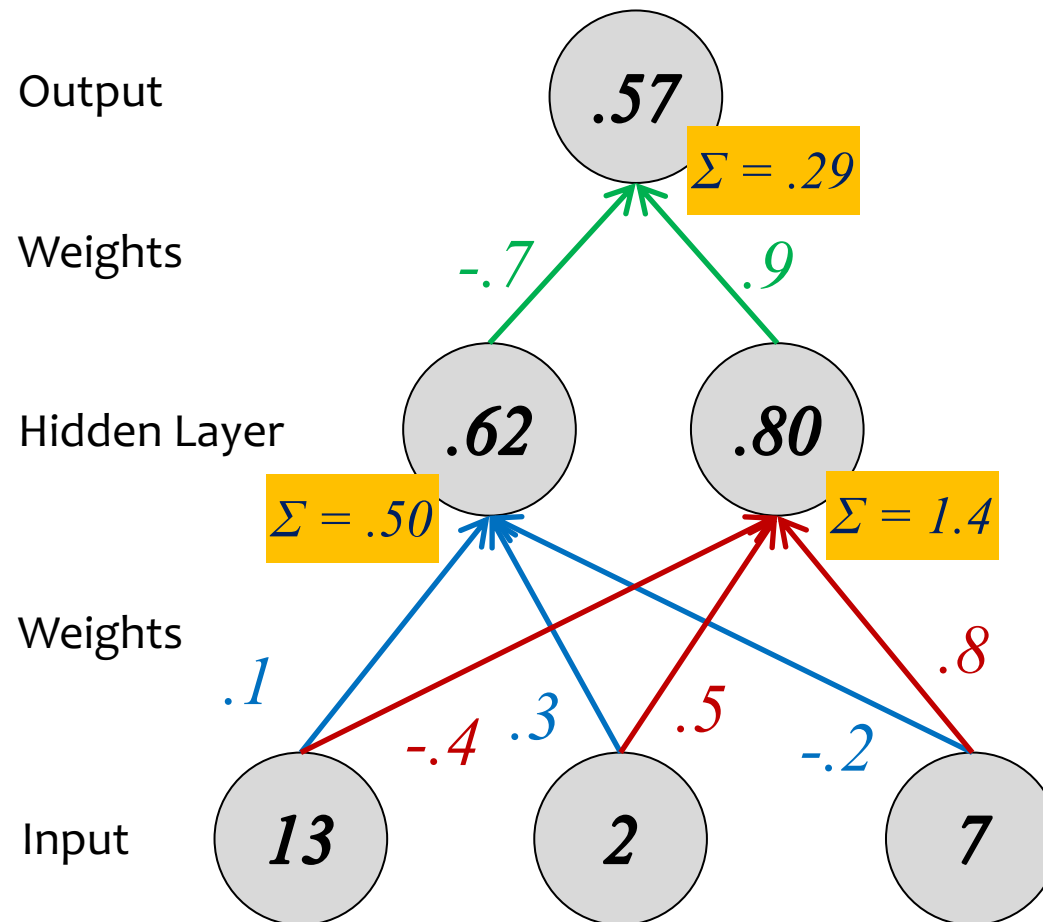
$$.80 = \sigma(1.4)$$

$$1.4 = 13(-.4) + 2(.5) + 7(.8)$$

$$.62 = \sigma(.50)$$

$$.50 = 13(.1) + 2(.3) + 7(-.2)$$

The computation of each neural network unit resembles binary logistic regression.



Except we only have the target value for  $y$  at training time!

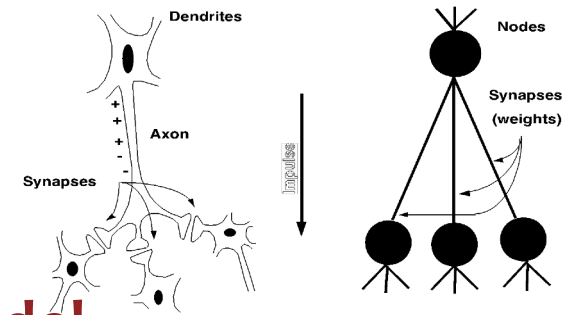
We have to learn to create “useful” values of  $z_1$  and  $z_2$  in the hidden layer.



The computation of each neural network unit resembles binary logistic regression.

# From Biological to Artificial

The motivation for Artificial Neural Networks comes from biology...



## Biological “Model”

- **Neuron:** an excitable cell
- **Synapse:** connection between neurons
- A neuron sends an **electrochemical pulse** along its *synapses* when a sufficient voltage change occurs
- **Biological Neural Network:** collection of neurons along some pathway through the brain

## Biological “Computation”

- Neuron switching time :  $\sim 0.001$  sec
- Number of neurons:  $\sim 10^{10}$
- Connections per neuron:  $\sim 10^{4-5}$
- Scene recognition time:  $\sim 0.1$  sec

## Artificial Model

- **Neuron:** node in a directed acyclic graph (DAG)
- **Weight:** multiplier on each edge
- **Activation Function:** nonlinear thresholding function, which allows a neuron to “fire” when the input value is sufficiently high
- **Artificial Neural Network:** collection of neurons into a DAG, which define some differentiable function

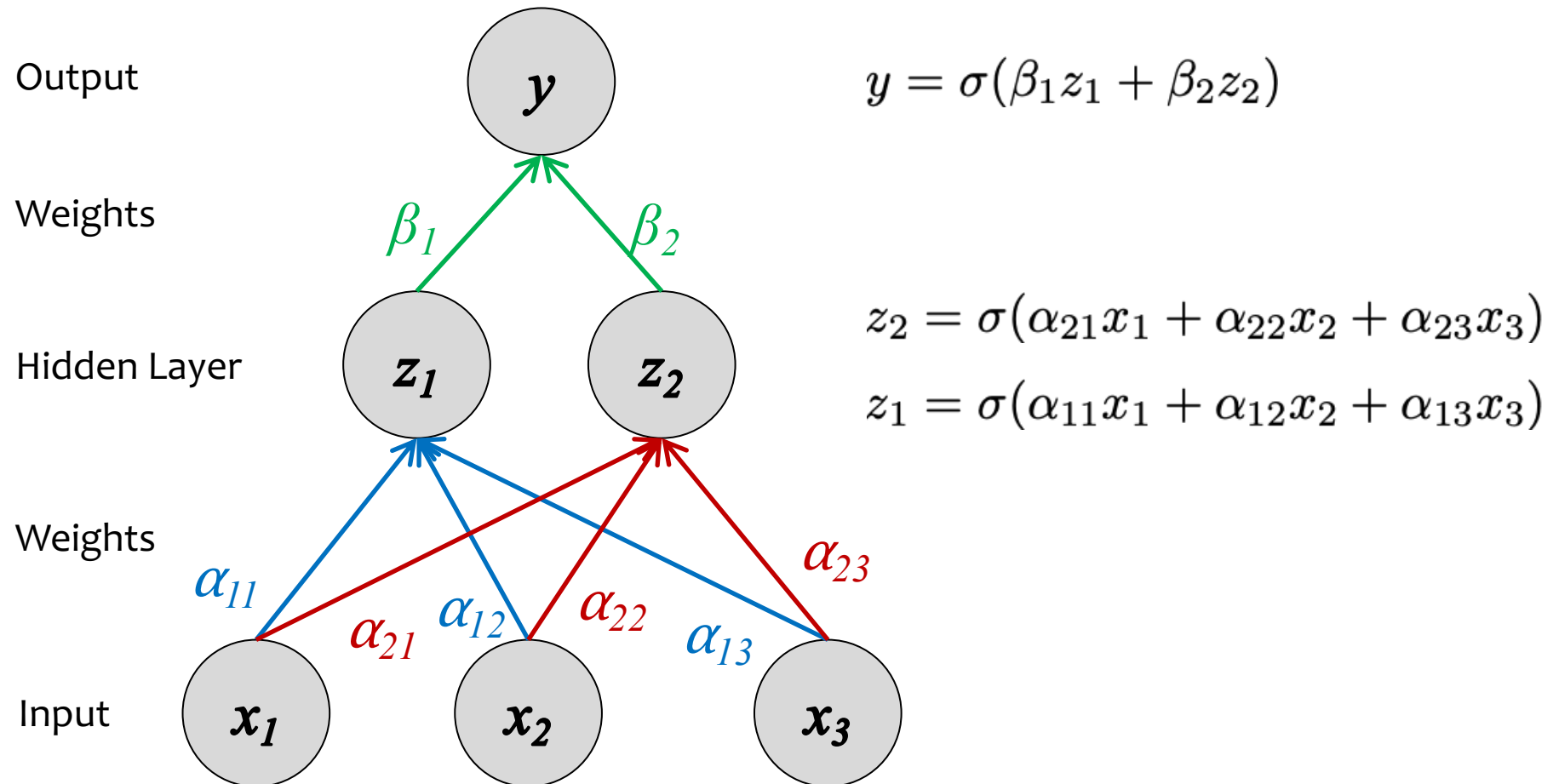
## Artificial Computation

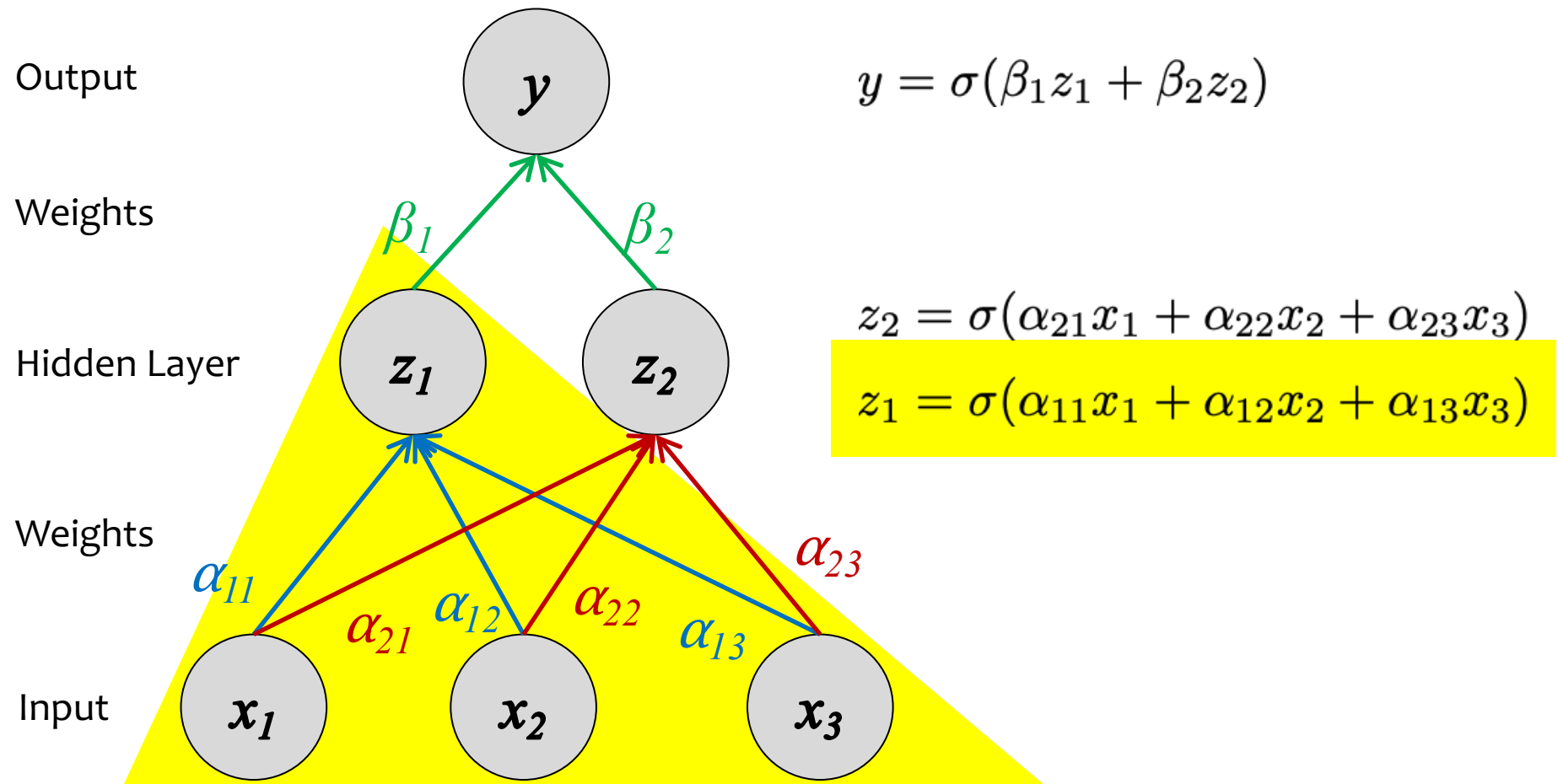
- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed processes

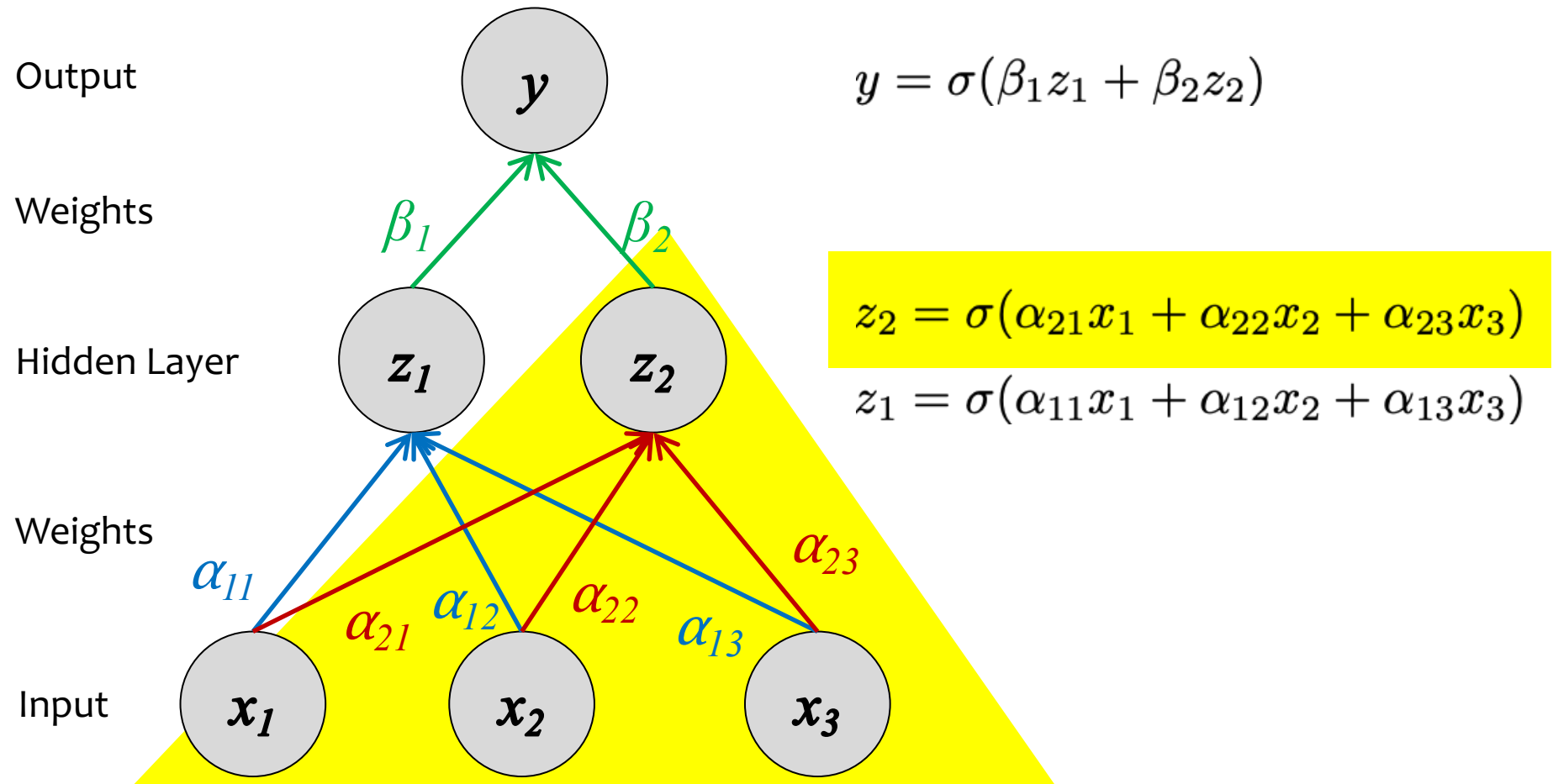
# **DEFINING A 1-HIDDEN LAYER NEURAL NETWORK**

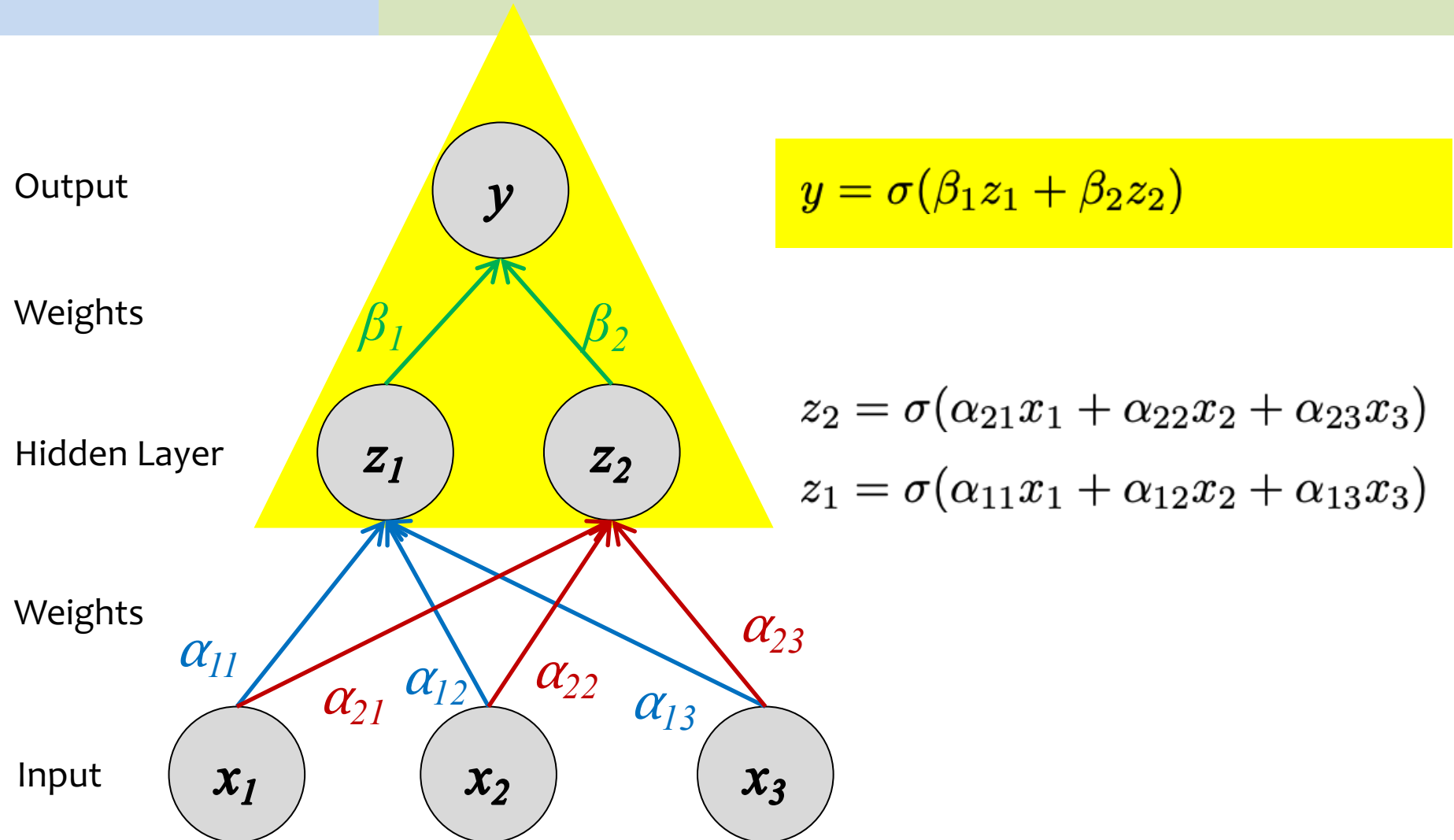


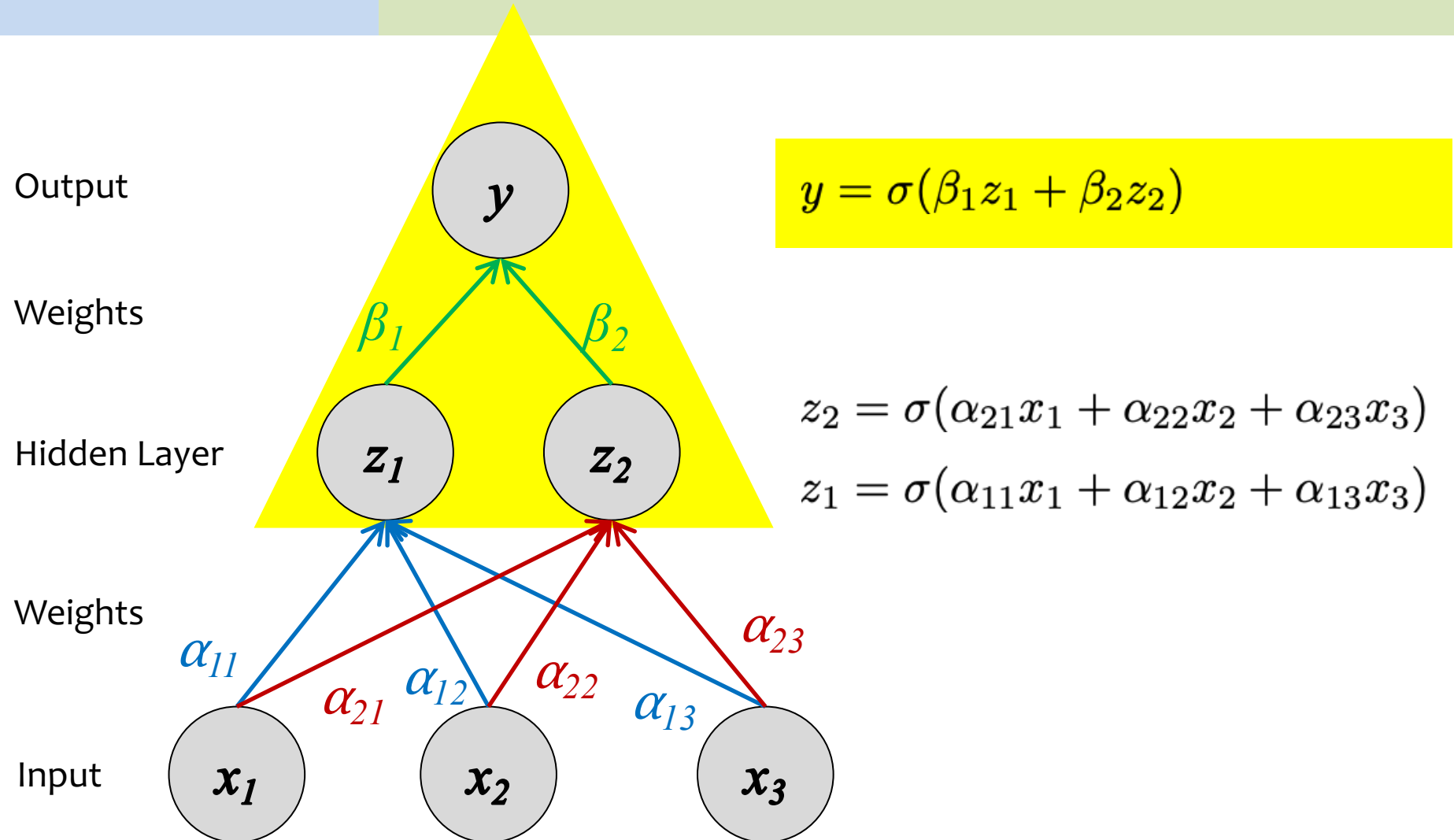
# Example: Neural Network with One Hidden Layer







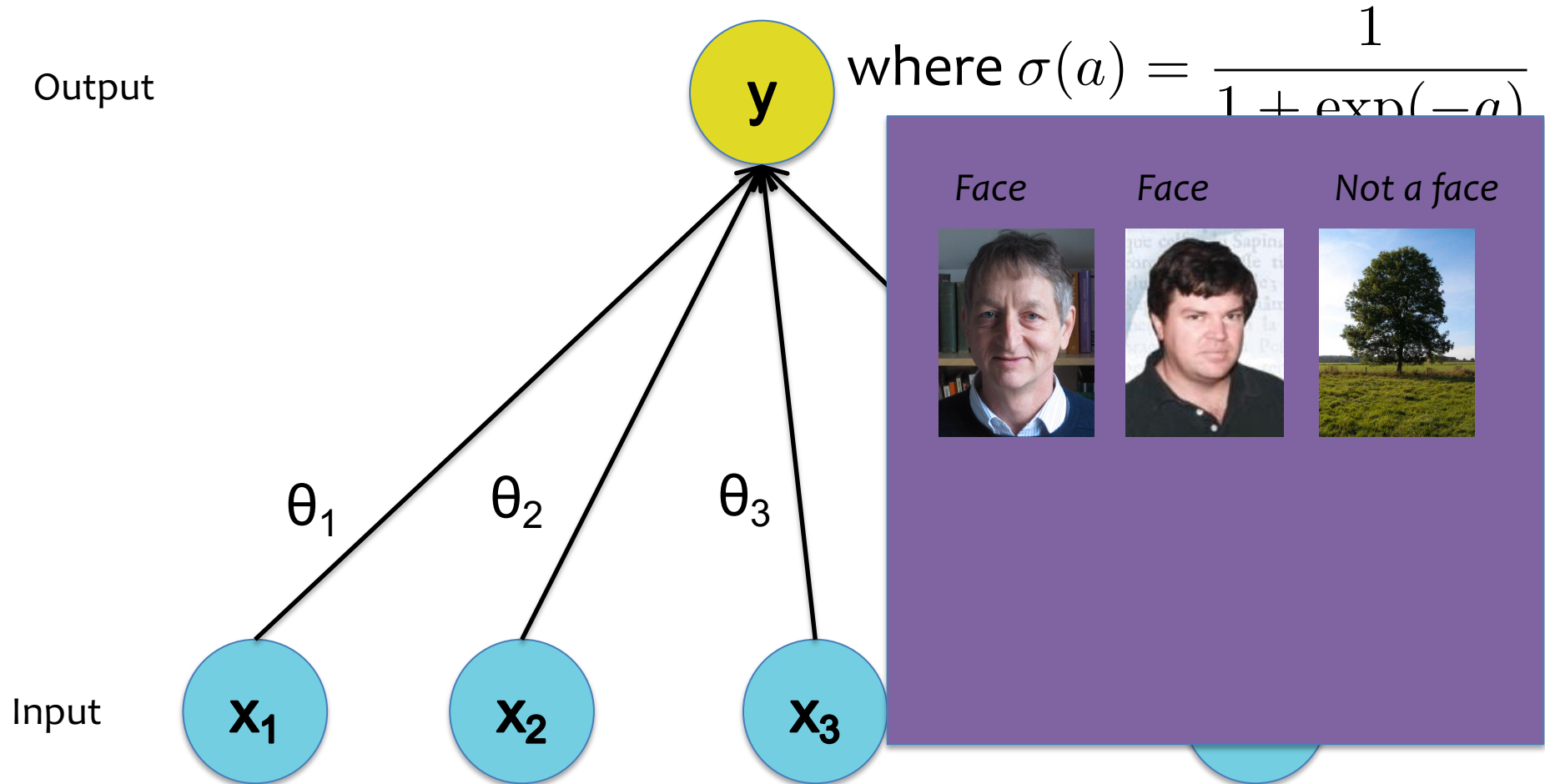




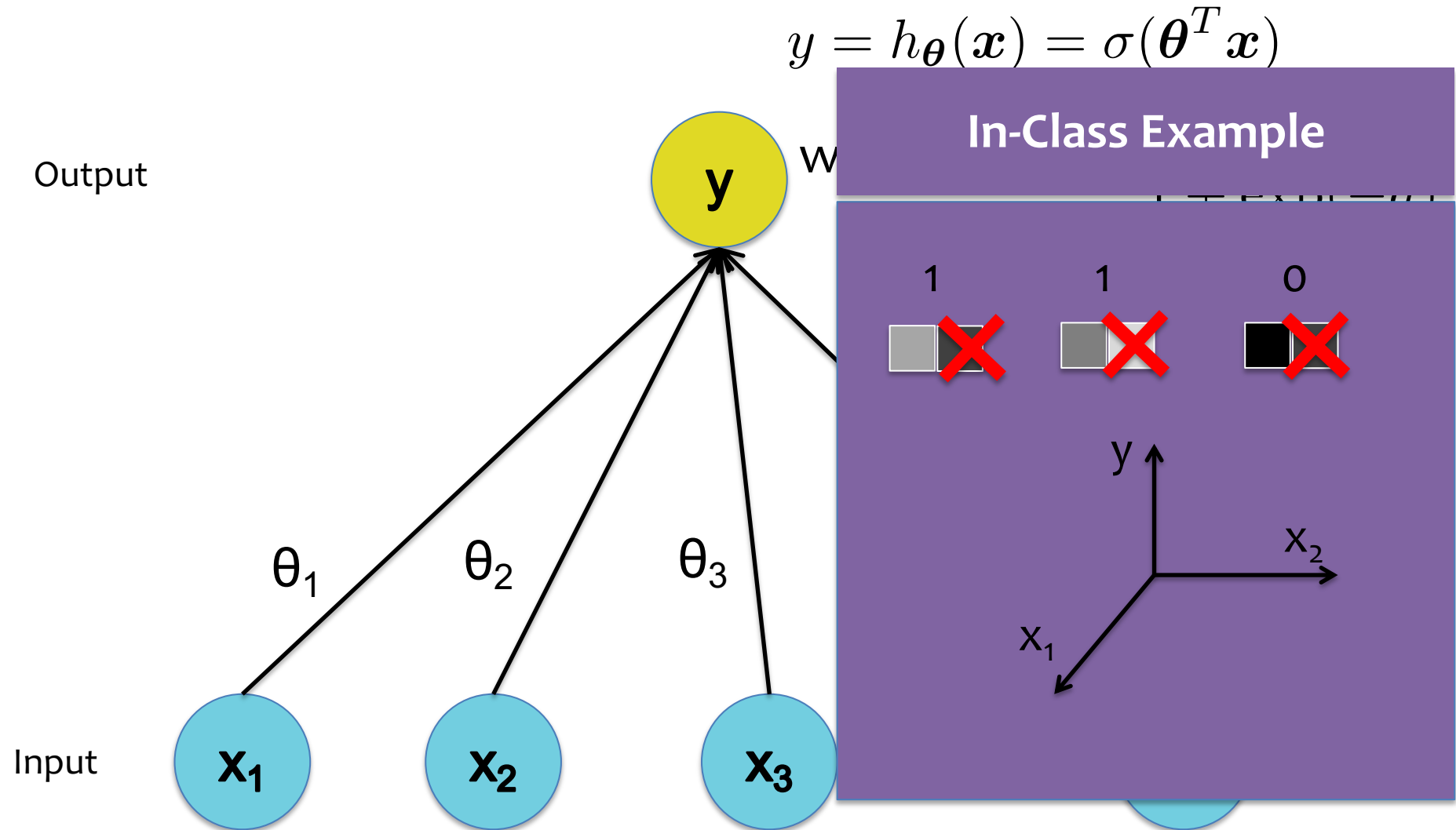
# **NONLINEAR DECISION BOUNDARIES AND NEURAL NETWORKS**

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

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



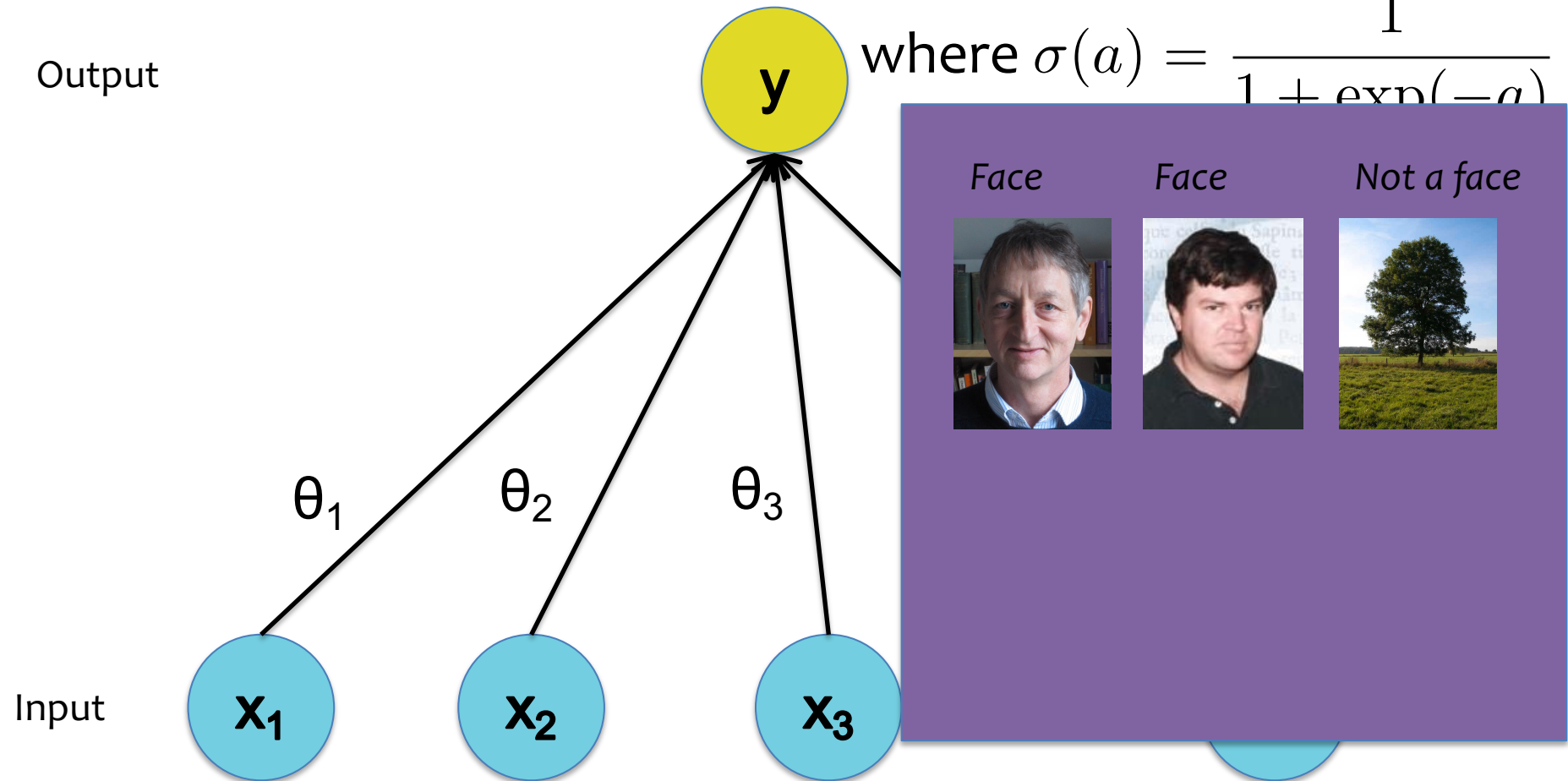


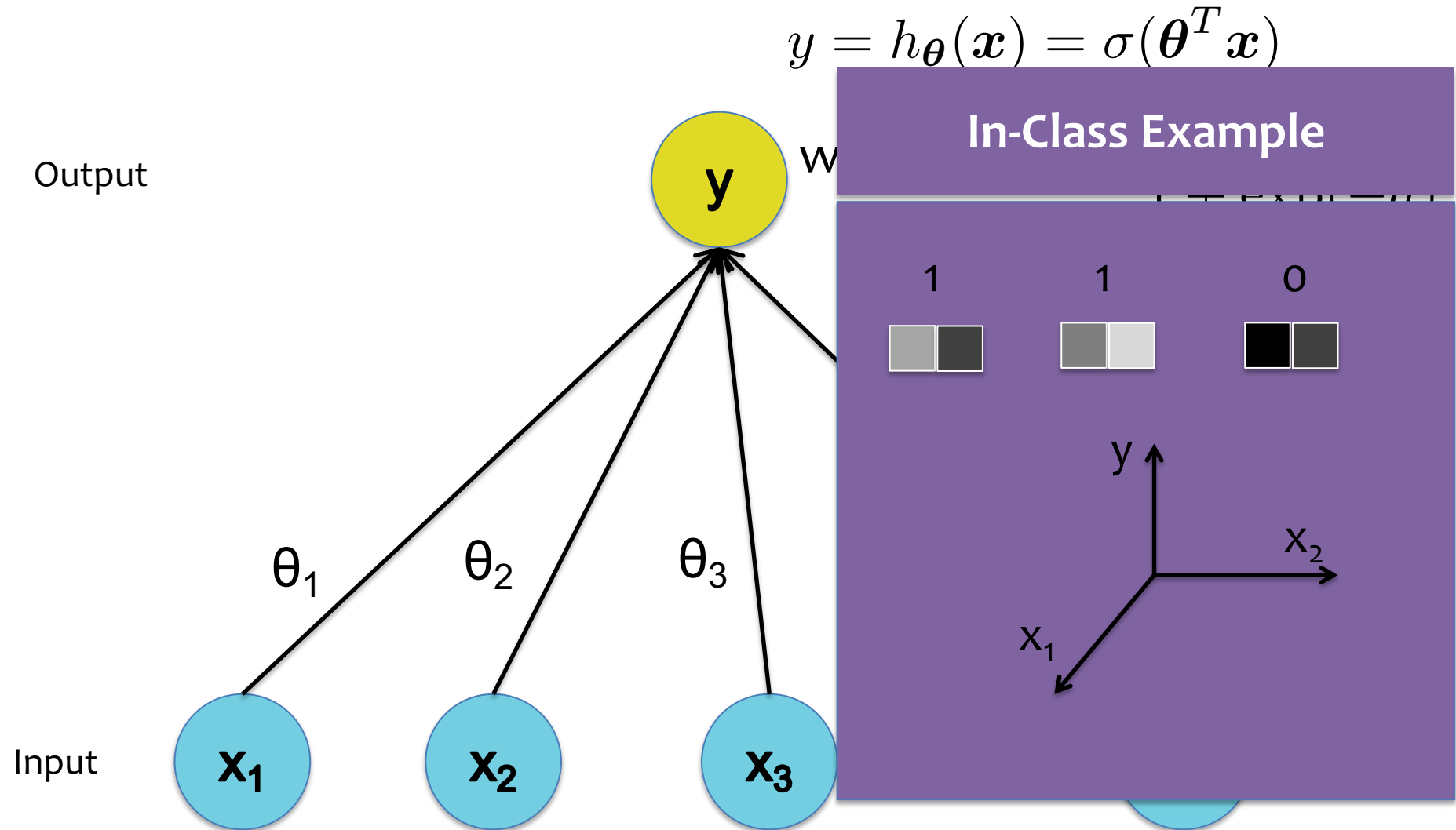


# 1D Face Recognition

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

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





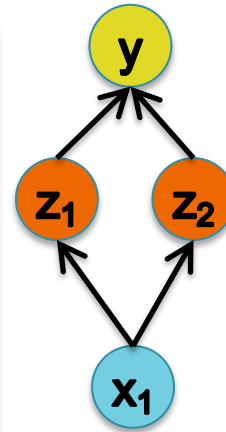
# Neural Network Parameters

## Question:

Suppose you are training a one-hidden layer neural network with sigmoid activations for binary classification.



**True or False:** There is a unique set of parameters that maximize the likelihood of the dataset above.



## Answer:

# Neural Network Architectures

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

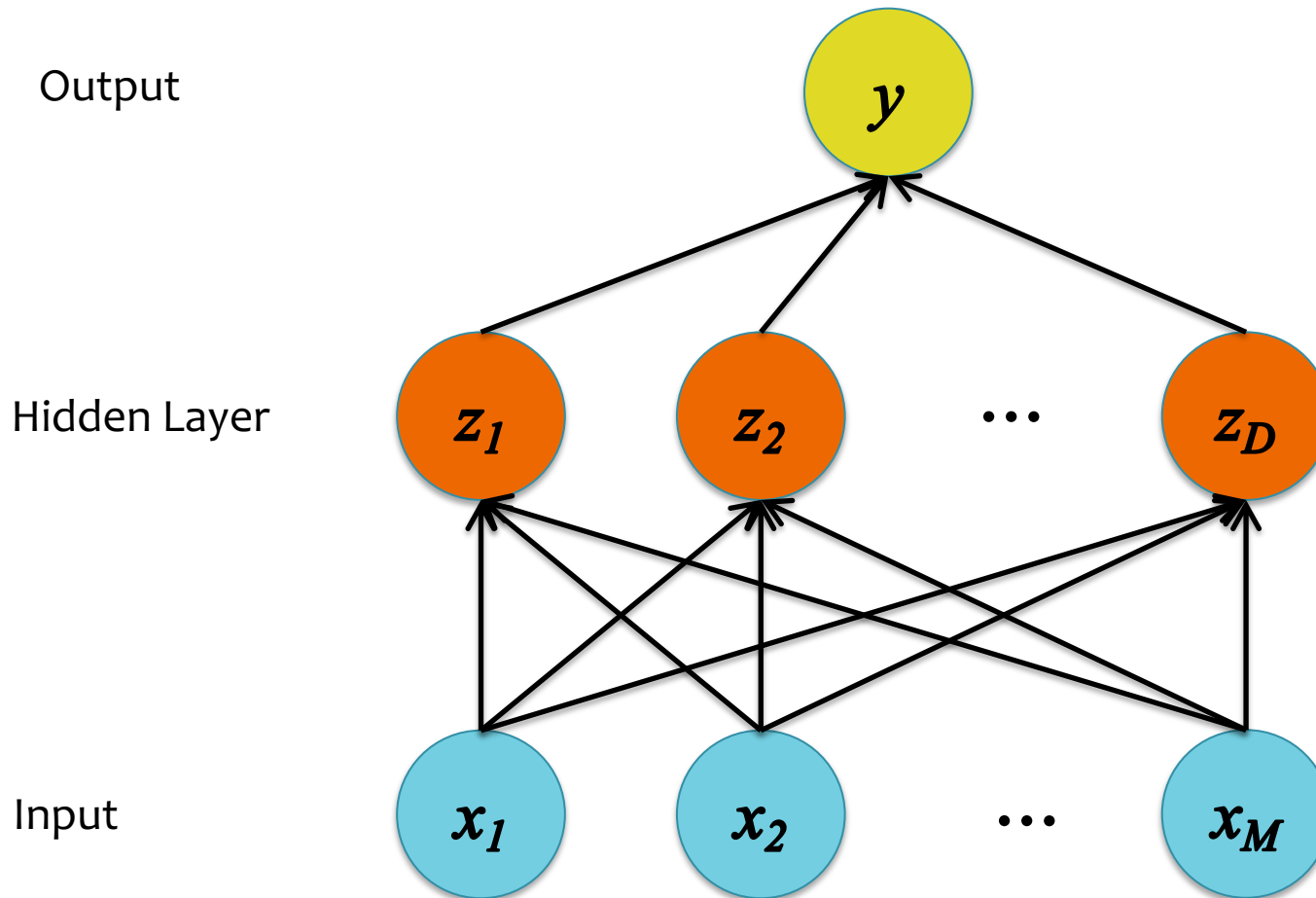
1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters

# **BUILDING WIDER NETWORKS**

$$D = M$$

# Building a Neural Net

*Q: How many hidden units,  $D$ , should we use?*



The hidden units could learn to be...

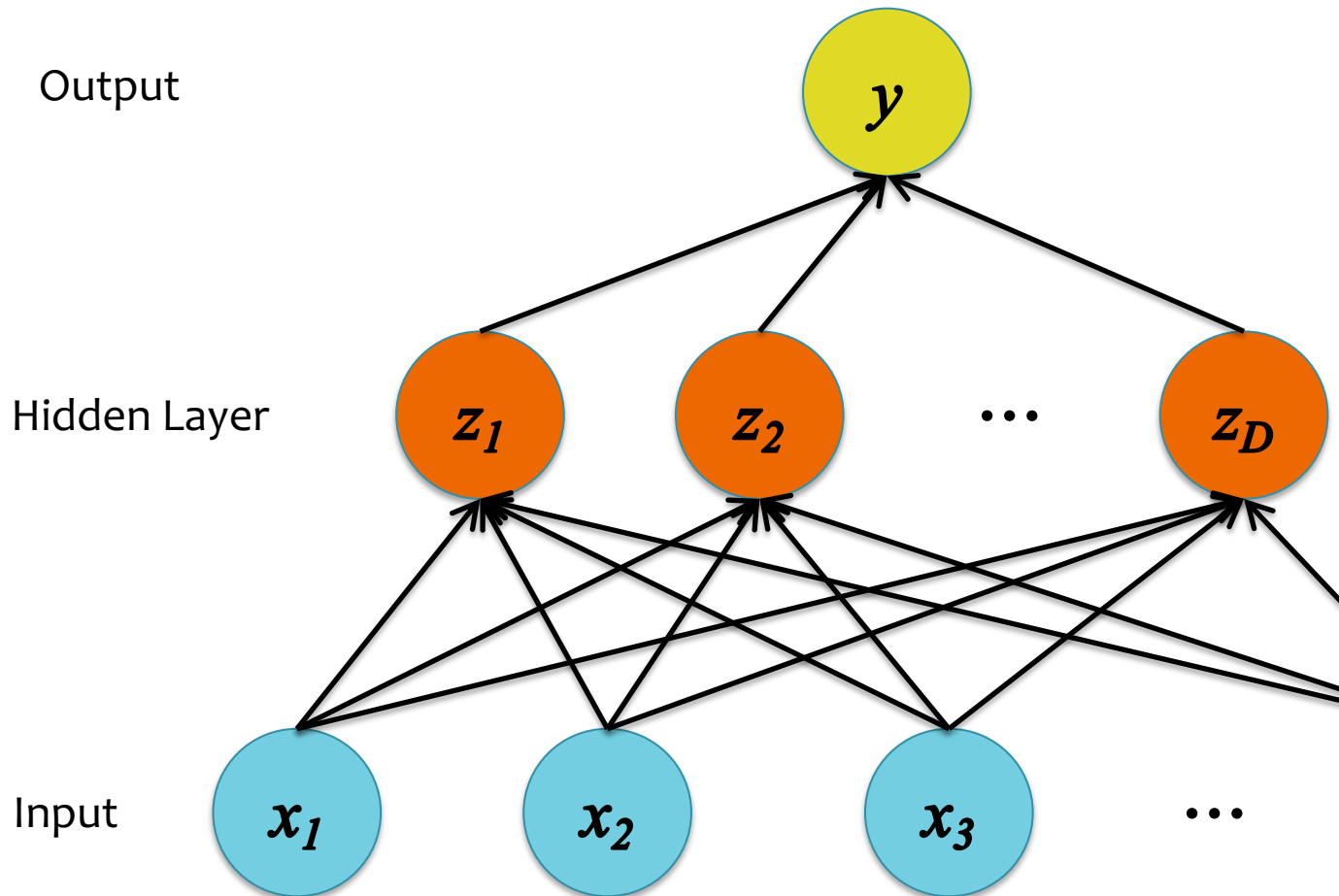
- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above



$$D < M$$

# Building a Neural Net

*Q: How many hidden units,  $D$ , should we use?*



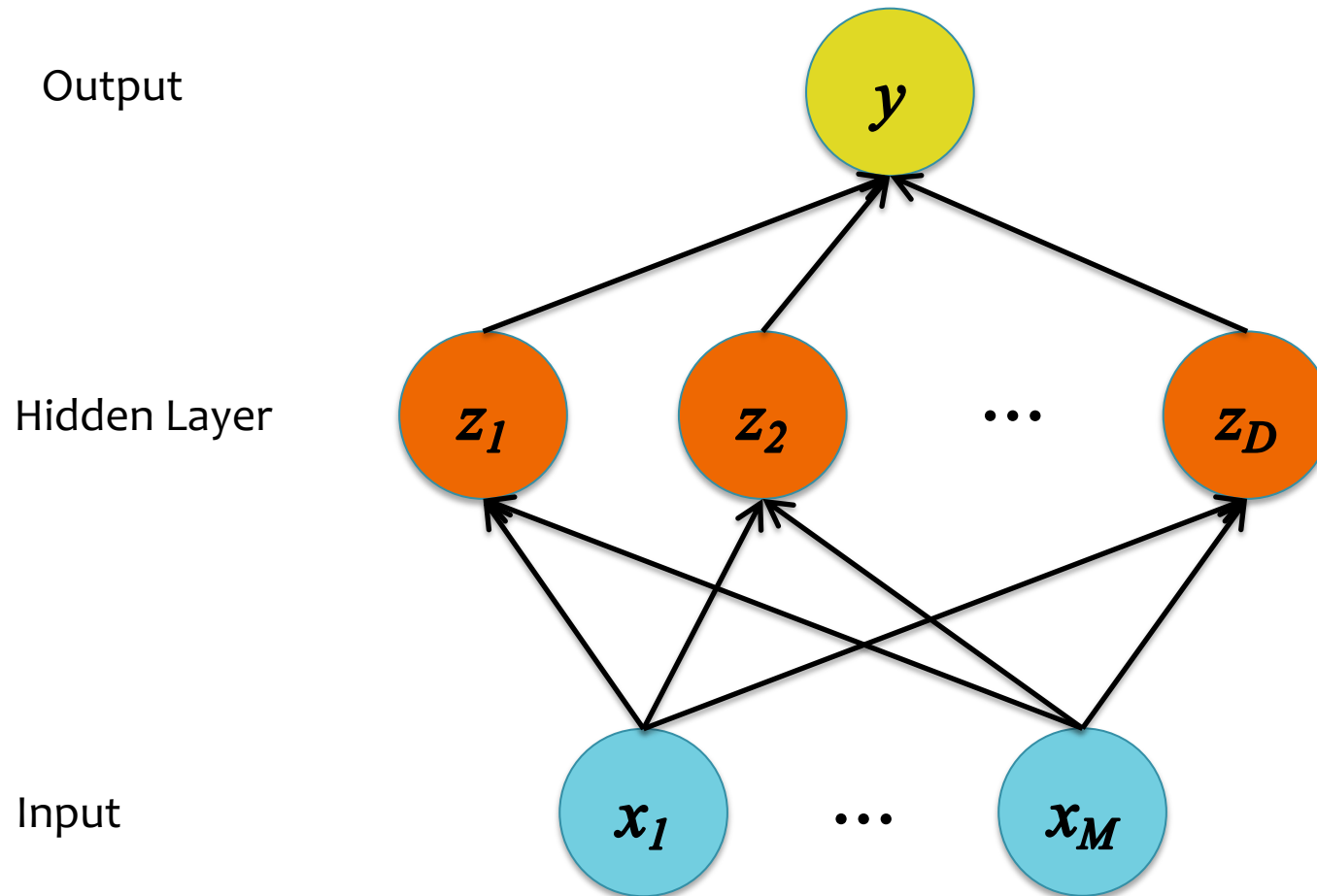
The hidden units could learn to be...

- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above

$D > M$

# Building a Neural Net

*Q: How many hidden units,  $D$ , should we use?*



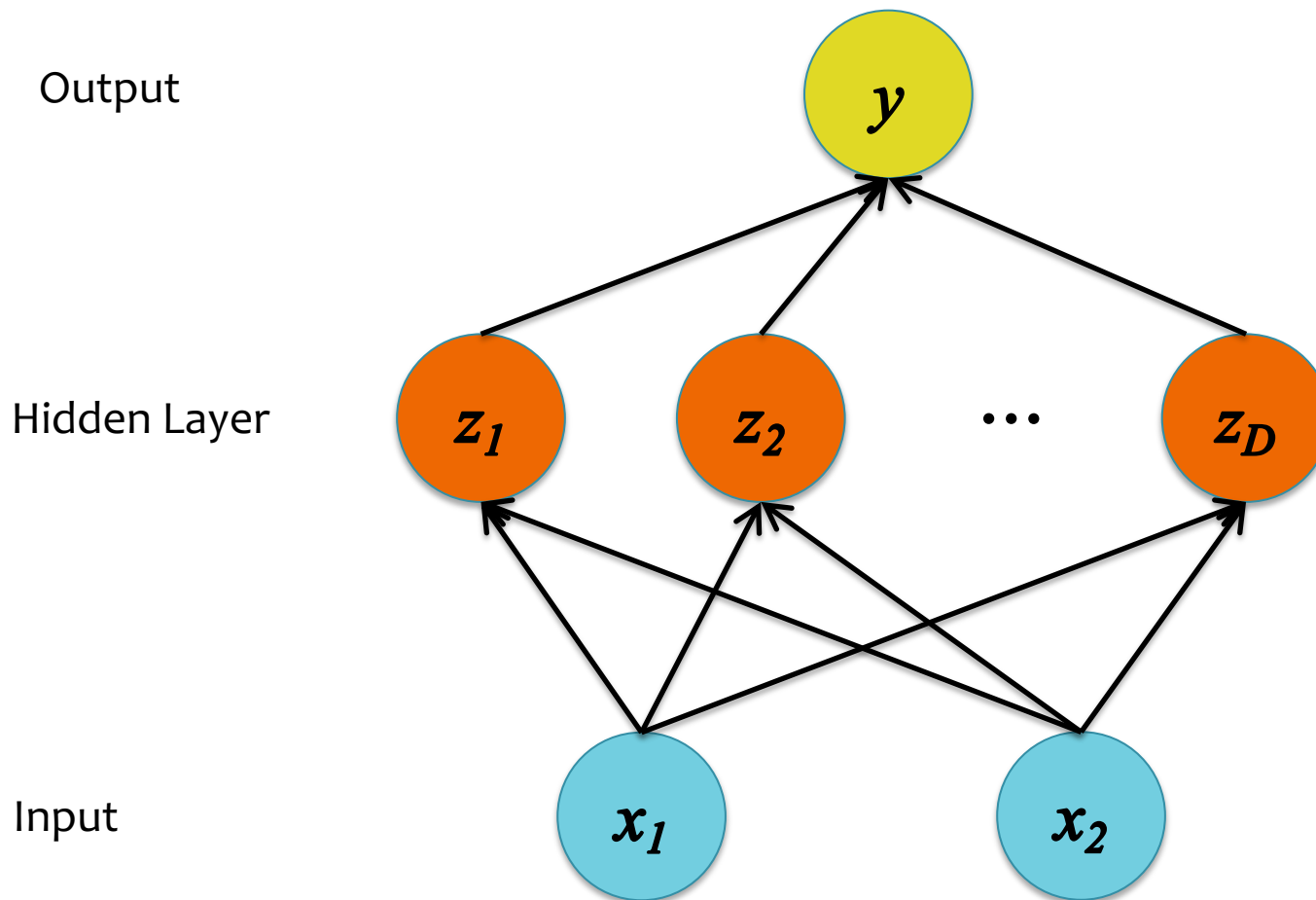
The hidden units could learn to be...

- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above

$$D \geq M$$

# Building a Neural Net

In the following examples, we have two input features,  $M=2$ , and we vary the number of hidden units,  $D$ .



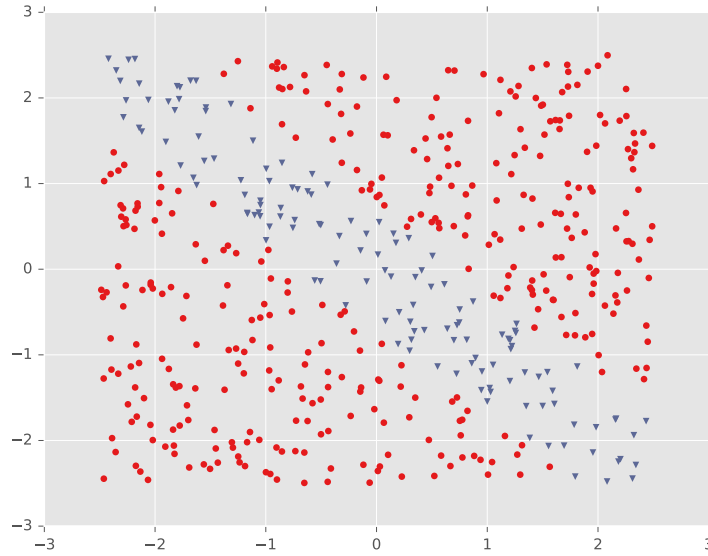
The hidden units could learn to be...

- a selection of the most useful features
- nonlinear combinations of the features
- a lower dimensional projection of the features
- a higher dimensional projection of the features
- a copy of the input features
- a mix of the above

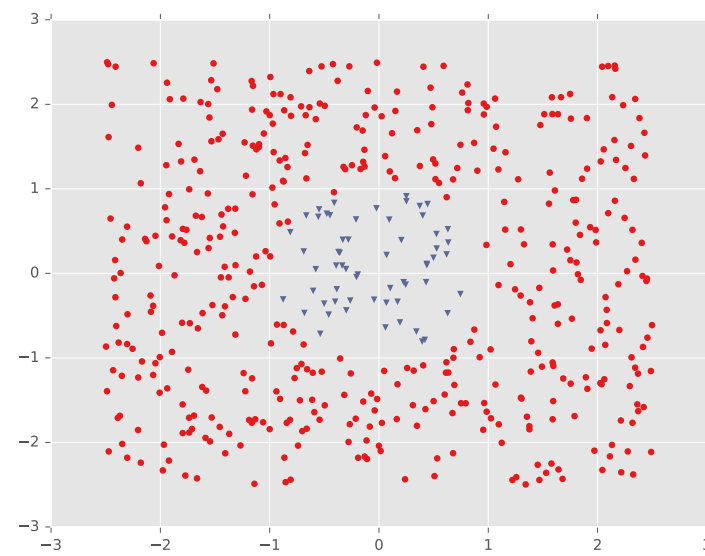
Examples 1 and 2

# **DECISION BOUNDARY EXAMPLES**

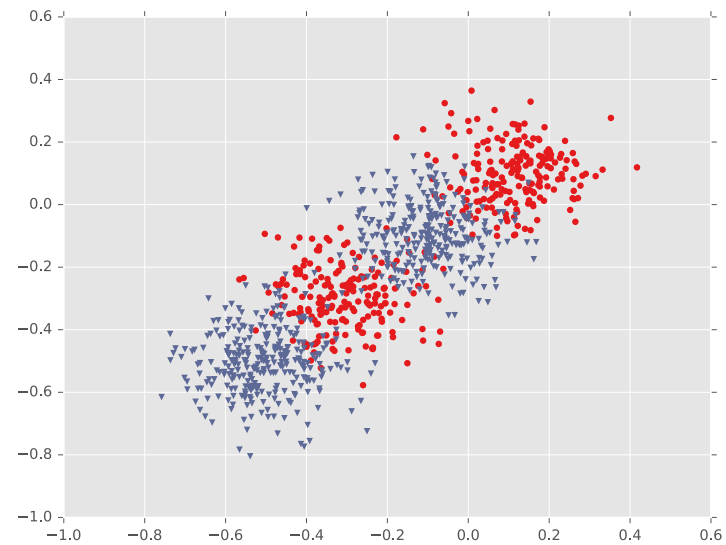
Example #1: Diagonal Band



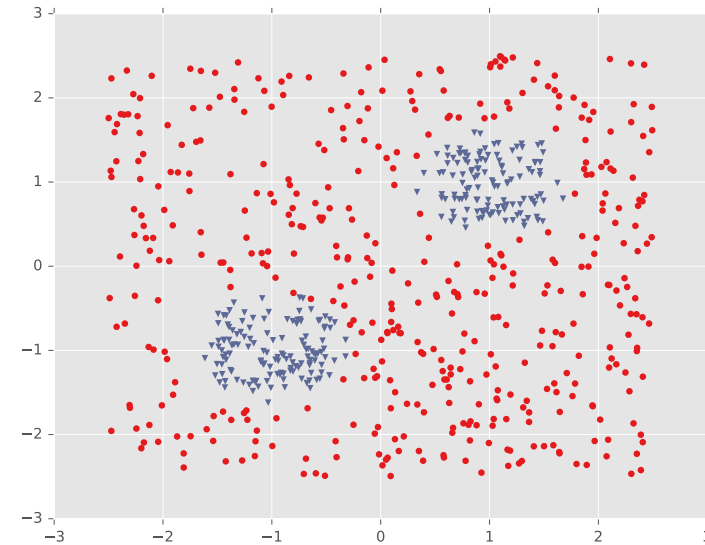
Example #2: One Pocket



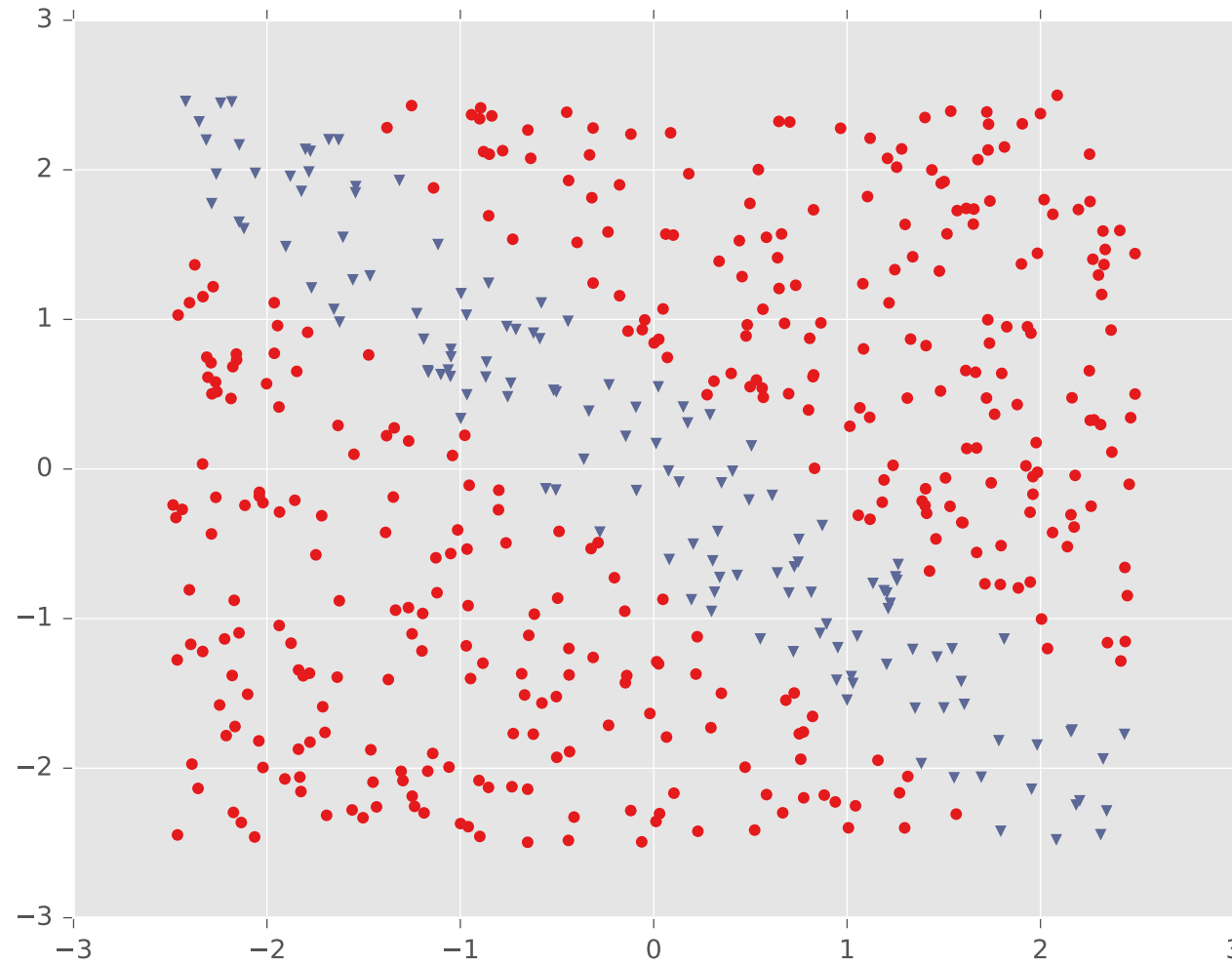
Example #3: Four Gaussians



Example #4: Two Pockets



# Example #1: Diagonal Band

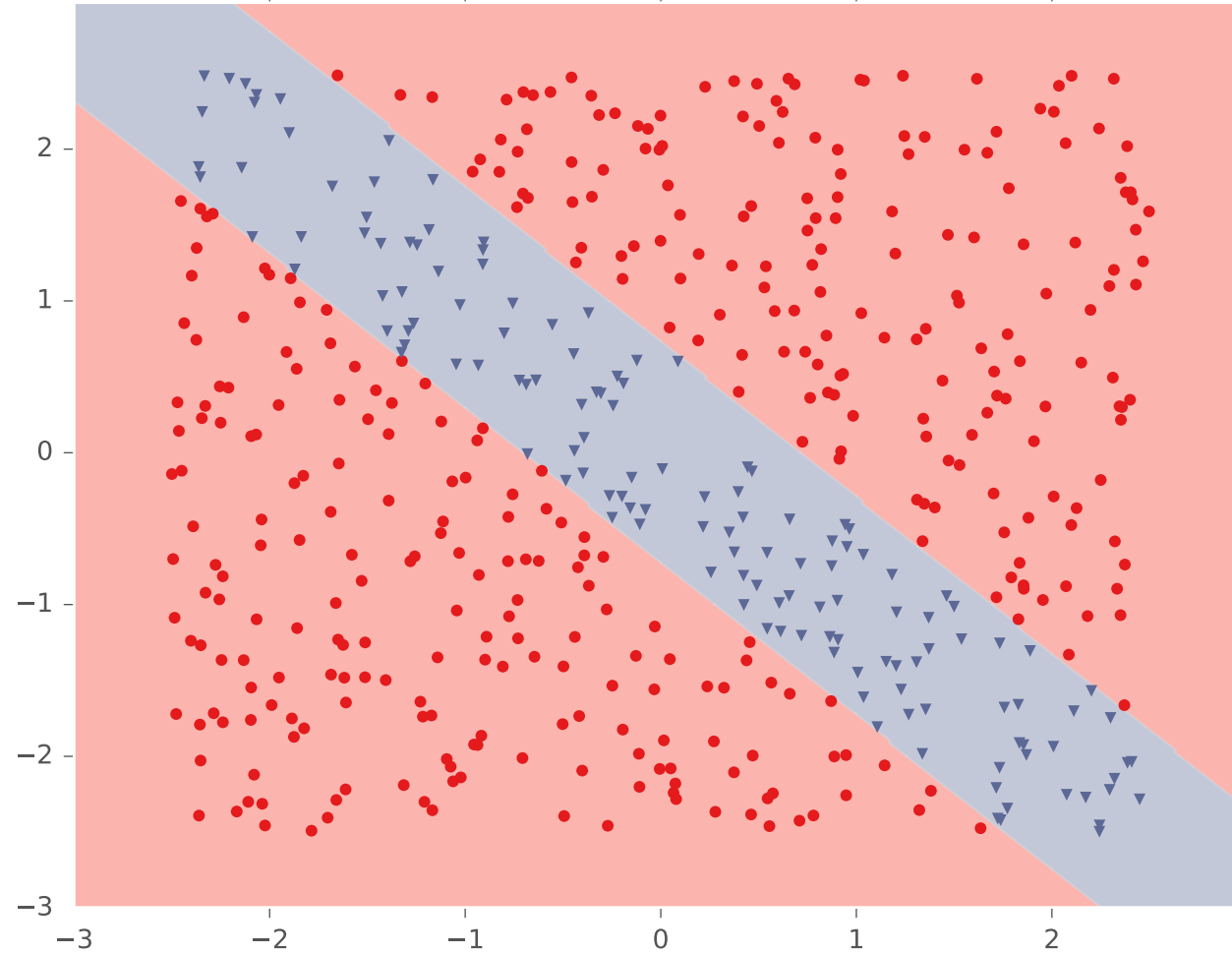


# Example #1: Diagonal Band



# Example #1: Diagonal Band

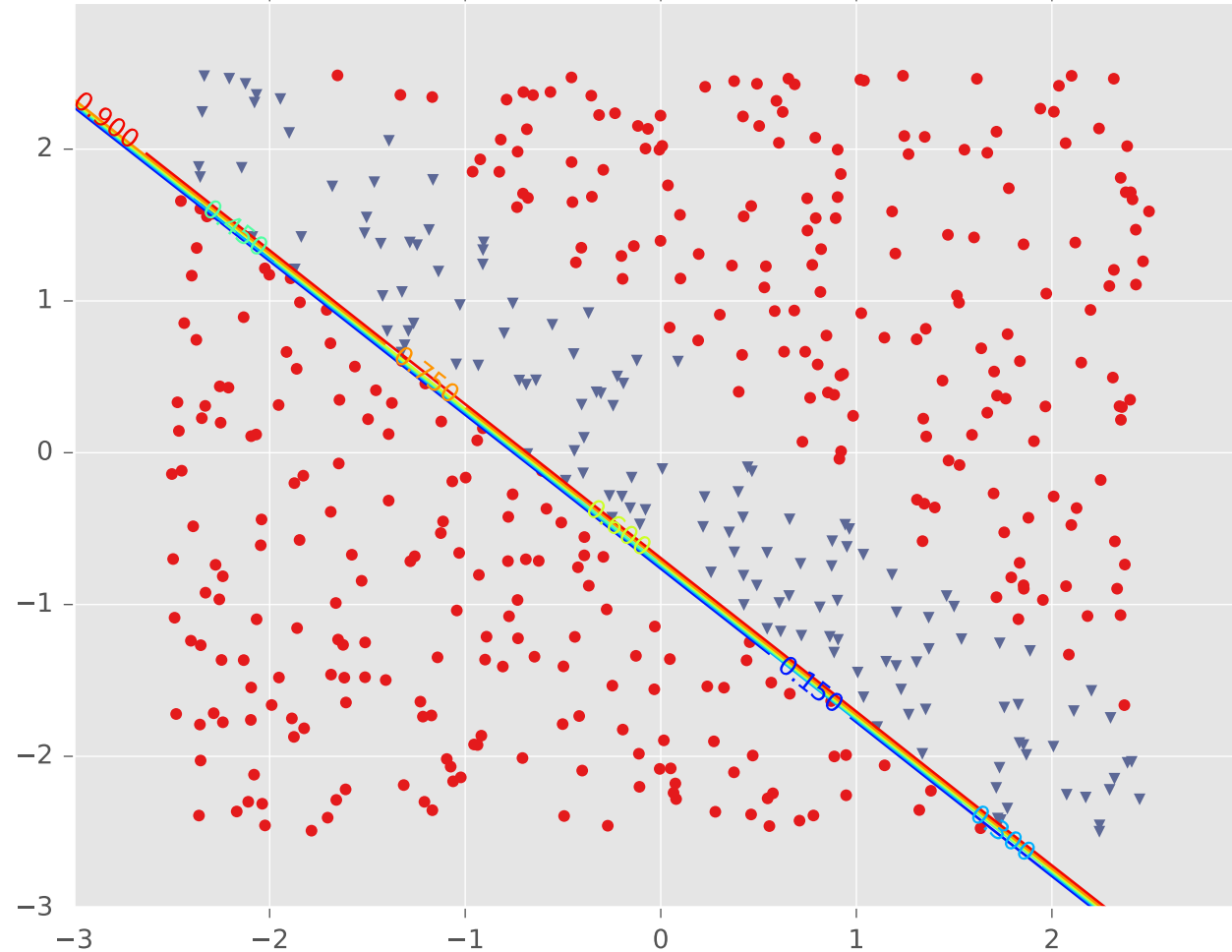
Tuned Neural Network (hidden=2, activation=logistic)





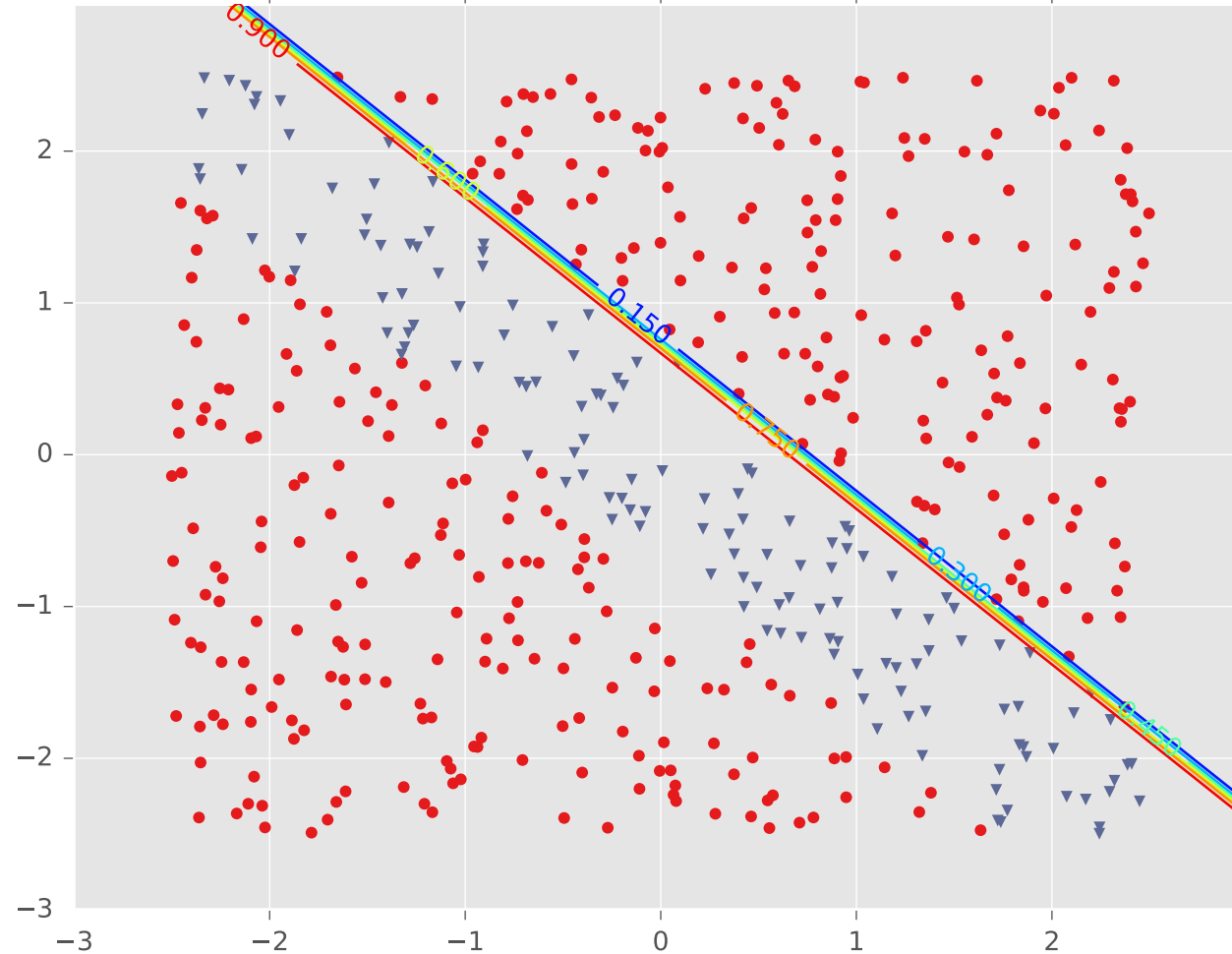
# Example #1: Diagonal Band

LR1 for Tuned Neural Network (hidden=2, activation=logistic)

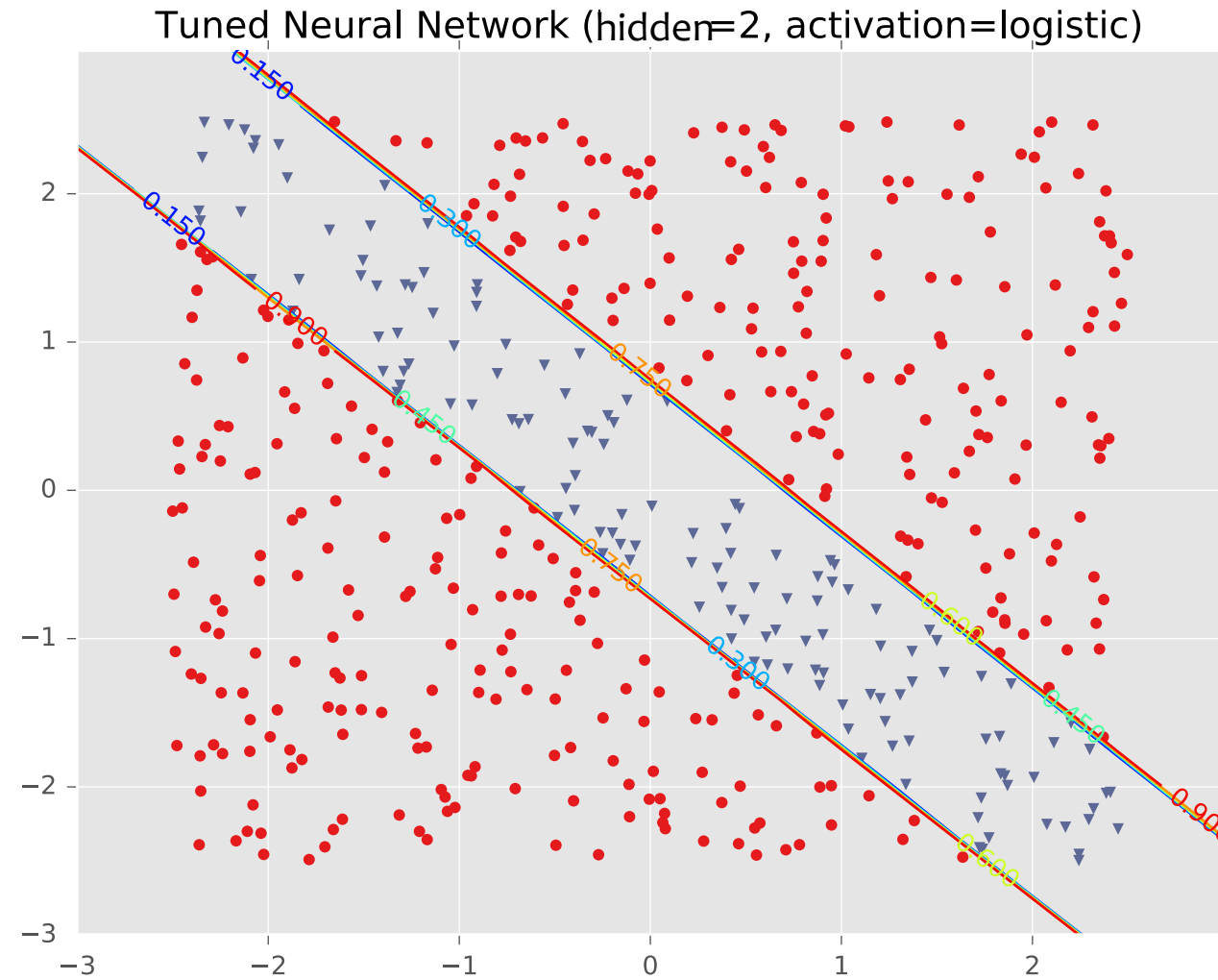


# Example #1: Diagonal Band

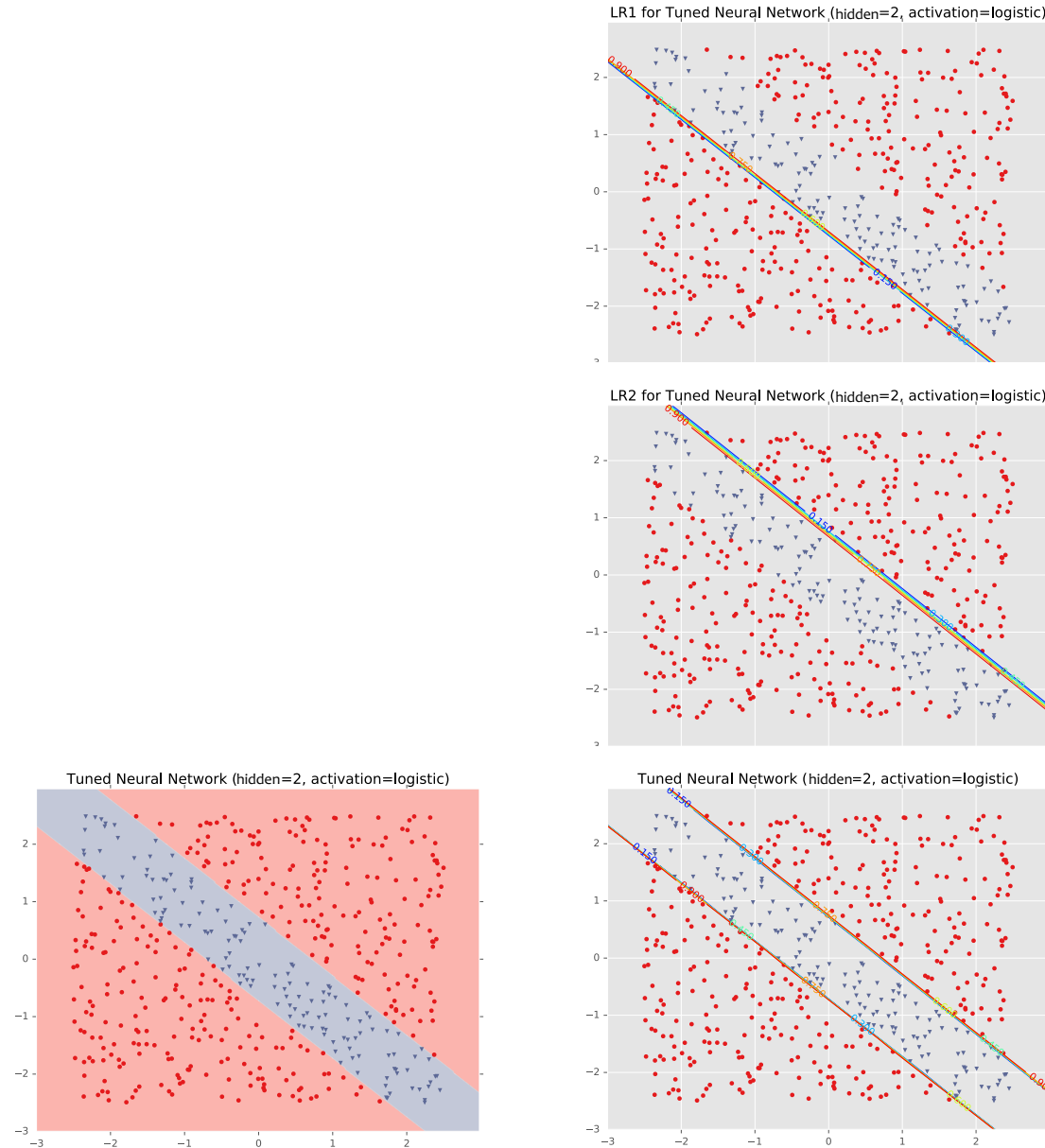
LR2 for Tuned Neural Network (hidden=2, activation=logistic)



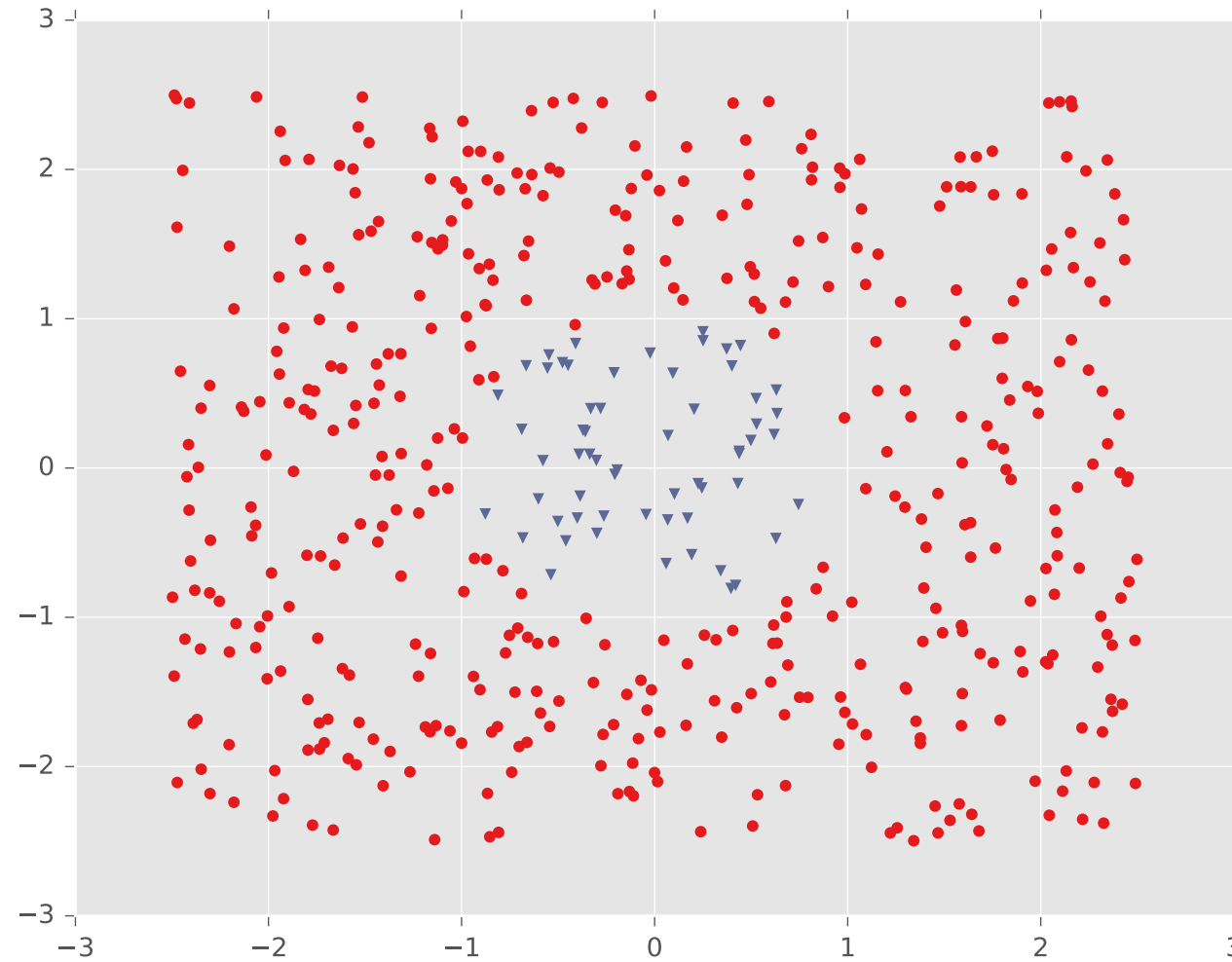
# Example #1: Diagonal Band



# Example #1: Diagonal Band



# Example #2: One Pocket

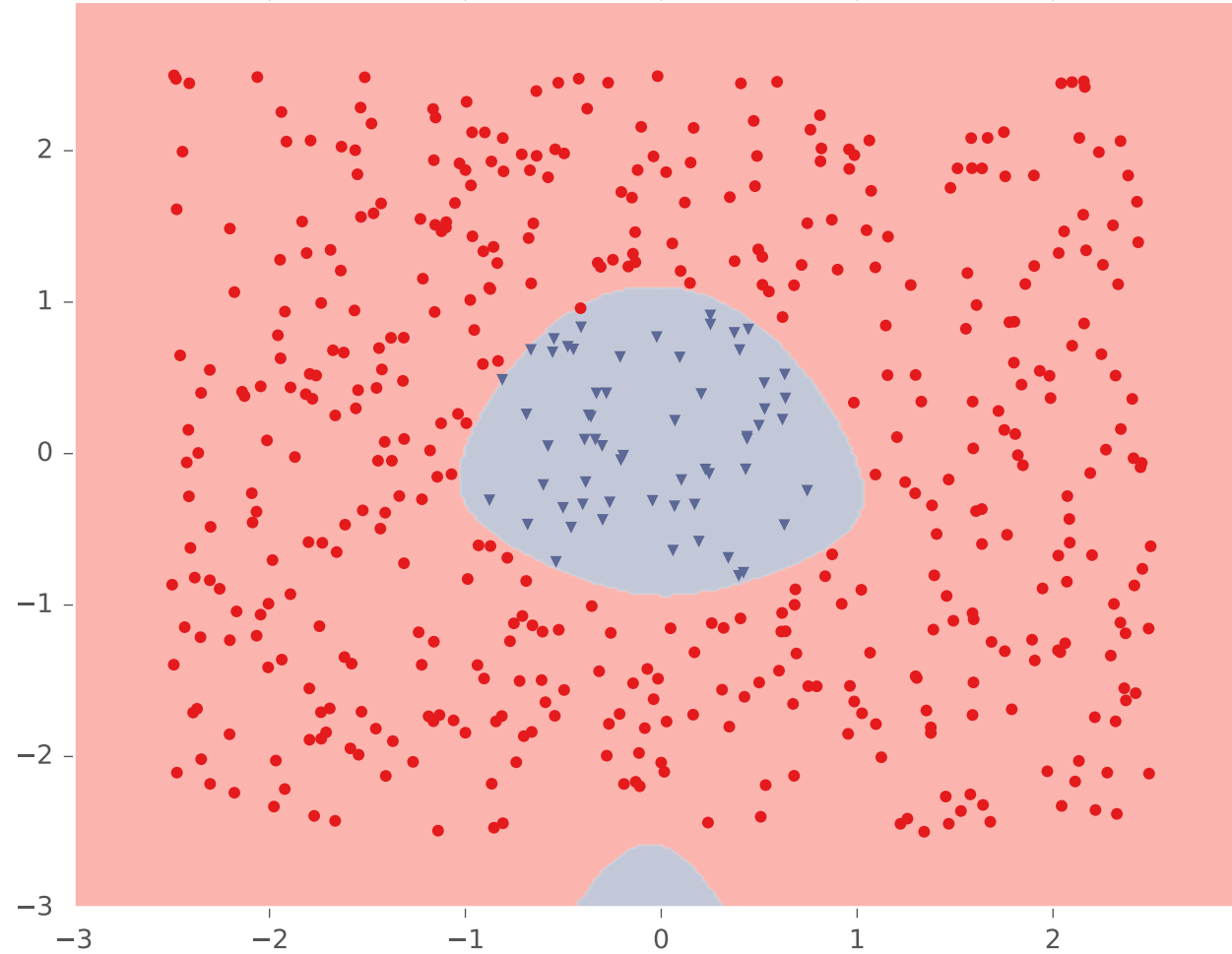


# Example #2: One Pocket



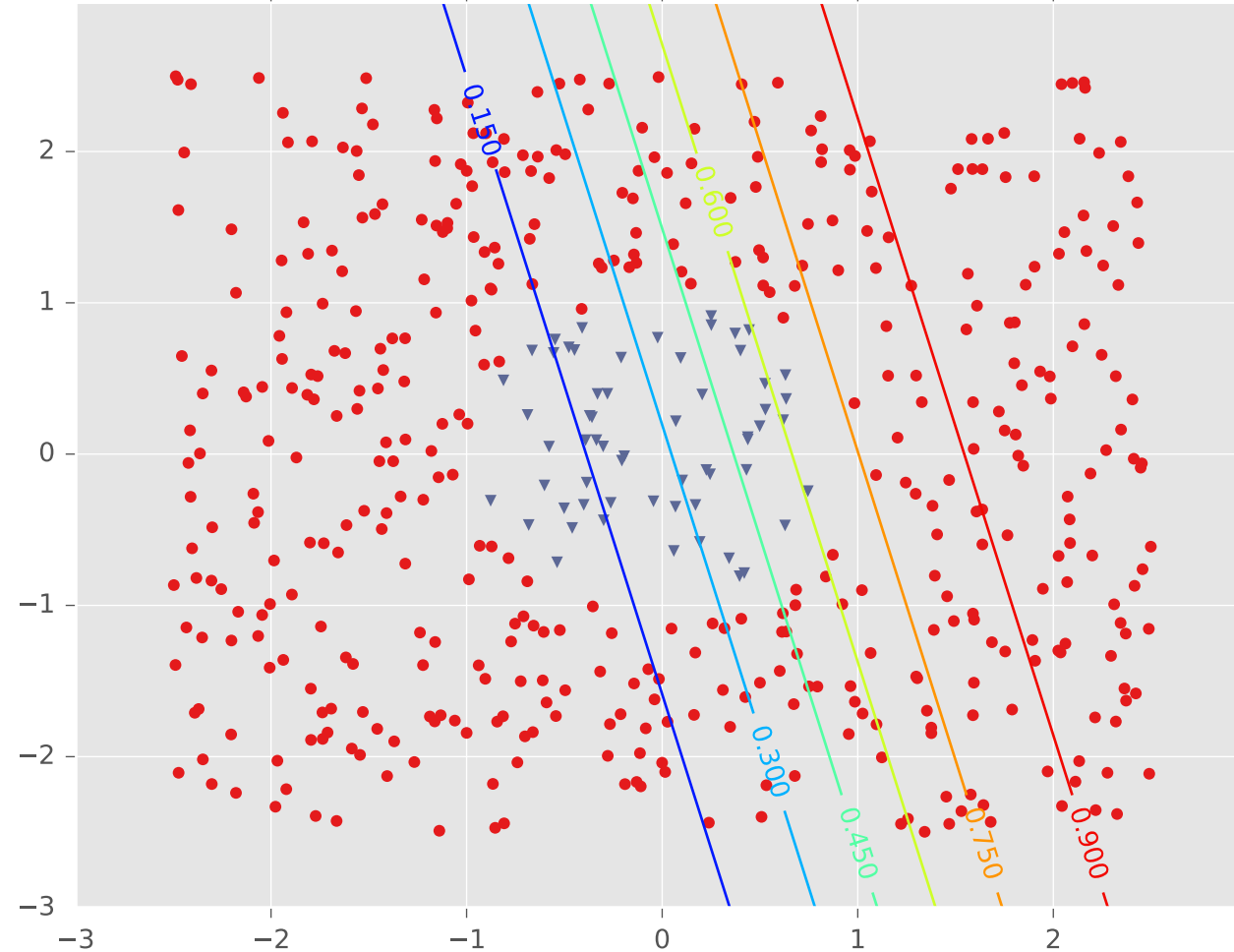
# Example #2: One Pocket

Tuned Neural Network (hidden=3, activation=logistic)



# Example #2: One Pocket

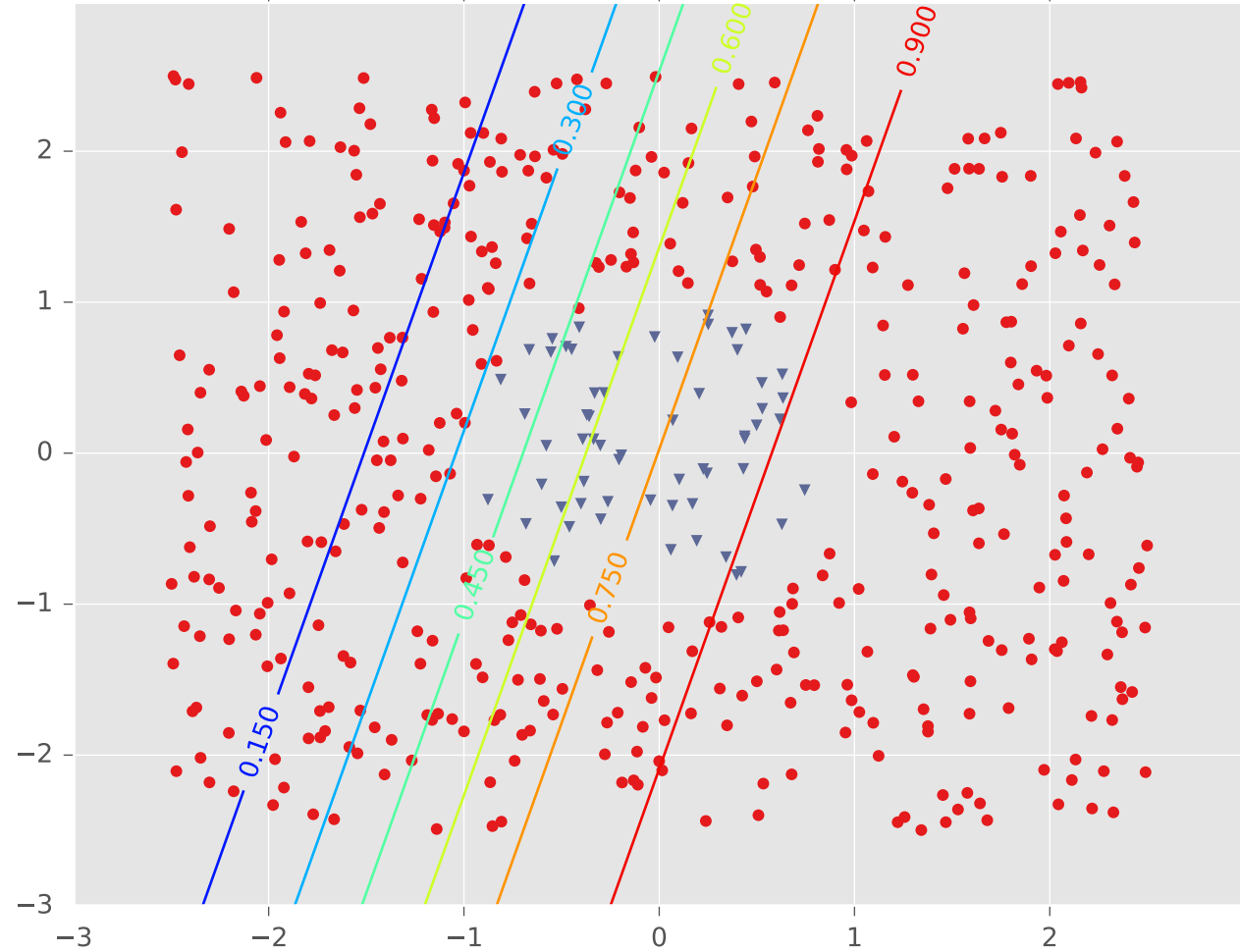
LR1 for Tuned Neural Network (hidden=3, activation=logistic)





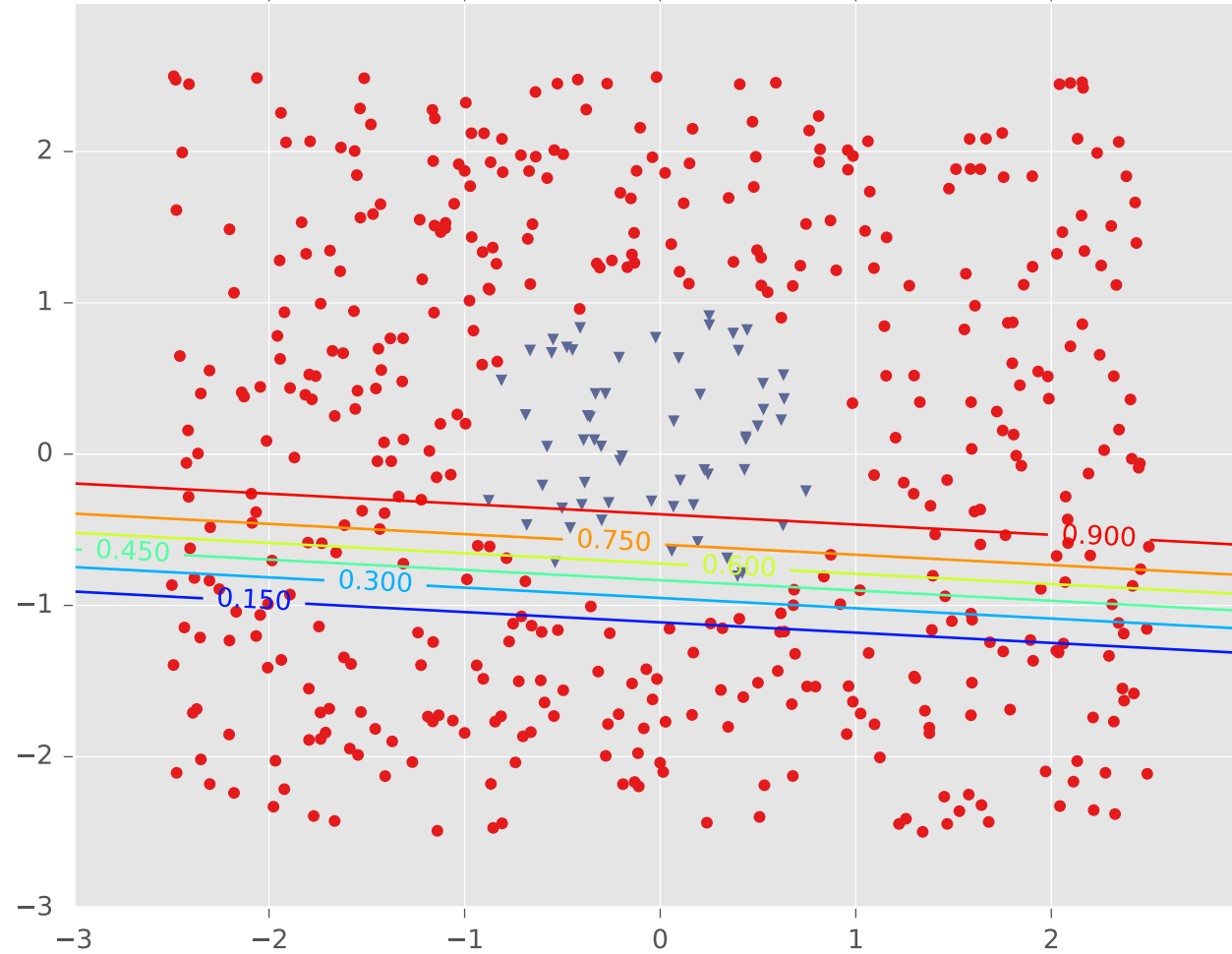
# Example #2: One Pocket

LR2 for Tuned Neural Network (hidden=3, activation=logistic)



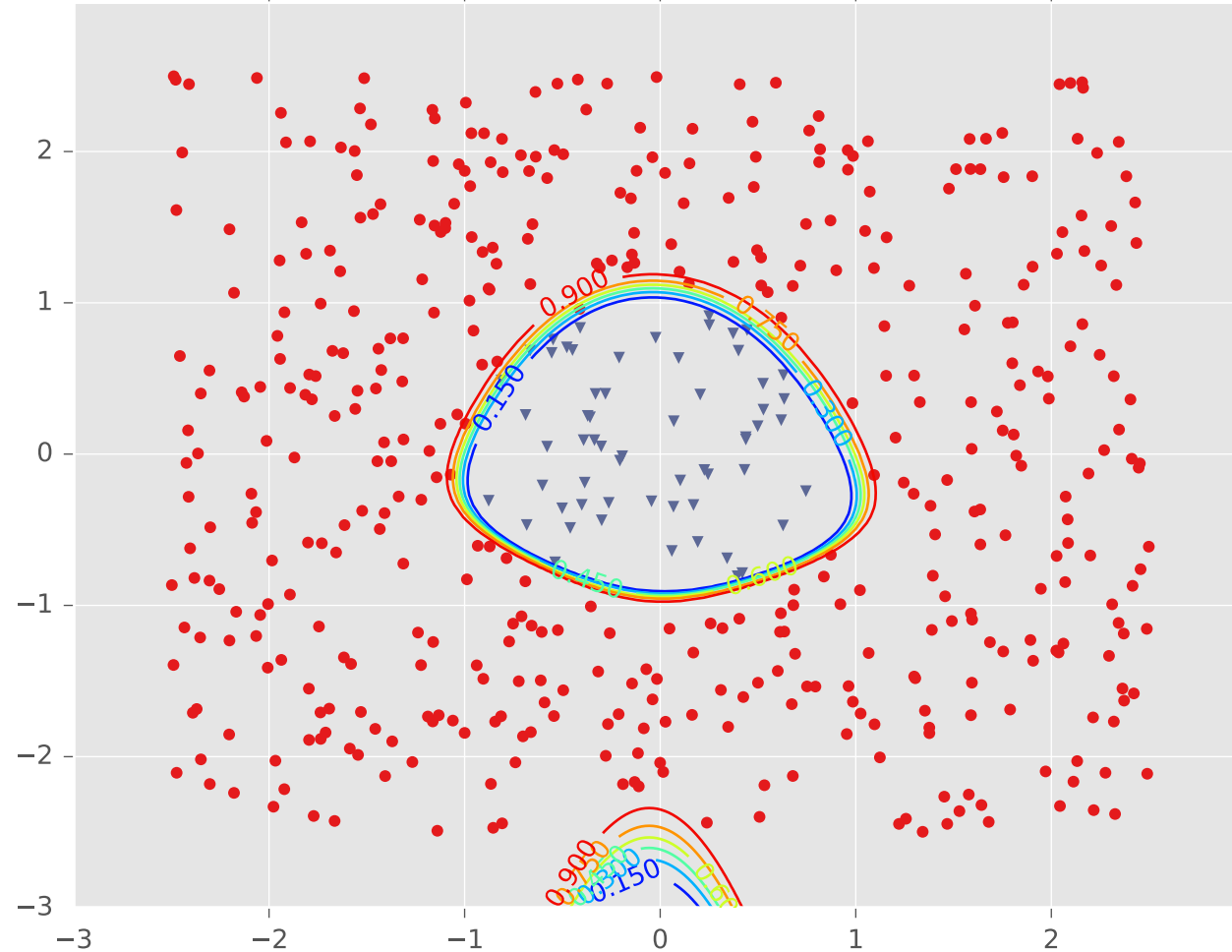
# Example #2: One Pocket

LR3 for Tuned Neural Network (hidden=3, activation=logistic)

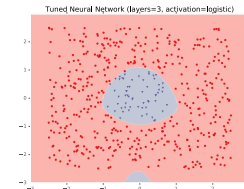
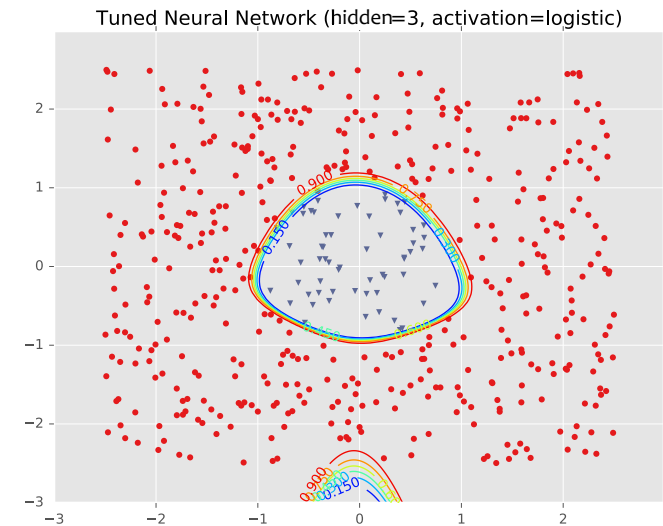
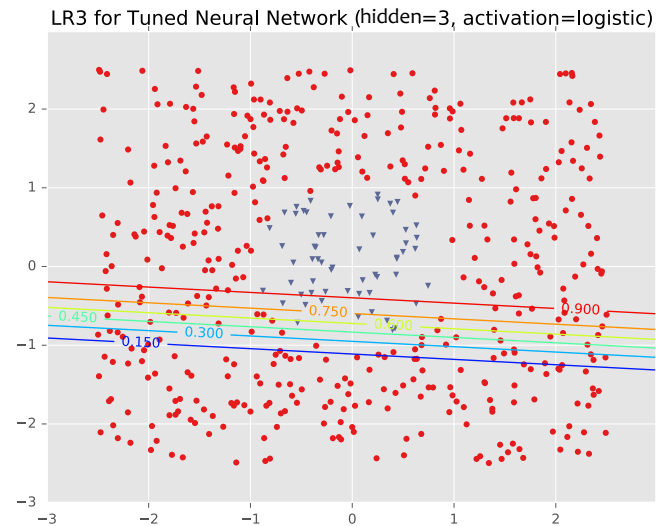
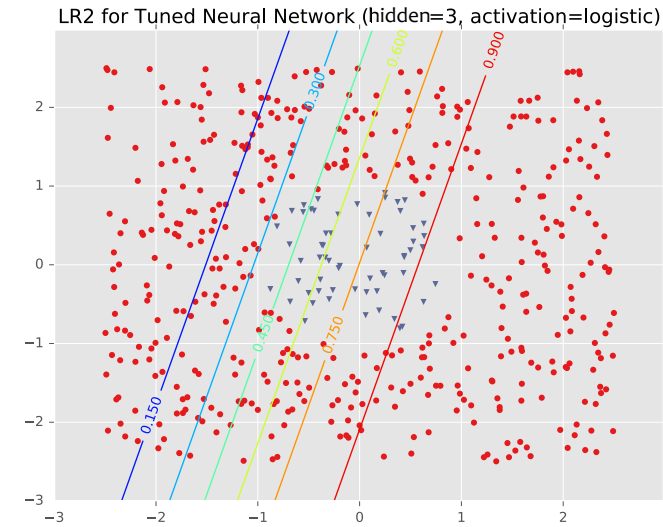
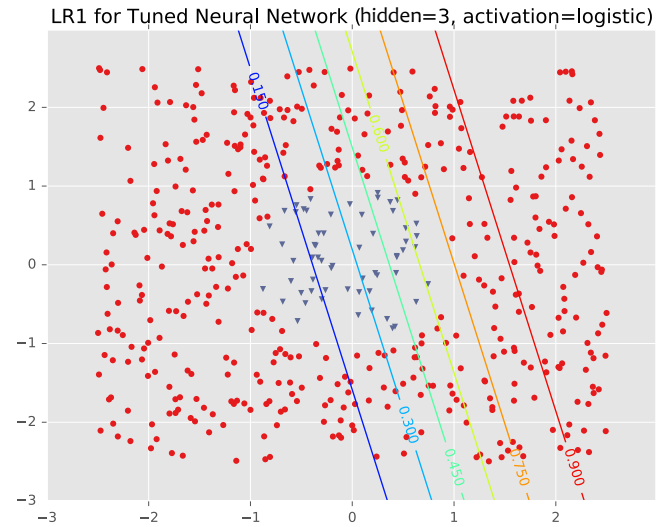


# Example #2: One Pocket

Tuned Neural Network (hidden=3, activation=logistic)



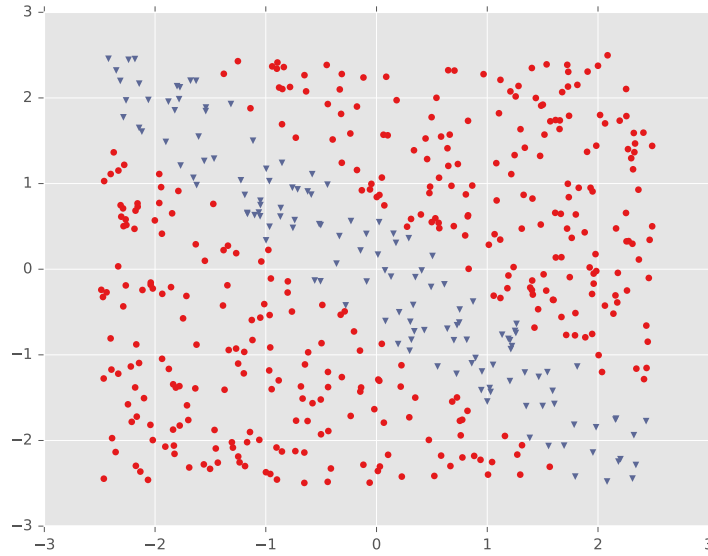
# Example #2: One Pocket



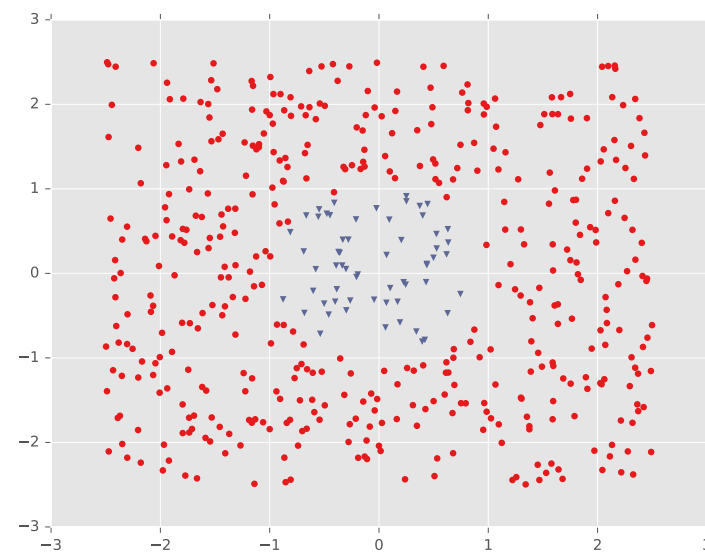
Examples 3 and 4

# **DECISION BOUNDARY EXAMPLES**

### Example #1: Diagonal Band



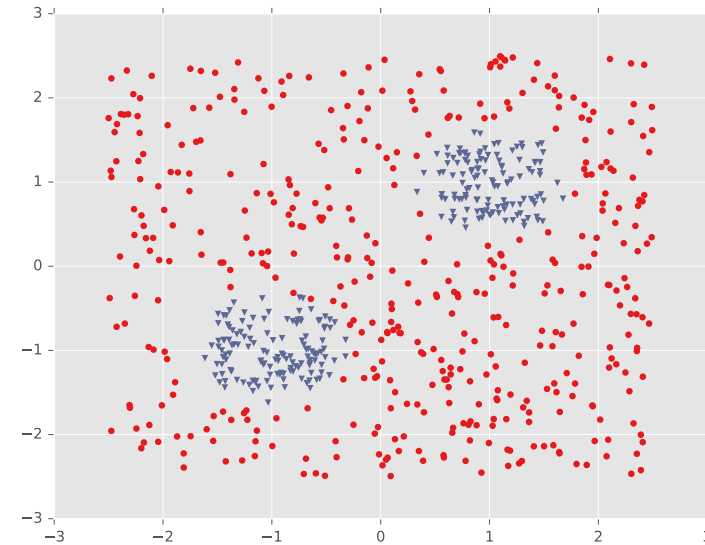
### Example #2: One Pocket



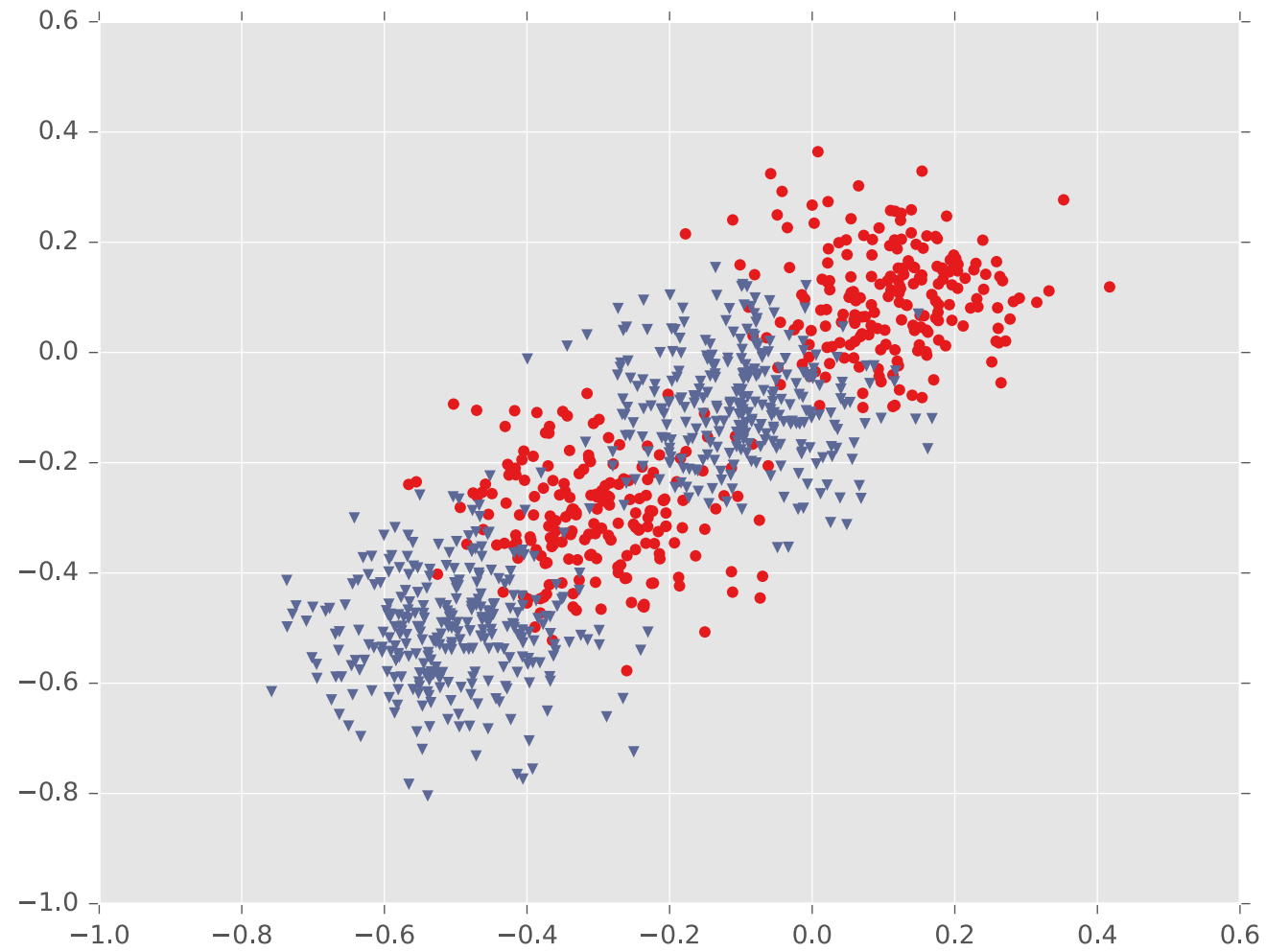
### Example #3: Four Gaussians



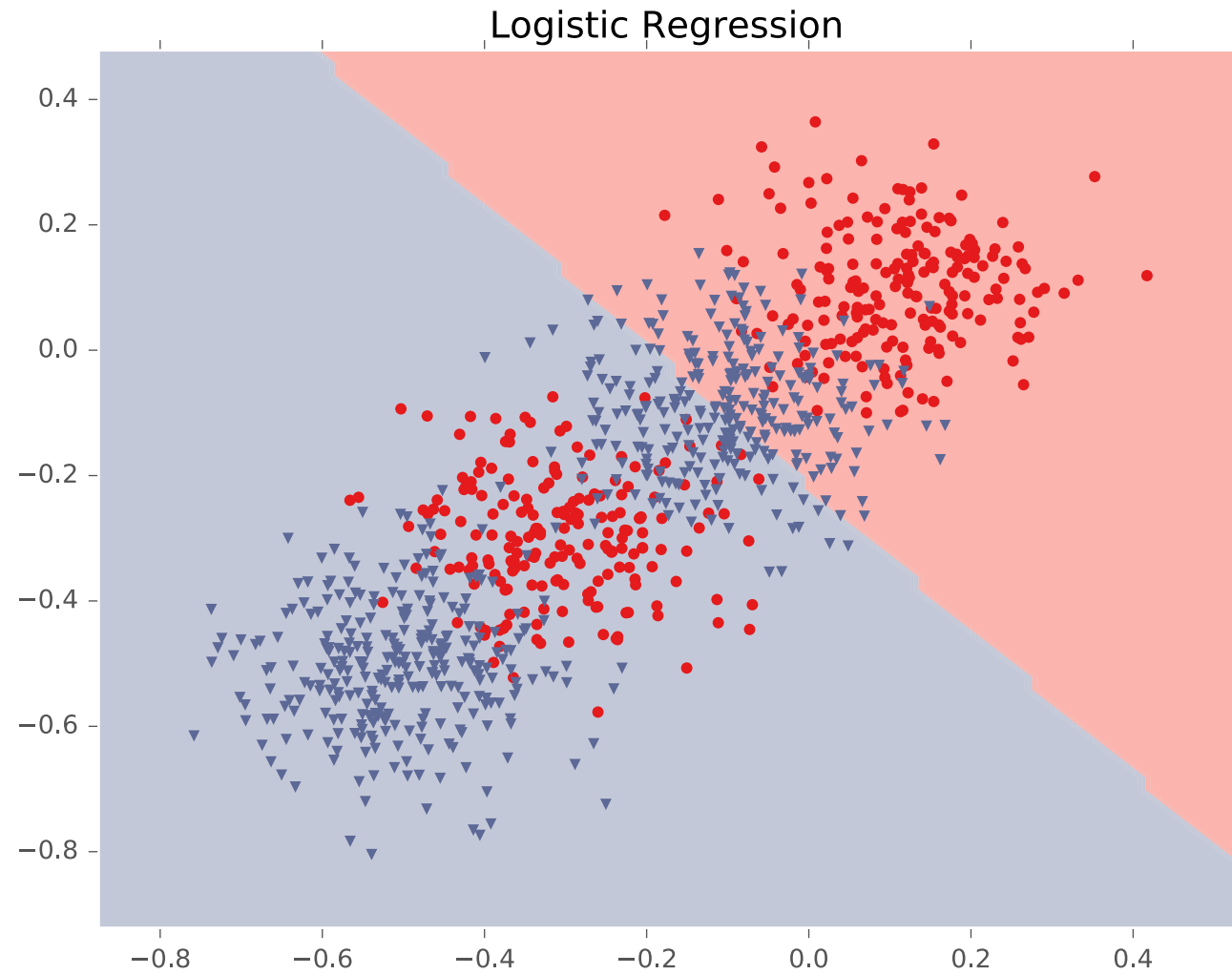
### Example #4: Two Pockets



# Example #3: Four Gaussians

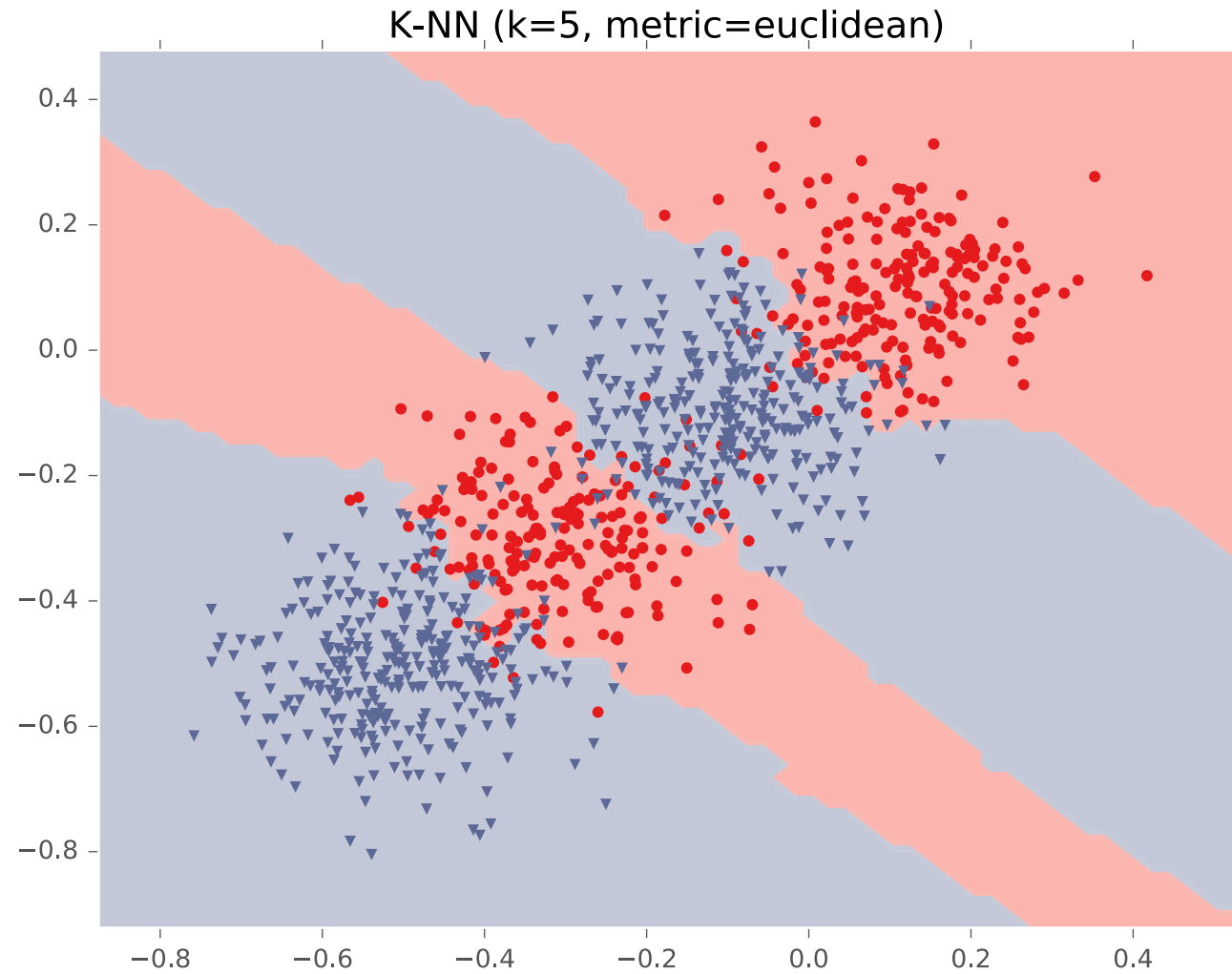


# Example #3: Four Gaussians

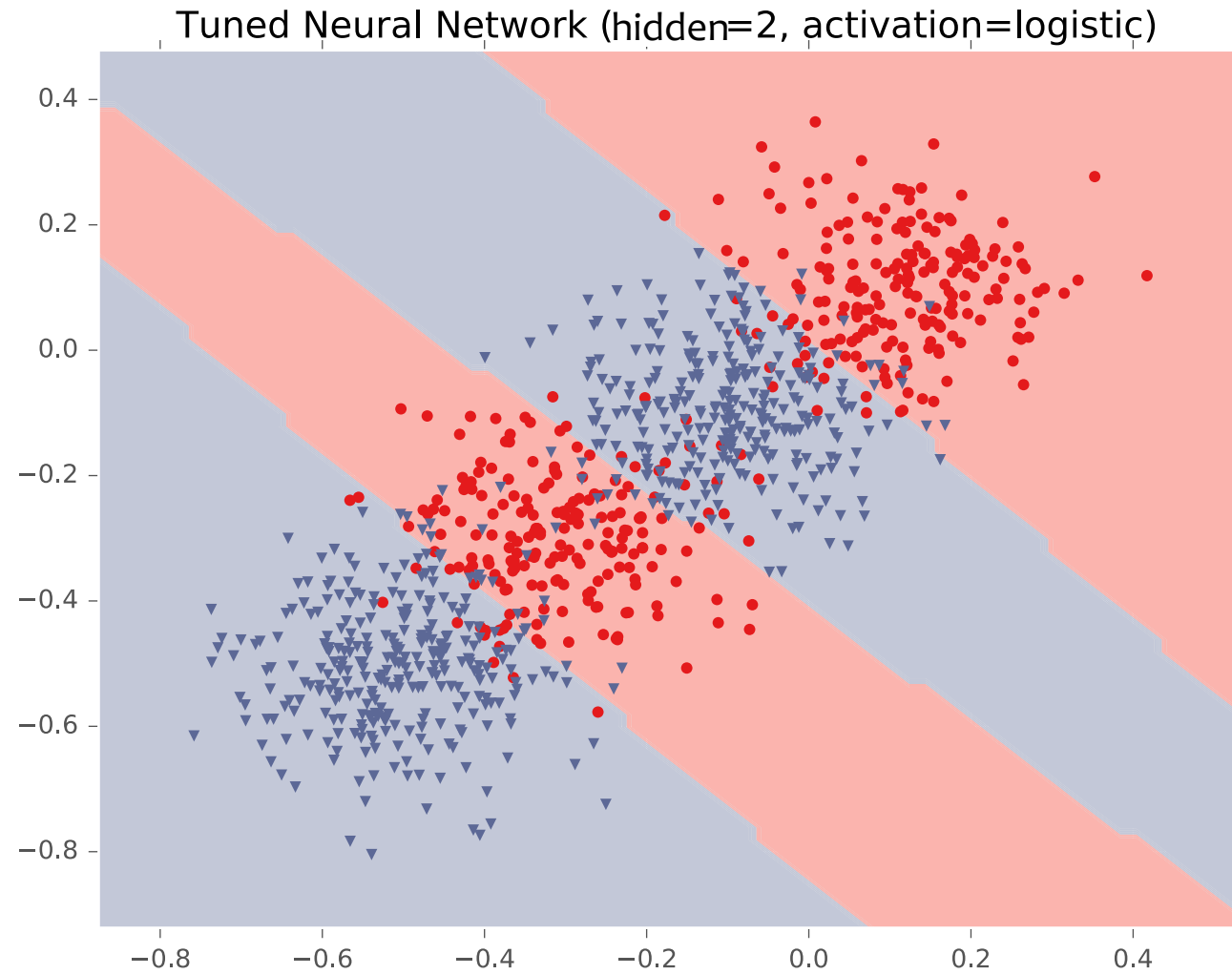




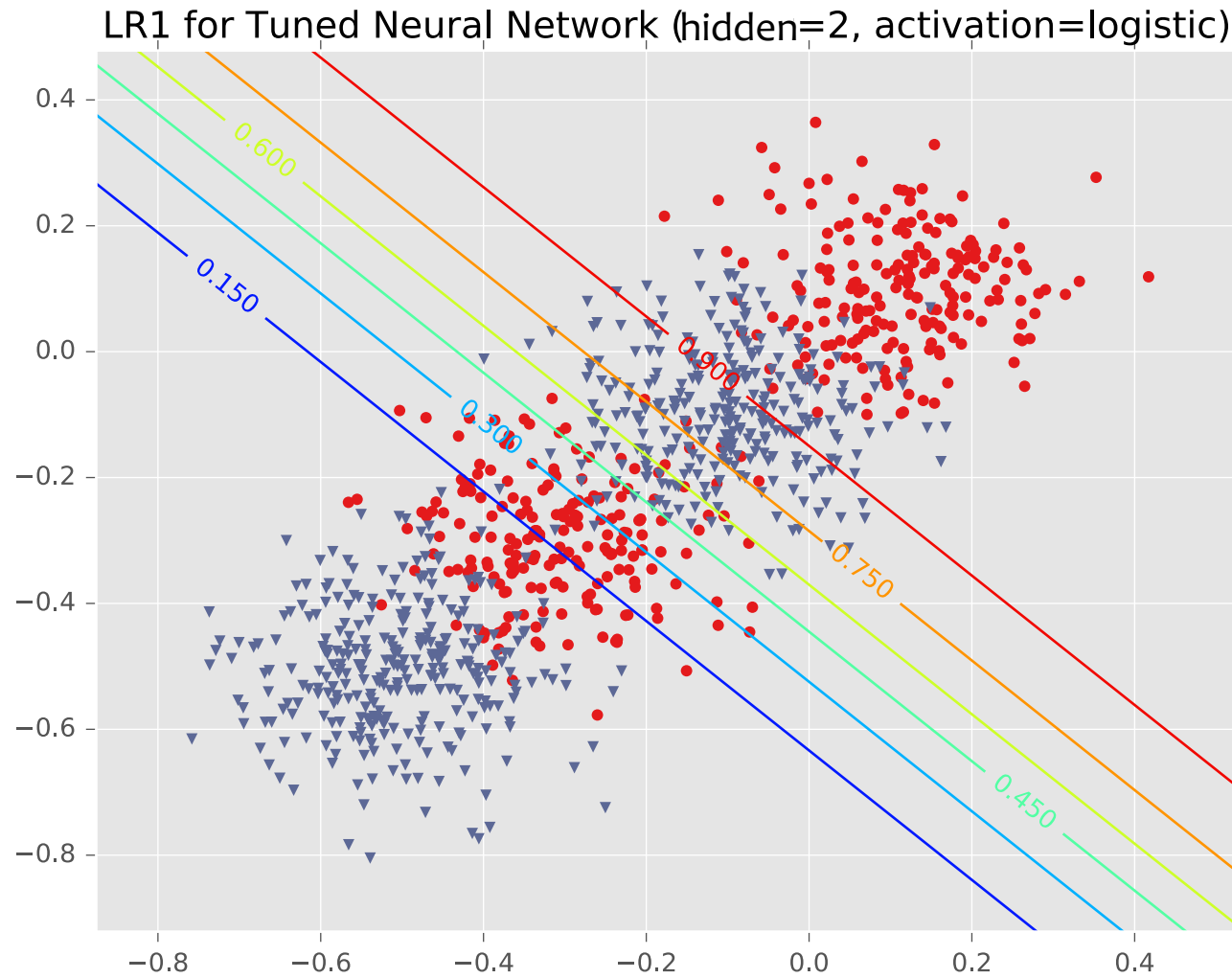
# Example #3: Four Gaussians



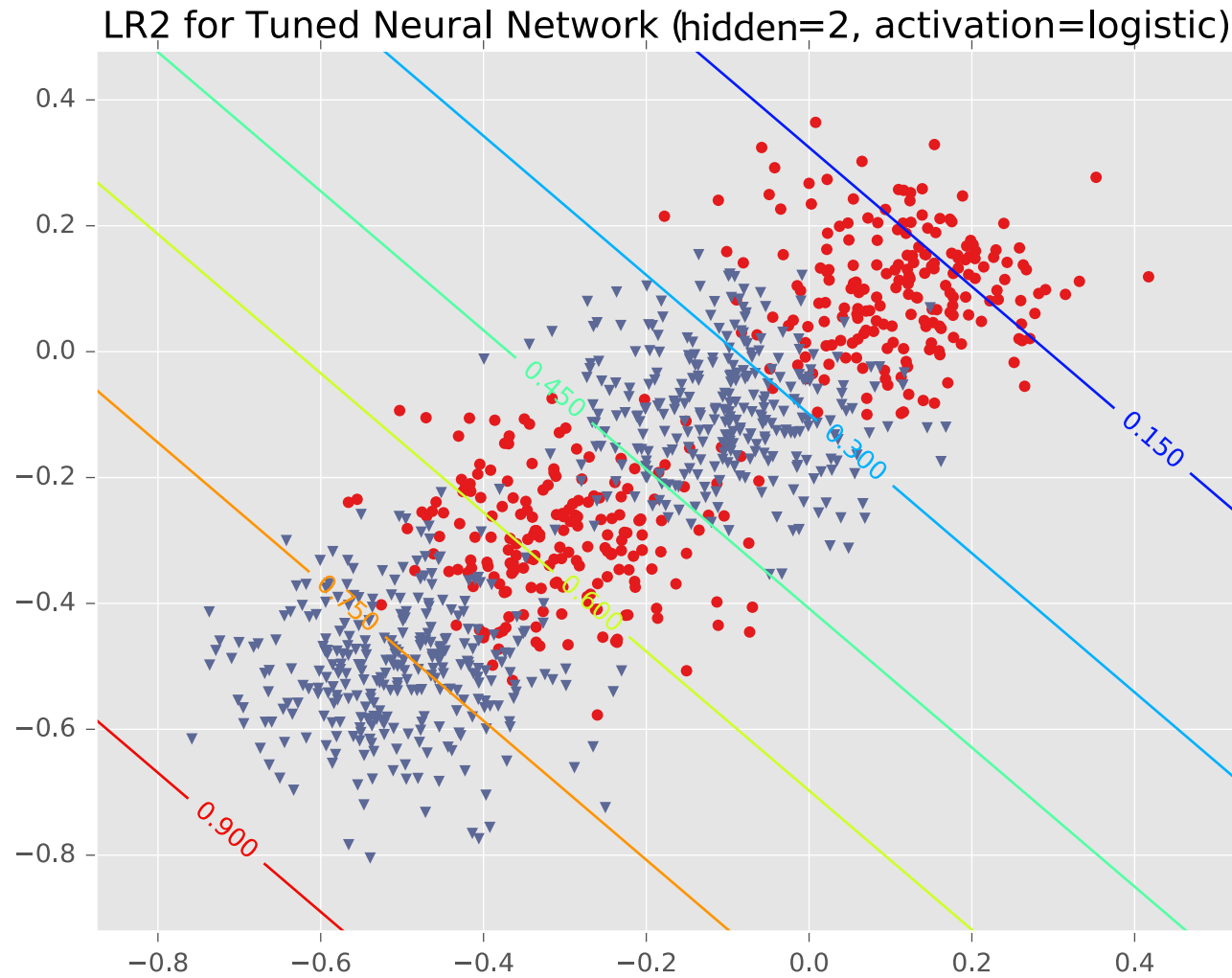
# Example #3: Four Gaussians



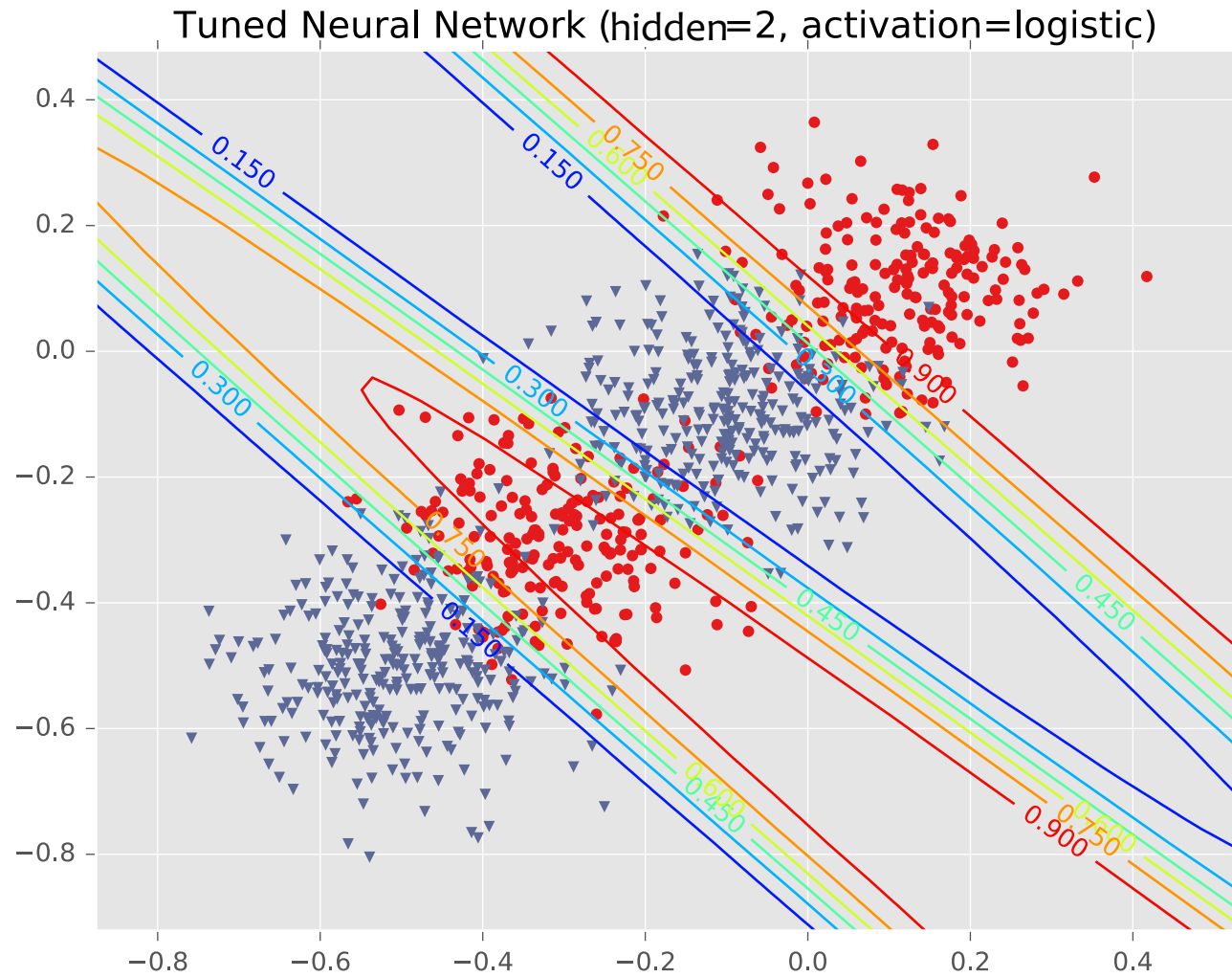
# Example #3: Four Gaussians



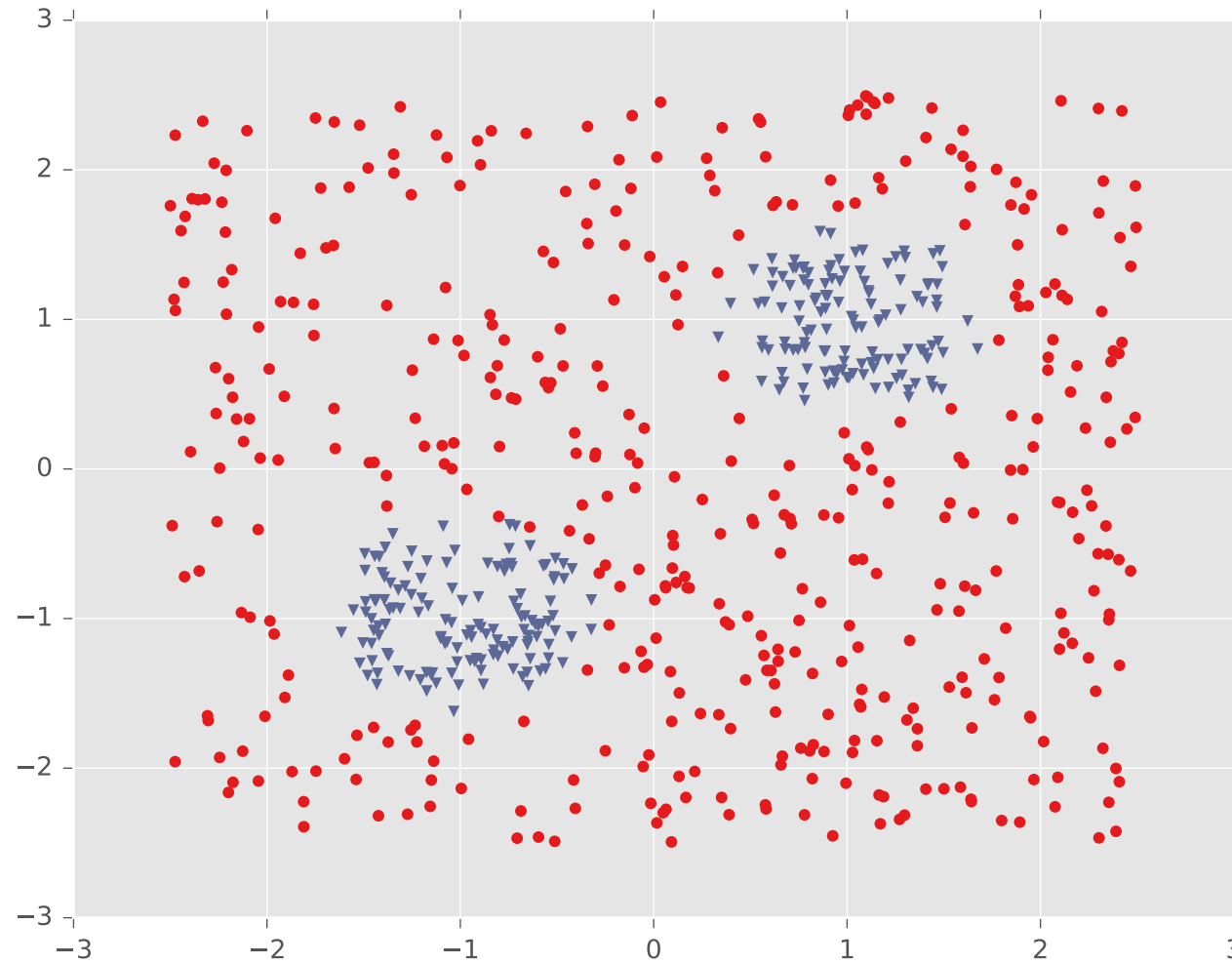
# Example #3: Four Gaussians



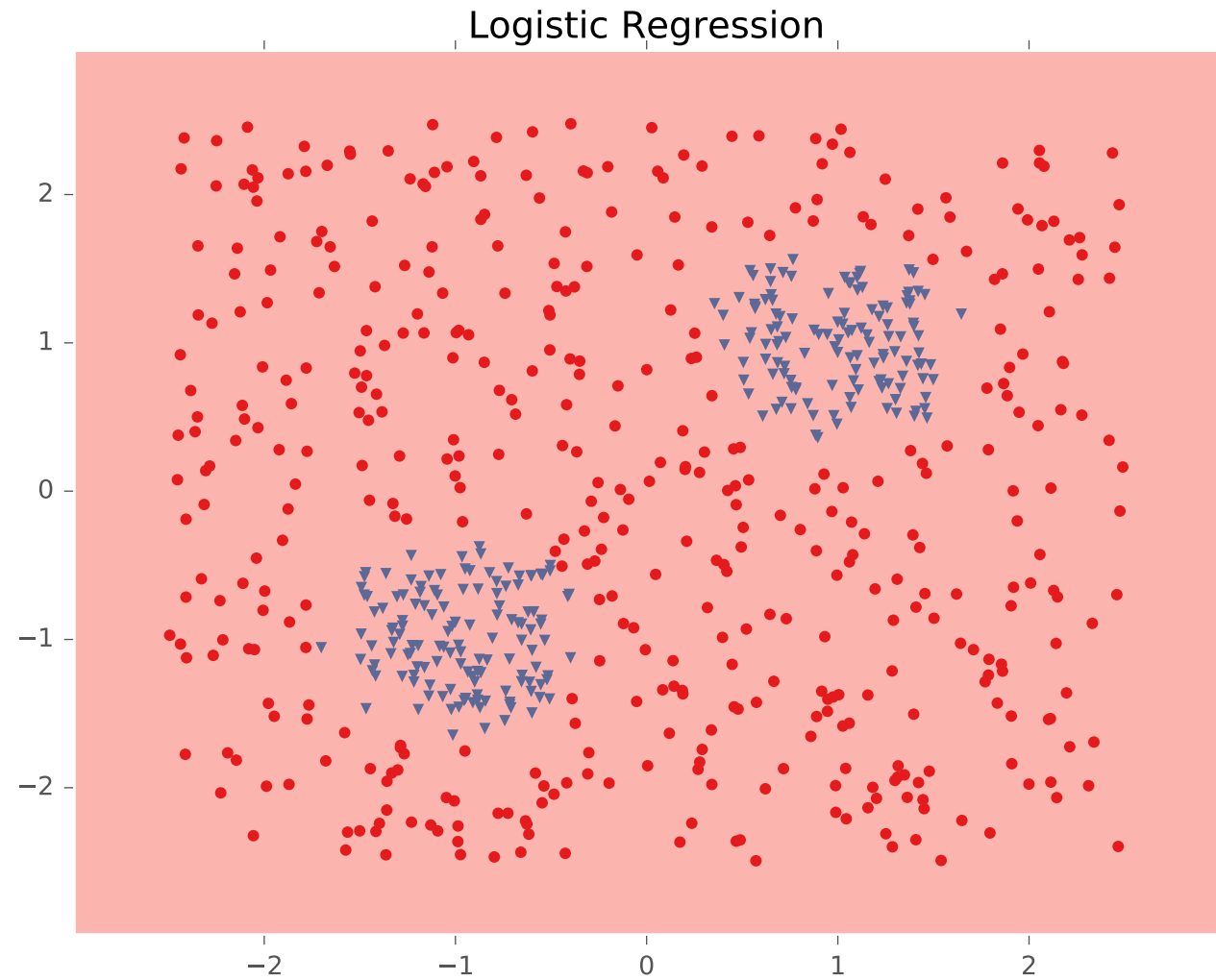
# Example #3: Four Gaussians



# Example #4: Two Pockets

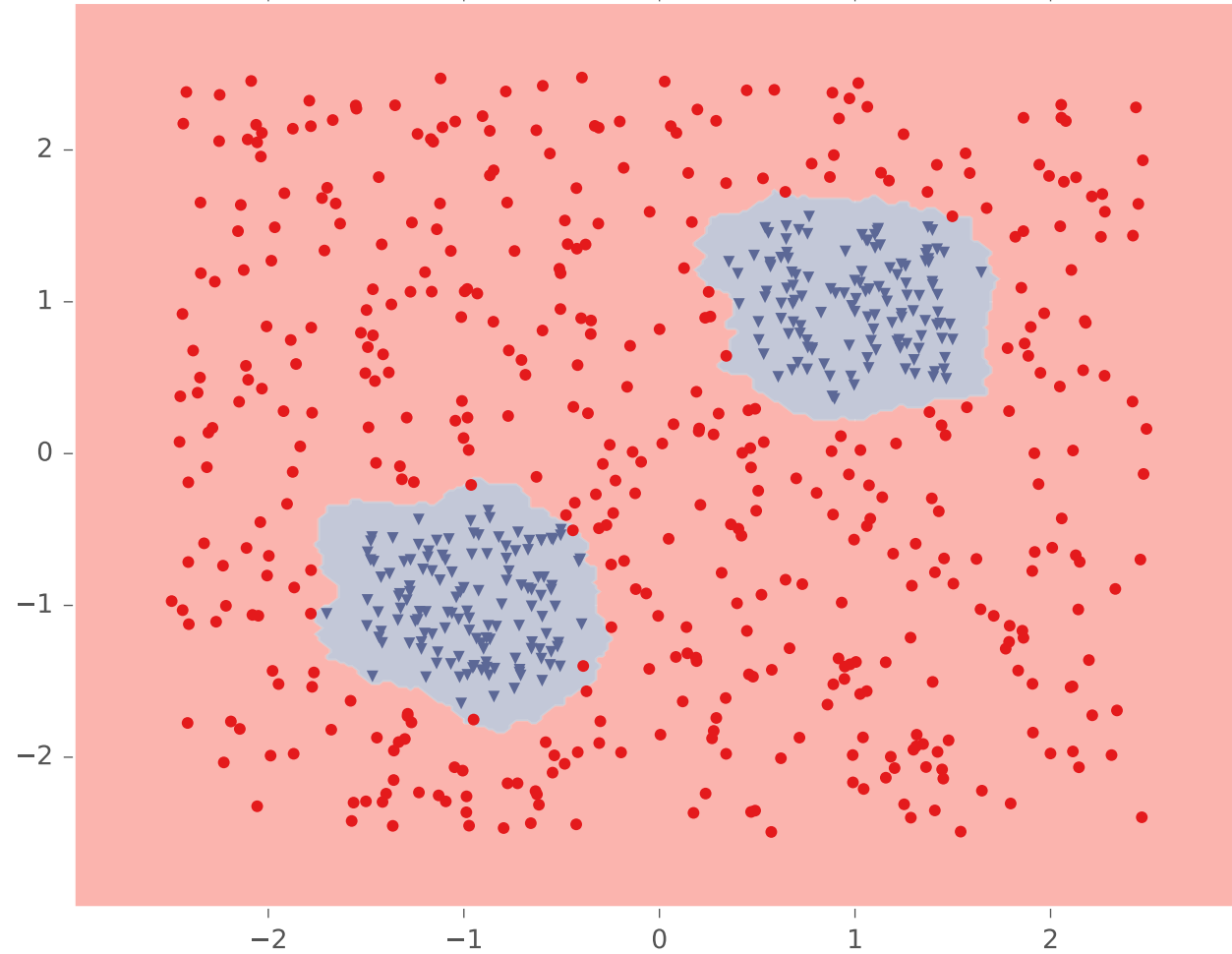


# Example #4: Two Pockets



# Example #4: Two Pockets

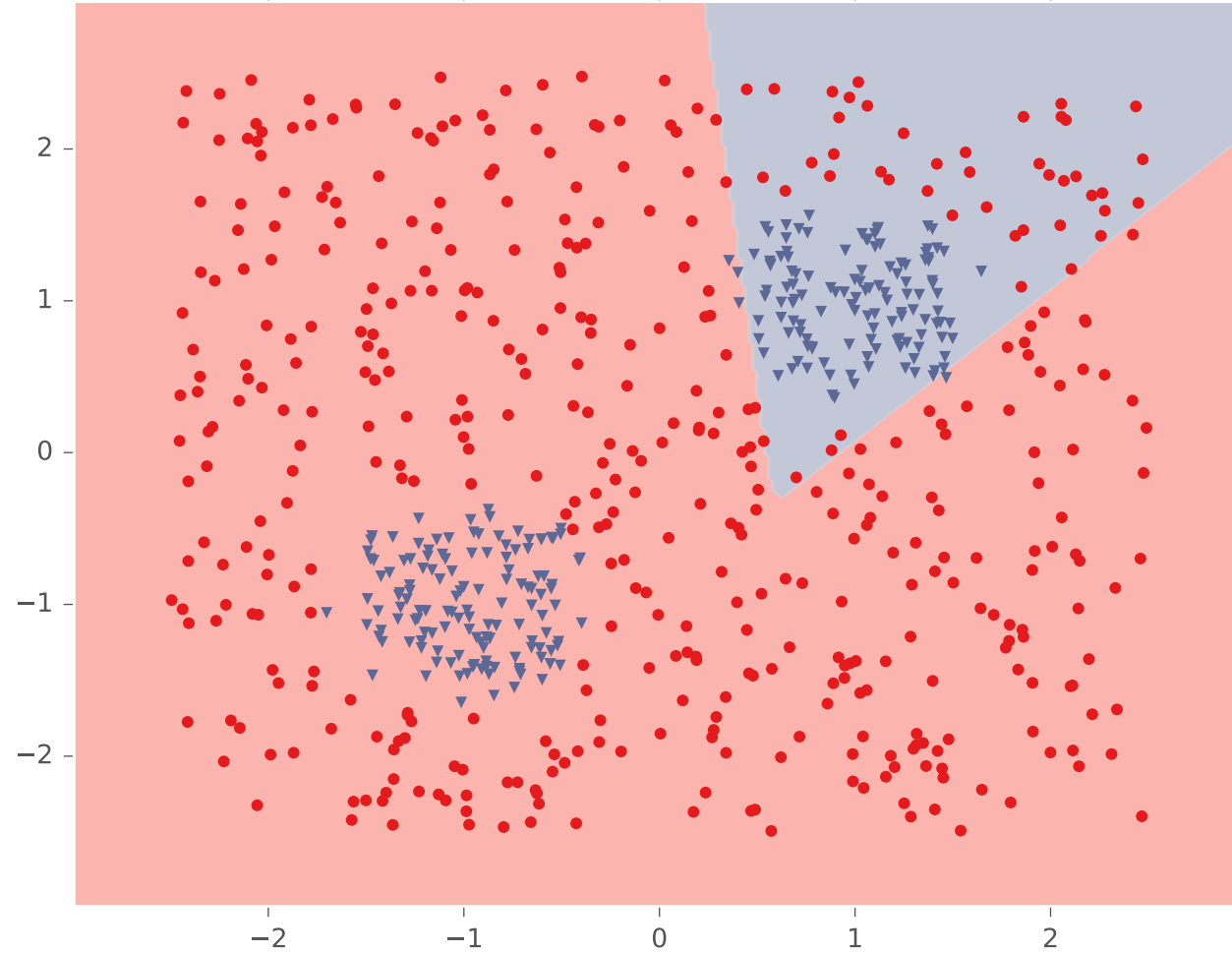
K-NN (k=5, metric=euclidean)





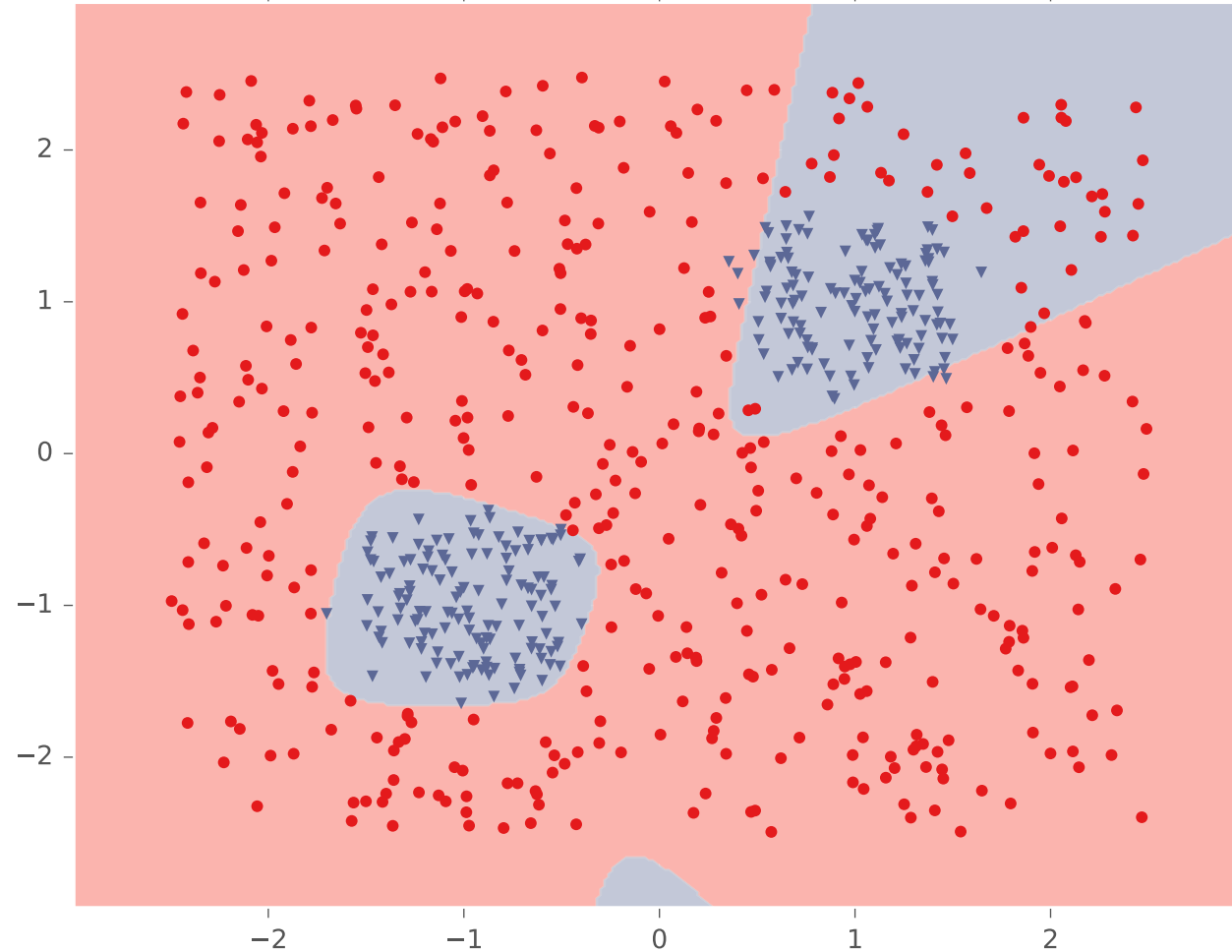
# Example #4: Two Pockets

Tuned Neural Network (hidden=2, activation=logistic)



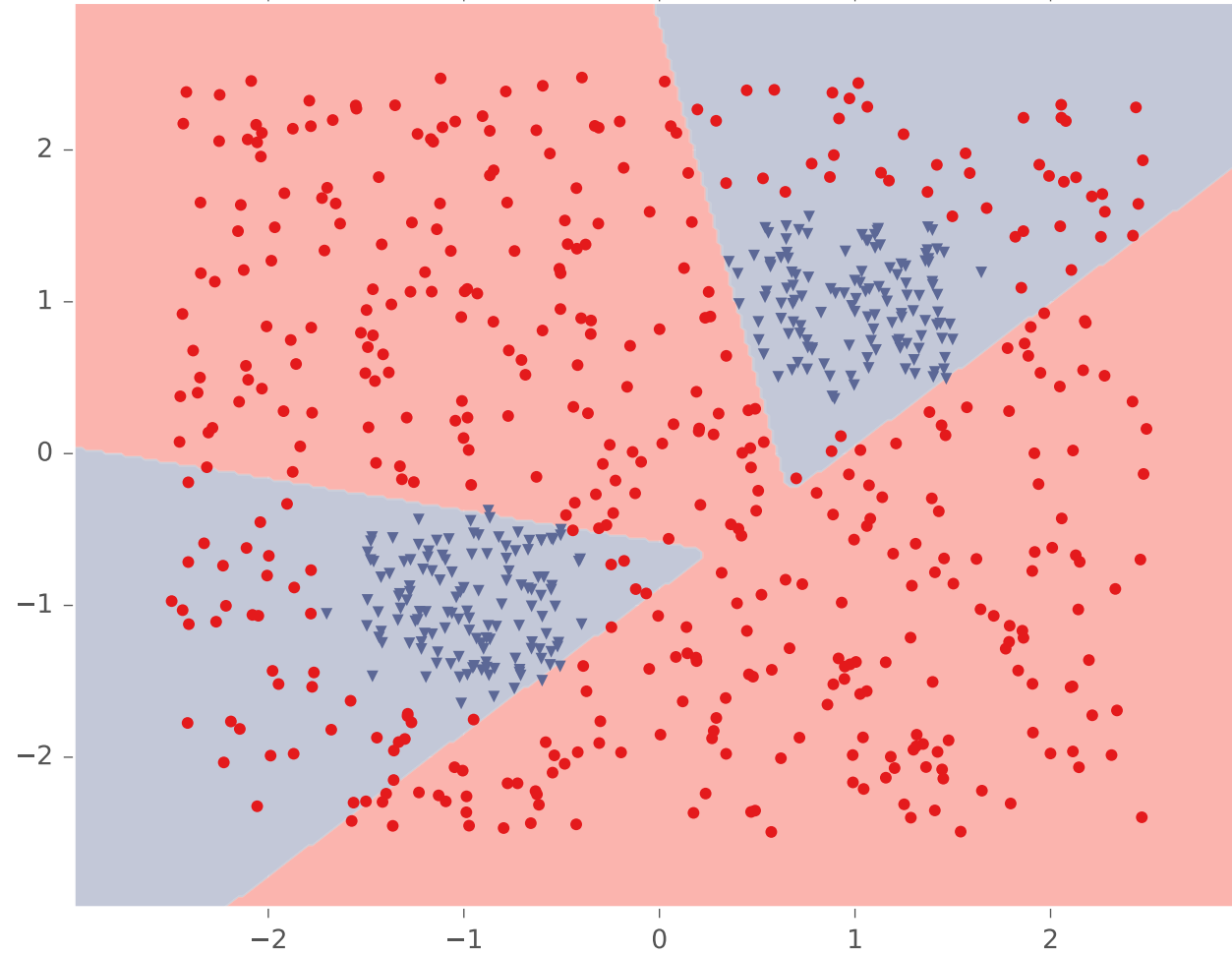
# Example #4: Two Pockets

Tuned Neural Network (hidden=3, activation=logistic)



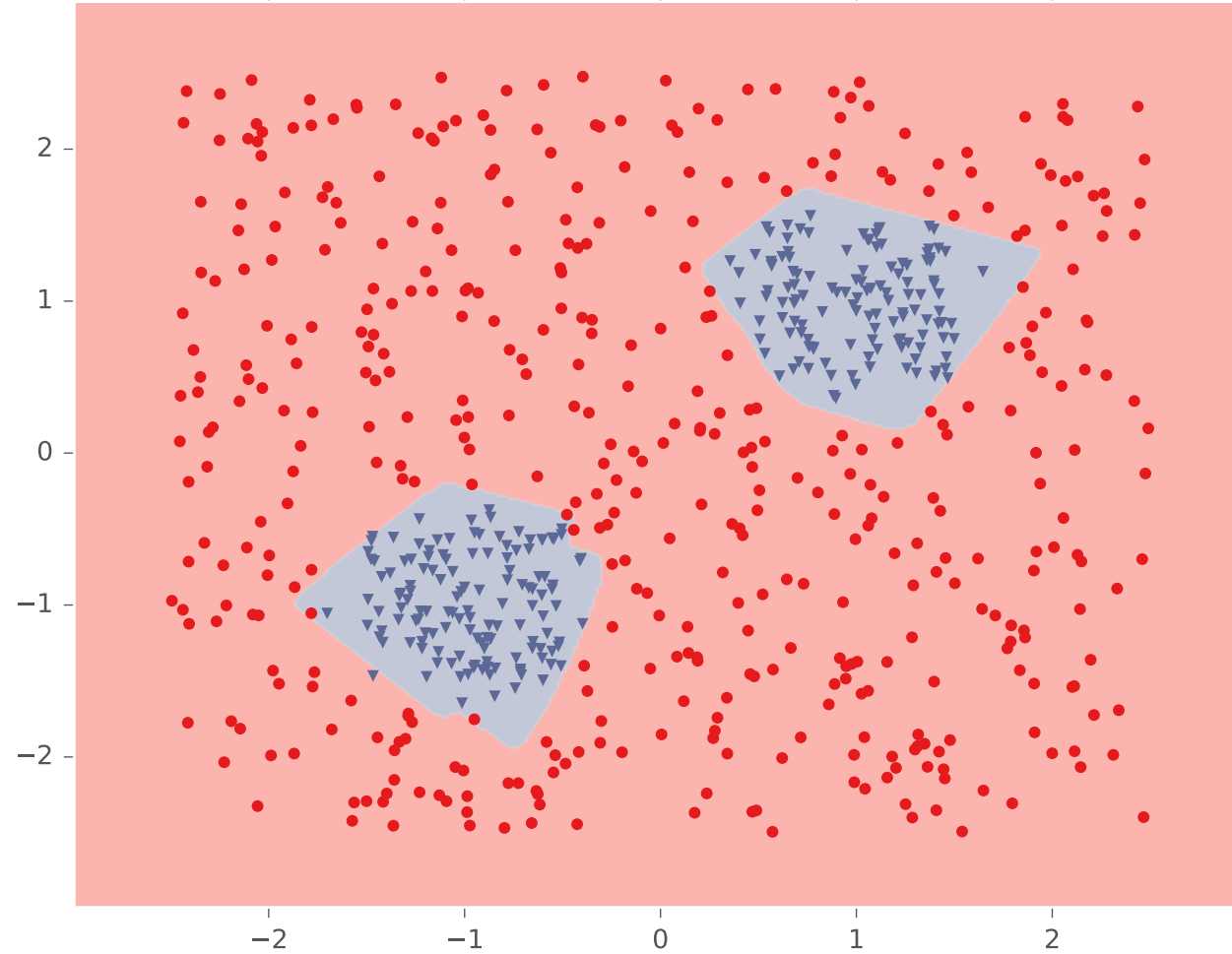
# Example #4: Two Pockets

Tuned Neural Network (hidden=4, activation=logistic)



# Example #4: Two Pockets

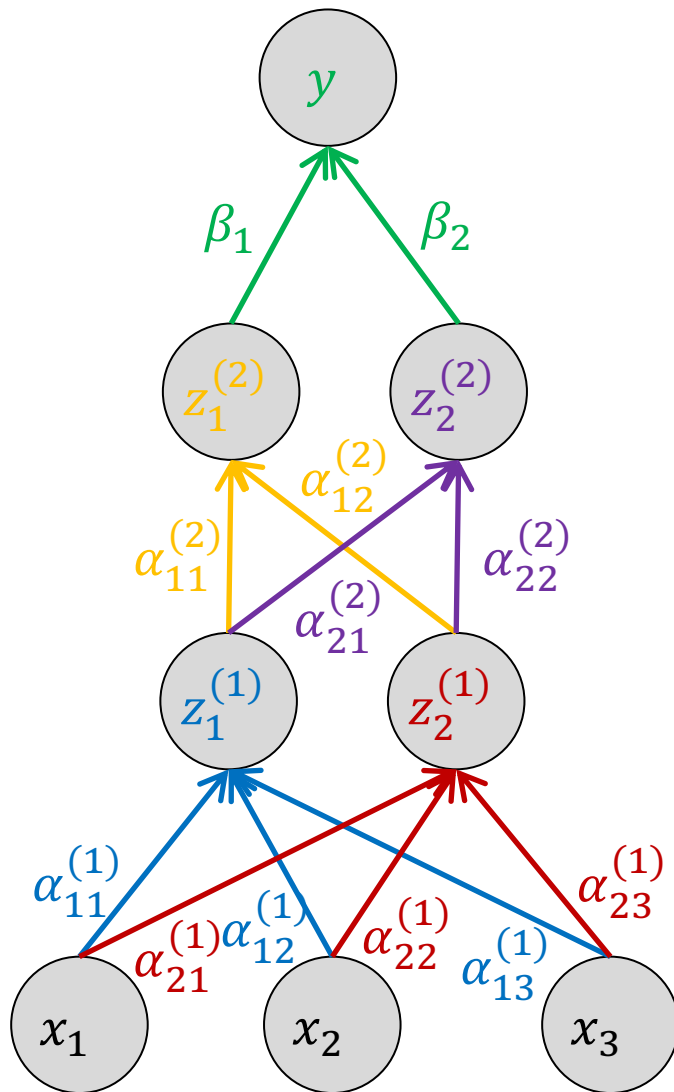
Tuned Neural Network (hidden=10, activation=logistic)



# **BUILDING DEEPER NETWORKS**

# Neural Network

*Example: Neural Network with 2 Hidden Layers and 2 Hidden Units*



$$z_1^{(1)} = \sigma(\alpha_{11}^{(1)} x_1 + \alpha_{12}^{(1)} x_2 + \alpha_{13}^{(1)} x_3 + \alpha_{10}^{(1)})$$

$$z_2^{(1)} = \sigma(\alpha_{21}^{(1)} x_1 + \alpha_{22}^{(1)} x_2 + \alpha_{23}^{(1)} x_3 + \alpha_{20}^{(1)})$$

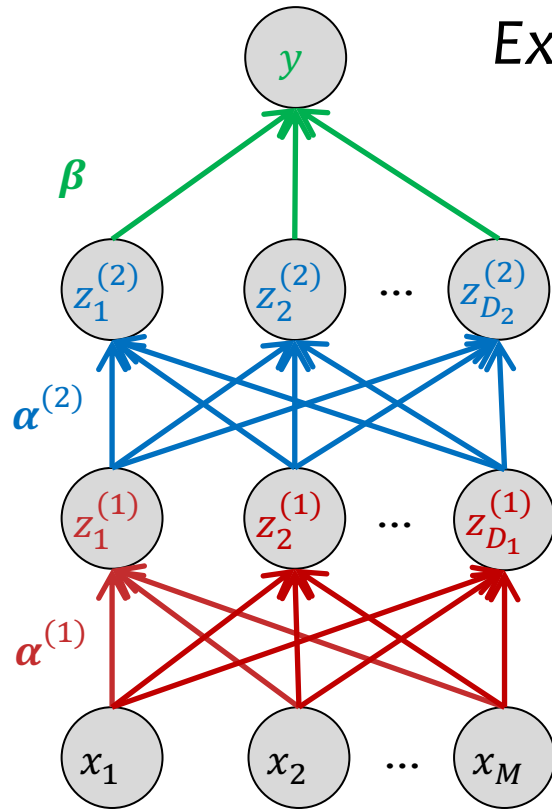
$$z_1^{(2)} = \sigma(\alpha_{11}^{(2)} z_1^{(1)} + \alpha_{12}^{(2)} z_2^{(1)} + \alpha_{10}^{(2)})$$

$$z_2^{(2)} = \sigma(\alpha_{21}^{(2)} z_1^{(1)} + \alpha_{22}^{(2)} z_2^{(1)} + \alpha_{20}^{(2)})$$

$$y = \sigma(\beta_1 z_1^{(2)} + \beta_2 z_2^{(2)} + \beta_0)$$

# Neural Network (Matrix Form)

*Example: Arbitrary Feed-forward Neural Network*



$$\beta \in \mathbb{R}^{D_2}$$

$$\beta_0 \in \mathbb{R}$$

$$\alpha^{(2)} \in \mathbb{R}^{D_1 \times D_2}$$

$$\mathbf{b}^{(2)} \in \mathbb{R}^{D_2}$$

$$\alpha^{(1)} \in \mathbb{R}^{M \times D_1}$$

$$\mathbf{b}^{(1)} \in \mathbb{R}^{D_1}$$

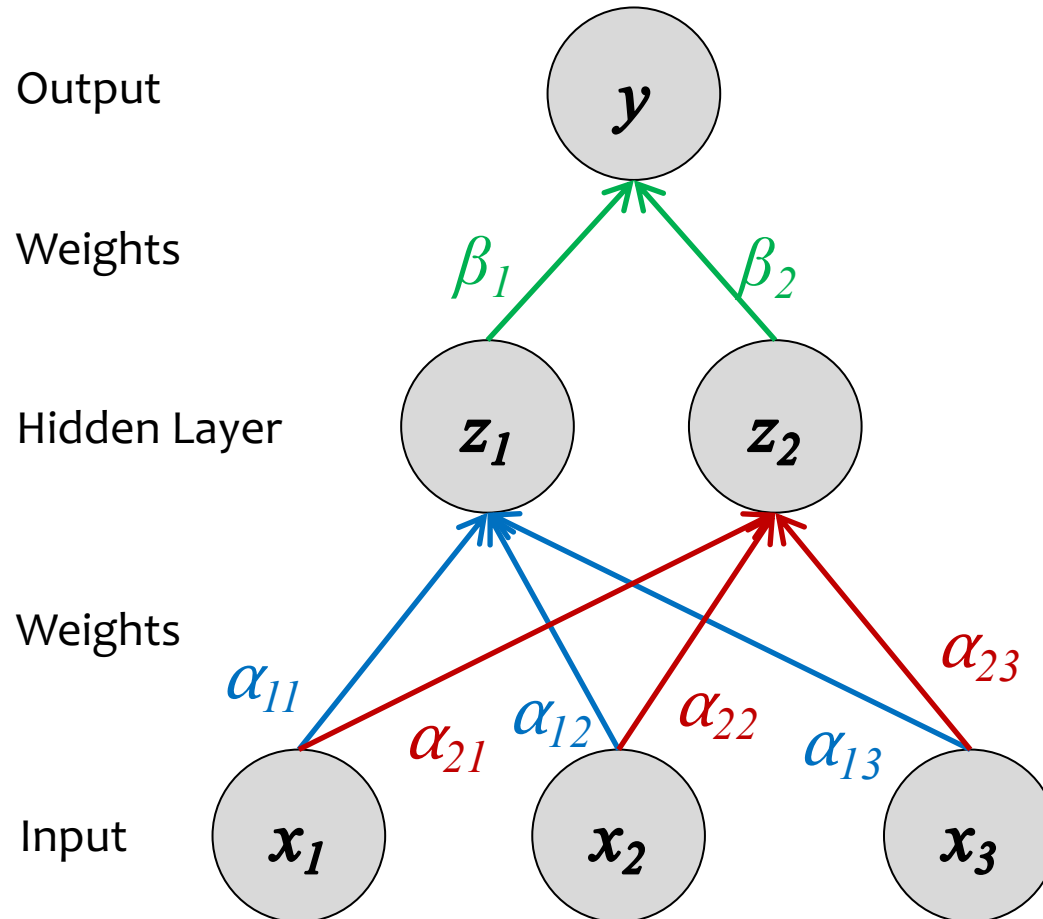
$$y = \sigma((\beta)^T \mathbf{z}^{(2)} + \beta_0)$$

$$\mathbf{z}^{(2)} = \sigma((\alpha^{(2)})^T \mathbf{z}^{(1)} + \mathbf{b}^{(2)})$$

$$\mathbf{z}^{(1)} = \sigma((\alpha^{(1)})^T \mathbf{x} + \mathbf{b}^{(1)})$$

# Neural Network (Vector Form)

Neural Network with 1 Hidden Layers  
and 2 Hidden Units (Matrix Form)



$$y = \sigma(\beta^T \mathbf{z})$$

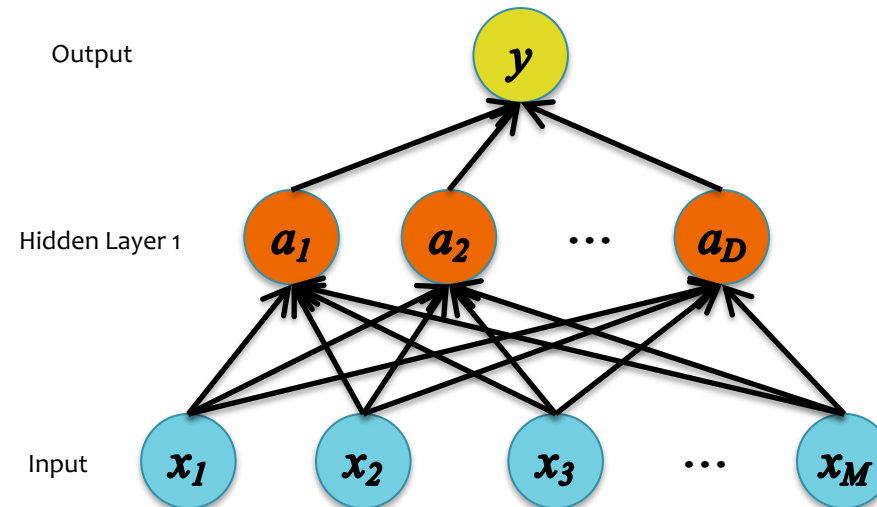
$$z_2 = \sigma(\alpha_{2,\cdot}^T \mathbf{x})$$

$$z_1 = \sigma(\alpha_{1,\cdot}^T \mathbf{x})$$



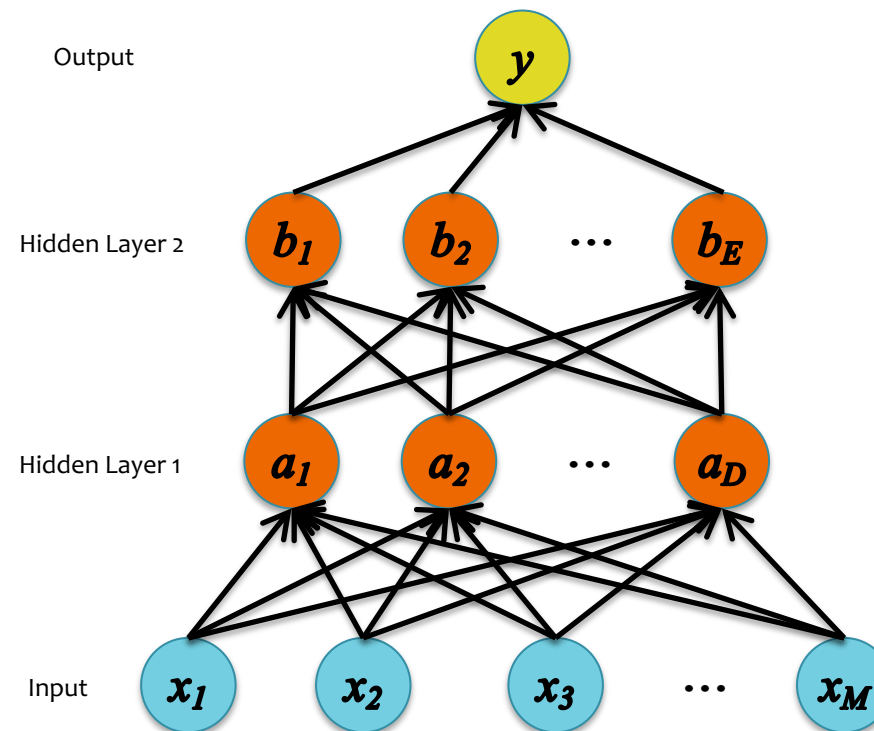
# Deeper Networks

*Q: How many layers should we use?*



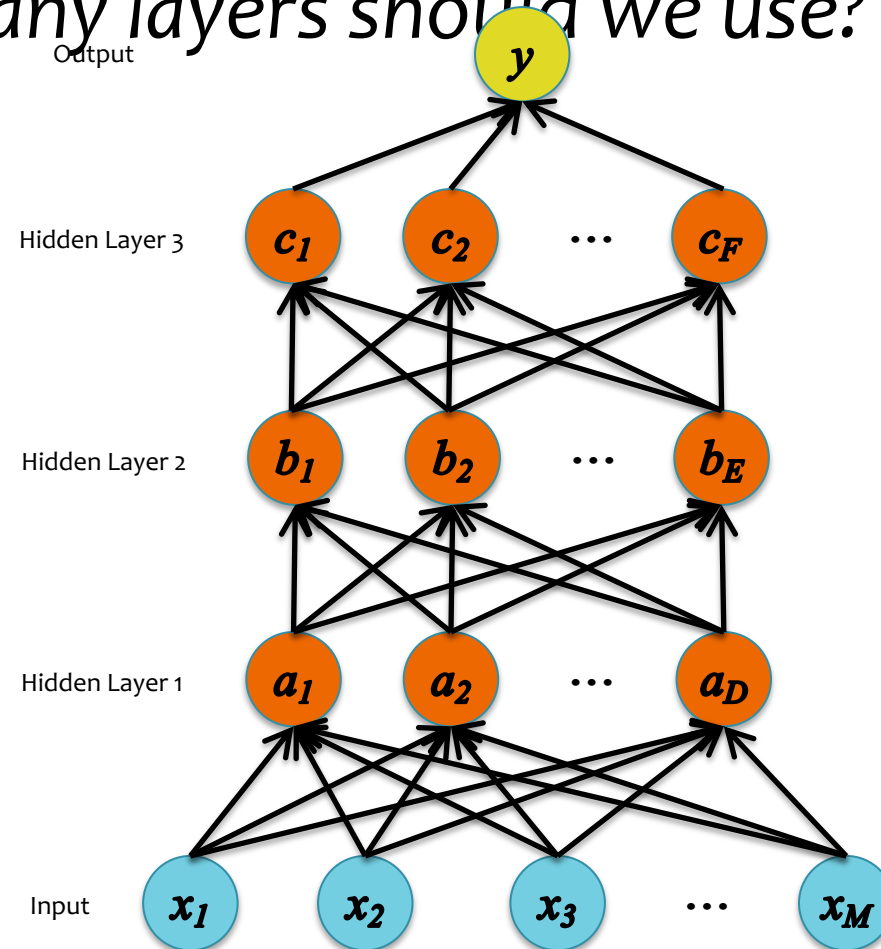
# Deeper Networks

*Q: How many layers should we use?*



# Deeper Networks

Q: *How many layers should we use?*



# Deeper Networks

*Q: How many layers should we use?*

- **Theoretical answer:**

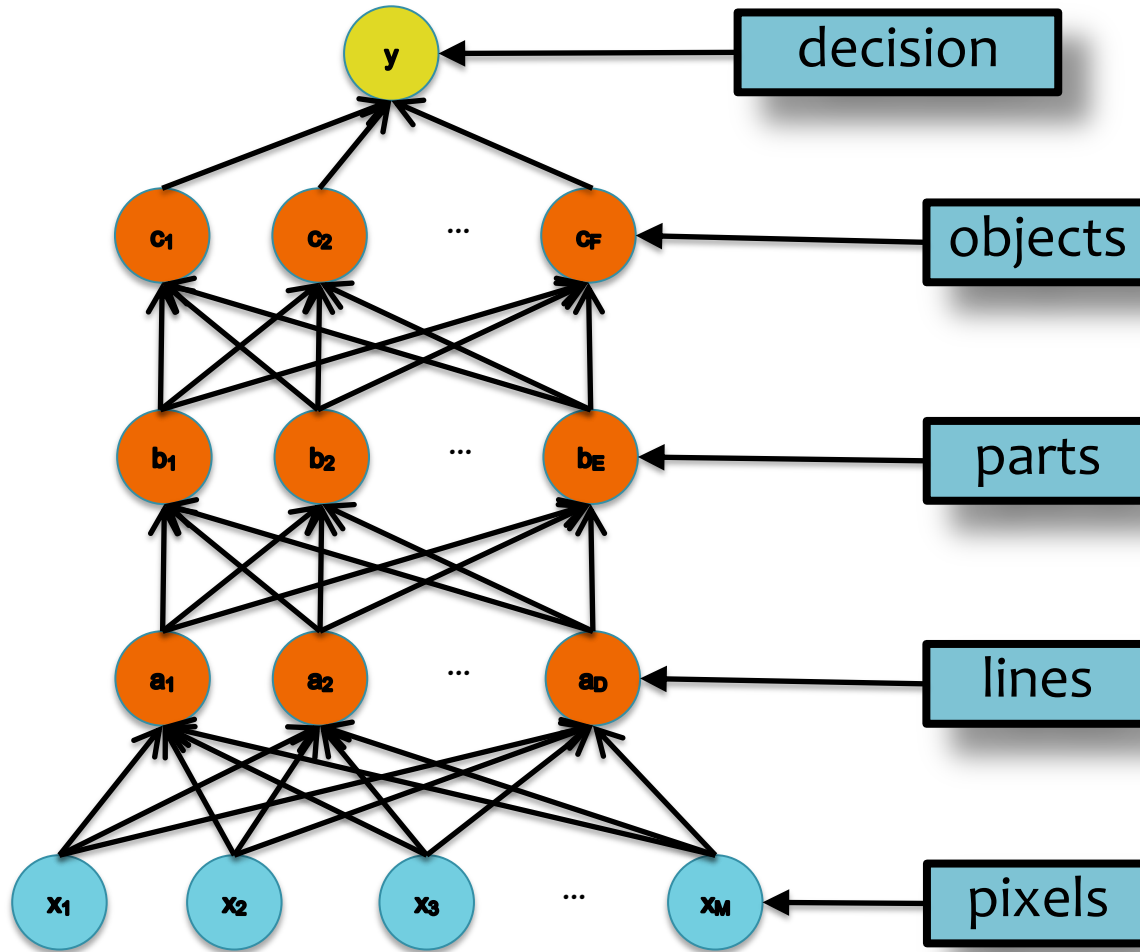
- A neural network with 1 hidden layer is a **universal function approximator**
- Cybenko (1989): For any continuous function  $g(\mathbf{x})$ , there exists a 1-hidden-layer neural net  $h_{\theta}(\mathbf{x})$  s.t.  $|h_{\theta}(\mathbf{x}) - g(\mathbf{x})| < \epsilon$  for all  $\mathbf{x}$ , assuming sigmoid activation functions

- **Empirical answer:**

- Before 2006: “Deep networks (e.g. 3 or more hidden layers) are too hard to train”
- After 2006: “Deep networks are easier to train than shallow networks (e.g. 2 or fewer layers) for many problems”

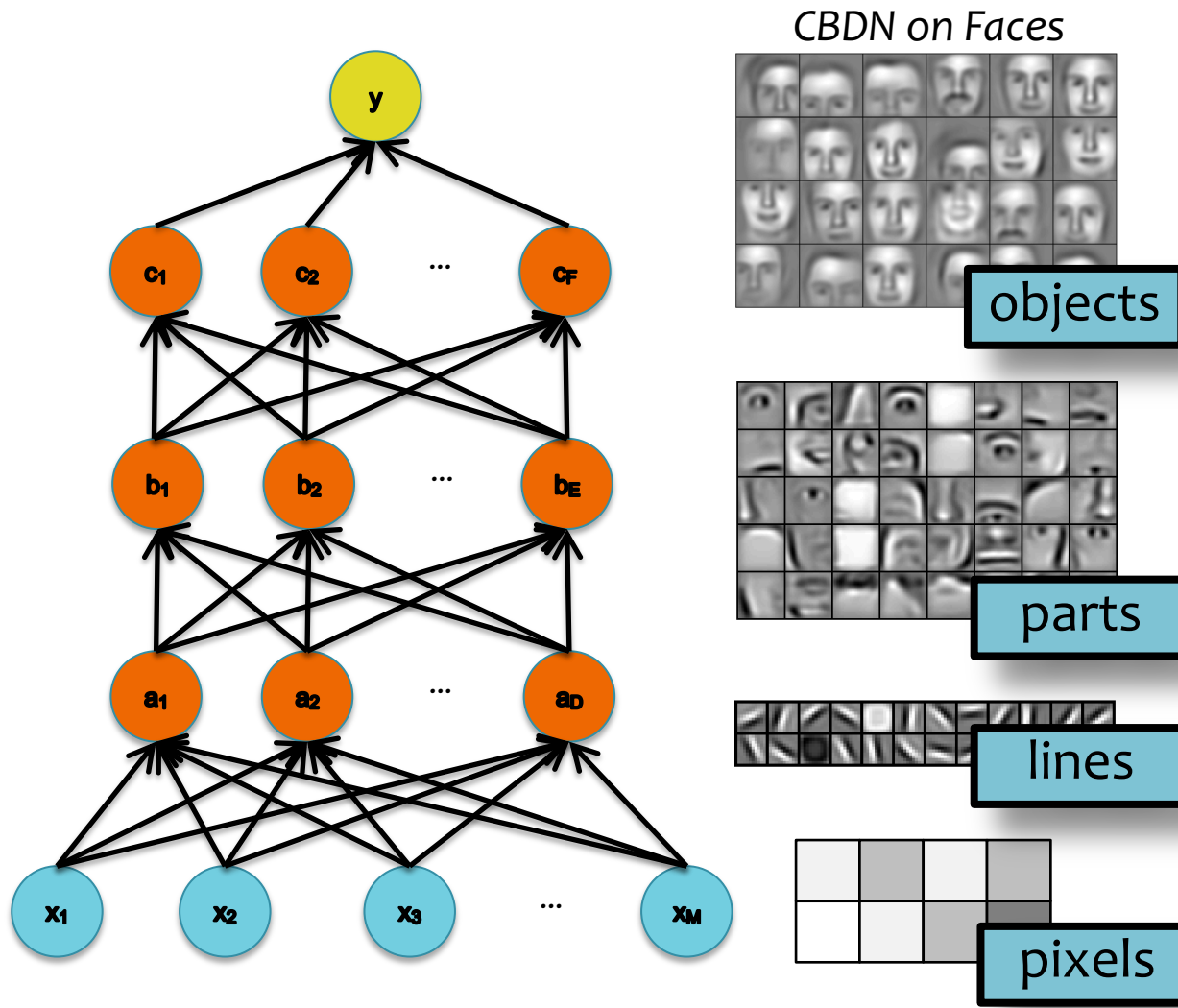
Big caveat: You need to know and use the right tricks.

# Feature Learning



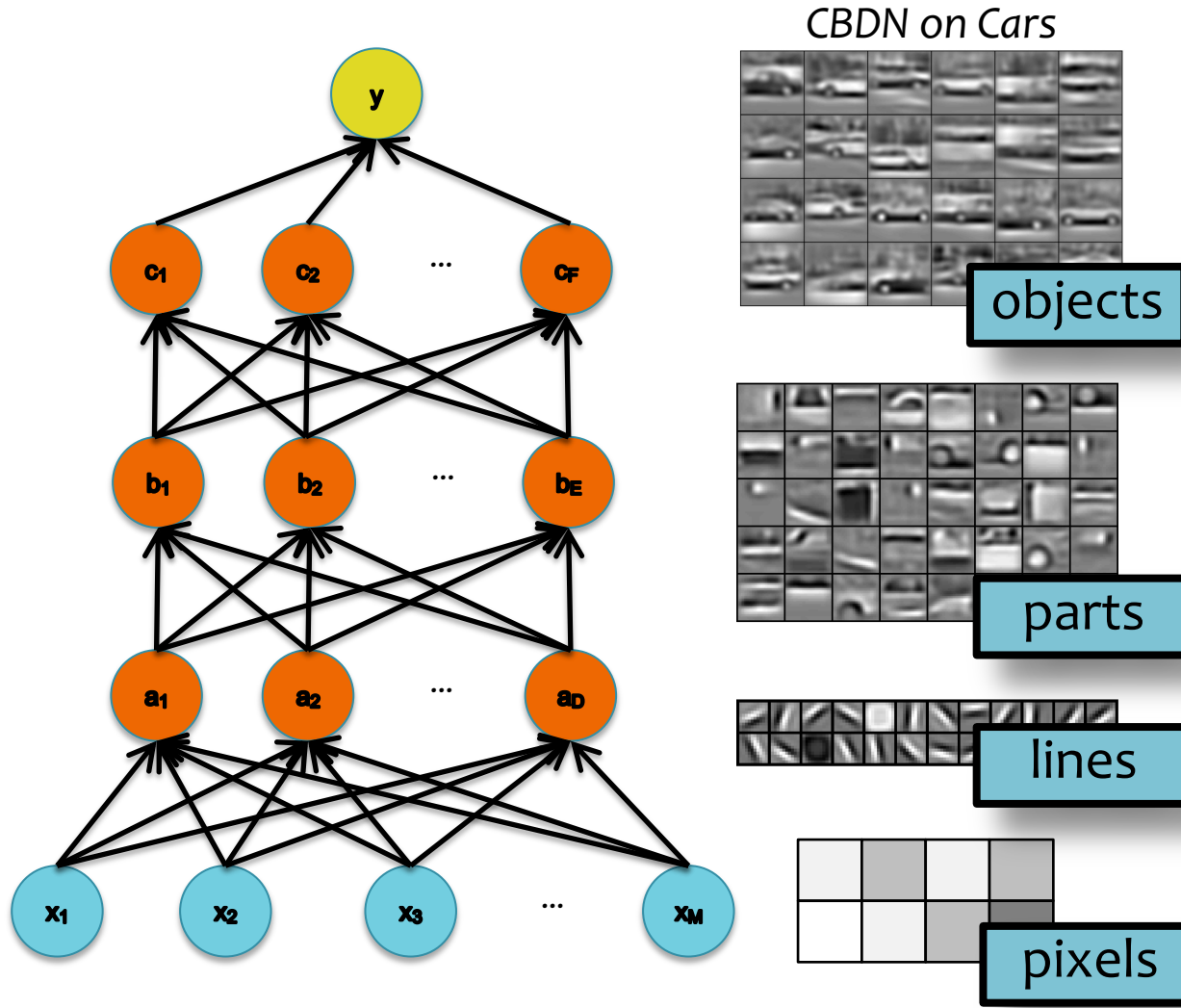
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

# Feature Learning



- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

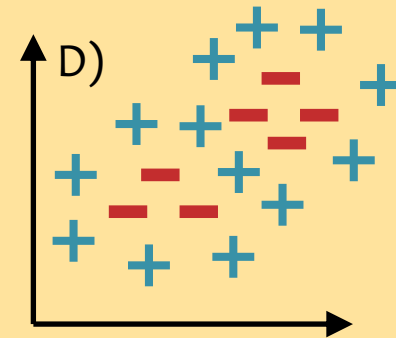
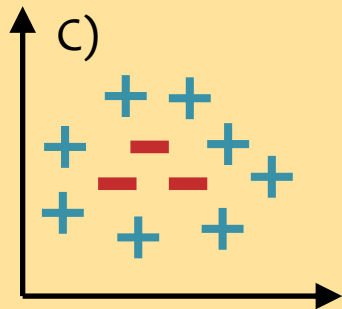
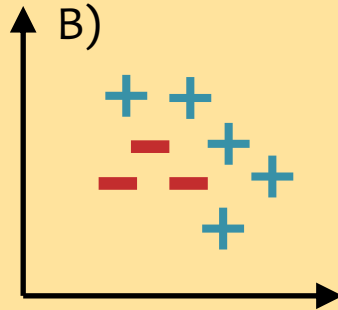
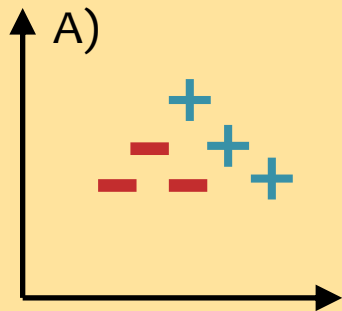
# Feature Learning



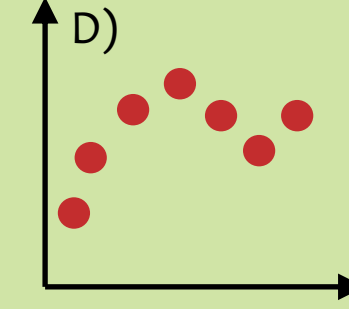
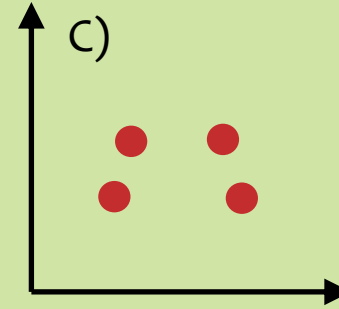
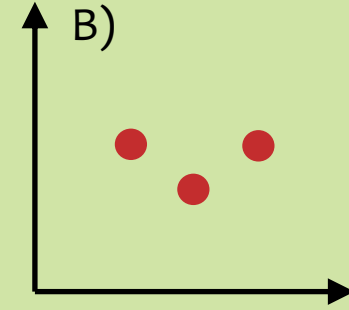
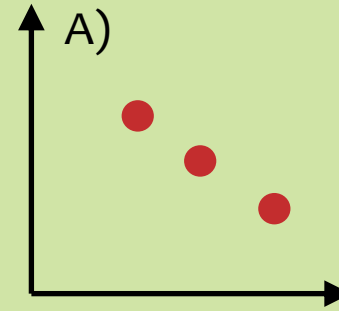
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

# Neural Network Errors

**Question X:** For which of the datasets below does there exist a one-hidden layer neural network that achieves zero *classification* error? **Select all that apply.**



**Question Y:** For which of the datasets below does there exist a one-hidden layer neural network for *regression* that achieves *nearly* zero MSE? **Select all that apply.**





# Neural Network Architectures

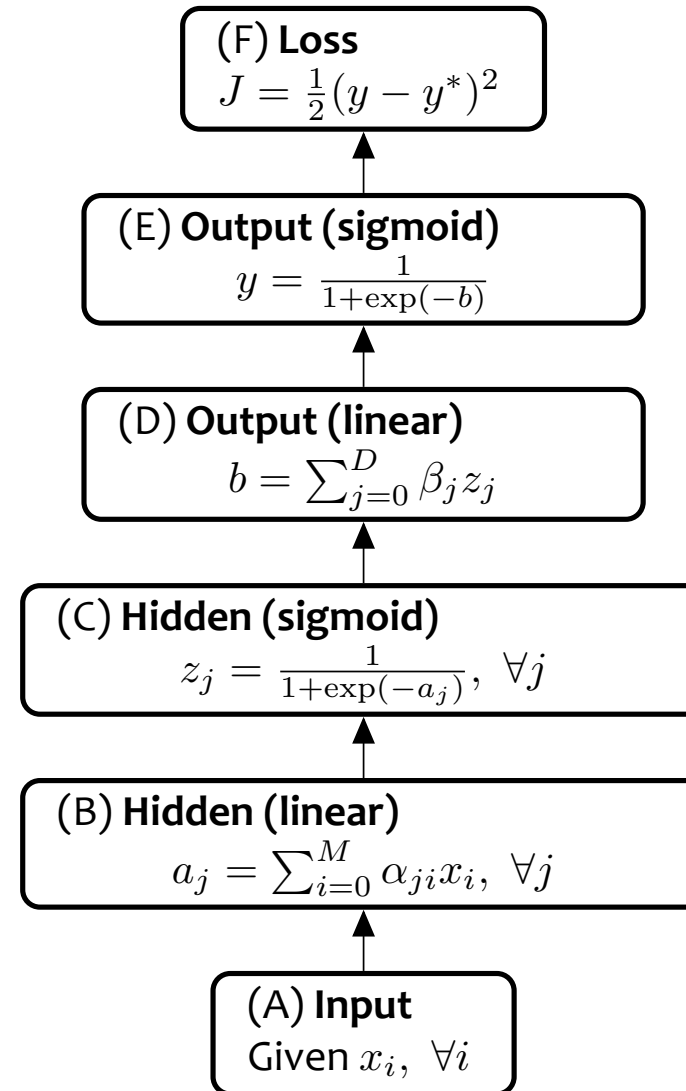
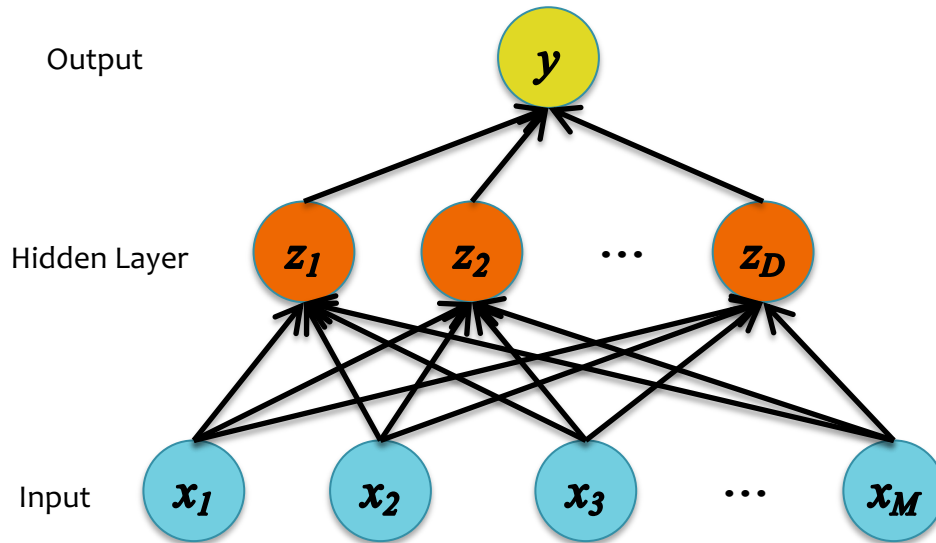
Even for a basic Neural Network, there are many design decisions to make:

1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters

# **ACTIVATION FUNCTIONS**

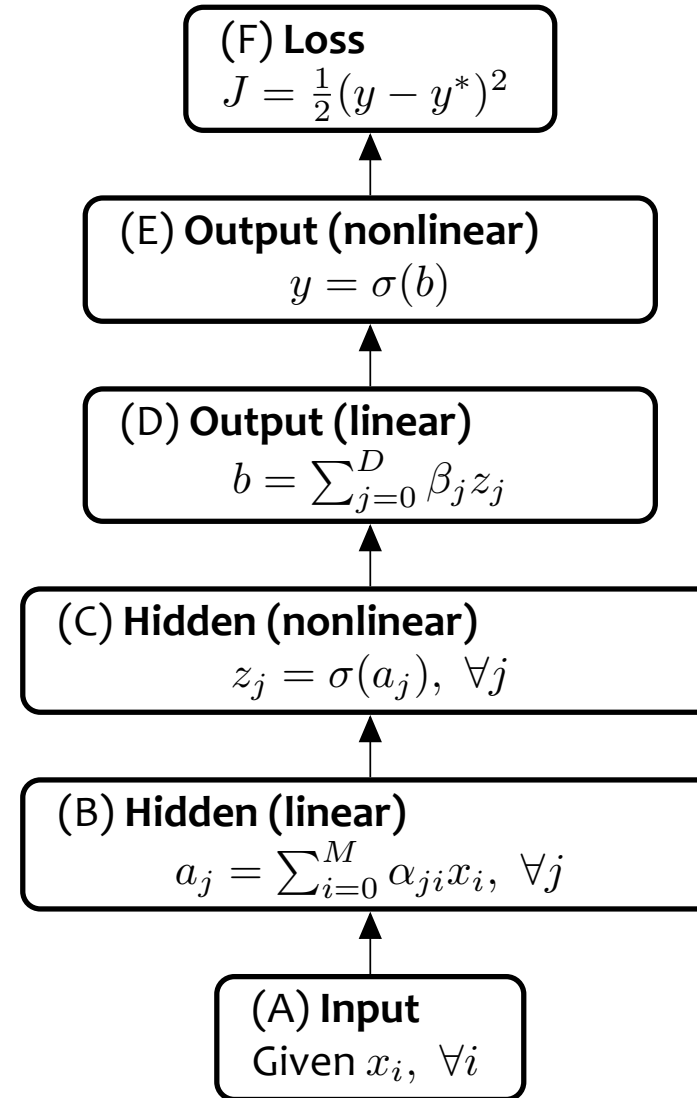
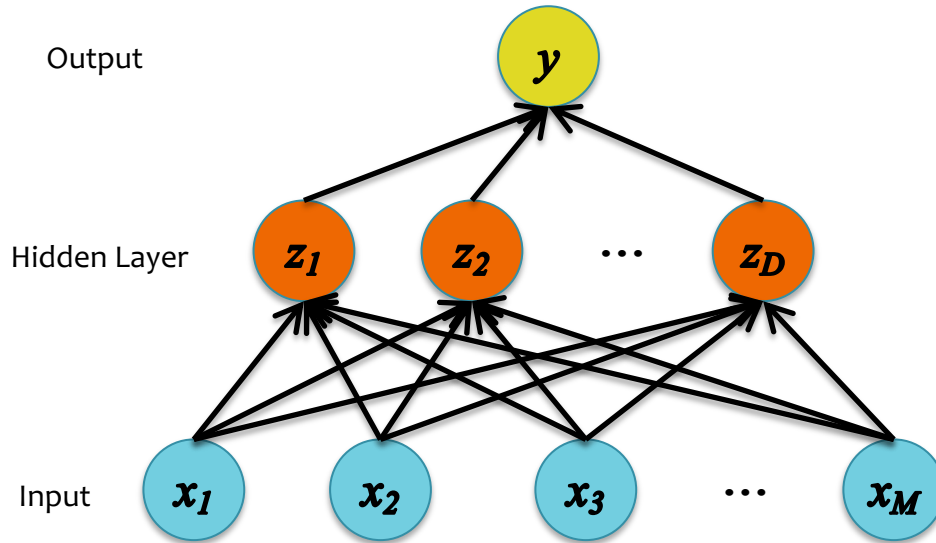
# Activation Functions

Neural Network with sigmoid  
activation functions



# Activation Functions

Neural Network with arbitrary nonlinear activation functions

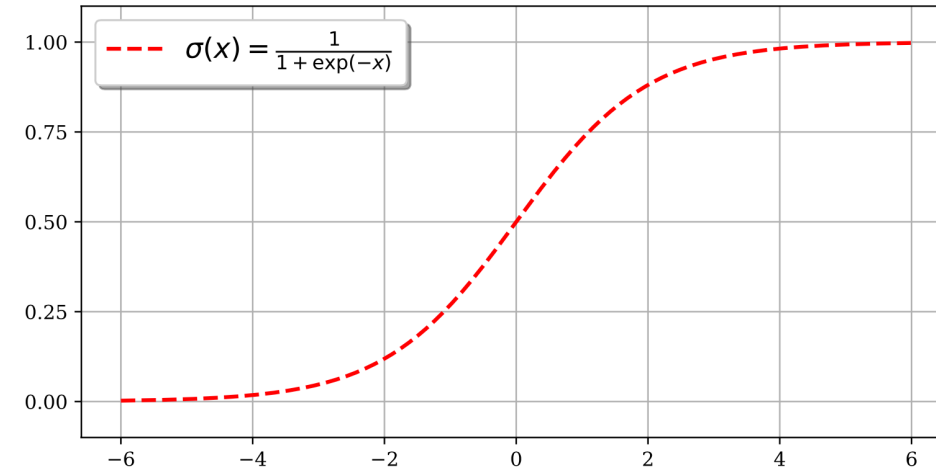


# Activation Functions

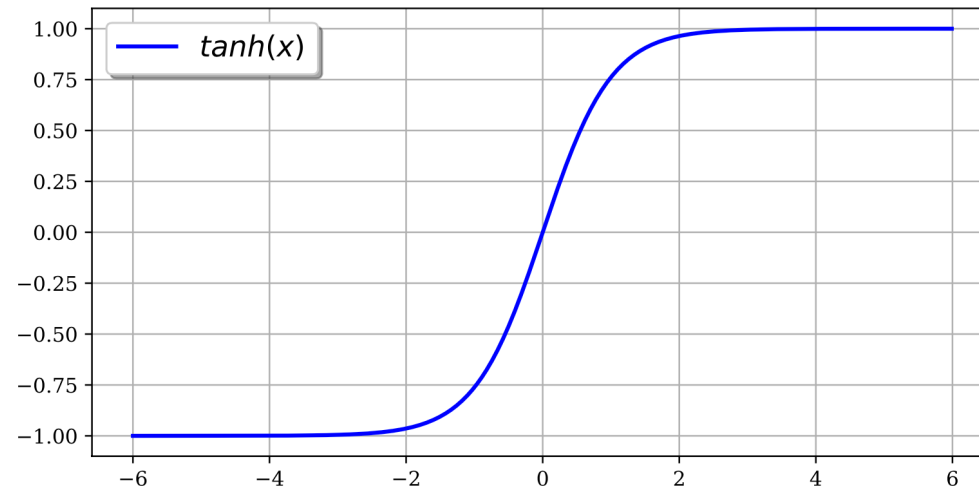
So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function...

...but the sigmoid is not widely used in modern neural networks

**Sigmoid (aka. logistic) function**



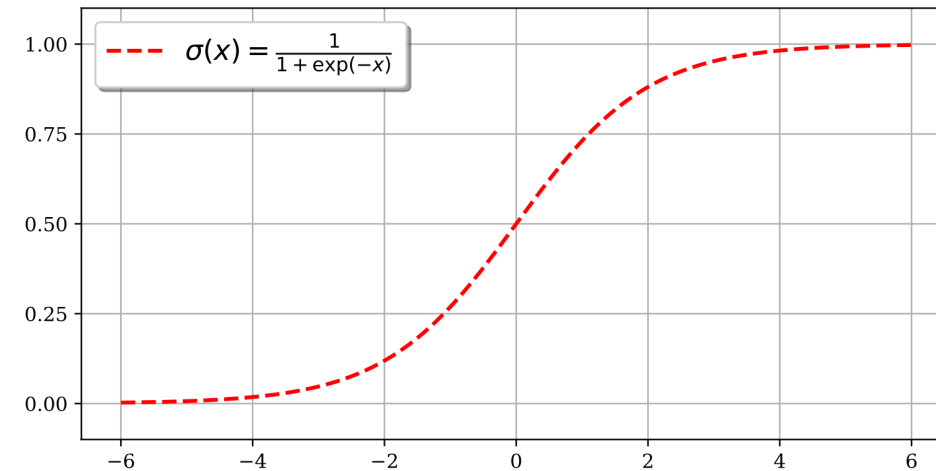
**Hyperbolic tangent function**



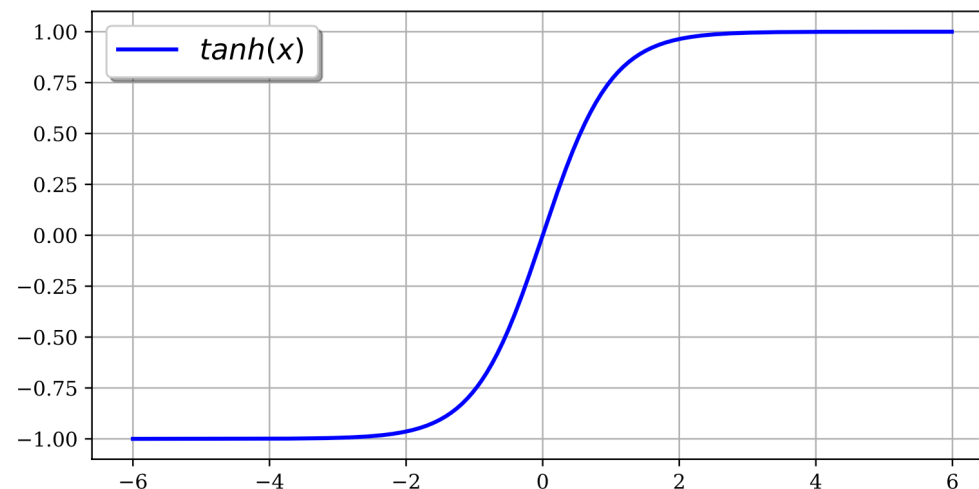
# Activation Functions

- sigmoid,  $\sigma(x)$ 
  - output in range (0,1)
  - good for probabilistic outputs
- hyperbolic tangent,  $\tanh(x)$ 
  - similar shape to sigmoid, but output in range (-1,+1)

Sigmoid (aka. logistic) function



Hyperbolic tangent function



## Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010

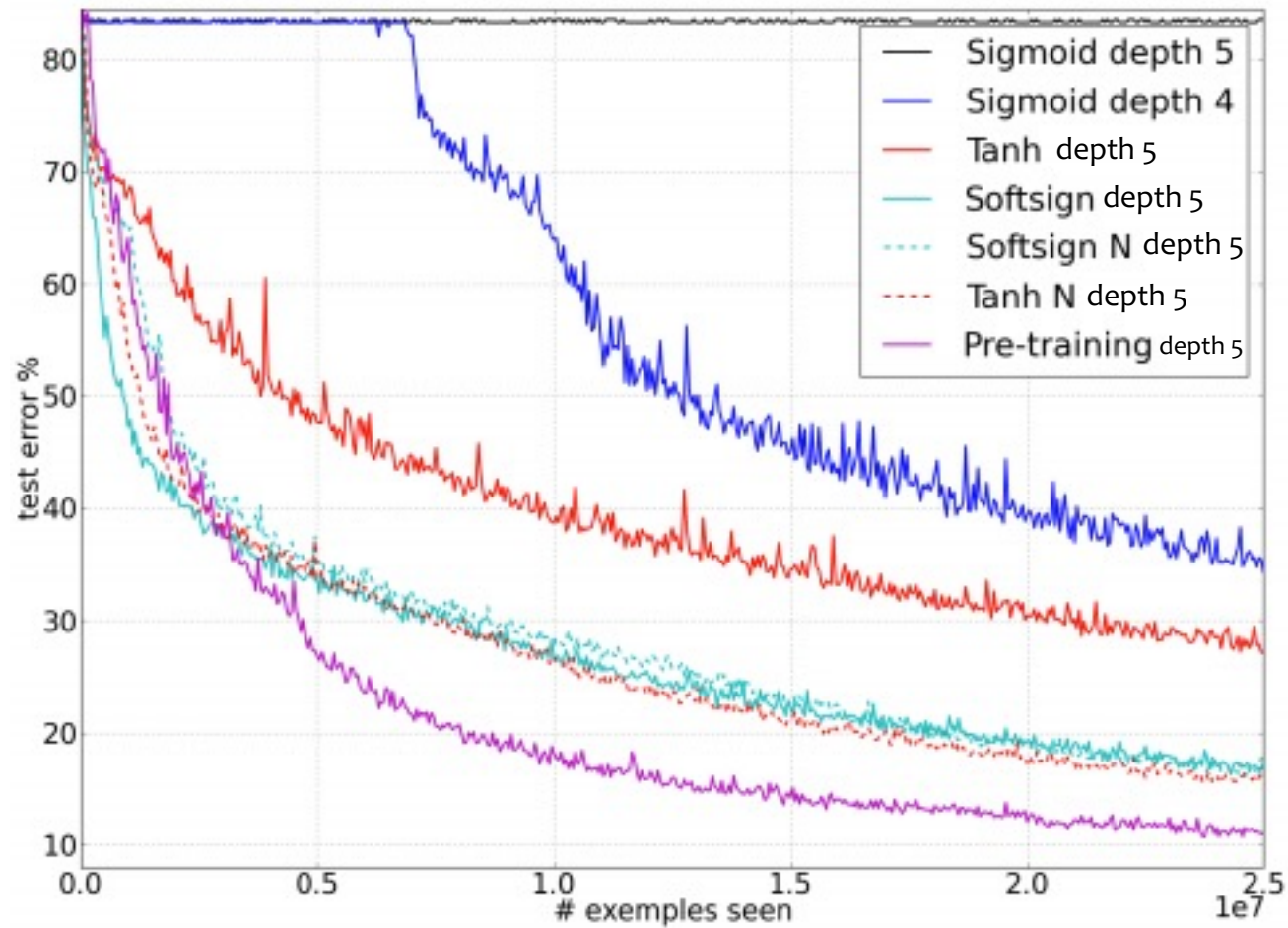
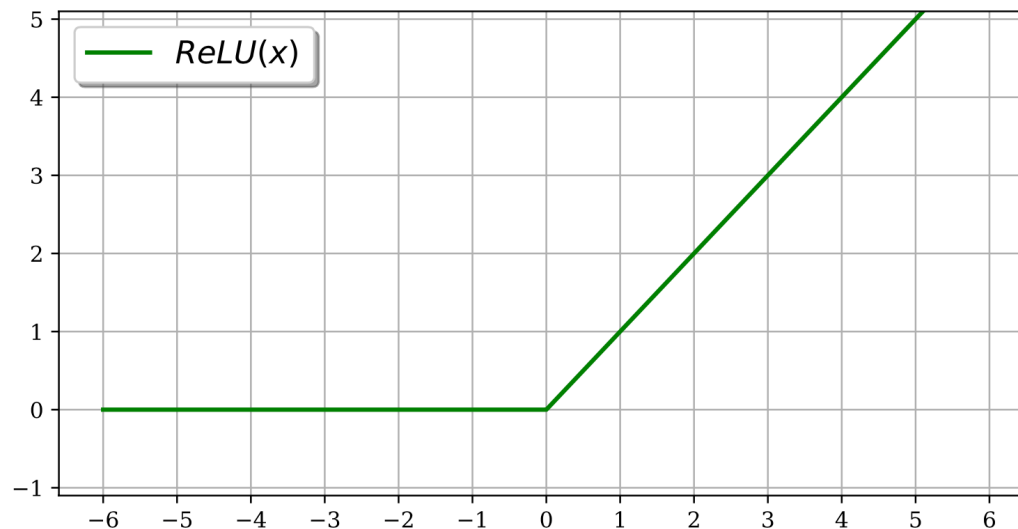


Figure from Glorot & Bentio (2010)

# Activation Functions

- Rectified Linear Unit (ReLU)
  - avoids the vanishing gradient problem
  - derivative is fast to compute

$$\text{ReLU}(x) = \max(0, x)$$





# Activation Functions

- Rectified Linear Unit (ReLU)

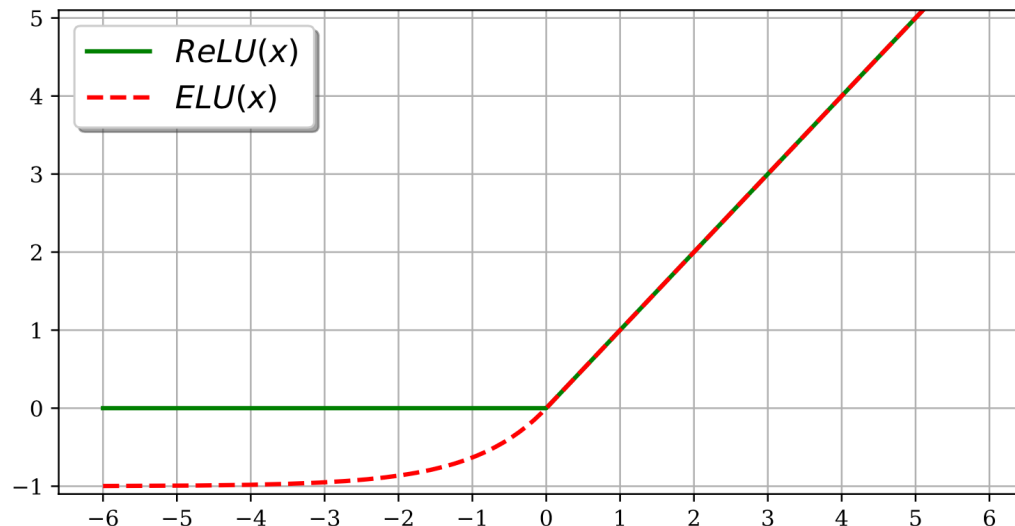
- avoids the vanishing gradient problem
- derivative is fast to compute

$$\text{ReLU}(x) = \max(0, x)$$

- Exponential Linear Unit (ELU)

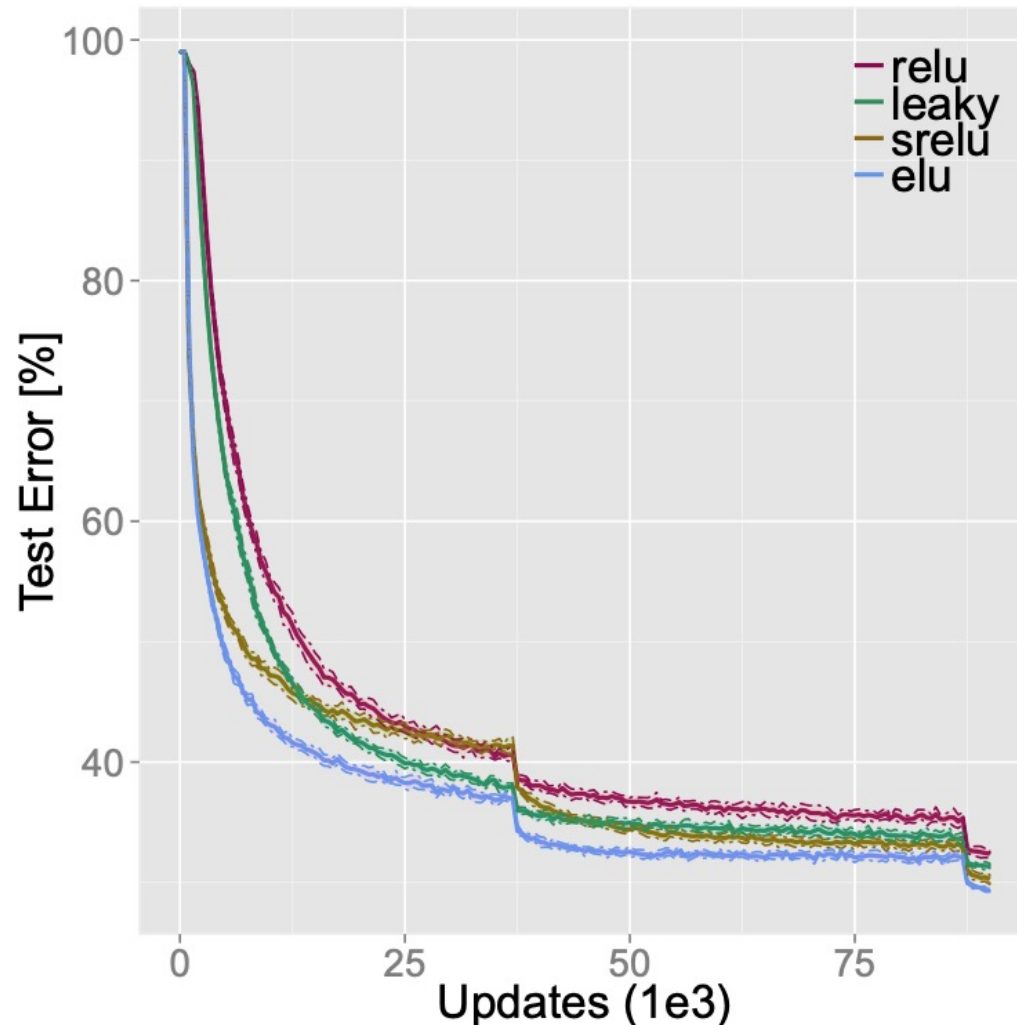
- same as ReLU on positive inputs
- unlike ReLU, allows negative outputs and smoothly transitions for  $x < 0$

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(\exp(x) - 1), & \text{if } x \leq 0 \end{cases}$$



# Activation Functions

Image Classification Benchmark (CIFAR-10)



1. Training loss converges fastest with ELU
2. ELU(x) yields lower test error than ReLU(x) on CIFAR-10

# Neural Networks Objectives

*You should be able to...*

- Explain the biological motivations for a neural network
- Combine simpler models (e.g. linear regression, binary logistic regression, multinomial logistic regression) as components to build up feed-forward neural network architectures
- Explain the reasons why a neural network can model nonlinear decision boundaries for classification
- Compare and contrast feature engineering with learning features
- Identify (some of) the options available when designing the architecture of a neural network
- Implement a feed-forward neural network