10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Regularization + Neural Networks

Matt Gormley
Lecture 11
Feb. 22, 2023

Reminders

- Homework 4: Logistic Regression
  - Out: Fri, Feb 17
  - Due: Sun, Feb. 26 at 11:59pm
- Lecture on Friday

REGULARIZATION

Overfitting

Definition: The problem of **overfitting** is when the model captures the noise in the training data instead of the underlying structure

Overfitting can occur in all the models we've seen so far:

- Decision Trees (e.g. when tree is too deep)
- KNN (e.g. when k is small)
- Perceptron (e.g. when sample isn't representative)
- Linear Regression (e.g. with nonlinear features)
- Logistic Regression (e.g. with many rare features)

Motivation: Regularization

- **Occam's Razor:** prefer the simplest hypothesis

- What does it mean for a hypothesis (or model) to be **simple**?
  1. small number of features (**model selection**)
  2. small number of "important" features (**shrinkage**)

Regularization

- **Given** objective function: $J(\theta)$
- **Goal** is to find: $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \, J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$
- **Key idea:** Define regularizer $r(\theta)$ s.t. we tradeoff between fitting the data and keeping the model simple
- **Choose form of $r(\theta)$:**
  - Example: q-norm (usually p-norm): $||\boldsymbol{\theta}||_q = \left(\sum_{m=1}^{M} |\theta_m|^q \right)^{\frac{1}{q}}$

q	$r(\boldsymbol{\theta})$	**yields parameters that are...**	**name**	**optimization notes**
0	$\|\|\boldsymbol{\theta}\|\|_0 = \sum \mathbb{1}(\theta_m \neq 0)$	zero values	L0 reg.	no good computational solutions
1	$\|\|\boldsymbol{\theta}\|\|_1 = \sum \|\theta_m\|$	zero values	L1 reg.	subdifferentiable
2	$(\|\|\boldsymbol{\theta}\|\|_2)^2 = \sum \theta_m^2$	small values	L2 reg.	differentiable

Regularization Examples

Add an **L2 regularizer** to Linear Regression (aka. Ridge Regression)

$$J_{\text{RR}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_2^2}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{m=1}^{M} \theta_m^2}$$

Add an **L1 regularizer** to Linear Regression (aka. LASSO)

$$J_{\text{LASSO}}(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_1}$$

$$= \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{m=1}^{M} |\theta_m|}$$

Regularization Examples

Add an **L2 regularizer** to Logistic Regression

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_2^2}$$

$$= \frac{1}{N} \sum_{i=1}^{N} -\log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}) + \boxed{\lambda \sum_{m=1}^{M} \theta_m^2}$$

Add an **L1 regularizer** to Logistic Regression

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_1}$$

$$= \frac{1}{N} \sum_{i=1}^{N} -\log p(y^{(i)} \mid \mathbf{x}^{(i)}, \boldsymbol{\theta}) + \boxed{\lambda \sum_{m=1}^{M} |\theta_m|}$$

Regularization

Question:

Suppose we are minimizing J'(θ) where

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$$

As λ increases, the minimum of J'(θ) will…

A. …move towards the midpoint between J(θ) and r(θ)

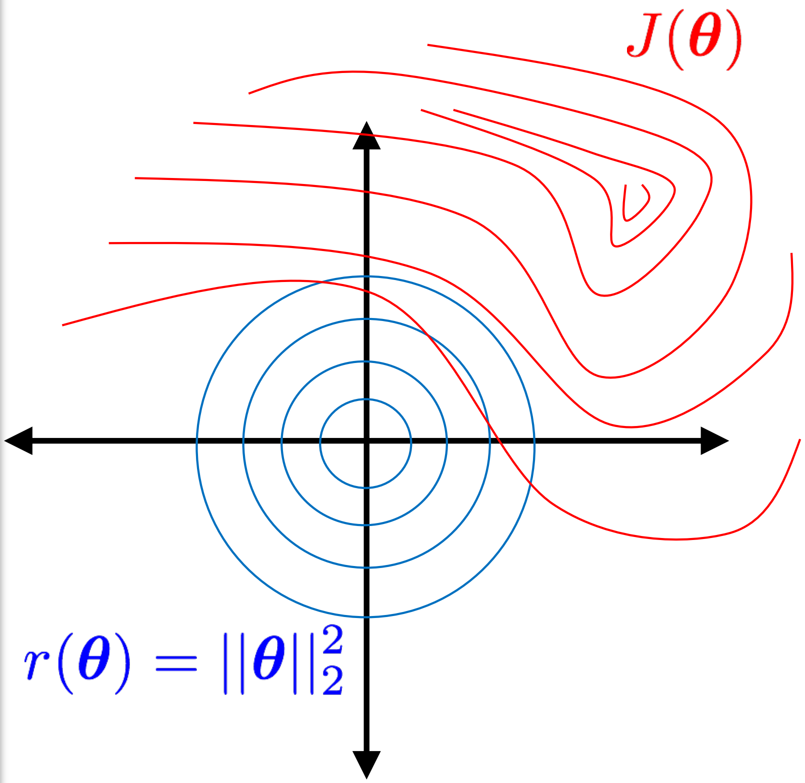
B. …move towards the minimum of J(θ)

C. …move towards the minimum of r(θ)

D. …move towards a theta vector of positive infinities

E. …move towards a theta vector of negative infinities

F. …stay the same



Regularization

Don't Regularize the Bias (Intercept) Parameter!

- In our models so far, the bias / intercept parameter is usually denoted by θ_0 -- that is, the parameter for which we fixed $x_0 = 1$
- Regularizers always avoid penalizing this bias / intercept parameter
- Why? Because otherwise the learning algorithms wouldn't be invariant to a shift in the y-values

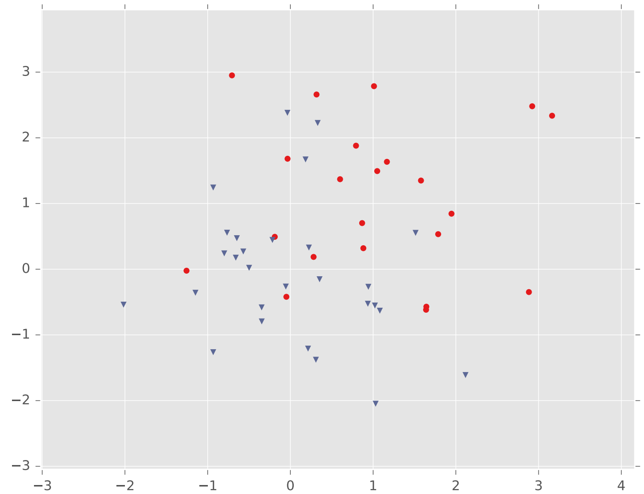
Standardizing Data

- It's common to *standardize* each feature by subtracting its mean and dividing by its standard deviation
- For regularization, this helps all the features be penalized in the same units
(e.g. convert both centimeters and kilometers to z-scores)

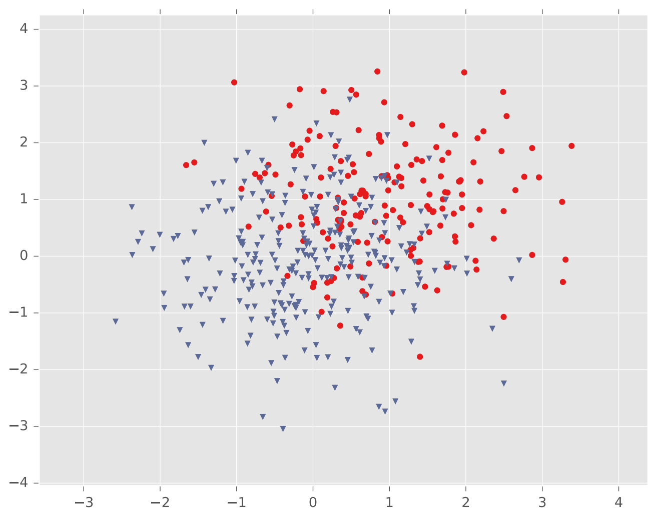
REGULARIZATION EXAMPLE: LOGISTIC REGRESSION

Example: Logistic Regression

Training Data



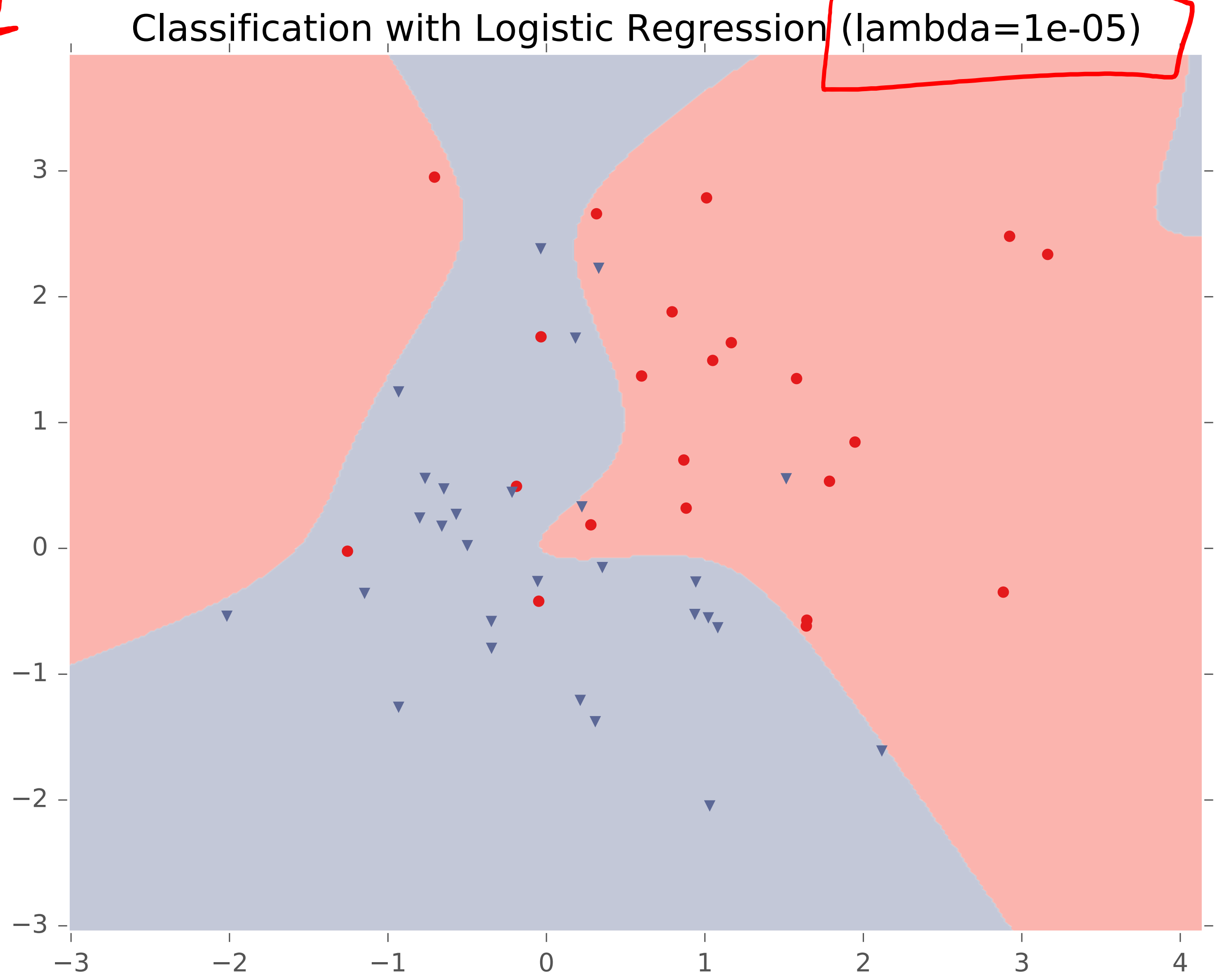
Test Data



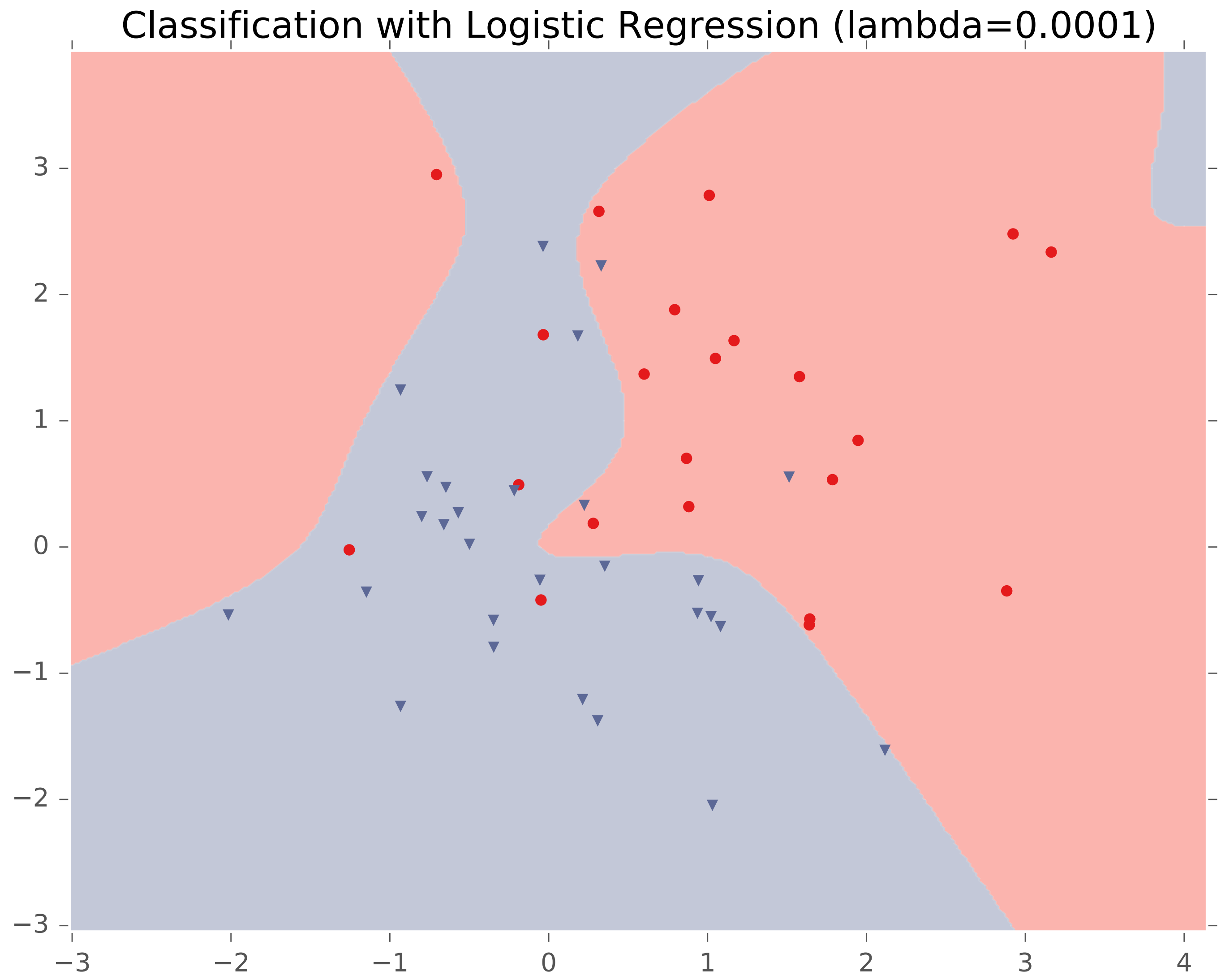
- For this example, we construct **nonlinear features** (i.e. feature engineering)
- Specifically, we add **polynomials up to order 9** of the two original features x_1 and x_2
- Thus our classifier is **linear** in the **high-dimensional feature space,** but the decision boundary is **nonlinear** when visualized in **low-dimensions** (i.e. the original two dimensions)

Example: Logistic Regression

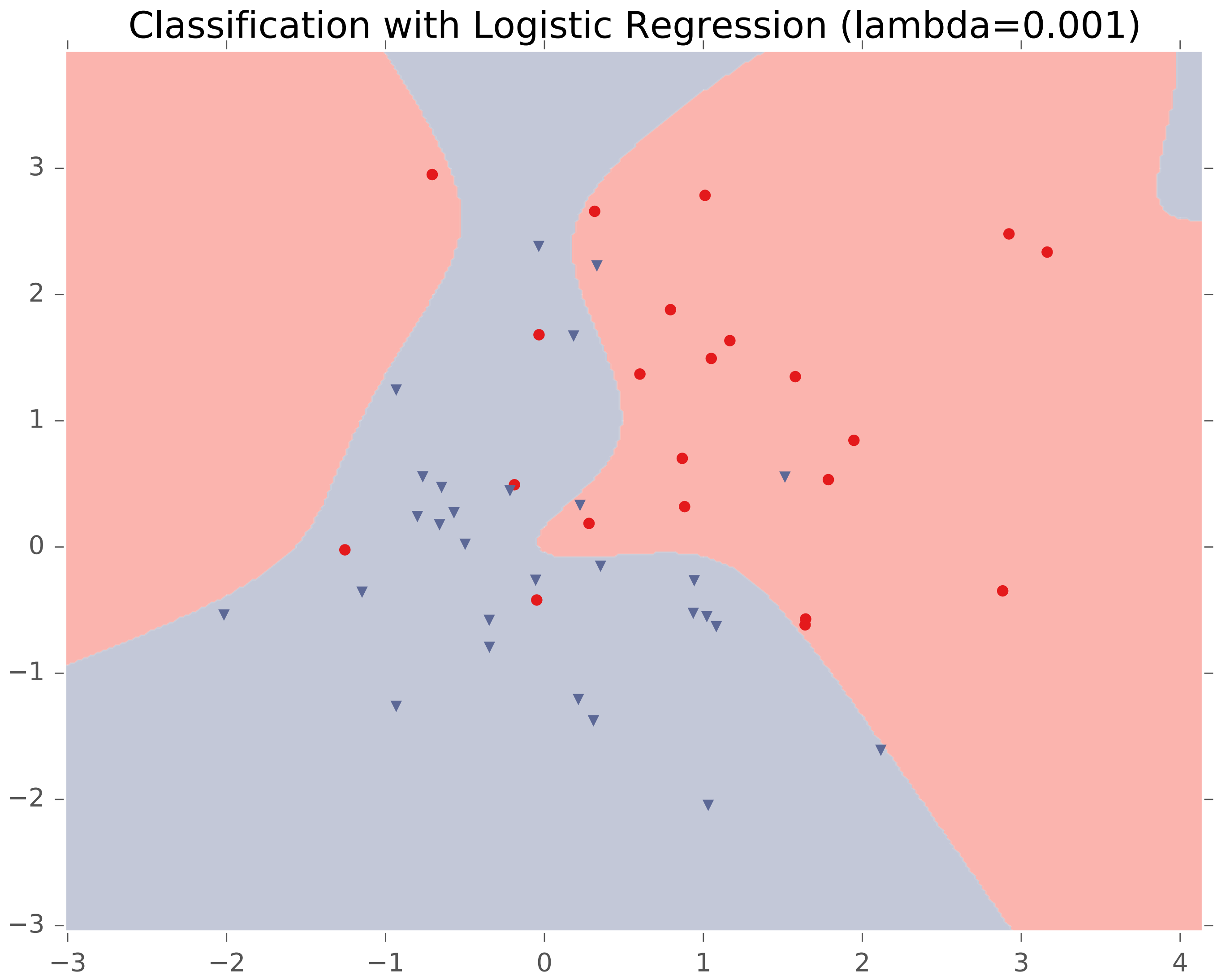
$J(\theta) + \lambda r(\theta)$



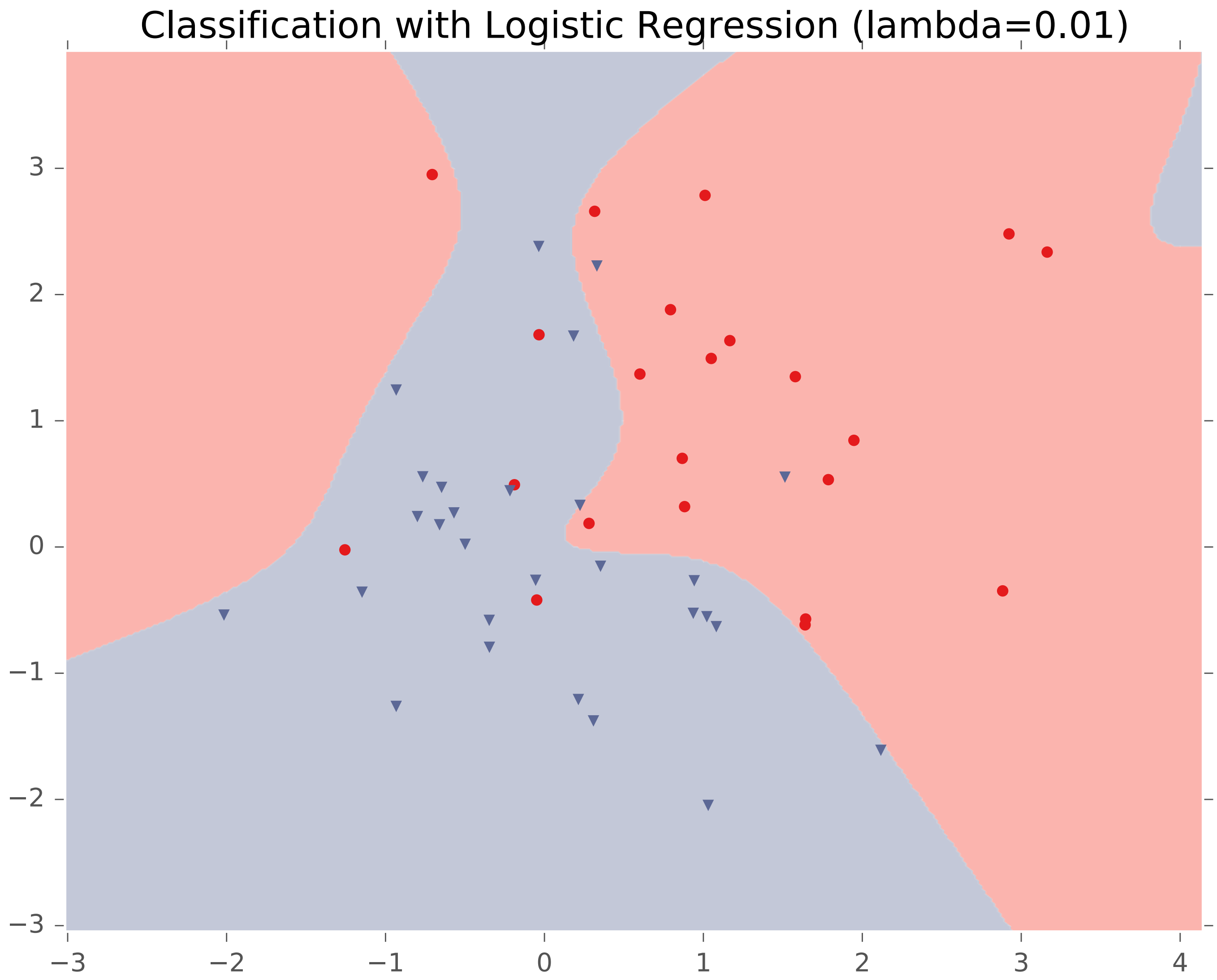
Example: Logistic Regression



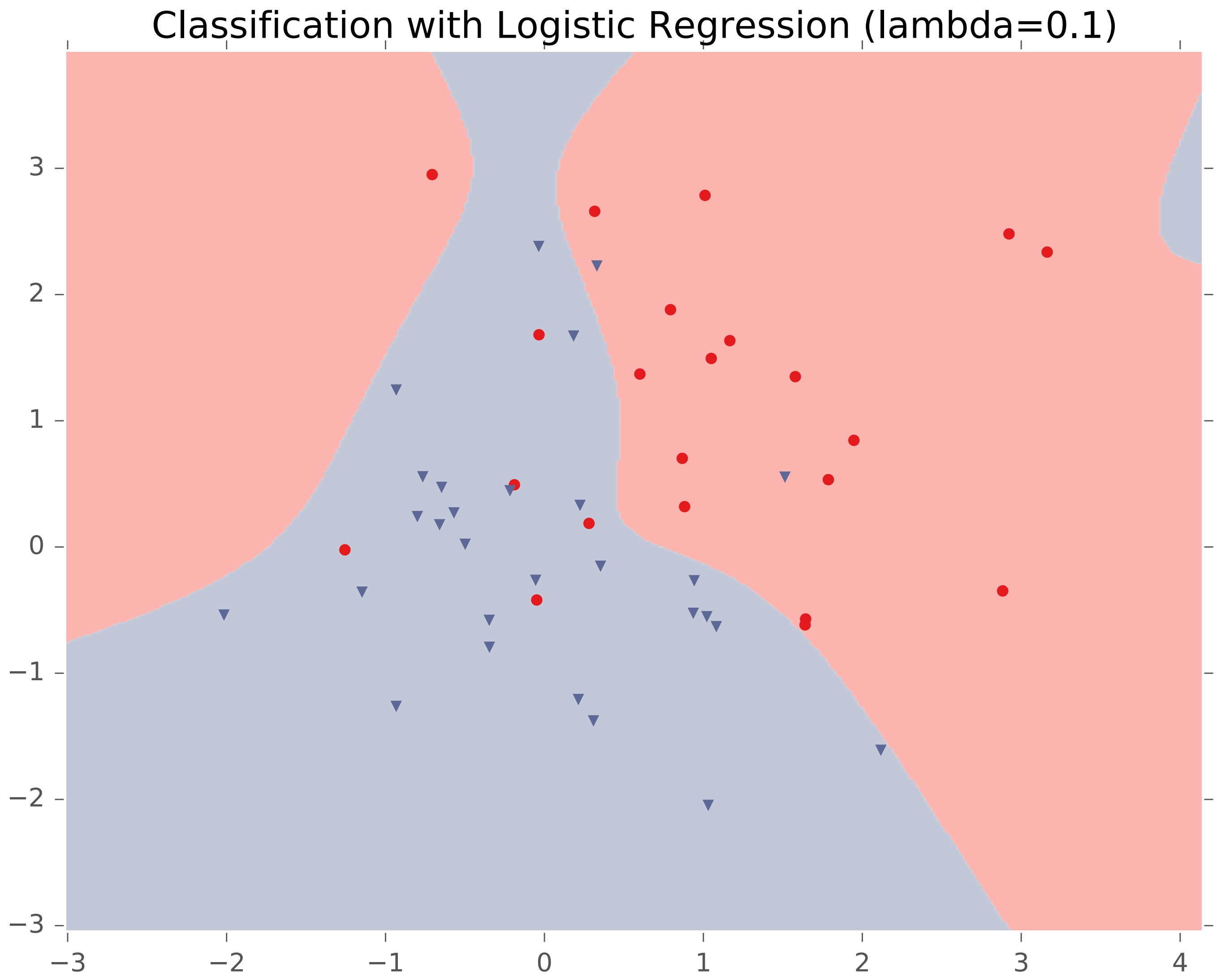
Example: Logistic Regression



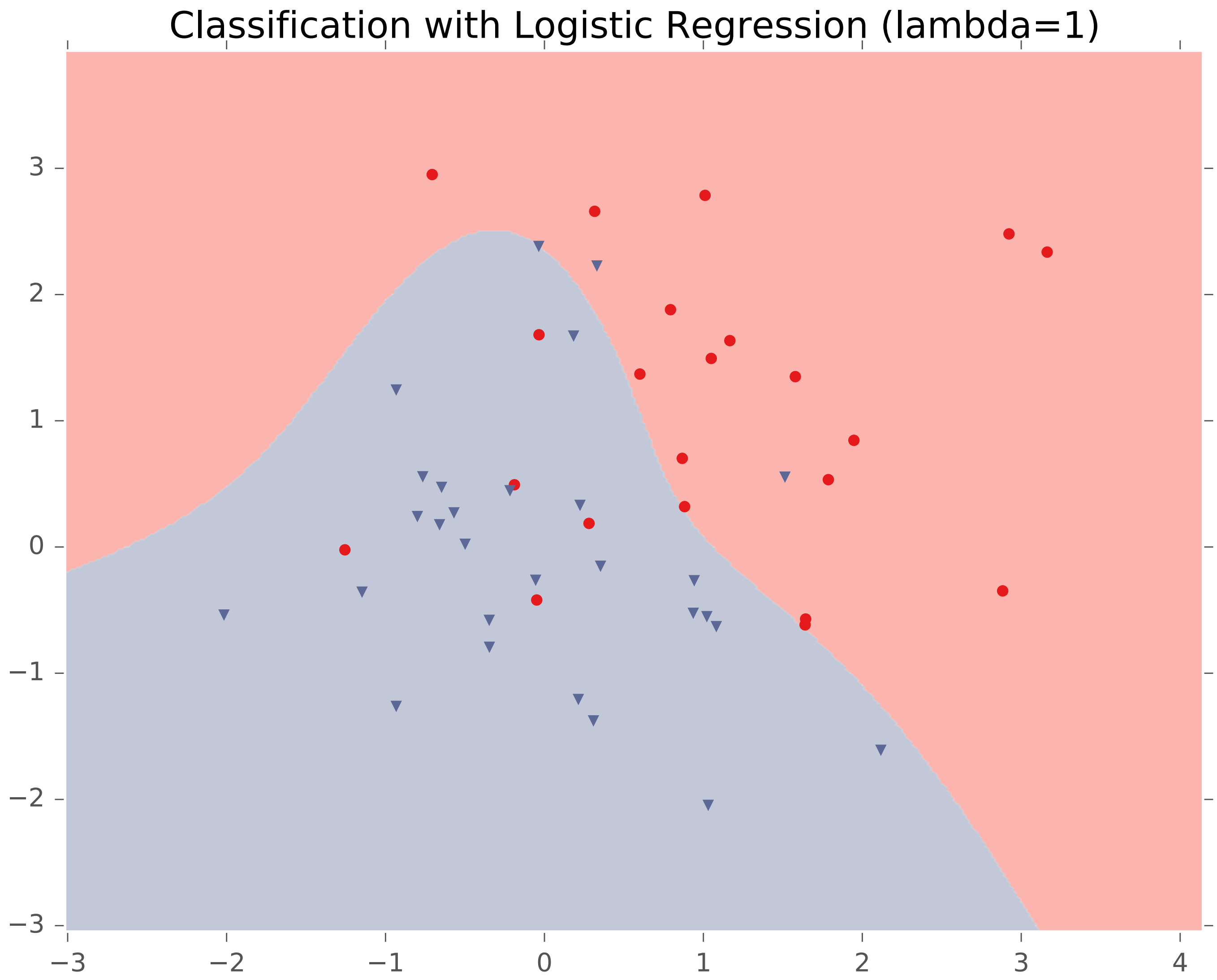
Example: Logistic Regression



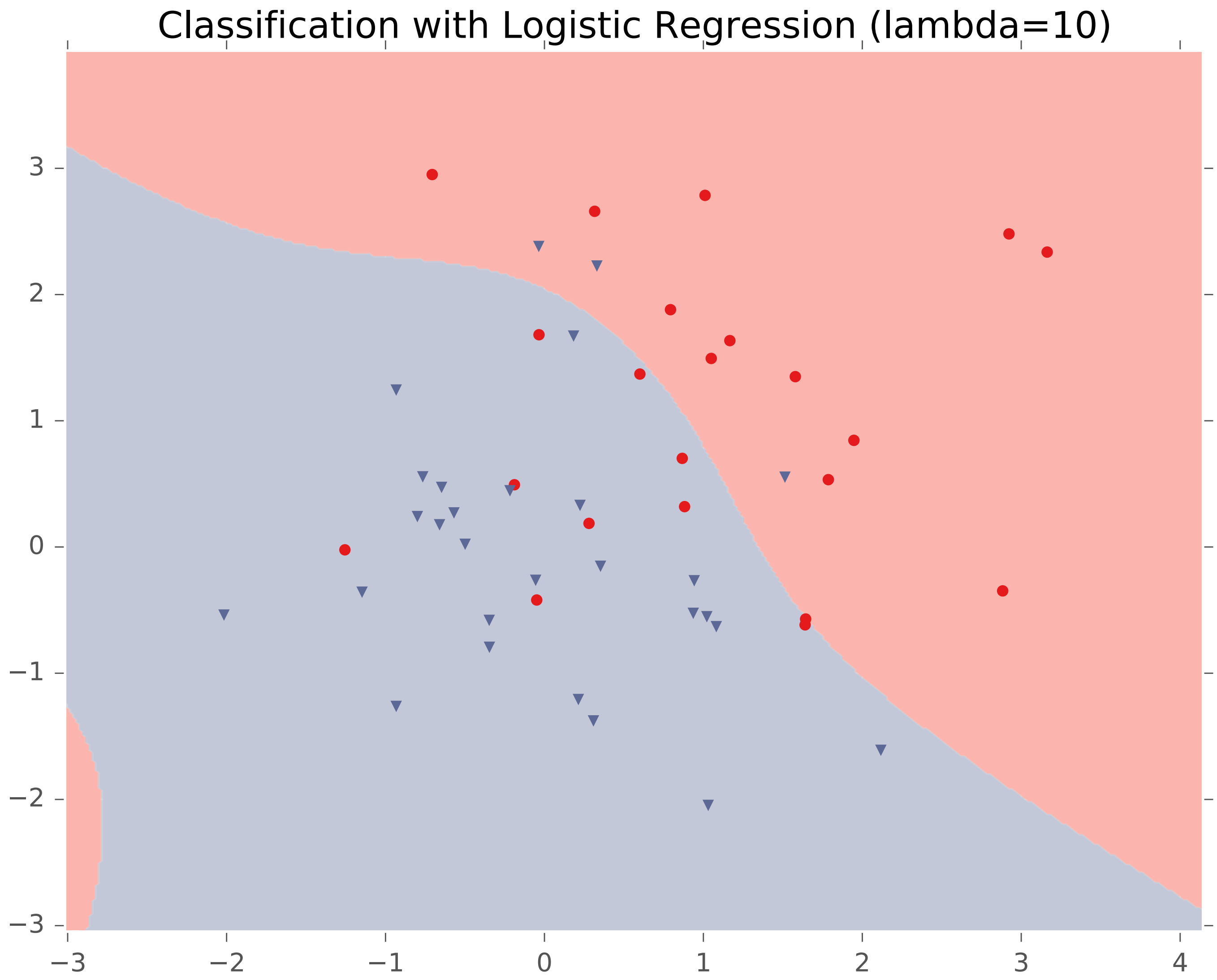
Example: Logistic Regression



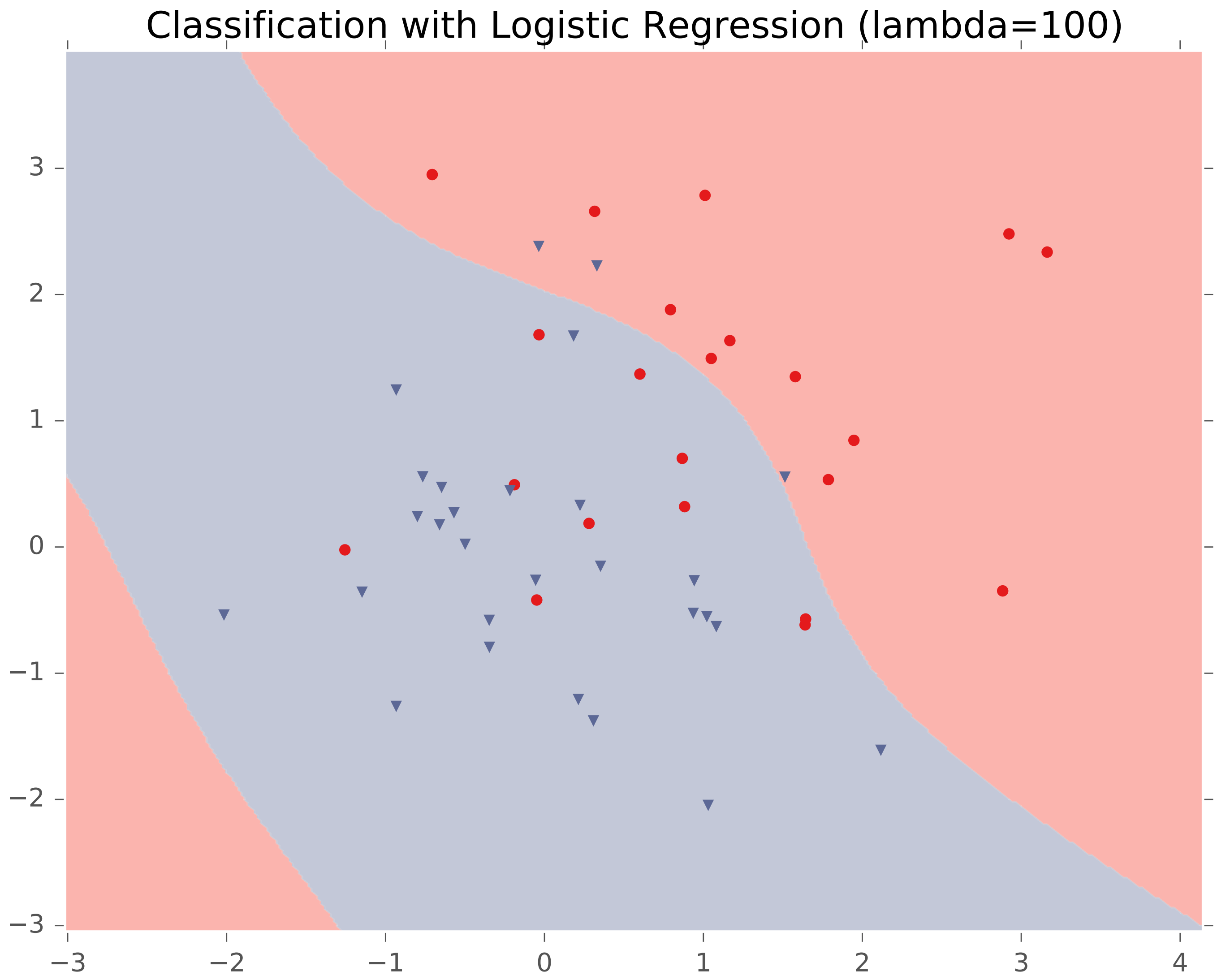
Example: Logistic Regression



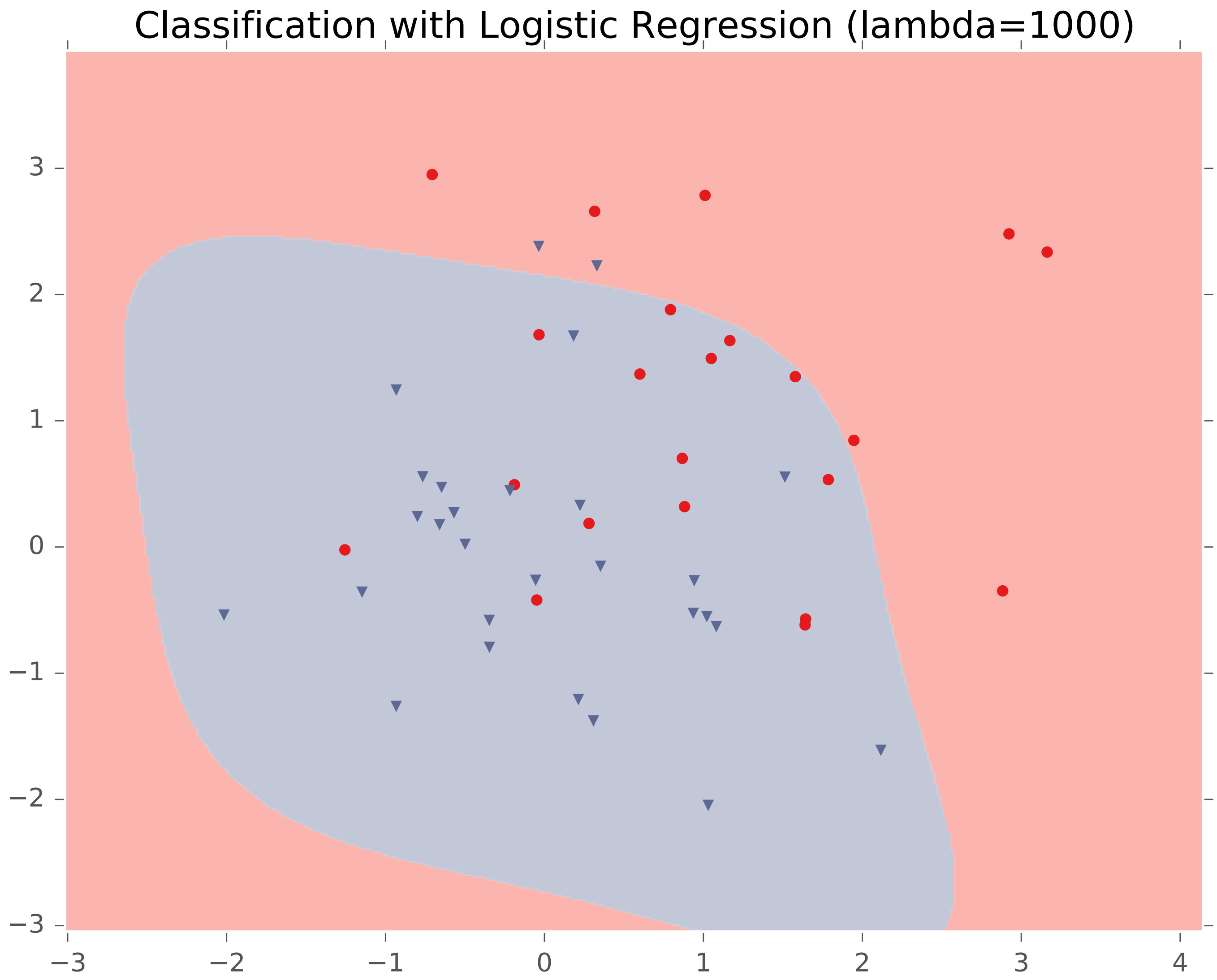
Example: Logistic Regression



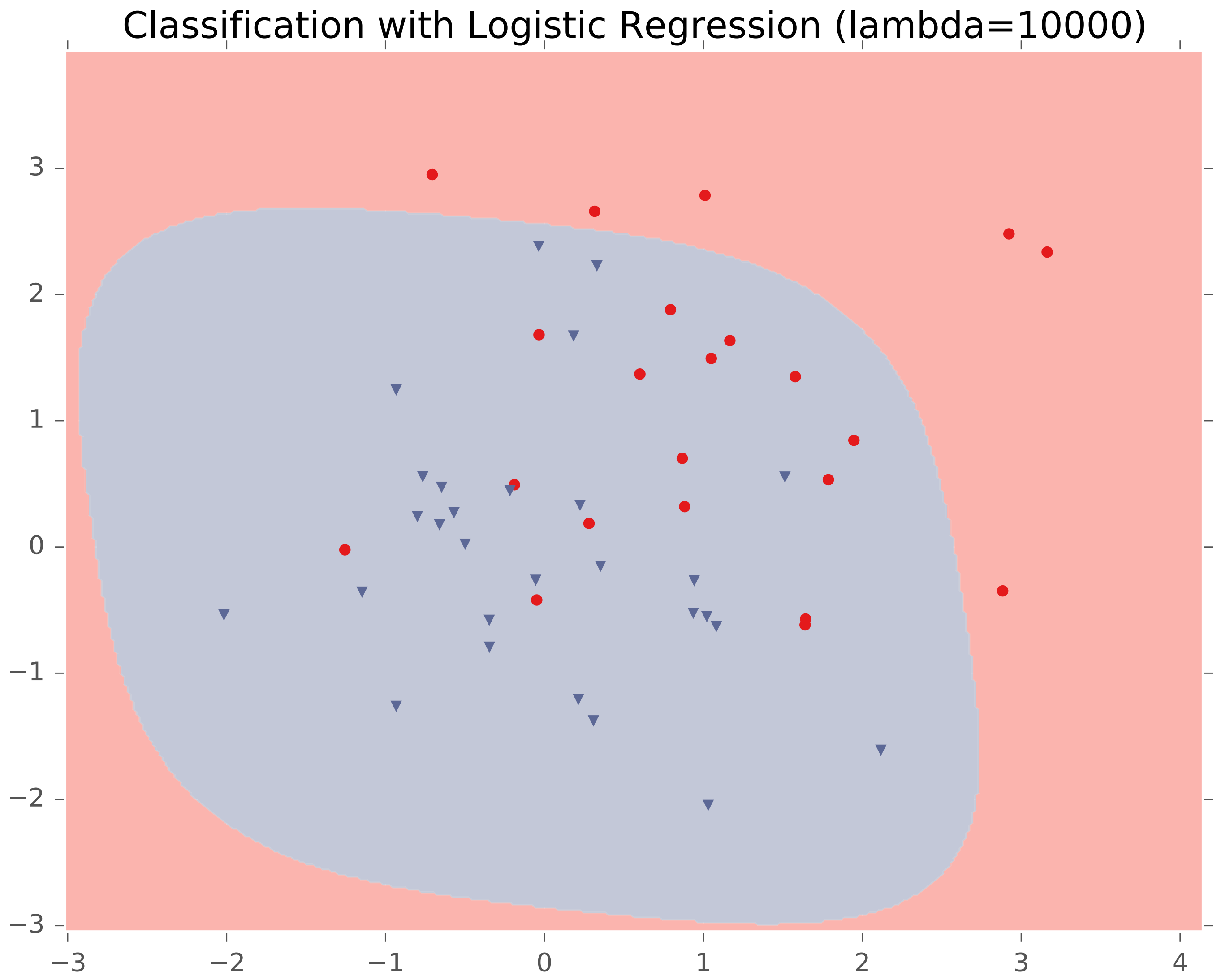
Example: Logistic Regression



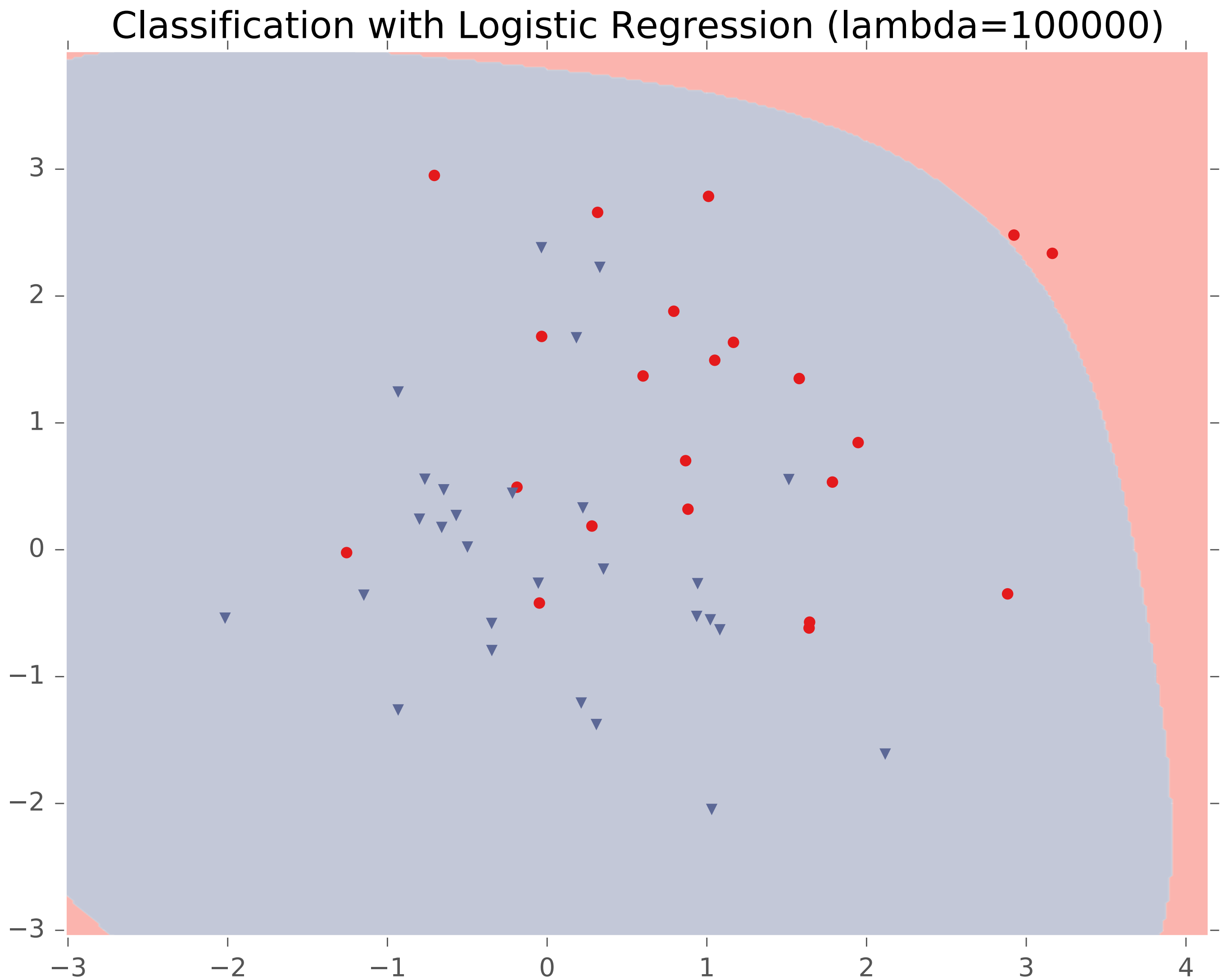
Example: Logistic Regression



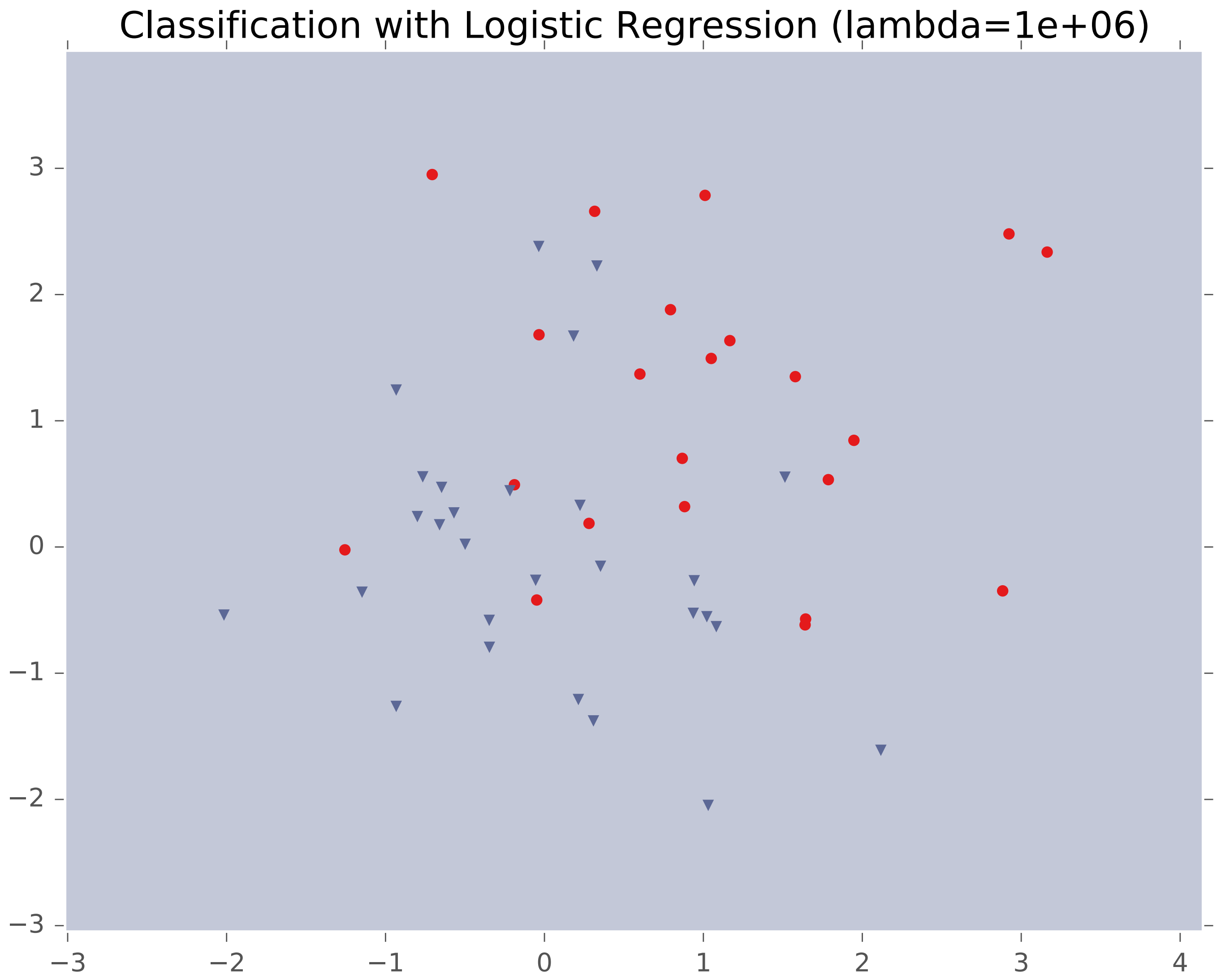
Example: Logistic Regression



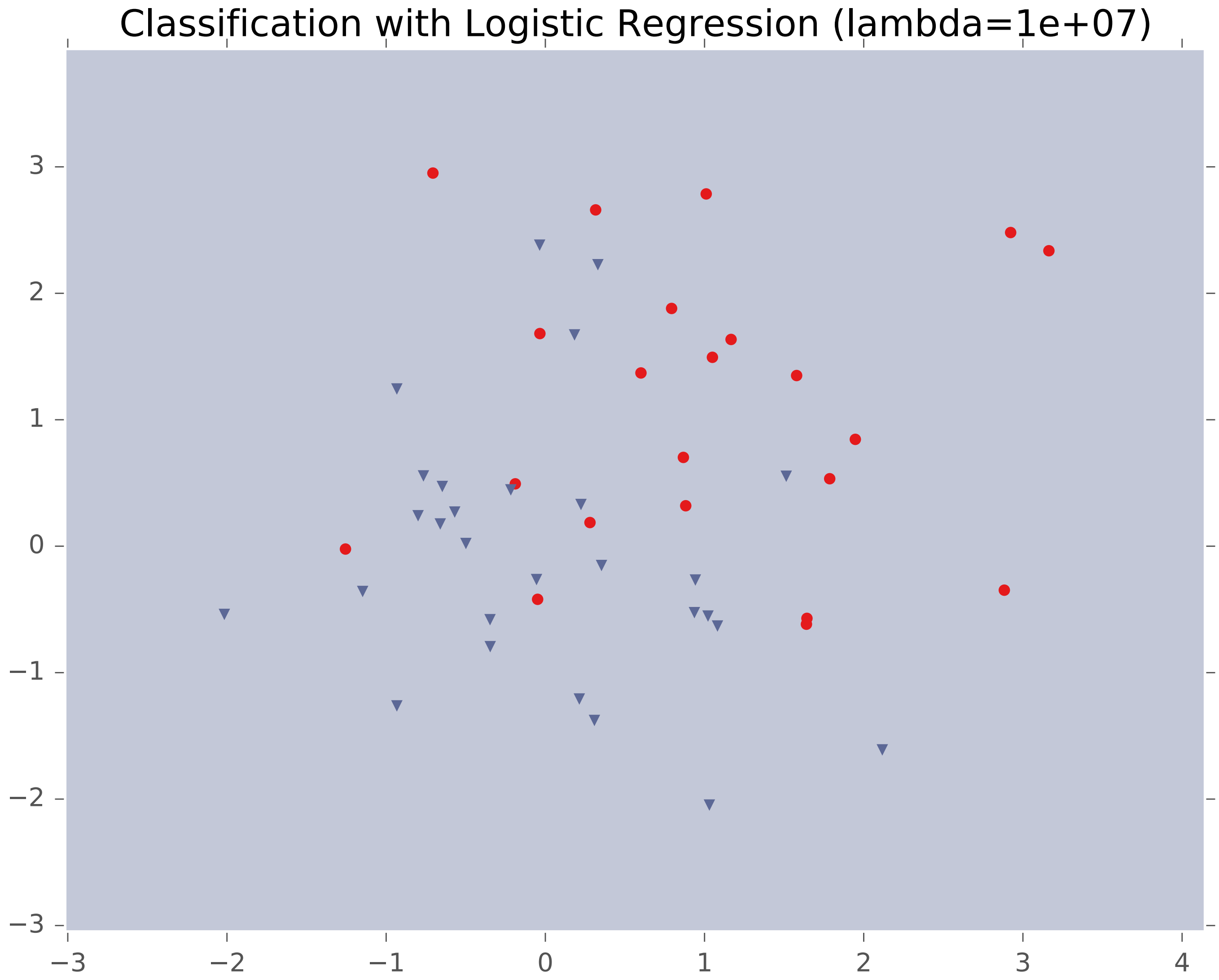
Example: Logistic Regression



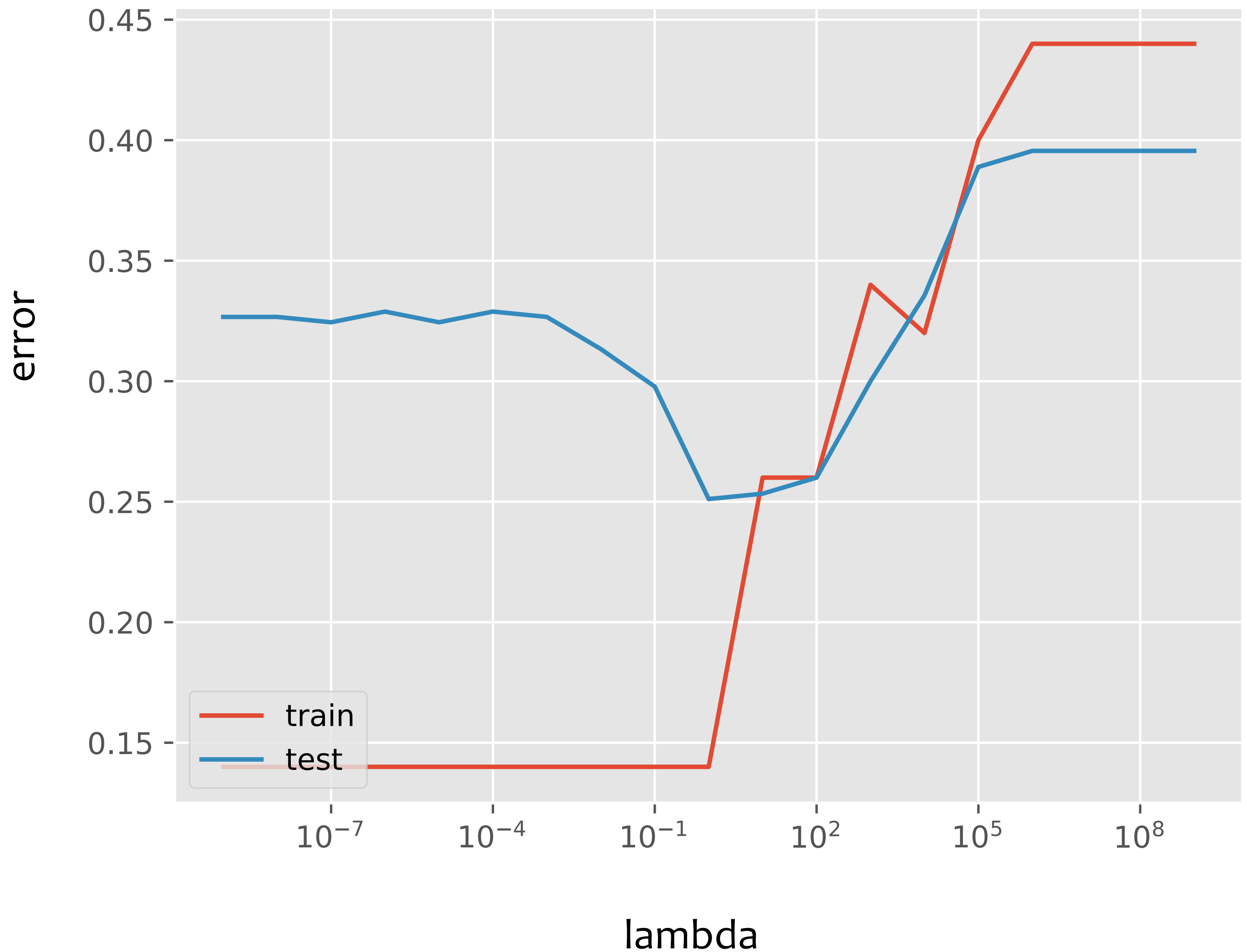
Example: Logistic Regression



Example: Logistic Regression



Example: Logistic Regression



OPTIMIZATION FOR L1 REGULARIZATION

Optimization for L1 Regularization

Can we apply SGD to the LASSO learning problem?

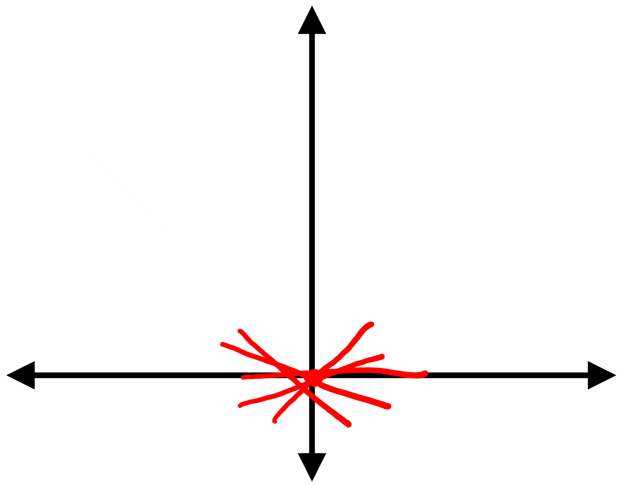
$$\underset{\boldsymbol{\theta}}{\text{argmin}} \;\; J_{\text{LASSO}}(\boldsymbol{\theta})$$

$$\begin{aligned} J_{\text{LASSO}}(\boldsymbol{\theta}) &= J(\boldsymbol{\theta}) + \boxed{\lambda ||\boldsymbol{\theta}||_1} \\ &= \frac{1}{2}\sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})^2 + \boxed{\lambda \sum_{k=1}^{K} |\theta_k|} \end{aligned}$$

Optimization for L1 Regularization

- Consider the absolute value function:

$$r(\boldsymbol{\theta}) = \lambda \sum_{k=1}^{K} |\theta_k|$$



- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)

Def: A vector $g \in \mathbb{R}^M$ is called a **subgradient** of a function $f(\mathbf{x}) : \mathbb{R}^M \to \mathbb{R}$ at the point $\mathbf{x}$ if, for all $\mathbf{x}' \in \mathbb{R}^M$, we have:

$$f(\mathbf{x}') \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{x}' - \mathbf{x})$$

Optimization for L1 Regularization

- The L1 penalty is subdifferentiable (i.e. not differentiable at 0)
- An array of optimization algorithms exist to handle this issue:
  - Subgradient descent
  - Stochastic subgradient descent
  - Coordinate Descent
  - Othant-Wise Limited memory Quasi-Newton (OWL-QN) (Andrew & Gao, 2007) and provably convergent variants
  - Block coordinate Descent (Tseng & Yun, 2009)
  - Sparse Reconstruction by Separable Approximation (SpaRSA) (Wright et al., 2009)
  - Fast Iterative Shrinkage Thresholding Algorithm (FISTA) (Beck & Teboulle, 2009)

Basically the same as GD and SGD, but you use one of the subgradients when necessary

Regularization as MAP

- L1 and L2 regularization can be interpreted as **maximum a-posteriori (MAP) estimation** of the parameters
- To be discussed later in the course…

Takeaways

1. **Nonlinear basis functions** allow **linear models** (e.g. Linear Regression, Logistic Regression) to capture **nonlinear** aspects of the original input
2. Nonlinear features are **require no changes to the model** (i.e. just preprocessing)
3. **Regularization** helps to avoid **overfitting**
4. **Regularization** and **MAP estimation** are equivalent for appropriately chosen priors

Feature Engineering / Regularization Objectives

You should be able to… Q1: What questions do you have?

- Engineer appropriate features for a new task
- Use feature selection techniques to identify and remove irrelevant features
- Identify when a model is overfitting
- Add a regularizer to an existing objective in order to combat overfitting
- Explain why we should **not** regularize the bias term
- Convert linearly inseparable dataset to a linearly separable dataset in higher dimensions
- Describe feature engineering in common application areas

NEURAL NETWORKS

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:

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

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

A Recipe for

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

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

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

A Recipe for

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)

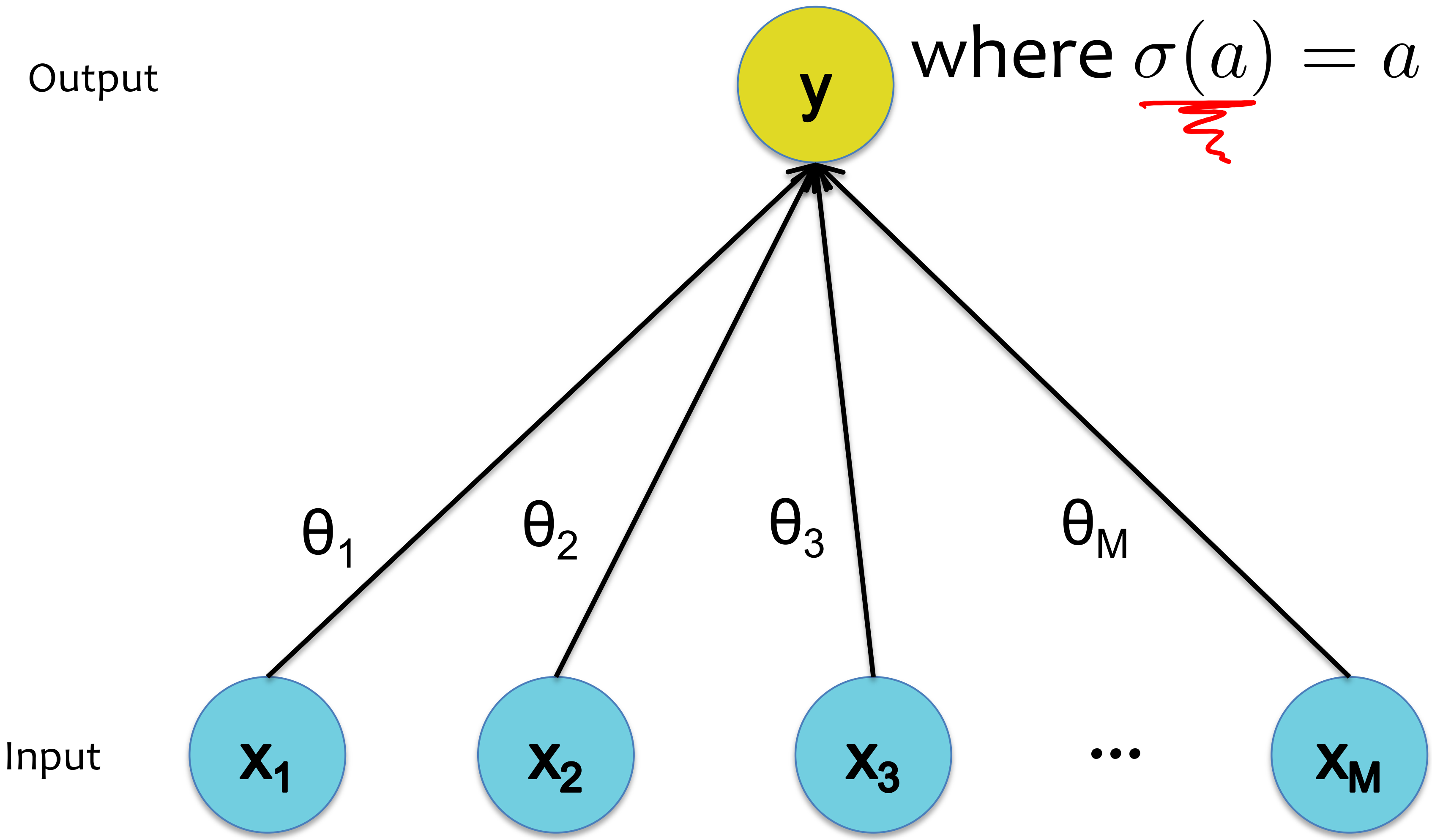
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Decision Functions

Linear Regression

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

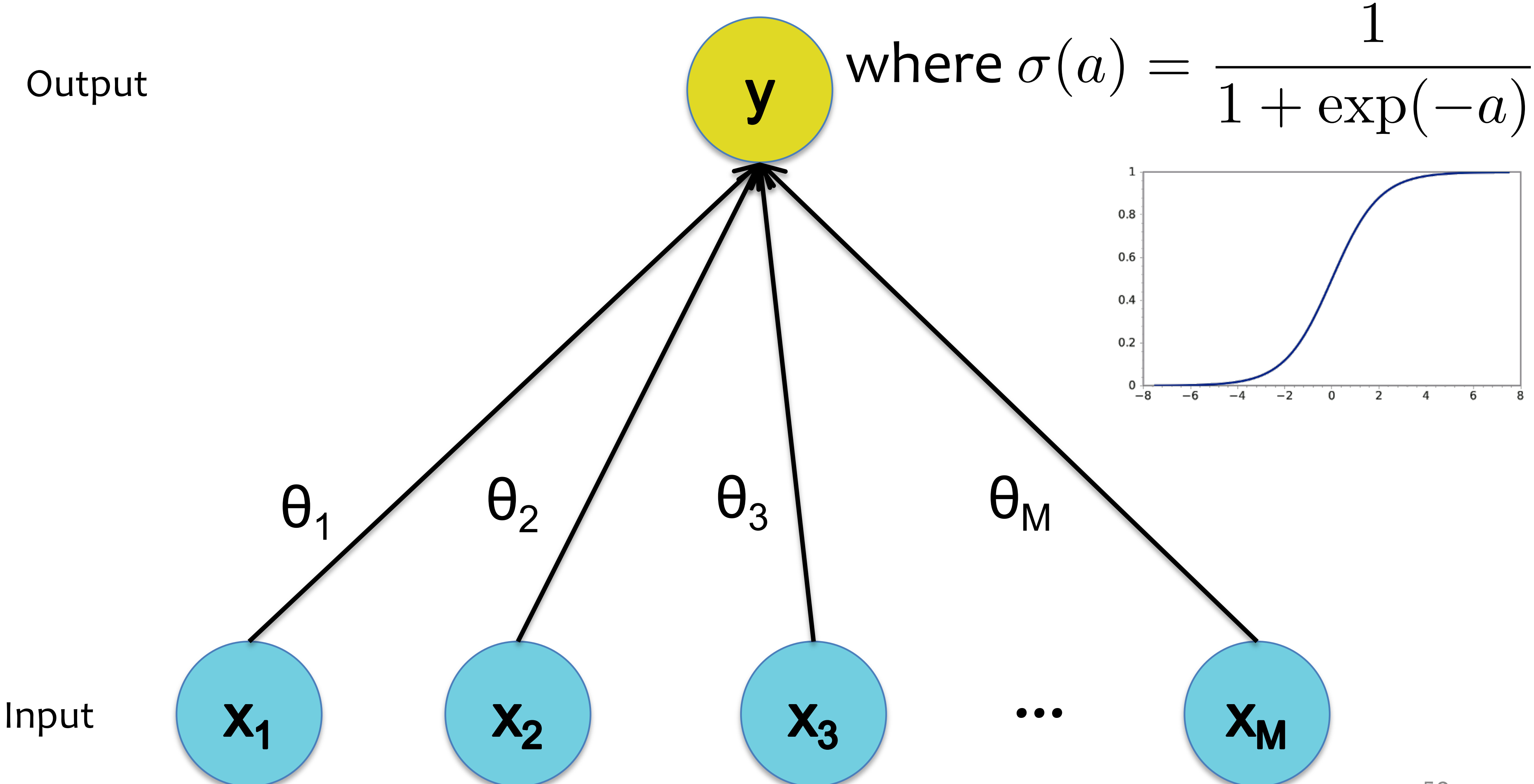
where $\sigma(a) = a$



Logistic Regression

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

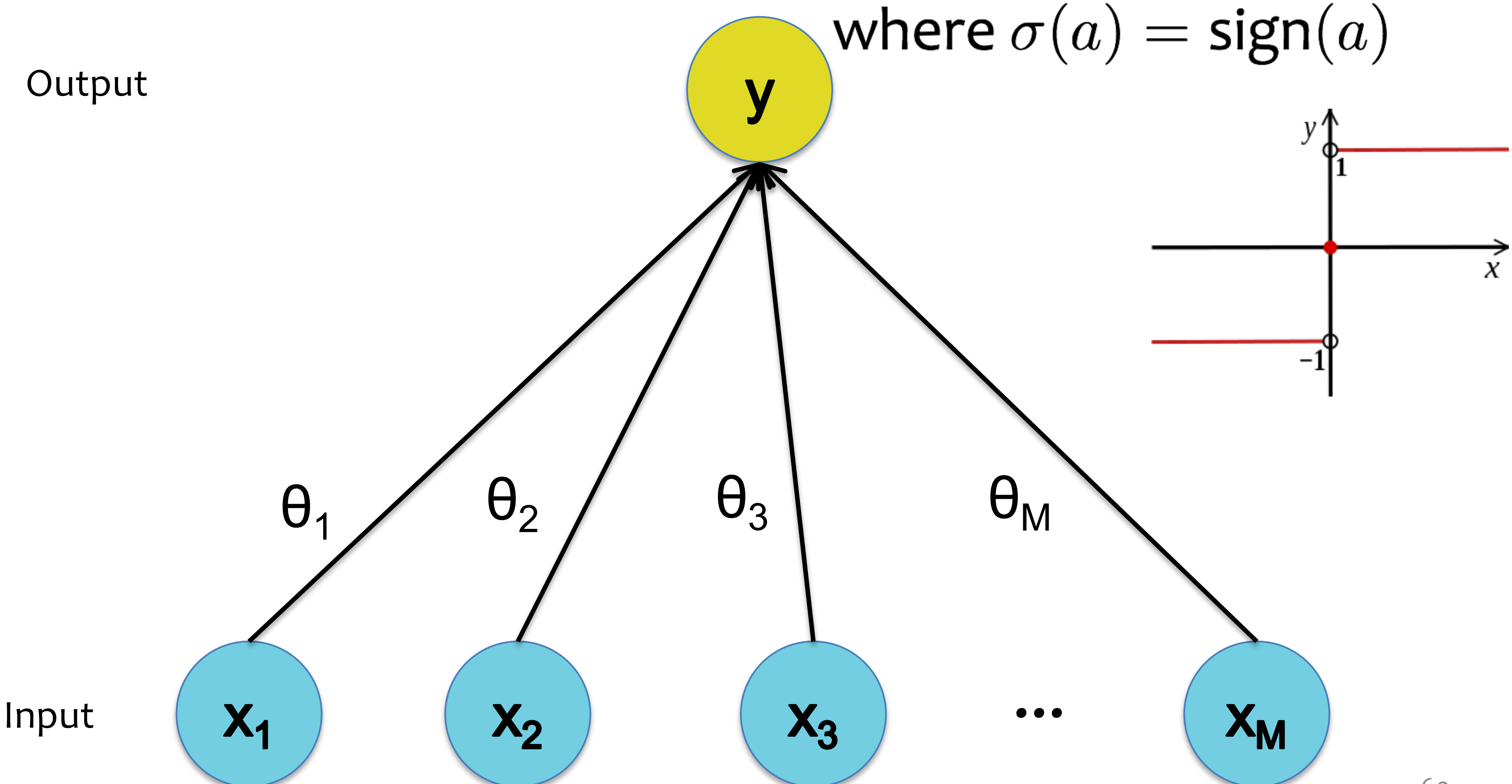
where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



Perceptron

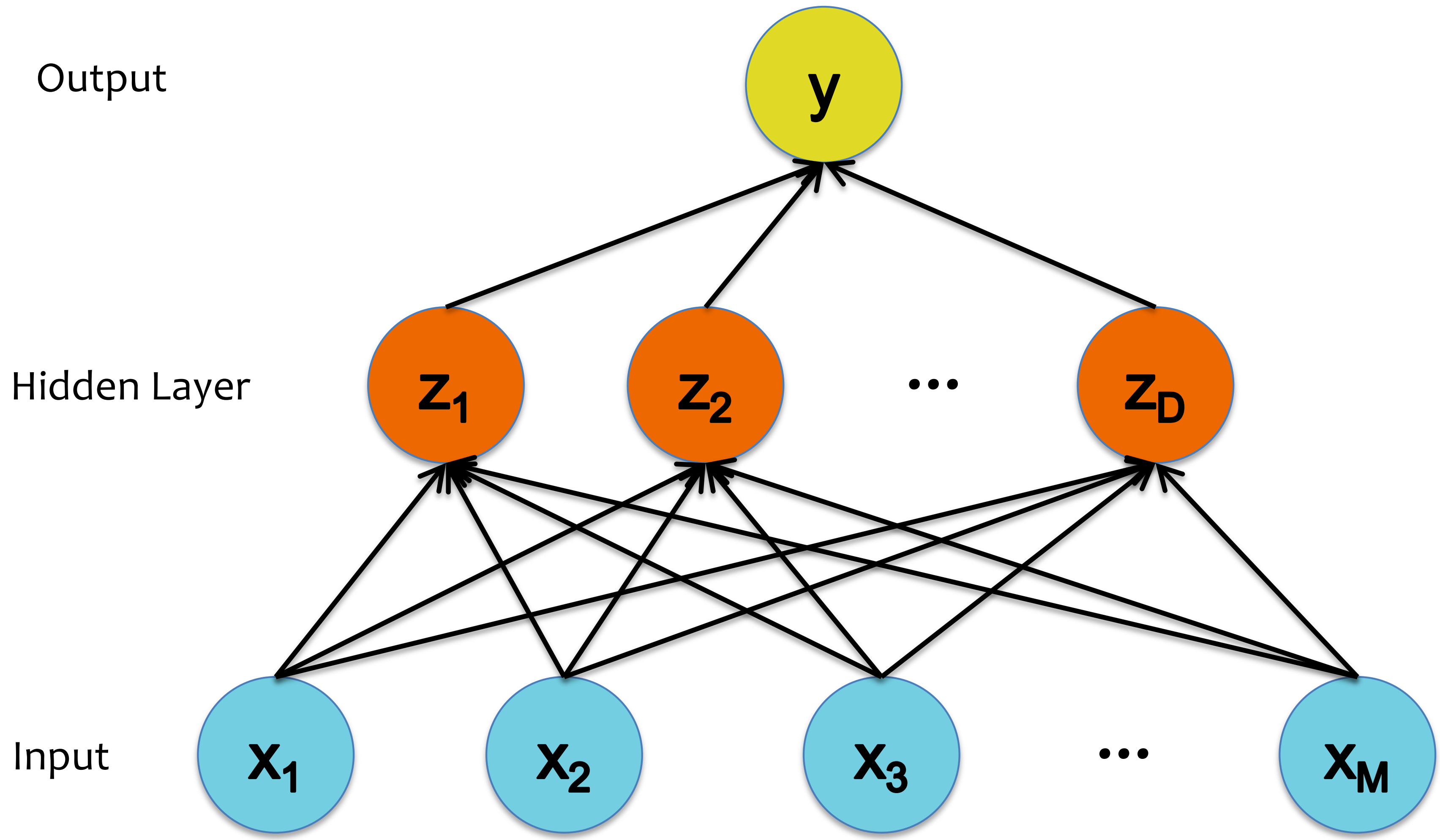
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

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



Decision Functions

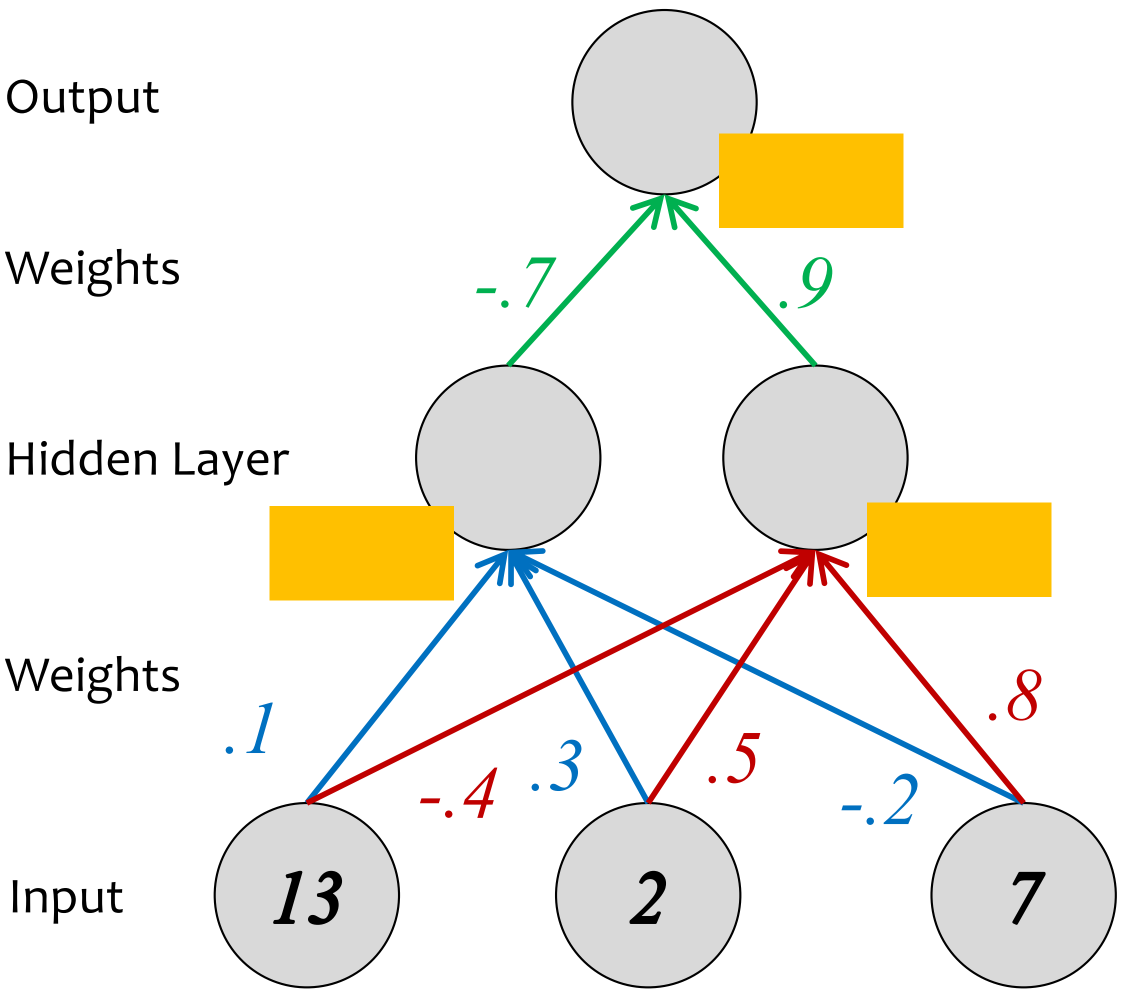
Neural Network



COMPONENTS OF A NEURAL NETWORK

Decision Functions

Neural Network

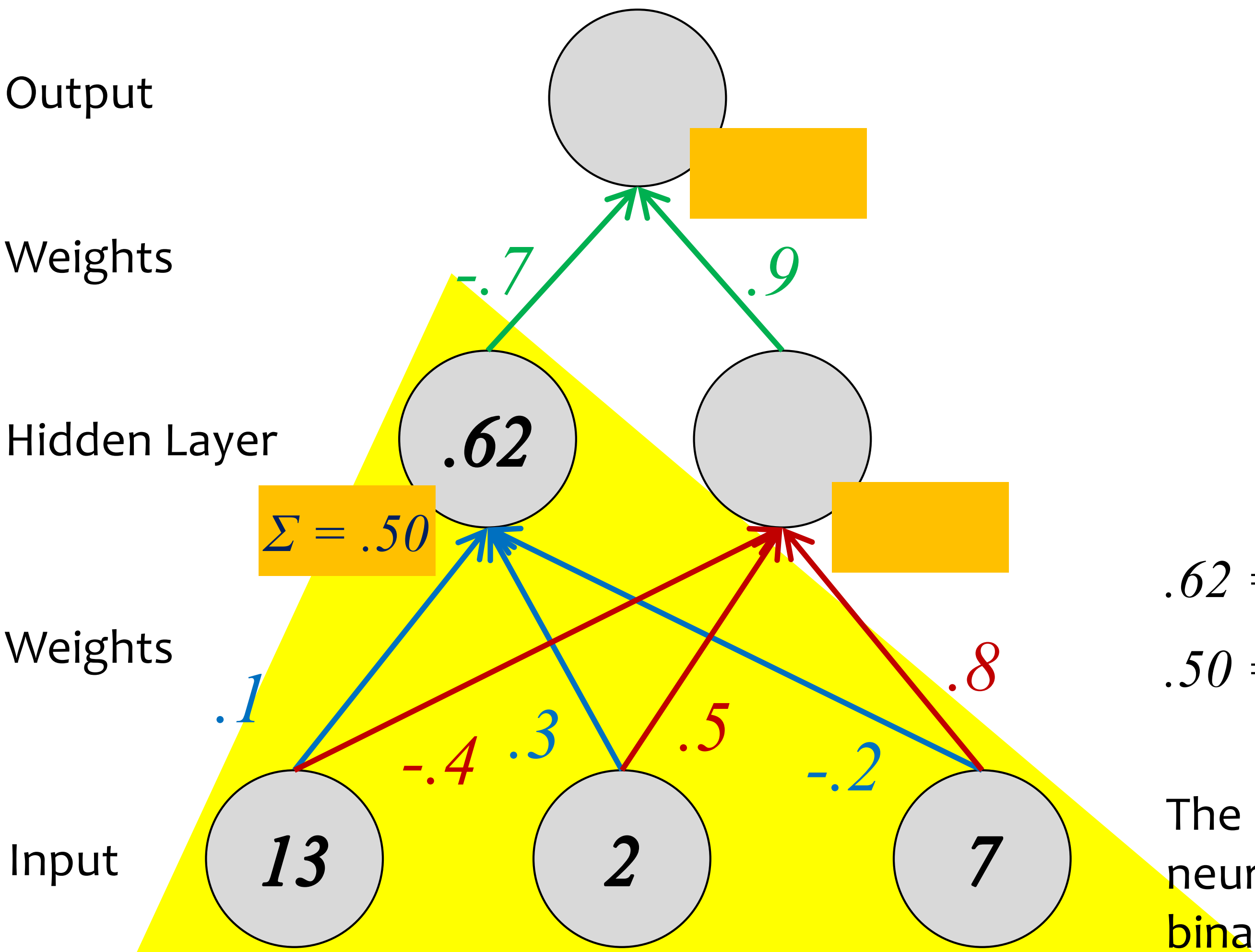


Suppose we already learned the weights of the neural network.

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



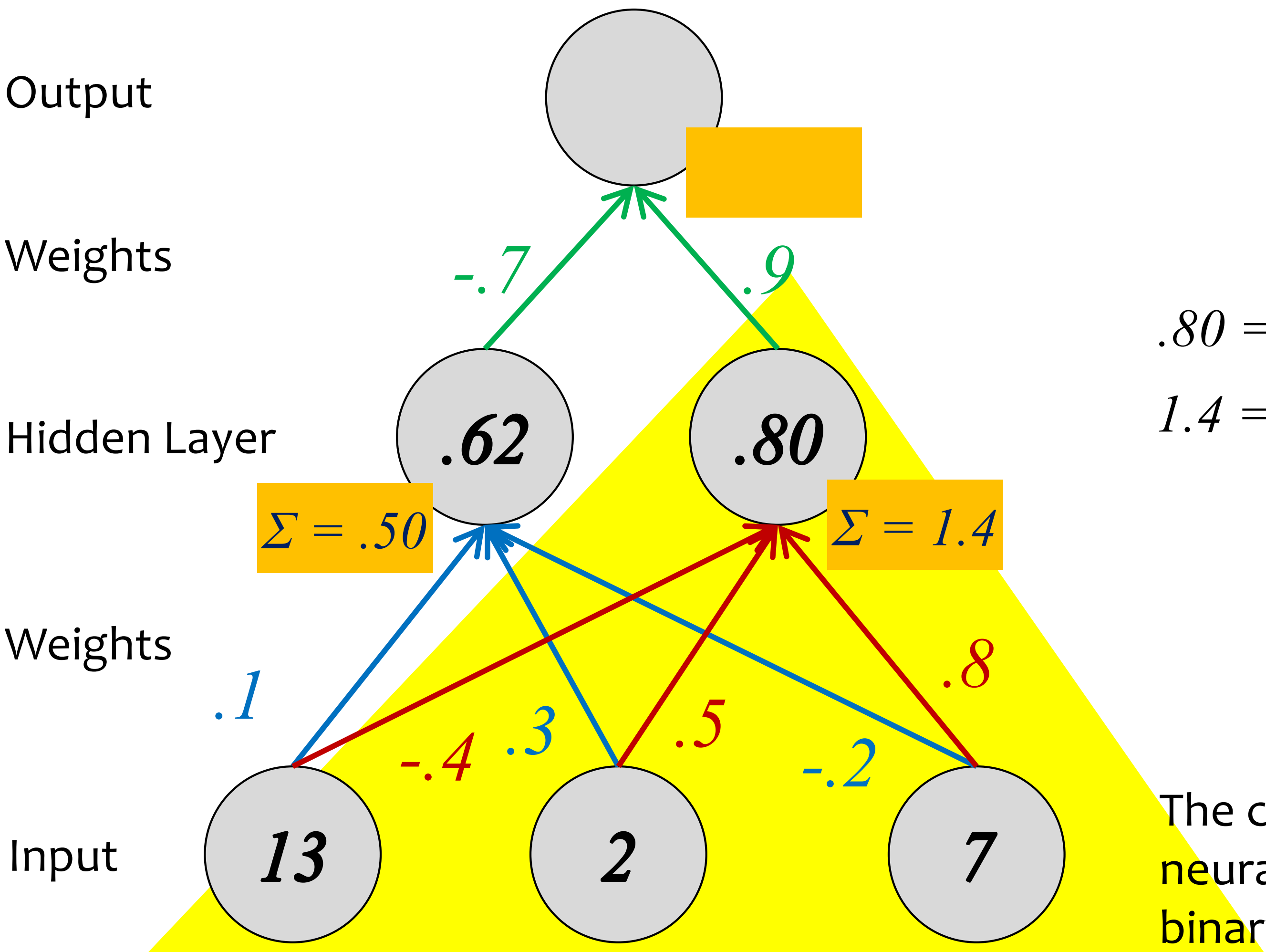
$.62 = \sigma(.50)$

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

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

Decision Functions

Neural Network



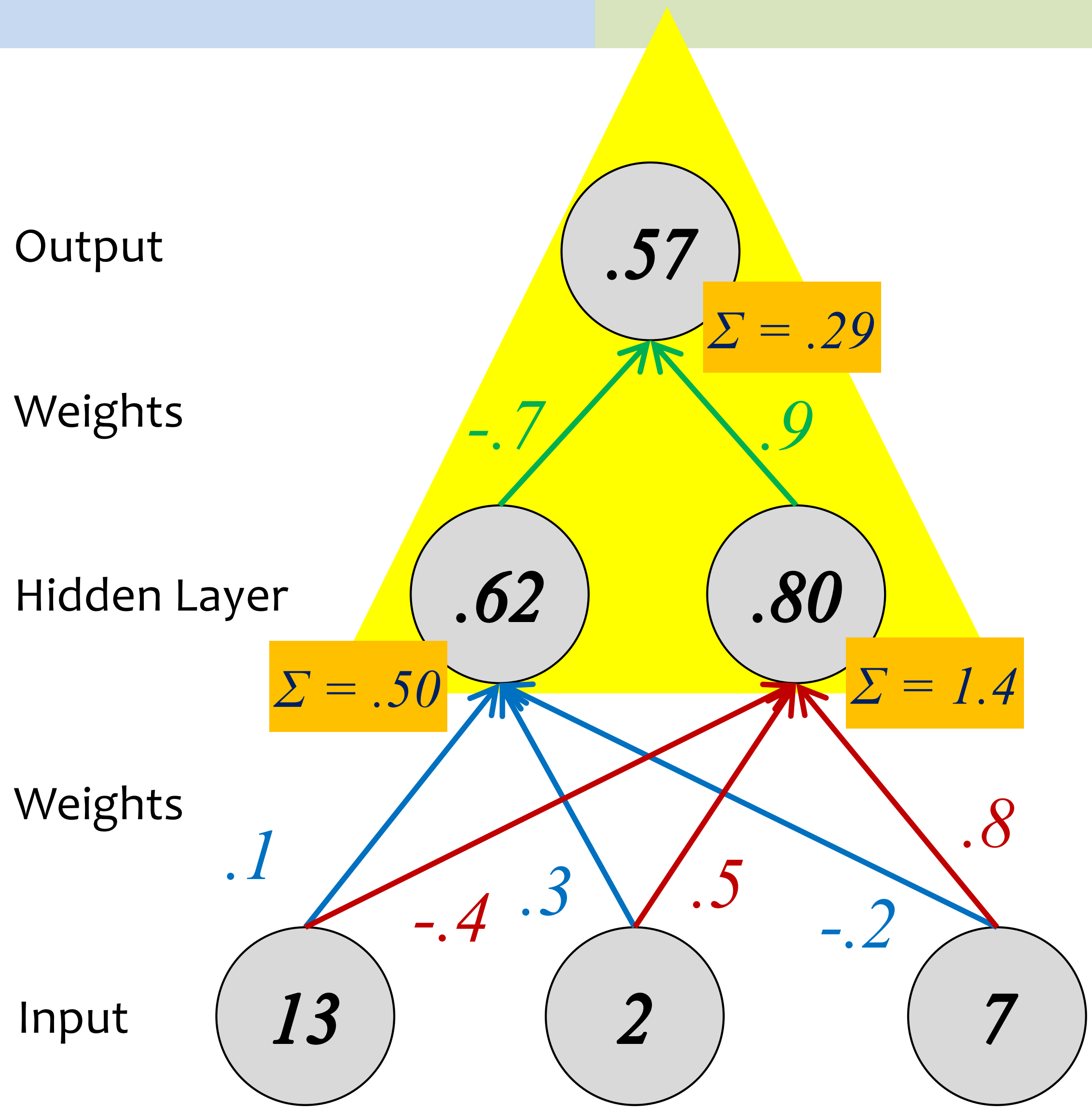
$.80 = \sigma(1.4)$

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

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

Decision Functions

Neural Network



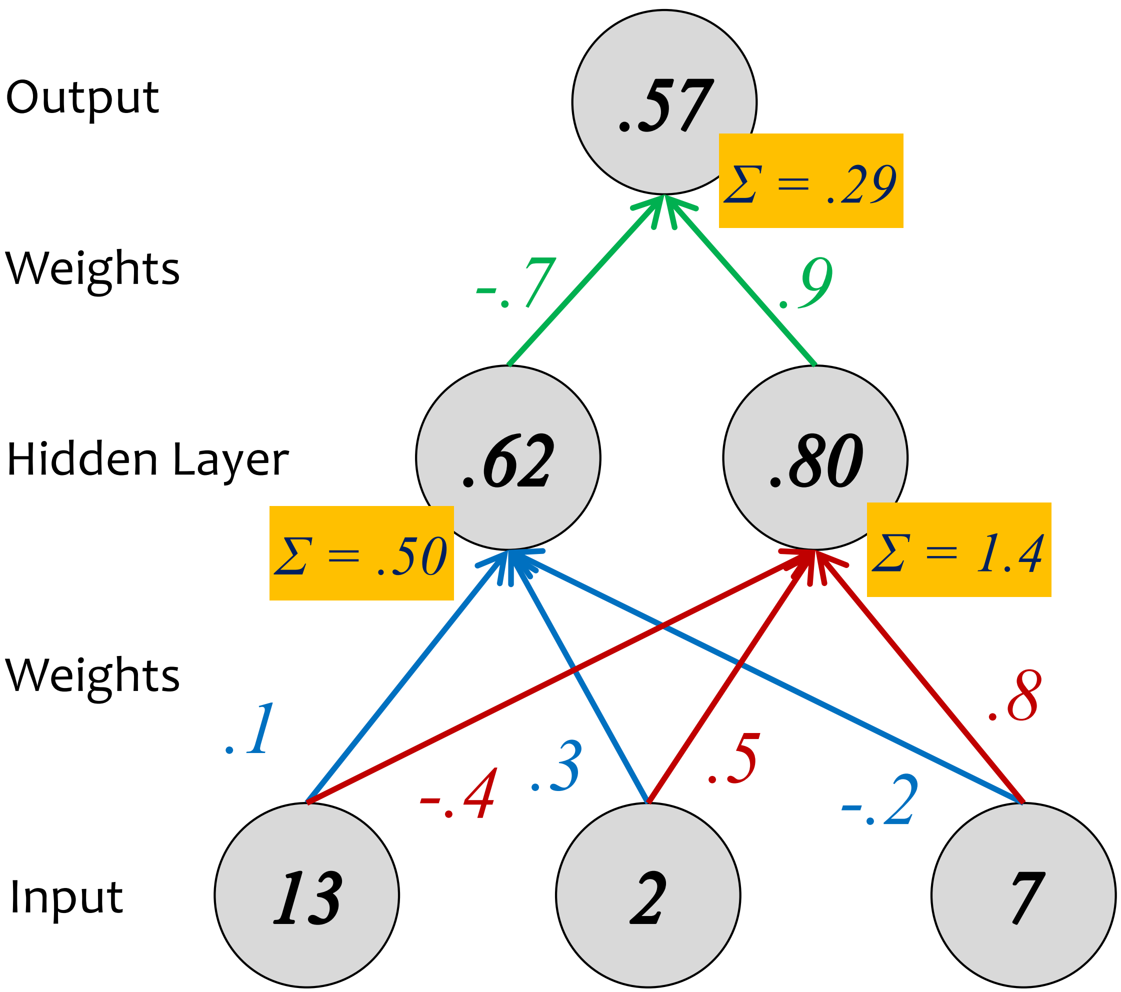
$$.57 = \sigma(.29)$$

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

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

Decision Functions

Neural Network



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

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

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

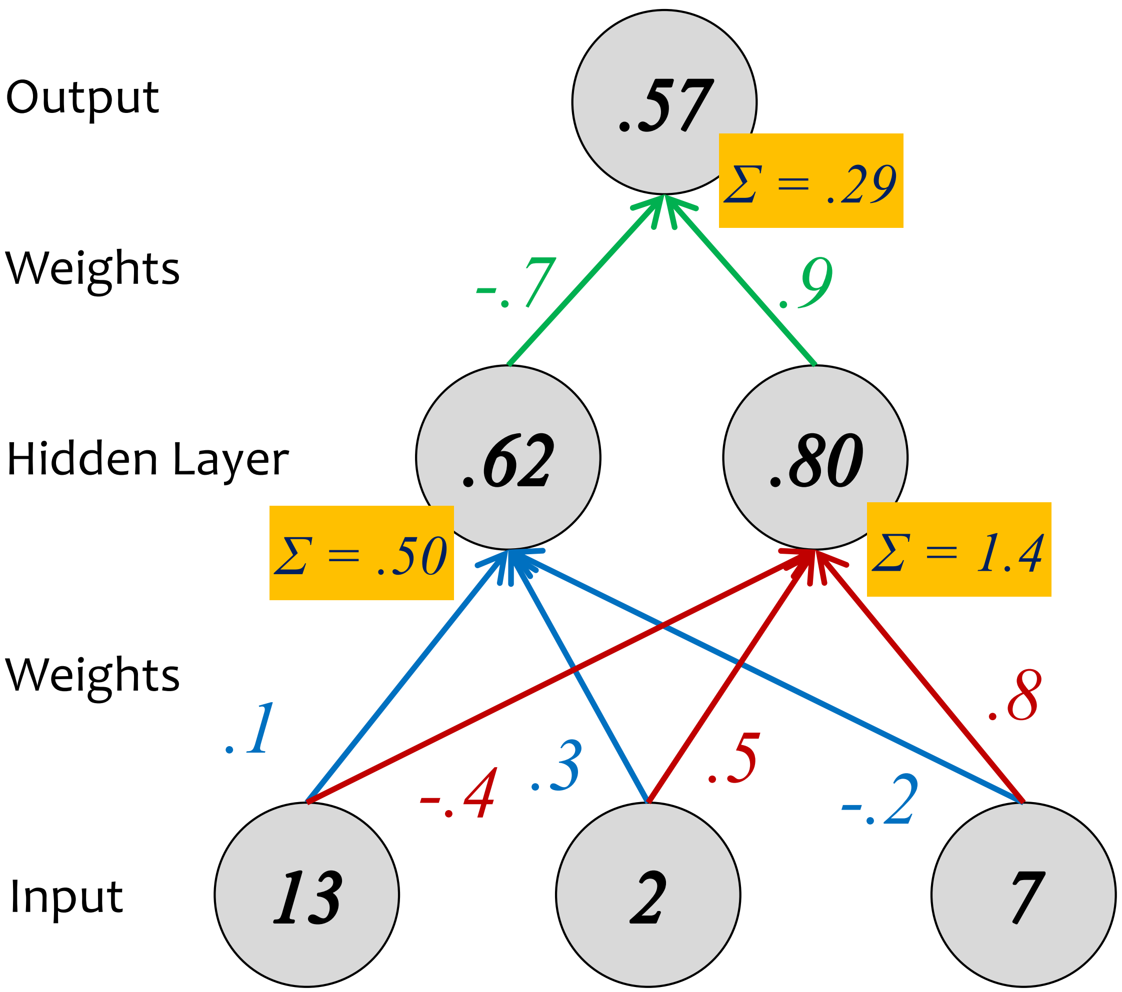
$$1.4 = 13(-.4) + 2(.5) + 7(.8)$$

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

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

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

Neural Network



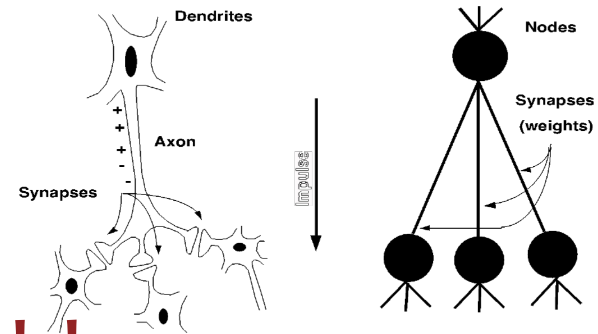
Except we only have the target value for y at training time!

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

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

Biological "Computation"

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

Artificial Computation

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

Slide adapted from Eric Xing

DEFINING A 1-HIDDEN LAYER NEURAL NETWORK

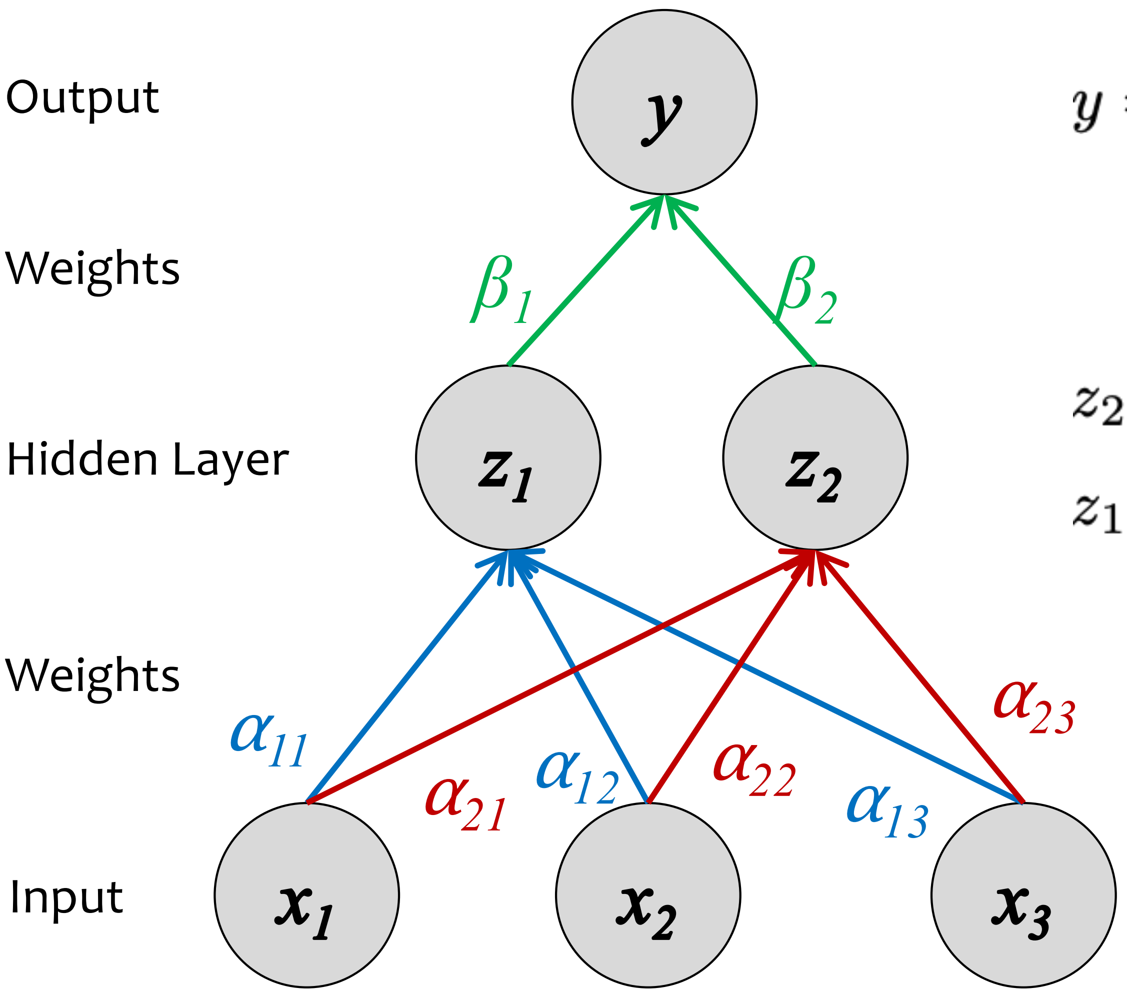
Neural Networks

Chalkboard

– Example: Neural Network w/1 Hidden Layer

Decision Functions

Neural Network



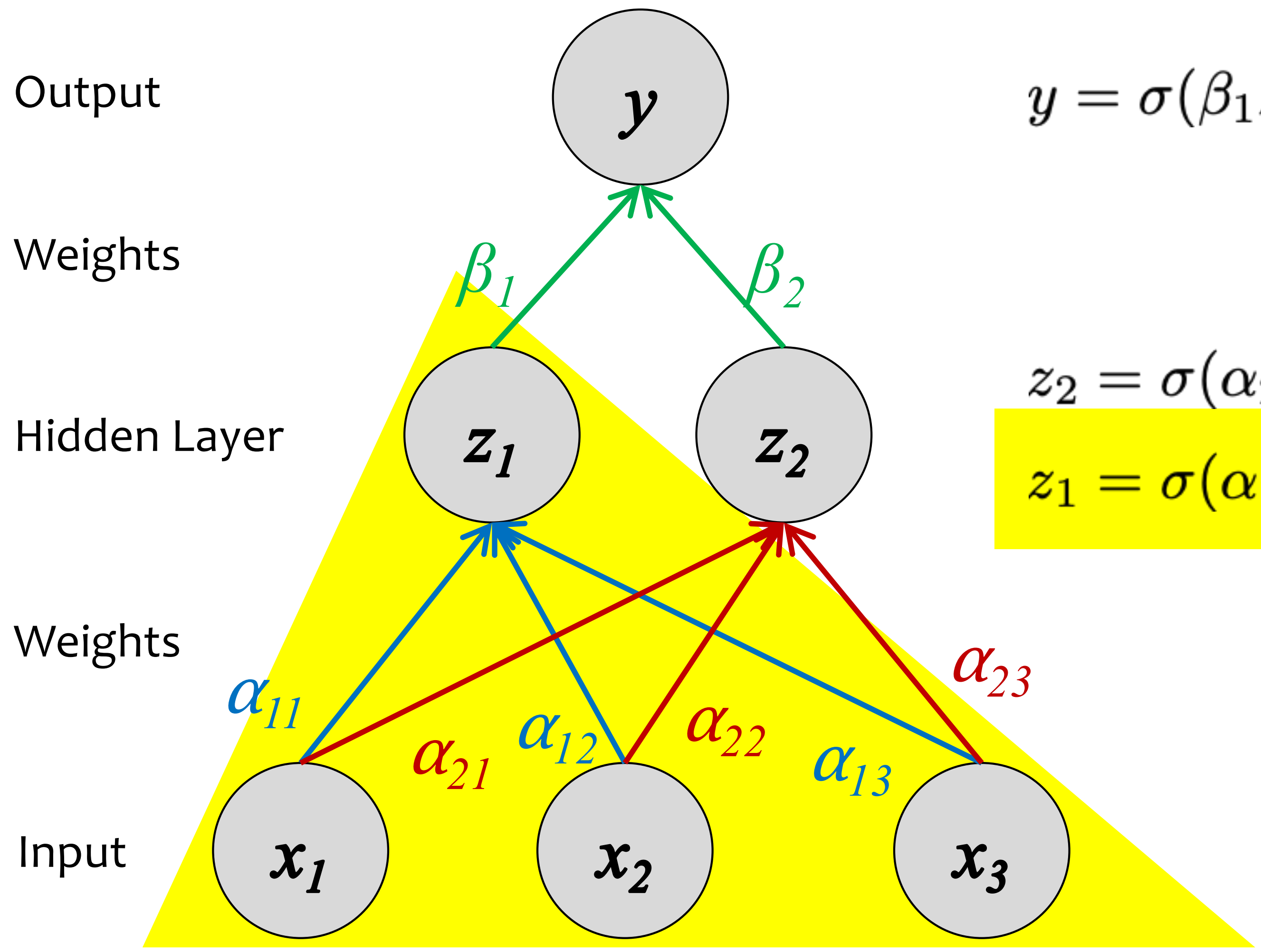
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network

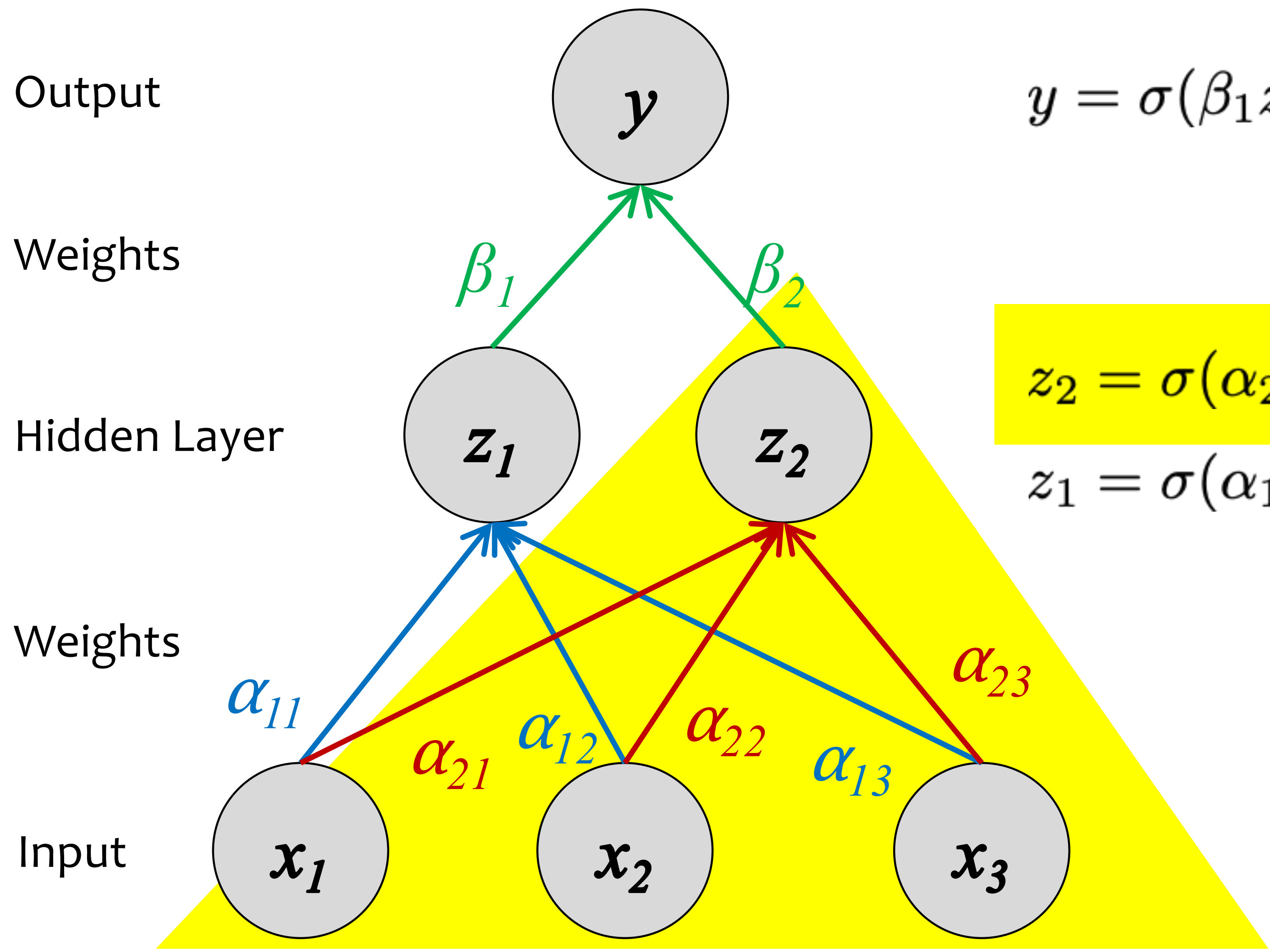


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Neural Network



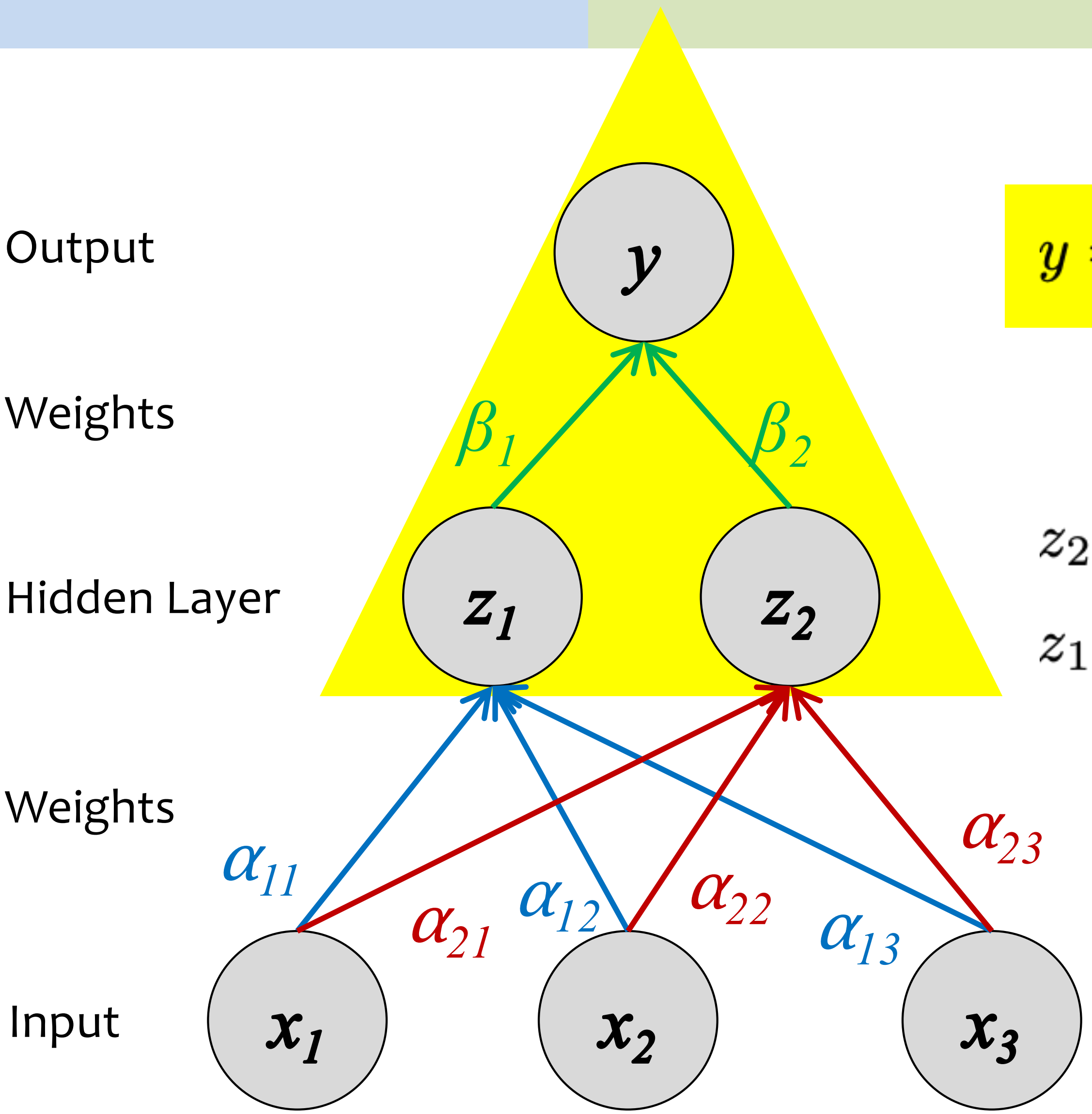
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network



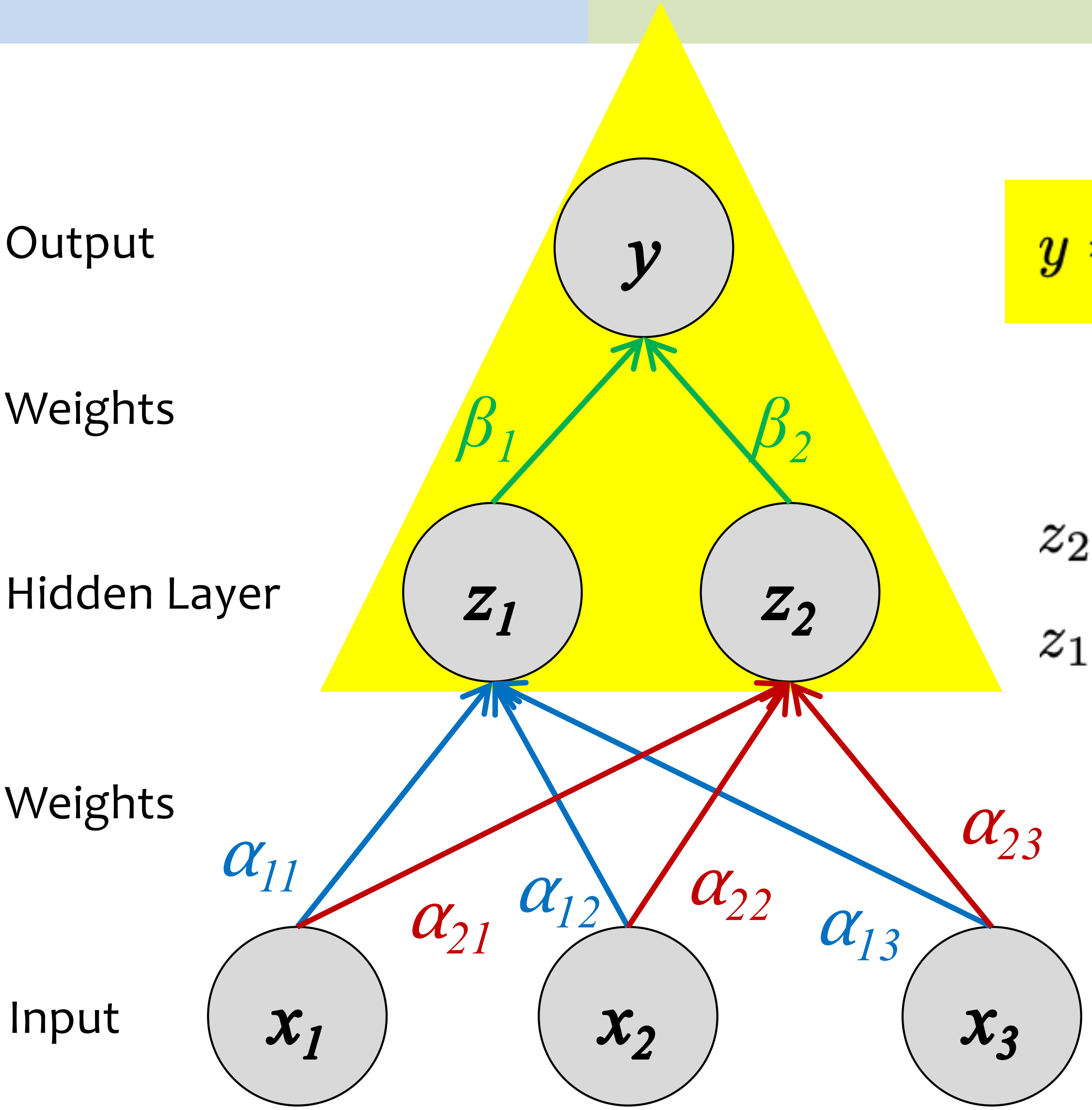
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network



$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

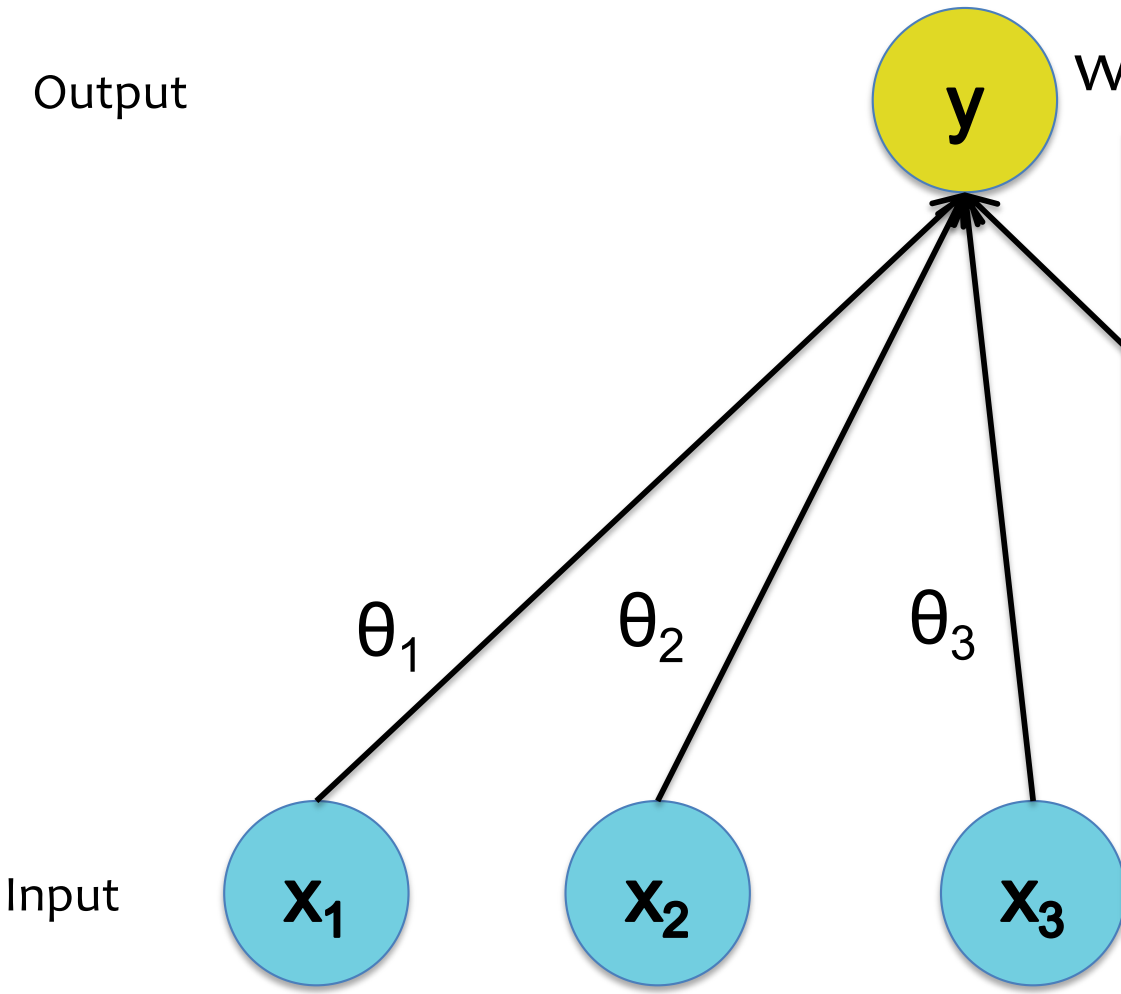
NONLINEAR DECISION BOUNDARIES AND NEURAL NETWORKS

Decision Functions

Logistic Regression

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

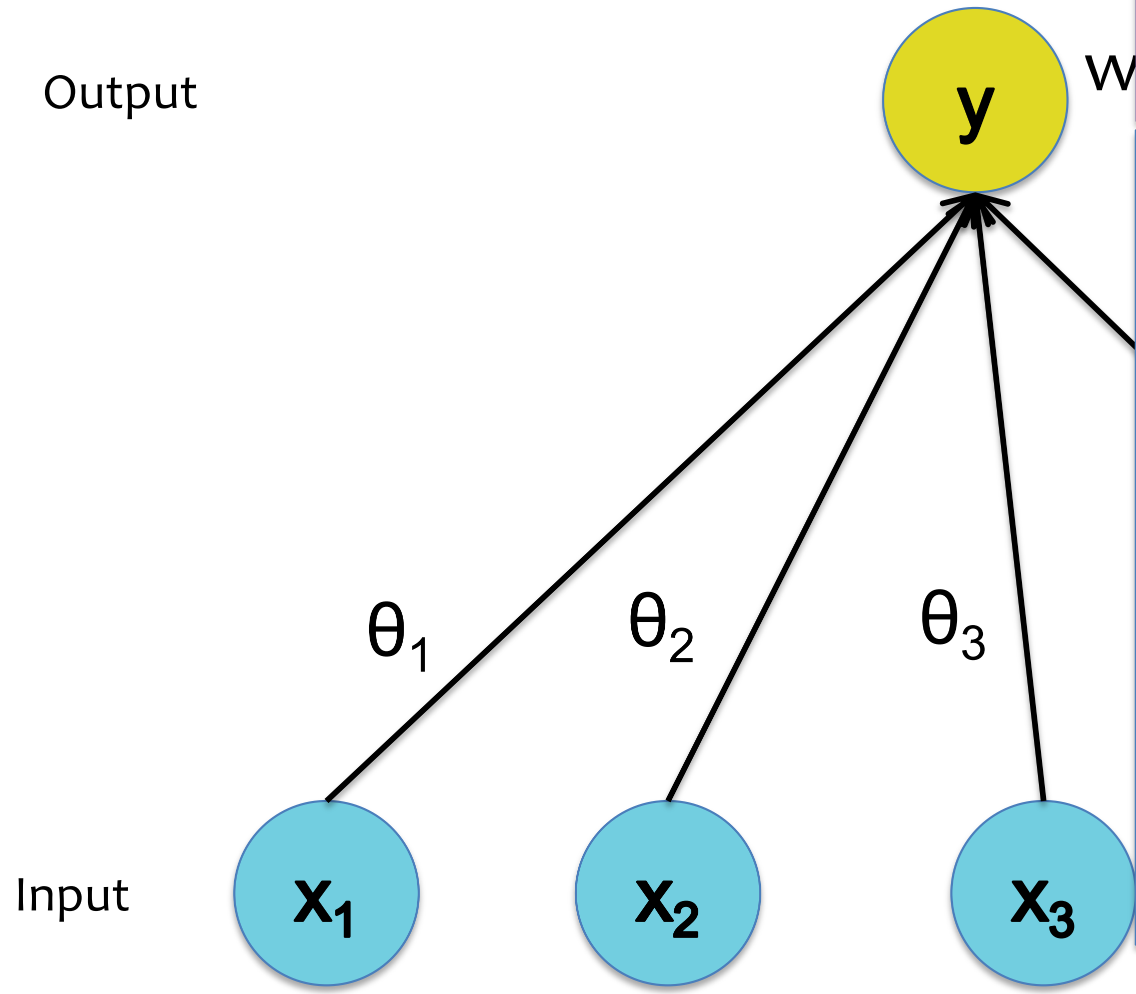
where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



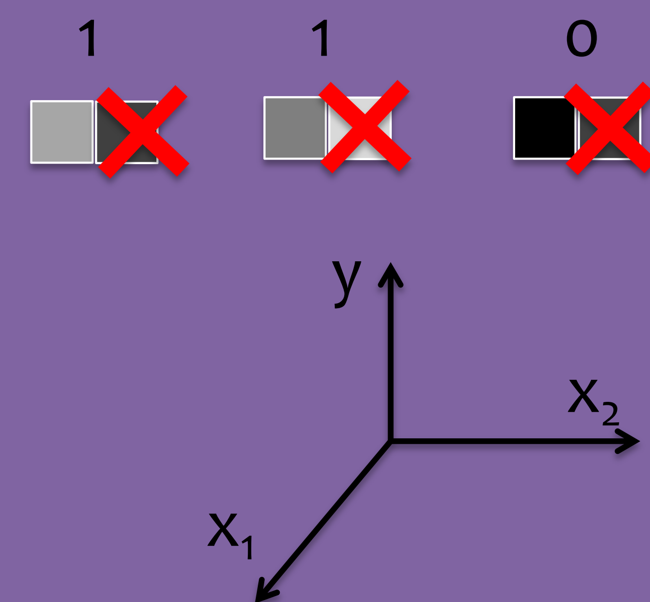
Decision Functions

Logistic Regression

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



In-Class Example



Neural Networks

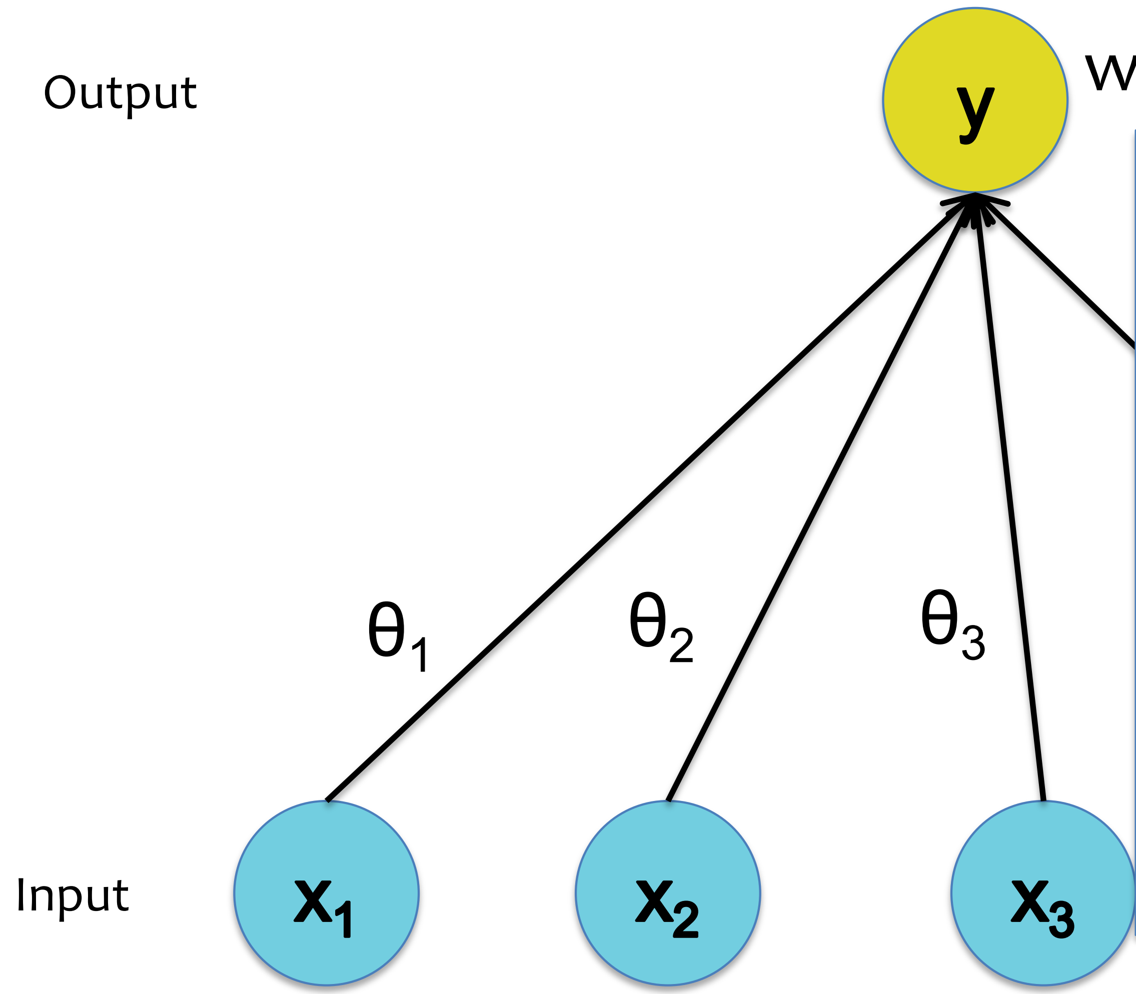
Chalkboard

- 1D Example from linear regression to logistic regression
- 1D Example from logistic regression to a neural network

Logistic Regression

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

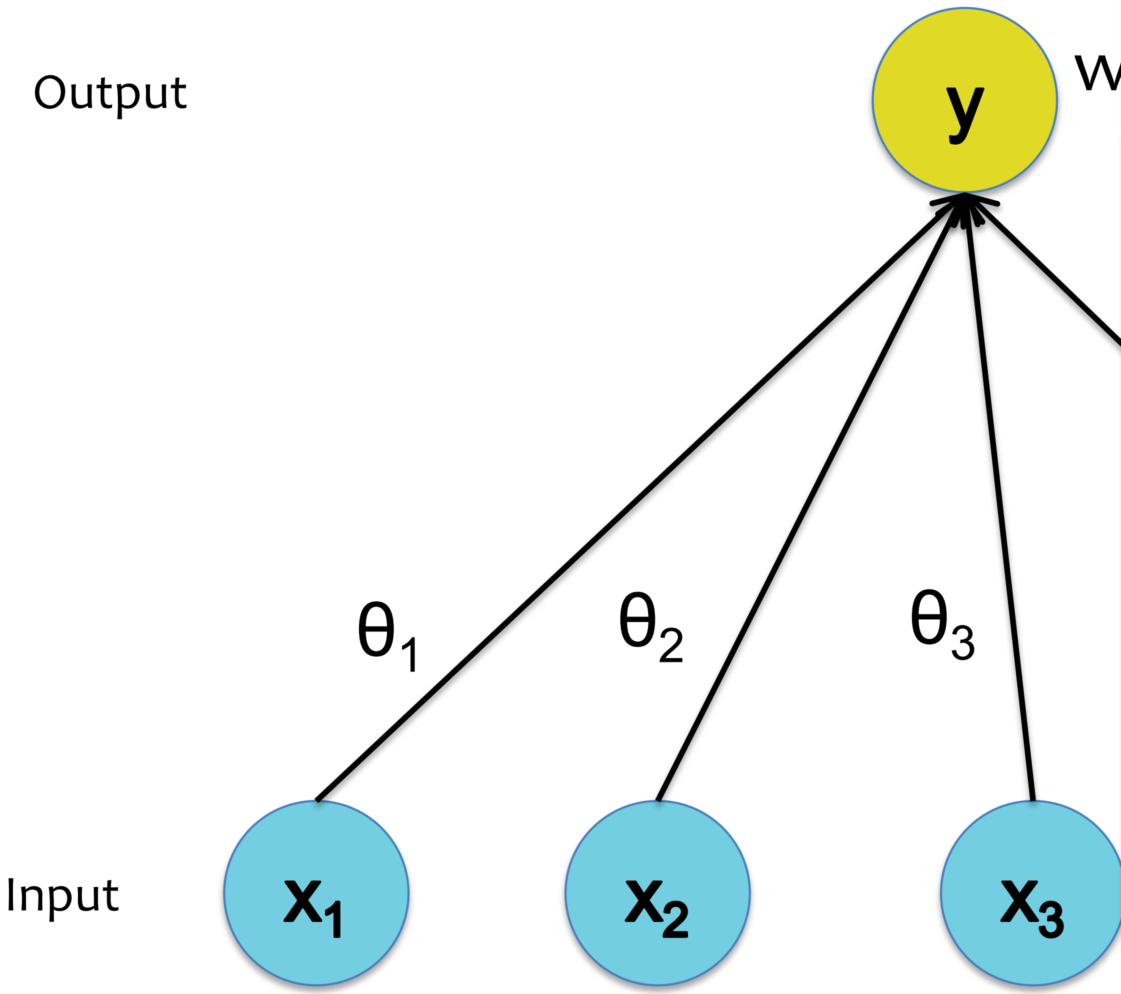
where $\sigma(a) = \frac{1}{1 + \exp(-a)}$



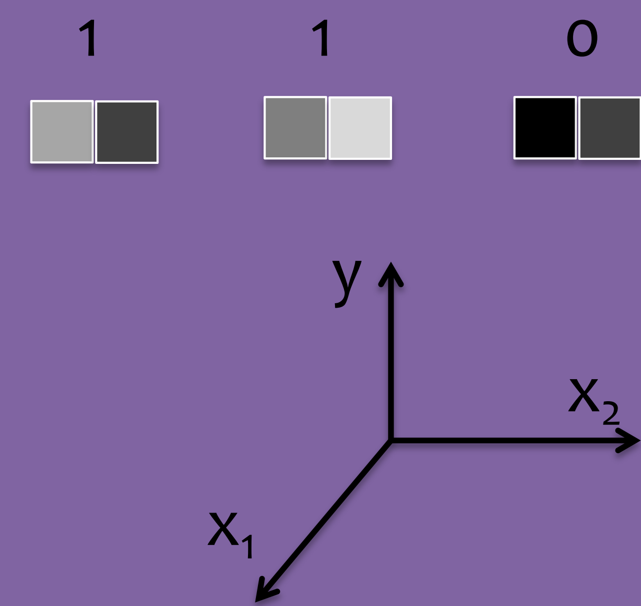
Decision Functions

Logistic Regression

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



In-Class Example



Neural Network Parameters

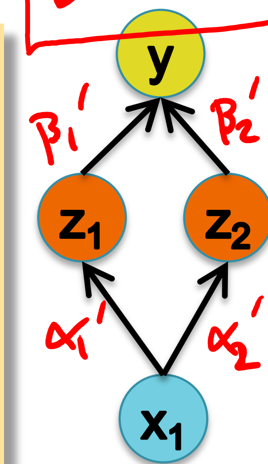
55%

Q2: A = True $\boxed{B = \text{False}}$ C = toxic

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.

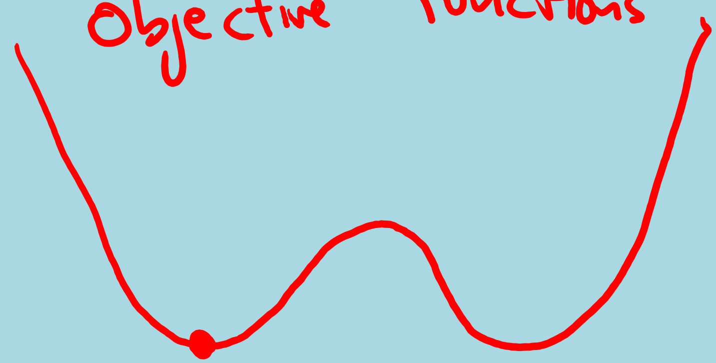


$\alpha_1' = \alpha_2$ $\alpha_2' = \alpha_1$

$\beta_1' = \beta_2$ $\beta_2' = \beta_1$

Answer:

NNs have nonconvex objective functions



ARCHITECTURES

Neural Network Architectures

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

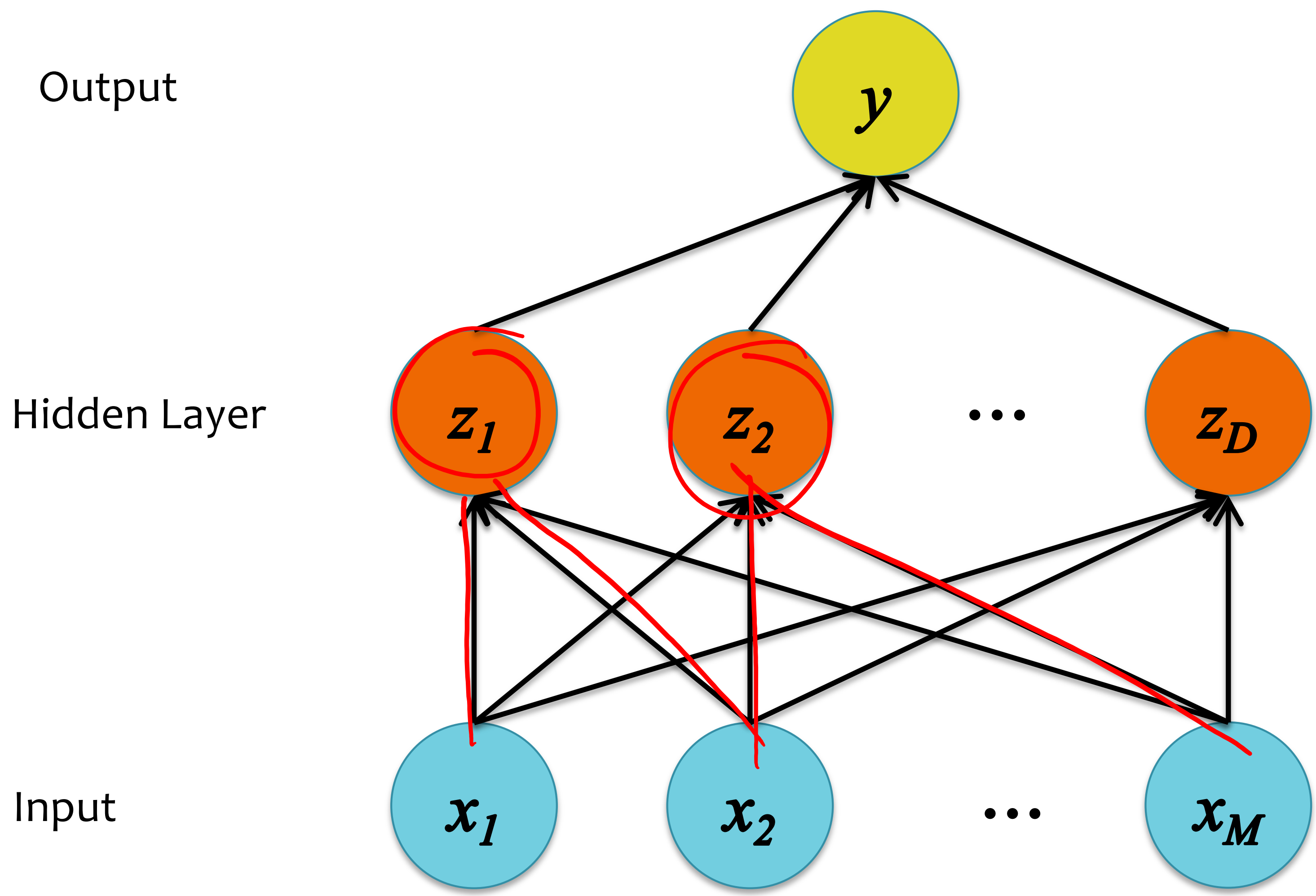
1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters

BUILDING WIDER NETWORKS

D = M

Building a Neural Net

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

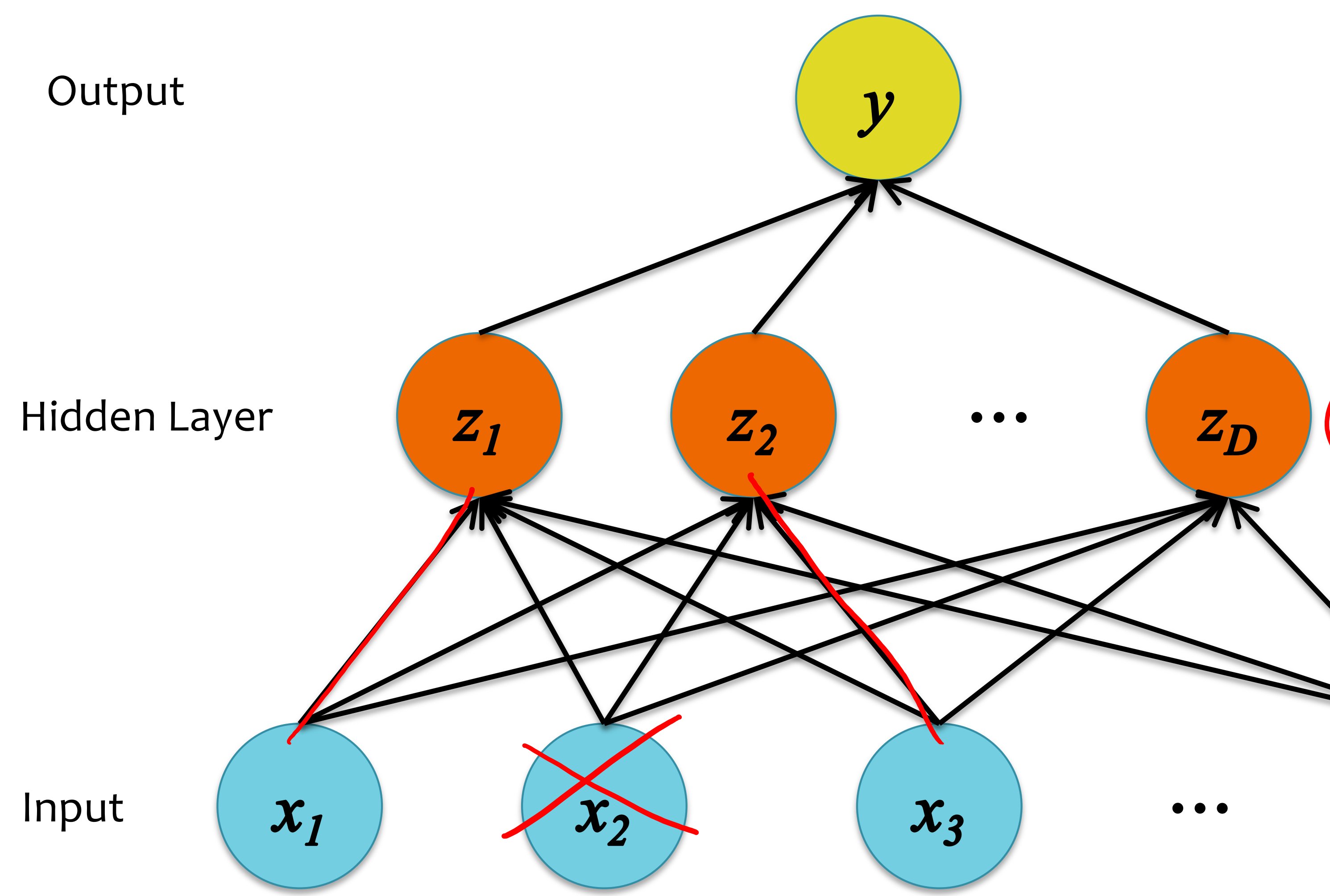


The hidden units could learn to be…

- 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

Building a Neural Net

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



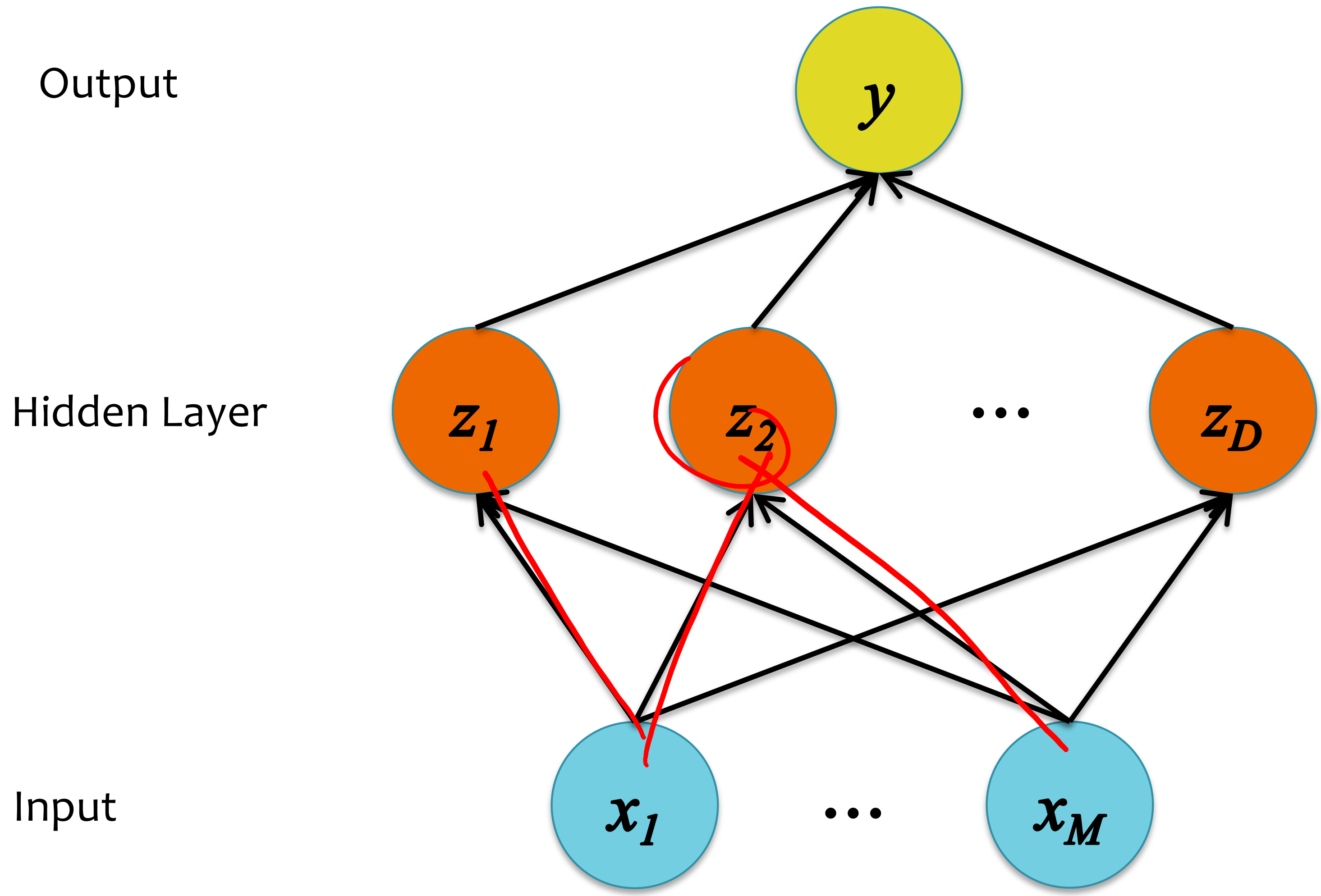
The hidden units could learn to be…

- 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?



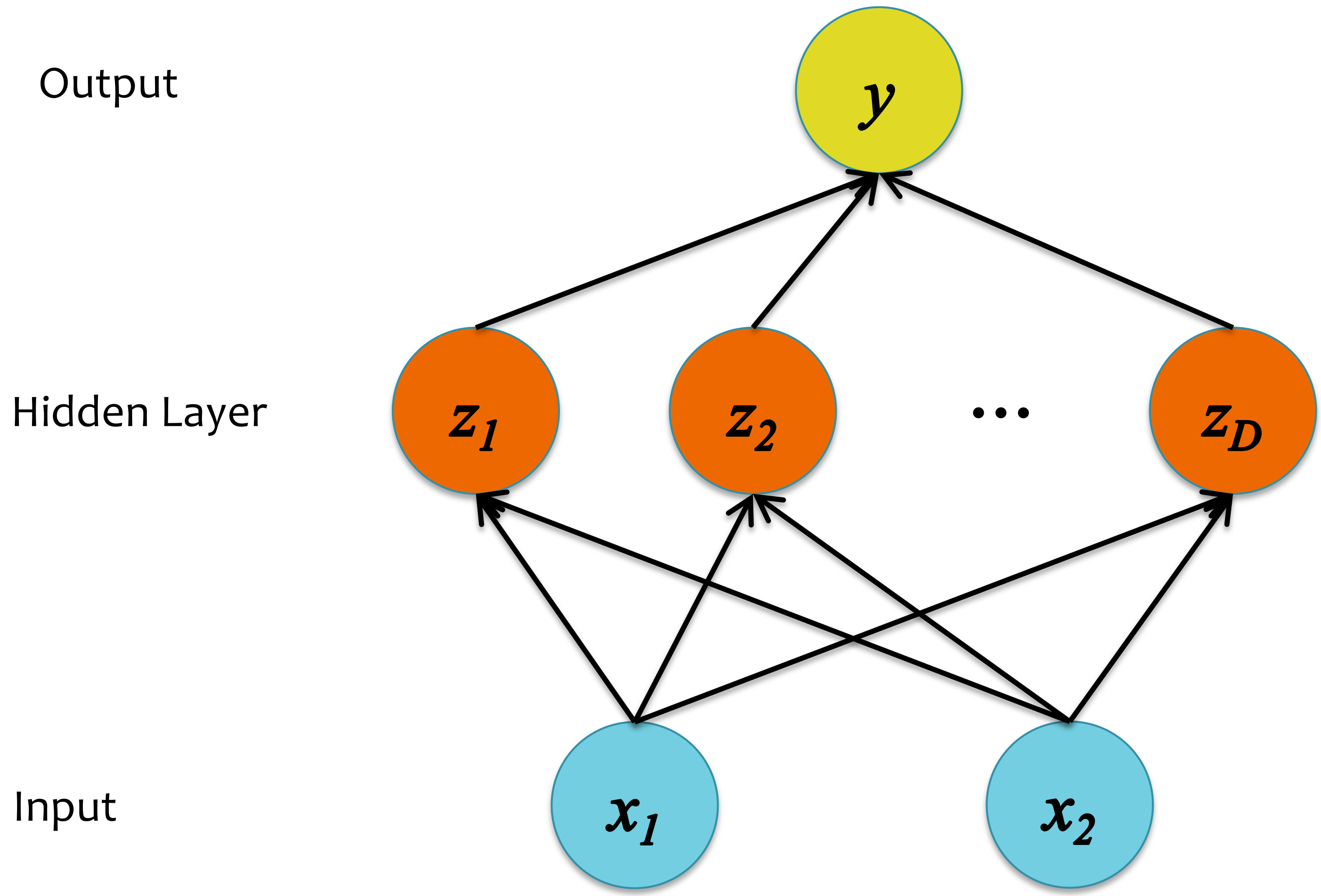
The hidden units could learn to be…

- 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

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



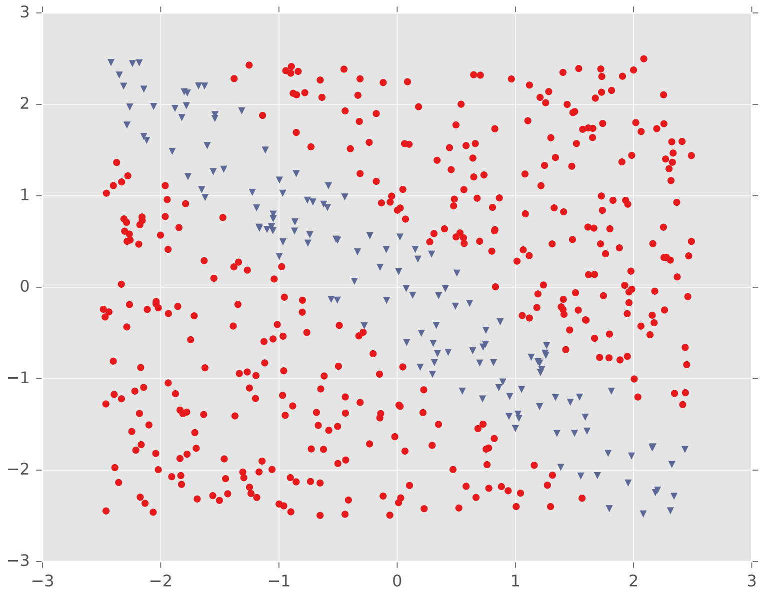
The hidden units could learn to be…

- 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

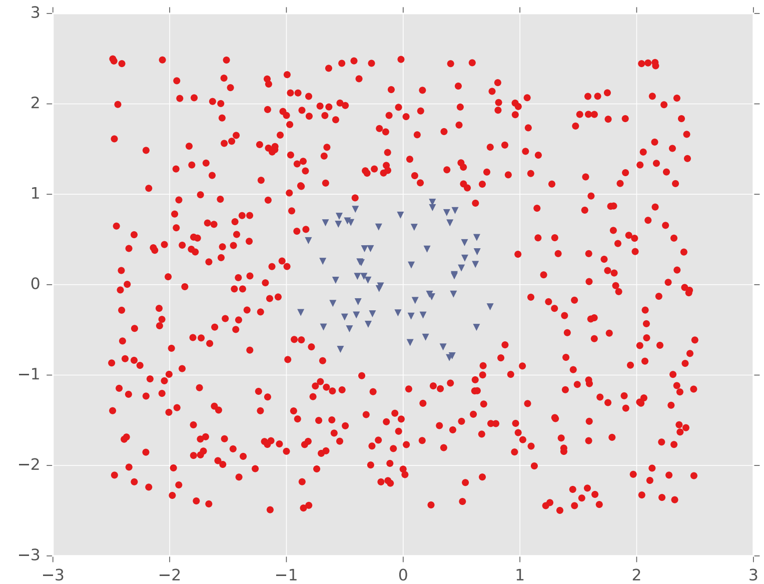
Examples 1 and 2

DECISION BOUNDARY EXAMPLES

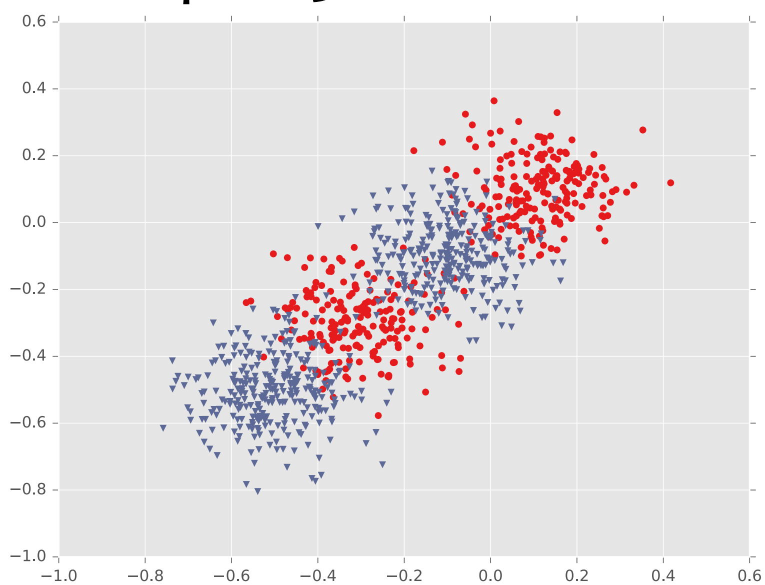
Example #1: Diagonal Band



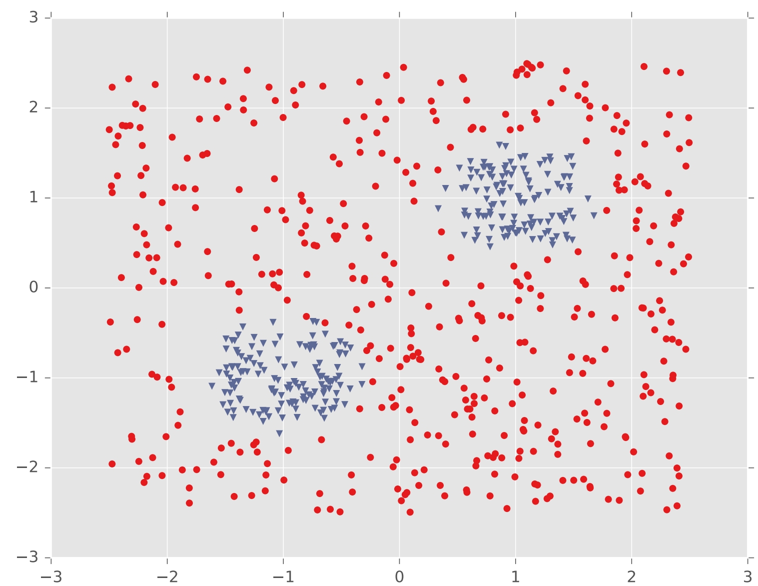
Example #2: One Pocket



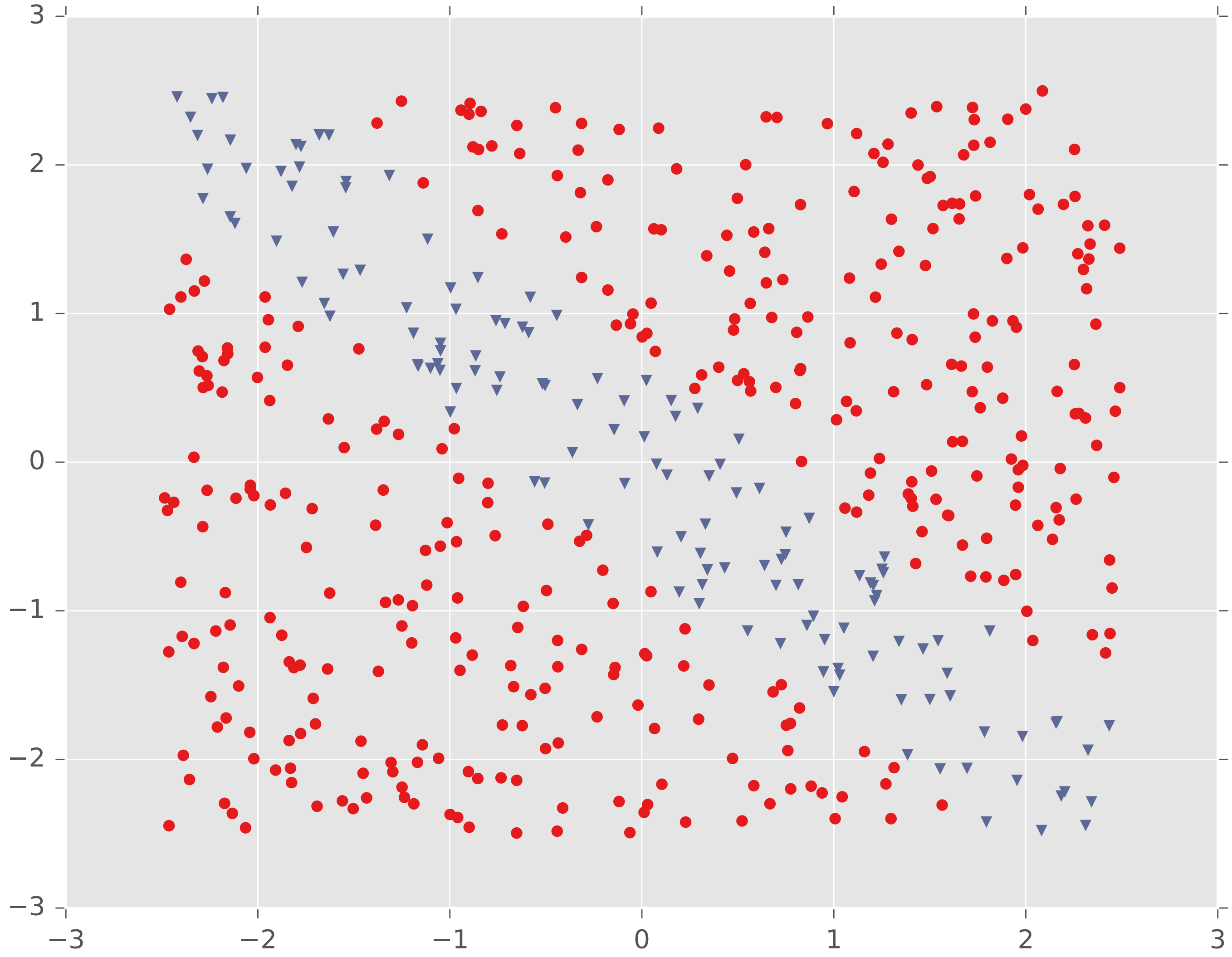
Example #3: Four Gaussians



Example #4: Two Pockets

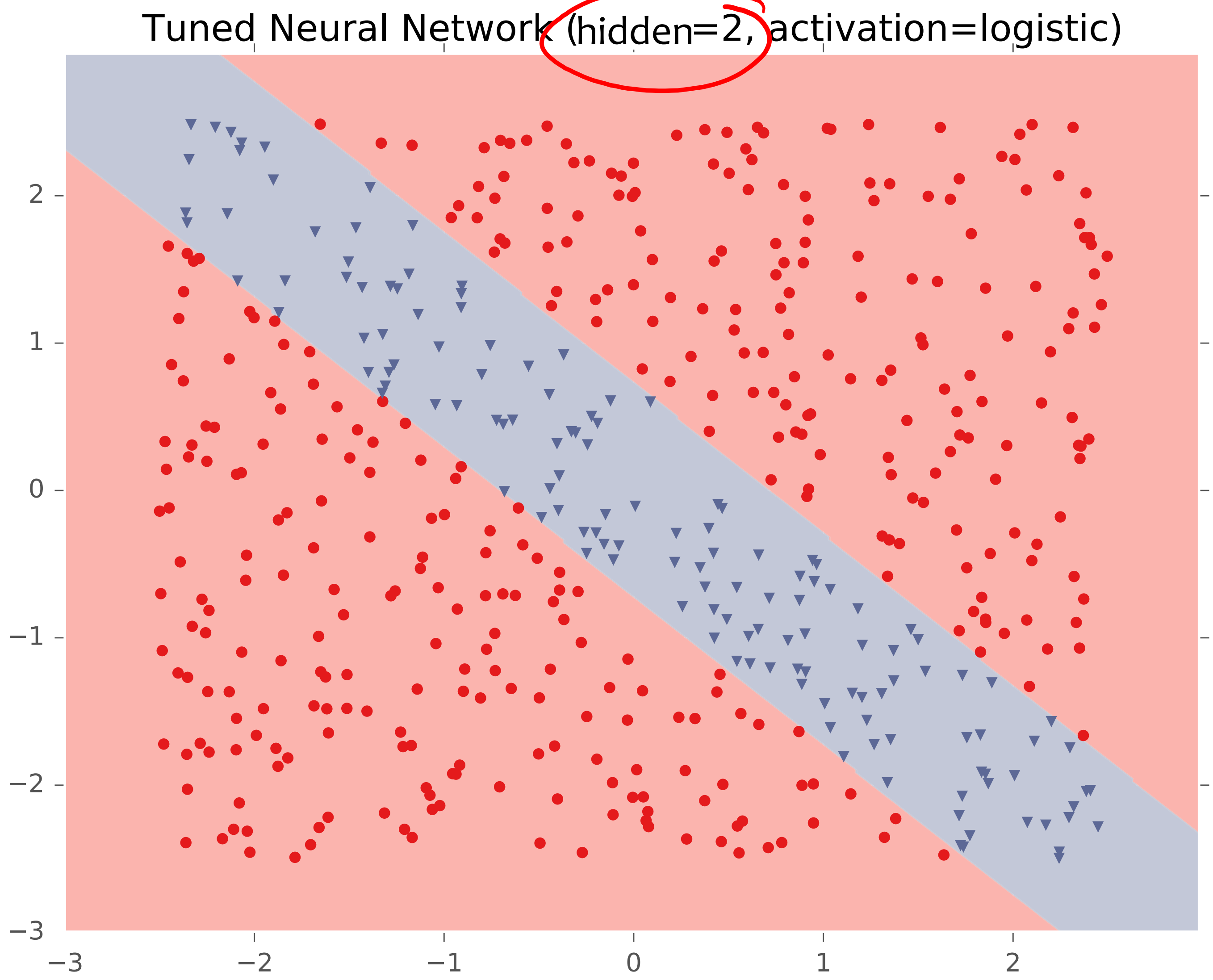


Example #1: Diagonal Band

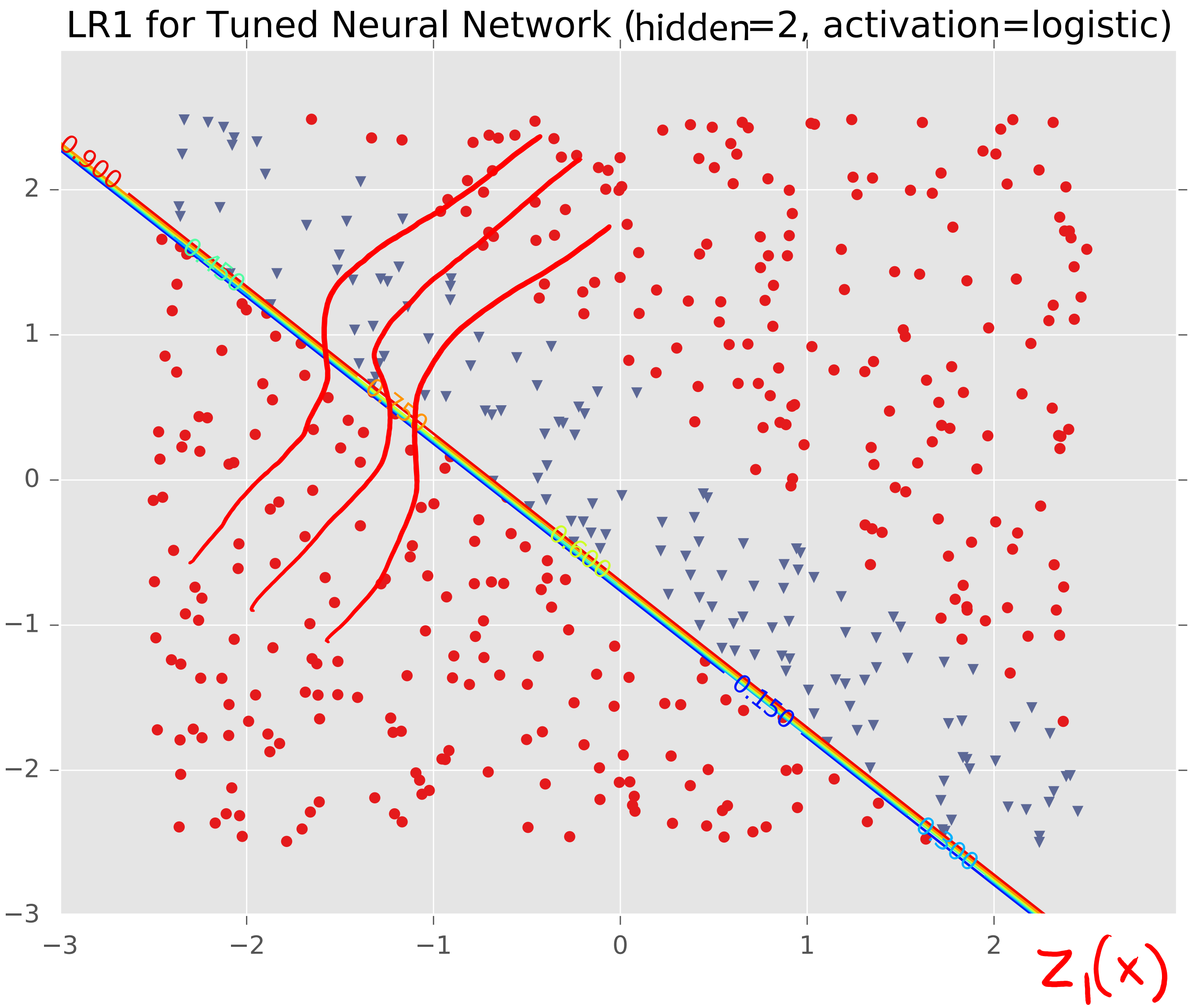


Example #1: Diagonal Band

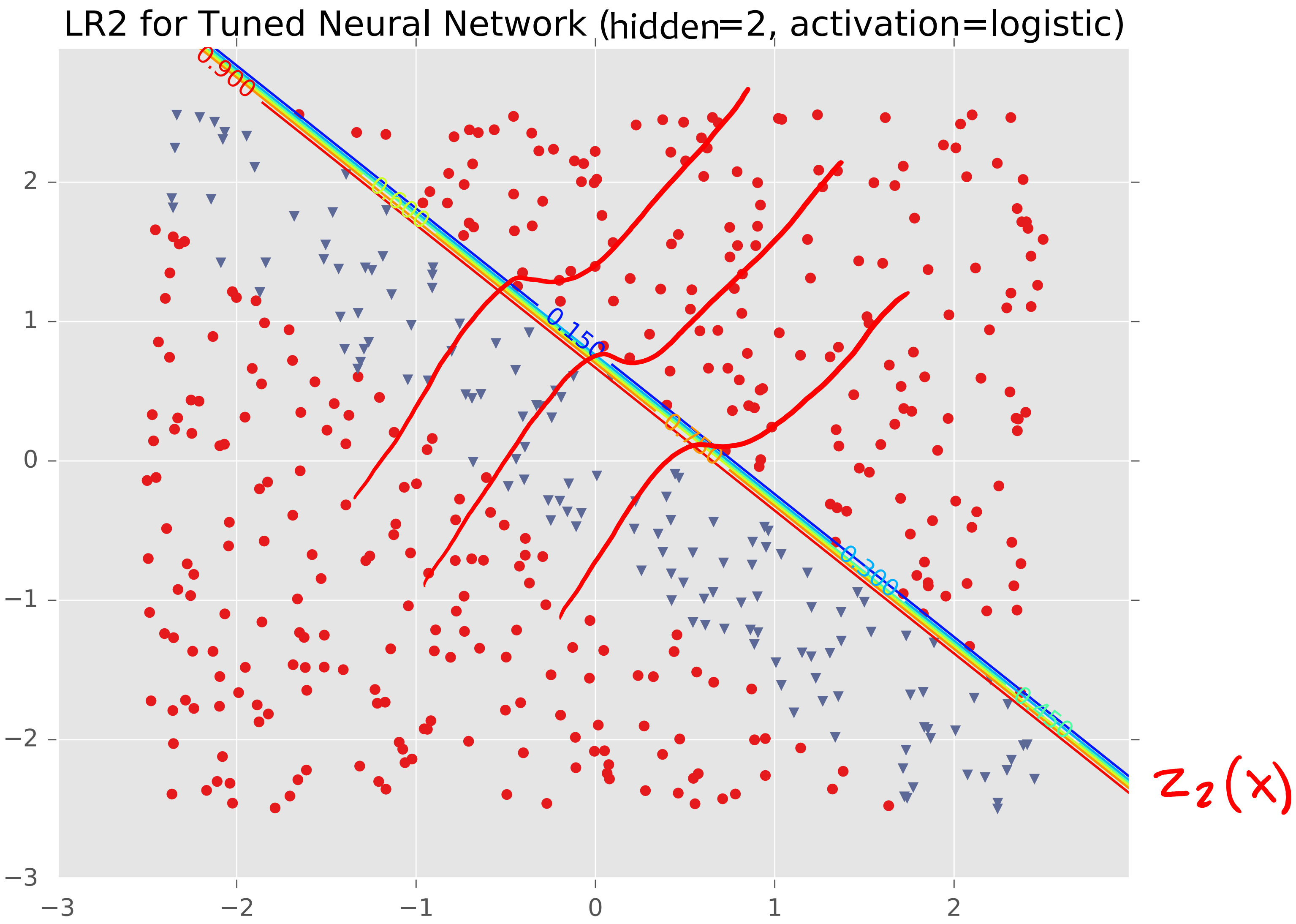
Example #1: Diagonal Band



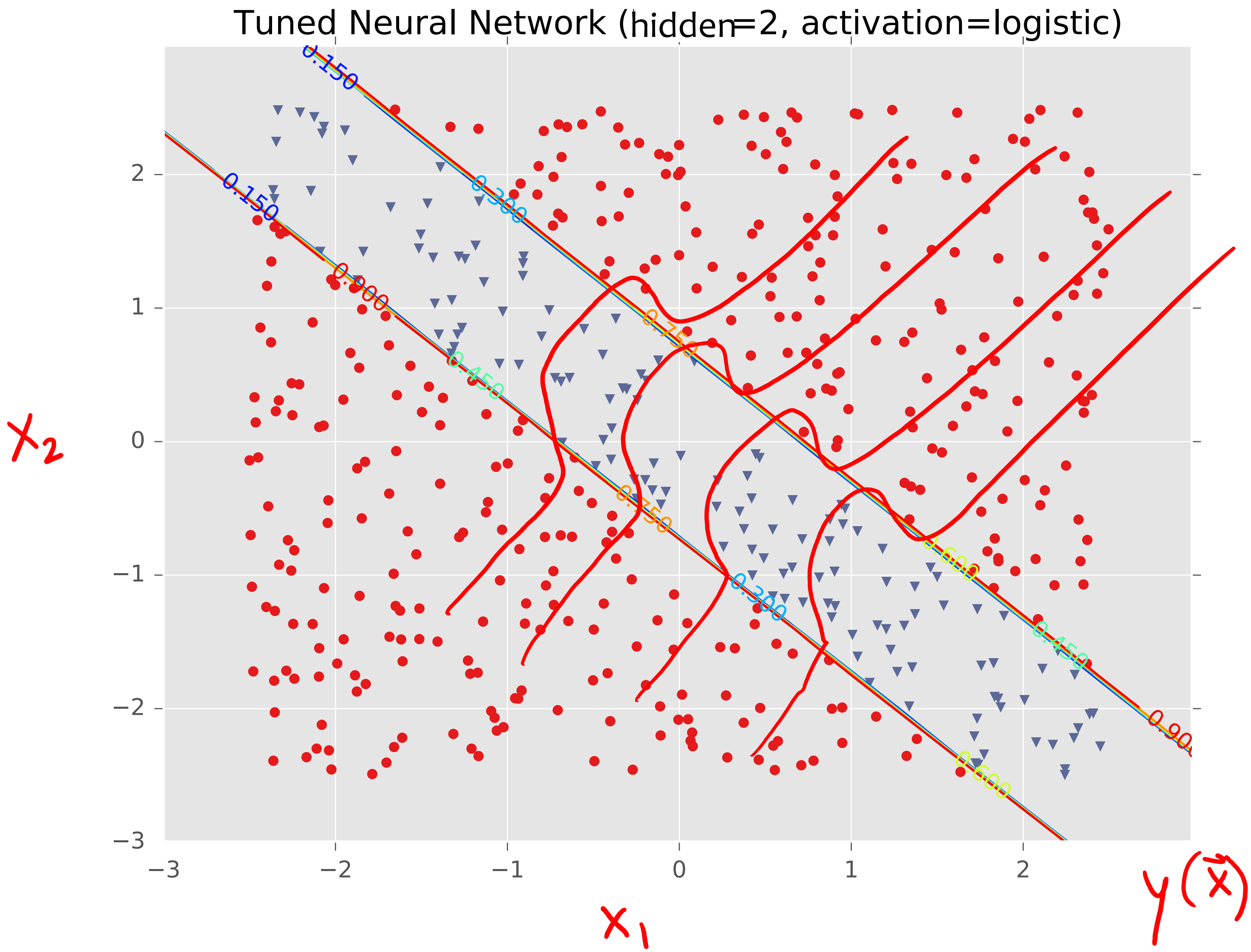
Example #1: Diagonal Band



Example #1: Diagonal Band

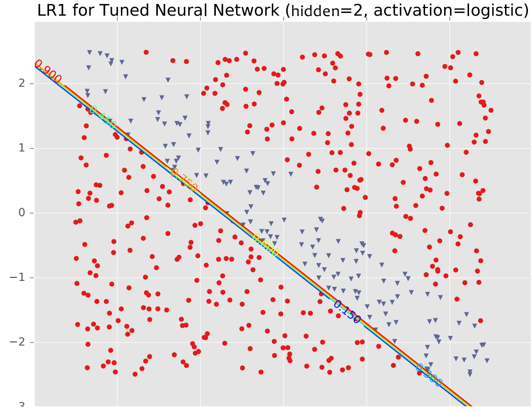


Example #1: Diagonal Band

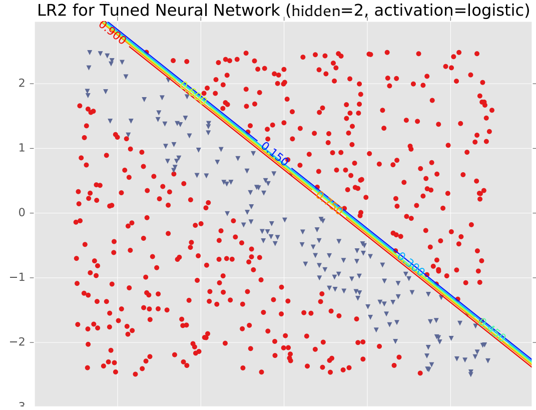


Example #1: Diagonal Band

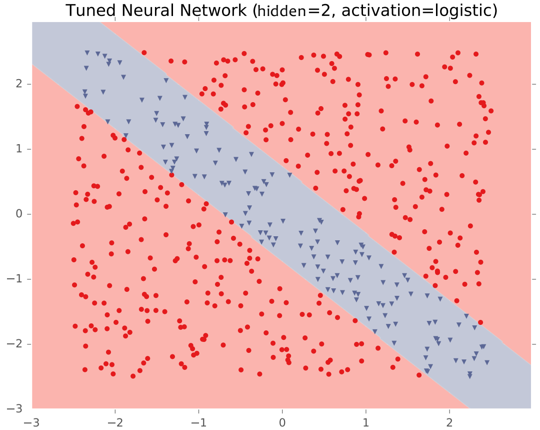
LR1 for Tuned Neural Network (hidden=2, activation=logistic)



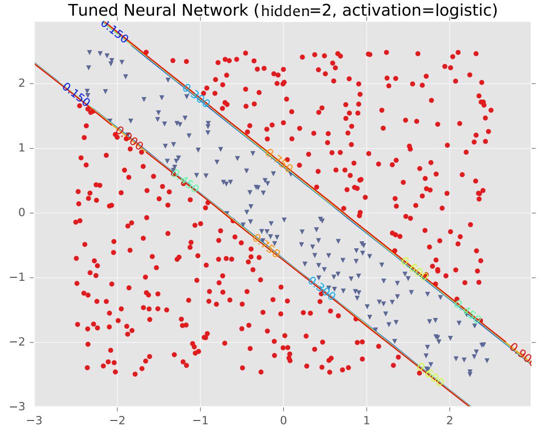
LR2 for Tuned Neural Network (hidden=2, activation=logistic)



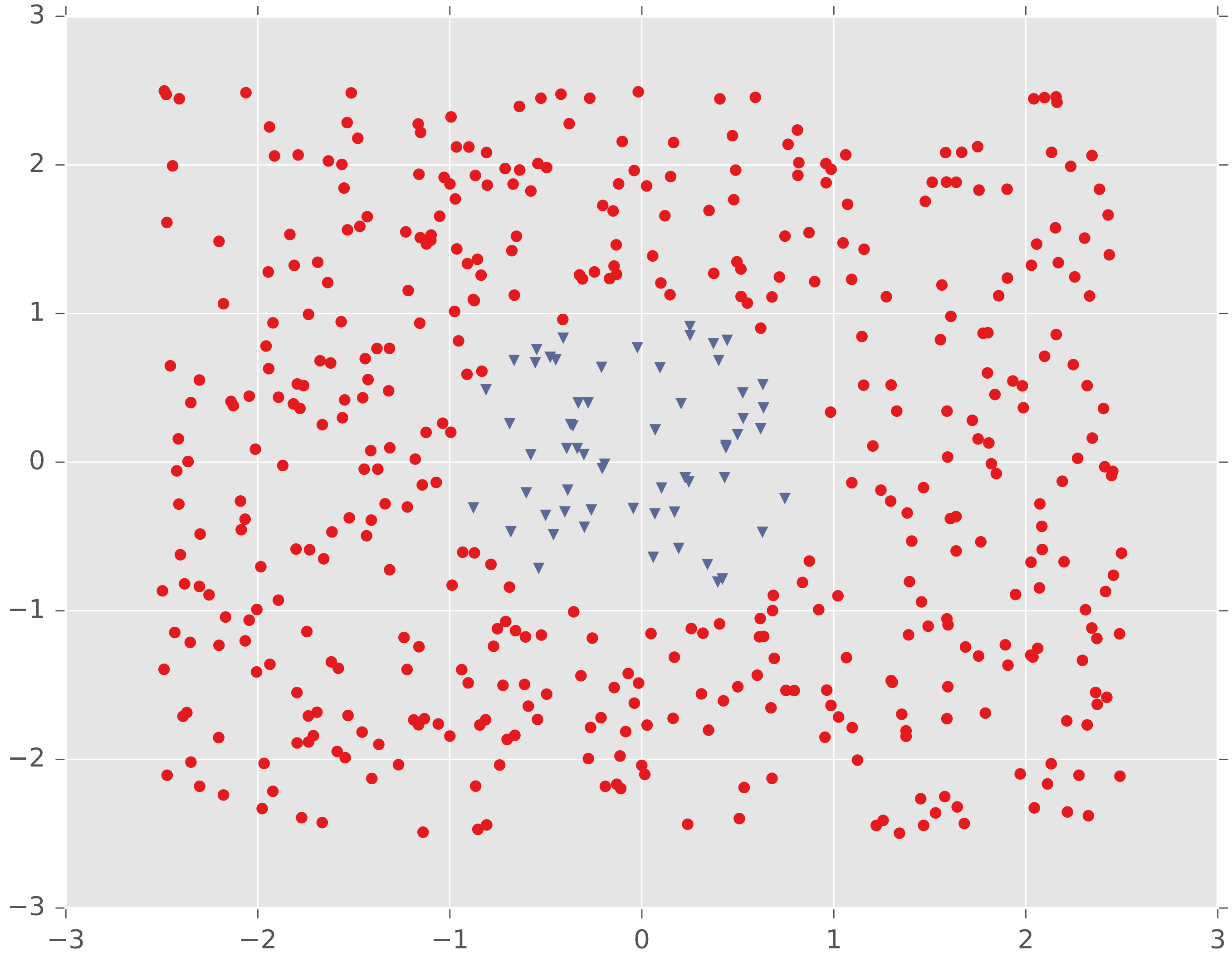
Tuned Neural Network (hidden=2, activation=logistic)



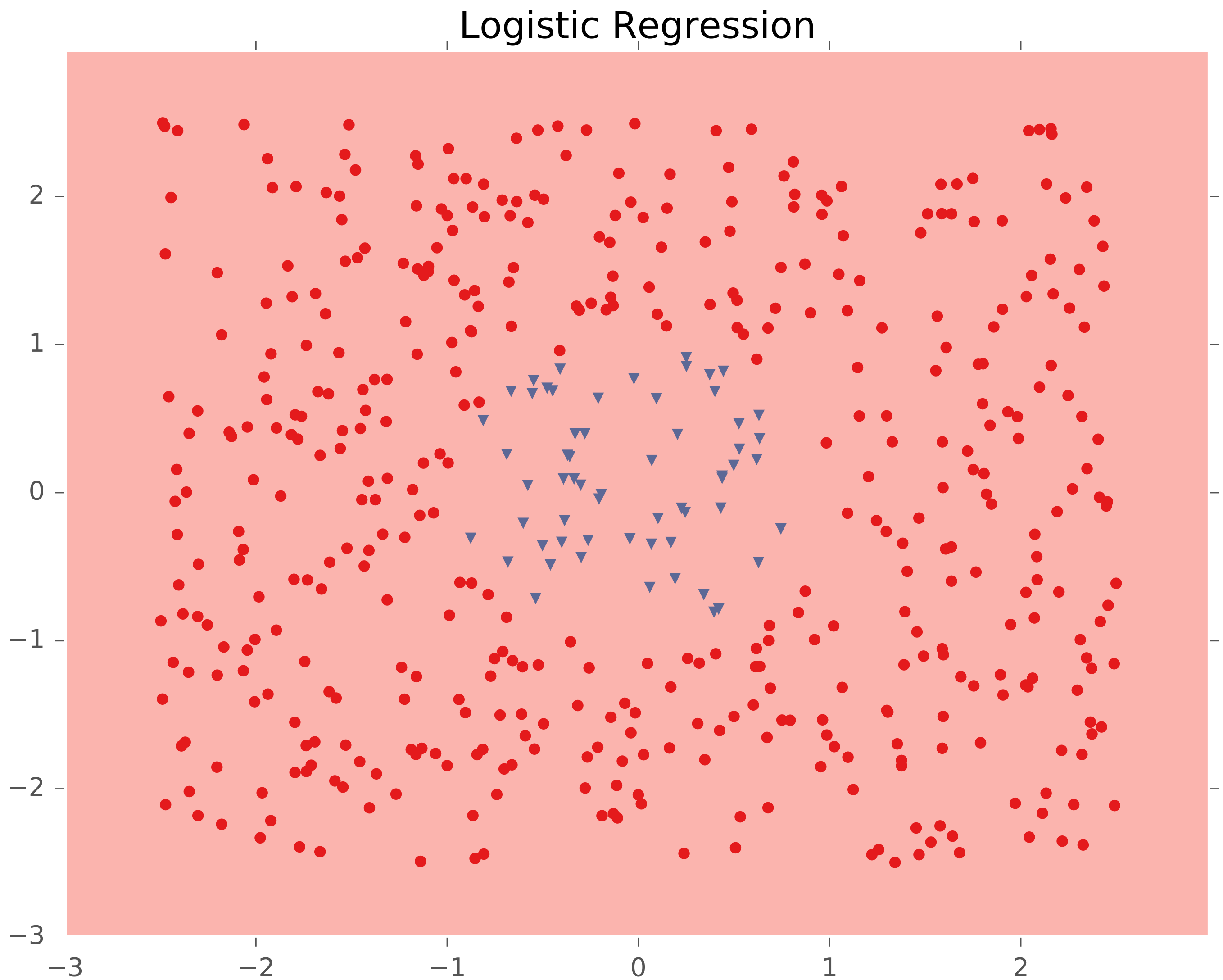
Tuned Neural Network (hidden=2, activation=logistic)



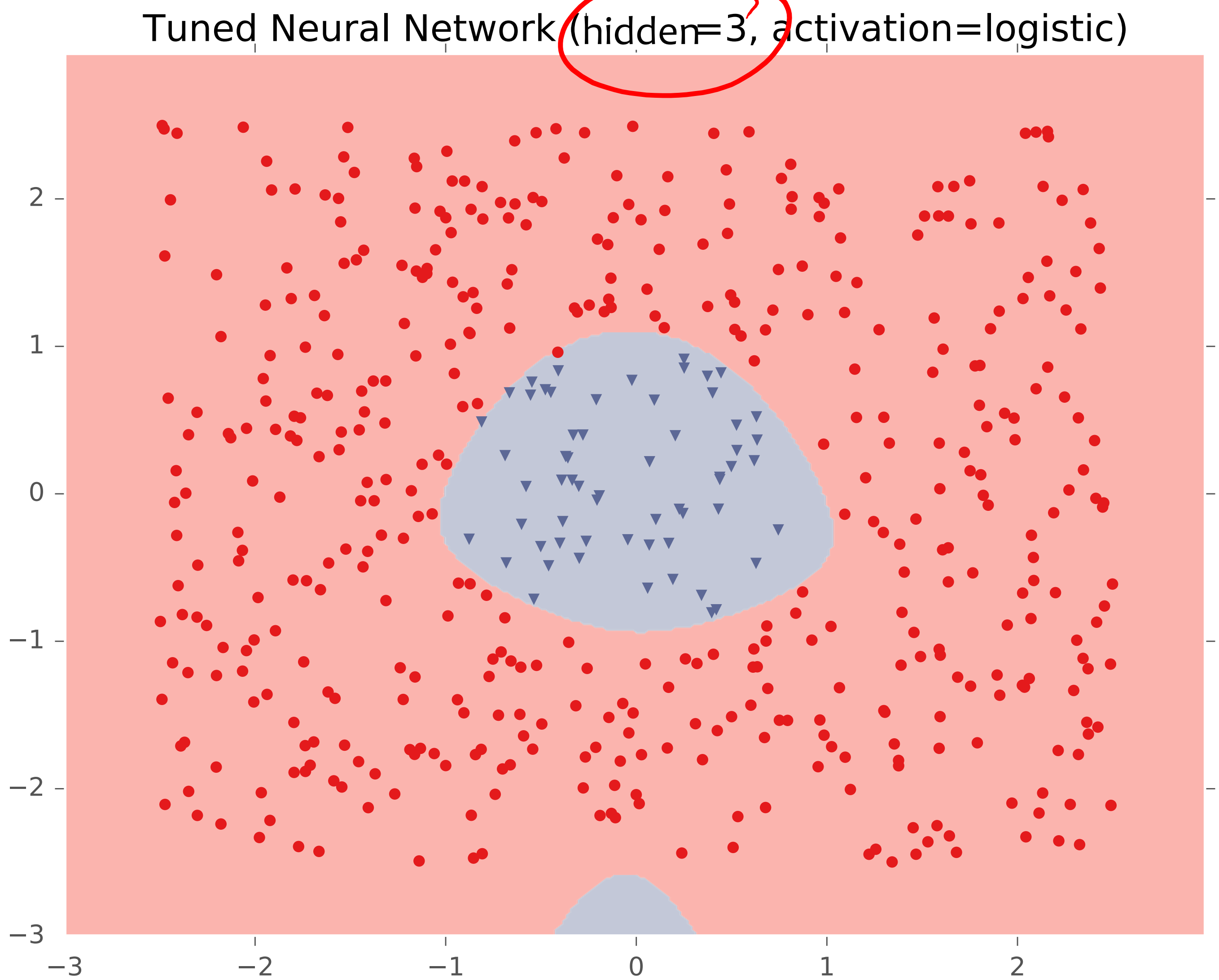
Example #2: One Pocket



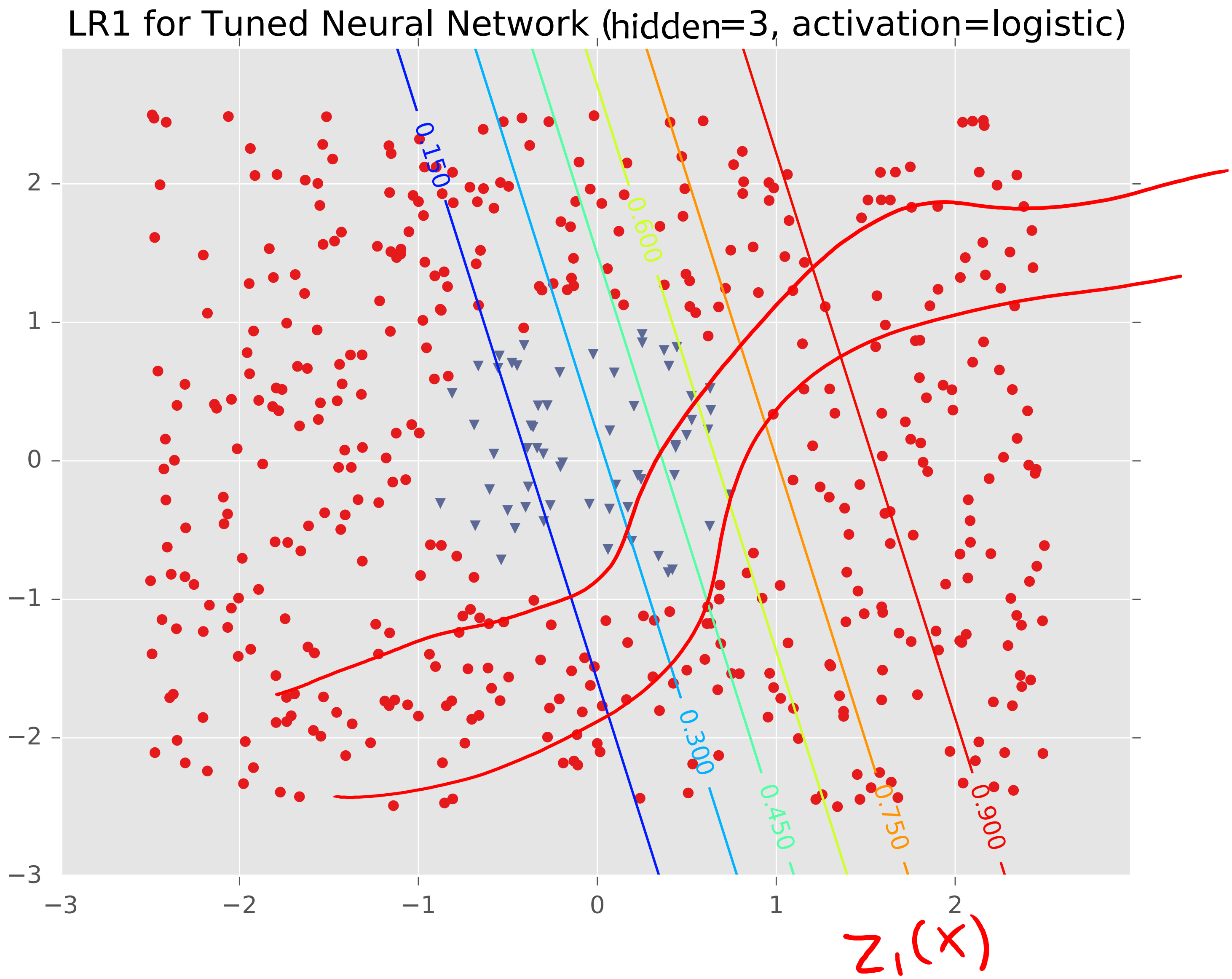
Example #2: One Pocket



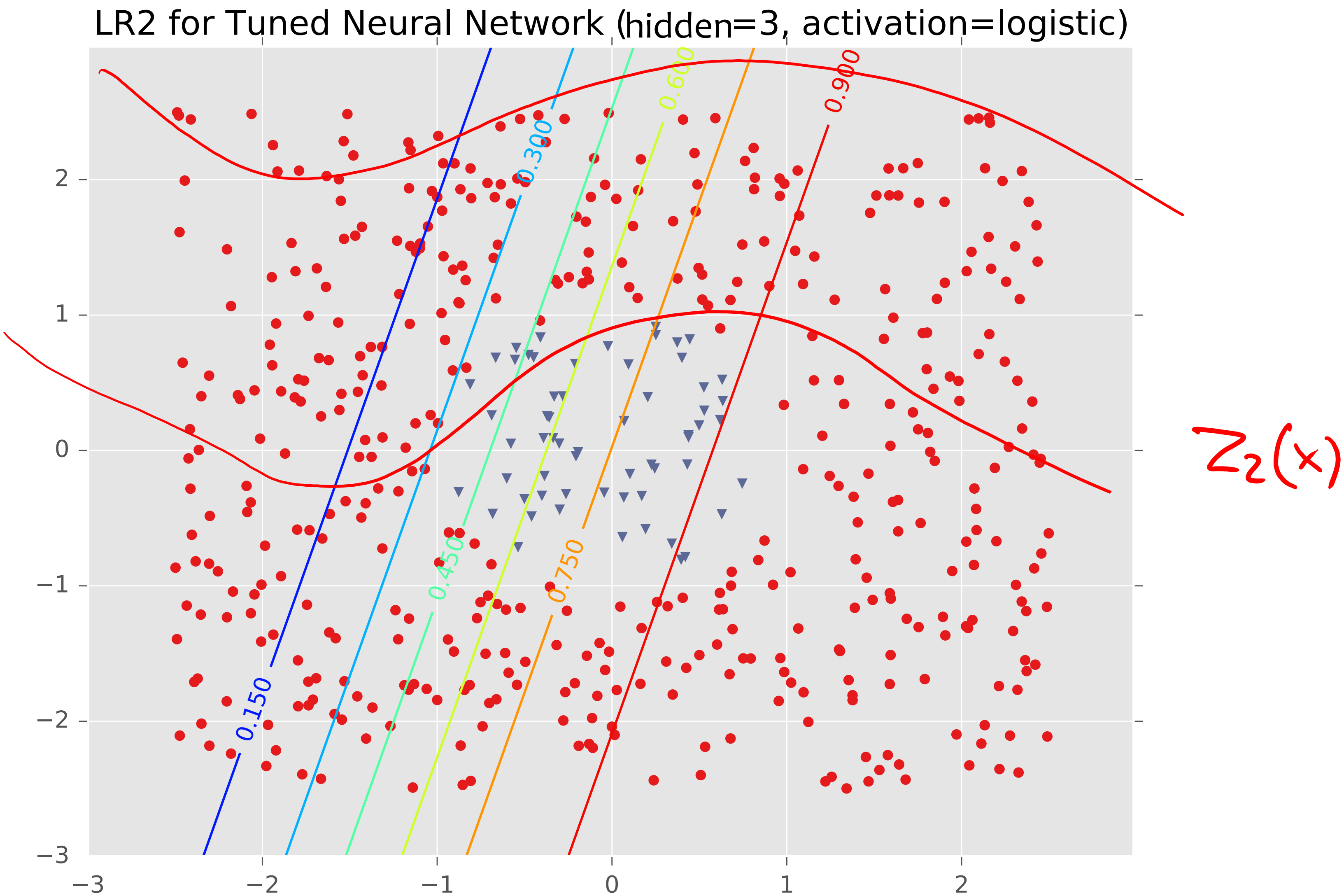
Example #2: One Pocket



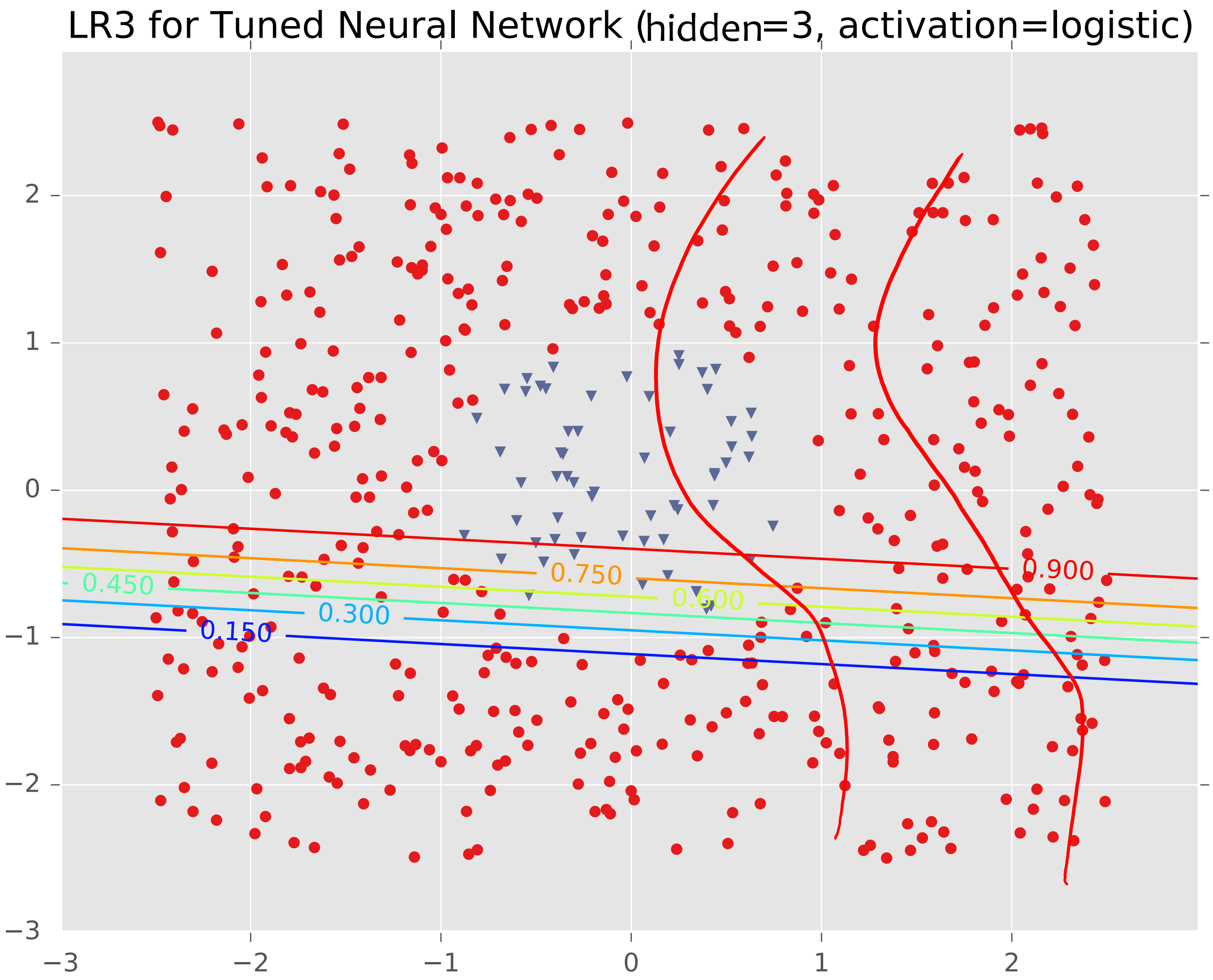
Example #2: One Pocket



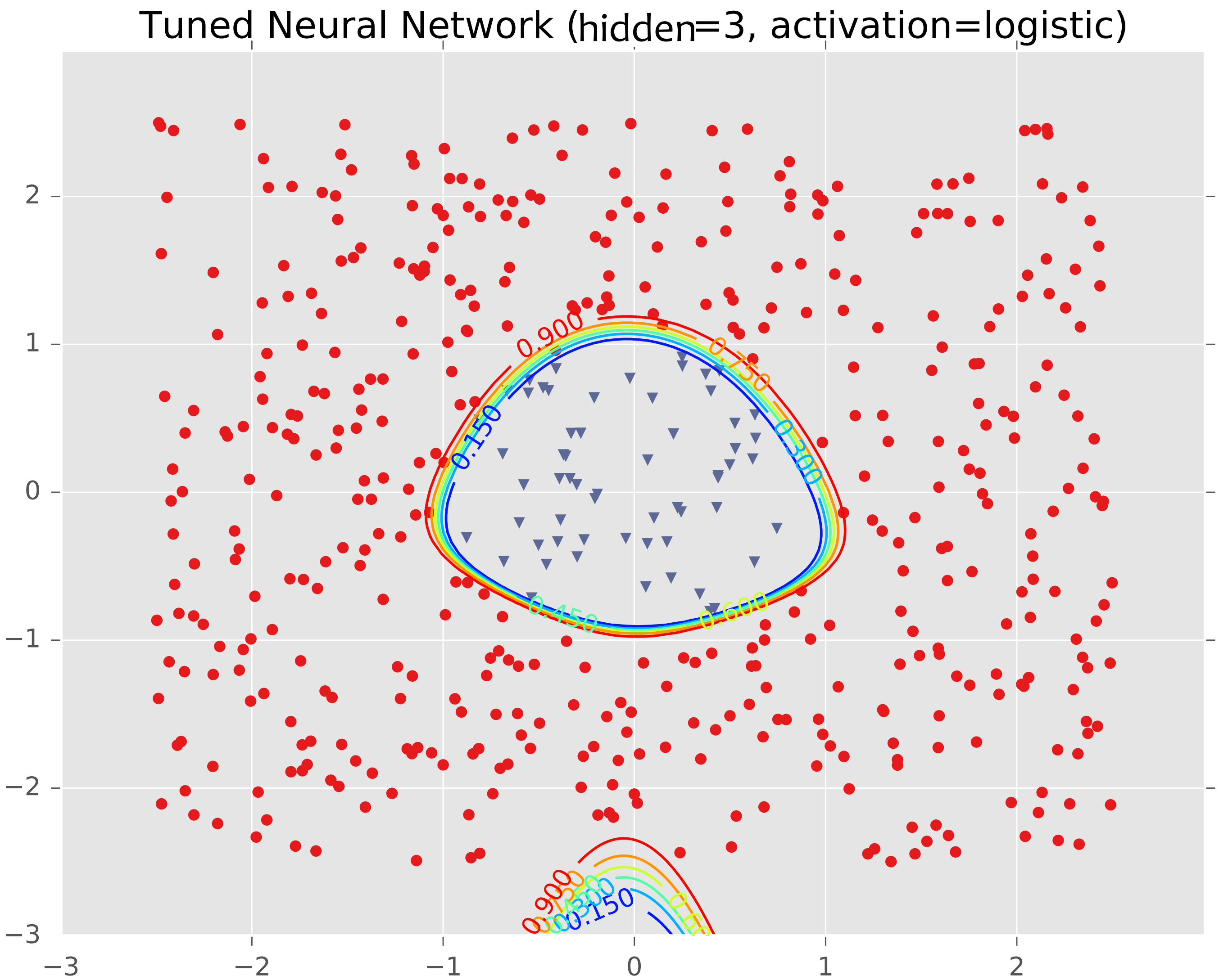
Example #2: One Pocket



Example #2: One Pocket

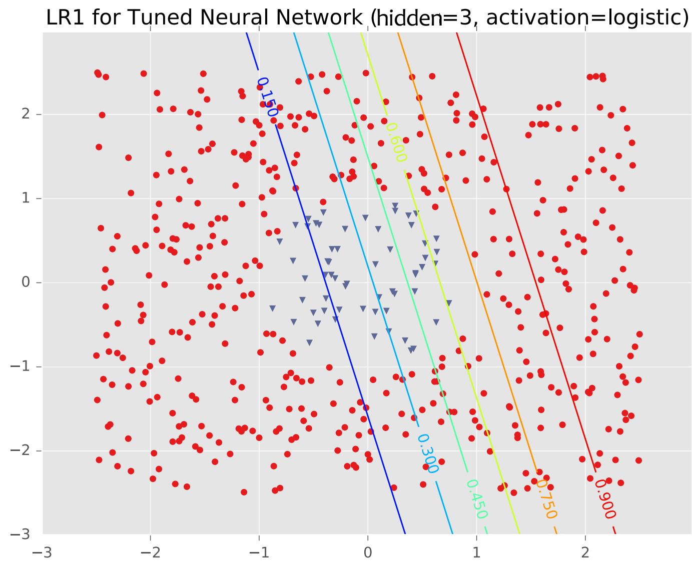


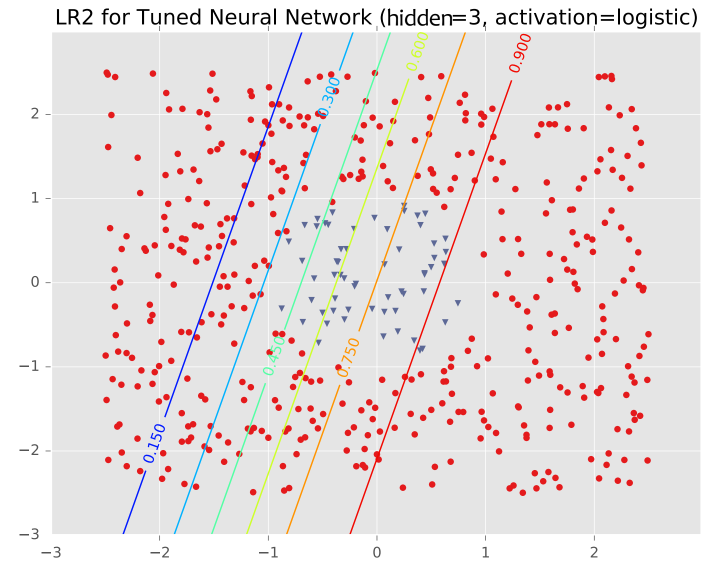
Example #2: One Pocket

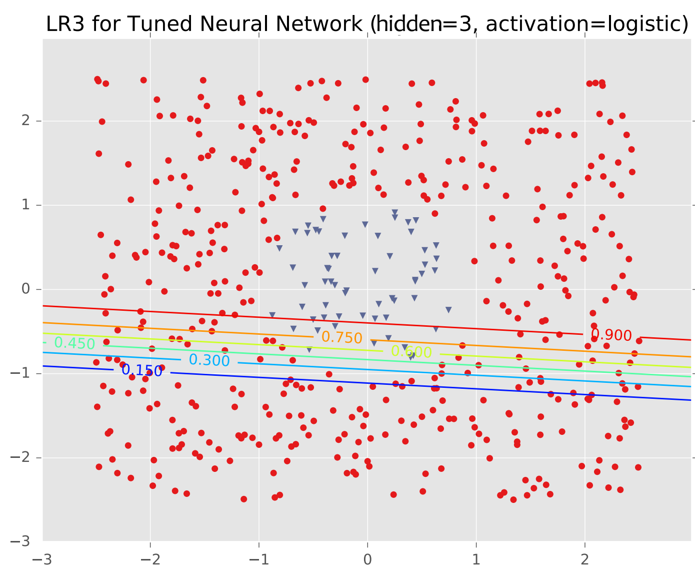


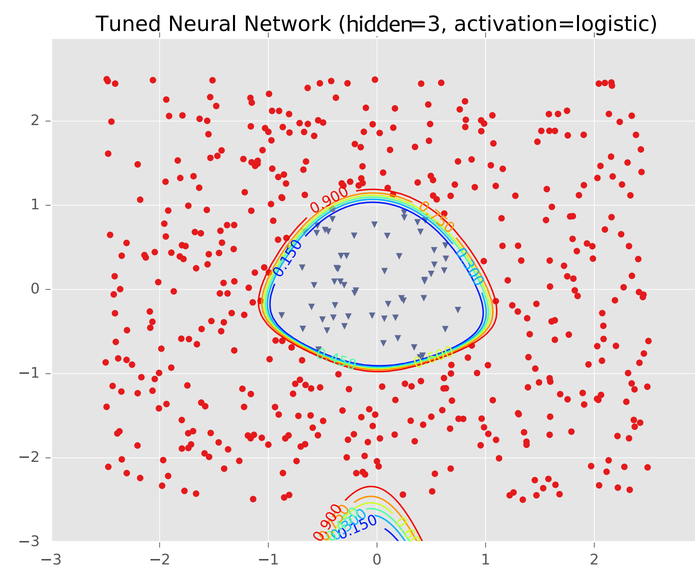
$y(x)$

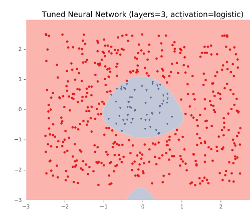
Example #2: One Pocket







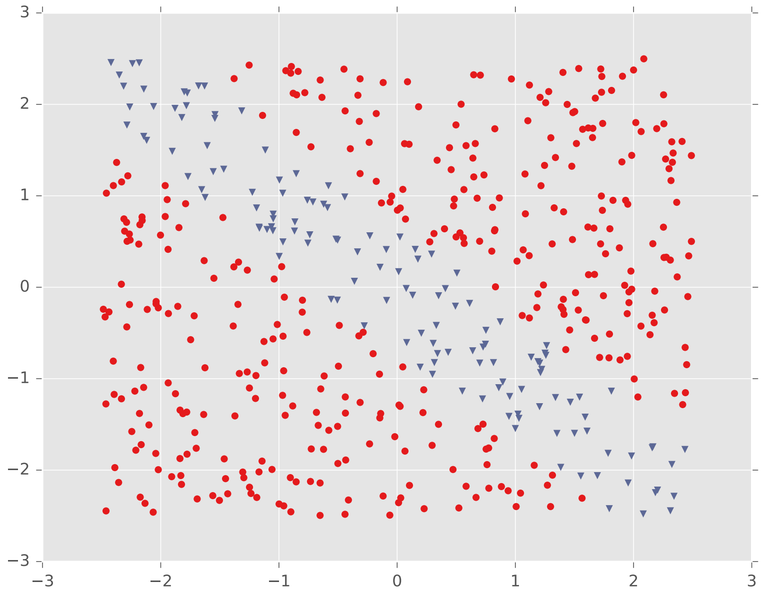




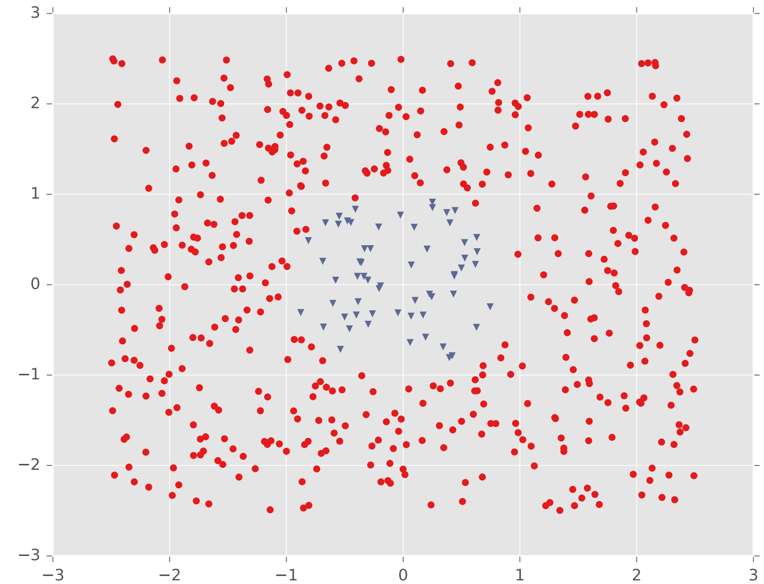
Examples 3 and 4

DECISION BOUNDARY EXAMPLES

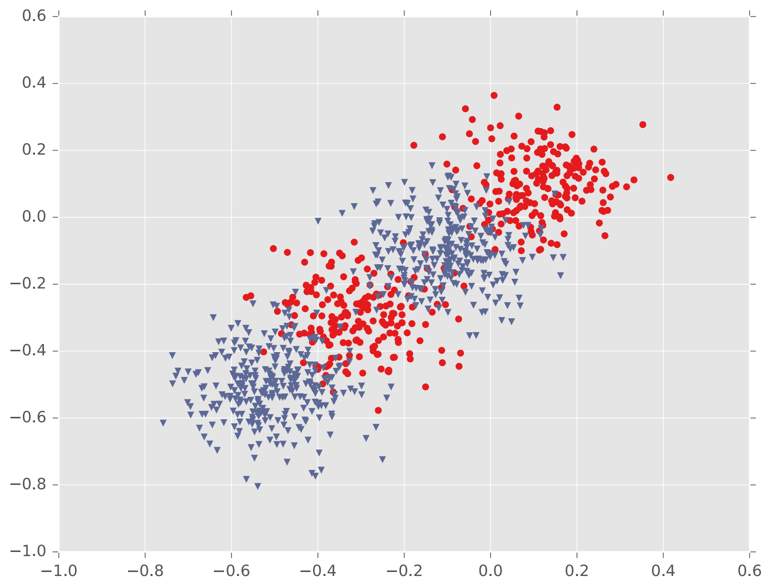
Example #1: Diagonal Band



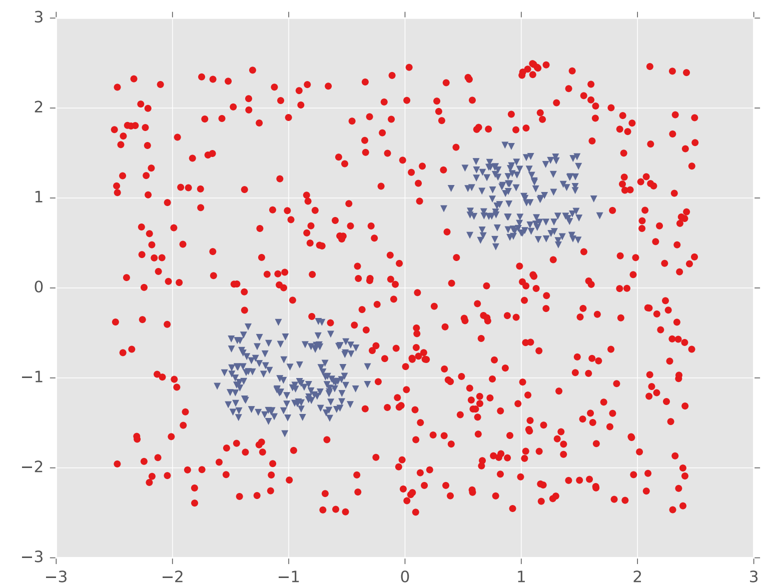
Example #2: One Pocket



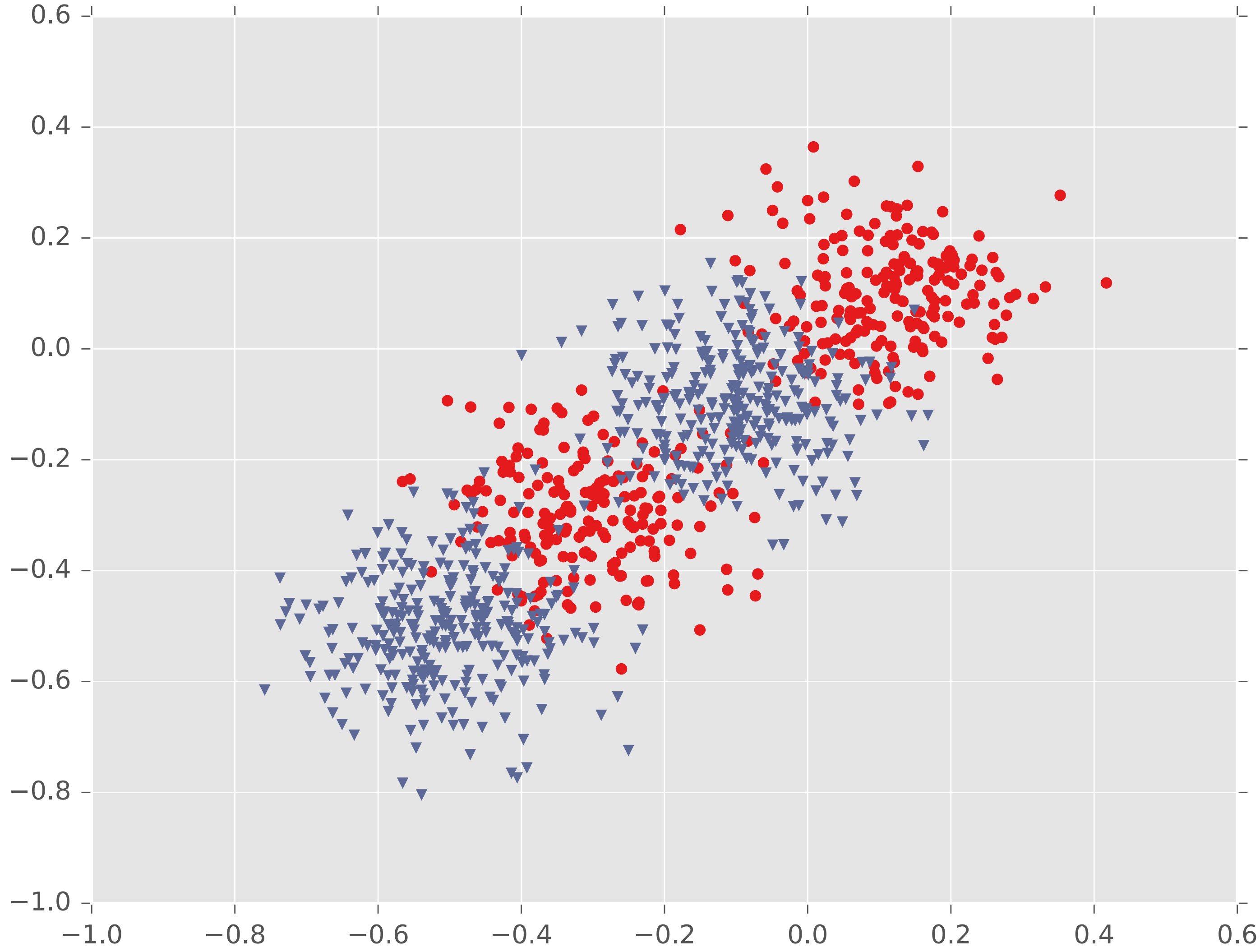
Example #3: Four Gaussians



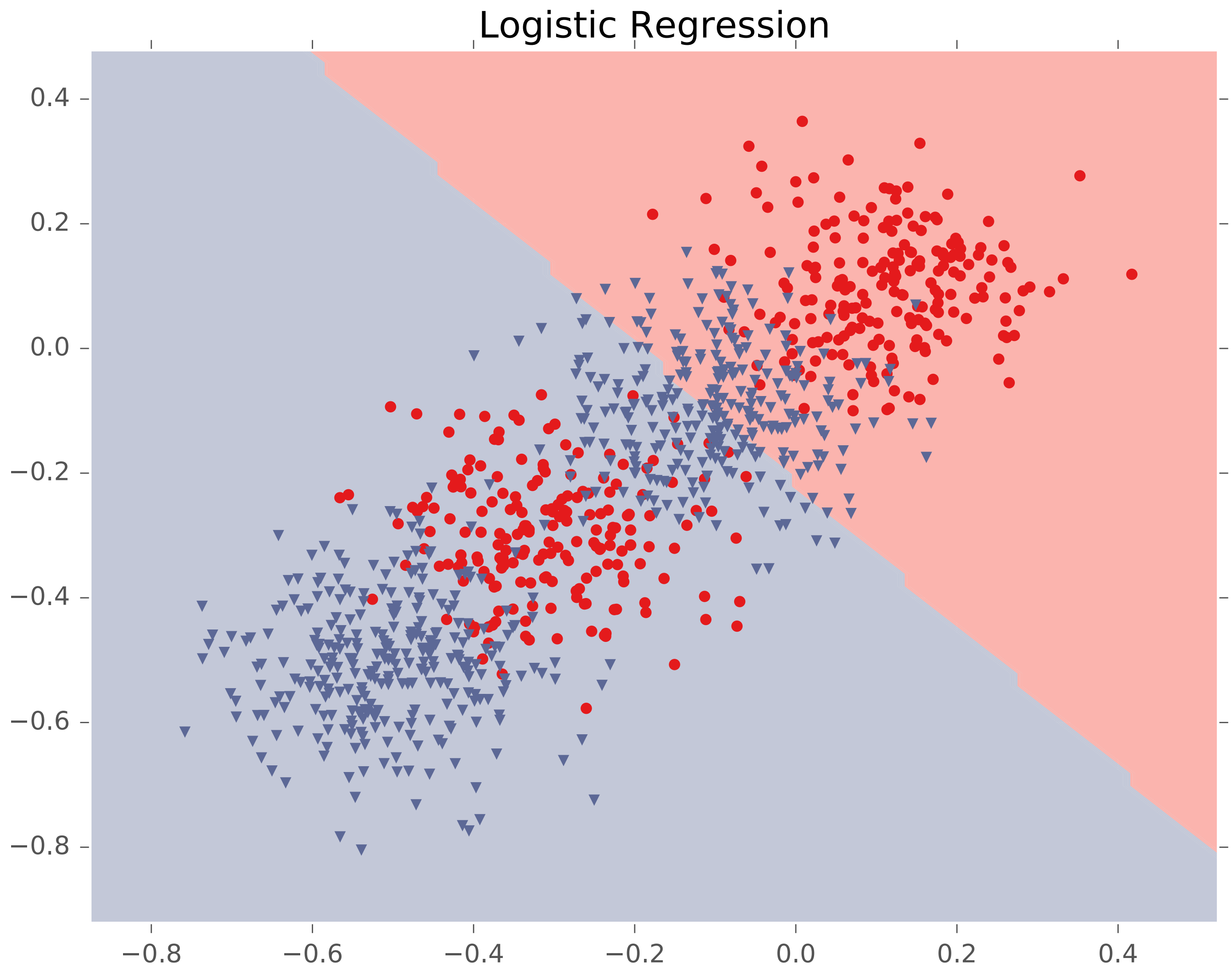
Example #4: Two Pockets



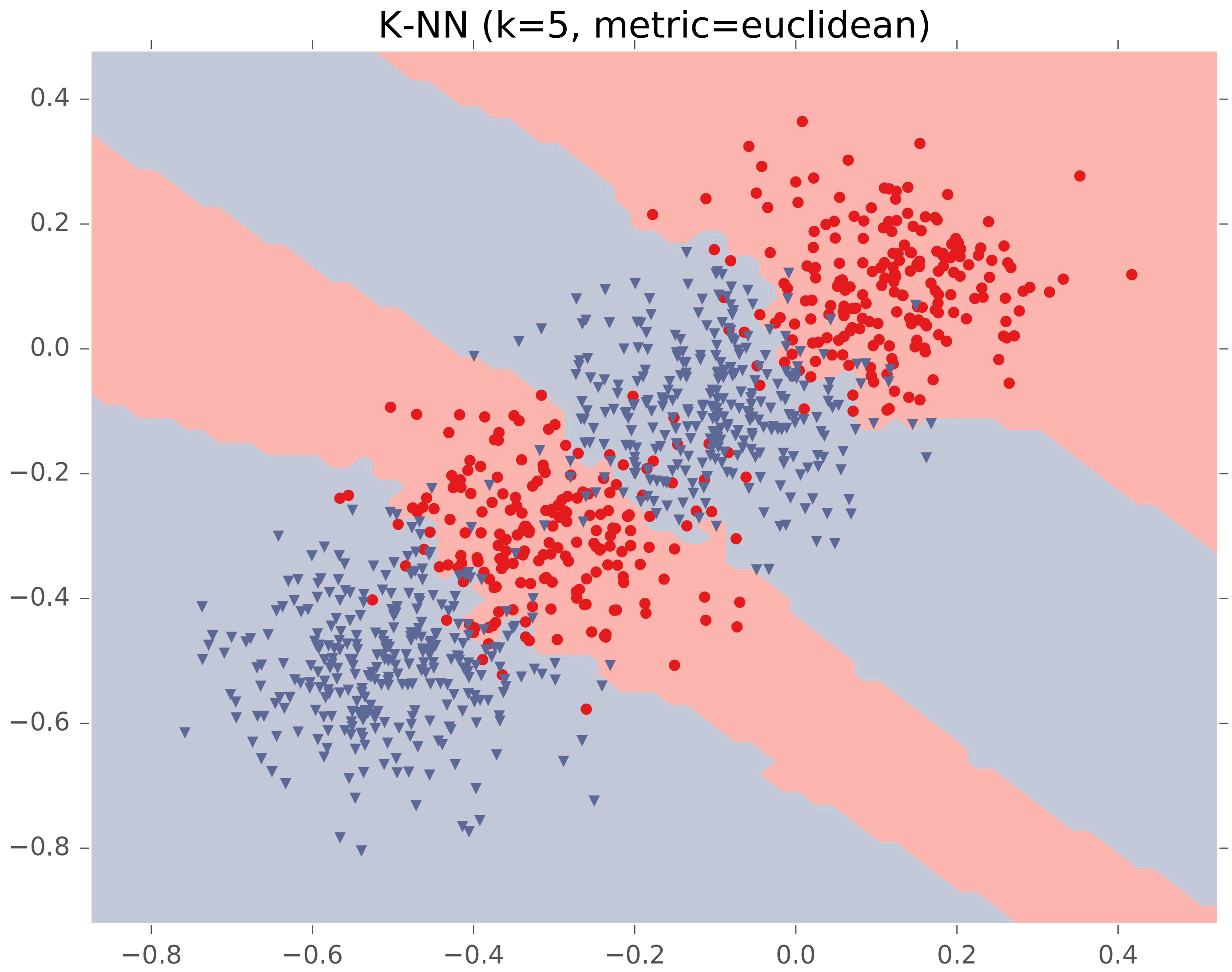
Example #3: Four Gaussians



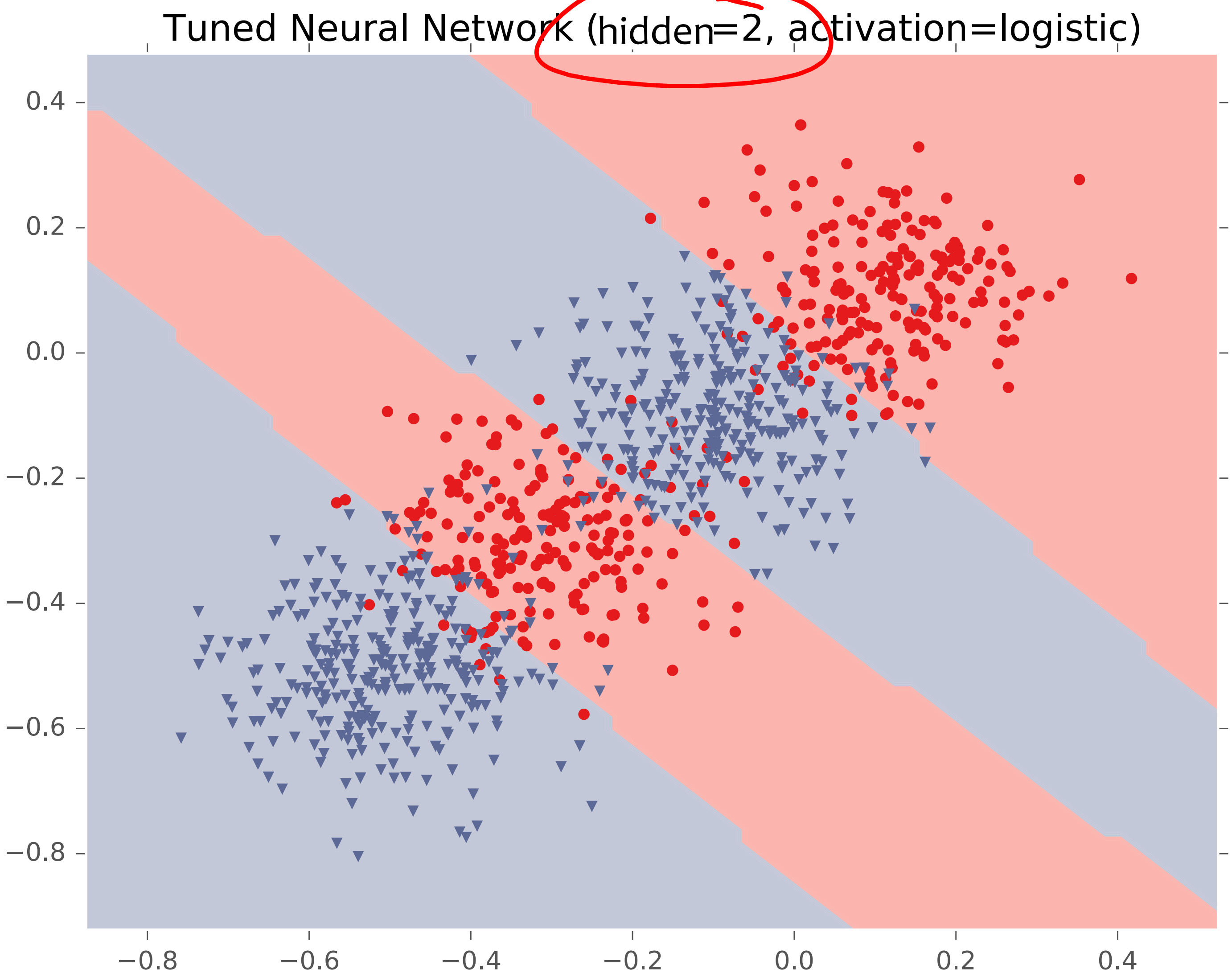
Example #3: Four Gaussians



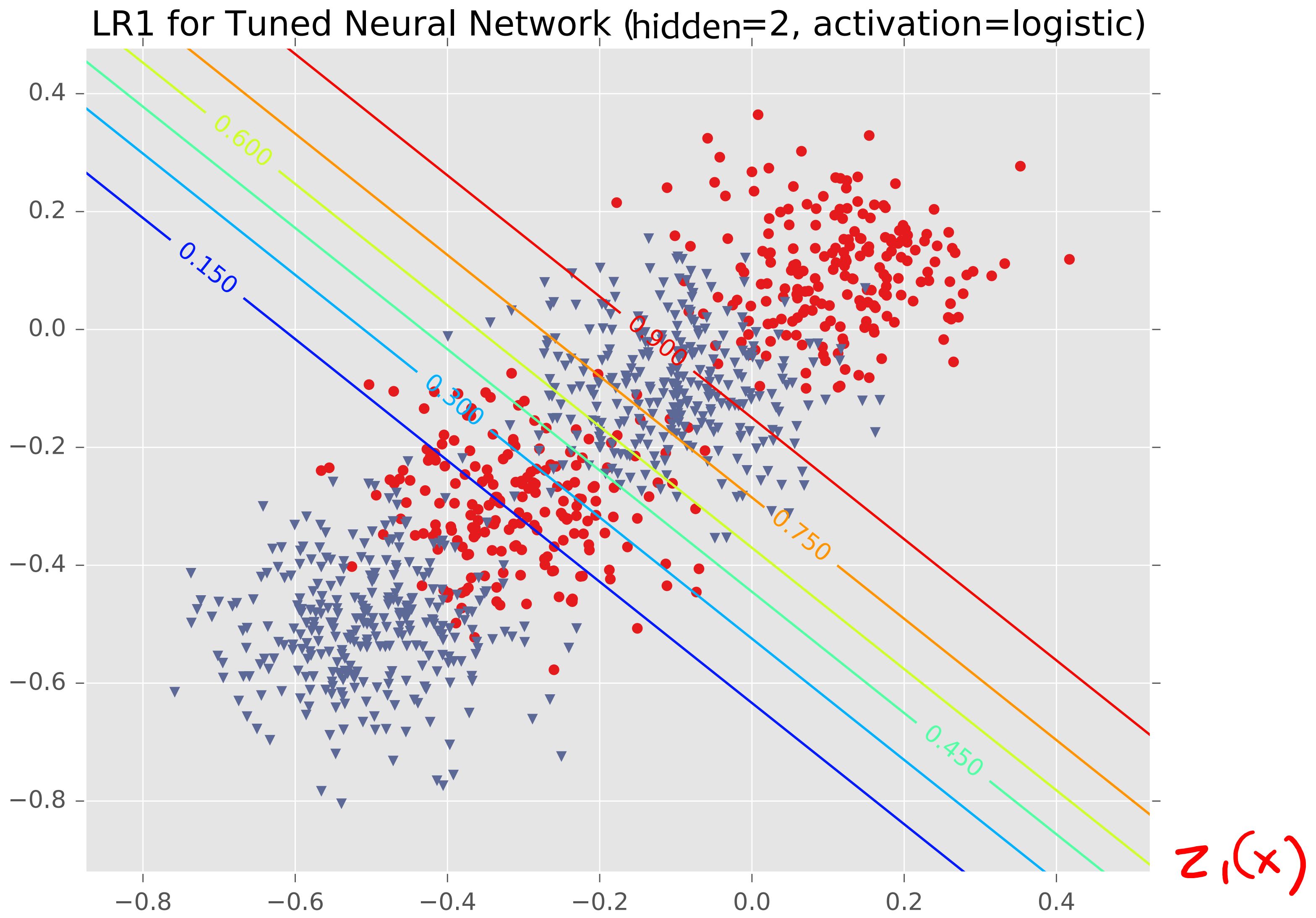
Example #3: Four Gaussians



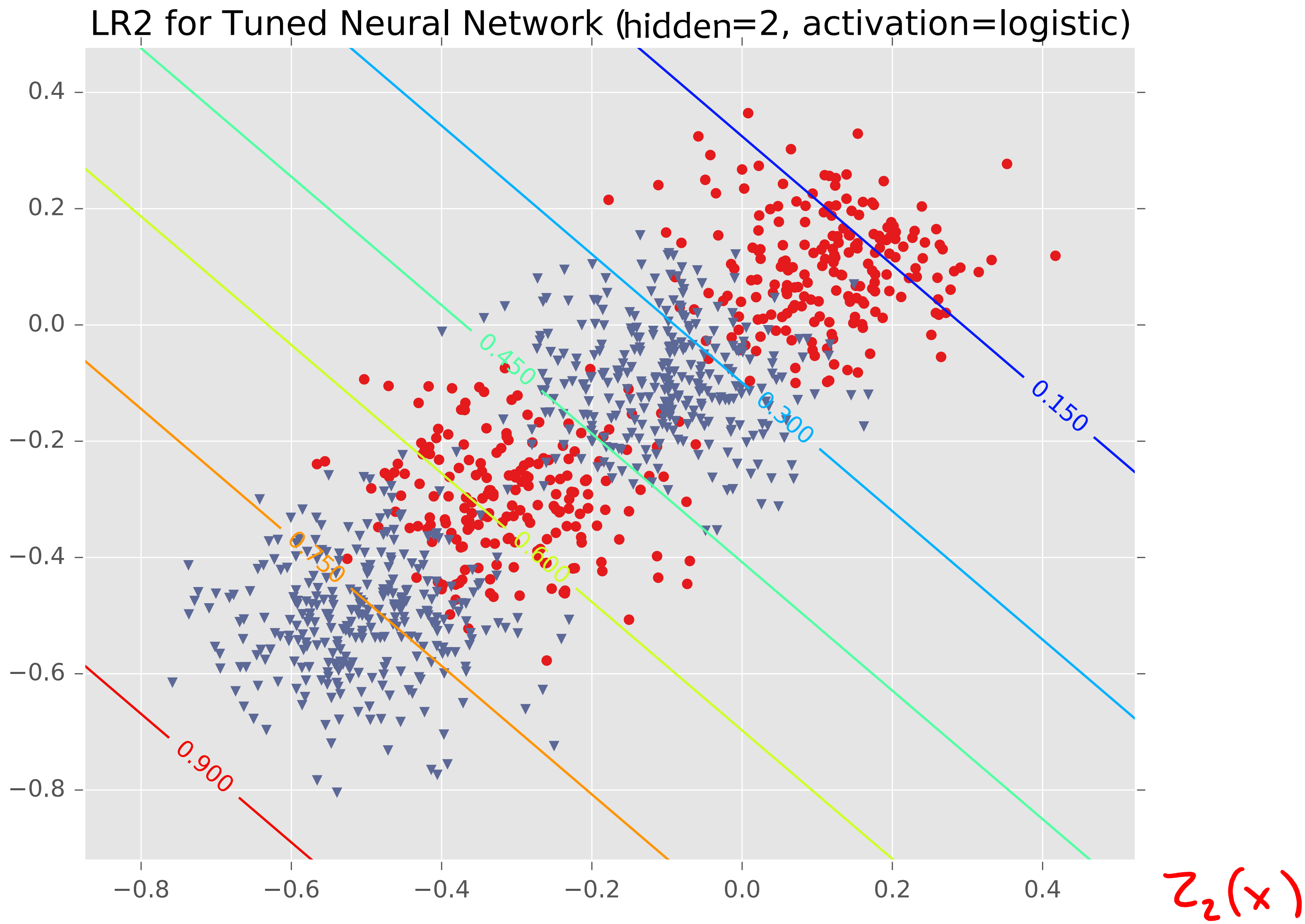
Example #3: Four Gaussians



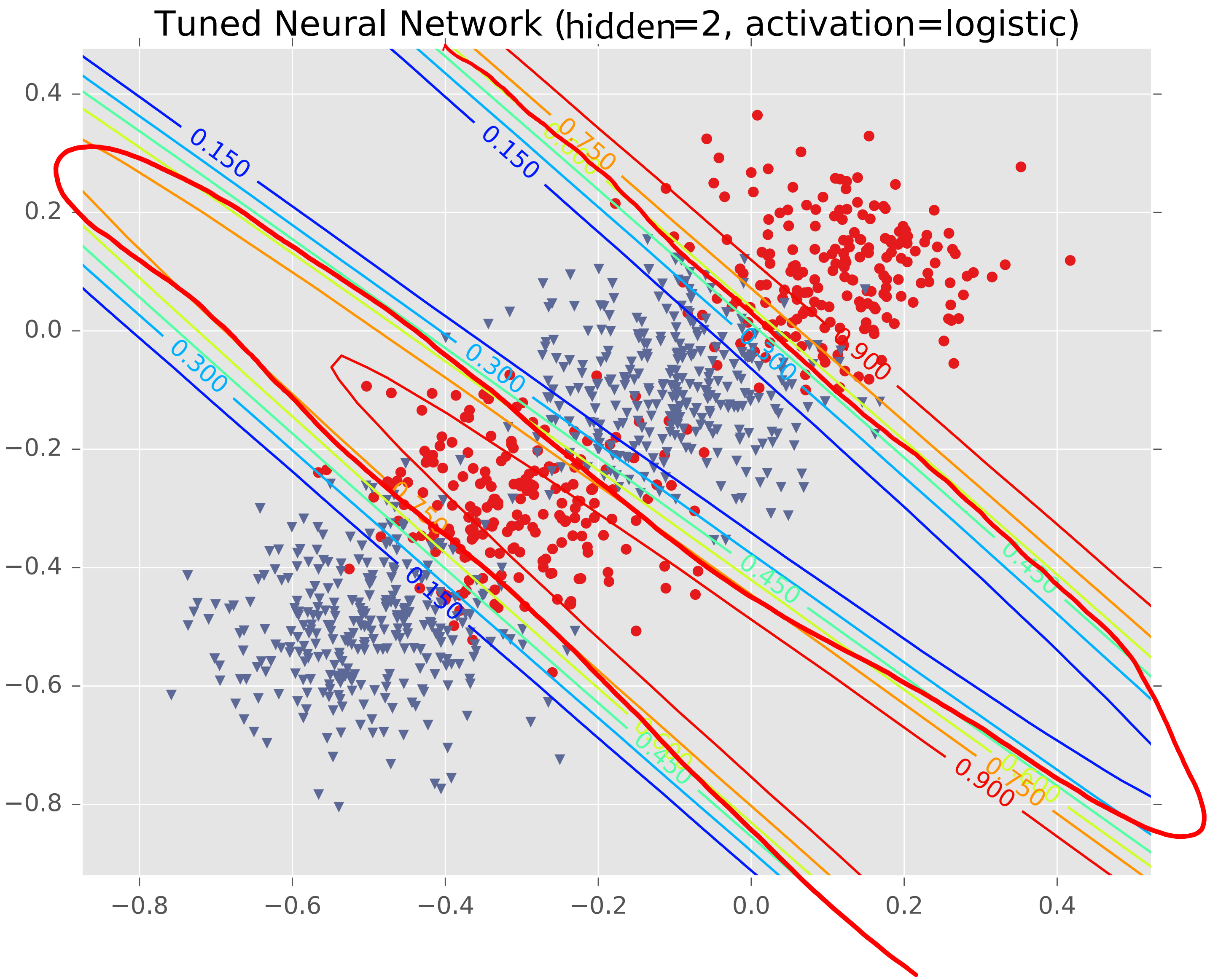
Example #3: Four Gaussians



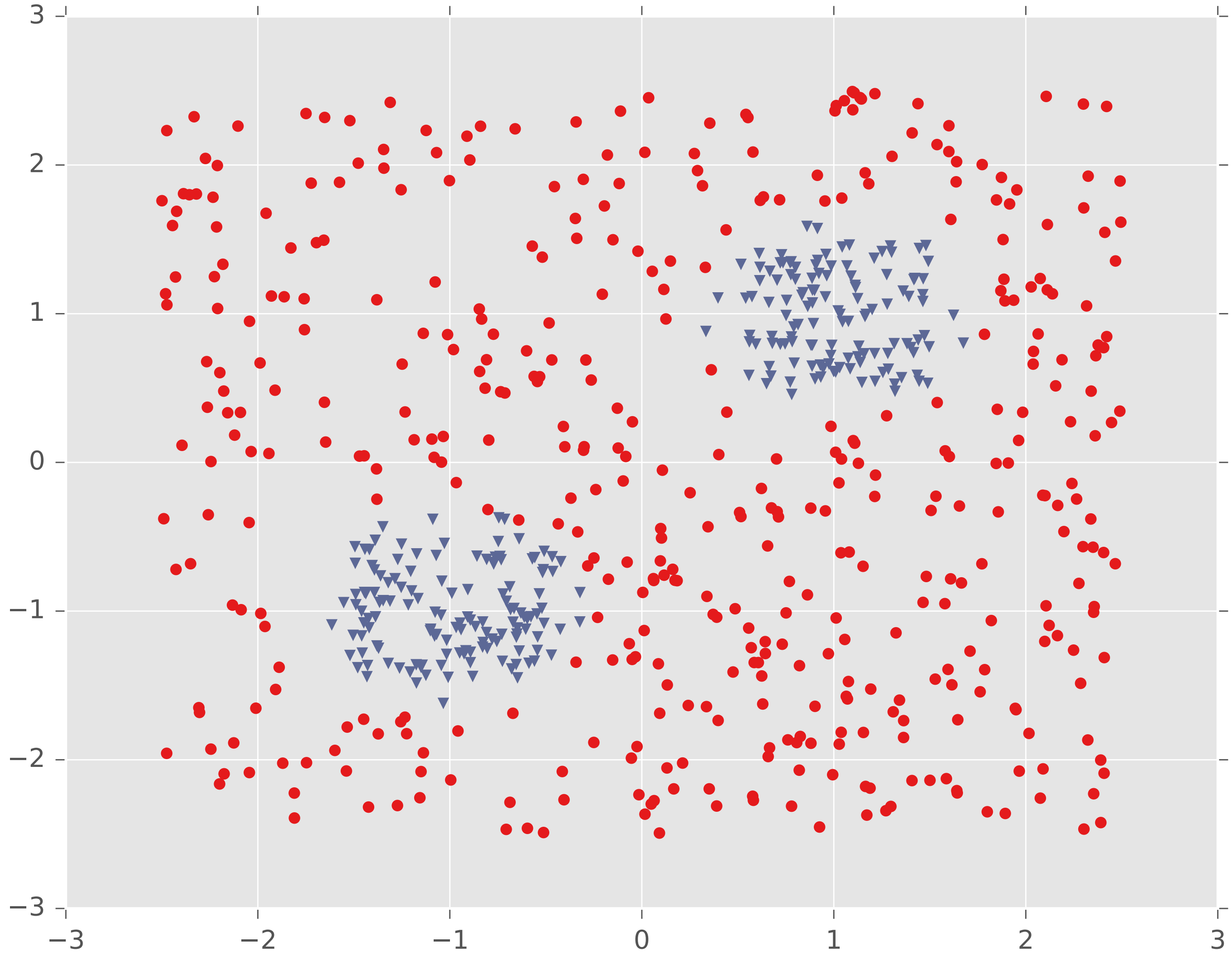
Example #3: Four Gaussians



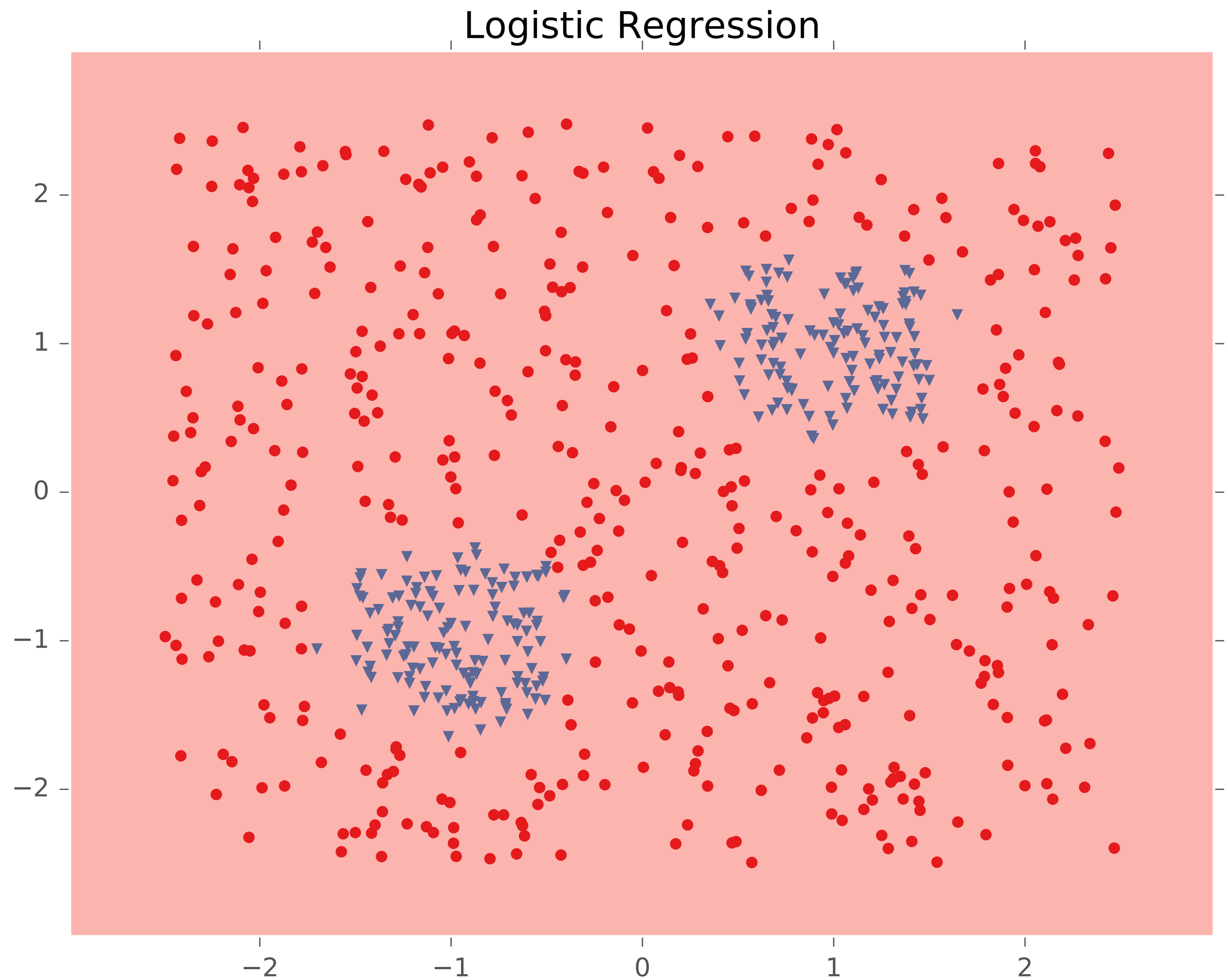
Example #3: Four Gaussians



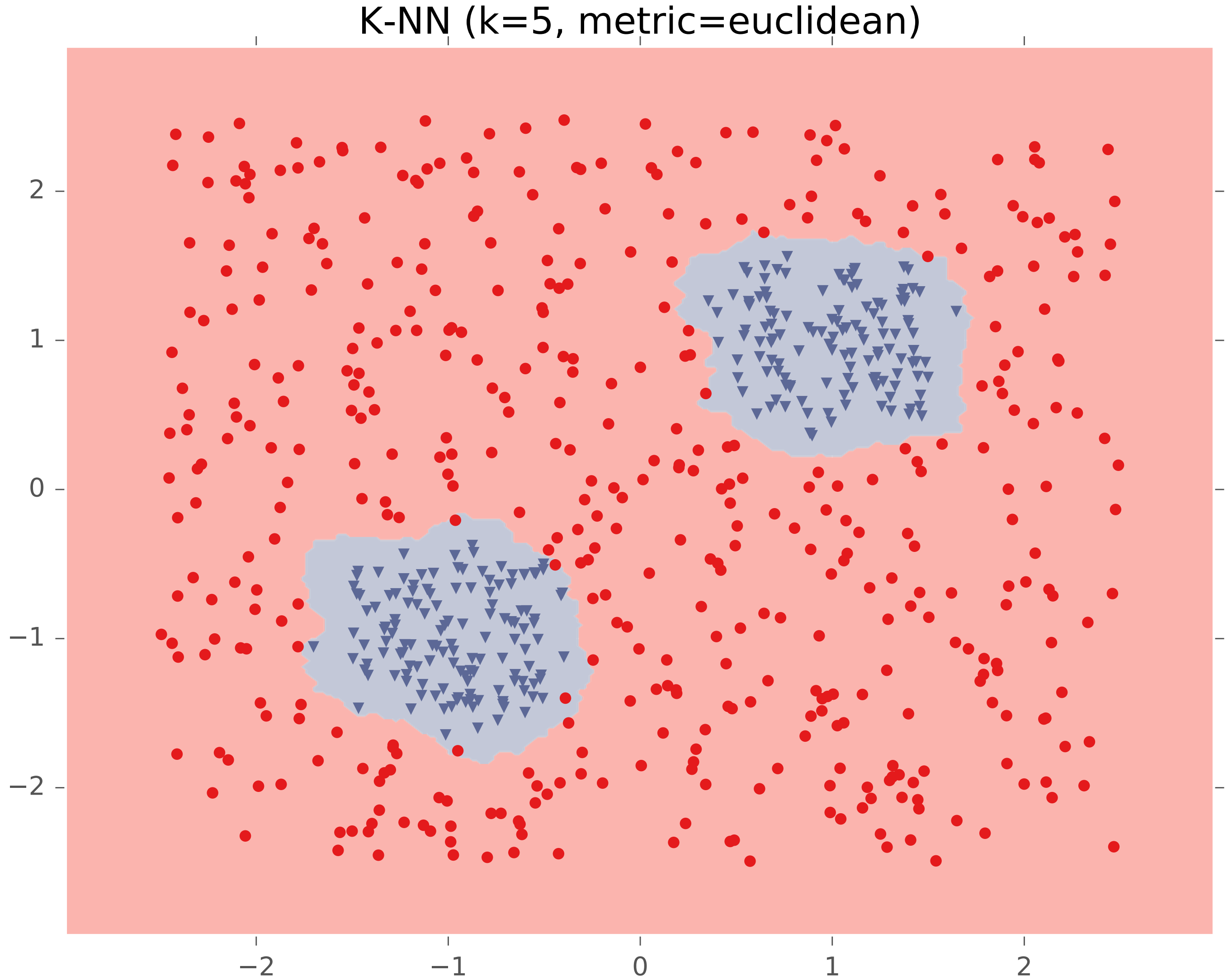
Example #4: Two Pockets



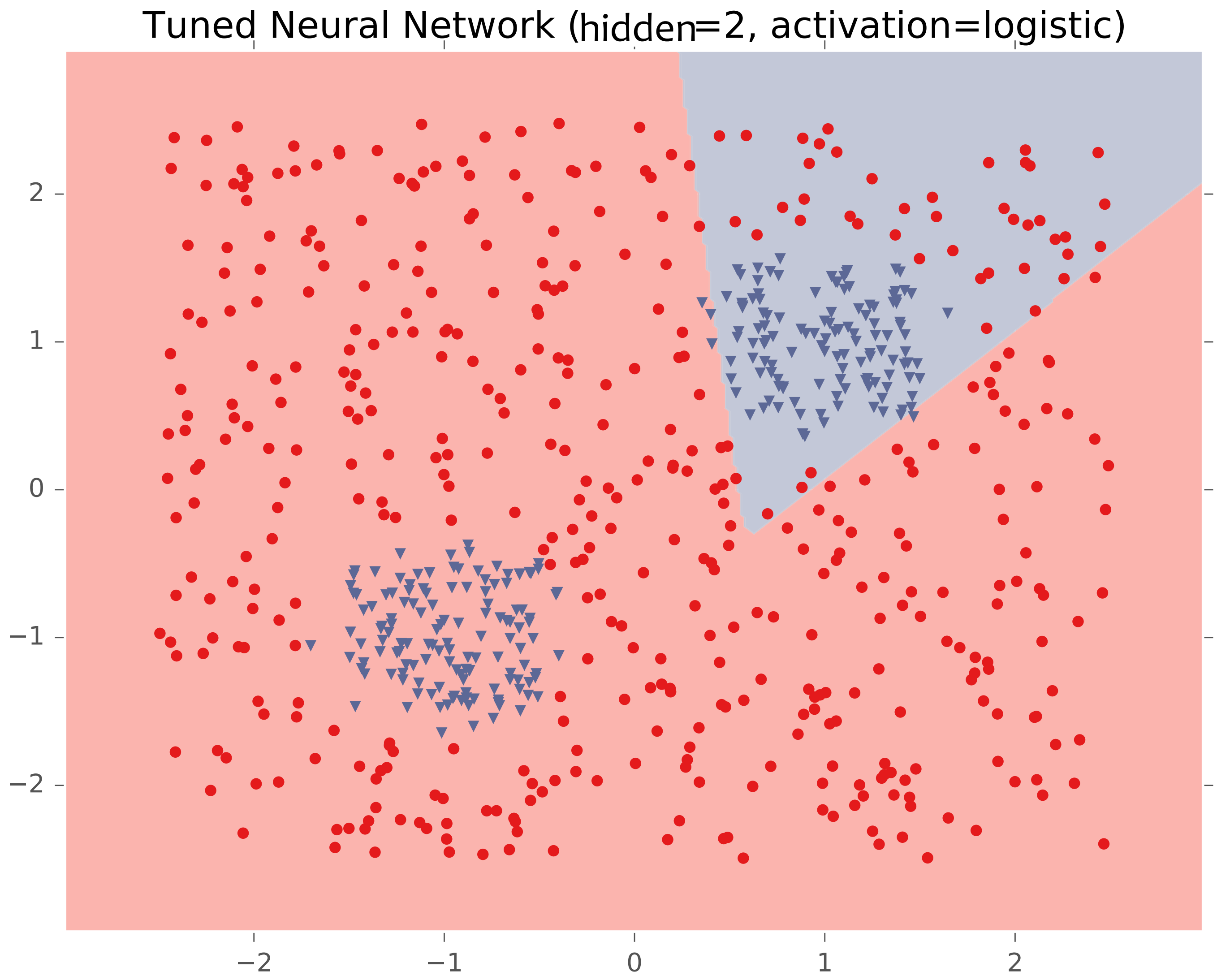
Example #4: Two Pockets



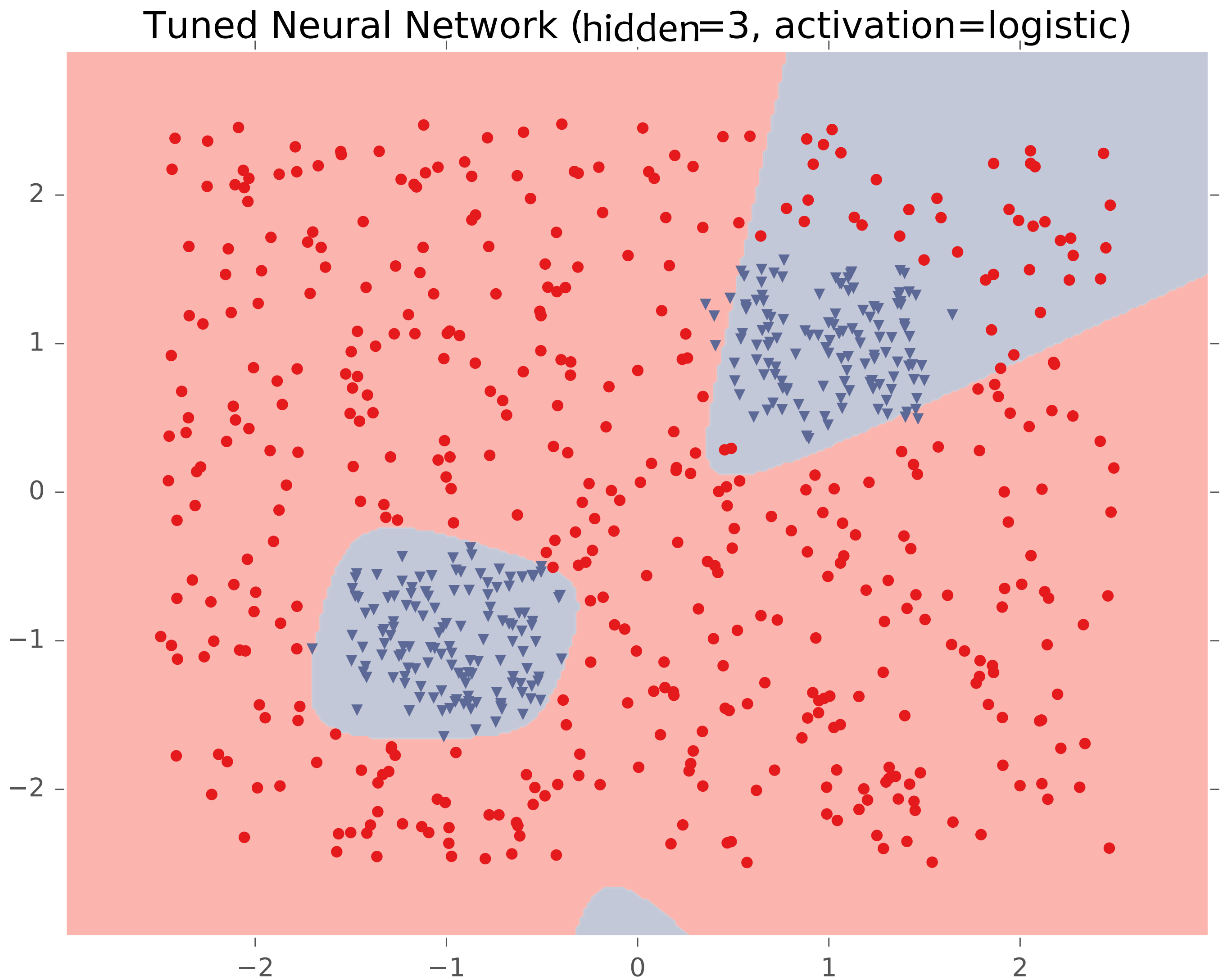
Example #4: Two Pockets



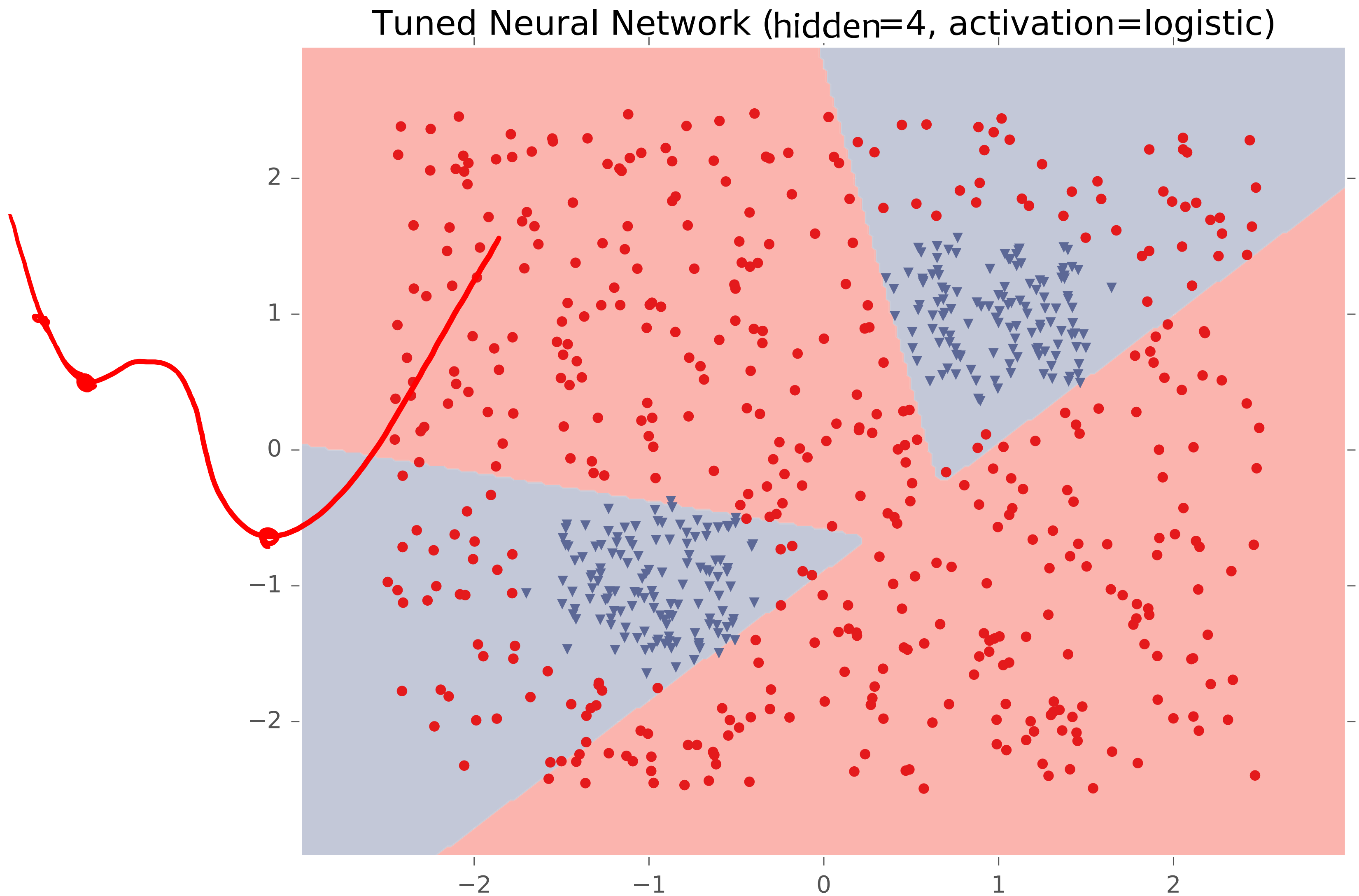
Example #4: Two Pockets



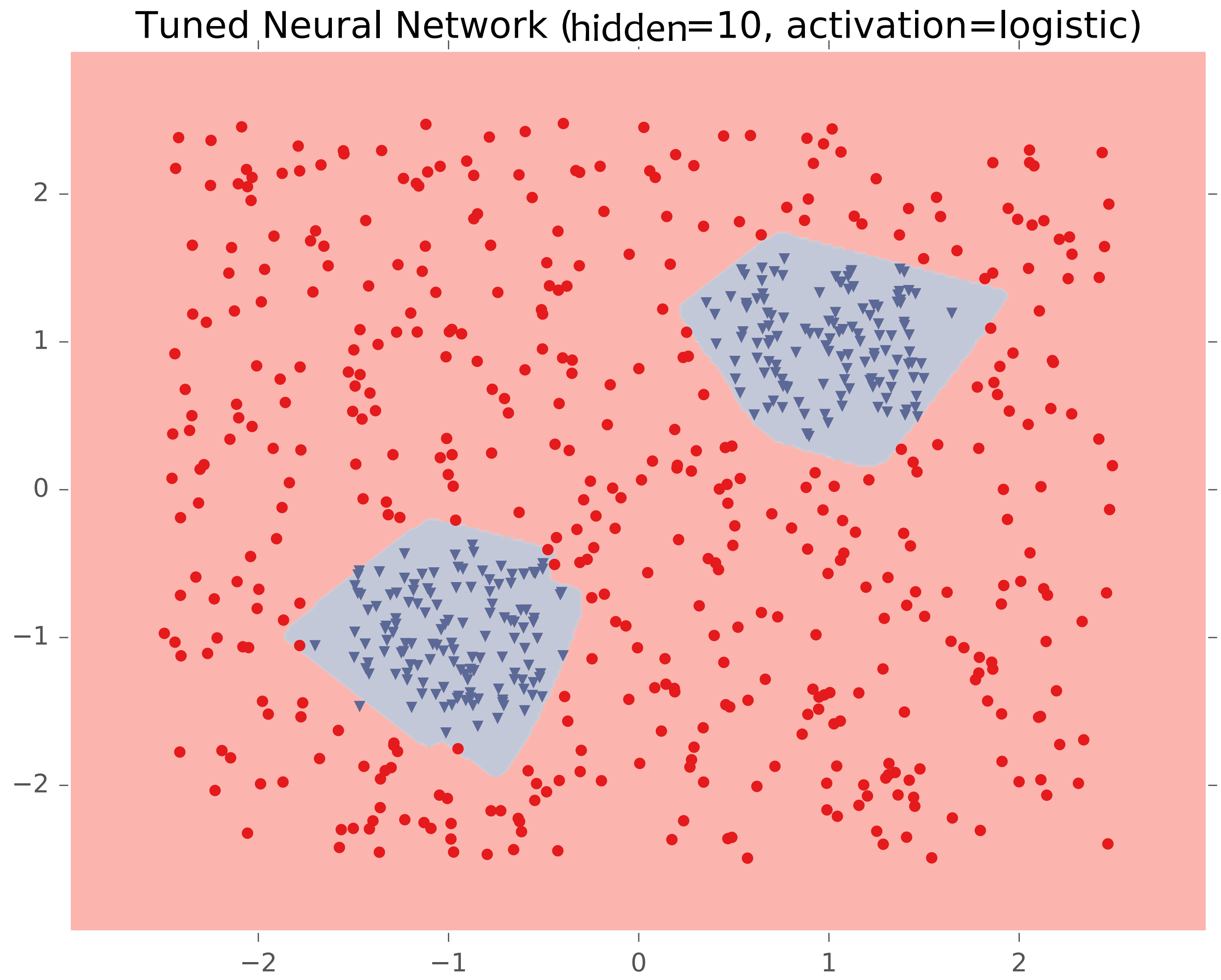
Example #4: Two Pockets



Example #4: Two Pockets



Example #4: Two Pockets



BUILDING DEEPER NETWORKS

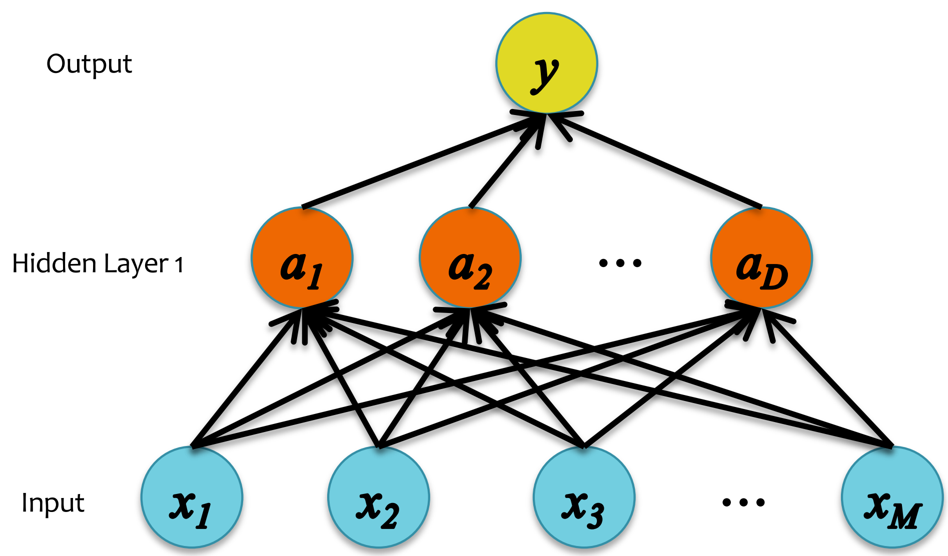
Neural Networks

Whiteboard

- Example: Neural Network w/2 Hidden Layers
- Example: Feed Forward Neural Network (matrix form)

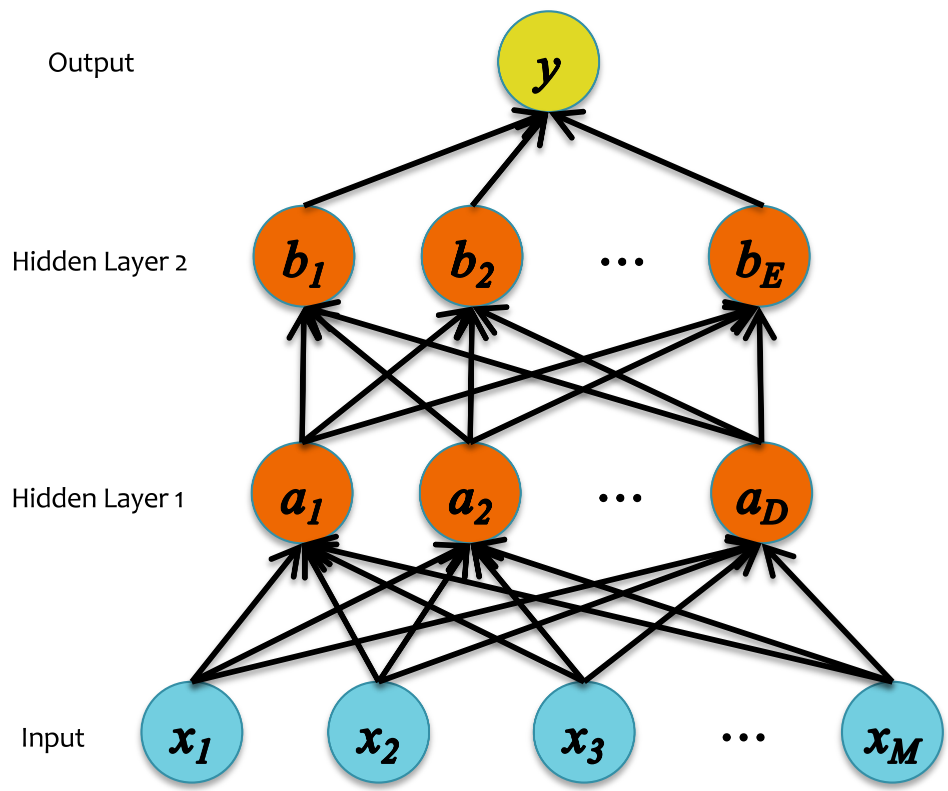
Deeper Networks

Q: How many layers should we use?



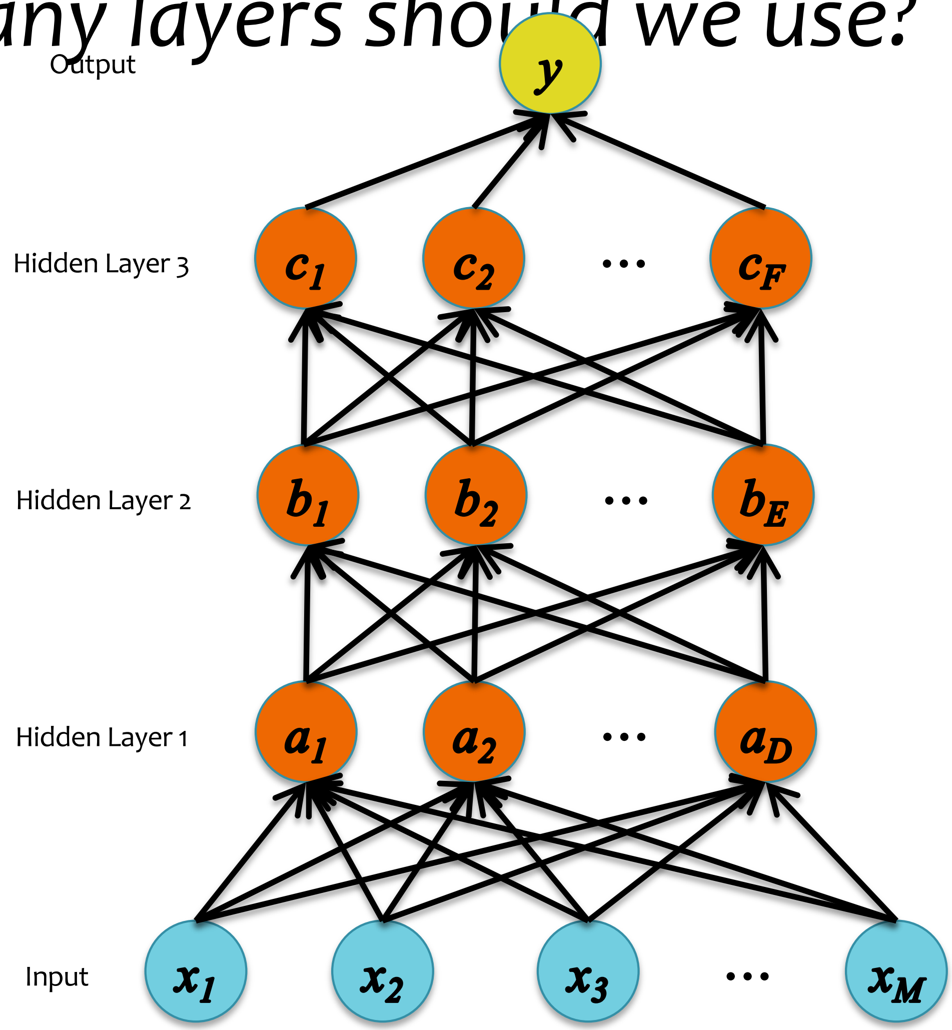
Deeper Networks

Q: How many layers should we use?



Deeper Networks

Q: How many layers should we use?



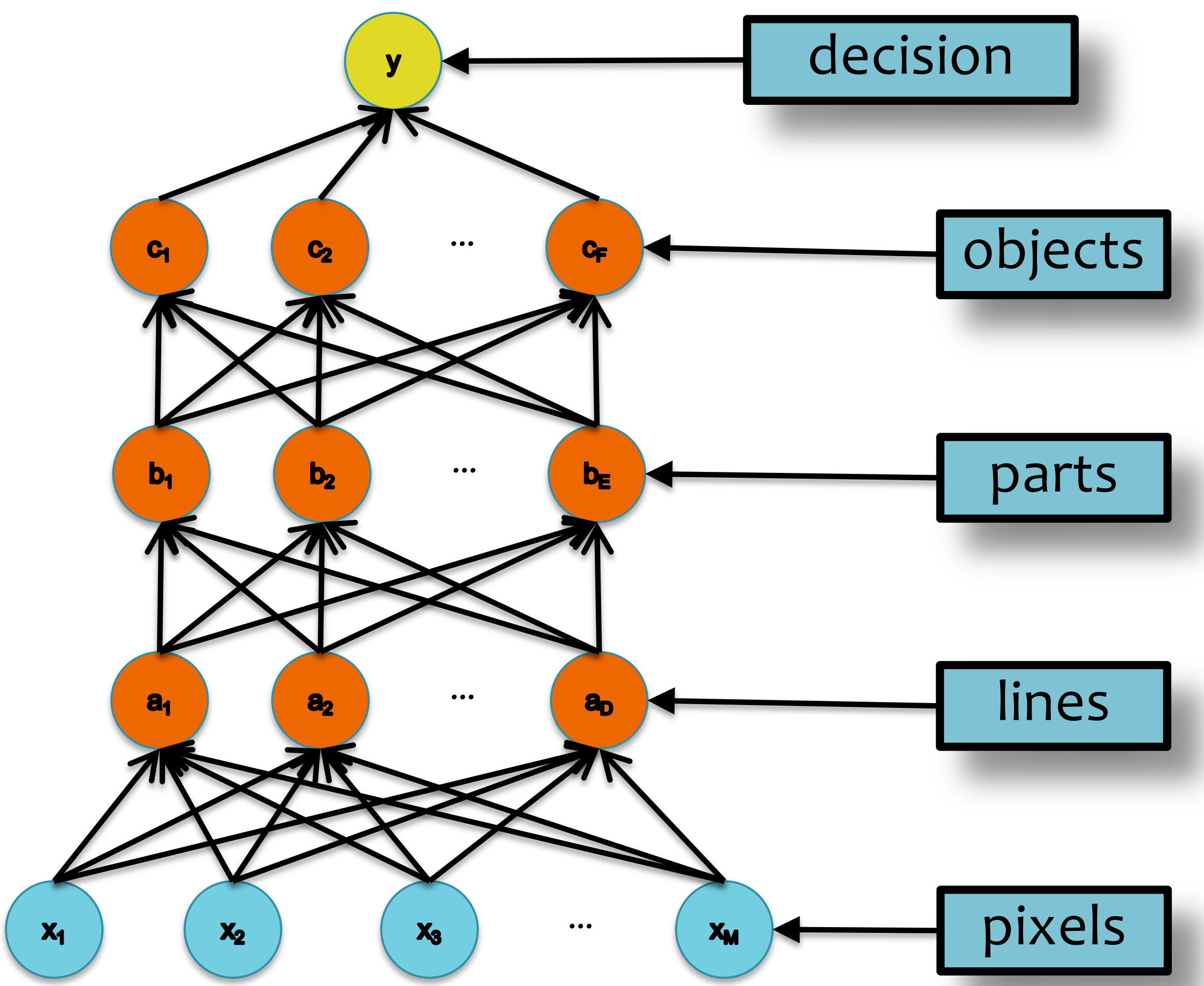
Deeper Networks

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.

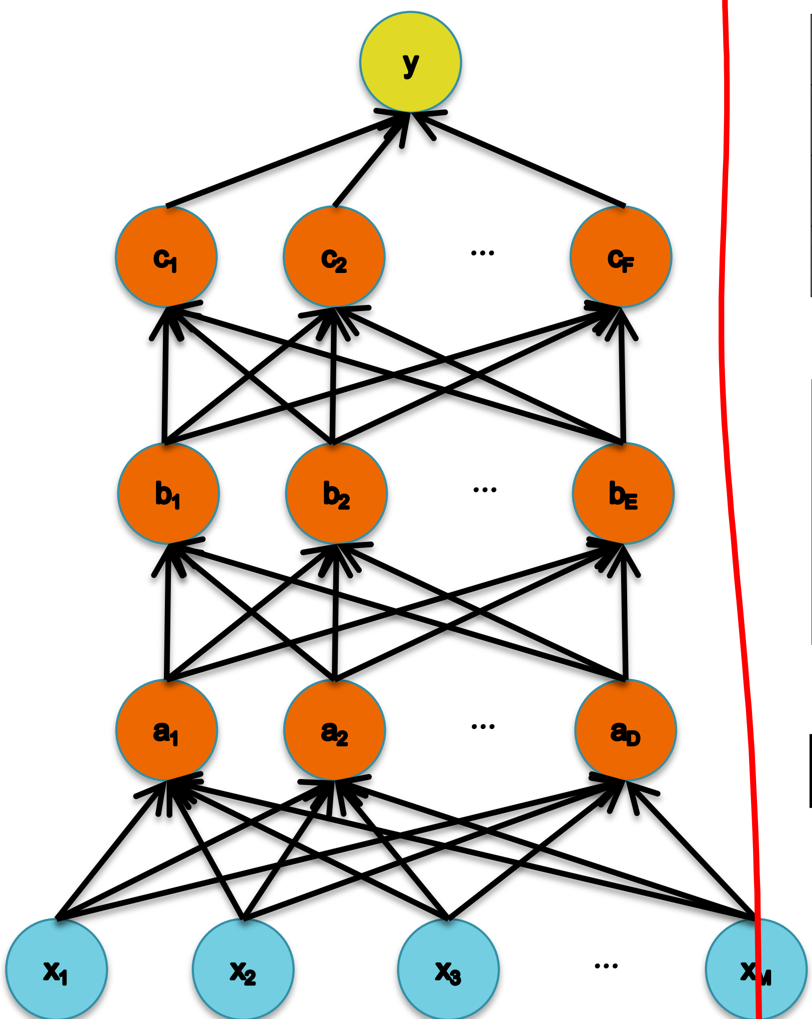
Feature Learning



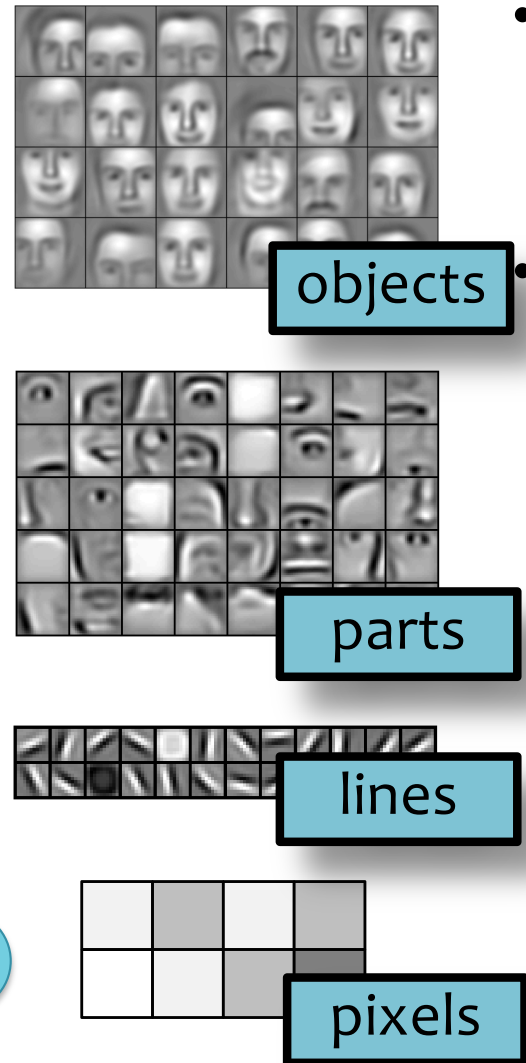
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning



CBDN on Faces

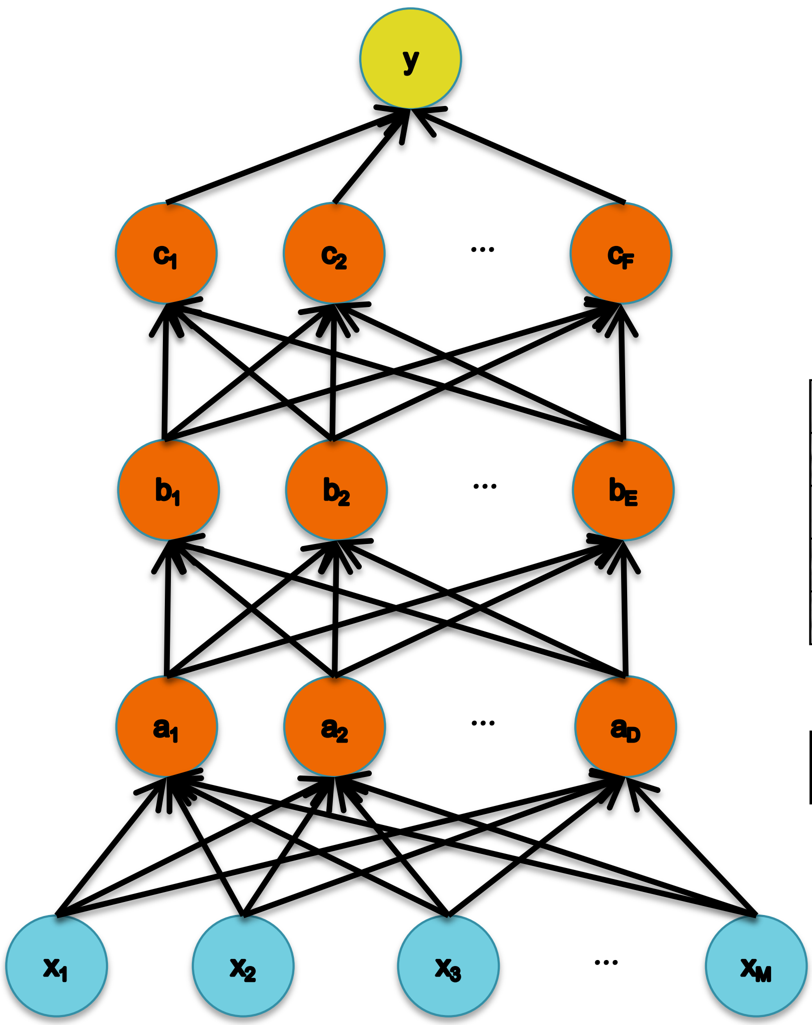


CNN

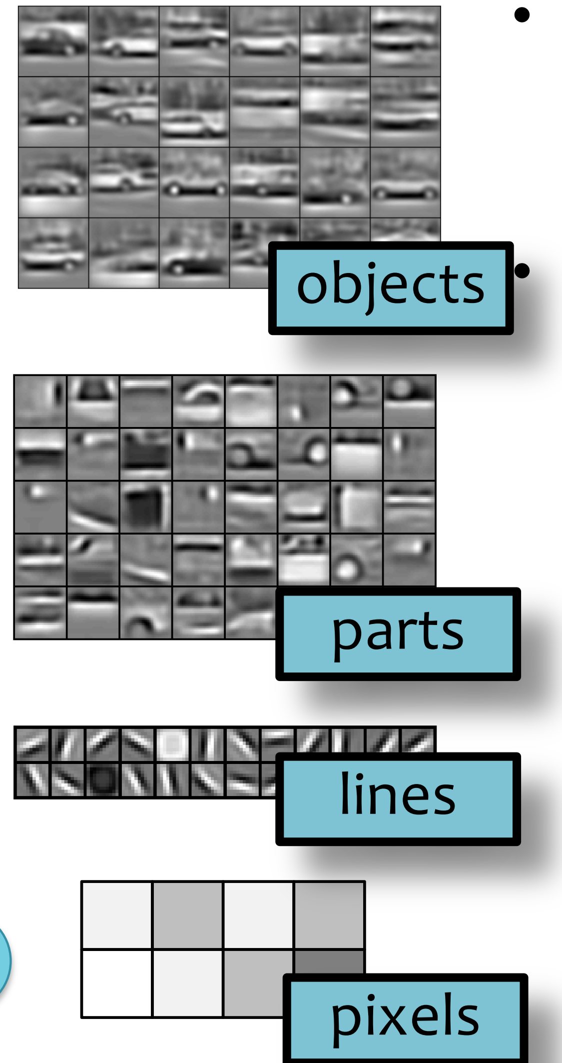
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning



CBDN on Cars



- **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

Figures from Lee et al. (ICML 2009)