10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

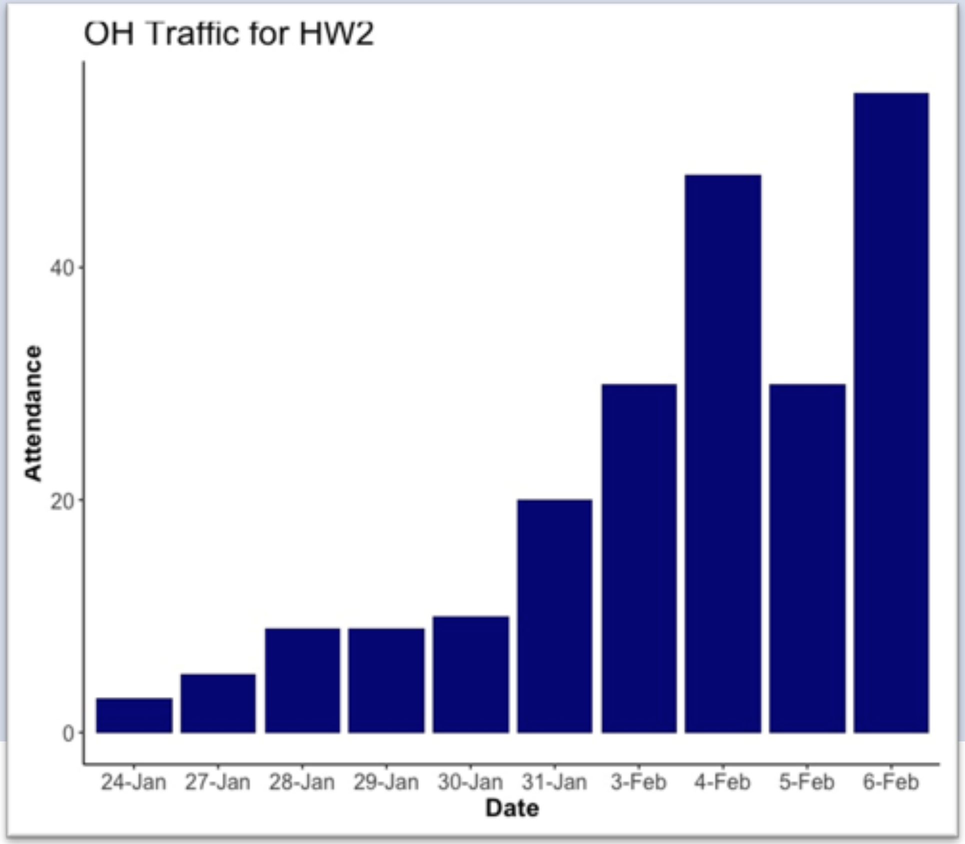
Neural Networks

Matt Gormley & Henry Chai
Lecture 11
Oct. 6, 2021

Q&A

Q: How can I get more one-on-one interaction with the course staff?

A: Attend office hours as soon after the homework release as possible!



Q&A

Q: "Why are there fewer OHs now?"

A: Wrong question. I think what you meant was:

"I just noticed that you guys are modeling office hour demand and adaptively scaling the number of office hours and number of TAs present to maximize contact time when I really need it! How can I be more like your awesome TAs?"

Great question. Spend more time talking with them at OHs, whenever you want and we'll adapt.

And yes, we are actually increasing the (amortized) amount of OHs per TA, but it's hard to observe if you're just looking at the calendar.

Reminders

- **Post-Exam Followup:**
  - Exam Viewing
  - Exit Poll: Exam 1
  - Grade Summary 1
- **Homework 4: Logistic Regression**
  - Out: Fri, Oct. 1
  - Due: Mon, Oct. 11 at 11:59pm
- **Swapped lecture/recitation:**
  - Lecture 12: Fri, Oct. 8

Q&A

Q: Am I good enough?

A: Exam 1 cannot answer that question for you. It can only answer the following:

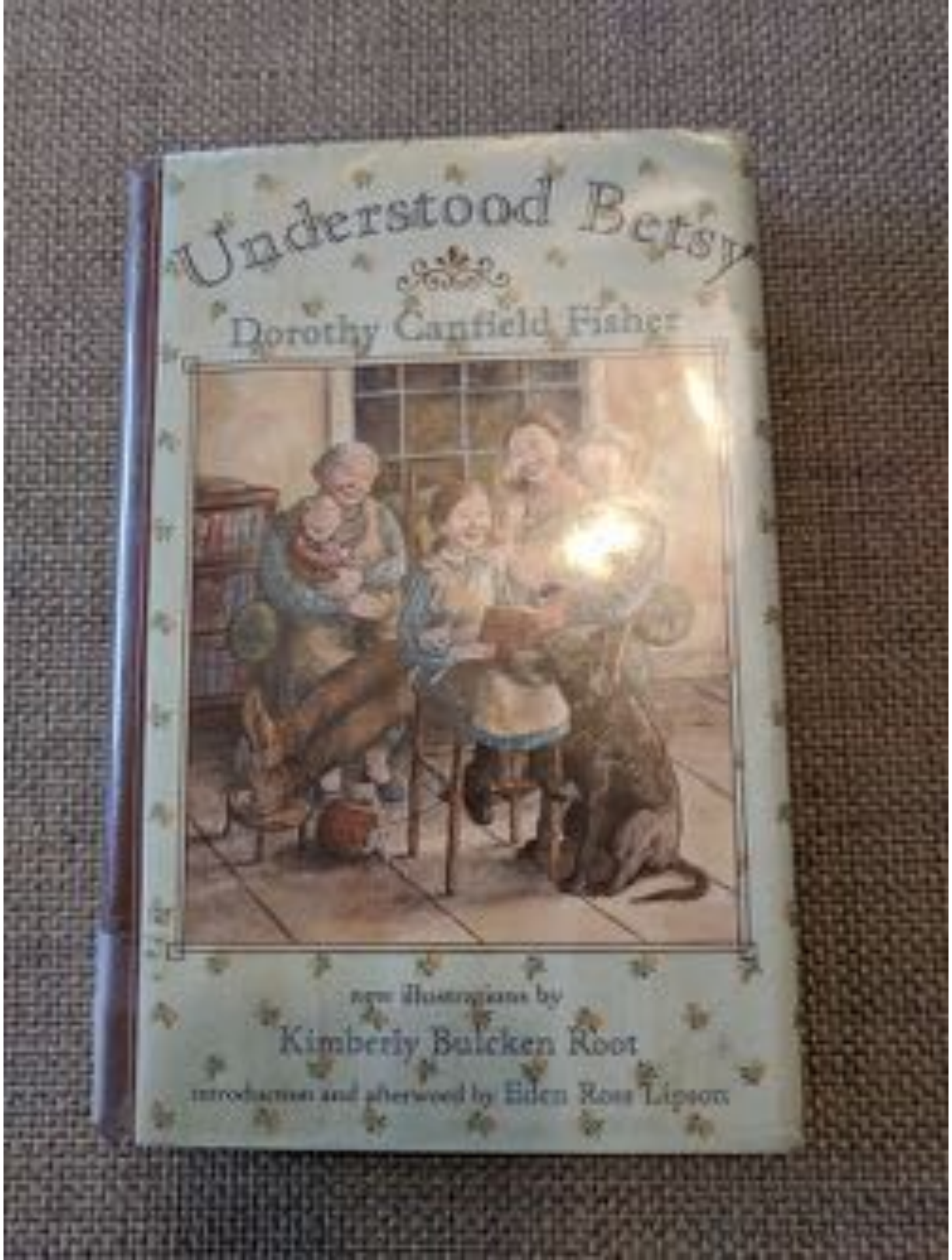
"How well did you perform on a timed standardized test taken on the 30th of September on the topics of decision trees, k-nearest neighbors, perceptron, and linear regression."

Q&A

Q: Can it answer any of these questions?

- "Will I someday become a machine learning scientist?"
- "Will I get that internship?"
- "How successful will I be in my future endeavors?"
- "Am I going to have impact on the world?"
- "How many licks does it take to get to the center of a Tootsie Pop™?"

A: No

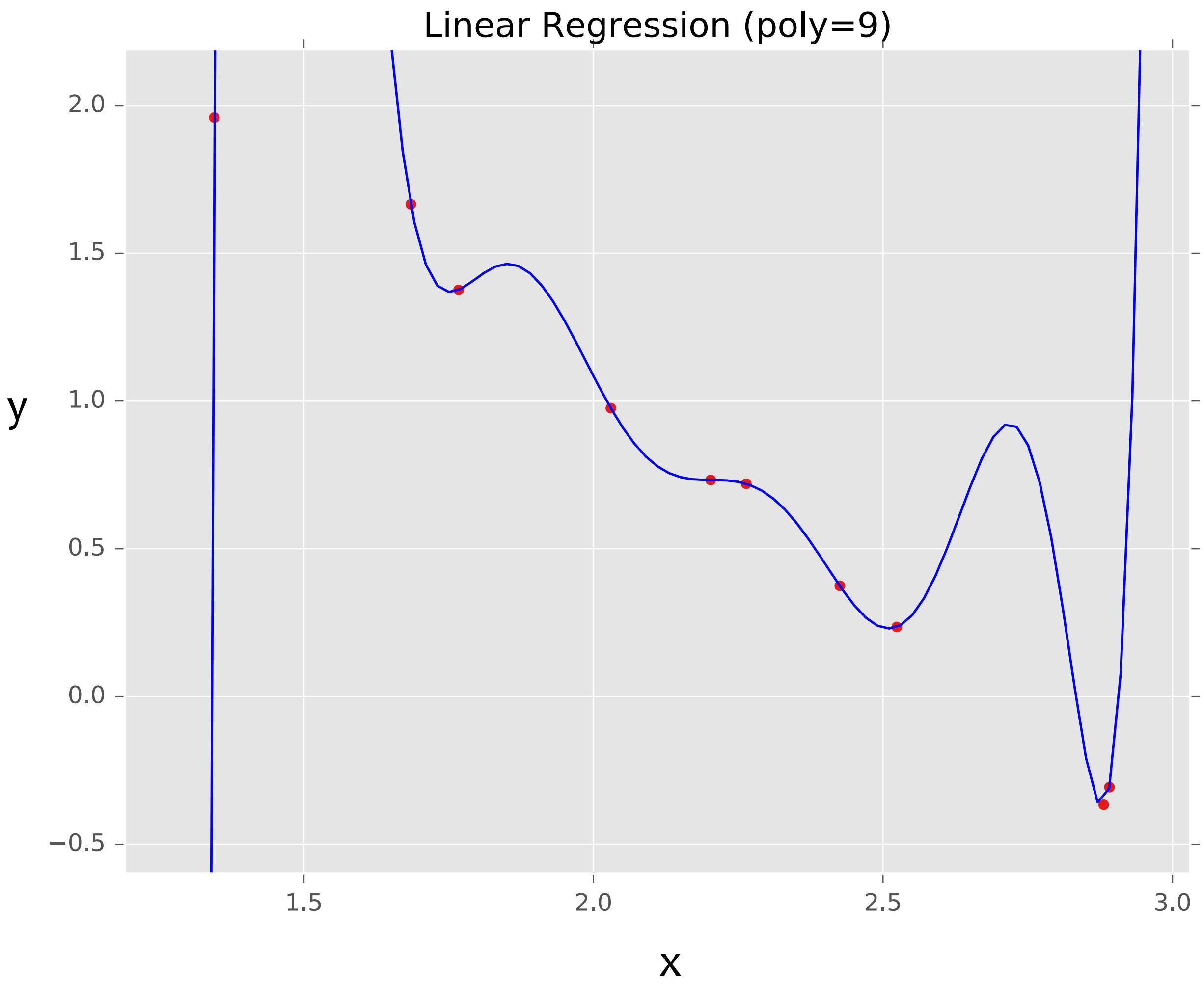

Understood Betsy
Dorothy Canfield Fisher
Kimberly Bulcken Root

REGULARIZATION

Example: Linear Regression

Goal: Learn y = $\mathbf{w}^T f(\mathbf{x}) + b$
where f(.) is a polynomial basis function

i	y	x	…	x^9
1	2.0	1.2	…	$(1.2)^9$
2	1.3	1.7	…	$(1.7)^9$
…	…	…	…	…
10	1.1	1.9	…	$(1.9)^9$



Polynomial Coefficients

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
θ_0	0.19	0.82	0.31	0.35
θ_1		-1.27	7.99	232.37
θ_2			-25.43	-5321.83
θ_3			17.37	48568.31
θ_4				-231639.30
θ_5				640042.26
θ_6				-1061800.52
θ_7				1042400.18
θ_8				-557682.99
θ_9				125201.43

Slide courtesy of William Cohen

Example: Linear Regression

Goal: Learn $y = \mathbf{w}^T f(\mathbf{x}) + b$ where f(.) is a polynomial basis function

i	y	x	…	x^9
1	2.0	1.2	…	$(1.2)^9$
2	1.3	1.7	…	$(1.7)^9$
3	0.1	2.7	…	$(2.7)^9$
4	1.1	1.9	…	$(1.9)^9$
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
98	…	…	…	…
99	…	…	…	…
100	0.9	1.5	…	$(1.5)^9$

- With just N = 10 points we overfit!
- But with N = 100 points, the overfitting (mostly) disappears
- **Takeaway:** more data helps prevent overfitting

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**)

$$\vec{x} = \begin{bmatrix} 10 \\ 11 \\ 17 \\ 9 \end{bmatrix} \qquad \vec{\theta} = \begin{bmatrix} 100 \\ -25 \\ 0.0001 \\ 0.001 \end{bmatrix} \approx \vec{\theta}' = \begin{bmatrix} 100 \\ -25 \end{bmatrix}$$

Regularization

- **Given** objective function: J(θ)
- **Goal** is to find: $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}}\, J(\boldsymbol{\theta}) + \lambda r(\boldsymbol{\theta})$
- **Key idea:** Define regularizer r(θ) s.t. we tradeoff between fitting the data and keeping the model simple
- **Choose form of r(θ):**
  – 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

Question: Q1

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)

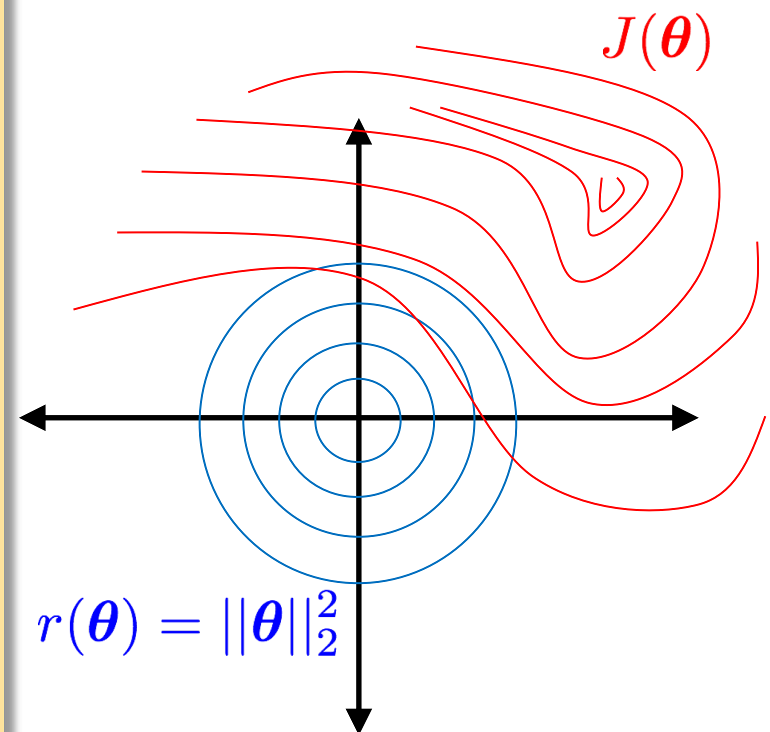
B. …move towards the minimum of J(θ)

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~~ toxic



▲ When survey is active, respond at **pollev.com/10301601polls**

Lecture 11: In-Class Poll

0 done

0 underway

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

When poll is active, respond at **pollev.com/10301601polls**

Question 1

A

B

C

D

E

F

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

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

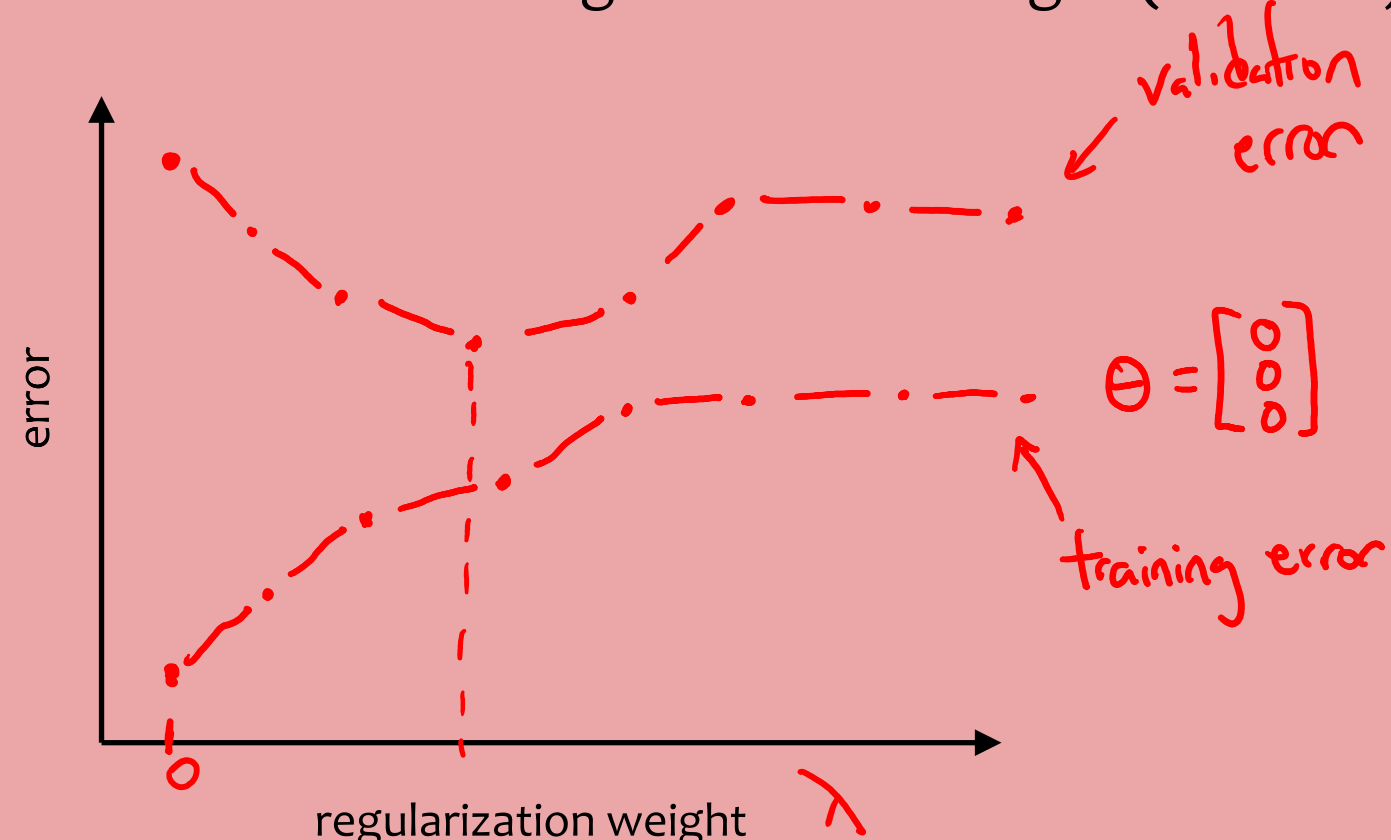
Whitening Data

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

Regularization Exercise

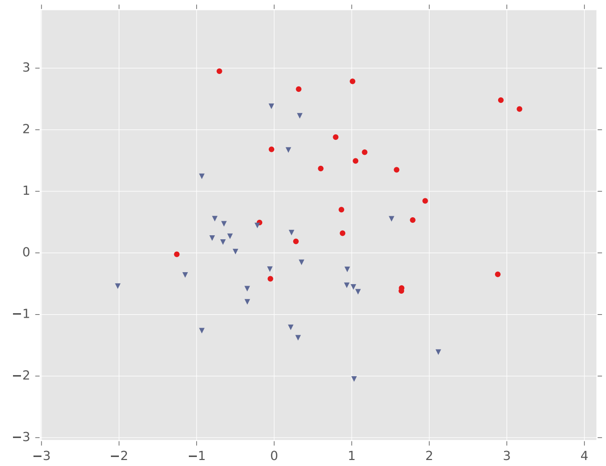
In-class Exercise

1. Plot train error vs. regularization weight (cartoon)
2. Plot validation error vs . regularization weight (cartoon)

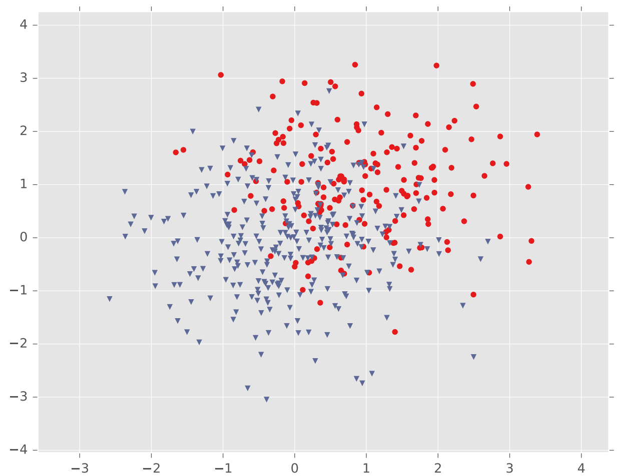


Example: Logistic Regression

Training Data

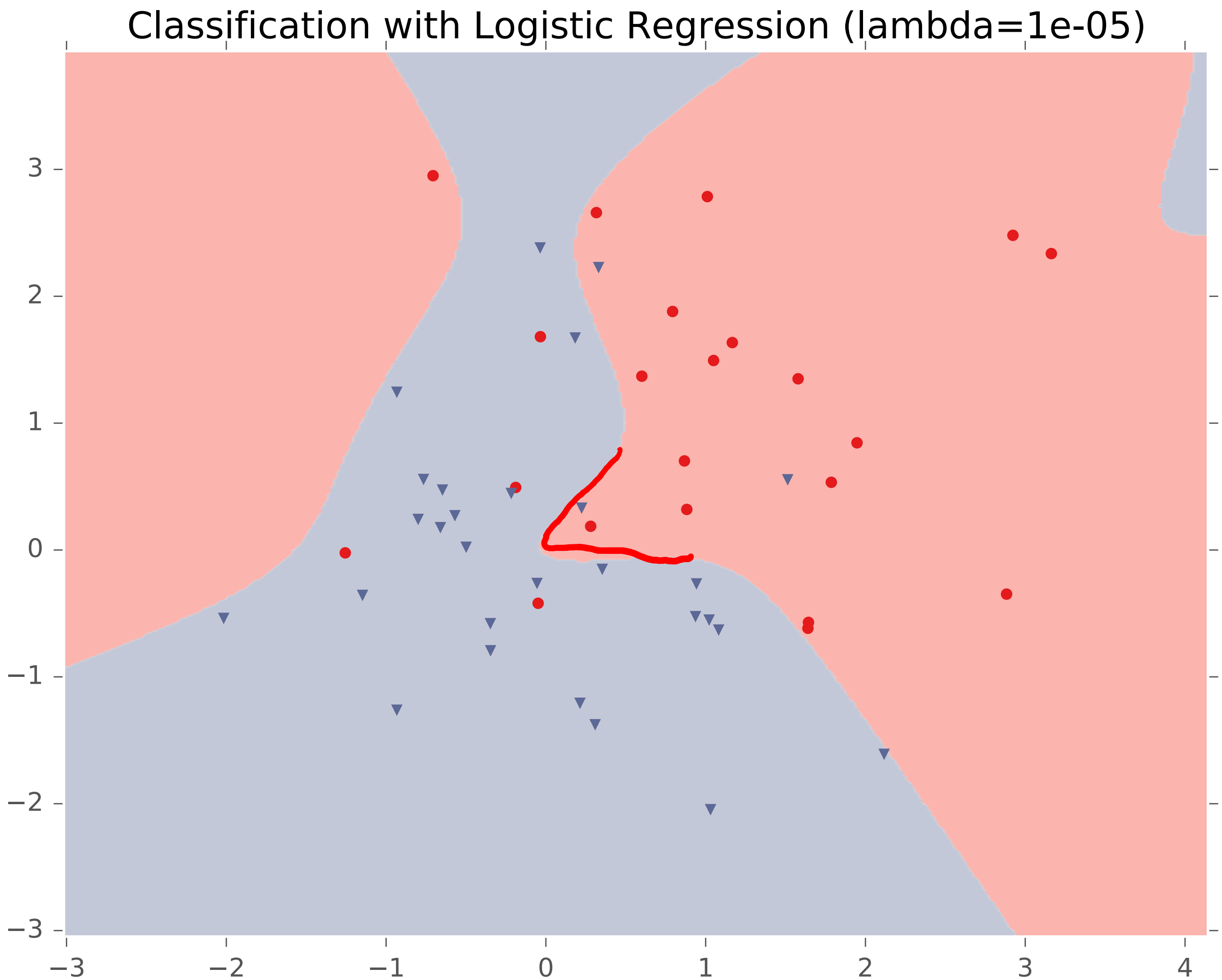


Test Data

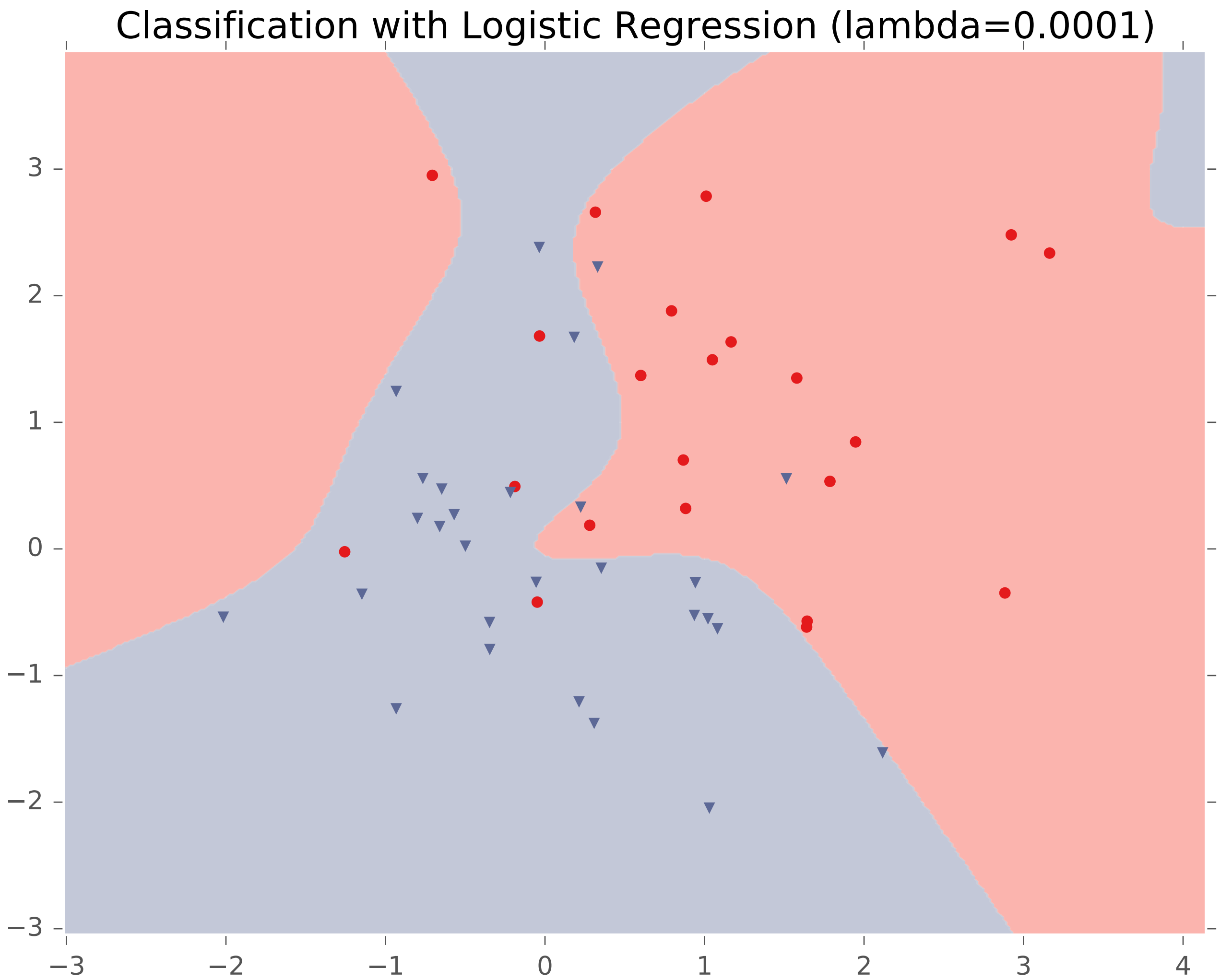


- For this example, we construct **nonlinear features** (i.e. feature engineering)
- Specifically, we add **polynomials up to order 9** of the two original features x_1 and x_2
- Thus our classifier is **linear** in the **high-dimensional feature space**, but the decision boundary is **nonlinear** when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

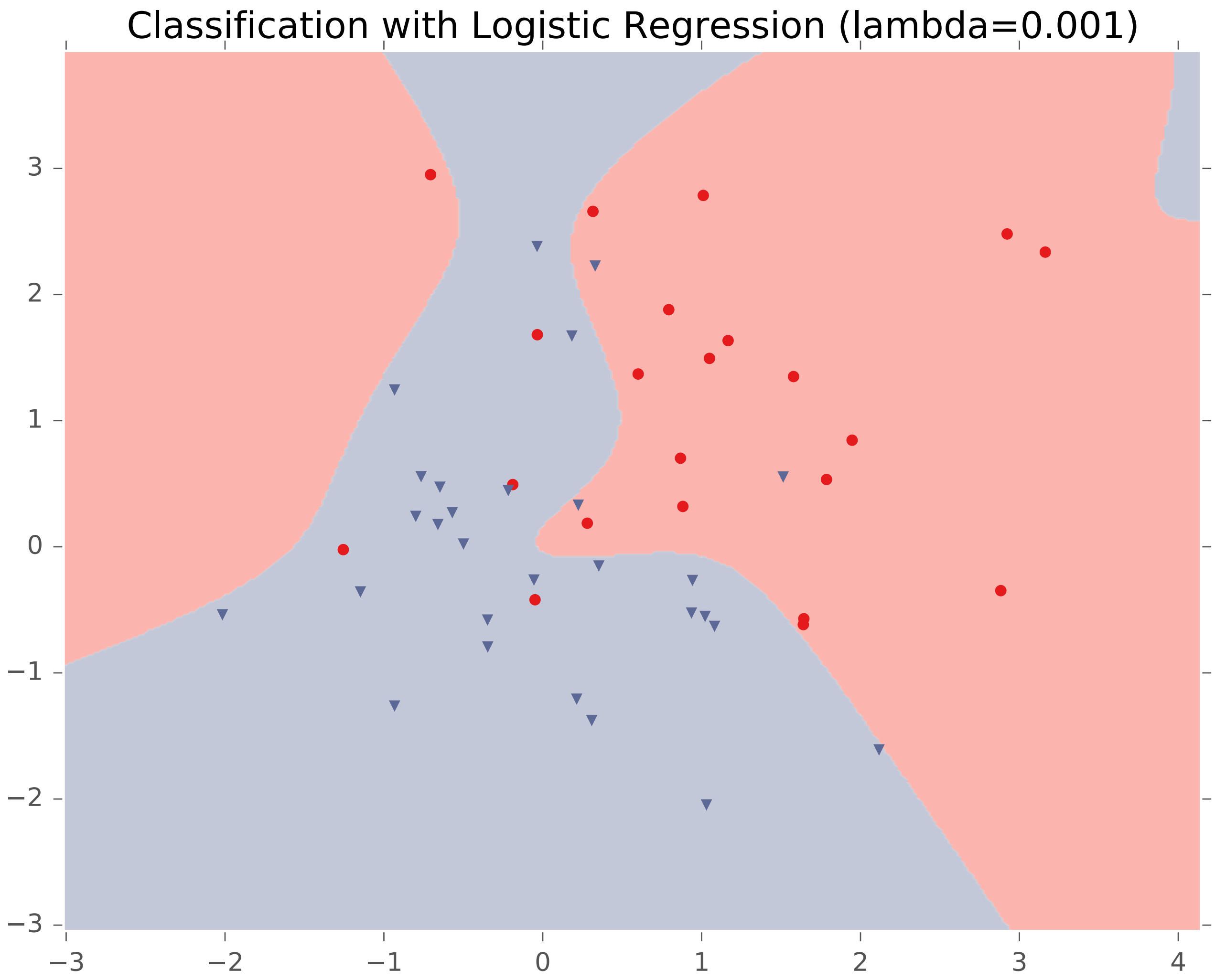
Example: Logistic Regression



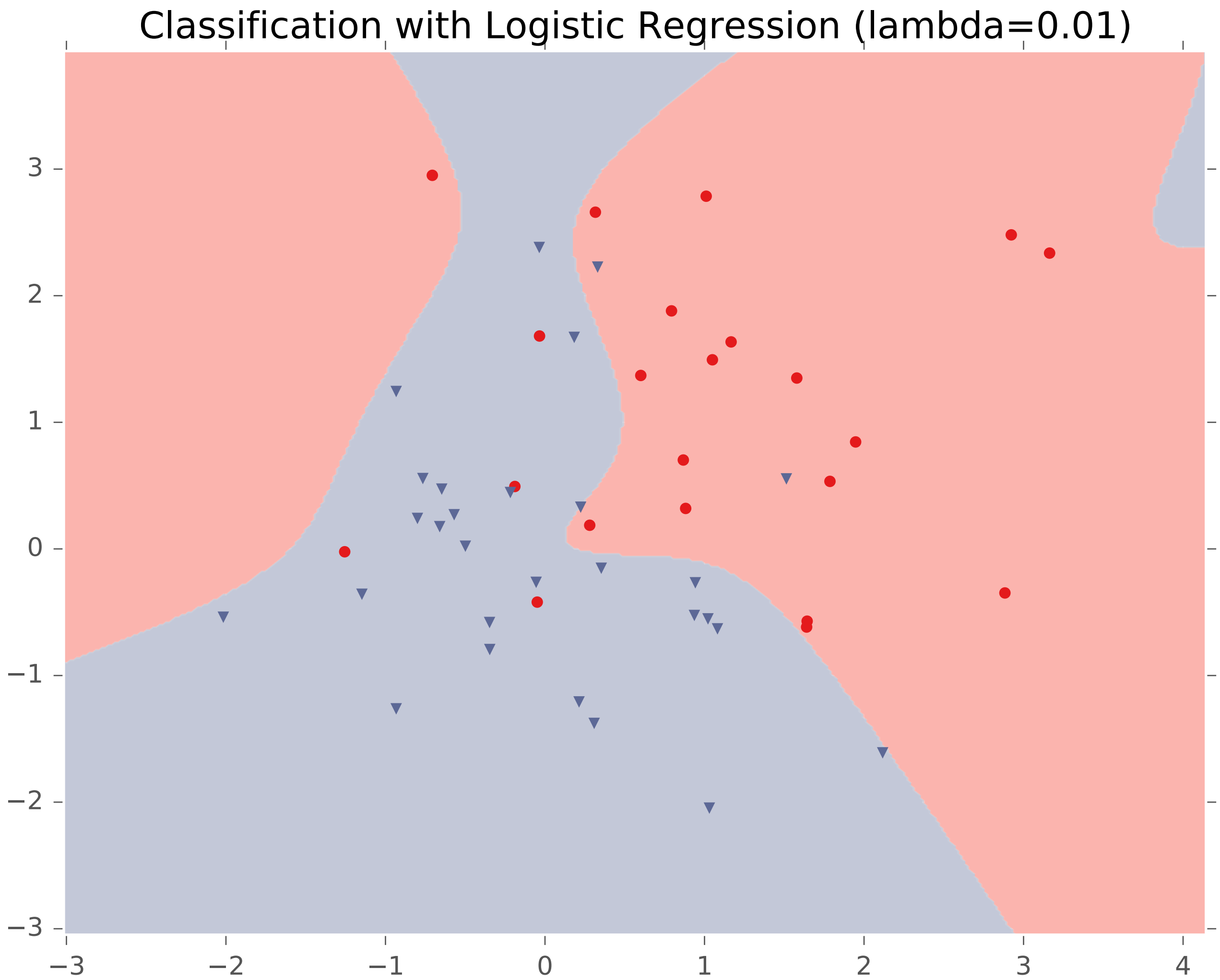
Example: Logistic Regression



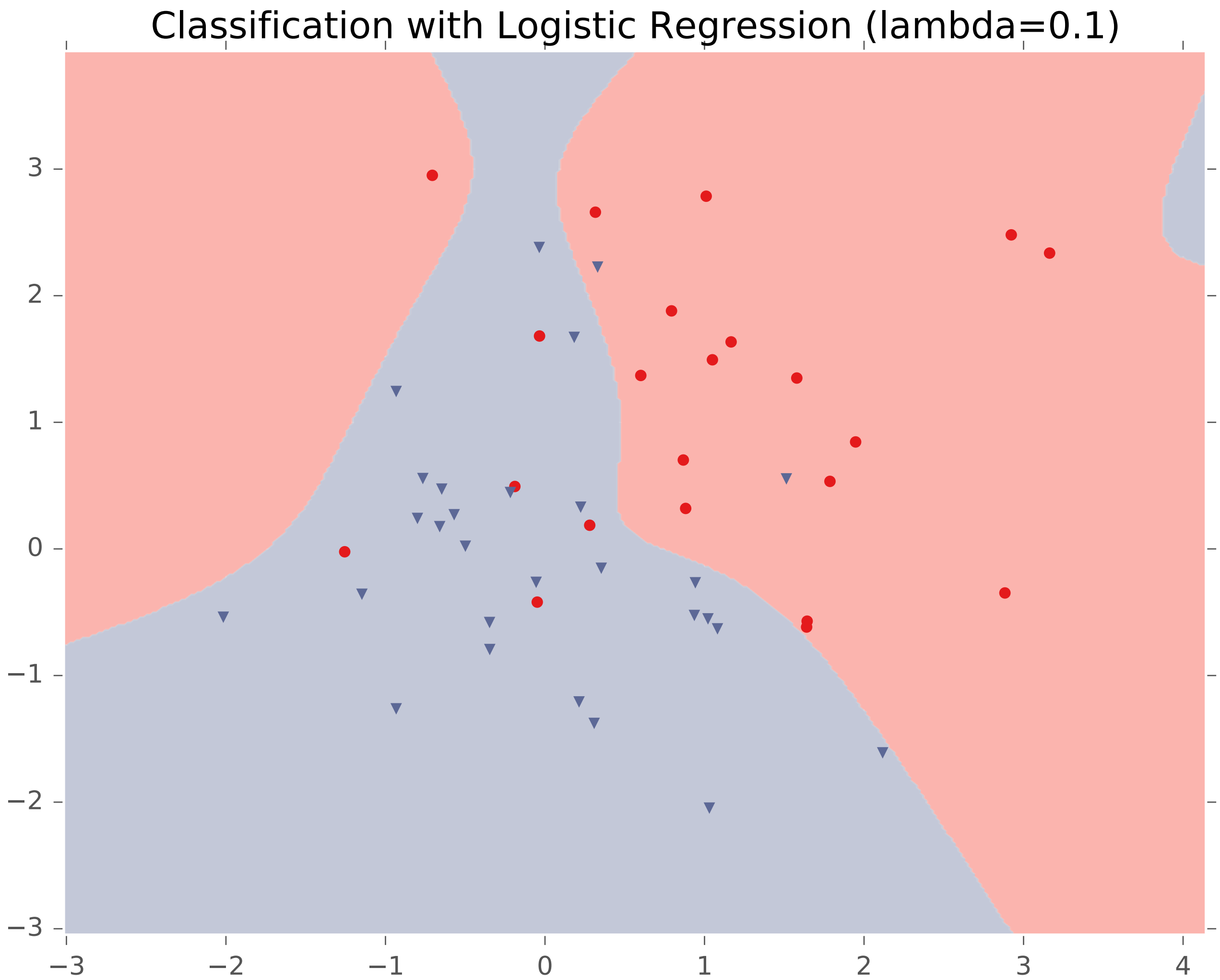
Example: Logistic Regression



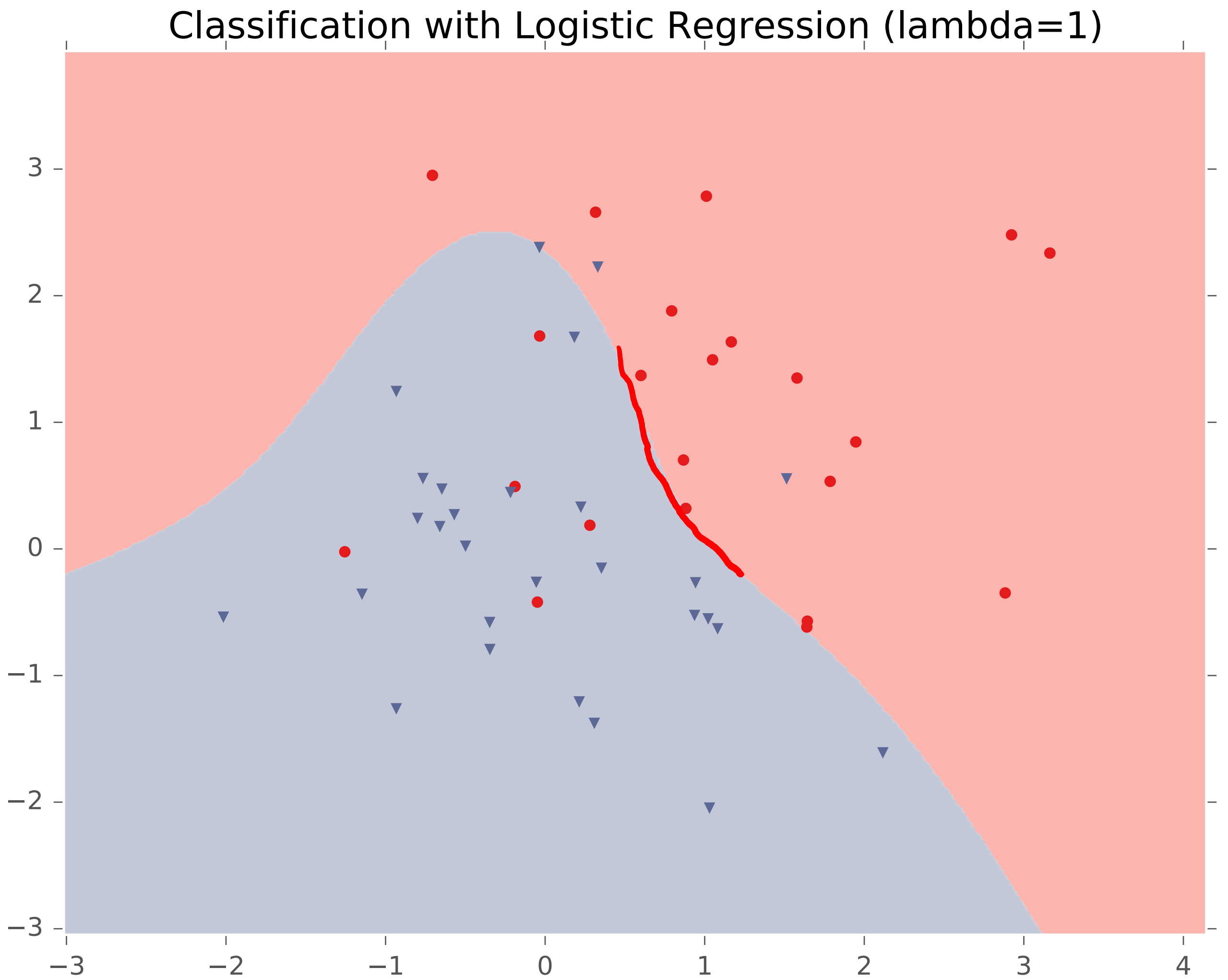
Example: Logistic Regression



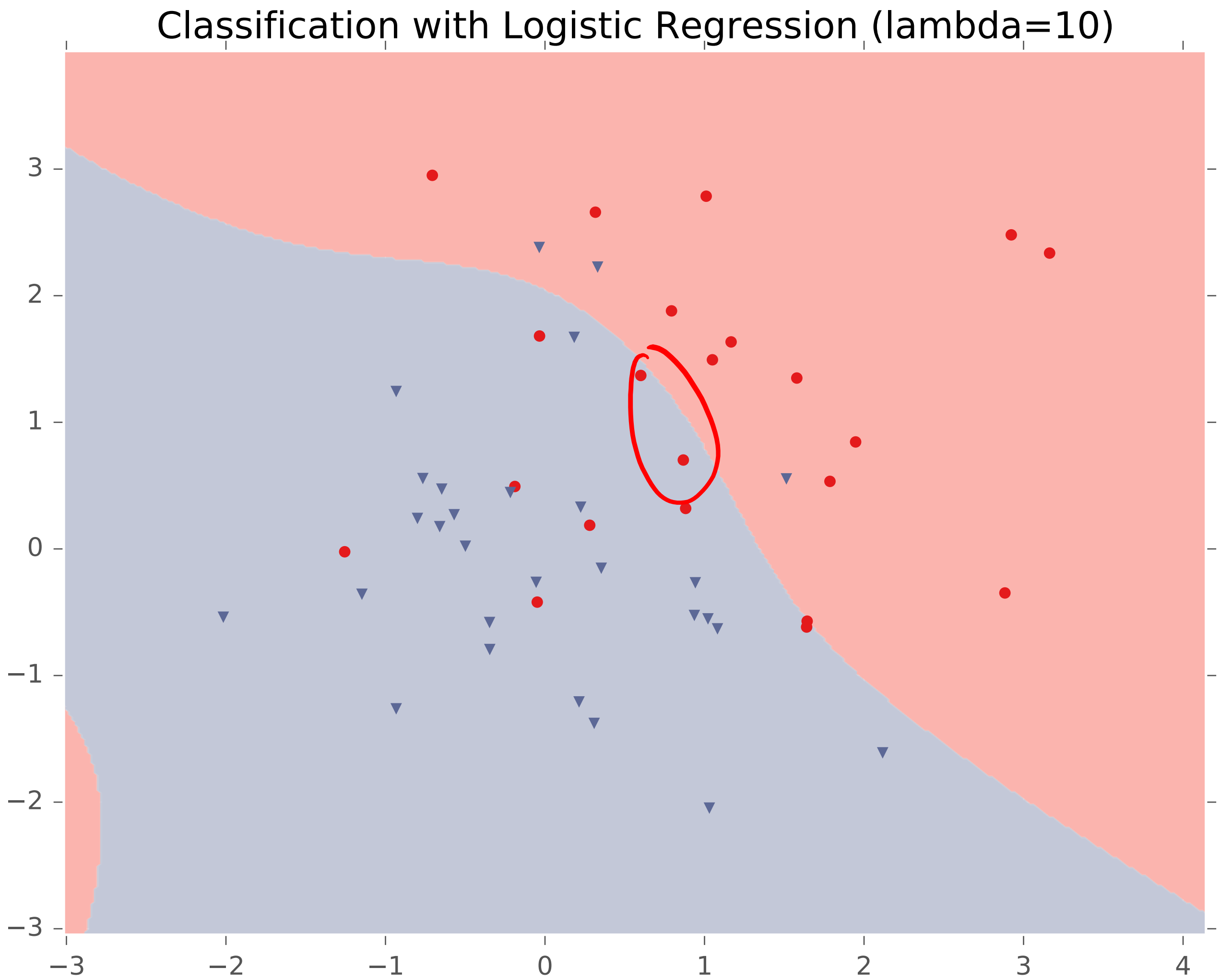
Example: Logistic Regression



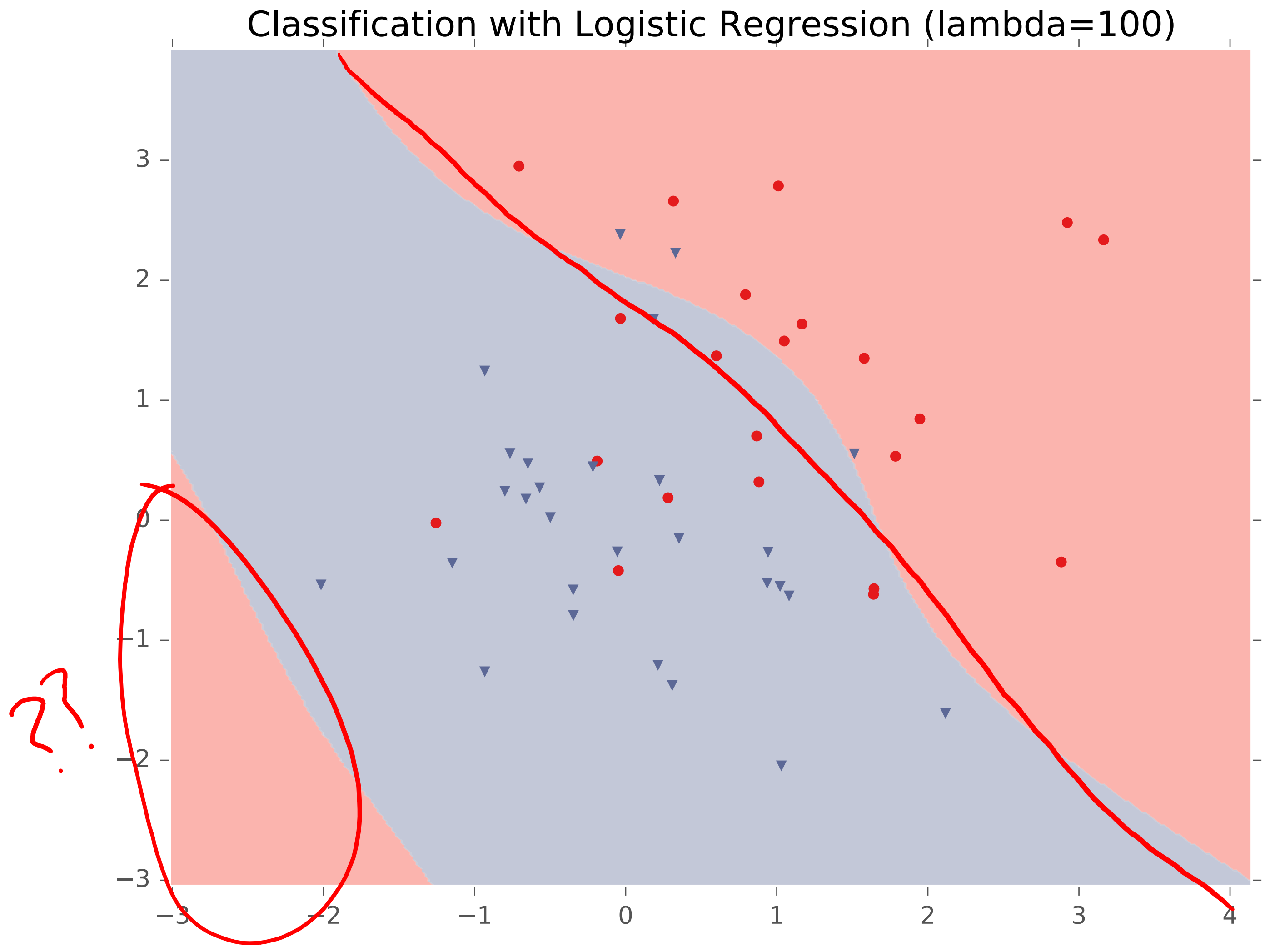
Example: Logistic Regression



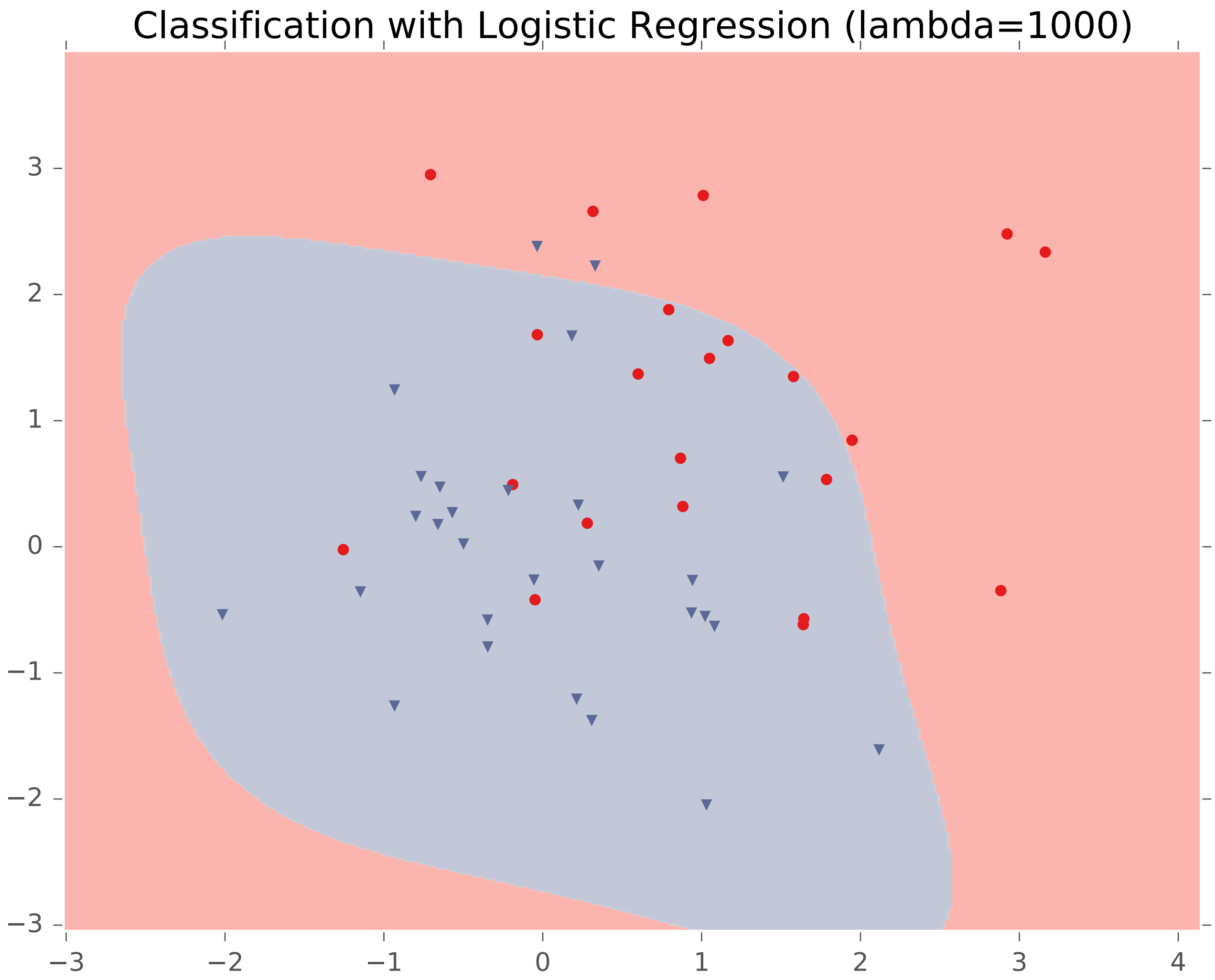
Example: Logistic Regression



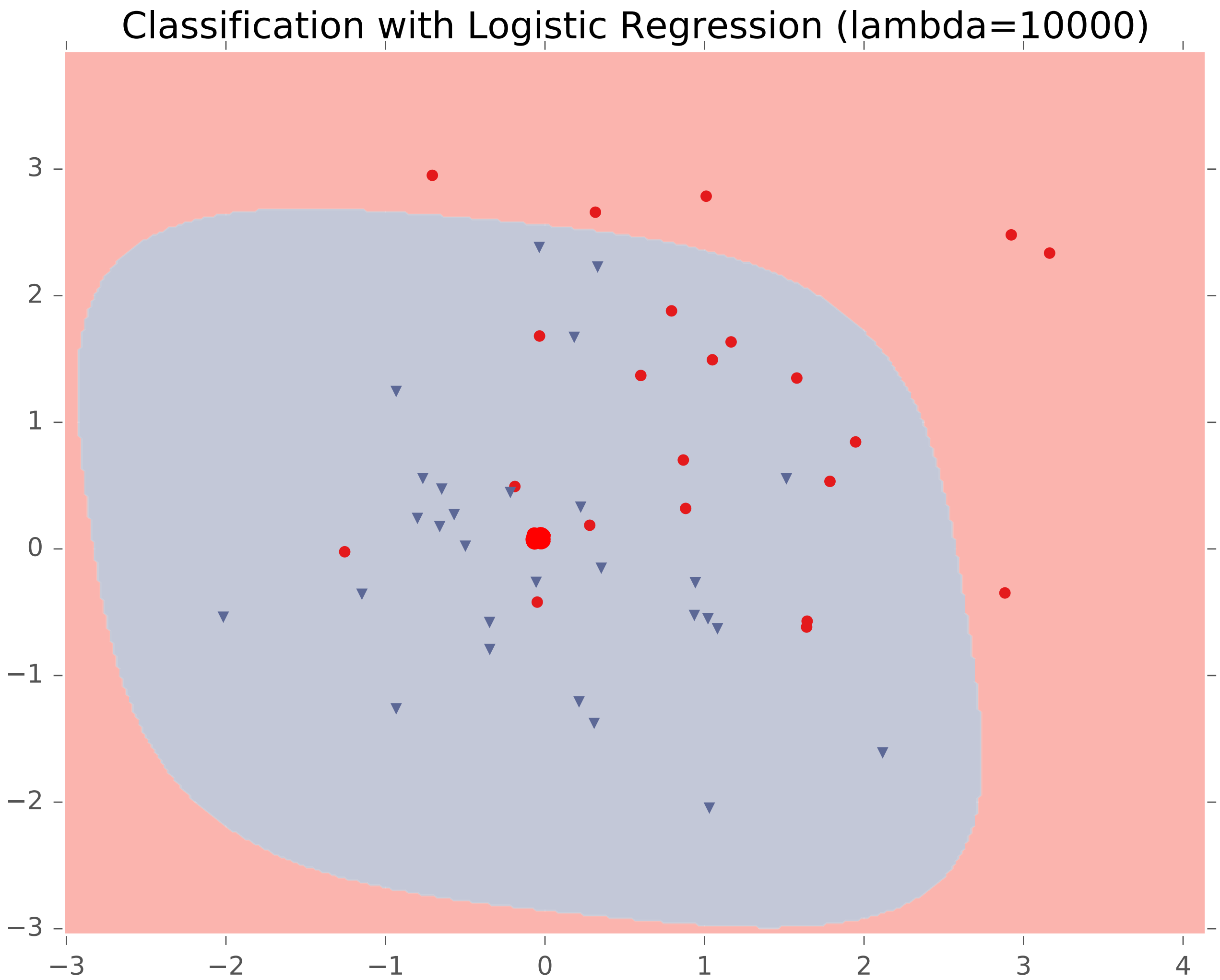
Example: Logistic Regression



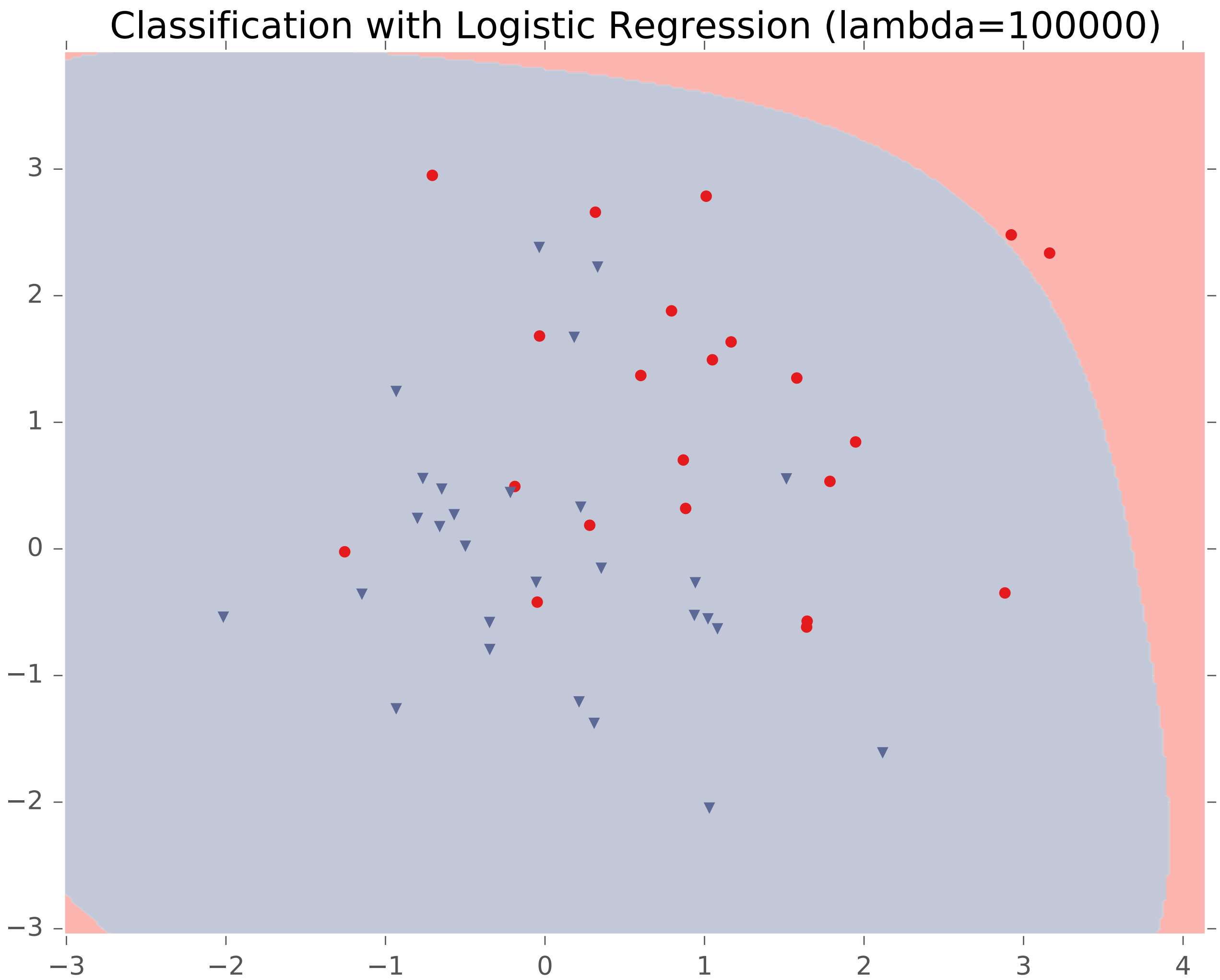
Example: Logistic Regression



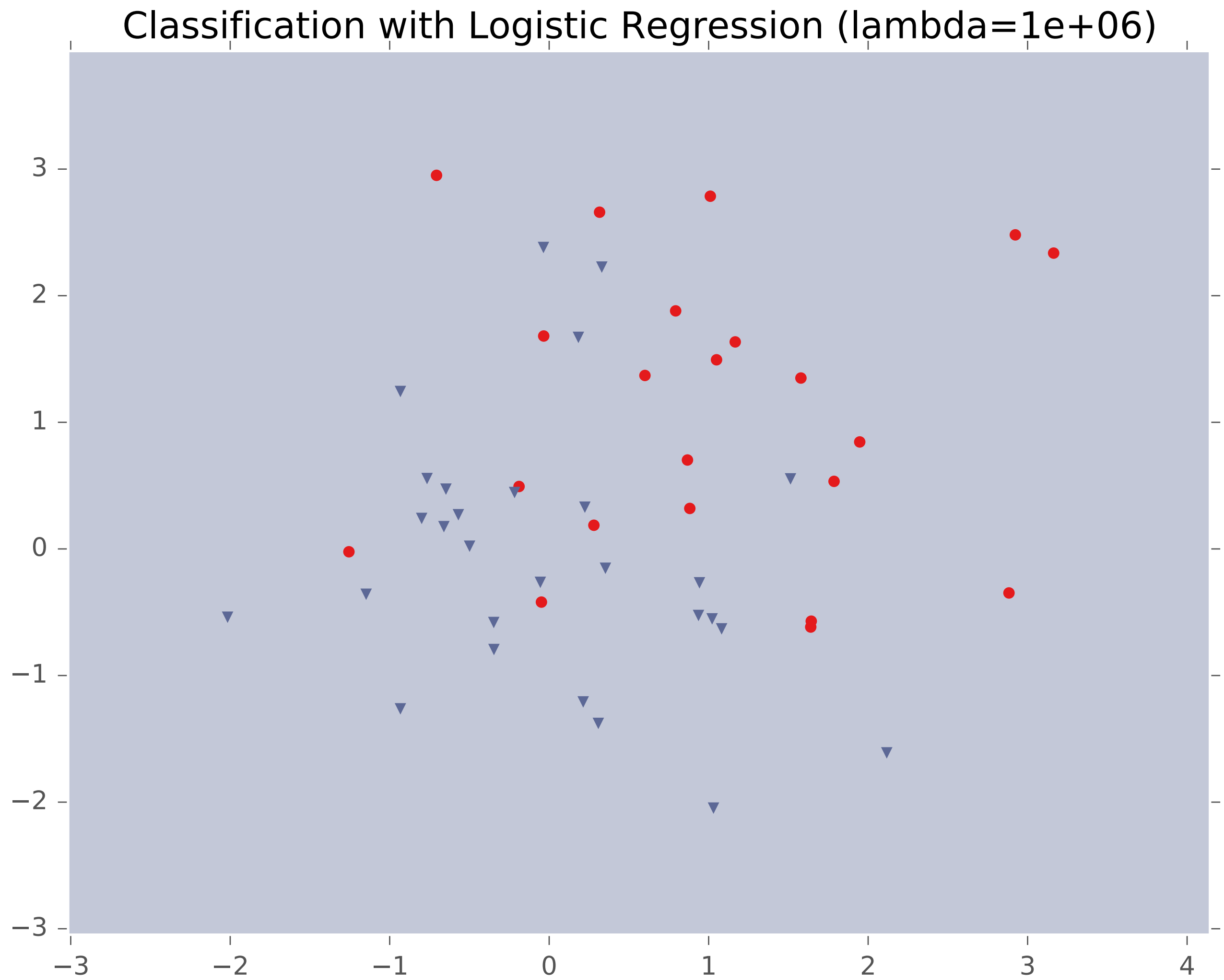
Example: Logistic Regression



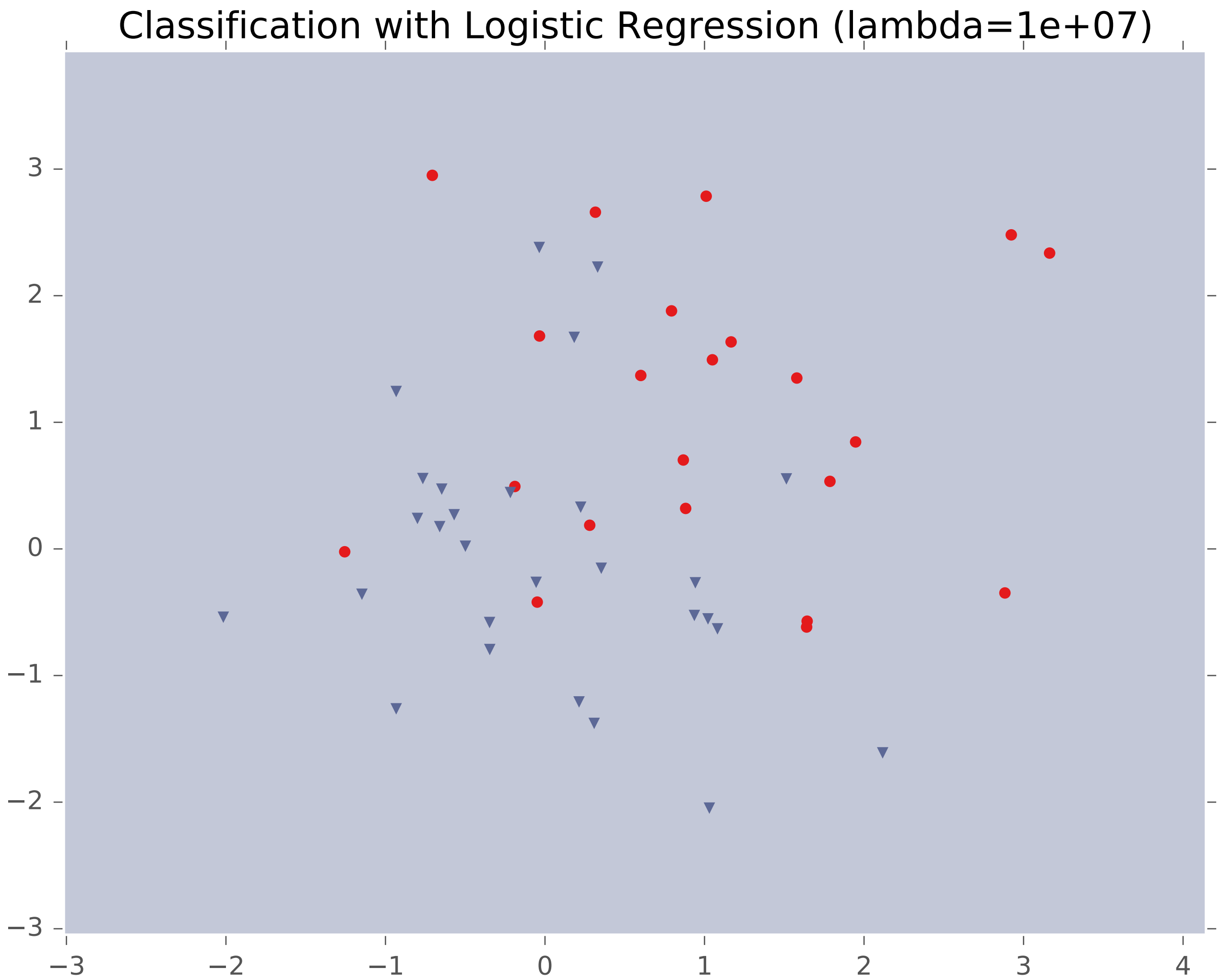
Example: Logistic Regression



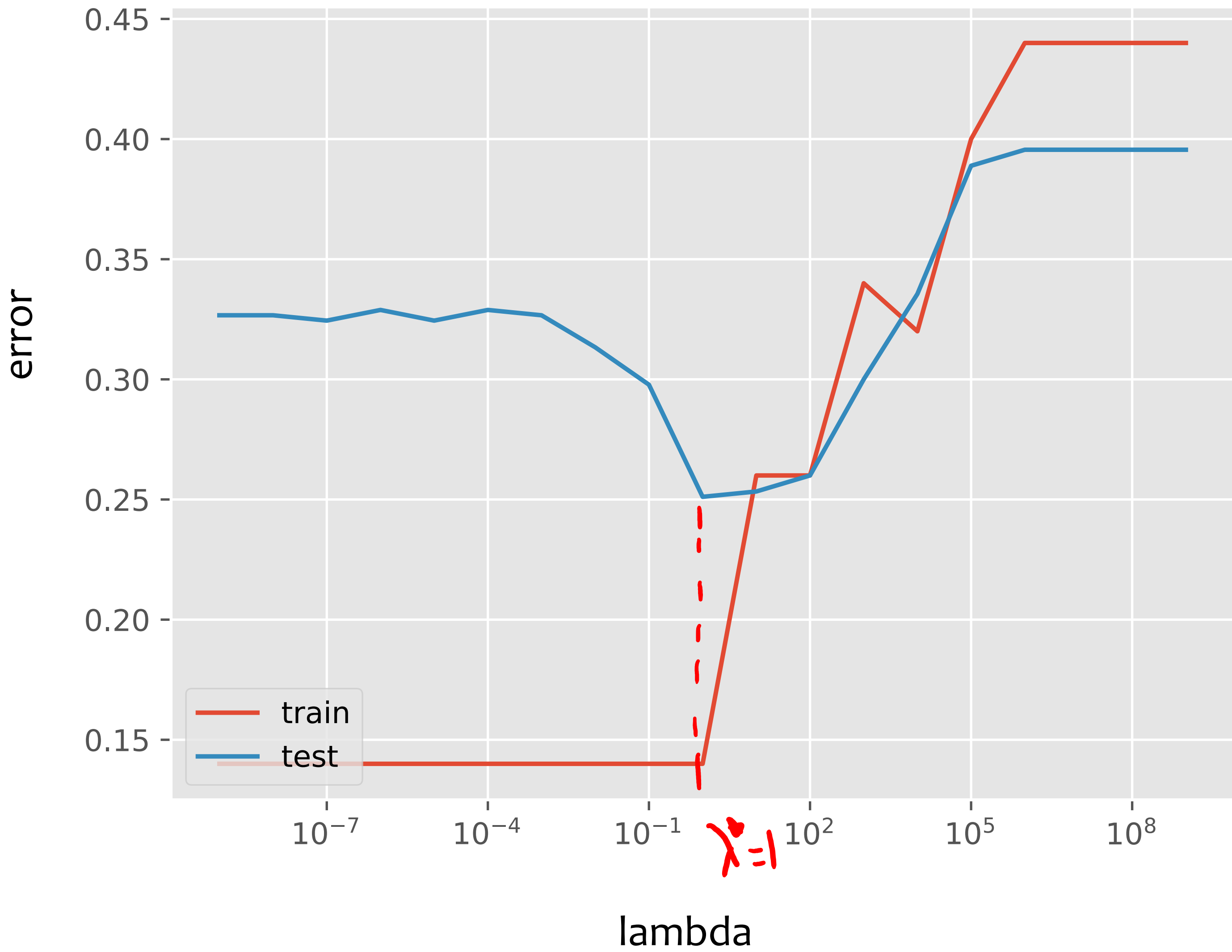
Example: Logistic Regression



Example: Logistic Regression



Example: Logistic Regression



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…

- 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

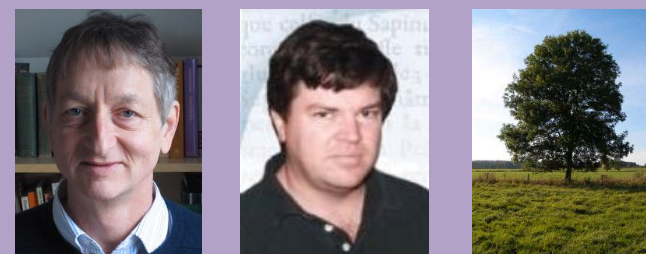
Background

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

$h_\theta(x_i)$

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

– 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:

empirical risk

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

4. Train with SGD:

(take small steps opposite the gradient)

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

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!

Goals for Today's Lecture

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

– Decision function

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

– Loss function

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

Train with SGD:

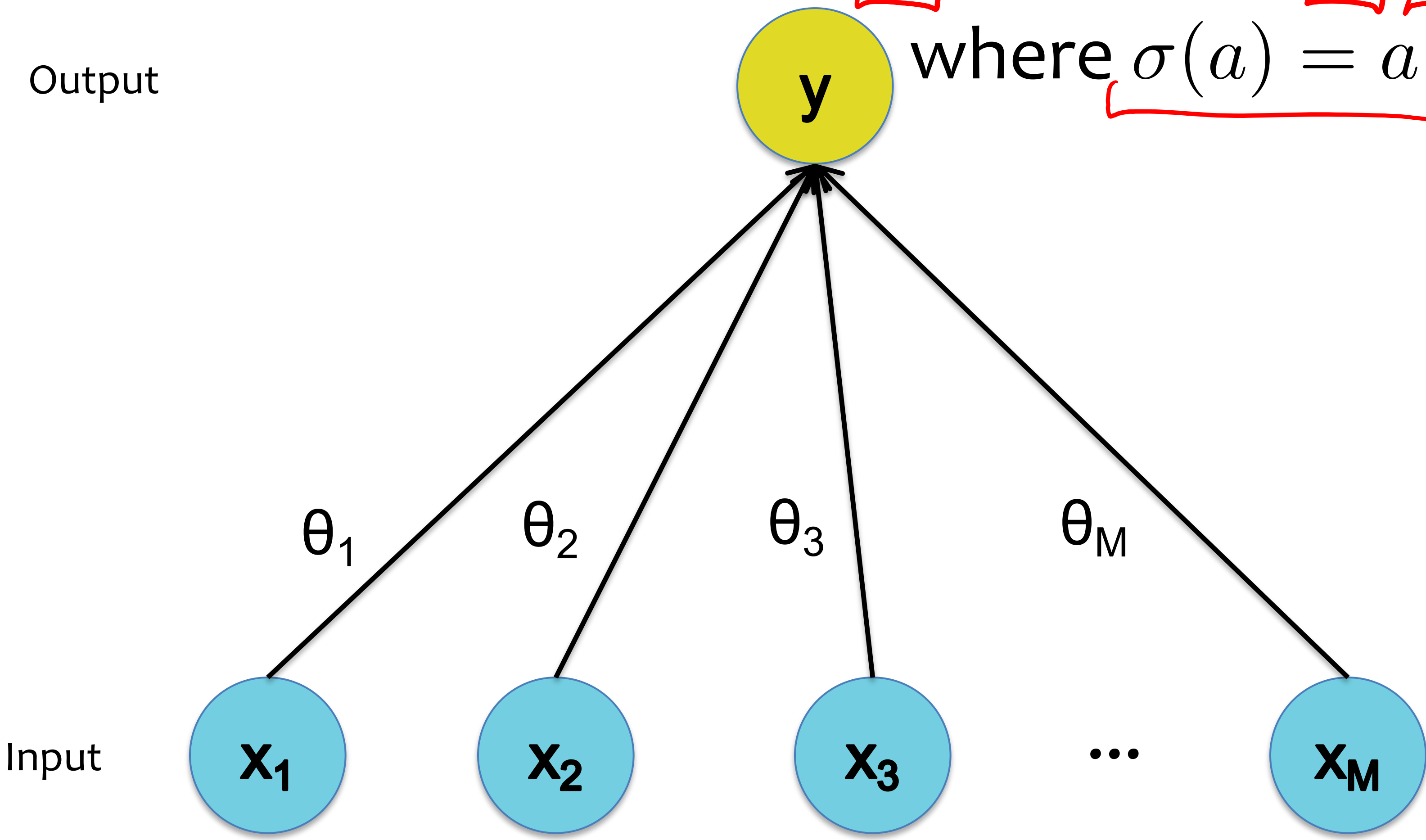
ke small steps opposite the gradient)

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

Linear Regression

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

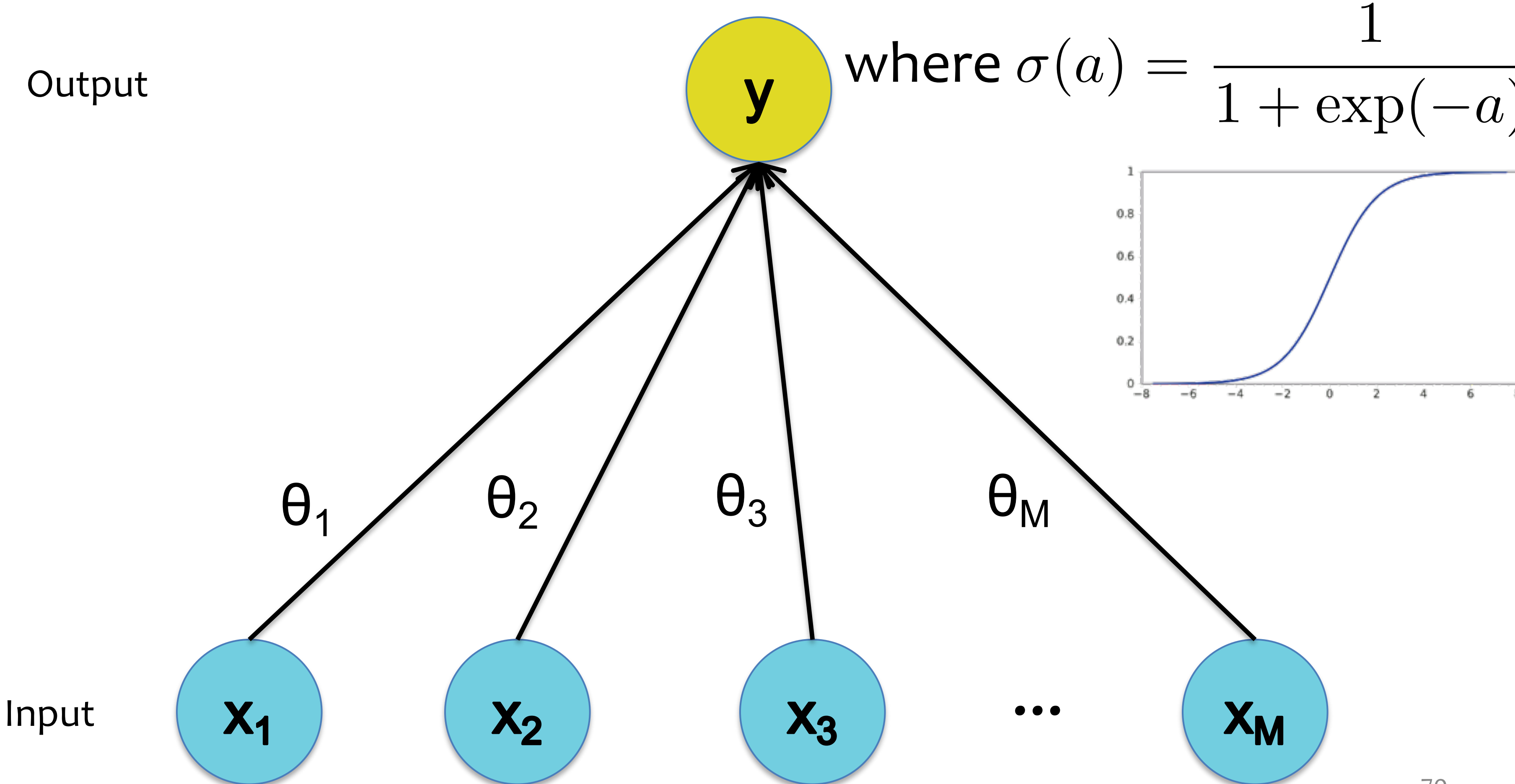
where $\sigma(a) = a$



Logistic Regression

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

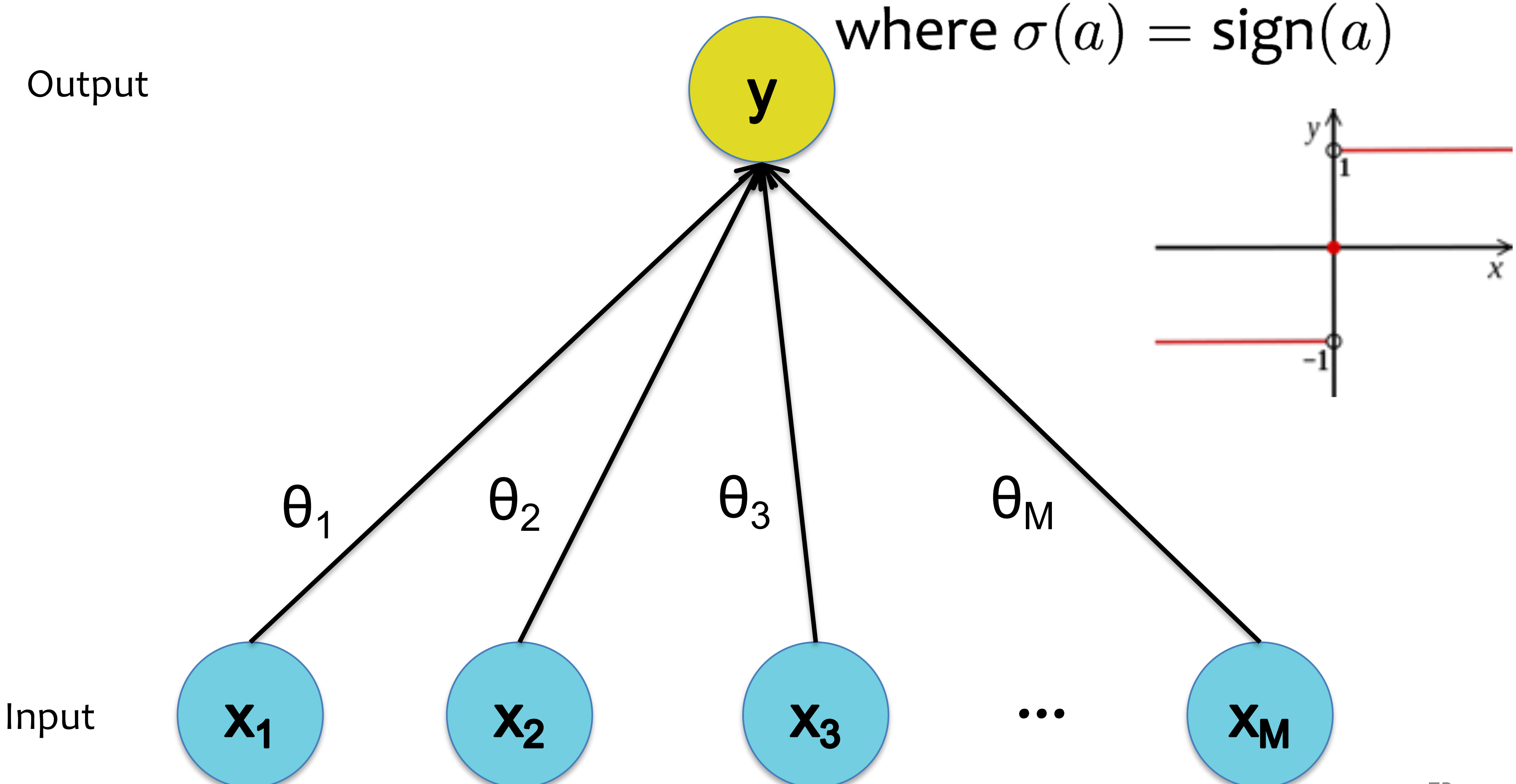
where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



Perceptron

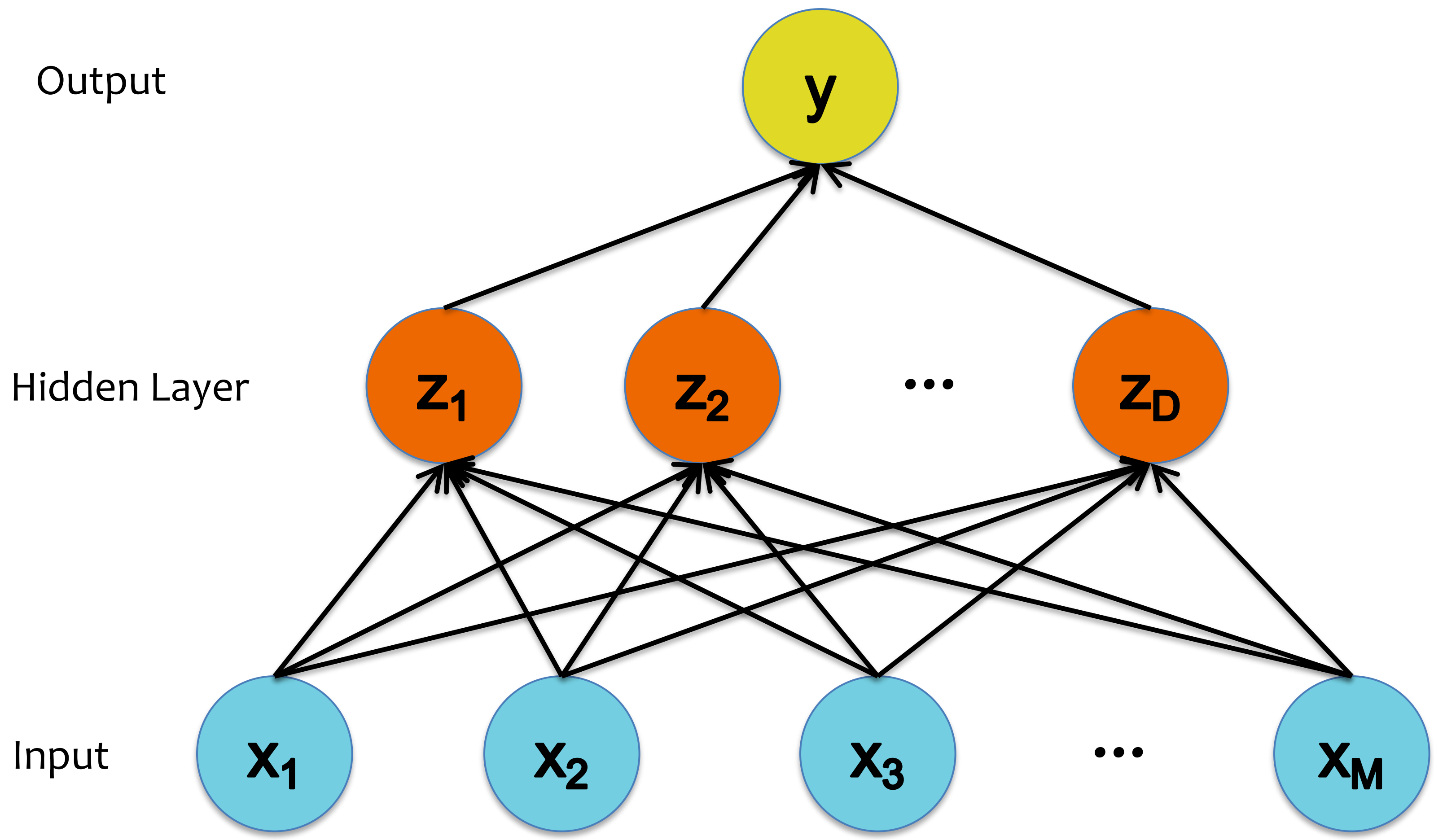
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

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



Decision Functions

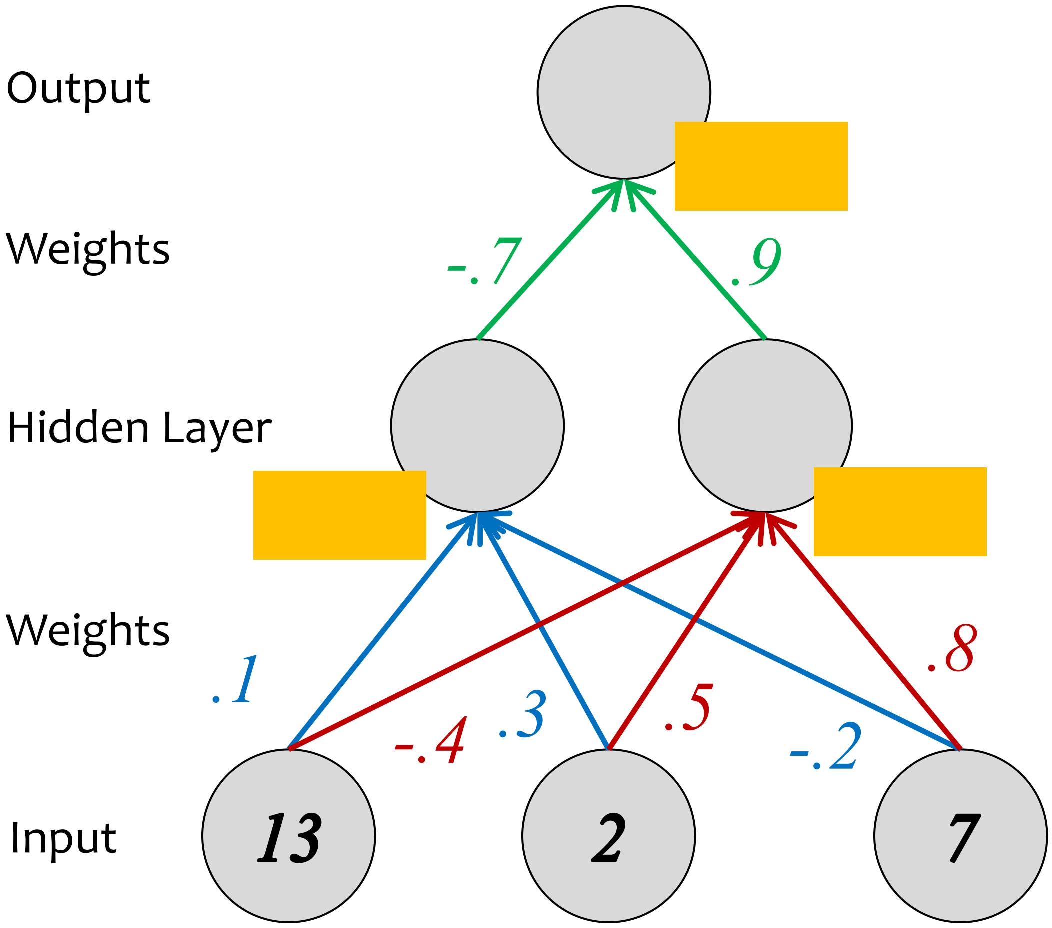
Neural Network



COMPONENTS OF A NEURAL NETWORK

Decision Functions

Neural Network

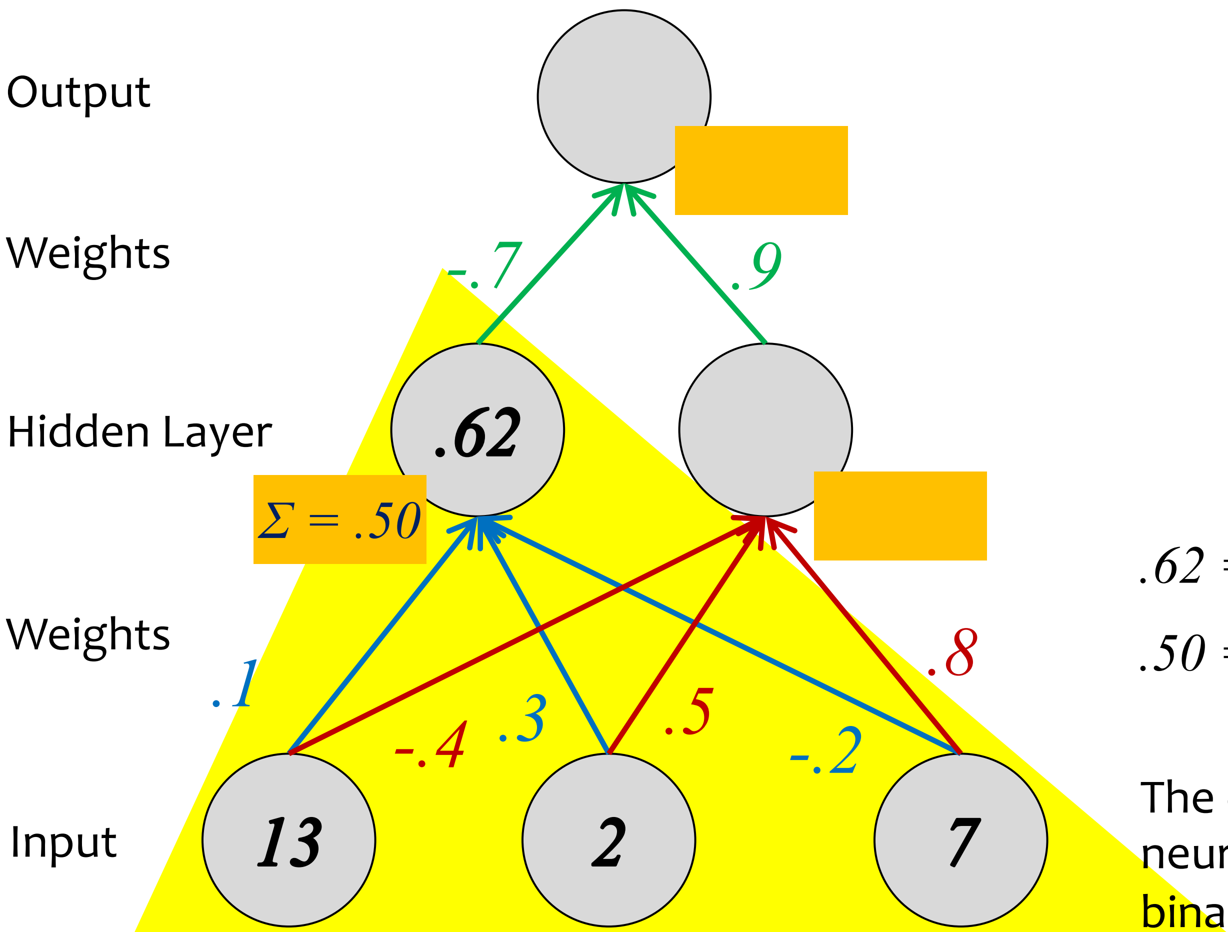


Suppose we already learned the weights of the neural network.

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



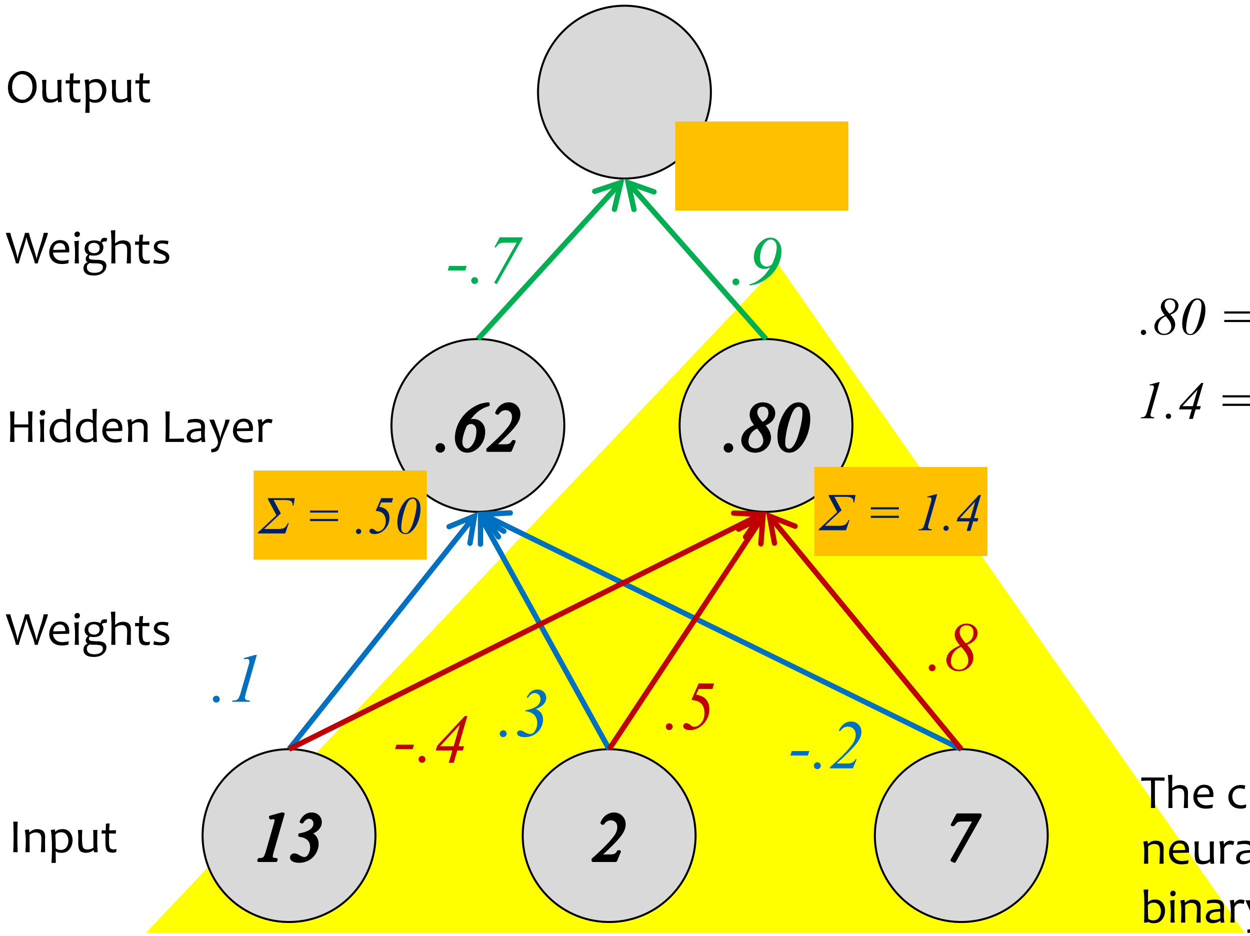
$$.62 = \sigma(.50)$$

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

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

Decision Functions

Neural Network

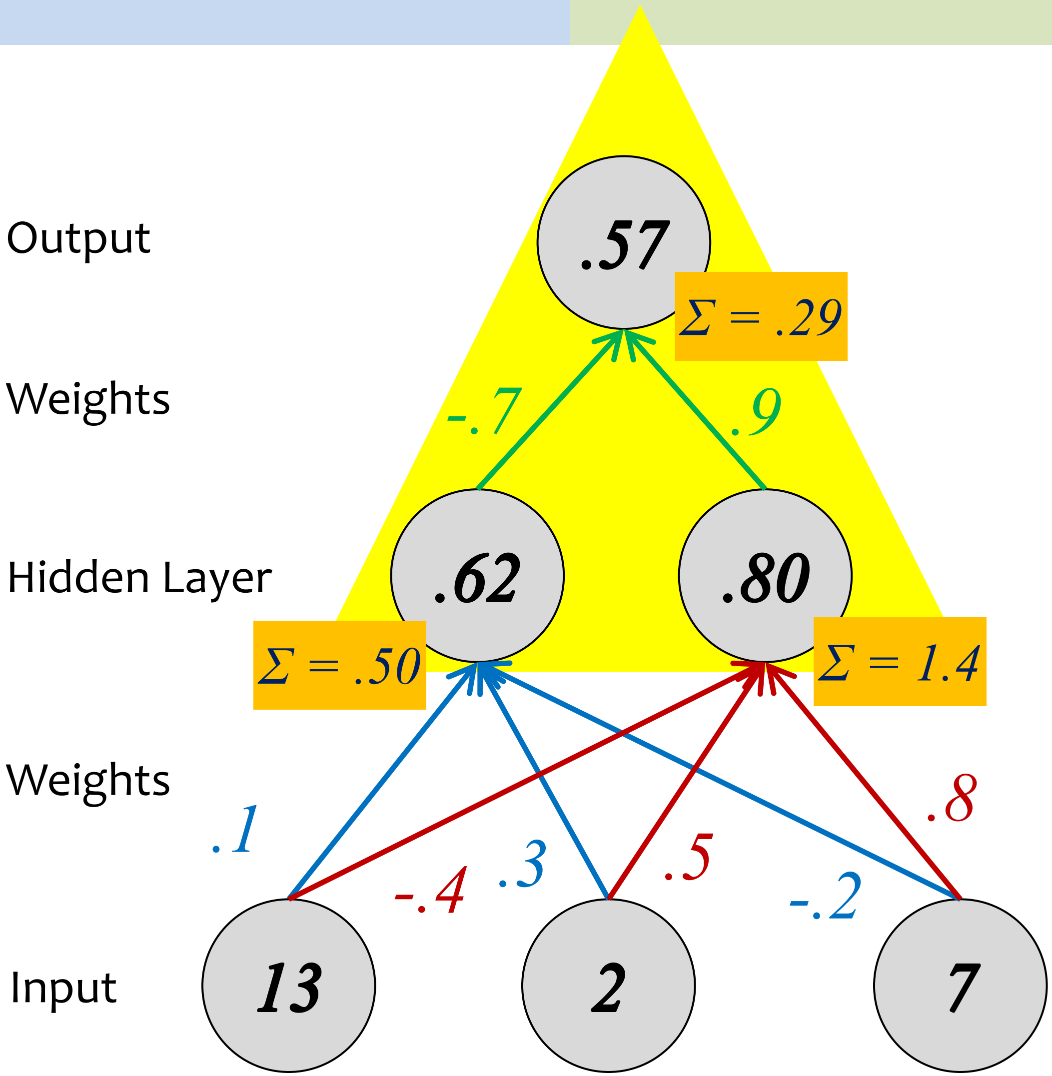


$$.80 = \sigma(1.4)$$

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

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

Neural Network



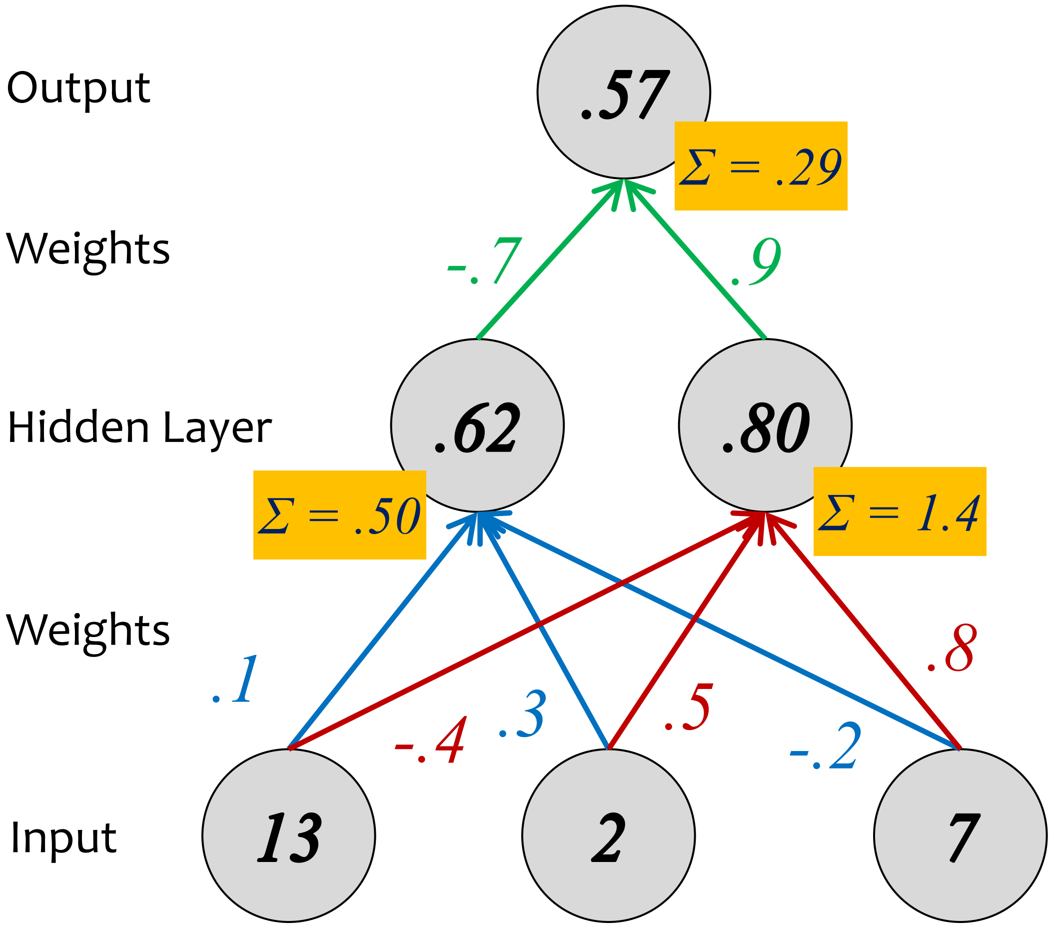
$$.57 = \sigma(.29)$$

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

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

Decision Functions

Neural Network



$.57 = \sigma(.29)$

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

$.80 = \sigma(1.4)$

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

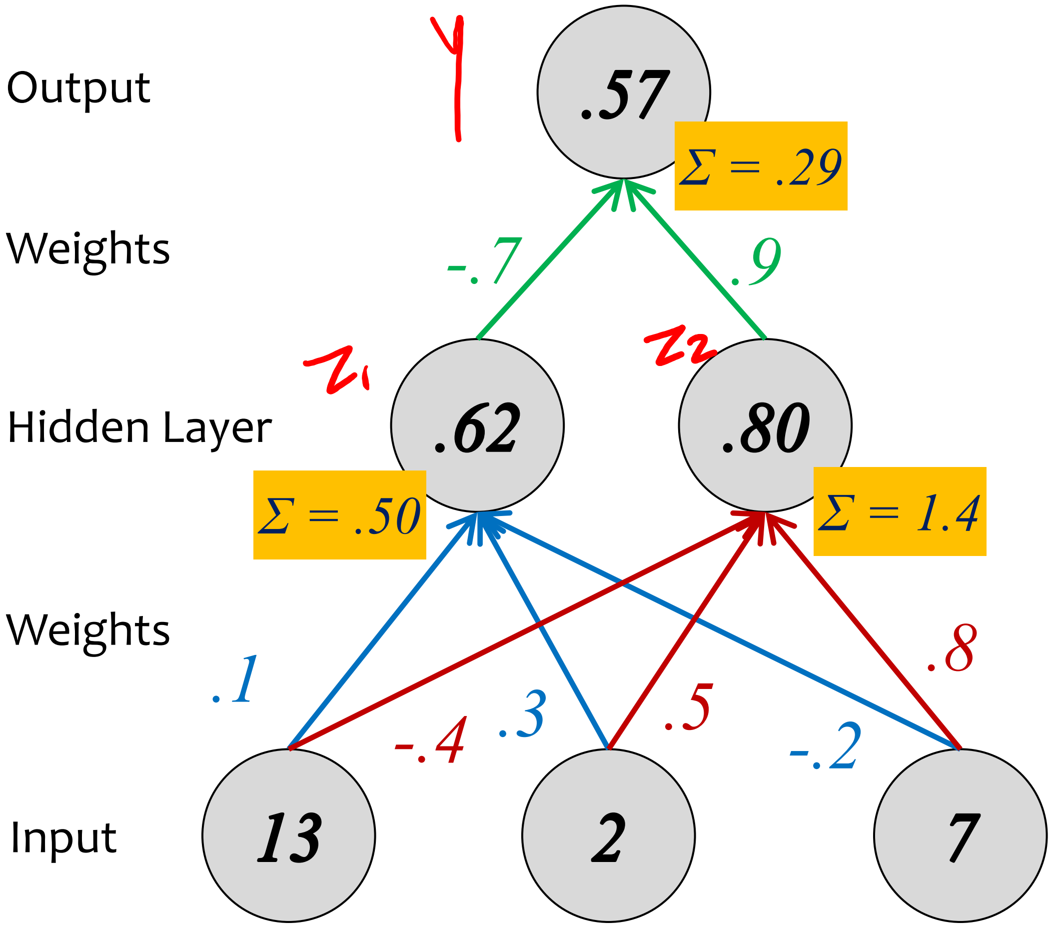
$.62 = \sigma(.50)$

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

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

Decision Functions

Neural Network



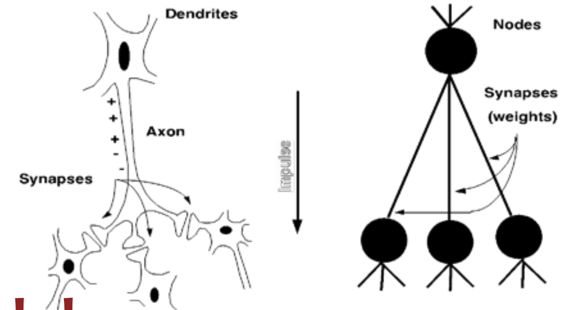
Except we only have the target value for y at training time!

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^{4-5}
- 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

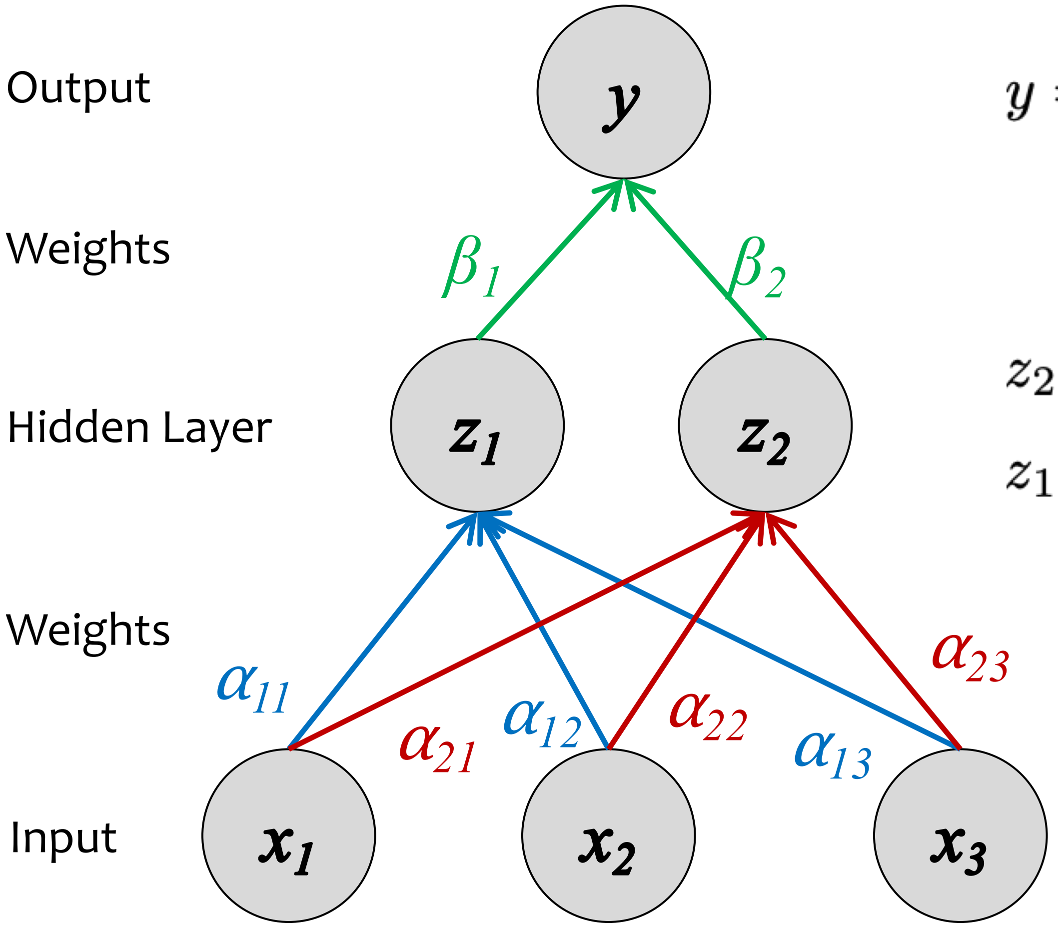
DEFINING A 1-HIDDEN LAYER NEURAL NETWORK

Neural Networks

Chalkboard

- Example: Neural Network w/1 Hidden Layer

Neural Network

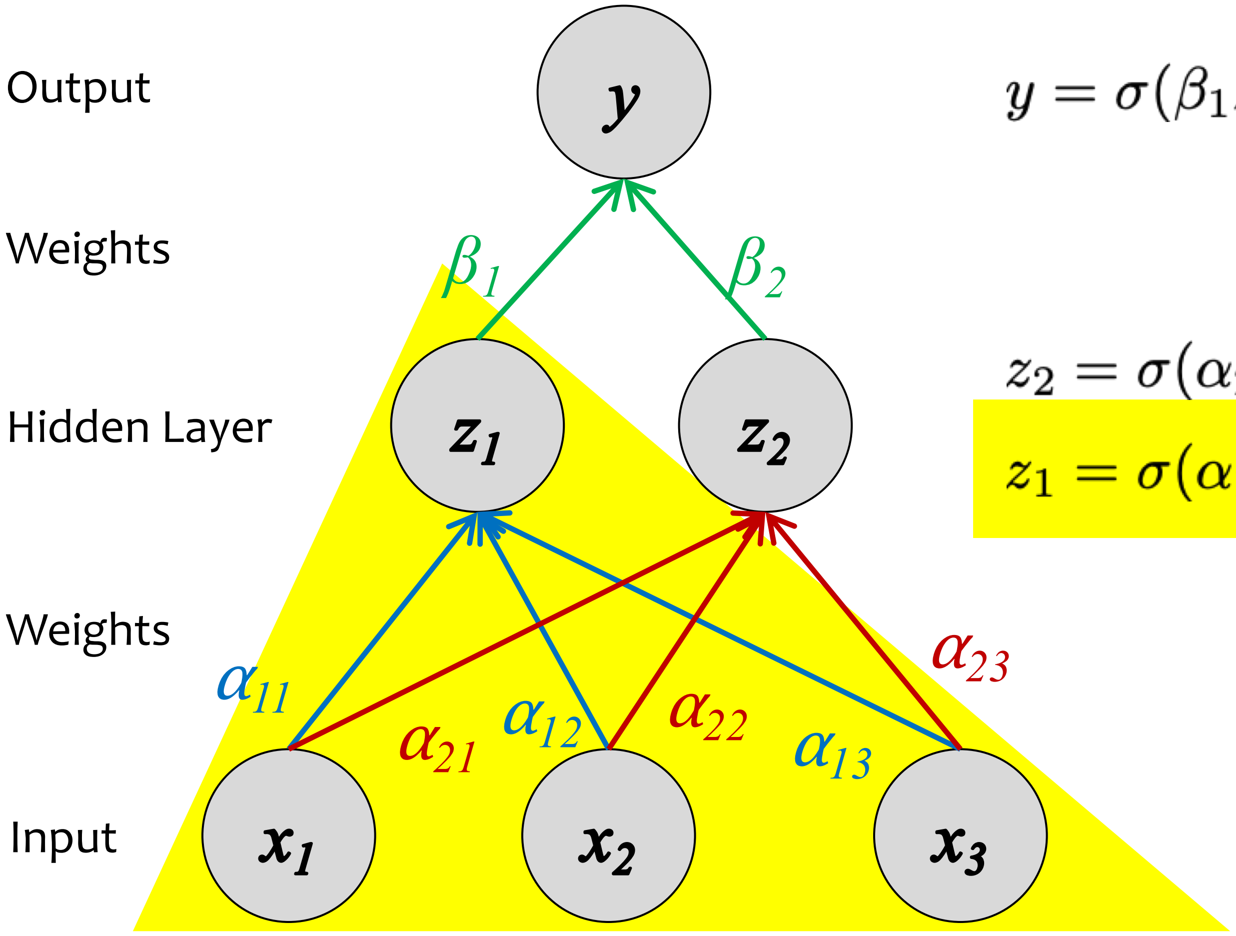


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Neural Network



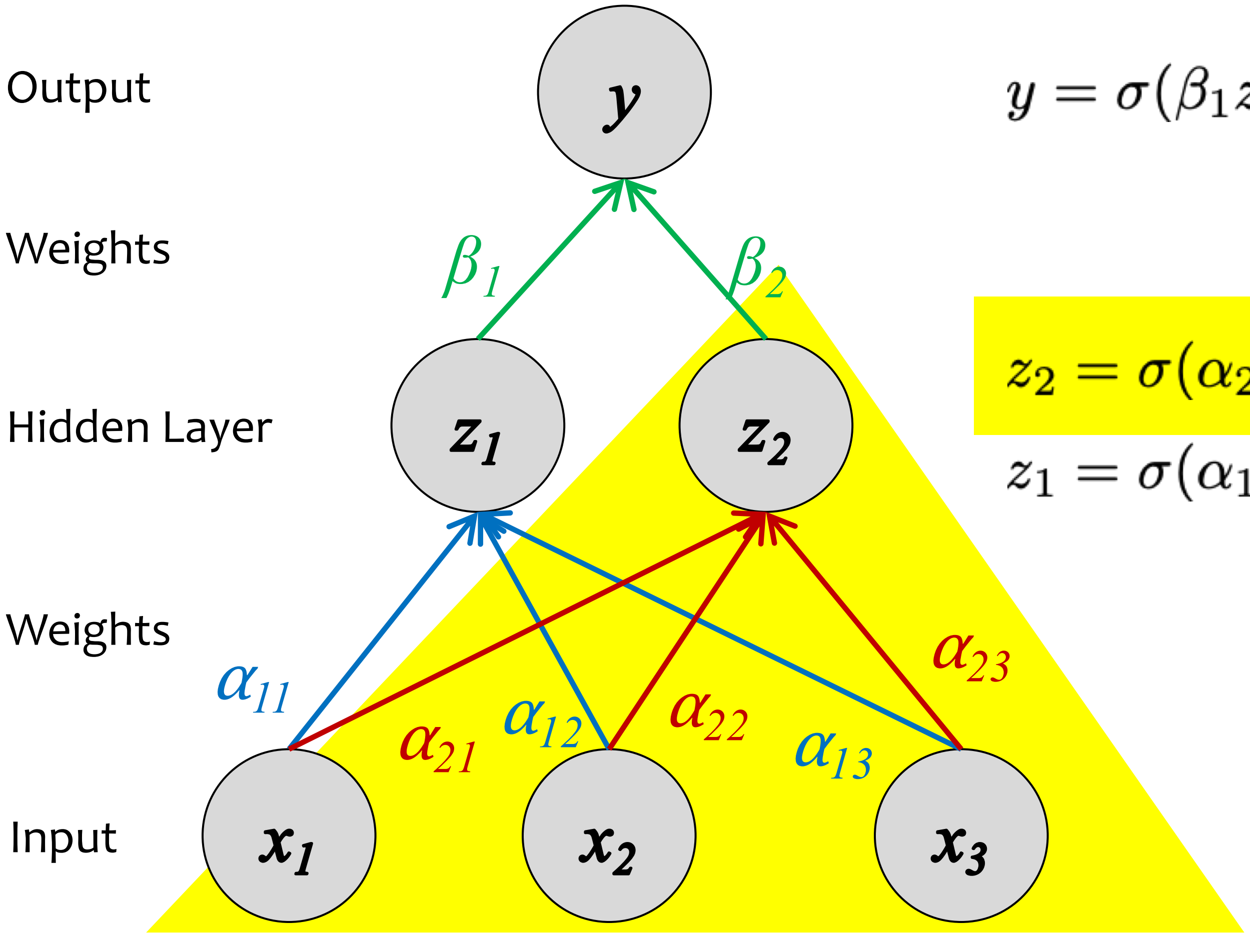
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network



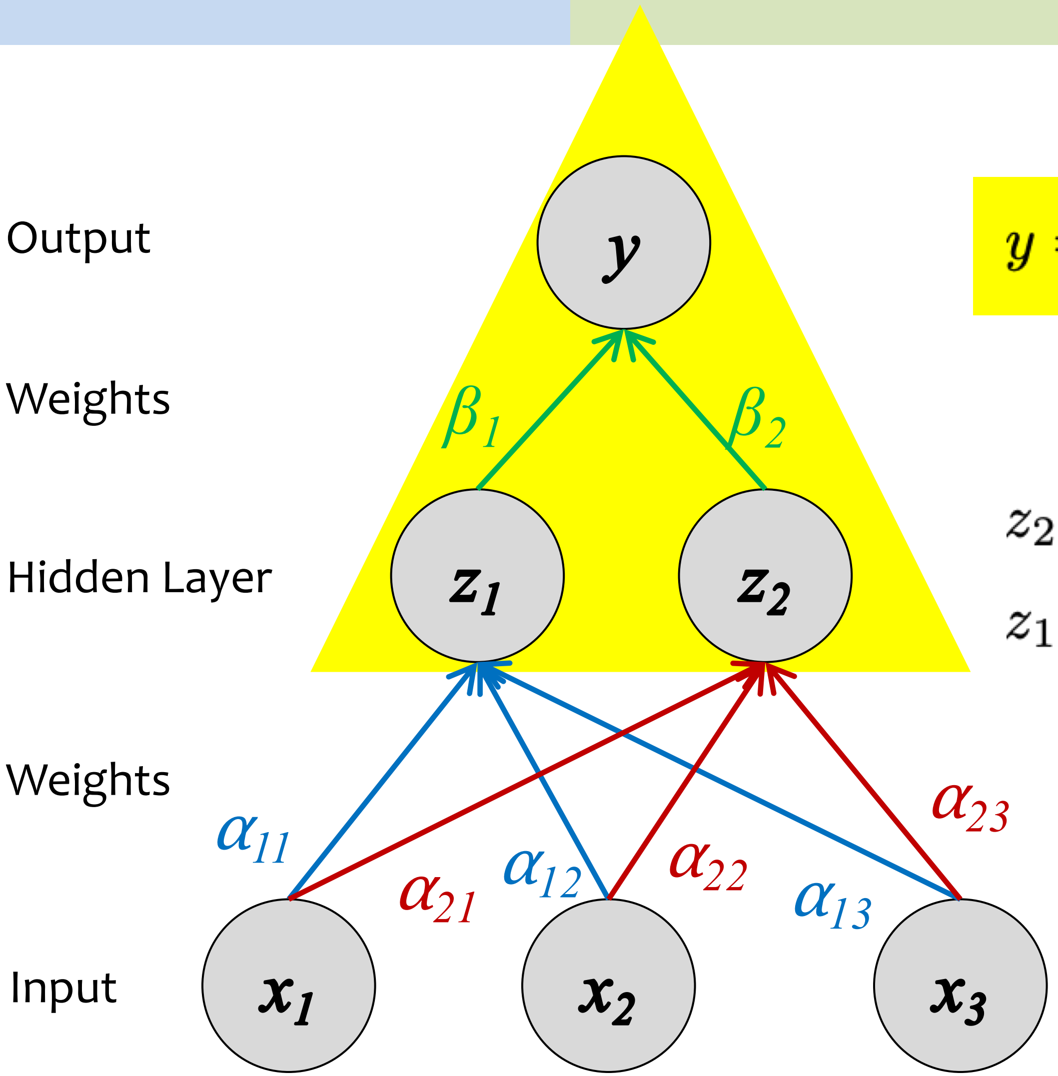
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network



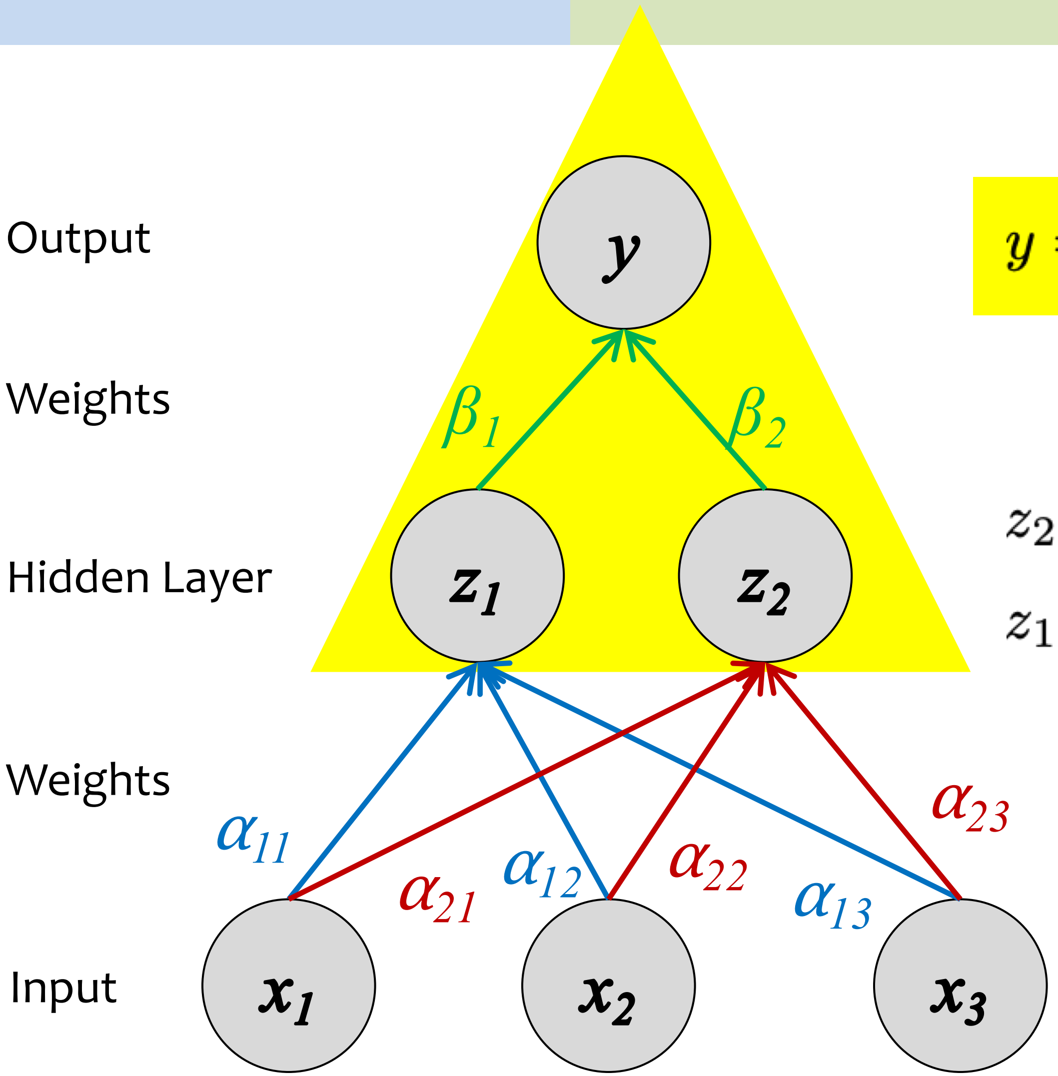
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network

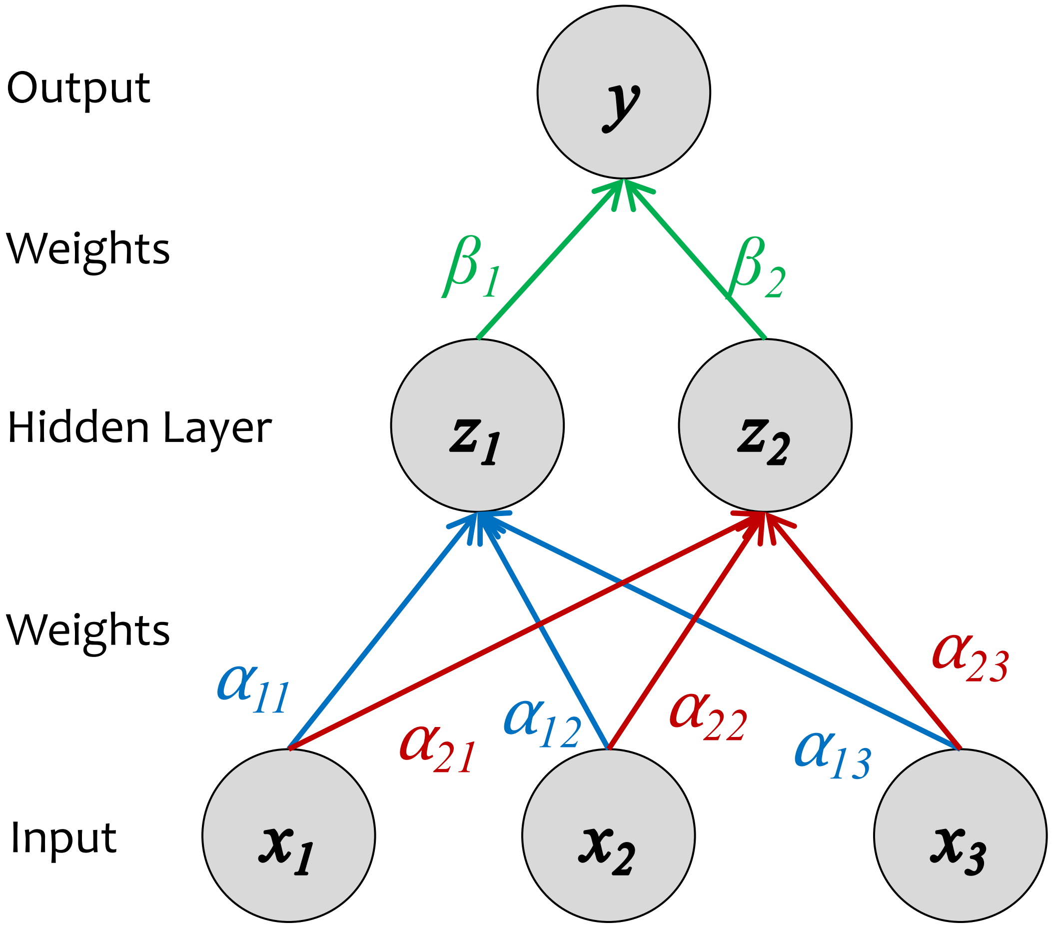


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Neural Network



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

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

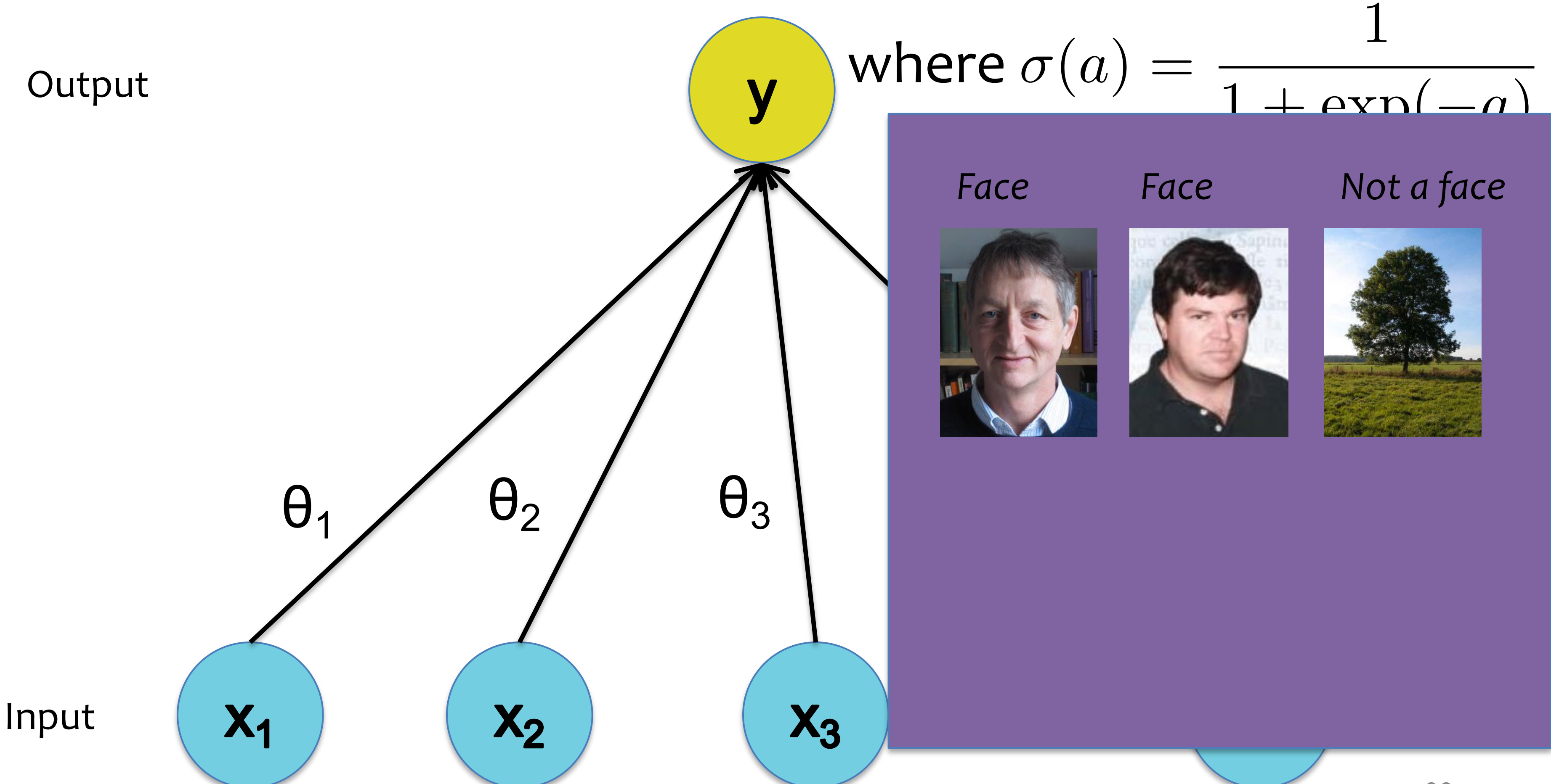
$$z_1 = \sigma(\boldsymbol{\alpha}_{1,\cdot}^T \mathbf{x})$$

NONLINEAR DECISION BOUNDARIES AND NEURAL NETWORKS

Logistic Regression

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

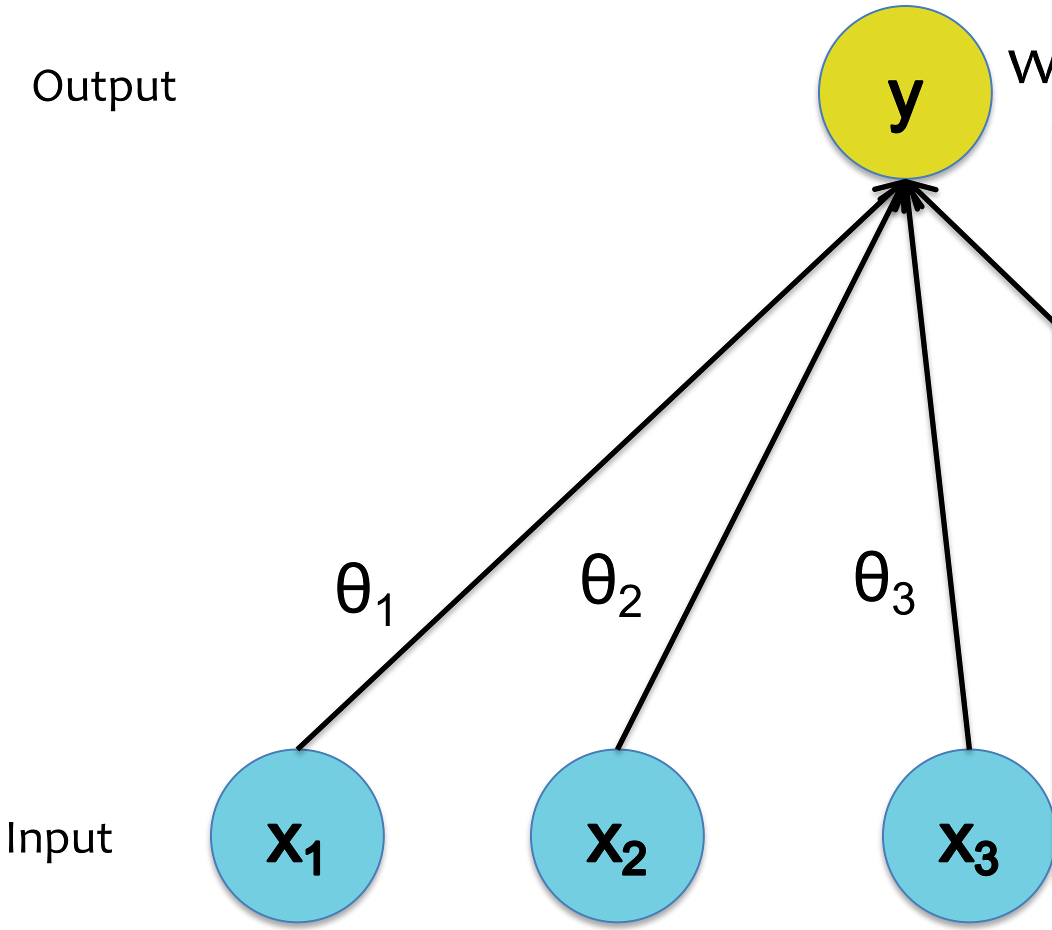
where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}$



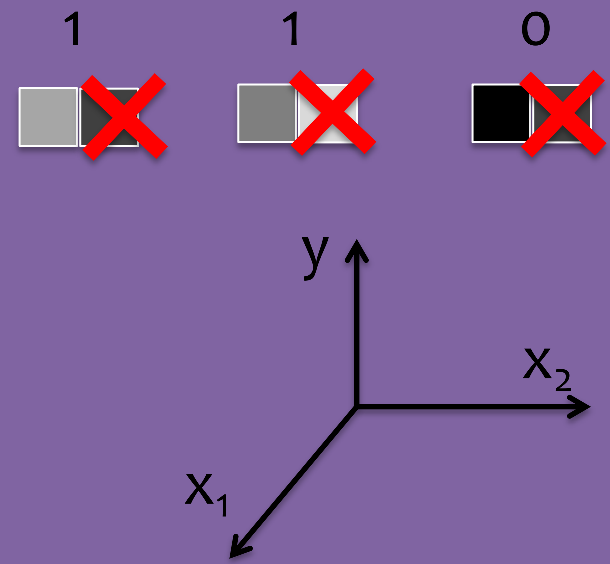
Decision Functions

Logistic Regression

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



In-Class Example



Neural Networks

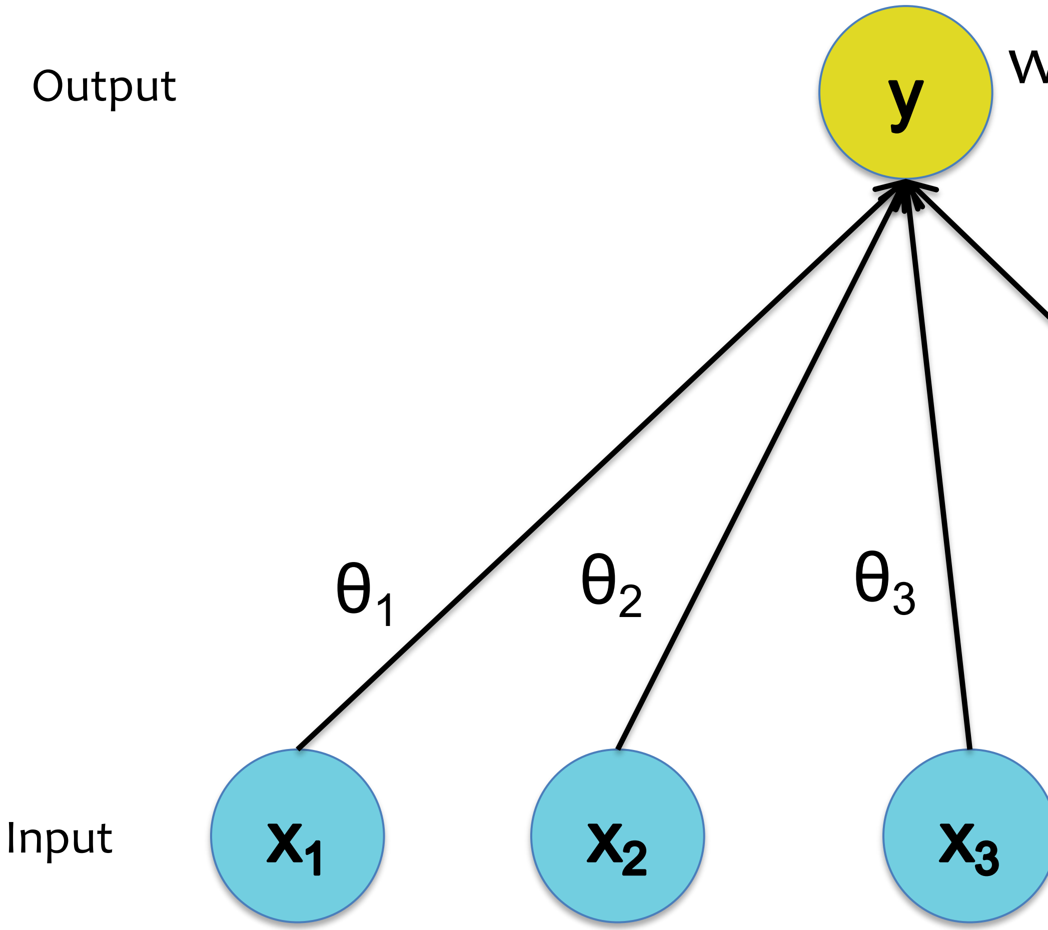
Chalkboard

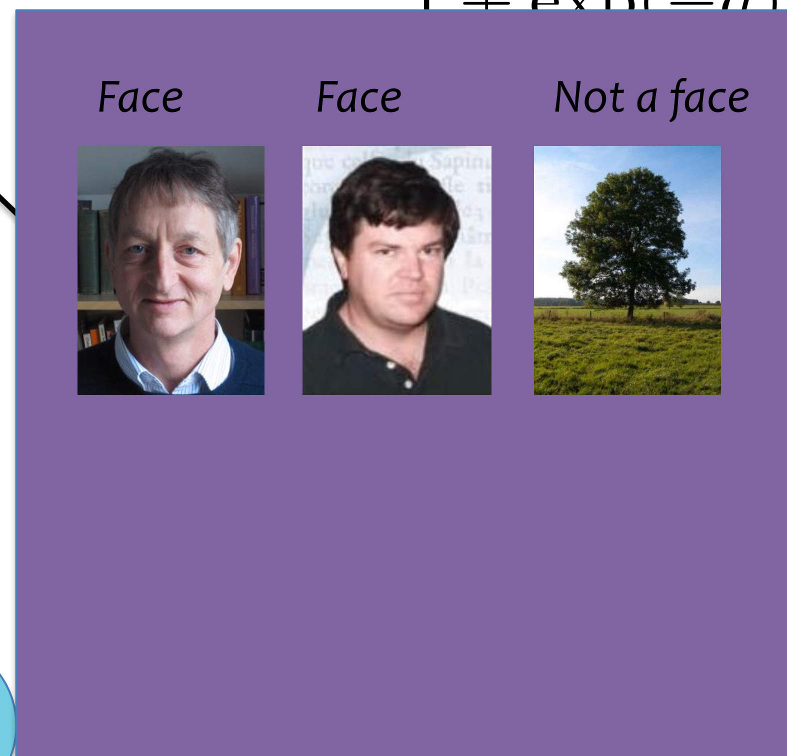
- 1D Example from linear regression to logistic regression
- 1D Example from logistic regression to a neural network

Logistic Regression

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

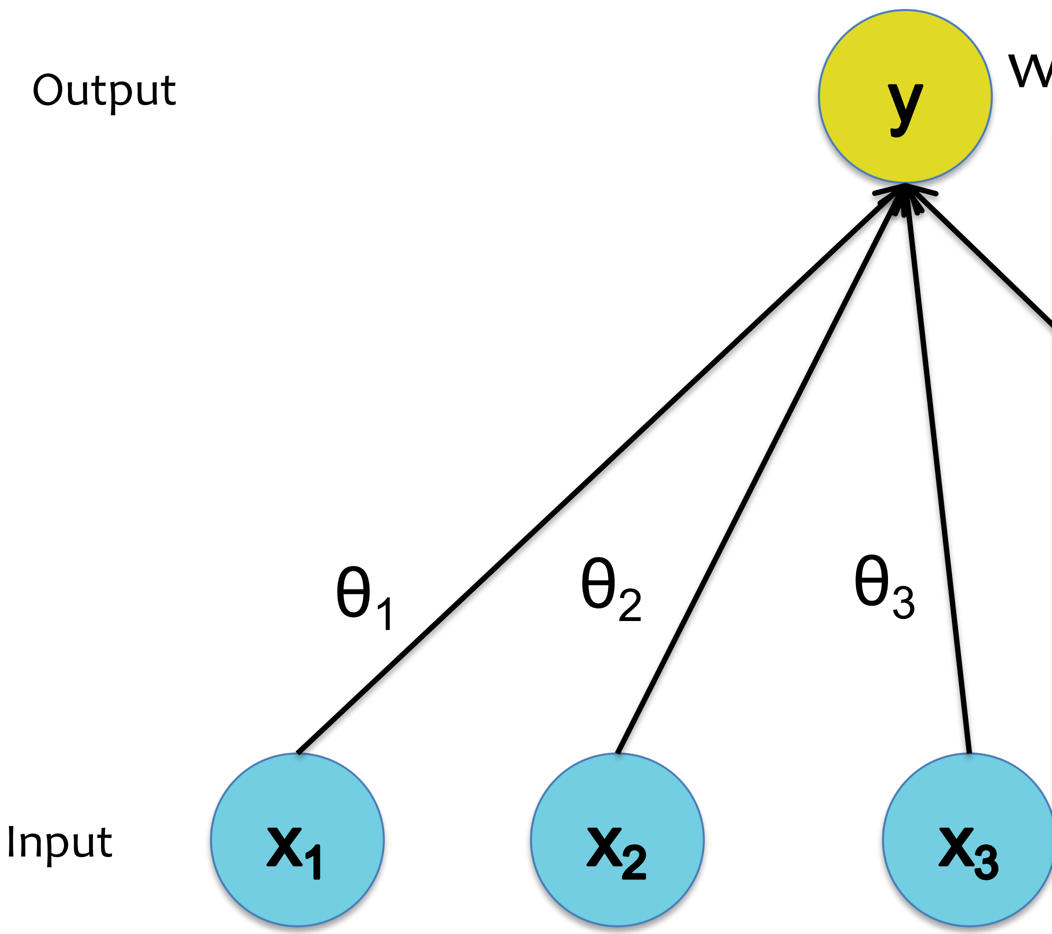
where $\sigma(a) = \frac{1}{1 + \exp(-a)}$



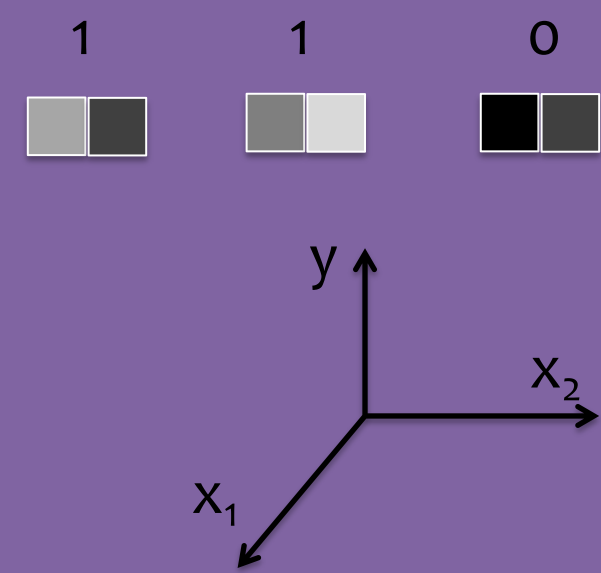


Logistic Regression

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



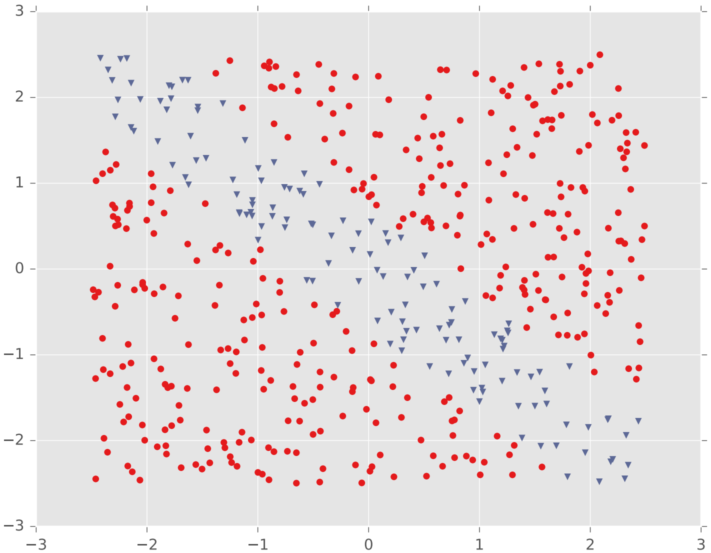
In-Class Example



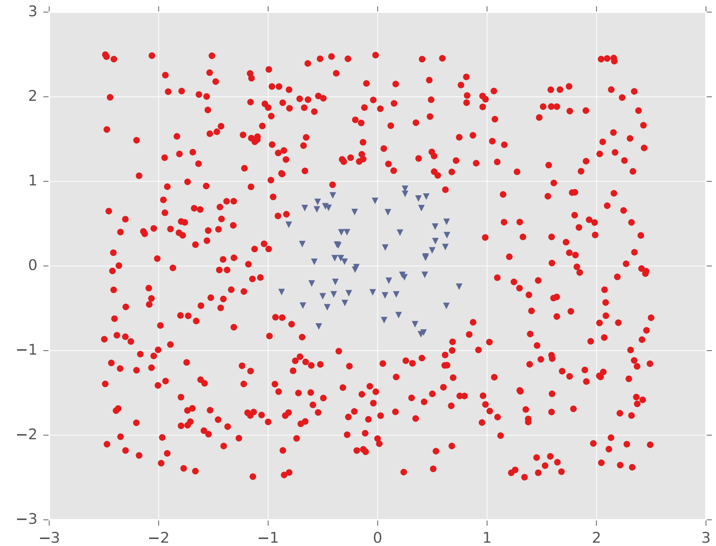
Examples 1 and 2

DECISION BOUNDARY EXAMPLES

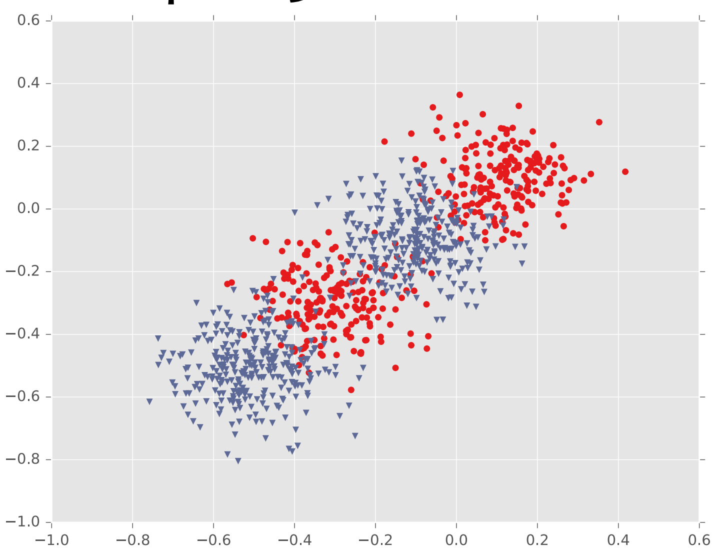
Example #1: Diagonal Band



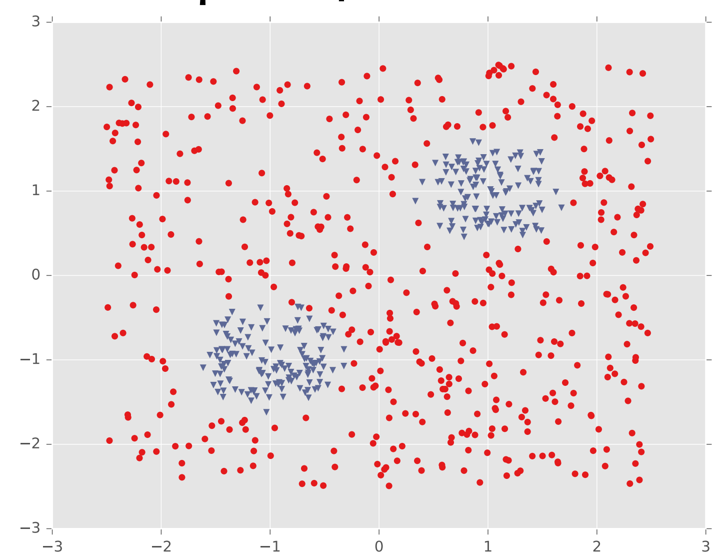
Example #2: One Pocket



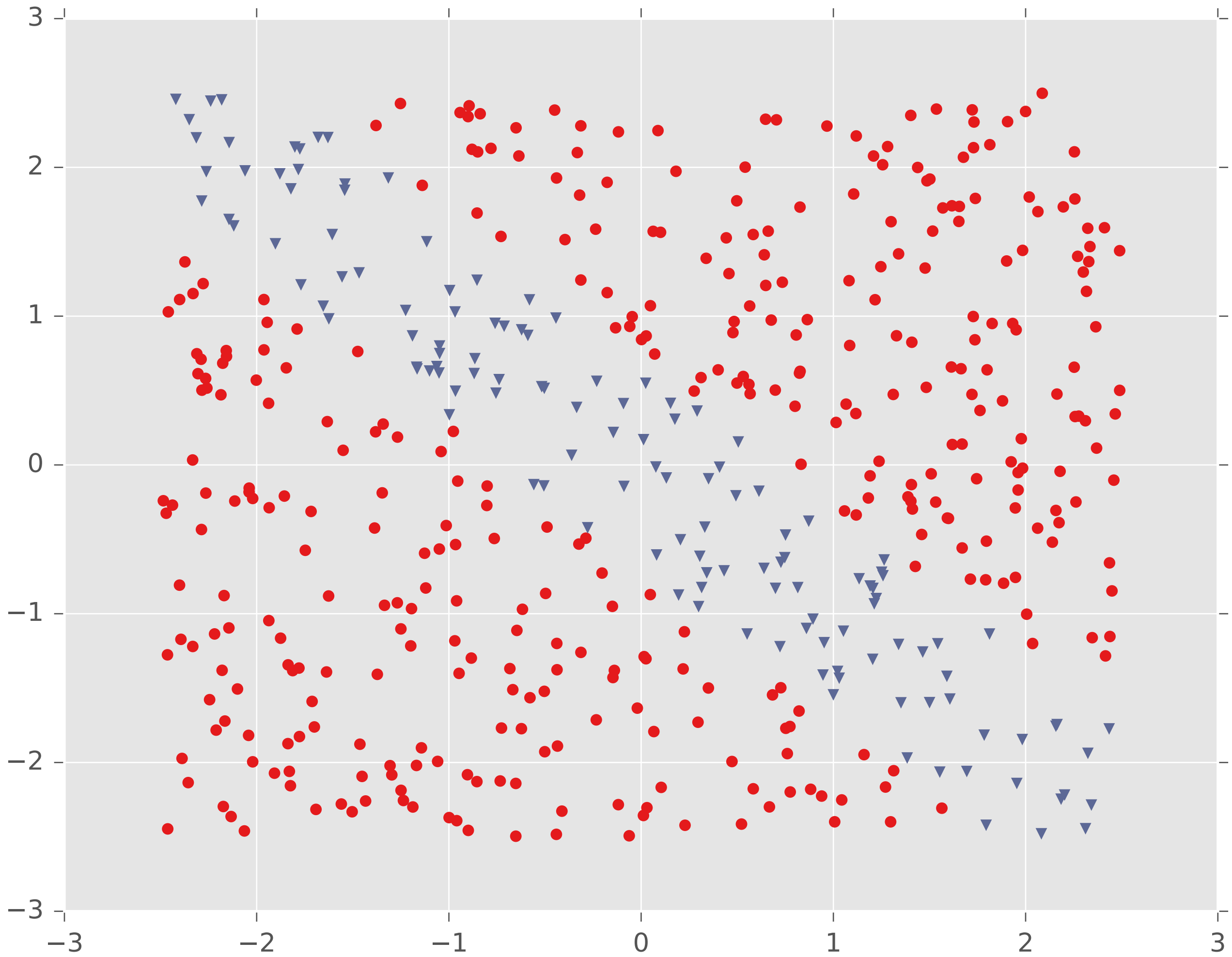
Example #3: Four Gaussians



Example #4: Two Pockets

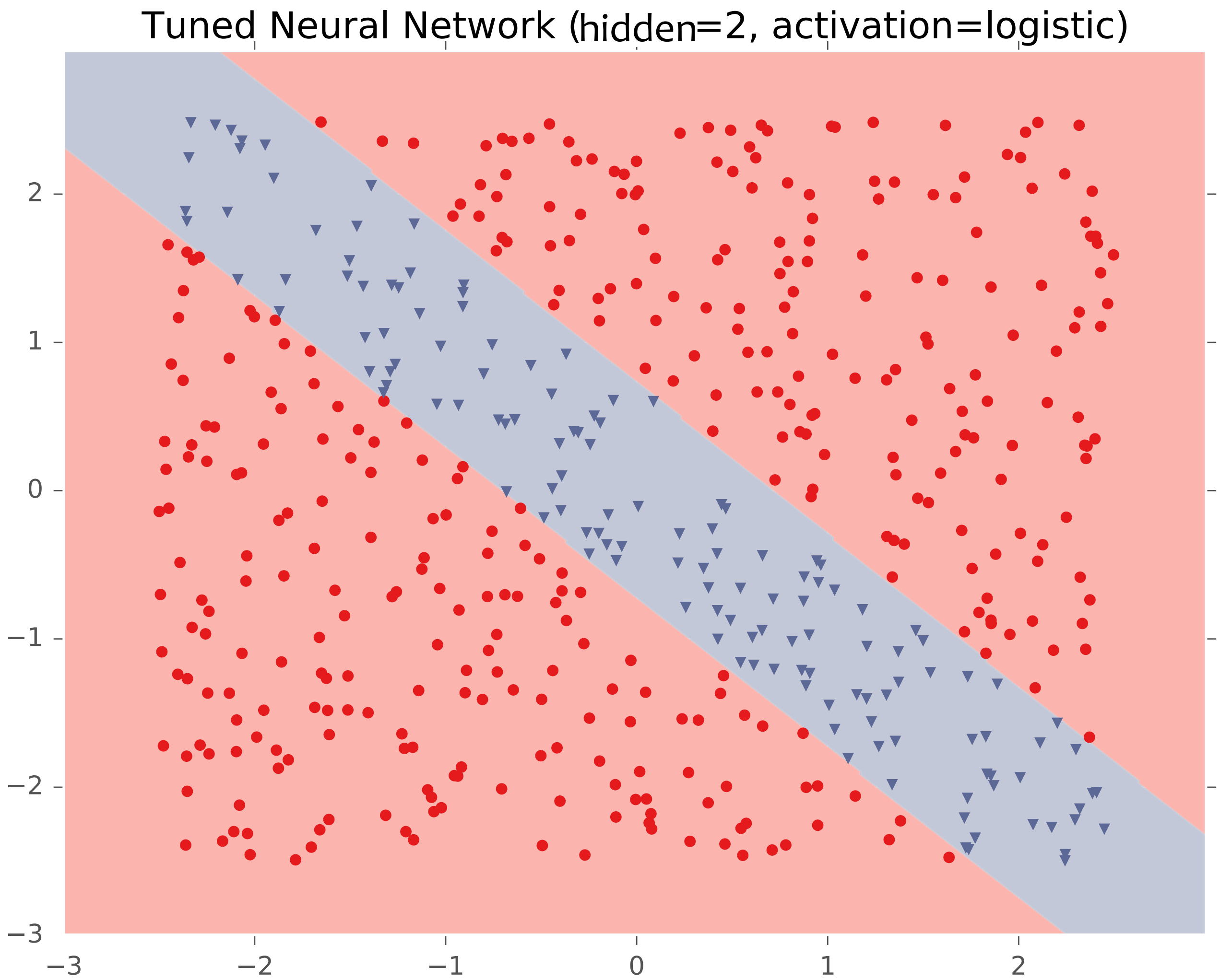


Example #1: Diagonal Band

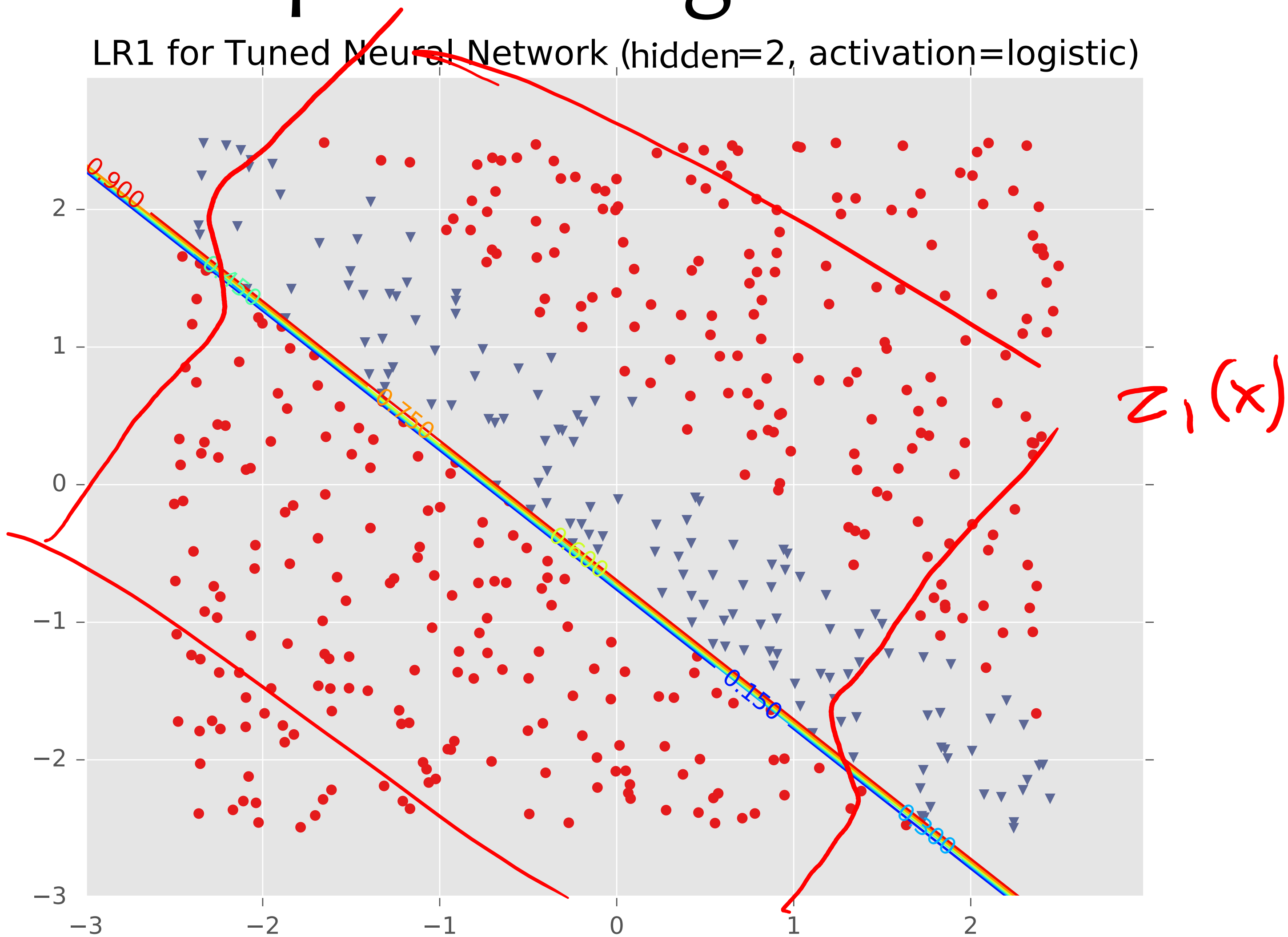


Example #1: Diagonal Band

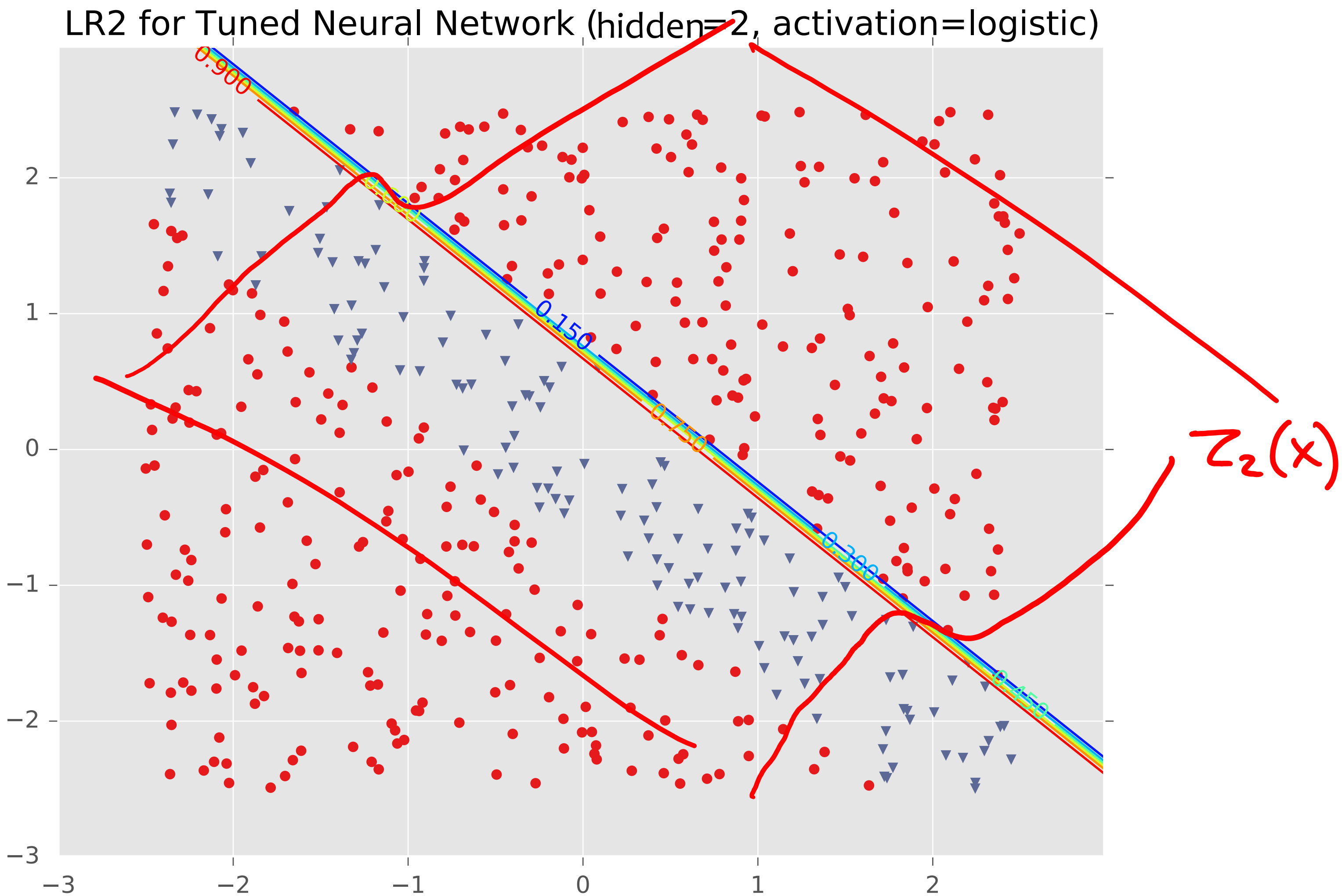
Example #1: Diagonal Band



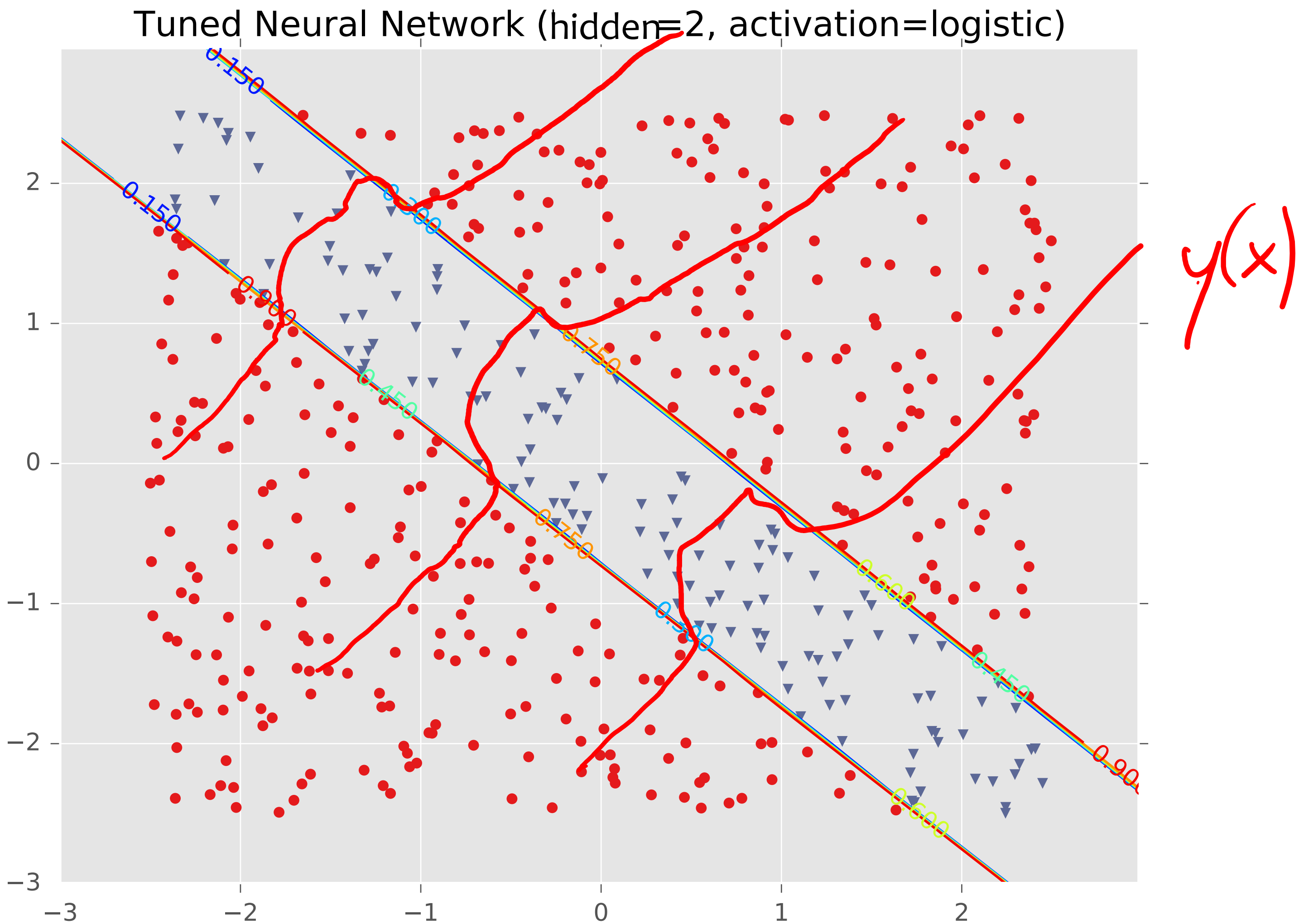
Example #1: Diagonal Band



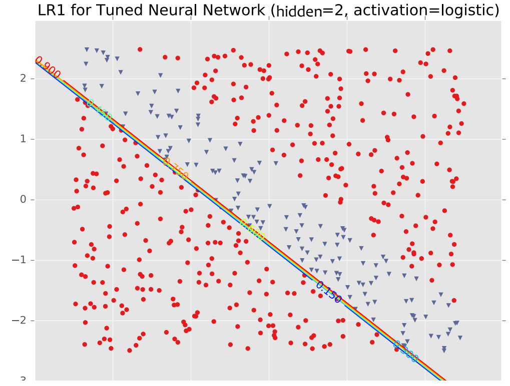
Example #1: Diagonal Band

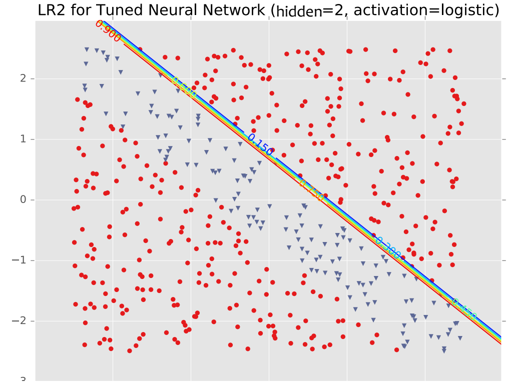


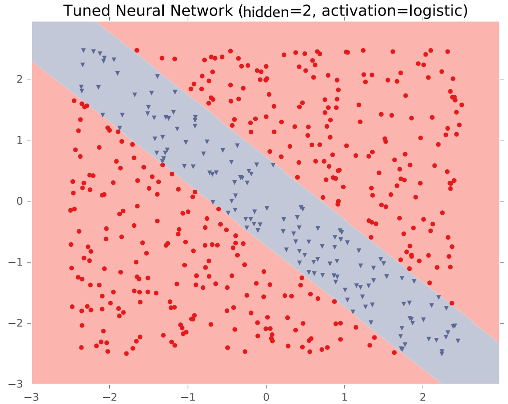
Example #1: Diagonal Band

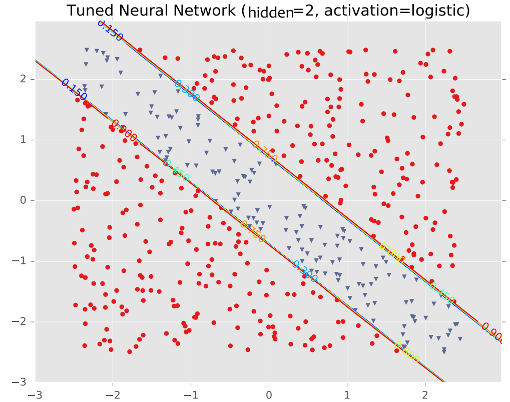


Example #1: Diagonal Band

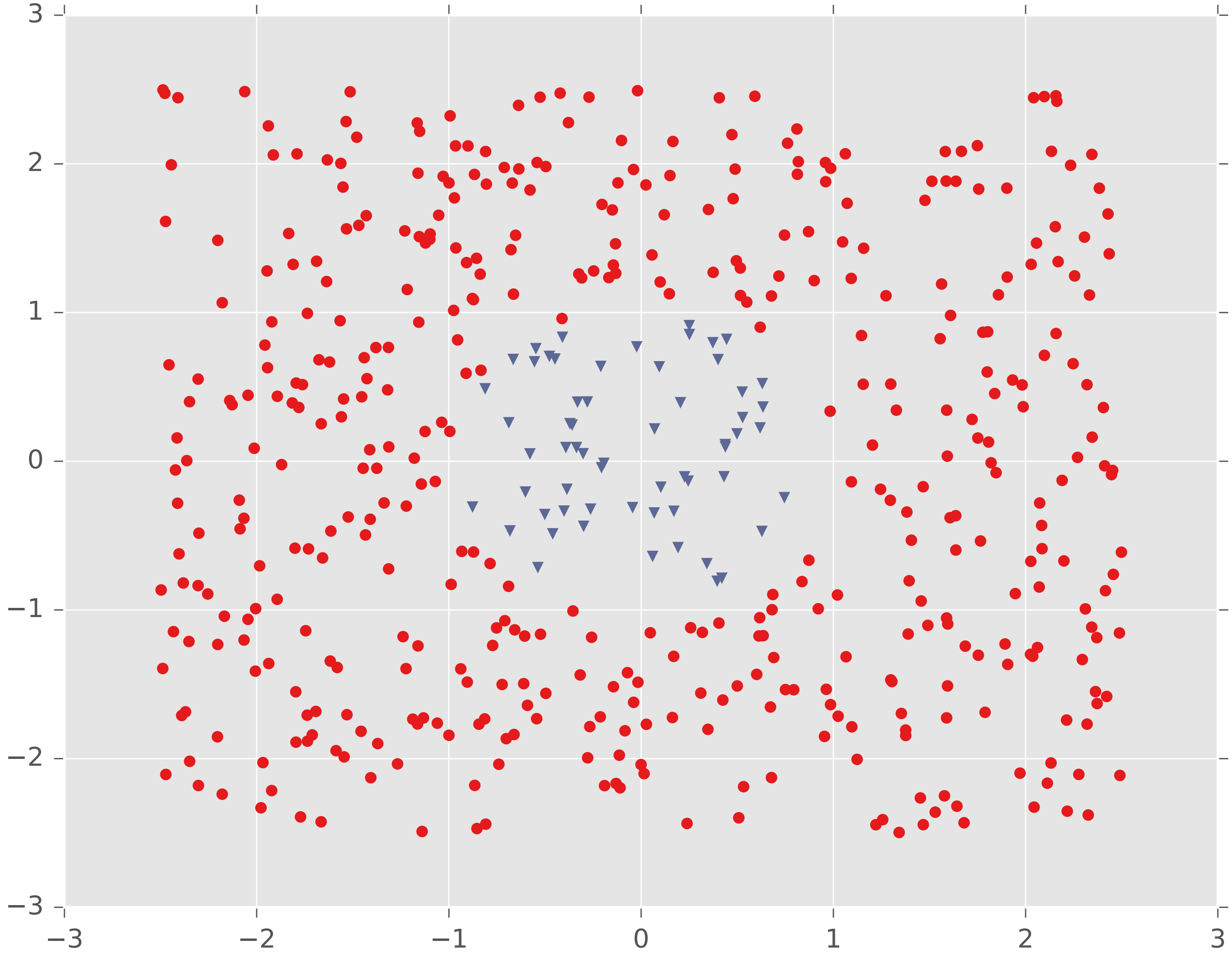






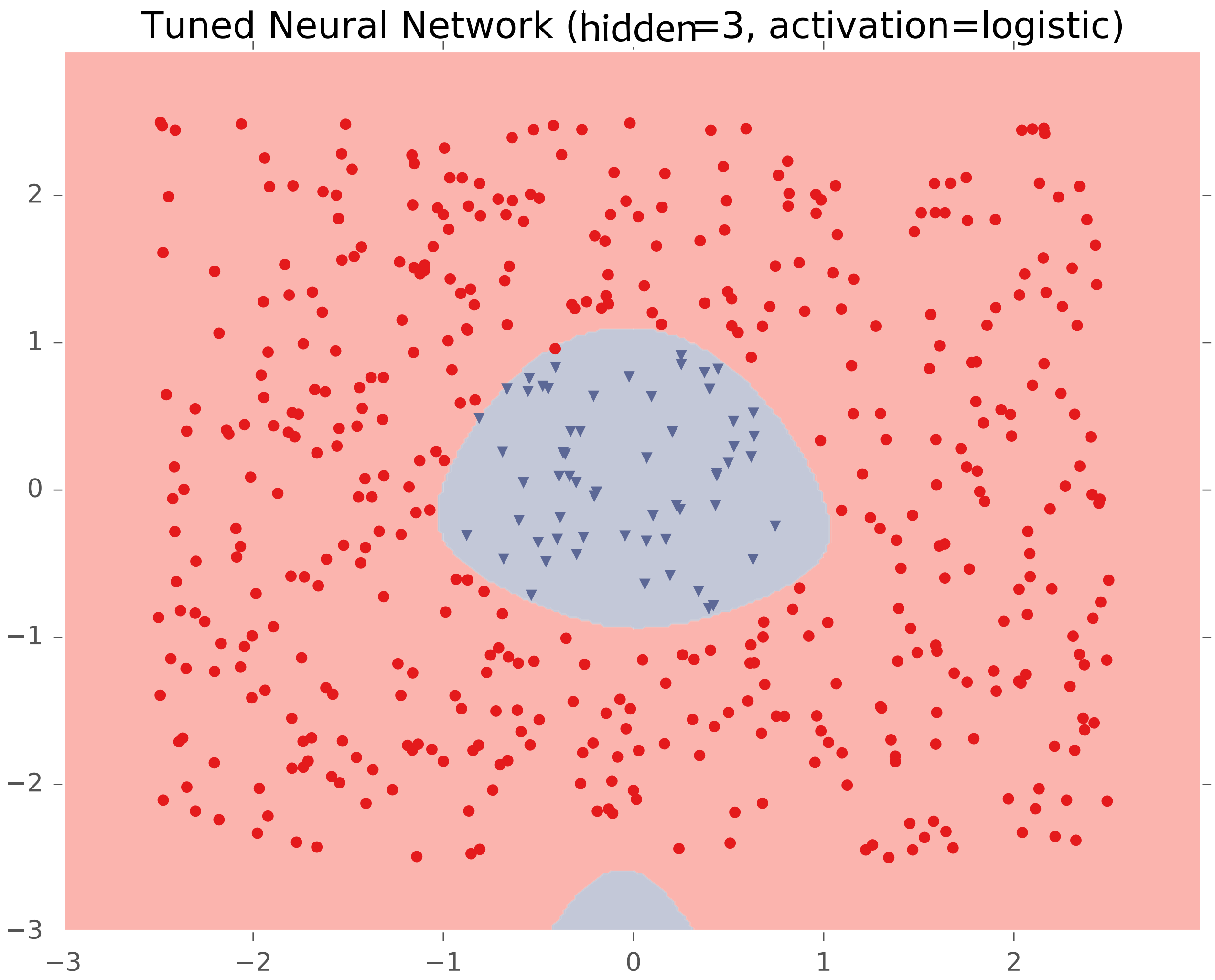


Example #2: One Pocket

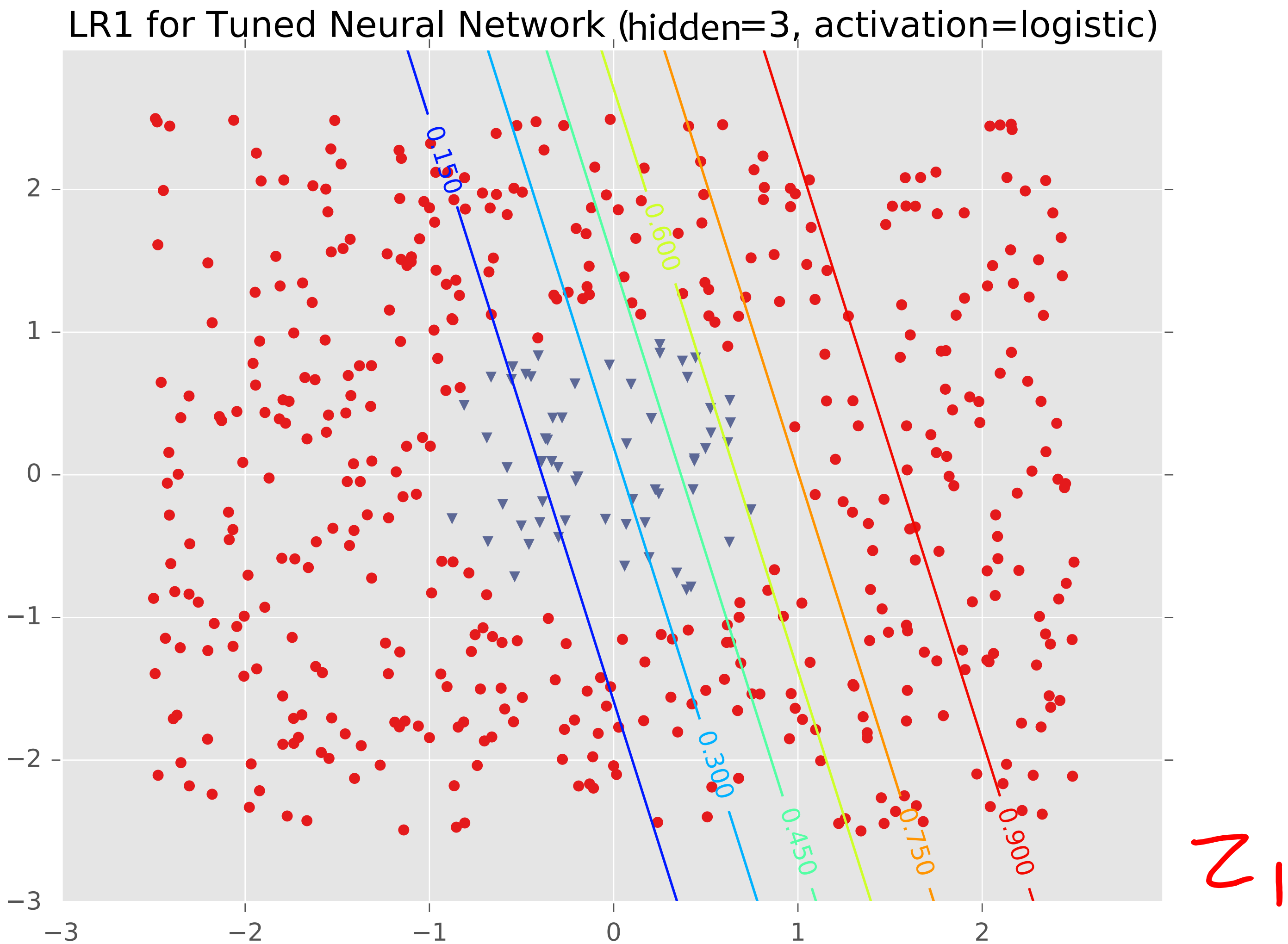


Example #2: One Pocket

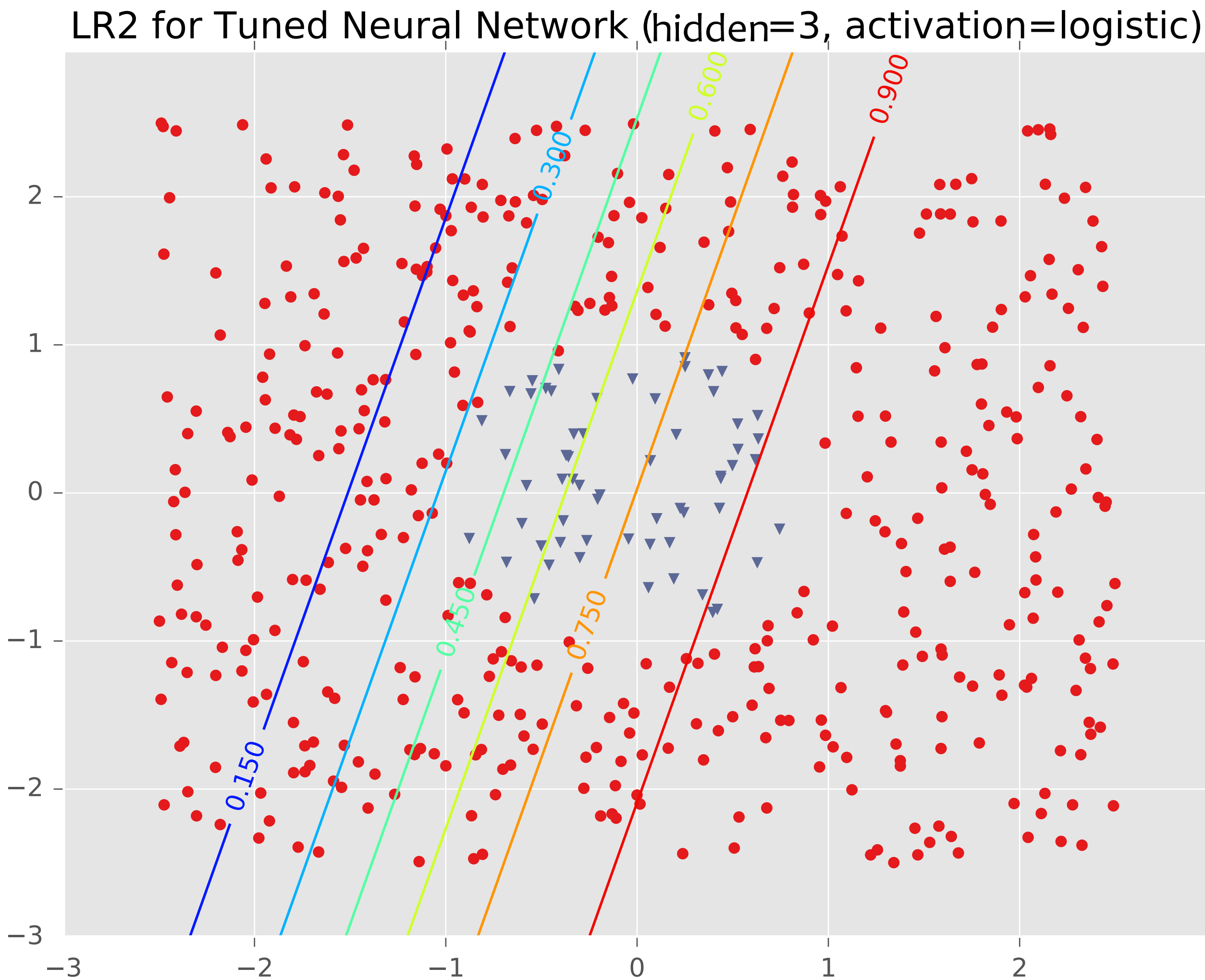
Example #2: One Pocket



Example #2: One Pocket

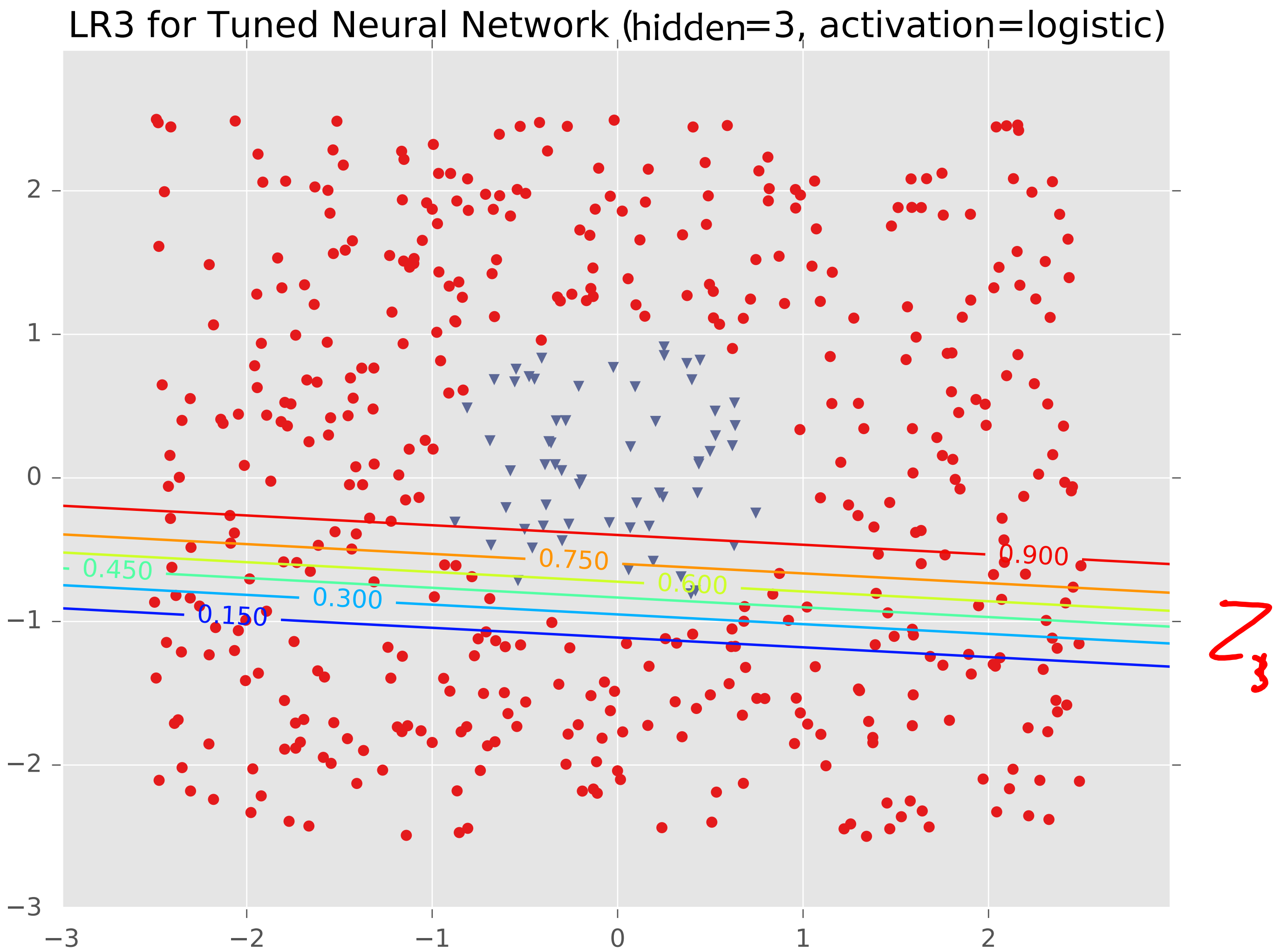


Example #2: One Pocket

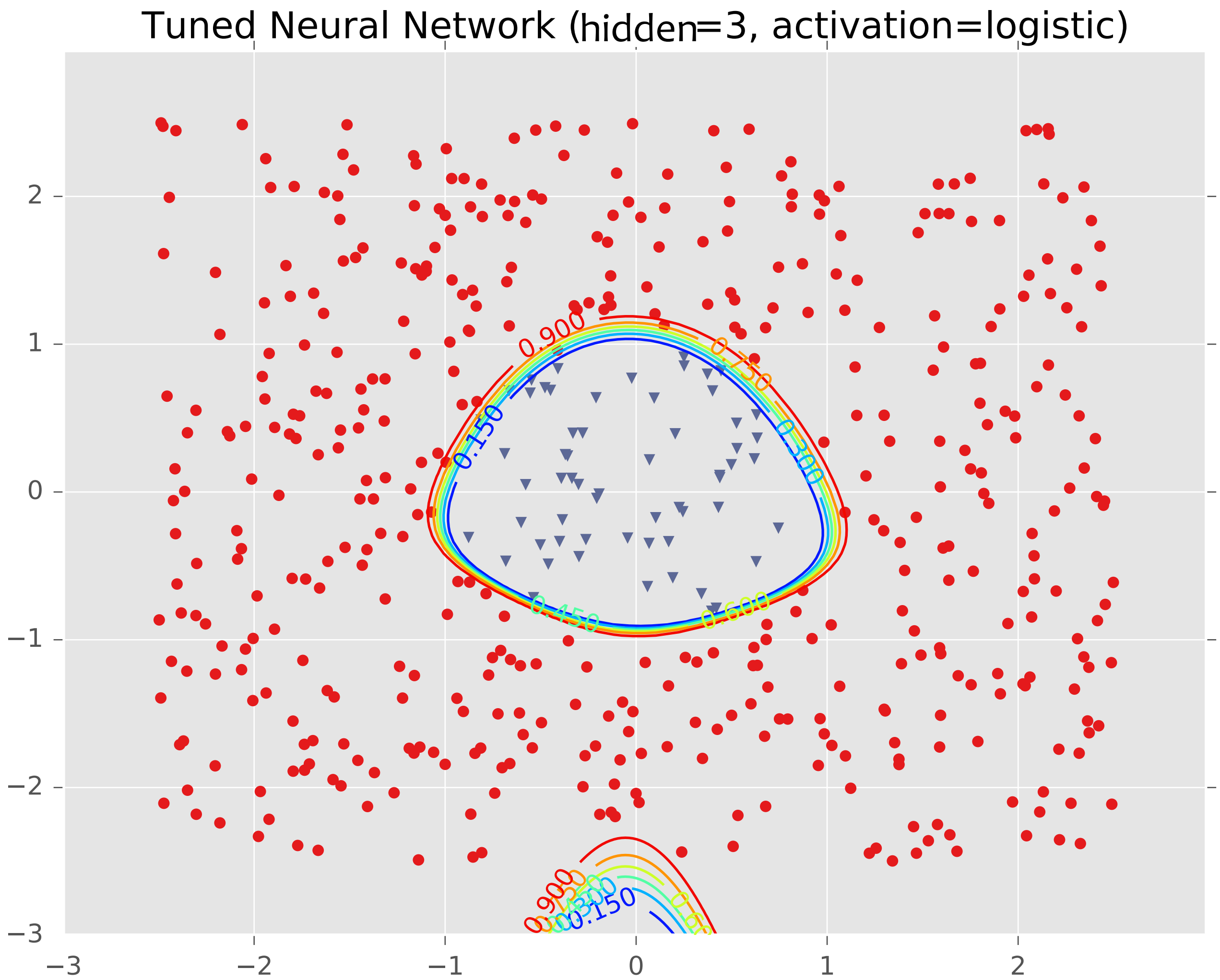


z_1

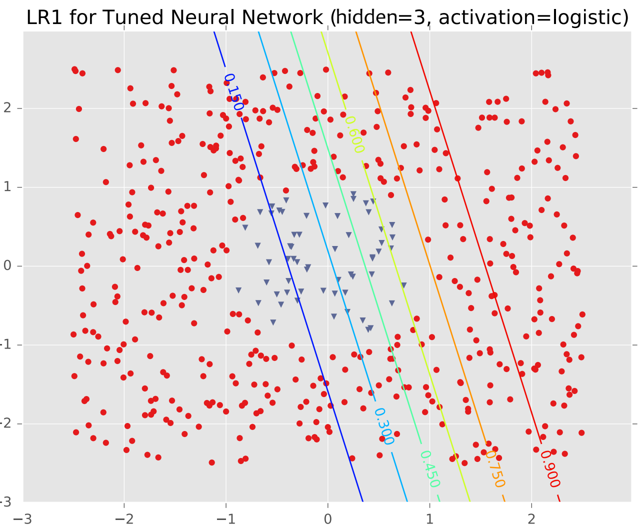
Example #2: One Pocket

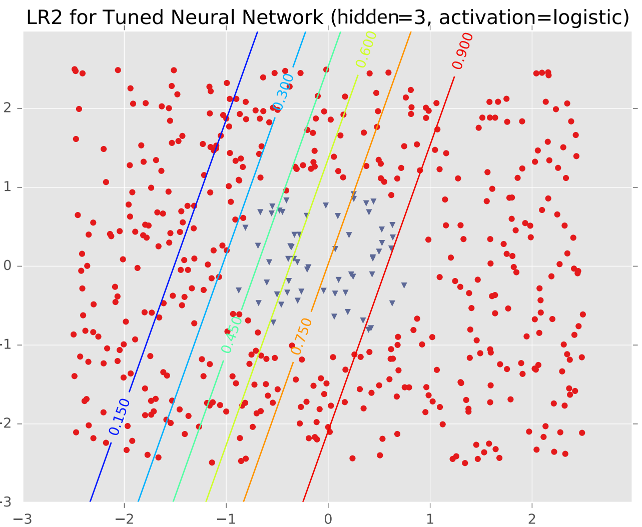


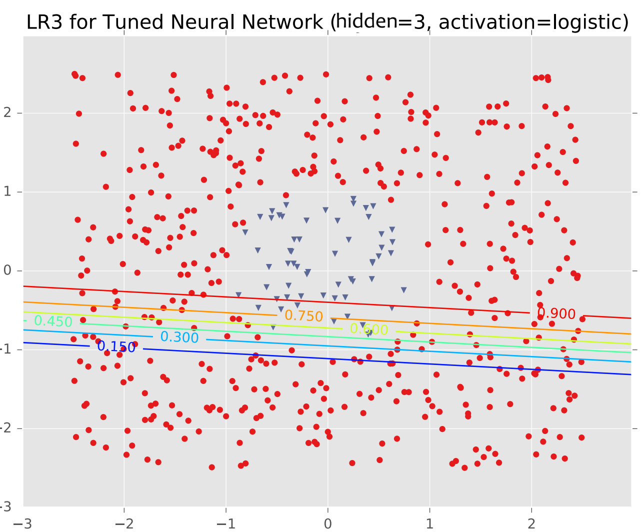
Example #2: One Pocket

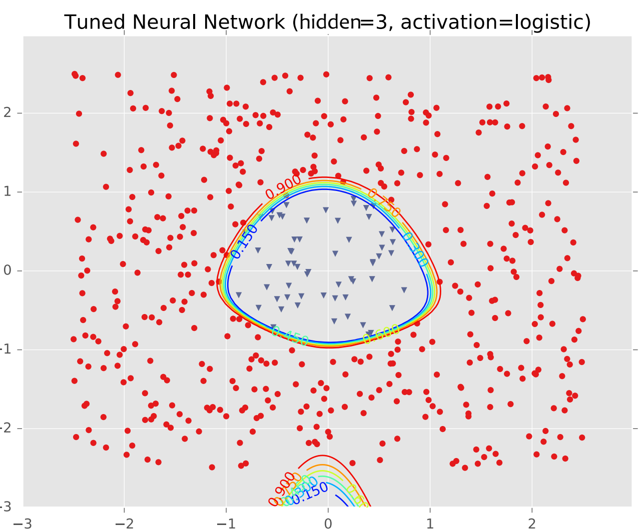


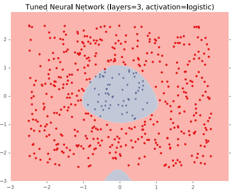
Example #2: One Pocket









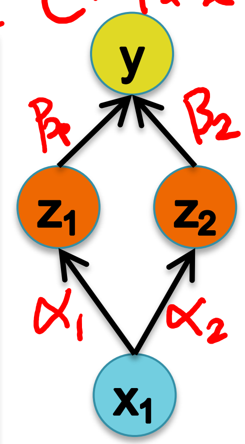


Neural Network Parameters

Question: Q2 A=toxic B=True C=False

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.



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

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

Answer:

This gives rise to a nonconvex objective function!

When poll is active, respond at **pollev.com/10301601polls**

Question 2

A

B

C

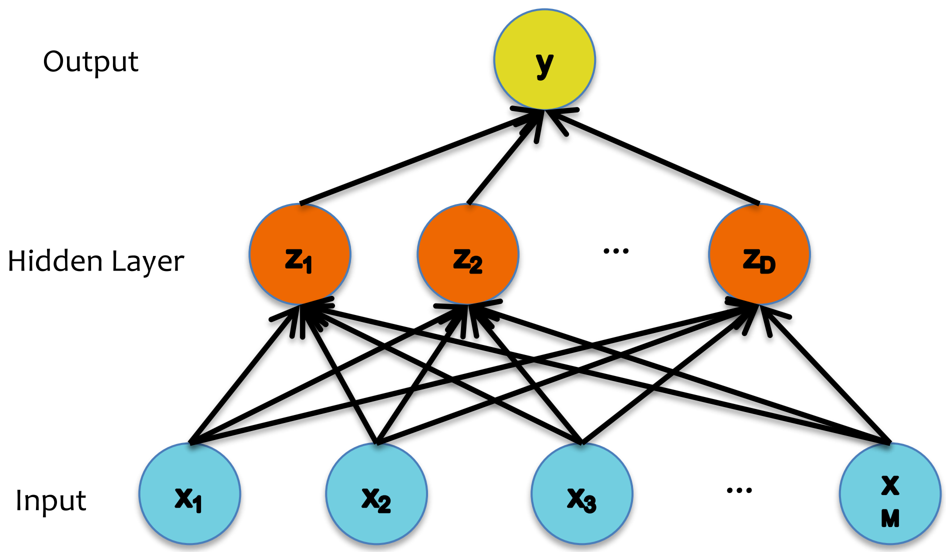
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

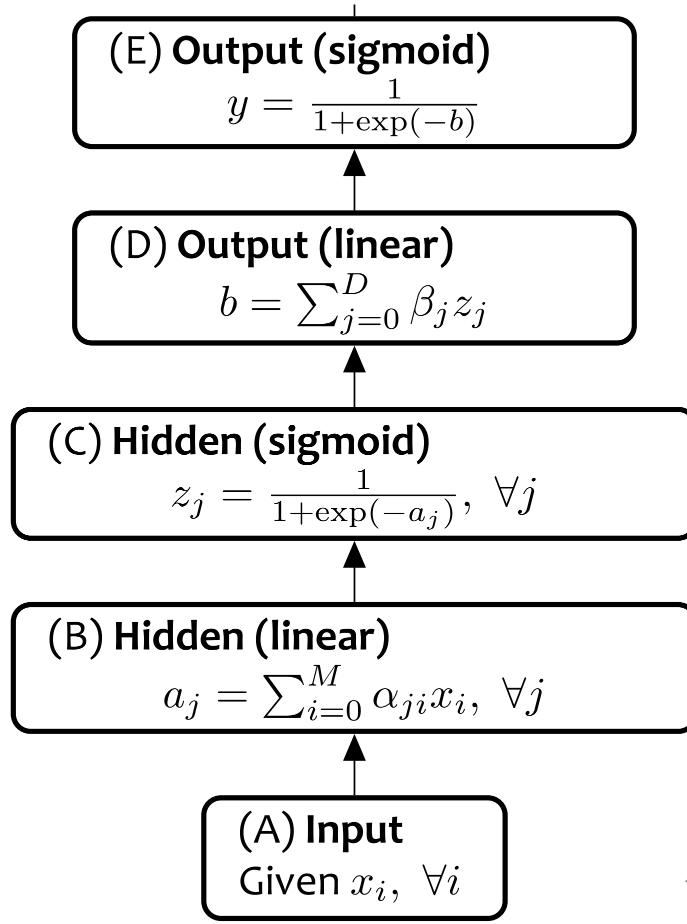
ARCHITECTURES

Decision Functions

Neural Network

Neural Network for Classification





Neural Networks

Chalkboard

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

Neural Network Architectures

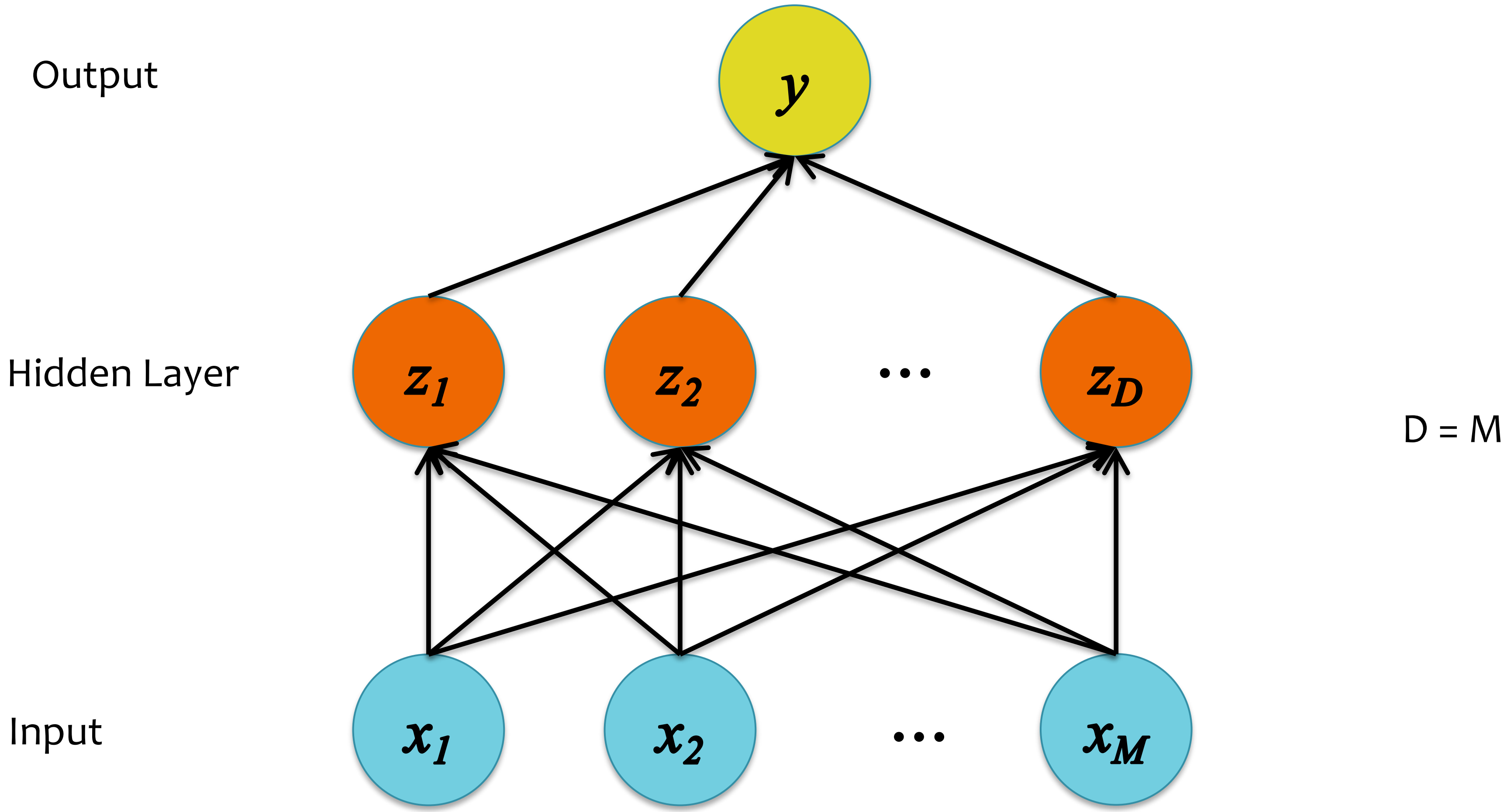
Even for a basic Neural Network, there are many design decisions to make:

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 DEEPER NETWORKS

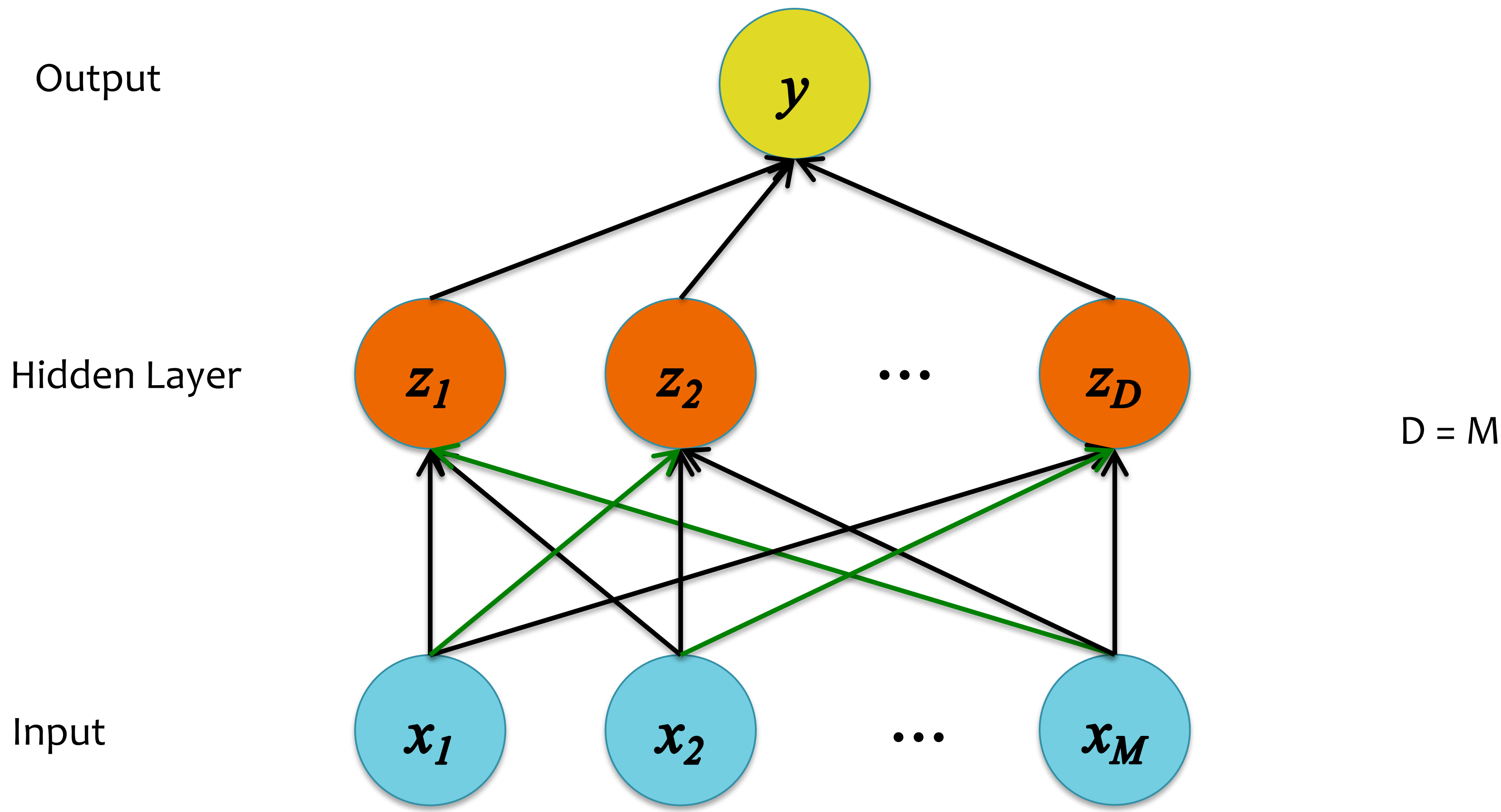
Building a Neural Net

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



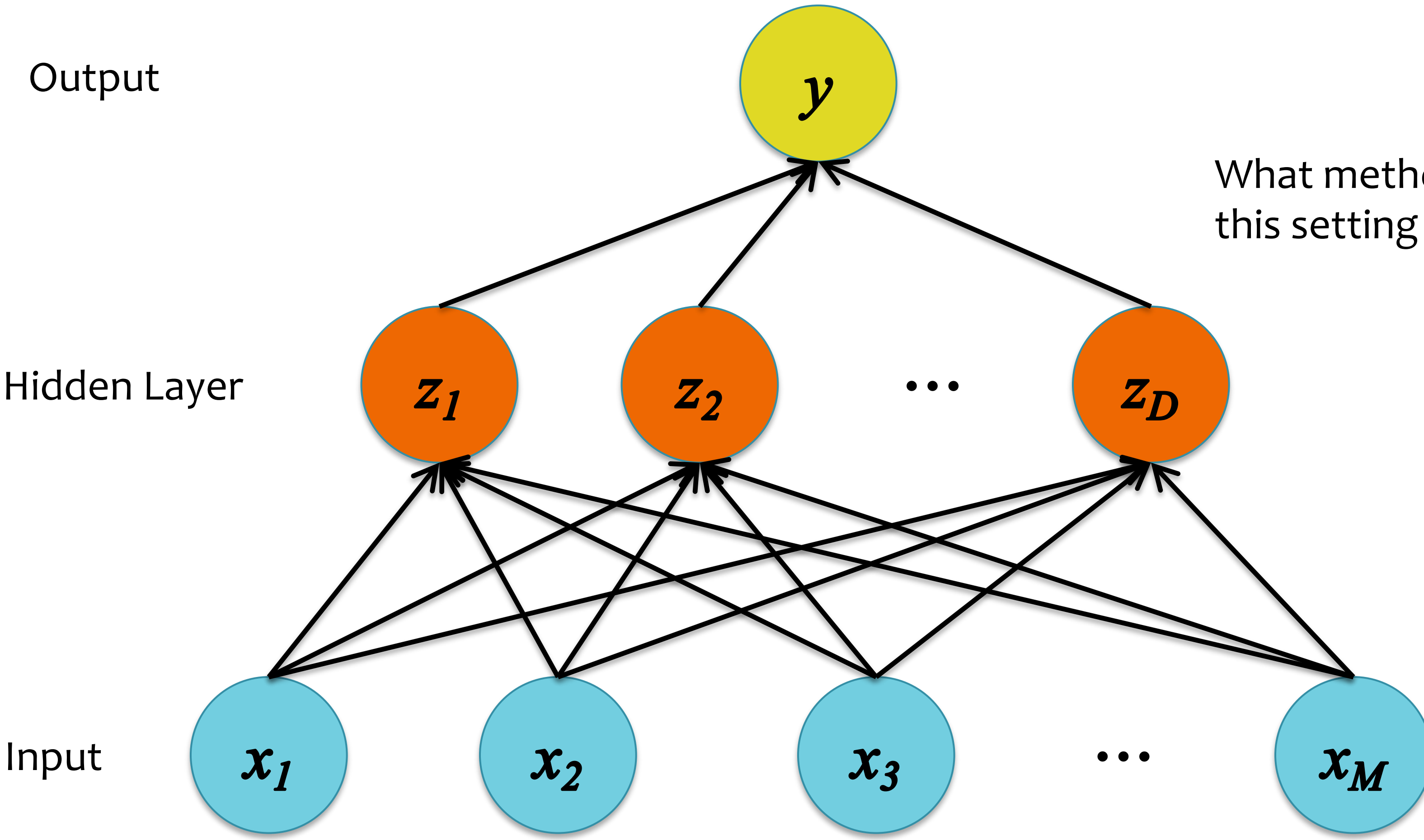
Building a Neural Net

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



Building a Neural Net

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

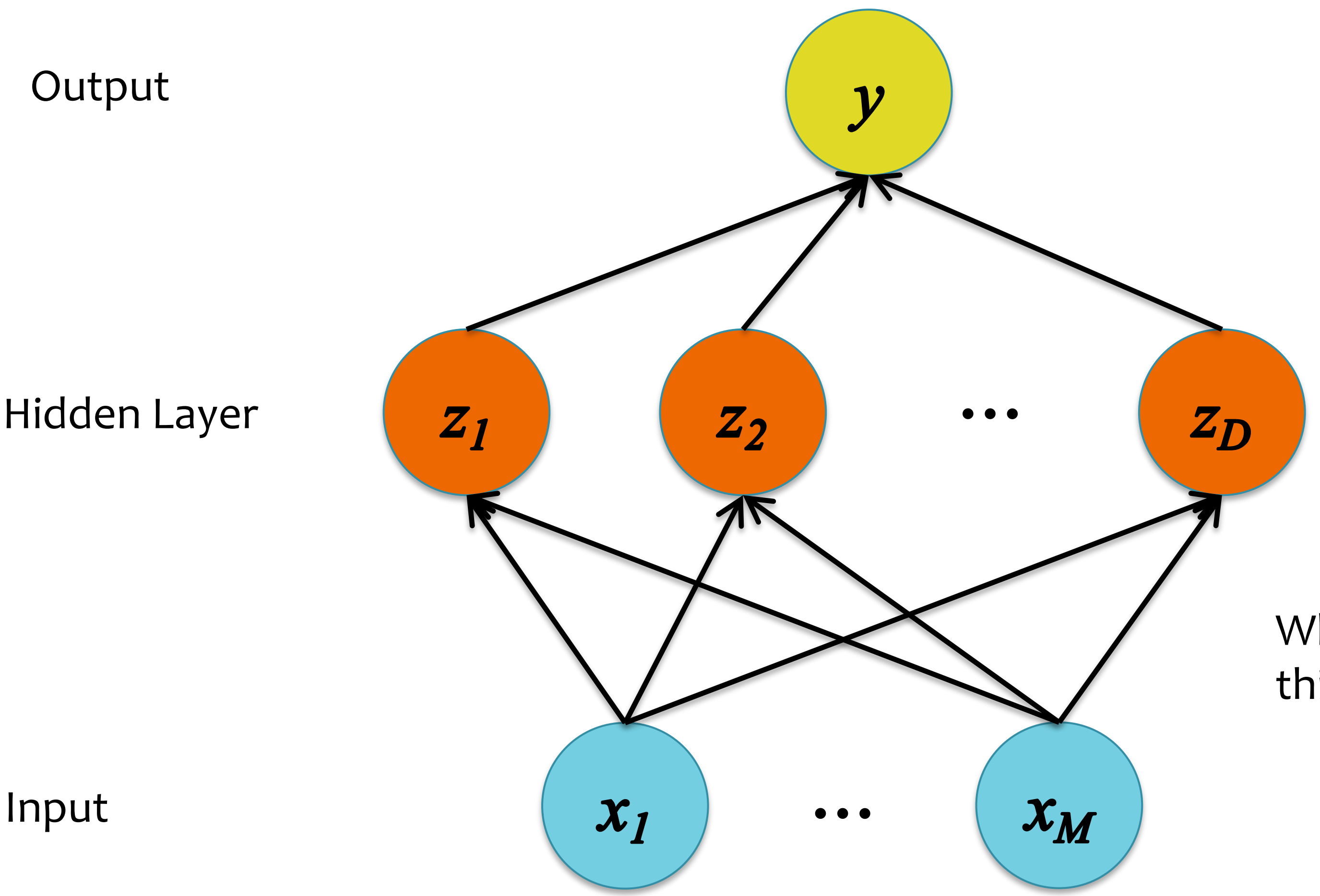


What method(s) is this setting similar to?

D < M

Building a Neural Net

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

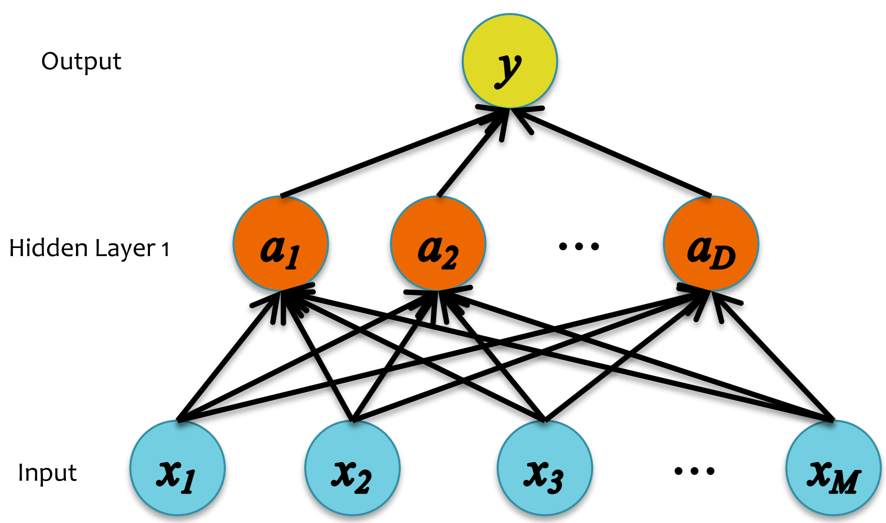


D > M

What method(s) is this setting similar to?

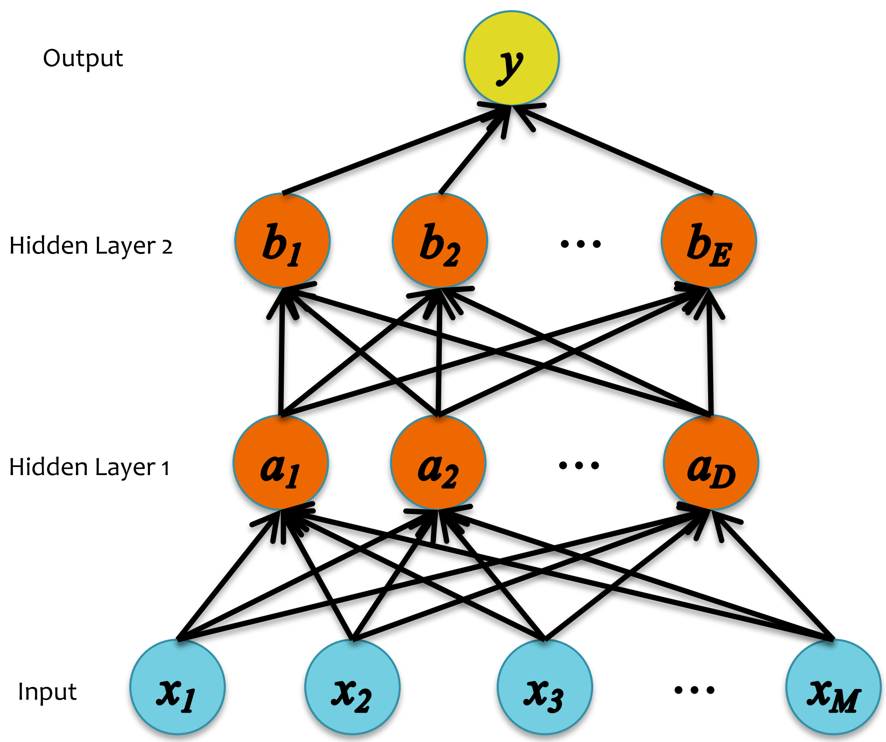
Deeper Networks

Q: How many layers should we use?



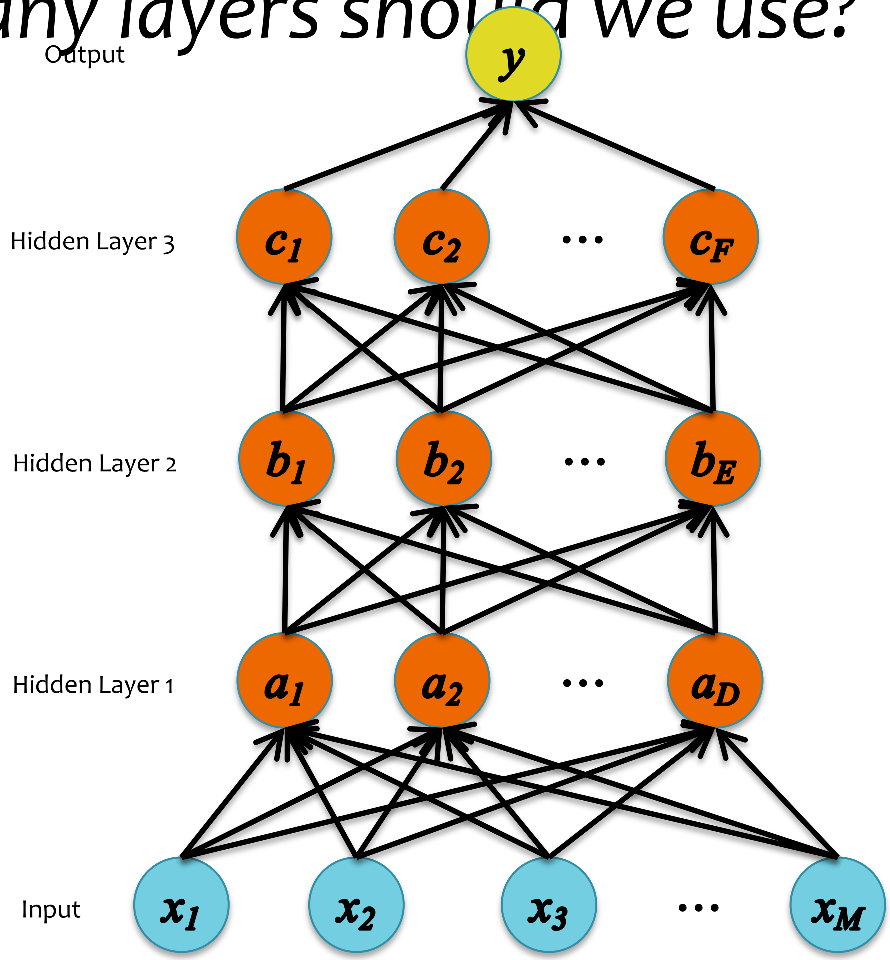
Deeper Networks

Q: How many layers should we use?



Deeper Networks

Q: How many layers should we use?



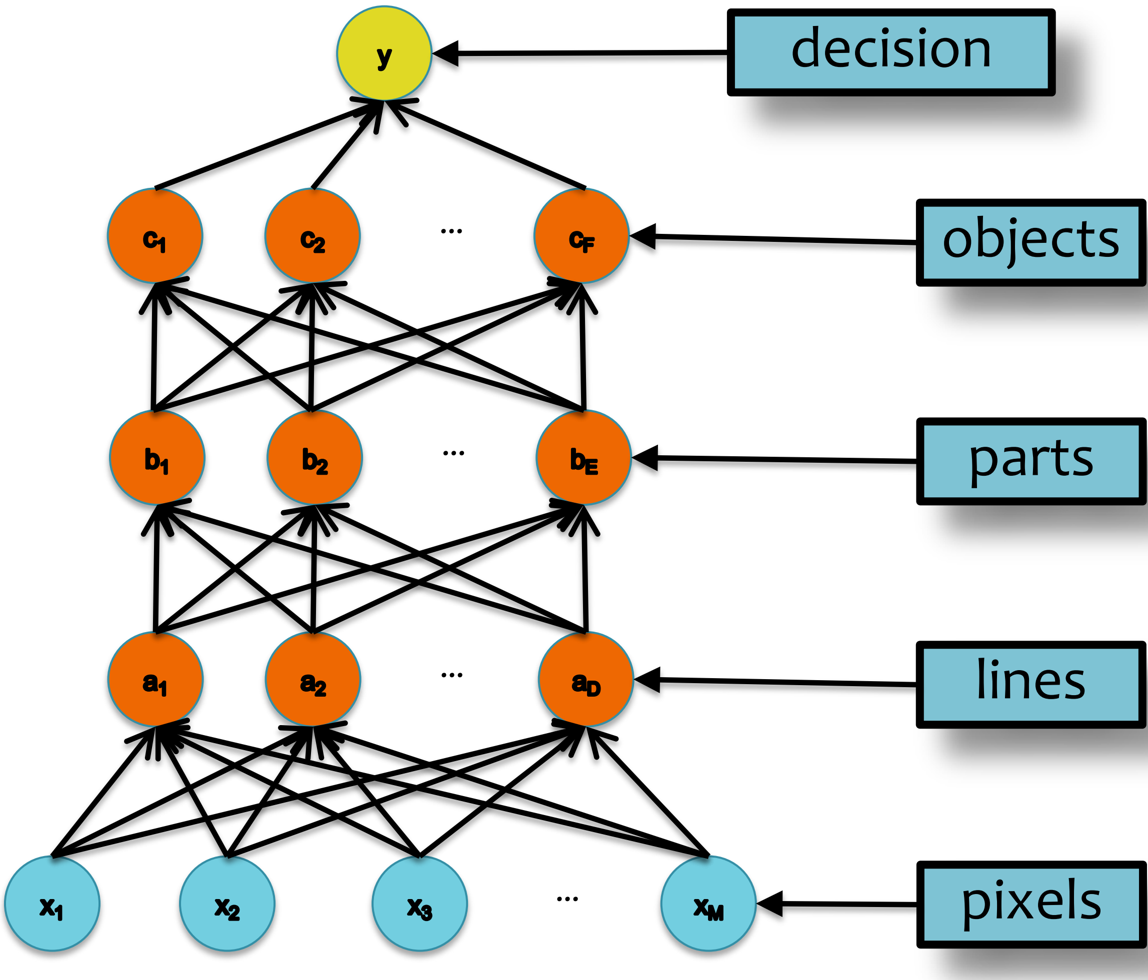
Deeper Networks

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(**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 **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.

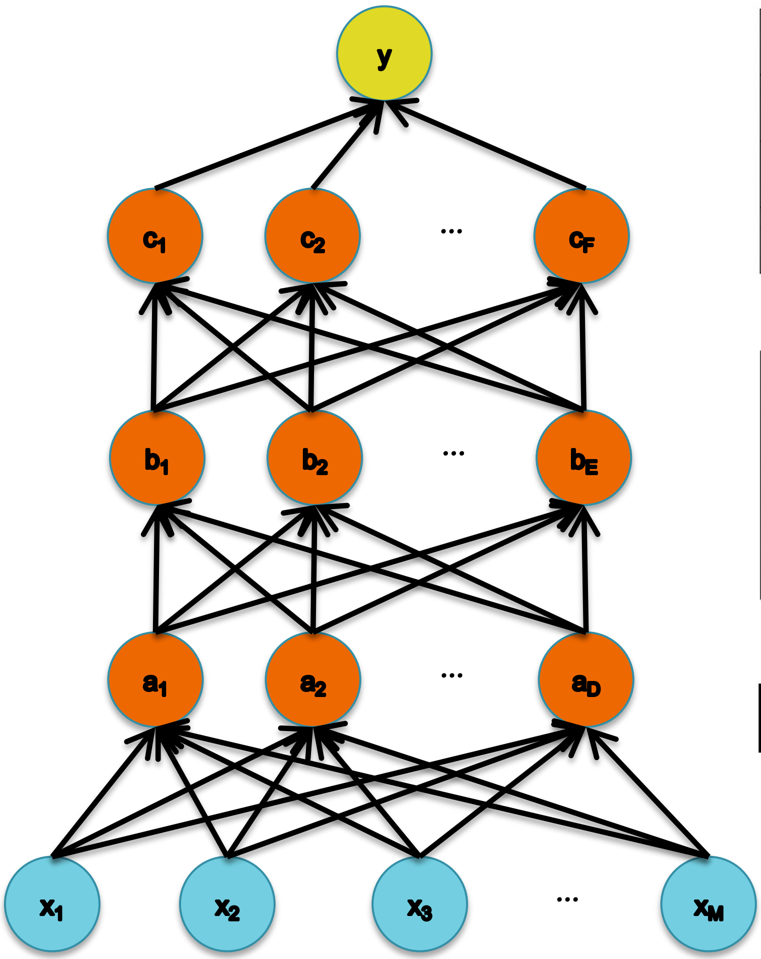
Feature Learning



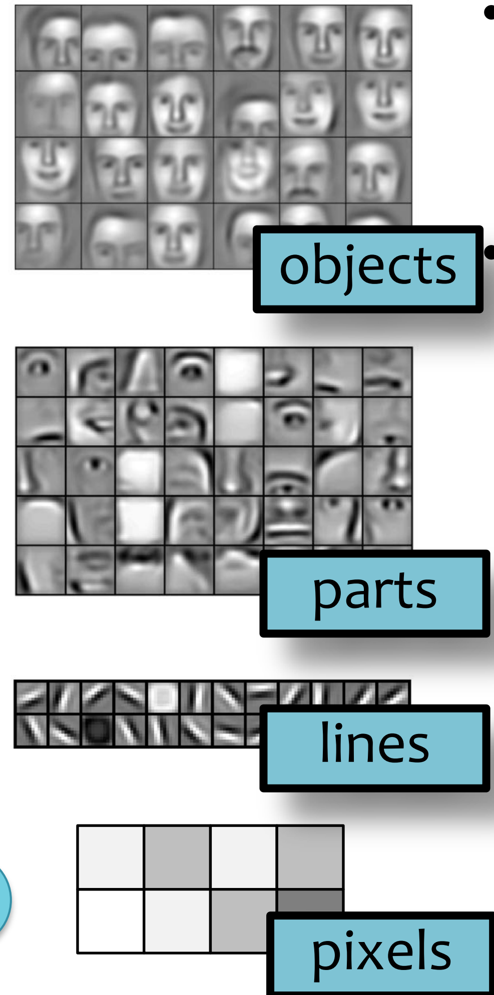
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning



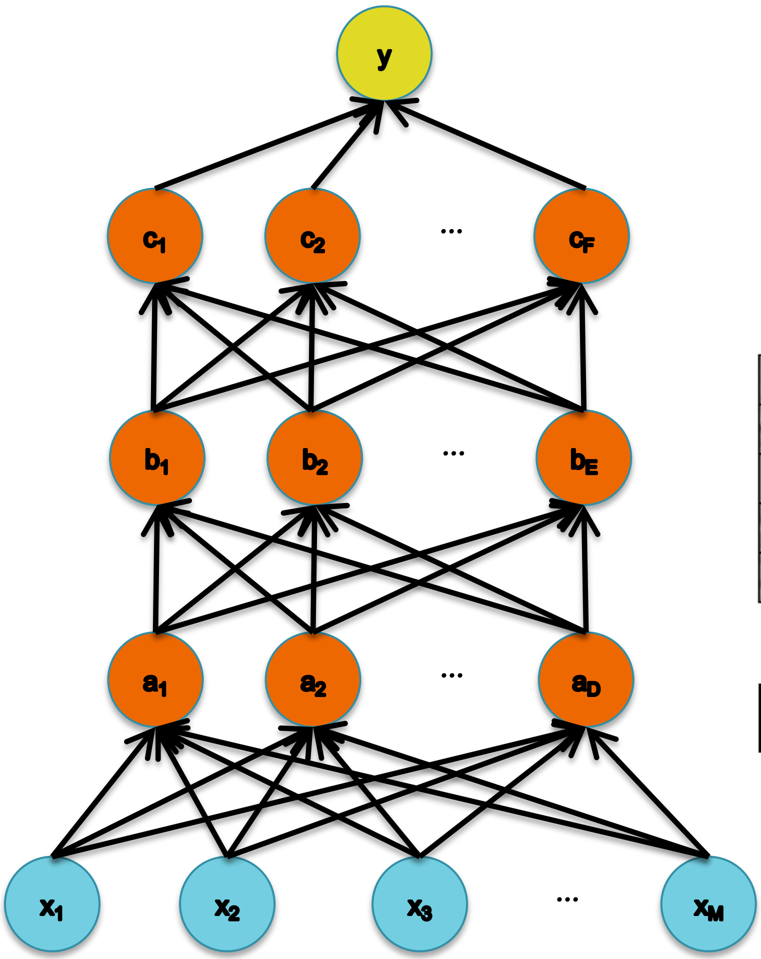
CBDN on Faces



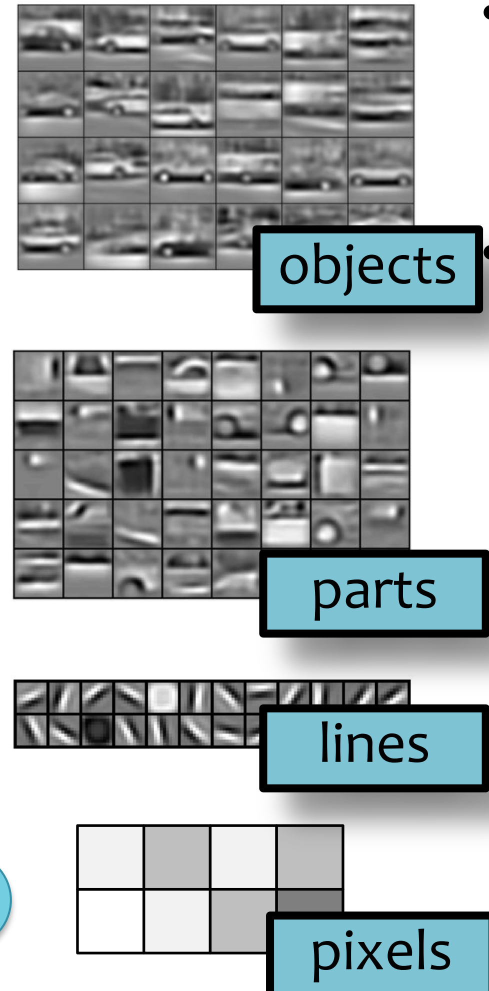
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning



CBDN on Cars



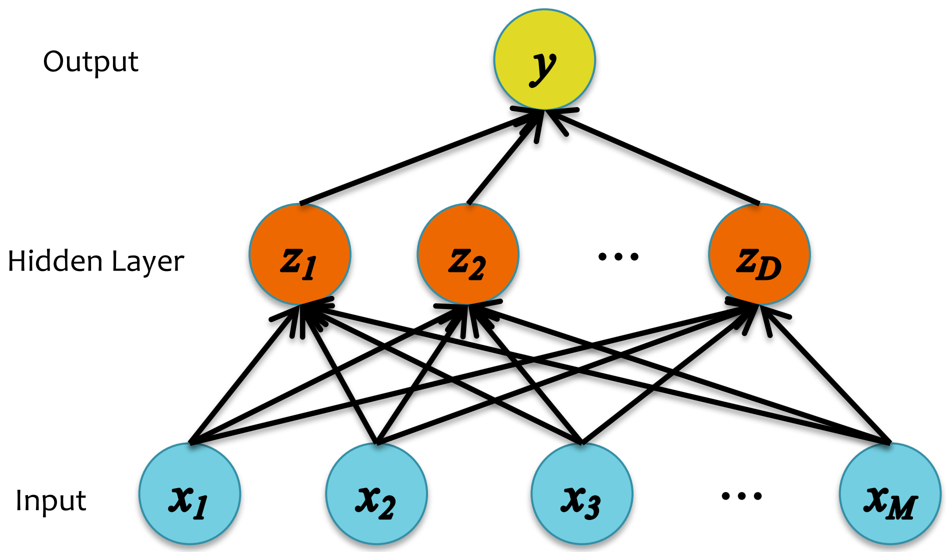
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

ACTIVATION FUNCTIONS

Activation Functions

Neural Network with sigmoid activation functions



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

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

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

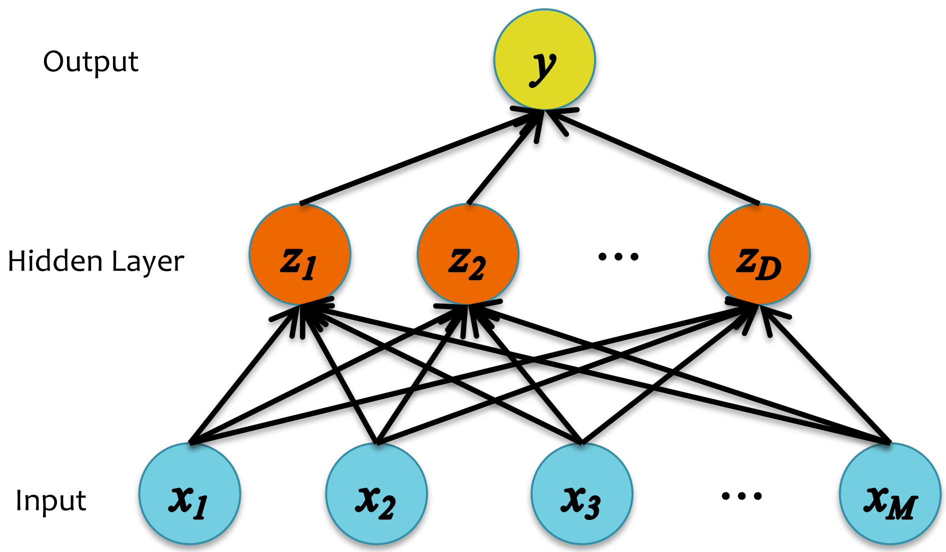
(C) **Hidden (sigmoid)**
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

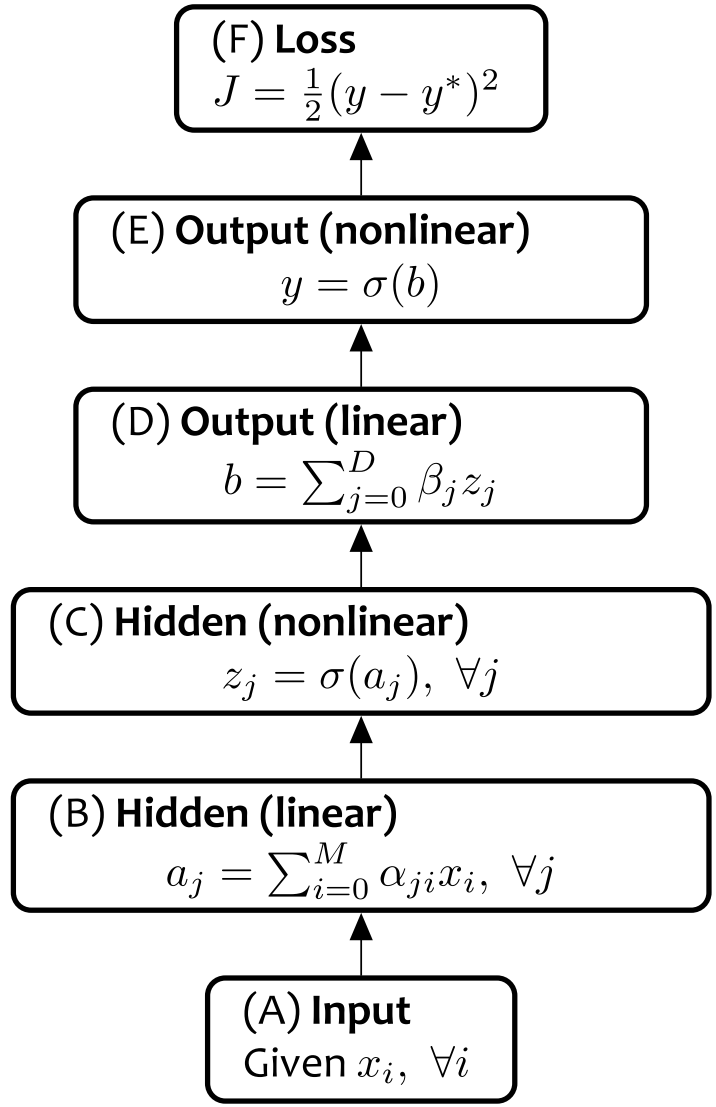
(B) **Hidden (linear)**
$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \ \forall j$

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

Activation Functions

Neural Network with arbitrary nonlinear activation functions



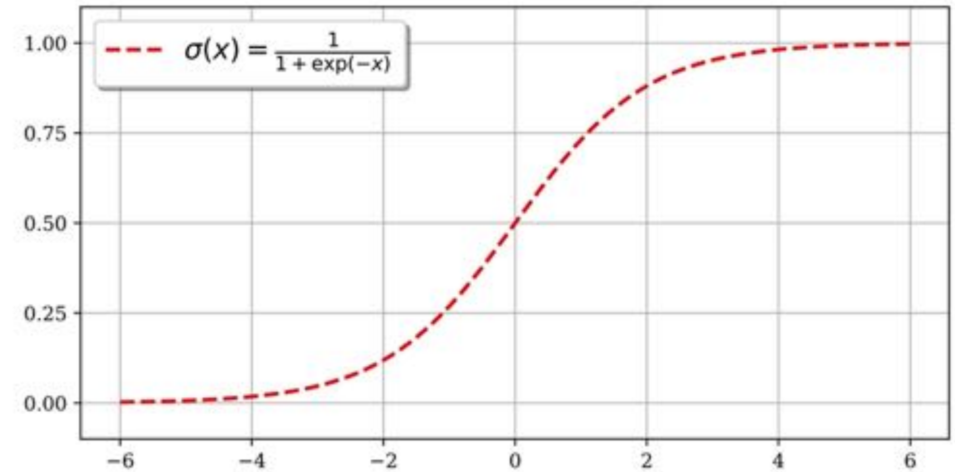


Activation Functions

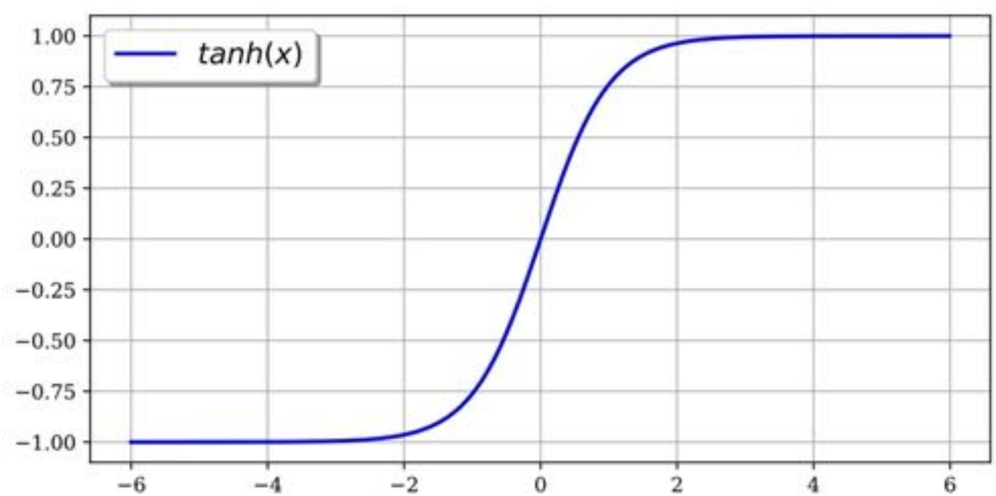
So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…

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

Sigmoid (aka. logistic) function



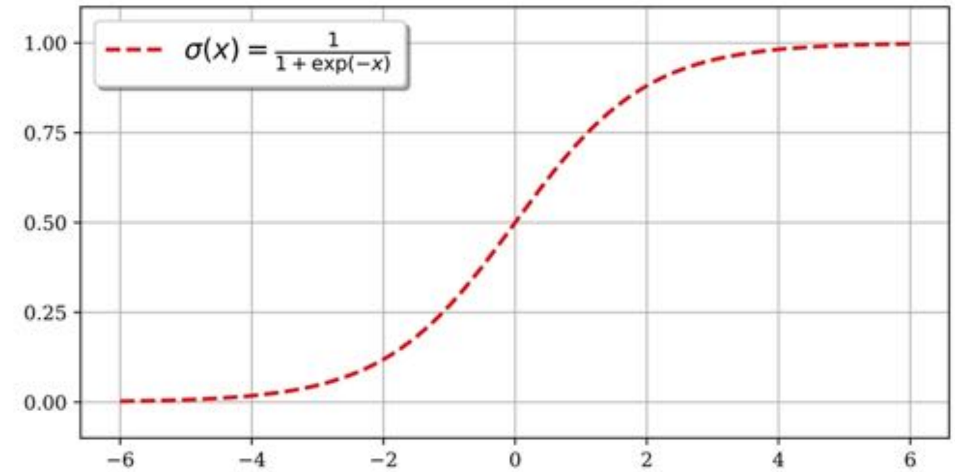
Hyperbolic tangent function



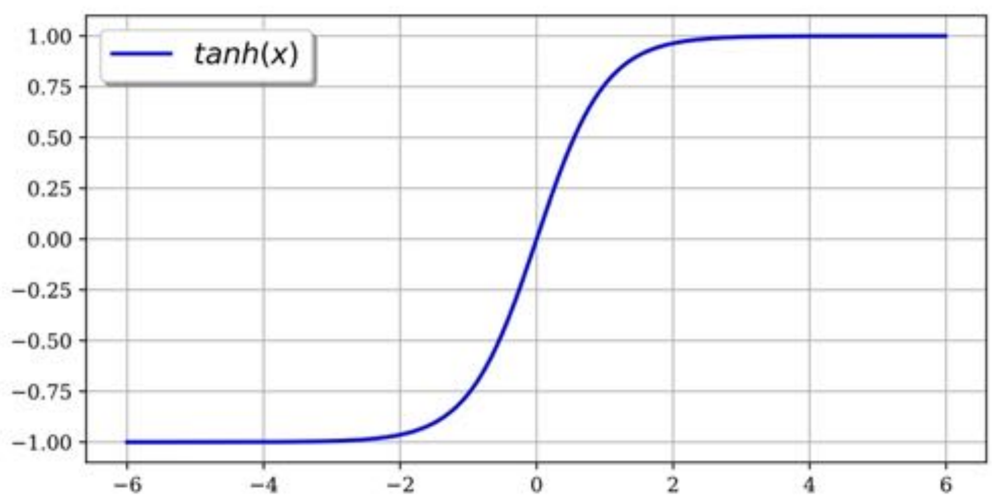
Activation Functions

- sigmoid, σ(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

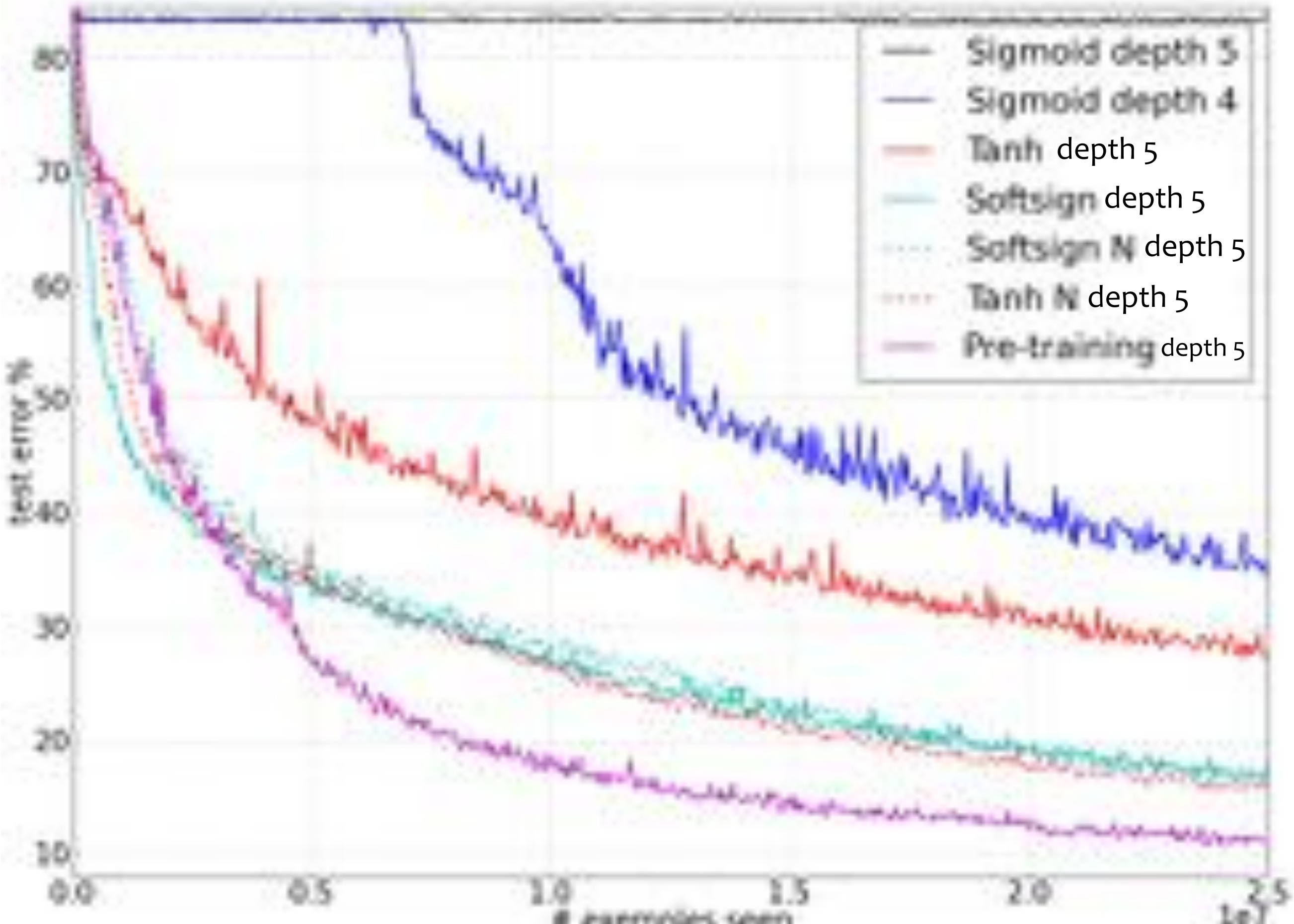


Hyperbolic tangent function



Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010



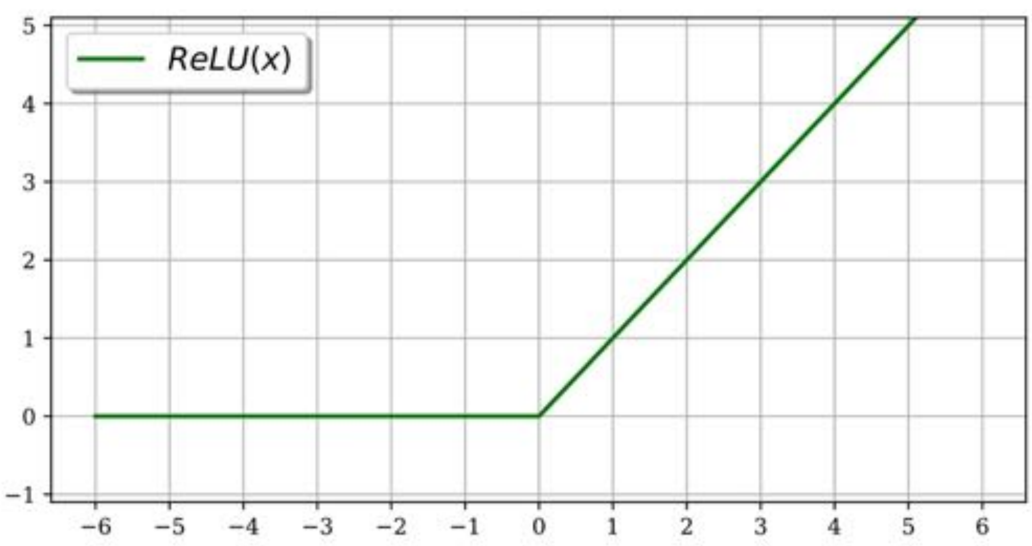
sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

Activation Functions

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

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



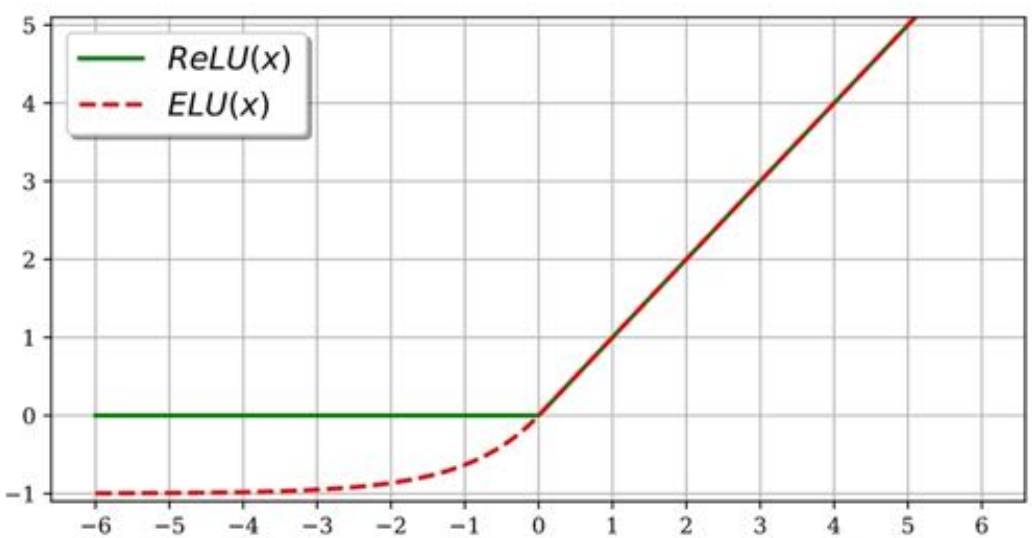
Activation Functions

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

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

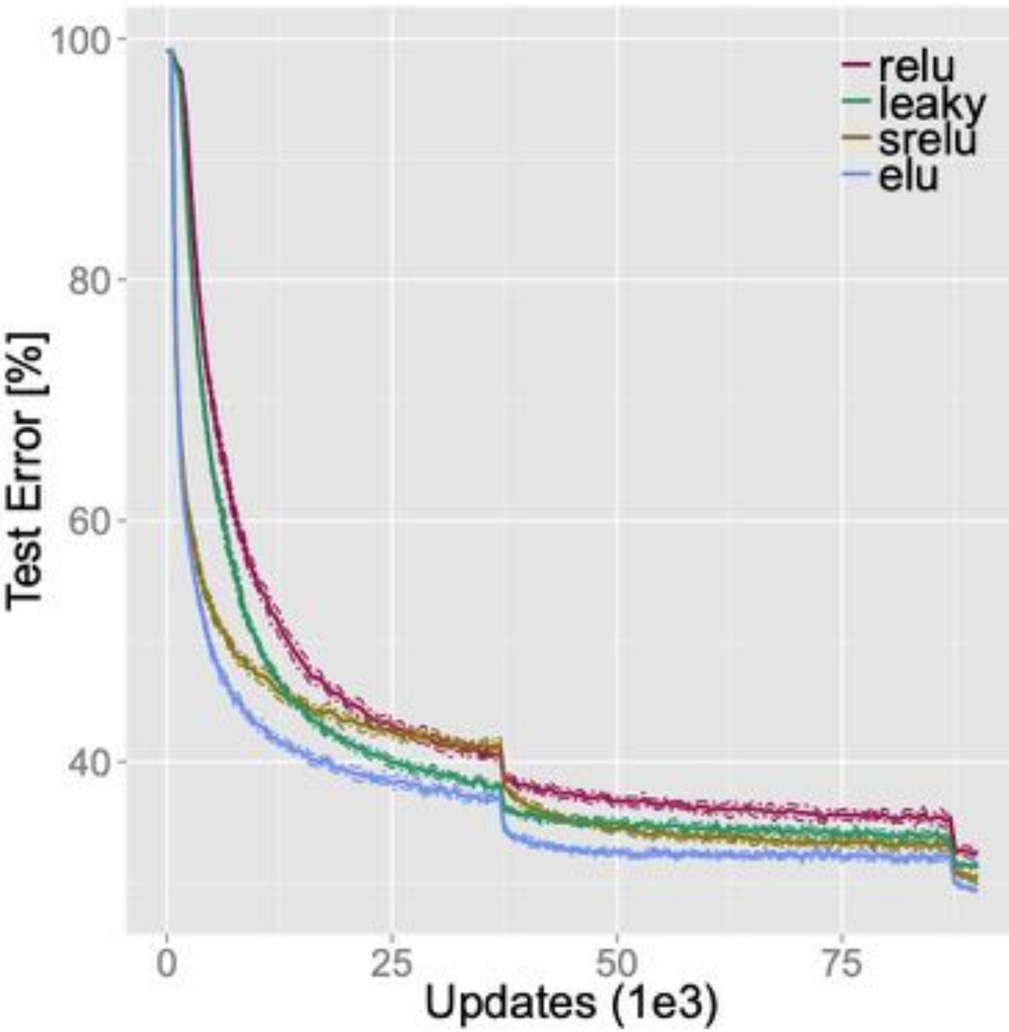
- Exponential Linear Unit (ELU)
  - same as ReLU on positive inputs
  - unlike ReLU, allows negative outputs and smoothly transitions for x < 0

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



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

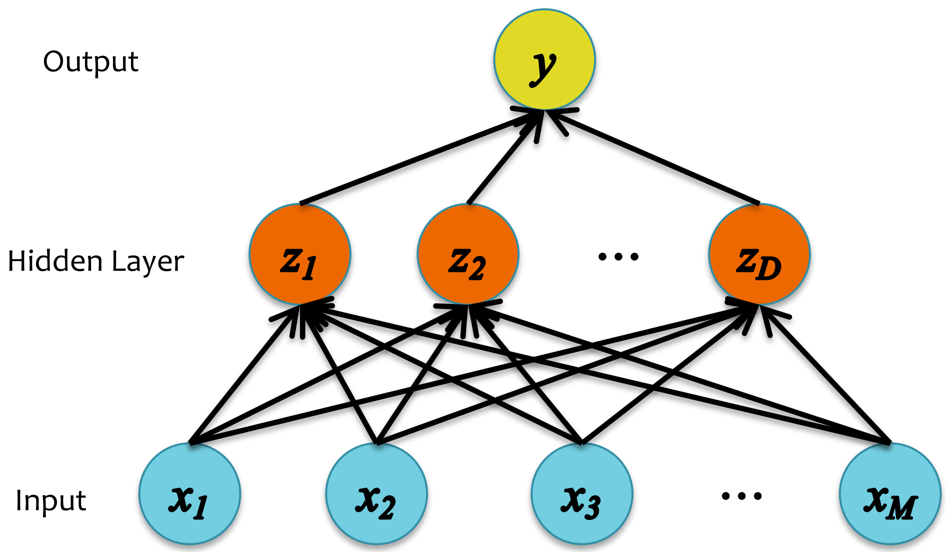
Figure from Clevert et al. (2016)

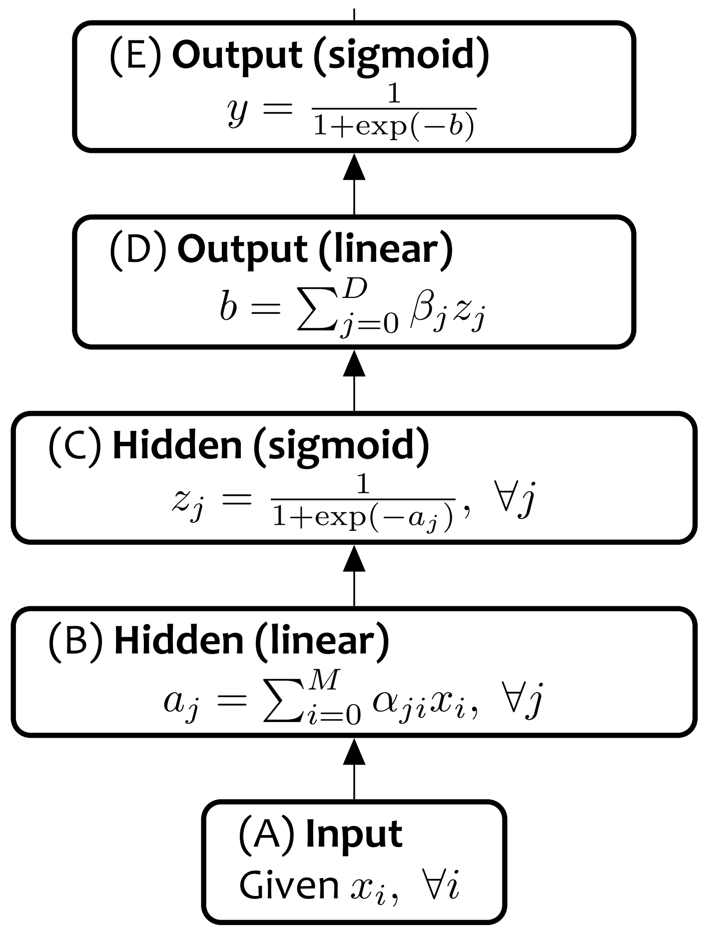
LOSS FUNCTIONS & OUTPUT LAYERS

Decision Functions

Neural Network

Neural Network for Classification

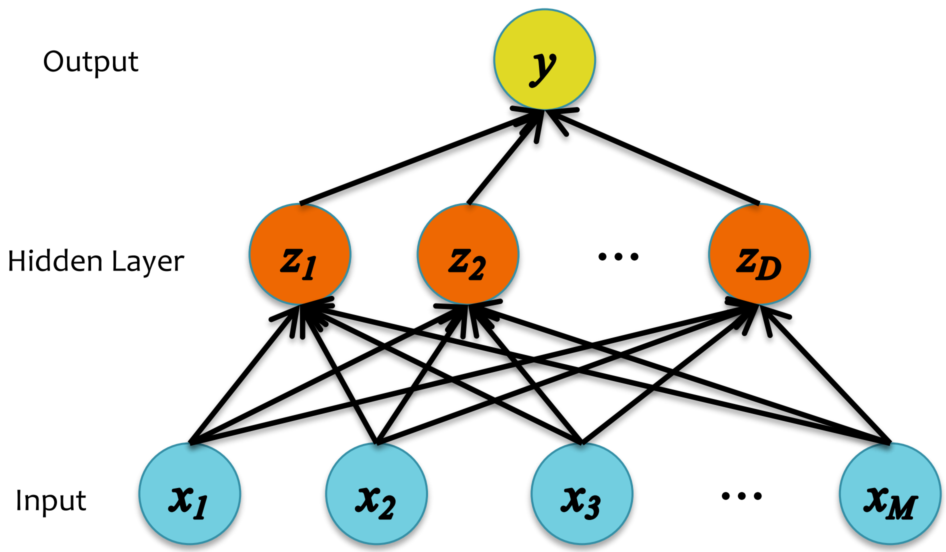




Decision Functions

Neural Network

Neural Network for Regression



(D) **Output (linear)**

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

(C) **Hidden (sigmoid)**

$$z_j = \frac{1}{1+\exp(-a_j)}, \quad \forall j$$

(B) **Hidden (linear)**

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i, \quad \forall j$$

(A) **Input**

Given $x_i, \quad \forall i$

Objective Functions for NNs

1. Quadratic Loss:
   - the same objective as Linear Regression
   - i.e. mean squared erroradd an additional “softmax” layer at the end of our network

Quadratic $J = \frac{1}{2}(y - y^*)^2$	$\frac{dJ}{dy} = y - y^*$

2. Cross-Entropy:
   - the same objective as Logistic Regression
   - i.e. negative log likelihood
   - This requires probabilities, so we add an additional “softmax” layer at the end of our network

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

Objective Functions for NNs

Cross-entropy vs. Quadratic loss

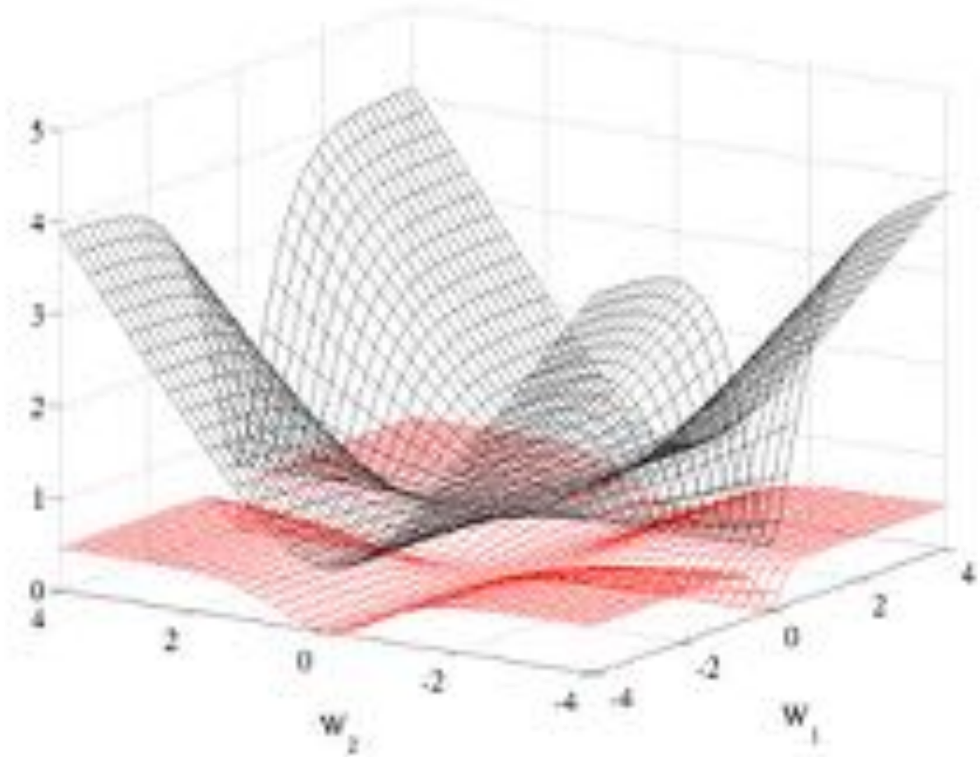
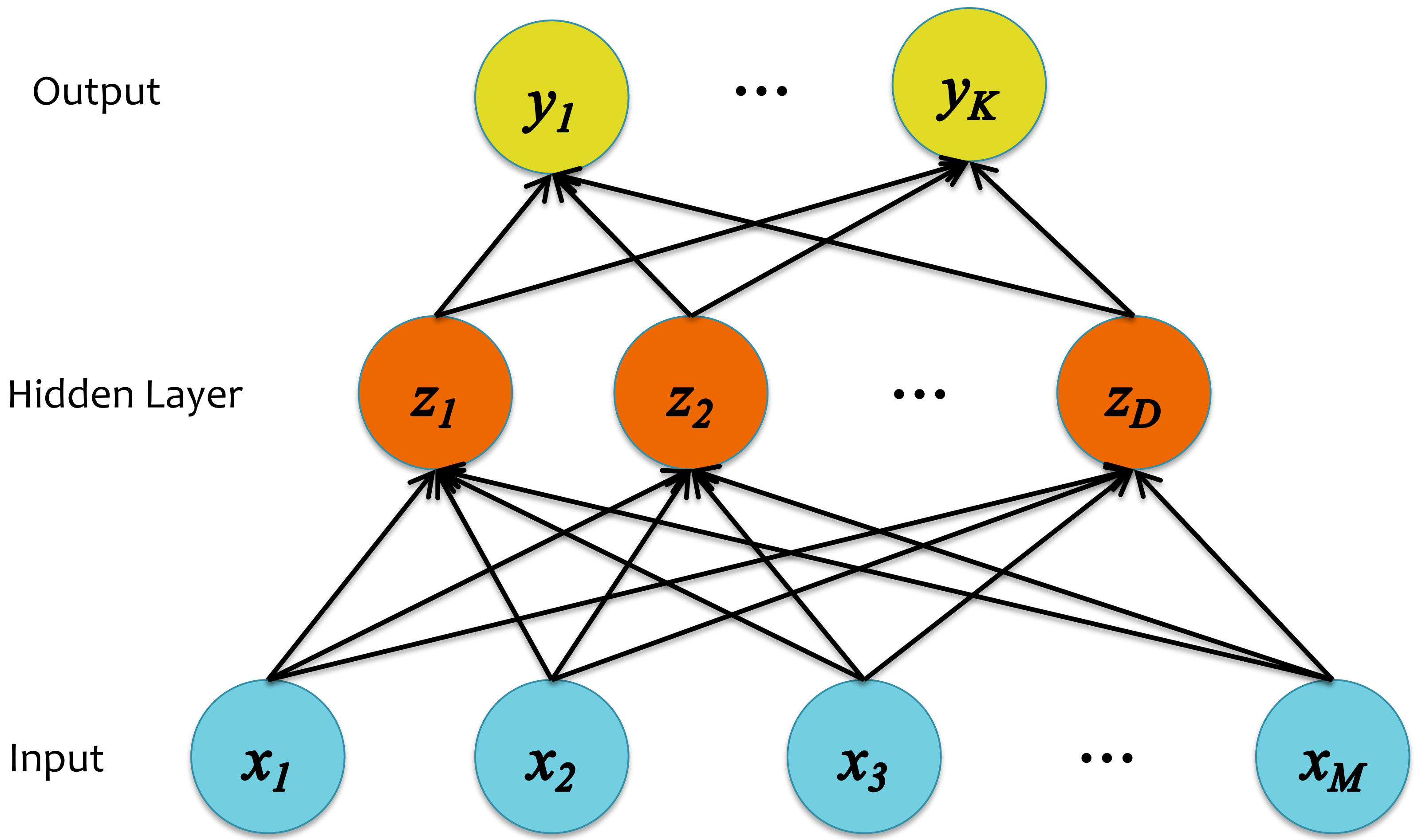


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Figure from Glorot & Bentio (2010)

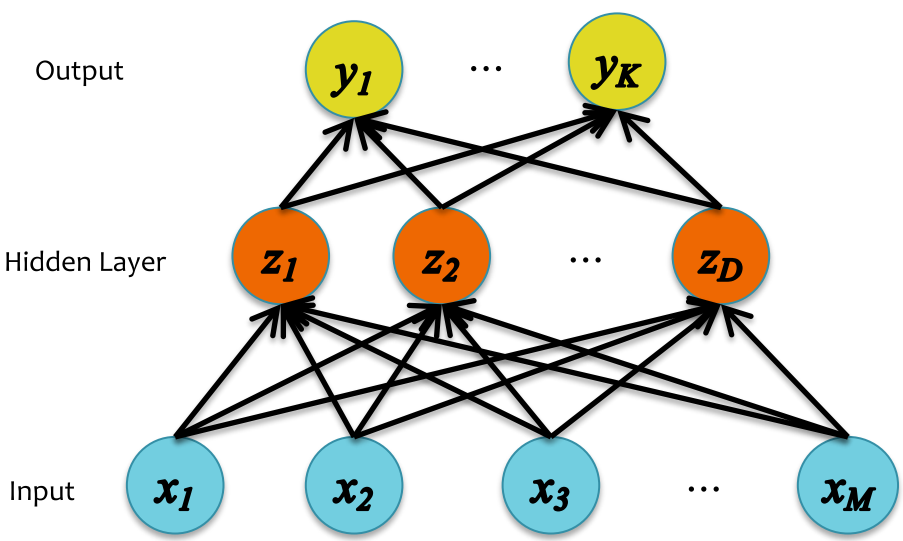
Multi-Class Output



Multi-Class Output

Softmax:

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



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

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

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

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

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

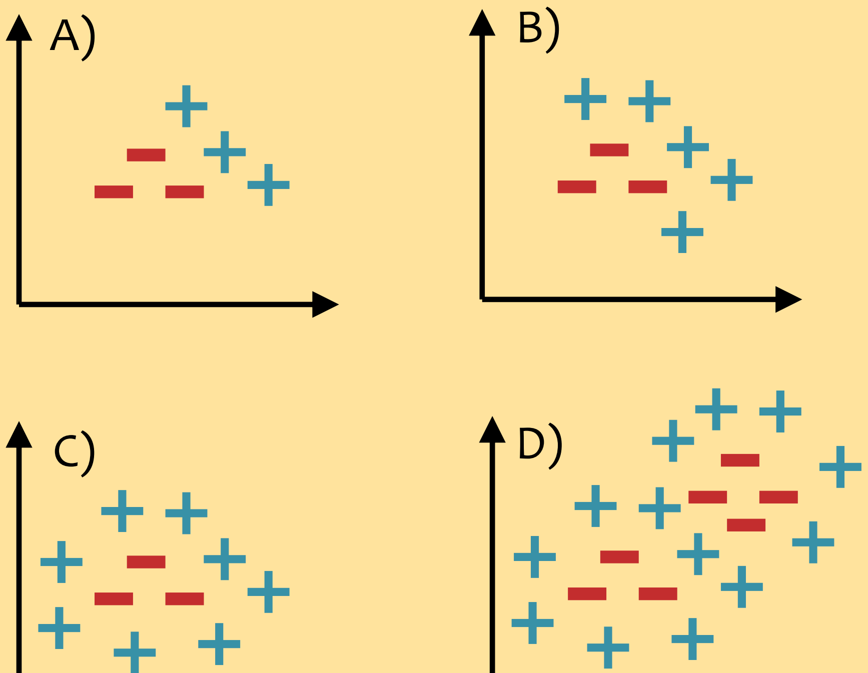
(A) Input
Given $x_i, \;\; \forall i$

Examples 3 and 4

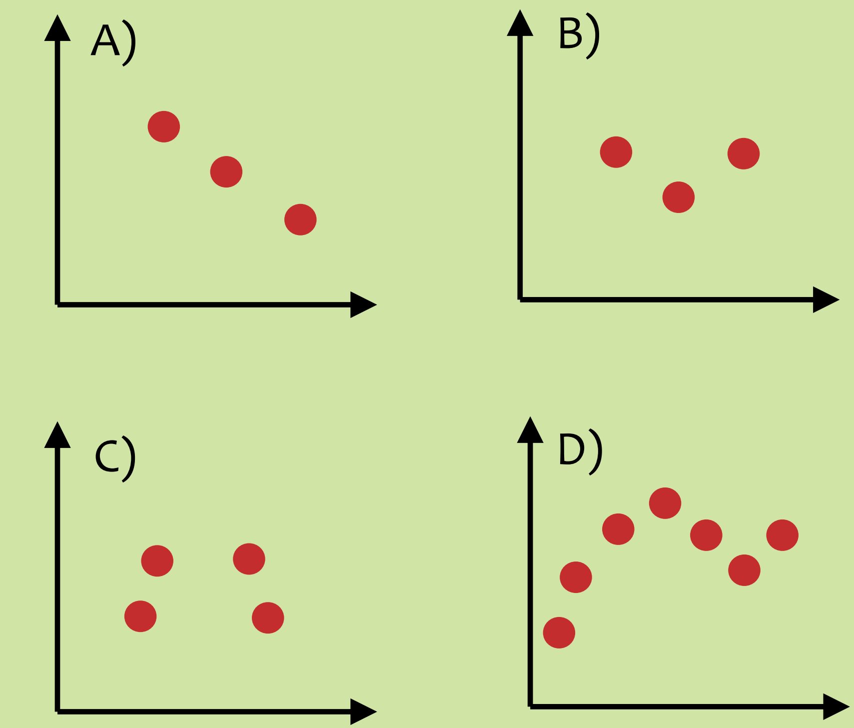
DECISION BOUNDARY EXAMPLES

Neural Network Errors

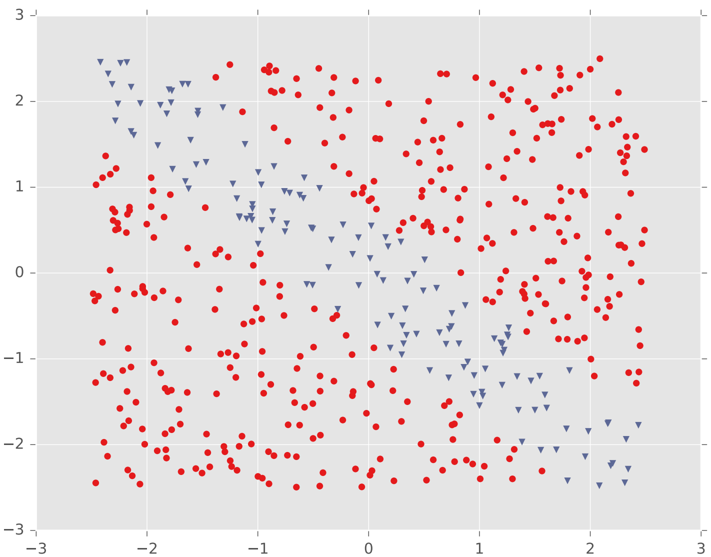
Question A: For which of the datasets below does there exist a one-hidden layer neural network that achieves zero *classification* error? **Select all that apply.**



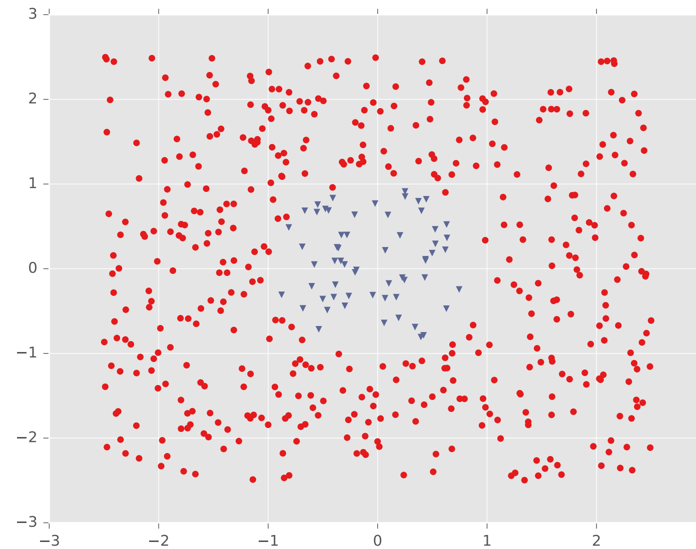
Question B: For which of the datasets below does there exist a one-hidden layer neural network for *regression* that achieves *nearly* zero MSE? **Select all that apply.**



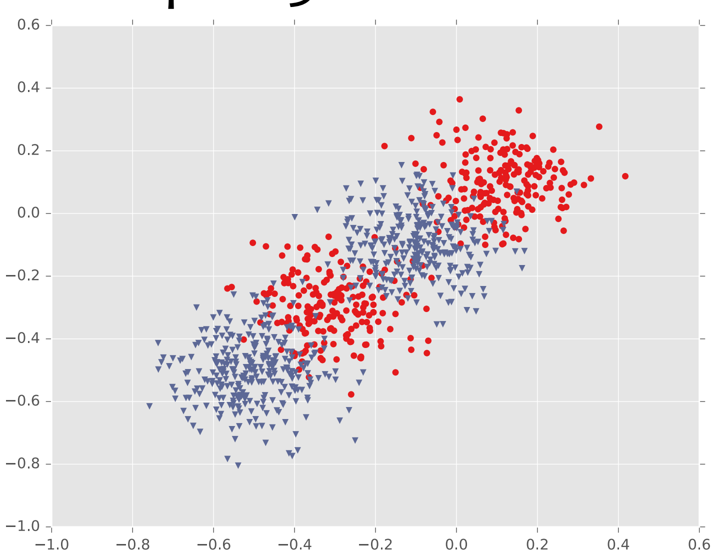
Example #1: Diagonal Band



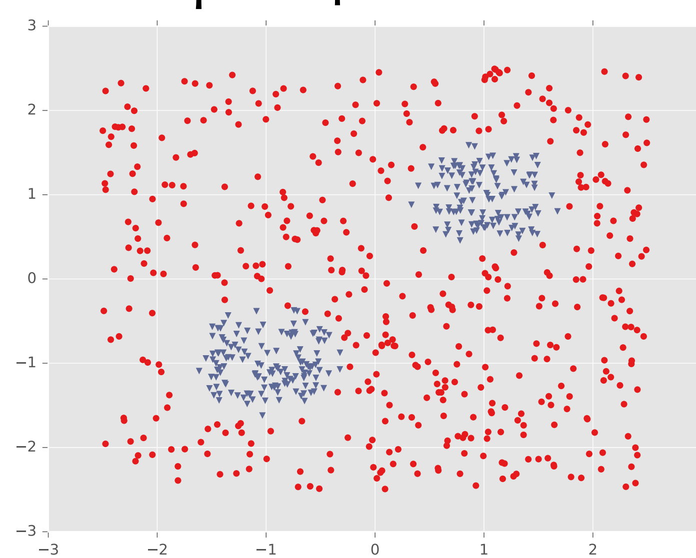
Example #2: One Pocket



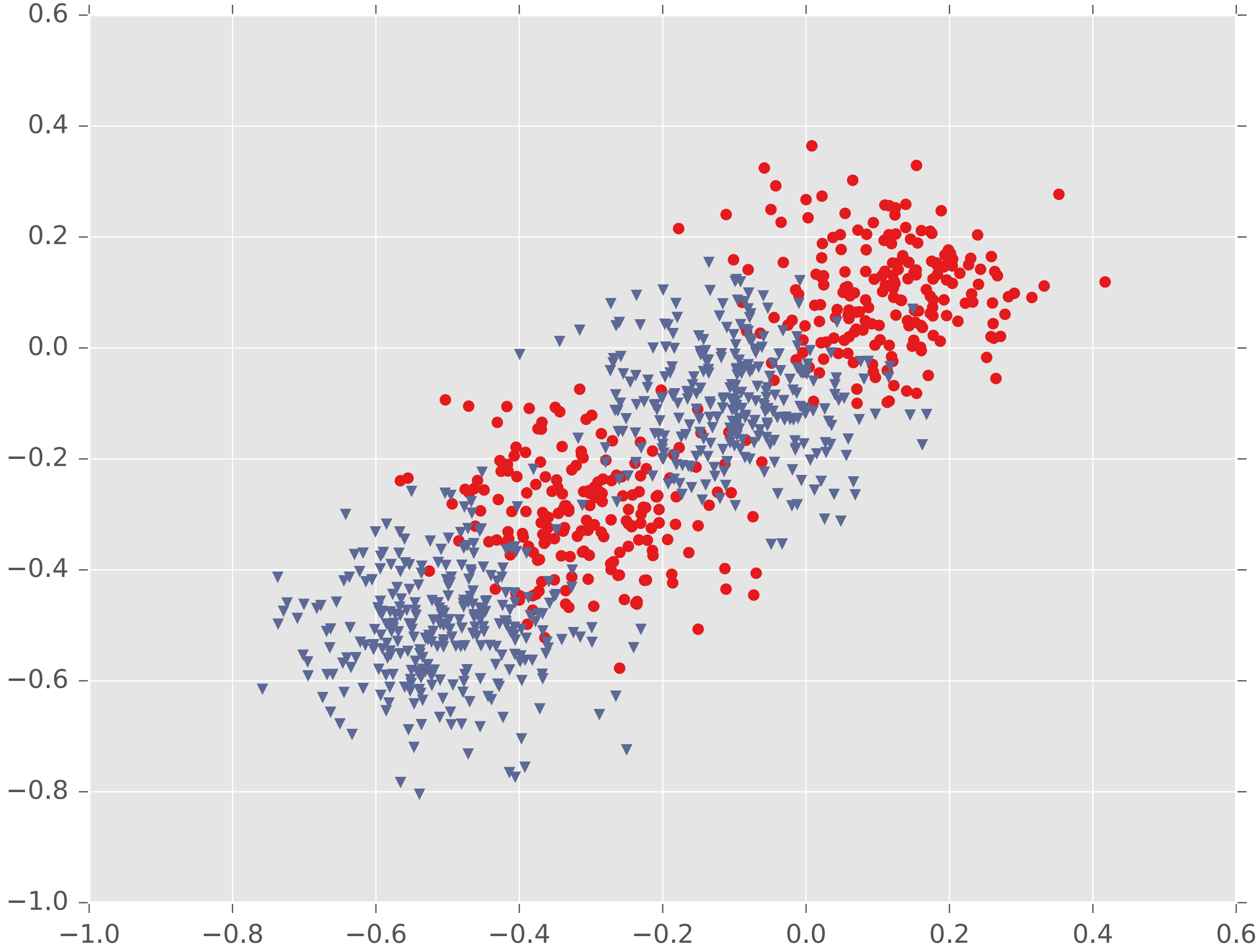
Example #3: Four Gaussians



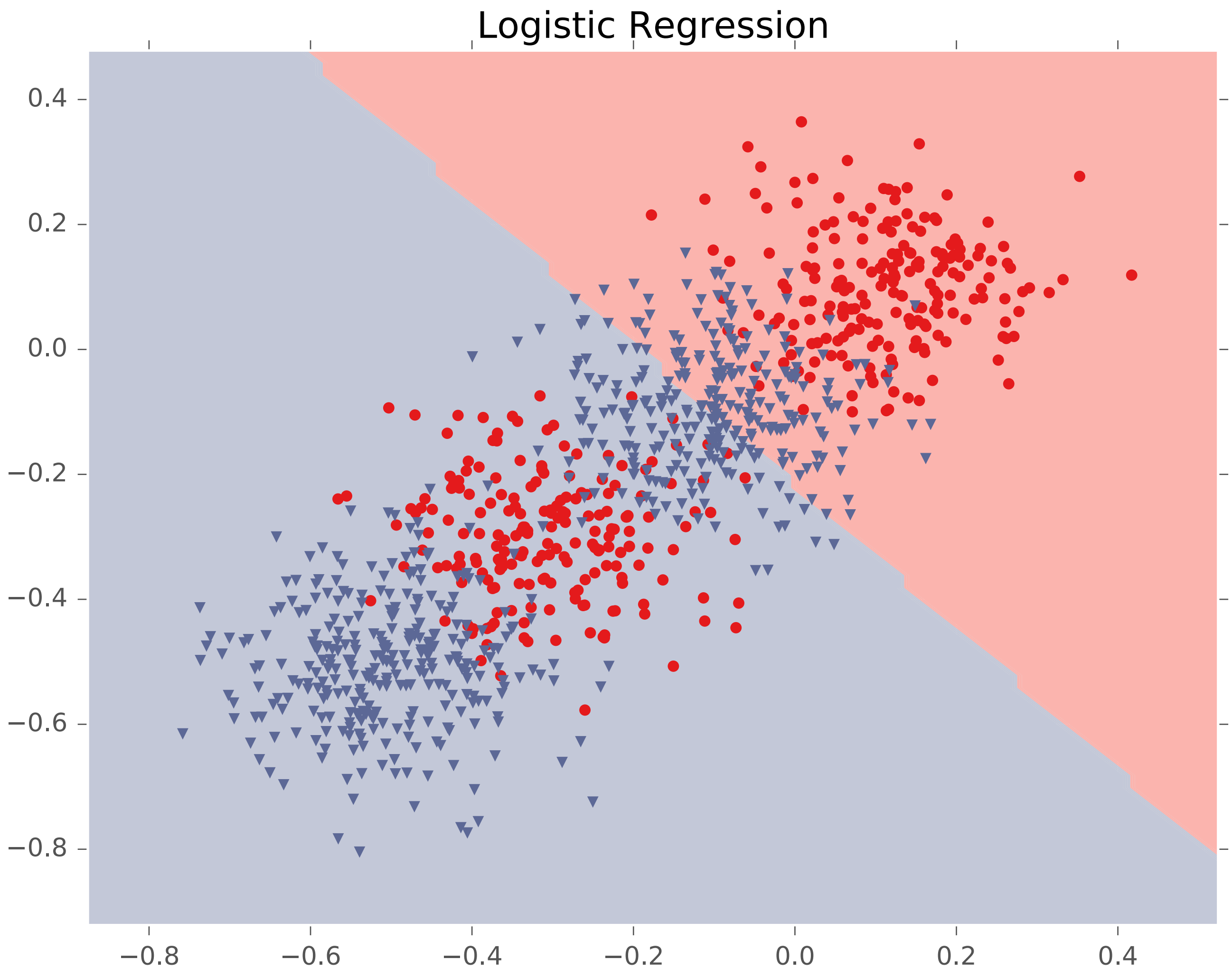
Example #4: Two Pockets



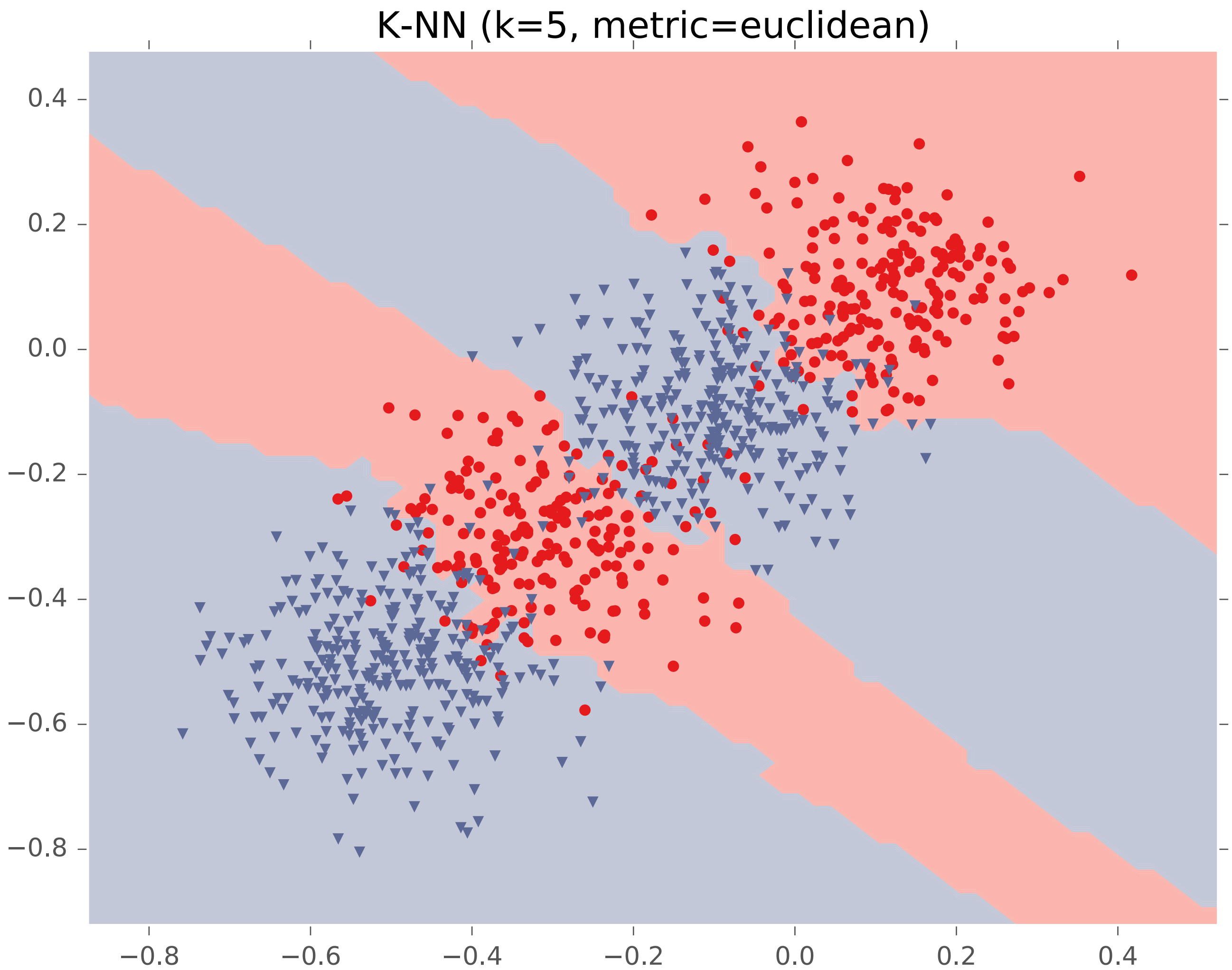
Example #3: Four Gaussians



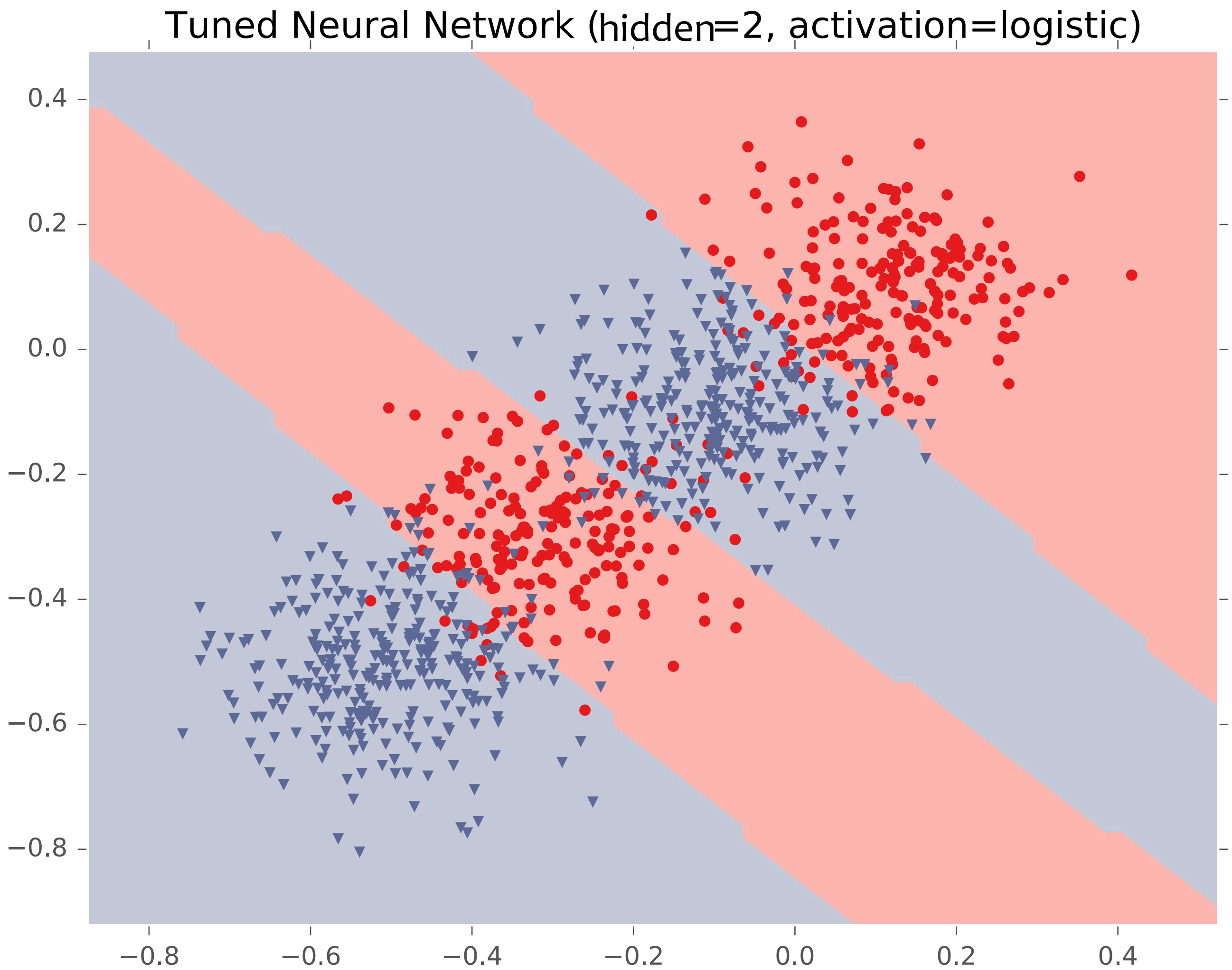
Example #3: Four Gaussians



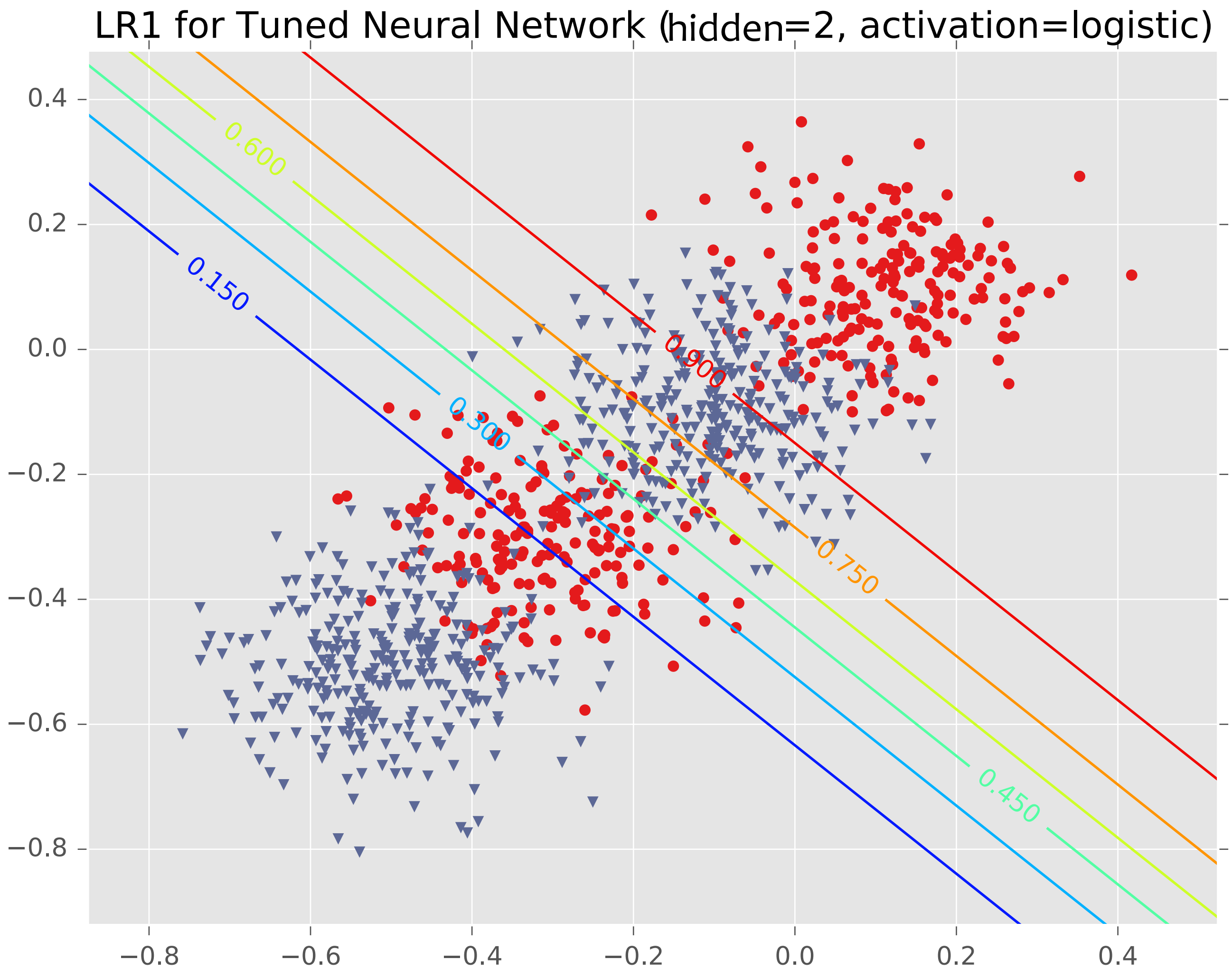
Example #3: Four Gaussians



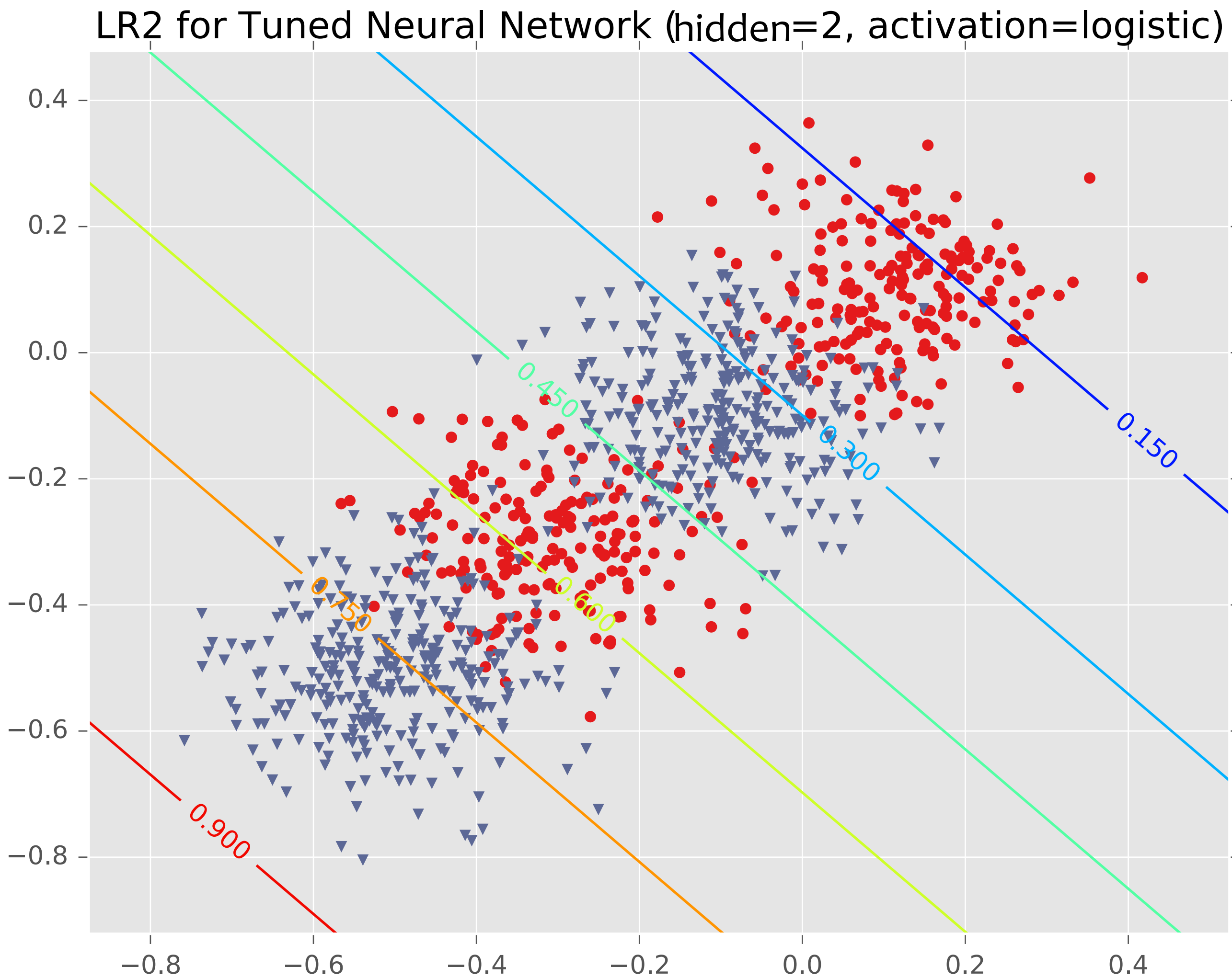
Example #3: Four Gaussians



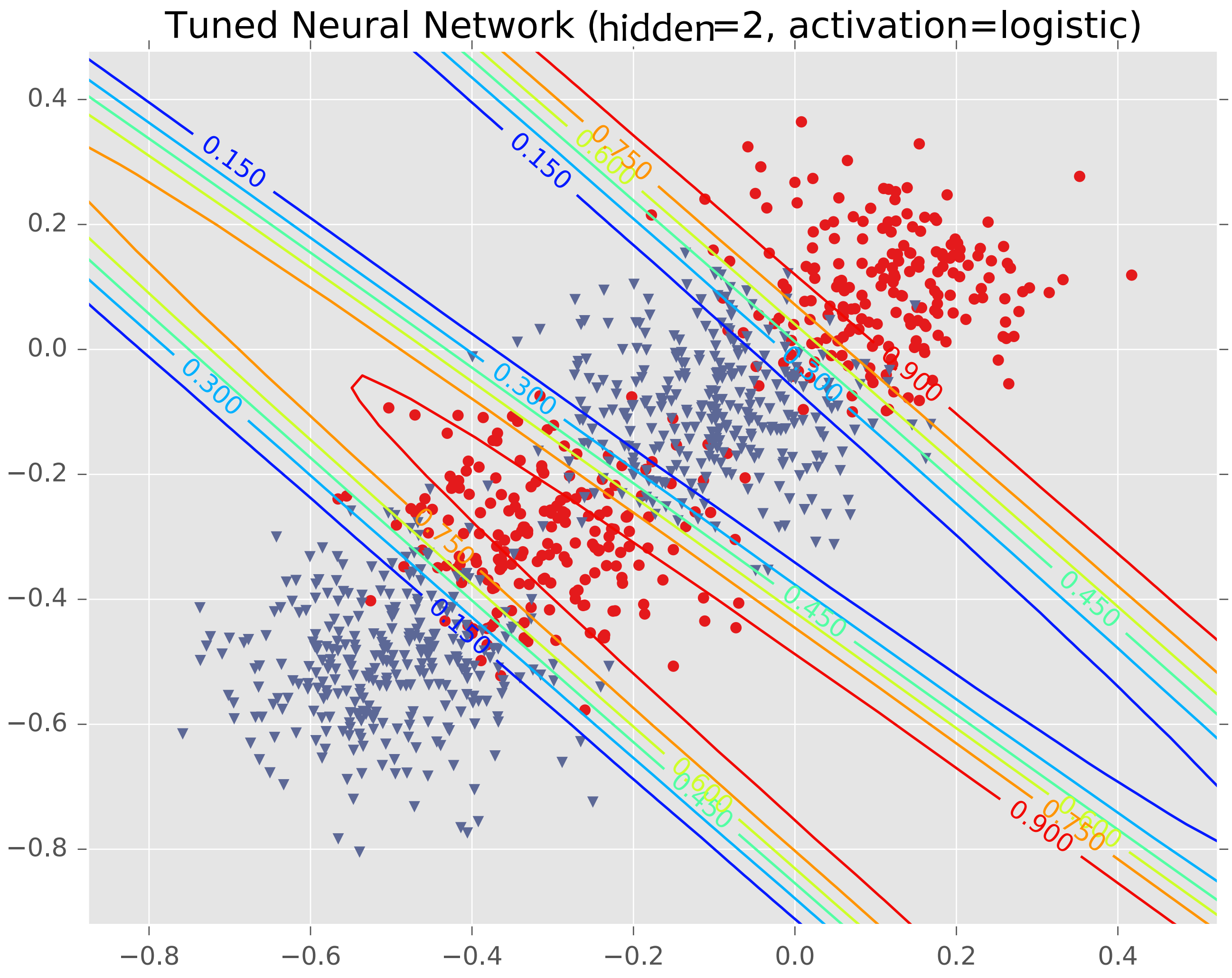
Example #3: Four Gaussians



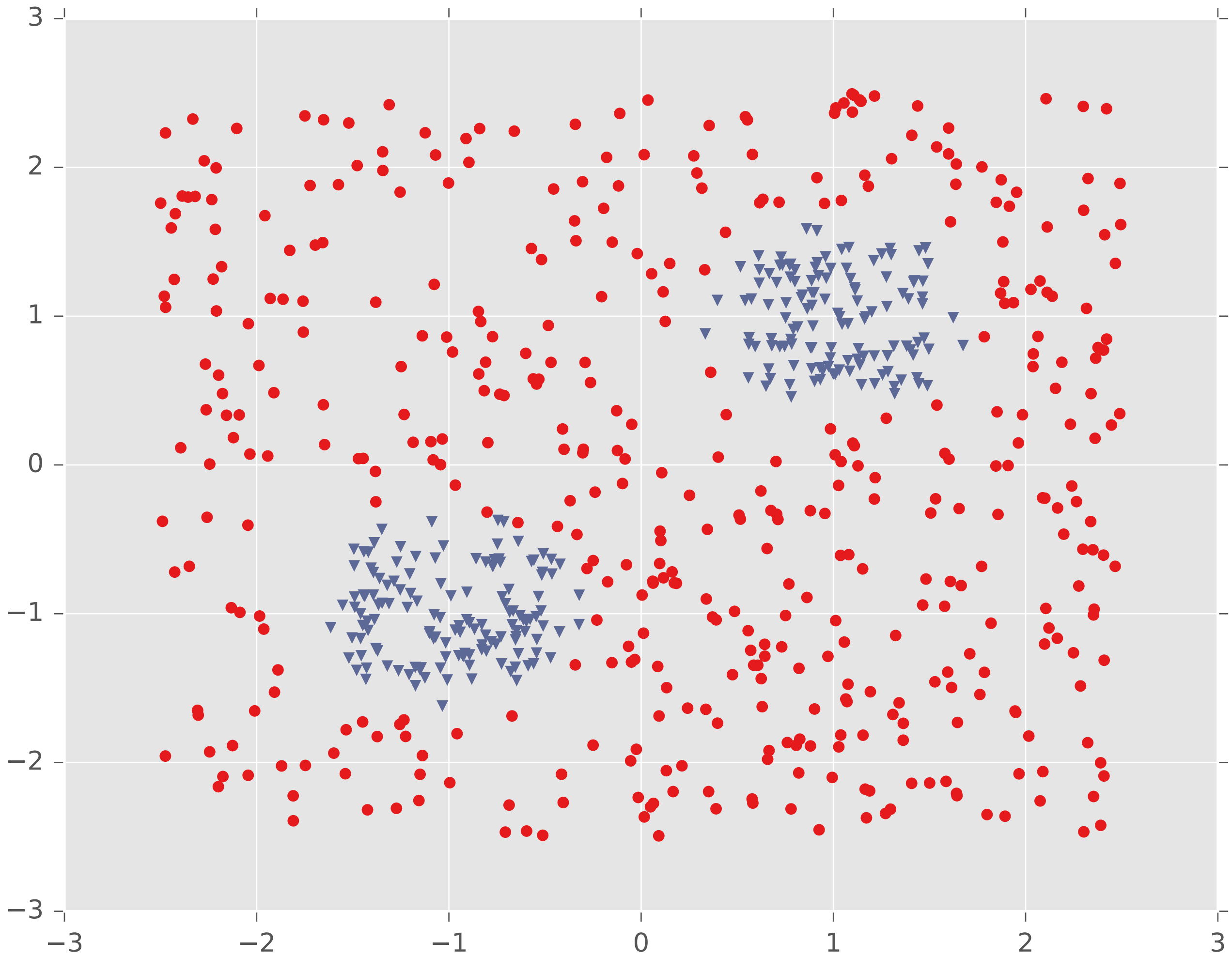
Example #3: Four Gaussians



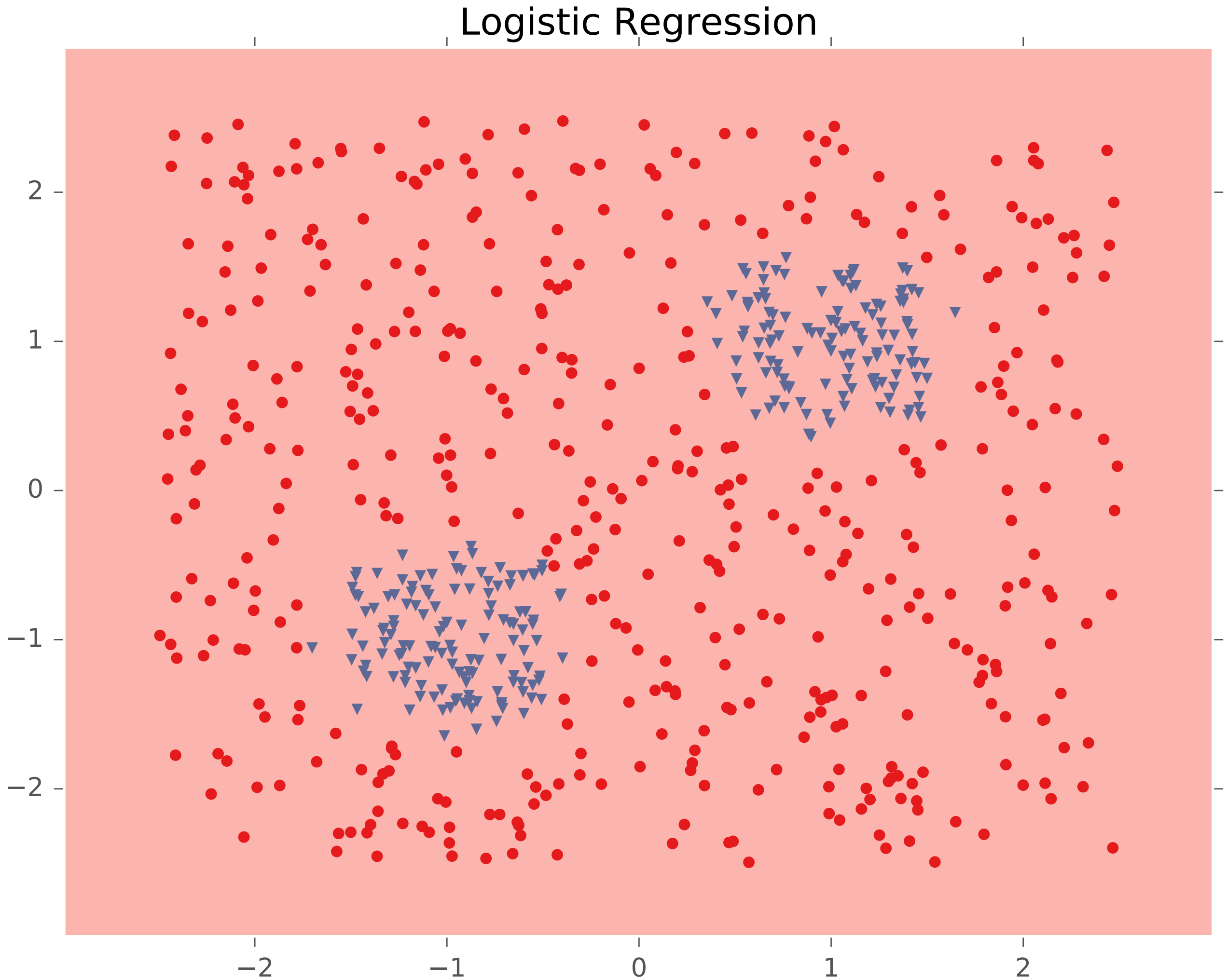
Example #3: Four Gaussians



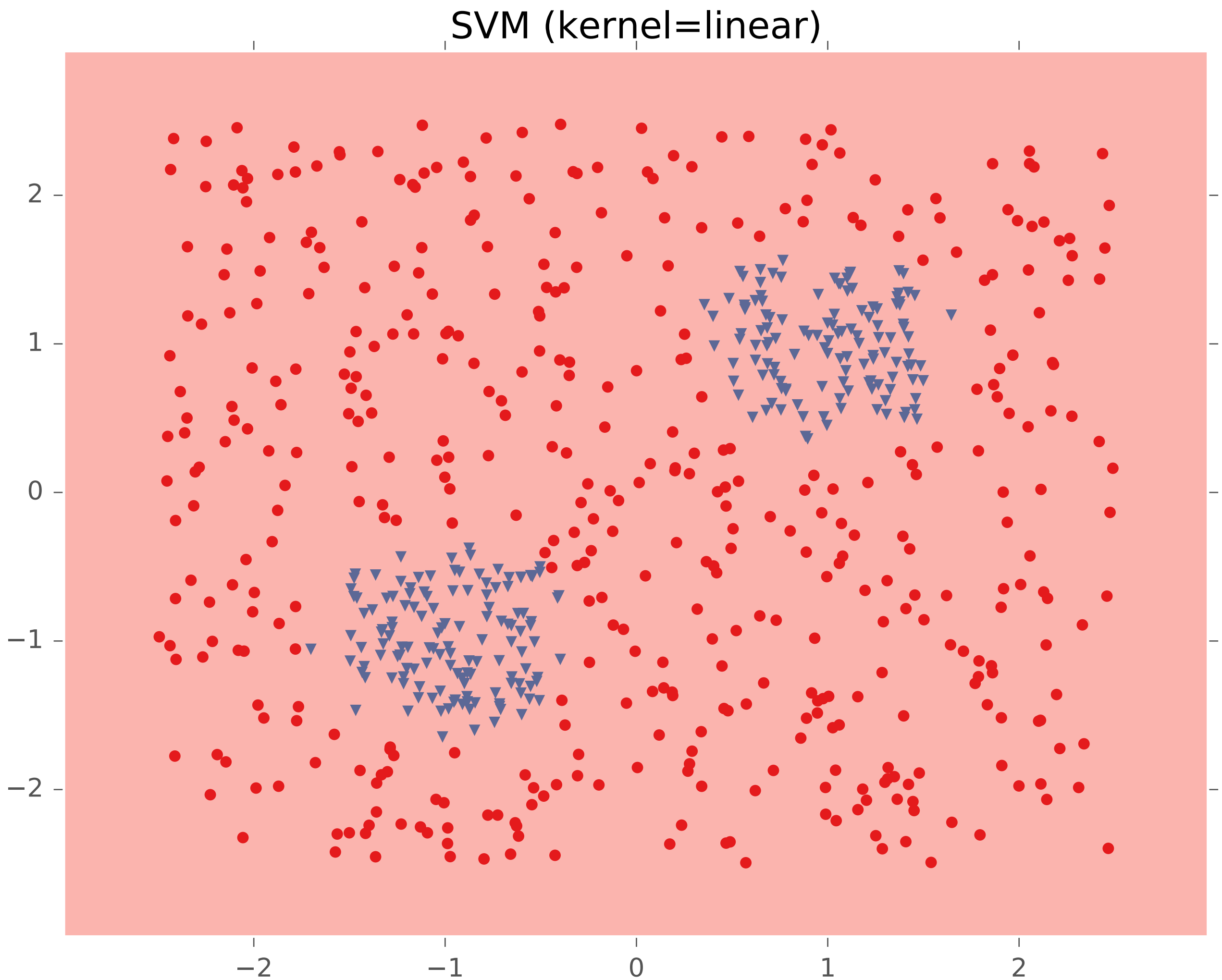
Example #4: Two Pockets



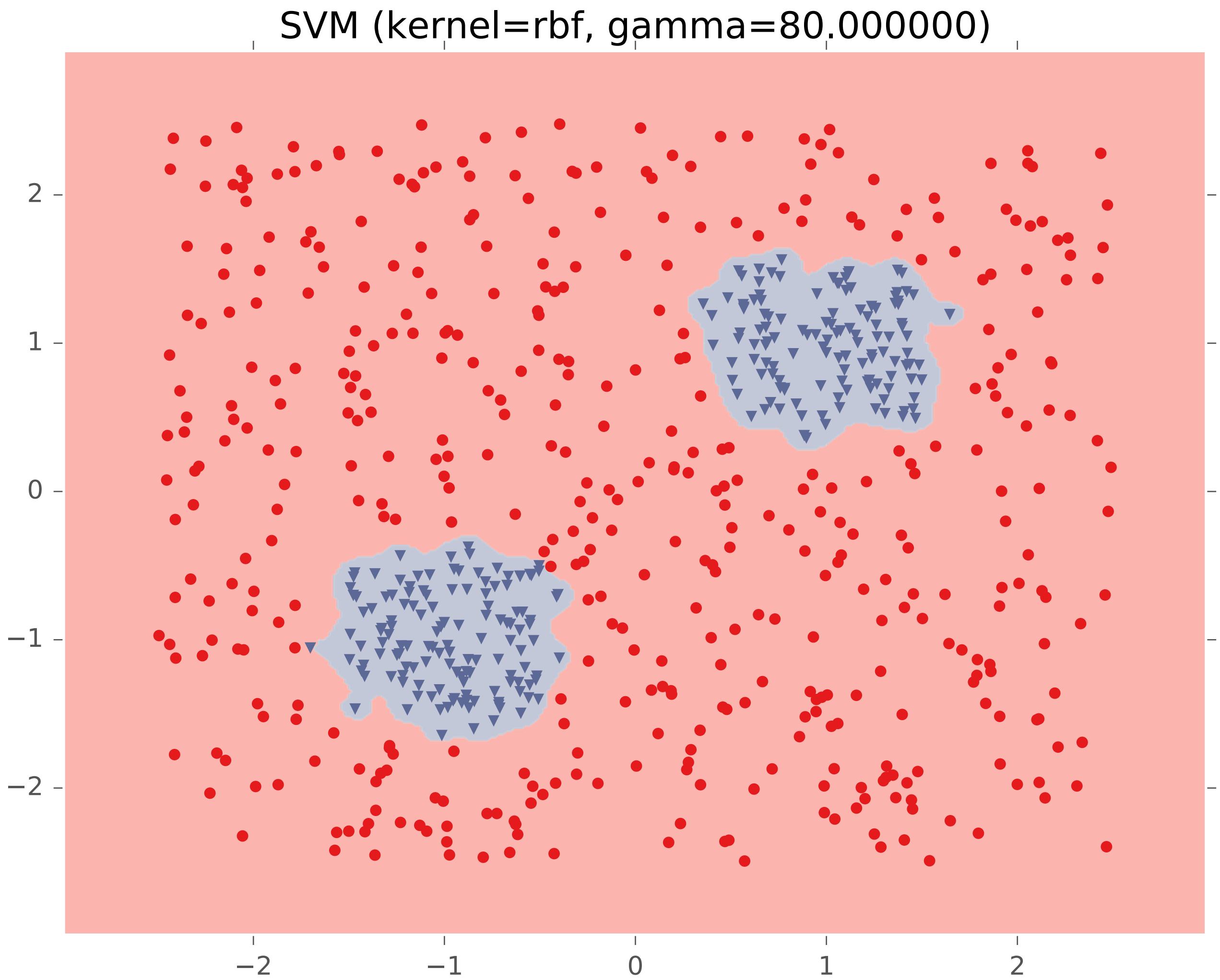
Example #4: Two Pockets



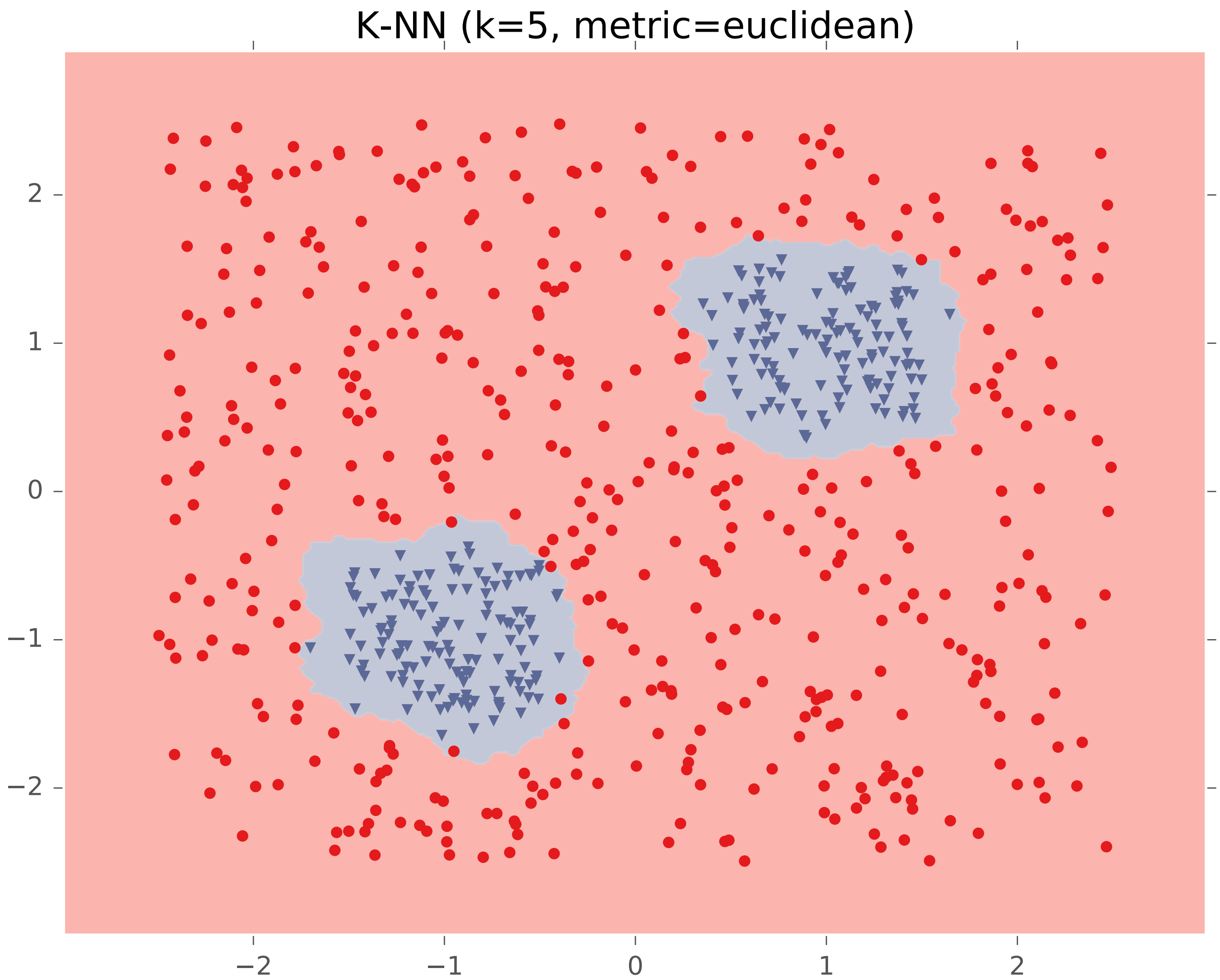
Example #4: Two Pockets



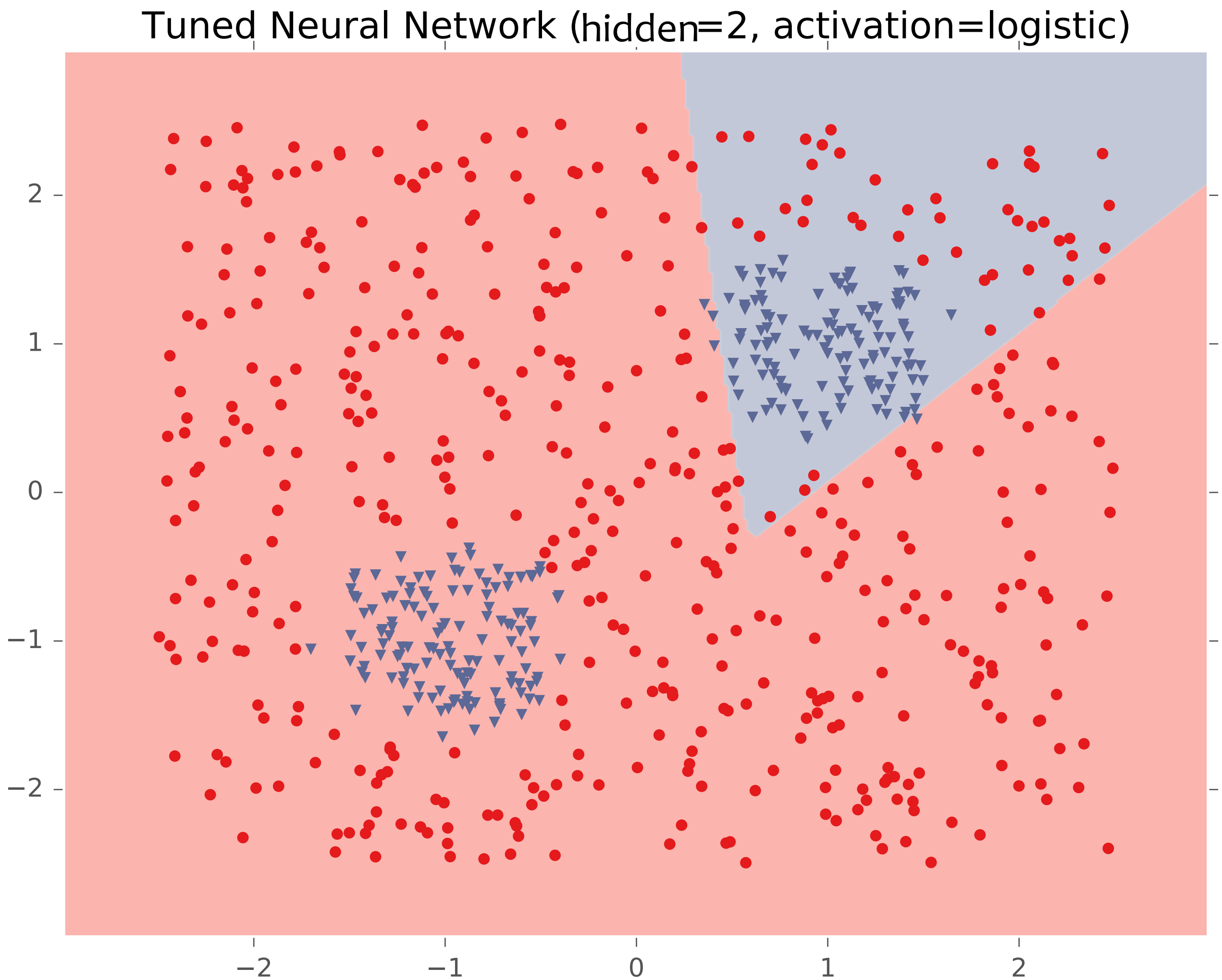
Example #4: Two Pockets



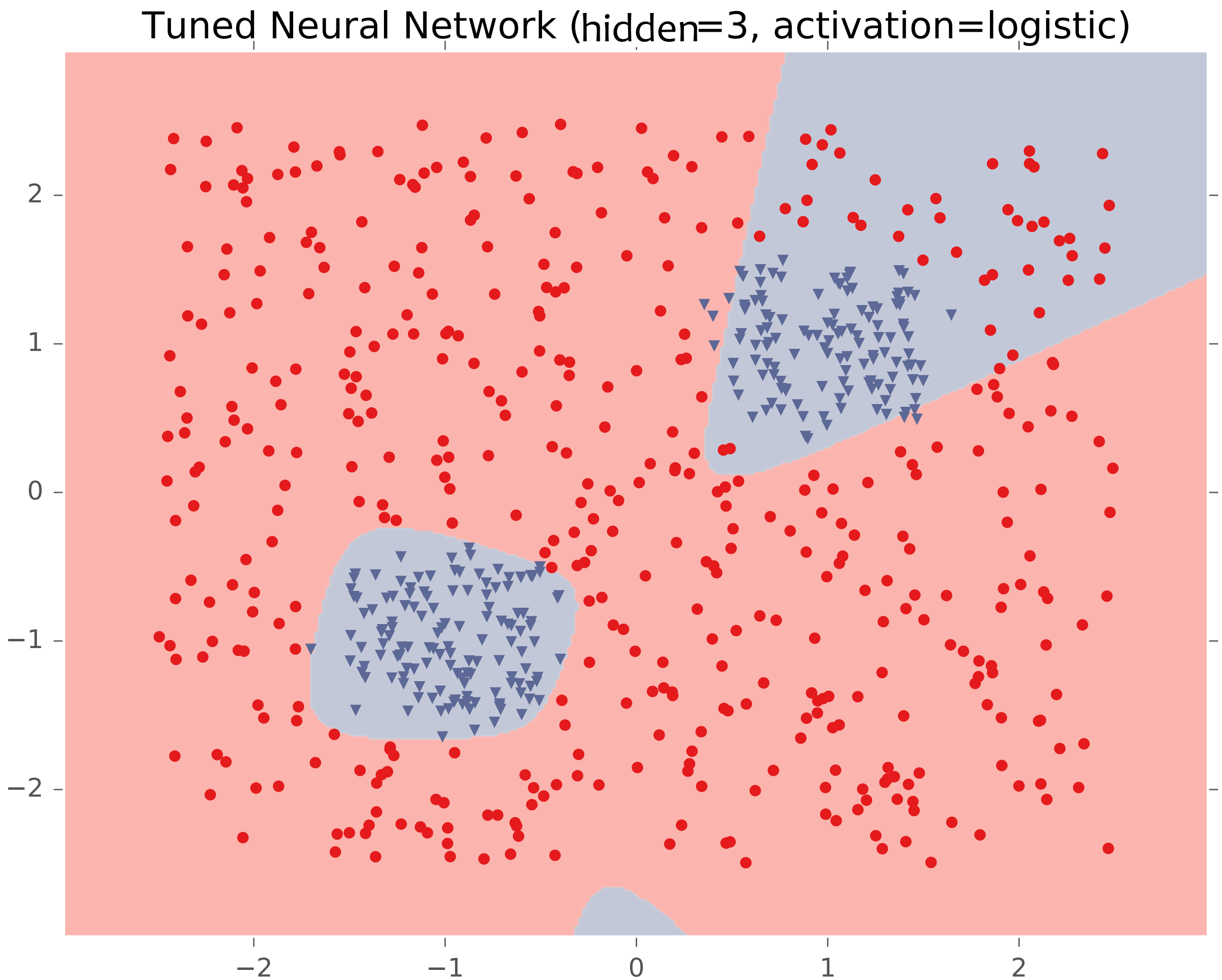
Example #4: Two Pockets



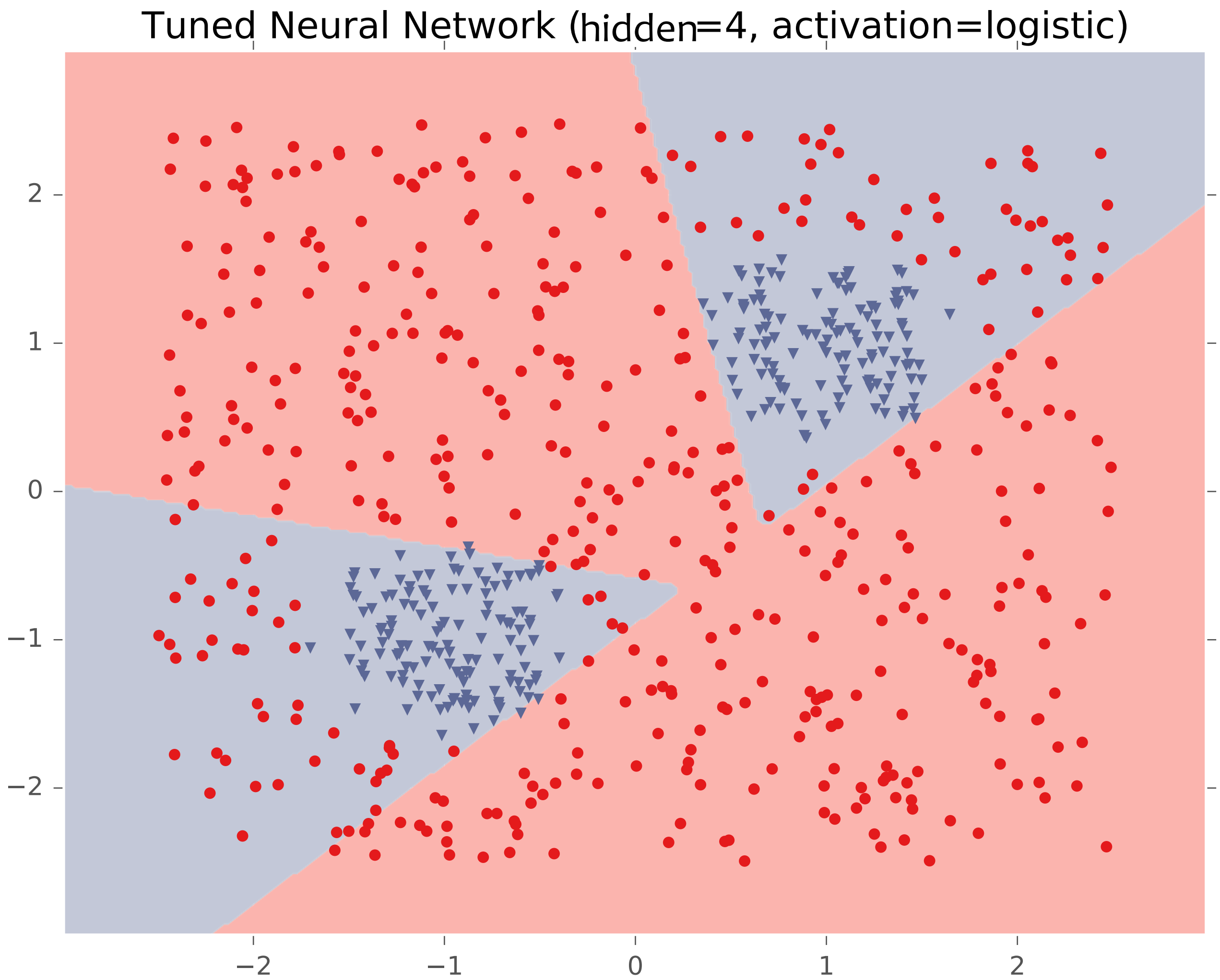
Example #4: Two Pockets



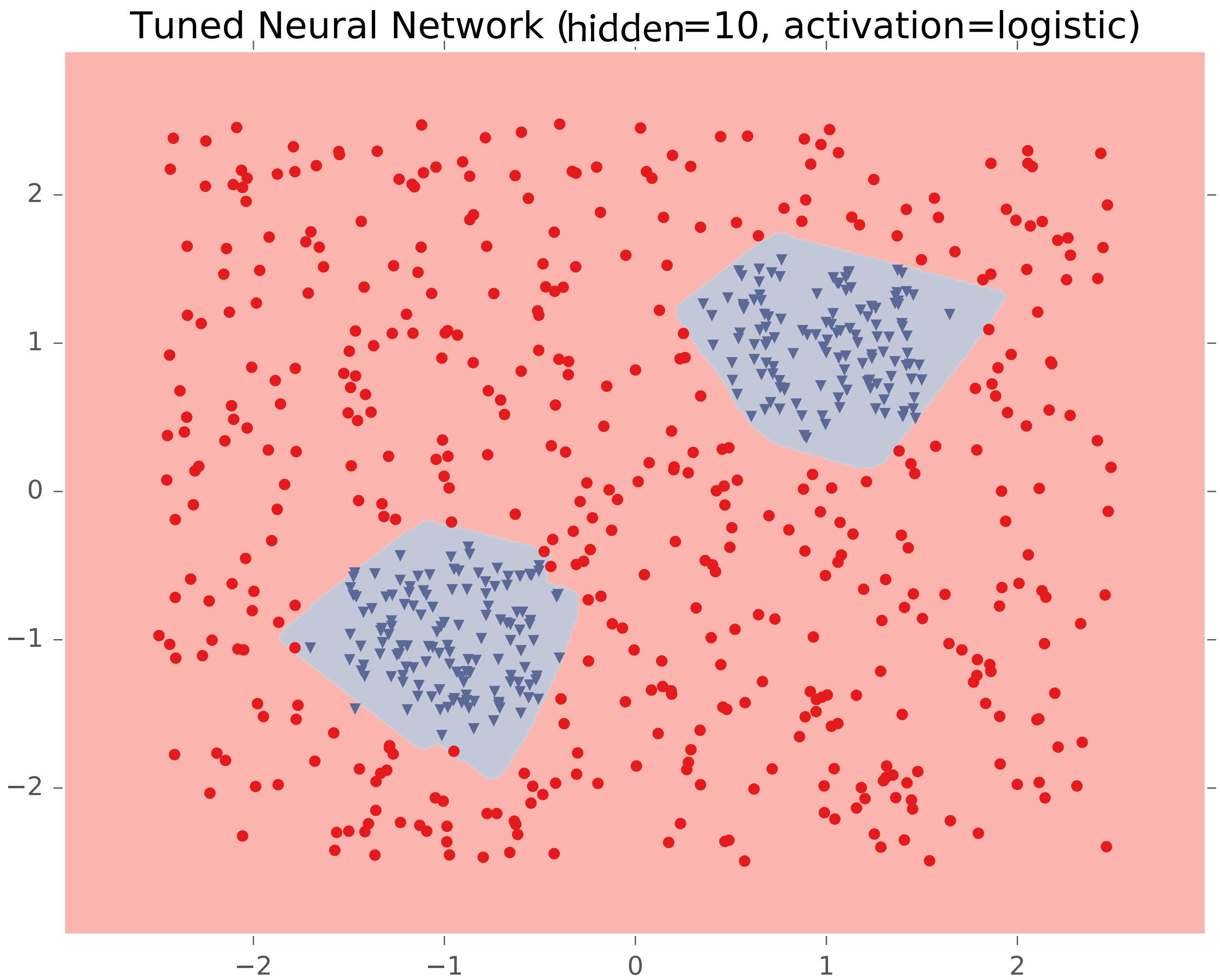
Example #4: Two Pockets



Example #4: Two Pockets



Example #4: Two Pockets



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