10-301/10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Neural Networks

Matt Gormley
Lecture 11
Oct. 4, 2023

Reminders

- **Homework 4: Logistic Regression**
  - Out: Fri, Sep 29
  - Due: Mon, Oct 9 at 11:59pm
- **Exam viewings**
- **Lecture on Friday**

A RECIPE FOR ML

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

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

Examples: Linear regression, Logistic regression, Neural Network

Examples: Mean-squared error, Cross Entropy

Background

A Recipe for Machine Learning

1. Given training data:

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

2. Choose each of these:

– Decision function

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

– Loss function

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

3. Define goal:

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

4. Train with SGD:

(take small steps opposite the gradient)

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

Background

A Recipe for

Gradients

Backpropagation can compute this gradient!

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

1. Given training dat

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

2. Choose each of t

– Decision function

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

– Loss function

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

opposite the gradient)

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

Backgrou

A Recipe for

Goals for Today's Lecture

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

1.

2. Choose each of these:

– Decision function

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

– Loss function

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

4. Train with SGD:

(take small steps opposite the gradient)

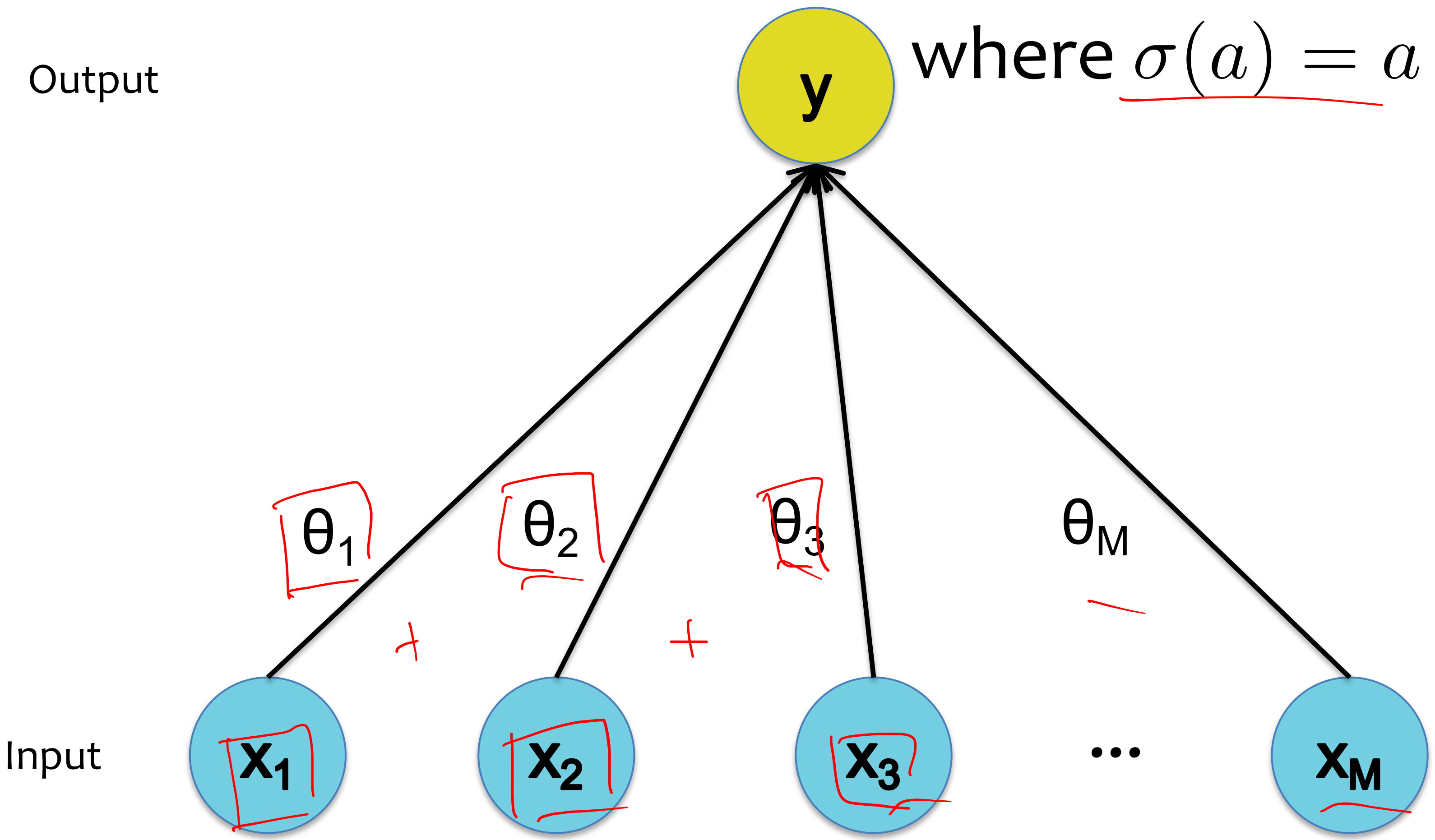
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Decision Functions

Linear Regression

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

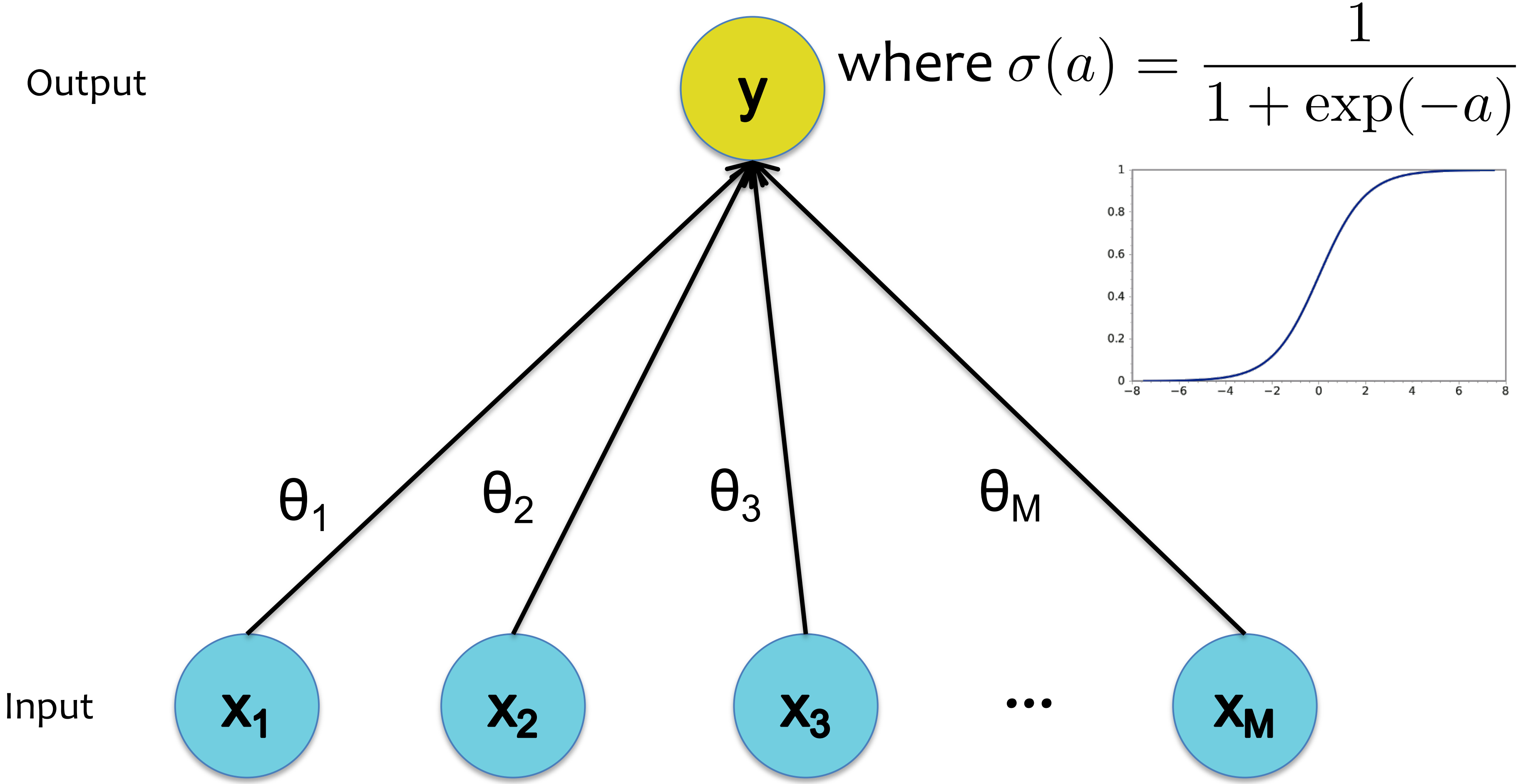
where $\sigma(a) = a$



Logistic Regression

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

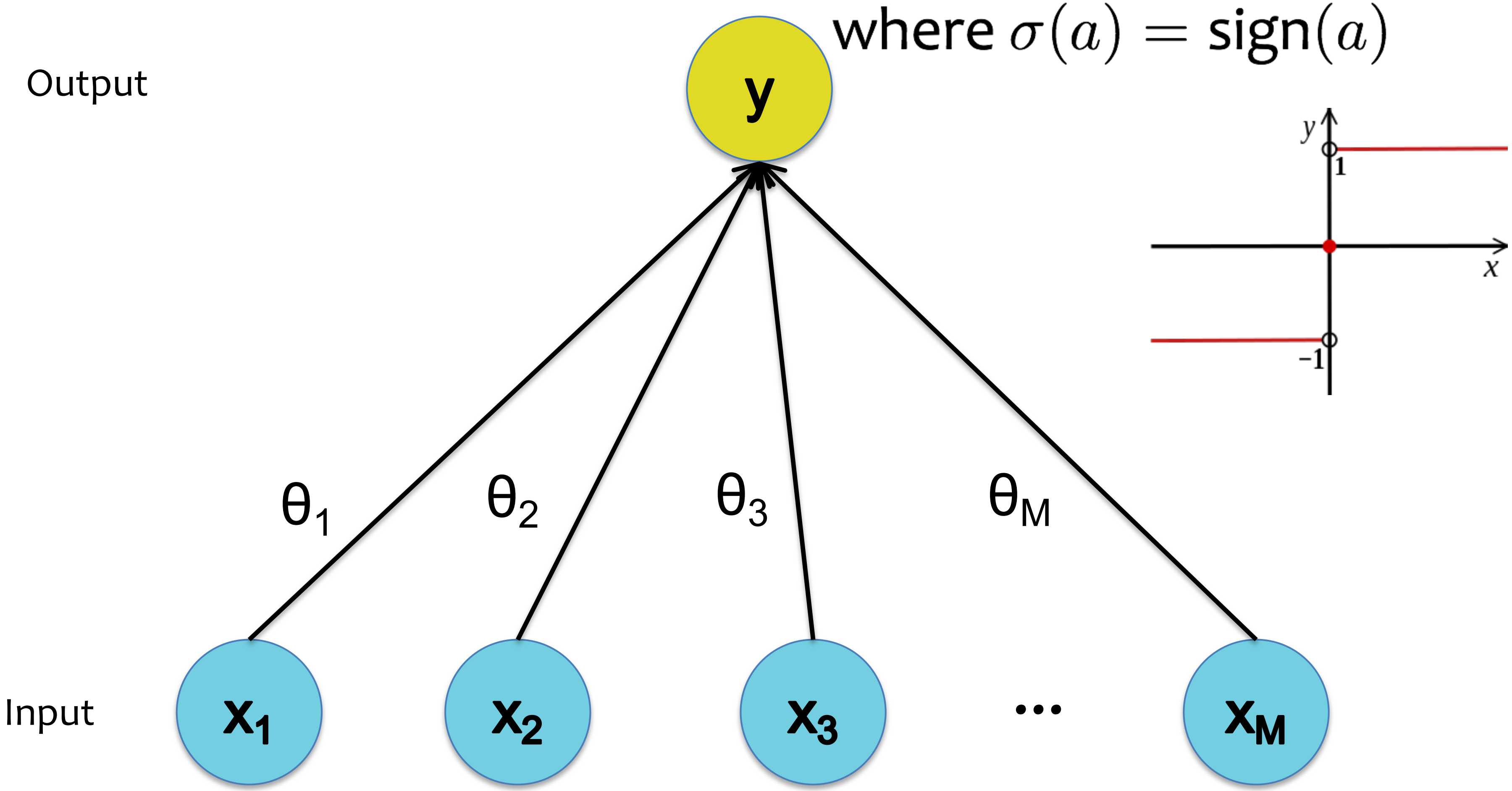
where $\sigma(a) = \frac{1}{1 + \exp(-a)}$



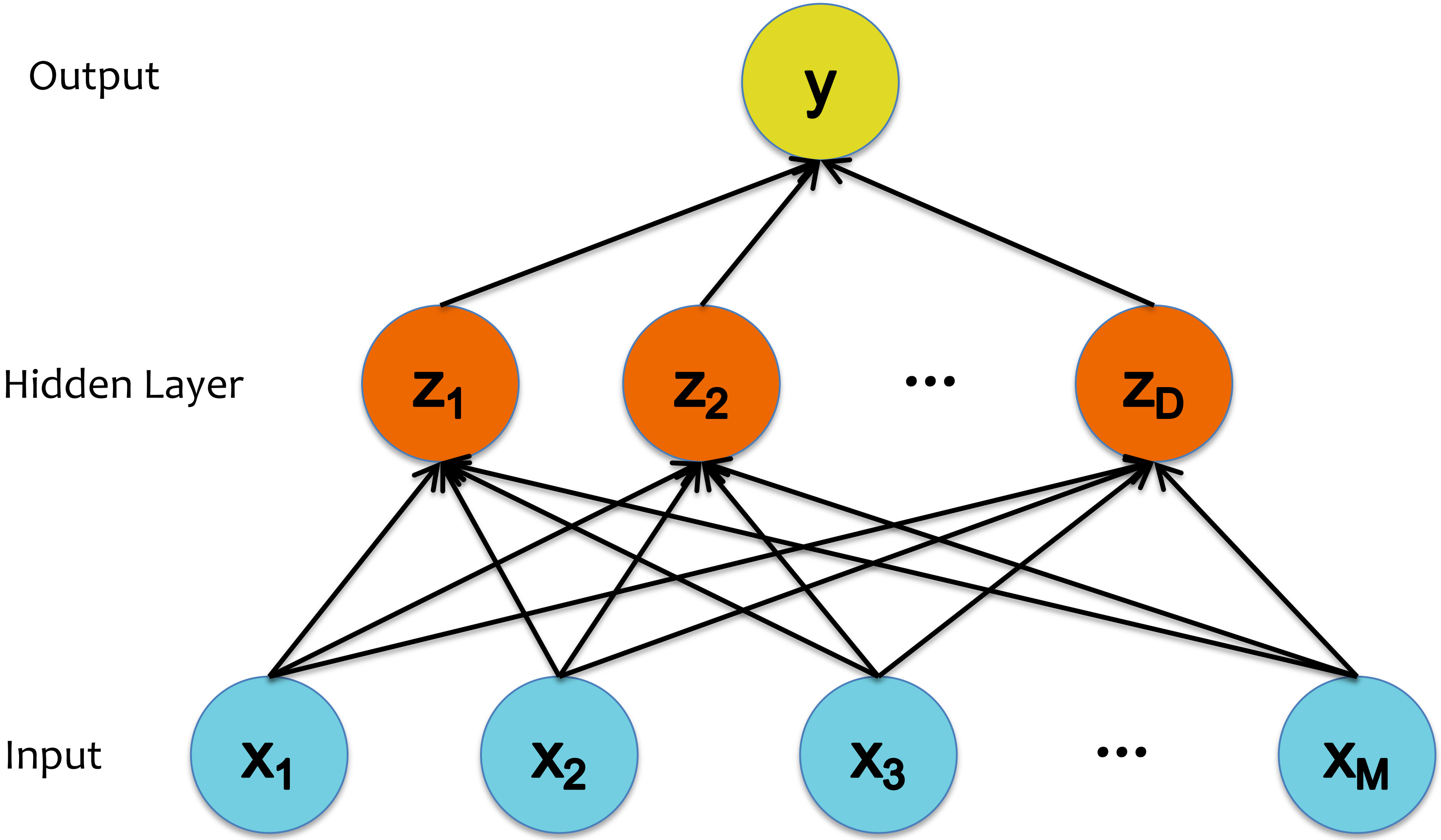
Perceptron

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

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

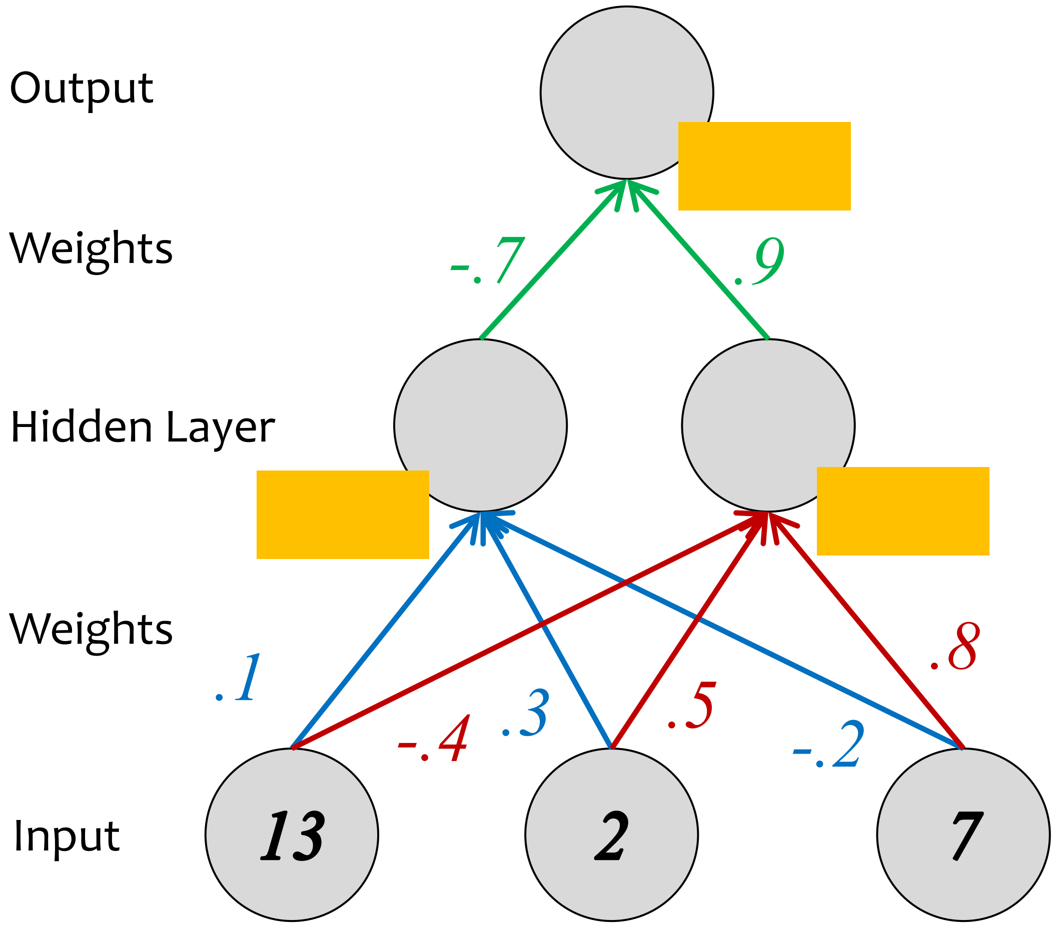


Neural Network



COMPONENTS OF A NEURAL NETWORK

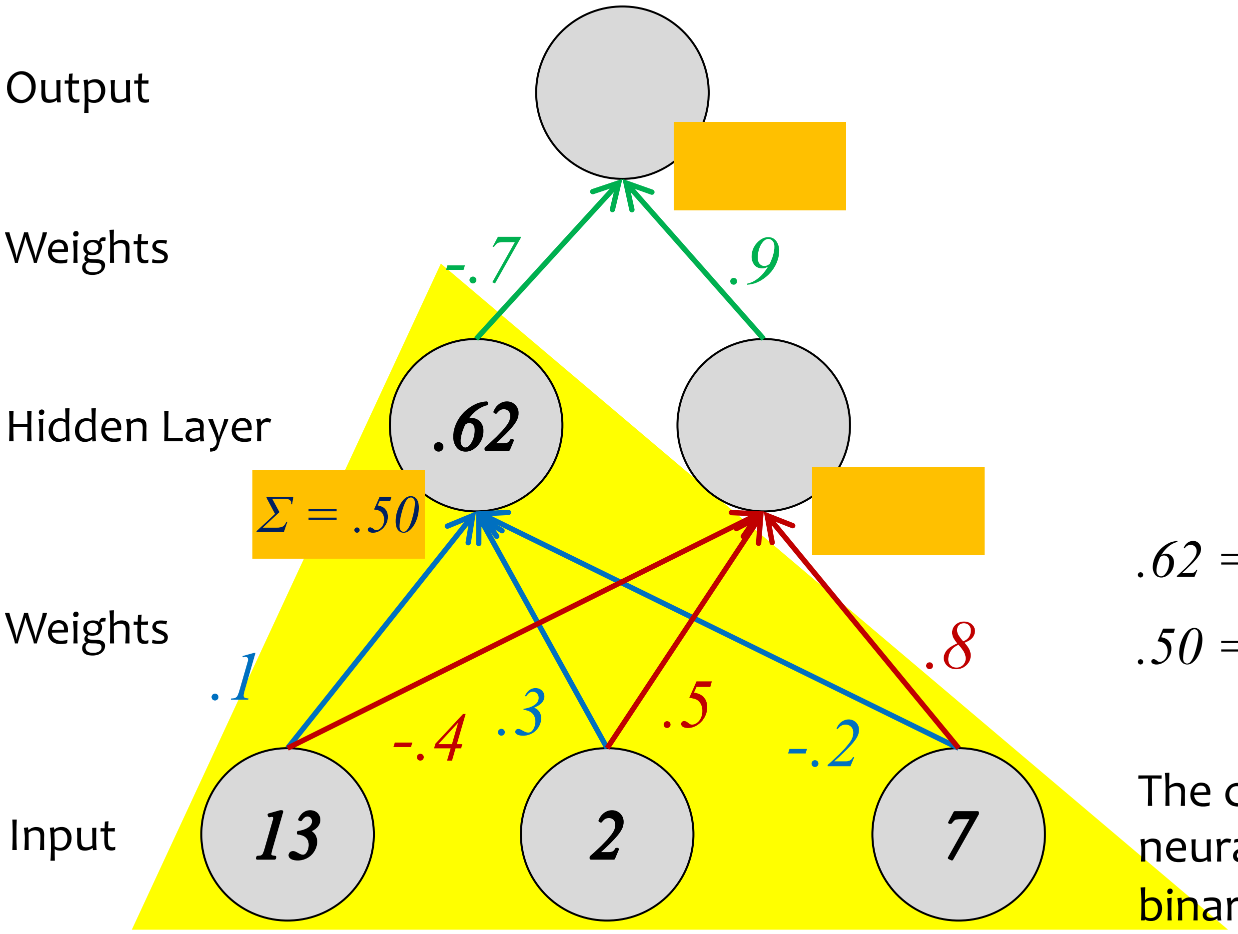
Neural Network



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.

Neural Network

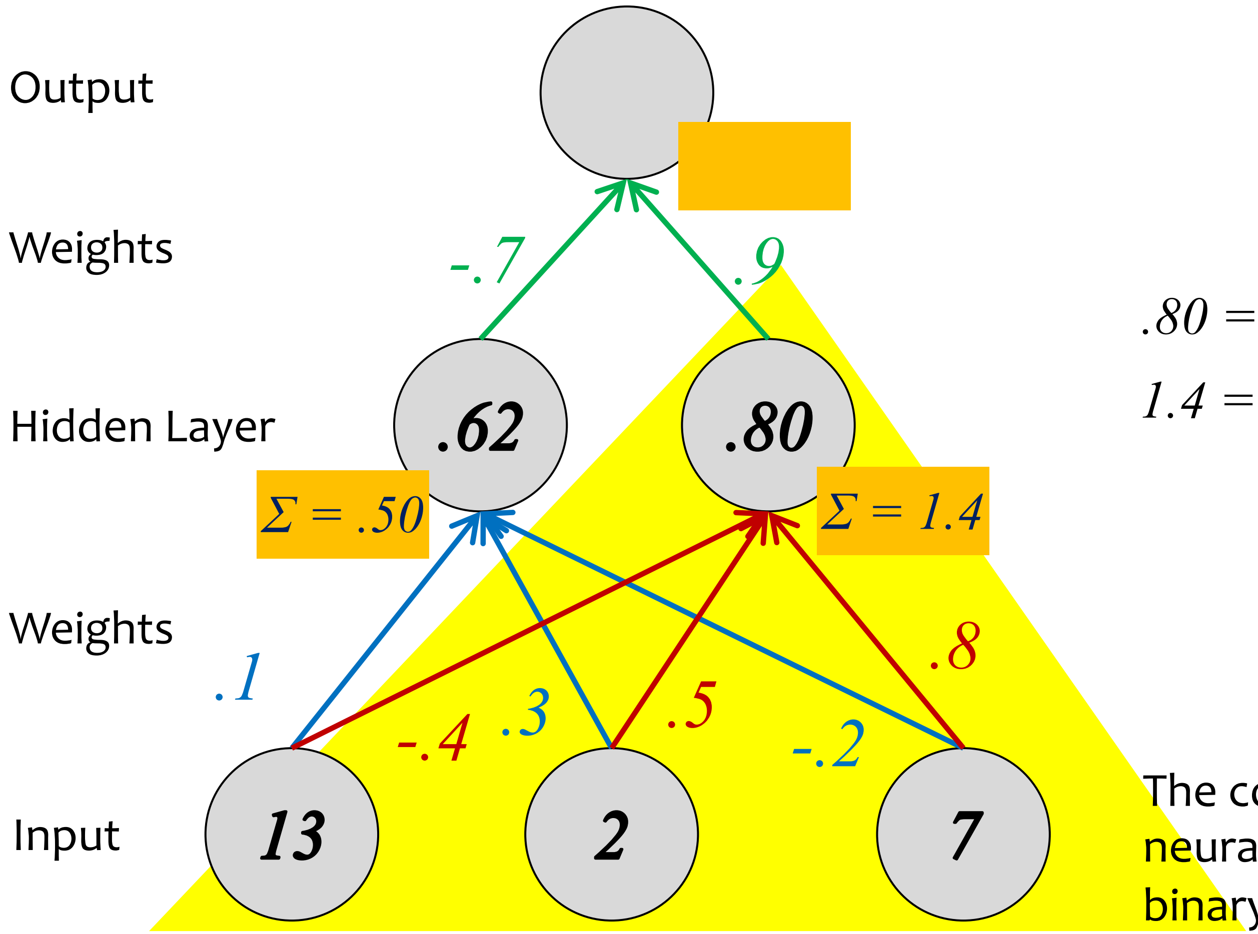


$.62 = \sigma(.50)$

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

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

Neural Network

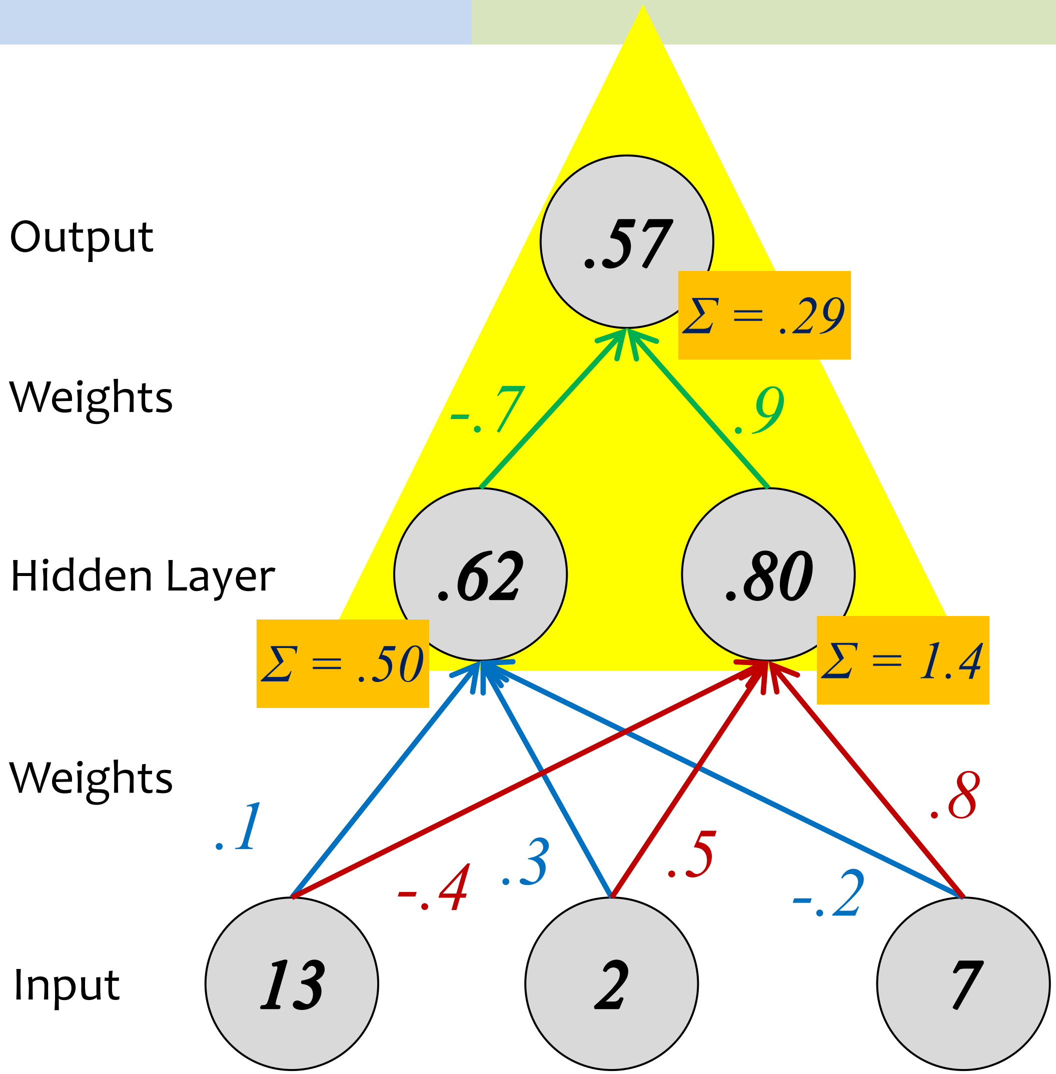


$.80 = \sigma(1.4)$

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

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

Neural Network

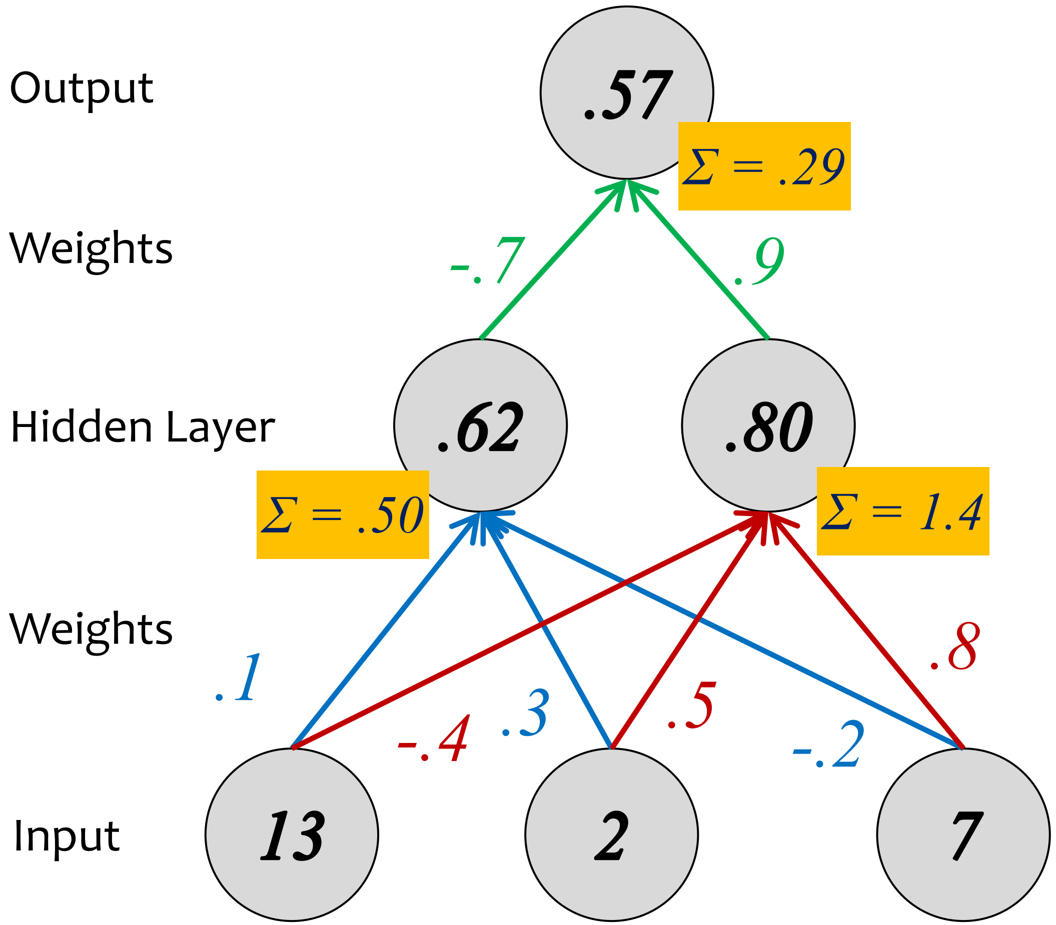


$.57 = \sigma(.29)$

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

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

Neural Network



$$.57 = \sigma(.29)$$
$$.29 = .62(-.7) + .80(.9)$$

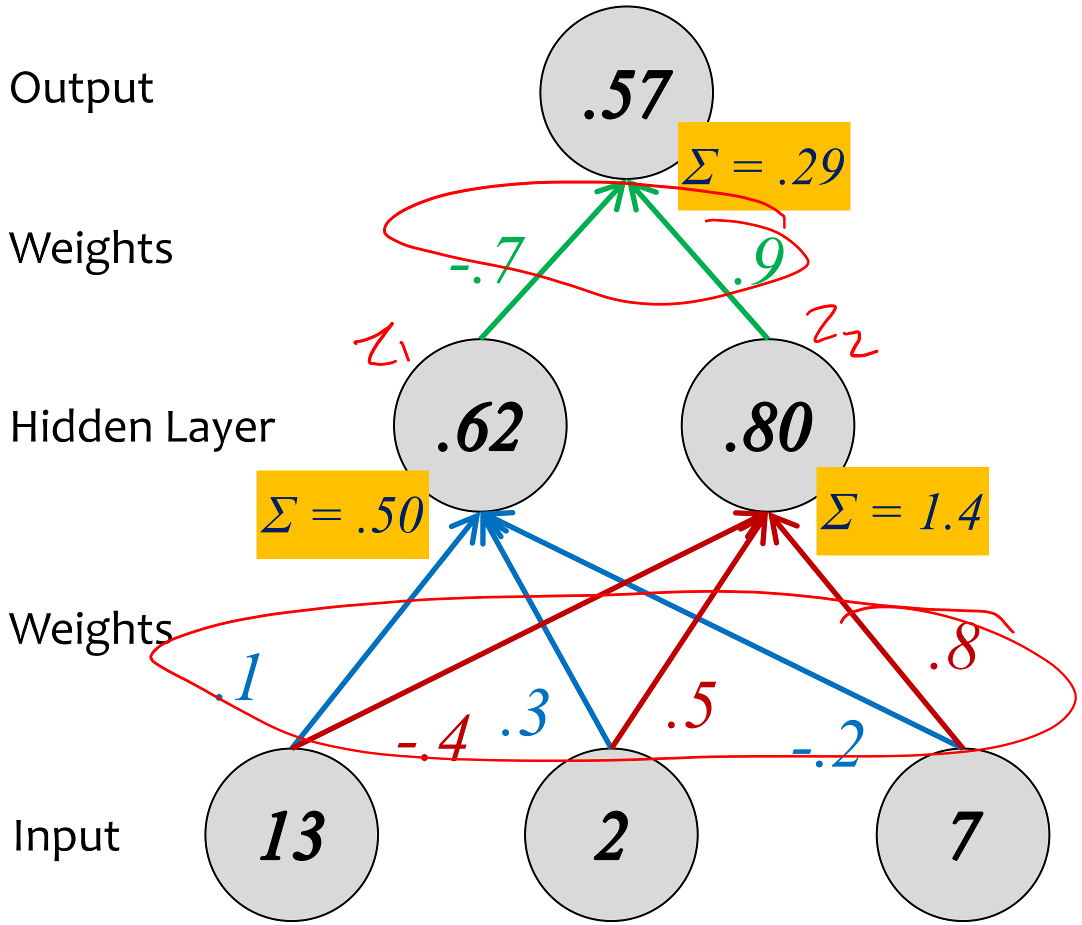
$$.80 = \sigma(1.4)$$
$$1.4 = 13(-.4) + 2(.5) + 7(.8)$$

$$.62 = \sigma(.50)$$
$$.50 = 13(.1) + 2(.3) + 7(-.2)$$

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

Decision Functions

Neural Network



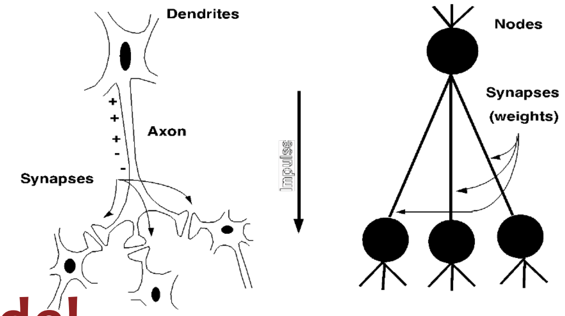
Except we only have the target value for y at training time!

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

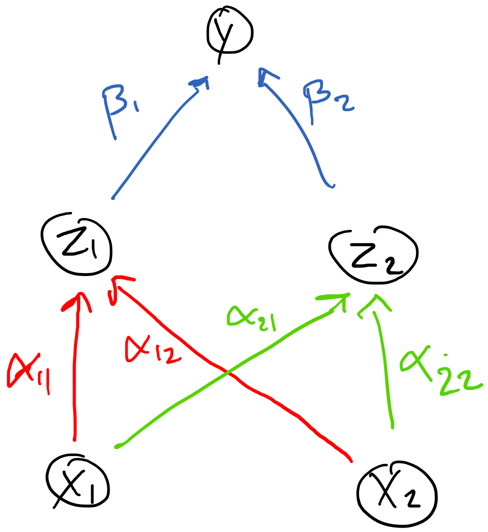
Artificial Computation

- 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

Example: Neural Network with One Hidden Layer



$x_m \in \mathbb{R}$

$z_m \in \mathbb{R}$ (general case)

$z_m \in (0,1)$ (for sigmoid act.)

Let σ be an activation function

If σ is sigmoid: $\sigma(a) = \frac{1}{1+\exp(-a)}$

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

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

$$y = \sigma(\beta_1 z_1 + \beta_2 z_2 + \beta_0)$$

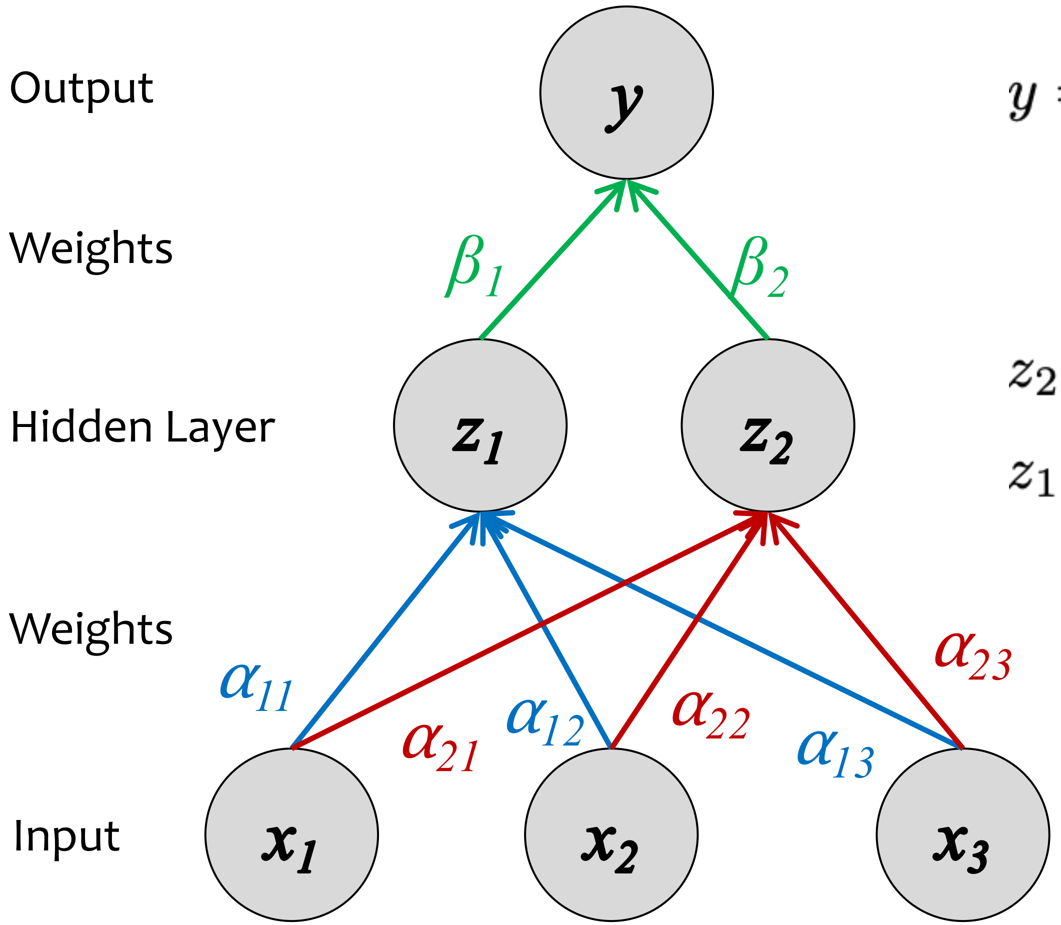
$$= \sigma(\beta_1 \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{10}) + \beta_2 \sigma(\alpha_{21} x_1 + \alpha_{22} x_2 + \alpha_{20}) + \beta_0)$$

$\Pr[Y=1 \mid \vec{x}, \alpha, \beta]$

To predict use Bayes Optimal Classifier

$$\hat{y} = h_{\alpha,\beta}(\vec{x}) = \begin{cases} 1 & \text{if } y \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Neural Network



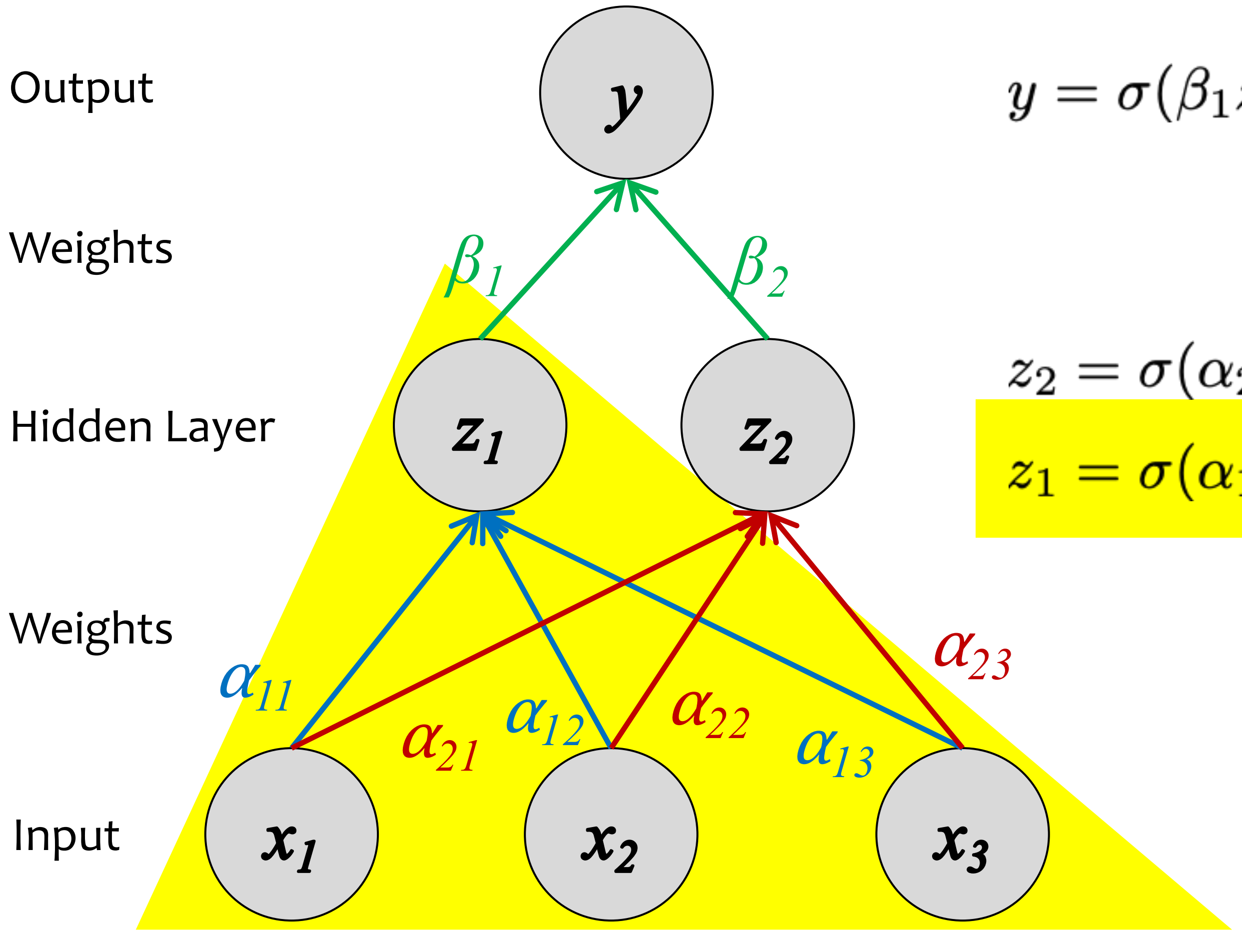
$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Decision Functions

Neural Network

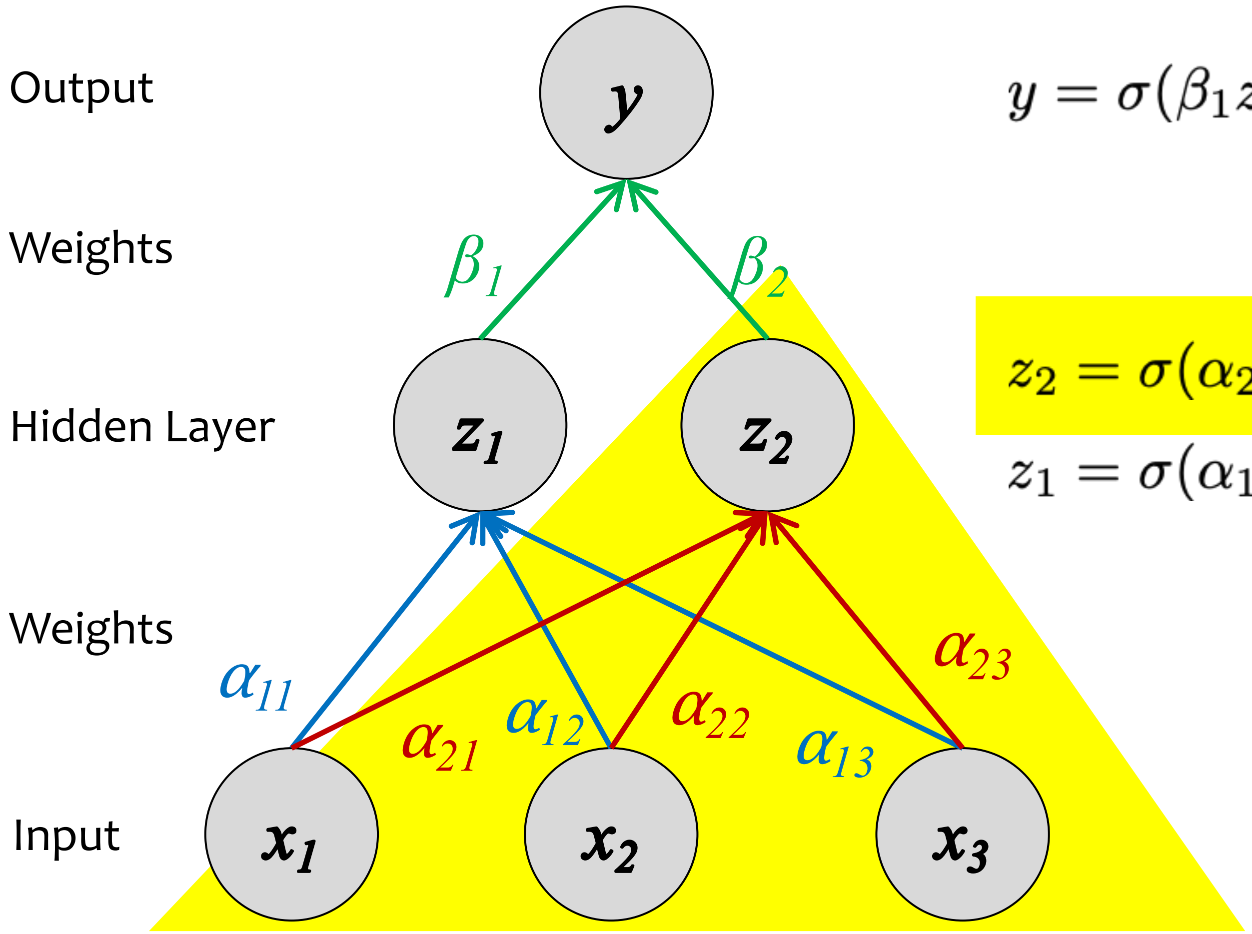


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Neural Network

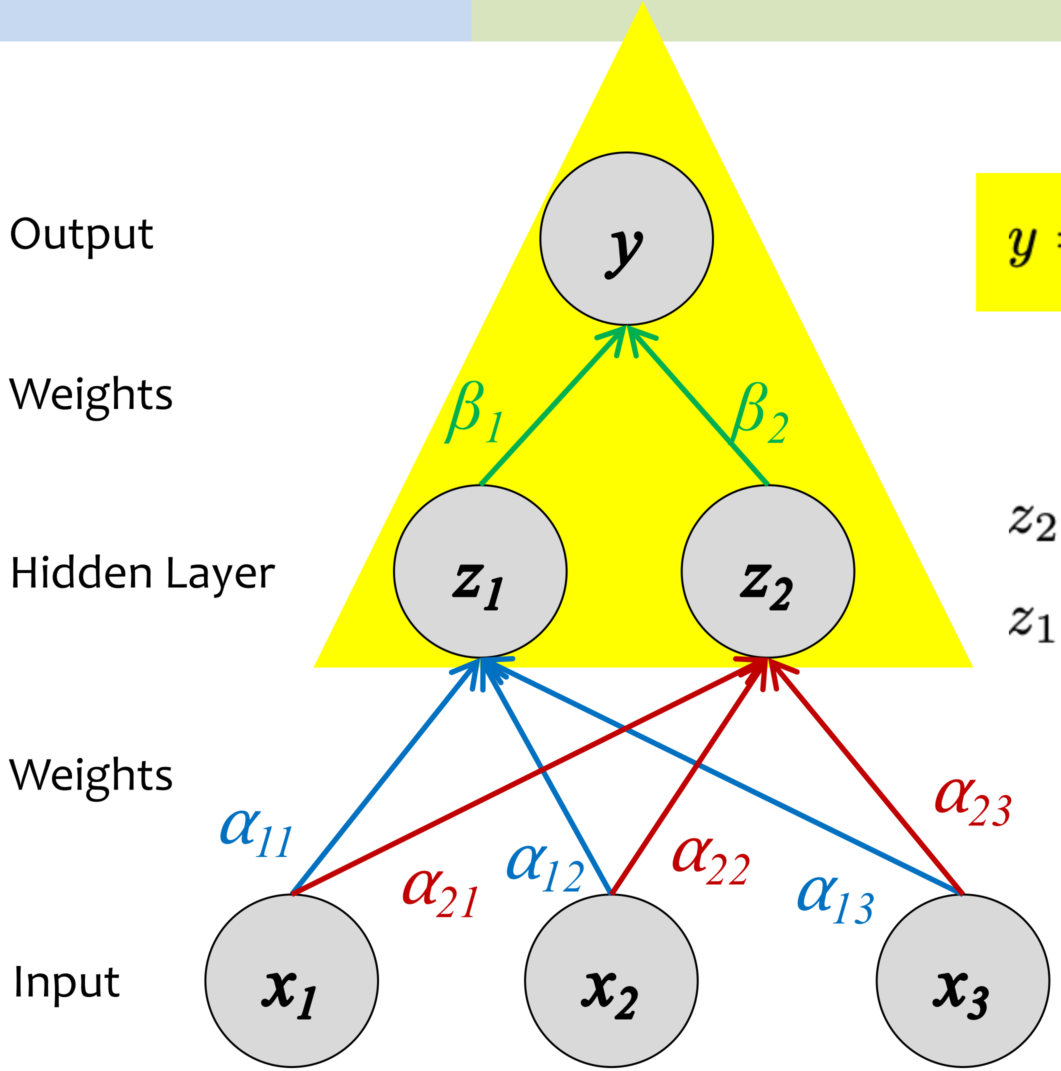


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Neural Network

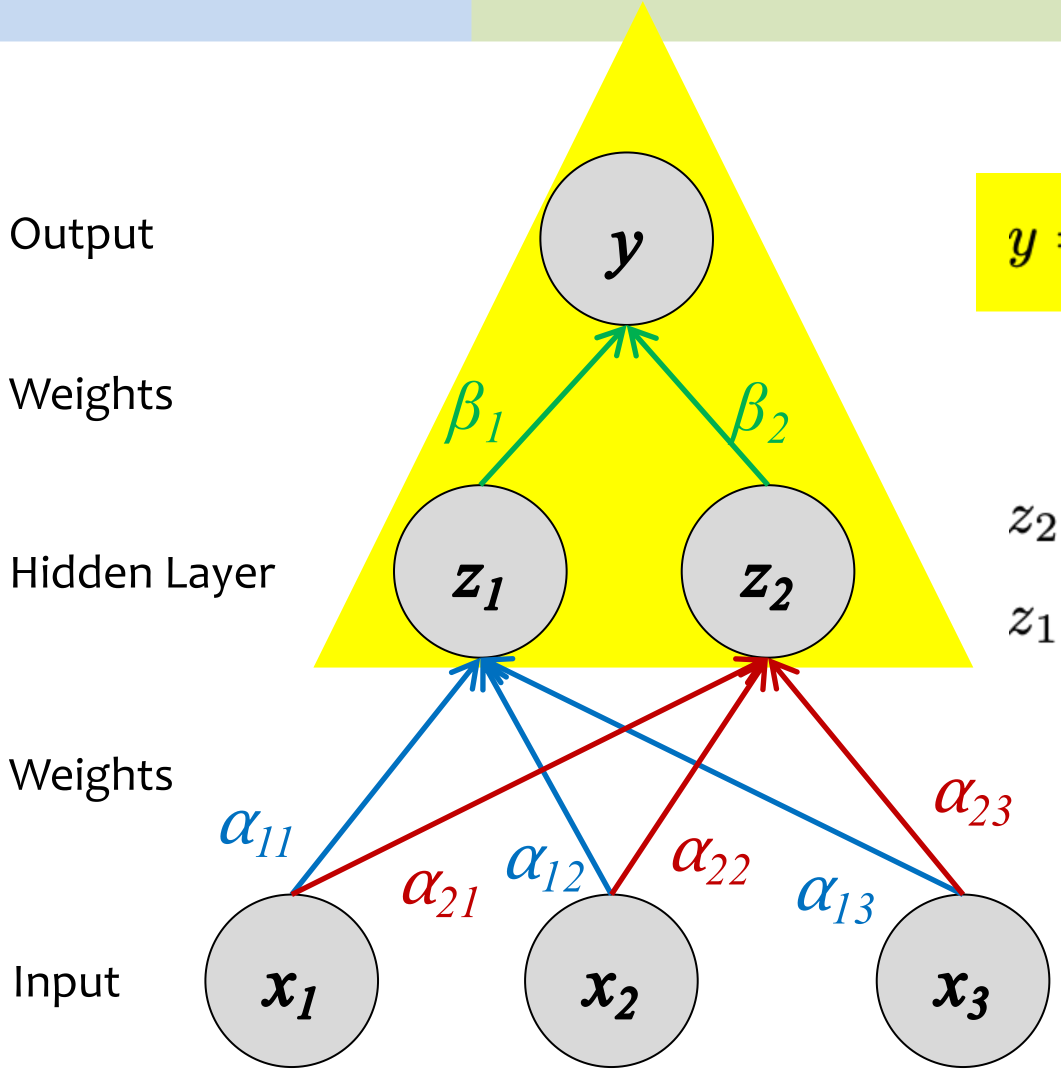


$$y = \sigma(\beta_1 z_1 + \beta_2 z_2)$$

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

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

Neural Network



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

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

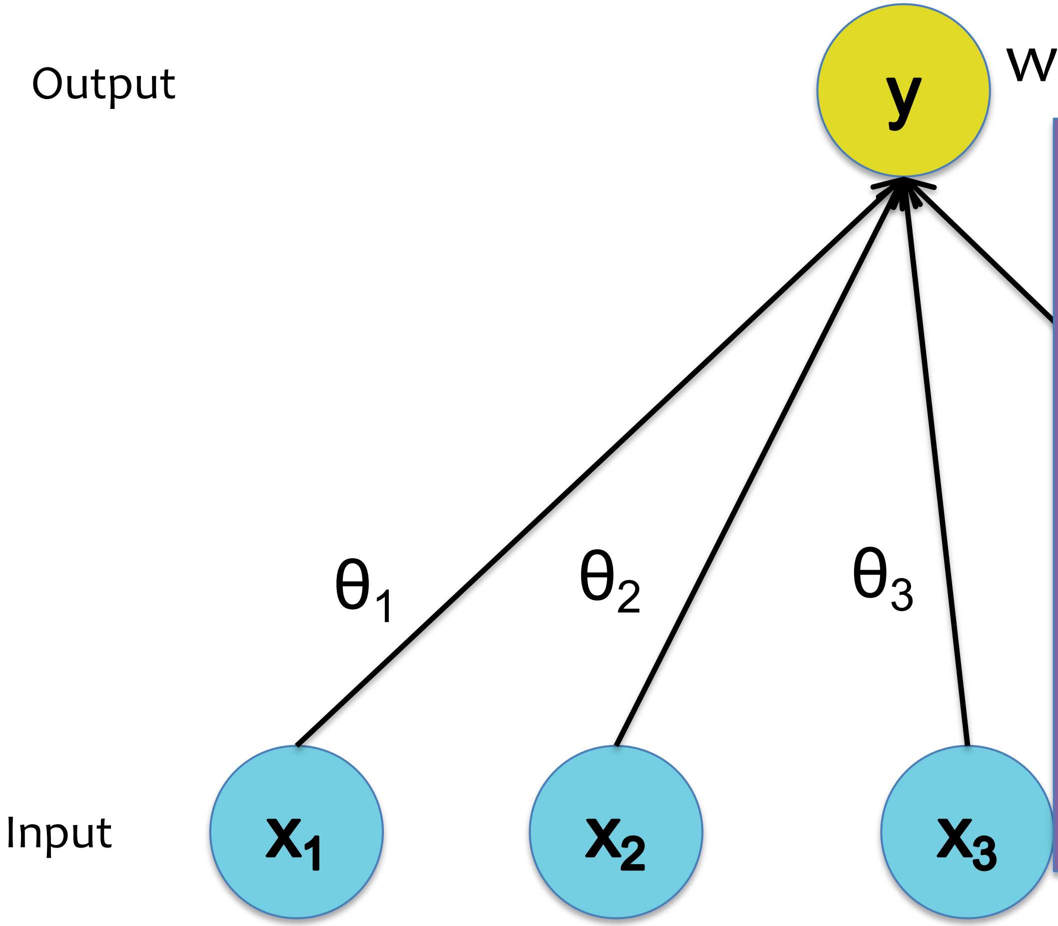
$$z_1 = \sigma(\alpha_{11} x_1 + \alpha_{12} x_2 + \alpha_{13} x_3)$$

NONLINEAR DECISION BOUNDARIES AND NEURAL NETWORKS

Logistic Regression

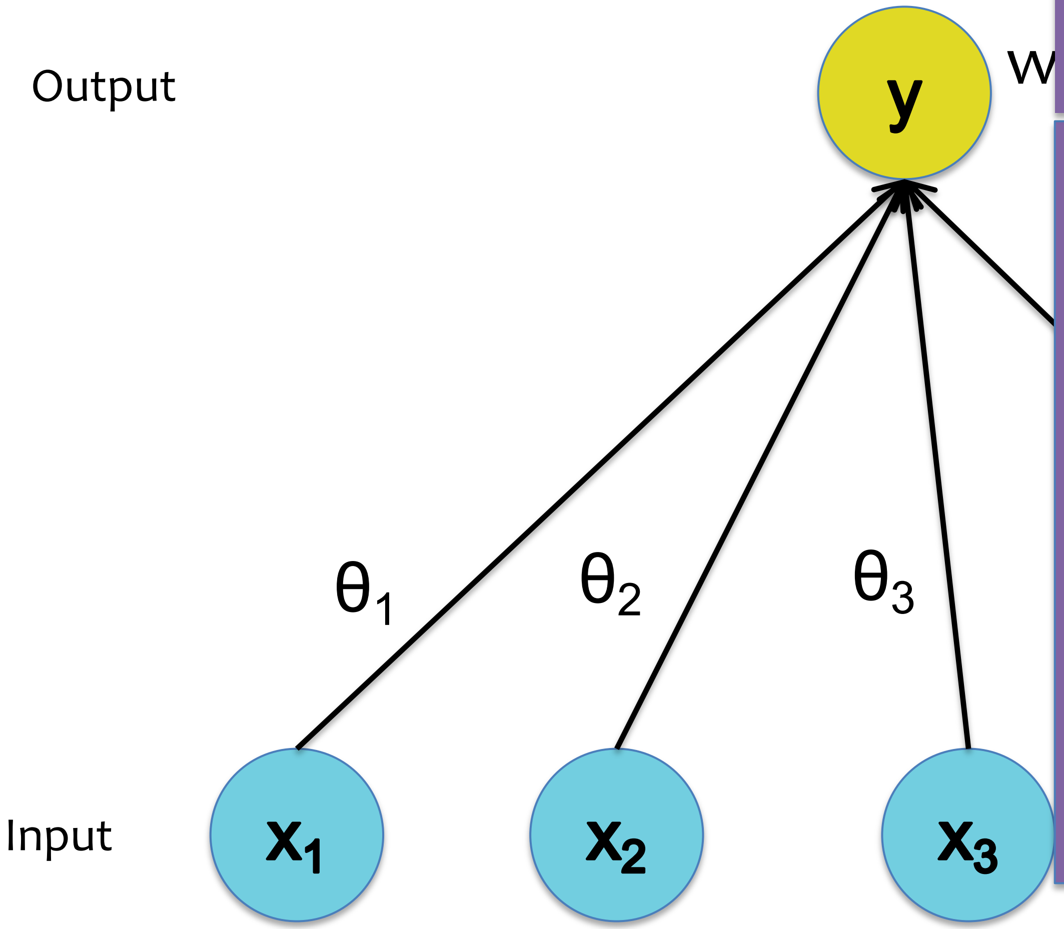
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

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

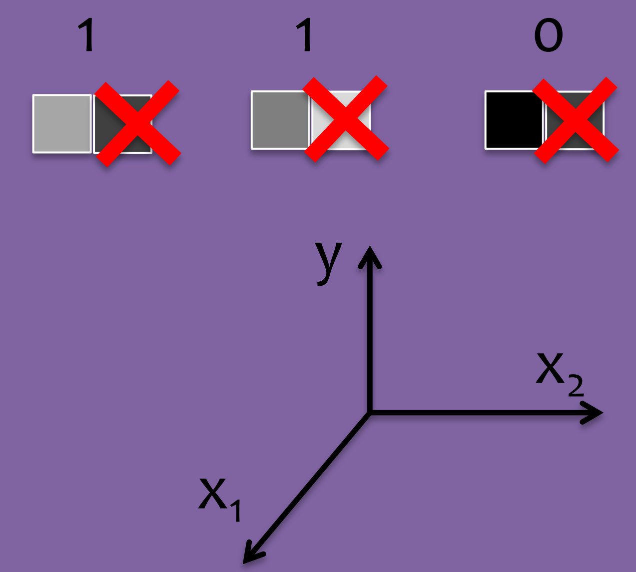


Logistic Regression

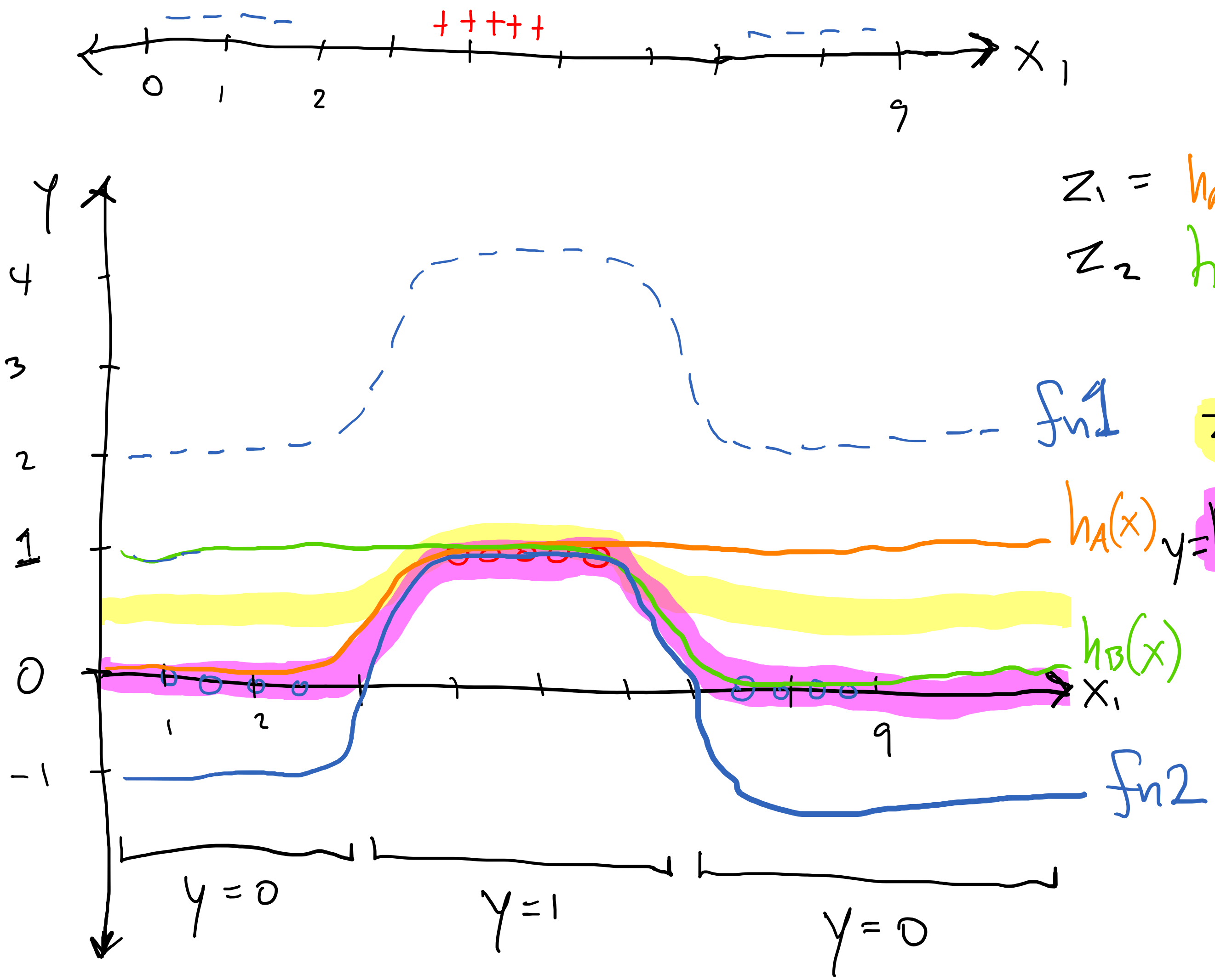
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$



In-Class Example



1D Face Recognition



$$z_1 = h_A(x) = \sigma(\theta_A x_1 + \theta_{A0})$$

$$z_2 \quad h_B(x) = \sigma(\theta_B x_1 + \theta_{B0})$$

$$z_C(x) = \frac{h_A(x) + h_B(x)}{2} \quad \text{fn1}$$

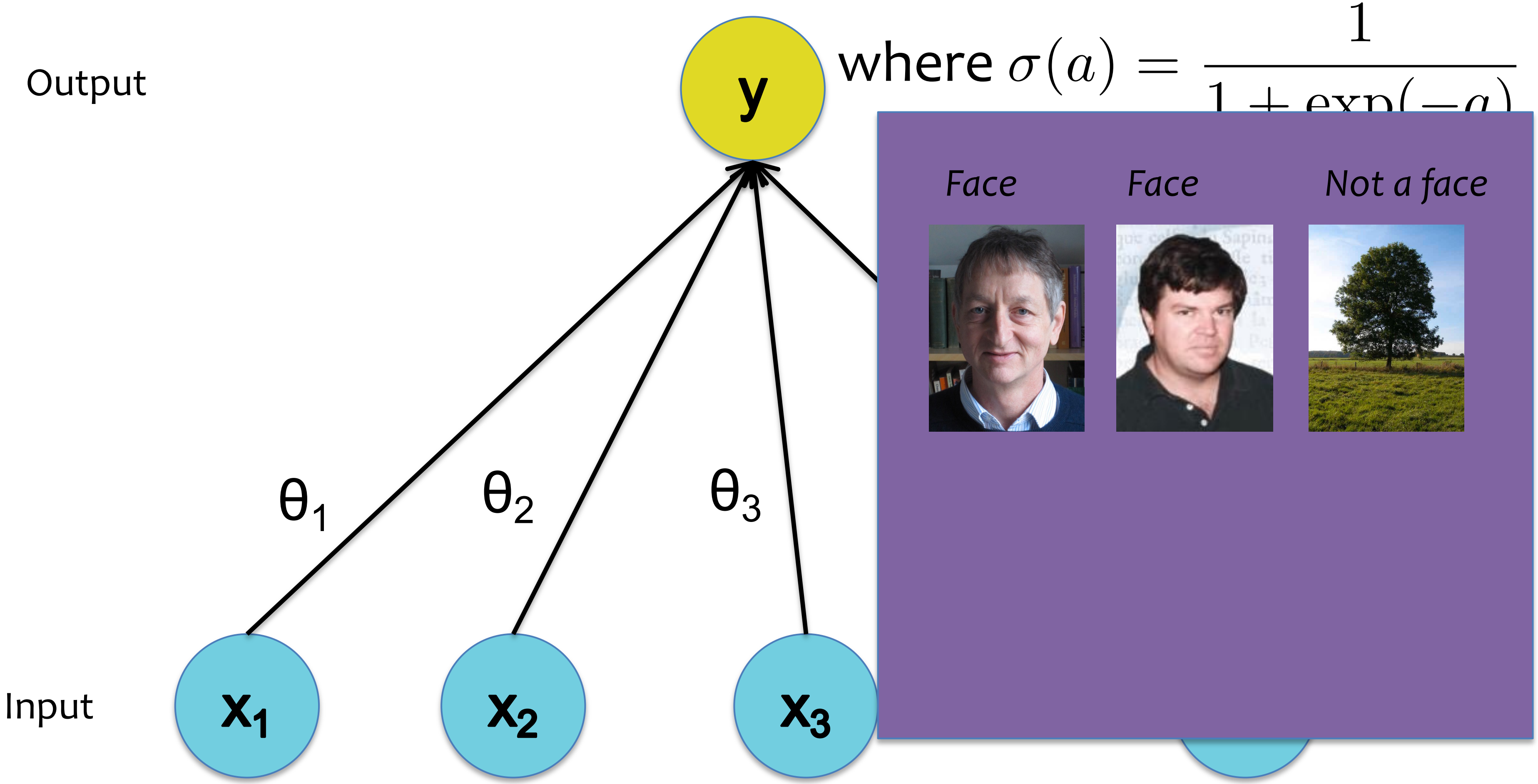
$$y = h_C(x) = \sigma(w_A h_A(x) + w_B h_B(x) + w_C) \quad \text{fn2}$$

where $w_A = 2$ $w_B = 2$ $w_C = -3$

Logistic Regression

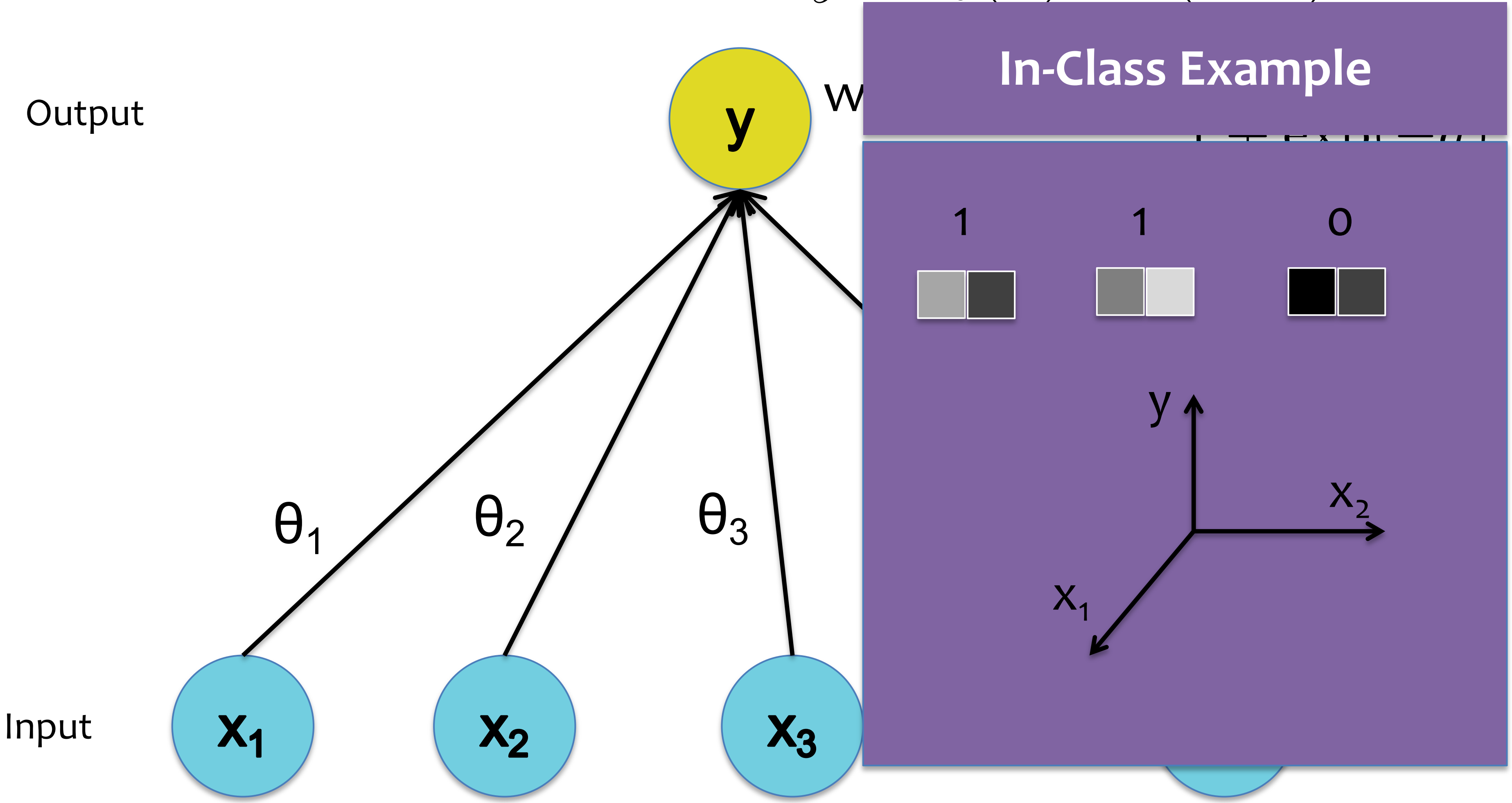
$$y = h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$$

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



Logistic Regression

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

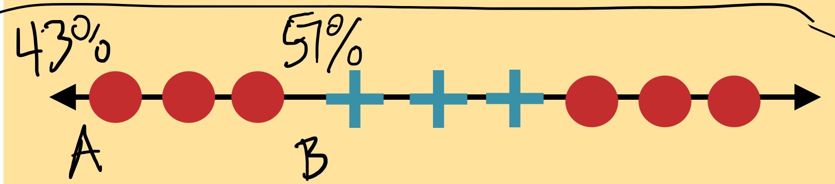


Neural Network Parameters

Q1

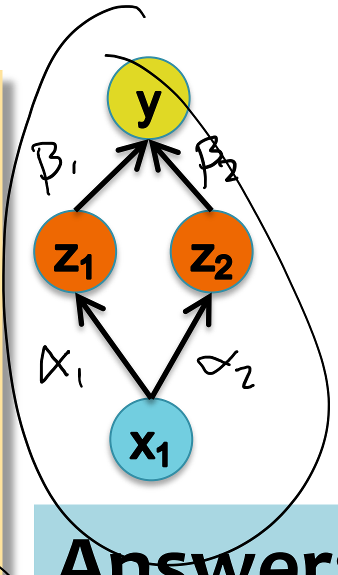
Question:

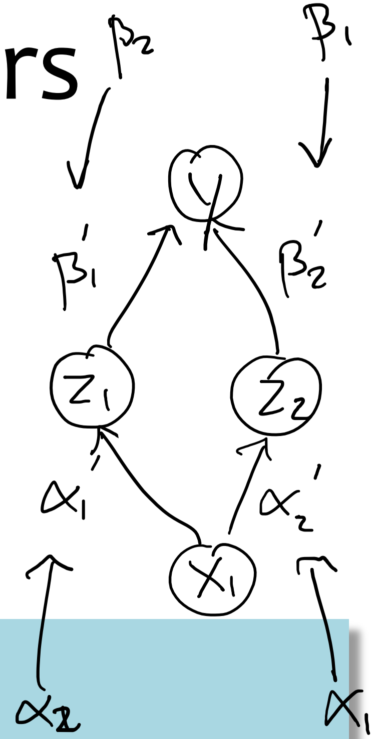
Suppose you are training a one-hidden layer neural network with sigmoid activations for binary classification.



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

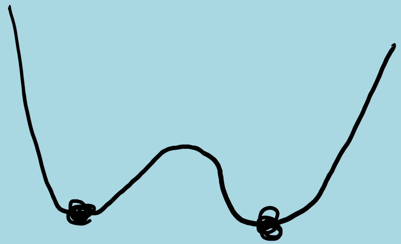
C = toxic





Answer:

⇒ objective function is nonconvex



Neural Network Architectures

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

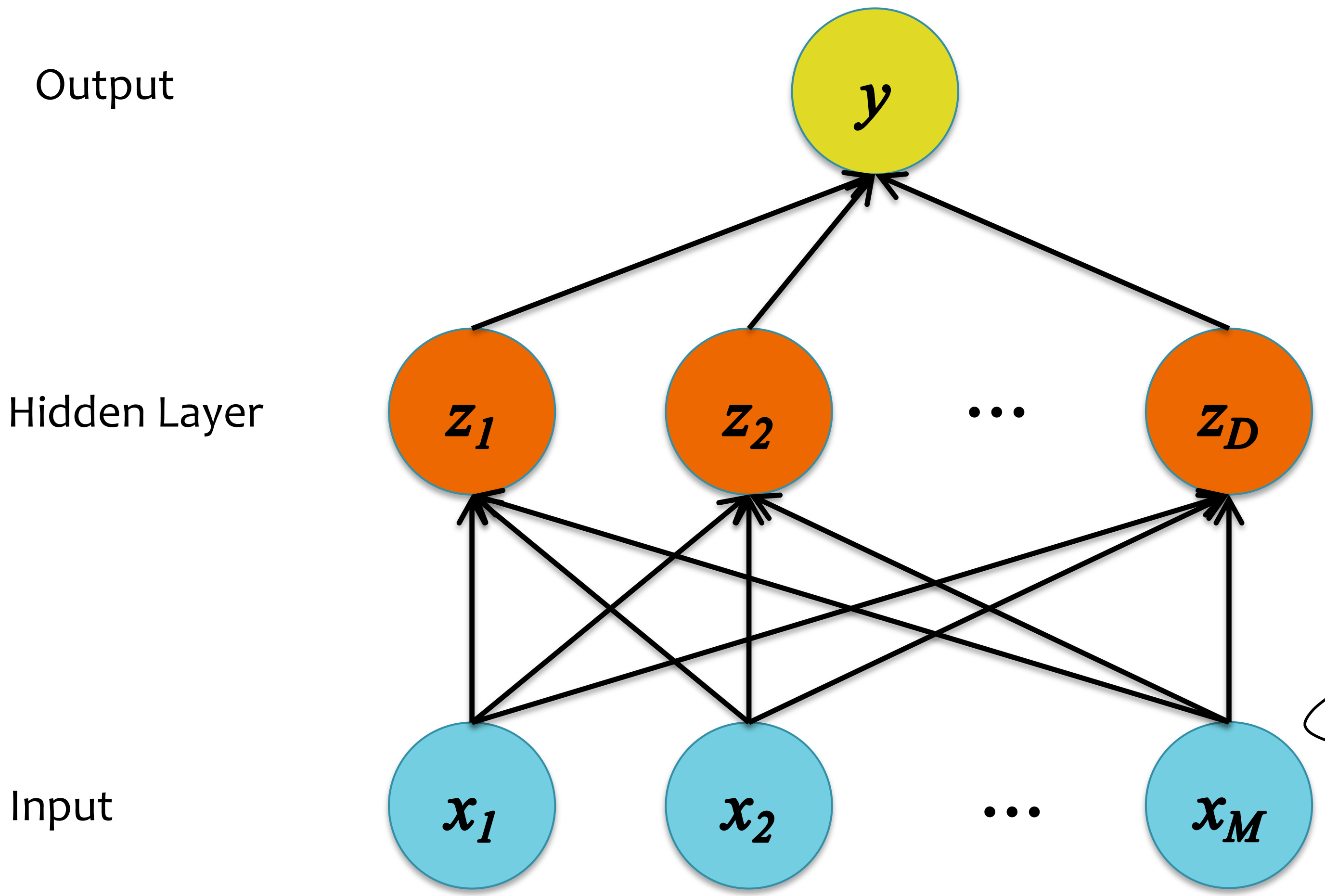
1. # of hidden layers (depth)
2. # of units per hidden layer (width)
3. Type of activation function (nonlinearity)
4. Form of objective function
5. How to initialize the parameters

BUILDING WIDER NETWORKS

D = M

Building a Neural Net

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

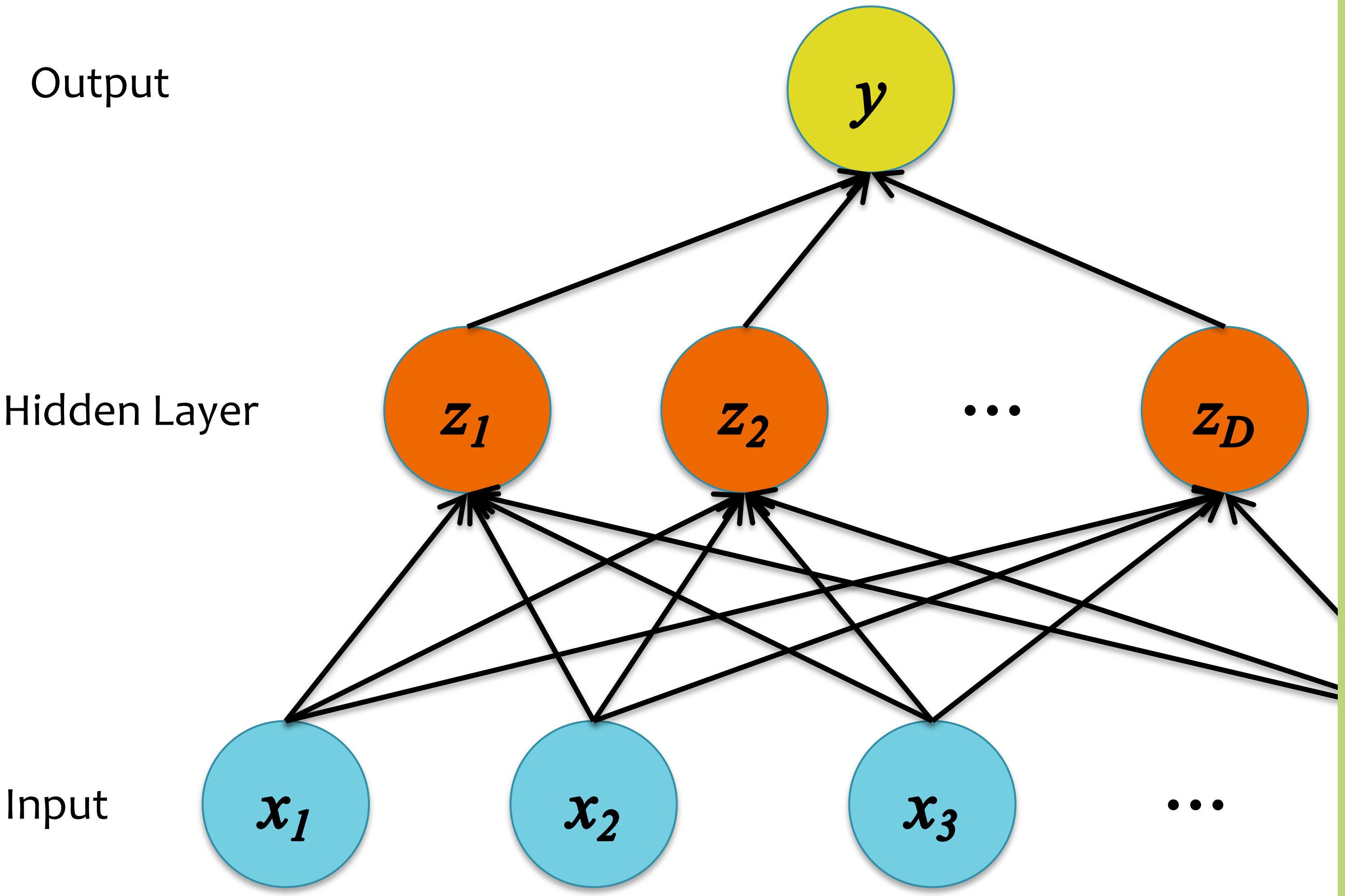


The hidden units could learn to be…

- 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?

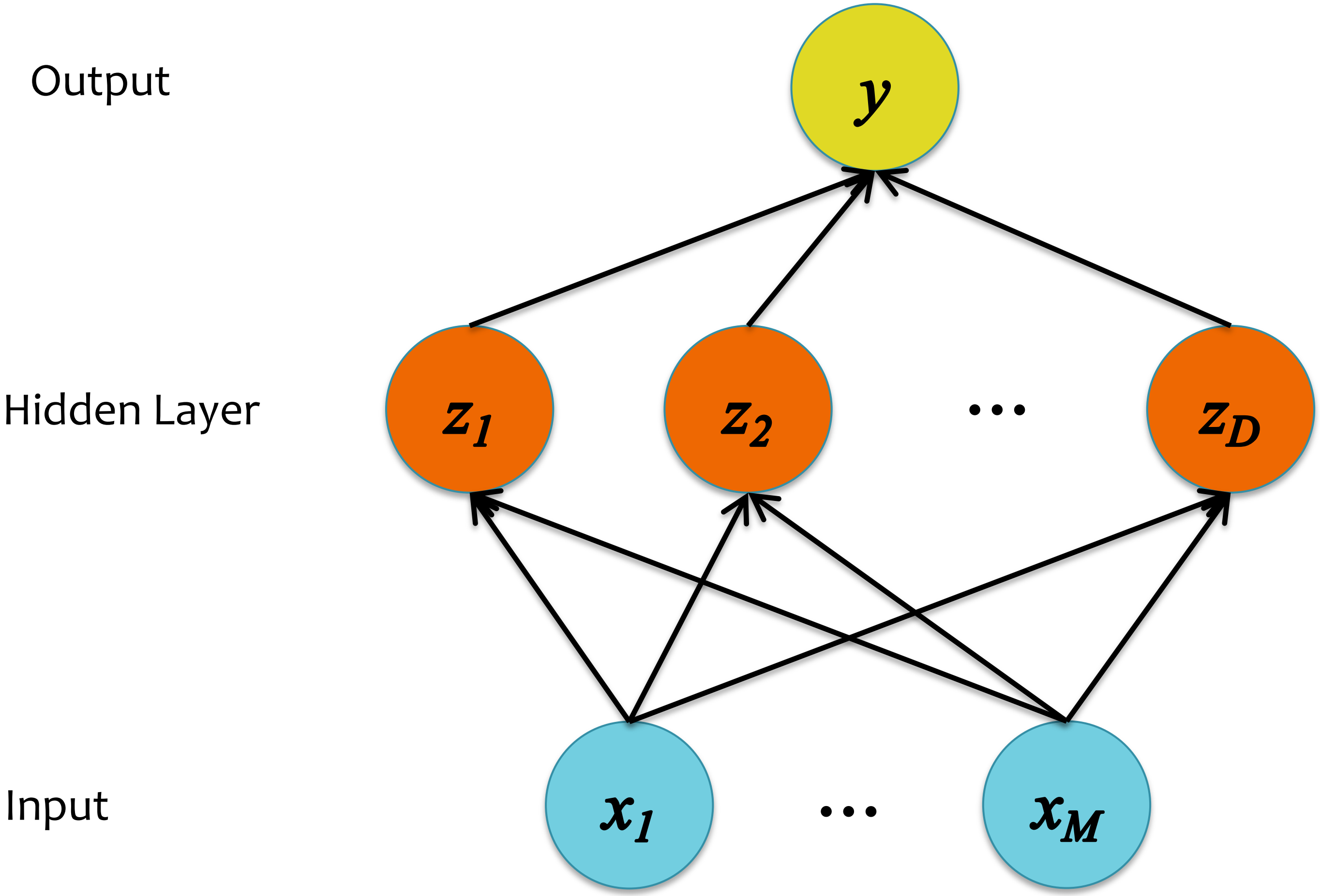


The hidden units could learn to be…

- 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

Building a Neural Net

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



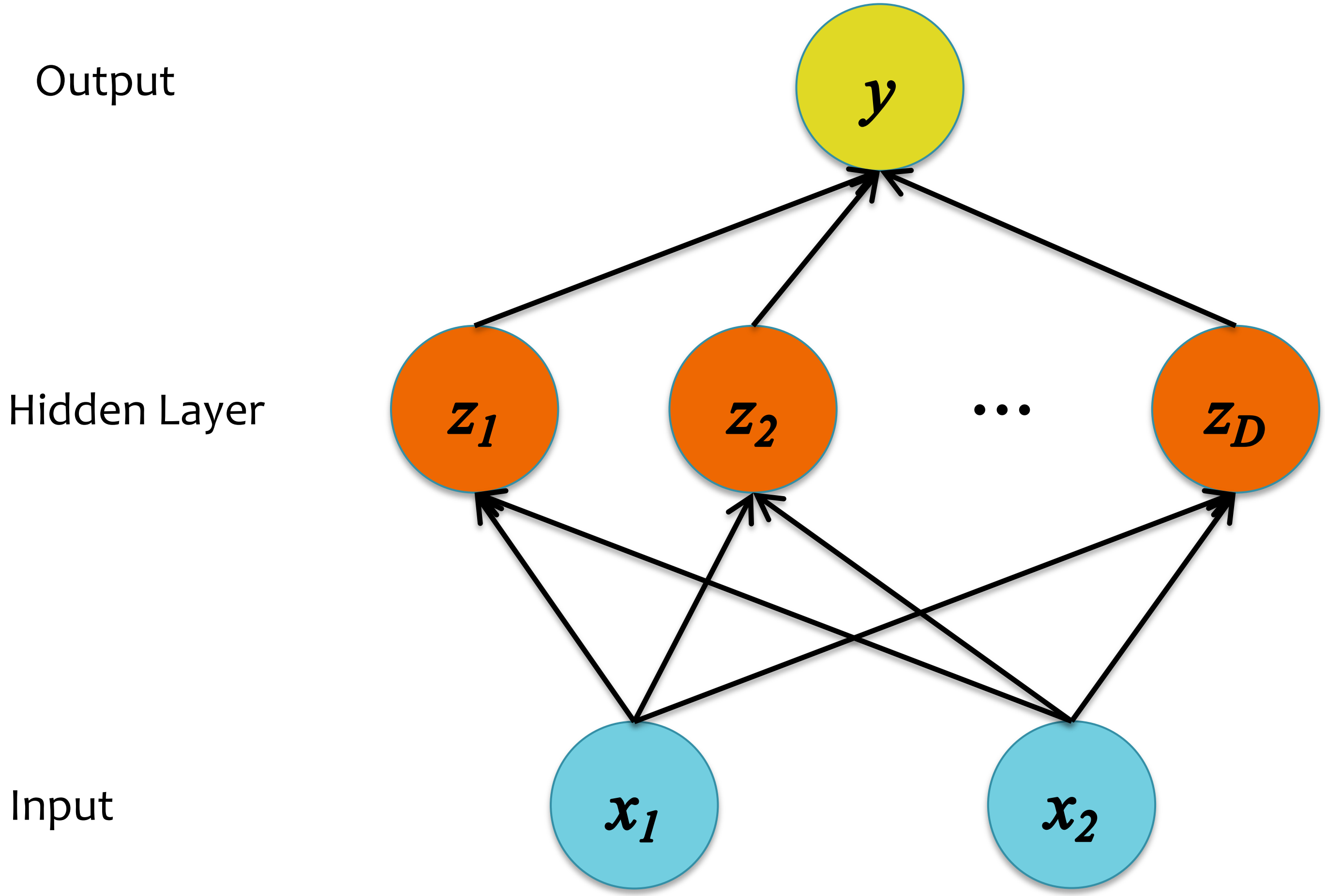
The hidden units could learn to be…

- 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.



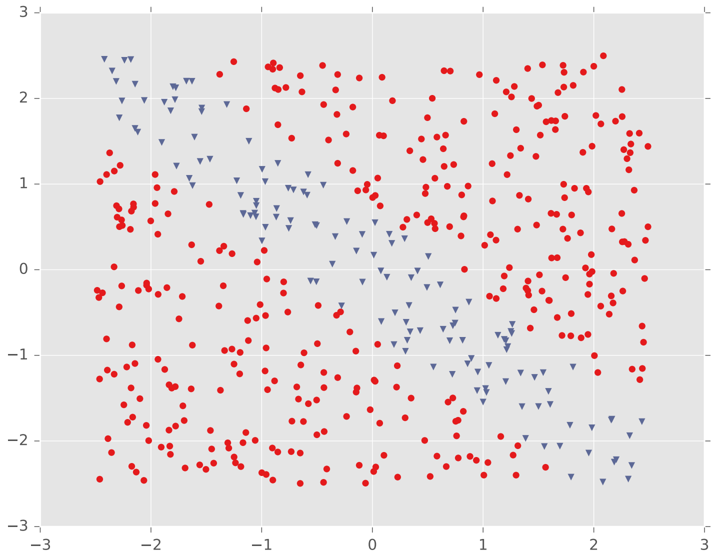
The hidden units could learn to be…

- 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

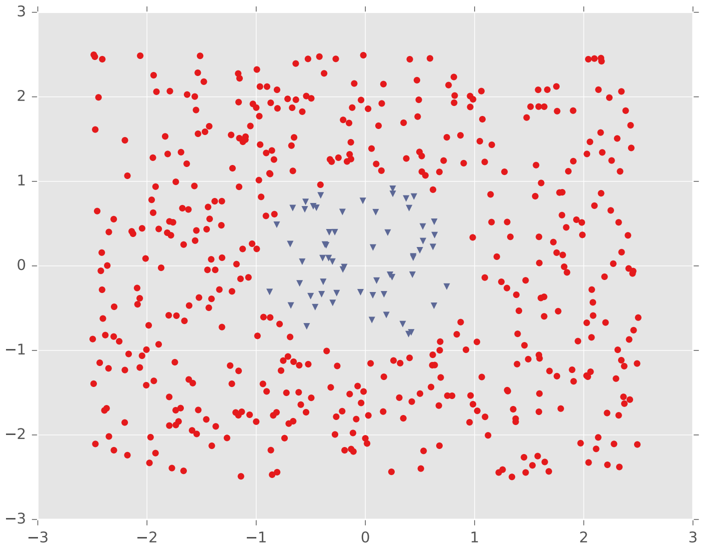
Examples 1 and 2

DECISION BOUNDARY EXAMPLES

Example #1: Diagonal Band

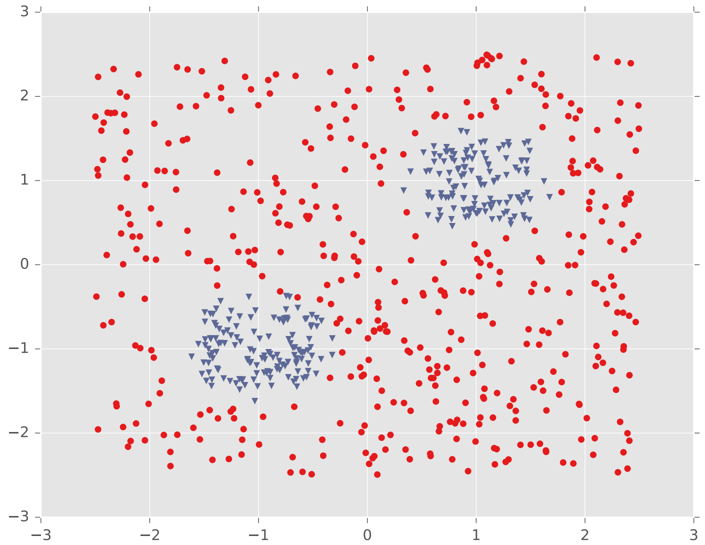


Example #2: One Pocket

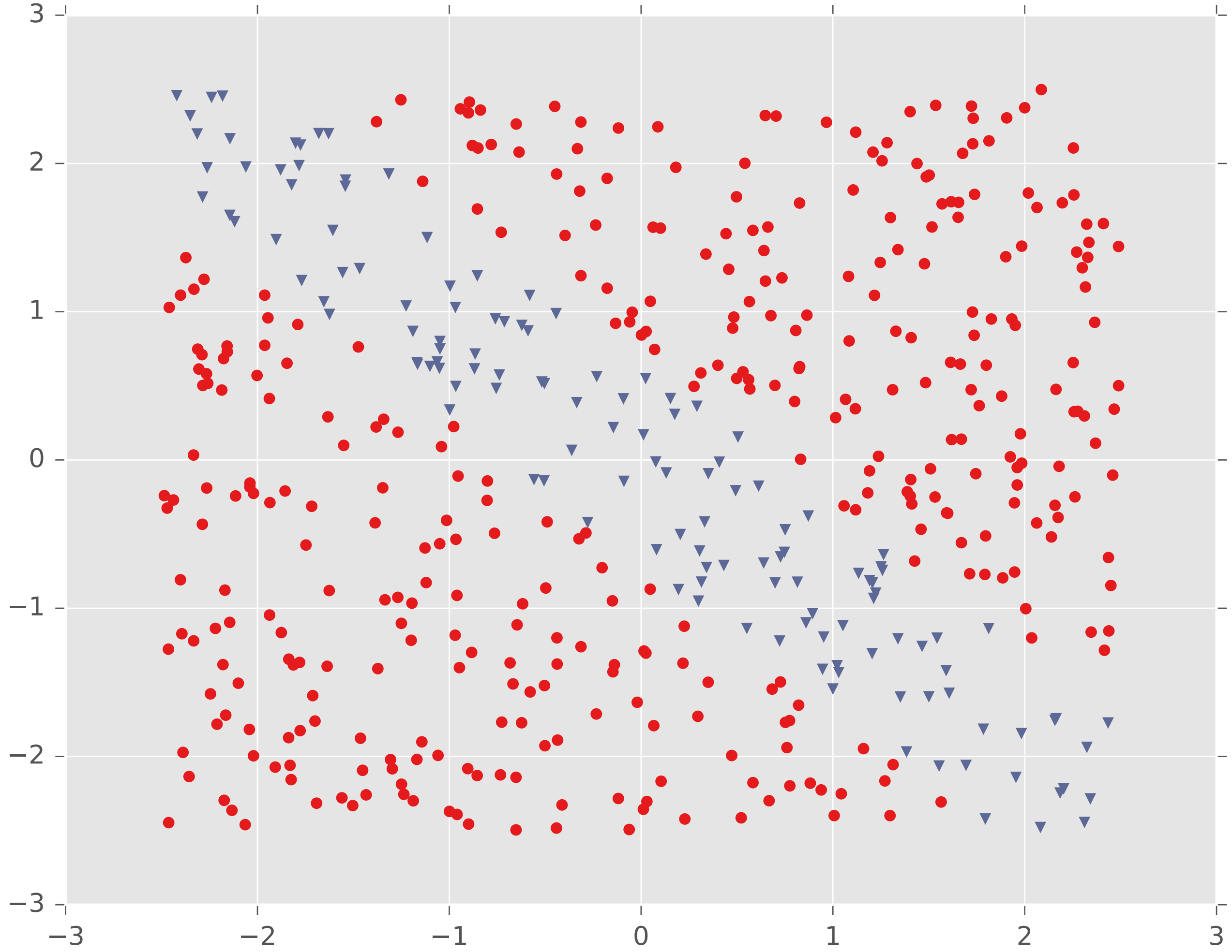


Example #3: Four Gaussians

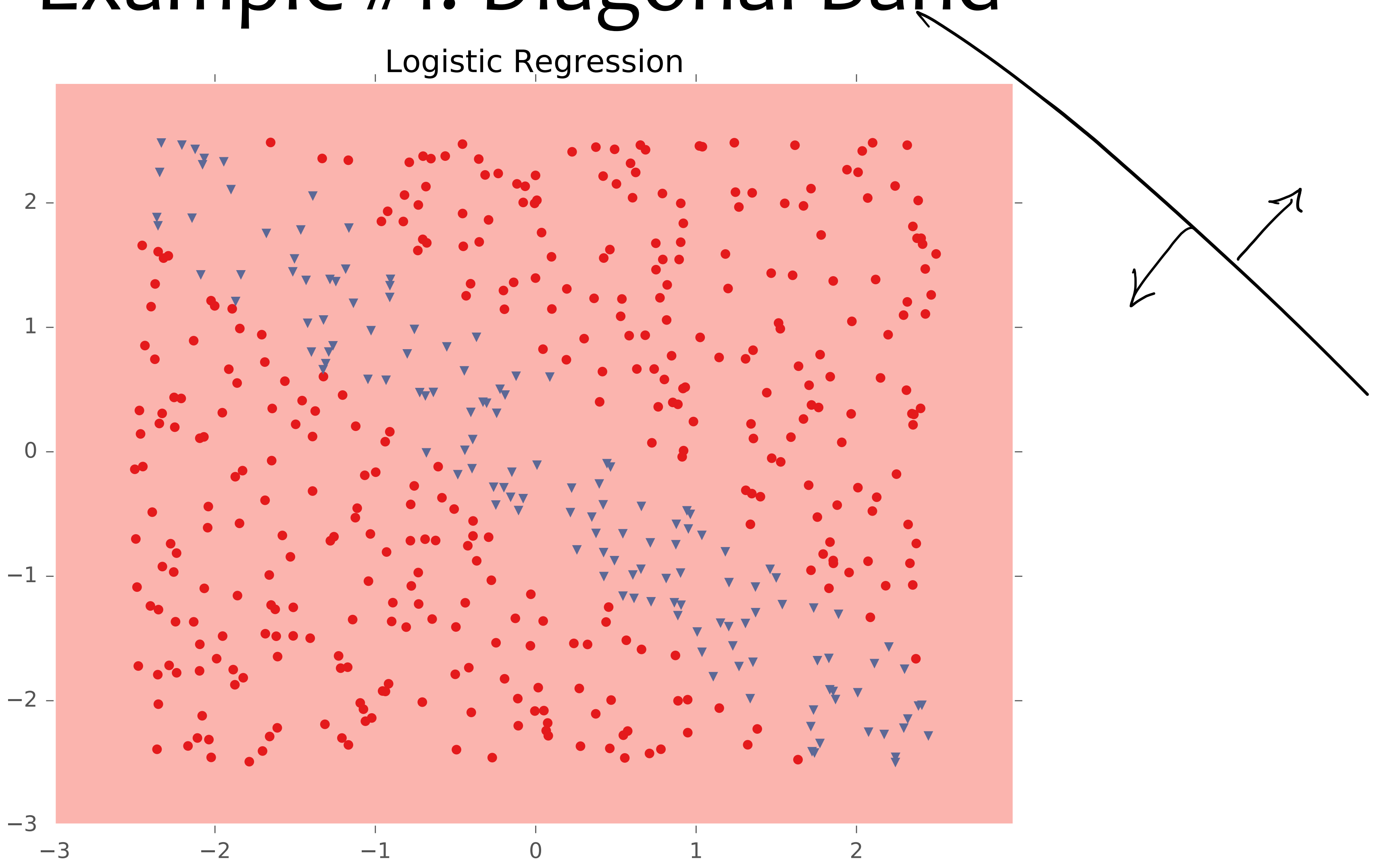
Example #4: Two Pockets



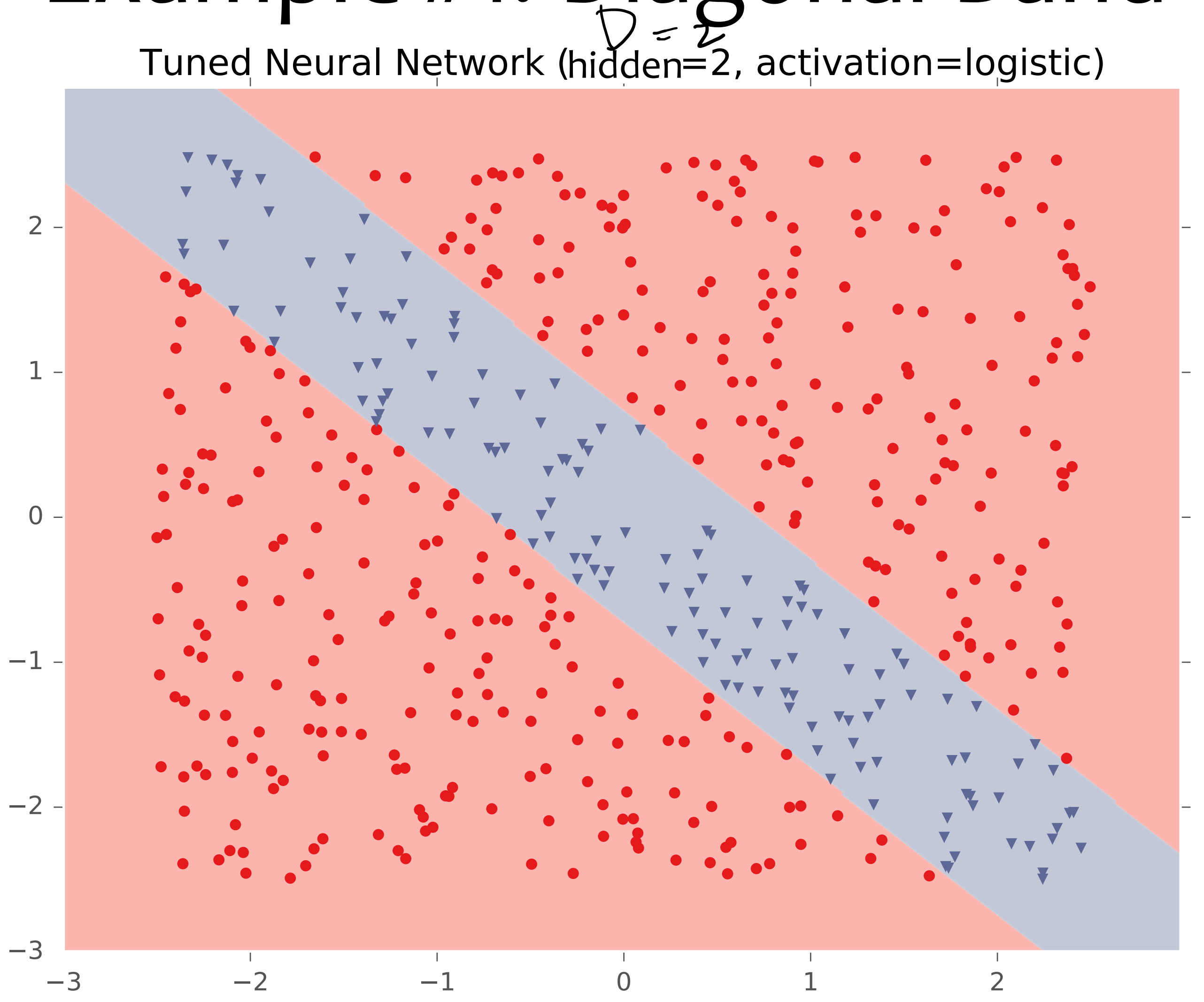
Example #1: Diagonal Band



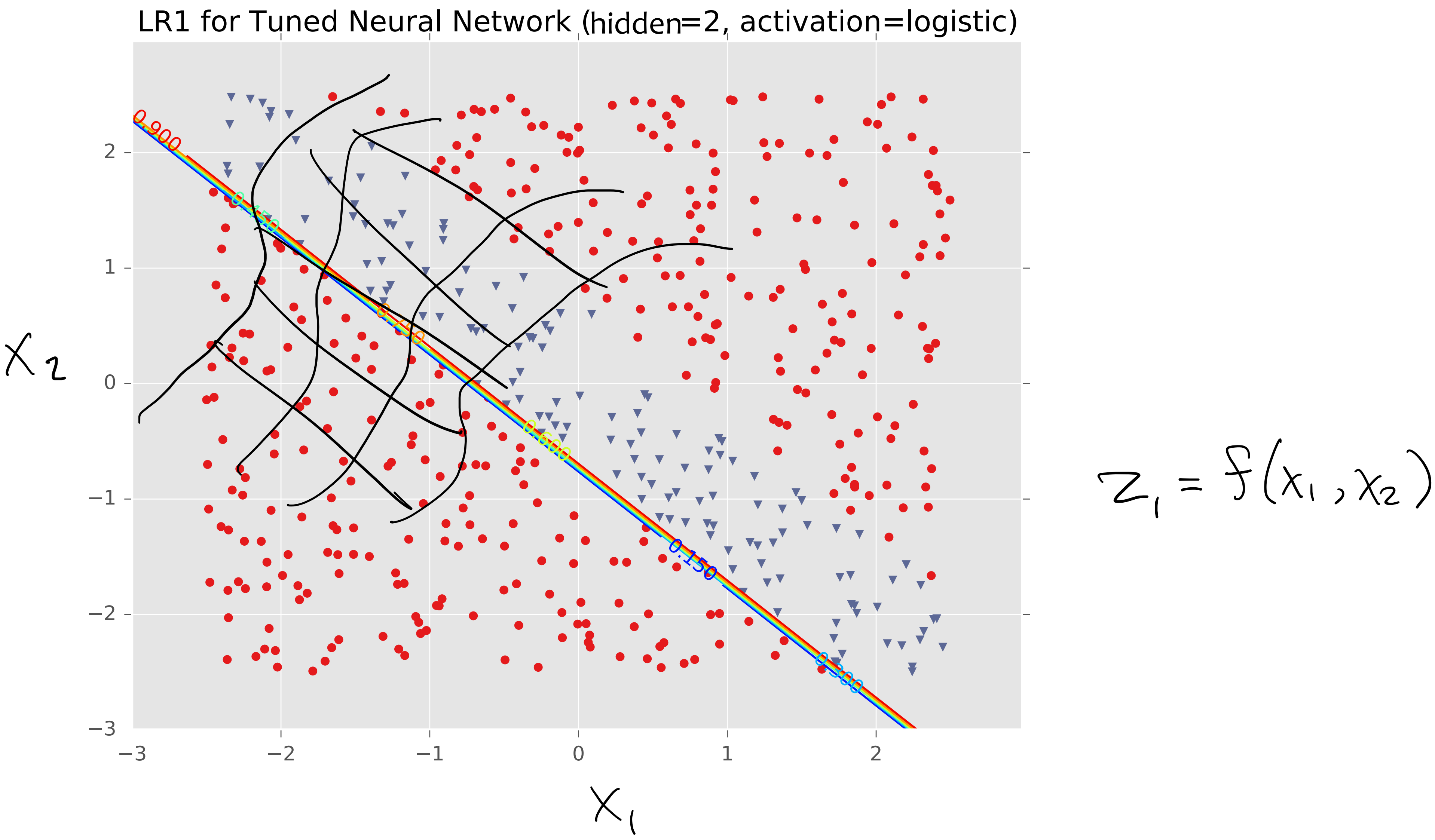
Example #1: Diagonal Band



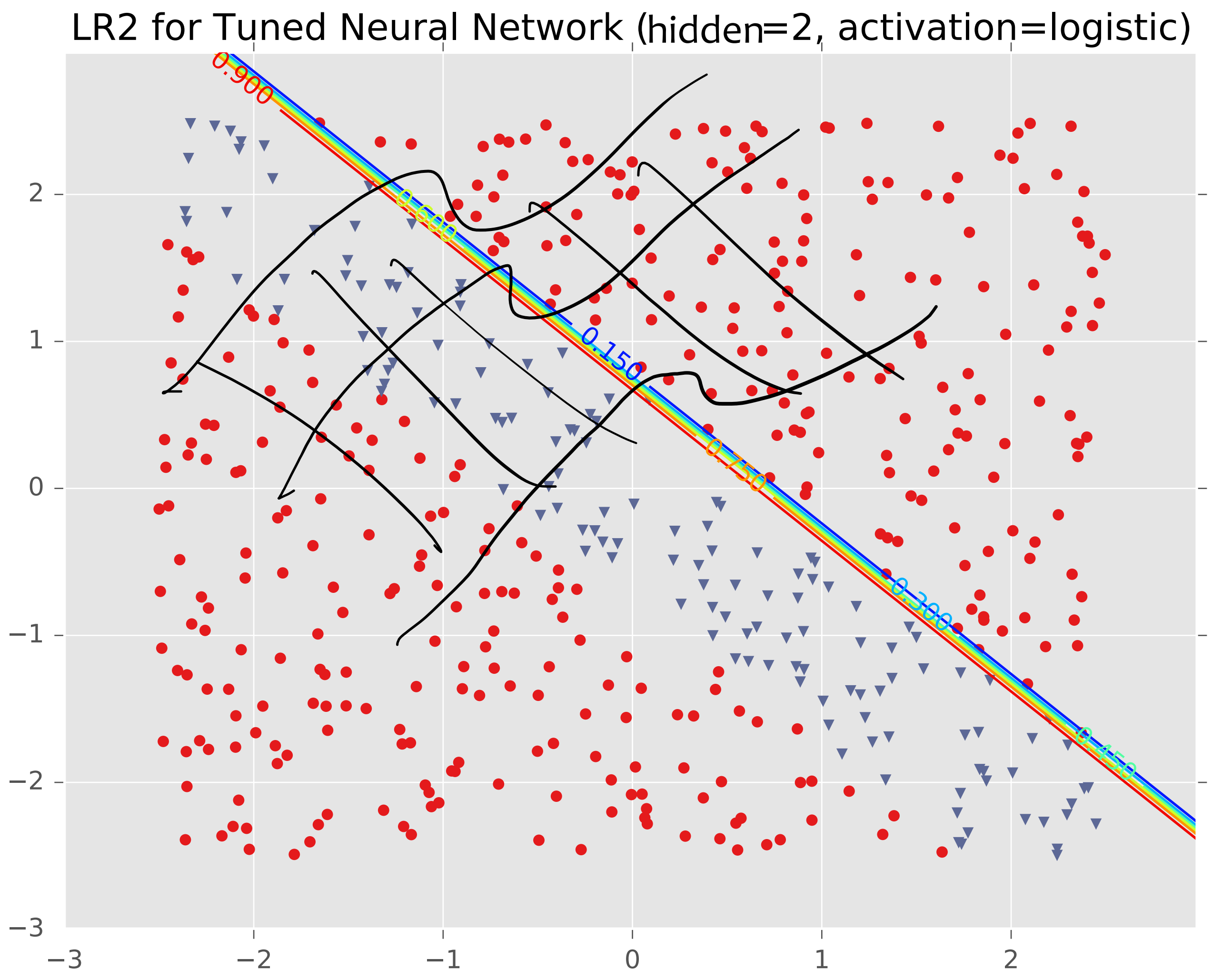
Example #1: Diagonal Band



Example #1: Diagonal Band

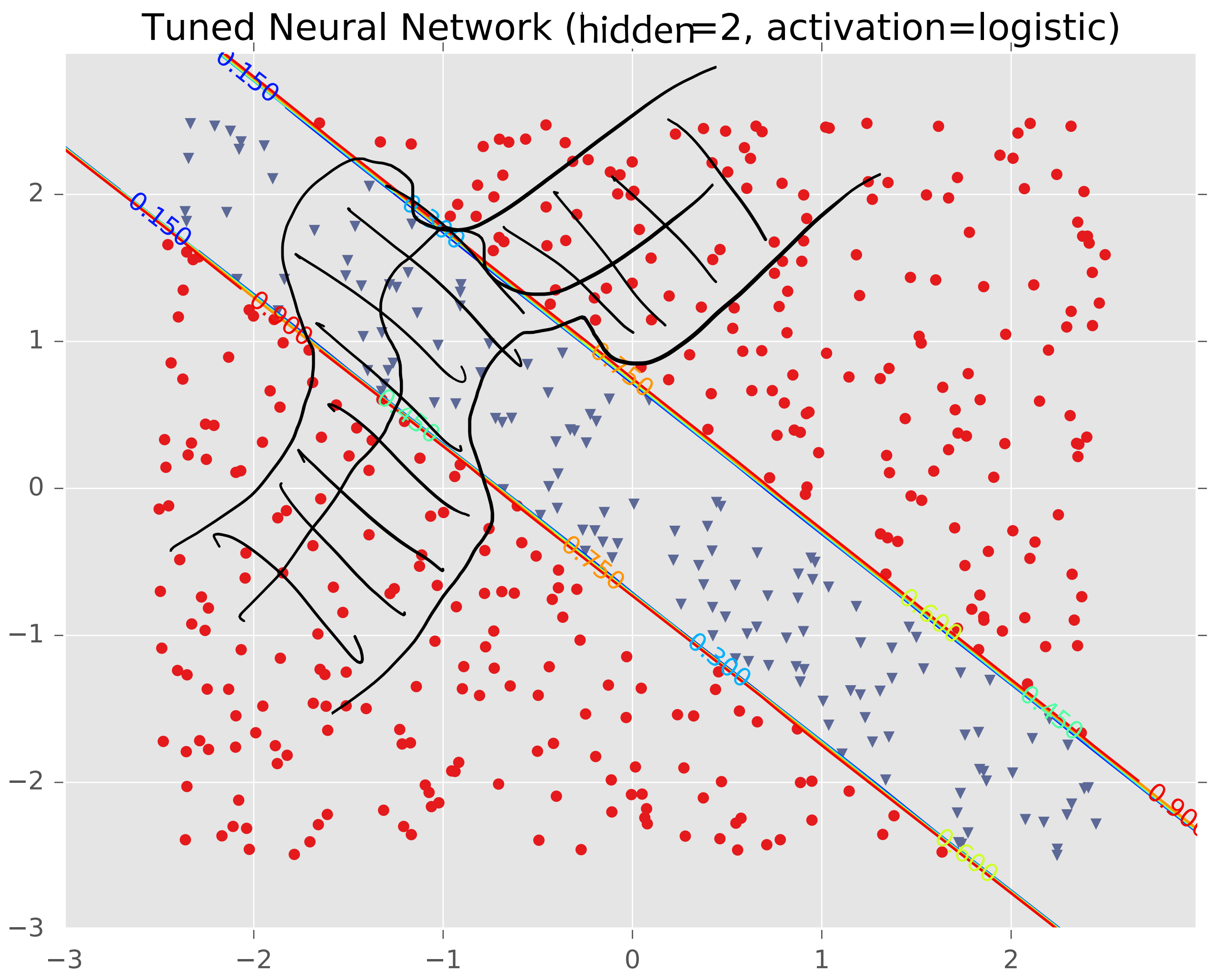


Example #1: Diagonal Band



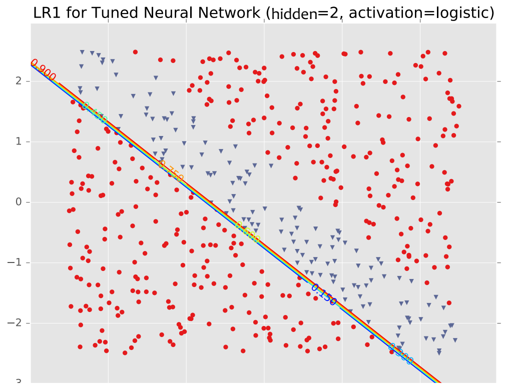
$$z_2 = g(x_1, x_2)$$

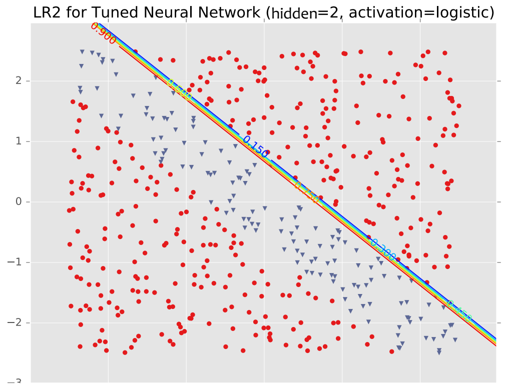
Example #1: Diagonal Band

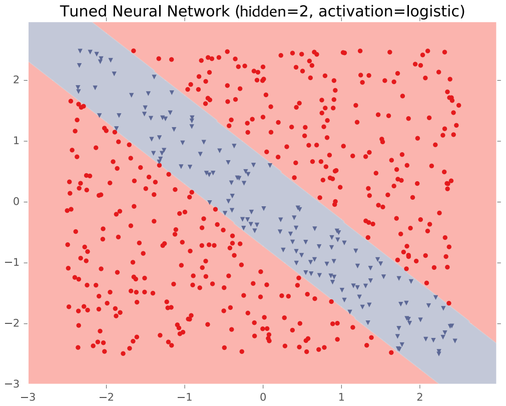


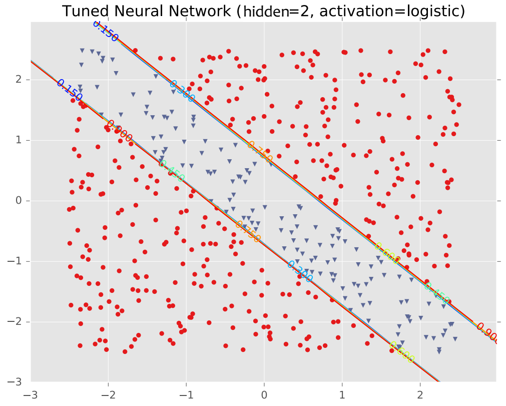
$y = h(x_1, x_2)$

Example #1: Diagonal Band

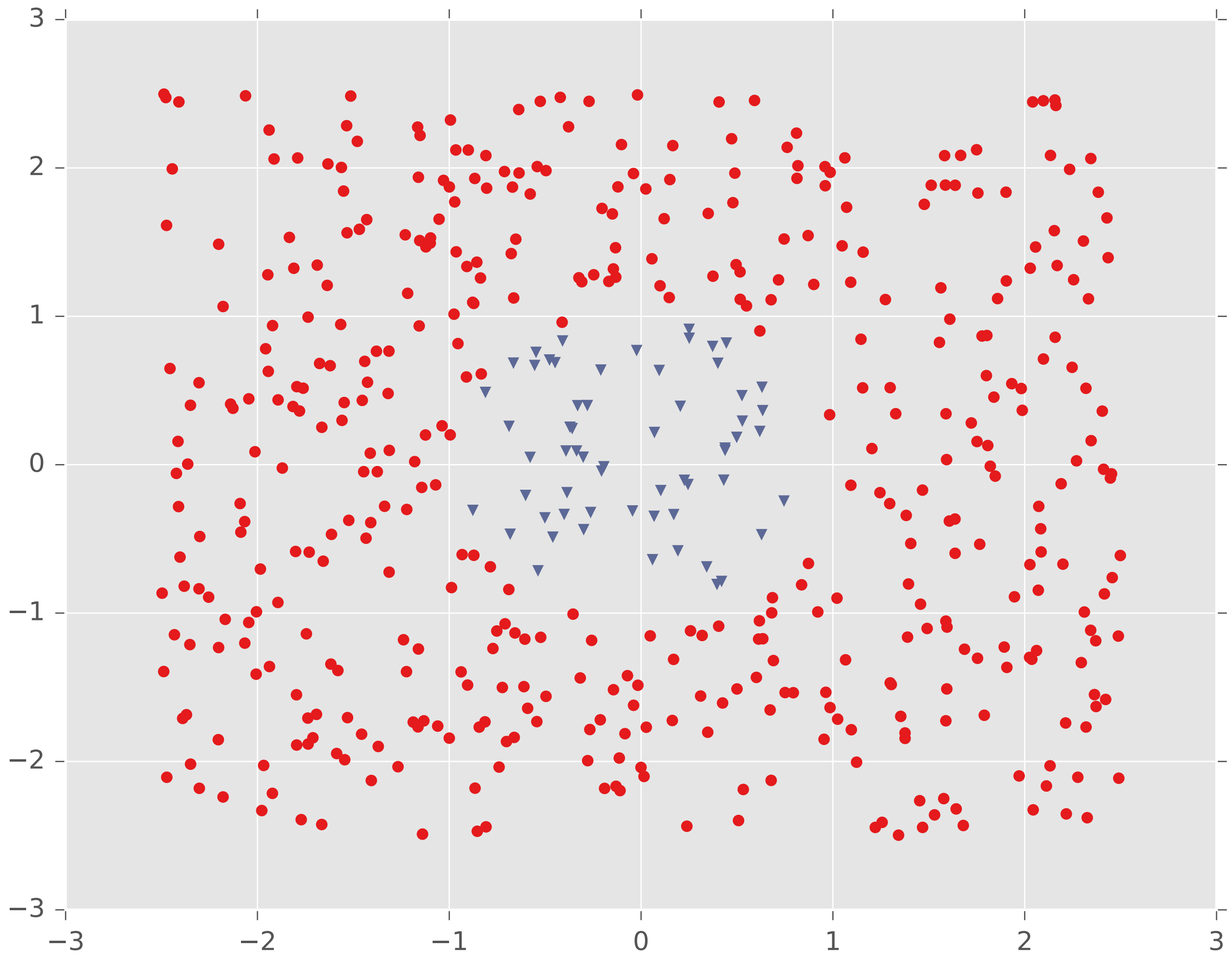




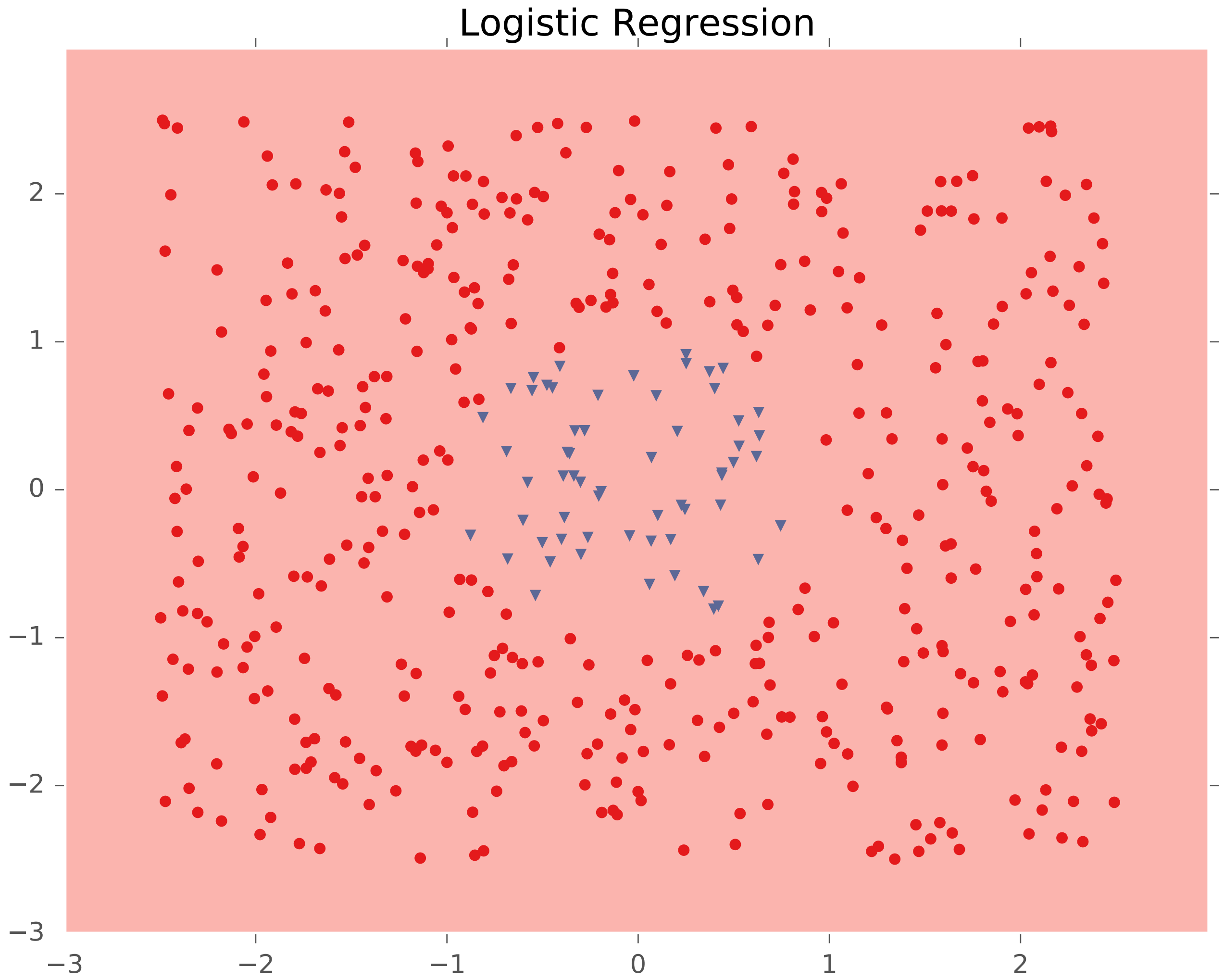




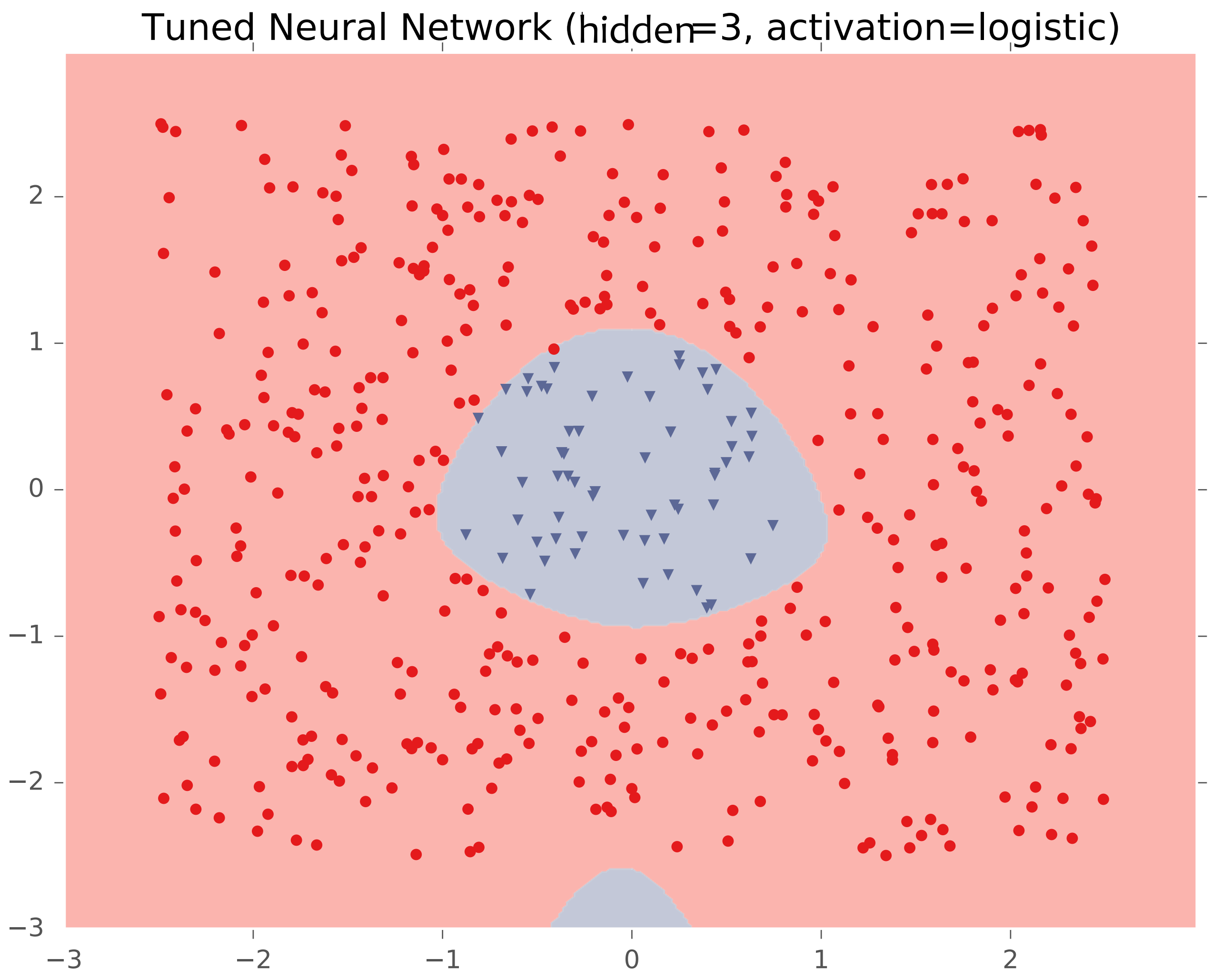
Example #2: One Pocket



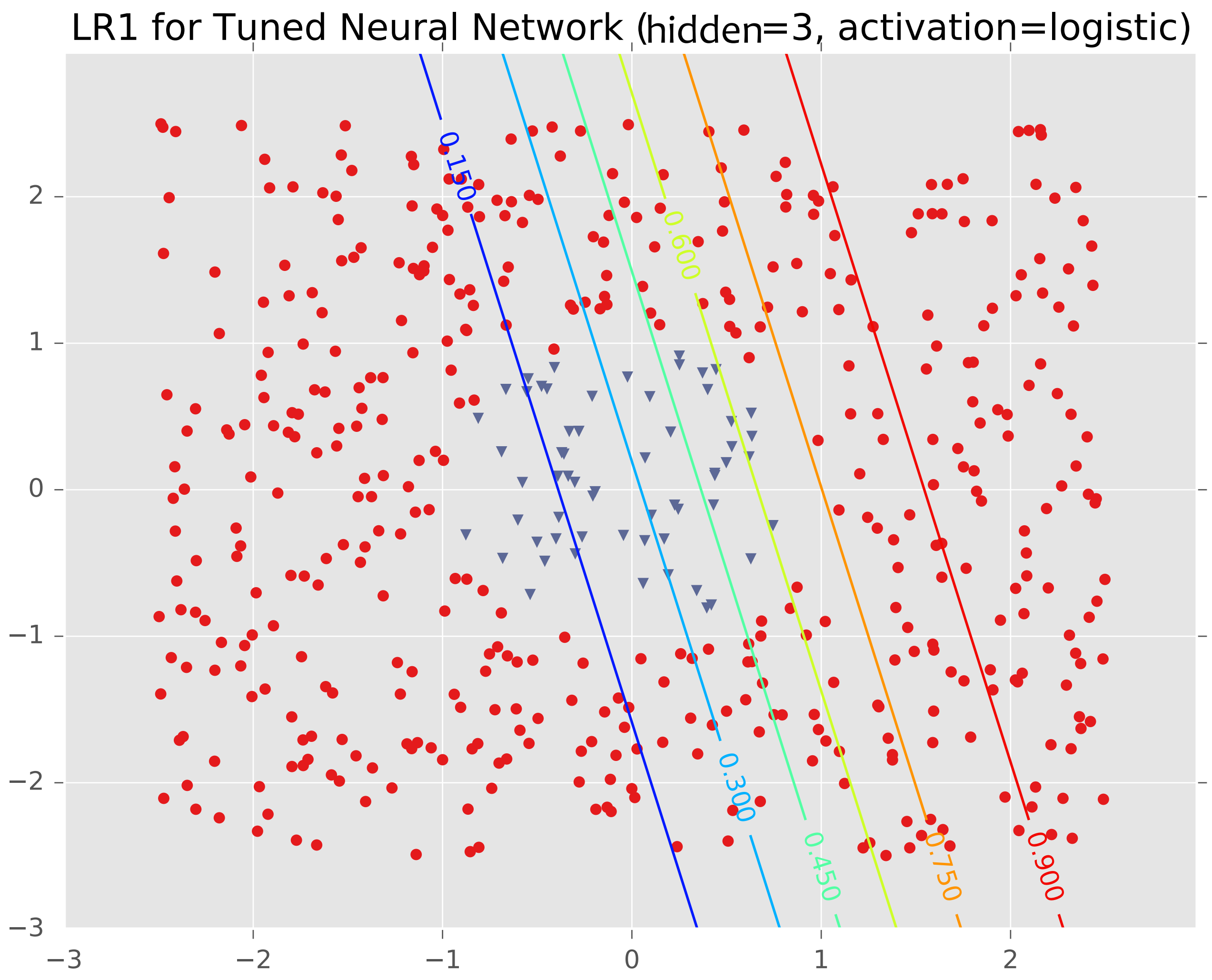
Example #2: One Pocket



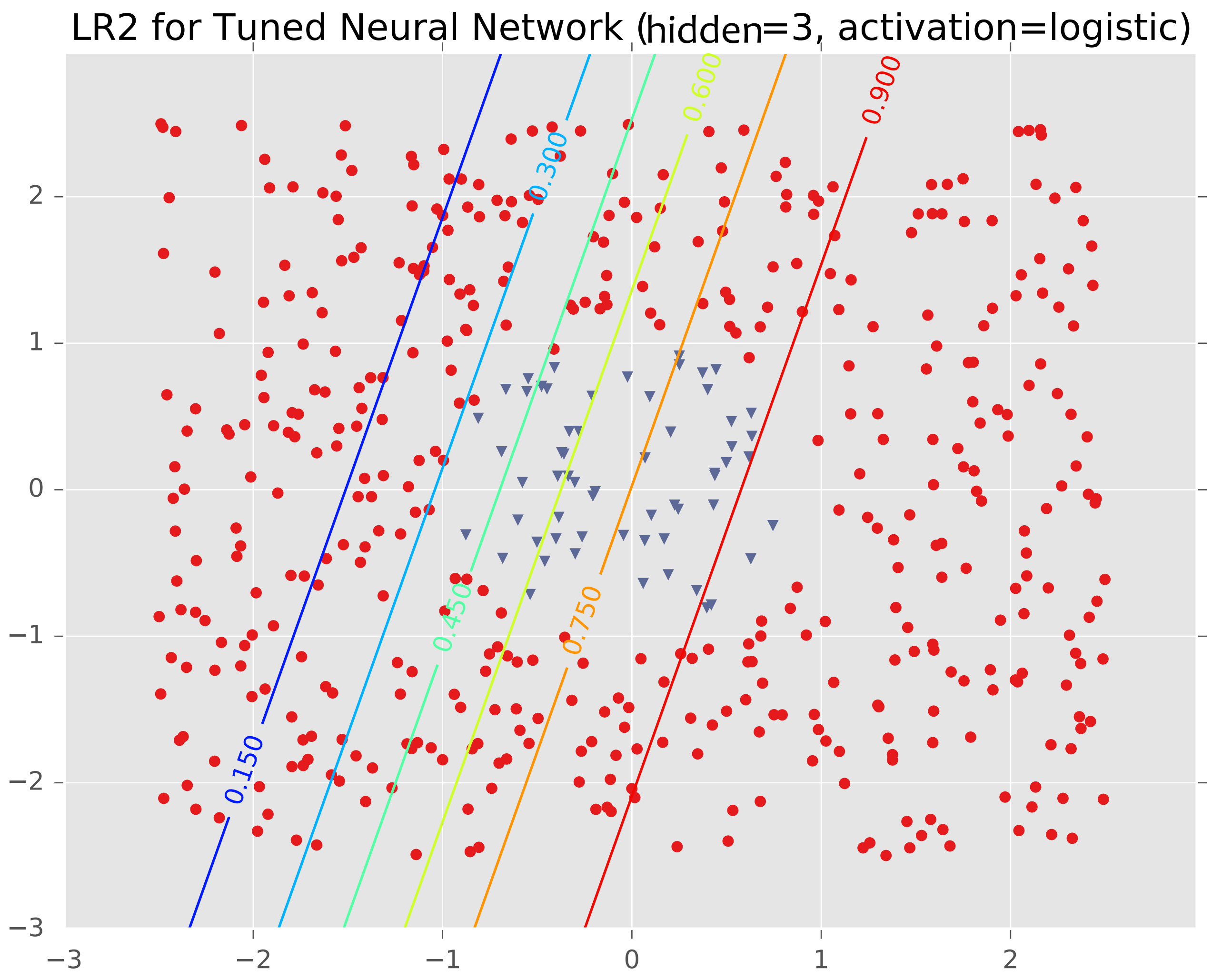
Example #2: One Pocket



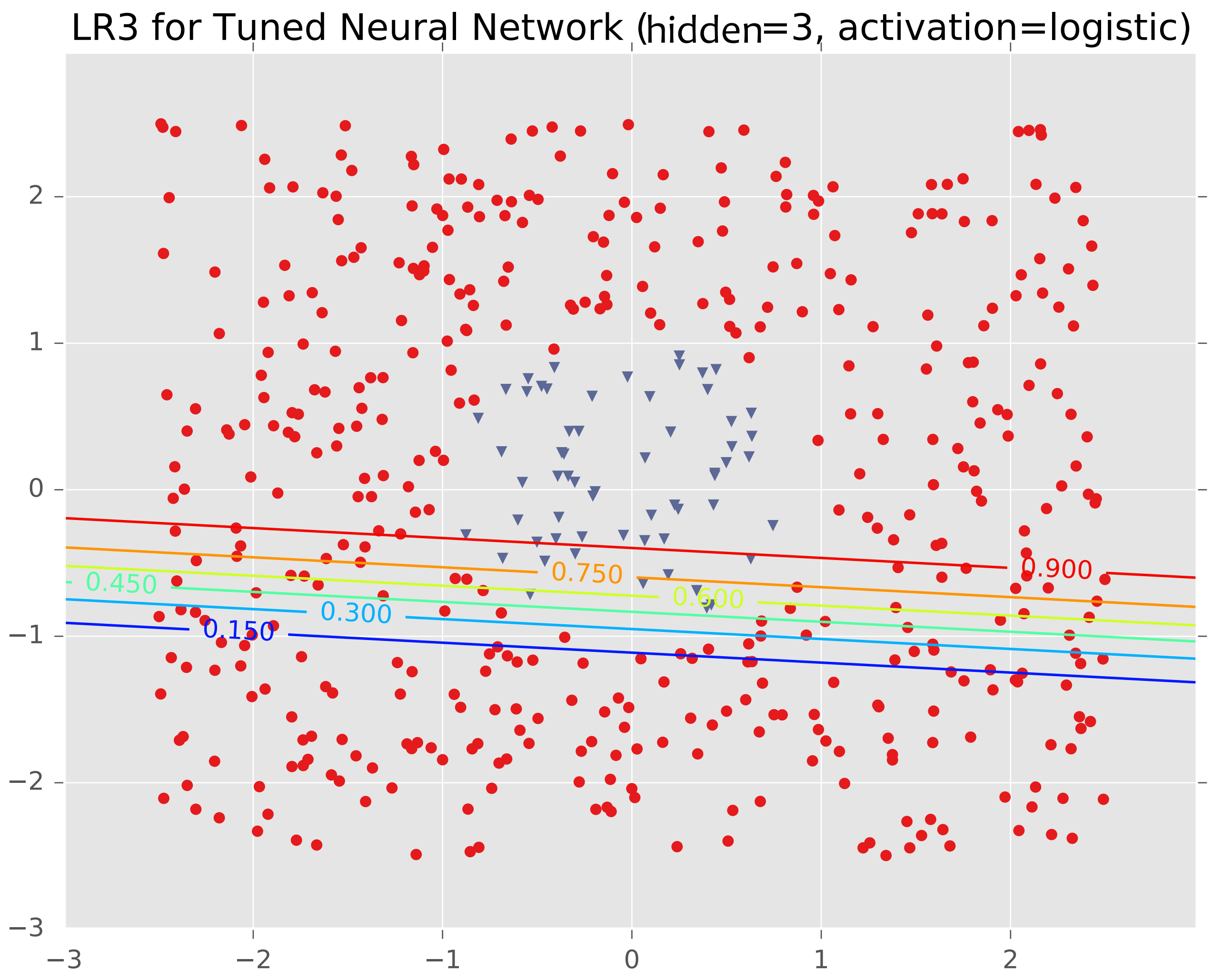
Example #2: One Pocket



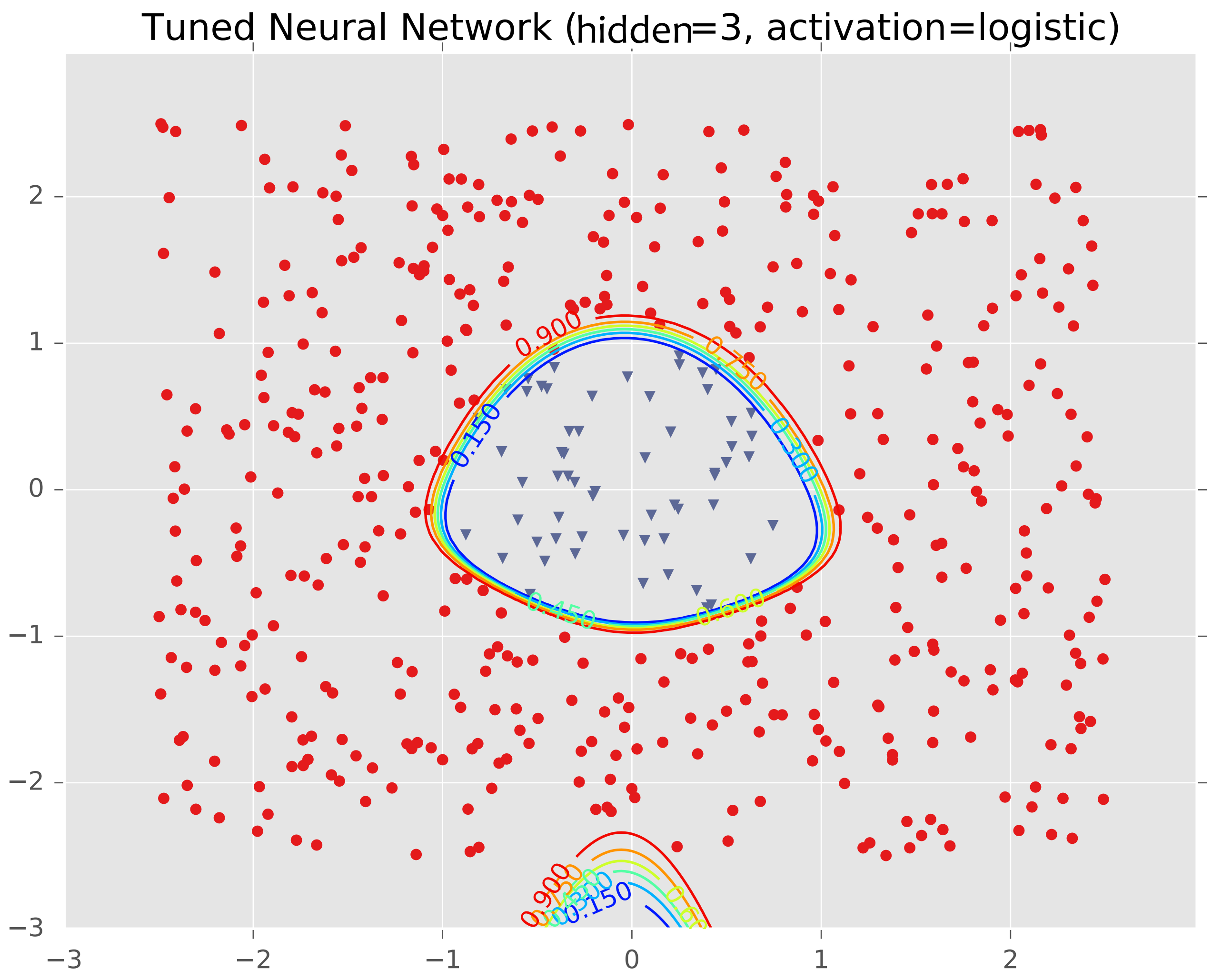
Example #2: One Pocket



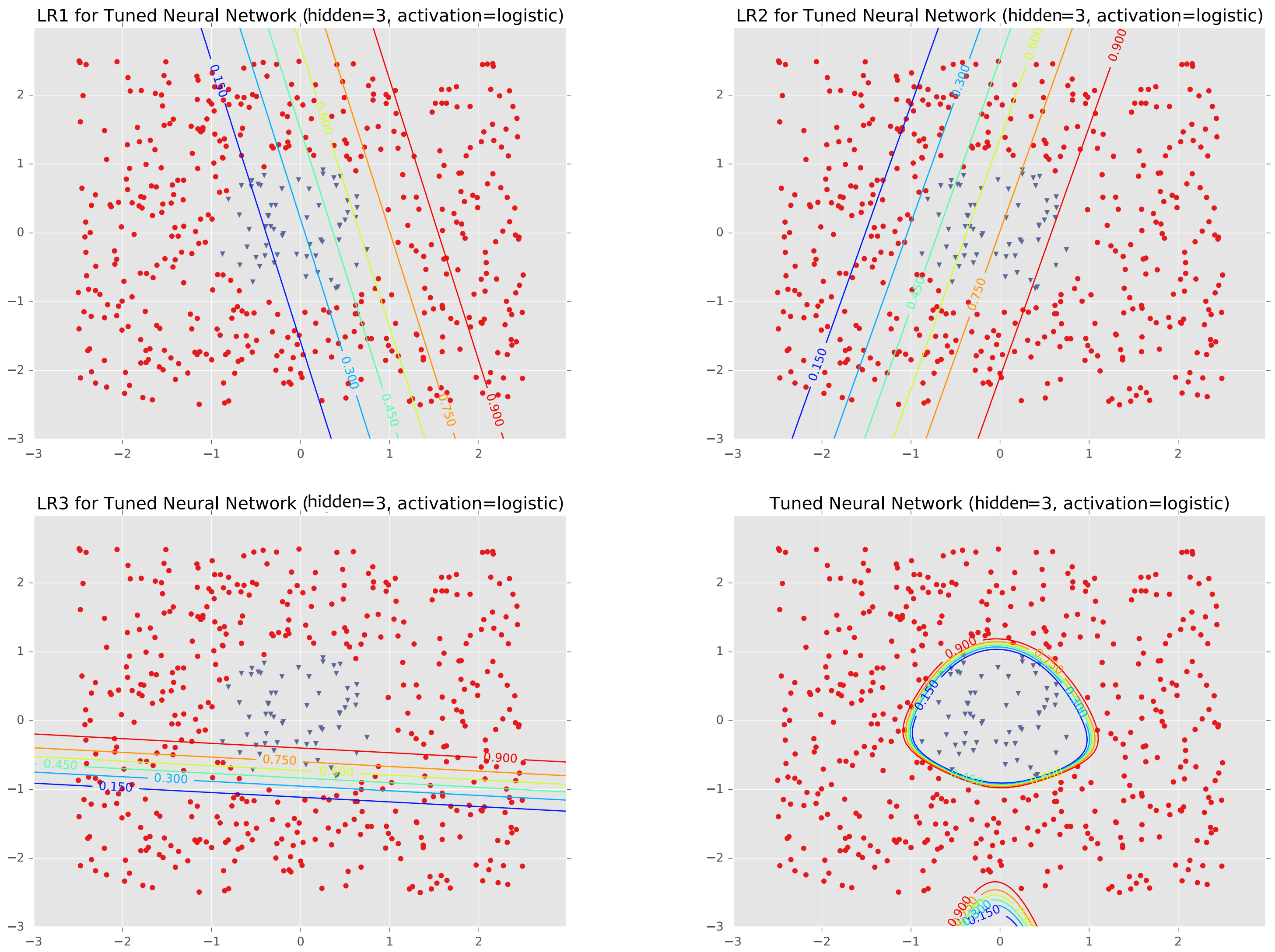
Example #2: One Pocket

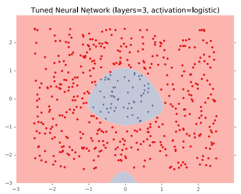


Example #2: One Pocket



Example #2: One Pocket

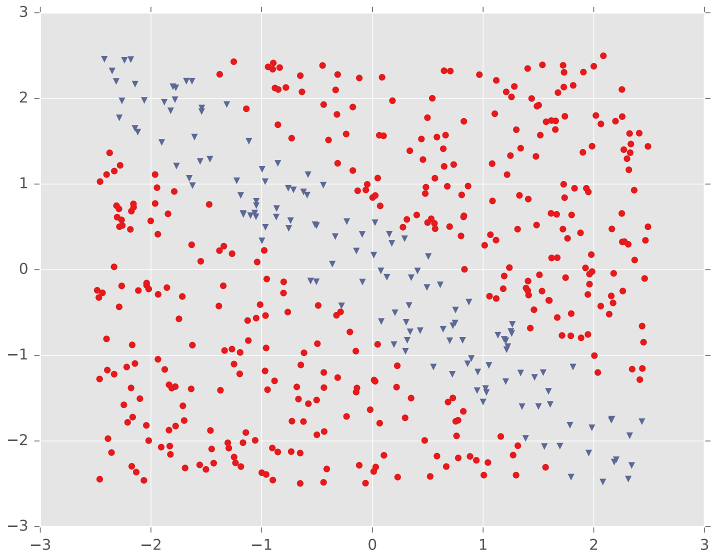




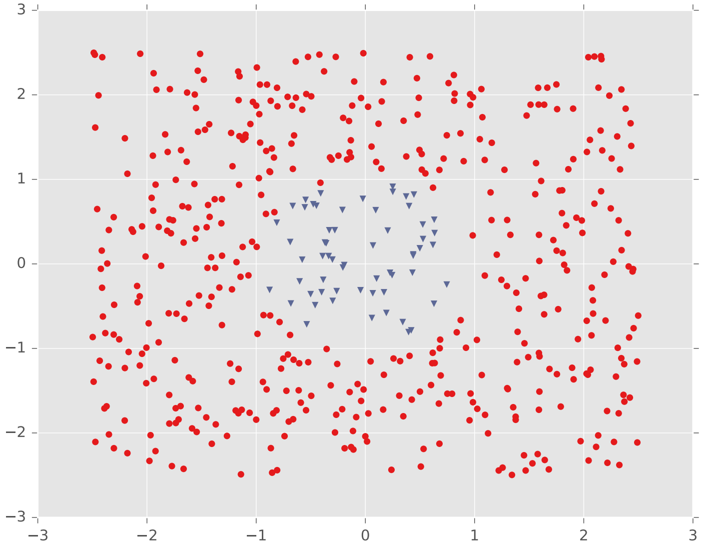
Examples 3 and 4

DECISION BOUNDARY EXAMPLES

Example #1: Diagonal Band

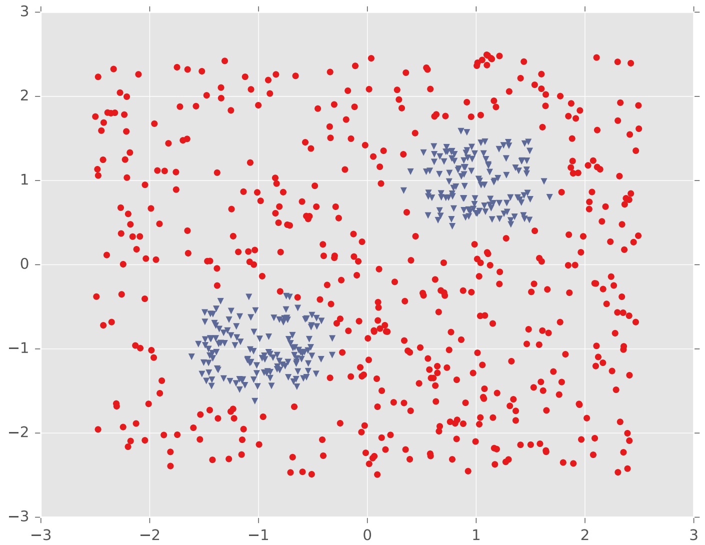


Example #2: One Pocket

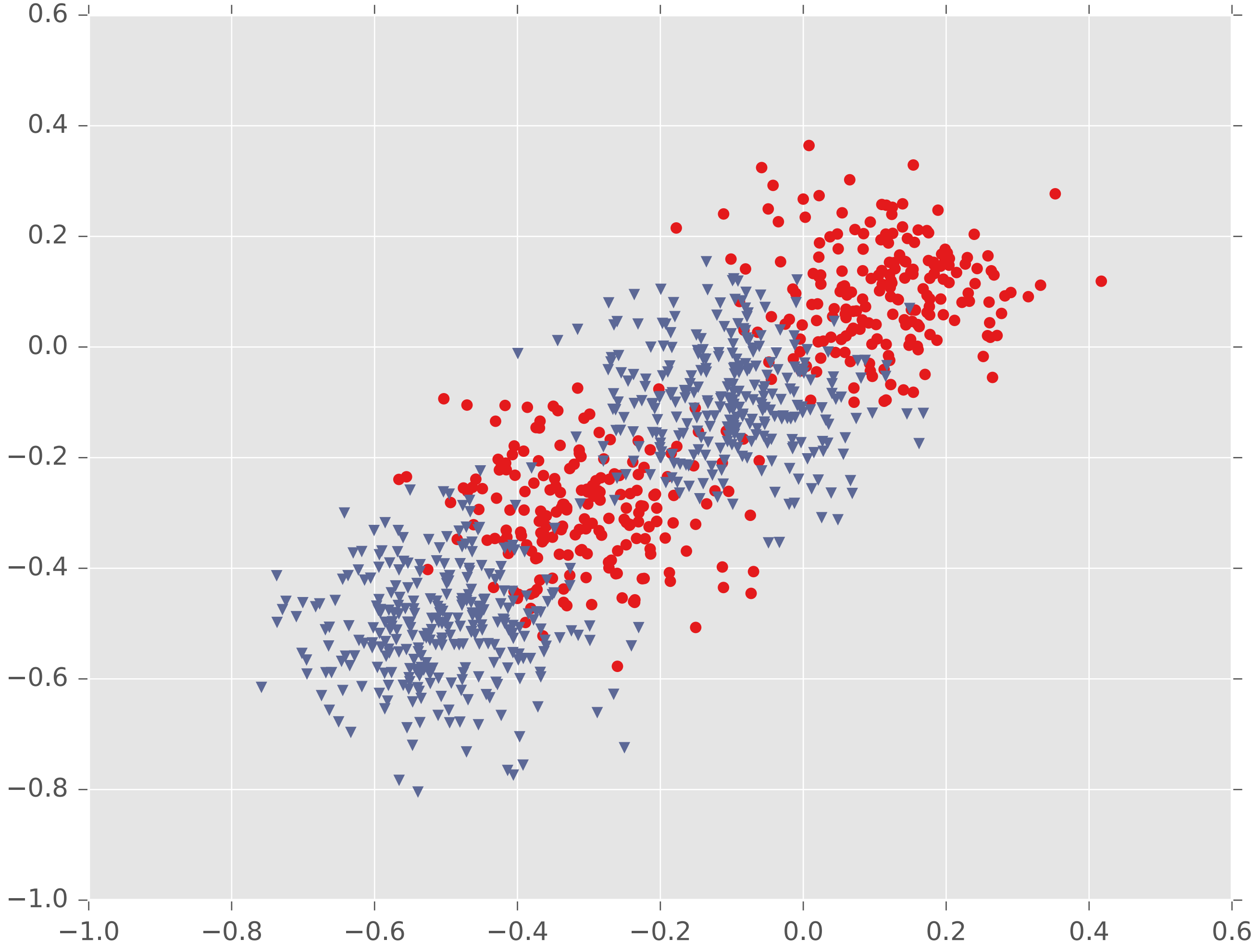


Example #3: Four Gaussians

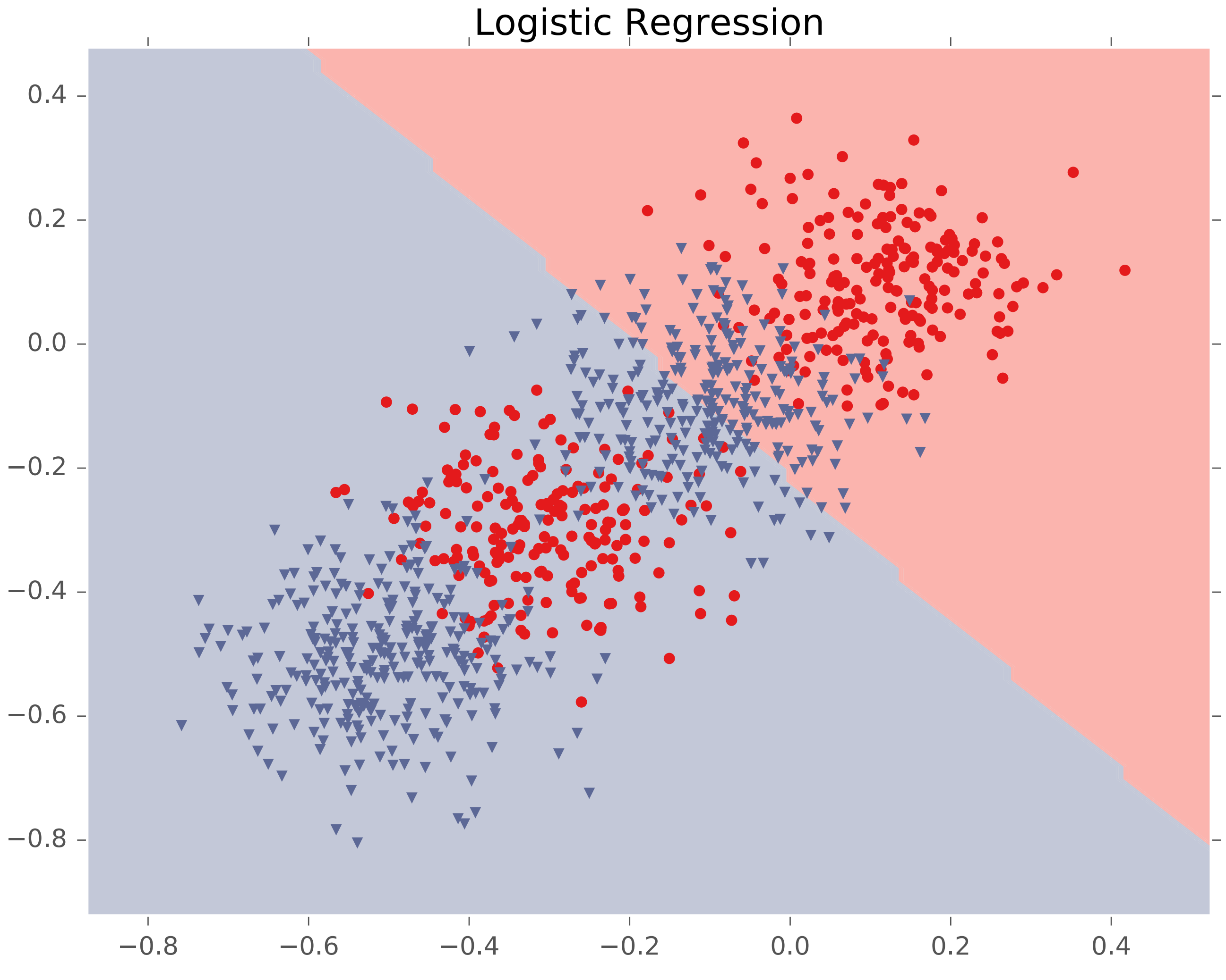
Example #4: Two Pockets



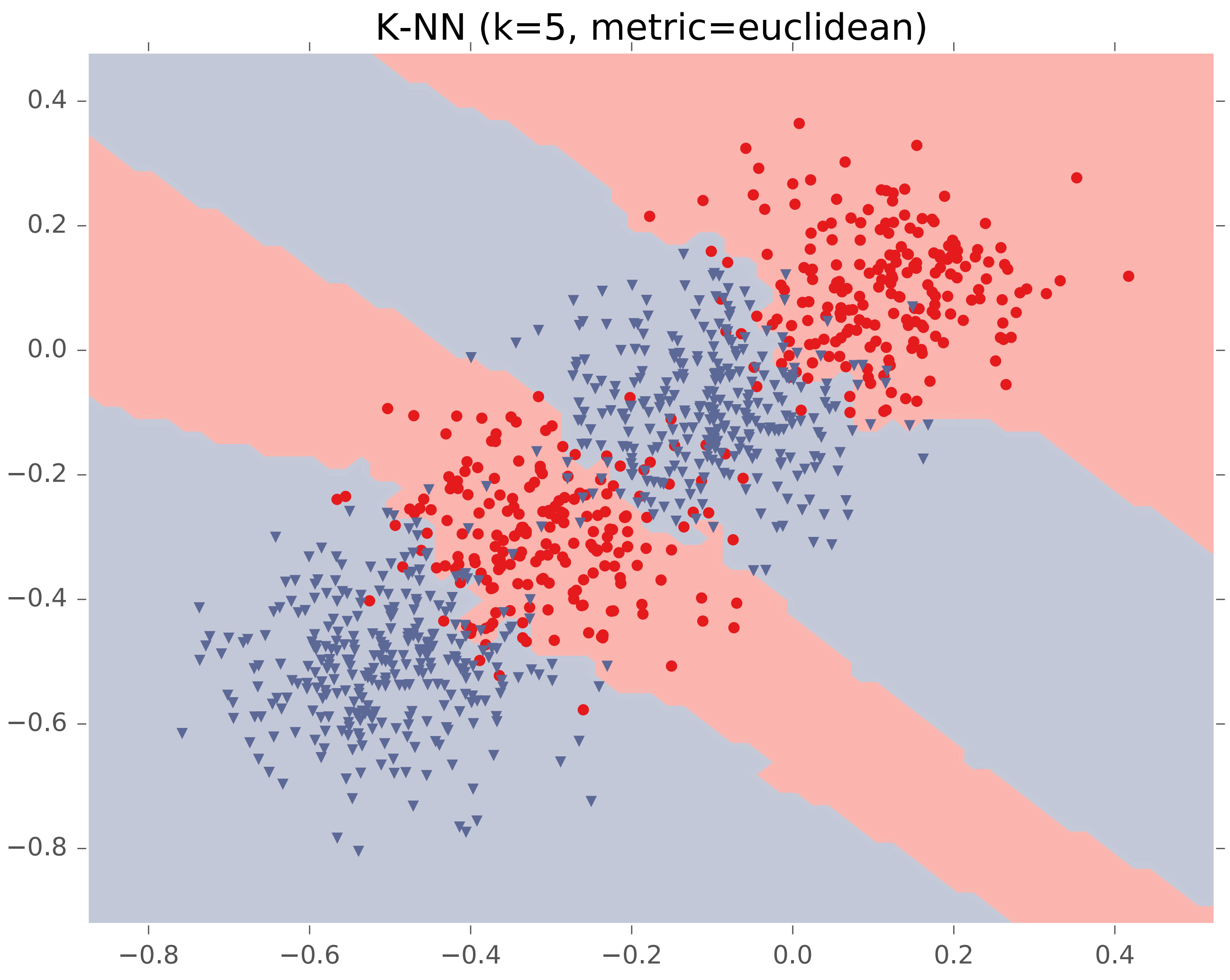
Example #3: Four Gaussians



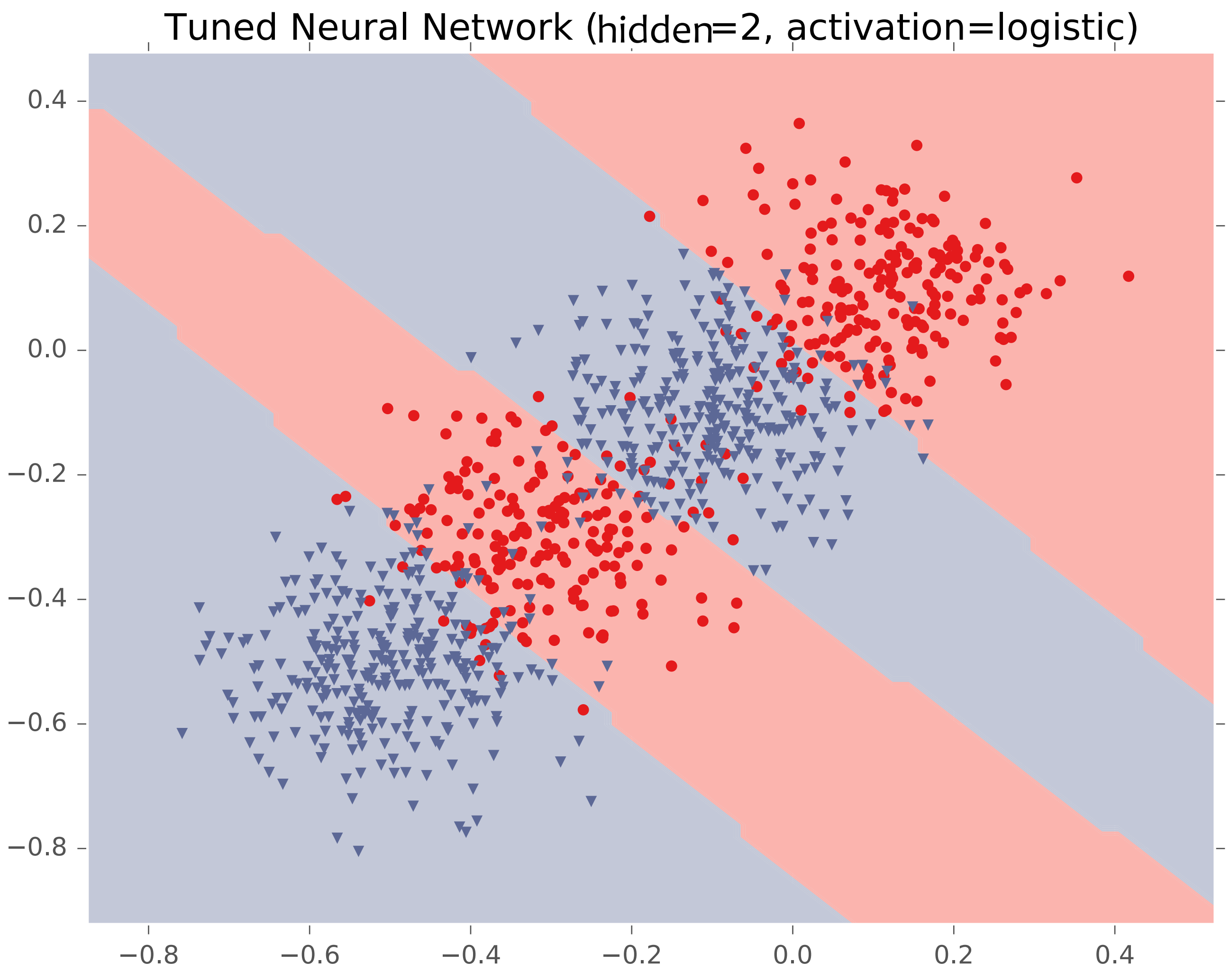
Example #3: Four Gaussians



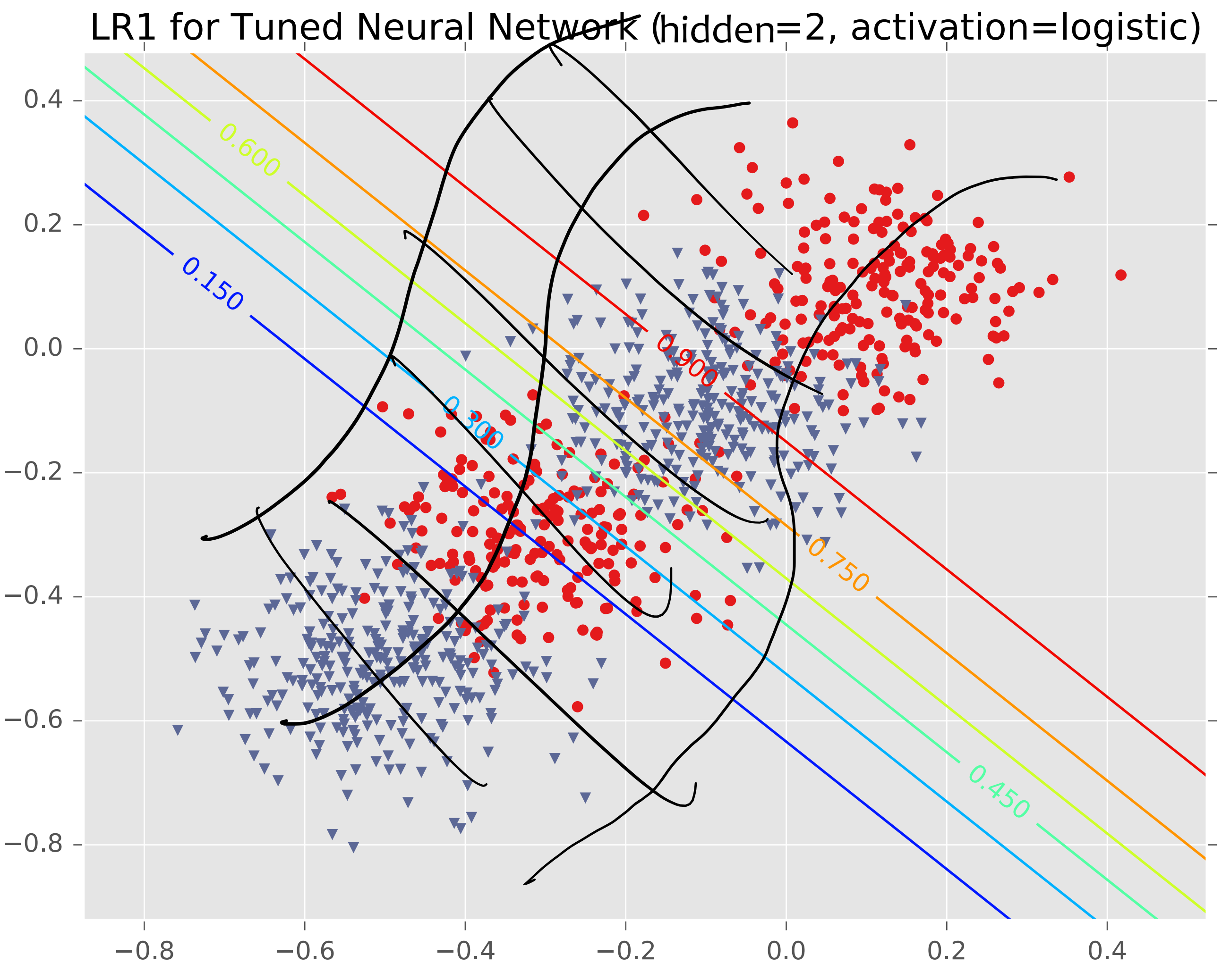
Example #3: Four Gaussians



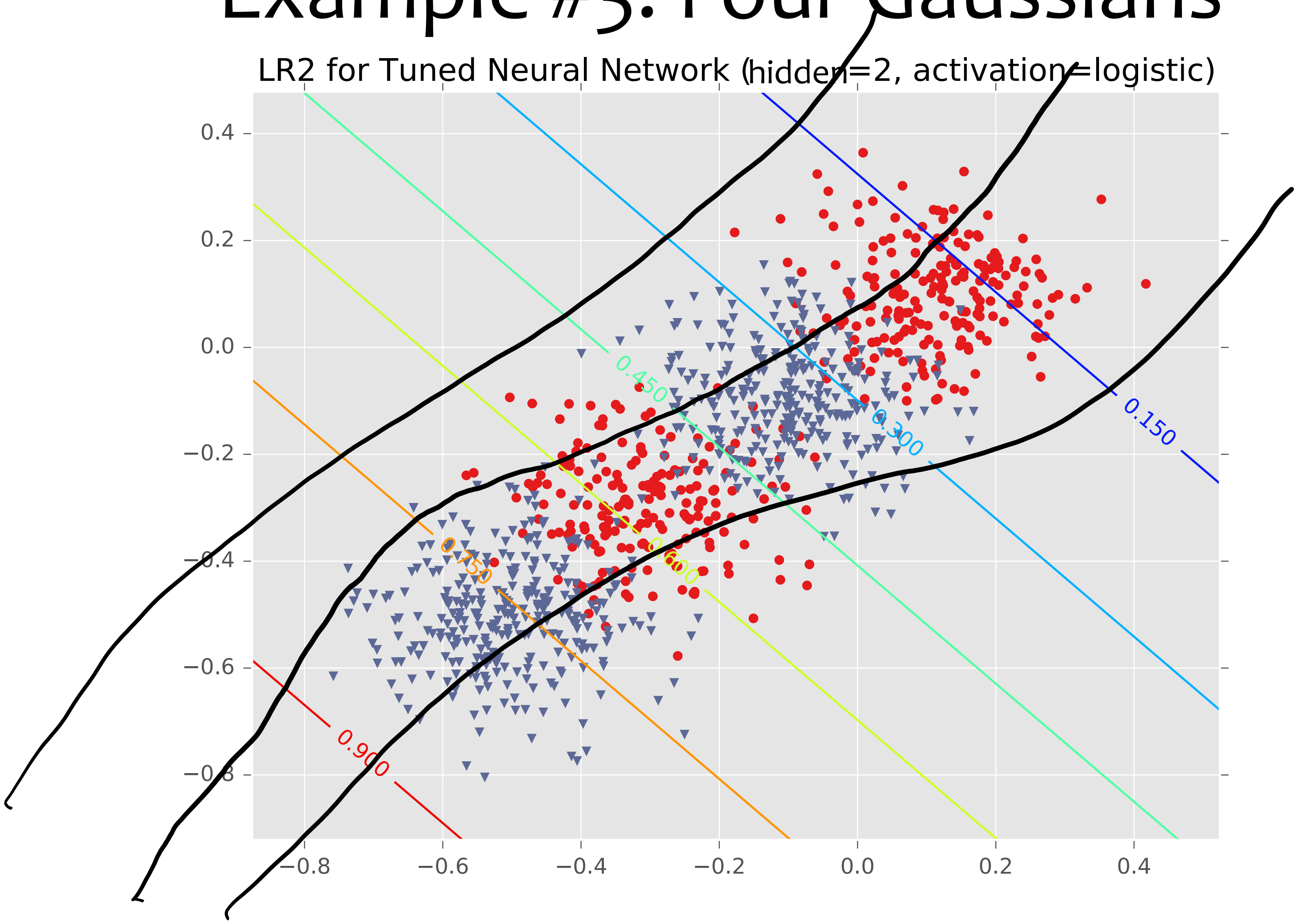
Example #3: Four Gaussians



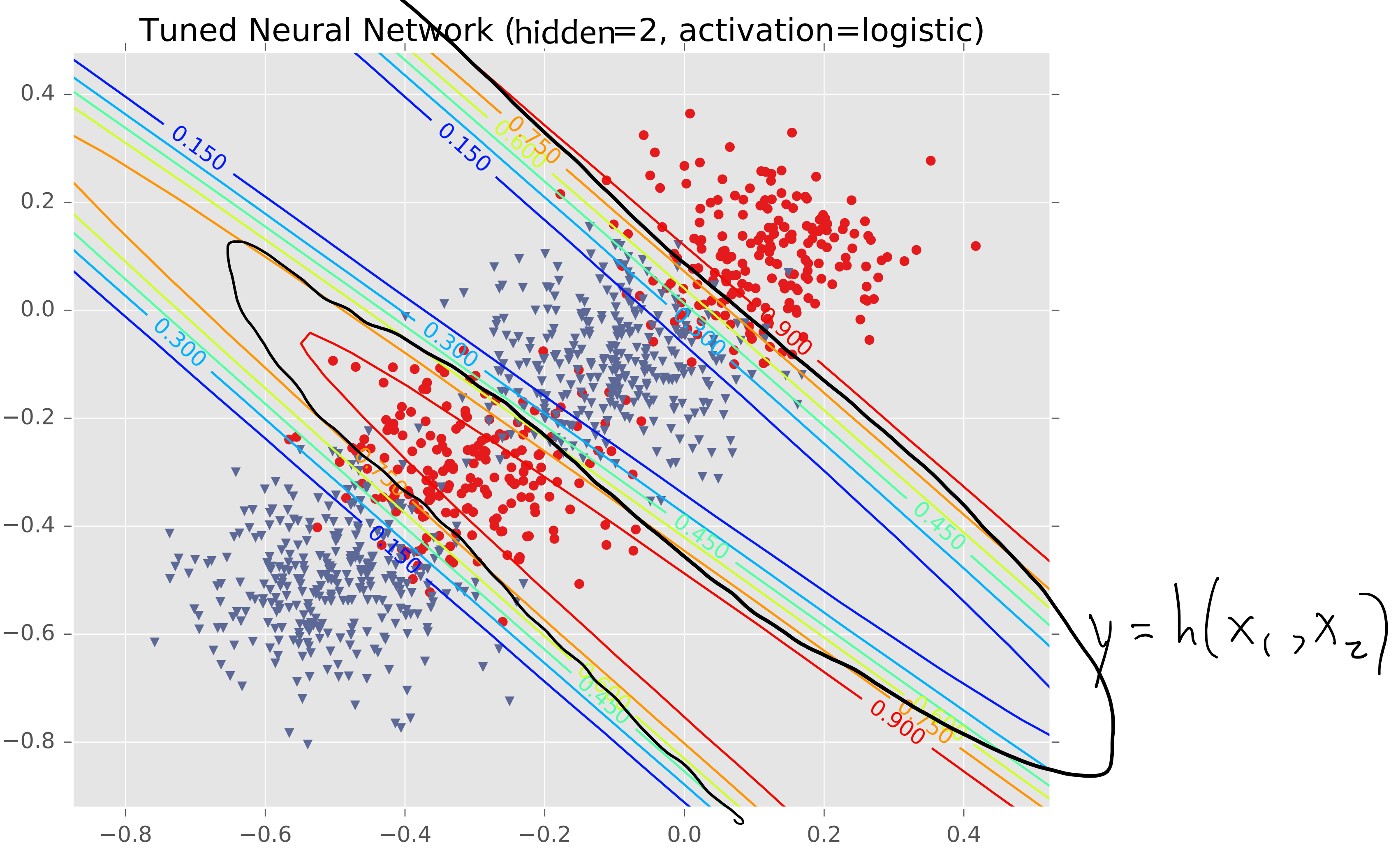
Example #3: Four Gaussians



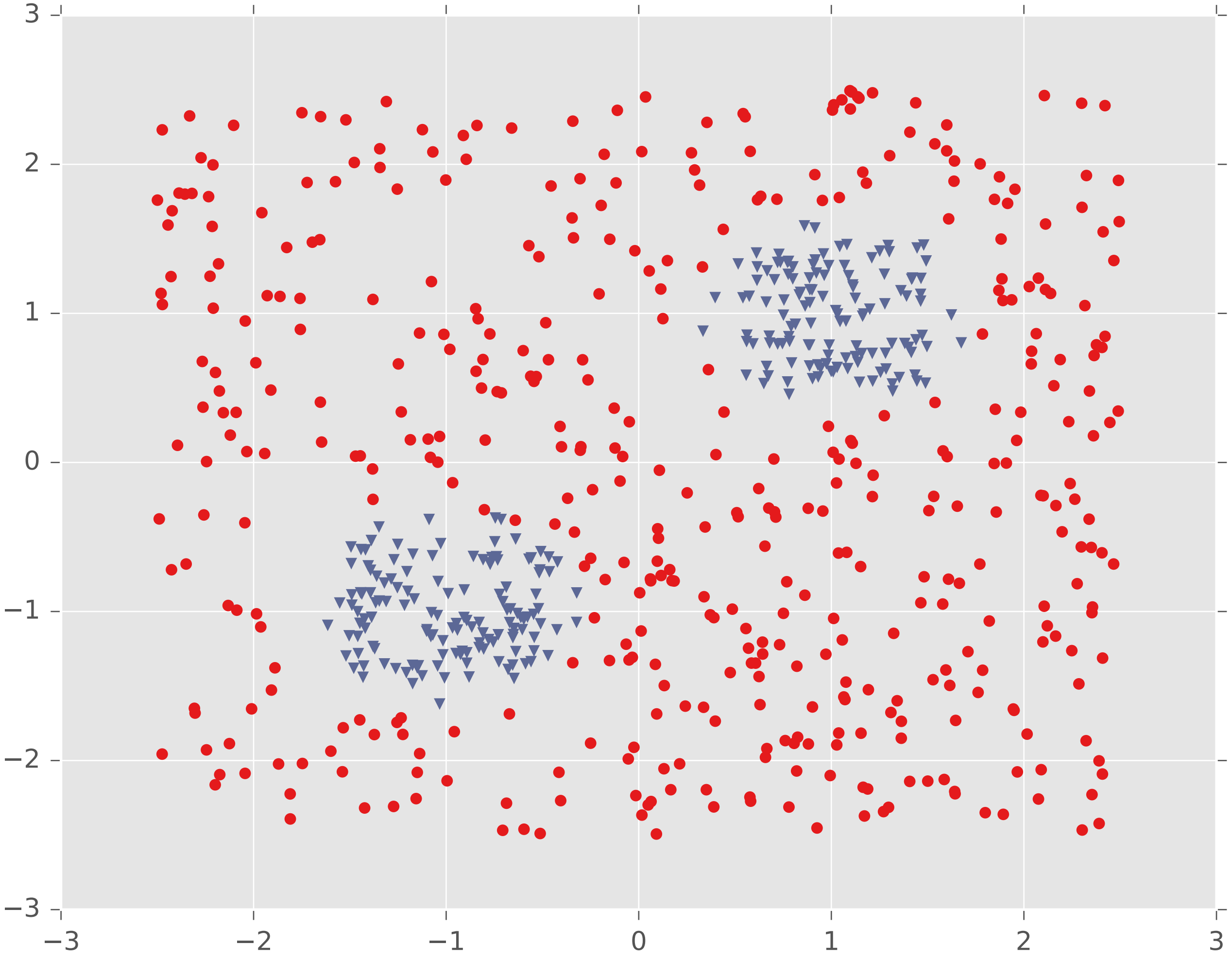
Example #3: Four Gaussians



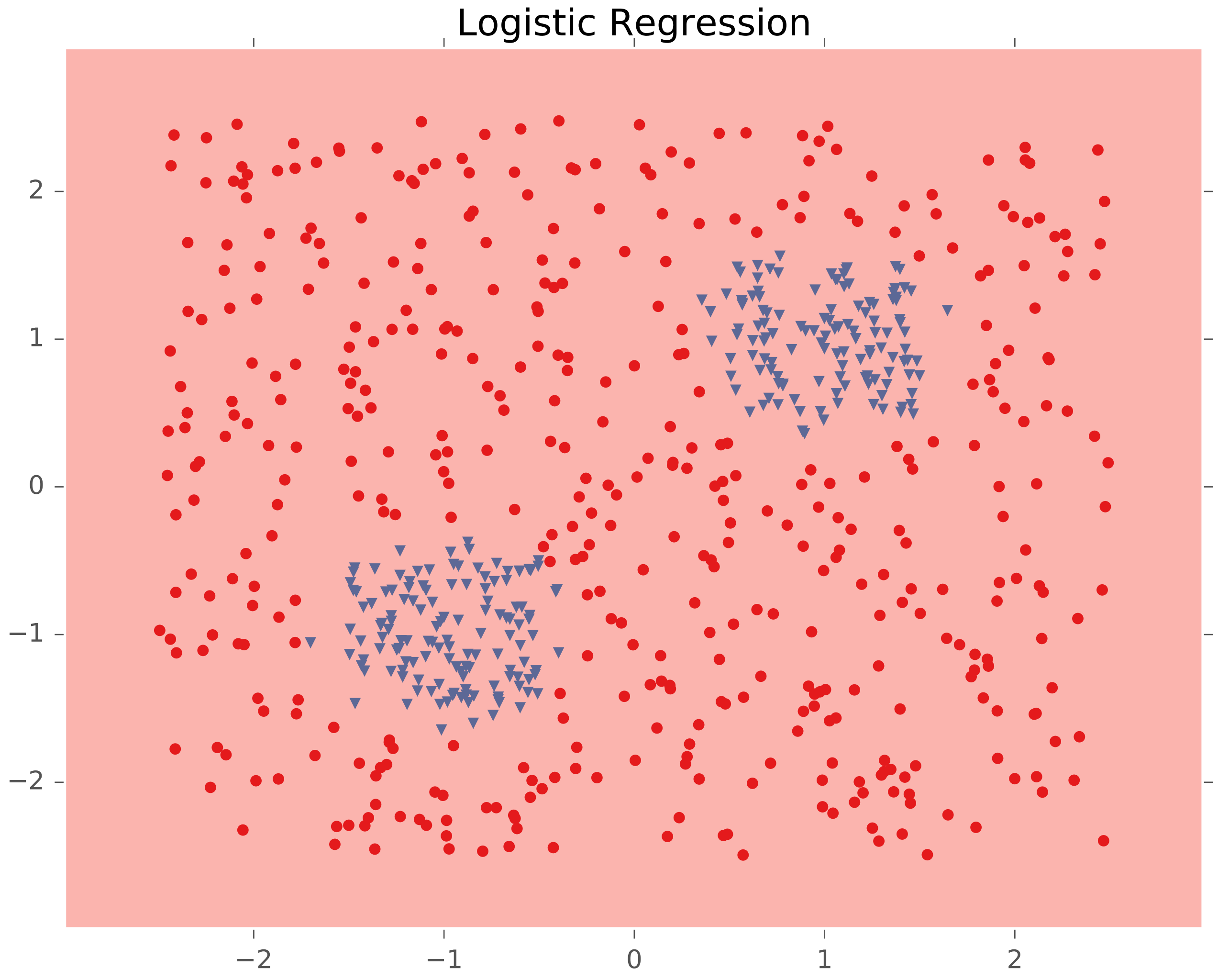
Example #3: Four Gaussians



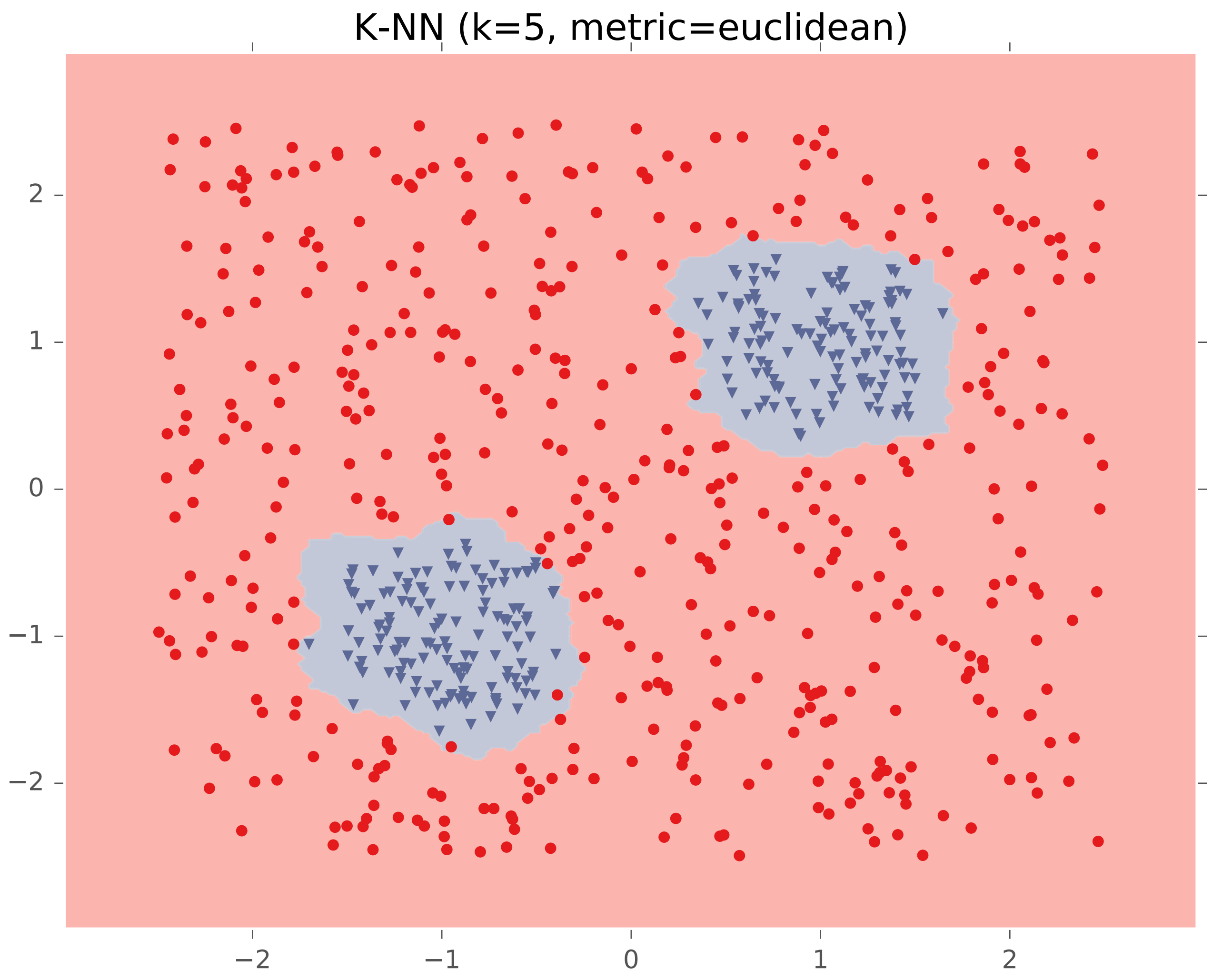
Example #4: Two Pockets



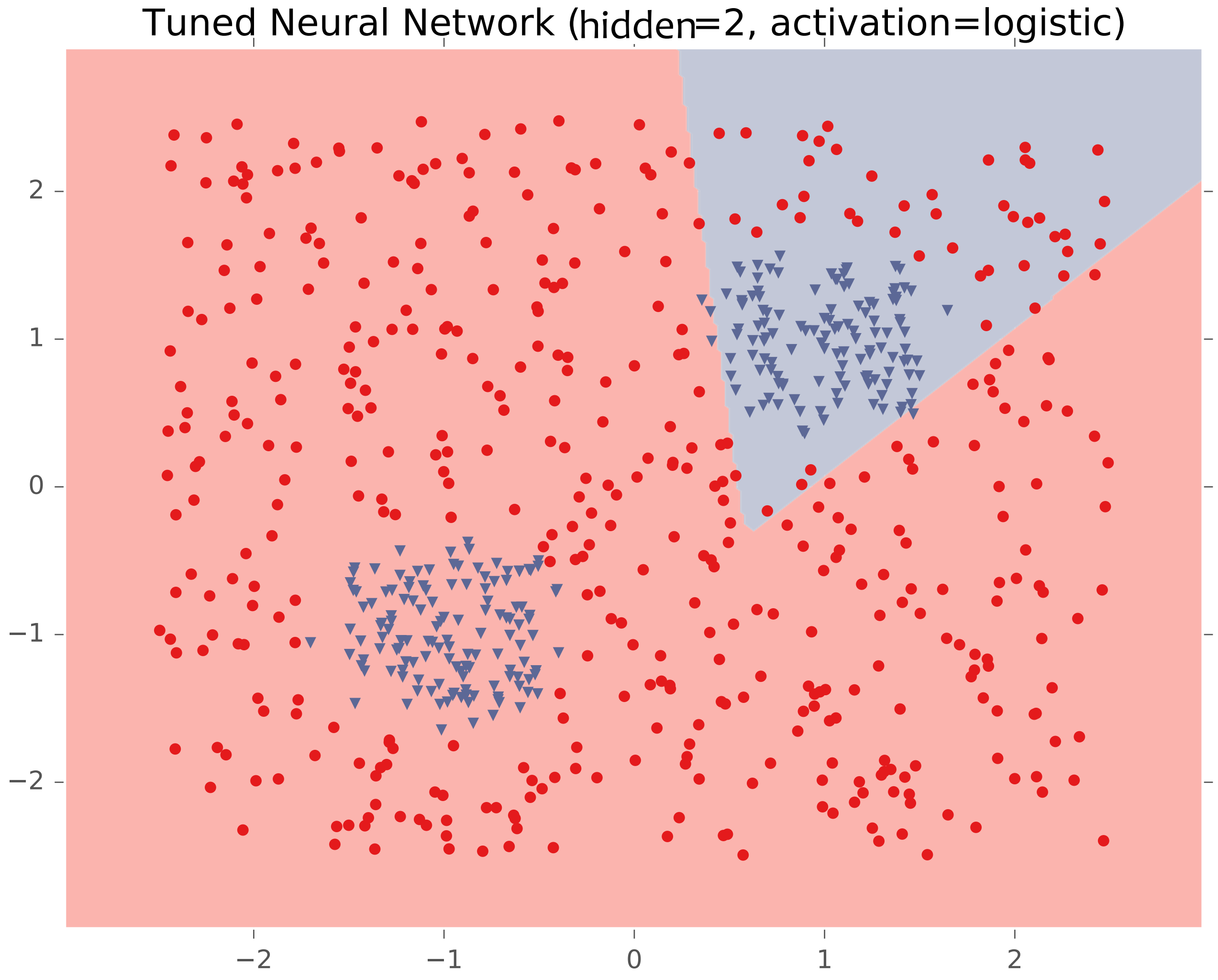
Example #4: Two Pockets



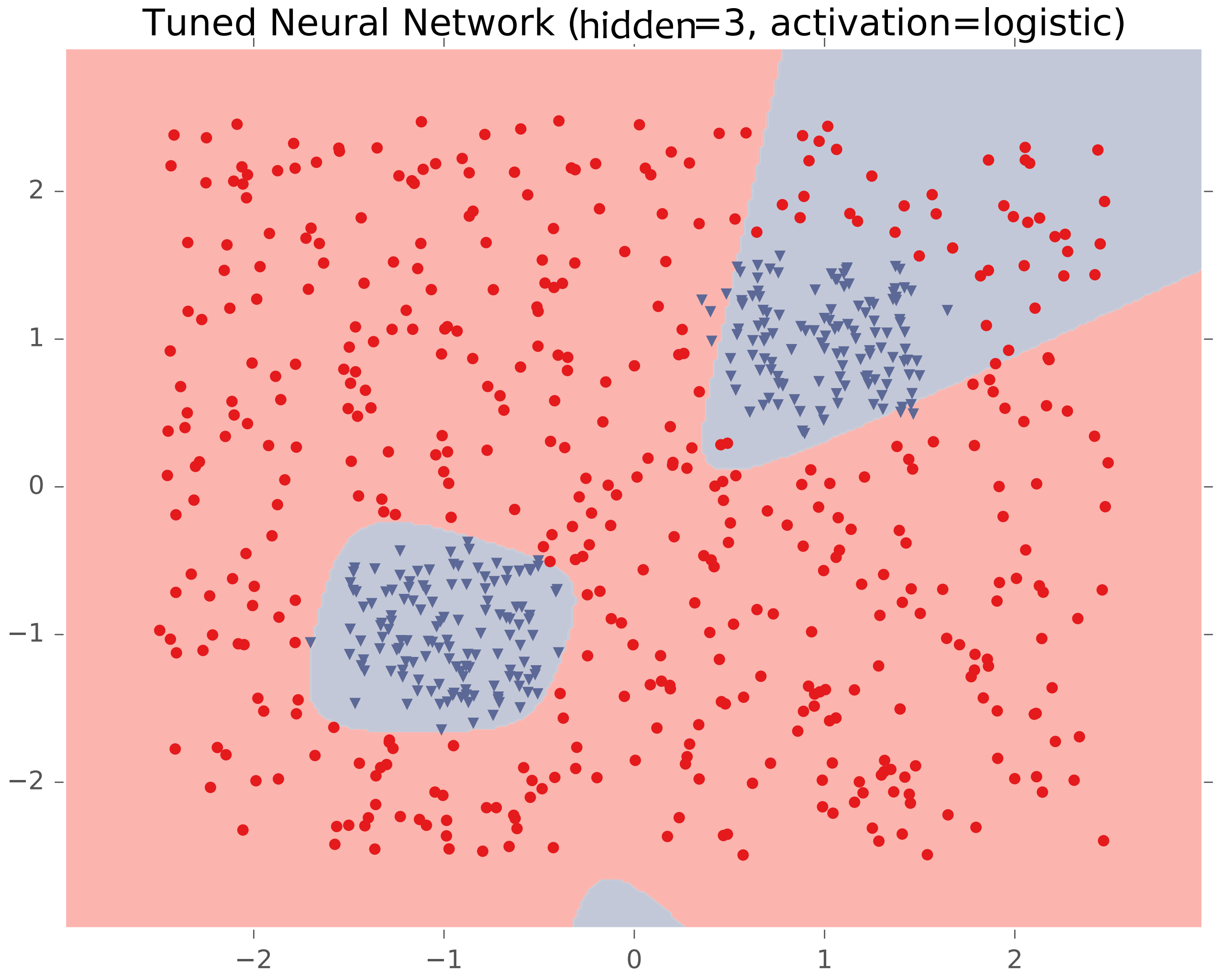
Example #4: Two Pockets



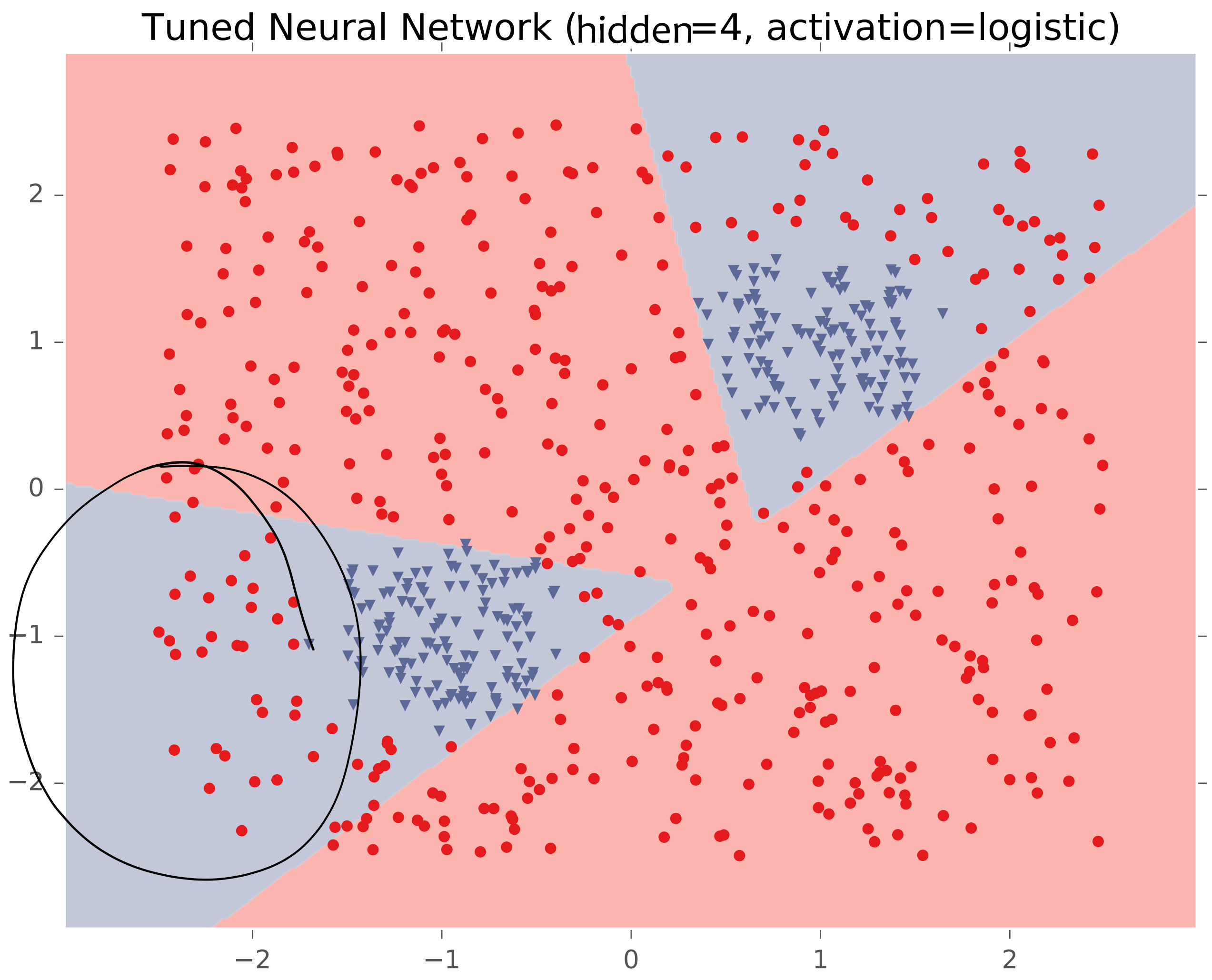
Example #4: Two Pockets



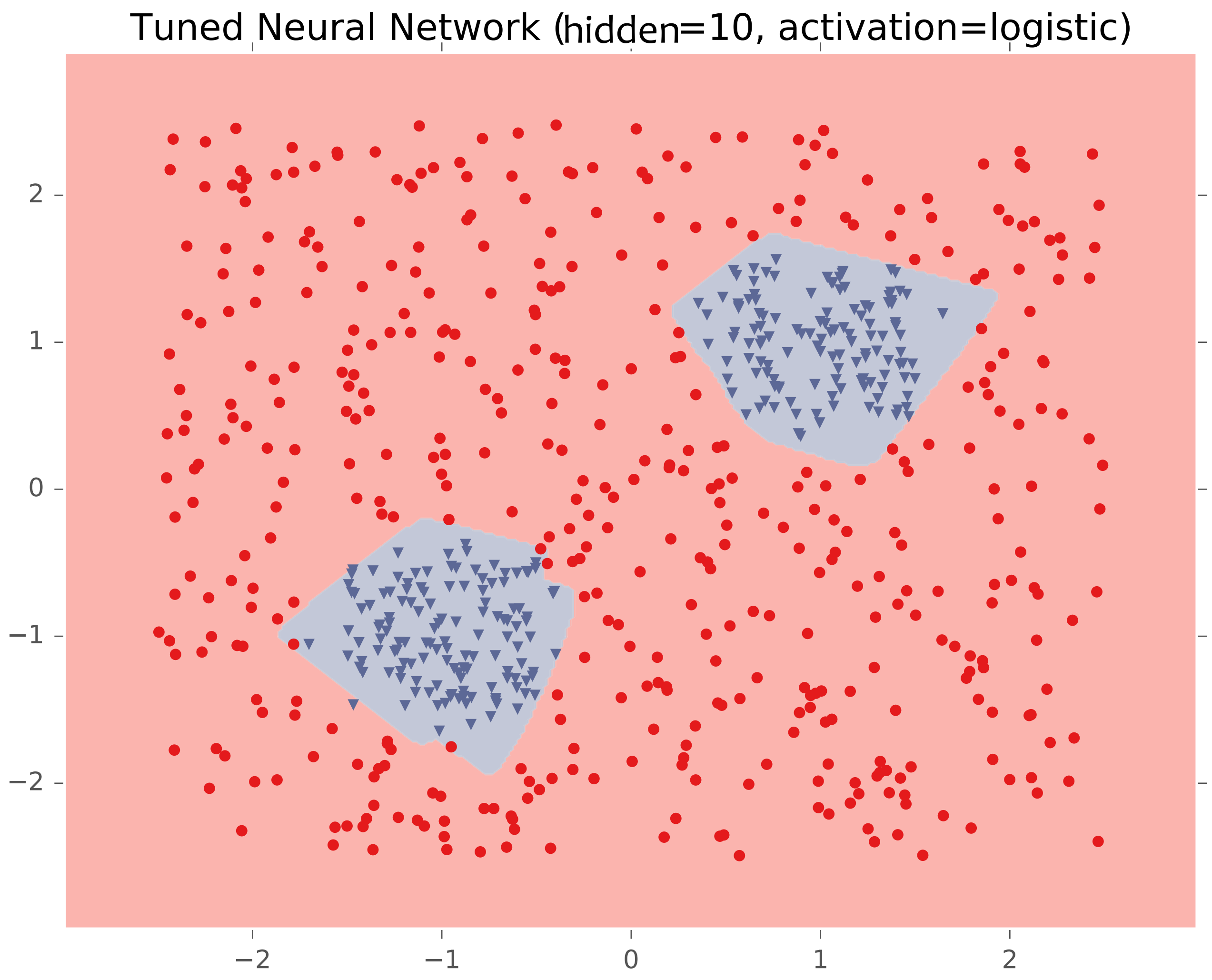
Example #4: Two Pockets



Example #4: Two Pockets

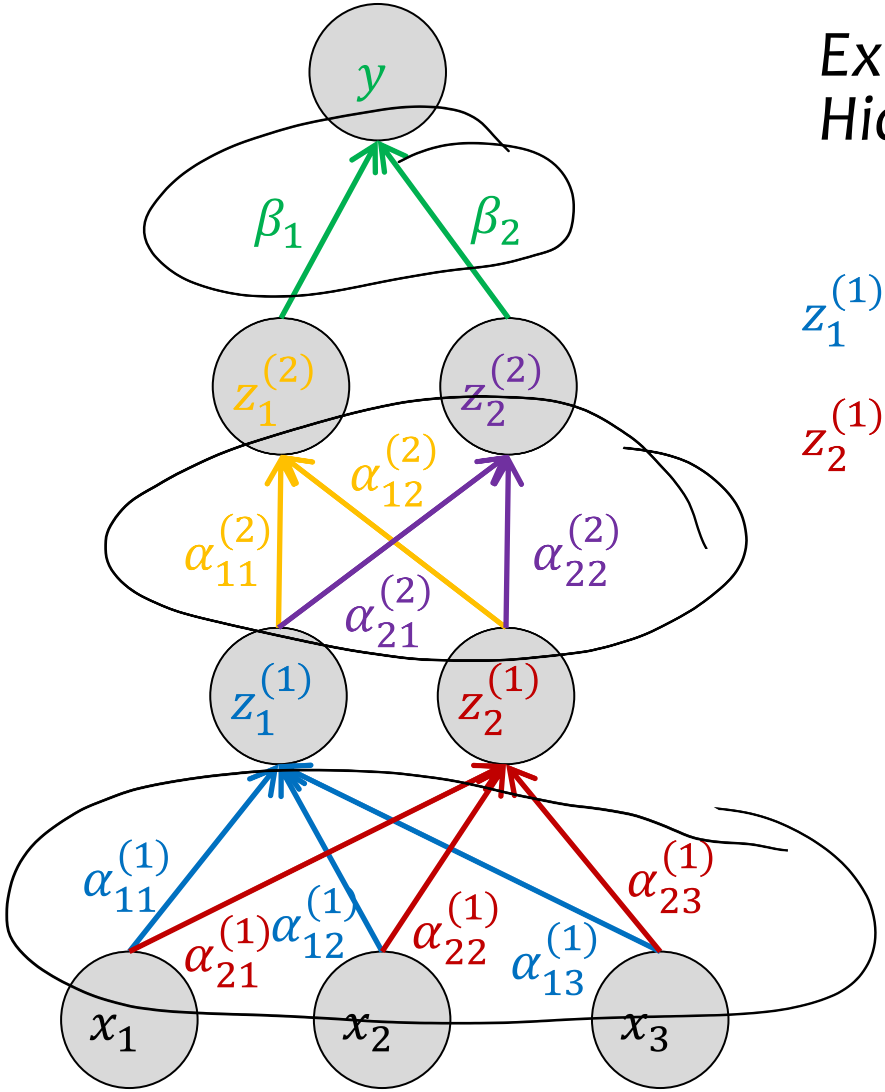


Example #4: Two Pockets



BUILDING DEEPER NETWORKS

Neural Network



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

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

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

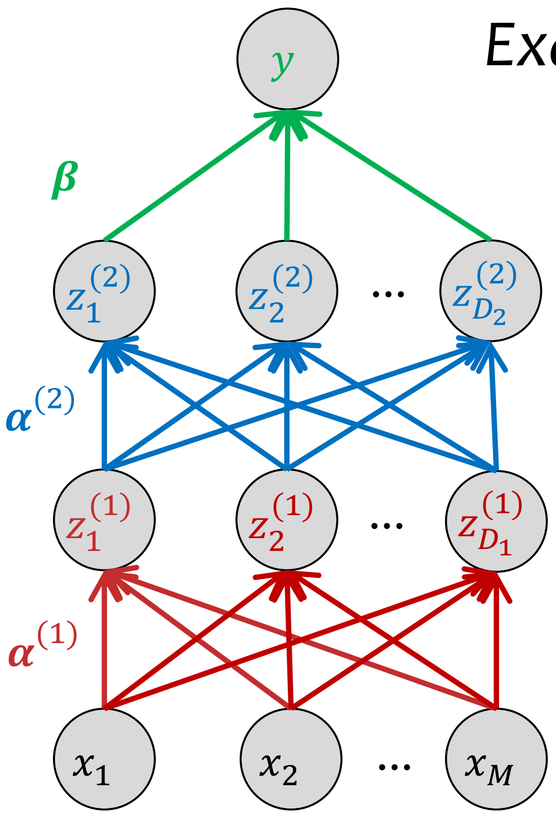
$$z_1^{(2)} = \sigma(\alpha_{11}^{(2)} z_1^{(1)} + \alpha_{12}^{(2)} z_2^{(1)} + \alpha_{10}^{(2)})$$

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

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

Neural Network (Matrix Form)

Example: Arbitrary Feed-forward Neural Network



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

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

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

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

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

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

1x1

1xD₂ D₂x1 + 1x1

$$y = \sigma((\boldsymbol{\beta})^T \boldsymbol{z}^{(2)} + \beta_0) \in \mathbb{R}$$

$$\boldsymbol{z}^{(2)} = \sigma((\boldsymbol{\alpha}^{(2)})^T \boldsymbol{z}^{(1)} + \boldsymbol{b}^{(2)})$$ D₂ x 1

$$\boldsymbol{z}^{(1)} = \sigma((\boldsymbol{\alpha}^{(1)})^T \boldsymbol{x} + \boldsymbol{b}^{(1)})$$ D₁ x 1

D₁xM Mx1 D₁x1

D₁x1

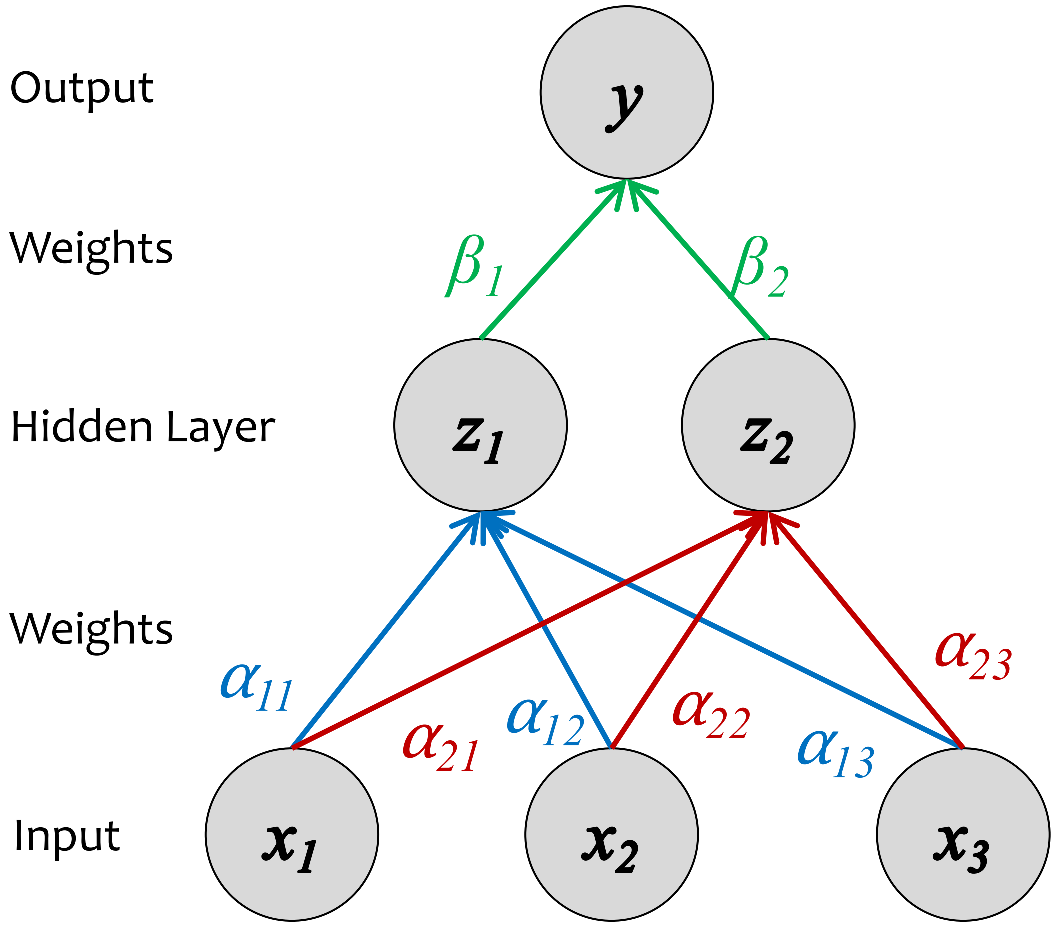
D₁ x 1

applied sigmoid element wise

$\vec{z} = \sigma(\vec{u})$

$\forall i, \; z_i = \sigma(u_i)$

Neural Network (Vector Form)



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

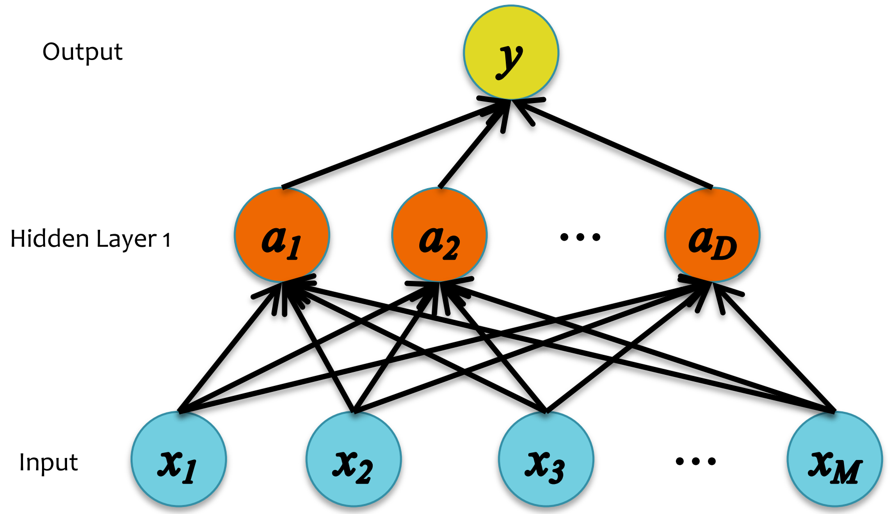
$$y = \sigma(\boldsymbol{\beta}^T \mathbf{z})$$

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

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

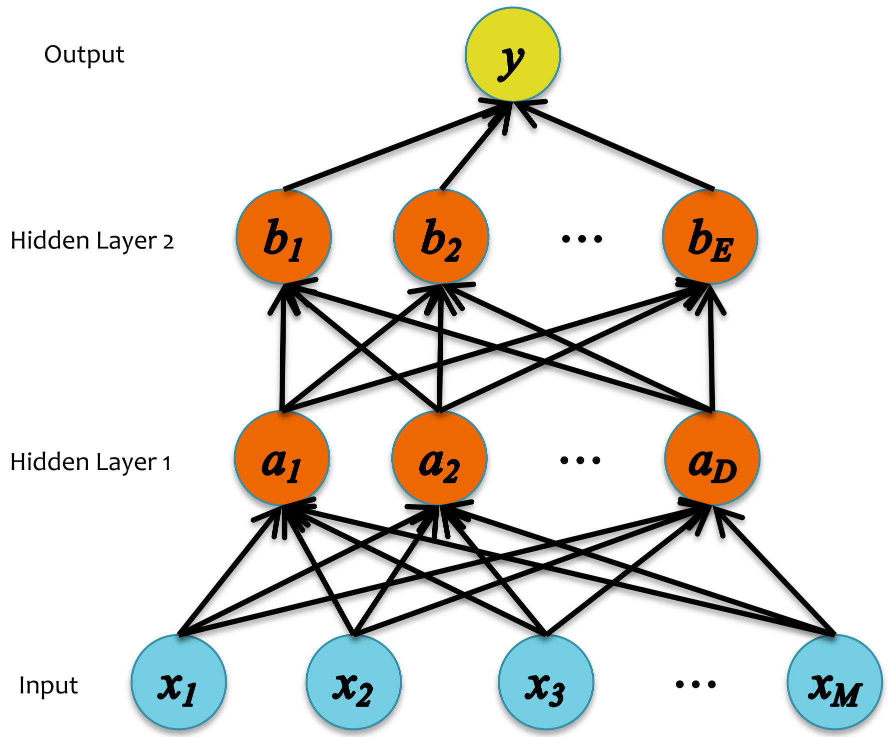
Deeper Networks

Q: How many layers should we use?



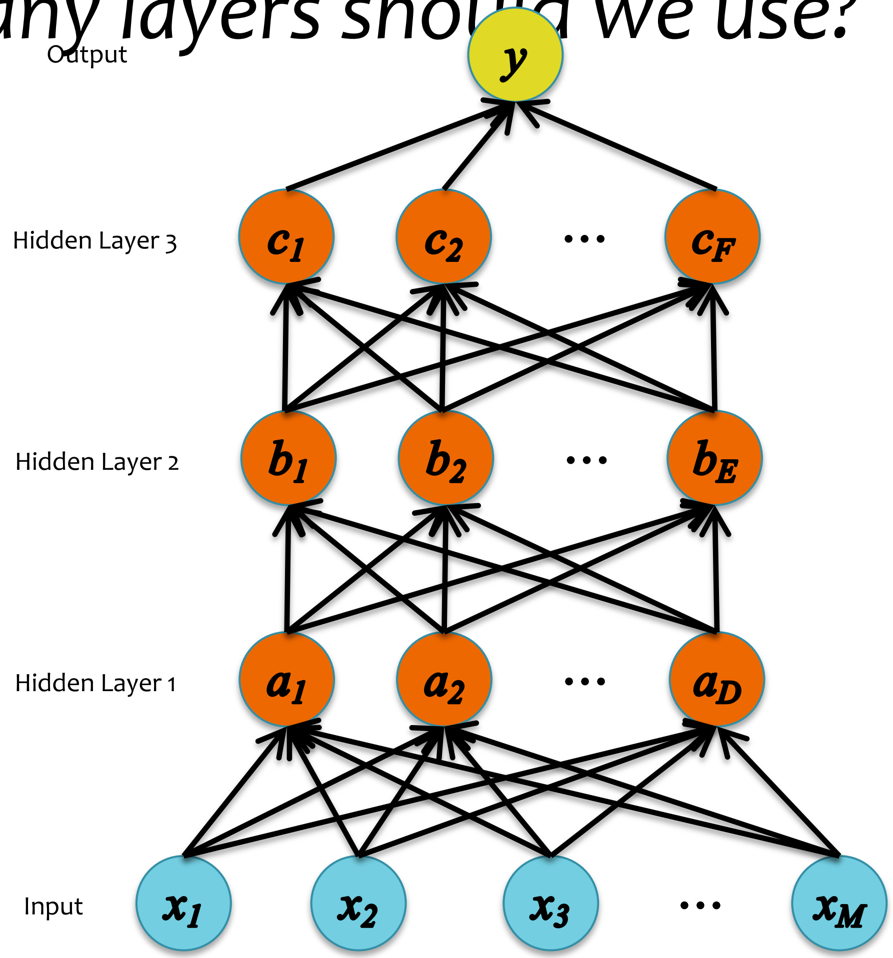
Deeper Networks

Q: How many layers should we use?



Deeper Networks

Q: How many layers should we use?



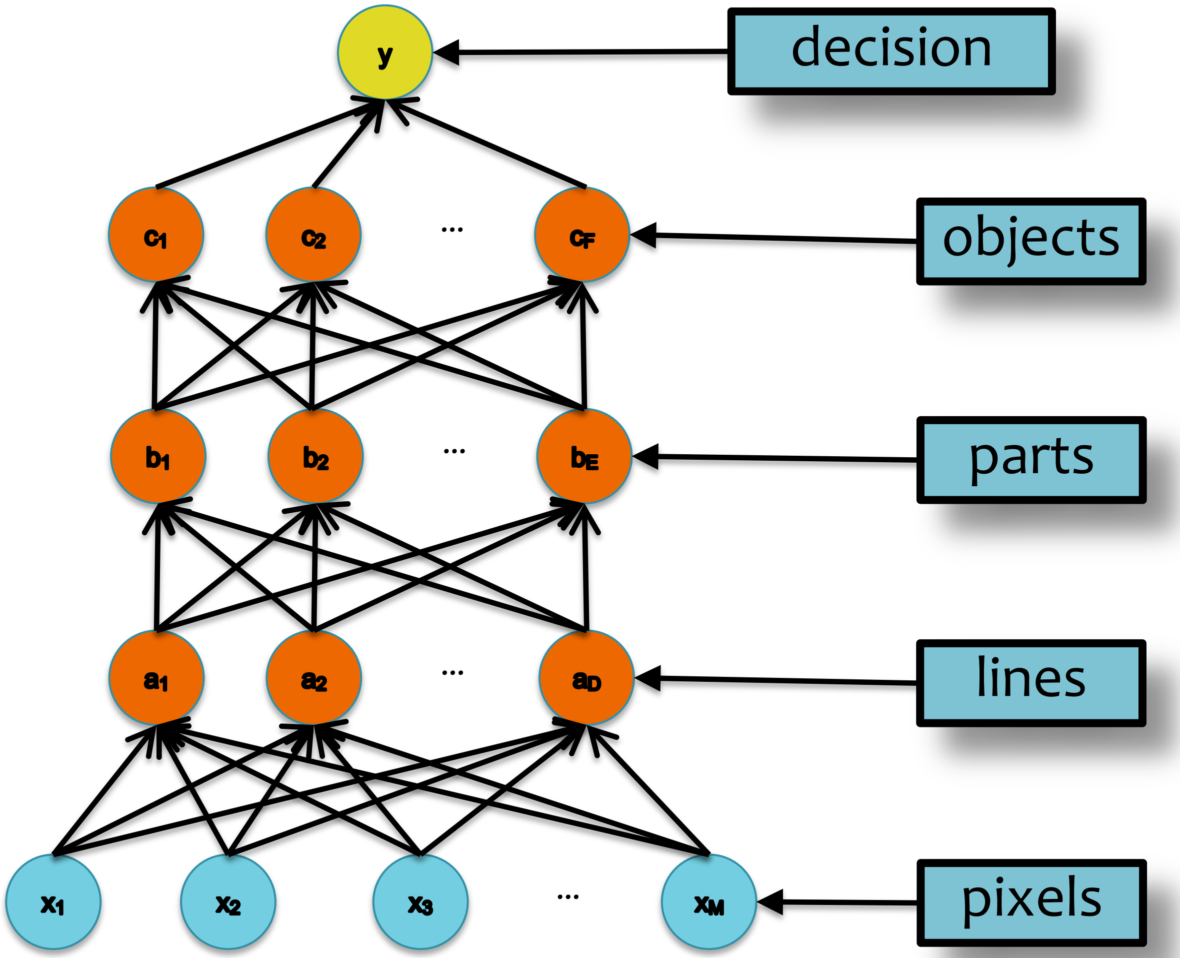
Deeper Networks

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.

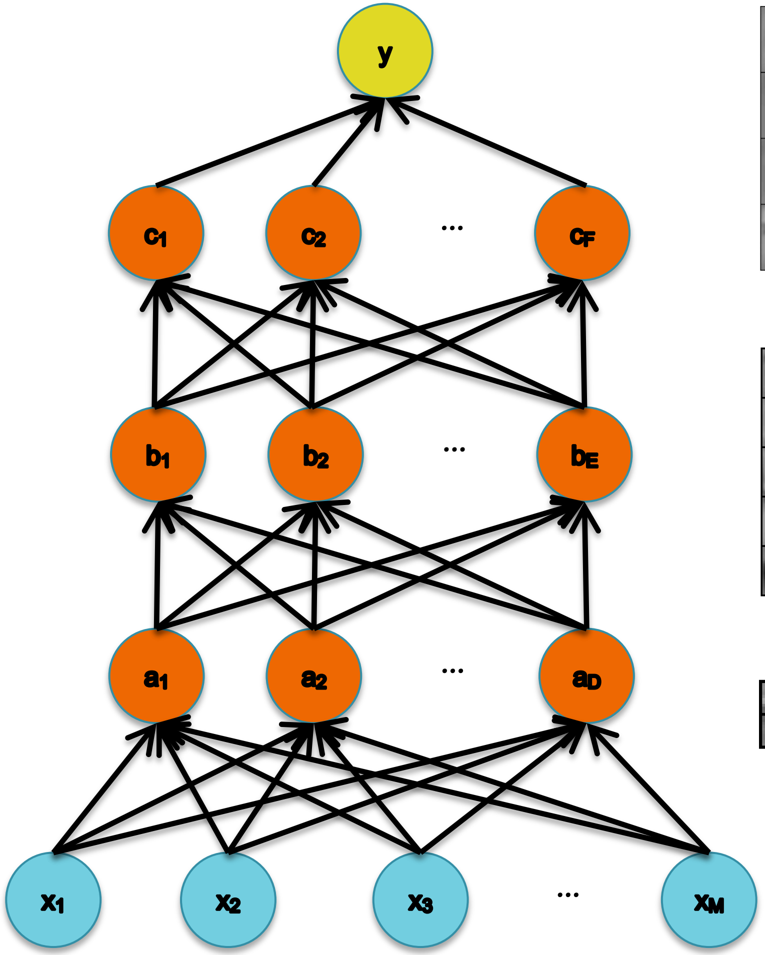
Feature Learning



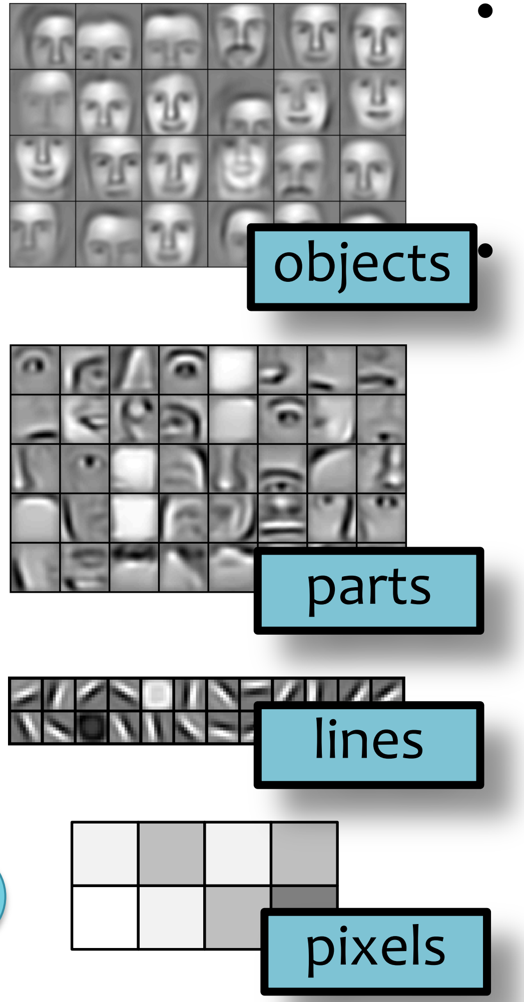
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning



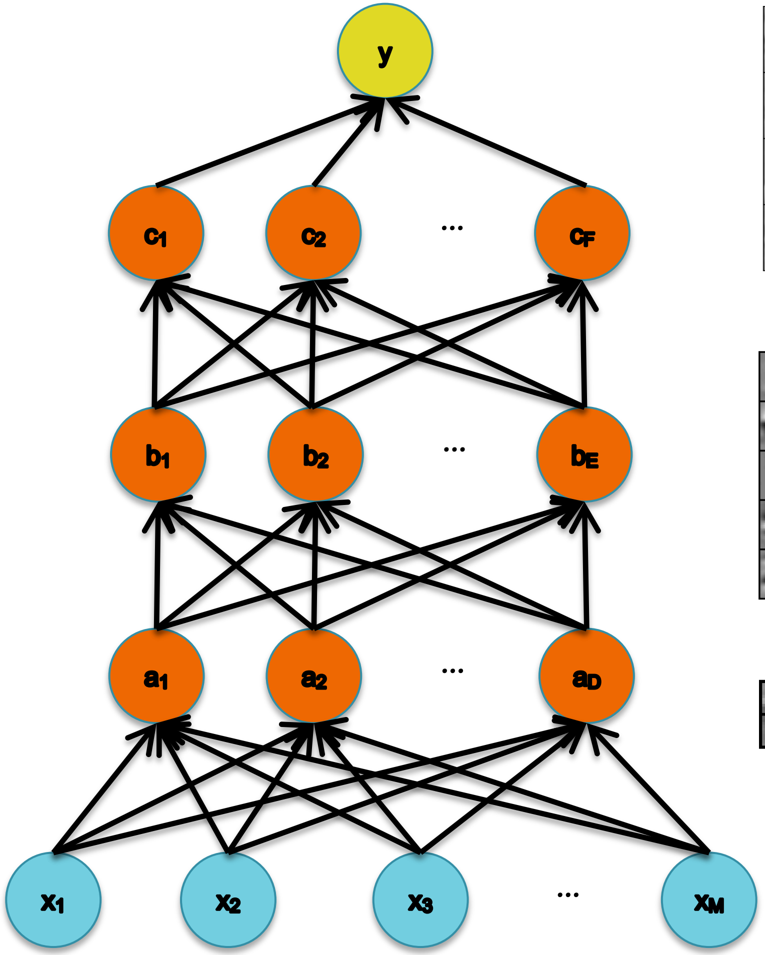
CBDN on Faces

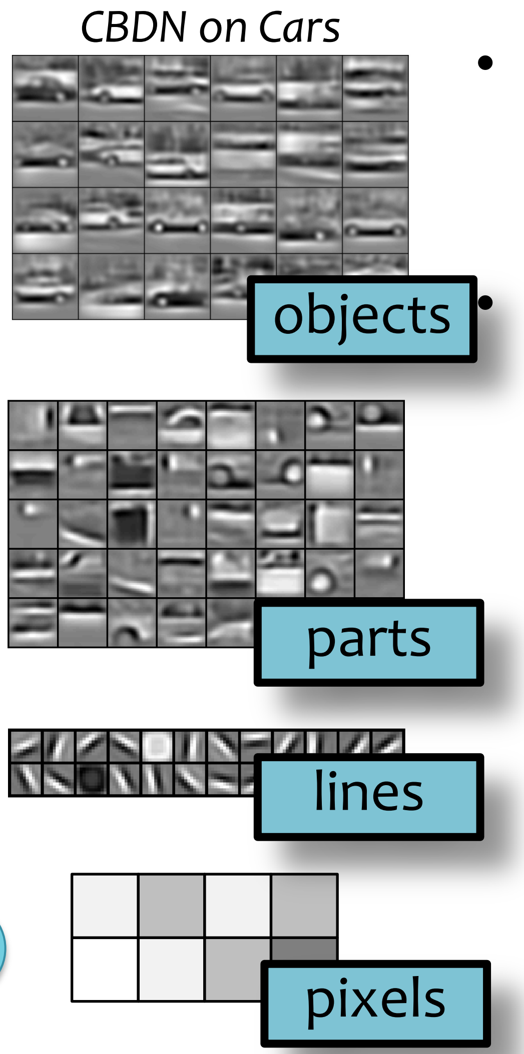


- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Feature Learning





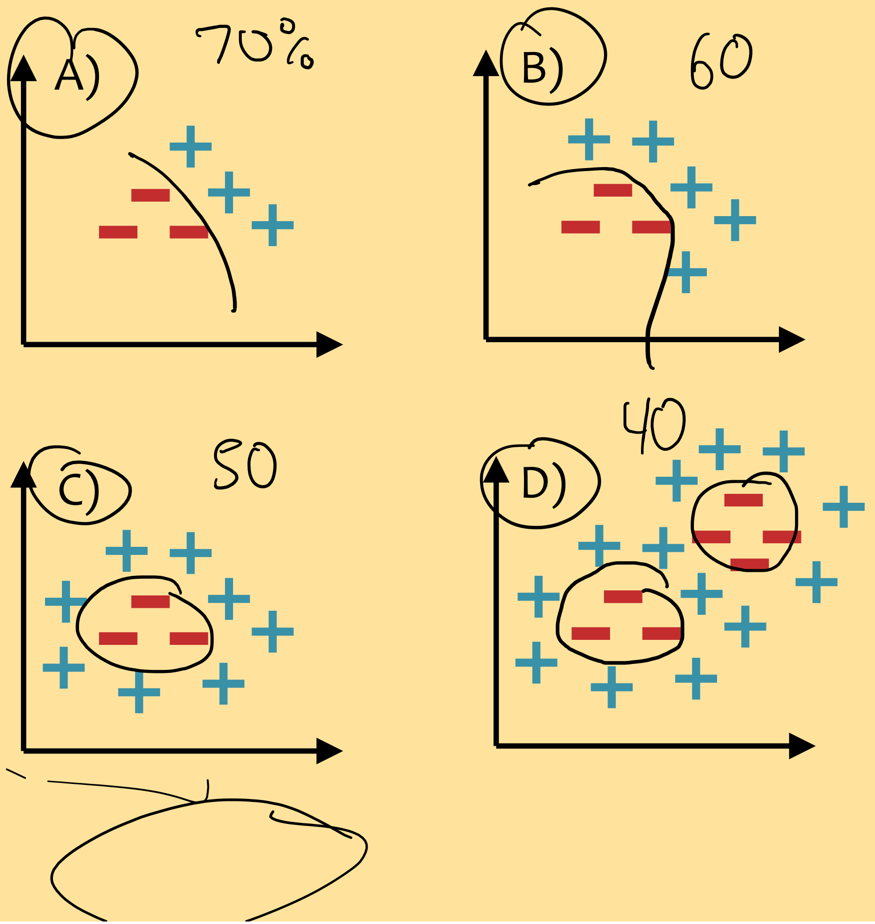
- **Traditional feature engineering:** build up levels of abstraction by hand
- **Deep networks** (e.g. convolution networks): learn the increasingly higher levels of abstraction from data
  - each layer is a learned feature representation
  - sophistication increases in higher layers

Figures from Lee et al. (ICML 2009)

Neural Network Errors

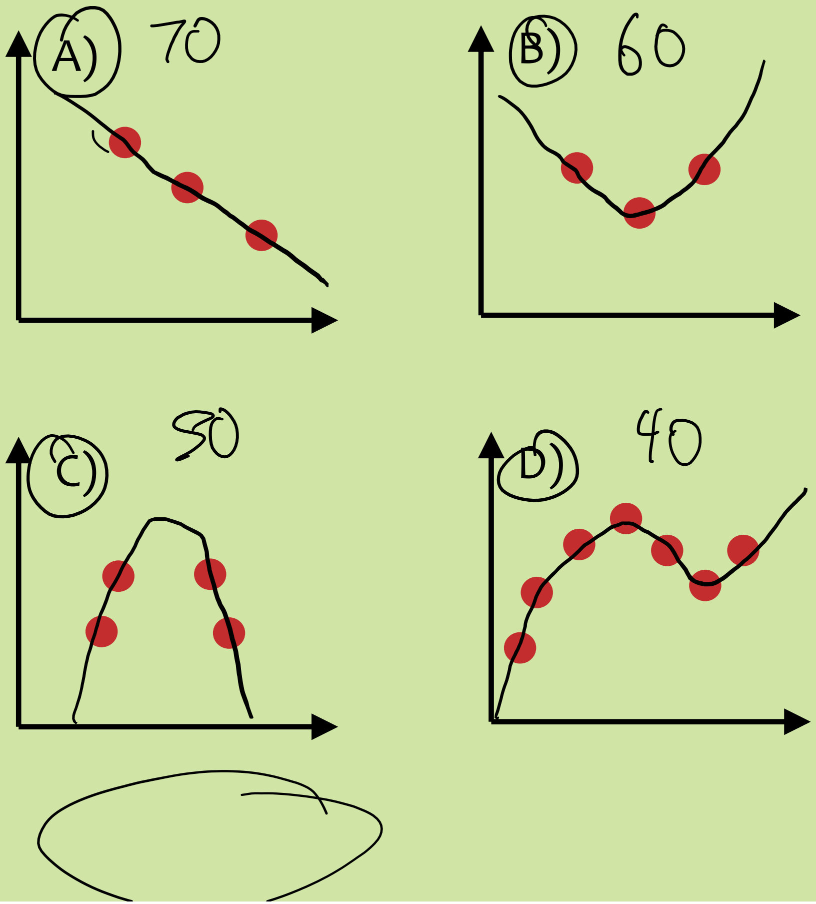
Q2

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.**



Q3

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

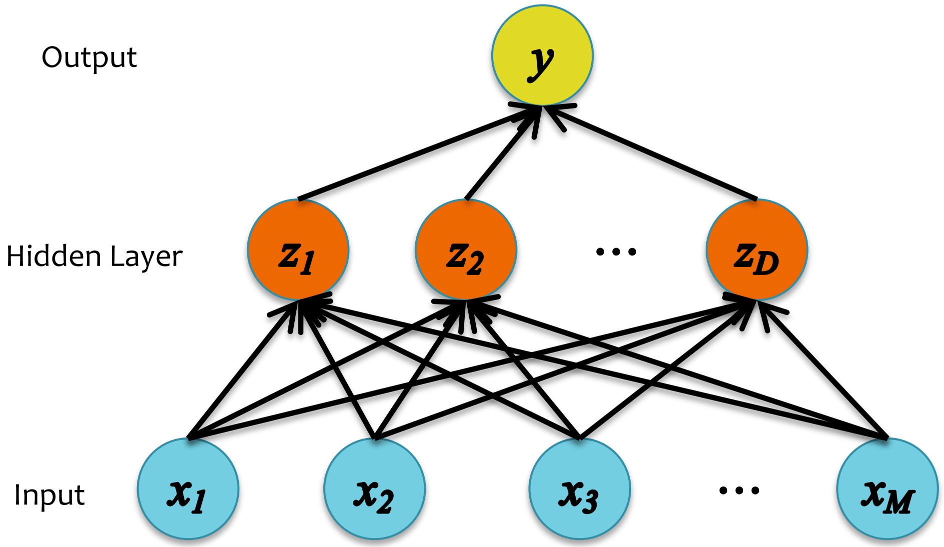
Even for a basic Neural Network, there are many design decisions to make:

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



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

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

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

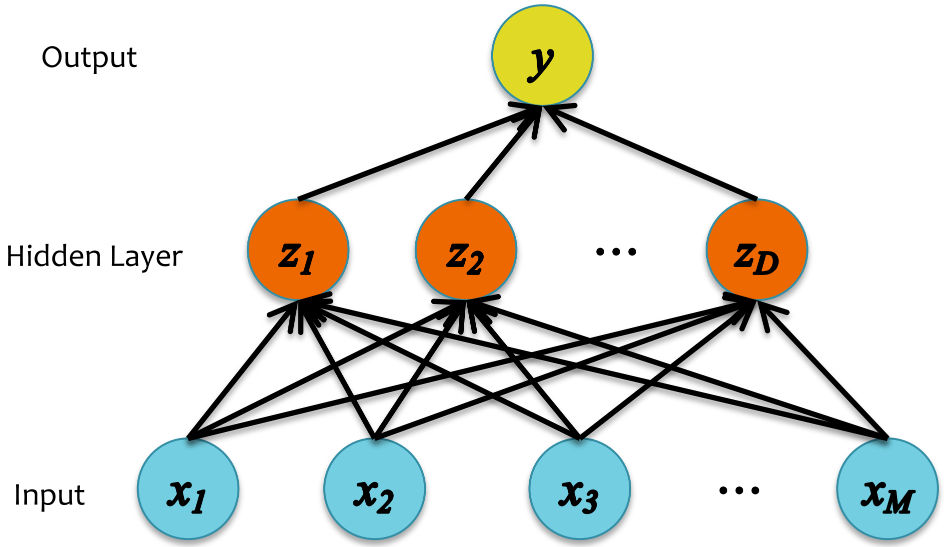
(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

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

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

Activation Functions

Neural Network with arbitrary nonlinear activation functions



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

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

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

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

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

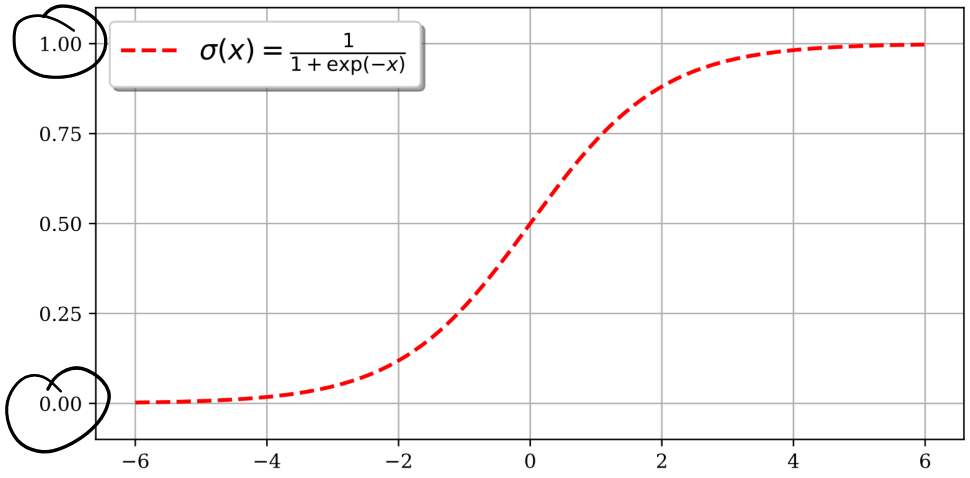
(A) Input
Given $x_i, \ \forall i$

Activation Functions

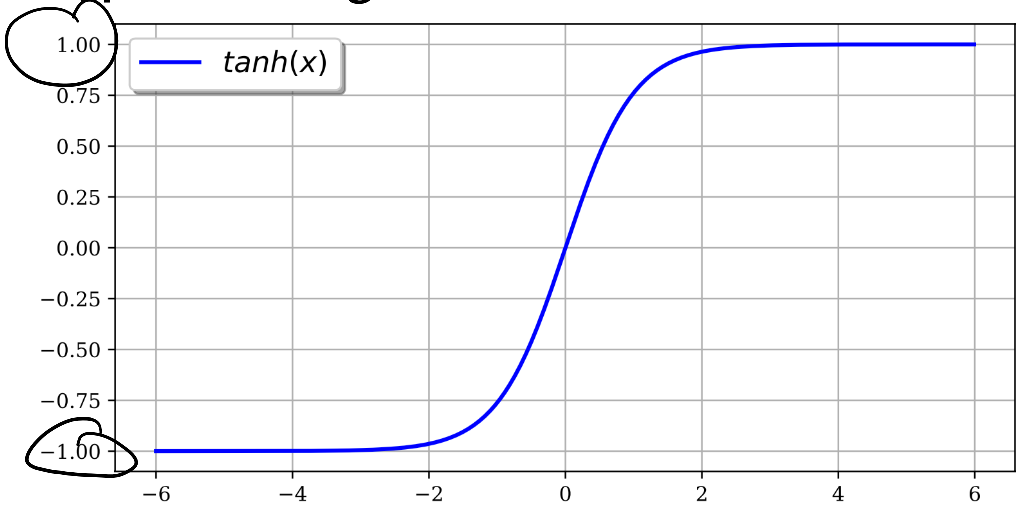
So far, we've assumed that the activation function (nonlinearity) is always the sigmoid function…

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

Sigmoid (aka. logistic) function



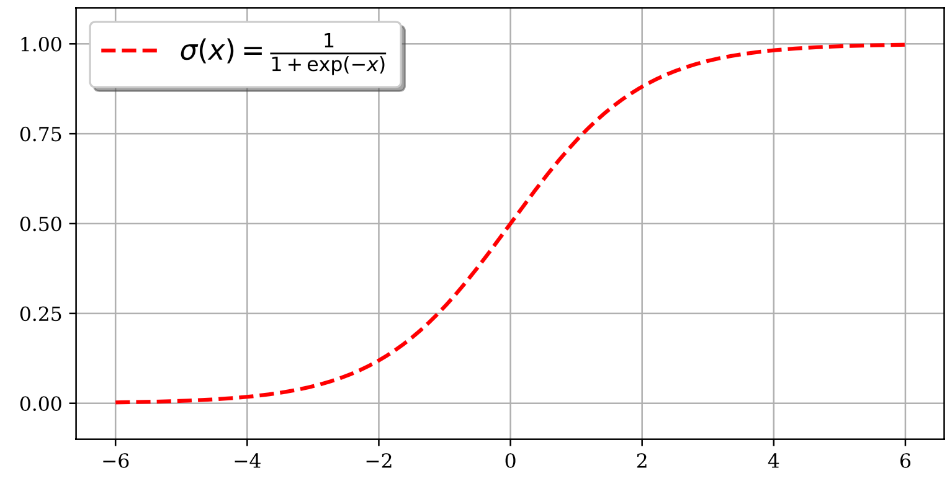
Hyperbolic tangent function



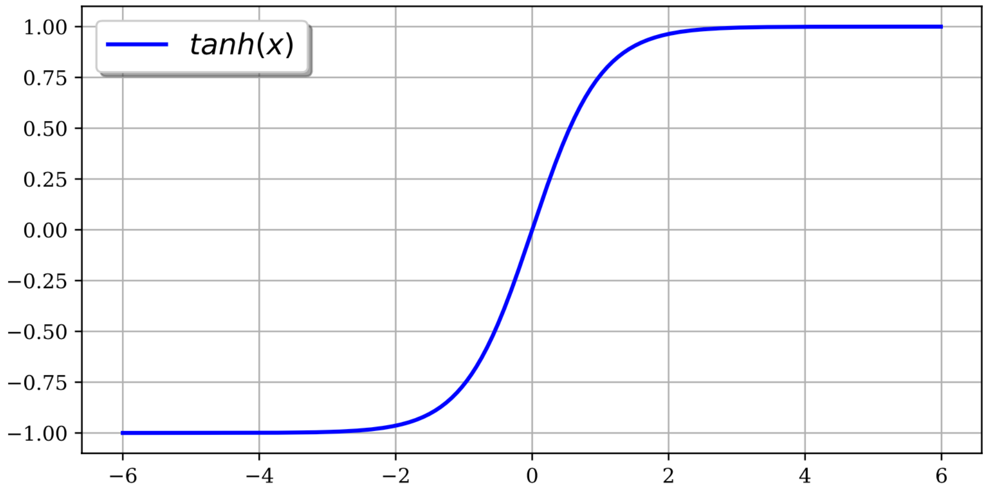
Activation Functions

- 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

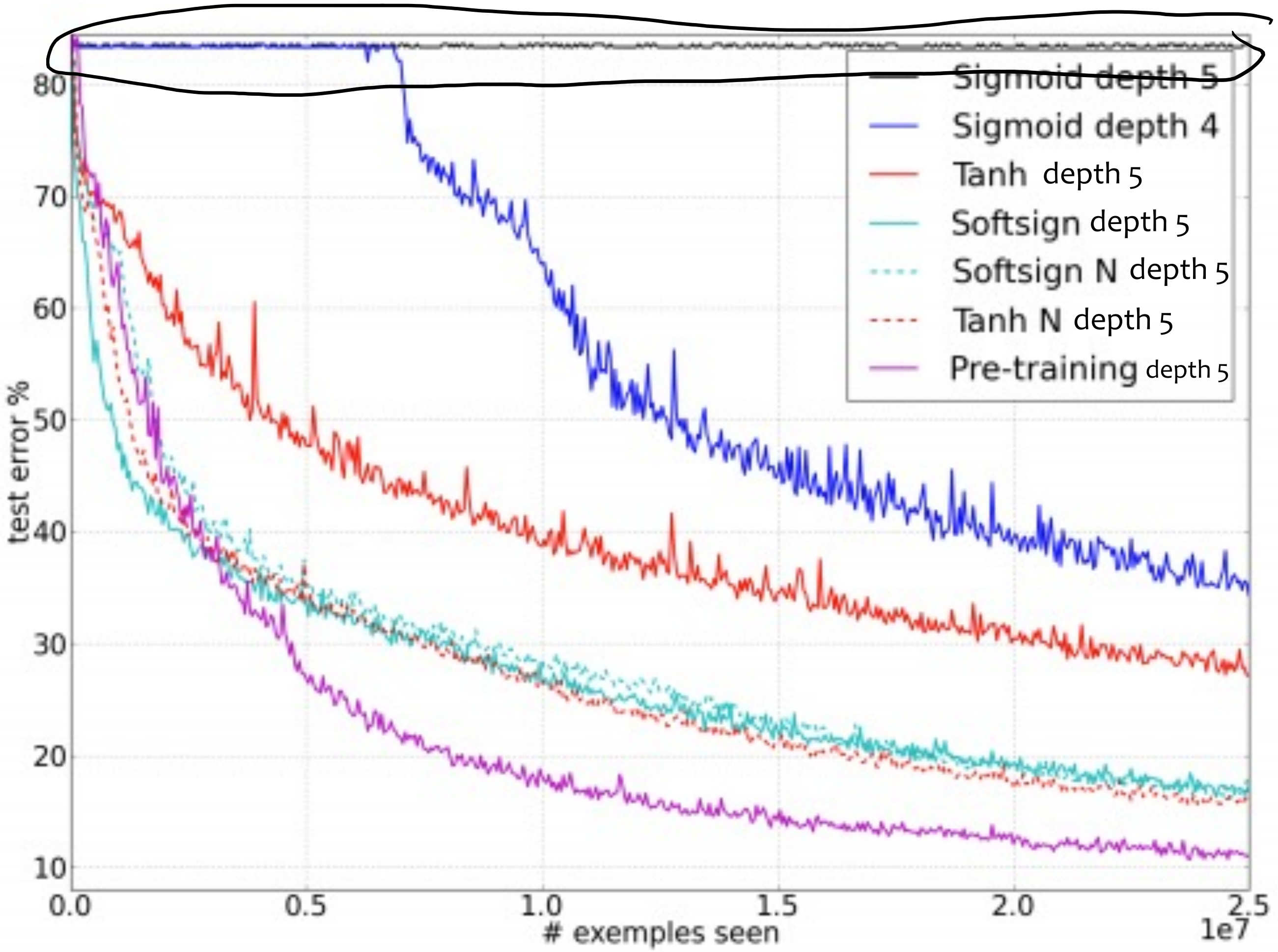


Hyperbolic tangent function



Understanding the difficulty of training deep feedforward neural networks

AI Stats 2010



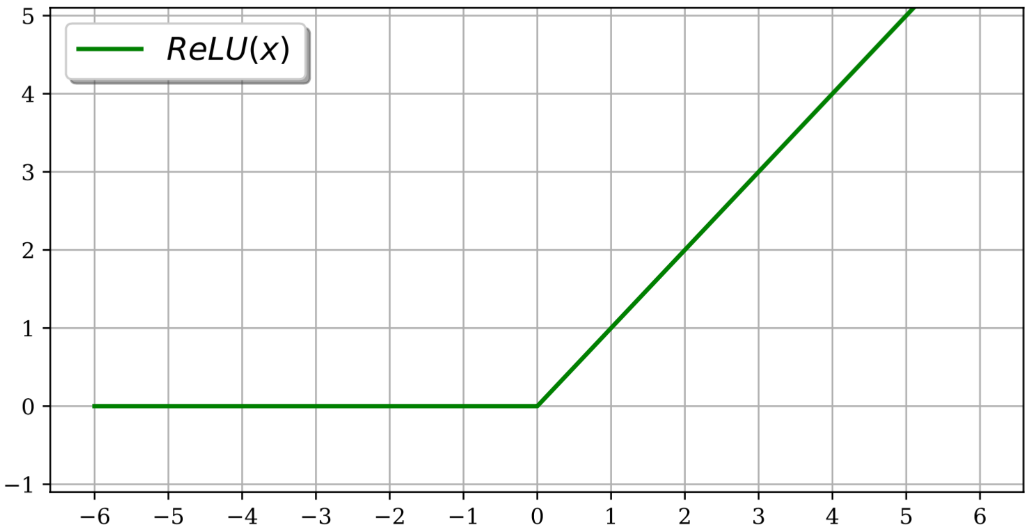
sigmoid
vs.
tanh

Figure from Glorot & Bentio (2010)

Activation Functions

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

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



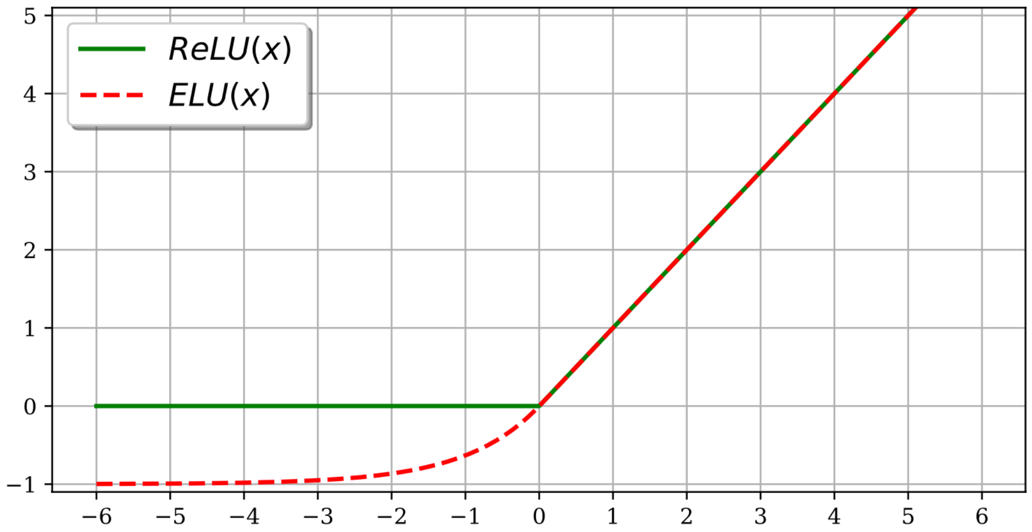
Activation Functions

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

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

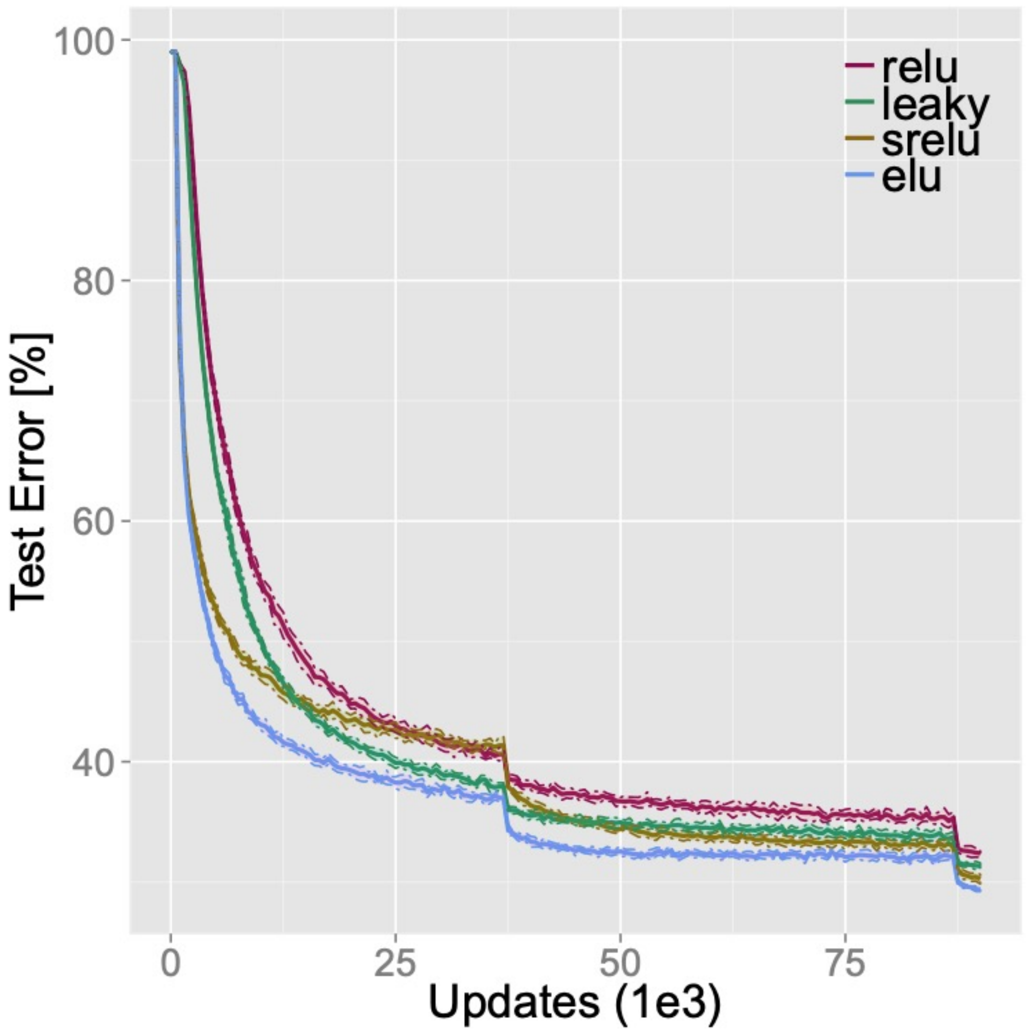
- Exponential Linear Unit (ELU)
  - same as ReLU on positive inputs
  - unlike ReLU, allows negative outputs and smoothly transitions for x < 0

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



Activation Functions

Image Classification Benchmark (CIFAR-10)

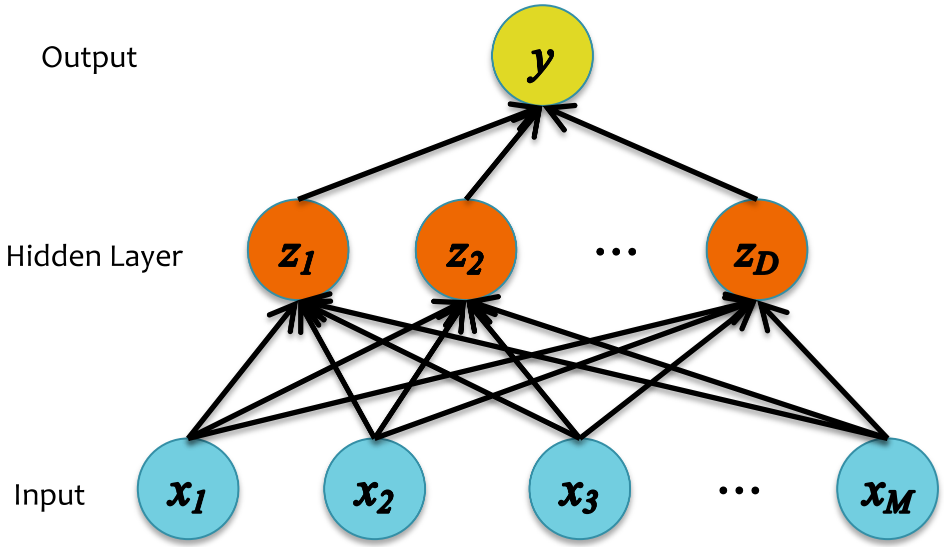


1. Training loss converges fastest with ELU
2. ELU(x) yields lower test error than ReLU(x) on CIFAR-10

Figure from Clevert et al. (2016)

LOSS FUNCTIONS & OUTPUT LAYERS

Neural Network for Classification



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

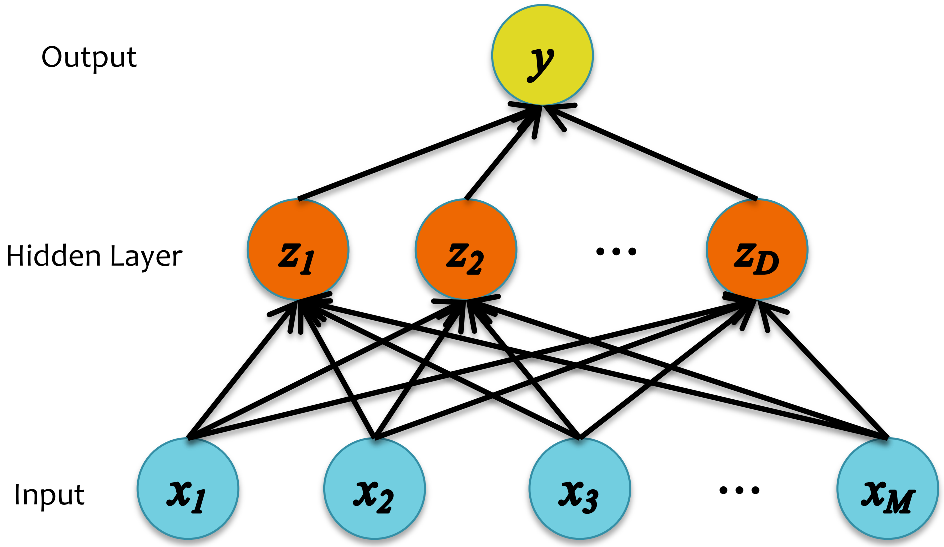
(D) Output (linear)
$b = \sum_{j=0}^{D} \beta_j z_j$

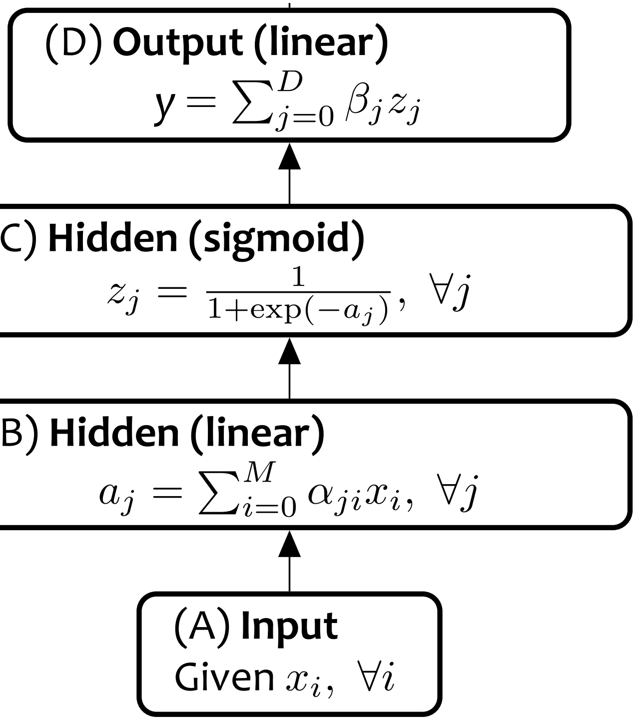
(C) Hidden (sigmoid)
$z_j = \frac{1}{1+\exp(-a_j)}, \ \forall j$

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

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

Neural Network for Regression





Objective Functions for NNs

1. Quadratic Loss:
   - the same objective as Linear Regression
   - i.e. mean squared error

$$J = \ell_Q(y, y^{(i)}) = \frac{1}{2}(y - y^{(i)})^2$$

$$\frac{dJ}{dy} = y - y^{(i)}$$

2. Binary Cross-Entropy:
   - the same objective as Binary Logistic Regression
   - i.e. negative log likelihood
   - This requires our output y to be a probability in [0,1]

$$J = \ell_{CE}(y, y^{(i)}) = -(y^{(i)} \log(y) + (1 - y^{(i)}) \log(1 - y))$$

$$\frac{dJ}{dy} = -\left(y^{(i)} \frac{1}{y} + (1 - y^{(i)}) \frac{1}{y - 1}\right)$$

Objective Functions for NNs

Cross-entropy vs. Quadratic loss

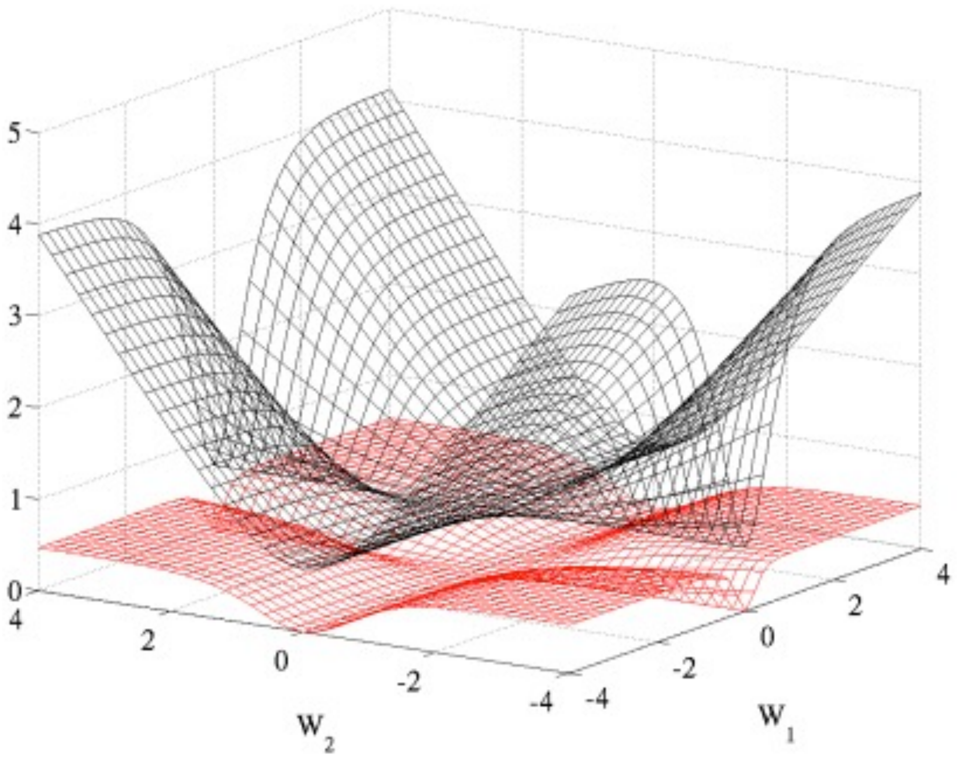
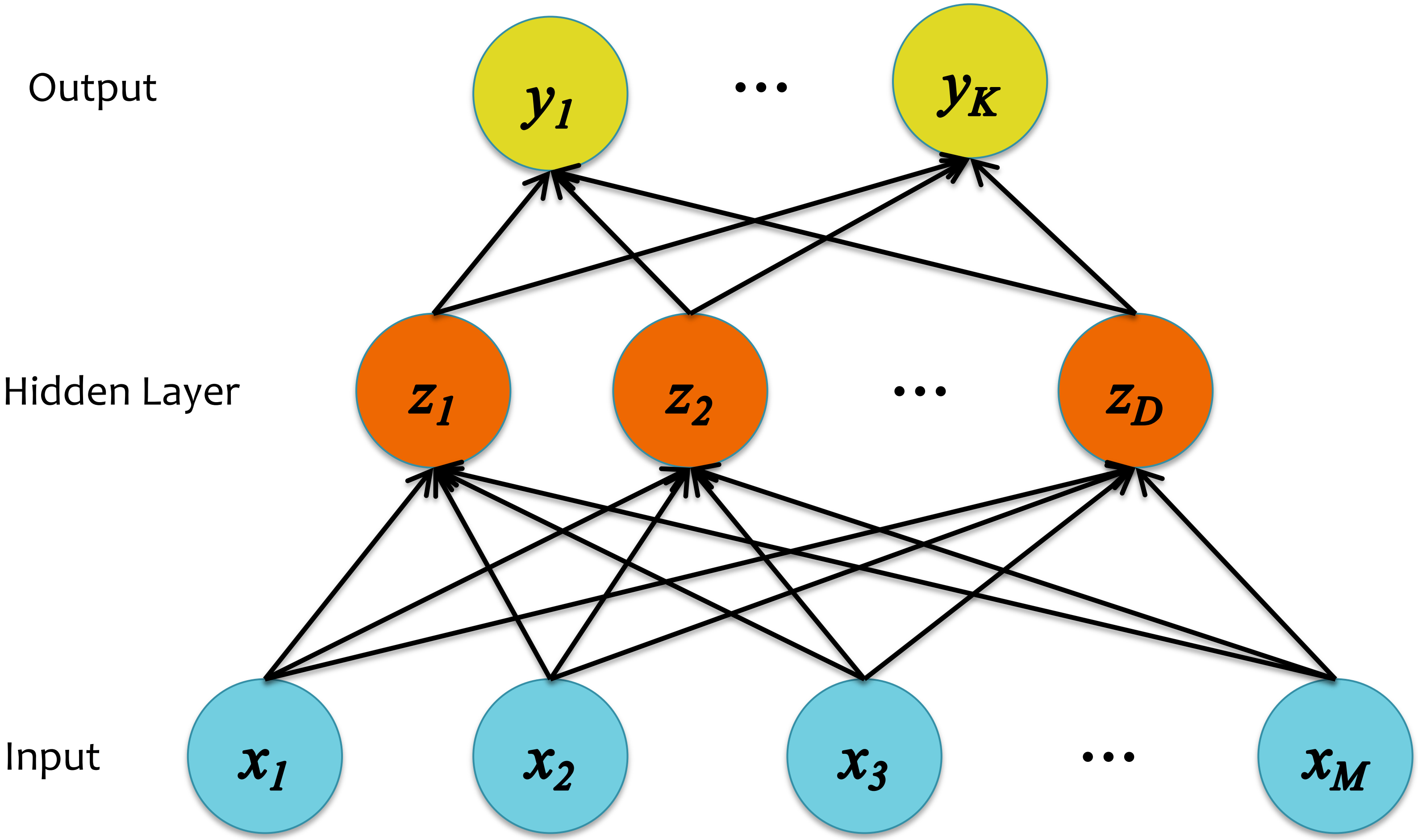


Figure 5: *Cross entropy (black, surface on top) and quadratic (red, bottom surface) cost as a function of two weights (one at each layer) of a network with two layers, W_1 respectively on the first layer and W_2 on the second, output layer.*

Figure from Glorot & Bentio (2010)

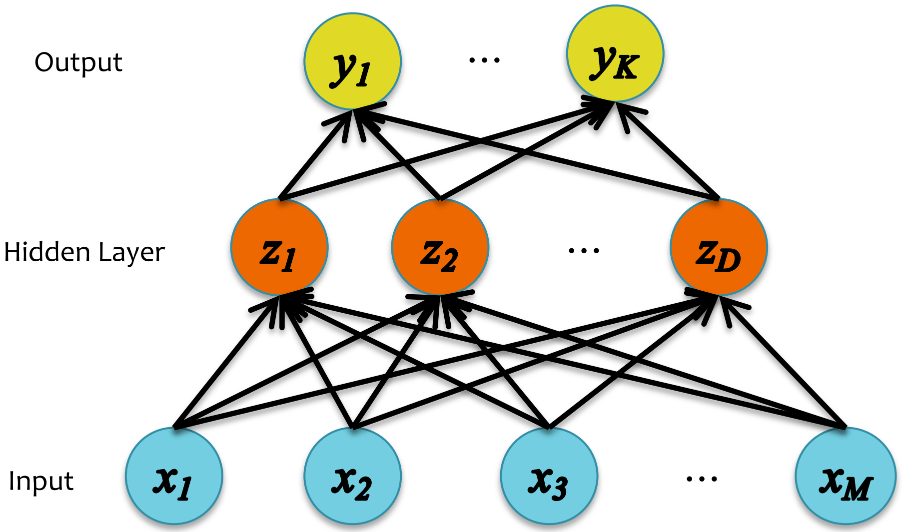
Multiclass Output



Multiclass 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 $x_i, \;\; \forall i$

Objective Functions for NNs

3. Cross-Entropy for Multiclass Outputs:
   - i.e. negative log likelihood for multiclass outputs
   - Suppose output is a random variable Y that takes one of K values
   - Let $\mathbf{y}^{(i)}$ represent our true label as a one-hot vector:

$\mathbf{y}^{(i)}$ =

0	0	0	1	0	0	…	0
1	2	3	4	5	6	…	K

   - Assume our model outputs a length K vector of probabilities:

$$\mathbf{y} = \text{softmax}(f_{\text{scores}}(\mathbf{x}, \boldsymbol{\theta}))$$

   - Then we can write the log-likelihood of a single training example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ as:

$$J = \ell_{CE}(\mathbf{y}, \mathbf{y}^{(i)}) = -\sum_{k=1}^{K} y_k^{(i)} \log(y_k)$$

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