ENGR-E 511; ENGR-E 399

Machine Learning for Signal Processing

Module 06:

Neural Networks

Minje Kim

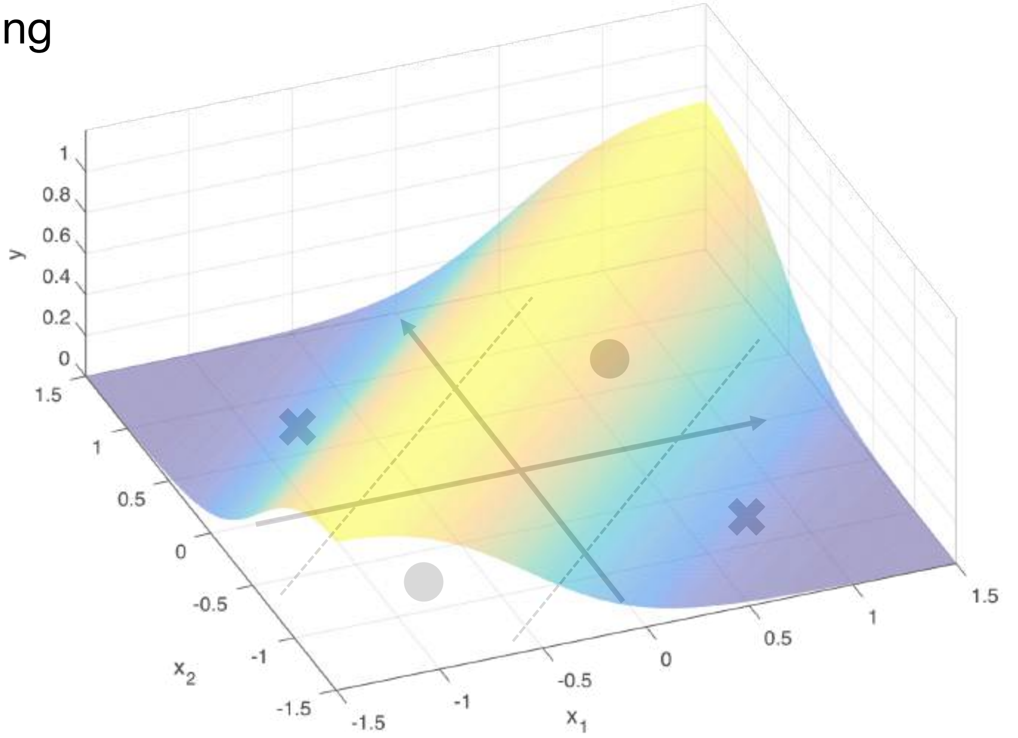
Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje



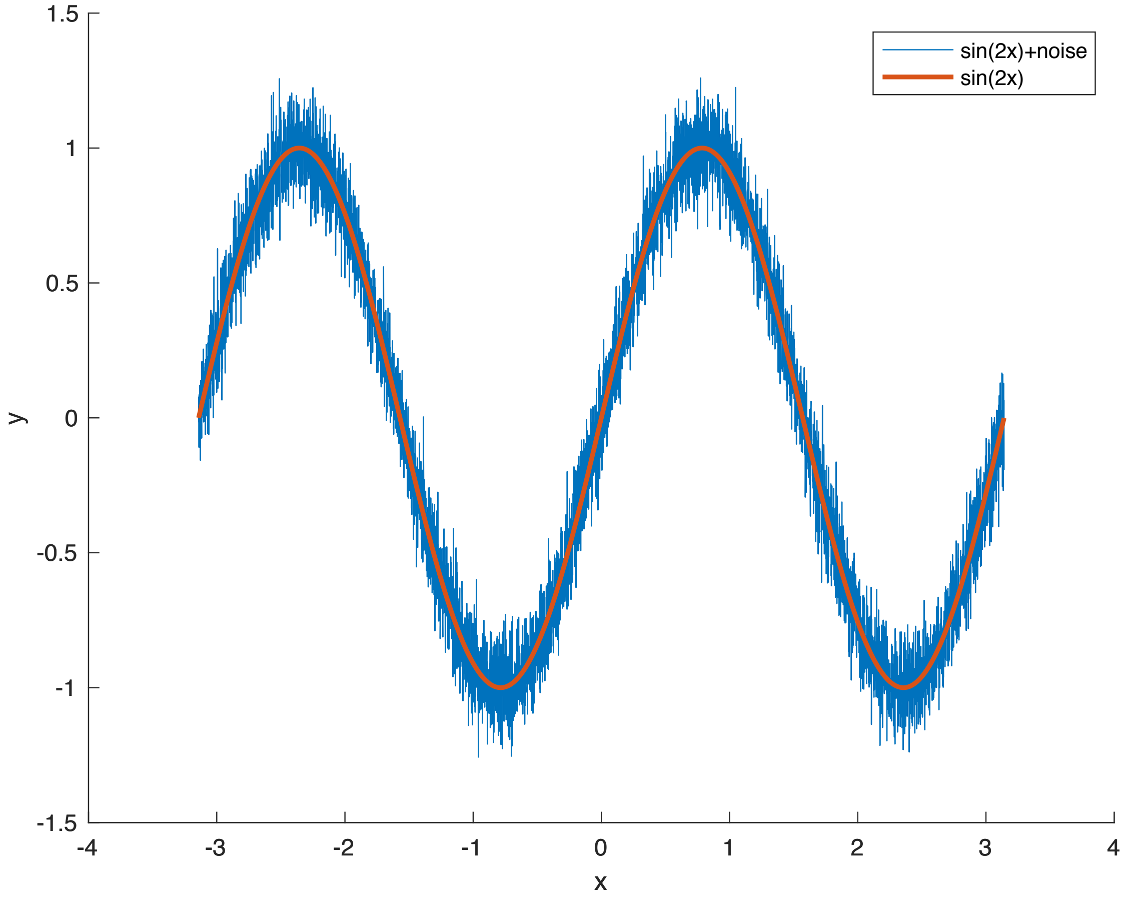
INDIANA UNIVERSITY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

Denoising Revisited

- A sine wave

- I observe a noisy signal (as always)
- I'm pretty sure that it is a sine wave
  - So I think my job is to find out its period



Denoising Revisited

- A sine wave

- We all are masters of optimization by now
  - Let's formulate this as an optimization problem
- I start from a set of data points

$$[x_1, x_2, \cdots, x_N]^\top \qquad [y_1, y_2, \cdots, y_N]^\top$$

  - I don't know a lot about this data set, but I feel that there's a relationship

$$y_i \approx \sin(\omega x_i)$$

- The goal is to find ω that minimizes the error

$$\mathcal{J} = \sum_i (y_i - \sin(\omega x_i))^2$$

  - Is this a convex function?

$$\frac{\partial \mathcal{J}}{\partial \omega} = -2 y_i x_i \cos(\omega x_i) + 2 x_i \sin(\omega x_i) \cos(\omega x_i)$$

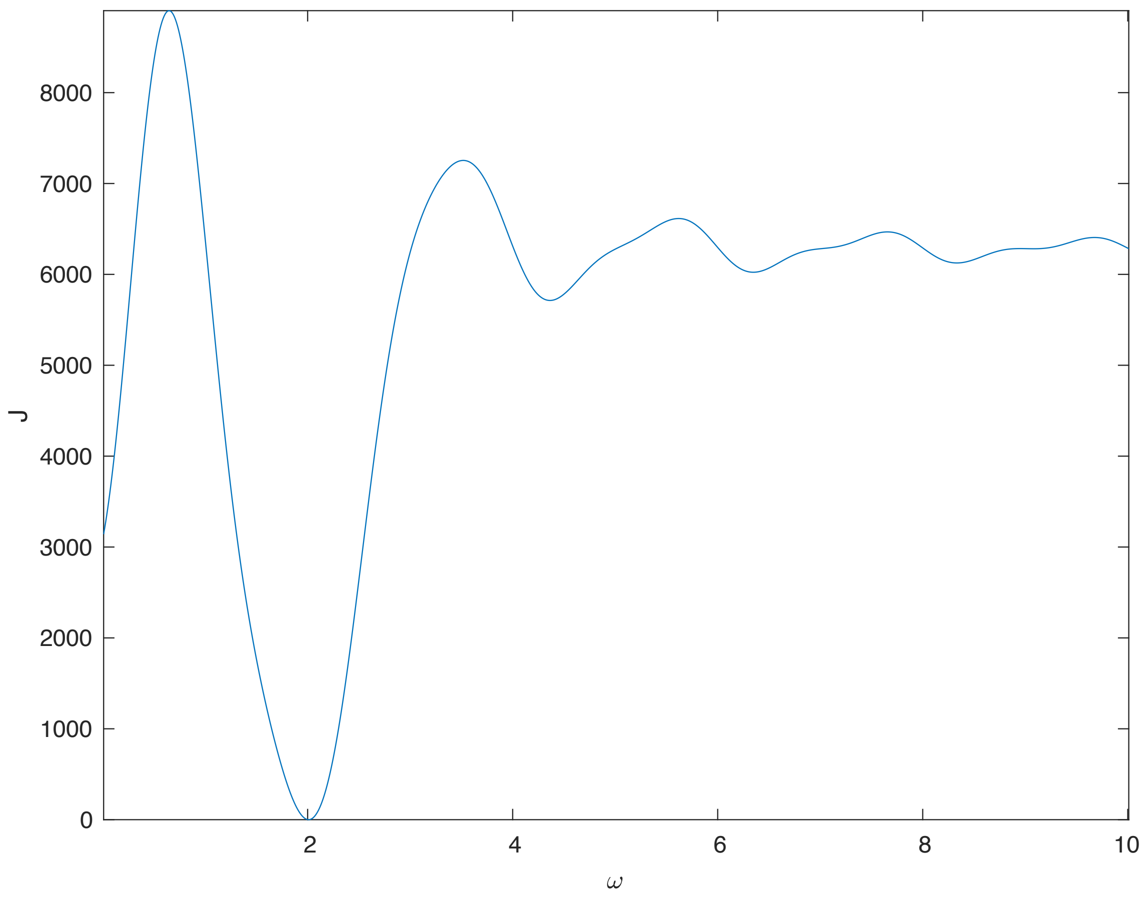
$$\frac{\partial \mathcal{J}^2}{\partial \omega^2} = 2 y_i x_i^2 \sin(\omega x_i) + 2 x_i^2 \cos(\omega x_i) \cos(\omega x_i) - 2 x_i^2 \sin(\omega x_i) \sin(\omega x_i)$$

    - Maybe not

Denoising Revisited

- A sine wave

- $\mathcal{J}$ as a function of ω
  - Too complicated, and I don't know what to do



Denoising Revisited

- A workaround: feature transform

- Even if we knew the ground truth (family of) function we are estimating, we give up
  - because a sine wave is too difficult for us to handle
- Instead, we're going to assume another trackable function

$$y_i \approx f(x_i; \boldsymbol{\Theta})$$

  - Any candidate family of functions?
    - That might be able to replace a piece of sine wave?
  - I'd use a 5th order polynomial function

$$y_i \approx f(x_i; \boldsymbol{\Theta}) = a_5 x_i^5 + a_4 x_i^4 + a_3 x_i^3 + a_2 x_i^2 + a_1 x_i + a_0$$

    - As there are 4 stationary points, it looks like a sine wave
- The new error function

$$\arg\min_{\boldsymbol{\Theta}} \sum_i \mathcal{E}(y_i || f(x_i; \boldsymbol{\Theta}) = \arg\min_{a_5, a_4, a_3, a_2, a_1, a_0} \sum_i \Big(y_i - (a_5 x_i^5 + a_4 x_i^4 + a_3 x_i^3 + a_2 x_i^2 + a_1 x_i + a_0) \Big)^2$$

  - Still look complicated?

Denoising Revisited

- A workaround: feature transform

- Let's form a new data matrix

$$\boldsymbol{X} = \begin{bmatrix} x_1^5 & x_2^5 & \cdots & x_N^5 \\ x_1^4 & x_2^4 & \cdots & x_N^4 \\ & & \vdots & \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

$$x_i^j$$

Polynomial order (j)

Sample index (i)

  - Then, we can rewrite the function in a linear form

$$\boldsymbol{y}^\top \approx f(\boldsymbol{x};\boldsymbol{a}) = \boldsymbol{a}^\top \boldsymbol{X} = [a_5 \ a_4 \ a_3 \ a_2 \ a_1 \ a_0] \begin{bmatrix} x_1^5 & x_2^5 & \cdots & x_N^5 \\ x_1^4 & x_2^4 & \cdots & x_N^4 \\ & & \vdots & \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

- You can think of this procedure in two ways:
  - **Approximating a function with another tractable, yet non-linear function**
  - **Converting the scalar input into a high-dim feature set and solve linearly**

Denoising Revisited

- A workaround: feature transform

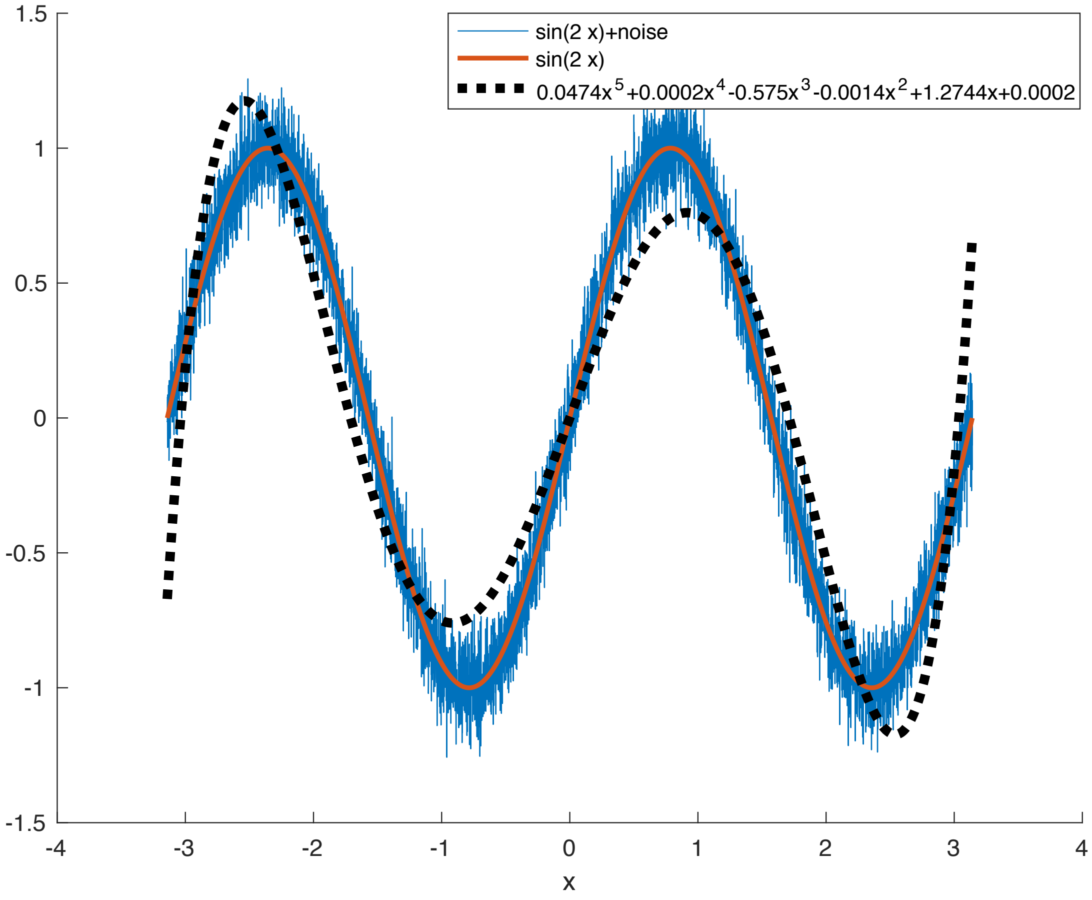
- Now the error function is quadratic

$$\arg\min_{\boldsymbol{a}} (\boldsymbol{y}^\top - \boldsymbol{a}^\top \boldsymbol{X})(\boldsymbol{y}^\top - \boldsymbol{a}^\top \boldsymbol{X})^\top$$

  - There's an analytic solution and we know how to solve it (see M01 L03)
  - Long story short…

$$\boldsymbol{a}^* = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\boldsymbol{y}$$

$$= [0.047, 0.0002, -0.575, -0.0014, 1.27, 0.0002]^\top$$



Some Questions
- Are we good?

- What if the signal is very long?
  - Many stationary points → polynomials with a high degree
- What if the signal looks complicated?
  - Polynomials don't make sense anymore
- Is there any parametric function that can potentially approximate any complicated target functions?
  - Universal approximation
  - This is our goal in learning neural networks
- Hint: we converted the data and solved it linearly
  - We'll cover this part in the nonlinear modeling lecture
  - But, long story short, neural networks can do this job anyway
- What's the difference between this denoising procedure and classification?
  - We'll cover it later in this lecture
  - But, long story short, they are not that different

Perceptron

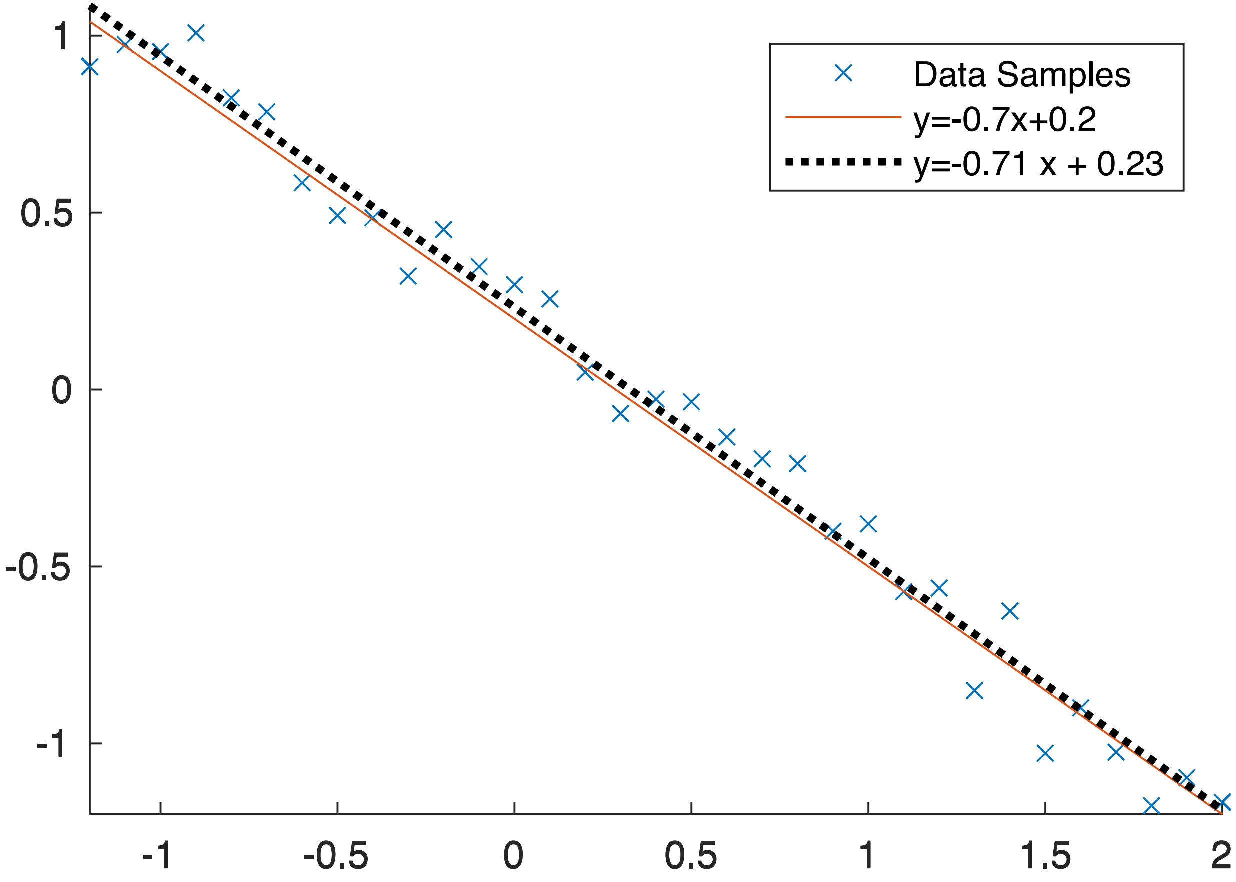
- Linear line fitting

- Let me solve a very simple regression problem

$$\boldsymbol{a} = [-0.7, 0.2]^\top$$

- The LMS solution:

$$\boldsymbol{a}^* = (\boldsymbol{X}\boldsymbol{X}^\top)^{-1}\boldsymbol{X}\boldsymbol{y}$$
$$= [-0.71, 0.23]^\top$$



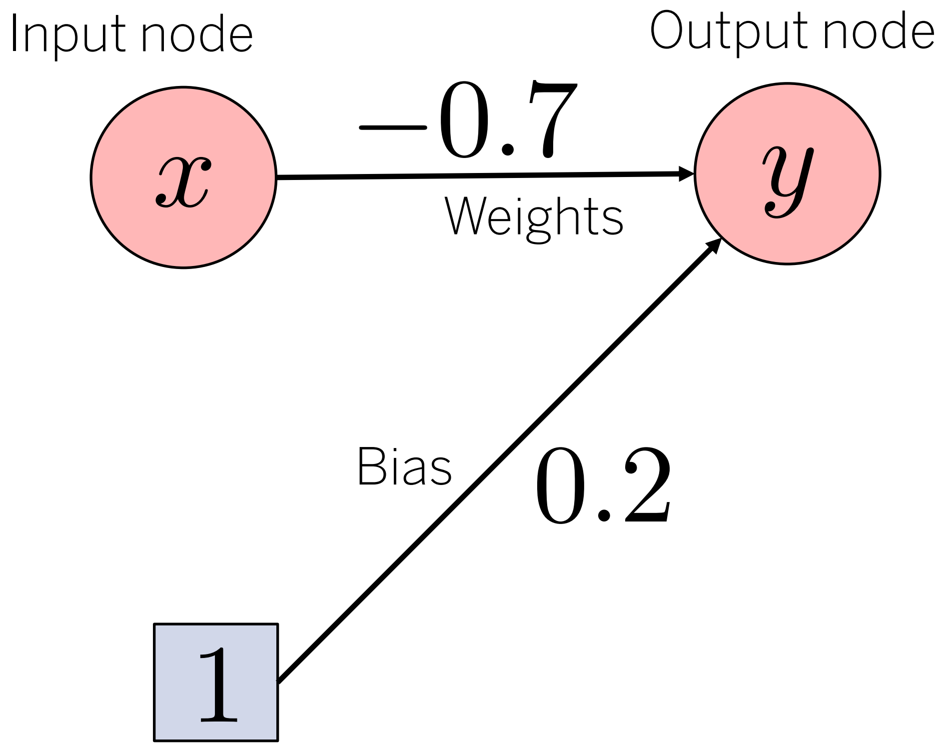
Perceptron

- Linear line fitting

- You can represent this as a network

$$y_t = [a_1 \quad a_0] \cdot \begin{bmatrix} X_{(2,t)} \\ X_{(1,t)} \end{bmatrix} \leftarrow 1$$

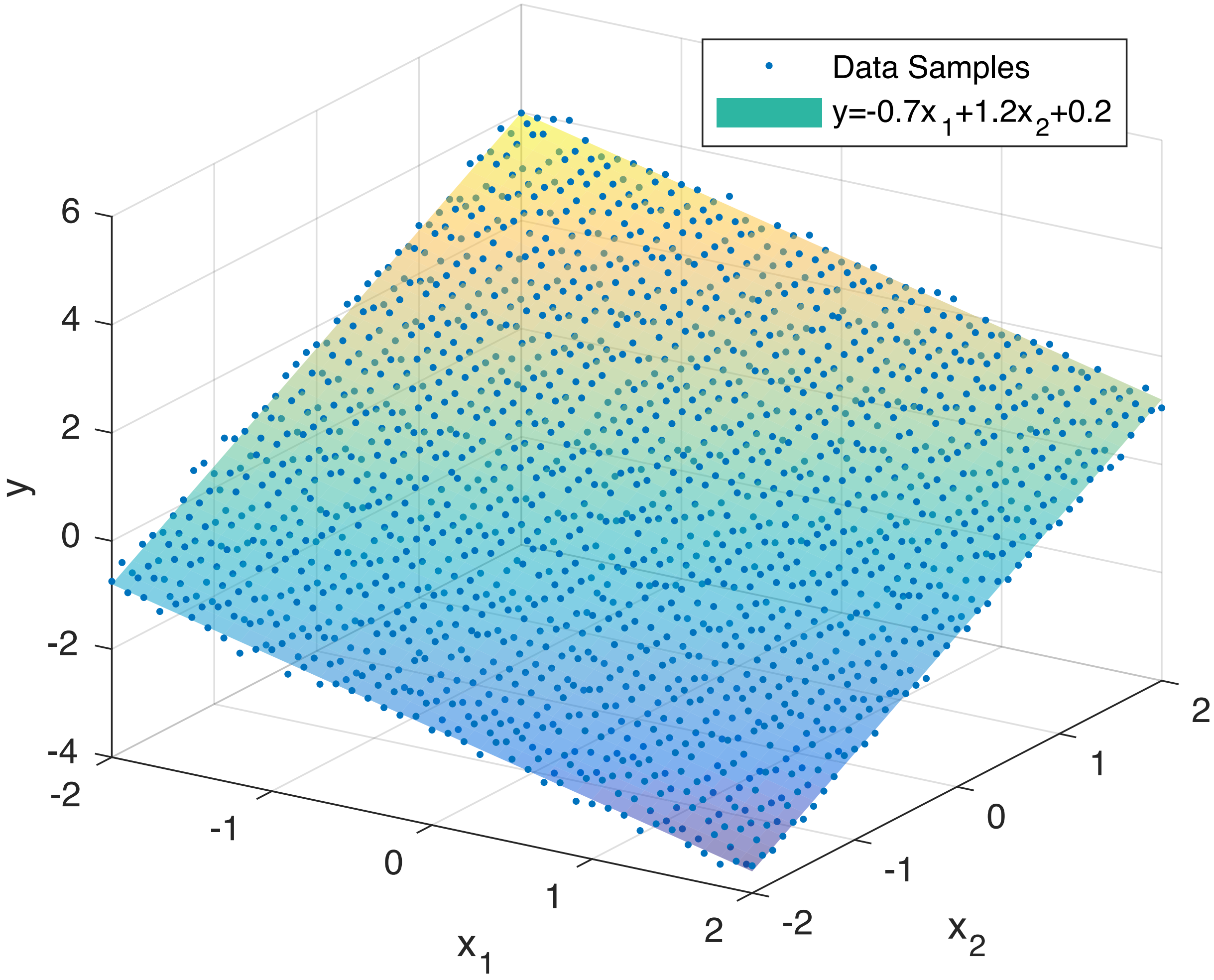
$$\boldsymbol{y} = \boldsymbol{a}^\top \boldsymbol{X}$$



Perceptron

- Linear surface

- Two input dimensions? (fig)



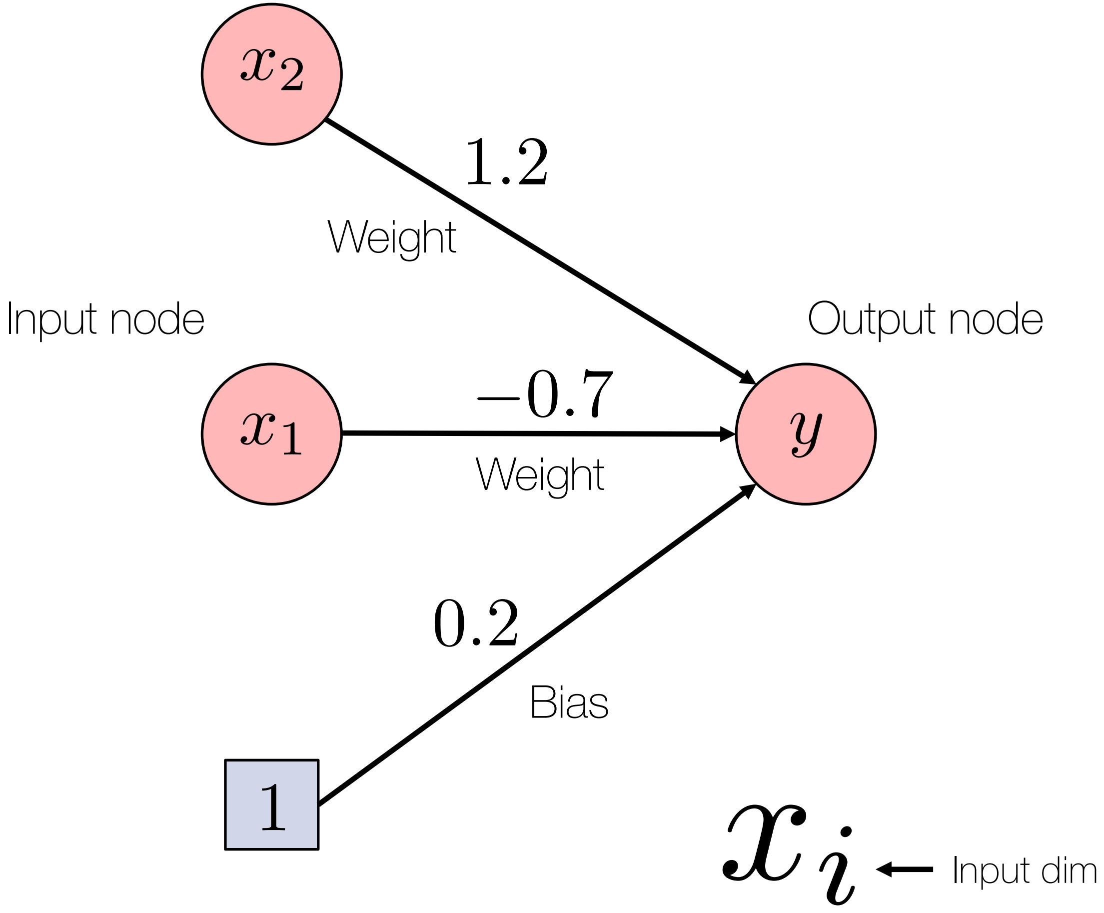
Perceptron

- Linear surface

- You can represent this as a network as well

$$y_t = [a_2 \quad a_1 \quad a_0] \cdot \begin{bmatrix} X_{(2,t)} \\ X_{(1,t)} \\ 1 \end{bmatrix}$$

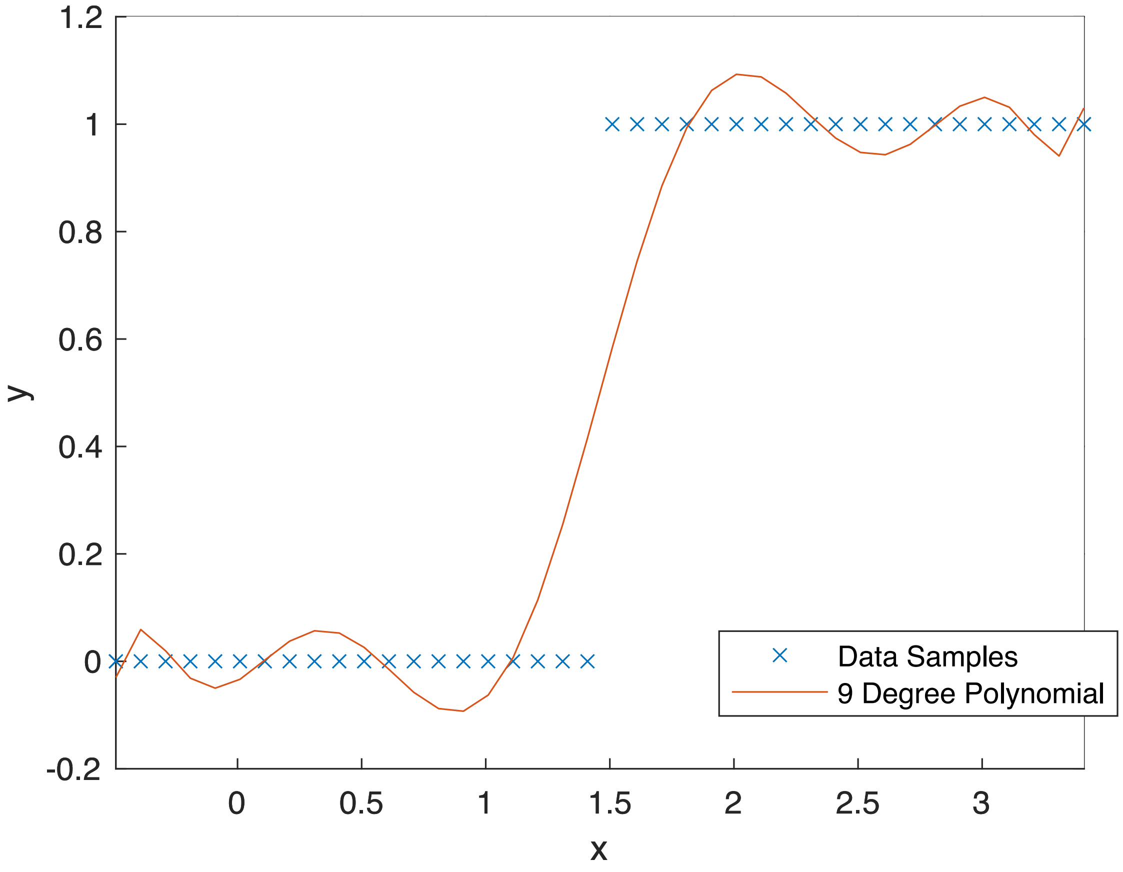
$$\boldsymbol{y} = \boldsymbol{a}^\top \boldsymbol{X}$$



Decision Making with Perceptrons

- Nonlinear?

- To remind you, we started from a set of observations (scalars for now)
- Each observation has a corresponding target variable
- We want to learn the mapping function
- Its not different from the linear line fitting cases
  - but the mapping function shouldn't be a dummy linear function anymore
  - How about another polynomial?

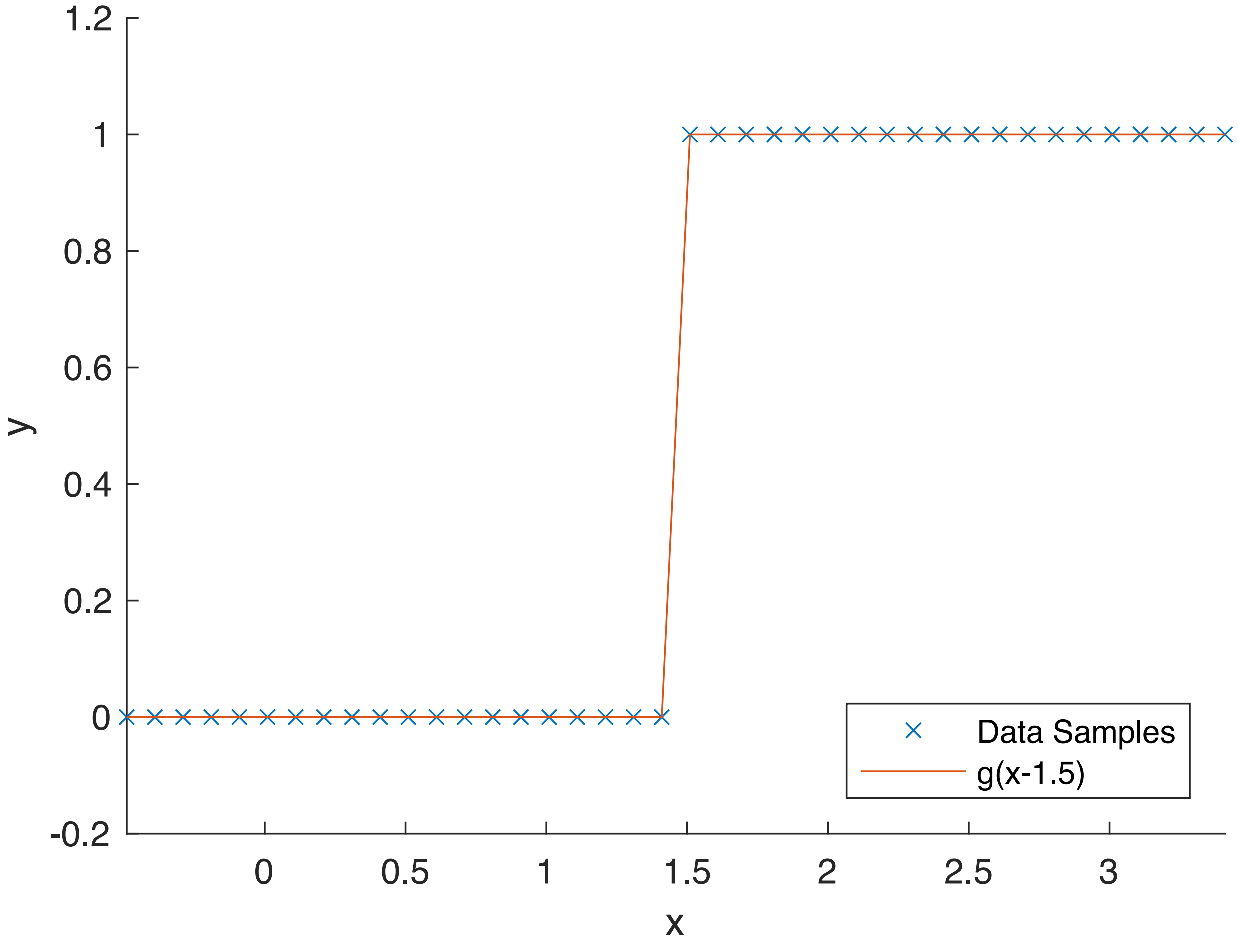


Decision Making with Perceptrons

- Nonlinear?

- A high degree polynomial works to some degree
- But what we see is a STEP!
  - Divided at around 1.5
- So, why don't we fit a step function?

$$g(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

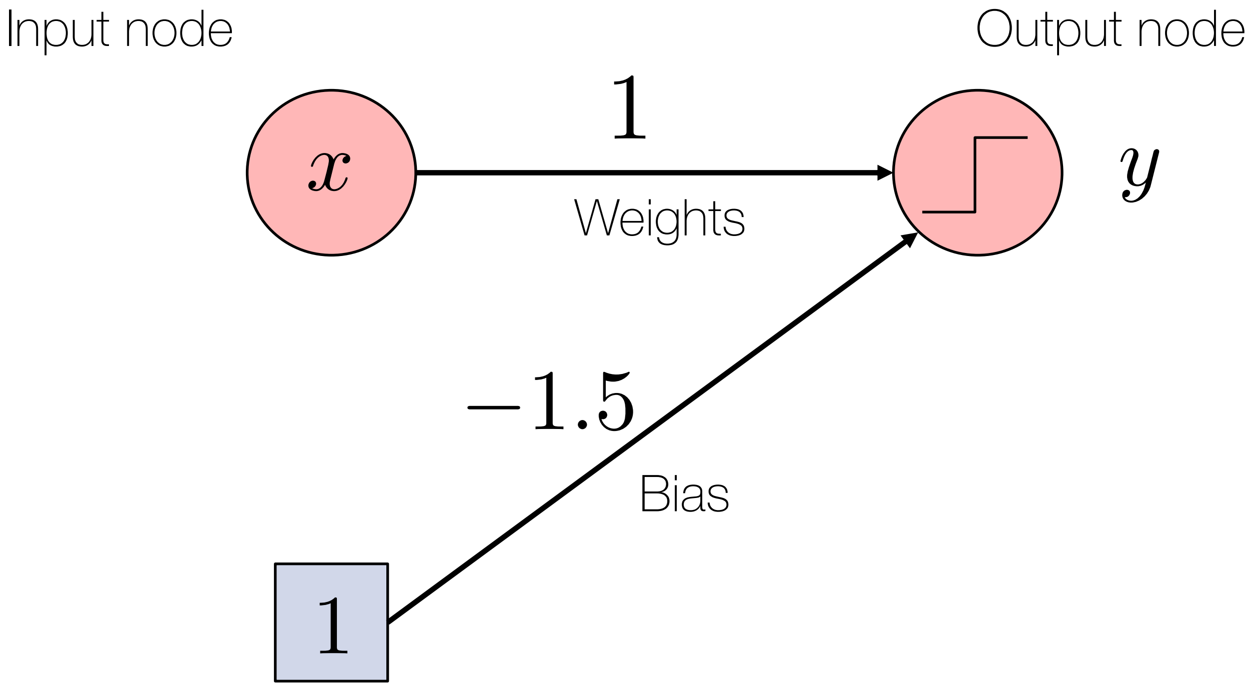


Decision Making with Perceptrons

- Nonlinear?

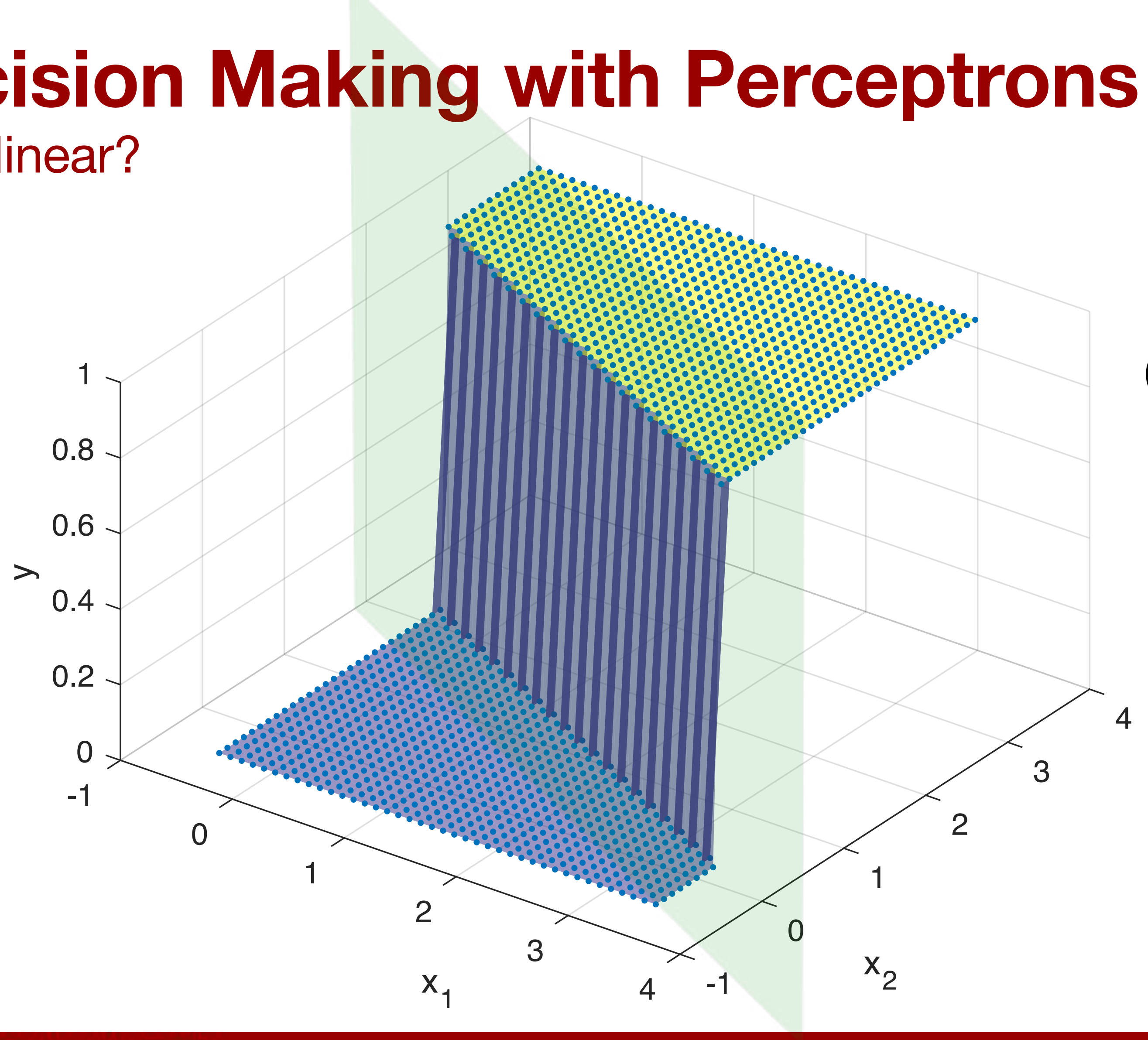
- Once again, you can represent this as a network as well

$$\boldsymbol{y} = g(\boldsymbol{a}^\top \boldsymbol{X})$$



Decision Making with Perceptrons

- Nonlinear?



$$x_2 = -0.5x_1 + 2$$

$$0.5x_1 + x_2 - 2 = 0$$

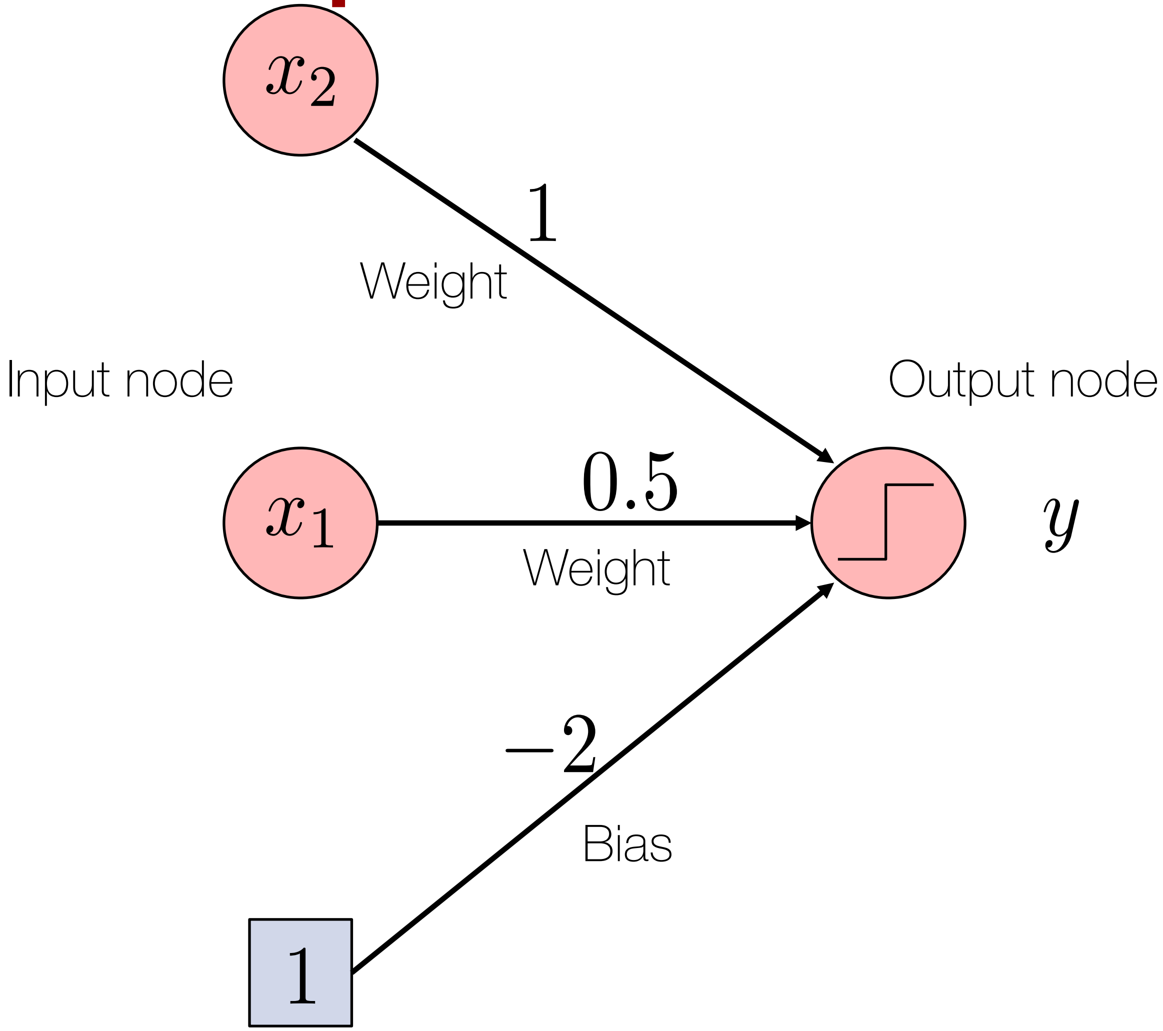
$$g(x_2 + 0.5x_1 - 2)$$

Decision Making with Perceptrons

- Nonlinear?

- Multi-dimensional step function

$$\boldsymbol{y} = g(\boldsymbol{a}^\top \boldsymbol{X})$$



Decision Making with Perceptrons

- Step functions

- I didn't tell you specifically, but
- The examples with step functions were for classification
  - e.g. This can be seen as a hyperplane

    $$x_2 = -0.5x_1 + 2$$

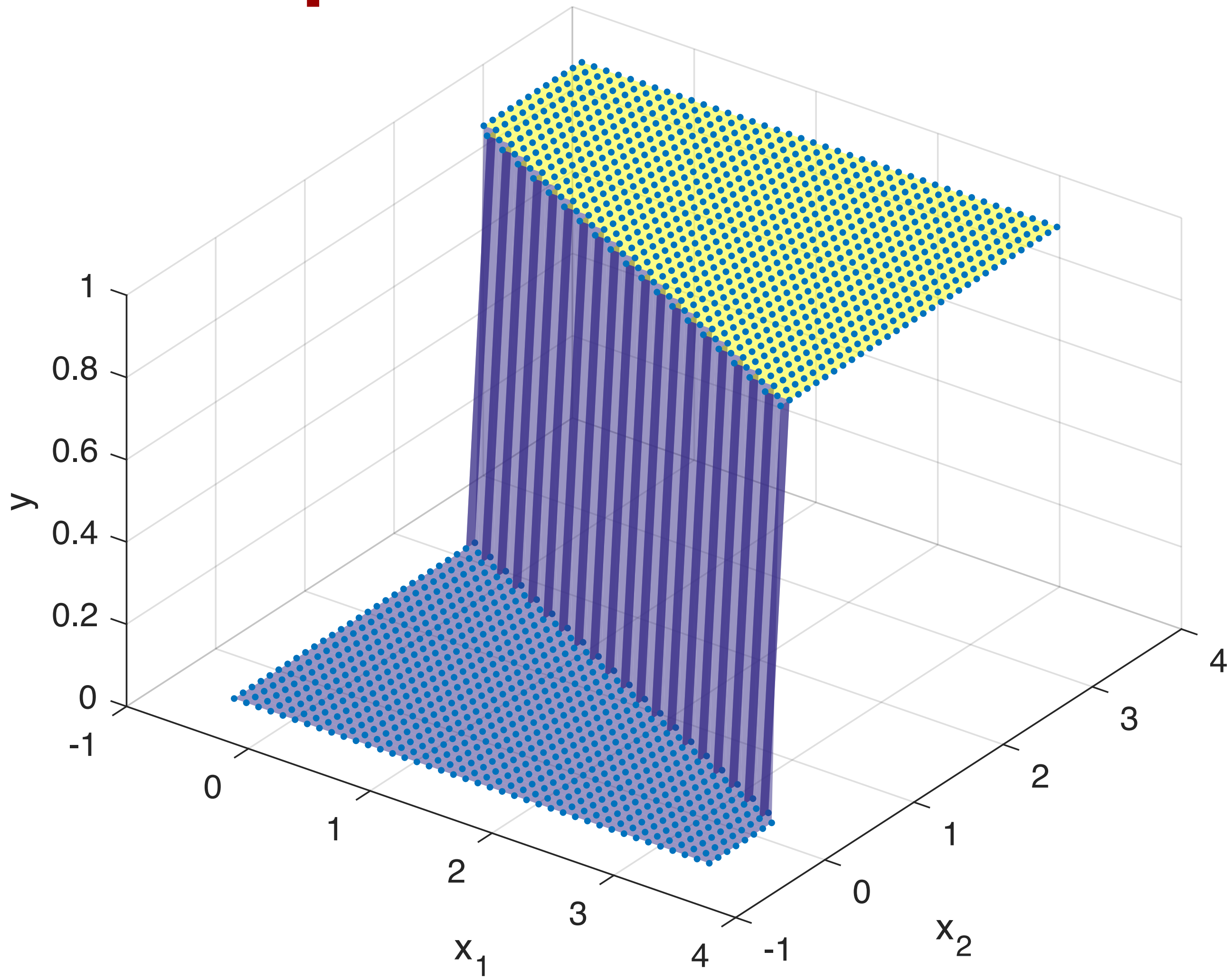
  - Class 1 $\quad \text{if } x_2 \geq -0.5x_1 + 2$

    $$\leftrightarrow g(x_2 + 0.5x_1 - 2) = 1$$

  - Class 2 $\quad \text{if } x_2 < -0.5x_1 + 2$

    $$\leftrightarrow g(x_2 + 0.5x_1 - 2) = 0$$

- Classification is not different from another function approximation
  - Except for the fact that the model tries to approximate a step function



Logistic Regression
- Step functions

- The data for training
  - We start from a set of data samples, a matrix $\boldsymbol{X} \in \mathbb{R}^{(D+1)\times N}$
  - We have another corresponding labeling information $\boldsymbol{y} \in \mathbb{R}^{1\times N}$
- The goal of the learning algorithm
  - Estimate the parameters that minimize the error
    - Between the training label and the network output

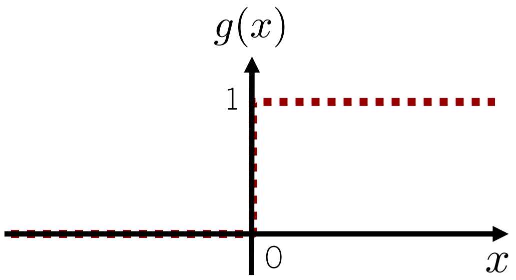
$$\mathcal{E}\left(\boldsymbol{y}||g(\boldsymbol{a}^\top \boldsymbol{X})\right) = \frac{1}{2}\sum_{i=1}^{N}\left(y_i - g\left(\boldsymbol{a}^\top \boldsymbol{X}_{(:,i)}\right)\right)^2 = \frac{1}{2}\left(\boldsymbol{y} - g\left(\boldsymbol{a}^\top \boldsymbol{X}\right)\right)\left(\boldsymbol{y} - g\left(\boldsymbol{a}^\top \boldsymbol{X}\right)\right)^\top$$

- Gradient descent?
  - Partial differentiation w.r.t. the parameters

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{a}} = \boldsymbol{X}\Big(\big(\boldsymbol{y} - g(\boldsymbol{a}^\top \boldsymbol{X})\big) \odot g'(\boldsymbol{a}^\top \boldsymbol{X})\Big)^\top$$

  - **The step function is not differentiable!**

$$g(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$$

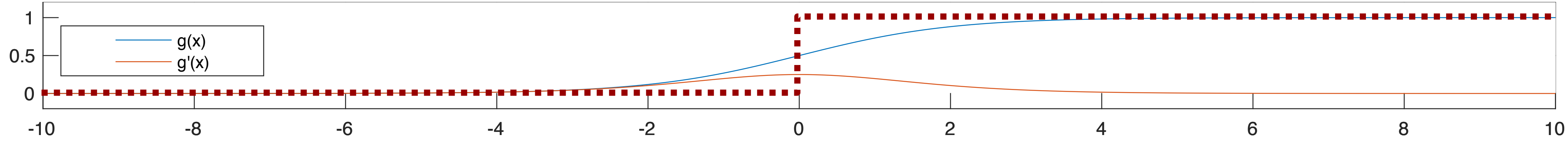


Logistic Regression

- Logistic function

- Step functions seem to be nice for classification, but it doesn't work
  - Because it's not differentiable
- What can we do?
  - Let's soften it up
  - Hint: you've already seen it!
- The logistic function

$$g(x) = \frac{1}{1+e^{-x}} \qquad\qquad g'(x) = g(x)\big(1-g(x)\big)$$



- Gradient descent

Derivative of logistic

$$\frac{\partial \mathcal{E}}{\partial \boldsymbol{a}} = \boldsymbol{X}\Big(\big(\boldsymbol{y}-g(\boldsymbol{a}^\top\boldsymbol{X})\big)\odot g'(\boldsymbol{a}^\top\boldsymbol{X})\Big)^\top = \boldsymbol{X}\Big(\big(\boldsymbol{y}-g(\boldsymbol{a}^\top\boldsymbol{X})\big)\odot \boxed{g(\boldsymbol{a}^\top\boldsymbol{X})\odot\big((1-g(\boldsymbol{a}^\top\boldsymbol{X})\big)}\Big)^\top$$

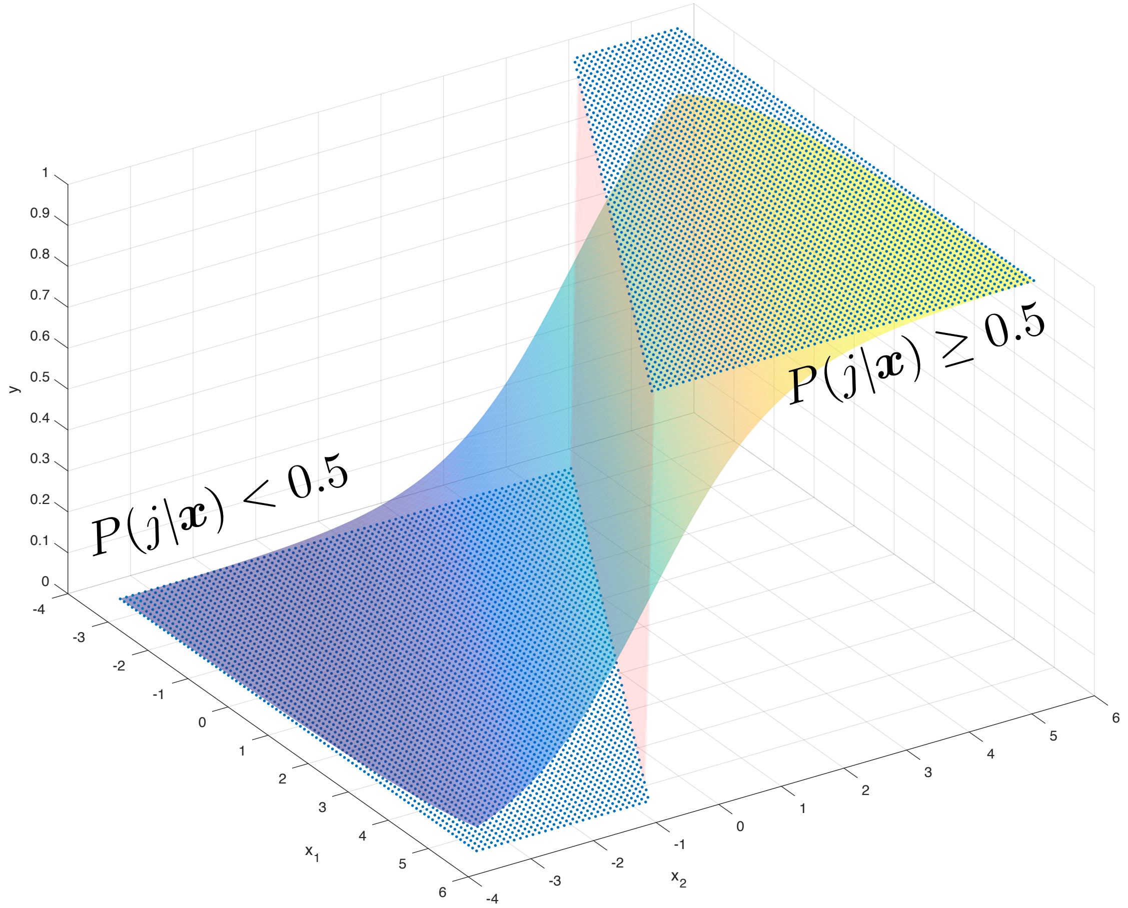
Logistic Regression

- Logistic function

- Now the decision is soft
  - and it works
- What the curved blanket says:
  - Corresponds to the posterior probability
- The decision boundary:

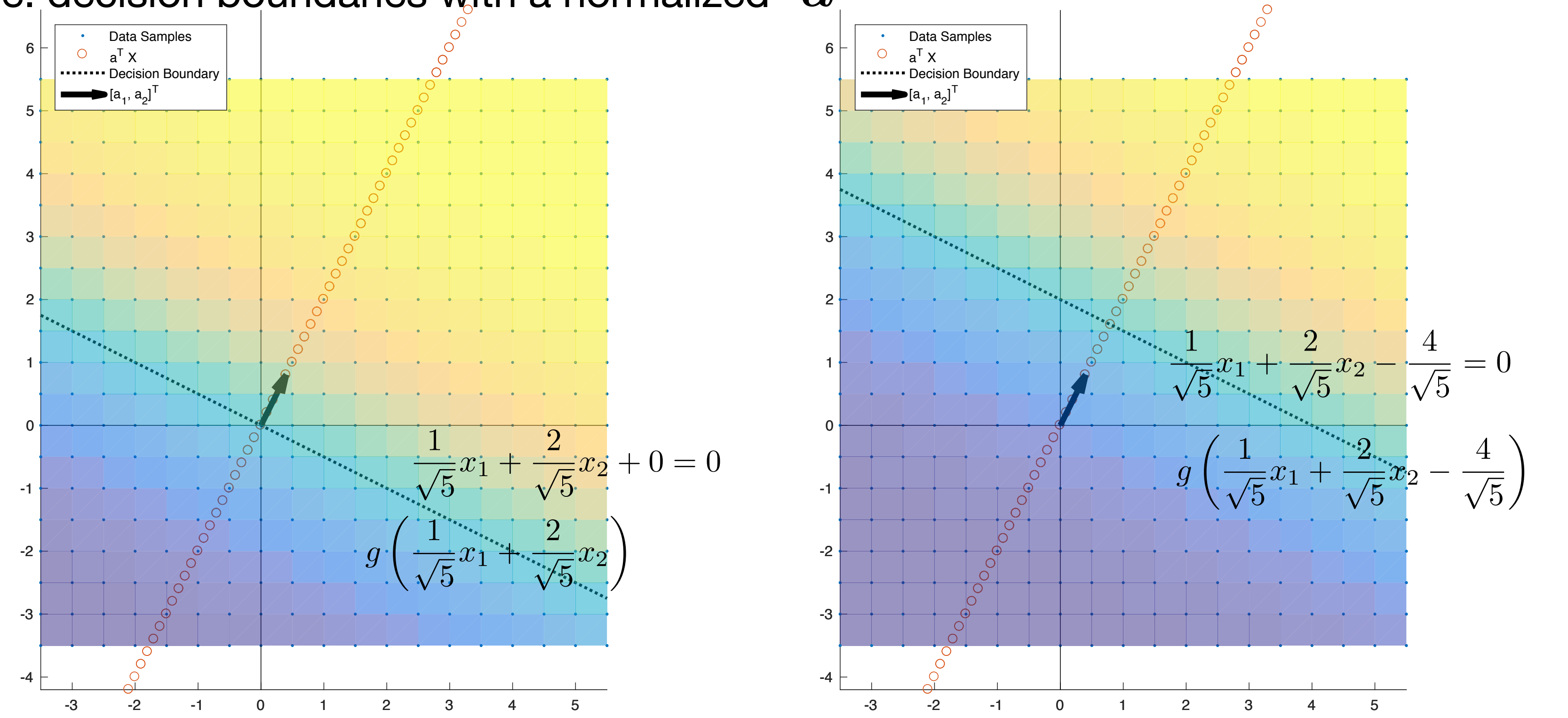
$$\log P(j|\boldsymbol{x}) = \log P(k|\boldsymbol{x})$$

$$g(x_2 + 0.5x_1 - 2) = 0.5$$



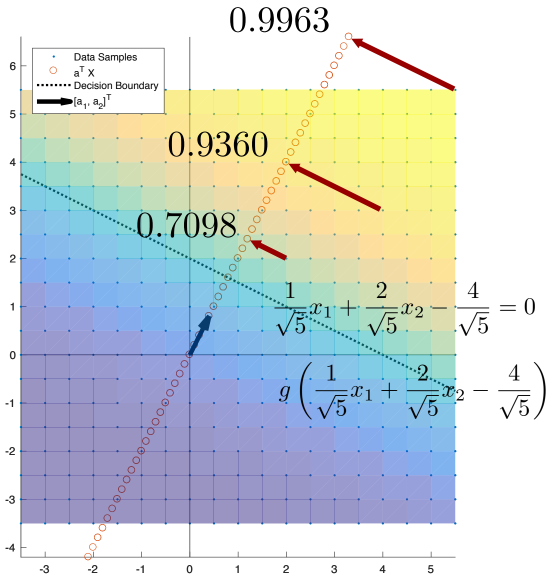
Logistic Regression
- Analysis of decision boundaries

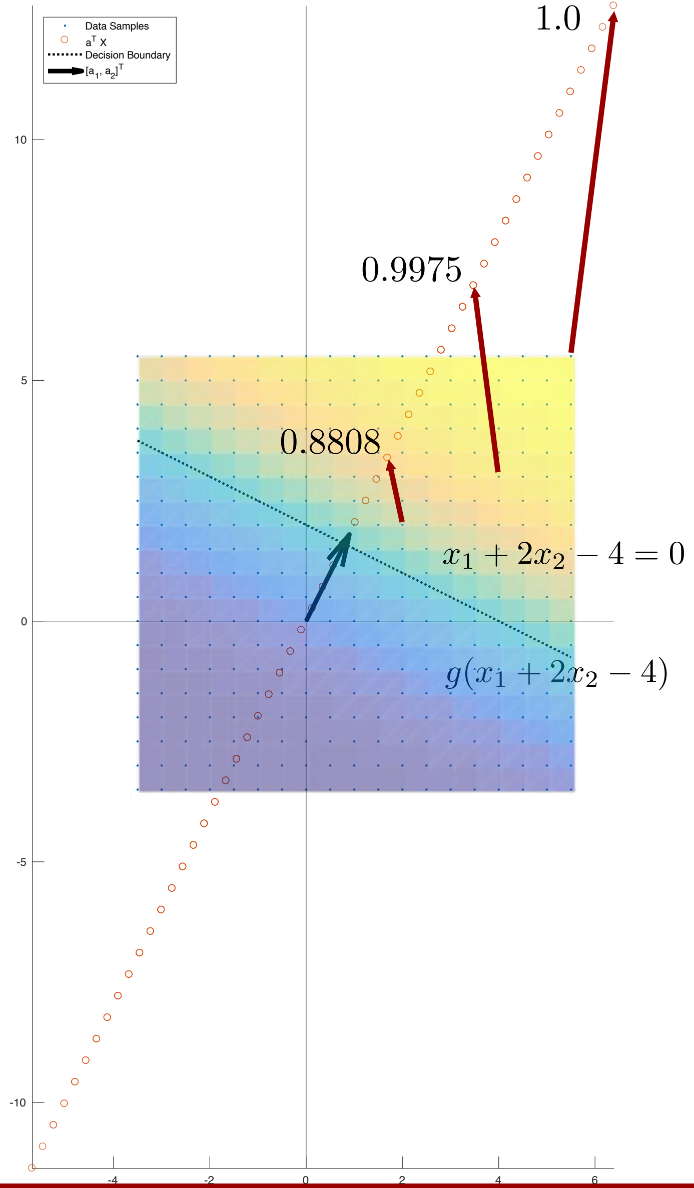
- Need a sharper decision?
  - It has something to do with your $\boldsymbol{a}$
- First example: decision boundaries with a normalized $\boldsymbol{a}$



Logistic Regression

- Analysis of decision boundaries





- Scaling $\boldsymbol{a}$ doesn't change the decision boundary

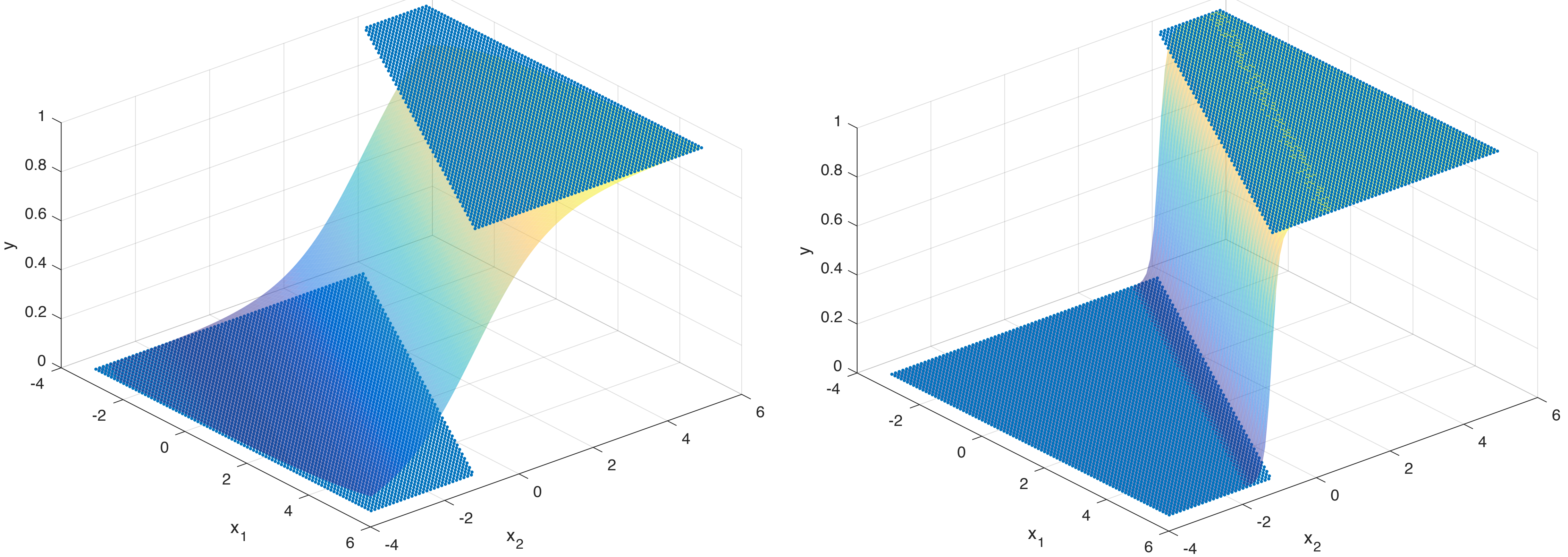
$$1x_1 + 2x_2 - 4 = 1mx_1 + 2mx_2 - 4m = 0$$

- But it stretches or shrinks the data distribution after the projection
  - The same effect that makes the logistic function sharper or smoother

Logistic Regression

- Analysis of decision boundaries

- If we actually run a learning algorithm on this data set for many iterations
  - The absolute values of the parameters in $\boldsymbol{a} = [a_2, a_1, a_0]^\top$ will get larger to make the logistic slope sharper



Logistic Regression

- Classification process

- The recipe for classification
  - Prepare labeled training data
  - Prepare your label vector $\boldsymbol{y}$
    - 0 for class A, 1 for class B
  - Estimate the parameters, e.g. by using gradient descent
    - I mean the weight vector $\boldsymbol{a}$
  - Your parameters will define the decision boundary
    - And the decision is natural thanks to the sigmoid functions

- Are we missing something?
  - Of course we are

Multilayer Perceptron

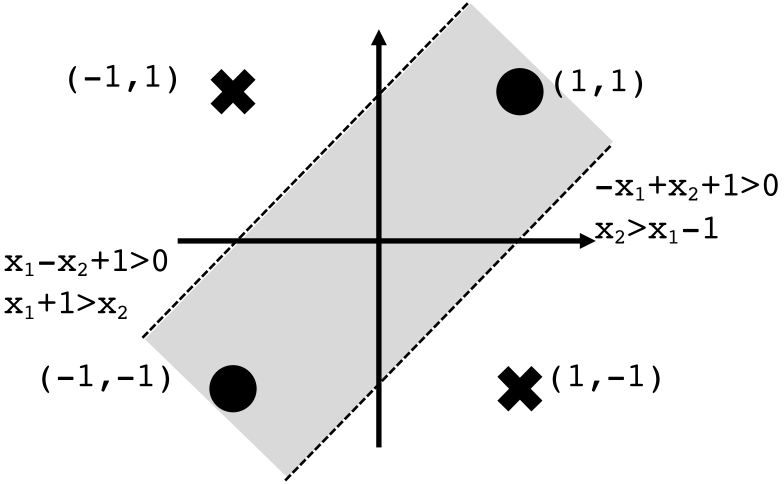
- The XOR problem

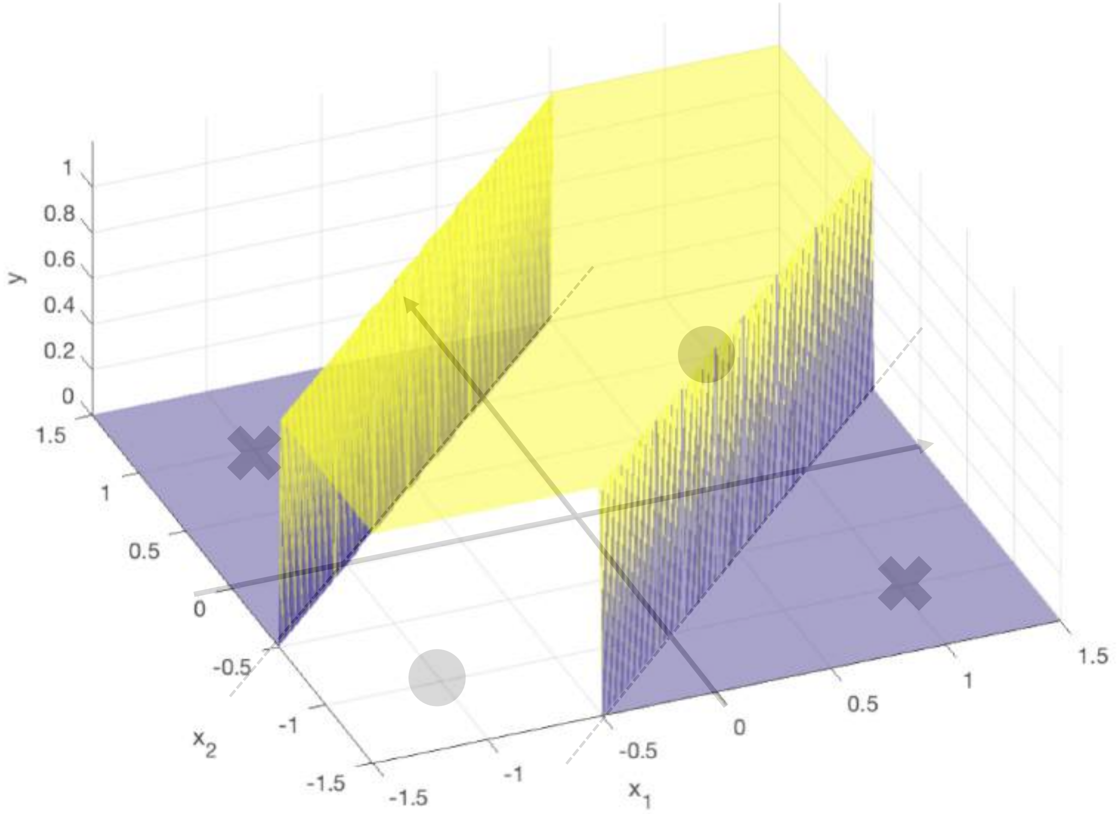
- In this lecture so far we saw that
  - Many problems are just a kind of function approximation
  - Linear line fitting or linear surface fitting was very easy: $\boldsymbol{y} \approx \boldsymbol{a}^\top \boldsymbol{X}$
  - Binary classification can be done by using a smooth step function, e.g. the logistic function:

    $\boldsymbol{y} \approx g(\boldsymbol{a}^\top \boldsymbol{X})$
  - They all can be represented as a network
    - Input nodes: the multidimensional data samples
    - Arrows: the weights and bias
    - Output nodes: the predicted targets
- What I will show you in the next slides
  - We can solve ALL (non-linear) function approximation problems by using the same structure
    - i.e. Linear combination of the input wrapped by a step function
    - Instead of guessing a particular parametric function (e.g. a high degree polynomial)
  - By increasing the number of layers

Multilayer Perceptron
- The XOR problem

- XOR is not linearly separable

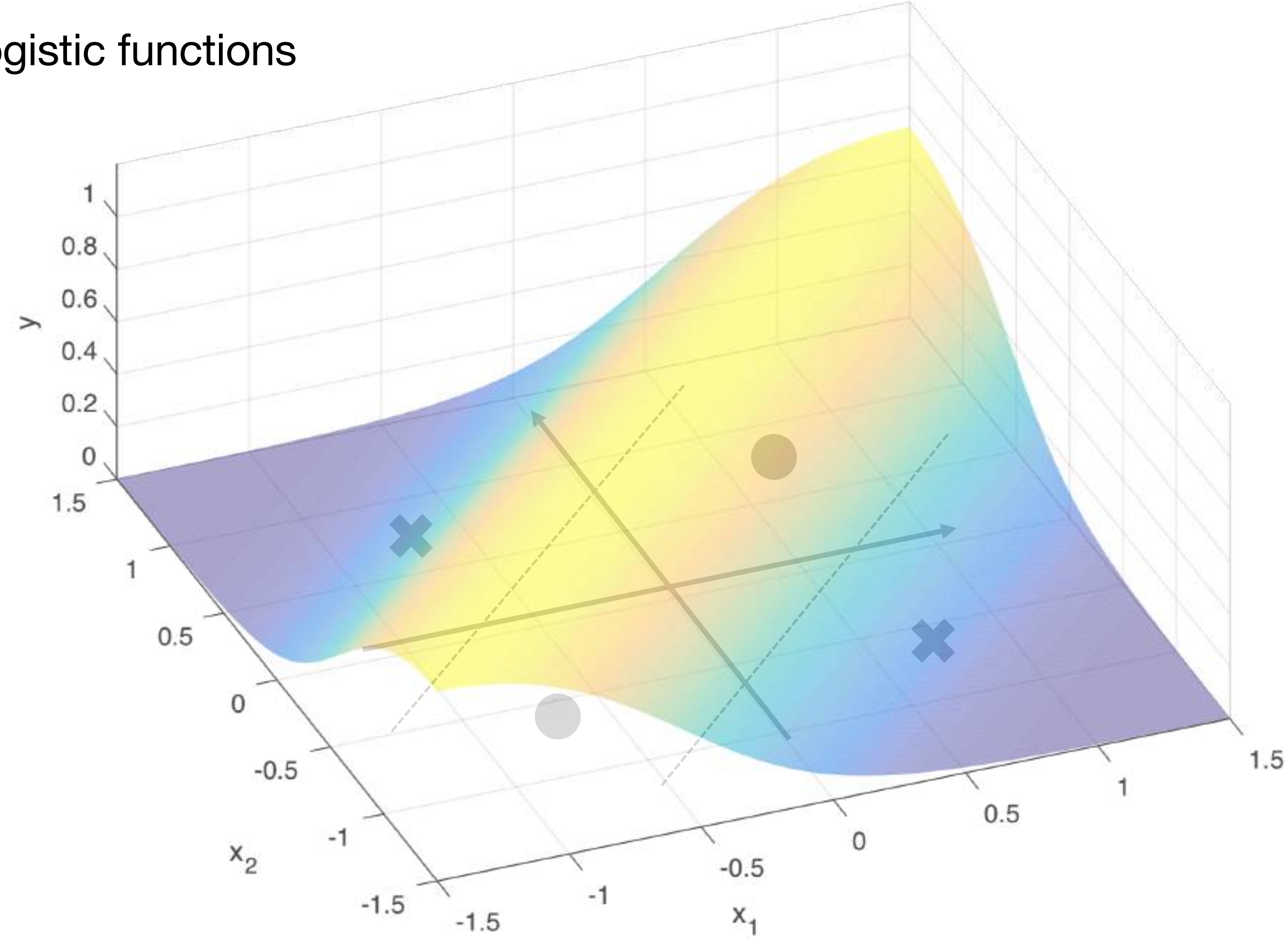




- Two hyperplanes can solve

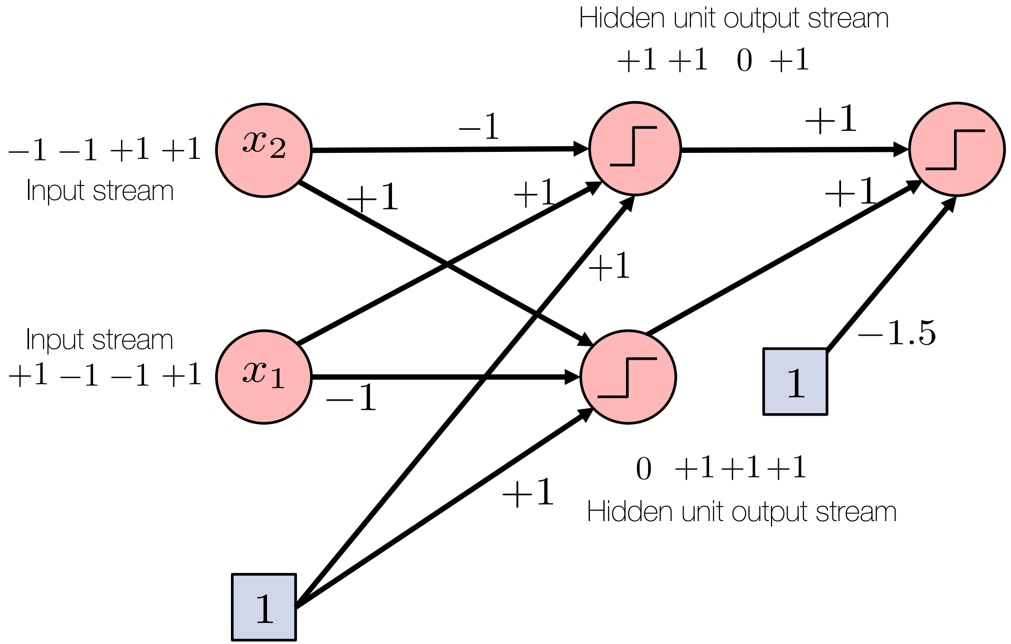
Multilayer Perceptron
- The XOR problem

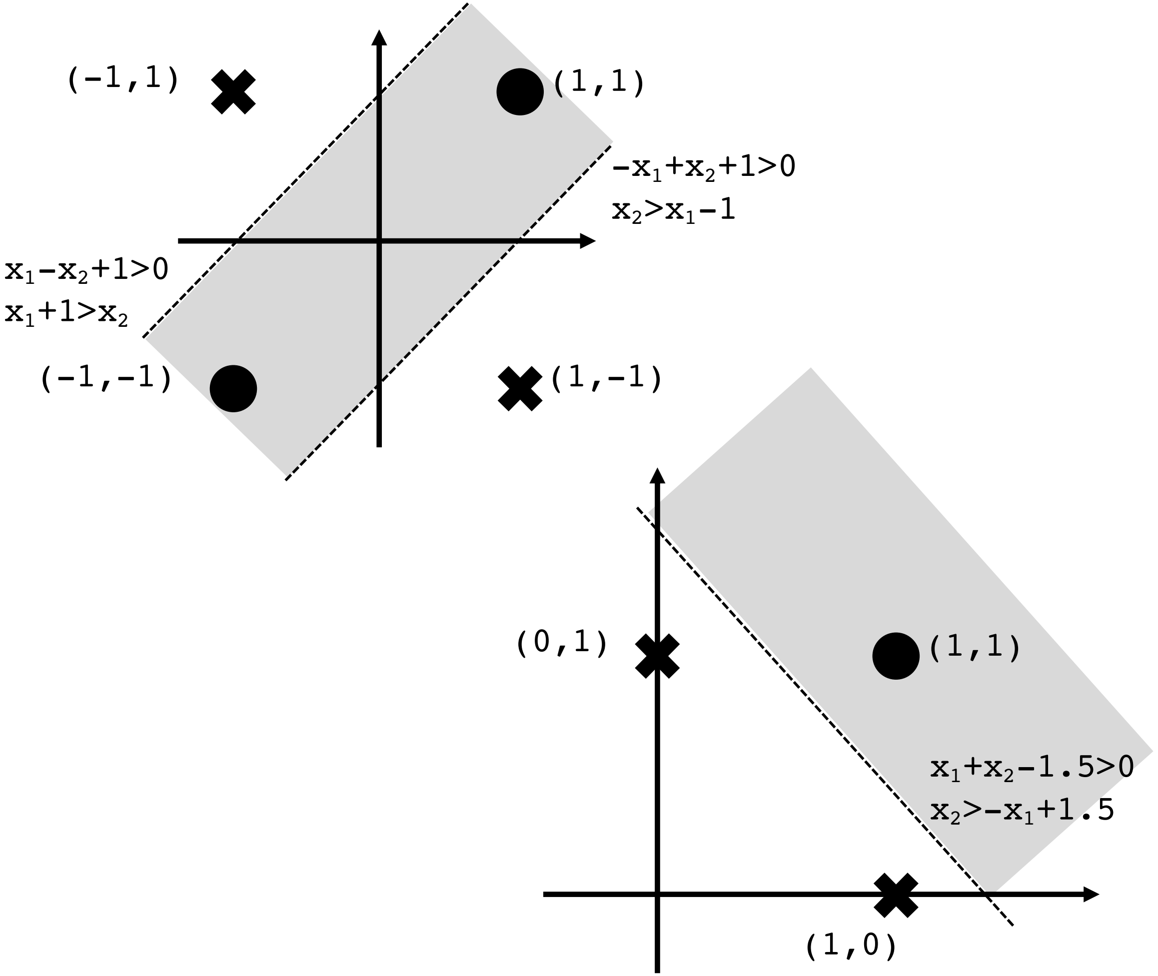
- I may need two logistic functions



Multilayer Perceptron

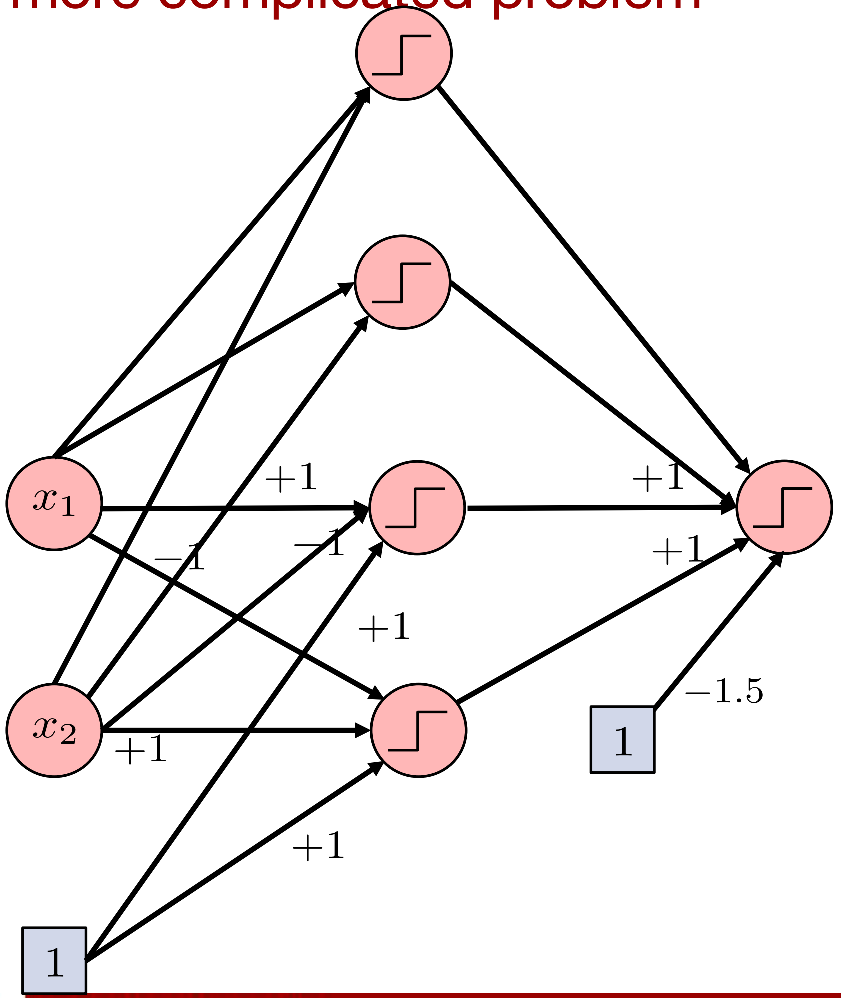
- The XOR problem

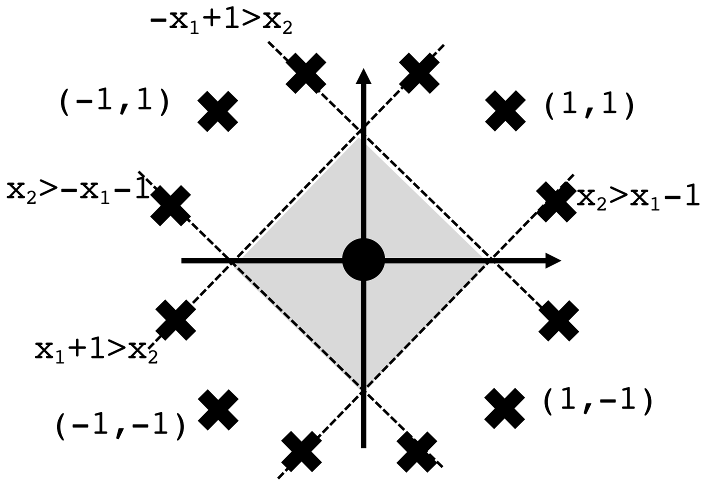


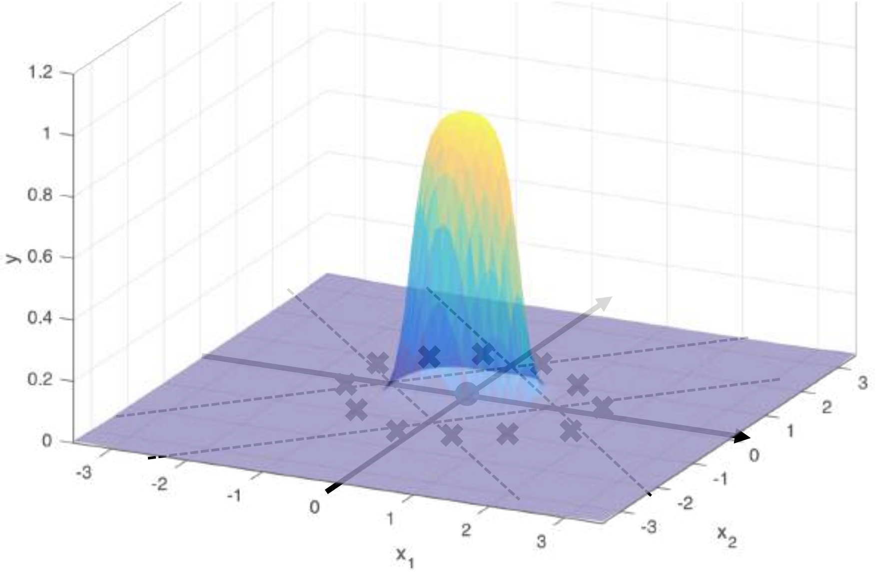


Multilayer Perceptron

- A more complicated problem



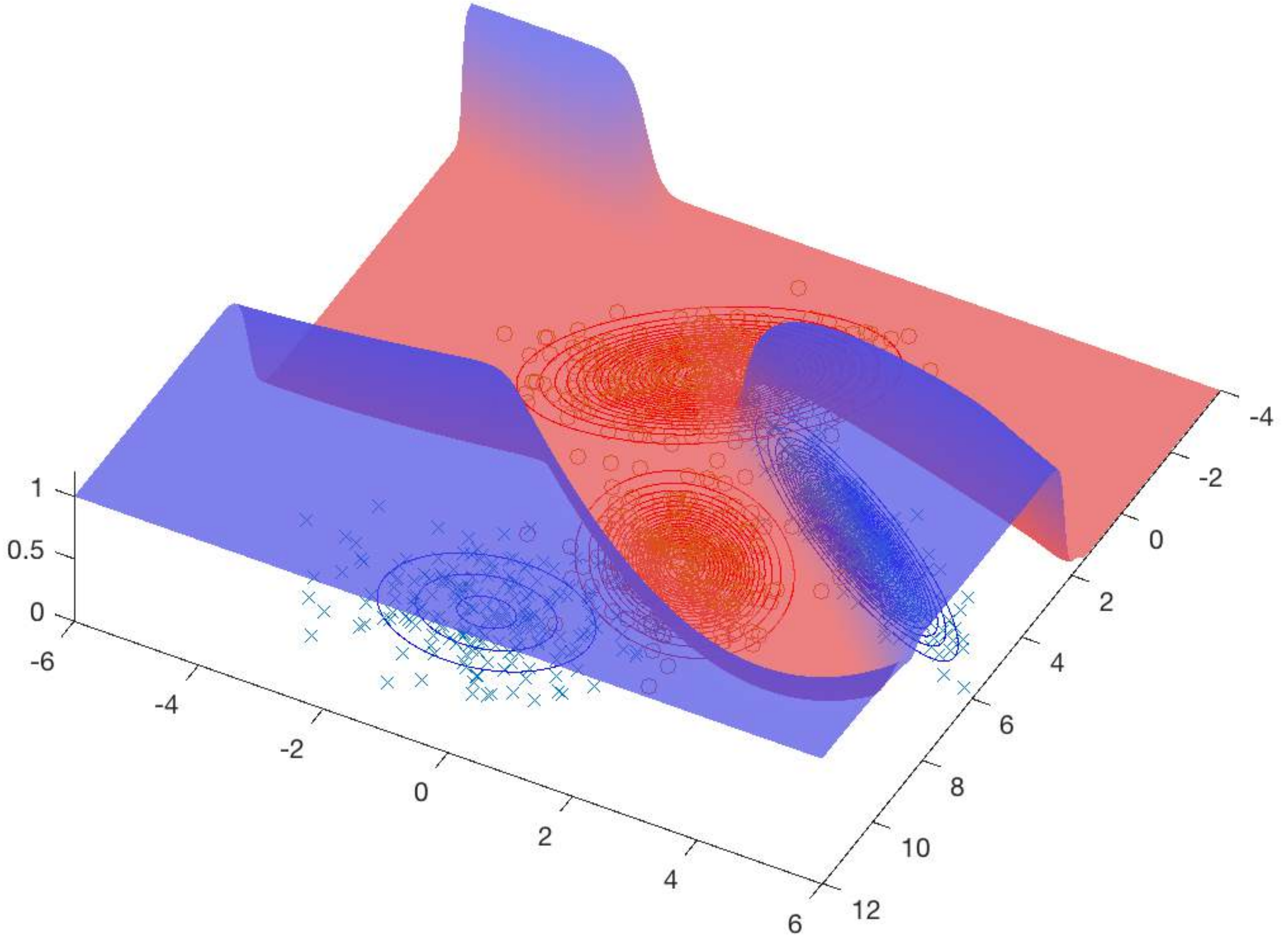




Multilayer Perceptron

- The universal approximation theorem

- If we're allowed to use as many hidden units as we want, we can approximate any target function
- The GMM classifier's posterior distribution



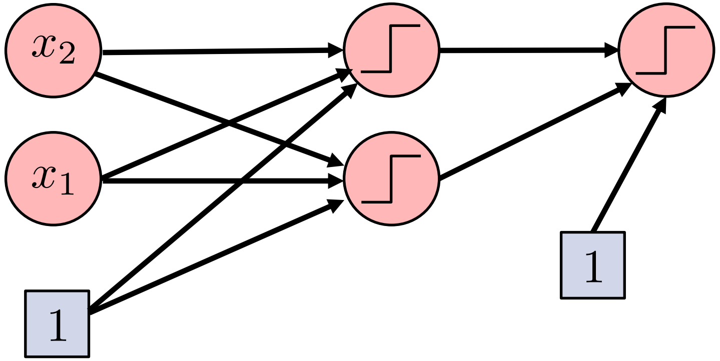
Multilayer Perceptron

- Error backpropagation

- First we randomly initialize the weights and forwardpropagate
  - Layer 1

$$\begin{bmatrix} z_2^{(1)} \\ z_1^{(1)} \end{bmatrix} = \begin{bmatrix} A_{22}^{(1)} & A_{21}^{(1)} & A_{20}^{(1)} \\ A_{12}^{(1)} & A_{11}^{(1)} & A_{10}^{(1)} \end{bmatrix} \cdot \begin{bmatrix} x_2^{(1)} \\ x_1^{(1)} \\ 1 \end{bmatrix} \qquad \boldsymbol{z}^{(1)} = \boldsymbol{A}^{(1)}\boldsymbol{x}^{(1)}$$

$$\begin{bmatrix} x_2^{(2)} \\ x_1^{(2)} \end{bmatrix} = g\left(\begin{bmatrix} z_2^{(1)} \\ z_1^{(1)} \end{bmatrix}\right) \qquad \boldsymbol{x}^{(2)} = g(\boldsymbol{z}^{(1)})$$



  - Final layer

$$z_1^{(2)} = \begin{bmatrix} A_{12}^{(2)} & A_{11}^{(2)} & A_{10}^{(2)} \end{bmatrix} \cdot \begin{bmatrix} x_2^{(2)} \\ x_1^{(2)} \\ 1 \end{bmatrix} \qquad z^{(2)} = \boldsymbol{A}^{(2)}\boldsymbol{x}^{(2)}$$

$$\hat{y} = g(z^{(2)})$$

  - Calculate the error $\mathcal{E} = \frac{1}{2}(\hat{y} - y)^2$

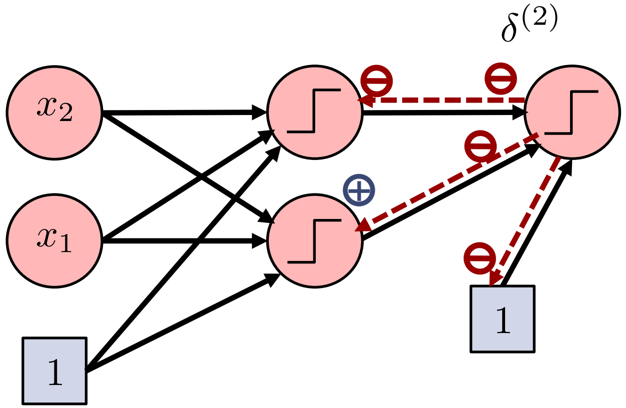
Multilayer Perceptron

- Error backpropagation

$$\boldsymbol{z}^{(1)} = \boldsymbol{A}^{(1)}\boldsymbol{x}^{(1)}$$
$$\boldsymbol{x}^{(2)} = g(\boldsymbol{z}^{(1)})$$
$$z^{(2)} = \boldsymbol{A}^{(2)}\boldsymbol{x}^{(2)}$$
$$\hat{y} = g(z^{(2)})$$
$$\mathcal{E} = \frac{1}{2}(\hat{y} - y)^2$$

- If the error is zero, we're good. Move on to the next sample
- Otherwise, update $\boldsymbol{A}^{(2)} = \boldsymbol{A}^{(2)} - \rho\nabla\boldsymbol{A}^{(2)} = \boldsymbol{A}^{(2)} - \rho\frac{\partial\mathcal{E}}{\partial\boldsymbol{A}^{(2)}}$

$$\frac{\partial\mathcal{E}}{\partial\boldsymbol{A}^{(2)}} = \underbrace{\frac{\partial\mathcal{E}}{\partial\hat{y}}\frac{\partial\hat{y}}{\partial z^{(2)}}}_{\text{BP Error } \frac{\partial\mathcal{E}}{\partial z^{(2)}} = \delta^{(2)}}\frac{\partial z^{(2)}}{\partial\boldsymbol{A}^{(2)}} = \underbrace{(\hat{y}-y)g'(z^{(2)})\boldsymbol{x}^{(2)\top}}_{\text{\{BP error\} X \{the input of the layer\}}} = \delta^{(2)}\boldsymbol{x}^{(2)\top}$$



- Let's see why it works
  - Suppose some negative error $\hat{y} - y = -1$ (i.e. network output not big enough)
    - Then the BP error is negative
      - Because $g'(z^{(2)})$ is positive everywhere
    - **To fix this error we need to promote** $\hat{y}\Uparrow$
  - Given that, if the input was positive, say $x_1^{(2)} = 1$
    - We need to promote the weights coming from it $A_{11}^{(2)}\Uparrow$
    - The gradient $\nabla A_{11}^{(2)}$ must be negative → $-\nabla A_{11}^{(2)}$ is positive → $\nabla A_{11}^{(2)} = \delta_1^{(2)}x_1^{(2)} < 0$ ← This is why we need negative BP error
  - If the input was negative, say $x_2^{(2)} = -1$ (this doesn't happen with logistic activation though)
    - We demote $A_{12}^{(2)}\Downarrow$ → $-\nabla A_{12}^{(2)}$ should be negative → $\nabla A_{12}^{(2)} = \delta_2^{(2)}x_2^{(2)} > 0$
  - Same arguments hold for the positive error
- Negative error with positive input➔ promotes weights $\boldsymbol{A}^{(2)}\Uparrow$
- Negative error with negative input➔ demotes weights $\boldsymbol{A}^{(2)}\Downarrow$

Multilayer Perceptron

- Error backpropagation

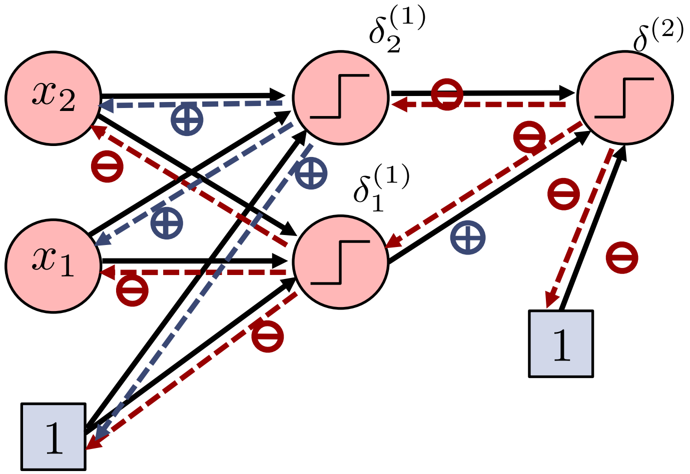
$$z^{(1)} = A^{(1)} x^{(1)}$$
$$x^{(2)} = g(z^{(1)})$$
$$z^{(2)} = A^{(2)} x^{(2)}$$
$$\hat{y} = g(z^{(2)})$$
$$\mathcal{E} = \frac{1}{2}(\hat{y} - y)^2$$

- Now let's work on the first layer
  - Update $A^{(1)} = A^{(1)} - \rho \nabla A^{(1)} = A^{(1)} - \rho \frac{\partial \mathcal{E}}{\partial A^{(1)}}$

$$\frac{\partial \mathcal{E}}{\partial A^{(1)}} = \underbrace{\underbrace{\frac{\partial \mathcal{E}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z^{(2)}}}_{\text{BP Error (final layer) } \delta^{(2)}} \frac{\partial z^{(2)}}{\partial x^{(2)}} \frac{\partial x^{(2)}}{\partial z^{(1)}}}_{\text{BP Error (hidden layer) } \delta^{(1)}} \frac{\partial z^{(1)}}{\partial A^{(1)}} = \underbrace{\left\{ \left(A^{(2)^\top} \delta^{(2)} \right) \odot g'(z^{(1)}) \right\}}_{\text{BP Error (hidden layer) } \delta^{(1)}} x^{(1)^\top}$$

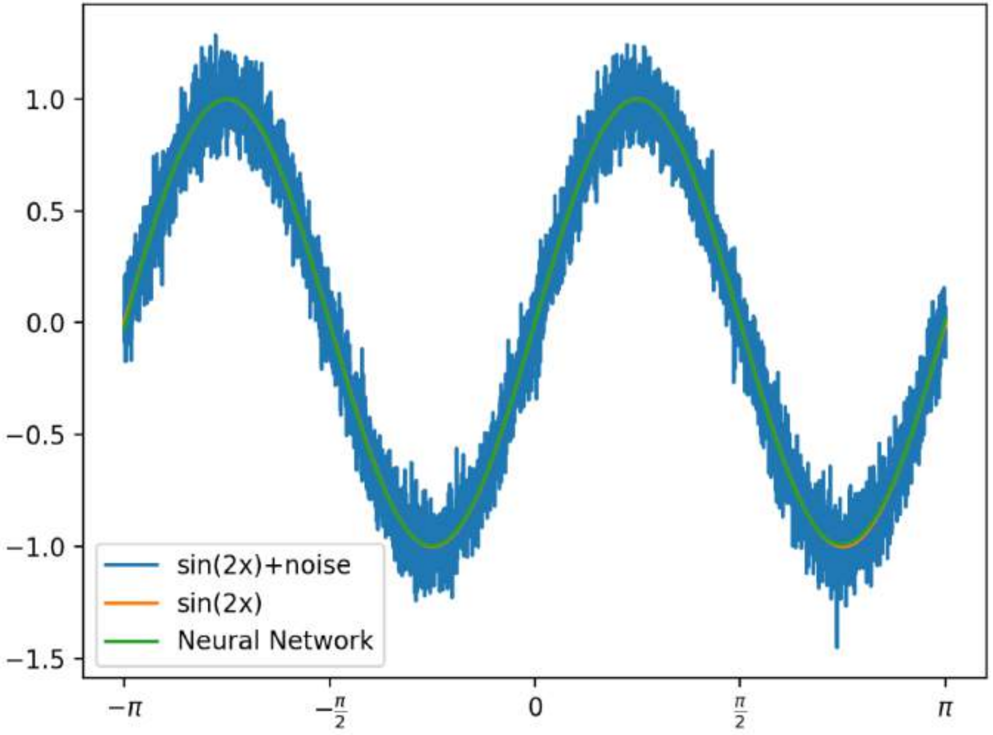
{BP error} times {the input of the layer}

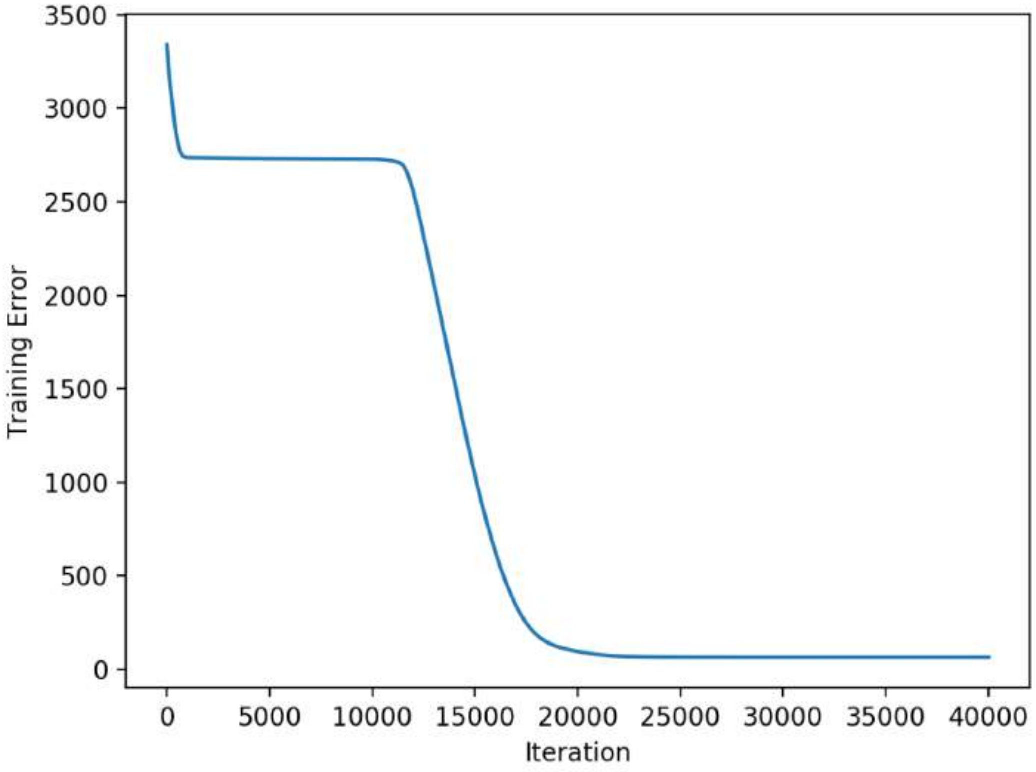
- Error in the hidden layer
  - If $\delta^{(2)} < 0 \ (\hat{y} - y < 0)$ and $A^{(2)}_{11} > 0$
    - Need to promote $x^{(2)}_1$ and, consequently, $z^{(1)}_1$
    - Promoting $z^{(1)}_1$ means reducing the final network error only if $\frac{\partial \mathcal{E}}{\partial z^{(1)}_1} < 0$
    - Therefore, $\delta^{(1)}_1 \propto \delta^{(2)} A^{(2)}_{11} < 0$ makes sense
  - If $\delta^{(2)} < 0 \ (\hat{y} - y < 0)$ and $A^{(2)}_{12} < 0$
    - Need to demote $x^{(2)}_2$ and $z^{(1)}_2$
    - $\frac{\partial \mathcal{E}}{\partial z^{(1)}_2} = \delta^{(1)}_2 \propto \delta^{(2)} A^{(2)}_{12} > 0$
  - **Error from the final layer, weighted by the second layer weights**



Multilayer Perceptron
- Does it work?

- Sine wave denoising

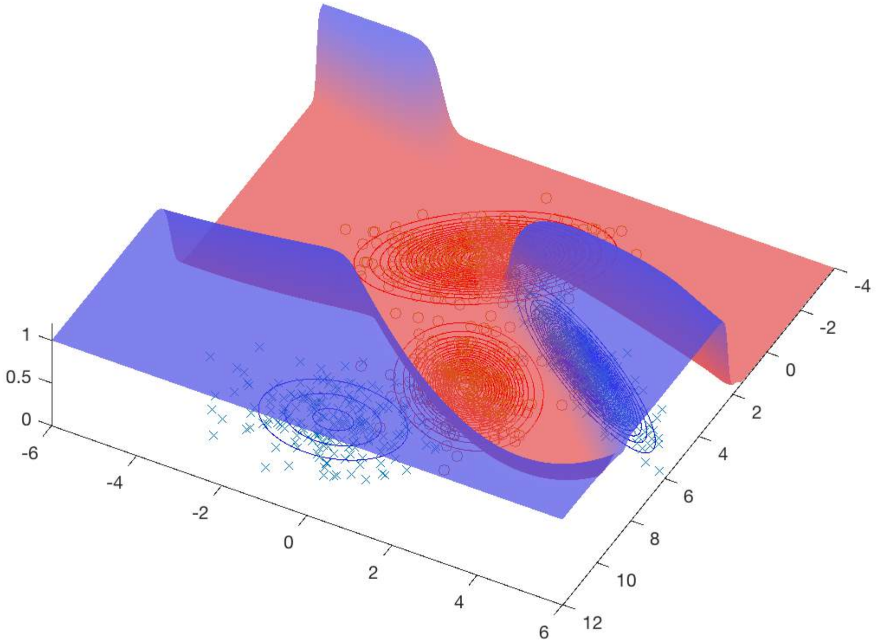




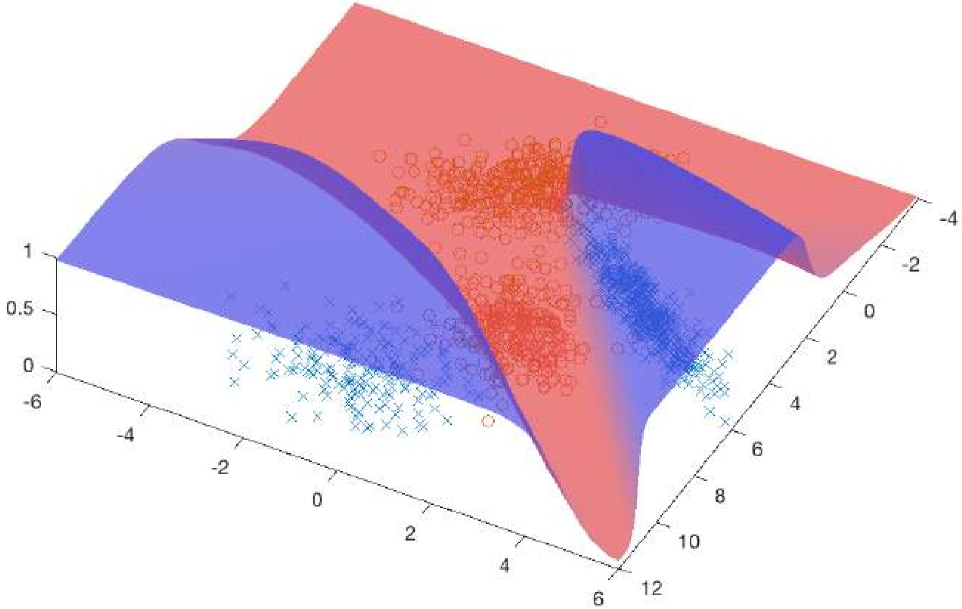
Multilayer Perceptron

- Does it work?

- GMM classifier



GMM Classifier



Neural Network

Multi-class Classification

- Logistic regression using a softmax activation

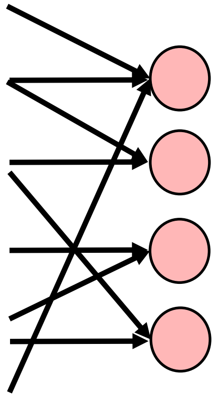
- You have 5 categories in your classification problem
  - How would you formulate your neural network output?
- Integers?
  - 1 : ★☆☆☆☆, 2 : ★★☆☆☆, ... , 5 : ★★★★★
- If the categories are not ordinal at all
  - 1 for RED, 2 for GREEN, 3 for ORANGE
  - 1 for “dog,” 2 for “lion,” 3 for “cat,” 4 for “wolf”
    - Error between “dog” and “lion” is larger than “dog” and “wolf”
- We can use something called one-hot vector
  - Different categories are equally different (Hamming distance)

$$\begin{array}{ccc} \begin{bmatrix}1\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\end{bmatrix} \\ \text{RED} & \text{GREEN} & \text{ORANGE} \end{array} \qquad \begin{array}{cccc} \begin{bmatrix}1\\0\\0\\0\end{bmatrix} & \begin{bmatrix}0\\1\\0\\0\end{bmatrix} & \begin{bmatrix}0\\0\\1\\0\end{bmatrix} & \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \\ \text{dog} & \text{cat} & \text{wolf} & \text{lion} \end{array}$$

Multi-class Classification

- Logistic regression using a softmax activation

- I want a smooth logistic-like function
  - But, a multidimensional one this time
  - Hopefully a probability vector
  - What should be the final layer activation function?

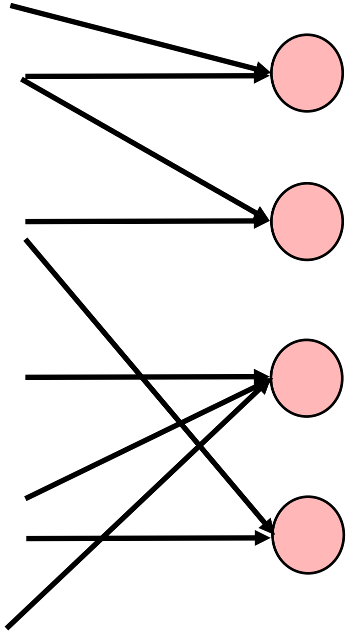


$$
\begin{array}{cccccc}
\begin{bmatrix} 0.80 \\ 0.05 \\ 0.10 \\ 0.05 \end{bmatrix} &
\begin{bmatrix} P(\text{dog}|x;\boldsymbol{\theta}) \\ P(\text{lion}|x;\boldsymbol{\theta}) \\ P(\text{cat}|x;\boldsymbol{\theta}) \\ P(\text{wolf}|x;\boldsymbol{\theta}) \end{bmatrix} &
\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} &
\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \\
\hat{y} & & \text{dog} & \text{cat} & \text{wolf} & \text{lion} \\
\text{doggish} & & & & &
\end{array}
$$

Multi-class Classification

- Logistic regression using a softmax activation

- For the multi-class neural networks we use **softmax activation** function

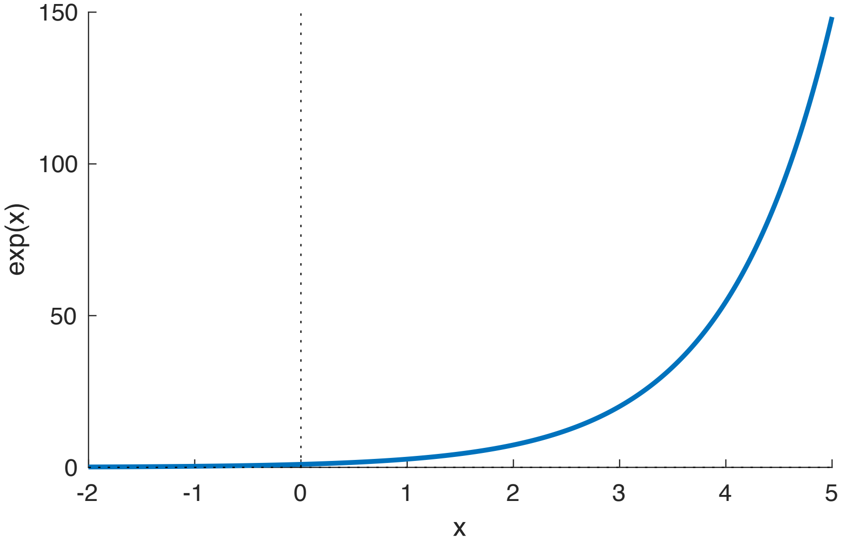


$$g^{(2)}\left(z_1^{(2)}\right) = \frac{\exp\left(\boldsymbol{A}_{(1,:)}\boldsymbol{x}^{(2)}\right)}{\sum_i \exp\left(\boldsymbol{A}_{(i,:)}\boldsymbol{x}^{(2)}\right)}$$

$$g^{(2)}\left(z_2^{(2)}\right) = \frac{\exp\left(\boldsymbol{A}_{(2,:)}\boldsymbol{x}^{(2)}\right)}{\sum_i \exp\left(\boldsymbol{A}_{(i,:)}\boldsymbol{x}^{(2)}\right)}$$

$$g^{(2)}\left(z_3^{(2)}\right) = \frac{\exp\left(\boldsymbol{A}_{(3,:)}\boldsymbol{x}^{(2)}\right)}{\sum_i \exp\left(\boldsymbol{A}_{(i,:)}\boldsymbol{x}^{(2)}\right)}$$

$$g^{(2)}\left(z_4^{(2)}\right) = \frac{\exp\left(\boldsymbol{A}_{(4,:)}\boldsymbol{x}^{(2)}\right)}{\sum_i \exp\left(\boldsymbol{A}_{(i,:)}\boldsymbol{x}^{(2)}\right)}$$



  - If you use cross entropy as your error, the BP error for the softmax regression is intuitive

$$\mathcal{E}(\boldsymbol{y}||\hat{\boldsymbol{y}}) = -\sum_i y_i \log(\hat{y}_i) \qquad \frac{\partial \mathcal{E}}{\partial \boldsymbol{A}} = (\hat{\boldsymbol{y}} - \boldsymbol{y})\boldsymbol{x}^{(1)}$$

Recap

- Combining smooth step functions
  - Can create valleys and hills
  - Can universally approximate your objective function
- Logistic regression
  - Classification is not different from function approximation (regression)
  - Logistic function can approximate the sharp step function
    - Binary classification
  - Softmax function can approximate the one-hot vector representation
- There are a lot of choices for the hidden unit activation
- Universal approximation theorem says one hidden layer is enough
  - Why deep learning?

Reading

- Chapter 4
  - Feel free to skip off-topic clauses though

Thank You!