

APS1070

Foundations of Data Analytics and
Machine Learning

Winter 2022

Week 10:

- *Polynomial Regression*
- *Optimization and Convexity*
- *Regularization*
- *Classification*
- *Neural Networks*



Slide Attribution

These slides contain materials from various sources. Special thanks to the following authors:

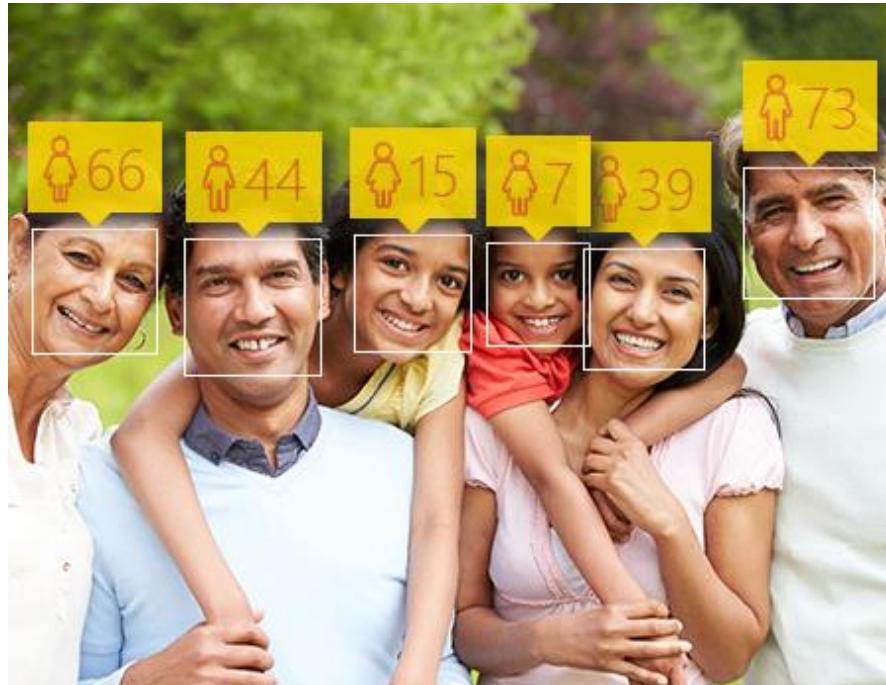
- Lisa Zhang
- Roger Grosse
- Jason Riordon

Last Time

- Linear Regression
 - Empirical Risk Minimization
 - Maximum Likelihood Estimation
 - Negative Log-likelihood
- Today we will continue with **nonlinear regression**.

Nonlinear Regression

Age Prediction



<https://www.how-old.net/>

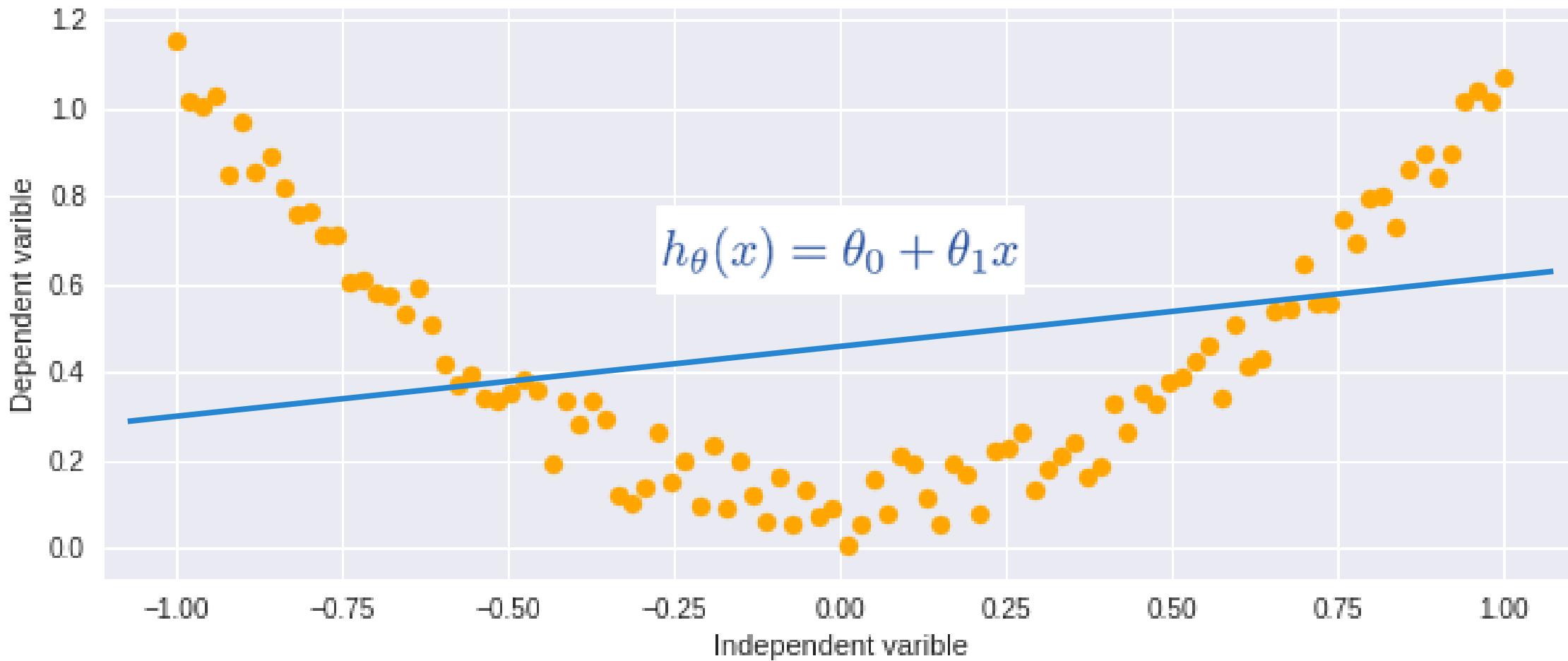
Stock Market Prediction

Daily Chart – AT&T



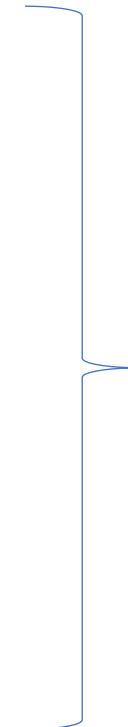
Deviations Below Strong & Established Linear Regression Uptrend are Buying Opportunities

Nonlinear Regression



Agenda

- Polynomial Regression
- Convexity and Optimization
- Logistic Regression
- Gradient Descent
- Regularization
- Multiclass Classification
- Neural Networks



Theme:
**Nonlinear Regression
(and Classification)**

Nonlinear Regression

Recap: Linear Regression

Hypothesis:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Parameters:

$$\theta_0, \theta_1$$

Cost Function:

$$J(\theta_0, \theta_1) = \frac{1}{2N} \sum_{i=1}^N (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Goal:

$$\text{minimize}_{\theta_0, \theta_1} J(\theta_0, \theta_1)$$

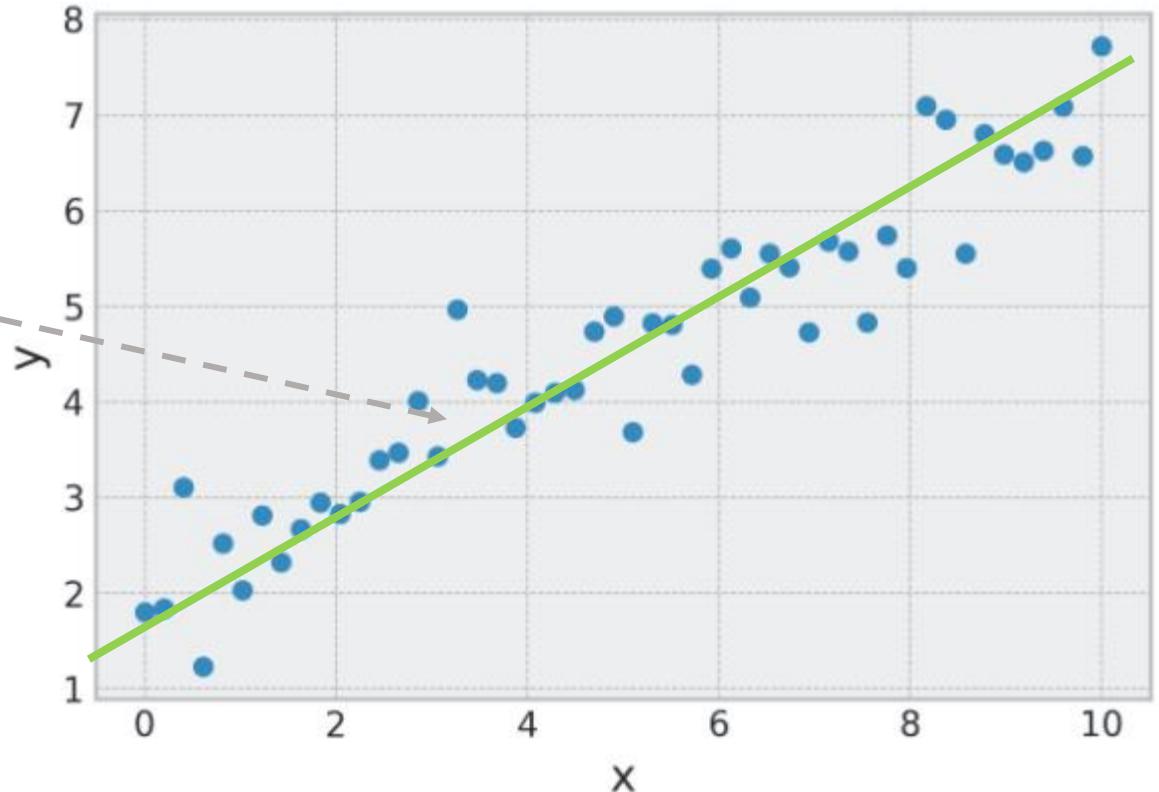
(1) direct solution

$$\frac{dJ}{d\theta} = 0$$

OR

(2) gradient descent

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}$$



Recap: Vectorization

Hypothesis:

$$h_{\theta}(\mathbf{X}) = \mathbf{X}\theta$$

Parameters:

$$\theta$$

Cost Function:

$$J(\theta) = \frac{1}{2N} \|y - \hat{y}\|^2$$

Goal:

$$\text{minimize}_{\theta} J$$

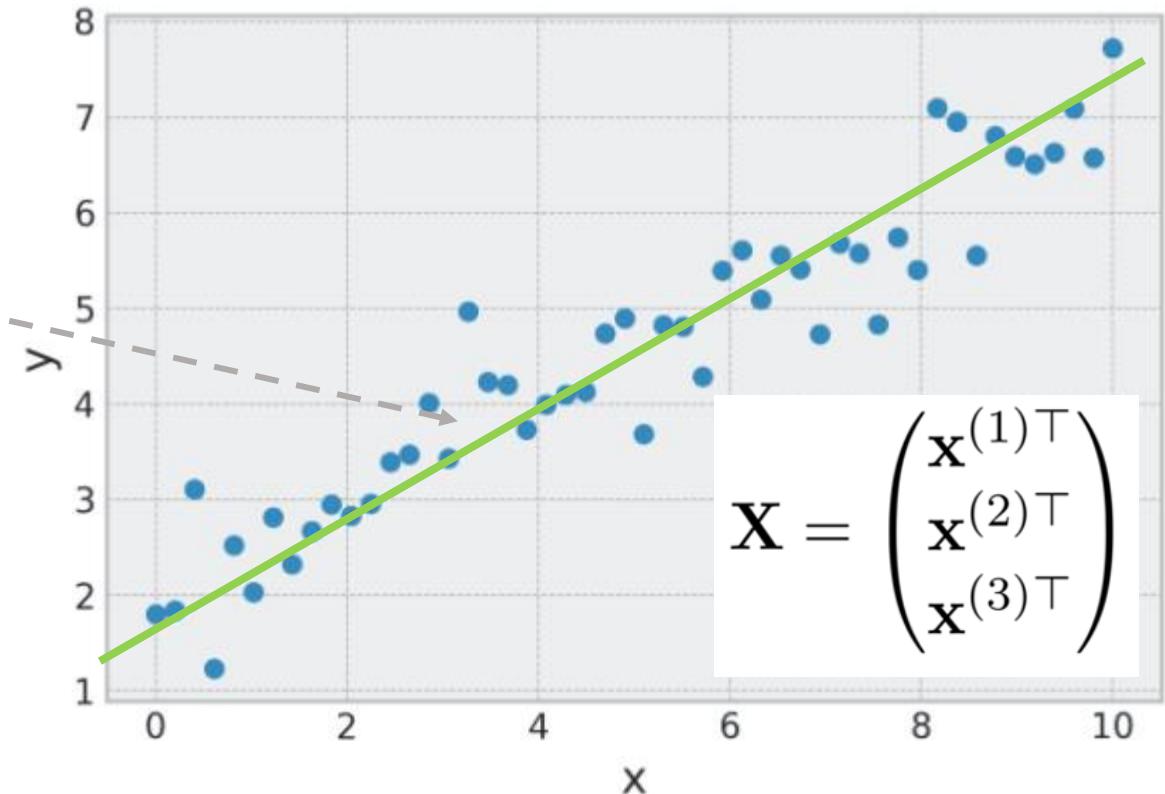
(1) direct solution

$$\frac{dJ}{d\theta} = 0$$

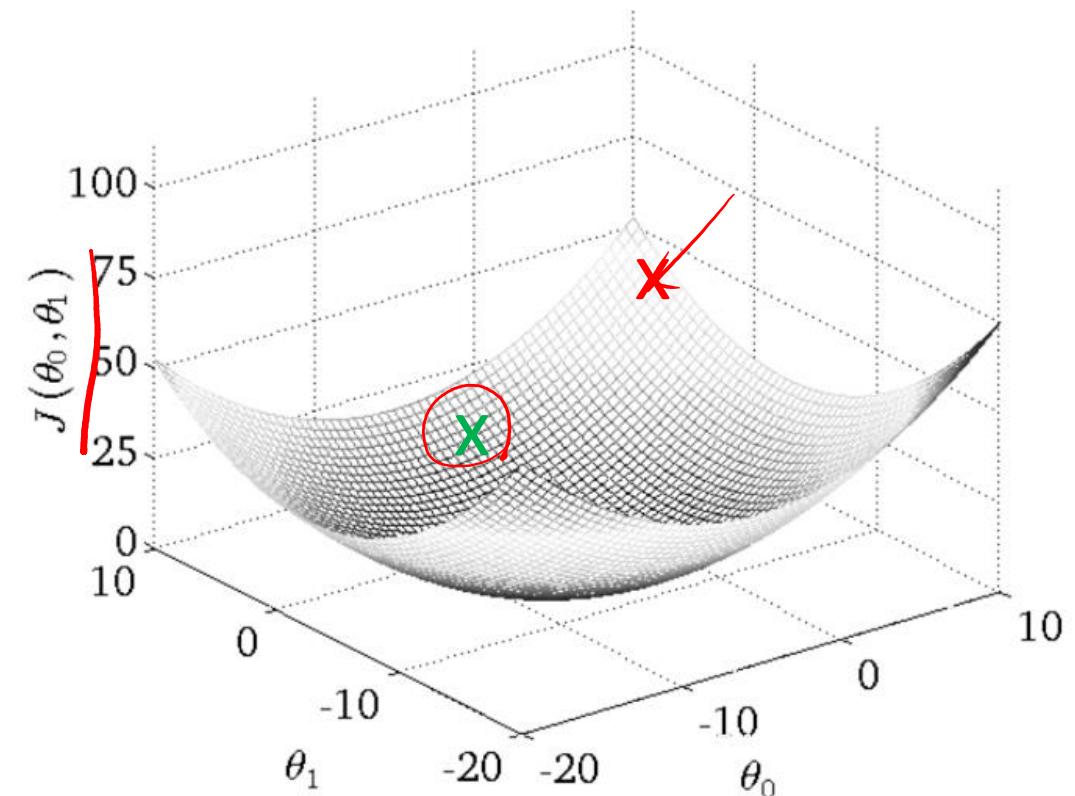
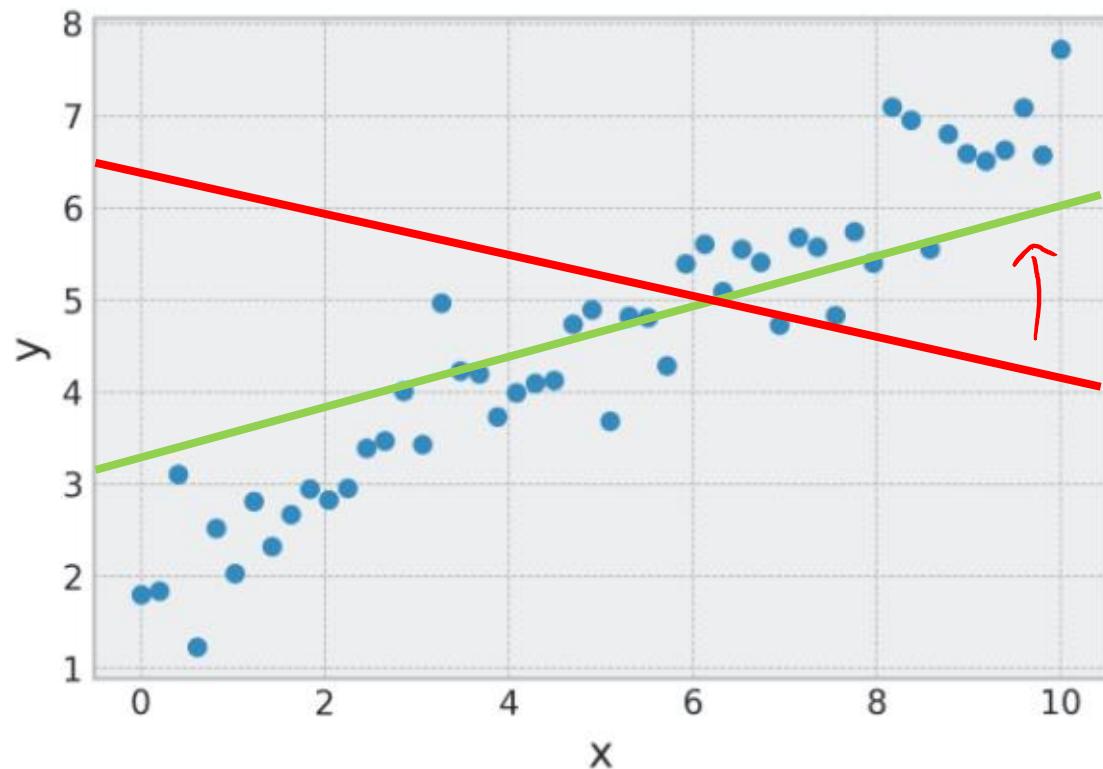
OR

(2) gradient descent

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta}$$

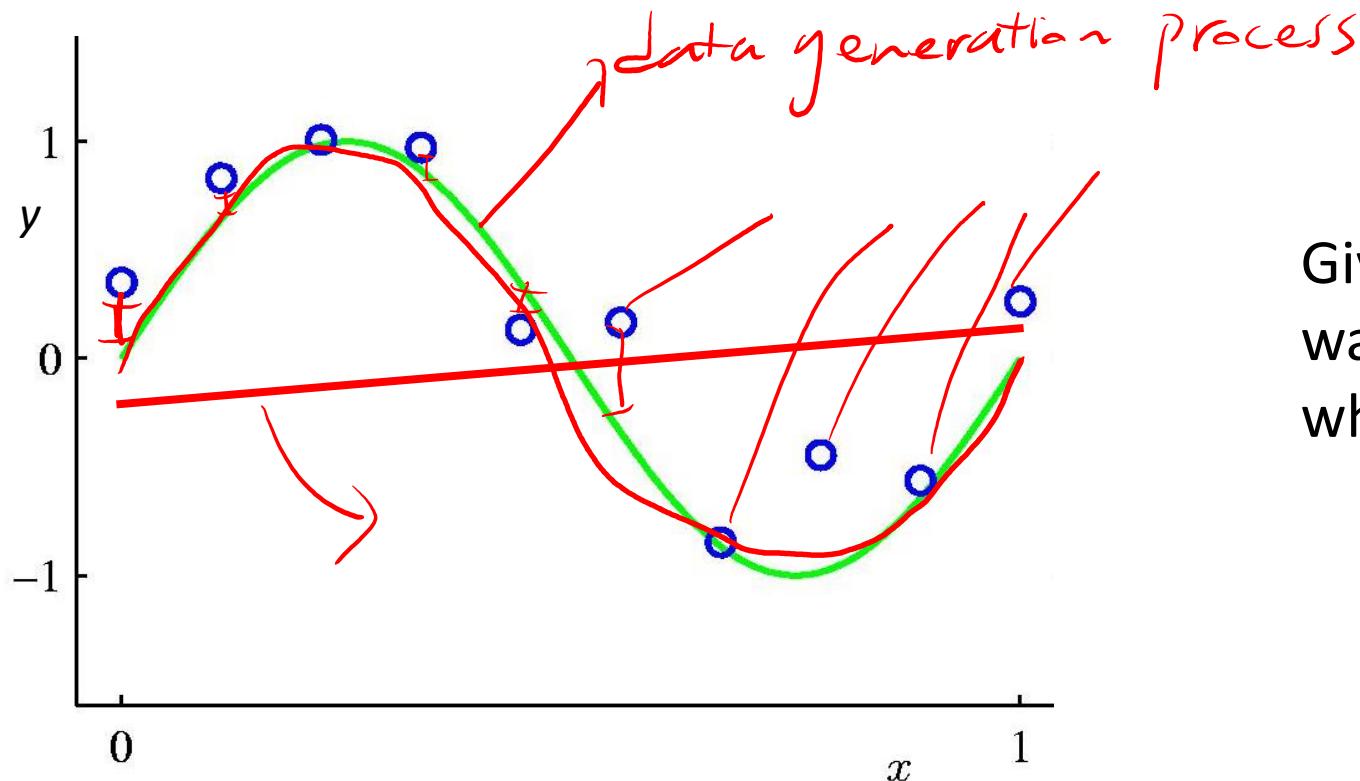


Recap: Convexity



Nonlinear Regression

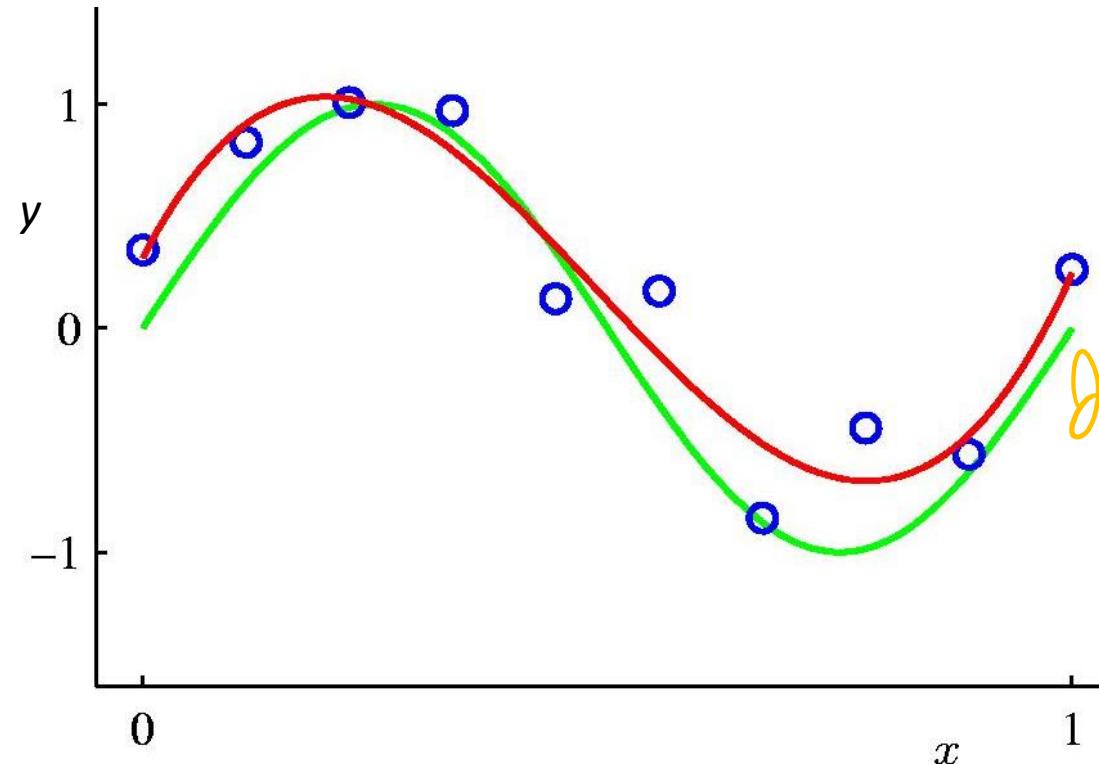
- Suppose we want to model the following data



Given noisy **sample** data we
want to find a **hypothesis** for
what **generated** the data

- Cannot be fit with a linear model...

Nonlinear Regression



➤ One option is to fit a low-degree polynomial:

$$\hat{y} = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x + \theta_0$$

This is known as
polynomial regression

Q: Does this mean we have to derive a whole new algorithm?

linear regression

non-linear regression

Feature Mapping

- Implement a polynomial transformation (feature mapping) by replacing input with polynomial of increasing order:

$$\psi(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \end{pmatrix}$$

➤ Hence our hypothesis can be written as:

$$\hat{y} = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x + \theta_0 \quad (1)$$

$$\hat{y} = \underline{\theta^T} \underline{\psi(x)}$$

The derivations and algorithms from last lecture remain the same! Why?

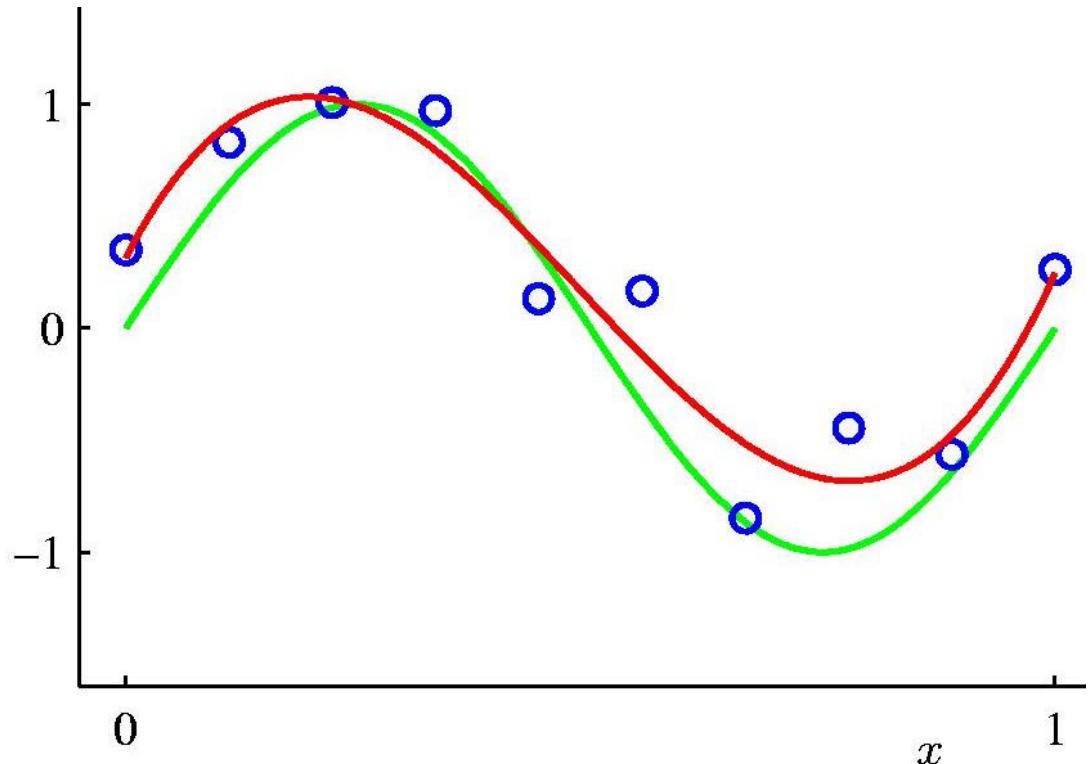
Polynomial Regression

- This doesn't require changing the algorithm, just **pretend $\psi(x)$ is the input vector.**

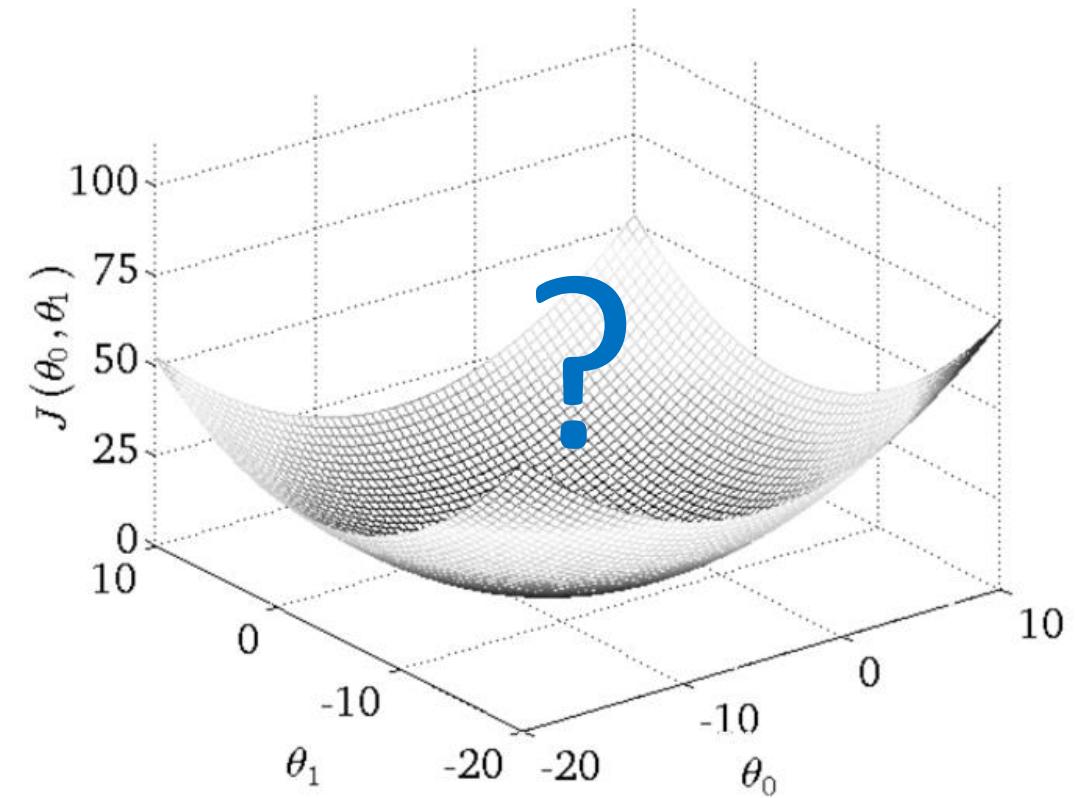
$$\hat{y} = \theta^T \psi(x) \quad \psi(x) = \begin{pmatrix} 1 \\ x \\ x^2 \\ x^3 \end{pmatrix}$$

- Feature maps let us fit nonlinear models
- Before deep learning, most of the effort in building a practical machine learning system was **feature engineering.**

Q: Convexity of Polynomial Regression?



$$\hat{y} = \theta^T \psi(x)$$



Direct Solution

- Polynomial regression is really a linear regression problem with some feature engineering.

$$\hat{y} = \theta^T \psi(x)$$
$$J(\theta) = \frac{1}{2N} \|y - \hat{y}\|^2$$

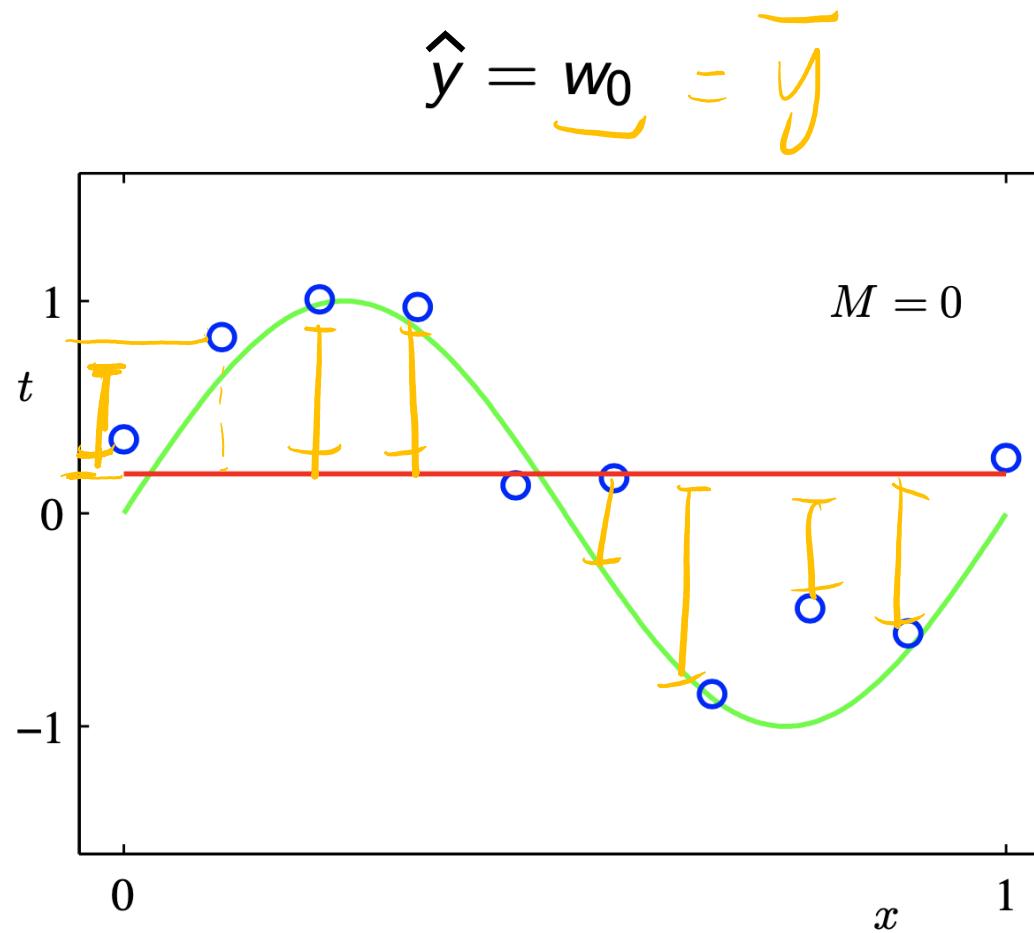
$\xrightarrow{\text{minimize cost}}$

$$\theta = (\psi^T \psi)^{-1} \psi^T y$$

*Requires that all columns
are linearly independent*

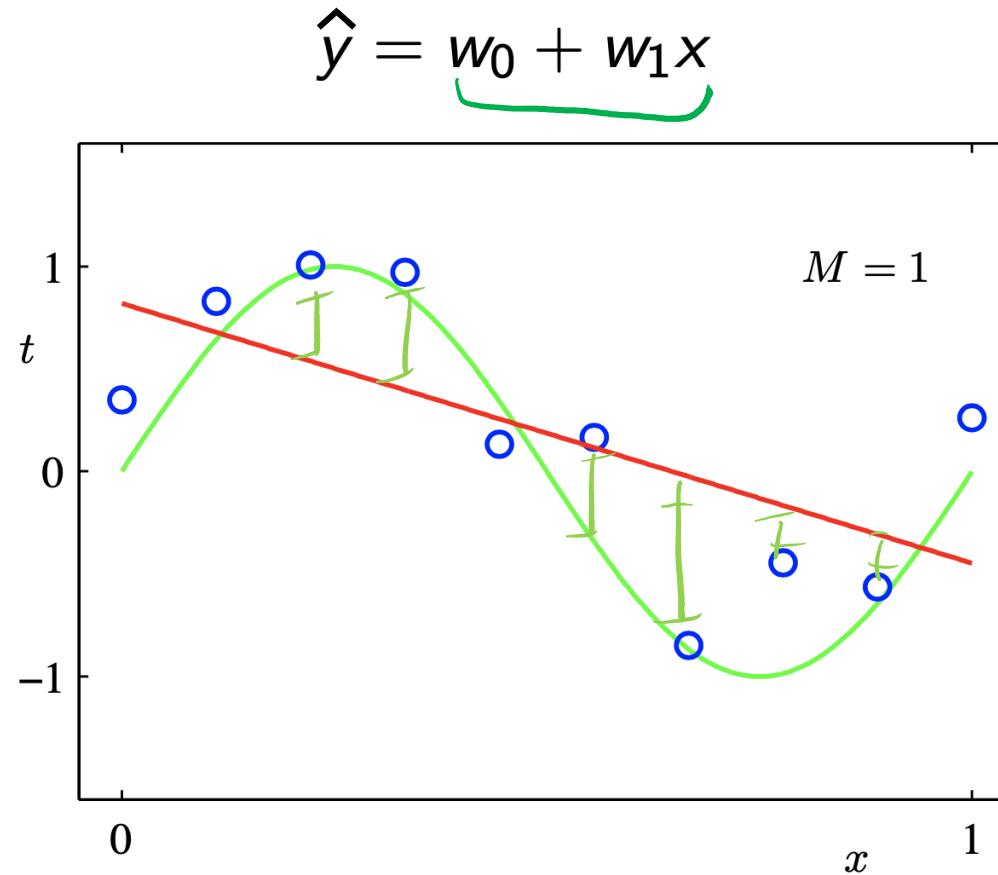
- Require that $\psi^T \psi \in \mathbb{R}^{D \times D}$ to be invertible. This is the case if and only if $\text{rank } (\psi) = D$.

Fitting Polynomial ($M = 0$)

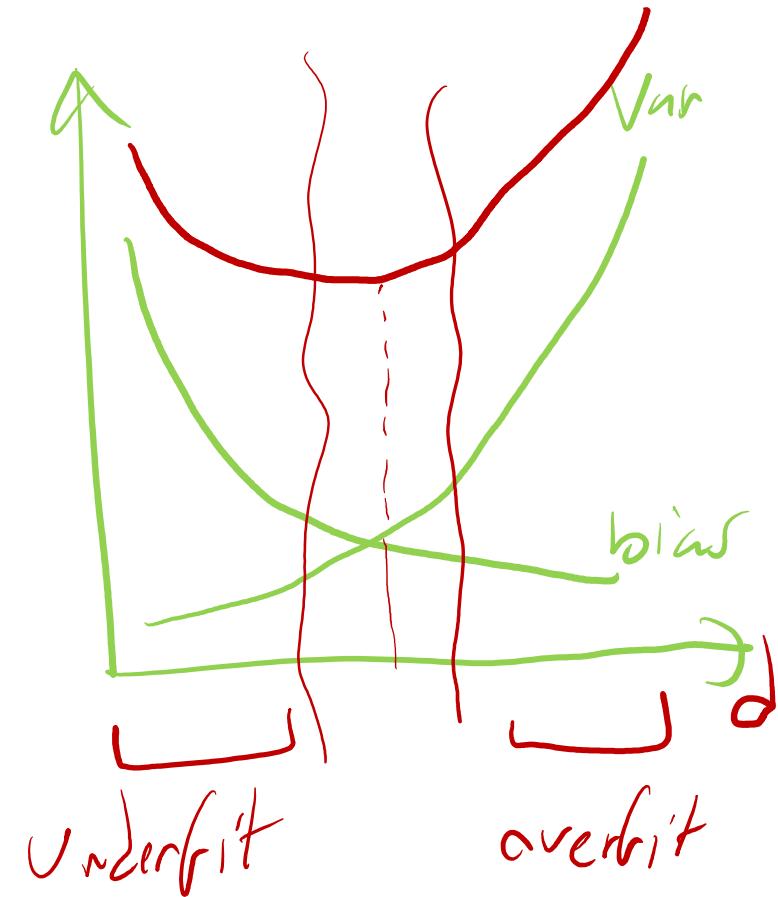
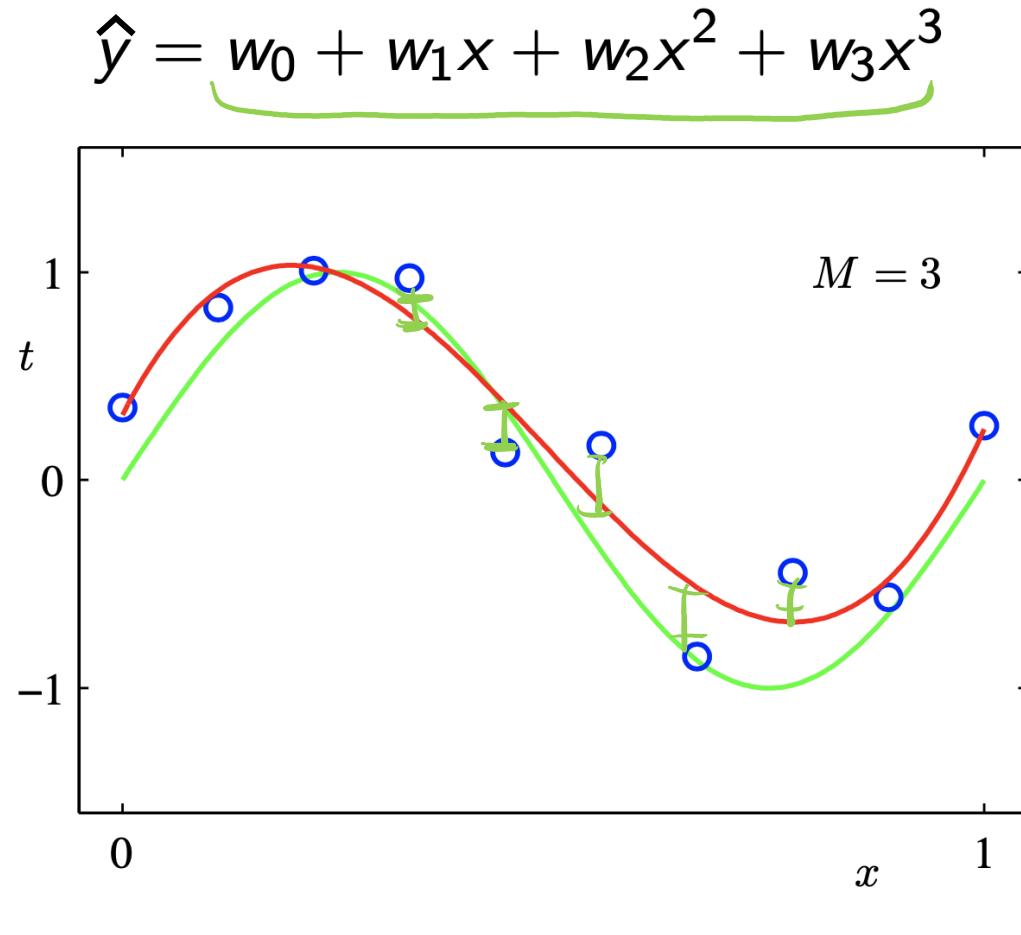


$$MSE = \text{bias}^2 + \text{Variance}$$

Fitting Polynomial ($M = 1$)

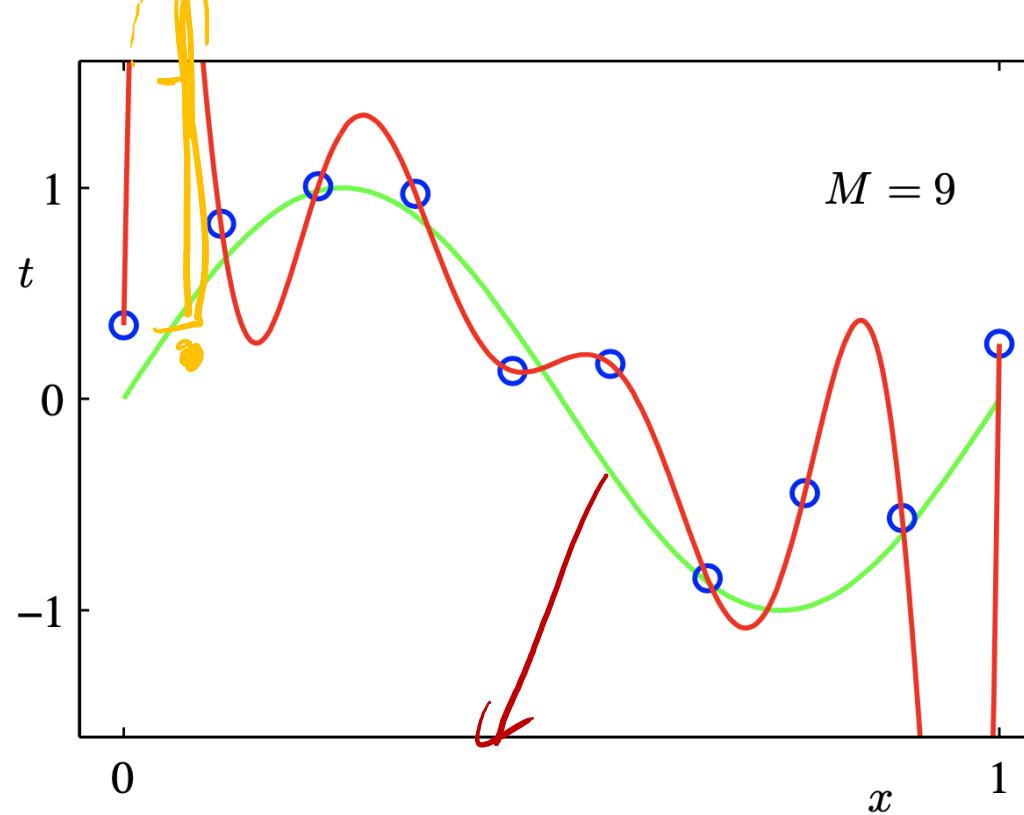


Fitting Polynomial ($M = 3$)



Fitting Polynomial ($M = 9$)

$$\hat{y} = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots + w_9x^9$$



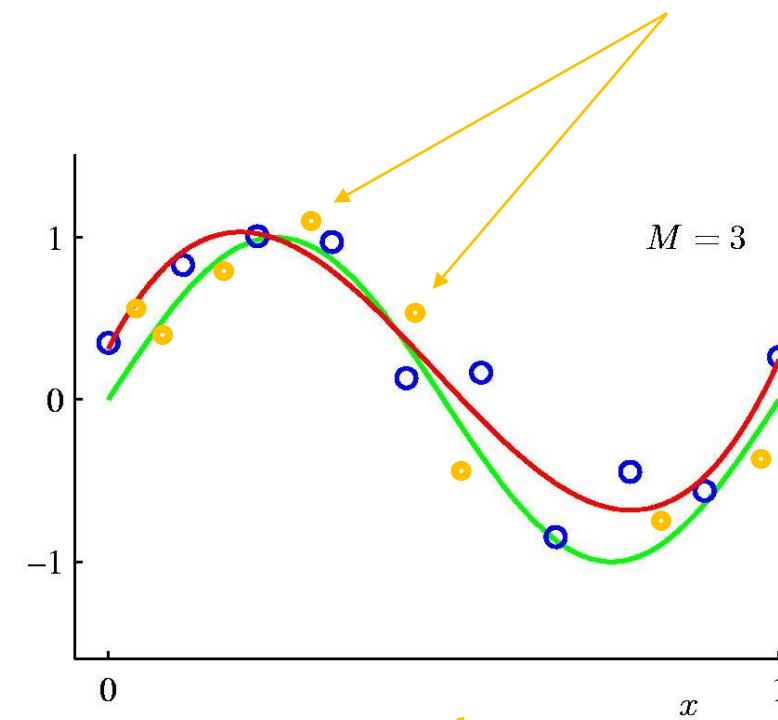
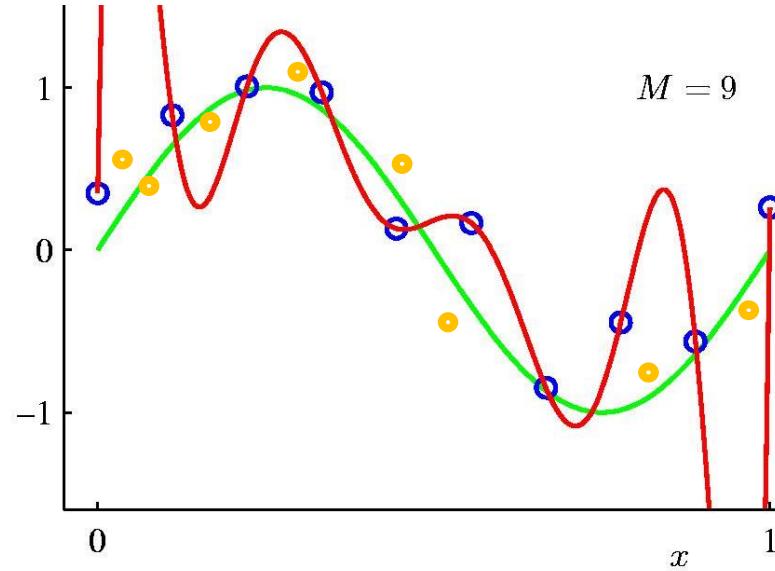
Fitting Polynomial ($M = 9$)

$$\hat{y} = w_0 + w_1x + w_2x^2 + w_3x^3 + \dots + w_9x^9$$



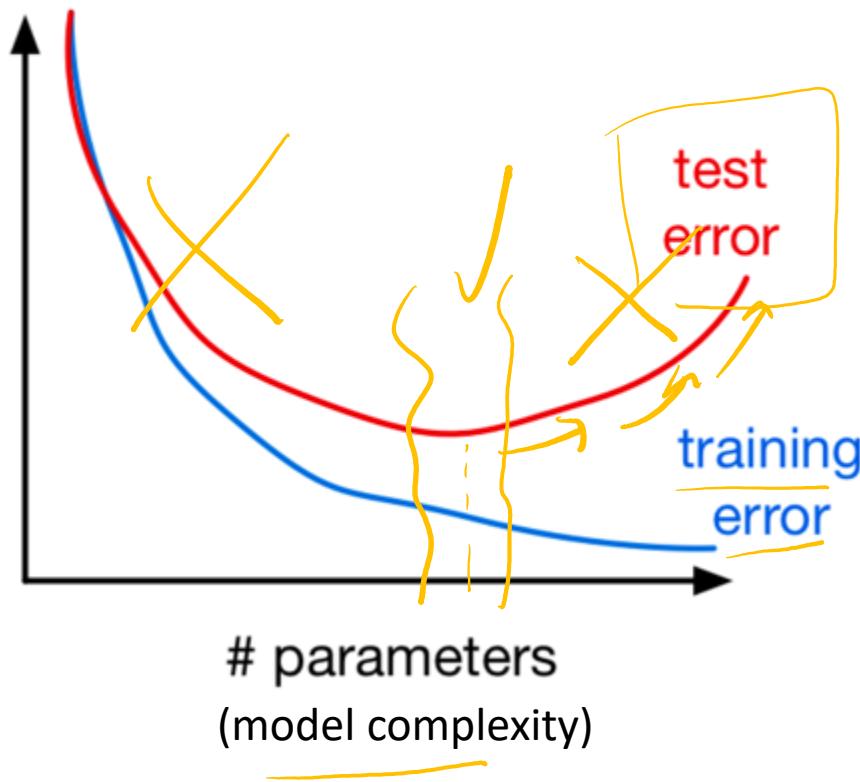
Generalize to New Samples

- We could give the hypothesis a higher complexity, or capacity to fit the data, but this may not generalize well to **new samples**



Generalization

- Training and test error as a function of # parameters:

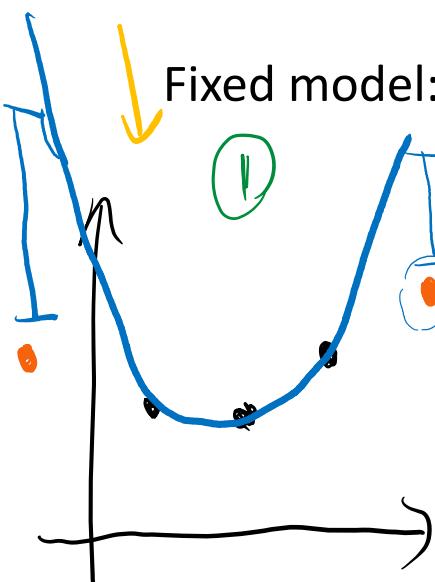


Generalization

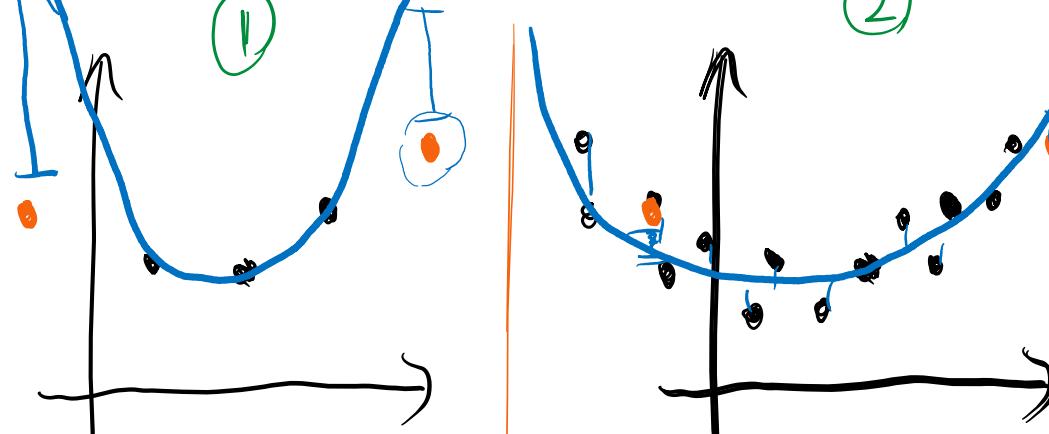
$$n \quad \boxed{n}$$

$$n \gg d$$

- Training and test error as a function of # training examples:



Fixed model: Polynomial of degree 2

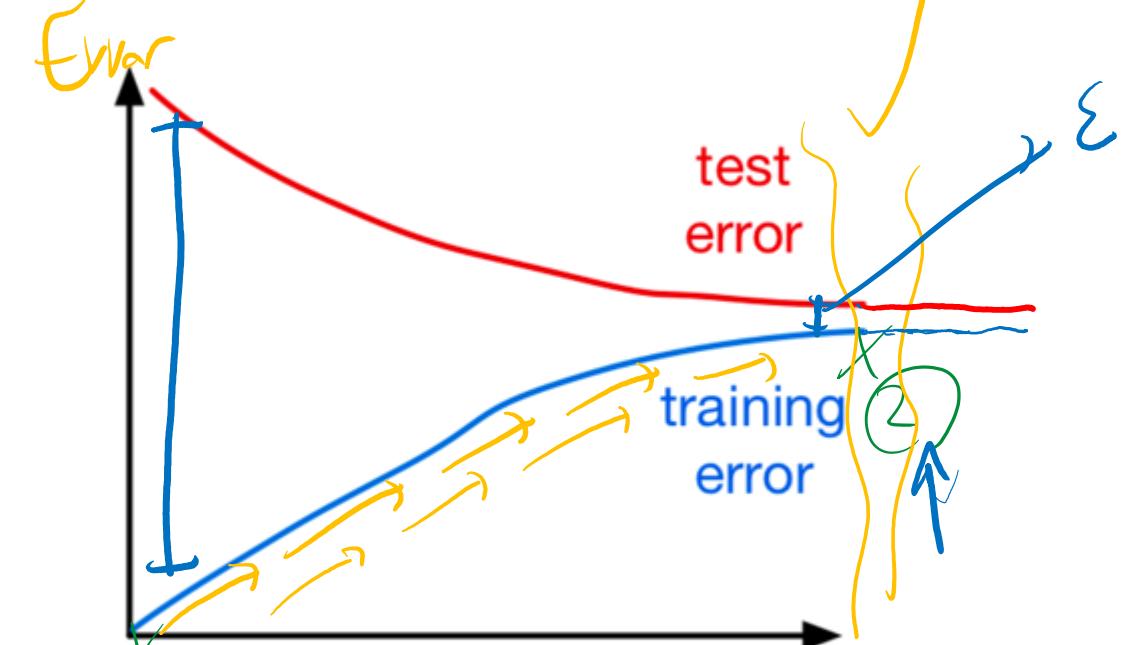


Training error: Zero

Test error: >> 0

Their difference: Huge

Model is Fixed



Training error: Some nonzero value a

Test error: $a+\epsilon$

Their difference: ϵ

Regularization

- The degree of the polynomial is a hyperparameter, just like k in KNN. We can tune it using a validation set.
- But restricting the size of a model is a crude solution, since you'll never be able to learn a more complex model, even if the data support it.
- **Another approach:** keep the model flexible, but **regularize** it
 - **Regularizer:** a function that quantifies how much we prefer one hypothesis vs another

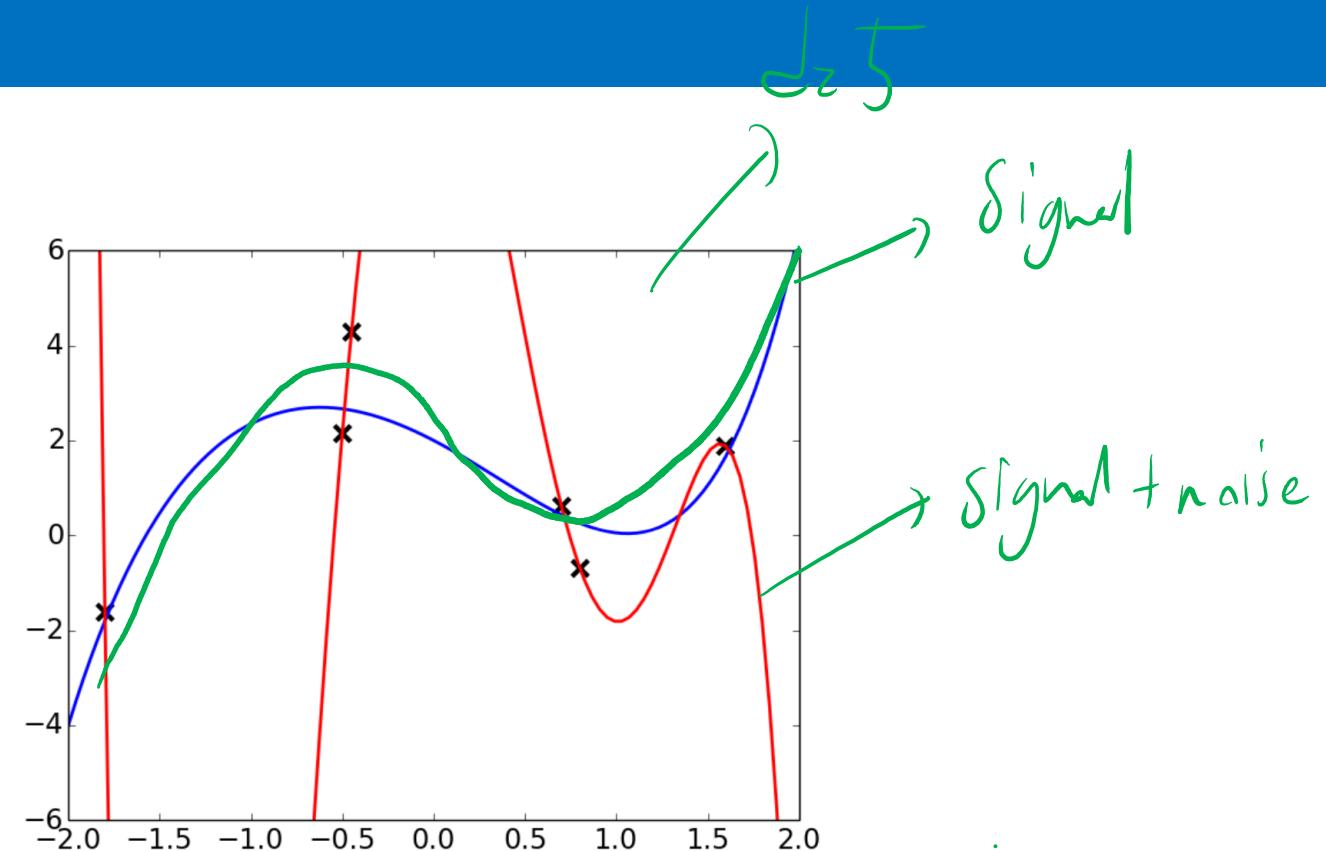
L^2 Regularization

Observation:

polynomials that overfit often have large coefficients

$$\frac{1}{2} \sum \theta_j^2$$

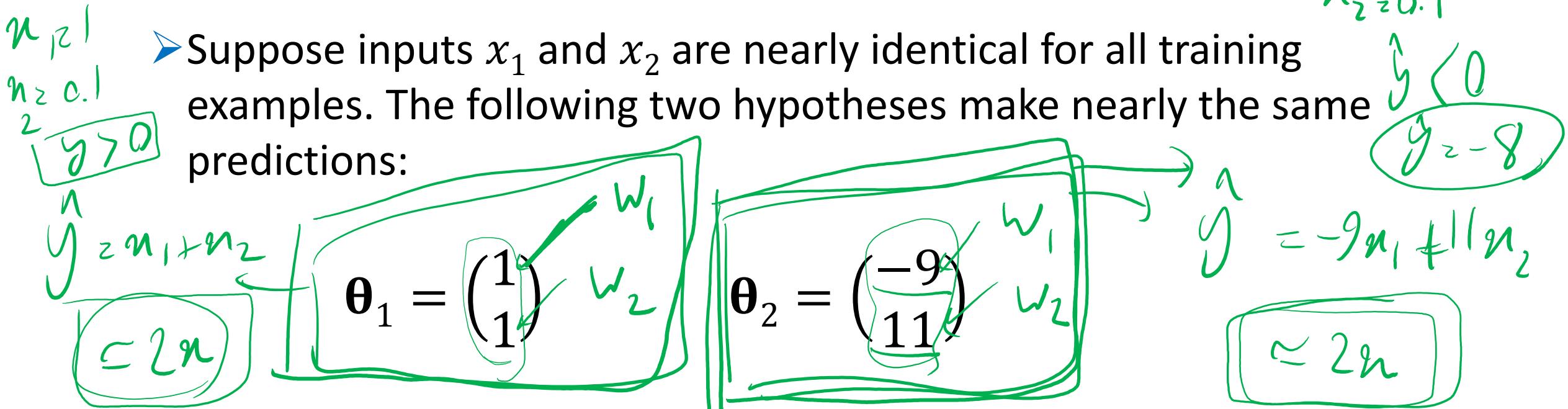
$$y = 0.1x^5 + 0.2x^4 + 0.75x^3 - x^2 - 2x + 2$$
$$y = -7.2x^5 + 10.4x^4 + 24.5x^3 - 37.9x^2 - 3.6x + 12$$



L² Regularization

➤ Another reason we want parameters (weights) to be small:

➤ Suppose inputs x_1 and x_2 are nearly identical for all training examples. The following two hypotheses make nearly the same predictions:



➤ But the second model might make weird predictions if the test distribution is slightly different (e.g. x_1 and x_2 match less closely).

L² Regularization

- We can **encourage the parameters to be small** by adding an L^2 penalty (regularizer) to our cost function:

$$\frac{1}{2} \|\theta\|^2 = \frac{1}{2} \sum_j \theta_j^2$$

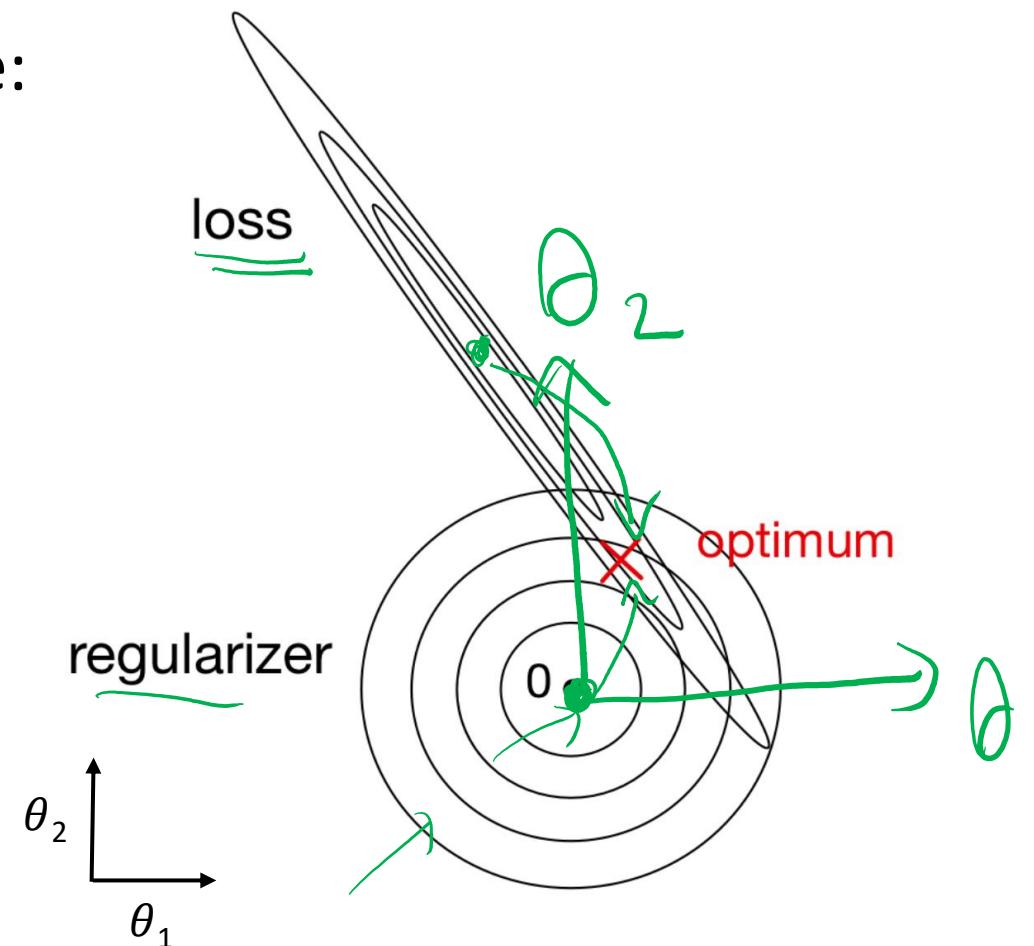
hyperparameter to be tuned

$$J_{reg} = J + \frac{\lambda}{2} \sum_j \theta_j^2$$

- The **regularized cost function** makes a tradeoff between fit to the data and the norm of the weights vector.

L^2 Regularization

- The geometric picture:



L² Regularization

$$\mathcal{J}(w, b) = \frac{1}{2N} \sum_{i=1}^N (y^{(i)} - t^{(i)})^2$$

- We can encourage the weights to be small by choosing as our regularizer the L^2 penalty.

$$\mathcal{R}(w) = \underbrace{\frac{1}{2} \|w\|^2}_{\text{Note: to be pedantic, the } L^2 \text{ norm is Euclidean distance, so we're really regularizing the squared } L^2 \text{ norm.}} = \frac{1}{2} \sum_j w_j^2.$$

- Note: to be pedantic, the L^2 norm is Euclidean distance, so we're really regularizing the *squared* L^2 norm.
- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.

$$\mathcal{J}_{\text{reg}} = \underbrace{\mathcal{J}}_{\text{Here, } \lambda \text{ is a hyperparameter that we can tune using a validation set.}} + \lambda \underbrace{\mathcal{R}}_{\text{The regularized cost function makes a tradeoff between fit to the data and the norm of the weights.}} = \mathcal{J} + \frac{\lambda}{2} \sum_j w_j^2$$

L² Regularization

- Recall the gradient descent update:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}}$$

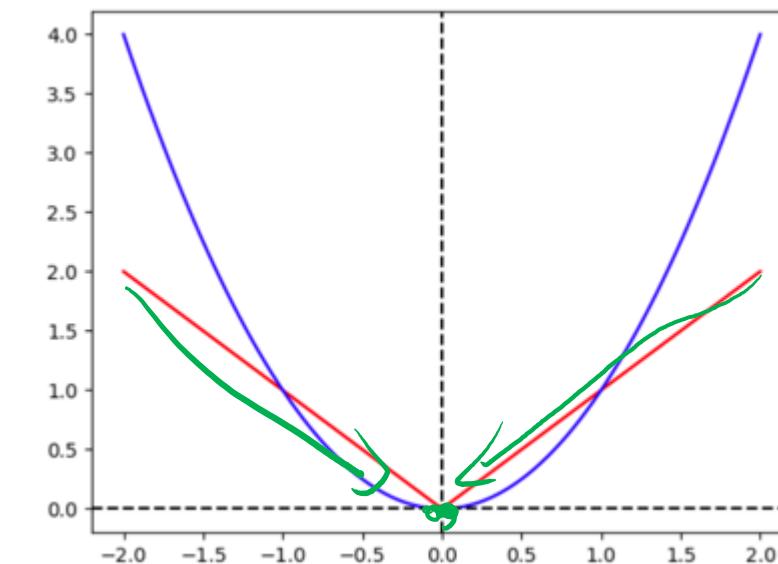
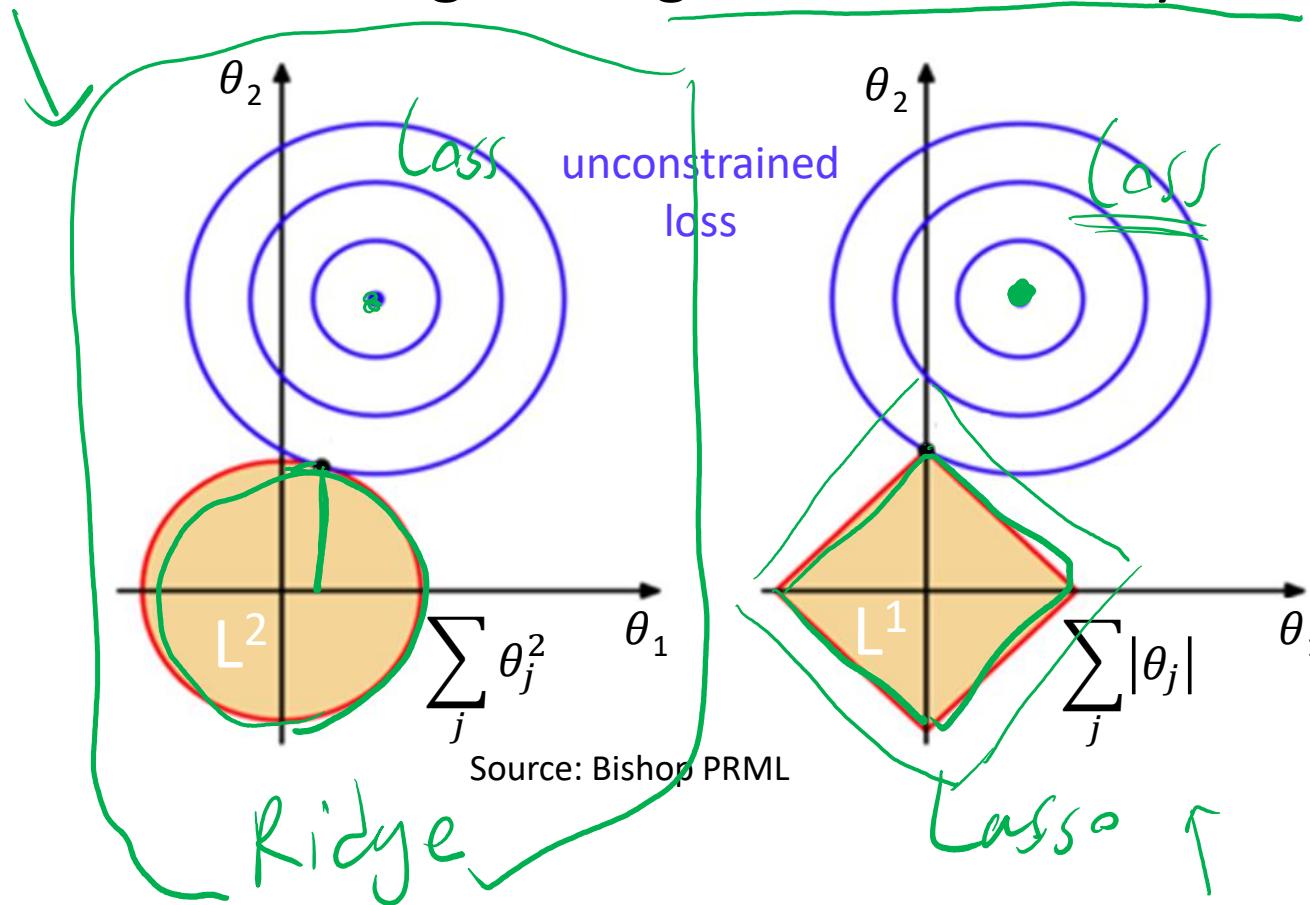
- The gradient descent update of the regularized cost has an interesting interpretation as **weight decay**:

$$\begin{aligned}\mathbf{w} &\leftarrow \mathbf{w} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \frac{\partial \mathcal{R}}{\partial \mathbf{w}} \right) \\ &= \mathbf{w} - \alpha \left(\frac{\partial \mathcal{J}}{\partial \mathbf{w}} + \lambda \mathbf{w} \right) \\ &= (1 - \alpha \lambda) \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \end{aligned}$$

↙ 0.999

L¹ vs L² Regularization

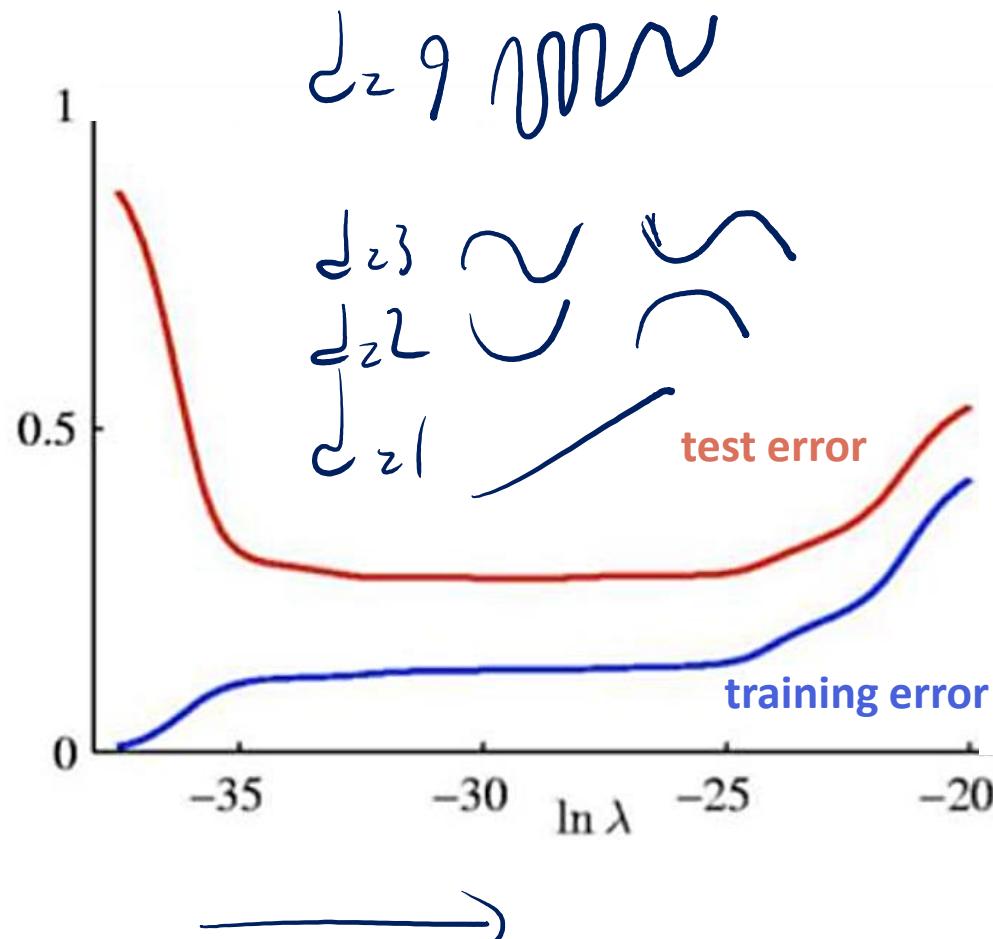
- The L^1 norm (or sum of absolute values) is another regularizer that encourages weights to be exactly zero.



L1-regularization tends to push parameters to zero

Generalization

- Training and test error as a function of regularization parameter λ



Overfit

$$\frac{1}{2} (5321)^2$$
$$\frac{1}{2} (1061800)^2$$

	$\lambda = 0$	$\lambda = e^{-18}$	$\lambda = 1$
θ_0	0.35	0.35	0.13
θ_1	232.37	4.74	-0.05
θ_2	-5321.83	-0.77	-0.06
θ_3	48568.31	-31.97	-0.05
θ_4	-231639.30	-3.89	-0.03
θ_5	640042.26	55.28	-0.02
θ_6	-1061800.52	41.32	-0.01
θ_7	1042400.18	-45.95	-0.00
θ_8	-557682.99	-91.53	0.00
θ_9	125201.43	72.68	0.01

ordinary least square regression

6 f

33

10:29

Ineffective Batch Size

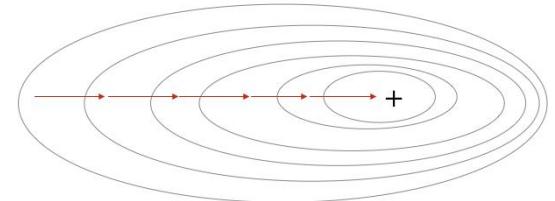
- Q: What happens if the batch size is too small? Too large?

➤ Too large:

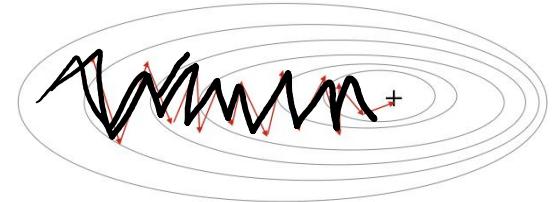
- Computationally expensive
- Average loss might not change very much as batch size grows

➤ Too small:

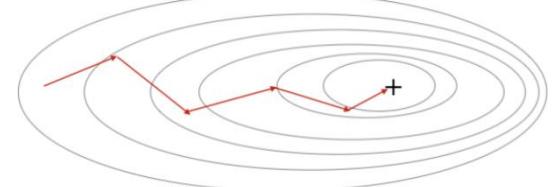
- We optimize a (possibly very) different loss function at each iteration
- Noisy
- SGD may actually take longer



Stochastic Gradient Descent



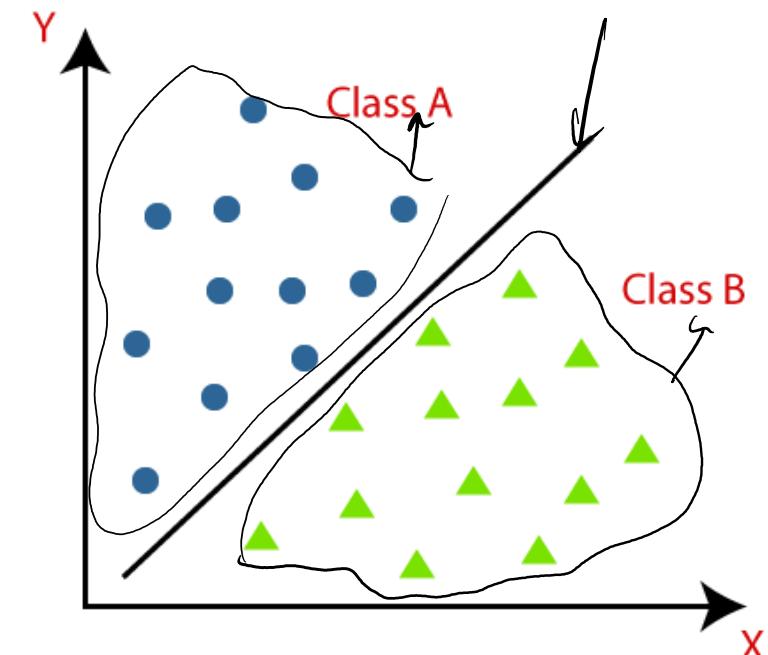
Mini-Batch Gradient Descent



Classification

Overview

- **Classification:** predicting a discrete-valued target
- **Binary classification:** number of target values is 2 (binary-valued)
- Examples:
 - predict where a patient has a disease given presence or absence of various symptoms
 - classify e-mails as spam or non-spam
 - predict whether a financial transaction is fraudulent



Binary Classification

- We can start with our linear function of x , but now we introduce a threshold :

$$z = \theta^T x + b$$

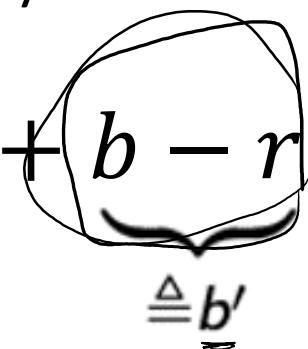
$$\hat{y} = \begin{cases} 1 & \text{if } z \geq r \\ 0 & \text{if } z < r \end{cases}$$

Binary Classification

➤ Eliminating the threshold

- Can make the threshold $r = 0$ without loss of generality.

$$\theta^T \mathbf{x} + b \geq 0 \quad \longleftrightarrow \quad \theta^T \mathbf{x} + b - r \geq 0$$



$$\trianglelefteq b'$$

➤ Eliminating the bias

- Add a dummy feature x_0 which always takes the value 1. The weight θ_0 is equivalent to the bias.

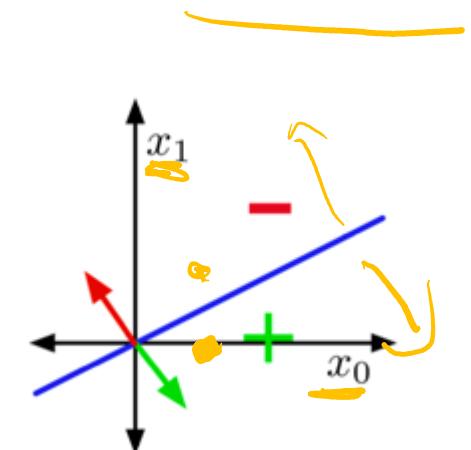
$$\mathbf{z} = \underline{\theta^T \mathbf{x}}$$

$$\hat{y} = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

The Geometric Picture

- Here we're visualizing the logical NOT example
- Training examples are points
- Hypothesis are half-spaces whose boundaries pass through the origin
- This boundary is the decision boundary
 - In 2-D it's a line, but think of it as a hyperplane
- If the training examples can be separated by a linear decision rule, they are linearly separable.

➤ Input Space, or Data Space:



NOT

x_0	x_1	t
1	0	1
1	1	0

The Geometric Picture

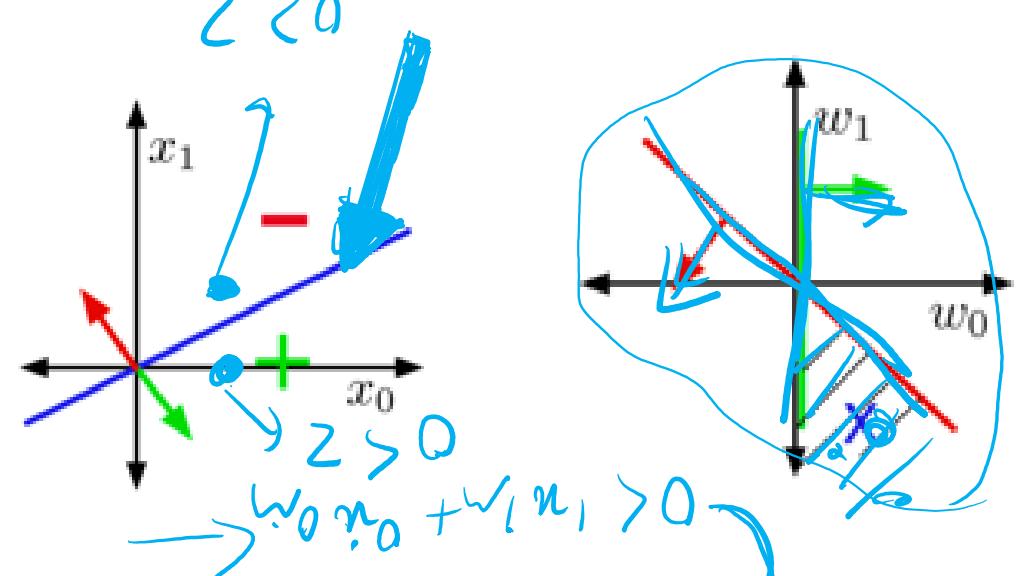
- Hypotheses are points
- Training examples are half-spaces whose boundaries pass through the origin
- The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is feasible

How do you optimize?

$$\begin{aligned} w_0 &= 1 \\ w_1 &= -2 \end{aligned}$$

$$w_0 n_0 + w_1 n_1 \geq 0$$

Weight Space



$$\begin{aligned} w_0 &> 0 \\ w_0 + w_1 &< 0 \end{aligned}$$
$$w_0 n_0 + w_1 n_1 < 0$$

Example 1

NOT

w_0	w_1	
x_0	x_1	t
1	0	1
1	1	0

Example 2

$$w_0x_0 + w_1x_1 + w_2x_2$$

$$\left\{ \begin{array}{l} w_0 < 0 \\ w_0 + w_2 < 0 \\ w_0 + w_1 < 0 \\ w_0 + w_1 + w_2 > 0 \end{array} \right.$$

$$\boxed{\begin{array}{l} w_0 = -1.5 \\ w_1 = 1 \\ w_2 = 1 \end{array}}$$

AND

x_0	x_1	x_2	t
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

How do you optimize?

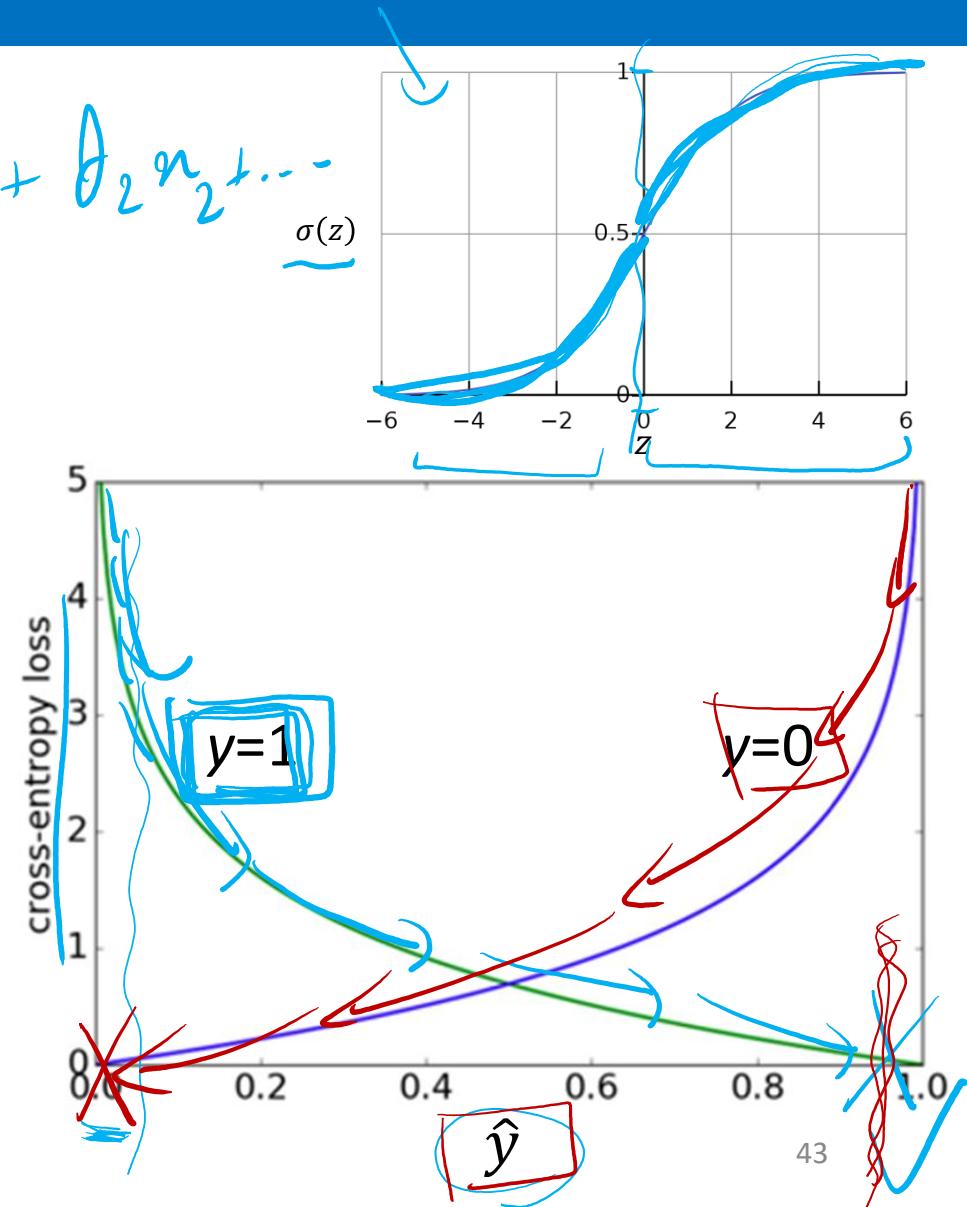
Logistic Regression

$$z = \theta^T x$$
$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$z = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$
$$= \sum_i \theta_i x_i$$

- Because the prediction $\hat{y} \in [0, 1]$, we can interpret it as the estimated probability that the label is positive ($y = 1$).
- **Cross-entropy loss captures** this intuition:

$$\begin{aligned}\mathcal{L}_{CE}(\hat{y}, y) &= \begin{cases} -\log \hat{y} & \text{if } y = 1 \\ -\log(1 - \hat{y}) & \text{if } y = 0 \end{cases} \\ &= -y \log \hat{y} - (1 - y) \log(1 - \hat{y})\end{aligned}$$



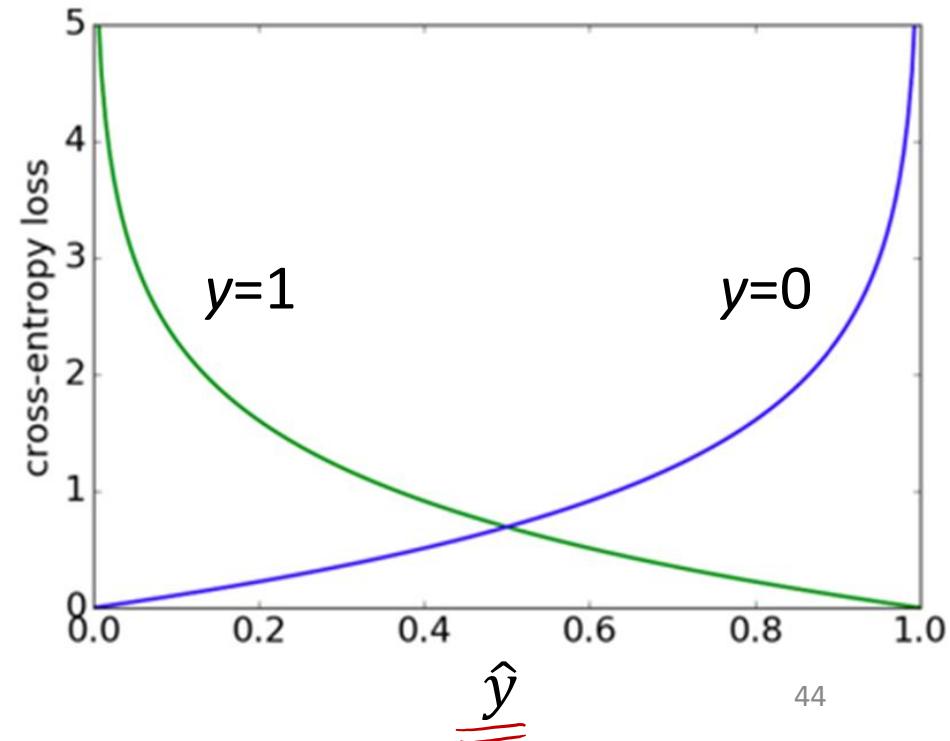
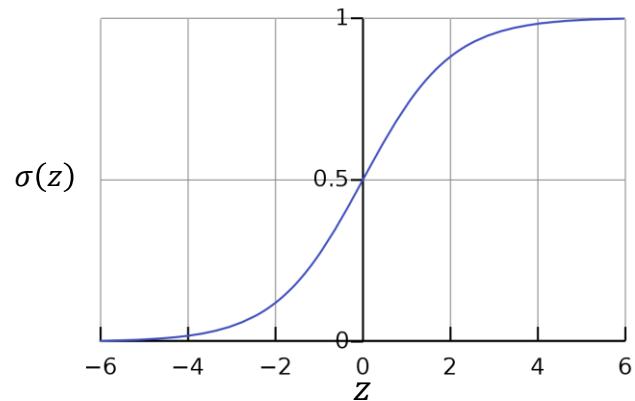
Logistic Regression with Cross-entropy Loss

$$z = \theta^T x$$

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}}$$

- Being 99% confident in the wrong answer is much more wrong than being only 90% confident.

$$\mathcal{L}_{CE}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$



Logistic Regression with Cross-entropy Loss

➤ Computation Problem:

- If $\hat{y} = \sigma(z)$ gets too close to 0 or 1, it may cause very subtle and hard-to-find bugs:

$$\hat{y} = \sigma(z)$$

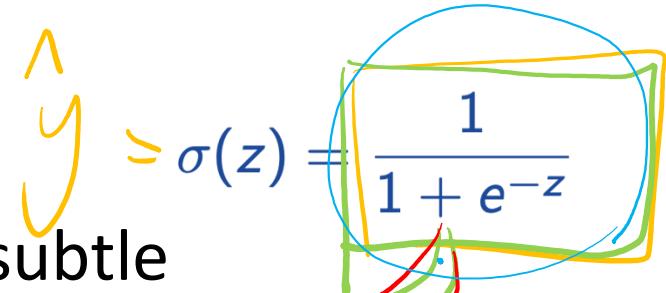
$$\mathcal{L}_{CE}$$

$$\mathcal{L}_{CE}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \Rightarrow \hat{y} \approx 0$$

$$\mathcal{L}_{CE}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \Rightarrow \text{computes } \log 0$$

- Instead, we combine the activation function and the loss into a single logistic-cross-entropy function.

$$\mathcal{L}_{LCE}(\sigma(z), y) = y \log(1 + e^{-z}) + (1 - y) \log(1 + e^z)$$



Example:

$$\mathcal{L}_{CE}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathcal{L}_{LCE}(\sigma(z), y) = y \log(1 + e^{-z}) + (1 - y) \log(1 + e^z)$$

Weight Updates

- Comparison of gradient descent updates:

- Linear regression: \hat{y}

\hat{y} z z

$$\theta \leftarrow \theta - \frac{\alpha}{N} \sum_{i=1}^N (\hat{y}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}$$

$$z = \theta^T \mathbf{x} + b$$

Using activation function

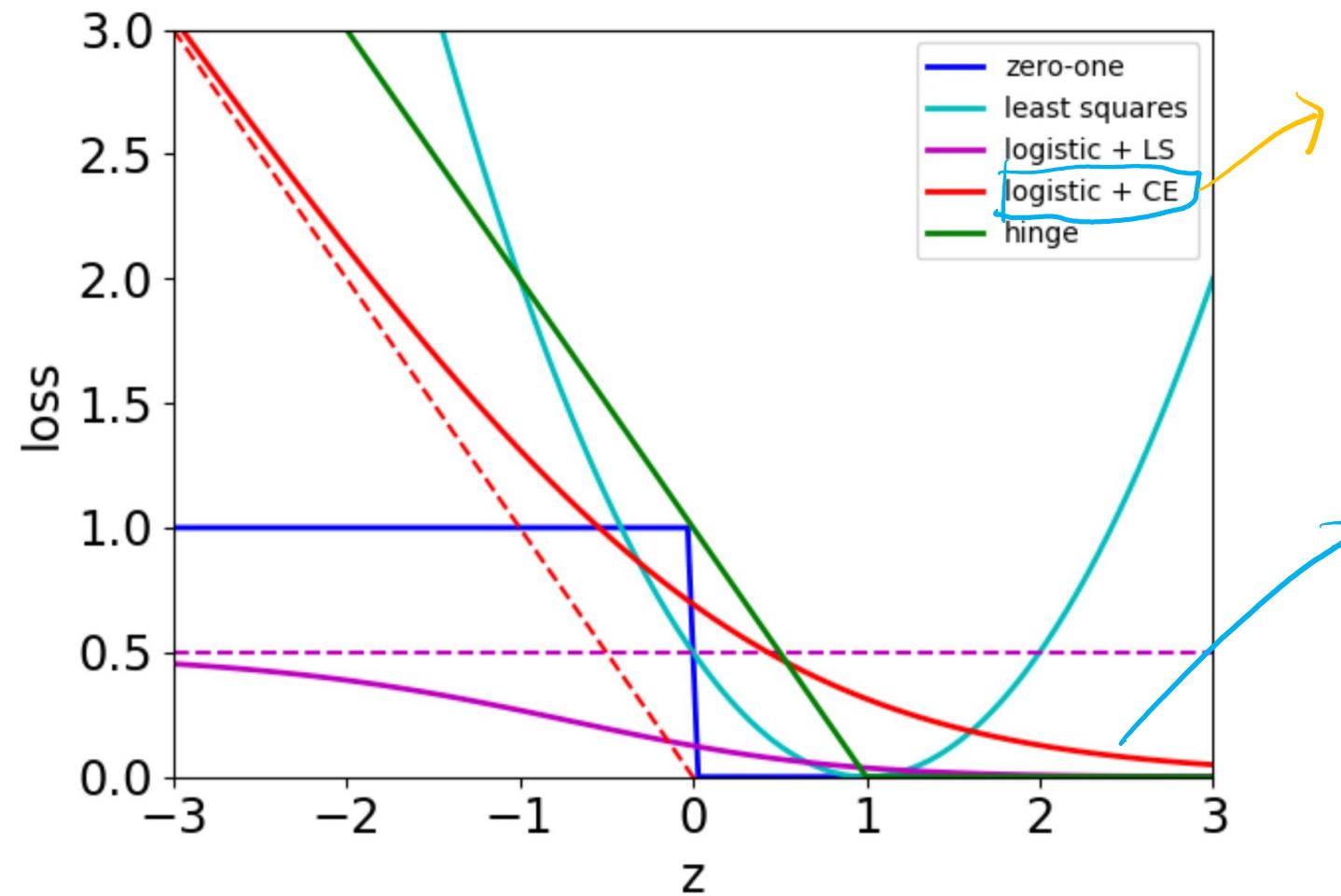
$$\hat{y} = \frac{1}{1 + e^{-z}}$$

- Logistic regression: \hat{y}

\hat{y} z $\sigma(z)$ z $\frac{1}{1+e^{-z}}$

$$\theta \leftarrow \theta - \frac{\alpha}{N} \sum_{i=1}^N (\hat{y}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}$$

Loss Summary



11 : 05

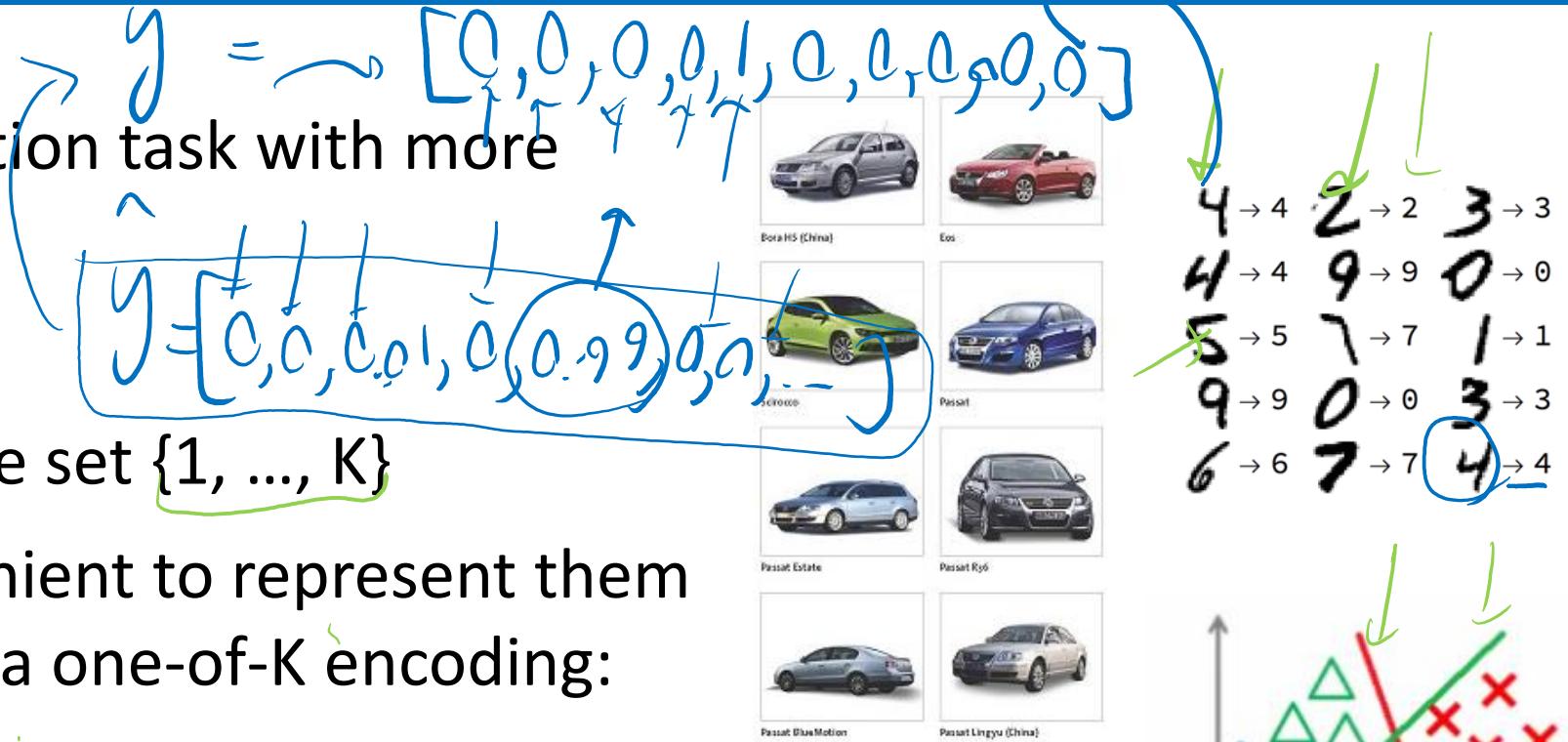
Multiclass Classification

Multiclass Classification

- What about classification task with more than two categories?
- Targets form a discrete set $\{1, \dots, K\}$
- It's often more convenient to represent them as one-hot vectors or a one-of-K encoding:

$y = [0, \dots, 0, 1, 0, \dots, 0]$ $^{1 \times k}$ entry k is a 1, all others are 0

class k



Multiclass Classification

➤ Now there are D input dimensions and K output dimensions, so we need $K \times D$ parameters, which we arrange as a matrix Θ .

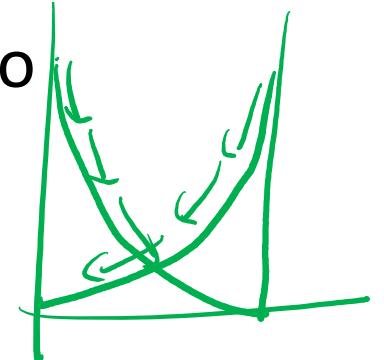
➤ Also, we have a K -dimensional vector \mathbf{b} of biases.

➤ Linear predictions:

$$\mathbf{x} = [x_0, x_1, x_2] \quad 1 \times 3$$

$$\mathbf{x} = [x_0, x_1, x_2] \quad \text{Vectorized: } 1 \times 3$$

$$\begin{aligned} \Theta^T &= [\theta_0, \theta_1, \theta_2] \\ z_k &= \sum_{j=0}^3 \theta_{kj} x_j + b_k \quad z \in \mathbb{R} \\ \Theta &= \begin{bmatrix} \theta_0^{(1)}, \theta_1^{(1)}, \theta_2^{(1)} \\ \theta_0^{(2)}, \theta_1^{(2)}, \theta_2^{(2)} \end{bmatrix} \\ z &= \Theta \mathbf{x} + \mathbf{b} \quad z \in \mathbb{R}^K \\ z &= \begin{bmatrix} z^{(1)}, z^{(2)}, \dots, z^{(K)} \end{bmatrix} \quad \mathbb{R}^K \end{aligned}$$



$$\hat{\mathbf{y}} = \left[\frac{e^1}{e^1 + e^2 + e^3}, \frac{e^2}{e^1 + e^2 + e^3}, \frac{e^3}{e^1 + e^2 + e^3} \right]^T \in \{0, 0, 0.9\}$$

Activation Function

- A natural activation function to use is the softmax function, a multivariable generalization of the logistic (sigmoid) function:

$$\hat{y}_k = \text{softmax}(z_1, \dots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}$$

Softmax makes differences larger – pushes values close to 1 to 1, values close to 0 to 0

- The input are called the logits.

$z = [1, 3, 7]$
➤ Outputs are positive and sum to 1 (interpreted as probabilities)

$$\hat{y} = \frac{1}{e^1 + e^3 + e^7} [e^1, e^3, e^7] = [0.002, 0.01, 0.98]$$

→ $\hat{y} = [0, 0, 1]$

Loss Function

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

Binary

$$\mathcal{L}_{CE}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Multi-class

$$\mathcal{L}_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = - \sum_{k=1}^K y_k \log \hat{y}_k$$

$$= -\underbrace{\mathbf{y}^T \log \hat{\mathbf{y}}}_{\text{gini index}}$$

≈ gini index

Softmax Regression

➤ Softmax regression:

$$\rightarrow z = \theta \mathbf{x} + \mathbf{b}$$

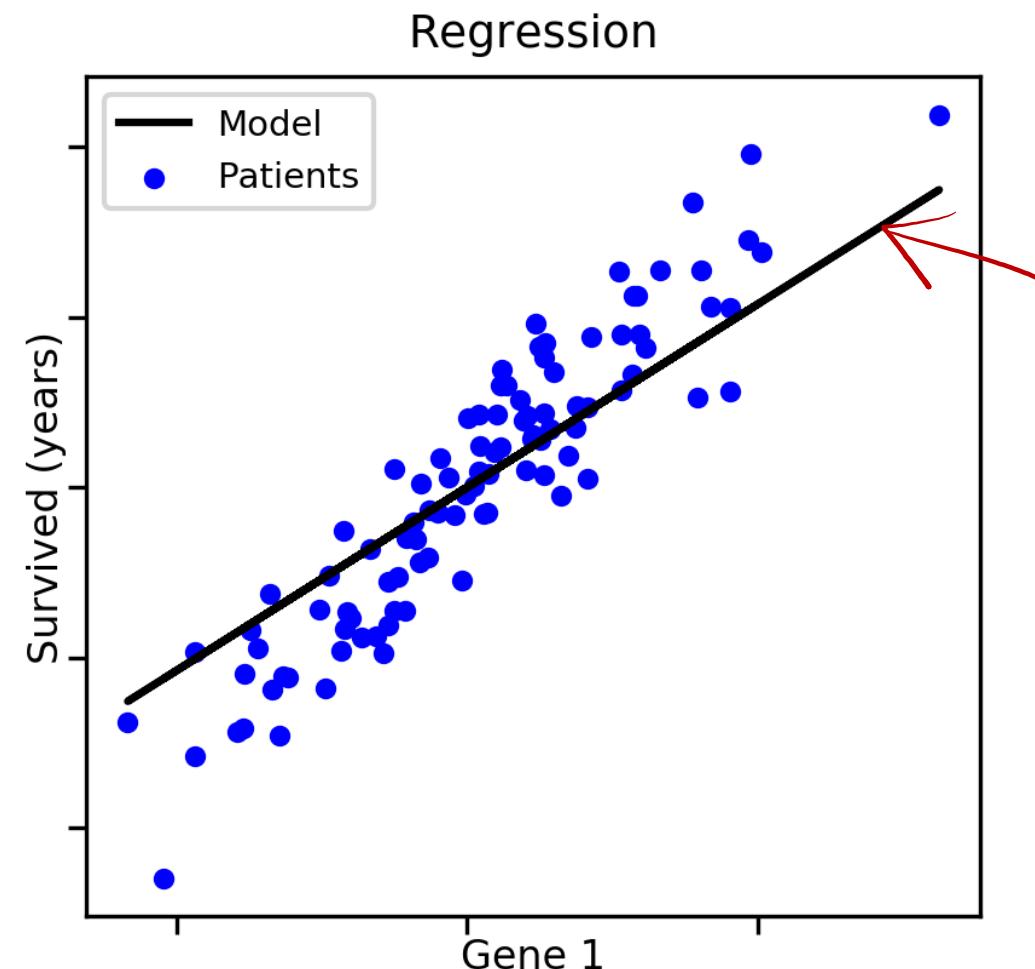
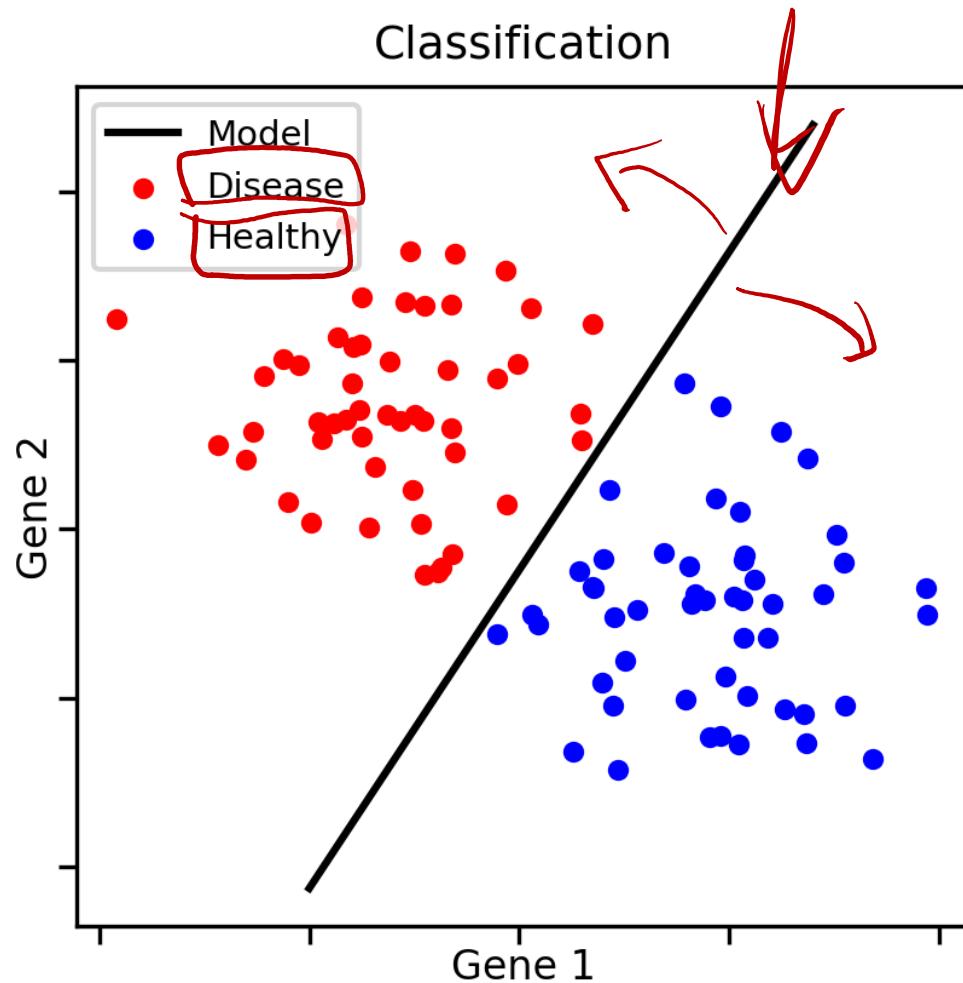
$$\rightarrow \hat{\mathbf{y}} = \text{softmax}(z)$$

$$\rightarrow \mathcal{L}_{CE}(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}^T \log \hat{\mathbf{y}}$$

➤ Gradient descent updates:

$$\frac{\partial \mathcal{L}_{CE}}{\partial \mathbf{z}} = \hat{\mathbf{y}} - \underline{\mathbf{y}}$$

Summary



Practice in Google Colab

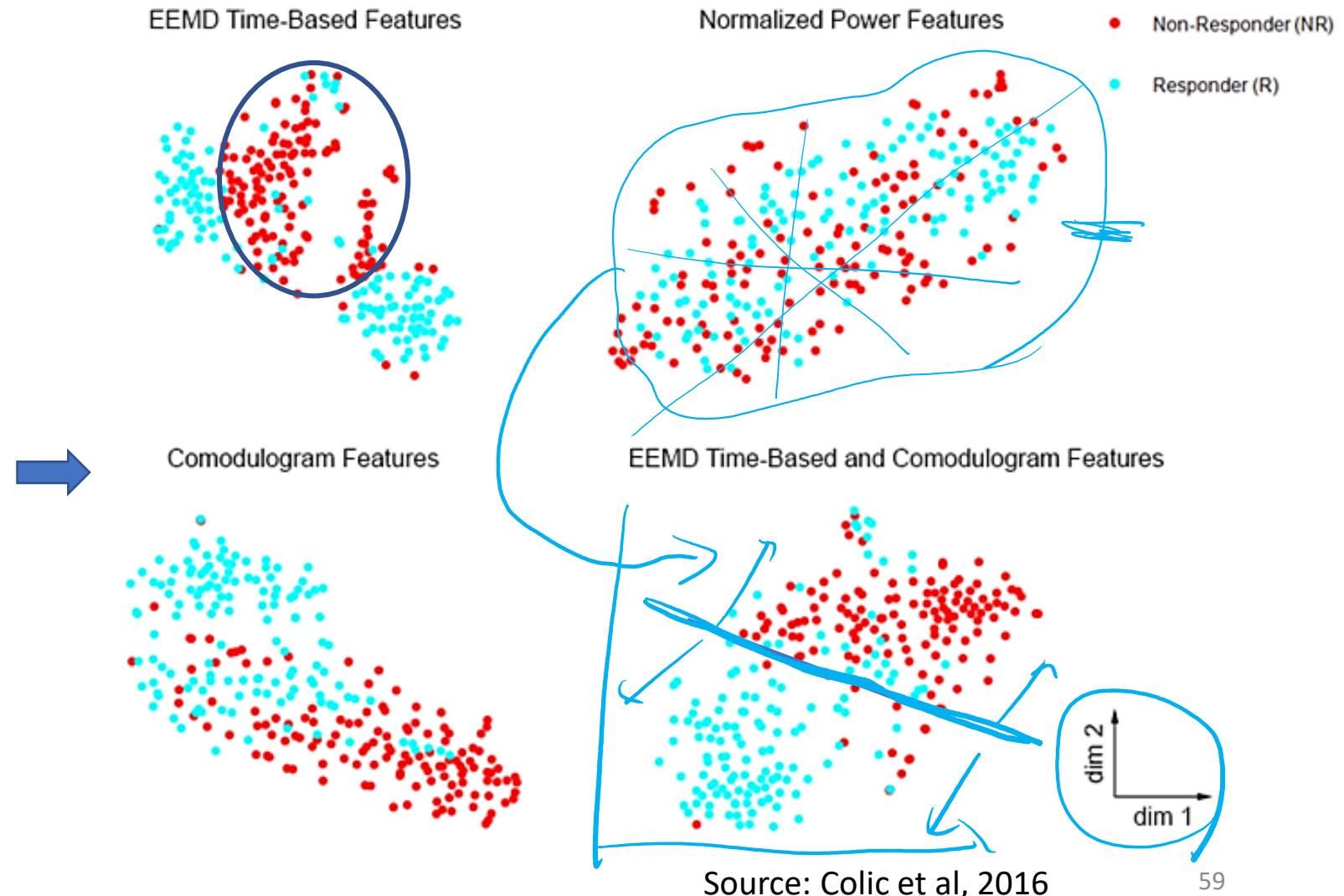
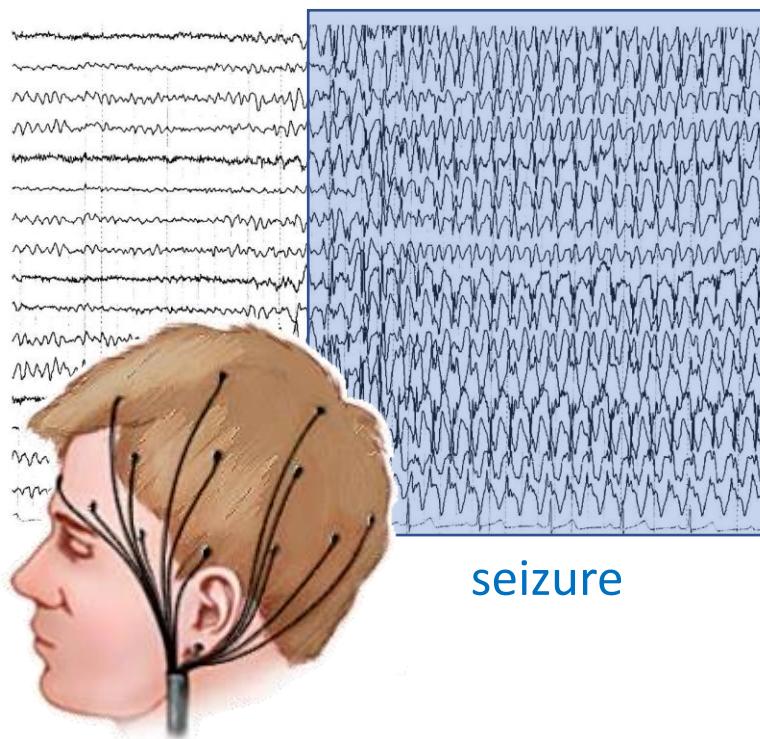
Neural Networks

Challenges with Feature Maps

- Feature maps can be useful for solving nonlinear regression and classification problems.
- Have several limitations:
 - The feature maps must be selected in advance
 - Not always easy to pick a good feature map and can take a long time to craft
 - In high dimensions the feature representations can explode

Example of Feature Engineering:

Objective: Predict **responders** from **non-responders** given raw electroencephalogram data.



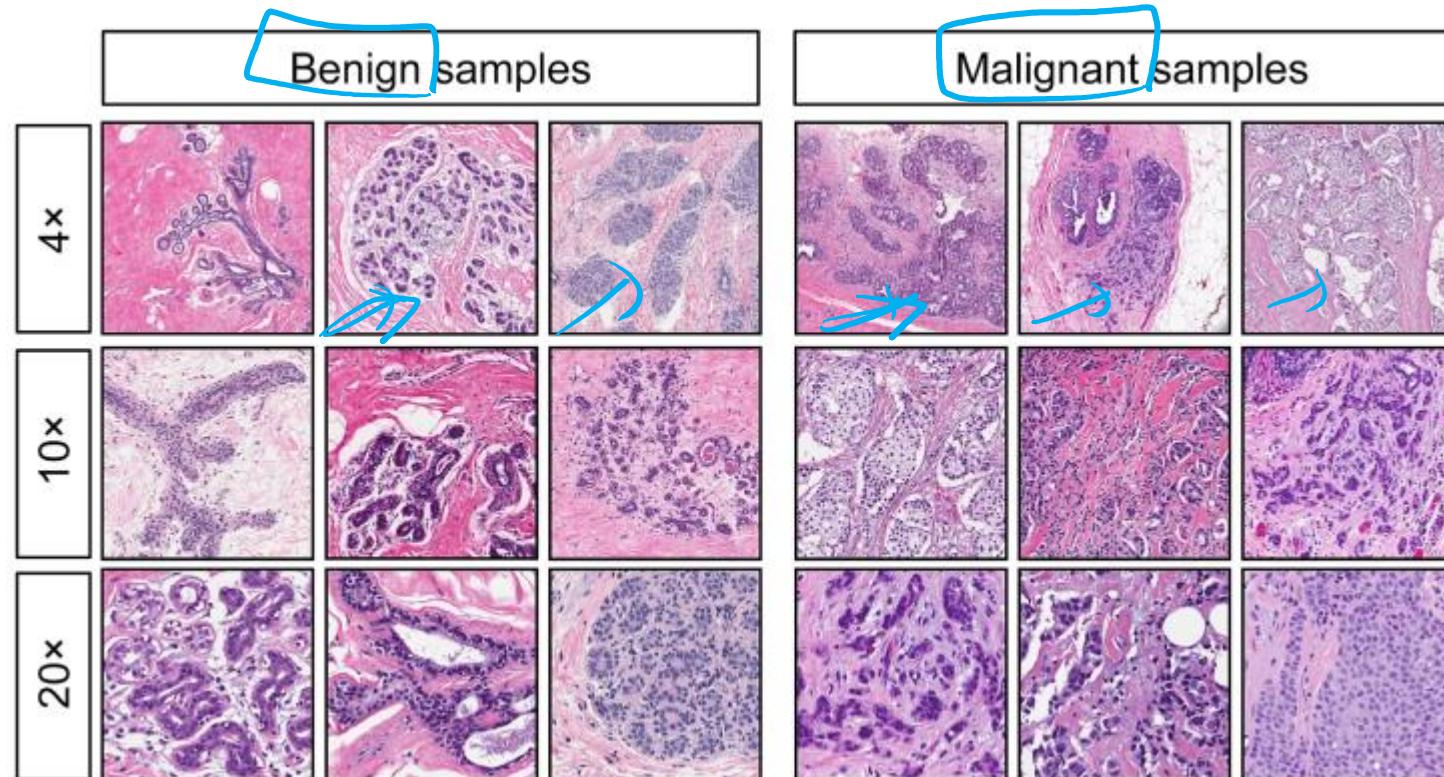
Challenges with Feature Maps

We need an algorithm that can learn good features for nonlinear regression and classification.

Motivating Example: Tumor Classification

Objective: Classify an image of a biopsy as cancerous (malignant) or benign (tumor)

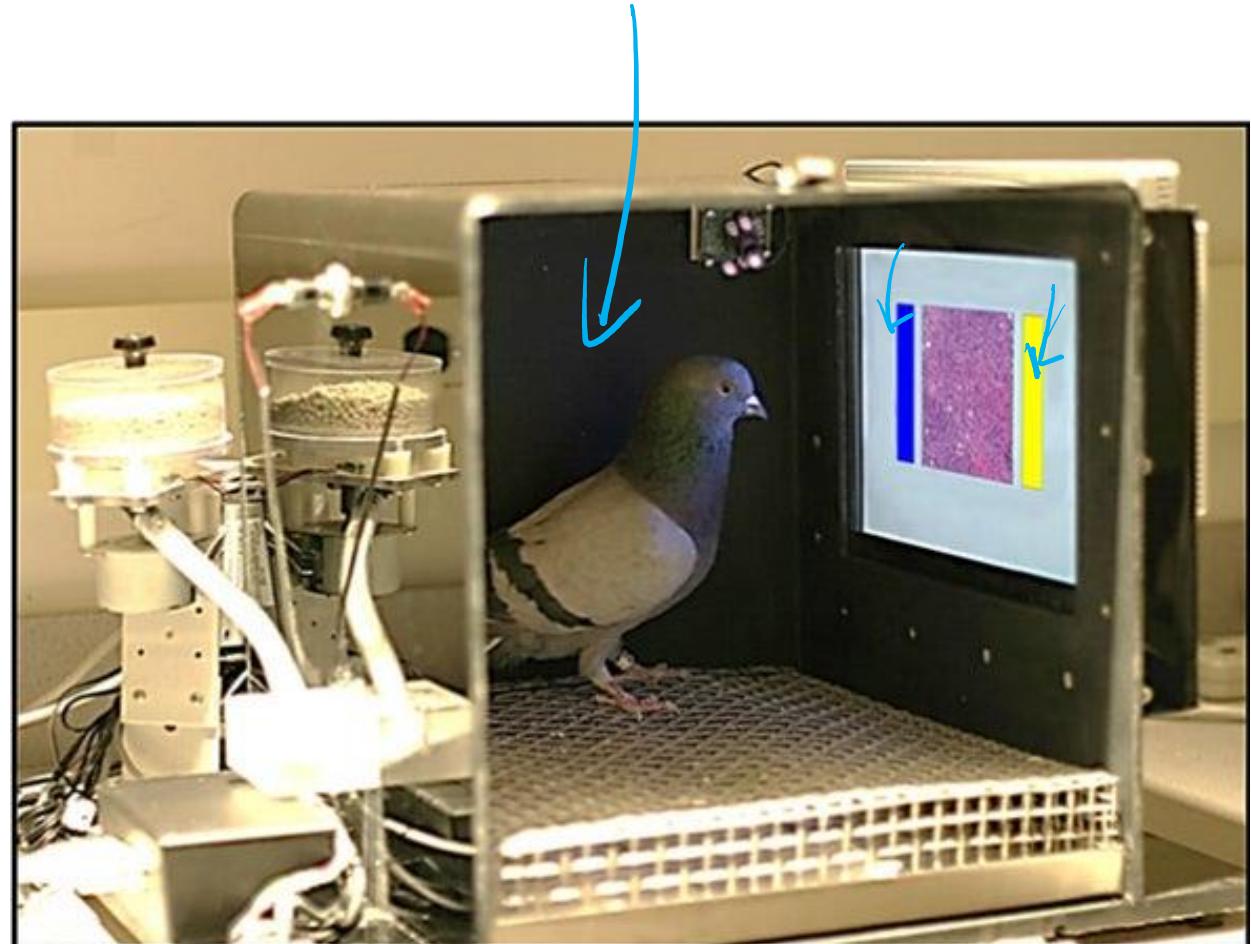
- Pathologists/radiologists train for years to do this!
- How would you solve this problem?



Source: [Levenson et al., 2015](#)

Maybe We Can Use Pigeons

- **Train a Pigeon!**
- A 2015 study suggests that the common pigeon can reliably distinguish between benign versus malignant tumors and, in doing so, could help researchers develop better cancer screening technologies.



Source: [Levenson et al., 2015](#)

Maybe We Can Use Pigeons

Training Algorithm:

1. Show an image of a magnified biopsy to pigeon
2. Pigeon pecks at one of two answer buttons on sides for malignant/benign
3. Correct classifications are rewarded with food pellets



Video: <https://www.youtube.com/watch?v=fIzGjnJLyS0>

Why are we talking about pigeons?

We need to answer similar questions in training a pigeon/artificial neural network:

- How will we reward correct responses?
- How do we train the neural network efficiently?
- How do we know the pigeon **didn't just memorize** the images we showed it?
- What are the ethics of trusting a pigeon to detect cancer?

How do Pigeons Work?

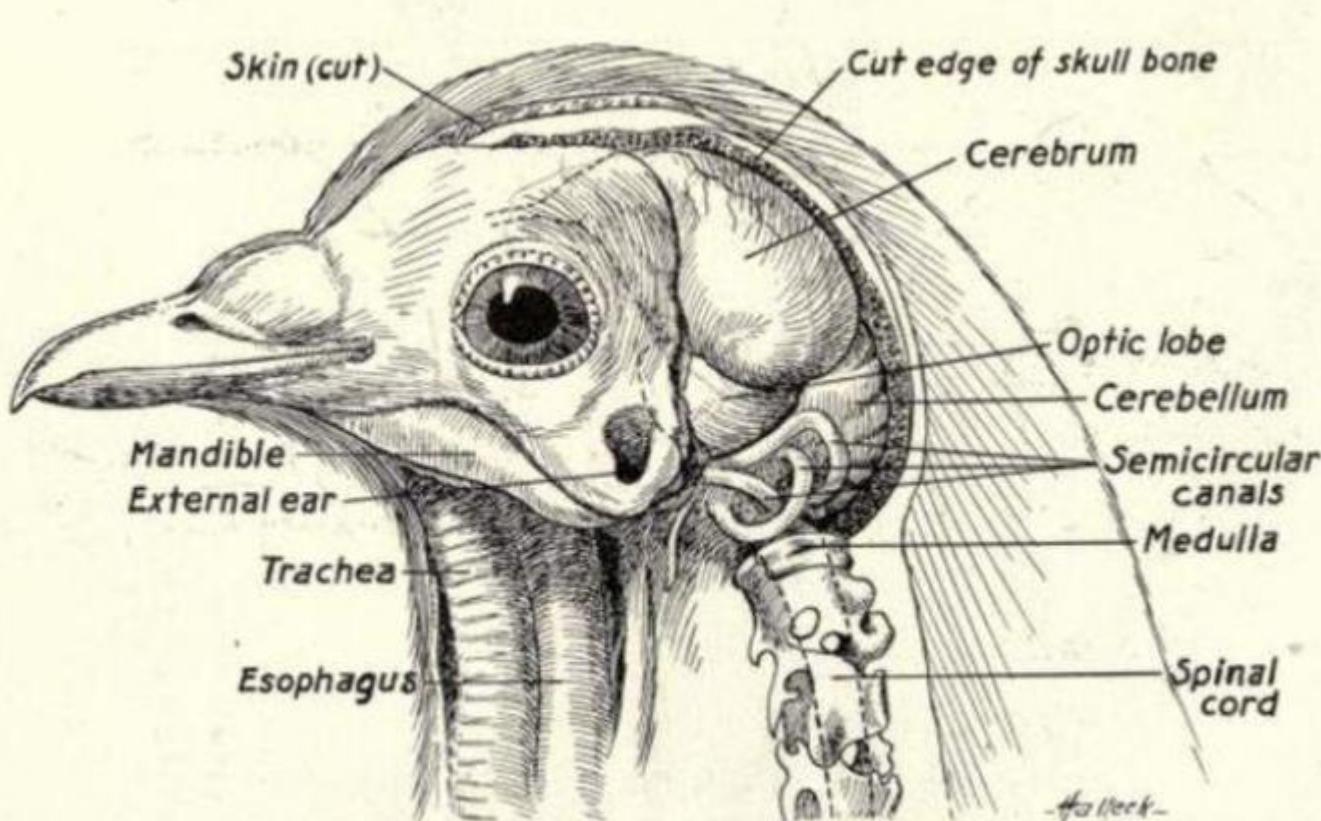
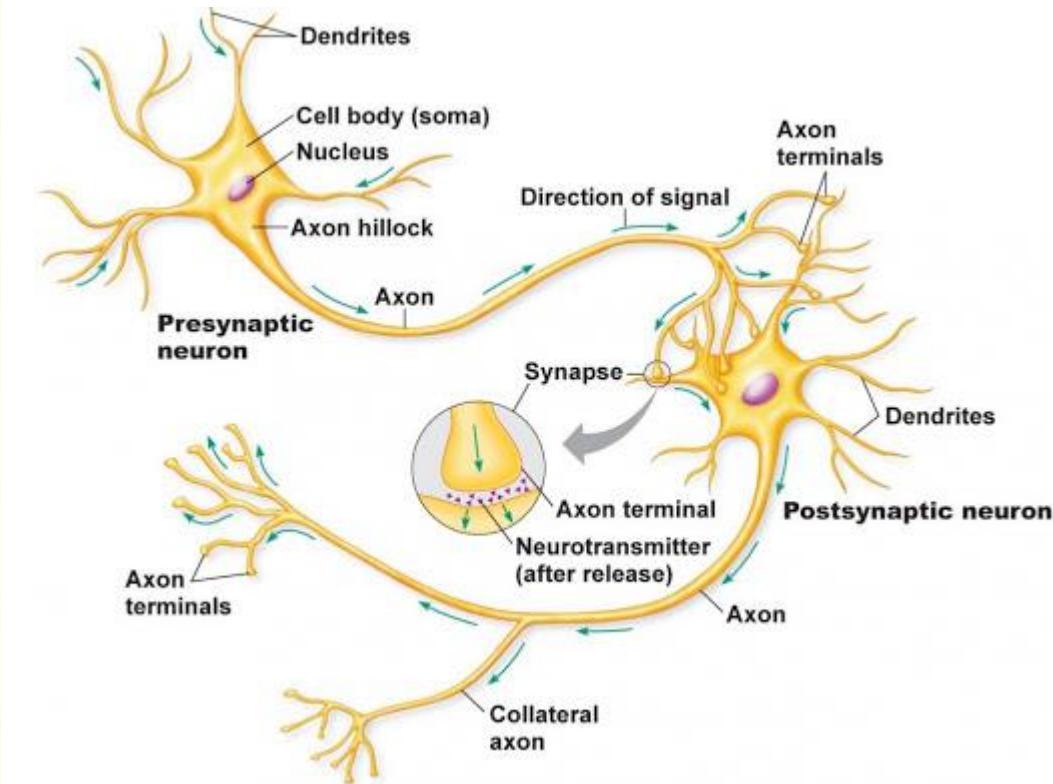


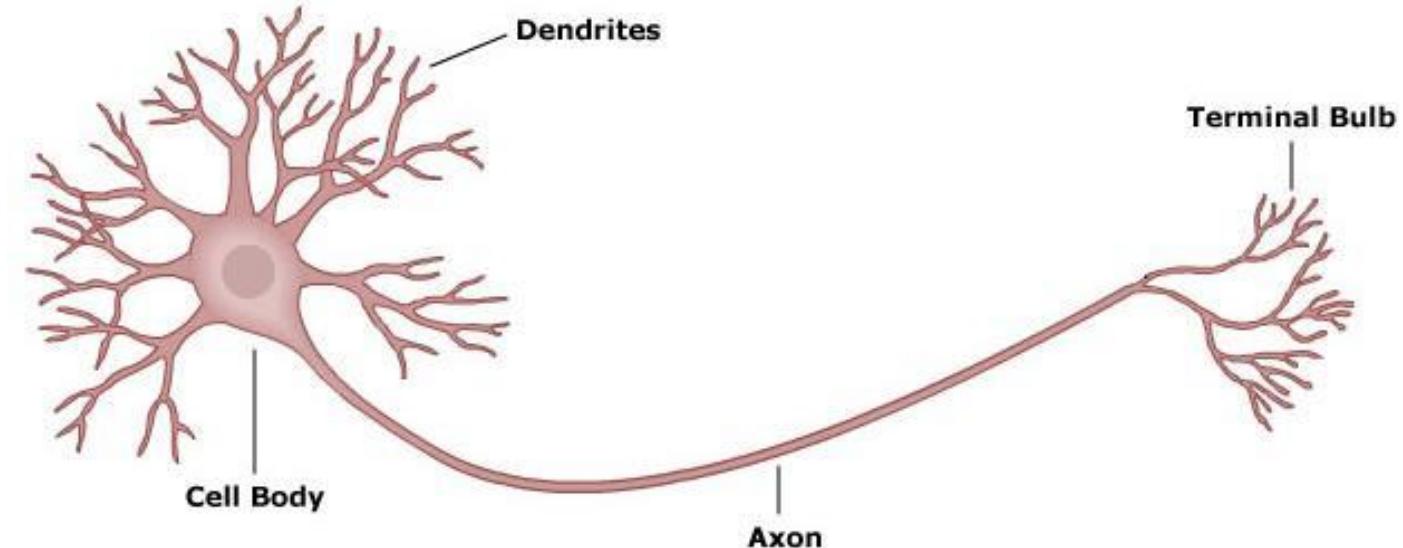
Fig. 167.—Lateral view of the head of a pigeon showing the brain, external auditory opening and semicircular canals.



Source: [Letsmaketeach](#)

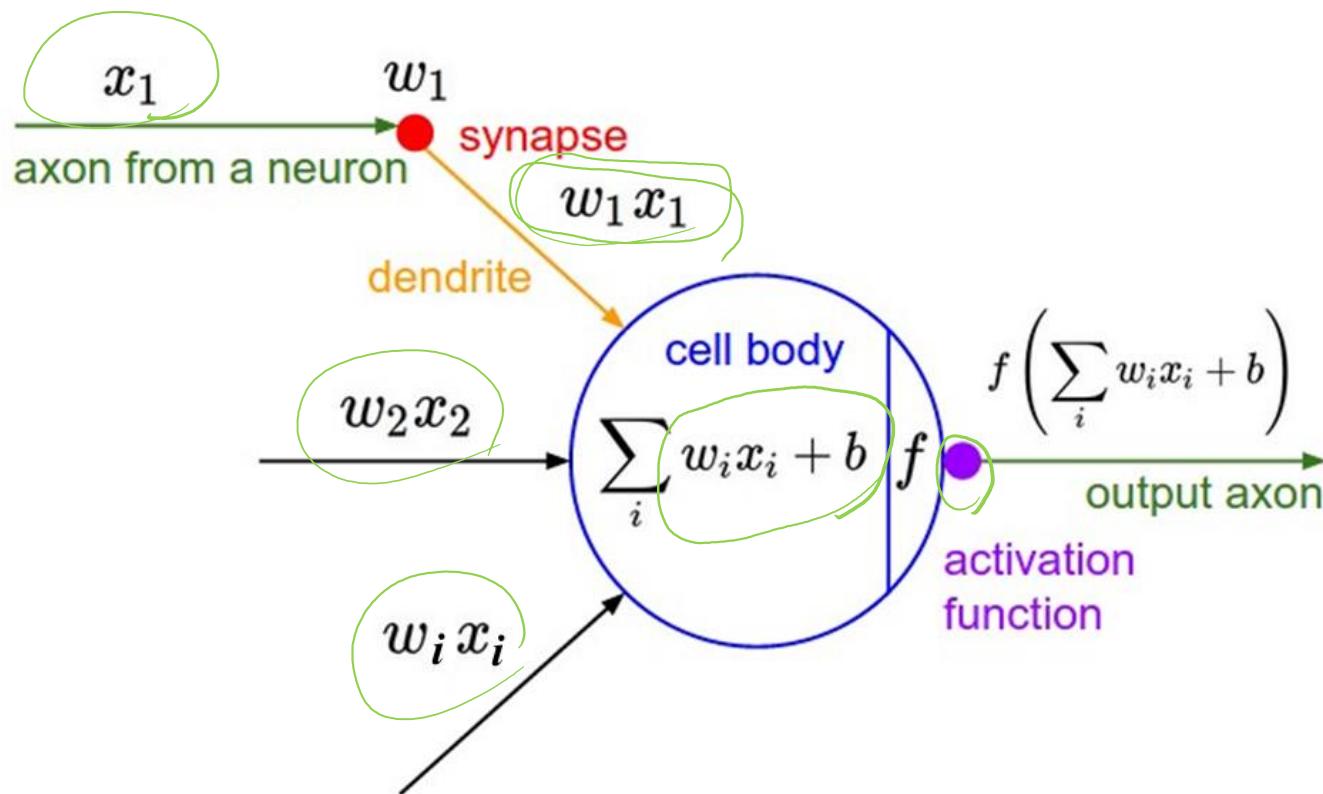
Simplified Neuron Anatomy

- **Dendrites:** are connected to other cells that provide information.
- **Cell body:** consolidates information from dendrites.
- **Axon:** an extension from the cell body that passes information to other neurons.
- **Synapse:** the area where the axon of one neuron and the dendrite of another connect.

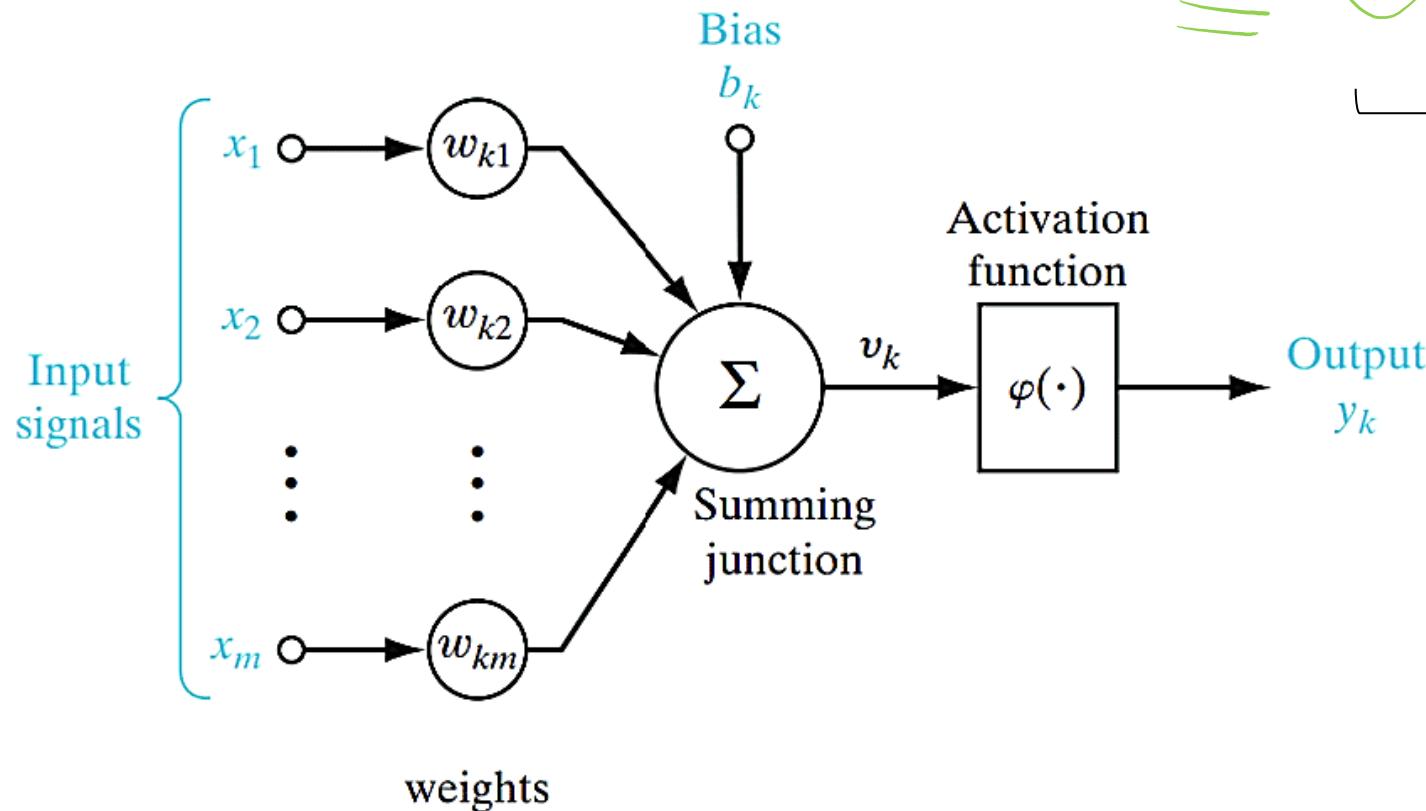


Artificial Neural Network

- Maybe we're not quite ready for pigeon doctors, but we can use the next best thing... an artificial pigeon (**artificial neural network**)



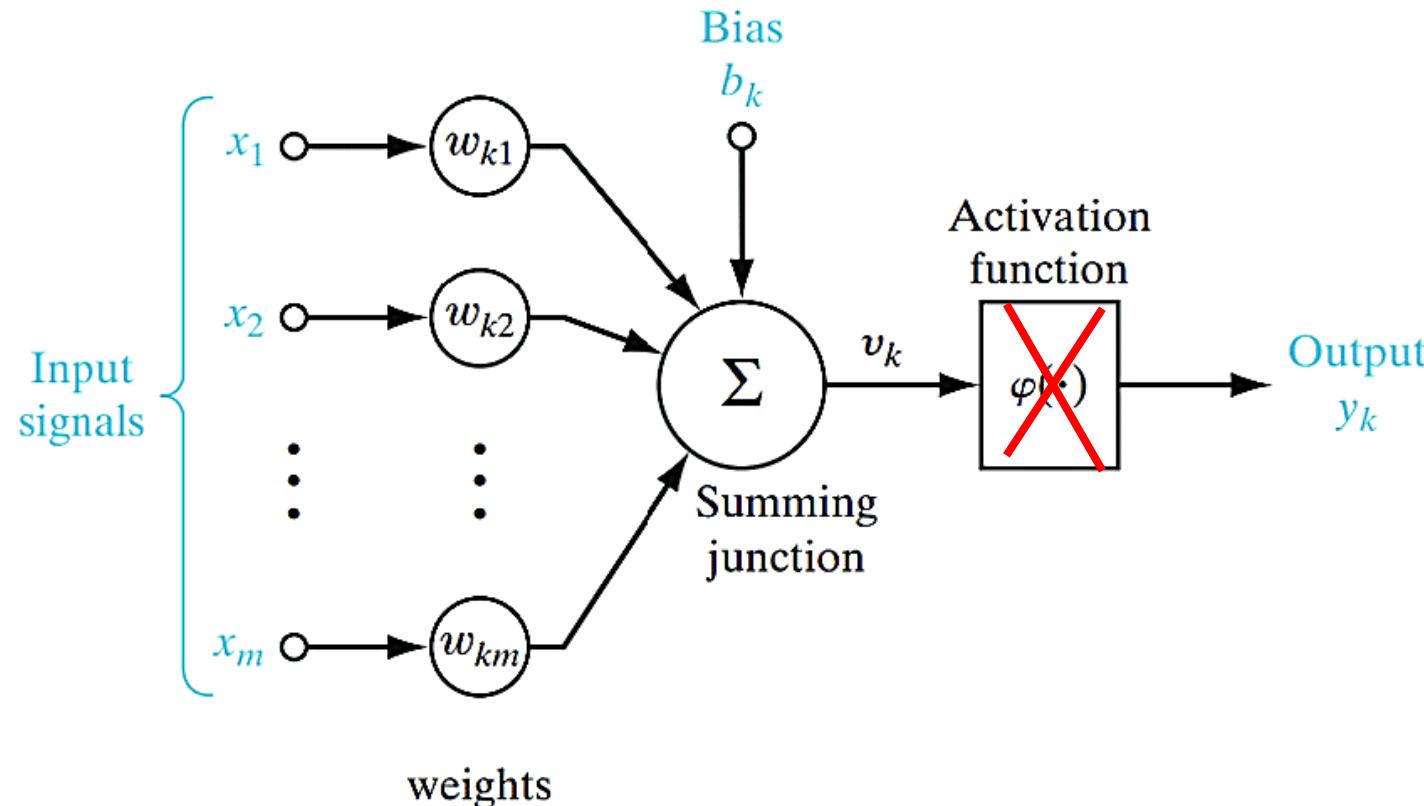
Artificial Neural Network



$$\hat{y} = \varphi \left(\sum_{i=0}^p w_i x_i = \mathbf{w}^T \mathbf{x} \right)$$

does this look familiar?

Artificial Neural Network



$$y = \cancel{\varphi} \left(\sum_{i=0}^p w_i x_i = \mathbf{w}^T \mathbf{x} + b \right)$$

Logistic Regression!

What will we have if
we exclude the
activation function?

Training / Learning Parameters

- In order to train an ANN we should define the error on our predictions.
- This is the same as with linear regression and logistic regression:

Mean Squared Error
(regression)

$$\frac{1}{2N} \sum_{n=1}^N \|\hat{y}_n - y_n\|^2$$

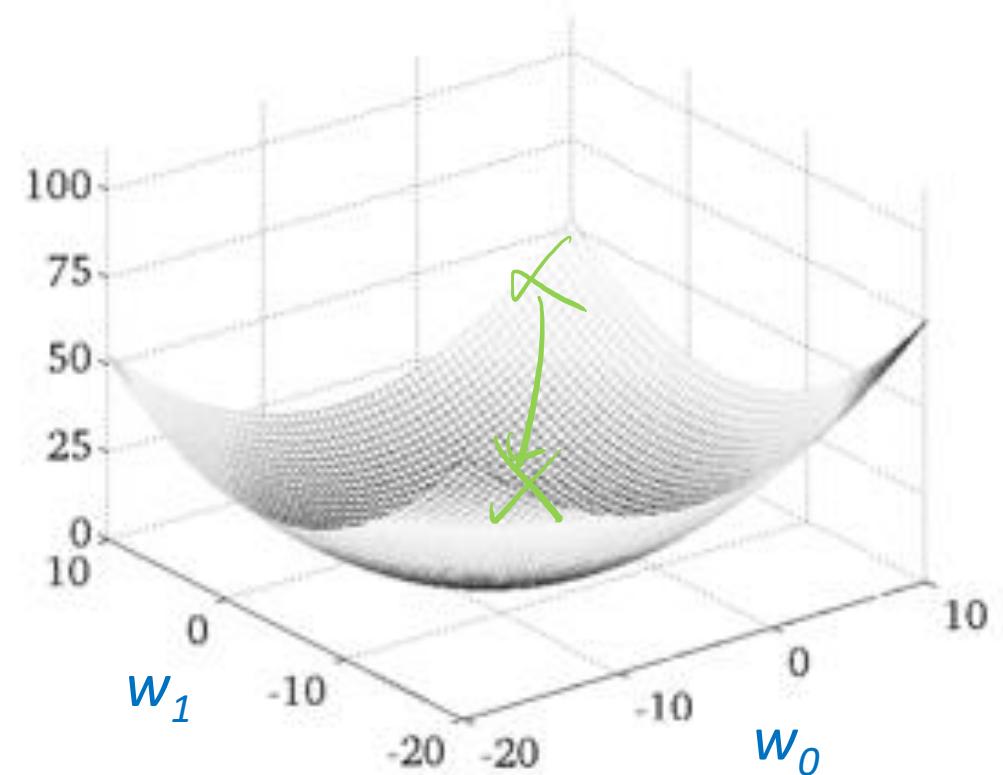
Cross-Entropy Loss
(classification)

$$-\frac{1}{N} \sum_{n=1}^N \sum_{k=1}^K t_{n,k} \log(\hat{y}_{n,k})$$

implement gradient descent to learn parameters!

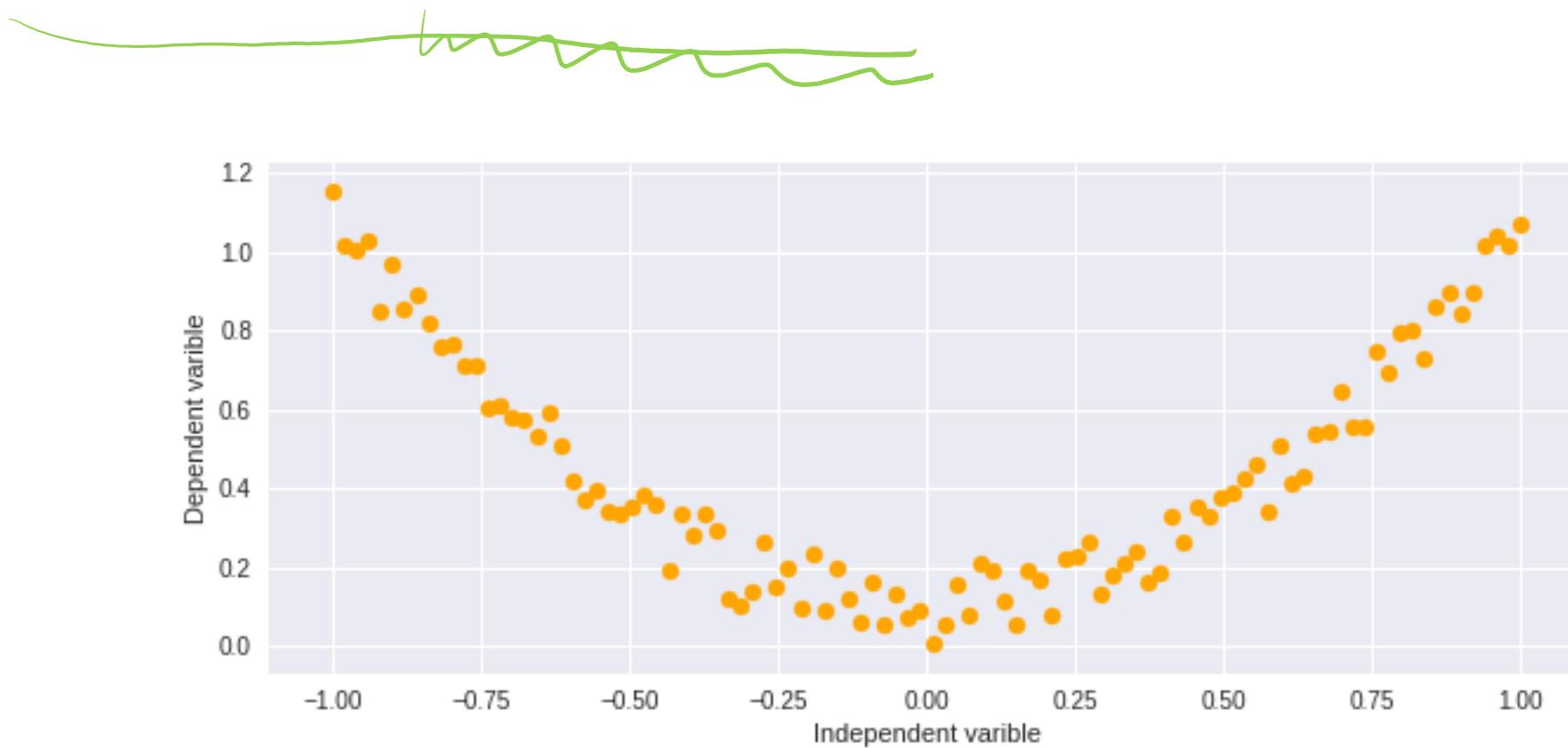
Gradients

- For this simple version of neural networks all the gradient calculations are the same as we've seen earlier...
- It probably wouldn't be a surprise to find out that to train this simple ANN is a convex problem (not true for all ANNs).

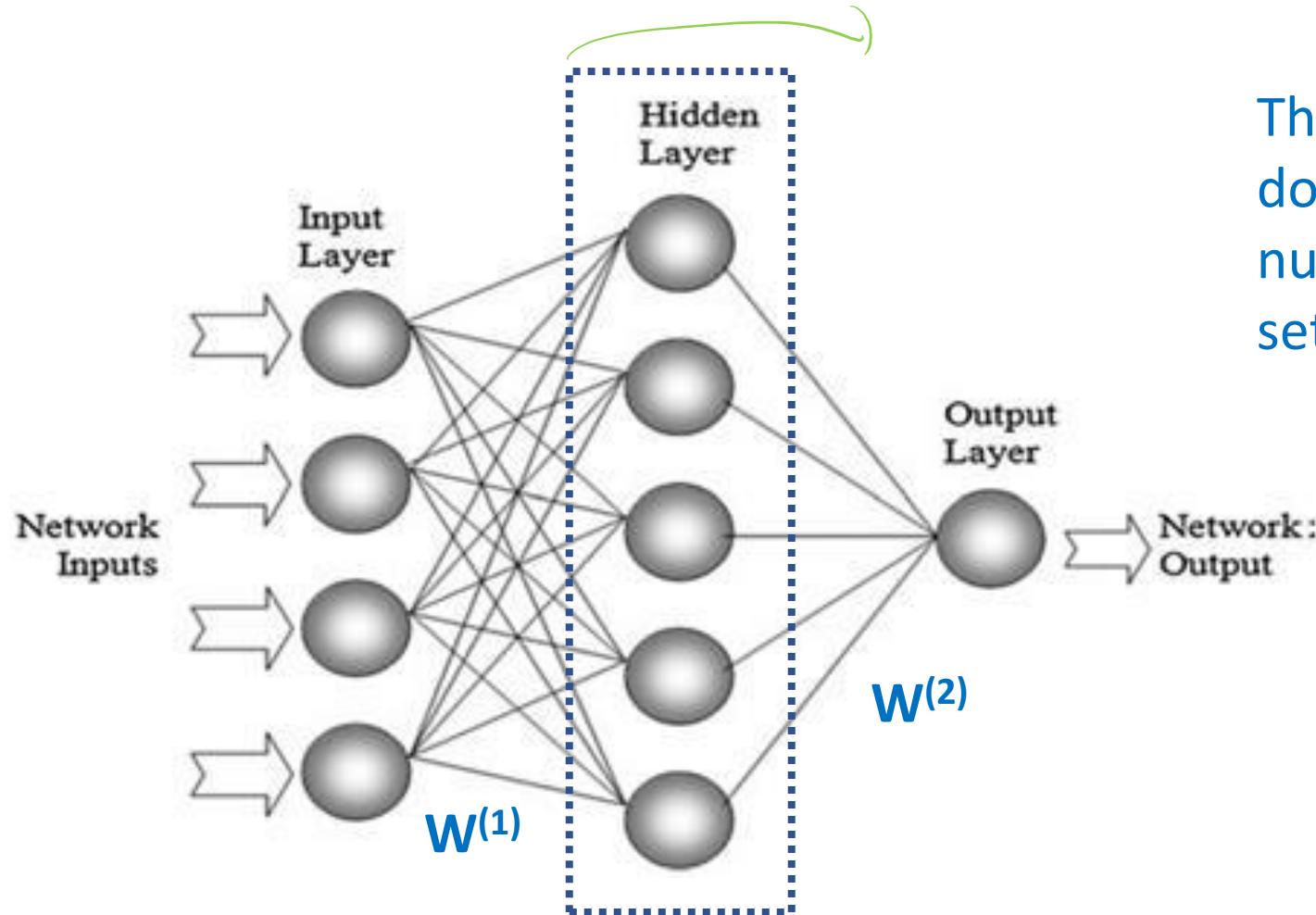


parameters (w) can
be multidimensional

Achieving Nonlinearity

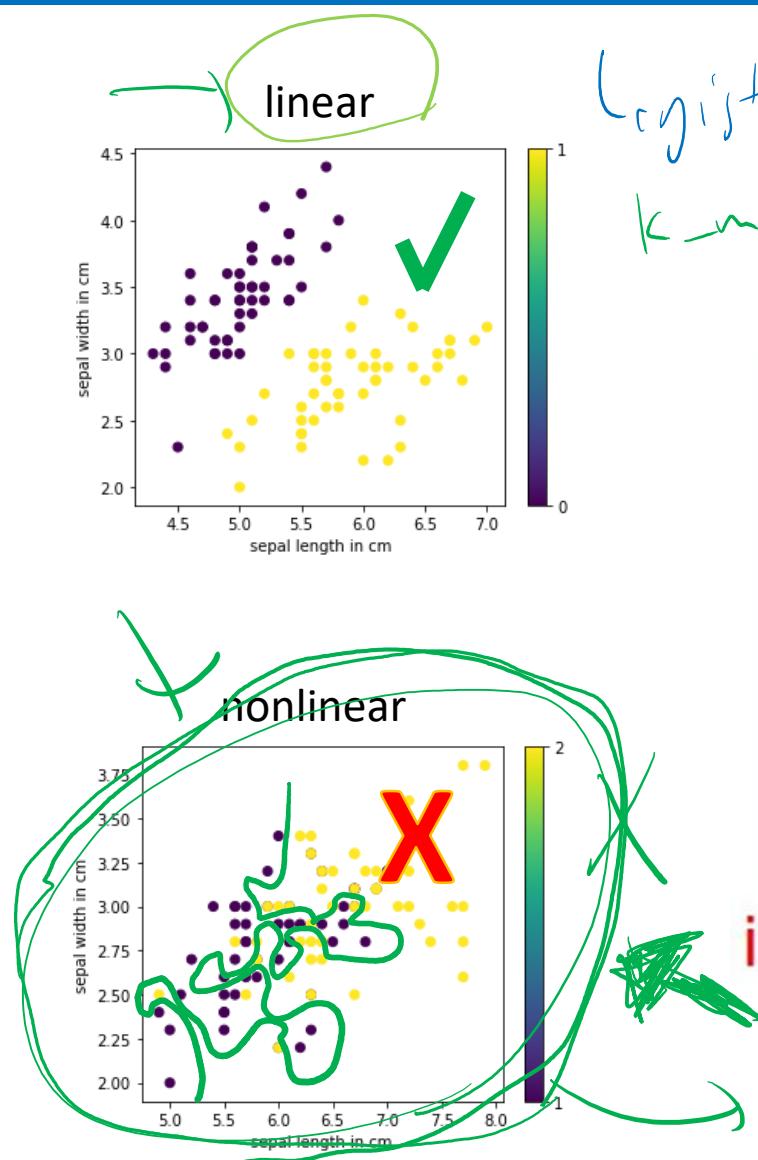


Add More Neurons?

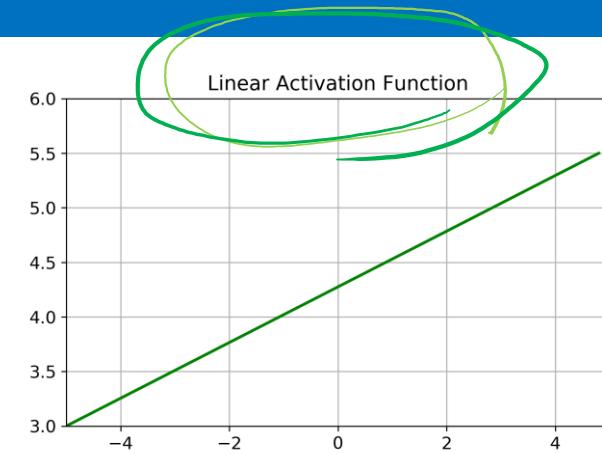
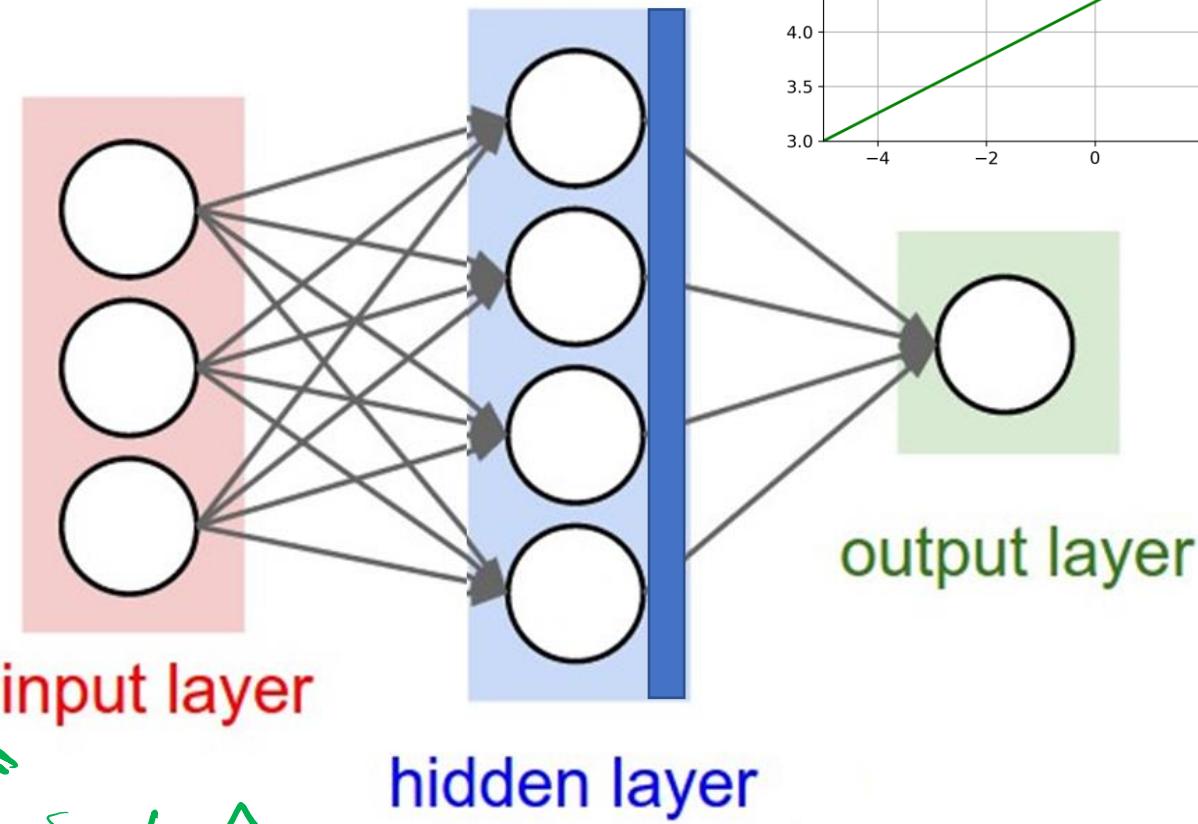


This is a 2-layer neural network. We do not count the input layer, so the number of layers equal number of sets of weights and biases.

2-layer ANN with Linear Activation

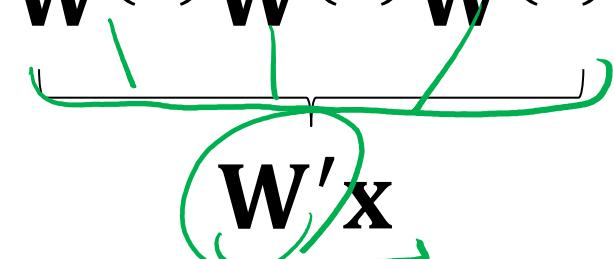


Logistic
k-means



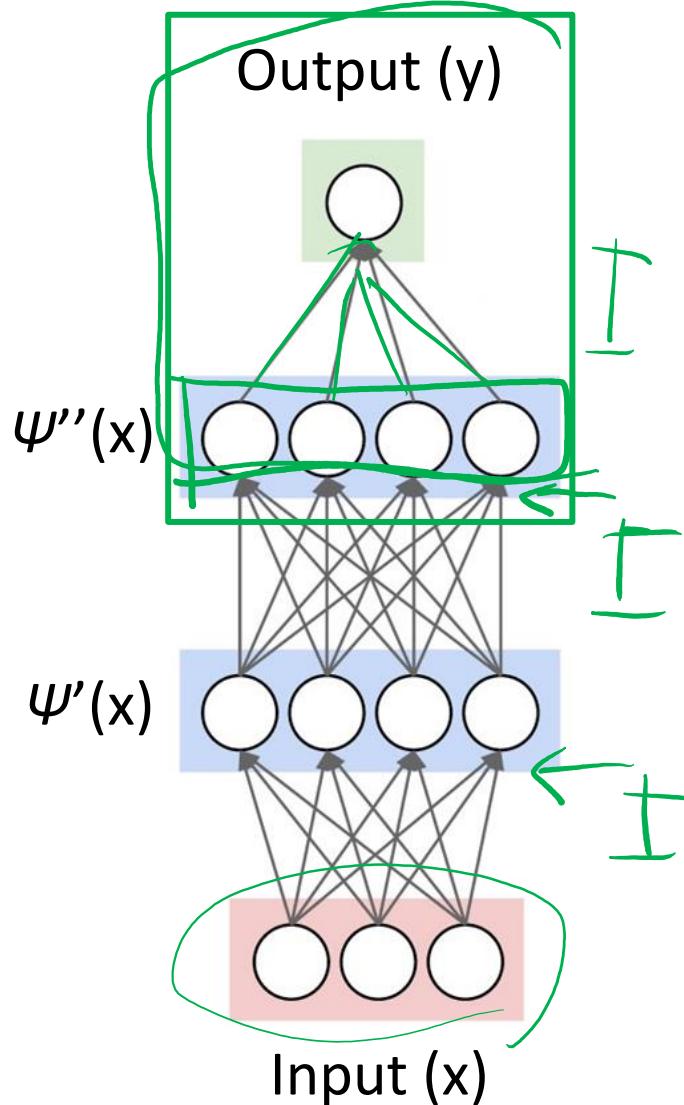
Expressive Power

- Adding more linear layers does not help increase the capacity of the model.
- Any sequence of linear layers can be equivalently represented with a single linear layer.

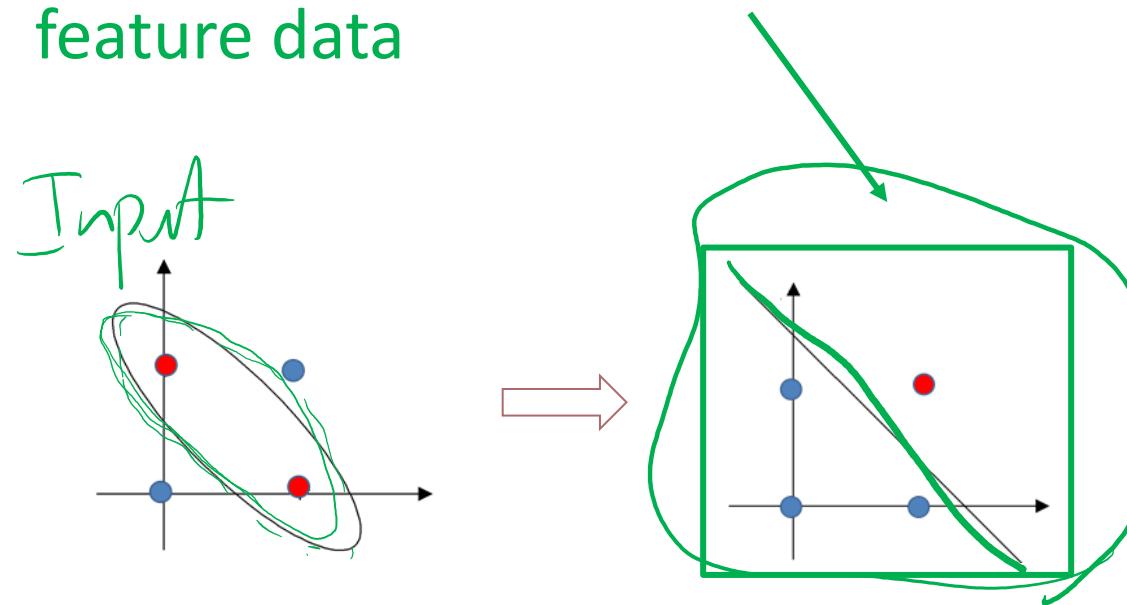
$$\hat{y} = \mathbf{W}^{(1)} \mathbf{W}^{(2)} \mathbf{W}^{(3)} \mathbf{x}$$

$$\mathbf{W}' \mathbf{x}$$

- Deep linear networks are no more expressive than **linear regression!**

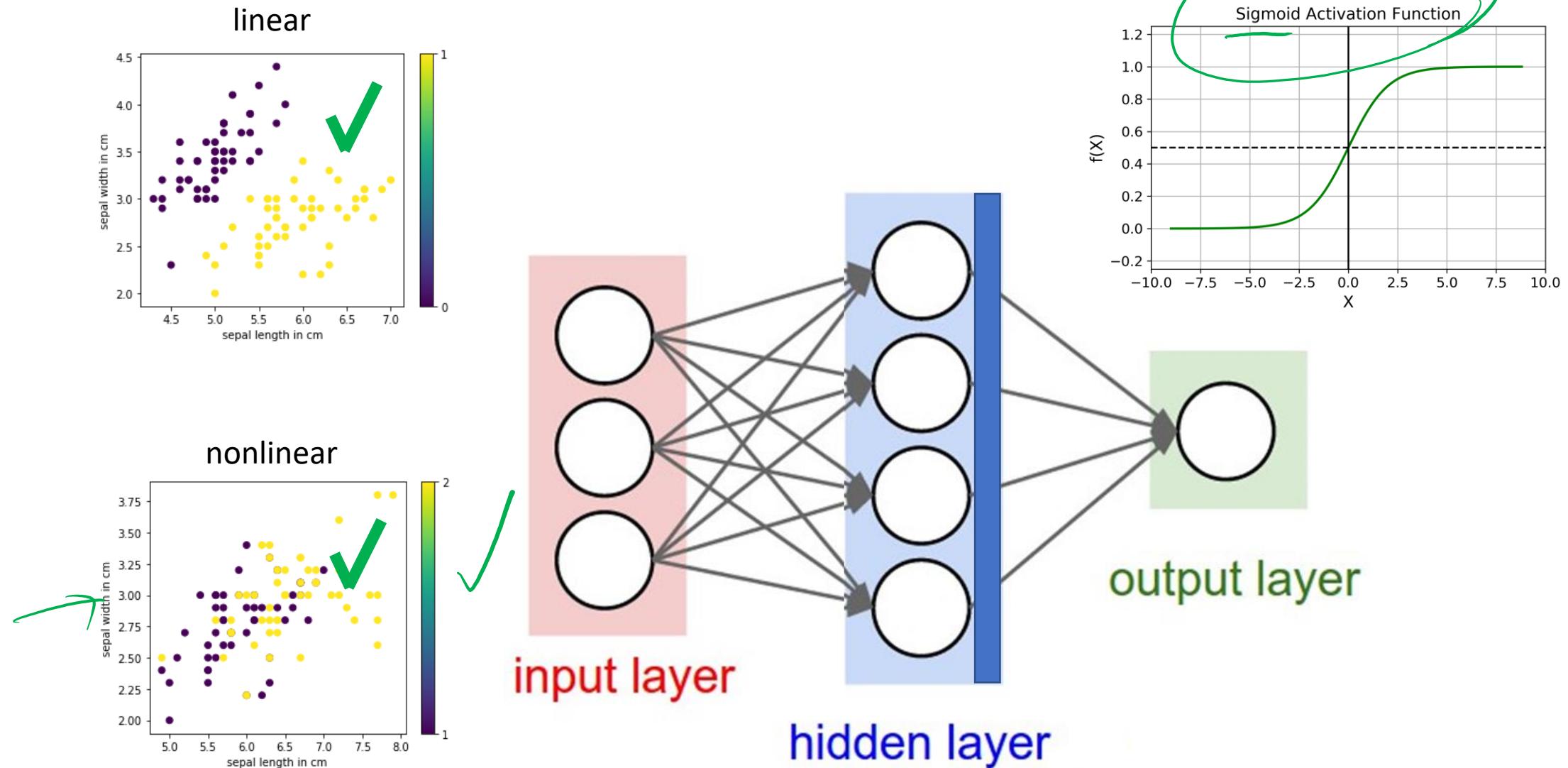
Need an Activation Function



- Neural Networks can be viewed as a **way of learning features**
- The goal being that the final layer is presented with **linearly separable** feature data



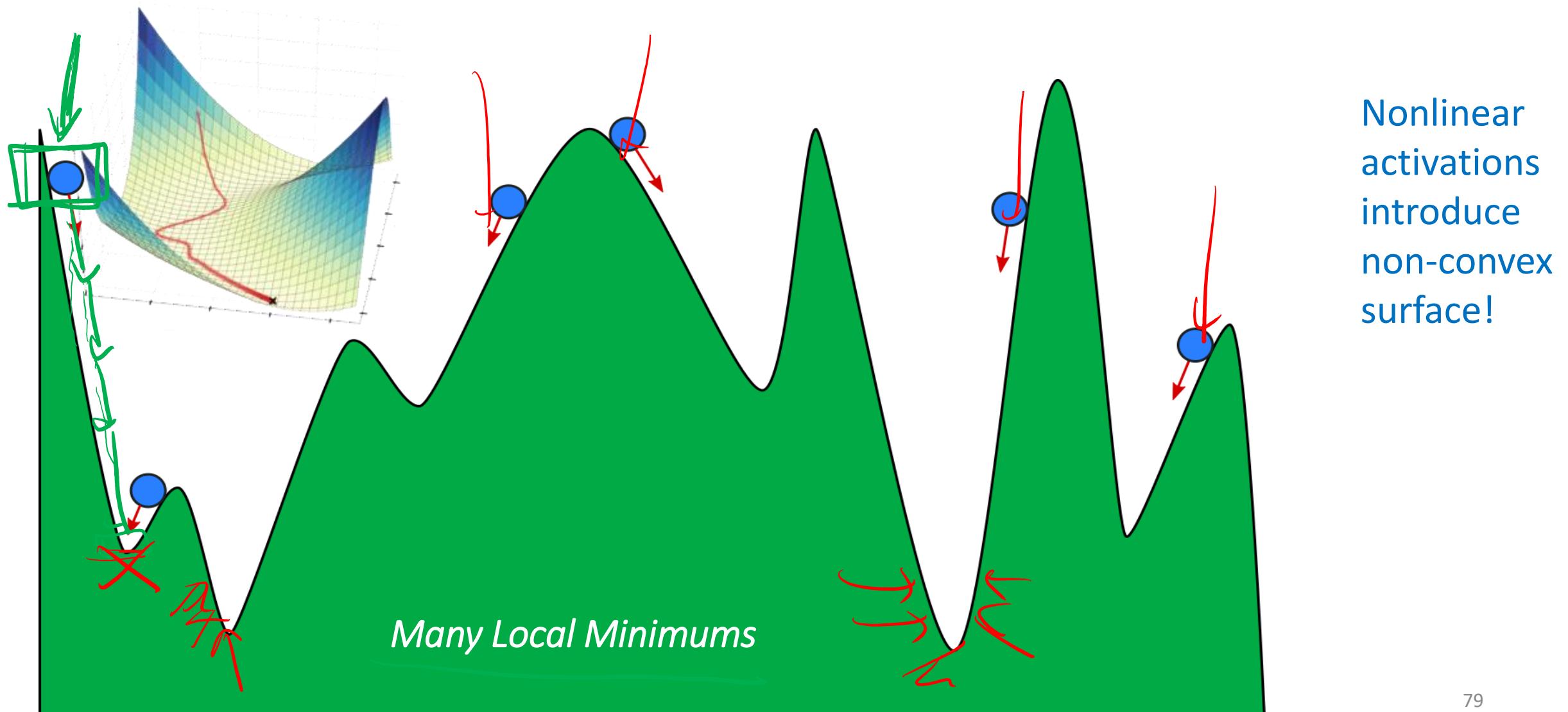
2-layer ANN with Nonlinear Activation



Expressive Power

- Multilayer feed-forward neural nets with nonlinear activation functions are **universal function approximators.**
- They can approximate any function arbitrarily well, but this comes at a cost...

Cost Function is Non-Convex!



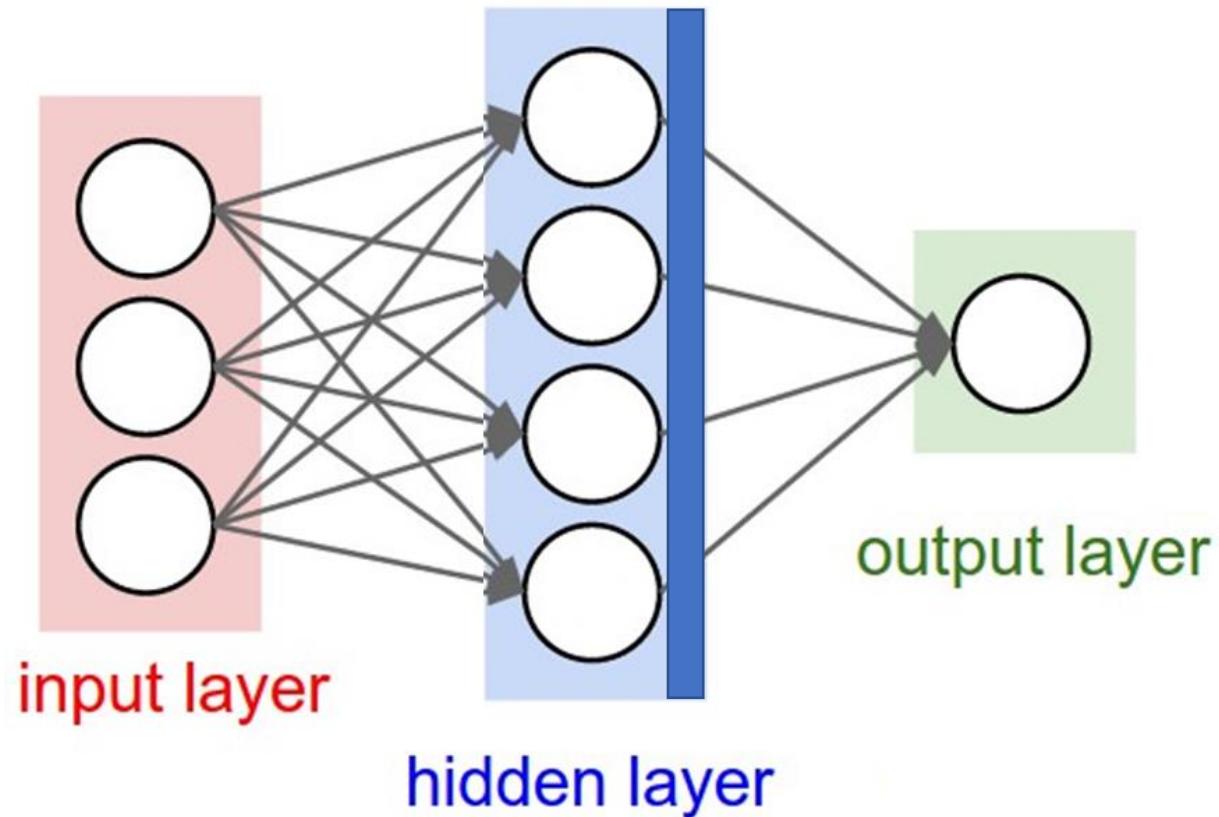
Tuning Neural Networks

Changing ANN Architecture:

- Number of hidden units
- Weights
- Activation Functions

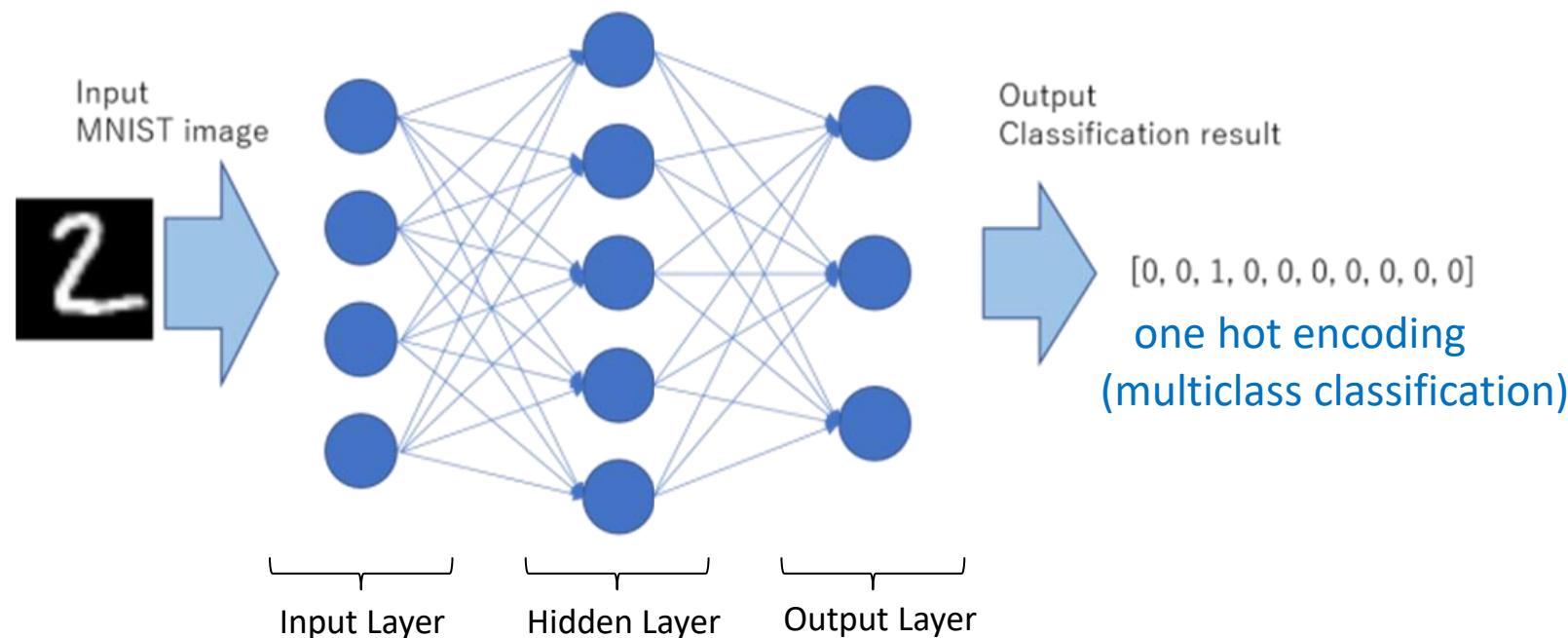
Applying Different Training Techniques:

- Number of iterations
- Learning rate and adaptive learning rate
- Momentum
- Batching of Data
- Regularization
- Dropout
- Feature Augmentation
- Many more...



Take-home Exercise: Discuss on Piazza

Q: Determine the gradients for a 2-layer artificial neural network with sigmoid activation on the hidden and output layers. The error is computed using mean squared error loss.



Next Time

- Week 10 Q&A session: Thursday and Friday
 - Project 4 is due on April 1st
- Week 11 Lecture – Deep Learning (and More)
 - Neural Network Architectures
 - Automatic Differentiation
 - Discrete Optimization

12:10

12:15