



Artificial Intelligence and Machine Learning

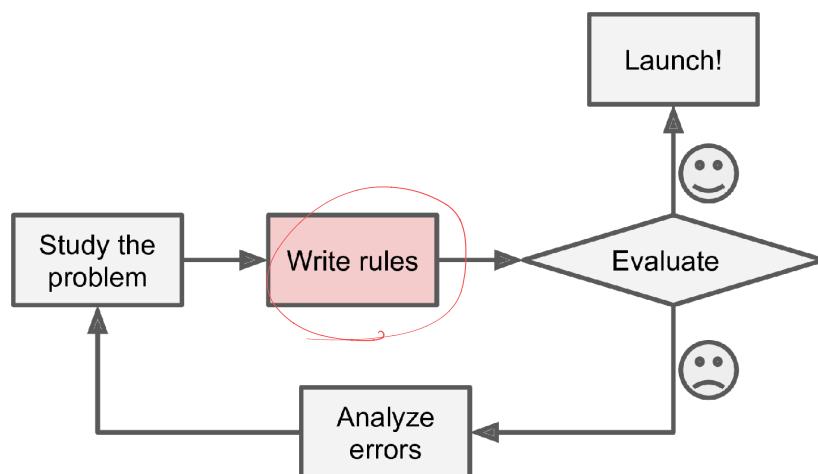
Linear Regression

Lecture 1: Outline

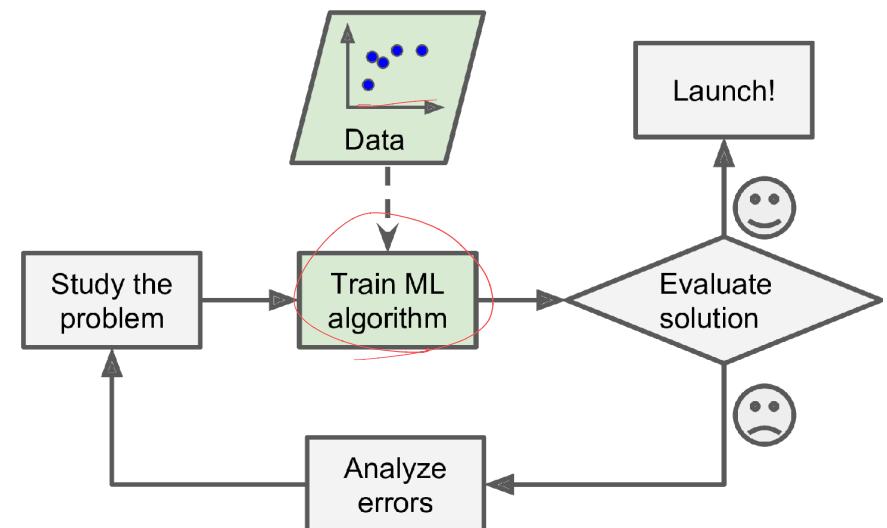
- Introduction to ML
- Linear Regression
- Optimization
- Linear Regression: Probabilistic Interpretation
- Bias-Variance Tradeoff
- Regularization

Introduction to ML

- **Machine Learning** is the science (and art) of programming computers so they can learn from data.

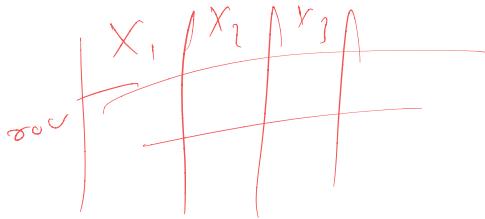


The traditional approach



The Machine Learning approach

Data Types

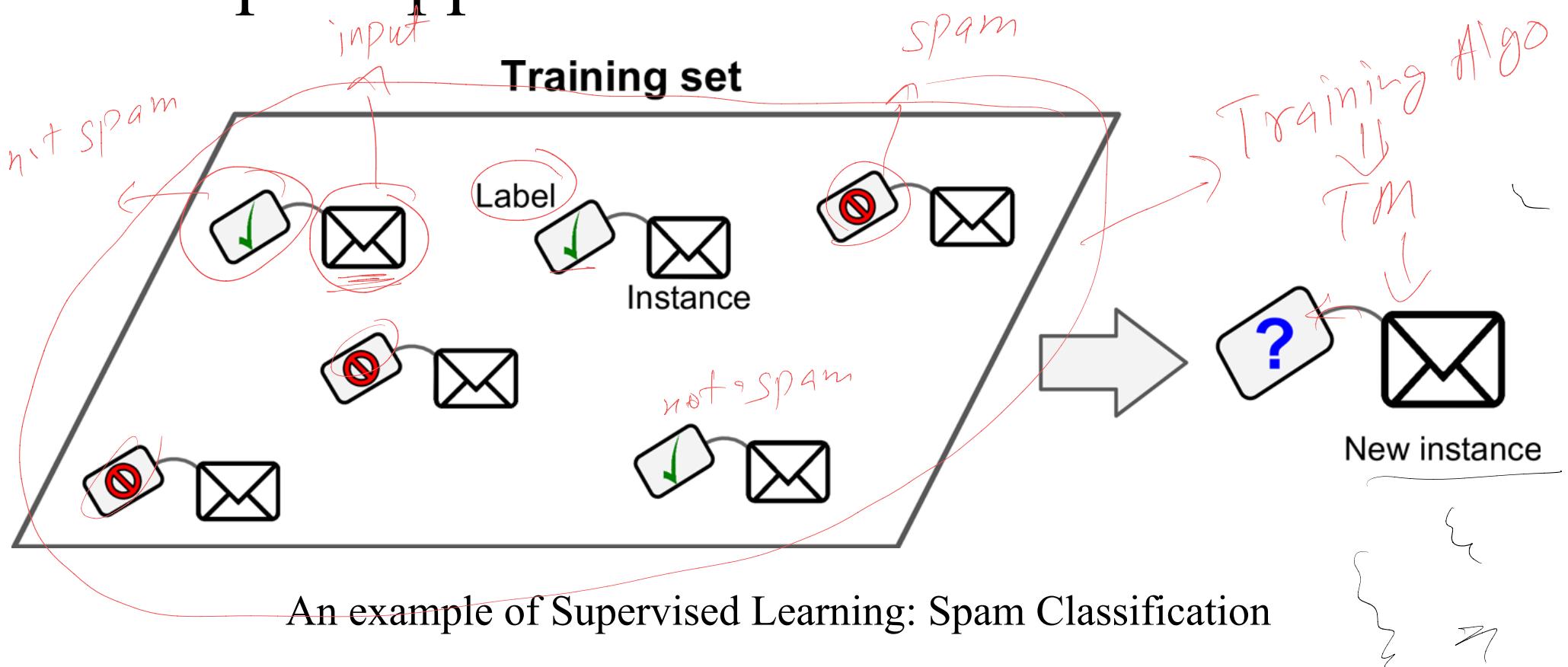


- **Tabular Data** (e.g., spreadsheets, databases)
 - Note: Columns are called **Features**. Rows are called **Samples**.
- **Time-Series Data** (e.g., stock prices, weather forecasts, IoT sensor data)
- **Text Data** (Natural Language Processing, e.g., emails, social media posts, documents)
- **Images and Videos** (Computer Vision, e.g., medical imaging, surveillance, facial recognition)
- **Audio Data** (Speech Recognition, Music Processing, e.g., voice commands, podcasts, sound classification)

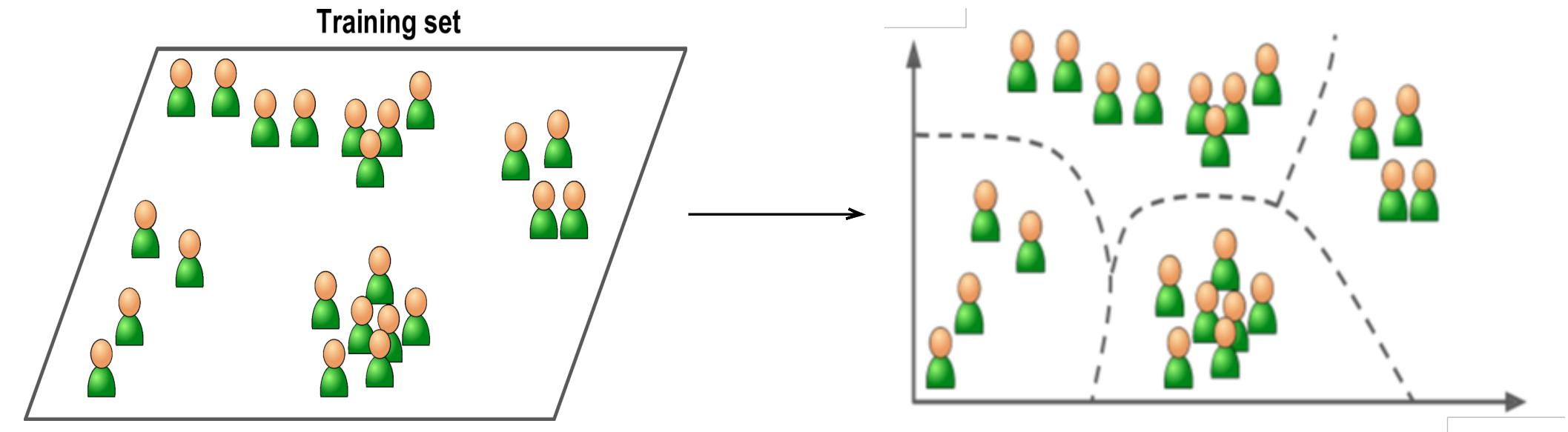
ML Algorithms Types

- **Supervised:** There is a target we want to predict.
 - **Regression:** Predict continuous value (e.g. house prices).
 - **Classification:** Predict discrete value (e.g. spam/not-spam).
- **Unsupervised:** There is no target. We are interested in things like:
 - **Clustering:** Grouping
 - **Dimensionality Reduction:** Reducing the Dimensions
 - **Anomaly Detection:** Detecting outliers

Example Application



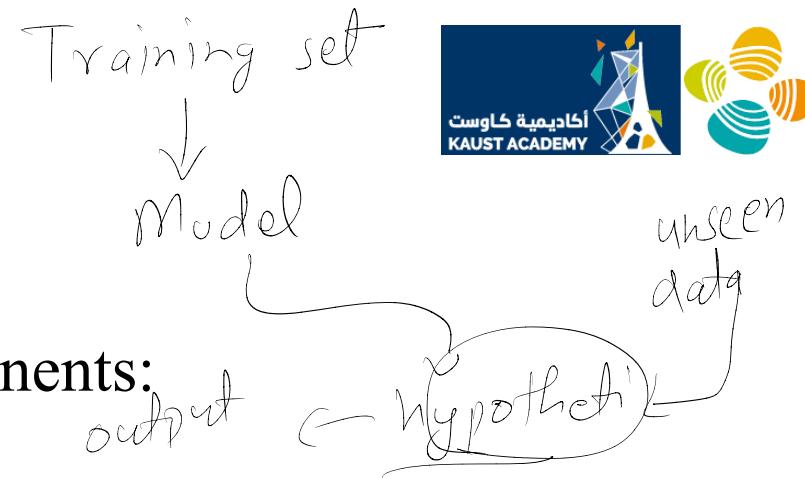
Example Application



An example of Unsupervised Learning: Clustering

How Does ML Work?

- Any ML system consists of three main components:



Hypothesis (Model): The function that approximates the target.

- E.g. Linear Regression, Logistic Regression, SVM, Decision Trees, NN,...

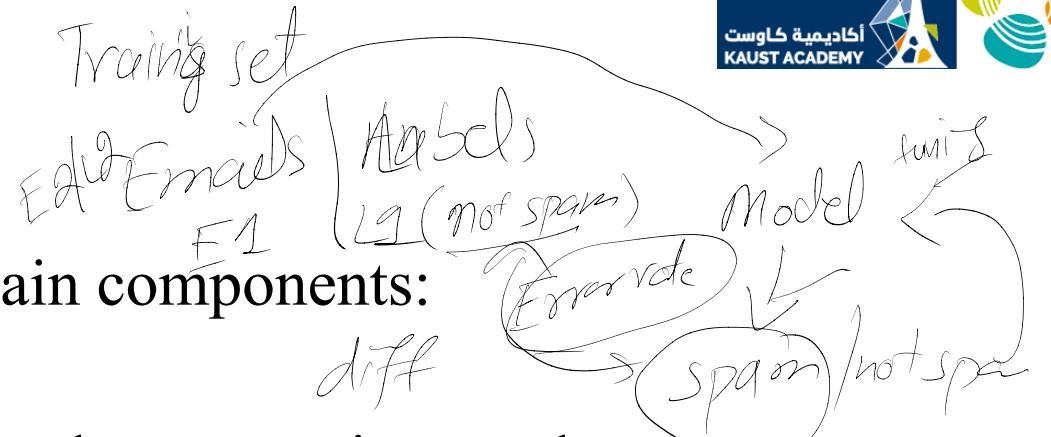
Optimizer: The mechanism for improving predictions of our model.

Loss Function: The measure of how wrong the predictions are.

Objective
function

How Does ML Work?

- Any ML system consists of three main components:



Hypothesis (Model): The function that approximates the target.

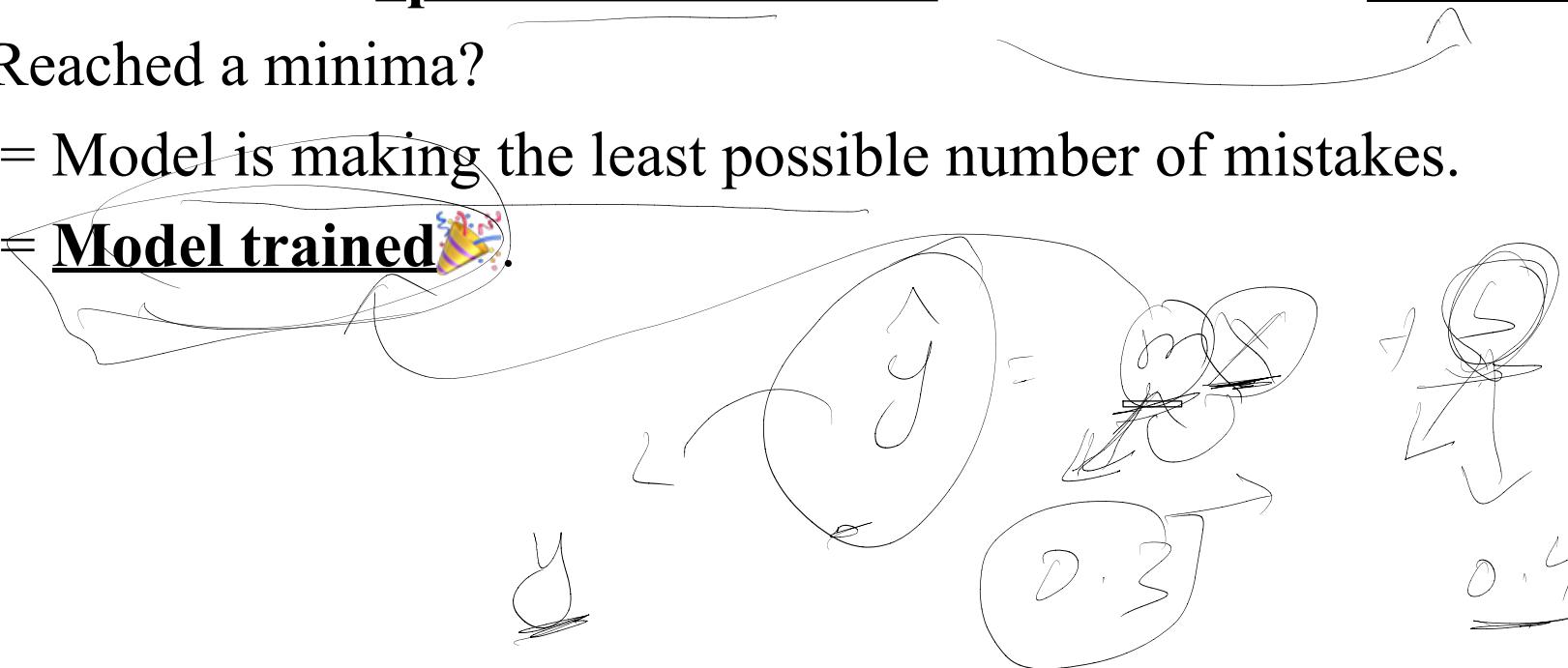
- E.g. Linear Regression, Logistic Regression, SVM, Decision Trees, NN,...

Optimizer: The mechanism for improving predictions of our model.

Loss Function: The measure of how wrong the predictions are.

How Does ML Work?

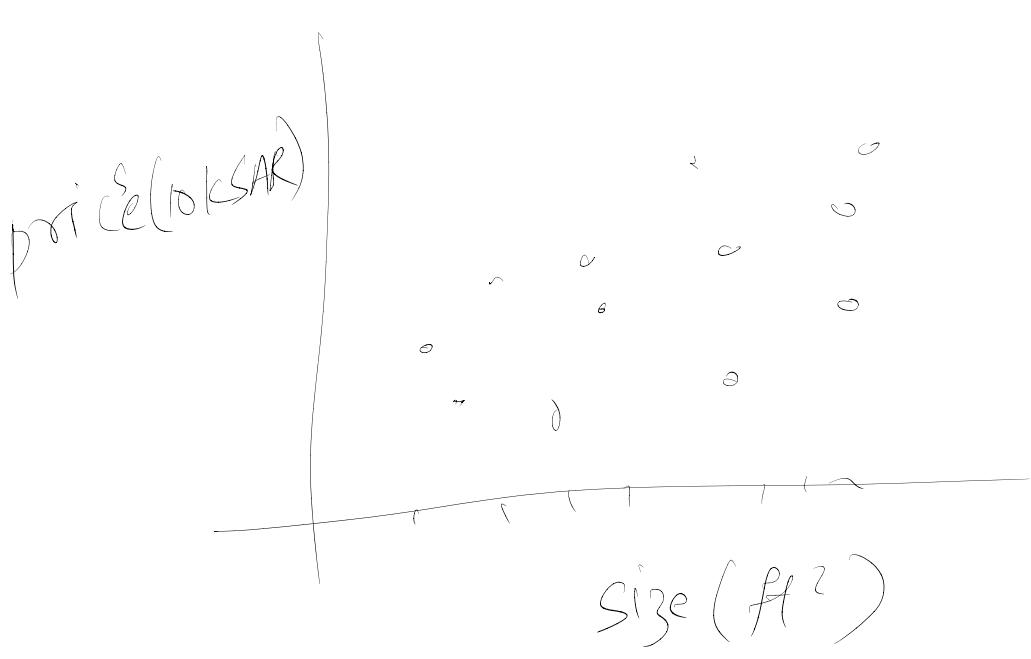
- We firstly define our task (classification/regression) then choose an appropriate model.
- We will use an optimization method to minimize the loss function.
- Reached a minima?
 - = Model is making the least possible number of mistakes.
 - = Model trained.



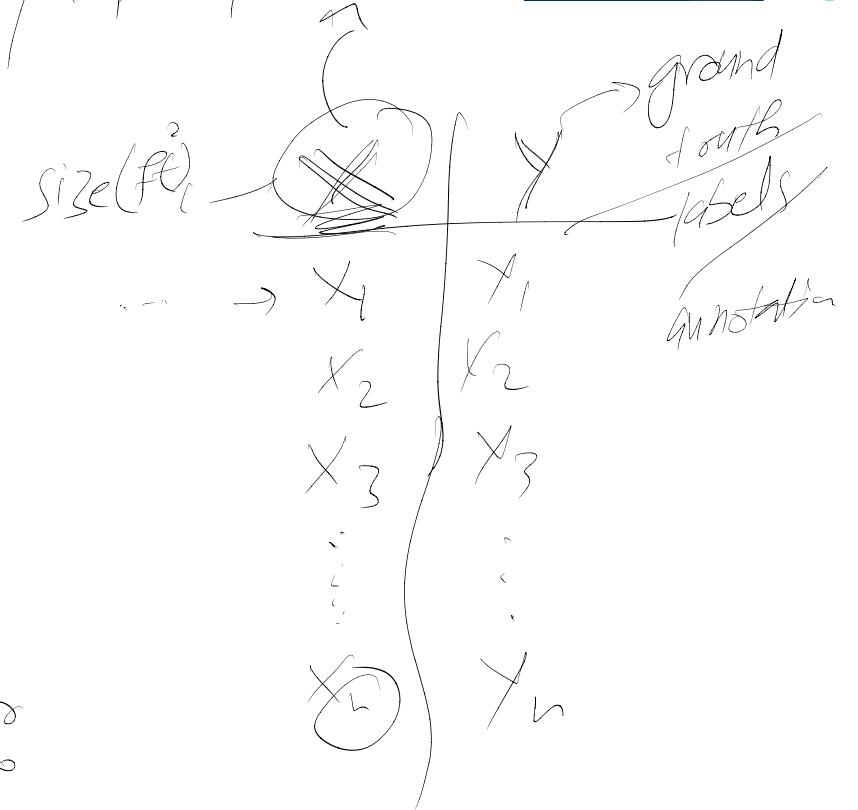
Linear Regression: Motivation

- Linear Regression is “still” one of the more widely used ML/DL Algorithms
- Easy to understand and implement
- Efficient to Solve
- We will use Linear Regression to Understand the concepts of:
 - Data
 - Models
 - Loss
 - Optimization

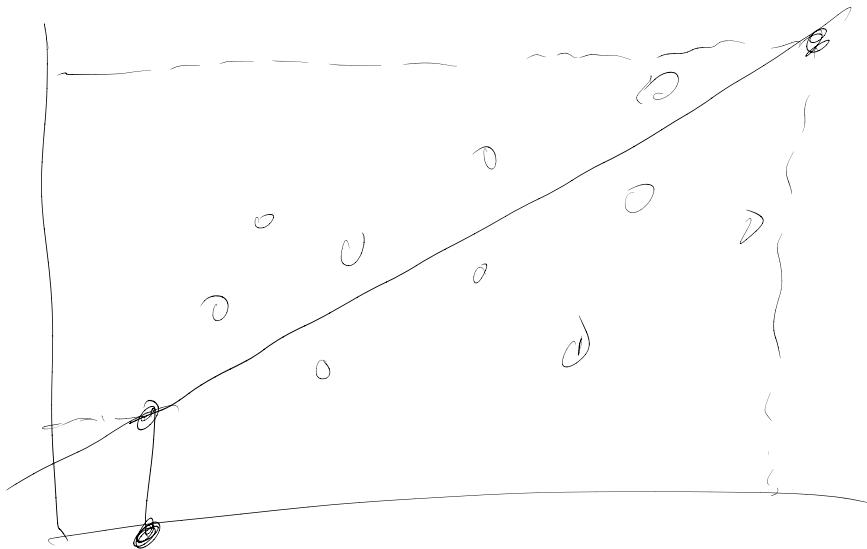
Linear Regression: Data

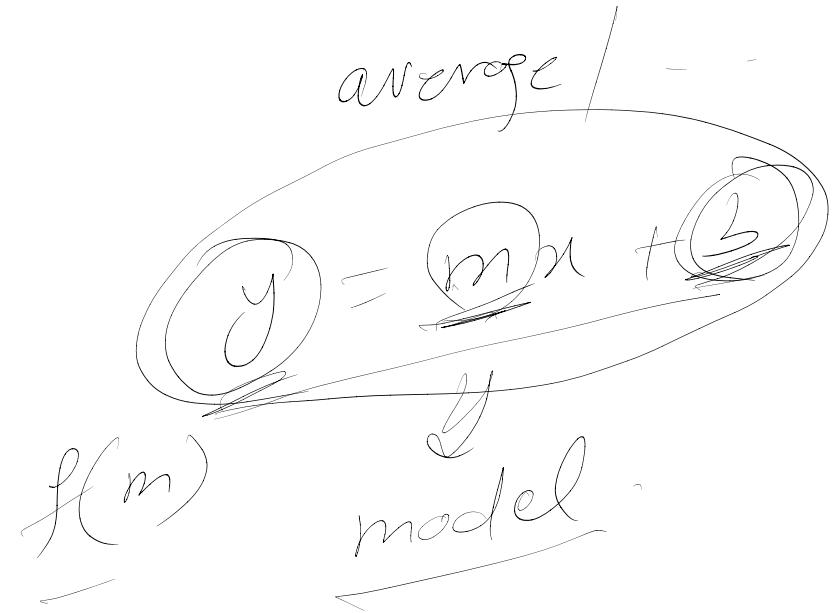


attribute / features / input / data



Linear Regression: Model





A hand-drawn diagram illustrating the linear regression equation $y = mx + b$. The equation is written in the center. To the right, a line is labeled "model". Above the line, the word "average" is written above a dashed horizontal line. The y-axis is labeled "y".

$$y = mx + b$$

average / -

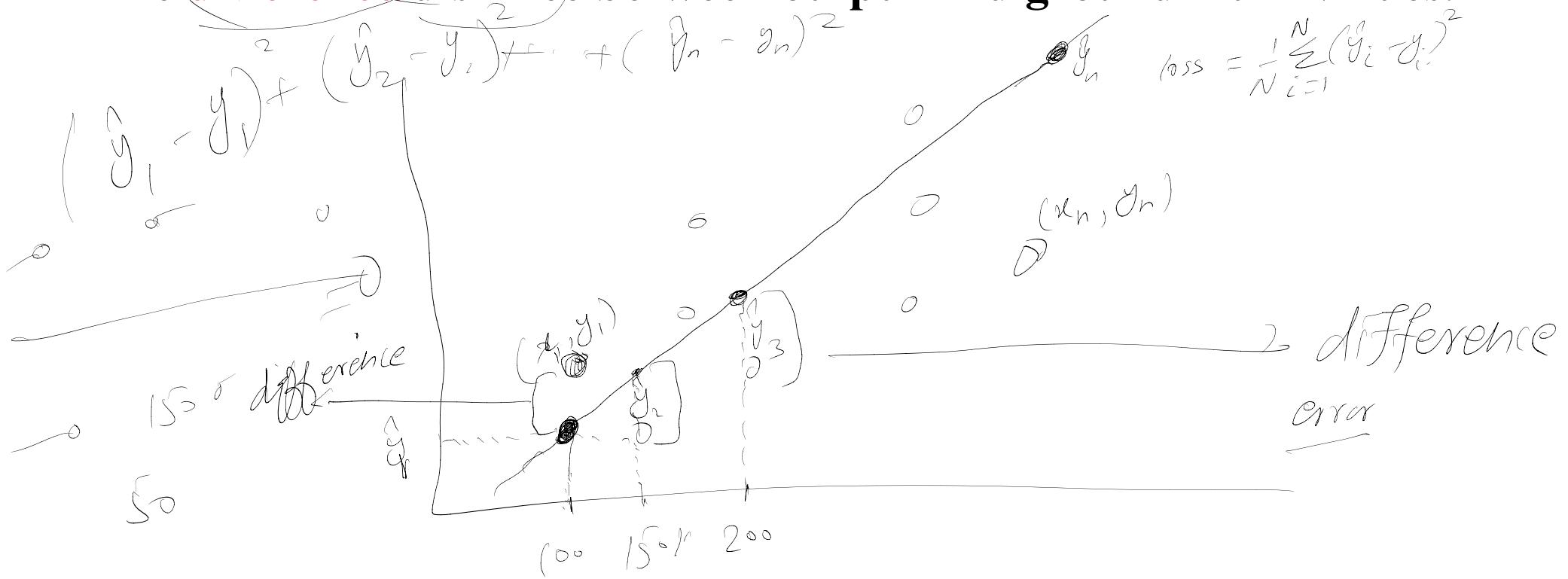
y

$f(m)$

model

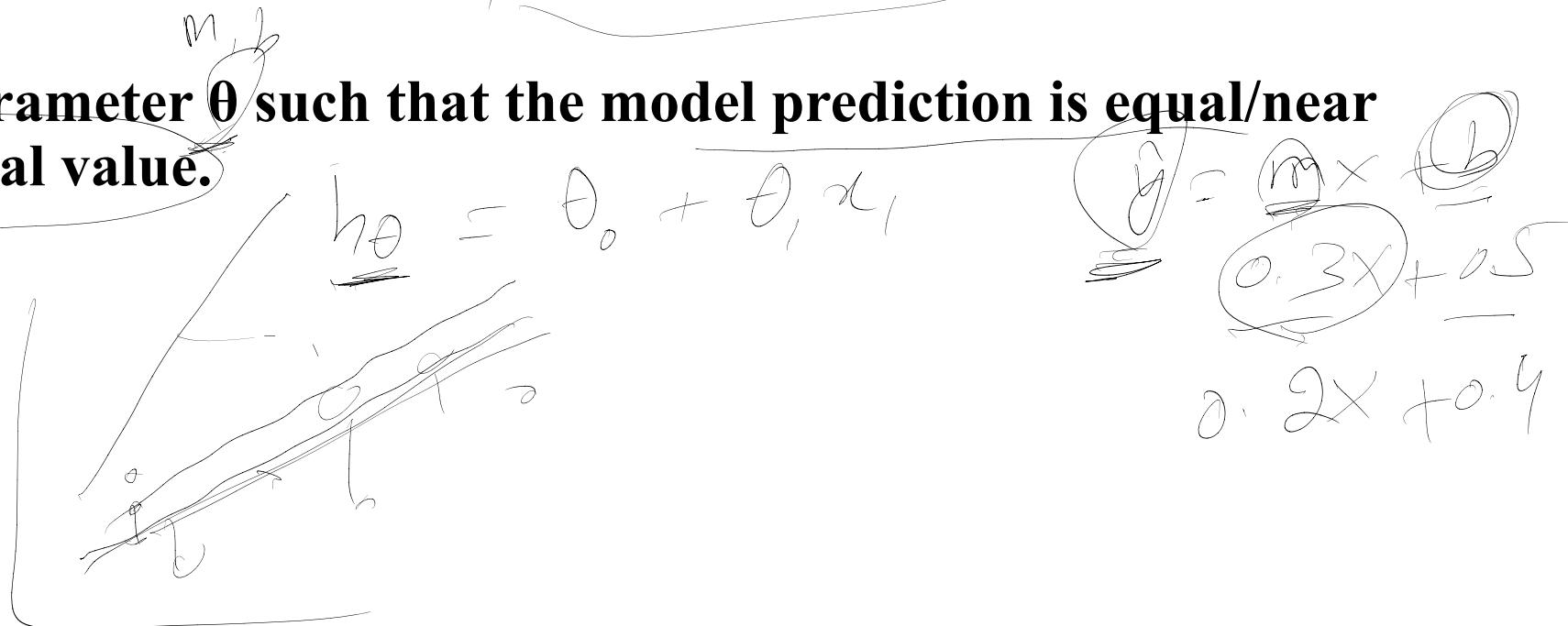
Linear Regression: Loss / Objective

- The difference/distance between output and ground truth values.

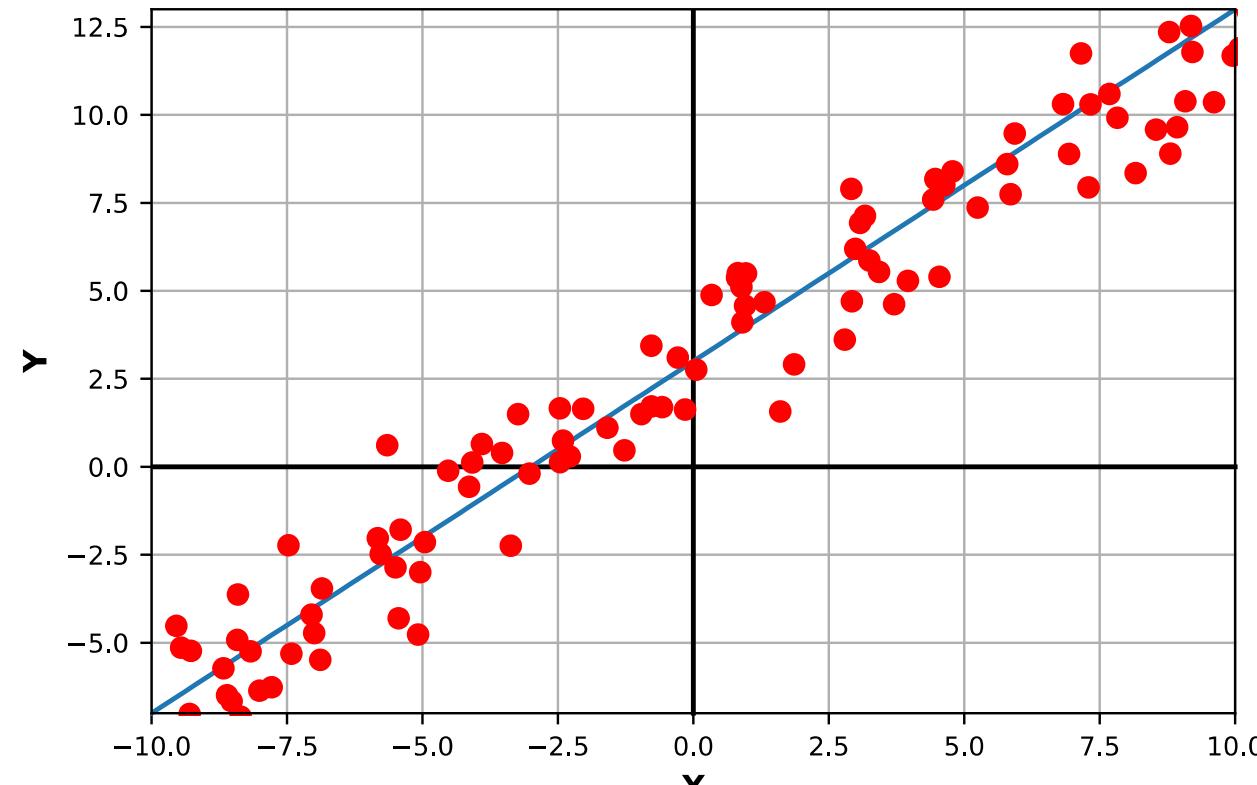


Linear Regression: Optimization

- Choose parameter θ such that the model prediction is equal/near to the actual value.



Simple Linear Regression



Model (*Linear*)

$$Y = mX + b$$

Y: Response Variable
X: Covariate / Ind.,
var/Regressors
m: slope
b: bias
 $\theta = \{m, b\}$

Simple Linear Regression

- **Hypothesis:**

$$\hat{y}_i = mx_i + b$$

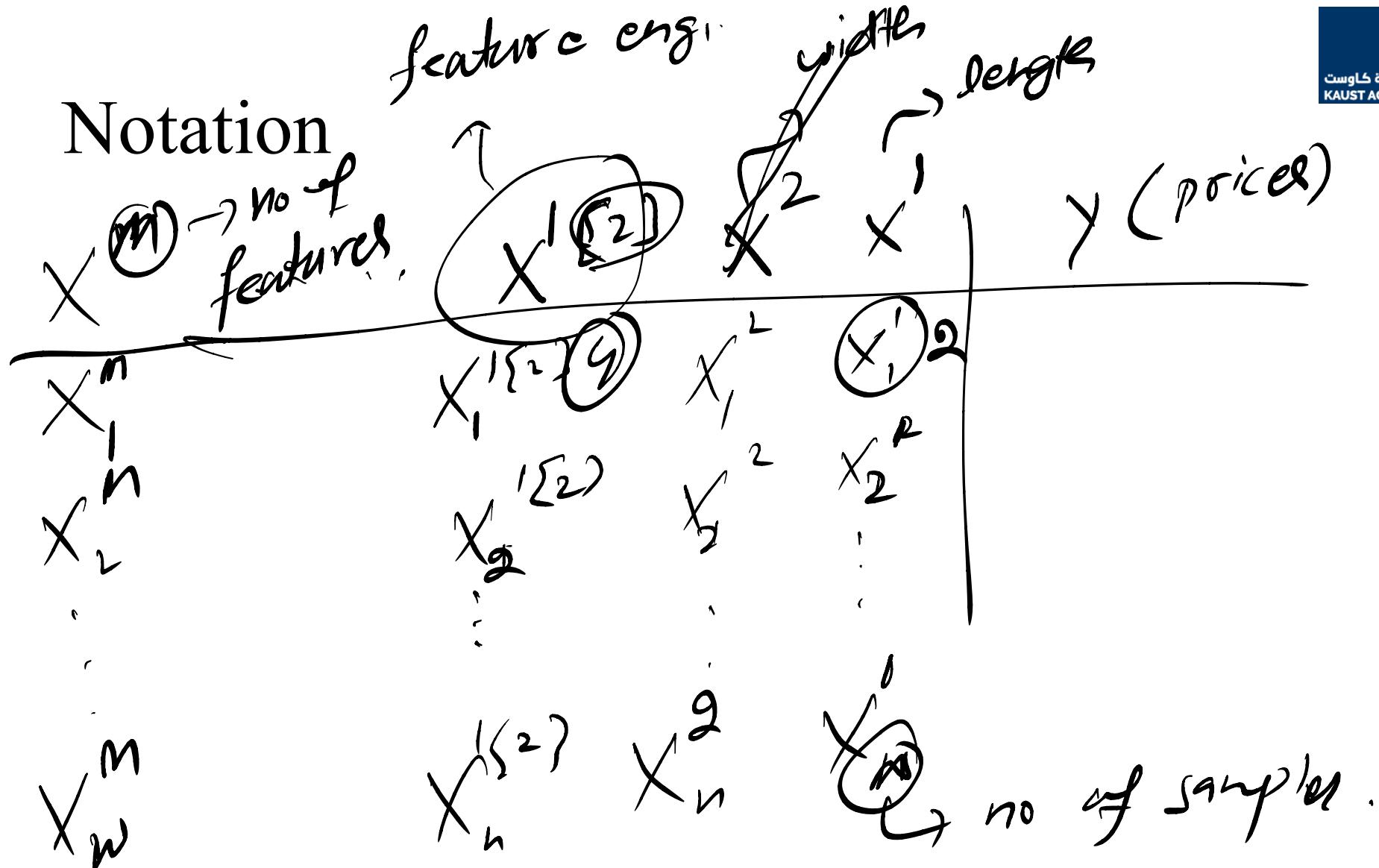
- **Input:** data $(x_i, y_i), i \in \{1, 2, \dots, N\}$
 - (e.g., house size x and price y)
- **Goal:** learn values of variable (m, b)

Notation

i → no of samples
 j → no of feature

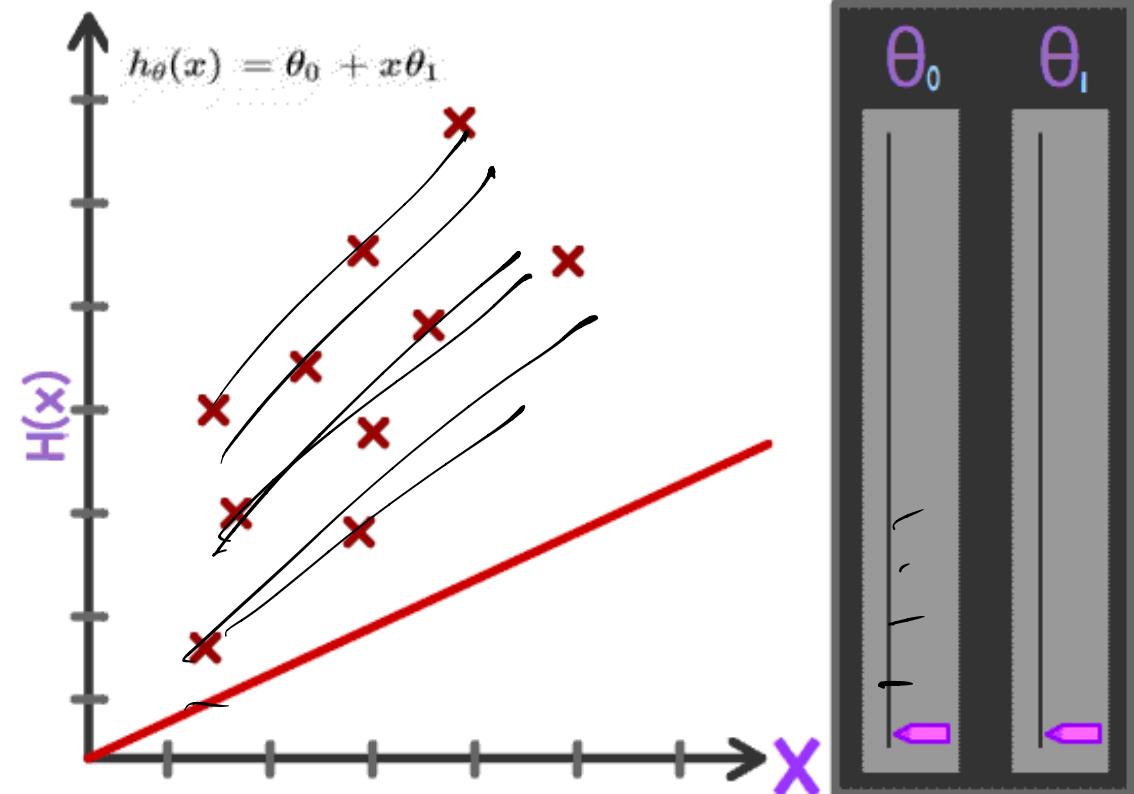


Notation



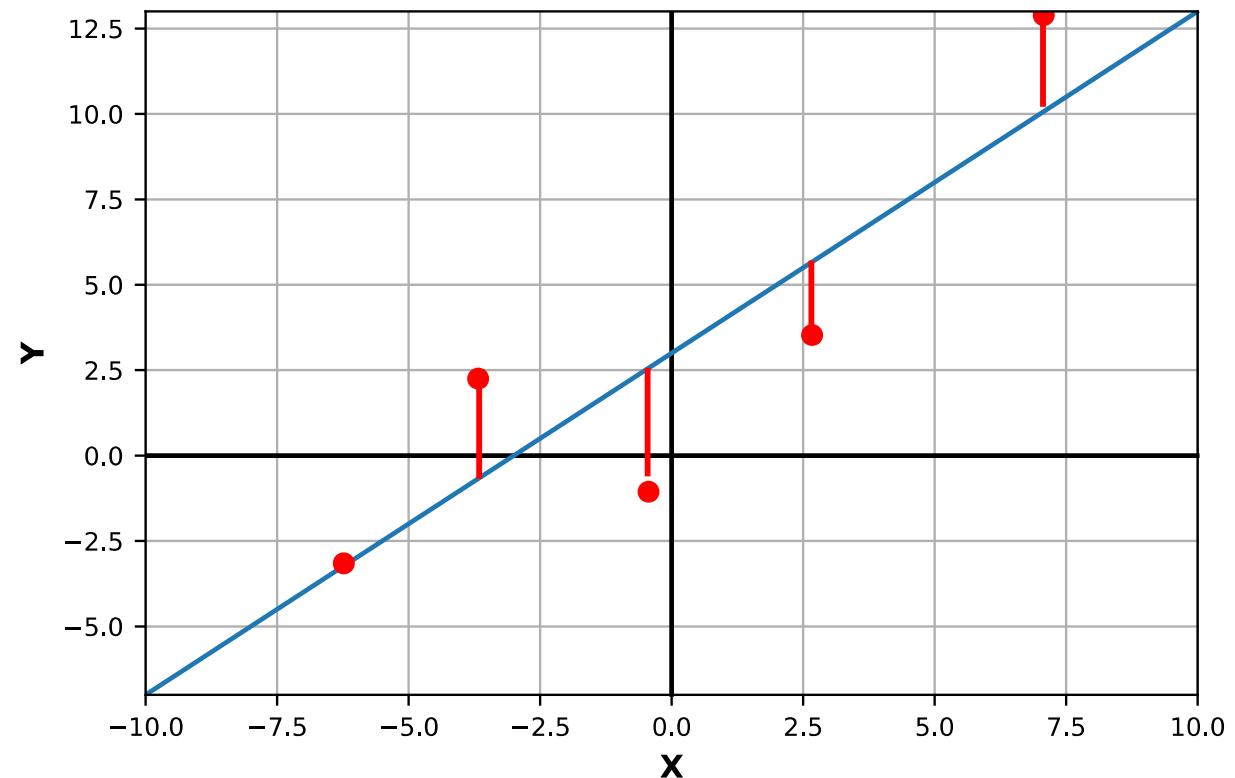
Solution Strategy for Solving the Problem

- There are countless possible lines.
- We want a line which is in some sense the “average line” that represents the data.
- Any ideas as to how we can do it?



Optimization

- To find the "best line," we should minimize the **distances** between our line's predictions and all the data points.
- How to define that mathematically?



Loss Function

- For one sample, this can be represented mathematically by:

$$(y - \hat{y}) \quad (\text{Error})$$

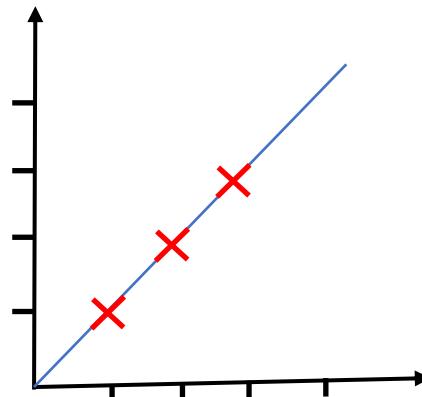
- But this could result in negative value if $\hat{y} > y$. Let's square it to remove the negative sign:

$$(y - \hat{y})^2 \quad (\text{Squared Error})$$

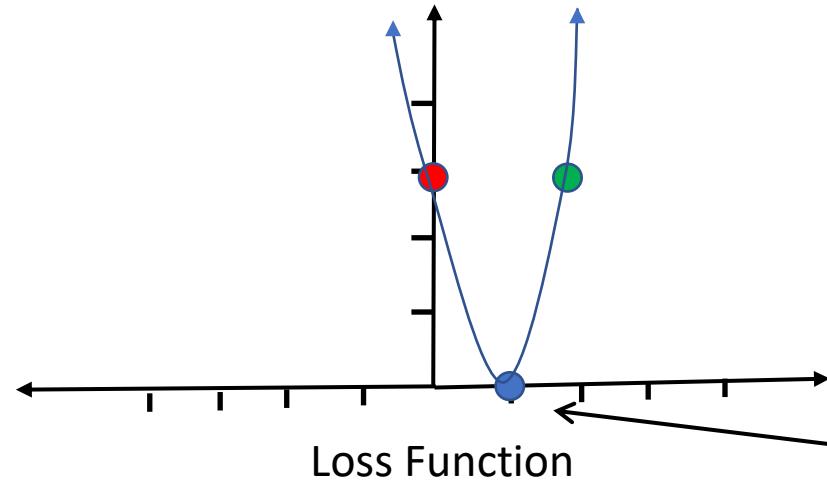
- But we have N samples, not only one. So, let's sum the errors and take the average:

$$\boxed{\text{Loss (MSE)} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2} \quad (\text{Mean Squared Error})$$

Intuition of Loss Function



Hypothesis



$$h(x) = mx$$

$$J(m) = \sum_{i=1}^3 (y_i - mx_i)^2$$

Notice: Lower is better.

$J(m = 0) = 14$

$J(m = 1) = 0$

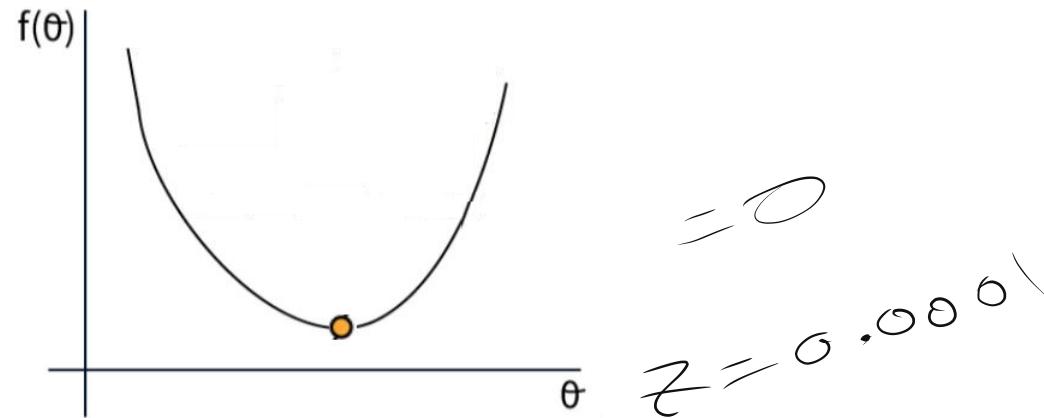
$J(m = 2) = 14$

How to find minima of a function (Review):

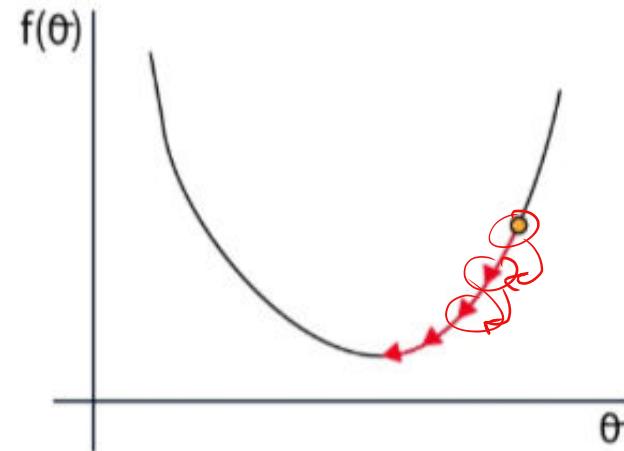
- There are two approaches to find the minima:
 - **Exact (Closed-form)**: Directly calculates the solution mathematically by solving for $f'(x) = 0$. *usefull in Lr, useless in the rest*
Important Note: can be used only with a very limited number of algorithms.
 - **Approximation (Iterative approach)**: Gradually improves the solution step by step.
Done by optimizers (e.g. Gradient Descent, ADAM,...etc).

How to find minima of a function (Review):

Closed-form:



Iterative:



- Example: $y = x^2$ (Solution: $x = 0$)
 - Closed-form Final Result: $x = 0$
 - Iterative Final Result: $x = 0.00001$ (close enough)

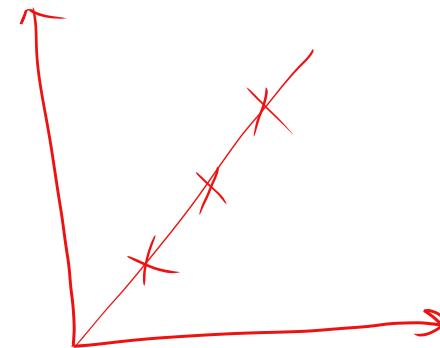
How to find minima of a function (Review):

- Let's try to solve this using the closed-form here: $\text{assume } \hat{y} = mx$

$$J(m) = \sum_{i=1}^3 (y_i - mx_i)^2 \quad J(m) = \sum_{i=1}^3 (i - mi)^2$$

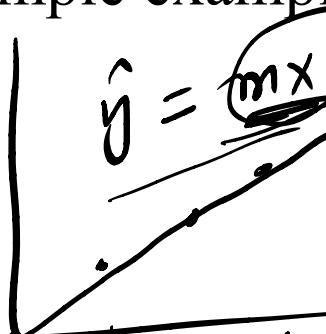
$$\frac{dJ(m)}{dm} = \frac{d}{dm} \sum_{i=1}^3 (i - mi)^2 \quad \frac{dJ(m)}{dm} = \sum_{i=1}^3 \frac{d}{dm} (i - mi)^2$$

$$\frac{dJ(m)}{dm} = \sum_{i=1}^3 -2i(i - mi) \quad \left\{ -2 \sum_{i=1}^3 i^2 + 2m \sum_{i=1}^3 i^2 = 0 \quad m = 1 \right.$$

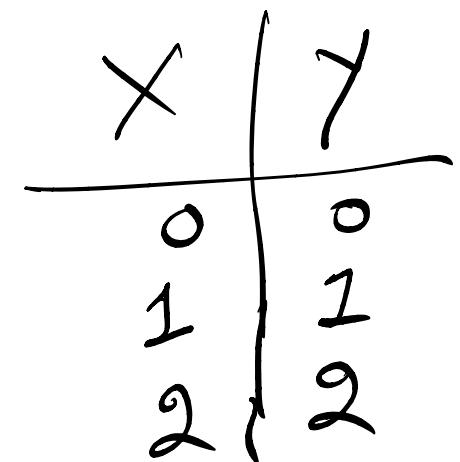


Example Data and its Solution

- Let's take a simple example (concrete data) and solve it for m .



$$J(m) = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$



Example Data and its Solution

- Let's take a simple example (concrete data) and solve it for m .

$$\begin{aligned}
 J(m) &= \frac{1}{N} \sum_{i=1}^N (m \cancel{x_i} - y)^2 \\
 &= \frac{1}{3} \left[(m \cdot 0 - 0)^2 + (m \cdot 1 - 1)^2 + (m \cdot 2 - 2)^2 \right] \\
 &= \frac{1}{3} \left[(m - 1)^2 + (2m - 2)^2 \right].
 \end{aligned}$$

Example Data and its Solution

- Let's take a simple example (concrete data) and solve it for m .

$$\mathcal{J}(0) = \frac{1}{3} \left[(\underline{x} - \bar{x})^2 + (\underline{y} - \bar{y})^2 + (\underline{z} - \bar{z})^2 \right]$$

$$\mathcal{J}(0) = \frac{1}{3} \left[1 + 4 \right] = \frac{5}{3}$$

Example Data and its Solution

- Let's take a simple example (concrete data) and solve it for m .

$$m_2 = \frac{1}{3} [0 + 0]$$

$$\boxed{J(1) = 2}$$

$$m_j \leftarrow m_j - \frac{\alpha}{\Delta m}$$

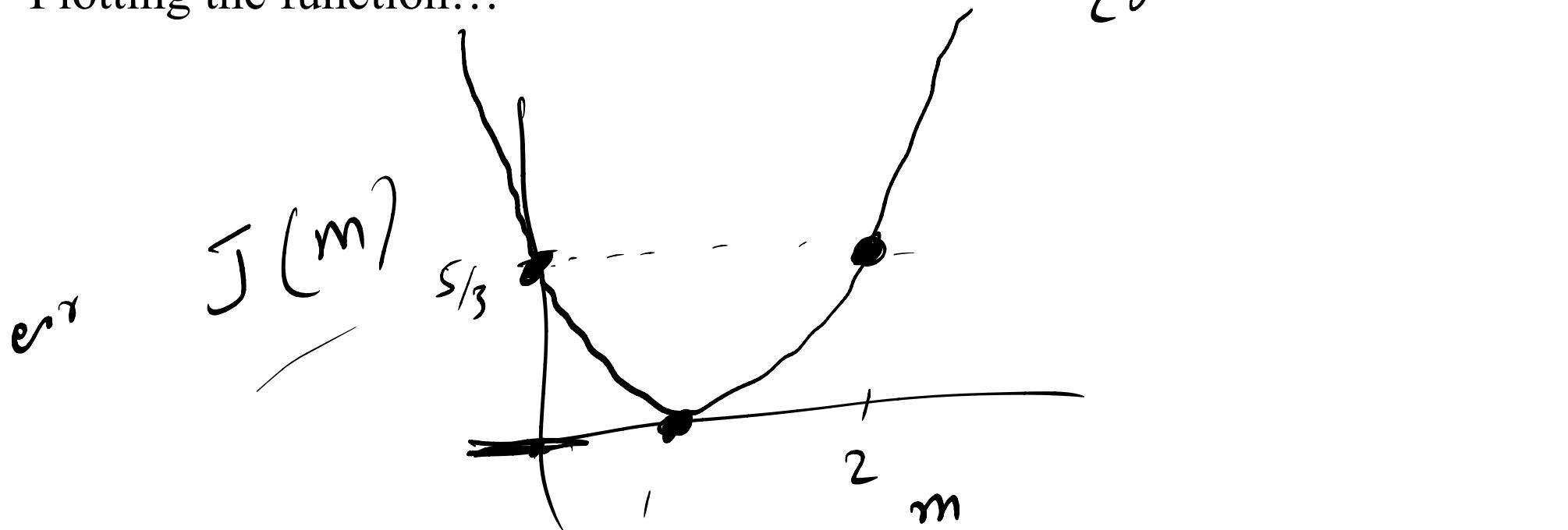
$$m \leftarrow j$$

$$m = \frac{2}{3} J(2) = \frac{1}{3} [0 + 4]$$

$$\boxed{J(2) = \frac{4}{3}}$$

Example Data and its Solution

- Plotting the function...



Example Data and its Solution

- Plotting the function...

$$J(m) = \frac{dJ}{dm} = 0 \quad \frac{dJ}{dm} = \frac{d}{dm} (\text{loss fn})$$

- Convex function i.e., it has only one critical point, which is the global minimal.
- Set derivative equal to zero and the critical point acquired is the minimal

5, 12

Example Data and its Solution

- Now solve it for the optimal value.

$$\begin{aligned}
 \frac{dJ}{dm} &= \frac{d}{dm} \left(\frac{1}{3} ((m-1)^2 + (2m-2)^2) \right) \\
 &= \frac{1}{3} \left[\frac{d}{dm} (m-1)^2 + \frac{d}{dm} (2m-2)^2 \right]. \\
 &= \frac{1}{3} \left[2(m-1) + 2(2m-2) \cdot 2 \right]. \\
 &= \frac{1}{3} [\underline{2m-2} + \underline{\underline{8m-8}}]
 \end{aligned}$$

Example Data and its Solution

- Now solve it for the optimal value.

$$\frac{dJ}{dm} = \frac{1}{3} \left\{ (2m - 10) \right\}$$

$$\frac{dJ}{dm} = 0$$

$$\cancel{\frac{1}{3}} m - \cancel{\frac{10}{3}} = 0$$

m = 1

Example Data and its Solution

- Now solve it for the optimal value.

① Select model based on data.
↳ find the loss \Rightarrow suitable me
↳ for the optimization.
② derivative = $\partial \rightarrow$ clos



Hypothesis Function with 2 Variables

- Let's setup regression for linear fur
- The hypothesis function is:

$$J = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

$$\hat{y}_i = mx_i + b$$

Gradient of the loss function

- We get the following expressions for the gradient of the cost function

$$\frac{\partial J}{\partial m} = \frac{1}{N} \sum_{i=1}^N -2(y_i - \hat{y}_i)x_i$$

$\frac{\partial}{\partial} (y_i - \hat{y}_i)^2$

$$\frac{\partial J}{\partial b} = \frac{1}{N} \sum_{i=1}^N -2(y_i - \hat{y}_i)$$

$(m x_i + b)$

Gradient of the loss function

- Simplifying the above expressions, we get:

$$\frac{\partial J}{\partial m} = \cancel{\frac{2}{N}} \sum_{i=1}^N y_i x_i + \cancel{\frac{2m}{N}} \sum_{i=1}^N x_i^2 + \cancel{\frac{2b}{N}} \sum_{i=1}^N x_i$$

$$\frac{\partial J}{\partial b} = \cancel{-\frac{2}{N}} \sum_{i=1}^N y_i + \cancel{\frac{2m}{N}} \sum_{i=1}^N x_i + \cancel{\frac{2b}{N}} \sum_{i=1}^N 1$$

Gradient of the loss function

- Setting the Gradient equal to 0, and solving for m and b, we get

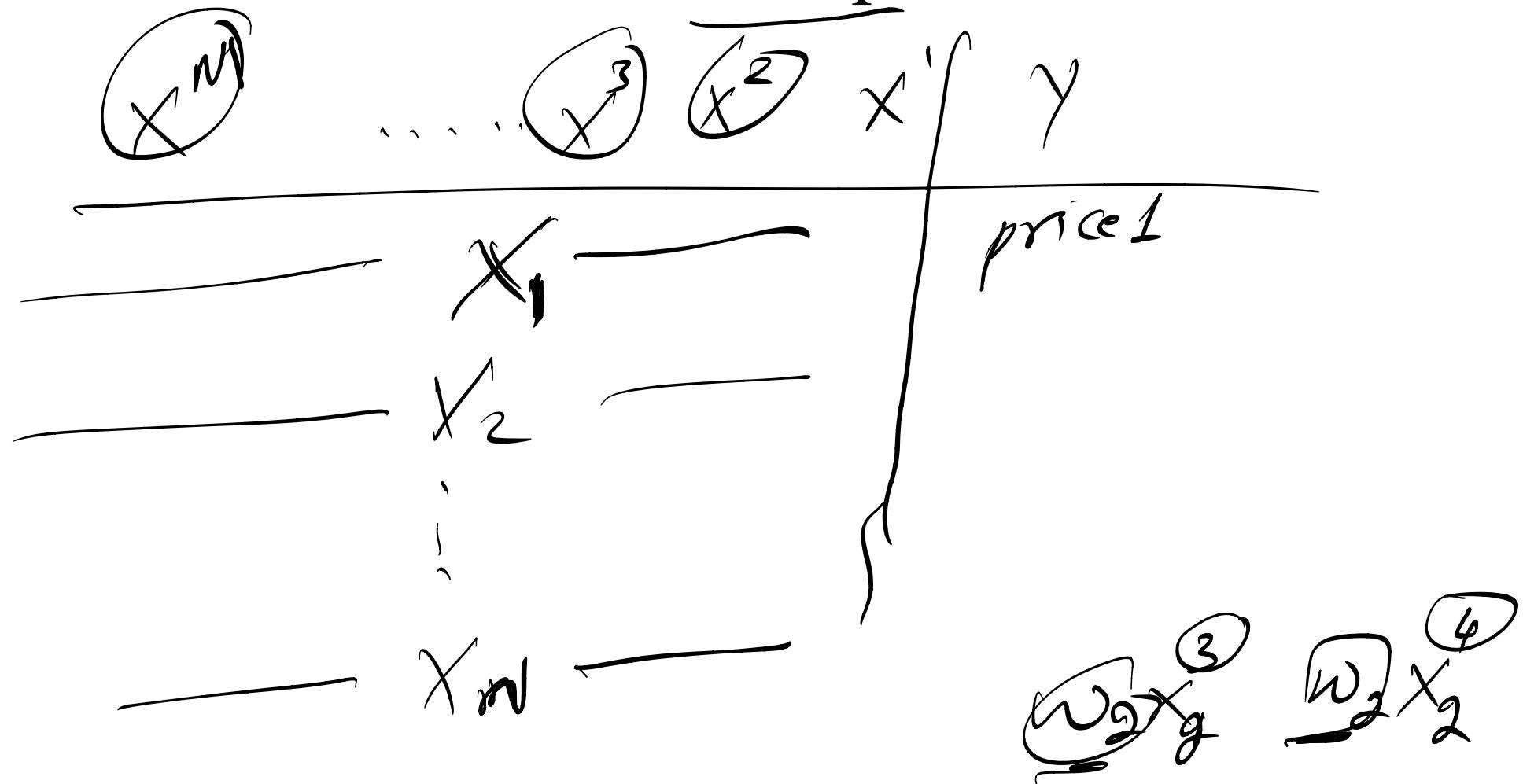
$$\left[\begin{matrix} \frac{\sum_i x_i^2}{N} & \frac{\sum_i x_i}{N} \\ \frac{\sum_i x_i}{N} & 1 \end{matrix} \right] \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \frac{\sum_i x_i y_i}{N} \\ \frac{\sum_i y_i}{N} \end{bmatrix}$$

~~A~~ . ~~B~~

$\theta = A^{-1} B$

$\theta = A^{-1} B$

Solution for Data with Multiple Features



Solution for Data with Multiple Features

$$\hat{y}_1 = \underline{1} \cdot \underline{w_0} + \underline{w_1} \underline{x_1^1} + \dots + \underline{w_m} \underline{x_1^m}$$

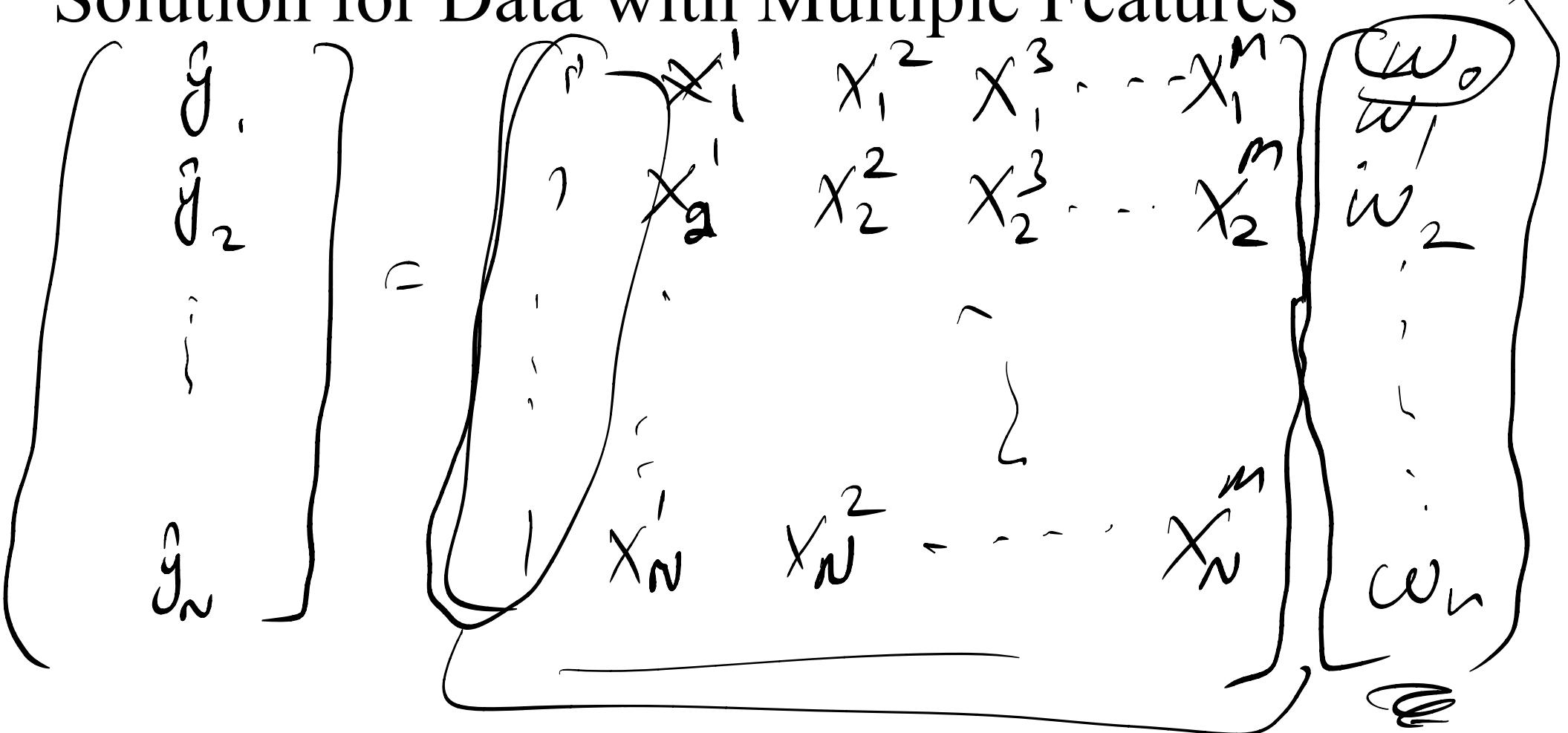
$$\hat{y}_2 = \underline{1} \cdot \underline{w_0} + \underline{w_1} \underline{x_2^1} + \dots + \underline{w_m} \underline{x_2^m}$$

$$\hat{y}_3 =$$

$$\vdots$$

$$\hat{y}_n = \underline{1} \cdot \underline{w_0} + \underline{w_1} \underline{x_n^1} + \underline{w_2} \underline{x_n^2} + \underline{w_3} \underline{x_n^3} + \dots + \underline{w_m} \underline{x_n^m}$$

Solution for Data with Multiple Features



Solution for Data with Multiple Features

$$\begin{aligned}
 \bar{Y} &= X \\
 \text{Loss} &= \sum_{i=1}^n (y_i - \hat{y}_i)^2 \\
 &= \sum_{i=1}^n (\epsilon_i)^2
 \end{aligned}$$

Solution for Data with Multiple Features

$$\begin{aligned}
 \text{Loss} &= \frac{1}{n} \sum_{i=1}^n \epsilon_i^2 \\
 &= \frac{1}{n} \underbrace{\epsilon \cdot \epsilon^T}_{\epsilon^T \cdot \epsilon} \\
 \text{Loss} &= \underbrace{\epsilon^T \cdot \epsilon}_{\sum (\epsilon_1^2 + \epsilon_2^2 + \dots + \epsilon_n^2)} \\
 &\quad \left[\epsilon_1^T \cdot \epsilon_1, \epsilon_2^T \cdot \epsilon_2, \dots, \epsilon_n^T \cdot \epsilon_n \right]
 \end{aligned}$$

Solution for Data with Multiple Features

$$\begin{aligned}
 \text{loss} &= \underline{\epsilon}^T \cdot \underline{\epsilon} \\
 &= (\hat{y} - y)^T \cdot (\hat{y} - y) \\
 \text{loss} &= (\underbrace{m_x - y}_m)^T (\underbrace{m_x - y}_m) = 0 \\
 \frac{\partial}{\partial m} (\underbrace{m_x - y}_m)^T (\underbrace{m_x - y}_m) &= 0
 \end{aligned}$$

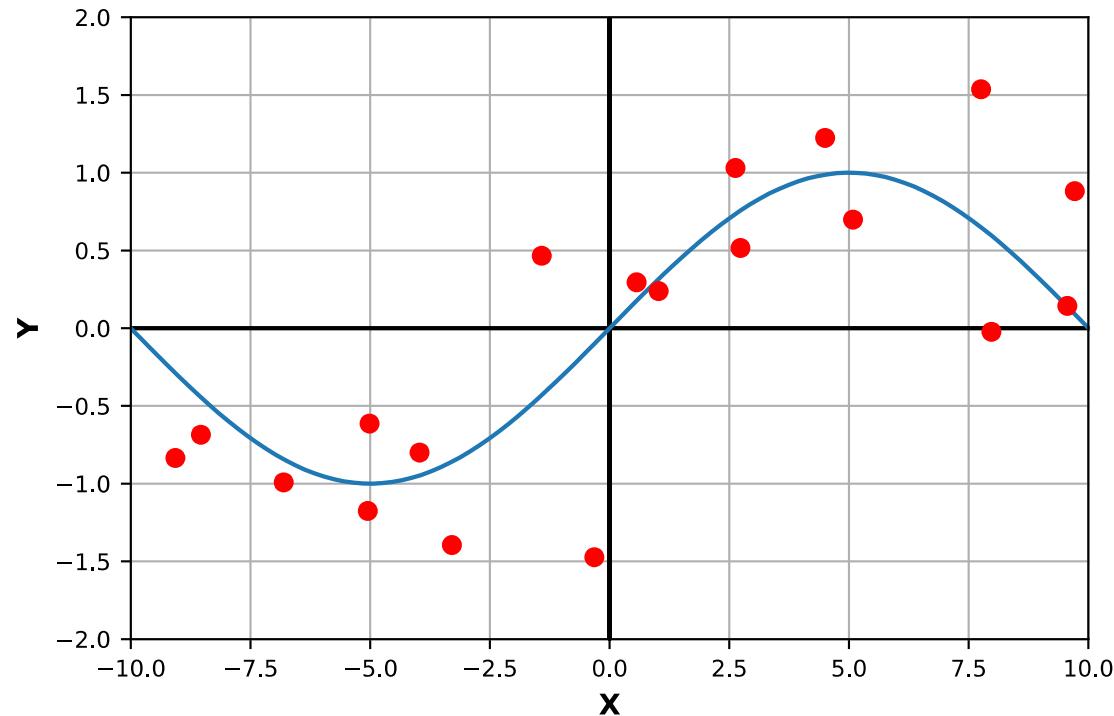
Solution for Data with Multiple Features

$$\theta = \frac{(X^T X)^{-1} (X^T y)}{A^{-1} B}$$

Normal Equation

Fitting Non-linear Data

- What if y is a non-linear function of x , will this approach still work?



Transforming the Feature Space (Feature Engineering)

- We can transform features x_i

$$x_i = (x_i^1, x_i^2, x_i^3, \dots, x_i^m)$$

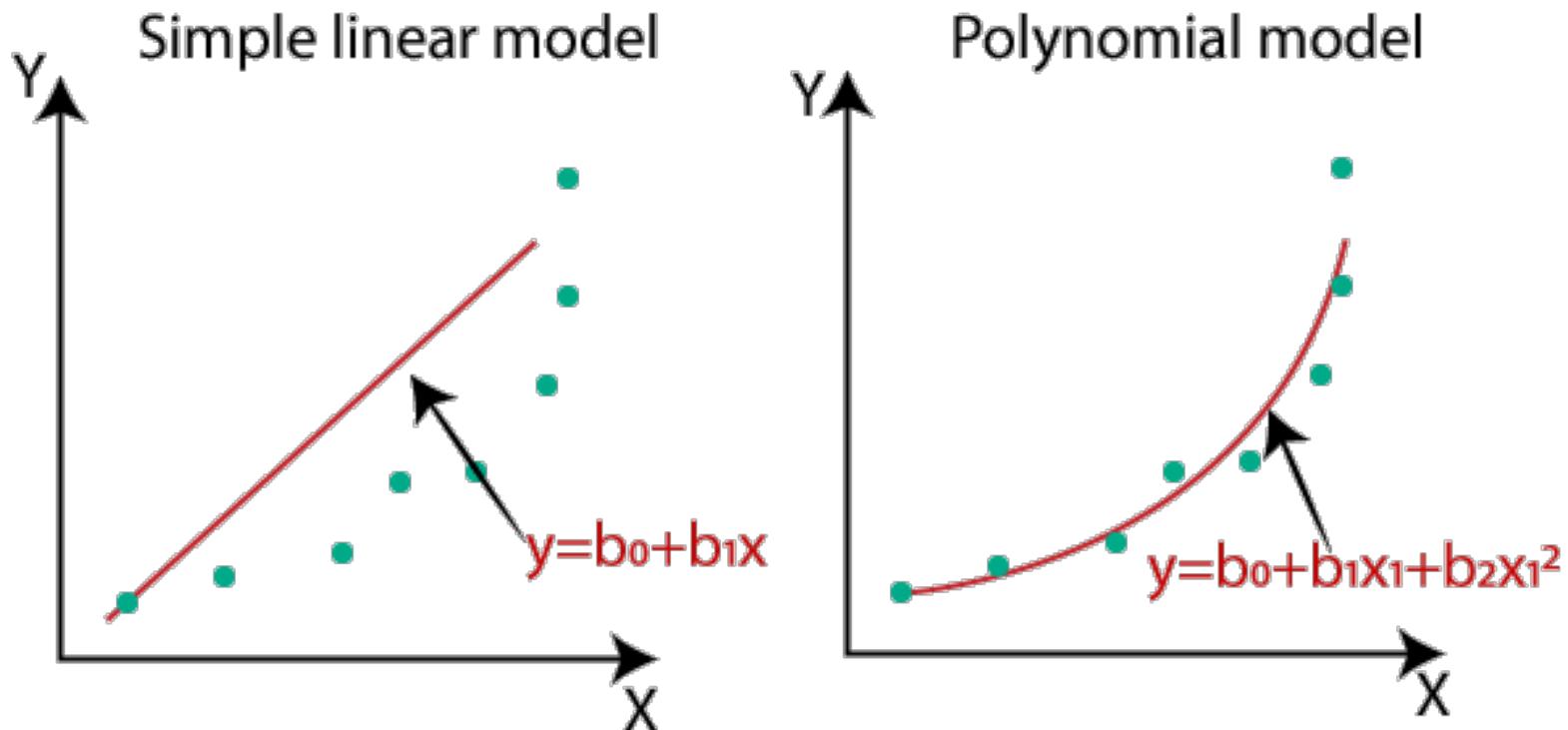
- We will apply some non-linear transformation ϕ :

$$\phi : \mathbb{R}^m \rightarrow \mathbb{R}^M$$

- For example, Polynomial transformation:

$$\phi(x_i) = \{1, x_i^1, x_i^{1,[2]}, \dots, x_i^{1,[k]}, x_i^2, x_i^{2,[2]}, \dots, x_i^{2,[k]}, \dots, x_i^m, x_i^{m,[2]}, \dots, x_i^{m,[k]}\}$$

Transforming the Feature Space (Feature Engineering)



Transforming the Feature Space (Feature Engineering)

Example: assume you have:

x_1 : Length
 x_2 : Width

You can add x_3 : Area = $x_1 * x_2$ to the dataset.

Other types:

- Cosine, splines, radial basis functions, etc.
- Encoding (Label encoding, One-hot,...)
- Domain-related features (e.g. financial measures)
- Time-related features (Day, month, year,...)

Issues with the Approach

- Assume we have 100 variables instead of 2.
- Calculating gradients like this can quickly become tedious
- **Notice:** Each term on either side of the expression can be written a dot product of two vectors (maybe we can calculate it more efficiently)?
- Let's explore if we can do something better through **vectorization** (Writing equations as matrices).

Vectorization

- To truly appreciate the power of vectorization. Let's make the problem a little more complex. The hypothesis function is now

$$\hat{y}_i = w_0 + w_1 x_i^1 + w_2 x_i^2 + \cdots + w_M x_i^M$$

- Where w_j are the unknown weights of the data x^j features of the input
- Next, we denote the discrepancy between y_i and \hat{y}_i as ϵ_i

$$y_i = \hat{y}_i + \epsilon_i$$

↑ error

Vectorization

- Now let's collect the above equation for all N datapoints

$$y_1 = \hat{y}_1 + \epsilon_1$$

$$y_2 = \hat{y}_2 + \epsilon_2$$

.

.

.

$$y_N = \hat{y}_N + \epsilon_N$$

Vectorization

- Replacing the values of \hat{y} , we get:

$$y_1 = w_0 + w_1 x_1^1 + w_2 x_1^2 + \dots + w_M x_1^M + \epsilon_1$$

$$y_2 = w_0 + w_1 x_2^1 + w_2 x_2^2 + \dots + w_M x_2^M + \epsilon_2$$

.

.

.

$$y_N = w_0 + w_1 x_N^1 + w_2 x_N^2 + \dots + w_M x_N^M + \epsilon_N$$

Vectorization

- Collecting the equations in matrix form:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} 1 & x_1^1 & x_1^2 & \dots & x_1^M \\ 1 & x_2^1 & x_2^2 & \dots & x_2^M \\ 1 & x_3^1 & x_3^2 & \dots & x_3^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N^1 & x_N^2 & \dots & x_N^M \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ \vdots \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \vdots \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Vectroization

- Notice the rows of the matrix on the right are data samples:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} \dots & \mathbf{x}_1 & \dots \\ \dots & \mathbf{x}_2 & \dots \\ \dots & \mathbf{x}_3 & \dots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \dots & \mathbf{x}_N & \dots \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ \vdots \\ w_M \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \vdots \\ \epsilon_N \end{bmatrix}$$

Vectorization

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$$

- Let's formalize some notaitons:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ y_N \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} \dots & \mathbf{x}_1 & \dots \\ \dots & \mathbf{x}_2 & \dots \\ \dots & \mathbf{x}_3 & \dots \\ & \ddots & \\ & \ddots & \\ \dots & \mathbf{x}_N & \dots \end{bmatrix} \quad \boldsymbol{\theta} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ \vdots \\ w_M \end{bmatrix} \quad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \vdots \\ \epsilon_N \end{bmatrix}$$

$$\mathbf{y} = \mathbf{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}$$

Cost function for the Vectorized form

- Notice that we are using the MSE cost function:

- Using the definition of epsilon we can write the above as:

$$J = \underbrace{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}_{\text{Using the definition of epsilon}} = \frac{1}{N} \sum_{i=1}^N (\epsilon_i)^2$$

- Using the definition of dot product the above can be written as:

$$J = \frac{1}{N} \sum_{i=1}^N (\epsilon_i)^2 = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon}$$

Optimization

- The optimization problem is now:

$$\min_{\theta} \epsilon^T \epsilon$$

$$\min_{\theta} \epsilon^T \epsilon = \min_{\theta} (\mathbf{y} - (\mathbf{X}\theta))^T (\mathbf{y} - (\mathbf{X}\theta))$$

- We will use chain rule to calculate the gradient of the cost function:

$$\frac{\partial}{\partial \theta} J = \frac{dJ}{d\epsilon} \nabla_{\theta} \epsilon$$

Linear Least Squares

- We get:

$$\frac{\partial}{\partial \theta} J = \mathbf{X}^T 2(\mathbf{y} - \mathbf{X}\theta)$$

- Setting it equal to zero we can solve for θ :

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

Closed-form solution for Linear Regression

$$x^2 + y^2 = r$$

$$2x^2 + 2y^2 = 2r$$

Hypothesis Function with 2 Variables

- Let's setup regression for linear function in two variables:
- The hypothesis function is:

$$\hat{y}_i = mx_i + b$$

- Similar to the previous problem our loss function is:

$$J = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

- Let's calculate the partial derivatives of the loss function w.r.t. m, b

ML Algorithms Perspectives:

- We can look into ML algorithms from two perspective:
 - **Loss Minimization** Problem (like what we did).
 - **Probability Maximization** Problem (using Maximum Likelihood Estimation).
- Both should result in the same solution.

Probalistic Interpretation of Linear Regression and MLE

- We can also look at the probabilistic Interpretation of Linear Regression.
- Keeping everything else same as the previous formulation

$$y_i = \mathbf{x}_i^T \boldsymbol{\theta} + \epsilon_i$$

- Now assume that $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, then $y_i \sim \mathcal{N}(\mathbf{x}_i^T \boldsymbol{\theta}, \sigma^2)$
- We can write the conditional distribution as :

$$\text{P}(y_i | \mathbf{x}_i) \sim \mathcal{N}(0, \sigma^2)$$

Probalistic Interpretation of LR

- Let's assume that all data point in the dataset are i.i.d. (independent identically distributed). Then we have:

$$\text{IP}(\mathcal{D}) = \prod_{i=1}^N \text{IP}(\mathbf{x}_i, y_i)$$

- Using Bayes Theorem we can write:

$$\prod_{i=1}^N \text{IP}(\mathbf{x}_i, y_i) = \prod_{i=1}^N \text{IP}(\mathbf{x}_i) \text{IP}(y_i | \mathbf{x}_i)$$

Maximum Likelihood Estimator

- In simple words, given the Dataset we want to find the values of the unknown parameters which maximize the probability of the Dataset.
- Using the definition of the conditional distribution we have

$$\text{P}(y_i | \mathbf{x}_i) = \frac{1}{\sigma\sqrt{2\pi}} \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta}))$$

- Using the definition we get

$$\prod_{i=1}^N \text{P}(\mathbf{x}_i, y_i) = \prod_{i=1}^N \text{P}(\mathbf{x}_i) \prod_{i=1}^N \frac{1}{\sigma\sqrt{2\pi}} \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta}))$$

Maximum Likelihood Estimator

- Let's try to maximaize:

$$\prod_{i=1}^N \text{P}(\mathbf{x}_i, y_i) = \prod_{i=1}^N \text{P}(\mathbf{x}_i) \prod_{i=1}^N \frac{1}{\sigma\sqrt{2\pi}} \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta}))$$

- Note that

$$\arg \max_{\boldsymbol{\theta}} \prod_{i=1}^N \text{P}(\mathbf{x}_i, y_i) = \arg \max_{\boldsymbol{\theta}} \prod_{i=1}^N \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2)$$

Maximum Likelihood Estimator

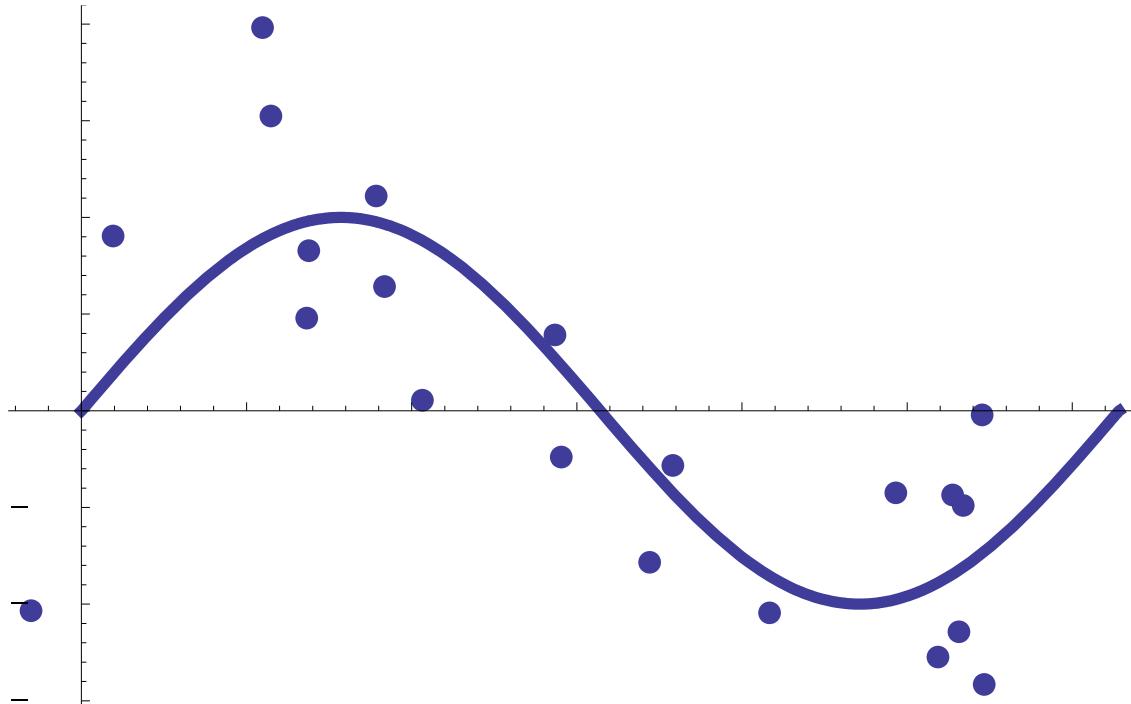
- Furthermore, since the right hand side of the above equation is monotonic in \theta the arg max will not change if we take log of the expression

$$\arg \max_{\theta} \prod_{i=1}^N \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2) = \arg \max_{\theta} \sum_{i=1}^N -(y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2$$

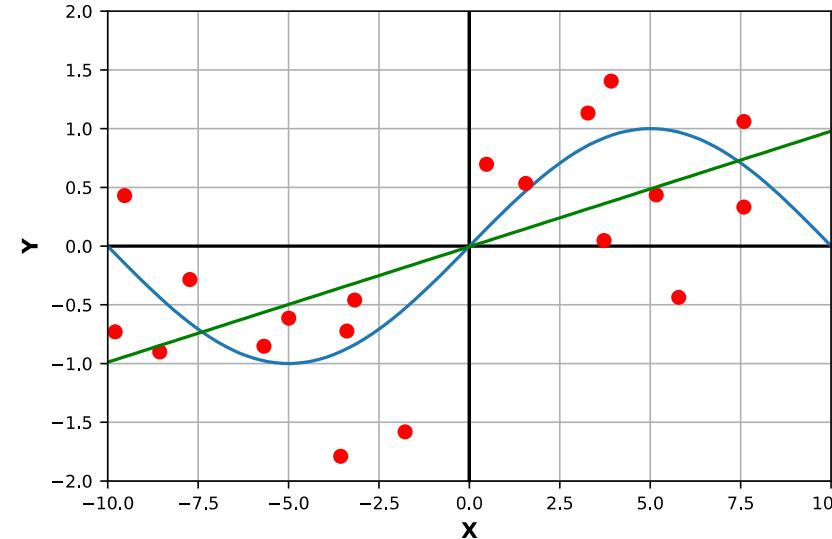
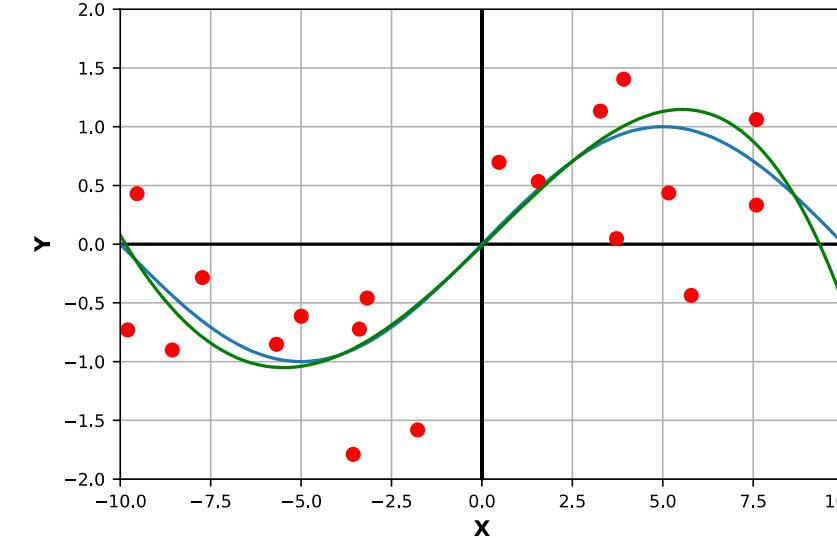
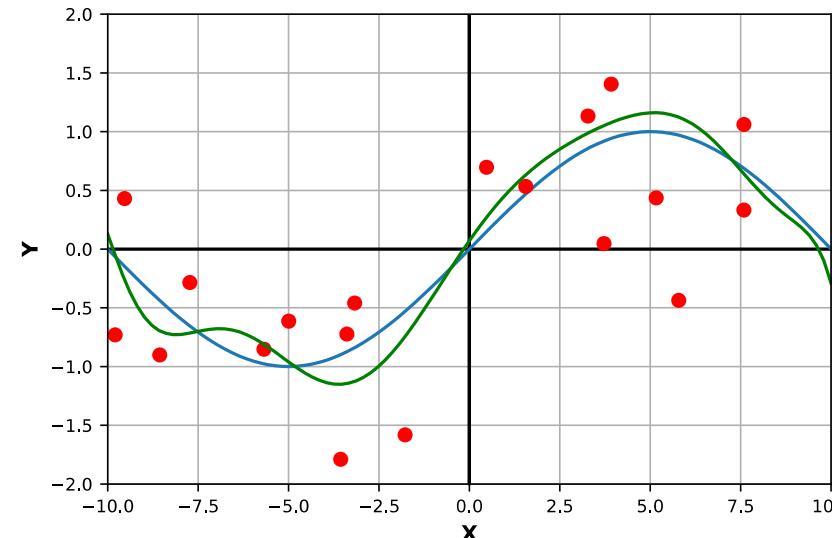
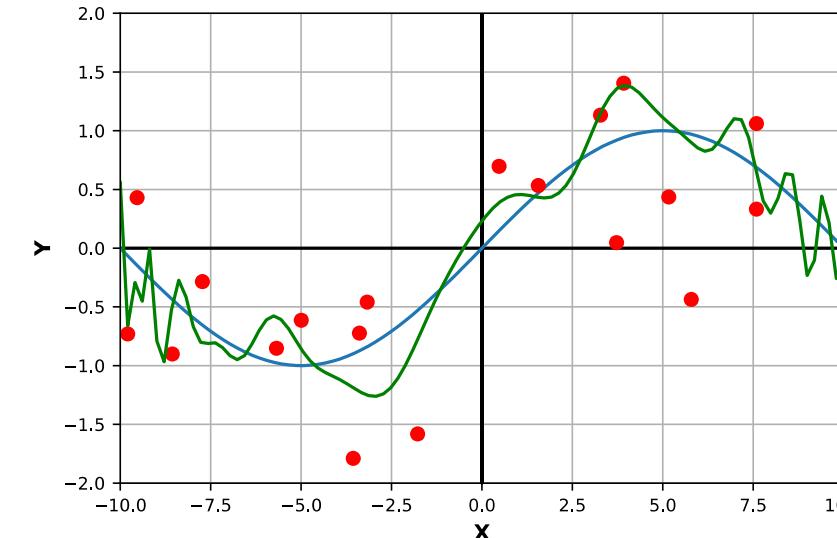
- Notice that the right hand side is minimising the MSE.
- Hence solution of minimizing the MSE is equivalent to Maximum Likelihood Estimator for linear regression

Bias and Variance

- What if Y has a non-linear response?

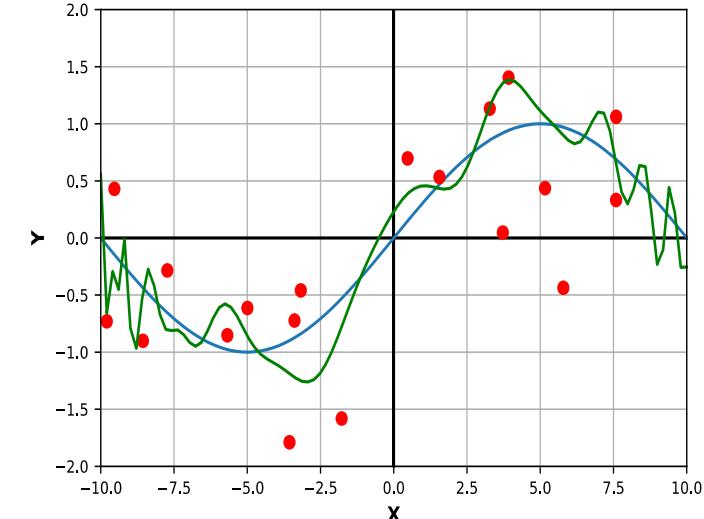
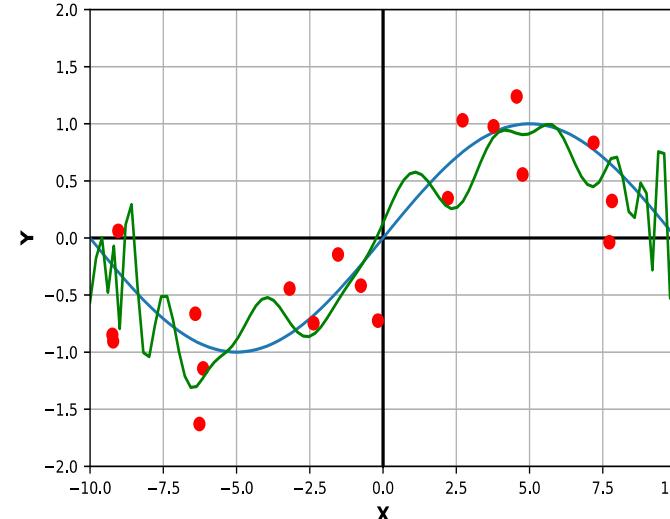
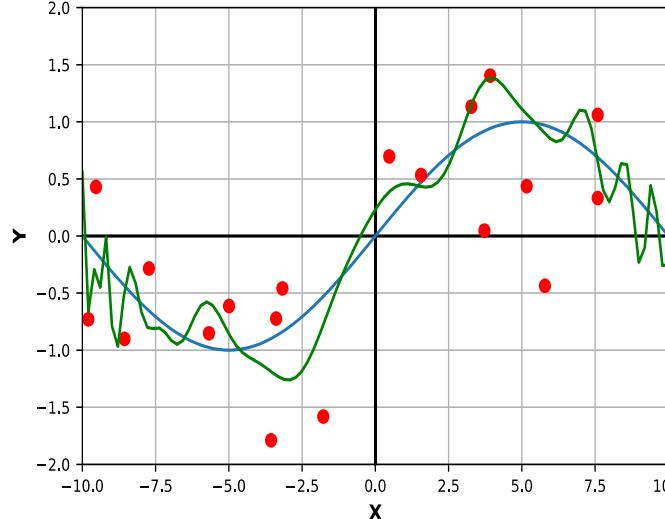
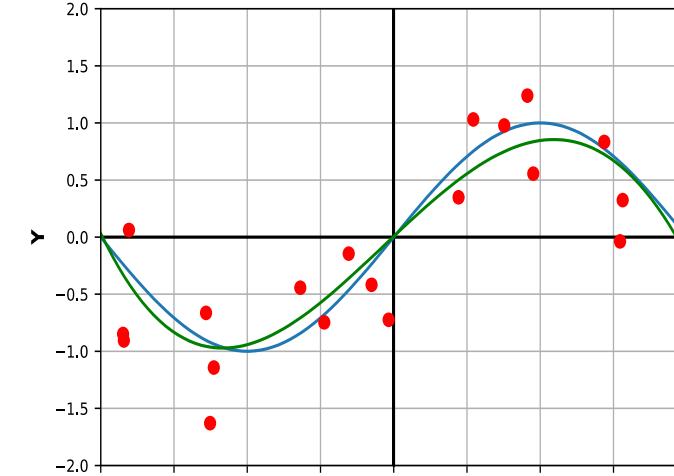
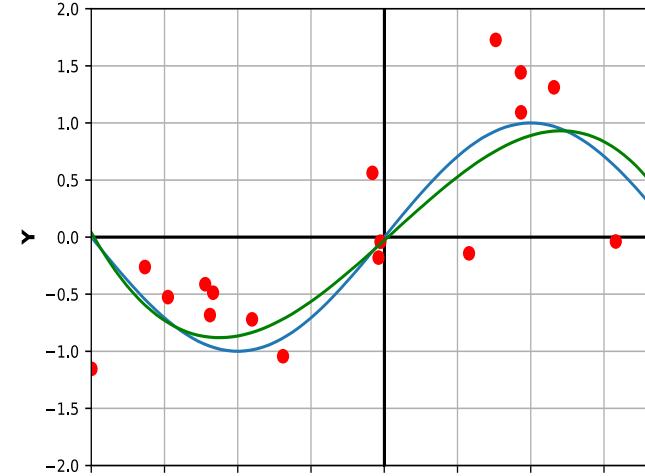
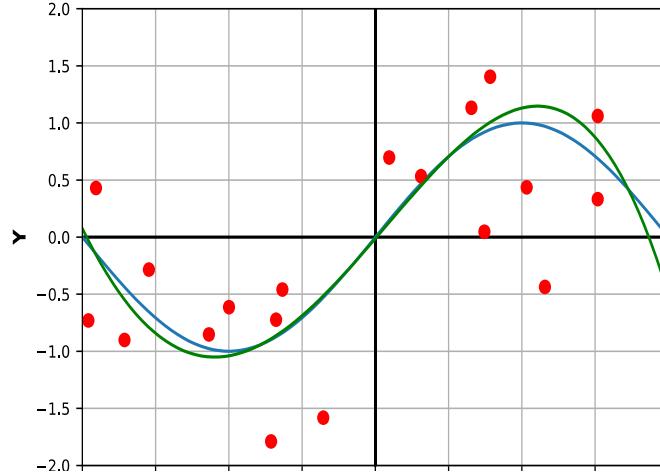


- Can we still use a linear model?

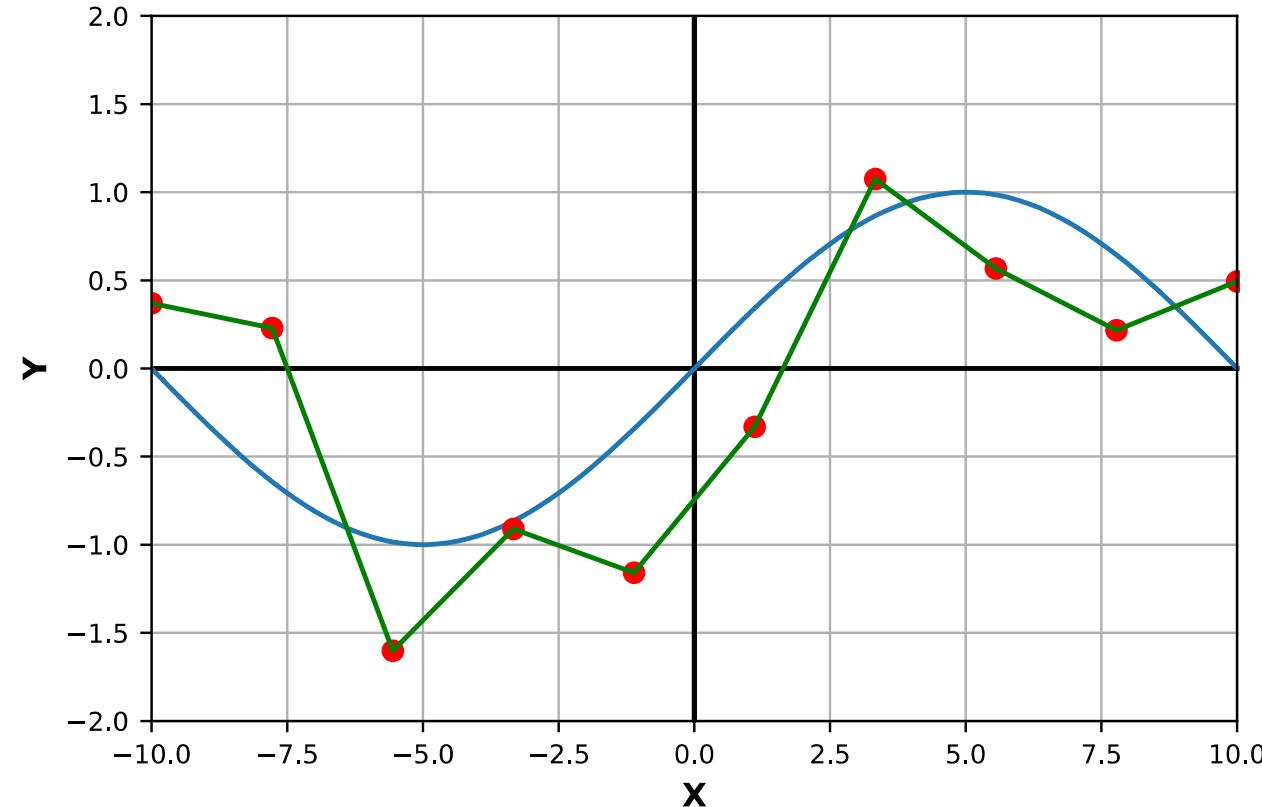
$\{1, x\}$

 $\{1, x, x^2, x^3, x^4\}$

 $\{1, x, x^2, \dots, x^9, x^{10}\}$

 $\{1, x, x^2, \dots, x^{99}, x^{100}\}$


What is Bias and Variance?

$$\{1, x, x^2, x^3, x^4\}$$



Real Bad Overfit?



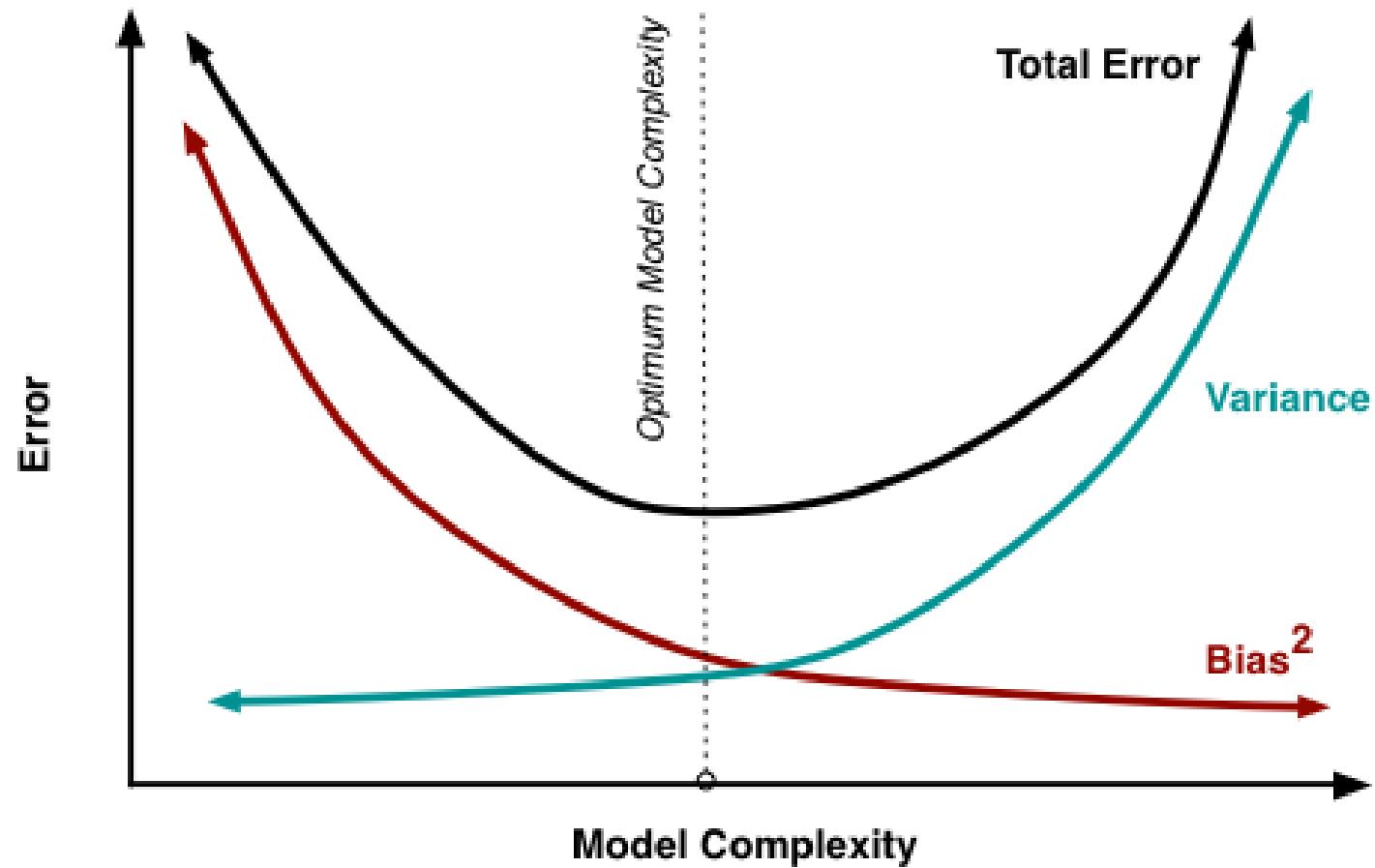
Bias-Variance Tradeoff



- So far we have minimized the error (loss) with respect to **training data**
 - Low training error does not imply good expected performance: **over-fitting**
- We would like to reason about the **expected loss (Prediction Risk)** over:
 - Training Data: $\{(y_1, x_1), \dots, (y_n, x_n)\}$
 - Test point: (y_*, x_*)
- We will decompose the expected loss into:

$$\mathbf{E}_{D,(y_*,x_*)} [(y_* - f(x_*|D))^2] = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

Bias Variance Plot



Data Split

- To ensure your model doesn't overfit to the training data, you should have another subset called **testing data**.
- You will evaluate your model against this subset, and based on its **metric score (e.g. accuracy)** you will decide if it's overfitting or not.
- But how should I split my data?

Data Split

- **Hold-out set:**
 - A portion of the dataset set aside and not used during training.
 - E.g. 80% for training and 20% for testing.

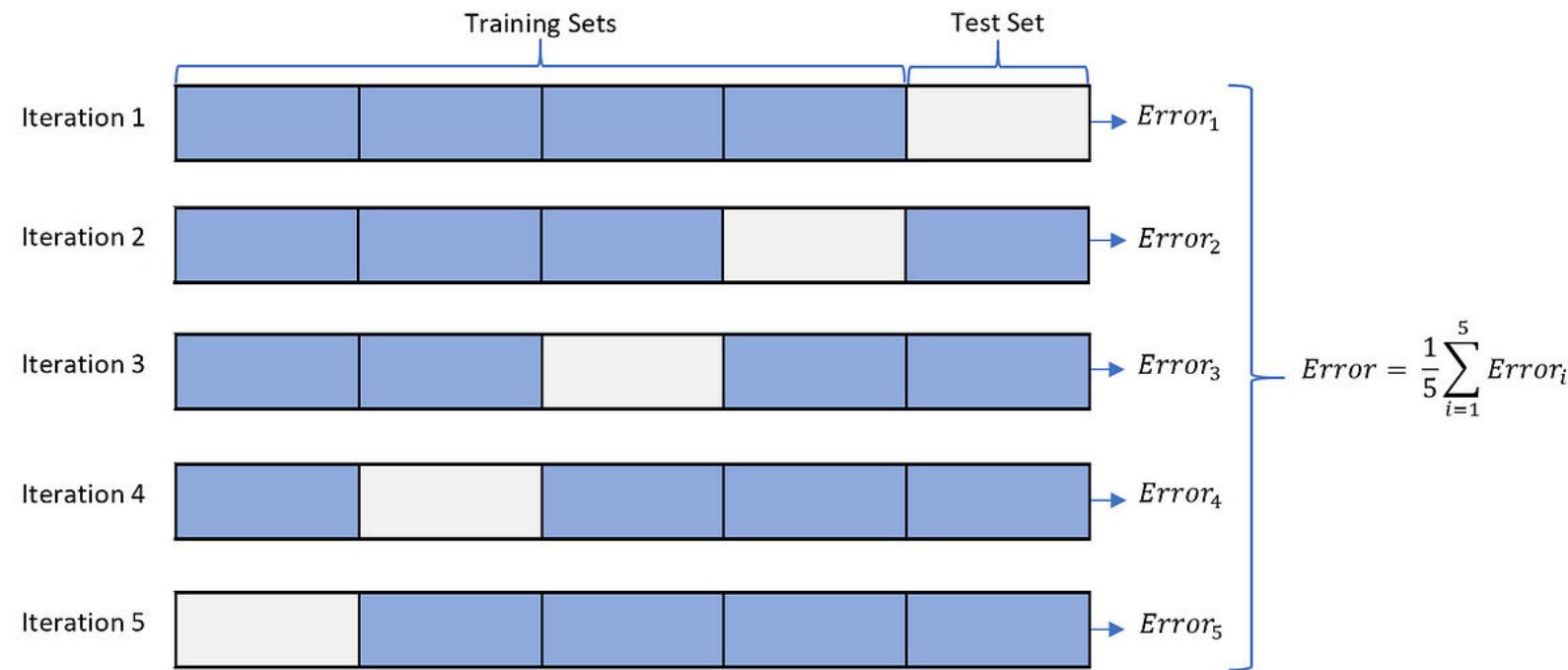
Issues:

- Imagine you have these labels: [1, 1, 2, 2, 2, 3, 3, 3, 3, 3] and you took last 30% as test: [3,3,3]. You didn't include 1 and 2 in test!
Solution: Always shuffle before split: [3, 2, 3, 1, 3, 2, 3, 1, 3, 2] → test: [1,3,2]
- My dataset is small. Taking 20% as test would not be representative!
Solution: Use KFold.

Data Split

- **K-Fold Cross Validation (CV):**

- Split data into k parts (folds), trains on $k - 1$ folds, test on the remaining fold, and repeats k times then average the scores.



Regularization

Regularization

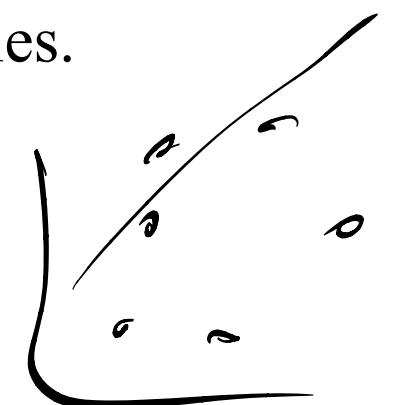
$$\omega = \begin{bmatrix} 1 \\ 5 \\ 0 \\ -1 \end{bmatrix}$$

- Non-linear function with multiple independent variables.
- What do we do for single variable?

$x^1 = x^3 \quad x^2 \quad x^1 \quad | \quad y$

 $\hat{y} = w_0 + w_1 x^1 + w_2 x^2 + w_3 x^3$
 $\hat{y} = 1 + 5x^1 + 0 + x^3$

one solution



$$nor = \sqrt{w_1^2 + w_2^2 + w_3^2}$$

Regularization

- Non-linear function with **multiple independent variables**.
- What do we do for single variable?

$$\hat{y} = 1 + 4x^1 + 0 + (x^3) + x^3$$

$$\hat{y} = 1 + 4x^1 + 2x^3$$

$$\frac{\partial J + 1}{\partial \theta} =$$

$\begin{pmatrix} 1 & 4 & 0 & 2 \end{pmatrix}$
$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

$$\begin{pmatrix} 1 & 4 & 0 & 2 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

$$nor = \sqrt{16 + 0} = \sqrt{18}$$

Regularization

- Non-linear function with **multiple independent variables**.
- What do we do for single variable?

$$\theta = (X^T X + \lambda I) (X^T y)$$

$(\bar{X} - y)^T (\bar{X} - y) + \lambda \theta^T \theta$

Regularization

- Non-linear function with **multiple independent variables**.
- What do we do for single variable?

$$\text{L} \left((\mathbf{x}^\top \boldsymbol{\theta} - y)^\top (\mathbf{x}^\top \boldsymbol{\theta} - y) + \sum_{j=1}^m |\theta_j| \right)$$

Regularization: An Overview

The idea of regularization revolves around modifying the loss function L ; in particular, we add a regularization term that penalizes some specified properties of the model parameters

$$L_{reg}(\beta) = L(\beta) + \lambda R(\beta),$$

where λ is a scalar that gives the weight (or importance) of the regularization term.

Fitting the model using the modified loss function L_{reg} would result in model parameters with desirable properties (specified by R).

LASSO Regression

Since we wish to discourage extreme values in model parameter, we need to choose a regularization term that penalizes parameter magnitudes. For our loss function, we will again use MSE.

Together our regularized loss function is:

$$L_{LASSO}(\beta) = \frac{1}{n} \sum_{i=1}^n |y_i - \beta^\top \mathbf{x}_i|^2 + \lambda \sum_{j=1}^J |\beta_j|.$$

Note that $\sum_{j=1}^J |\beta_j|$ is the l_1 norm of the vector β

$$\sum_{j=1}^J |\beta_j| = \|\beta\|_1$$

Ridge Regression

Alternatively, we can choose a regularization term that penalizes the squares of the parameter magnitudes. Then, our regularized loss function is:

$$L_{Ridge}(\beta) = \frac{1}{n} \sum_{i=1}^n |y_i - \beta^\top \mathbf{x}_i|^2 + \lambda \sum_{j=1}^J \beta_j^2.$$

Note that $\sum_{j=1}^J \beta_j^2$ is the square of the L_2 norm of the vector β

$$\sum_{j=1}^J \beta_j^2$$

$$\sum_{j=1}^J \beta_j^2 = \|\beta\|_2^2$$

Choosing λ



In both ridge and LASSO regression, we see that the larger our choice of the **regularization parameter λ** , the more heavily we penalize large values in β ,

- If λ is close to zero, we recover the MSE, i.e. ridge and LASSO regression is just ordinary regression.
- If λ is sufficiently large, the MSE term in the regularized loss function will be insignificant and the regularization term will force β_{ridge} and β_{LASSO} to be close to zero.

To avoid ad-hoc choices, we should select λ using cross-validation.

Solution to ridge regression:

$$\beta = (X^T X + \lambda I)^{-1} X^T Y$$

The solution to the LASSO regression:

LASSO has no conventional analytical solution, as the L1 norm has no derivative at 0. We can, however, use the concept of **subdifferential** or **subgradient** to find a manageable expression. See a-sec2 for details.

The solution of the Ridge/Lasso regression involves three steps:

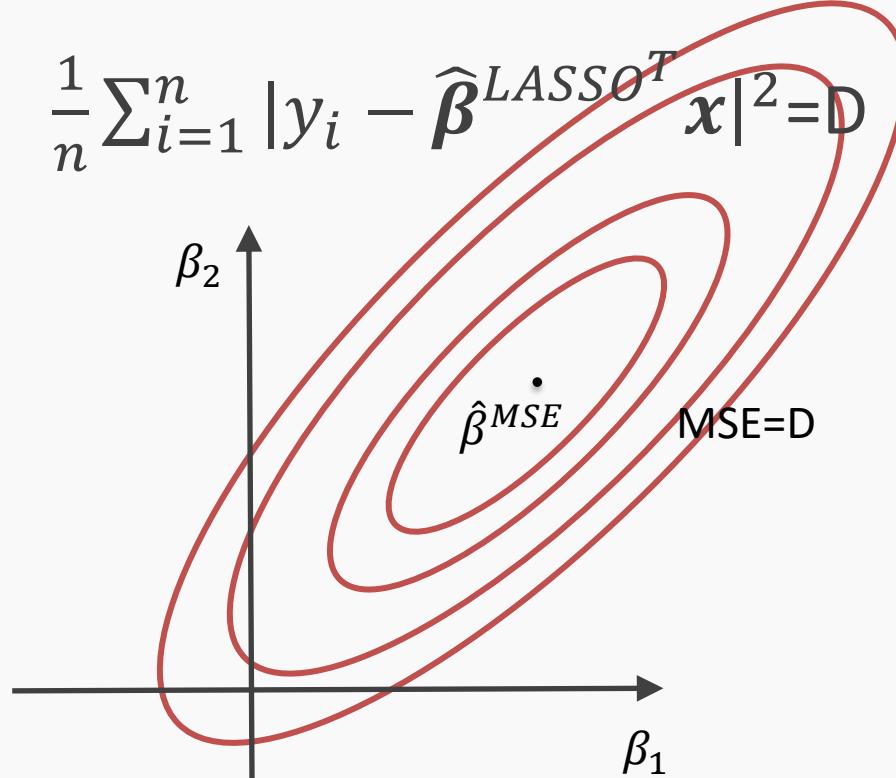
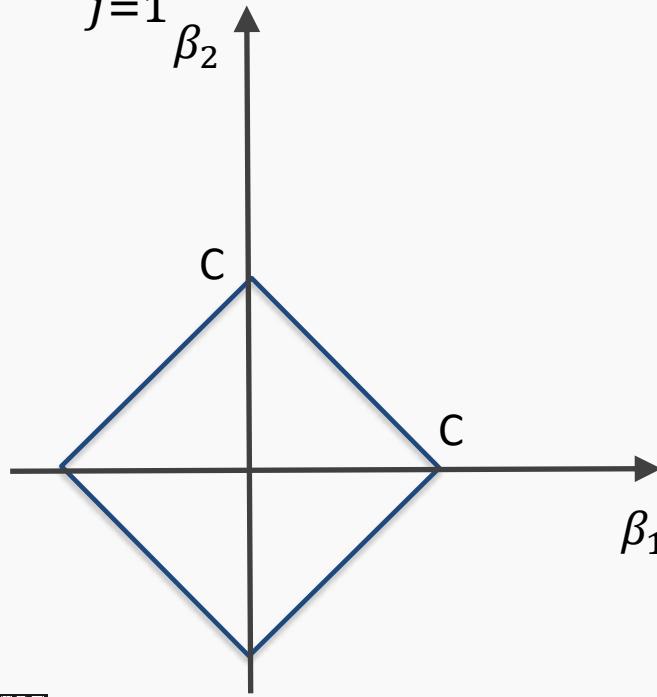
- Select λ
- Find the minimum of the ridge/Lasso regression loss function (using the formula for ridge) and record the **MSE on the validation/test set.**
- Find the λ that gives the smallest *MSE*

The Geometry of Regularization (LASSO)

$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^n |y_i - \boldsymbol{\beta}^T \mathbf{x}|^2 + \lambda \sum_{j=1}^J |\beta_j|$$

$$\hat{\boldsymbol{\beta}}^{LASSO} = \operatorname{argmin} L_{LASSO}(\boldsymbol{\beta})$$

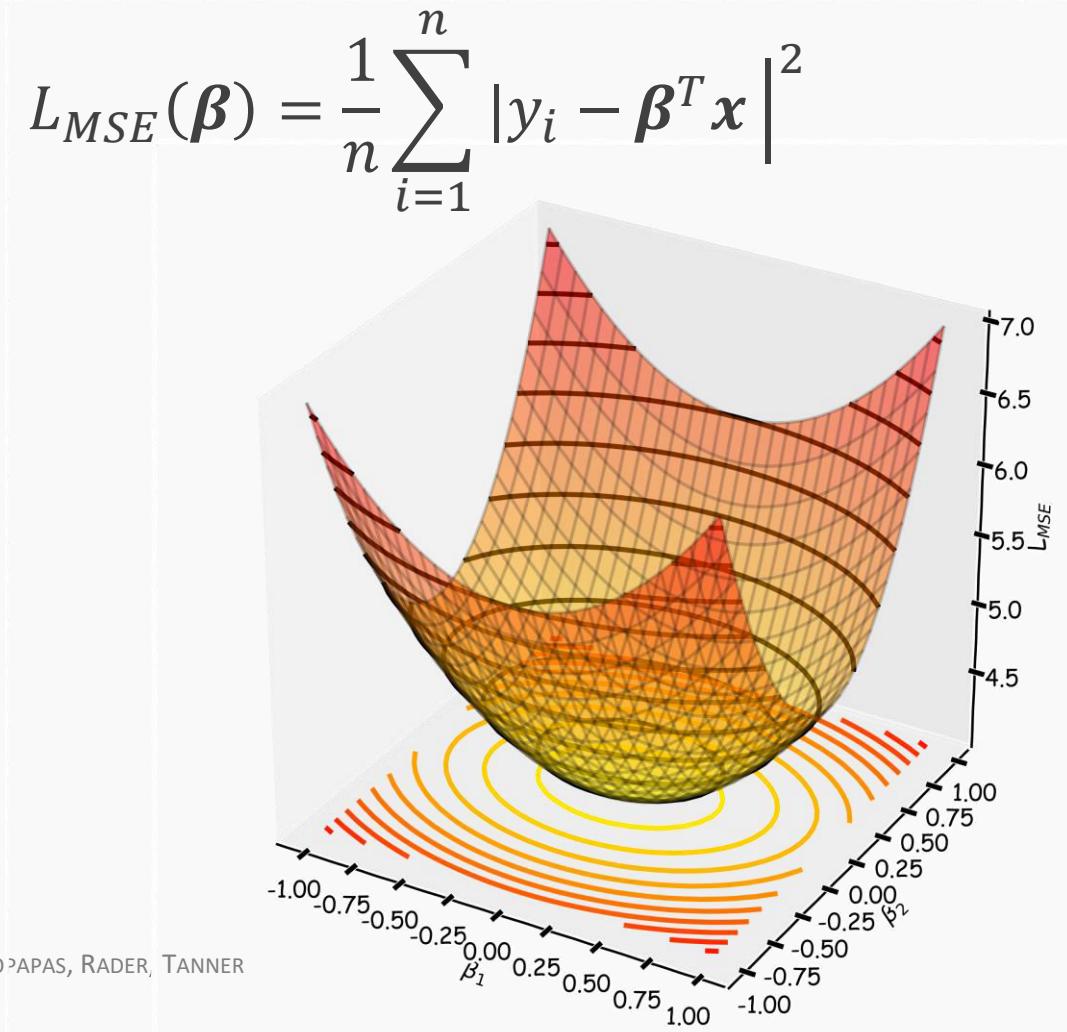
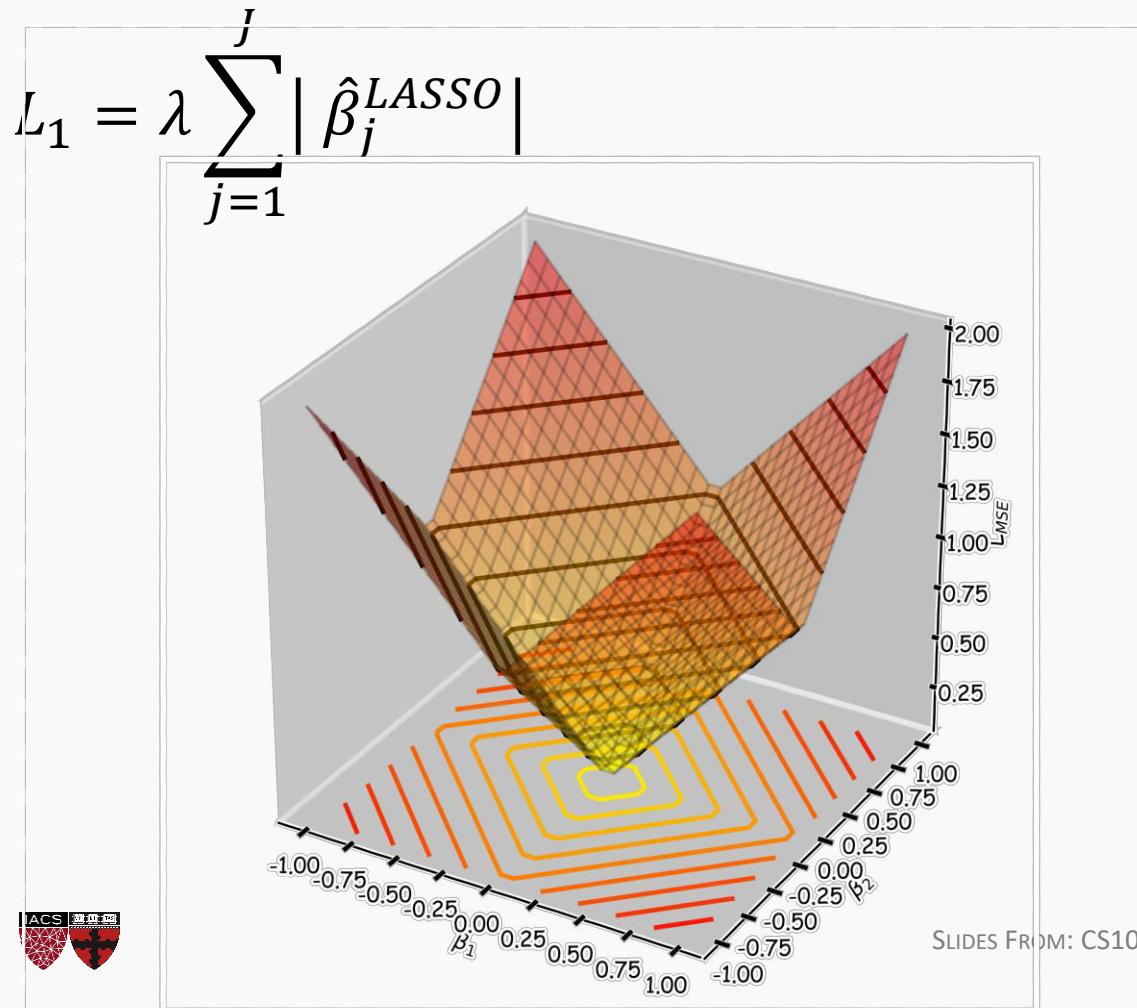
$$\lambda \sum_{j=1}^J |\hat{\beta}_j^{LASSO}| = C$$



The Geometry of Regularization (LASSO)

$$L_{LASSO}(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^n |y_i - \boldsymbol{\beta}^T \mathbf{x}|^2 + \lambda \sum_{j=1}^J |\beta_j|$$

$$\hat{\boldsymbol{\beta}}^{LASSO} = \operatorname{argmin} L_{LASSO}(\boldsymbol{\beta})$$

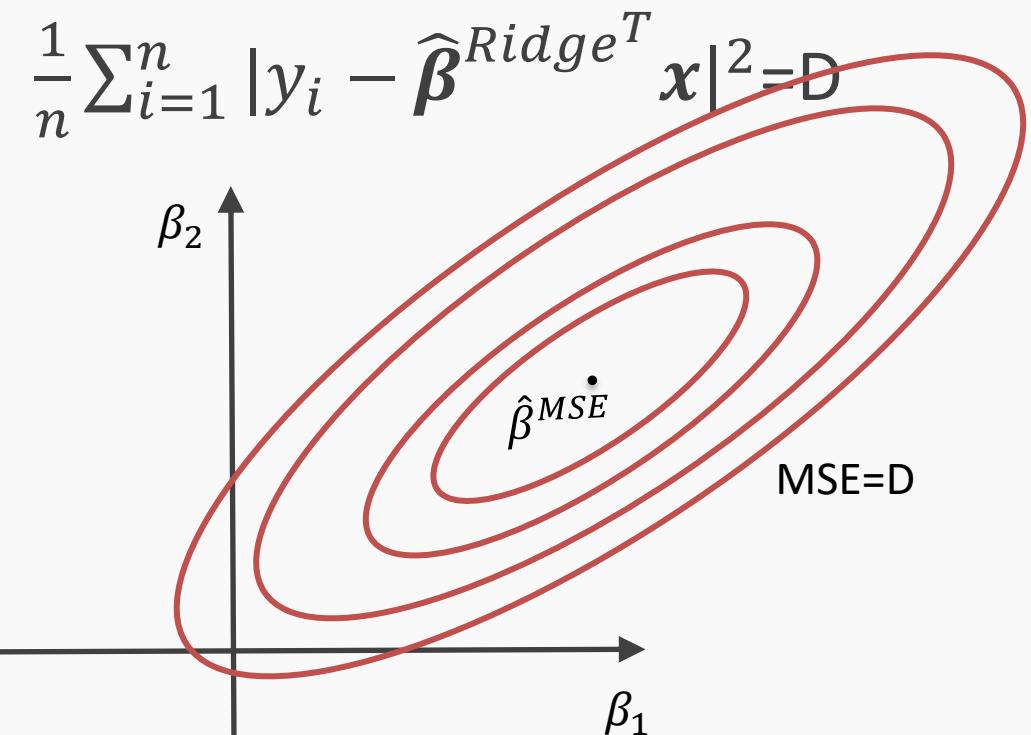
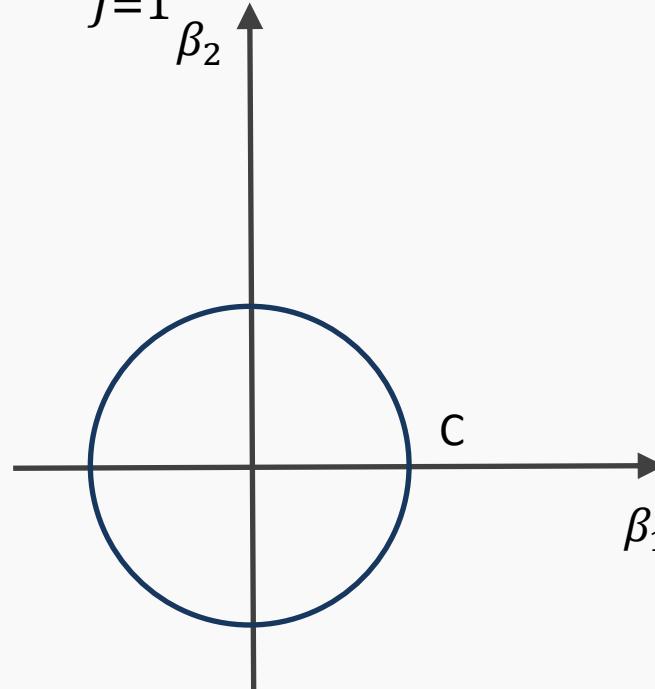


The Geometry of Regularization (Ridge)

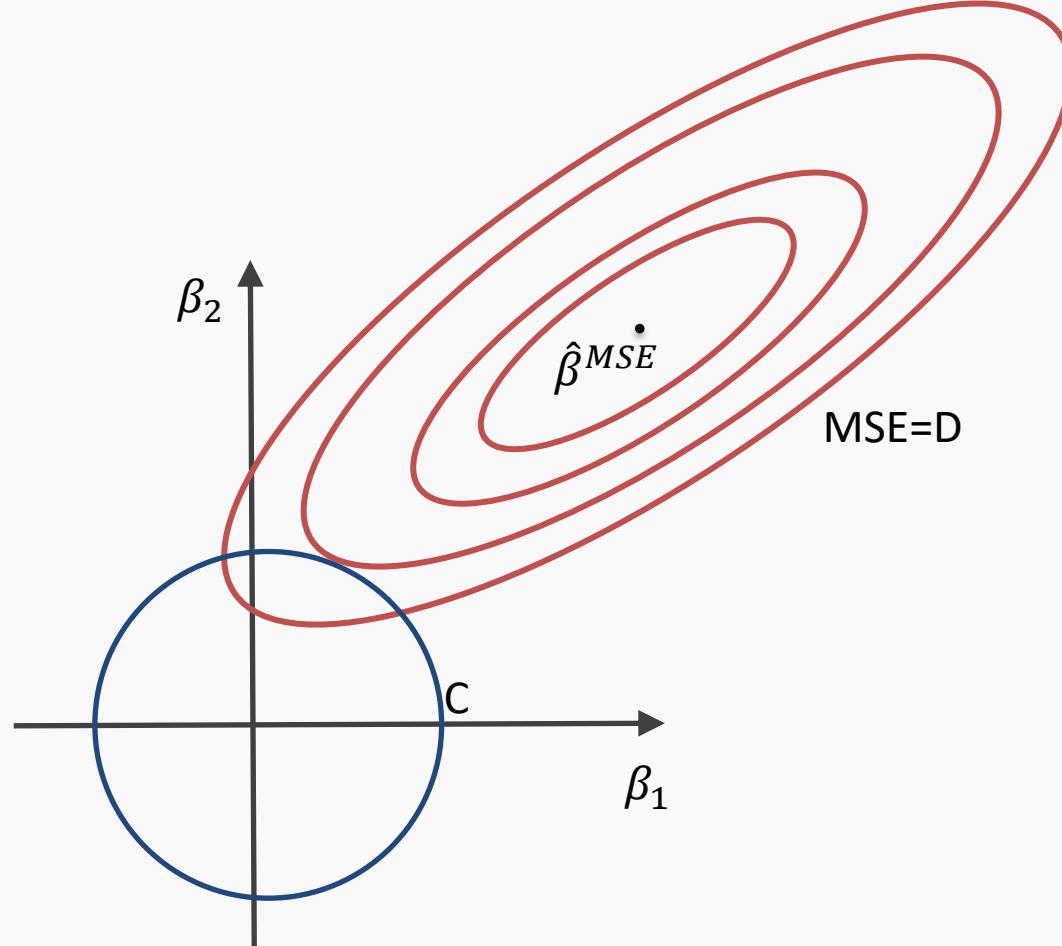
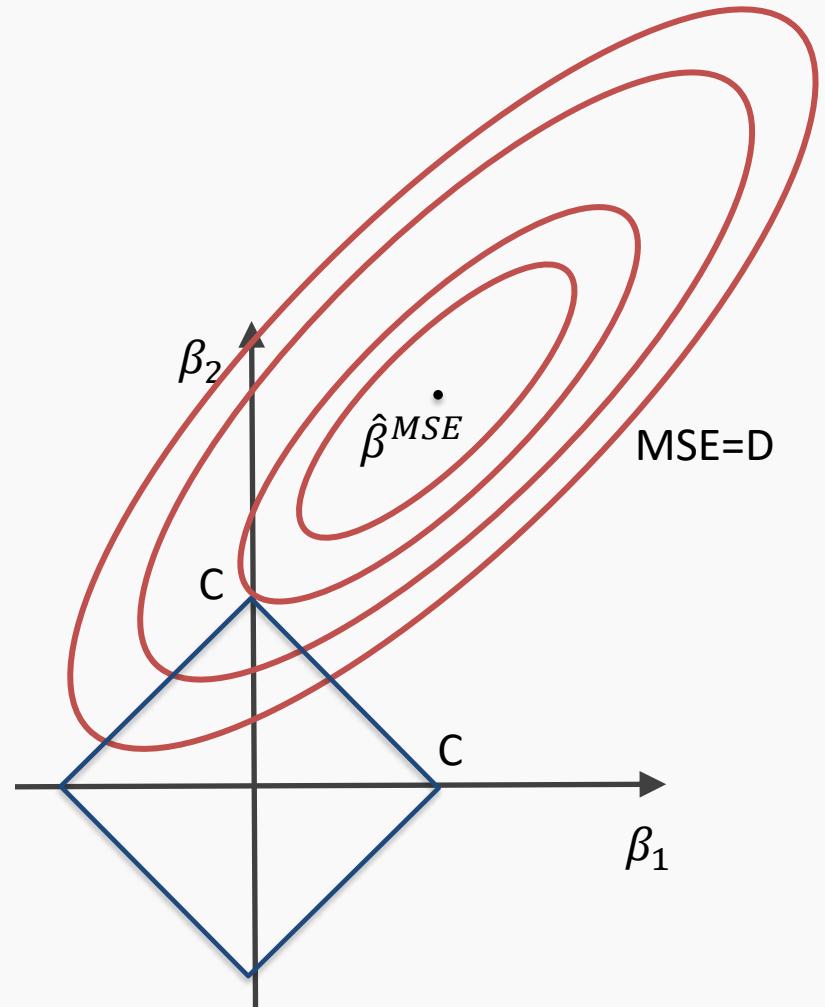
$$L_{Ridge}(\boldsymbol{\beta}) = \frac{1}{n} \sum_{i=1}^n |y_i - \boldsymbol{\beta}^T \mathbf{x}|^2 + \lambda \sum_{j=1}^J (\beta_j)^2$$

$$\hat{\boldsymbol{\beta}}^{Ridge} = \operatorname{argmin} L_{Ridge}(\boldsymbol{\beta})$$

$$\lambda \sum_{j=1}^J |\hat{\beta}_j^{Ridge}|^2 = C$$



The Geometry of Regularization



Variable Selection as Regularization



Since LASSO regression tend to produce zero estimates for a number of model parameters - we say that LASSO solutions are **sparse** - we consider LASSO to be a method for variable selection.

Many prefer using LASSO for variable selection (as well as for suppressing extreme parameter values) rather than stepwise selection, as LASSO avoids the statistic problems that arises in stepwise selection.

Question: What are the pros and cons of the two approaches?

Artificial Intelligence and Machine Learning

Classification

Lecture 2: Outline

- Linear Regression (Review)
- Intro to Classification
- Logistic Regression (Linear Classifier)
- Classification Metrics
- Other ML Models
- Optimizers

Recap

Design your model

- Input scalar linear model (line fitting)
- Fitting polynomials (synthetically designing features from a one-dimensional input)

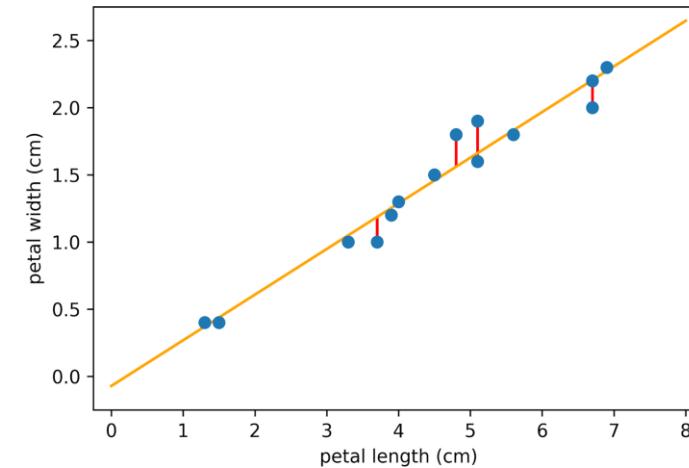
$$\{(\mathbf{x}_i, y_i)\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}$$

Design your loss function

- We used mean squared error loss throughout

Finding optimal parameter fitting

- Closed form solution to the linear least squares?
- Why is it linear leastsquares?
- Solution is closed form

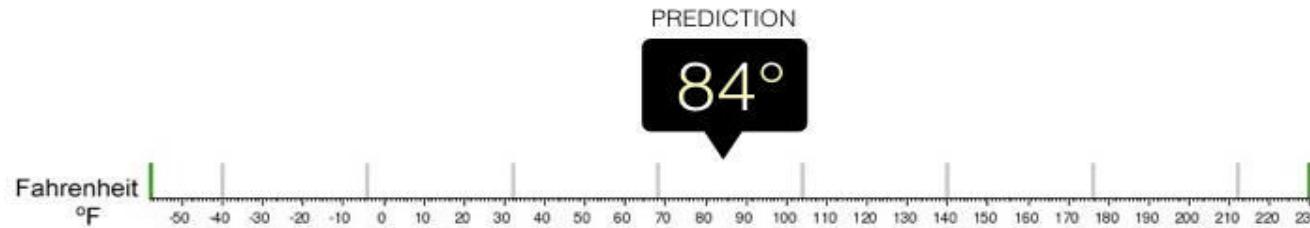


Regression VS classification



Regression

What is the temperature going to be tomorrow?

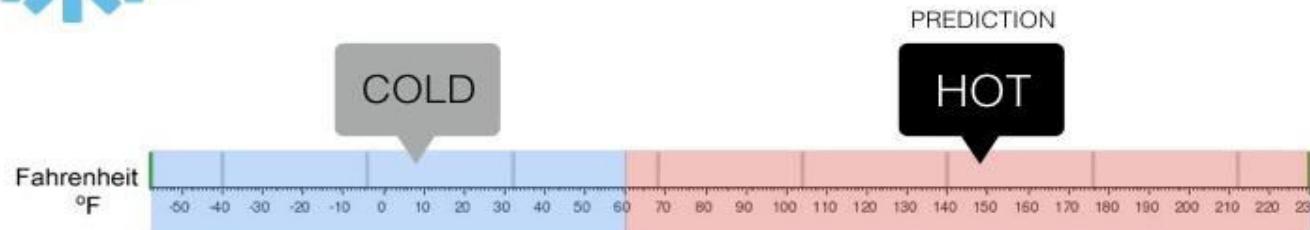


=> Continuous Values



Classification

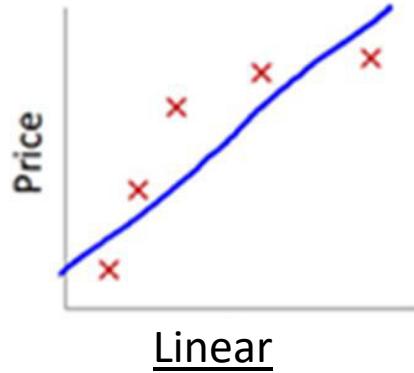
Will it be Cold or Hot tomorrow?



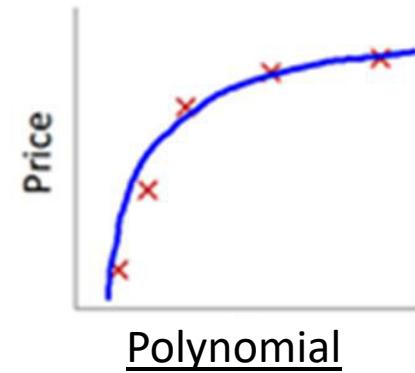
=> Discrete Values

Regression VS classification

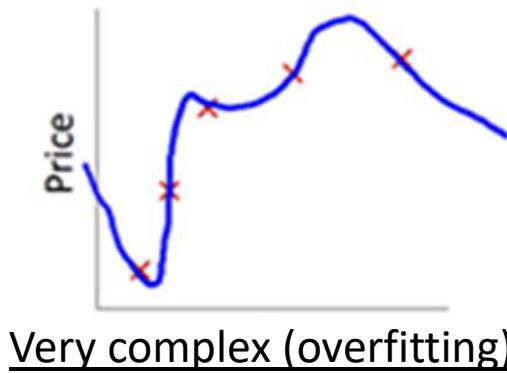
Regression:



Linear

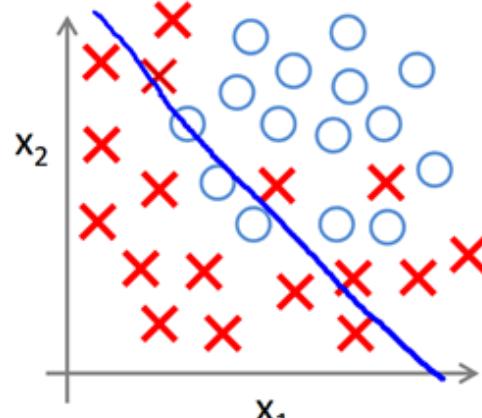


Polynomial

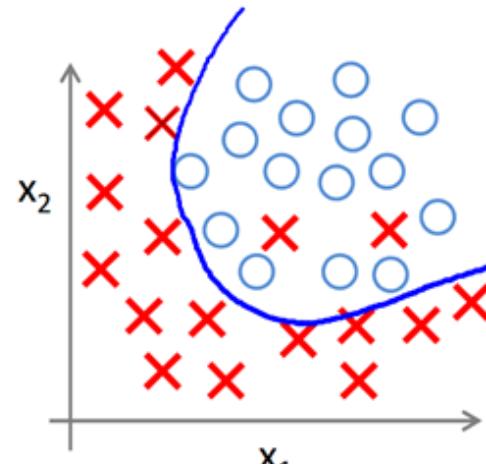


Very complex (overfitting)

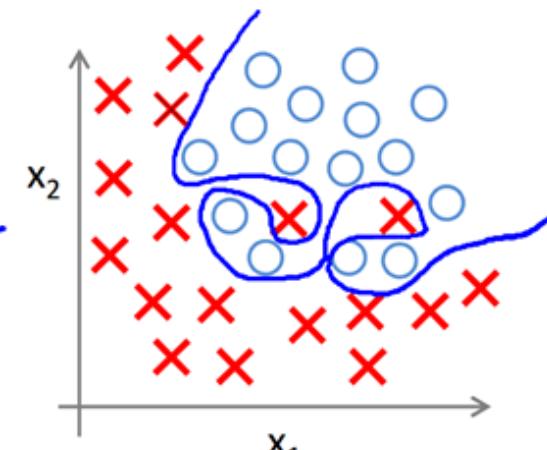
Classification:



Linear

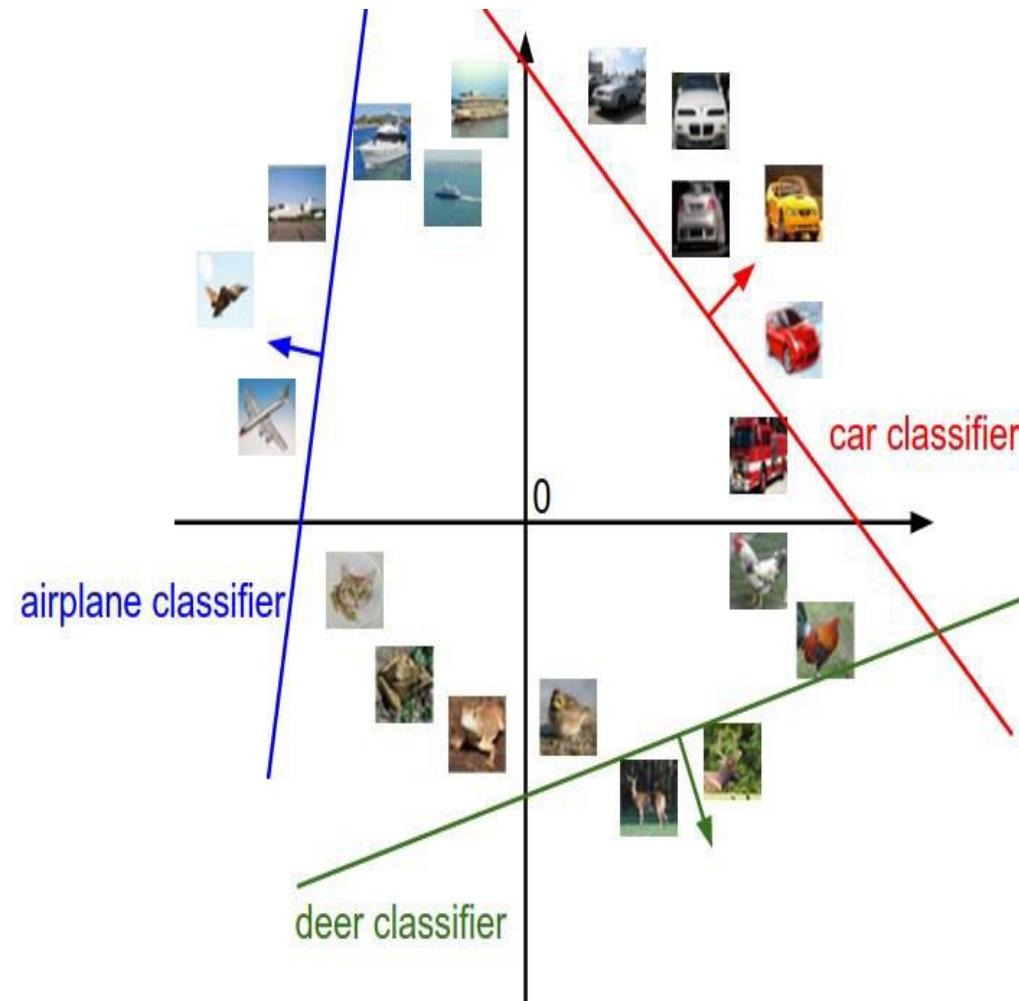


Polynomial



Very complex (overfitting)

Interpreting a Linear Classifier



$$f(x_i, W, b) = Wx_i + b$$

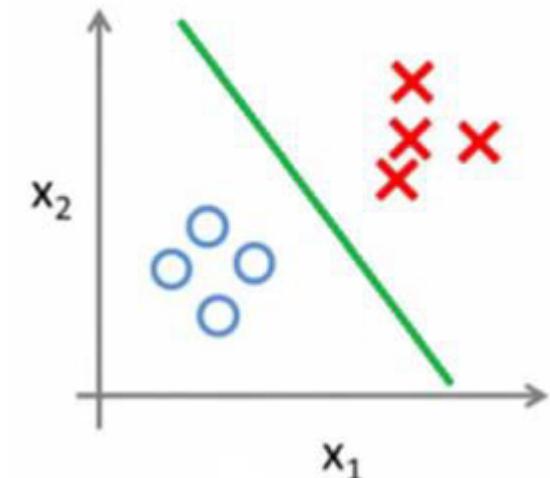


[32x32x3]
array of numbers 0...1
(3072 numbers total)

Classification Types

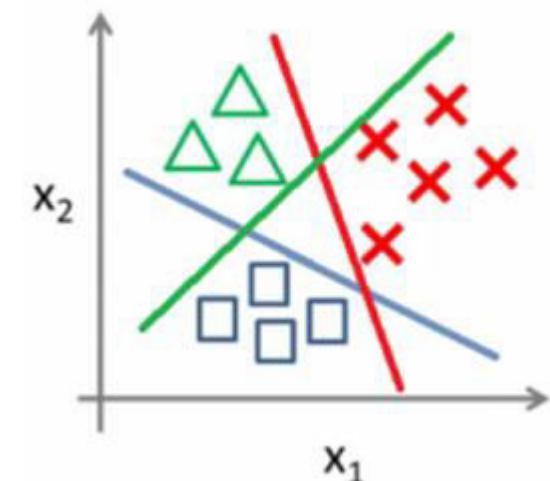
Binary Classification: Two possible outcomes (e.g., Yes/No, Spam/Not Spam).

How? Single Classifier.



Multiclass Classification: More than two outcomes (e.g., Class A, B, C).

How? Multiple *Single Classifiers*.

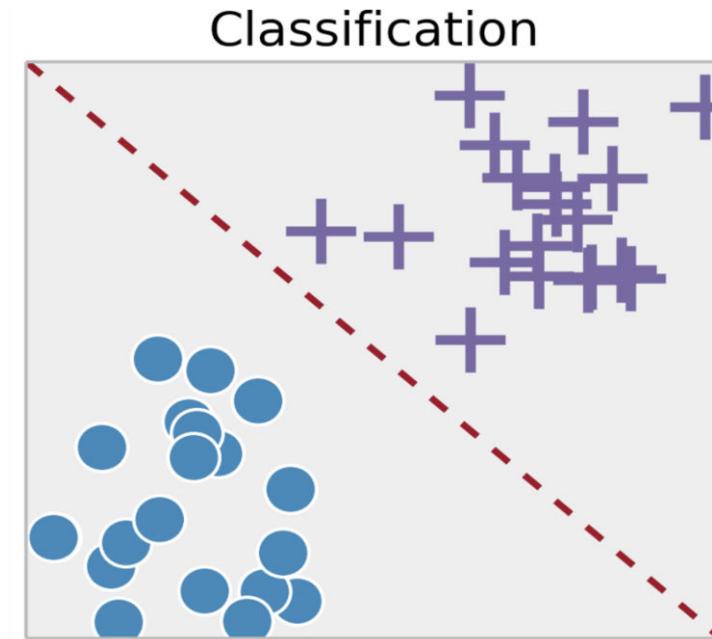


Linear Model in Disguise

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} = [w_0, w_1, \dots, w_m]^T$$

$$\mathbf{x} = [1, x^1, \dots, x^m]^T$$



$$\{(\mathbf{x}_i, y_i)\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1\}$$

Linear Model in Disguise

$$\hat{y} = \mathbf{w}^T \mathbf{x}$$

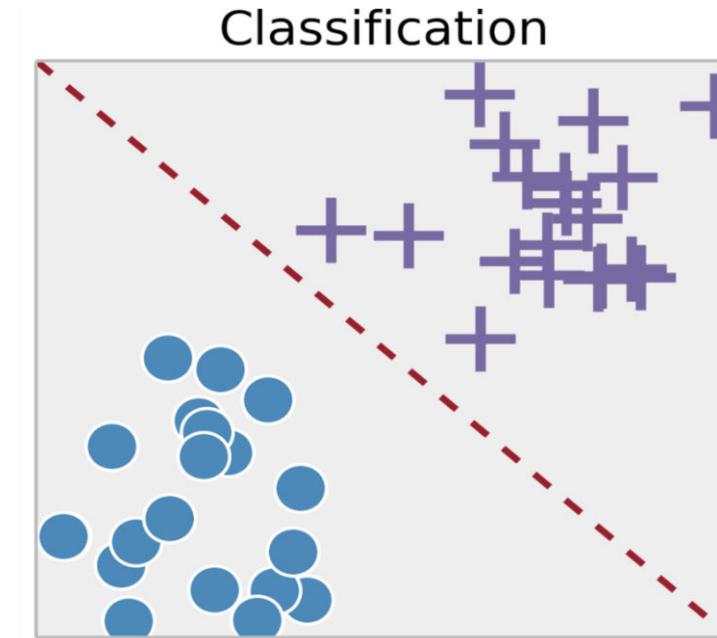
$$\mathbf{w} = [w_0, w_1, \dots, w_m]^T$$

$$\mathbf{x} = [1, x^1, \dots, x^m]^T$$

$$\hat{y} \approx y$$

Recall that the output/label y is binary

How to map the predictions to binary?



$$\{(\mathbf{x}_i, y_i)\}_{i=1}^N, \mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1\}$$

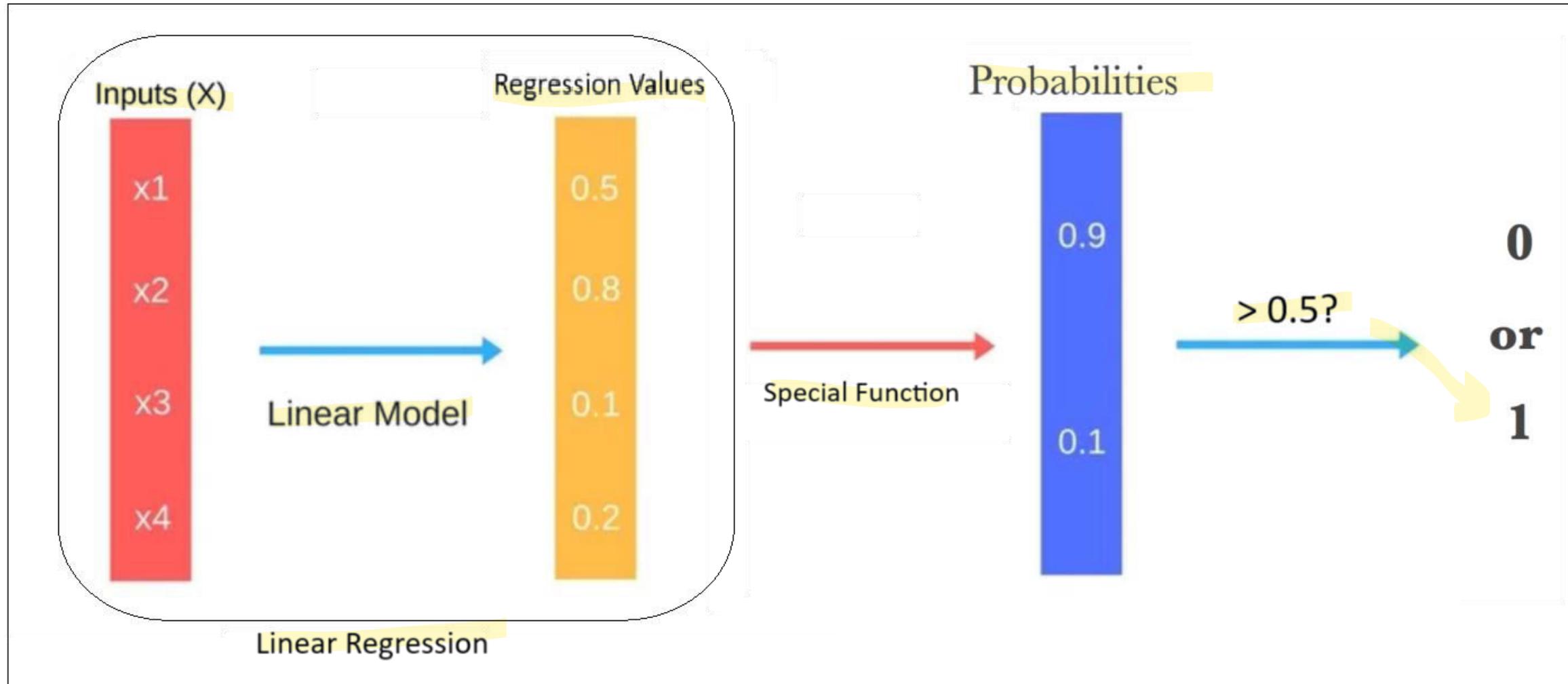
Classifiers

Classifiers are **regression** models with two key modifications:

- Pass the regression outputs through a **special function** to convert them into probabilities.
- Replace the Mean Squared Error loss with a **new loss function** designed for probabilities.

Applying these **two** to the **Linear Regression** should give **Linear Classifier** (called **Logistic Regression**).

Logistic Regression



Special Function

We are searching for a function that have the following characteristics:

1. Could convert any arbitrary input values to **[0,1] range (Probabilities)**.
2. **Smooth and Differentiable**. Because we want to find the minima later, right?

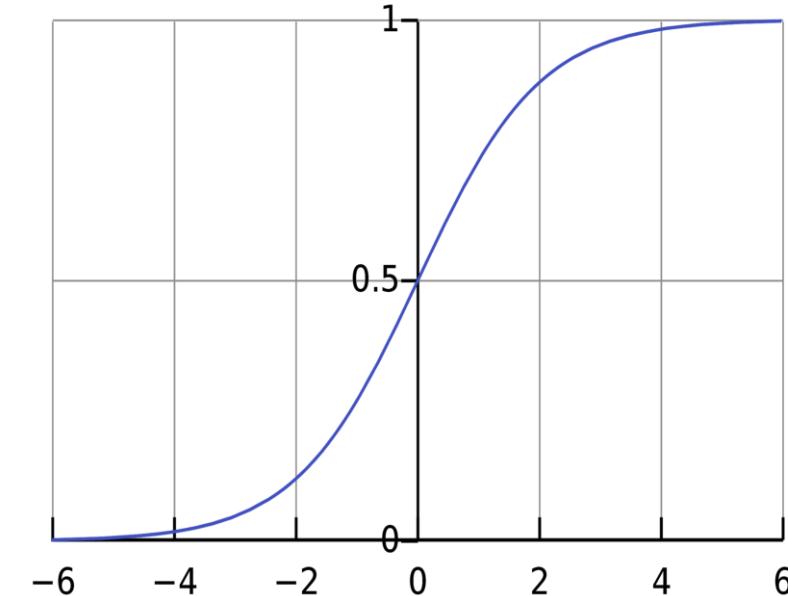
Any ideas?

Sigmoid Function (for Binary Classification)

(0, 1)

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

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$



Assuming you have negative and positive classes, its output would be the probability of the positive class.

But what if we have multiclass problem? 🤔

$$\lim_{z \rightarrow -\infty} \sigma(z) = 0$$

$$\lim_{z \rightarrow \infty} \sigma(z) = 1$$

Q

Softmax Function (for Multiclass Classification)

$$\text{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^n \exp(z_j)}$$

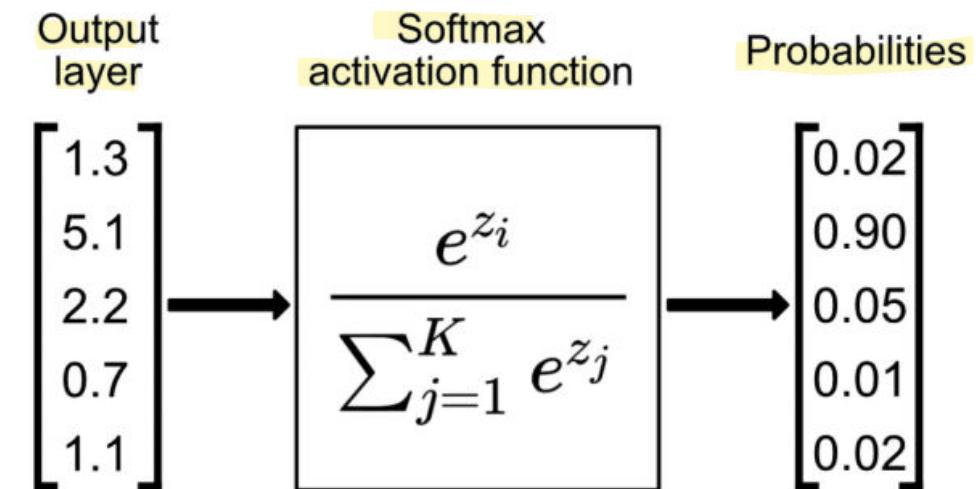
Numerator: The exponential of the i th class $\exp(z_i)$.

Denominator: The sum of the exponentials of all classes $\sum_{j=1}^n \exp(z_j)$.

E.g. Divide the cat value by the cat, dog and deer values to get the cat probability.

Assuming you have K classes, its output would be the probability of the i th class. So, you will run it K times to get the probability of each class.

Note: Sigmoid is a special case of Softmax. ✎
Optional: Can you prove it?



Classification Loss

We are searching for a loss that have the following characteristics:

1. **Probabilistic behaviour**: Should work with probabilities, not raw numbers.
2. **Sensitivity**: It should heavily penalize incorrect confident predictions (e.g., predicting 0.9 when the true label is 0).
3. **Differentiable**: Must be smooth and differentiable to enable optimization.

Any ideas?

Classification Loss

We are searching for a loss that have the following characteristics:

1. **Probabilistic behaviour:** Should work with probabilities, not raw numbers.
2. **Sensitivity:** It should heavily penalize incorrect confident predictions (e.g., predicting 0.9 when the true label is 0).
3. **Differentiable:** Must be smooth and differentiable to enable optimization.

Any ideas?

Logarithm is all you need :)

Binary Cross Entropy (LogLoss)

- For one sample, this can be represented mathematically by:

$$\text{loss}(y, p) = \begin{cases} -\log(p), & \text{if } y = 1 \\ -\log(1 - p), & \text{if } y = 0 \end{cases}$$

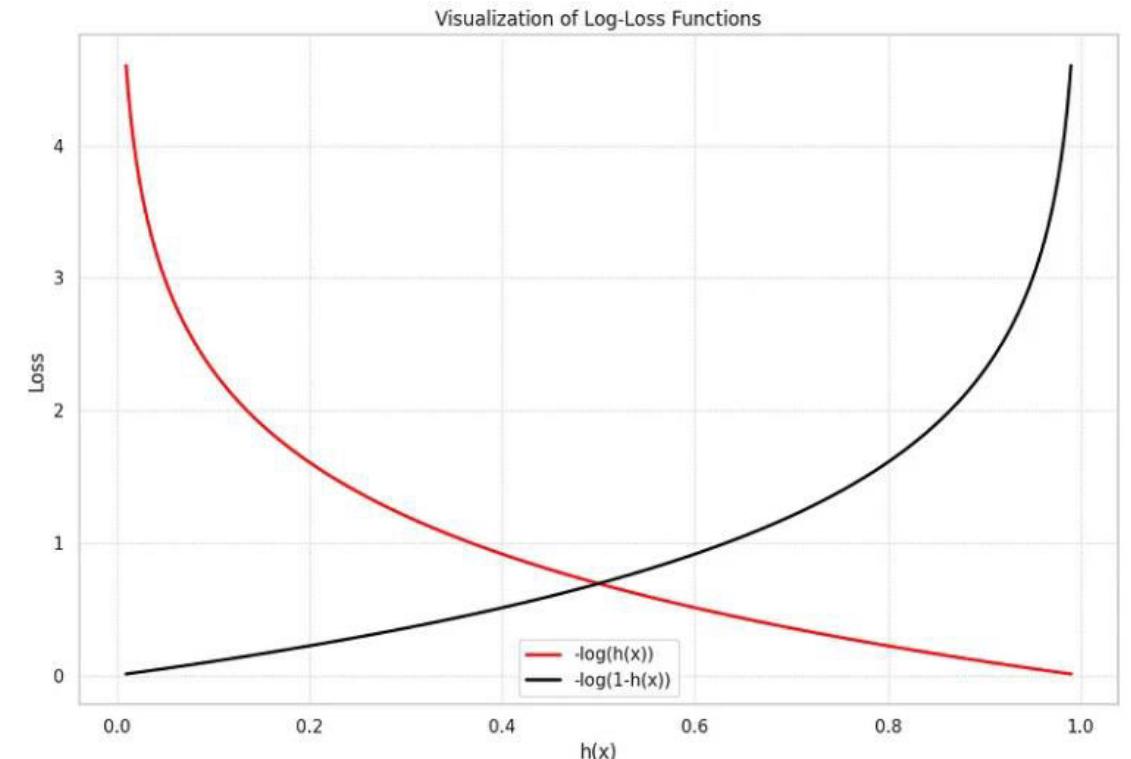
If $y = 1$ and prediction is close to 1, loss will be close to zero. (Great!)

If $y = 1$ and prediction is close to 0, loss will be close to infinity. (Very bad!)

Same behaviour for $y = 0$

If $y = 0$ and prediction is close to 1, loss will be close to zero. (Great!)

If $y = 0$ and prediction is close to 0, loss will be close to infinity. (Very bad!)



Binary Cross Entropy (LogLoss)

- For one sample, this can be represented mathematically by:

$$\text{loss}(y, p) = \begin{cases} -\log(p), & \text{if } y = 1 \\ -\log(1 - p), & \text{if } y = 0 \end{cases}$$

- Let's rewrite it in one line:

$$\text{loss}(y, p) = -(y * \log(p) + (1 - y) * \log(1 - p))$$

But we have many samples, not only one, right?

Binary Cross Entropy (LogLoss)

- For one sample, this can be represented mathematically by:

$$\text{loss}(y, p) = \begin{cases} -\log(p), & \text{if } y = 1 \\ -\log(1 - p), & \text{if } y = 0 \end{cases}$$

- Let's write it in one line:

$$\text{loss}(y, p) = -(y * \log(p) + (1 - y) * \log(1 - p))$$

- Sum and average:

$$\text{logloss}(Y, P) = -\frac{1}{N} \sum_{i=1}^N (y_i * \log(p_i) + (1 - y_i) * \log(1 - p_i))$$

Final derivative $\frac{d}{dx}$

How to find optimal Parameters?

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

Just like before, simply take

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$$

However, this does not have a nice closed solution
Just like the MSE case

How to find optimal Parameters?

We Gradient Descent

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

Just like before, simply take

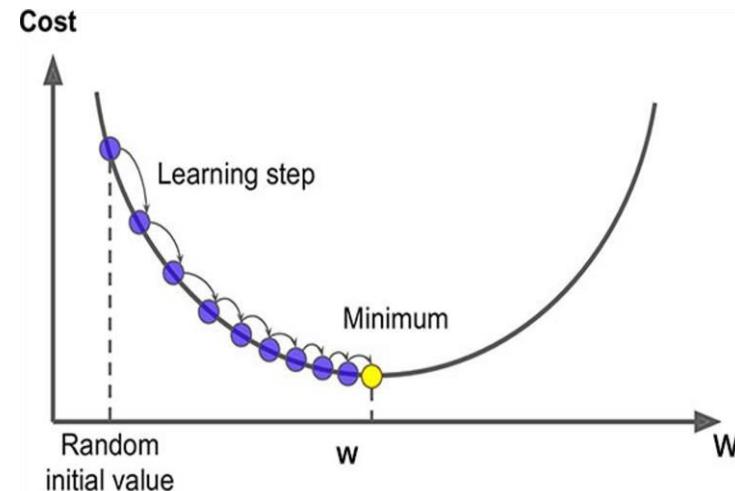
$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$$

However, this does not have a nice closed solution

Just like the MSE case

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

To Min Learning rate



Instead, use gradient descent (optimizer)!

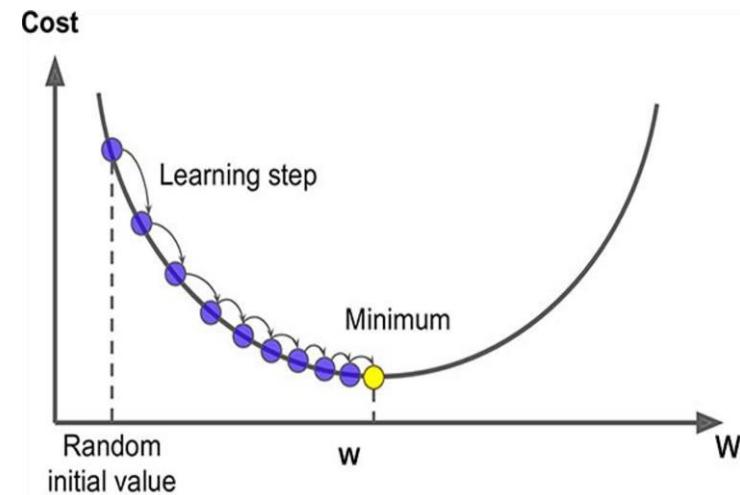
Logistic Regression

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

Learning rate

- We have a linear model for prediction
- For classification, we want to output a probability
- We map the prediction to probabilities with a sigmoid function
- We have a loss function (BCE) to compare models



Logistic Regression

Linear Regression	Logistic Regression
For Regression	For Classification
We predict the target value for any input value	We predict the probability that the input value belongs to the specific target
Target: Real Values	Target: Discrete values
Graph: Straight Line	Graph: S-curve

Classification Metrics

- **Accuracy:**

Accuracy measures the proportion of correctly classified samples out of the total number of samples.

$$\text{Accuracy} = \frac{\text{Correct Samples}}{\text{All Samples}}$$

Question: Why not optimizing for the accuracy directly instead of using LogLoss?

regression
 $\text{loss} = \text{matrix}$
 classification
 $\text{matrix} = \text{accuracy}$

Classification Metrics

- **Accuracy:**

Accuracy measures the proportion of correctly classified samples out of the total number of samples.

$$\text{Accuracy} = \frac{\text{Correct Samples}}{\text{All Sample}}$$

Could also be written as:

$$\text{Accuracy} = \frac{\underset{1}{\text{TP}} + \underset{0}{\text{TN}}}{\text{TP} + \text{TN} + \underset{\text{FP}}{\text{FP}} + \underset{\text{FN}}{\text{FN}}}$$

T = True
 F = False
 P = Positive
 N = Negative

Question: Why not optimizing for the accuracy directly instead of using LogLoss?

Non-differentiable

Classification Metrics

But accuracy isn't always enough!

Imagine an imbalanced dataset with 95% negative labels (e.g., "not spam").
A model predicting "negative" all the time will be 95% accurate but useless.

Classification Metrics

But accuracy isn't always enough!

Imagine an imbalanced dataset with 95% negative labels (e.g., "not spam").
A model predicting "negative" all the time will be 95% accurate but useless.

Different goals require different metrics

- Do you care more about **catching all positives** (e.g., cancer detection)?
- Or avoiding **false alarms** (e.g., fraud detection)?

Classification Metrics

- **Precision:**

Among all the positive predictions the model made,
how many are actually correct?

important when **minimizing false alarms** is critical,
like in fraud detection or spam filtering.

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$

Classification Metrics

- **Recall:**

Among all the actual positives, how many did the model correctly identify?

important when **catching all positives** is vital, such as in **cancer detection**.

$$Precision = \frac{True\ Positives\ (TP)}{True\ Positives\ (TP) + False\ Positives\ (FN)}$$

$$\frac{TP}{TP + FP}$$

Classification Metrics

- **F1 Score:**

What if both precision and recall are important?

Take their (harmonic) mean.

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

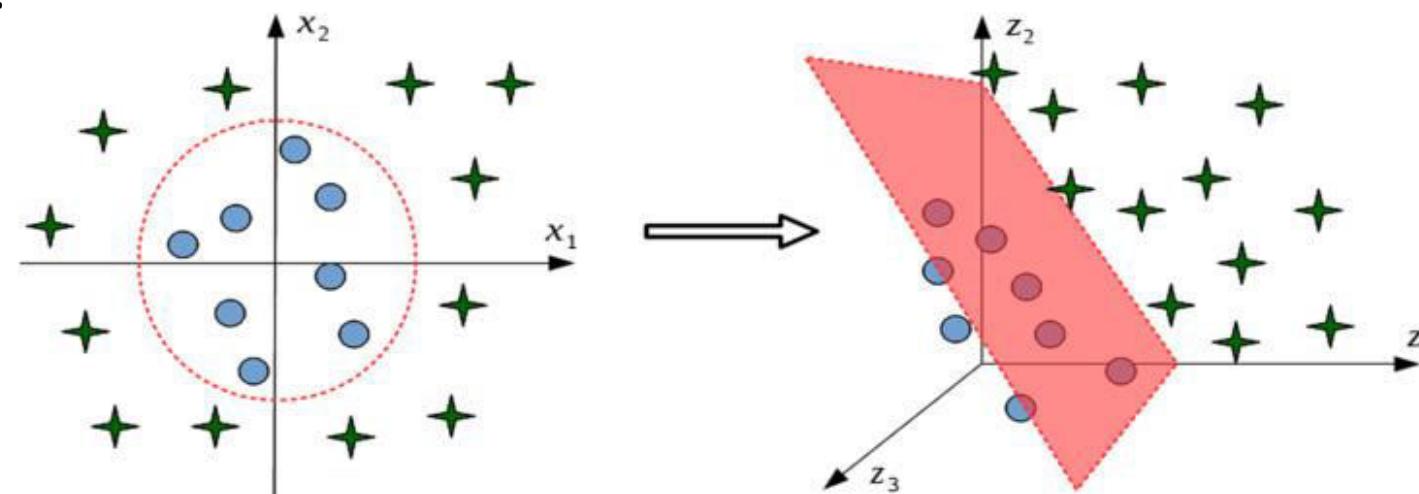
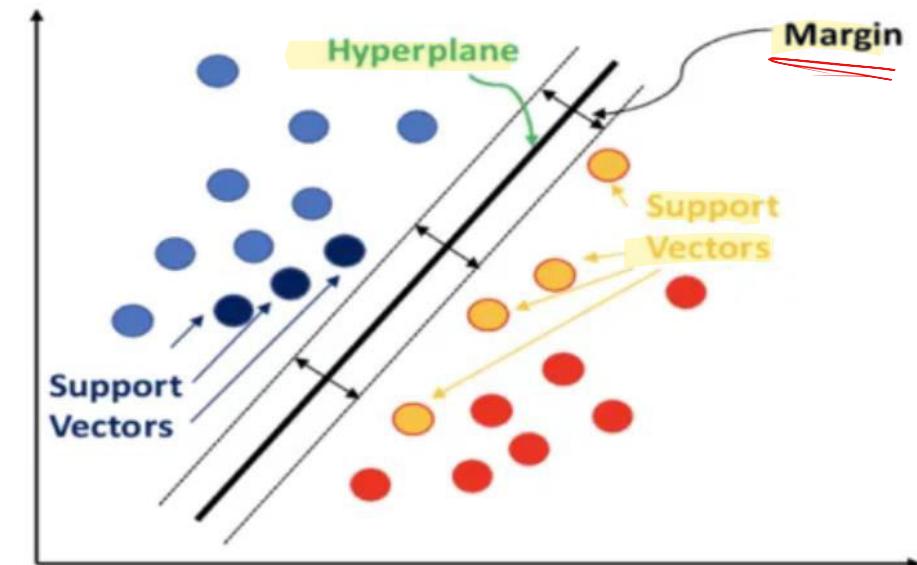


Other ML Models

- **Support Vector Machine (SVM):**

Key Idea:

- SVM finds the best boundary (hyperplane) that separates classes by maximizing the margin between them.
- Works well with high-dimensional data and small datasets.
- Can be computationally expensive for large datasets.

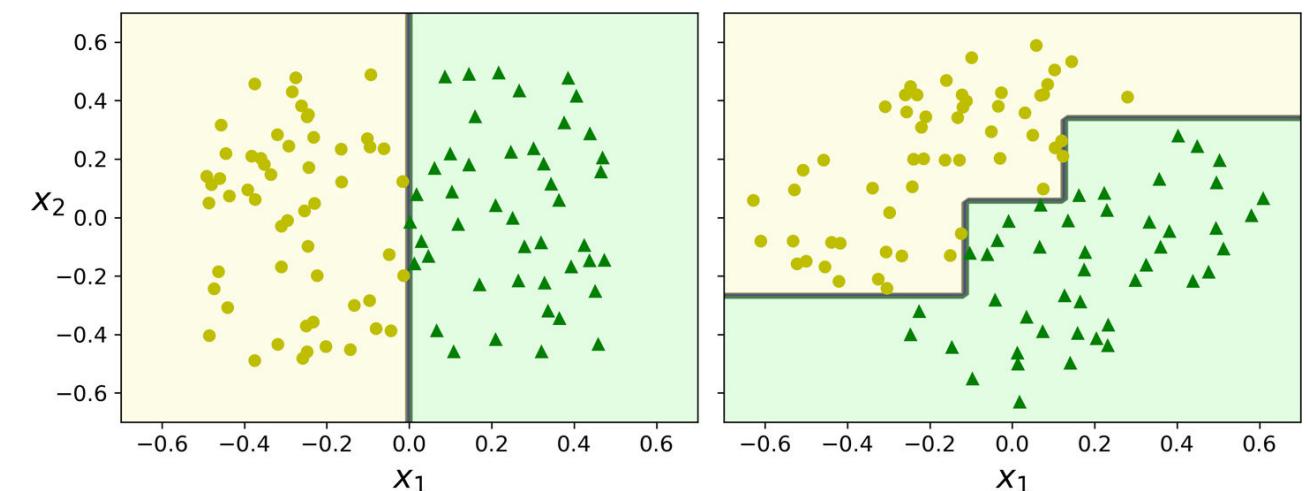
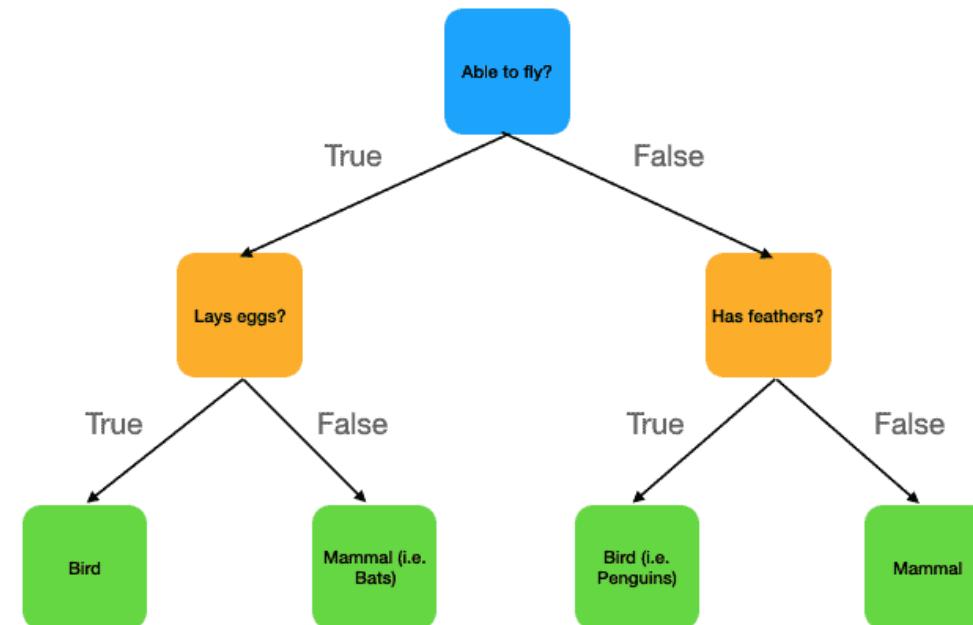


Other ML Models

- **Decision Trees:**

Key Idea:

- A tree-based model splits data into subsets based on feature thresholds, creating a flowchart-like structure for decision-making.
- Overfits very easily.
- could work bad if relation between features and the target is linear. (Check the right figure)



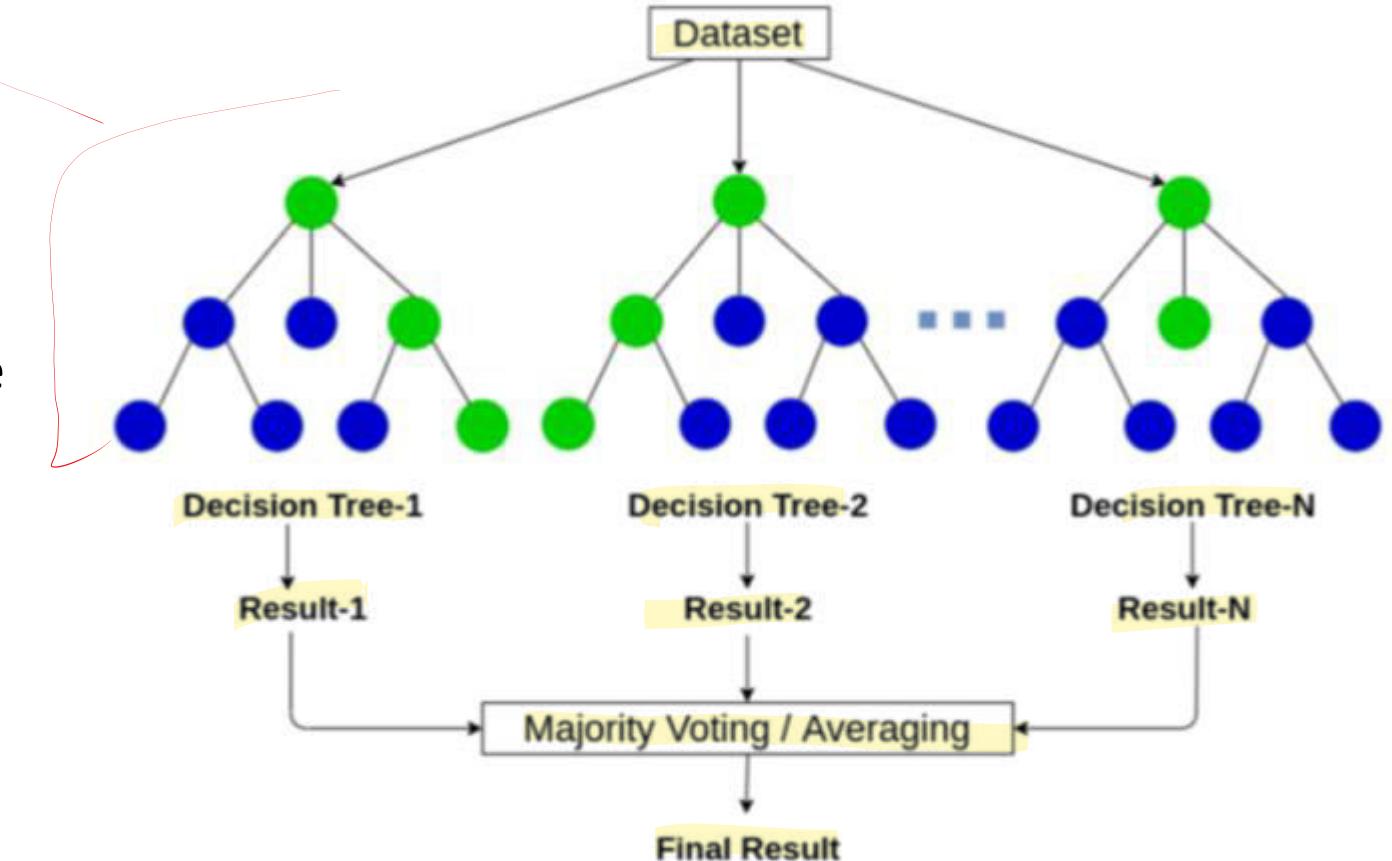
Other ML Models

- **Random Forest:**

Instead of training one tree, let's train a forest :)

Key Idea:

- Training K decision trees independently, where each one is fed with a different subset of samples and features.
- Then average the results (Regression) or Take the mode (Classification).
- Powerful model.

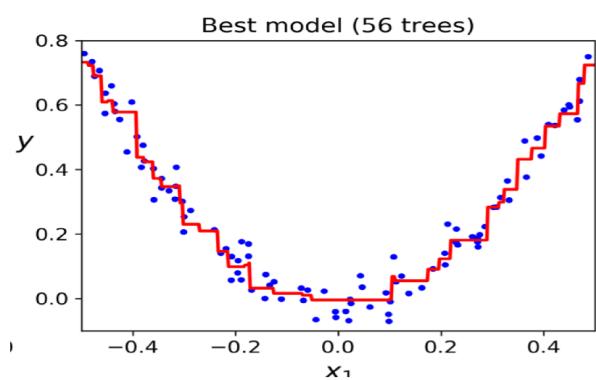
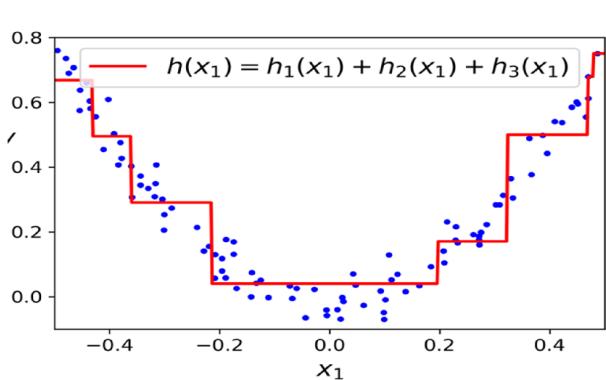
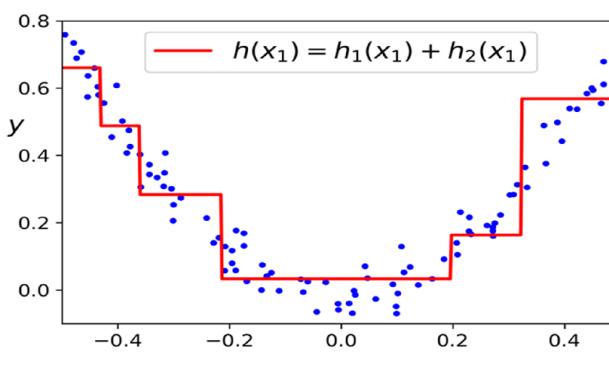
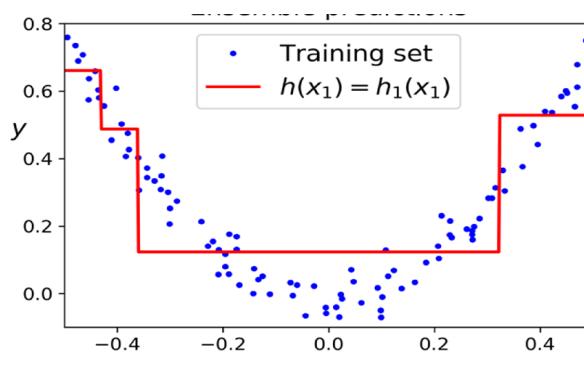
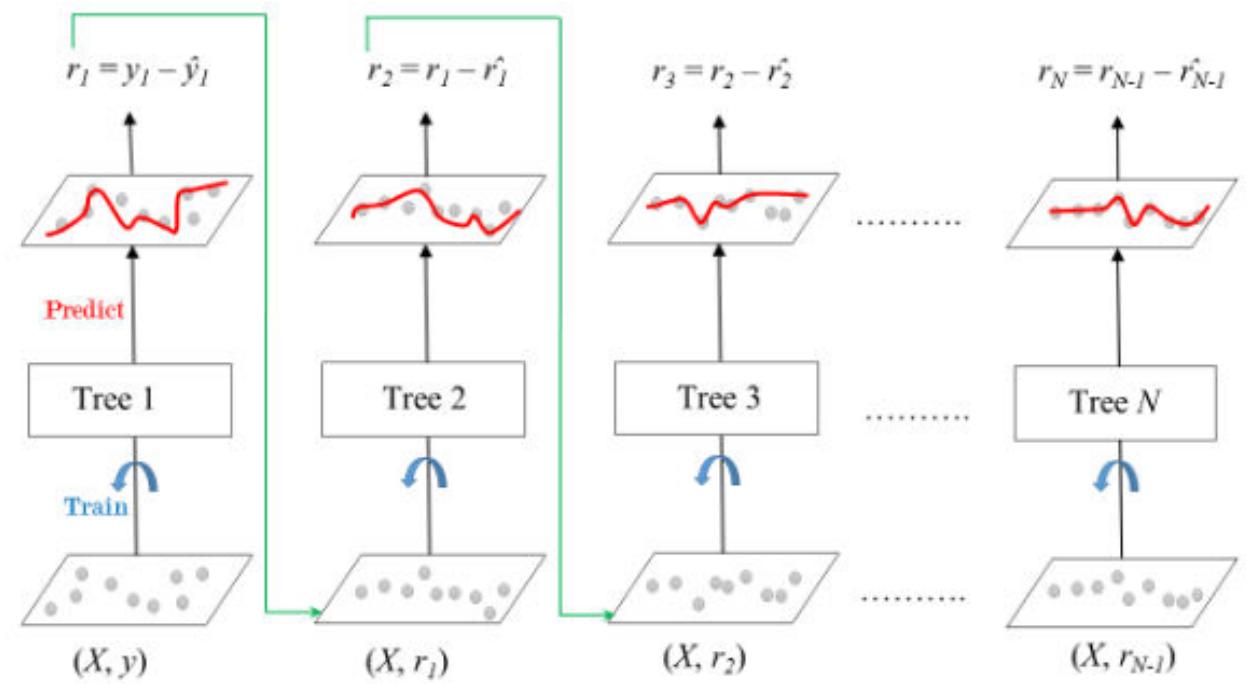


Other ML Models

- **Gradient Boosting:**

Key Idea:

- Training K decision trees independently, where each tree corrects the errors of the previous one.
- Many versions: XGBoost, LightGBM, CatBoost,...etc
- Top ML models.



A Slight Detour: A Look at Optimization Tools

in Deep Learning to arrive to the Min

Introduction

- **Optimizers** are algorithms that adjust the weights of the model to minimize the loss function during training.
- They are one of the main components of Deep Learning models.
- We will go through several types of optimizers in this section.

Direction of maximum increase and decrease for a function

- Gradient direction is the direction of maximum increase for a function
- Negative gradient is the direction of maximum decrease for a function

Gradient Descent

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Learning
rate \times gradient

Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

Mini-batch Gradient Descent

- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

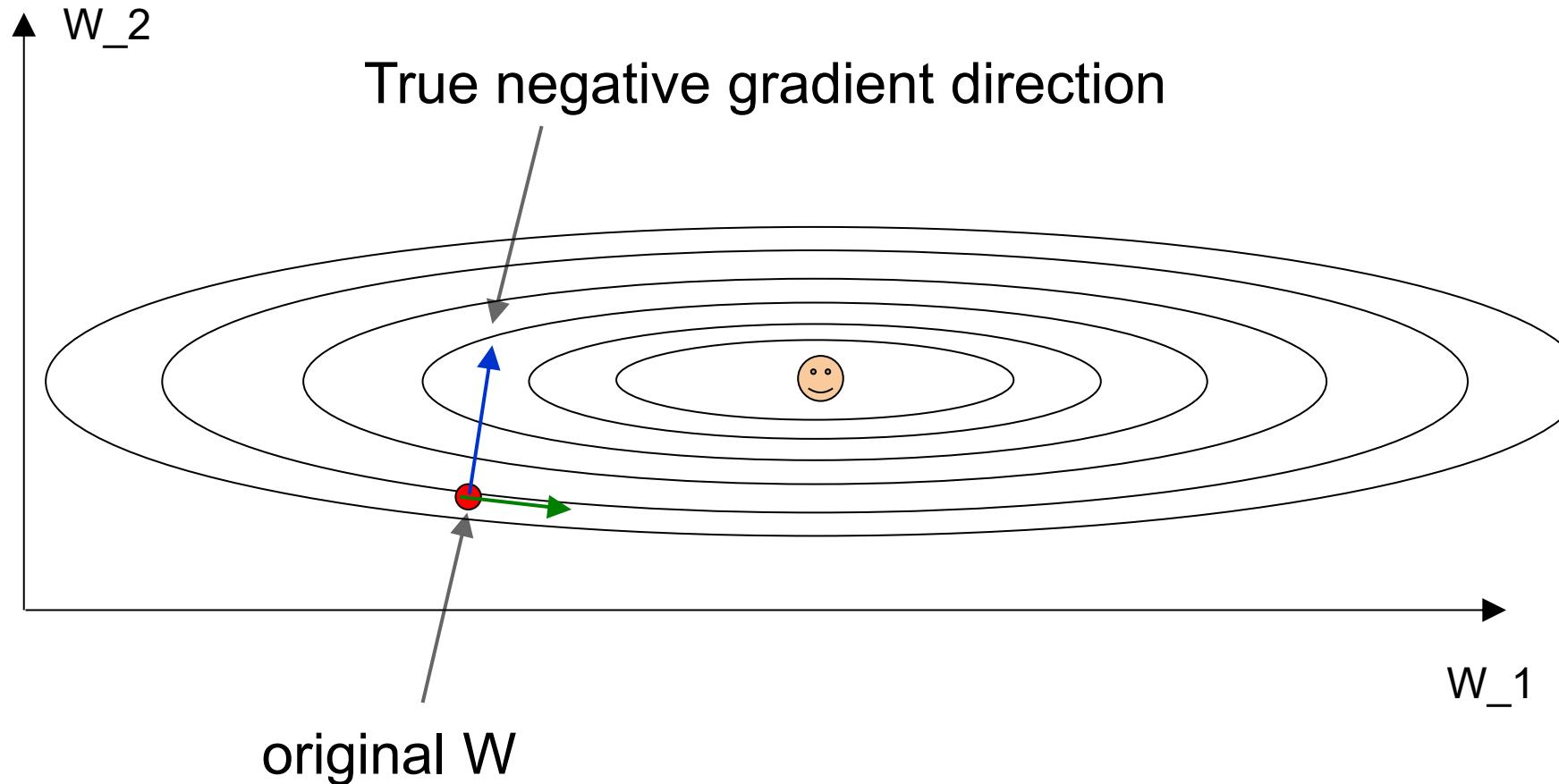
while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

batch size

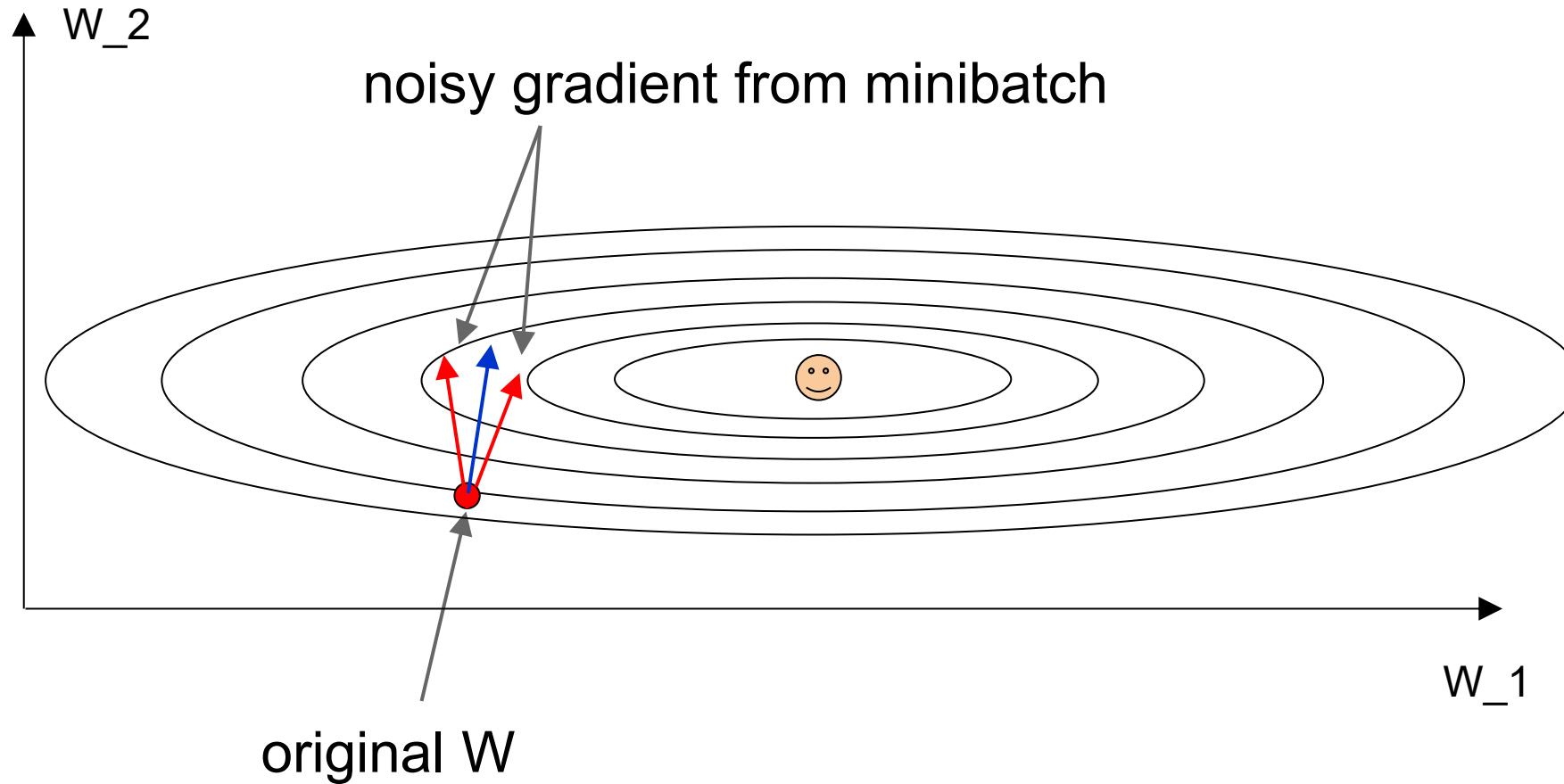
Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

we will look at more
fancy update formulas
(momentum, Adagrad,
RMSProp, Adam, ...)

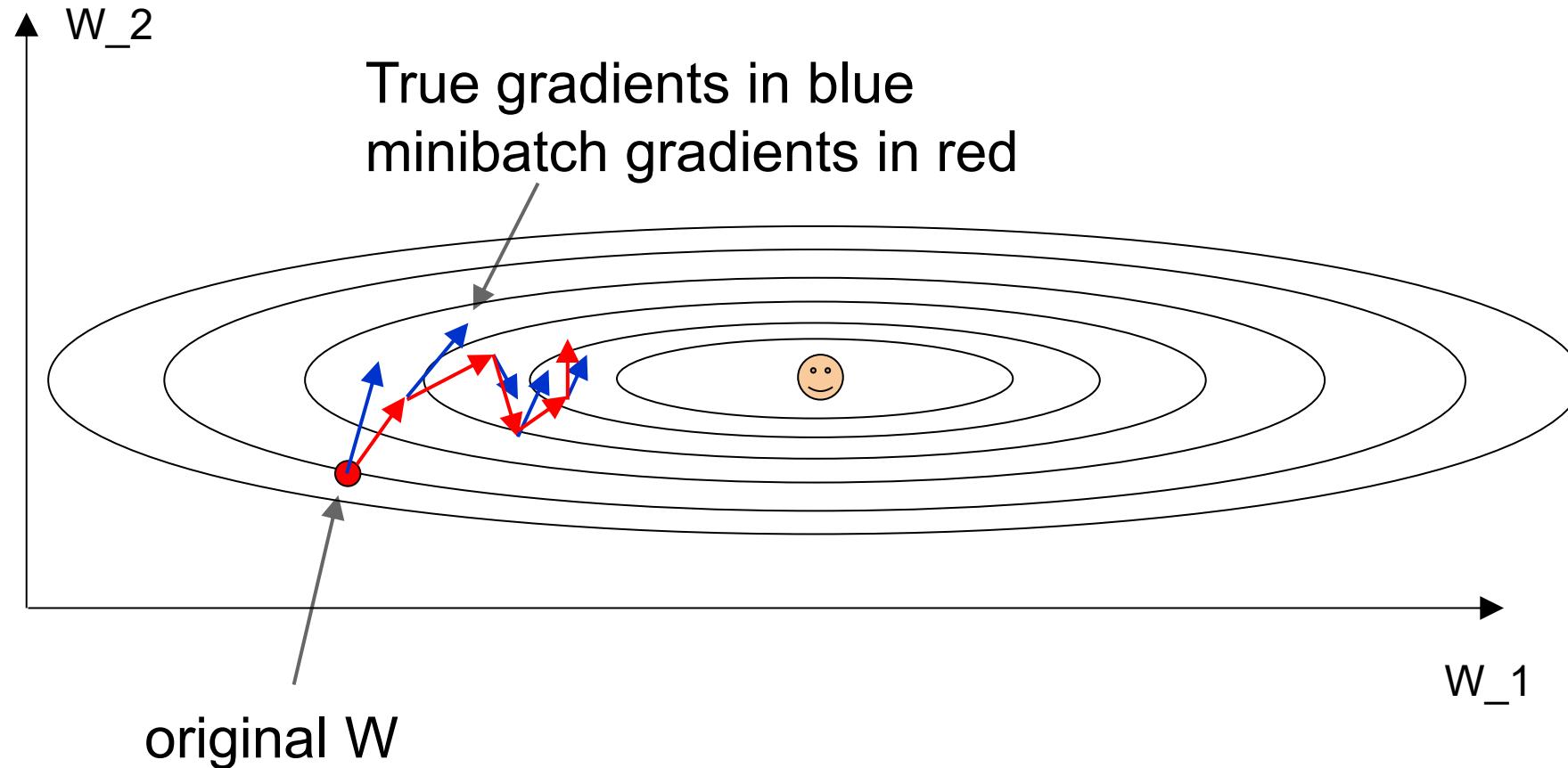
Many Sample
Minibatch updates



Stochastic Gradient

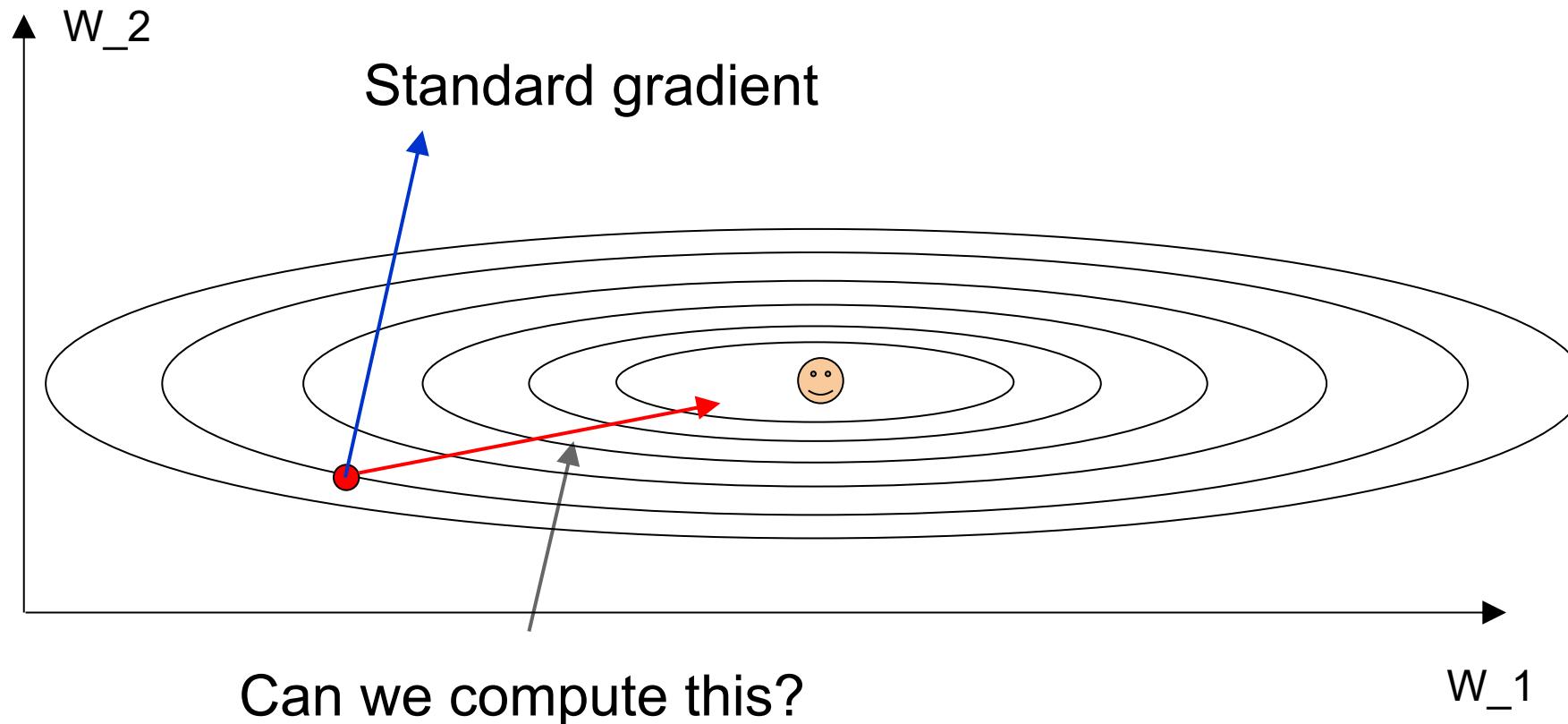


Stochastic Gradient Descent

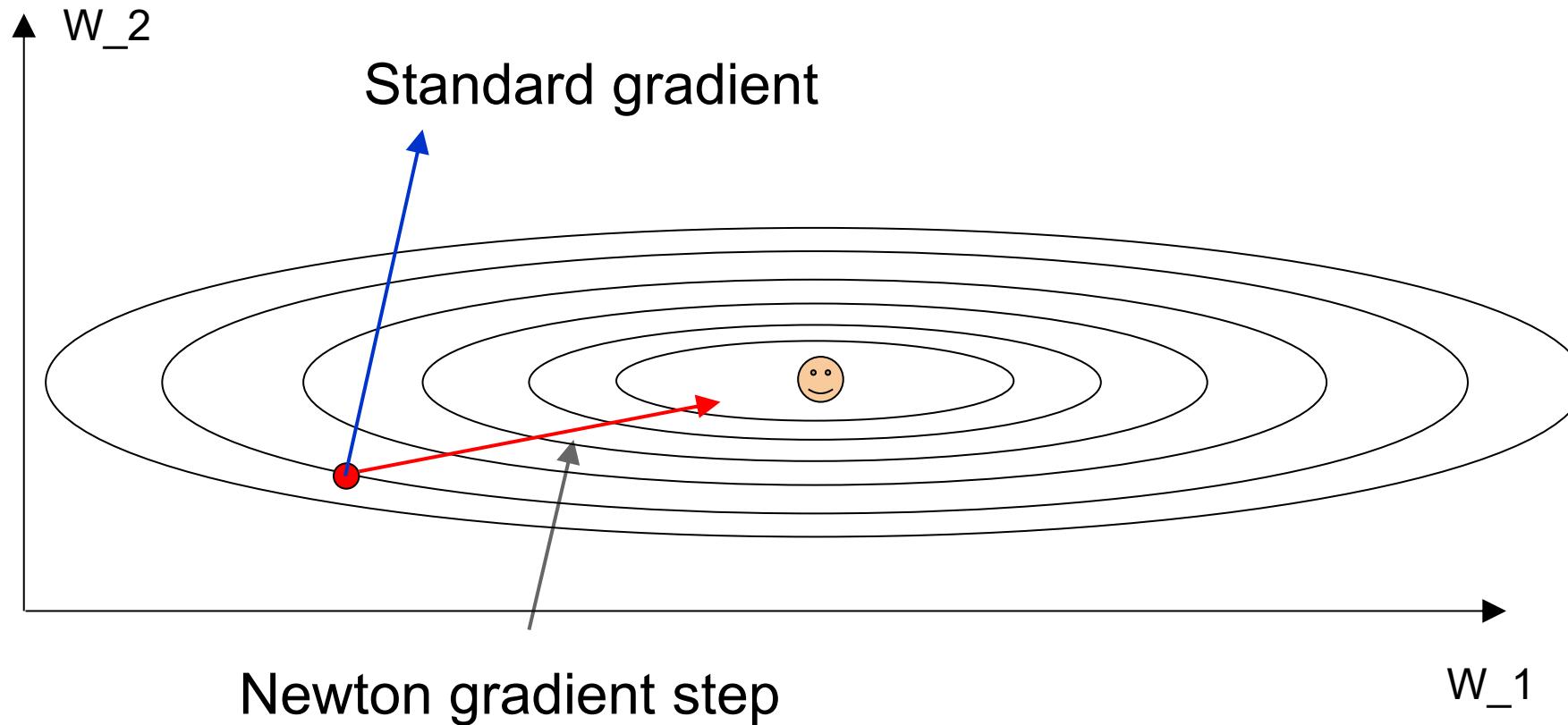


Gradients are noisy but still make good progress on average

You might be wondering...



Newton step



Newton's method for gradients:



The Newton update is:

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

Where H_f is the Hessian matrix of second derivatives of f .

Converges very fast, but rarely used in DL. Why?

Newton's method for gradients:



The Newton update is:

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

Converges very fast, but rarely used in DL. Why?

Too expensive: if x_n has dimension M, the Hessian $H_f(x_n)$ has dimension M^2 and takes $O(M^3)$ time to invert.

Newton's method for gradients:



The Newton update is:

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

Converges very fast, but rarely used in DL. Why?

Too expensive: if x_n has dimension M, the Hessian $H_f(x_n)$ has dimension M^2 and takes $O(M^3)$ time to invert.

Too unstable: it involves a high-dimensional matrix inverse, which has poor numerical stability. The Hessian may even be singular.

Momentum update

```
# Gradient descent update  
x += - learning_rate * dx
```

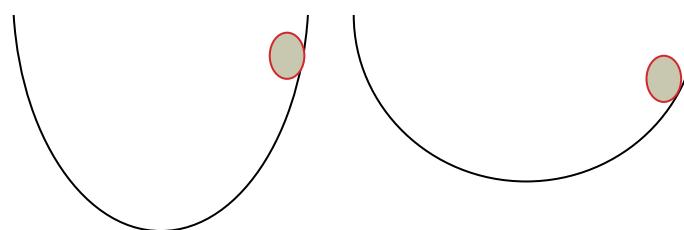


```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

- Physical interpretation as ball rolling down the loss function + friction (μ coefficient).
- μ = usually ~0.5, 0.9, or 0.99 (Sometimes annealed over time, e.g. from 0.5 -> 0.99)

Momentum update

```
# Gradient descent update  
x += - learning_rate * dx
```



```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

- Allows a velocity to “build up” along shallow directions
- Velocity becomes damped in steep direction due to quickly changing sign

AdaGrad update

improve for
previos method

[Duchi et al., 2011]



```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

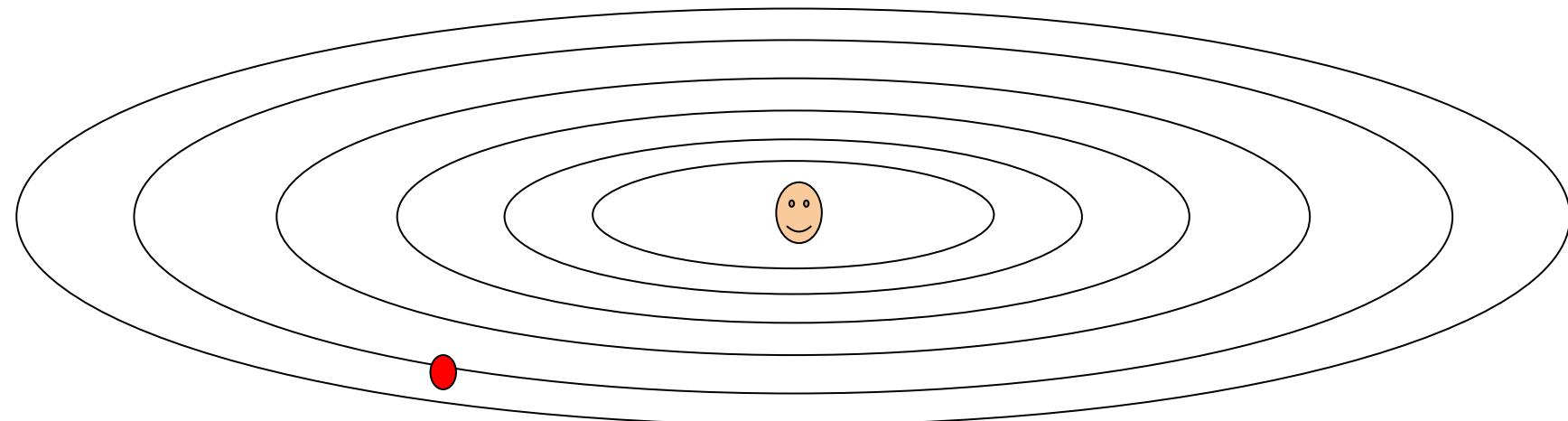
→ Scale Learning rate

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

to descrease the Lrate

AdaGrad update

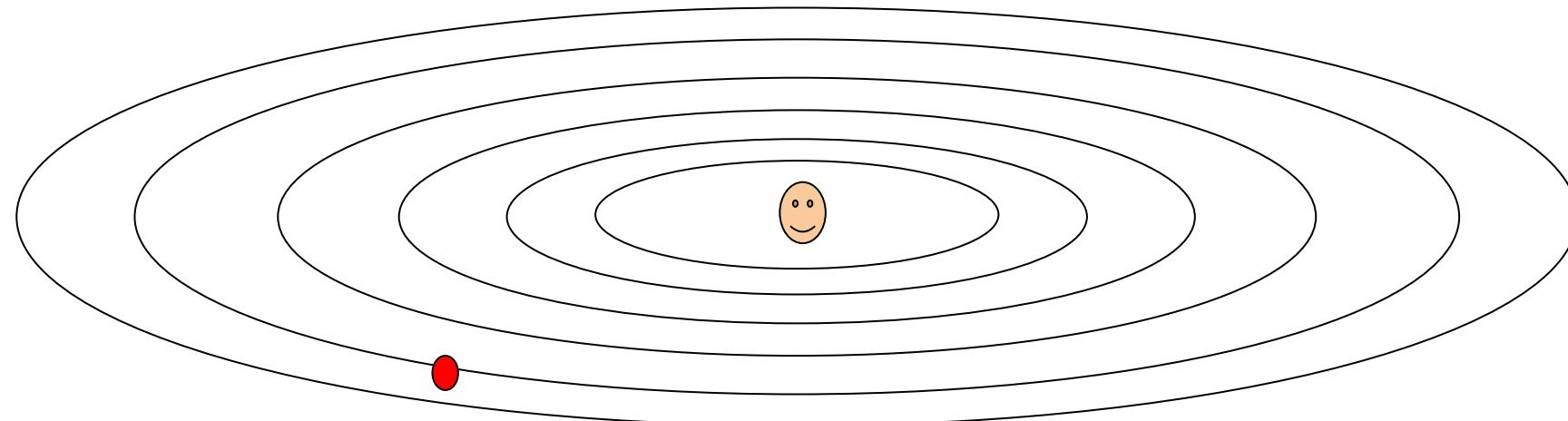
```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



Q: What happens with AdaGrad?

AdaGrad update

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



Q2: What happens to the step size over long time?

RMSProp update

[Tieleman and Hinton, 2012]



```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

rmsprop: A mini-batch version of rprop

- rprop is equivalent to using the gradient but also dividing by the size of the gradient.
 - The problem with mini-batch rprop is that we divide by a different number for each mini-batch. So why not force the number we divide by to be very similar for adjacent mini-batches?
- rmsprop: Keep a moving average of the squared gradient for each weight
$$MeanSquare(w, t) = 0.9 \, MeanSquare(w, t-1) + 0.1 \left(\frac{\partial E}{\partial w}(t) \right)^2$$
- Dividing the gradient by $\sqrt{MeanSquare(w, t)}$ makes the learning work much better (Tijmen Tieleman, unpublished).

Introduced in a slide in
Geoff Hinton's Coursera
class, lecture 6

rmsprop: A mini-batch version of rprop

- rprop is equivalent to using the gradient but also dividing by the size of the gradient.
 - The problem with mini-batch rprop is that we divide by a different number for each mini-batch. So why not force the number we divide by to be very similar for adjacent mini-batches?
- rmsprop: Keep a moving average of the squared gradient for each weight
$$\text{MeanSquare}(w, t) = 0.9 \text{ MeanSquare}(w, t-1) + 0.1 \left(\frac{\partial E}{\partial w}(t) \right)^2$$
- Dividing the gradient by $\sqrt{\text{MeanSquare}(w, t)}$ makes the learning work much better (Tijmen Tieleman, unpublished).

Cited by several papers as:

[52] T. Tieleman and G. E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude., 2012.

Introduced in a slide in Geoff Hinton's Coursera class, lecture 6

Adam update

[Kingma and Ba, 2014]



(incomplete, but close)

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

Adam update

[Kingma and Ba, 2014]



(incomplete, but close)

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

momentum

RMSProp-like

Looks a bit like RMSProp with momentum

Adam update

[Kingma and Ba, 2014]



(incomplete, but close)

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

momentum

RMSProp-like

Looks a bit like RMSProp with momentum

```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

Adam update

[Kingma and Ba, 2014]



```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
    dx = # ... evaluate gradient
    m = beta1*m + (1-beta1)*dx # update first moment
    v = beta2*v + (1-beta2)*(dx**2) # update second moment
    mb = m/(1-beta1**t) # correct bias
    vb = v/(1-beta2**t) # correct bias
    x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

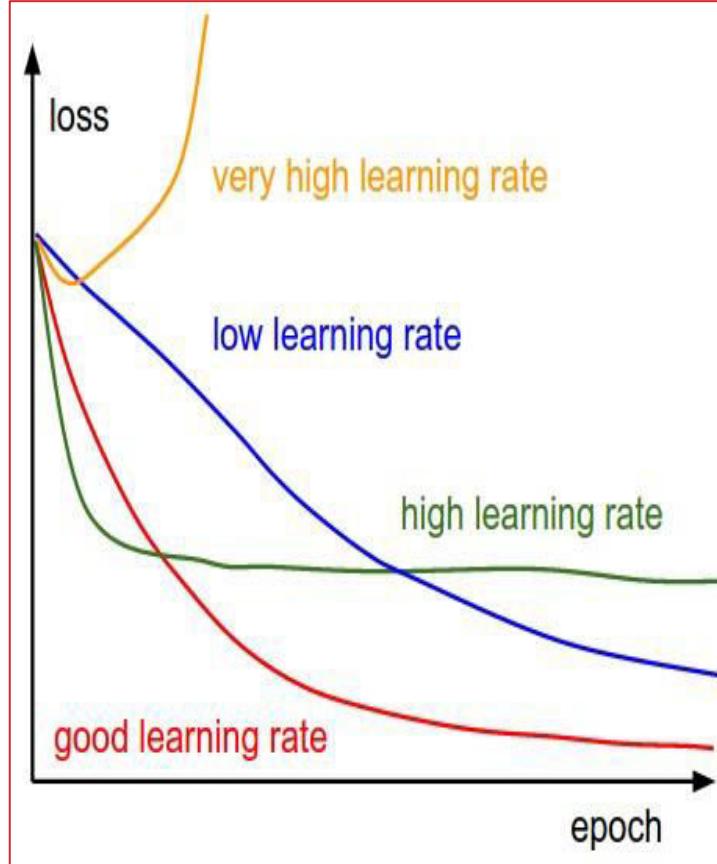
momentum skip local min

bias correction Head Start

RMSProp-like Scaning & Learning
rate, parameter vs Δt or N_t

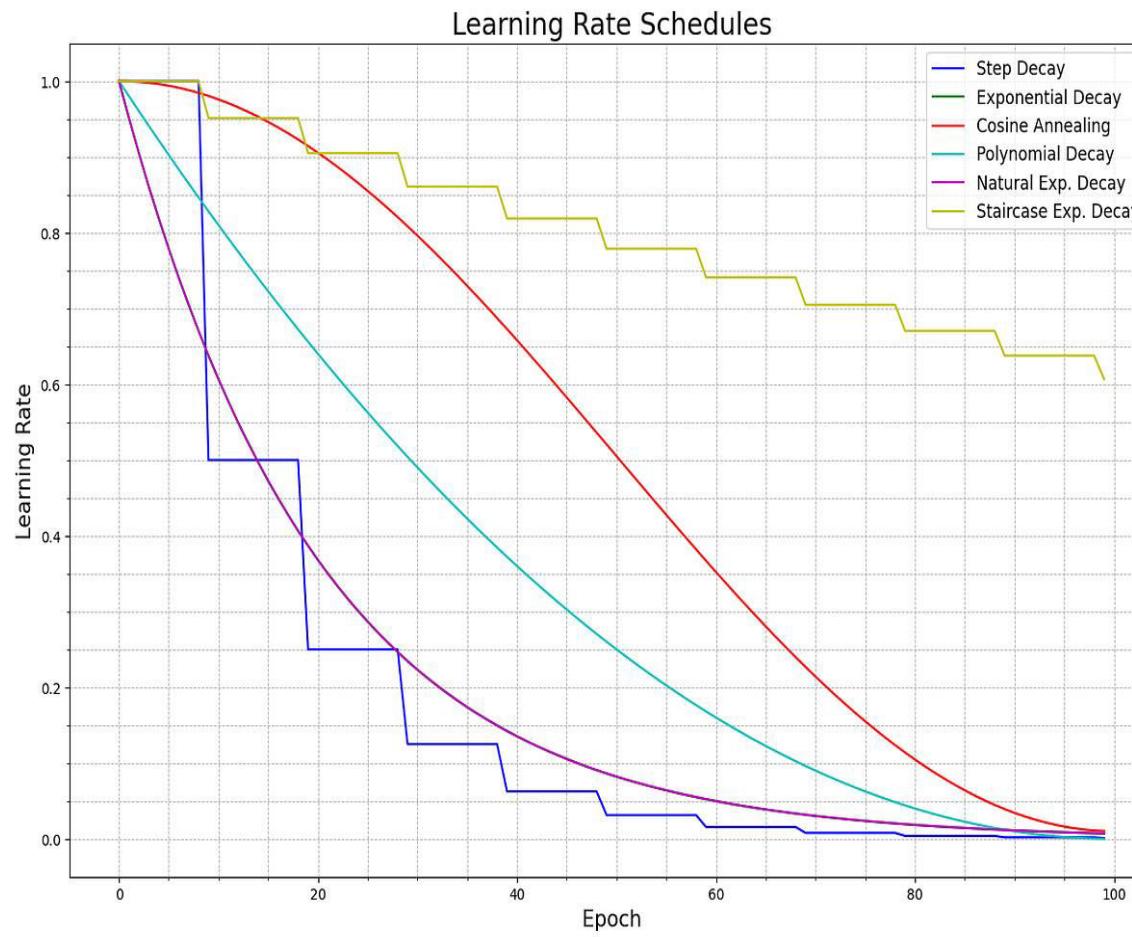
The bias correction compensates for the fact that m, v are initialized at zero and need some time to “warm up”.

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter.



Q: Which one of these learning rates is best to use?

SGD, SGD+Momentum, Adagrad, RMSProp, Adam all have learning rate as a hyperparameter.



⇒ Learning rate decay over time (LR Schedulers)!

Step decay:

e.g. decay learning rate by half every few epochs.

Exponential decay:

$$\alpha = \alpha_0 e^{-kt}$$

Cosine decay (Best):

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + \cos \left(\frac{T_{cur}}{T_{max}} \pi \right) \right)$$

Summary



- **Simple Gradient Methods** like SGD can make adequate progress to an optimum when used on minibatches of data.
- **Second-order** methods make much better progress toward the goal, but are more expensive and unstable.
- **Momentum:** is another method to produce better effective gradients.
- **ADAGRAD, RMSprop** diagonally scale the gradient.
- **ADAM** scales and applies momentum.



Artificial Intelligence and Machine Learning

Classification / Logistic Regression

Lecture 2: Outline

- Linear Regression (Review)
- Linear Regression (Remaining concepts)
- Intro to Classification
- Logistic Regression (Linear Classifier)
- Classification Metrics
- Other ML Models

Recap

Design your model

- Input scalar linear model (line fitting)
- Fitting polynomials (synthetically designing features from a one-dimensional input)

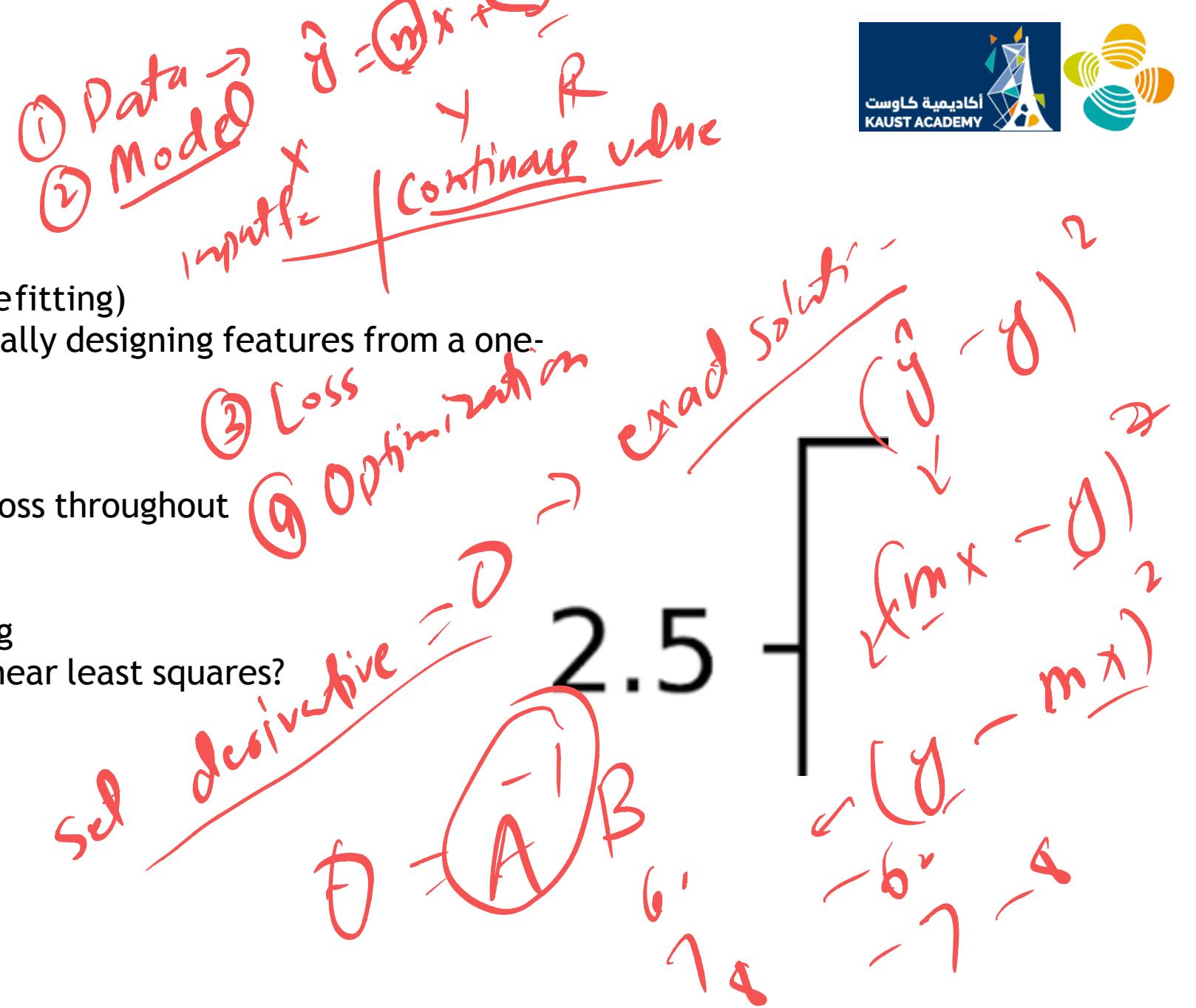
Design your loss function

- We used mean squared error loss throughout

Finding optimal parameter fitting

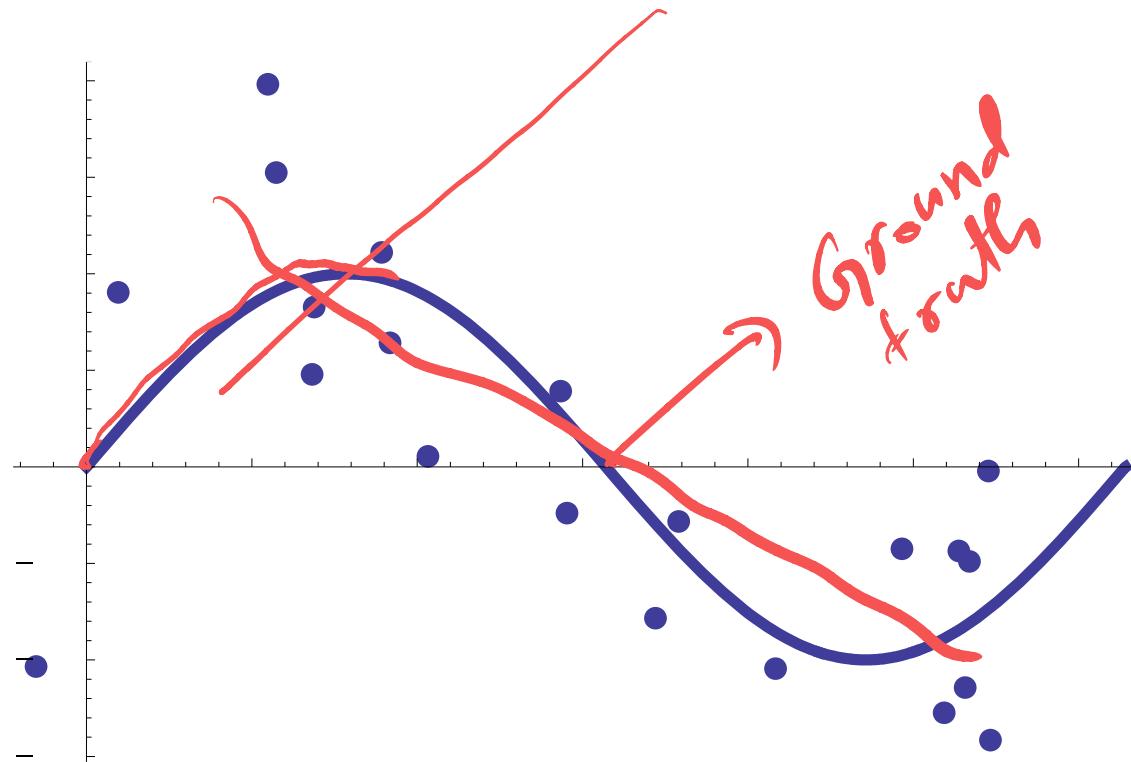
- Closed form solution to the linear least squares?
- Why is it linear least squares?
- Solution is closed form

gradient descent



Bias and Variance

- What if Y has a non-linear response?

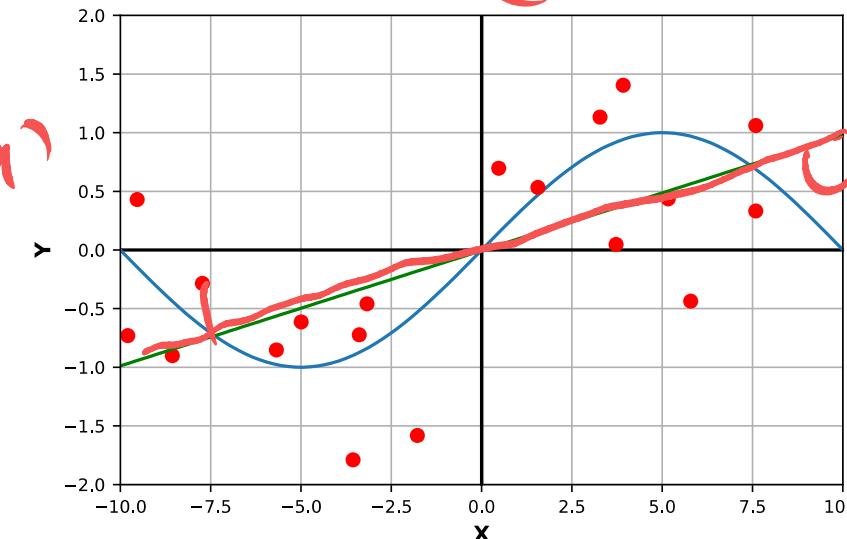


- Can we still use a linear model?

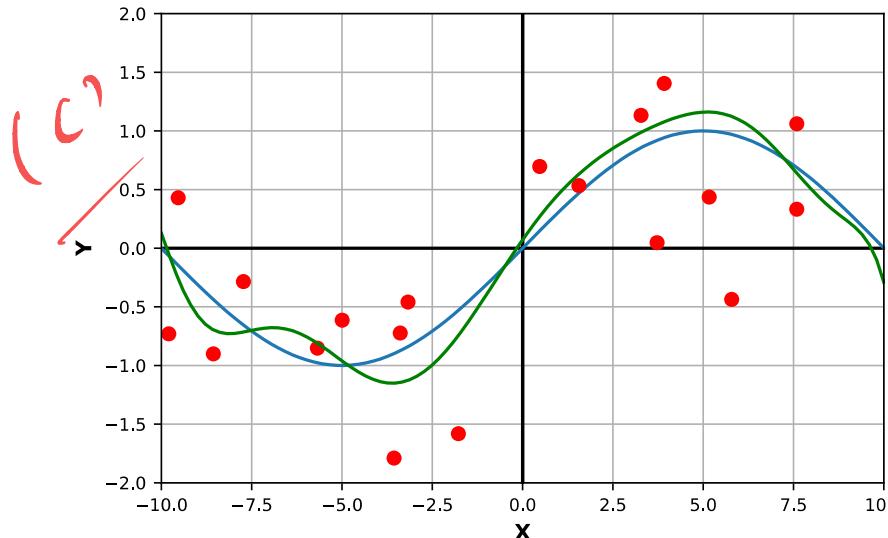
→ 2nd feature, 3rd feature



$\{1, \cancel{x}\}$

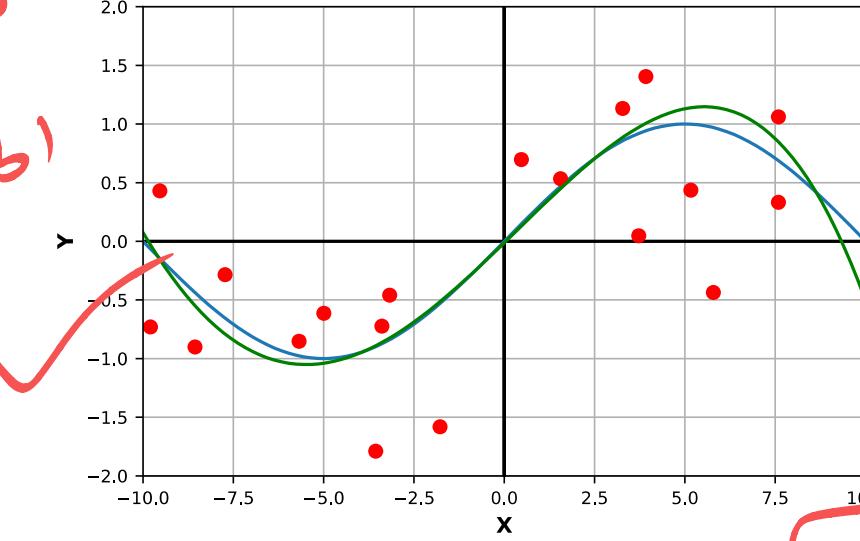


$\{1, x, x^2, \dots, x^9, x^{10}\}$



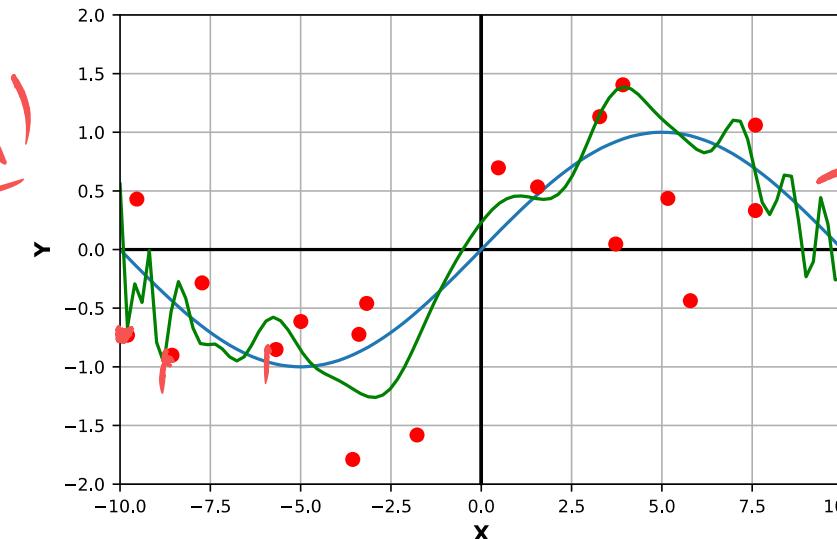
underfit

$\{1, \cancel{x}, \cancel{x^2}, \cancel{x^3}, \cancel{x^4}\}$



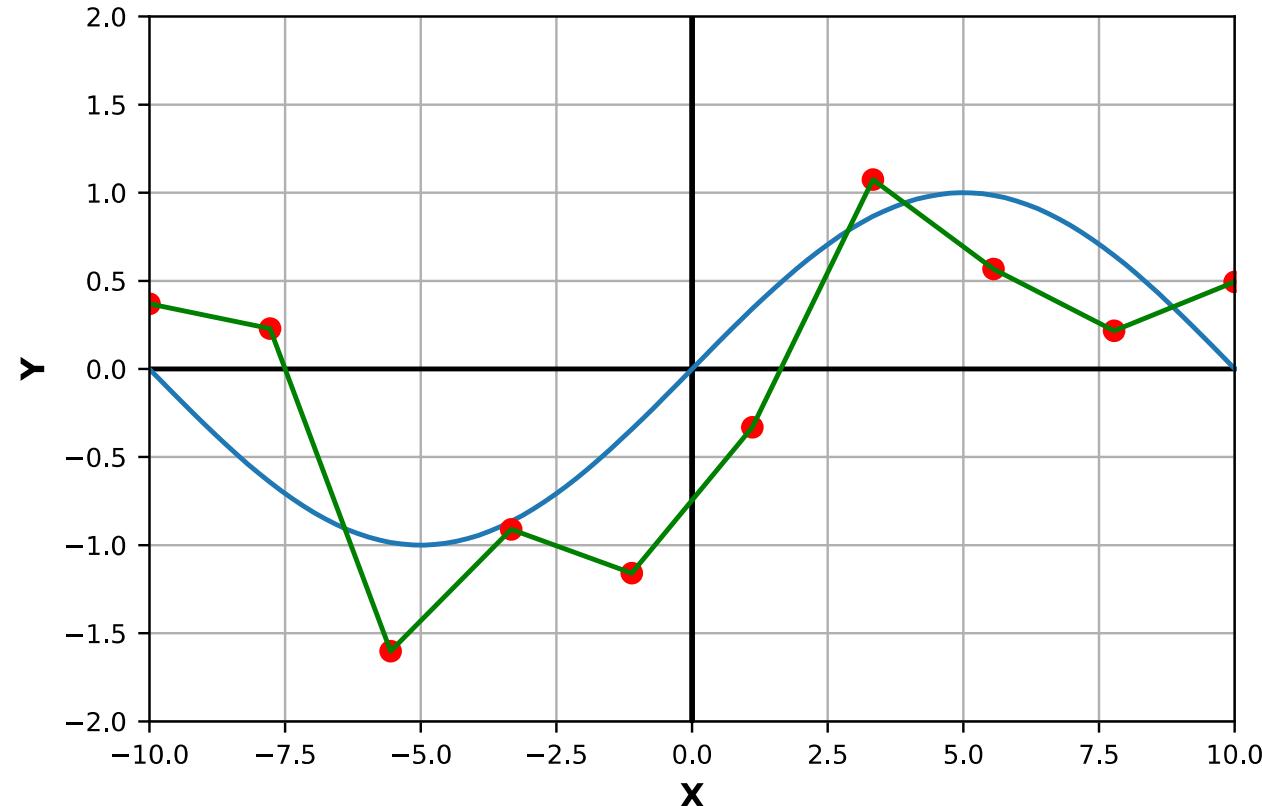
*green \rightarrow model
blue \rightarrow actual*

$\{1, x, x^2, \dots, x^{99}, x^{100}\}$



overfitting

Real Bad Overfit?



lowered error rates

Bias-Variance Tradeoff

- So far we have minimized the error (loss) with respect to **training data**

- Low training error does not imply good expected performance: **over-fitting**

- We would like to reason about the **expected loss** (**Prediction Risk**) over:

- Training Data: $\{(y_1, x_1), \dots, (y_n, x_n)\}$

- Test point: (y^*, x^*)

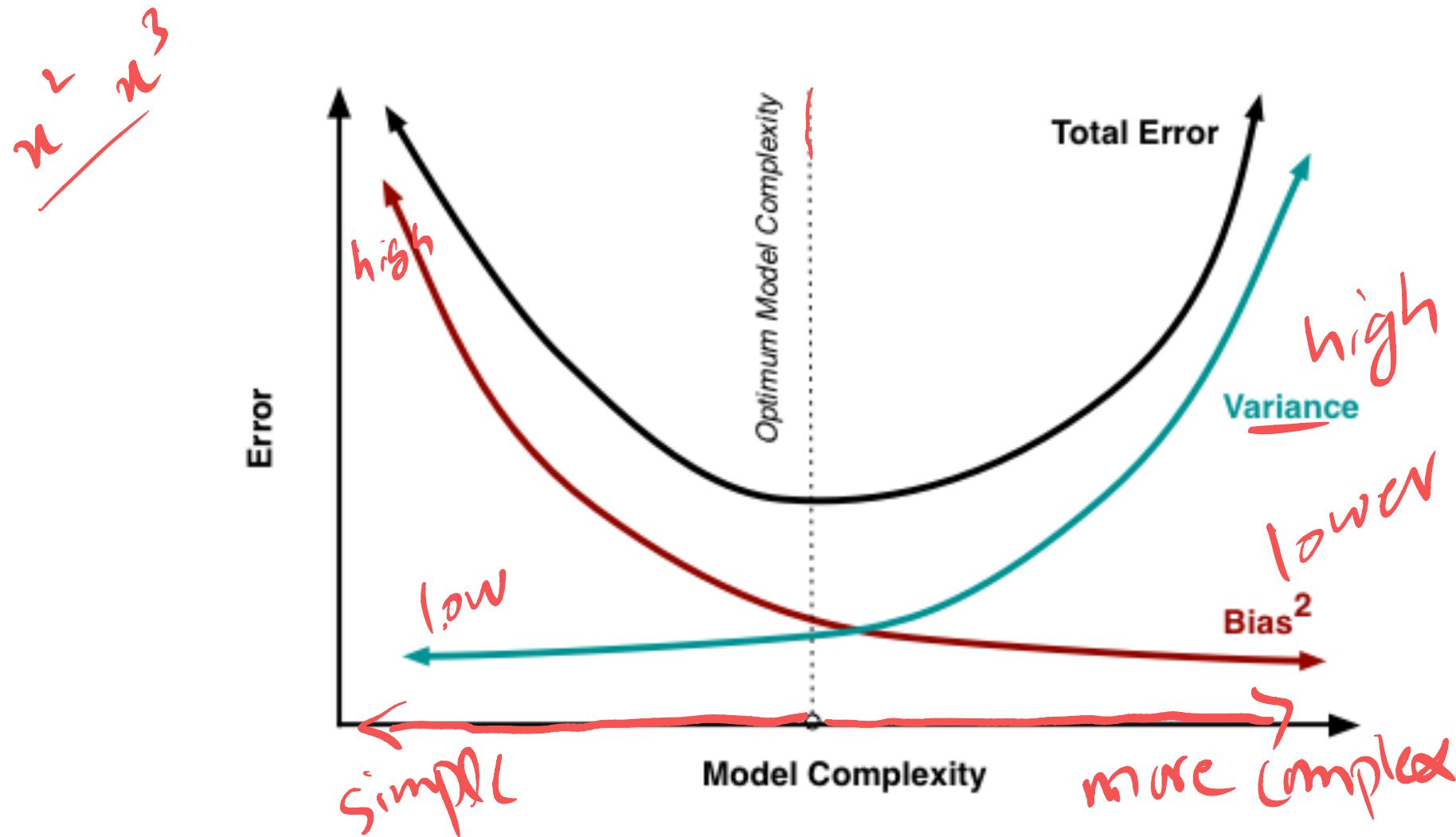
- We will decompose the expected loss into:

ansren

$$\mathbb{E}_{D, (y^*, x^*)} [(y^* - f(x^*|D))^2] = \text{Noise} + \text{Bias}^2 + \text{Variance}$$

Loss $\approx \sum_{i=1}^n (y_i - \hat{y}_i)^2$

Bias Variance Plot



Data Split



- To ensure your model doesn't overfit to the training data, you should have another **subset** called **testing data**.
- You will evaluate your model against this subset, and based on its **metric score (e.g. accuracy)** you will decide if it's overfitting or not.
- But how should I split my data?

Out of 100 → 20%^{sample}

Data Split



- **Hold-out set:**

- A portion of the dataset set aside and not used during training.
- E.g. 80% for training and 20% for testing.

Issues:

- Imagine you have these labels: [1, 1, 2, 2, 2, 3, 3, 3, 3, 3] and you took last 30% as test: [3,3,3]. You didn't include 1 and 2 in test!

Solution: Always shuffle before split: [3, 2, 3, 1, 3, 2, 3, 1, 3, 2] test: [1,3,2]

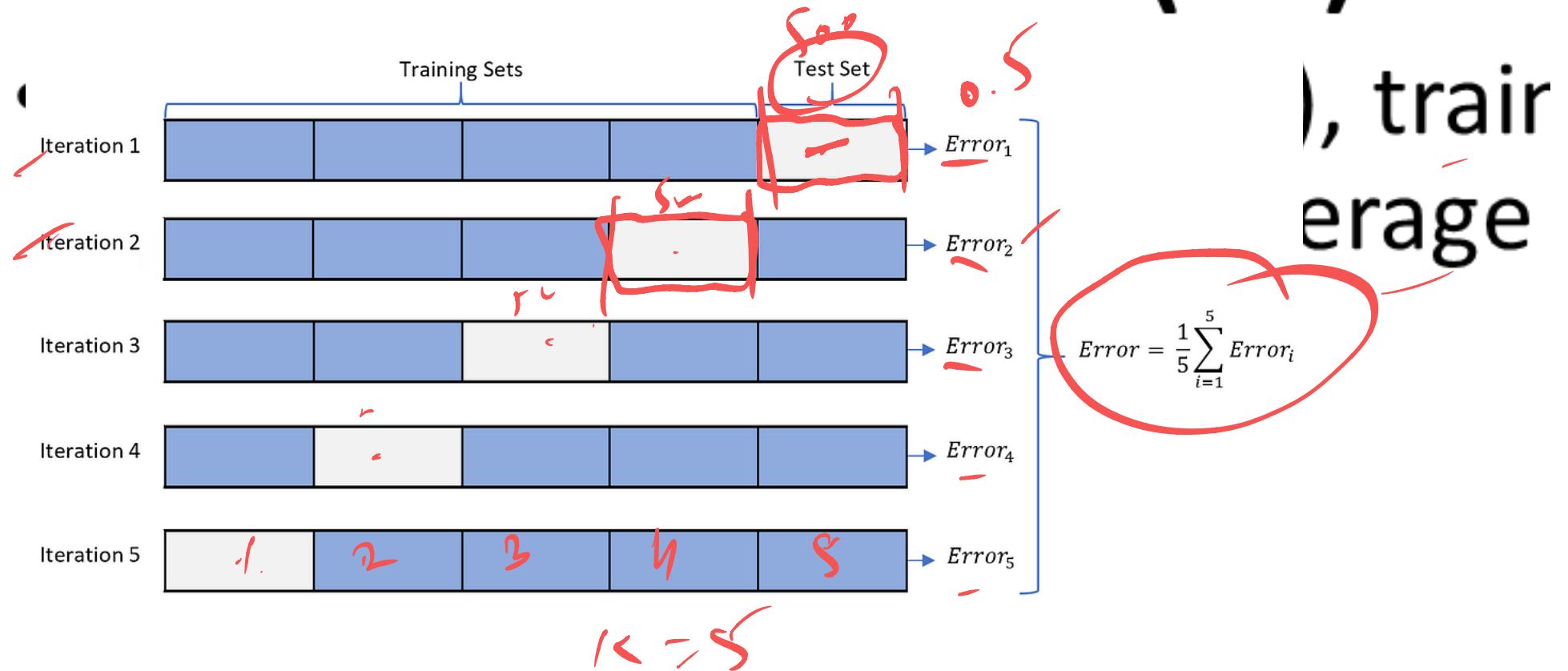
- My dataset is small. Taking 20% as test would not be representative!

Solution: Use KFold.

Data Split



- **K-Fold Cross Validation (CV):**



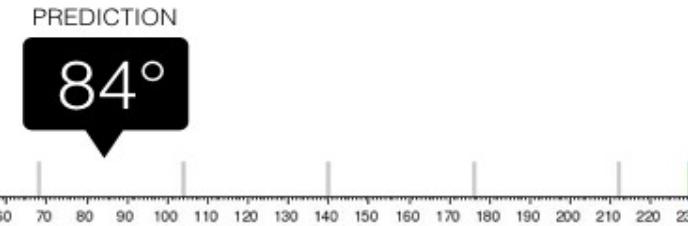
Regression VS classification

logistic
regression



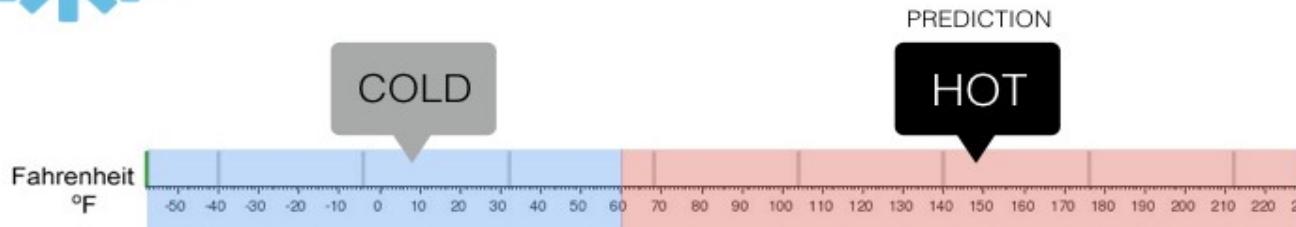
Regression

What is the temperature going to be tomorrow?



Classification

Will it be Cold or Hot tomorrow?



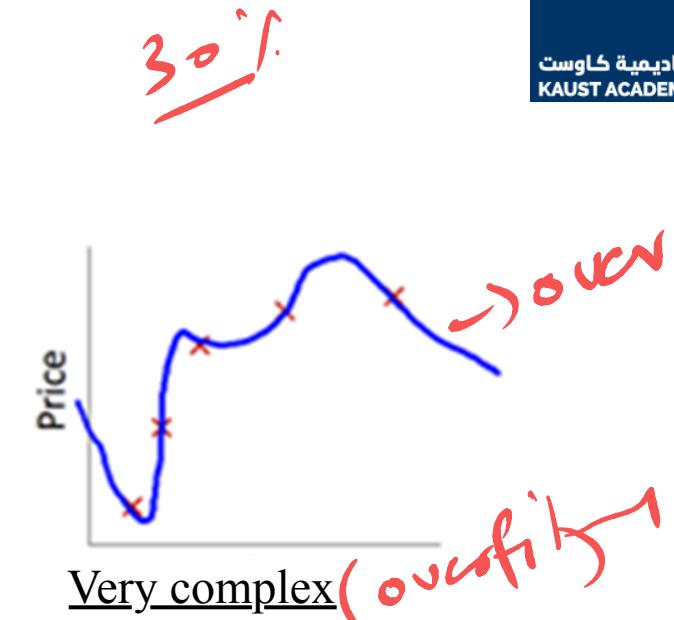
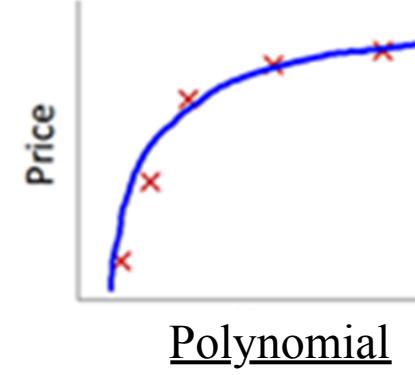
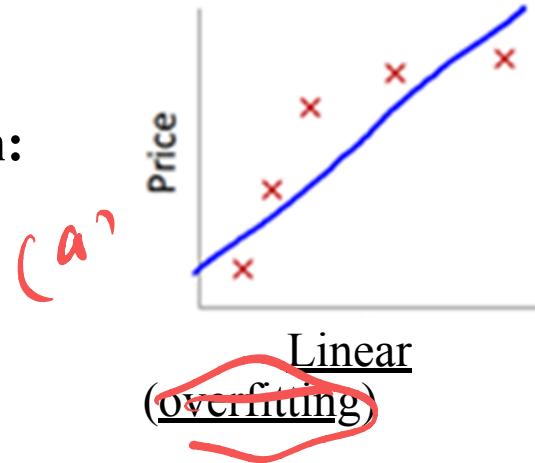
=> Continuous Values

rain
30°-
70°- clouds

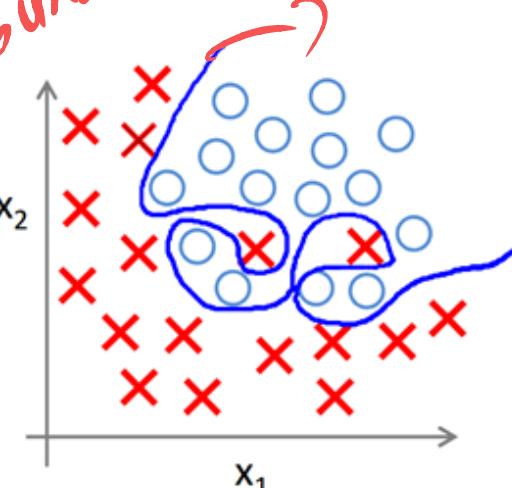
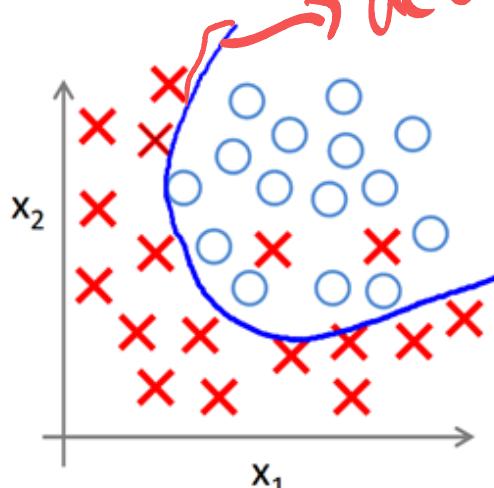
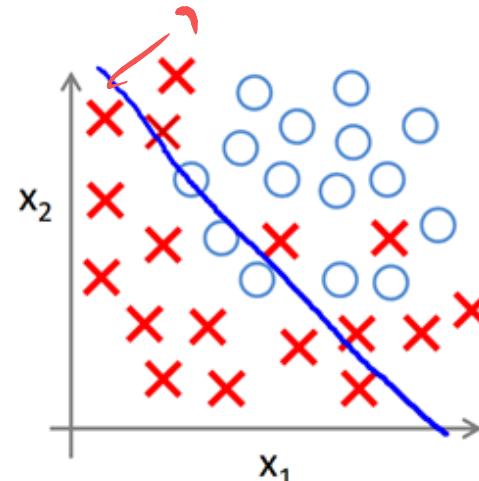
=> Discrete Values

Regression VS classification

Regression:



Classification:



decision boundary

Classification Types

Binary Classification: Two possible outcomes (e.g., Yes/No, Spam/Not Spam).

How? Single Classifier.

He is a good person

positive/neutral/not

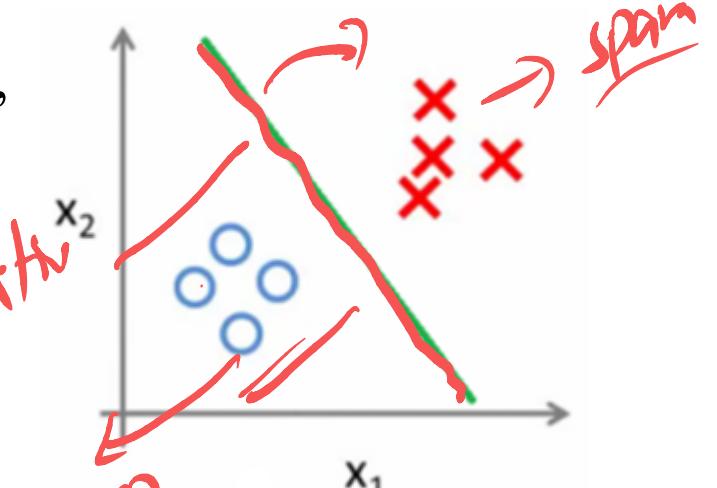
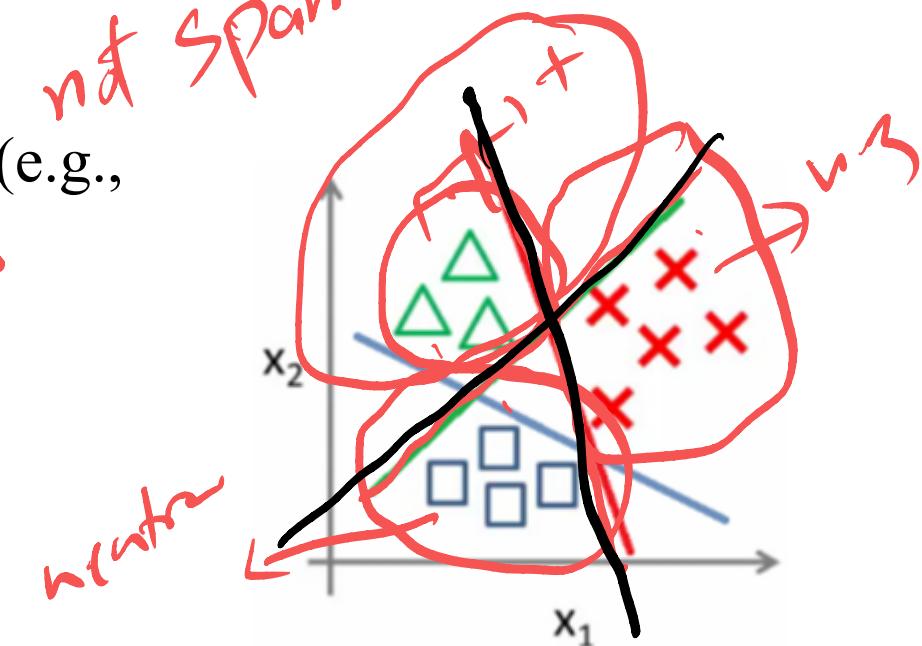
not Spam

Multiclass Classification: More than two outcomes (e.g., Class A, B, C).

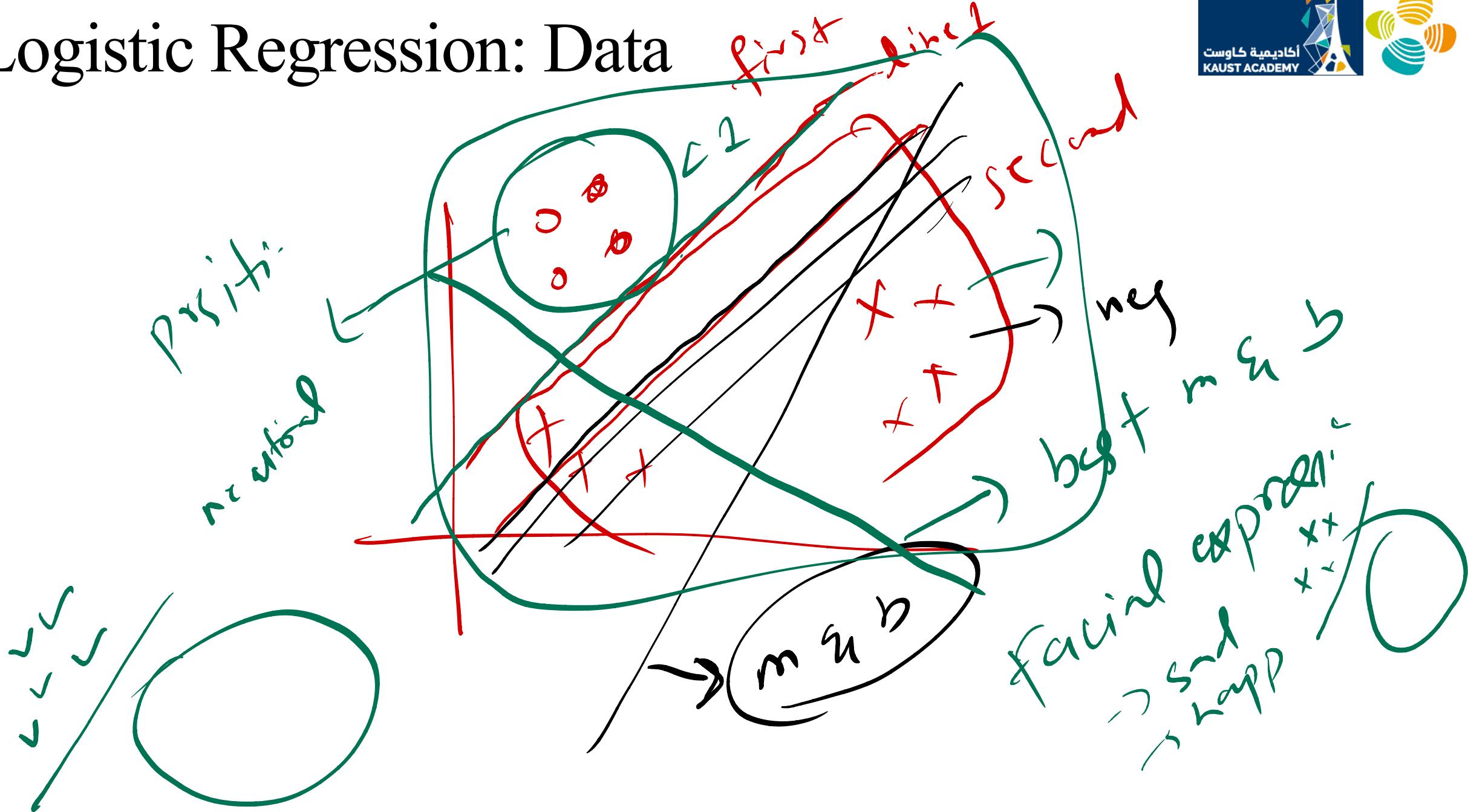
How? Multiple *Single Classifiers*.

sentiments classification

neutral



Logistic Regression: Data

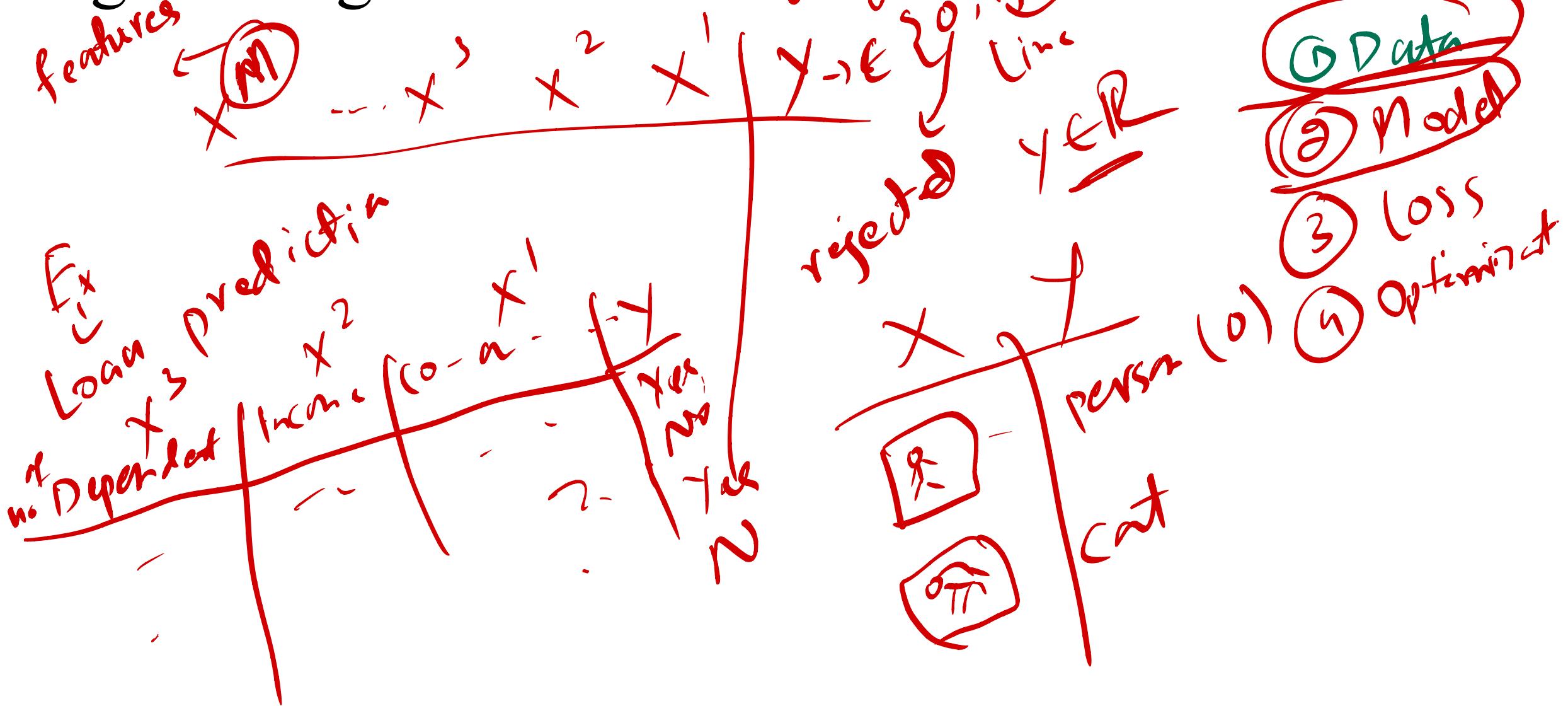


Logistic Regression: Data

c_1 happy vs
 c_2 Sad vs
 c_3 disgust vs
 sum \rightarrow one vs all



Logistic Regression: Data

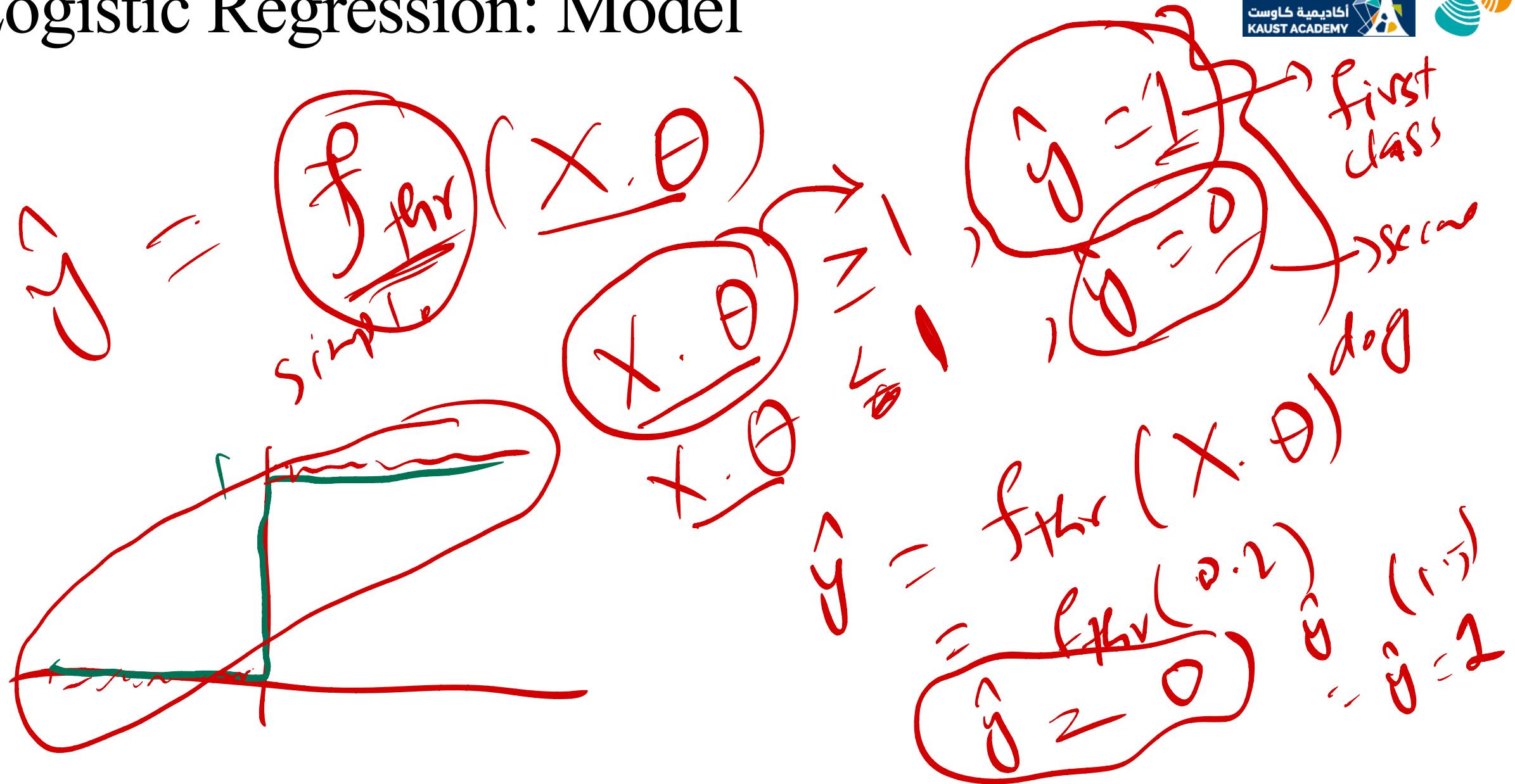


Logistic Regression: Model



- Linear regression
 - $y = f(x)$
 - continuous
 - $\hat{y} = f(x)$
 - \hat{y} = discrete value
 - Predicts memorability of image
 - image regres^{0.23}
- ~~Linear regression~~

Logistic Regression: Model



Logistic Regression: Model

- chances of min probability $\rightarrow \hat{y} = \frac{1}{1 + e^{-x^T \theta}}$
- computation of gradients $\rightarrow \nabla_{\theta} L(\theta) = \sum_{i=1}^m (\hat{y}_i - y_i) x_i$
- generalise to multi-class \rightarrow softmax function $\rightarrow \hat{y}_j = \frac{e^{x_j^T \theta_j}}{\sum_k e^{x_k^T \theta_k}}$
- n-confidence equally likely \rightarrow $\theta_0 = \theta_1 = \dots = \theta_n$
- dist \rightarrow $\theta_0 = \theta_1 = \dots = \theta_n$

Logistic Regression: Model

sigmoid function

$f(x) = \frac{1}{1 + \exp(-x)}$

$\delta_0 = \frac{1}{1 + \exp(-x_0 \cdot \theta)}$

approaching 0 as $x_0 \rightarrow -\infty$

approaching 1 as $x_0 \rightarrow \infty$

$\delta_0 = \frac{1}{1 + \exp(-x_0 \cdot \theta)}$

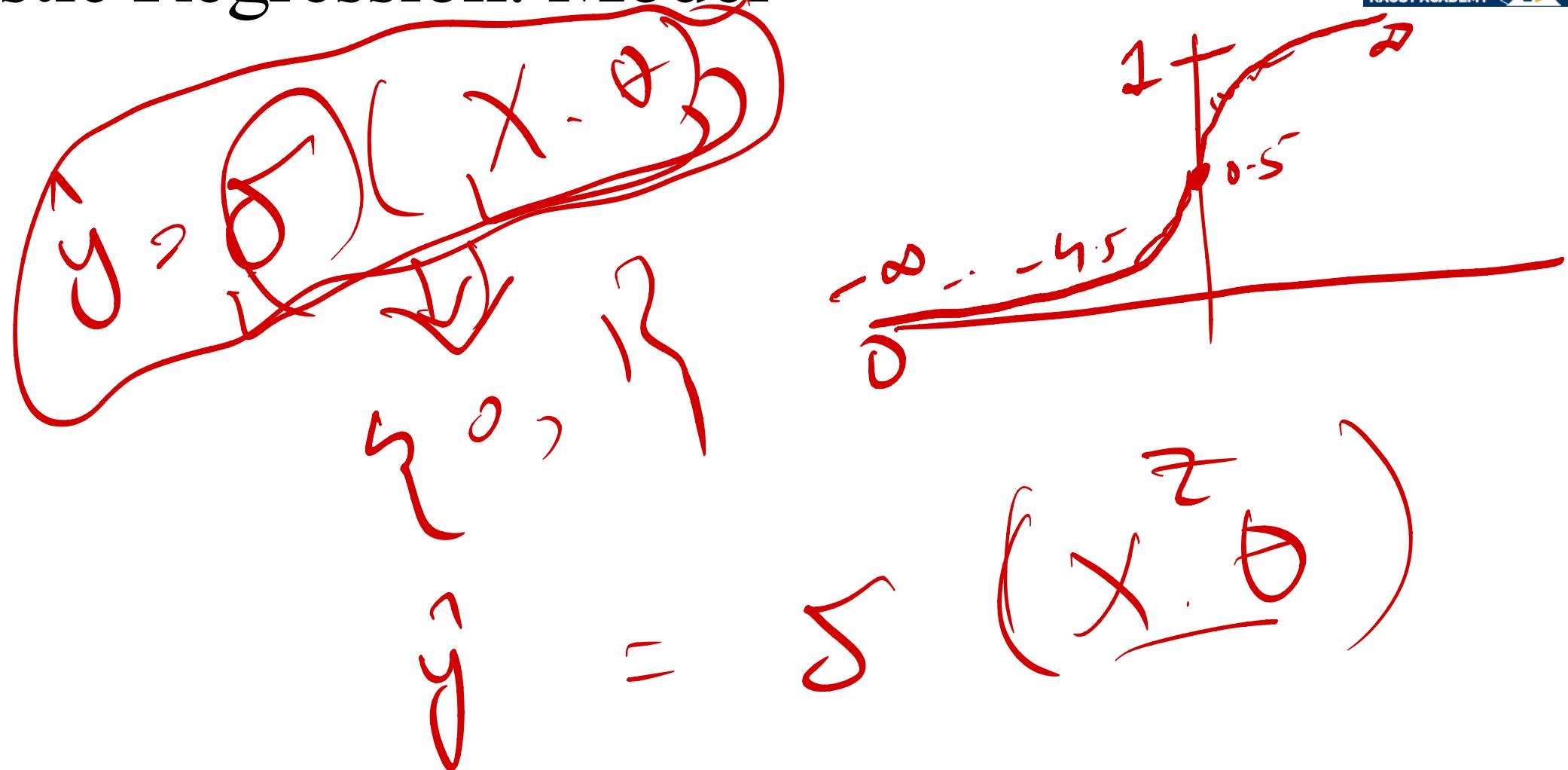
changing factor

$z = 0, \frac{1}{1 + \exp(-0)} = \frac{1}{1+1} = 0.5$

$z = \infty, \frac{1}{1 + \exp(-\infty)} = \frac{1}{1+0} = 1$

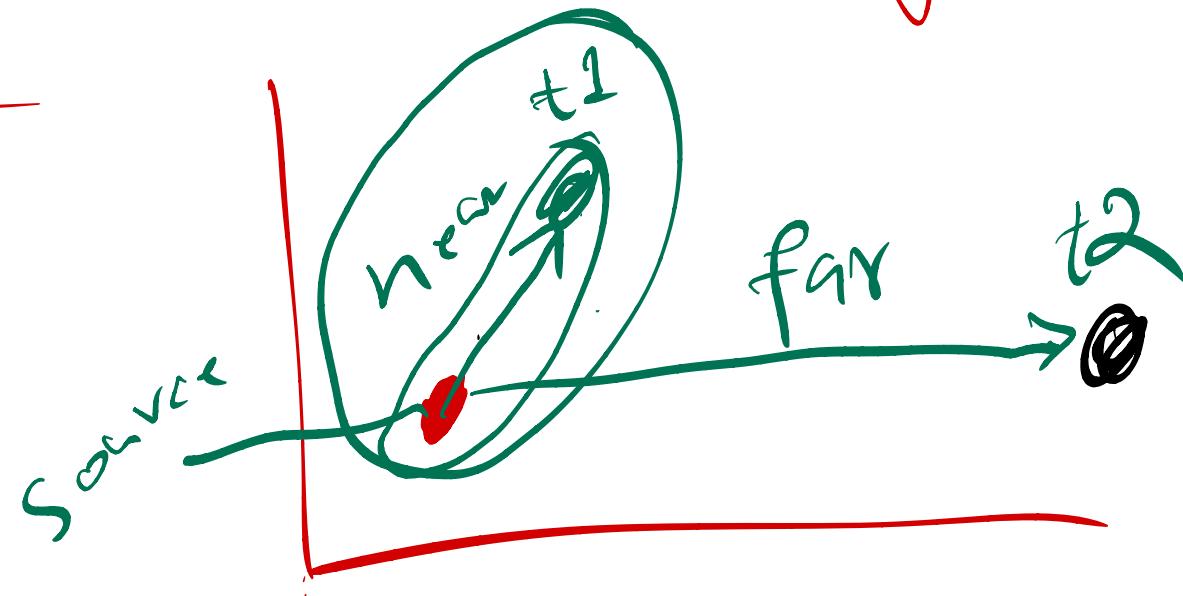
$z = -\infty, \frac{1}{1 + \exp(-(-\infty))} = \frac{1}{1+0} = 0$

Logistic Regression: Model



Logistic Regression: Loss

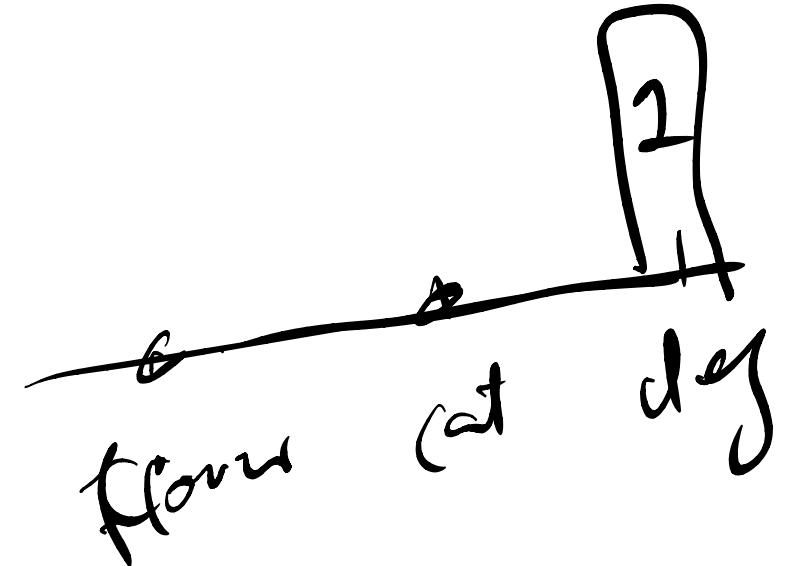
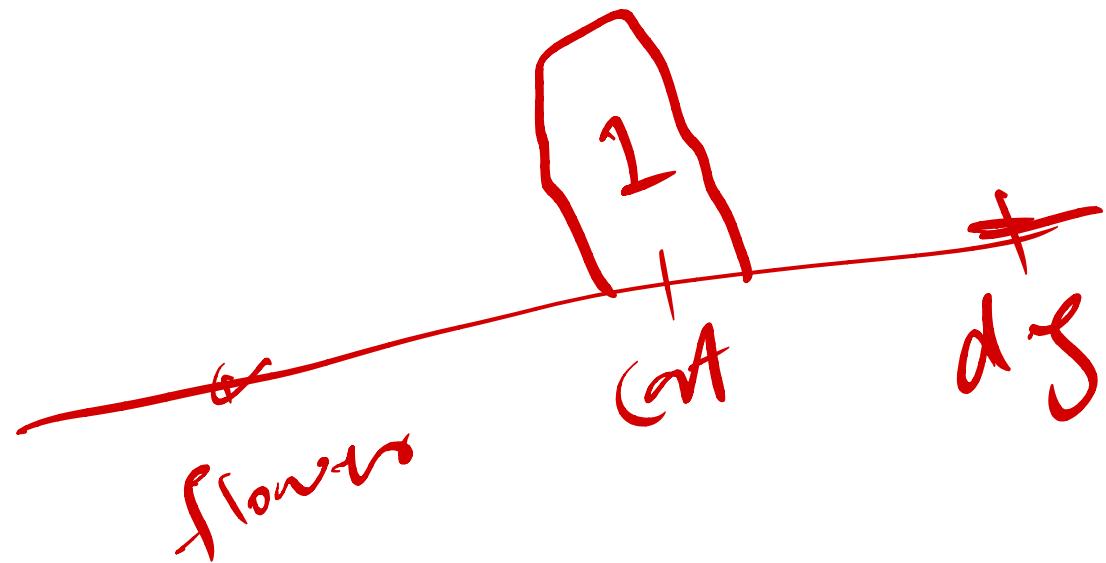
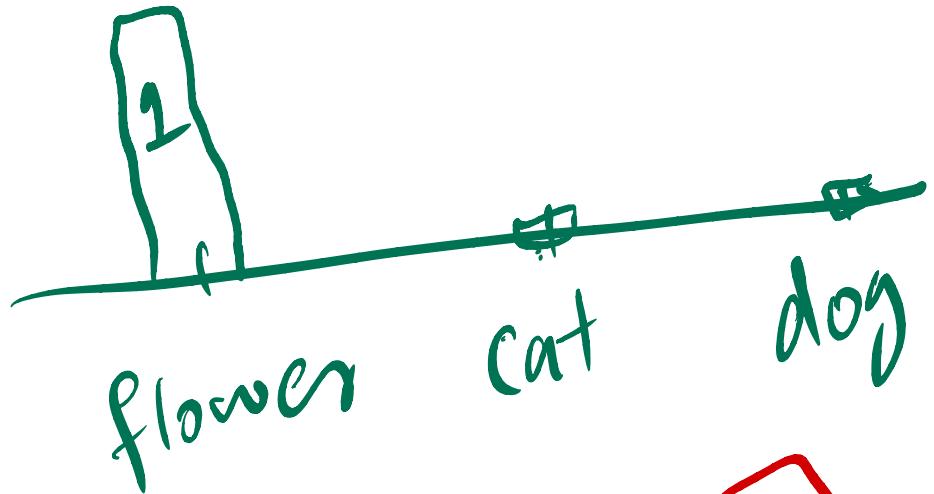
$$MSE = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$



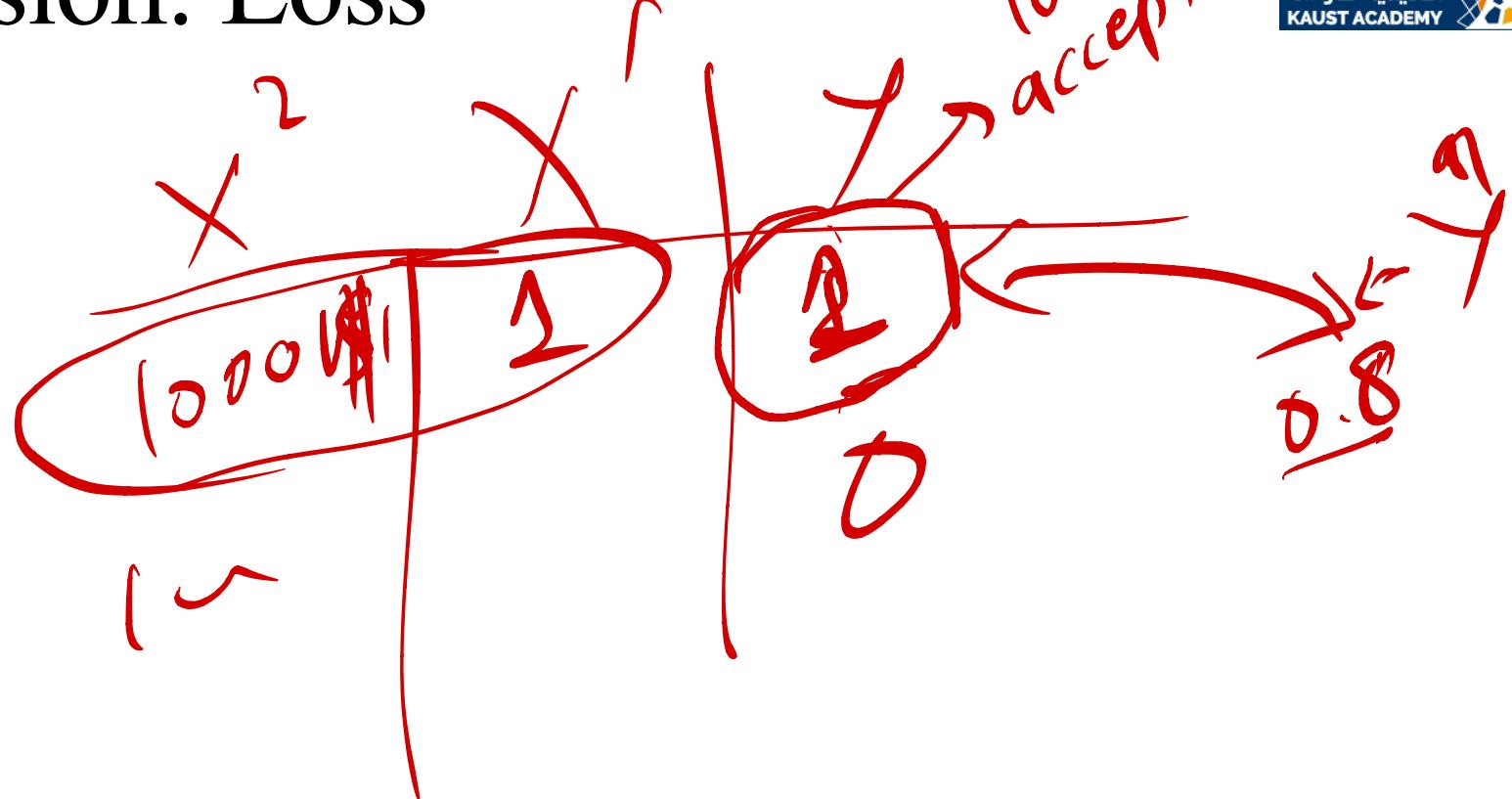
Compute the difference
below continue
values.

Logistic Regression: Loss

Logistic
classification
→ probabilities



Logistic Regression: Loss



KI
diagram
cross entropy

Logistic Regression: Loss

$$0 - 2(\hat{y} - y) = -y \log \hat{y} - (1-y) \log(1-\hat{y})$$

$y = 0$

$\hat{y} = 1$

$$J = - (1-y) \log(1-\hat{y})$$

$$J = - y \log \hat{y}$$

$$J = - \frac{1}{n} \sum_{i=1}^n y_i \log h(x_i)$$

very high loss

Logistic Regression: Loss

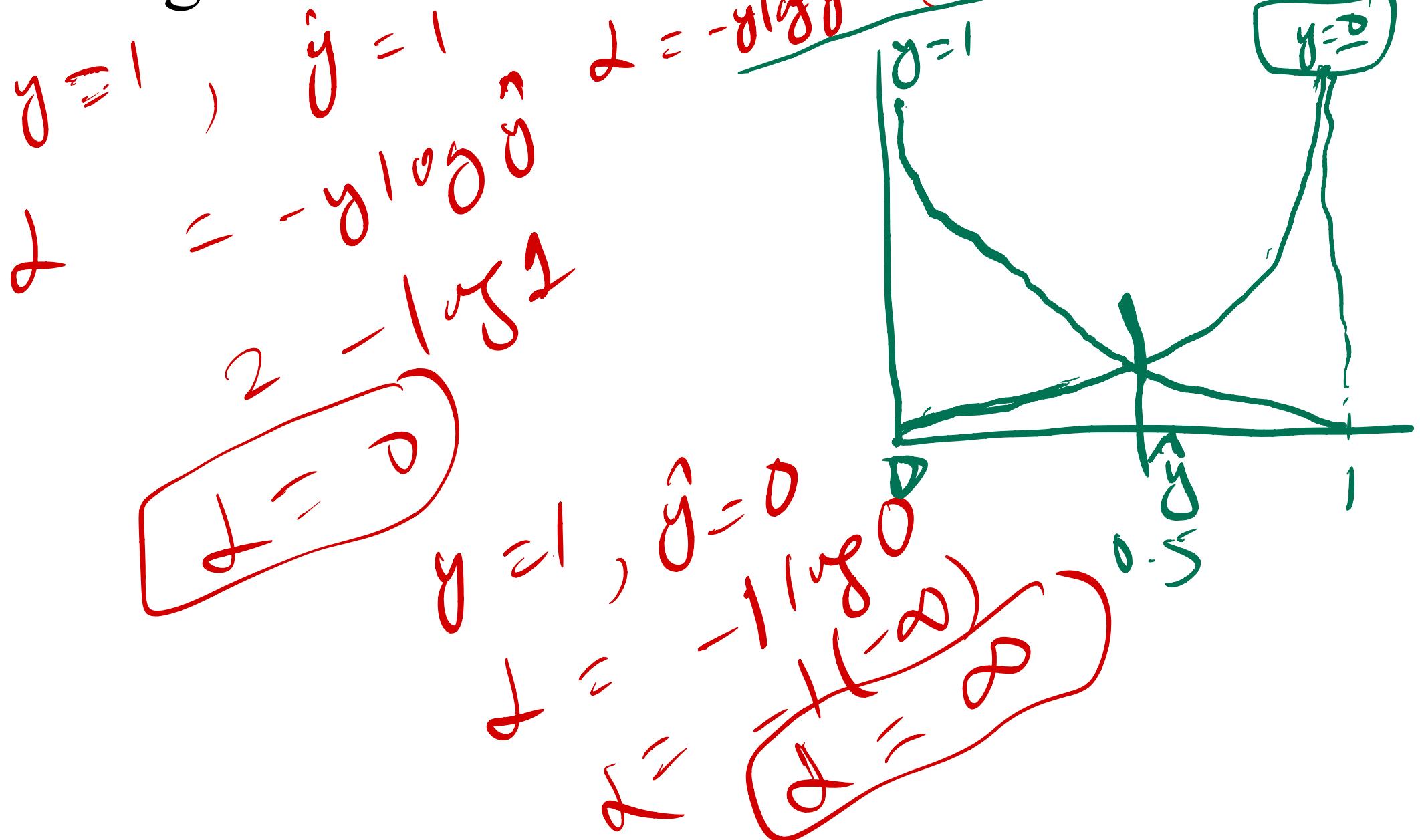


$$\text{Loss} = -y \ln(p) - (1-y) \ln(1-p)$$

Handwritten notes:

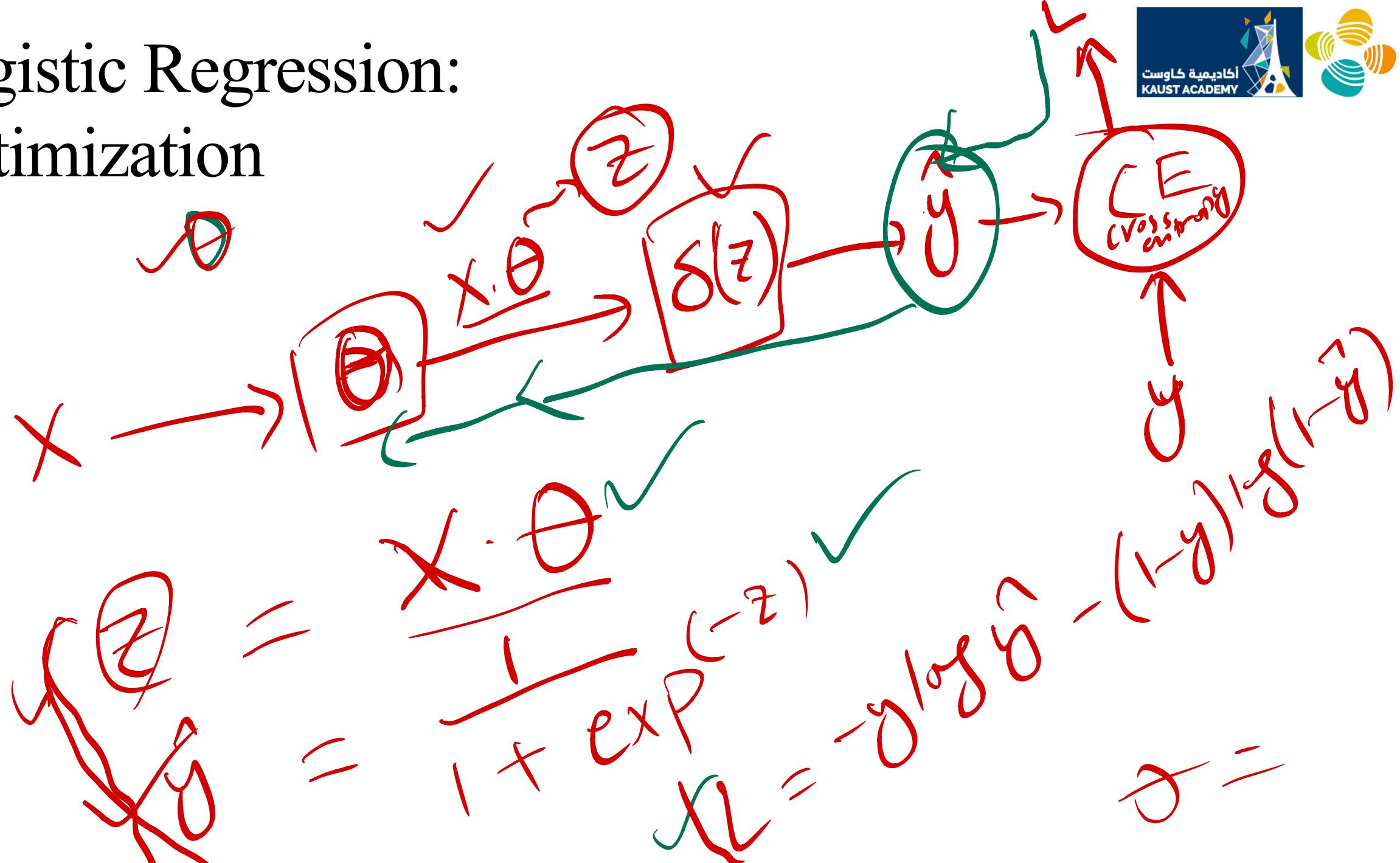
- $y=0, p=0$ (highlighted with a green oval)
- $y=1, p=1$
- $y=0.5, p=0.5$
- $y=0, p=1$
- $y=1, p=0$

Logistic Regression: Loss





Logistic Regression: Optimization



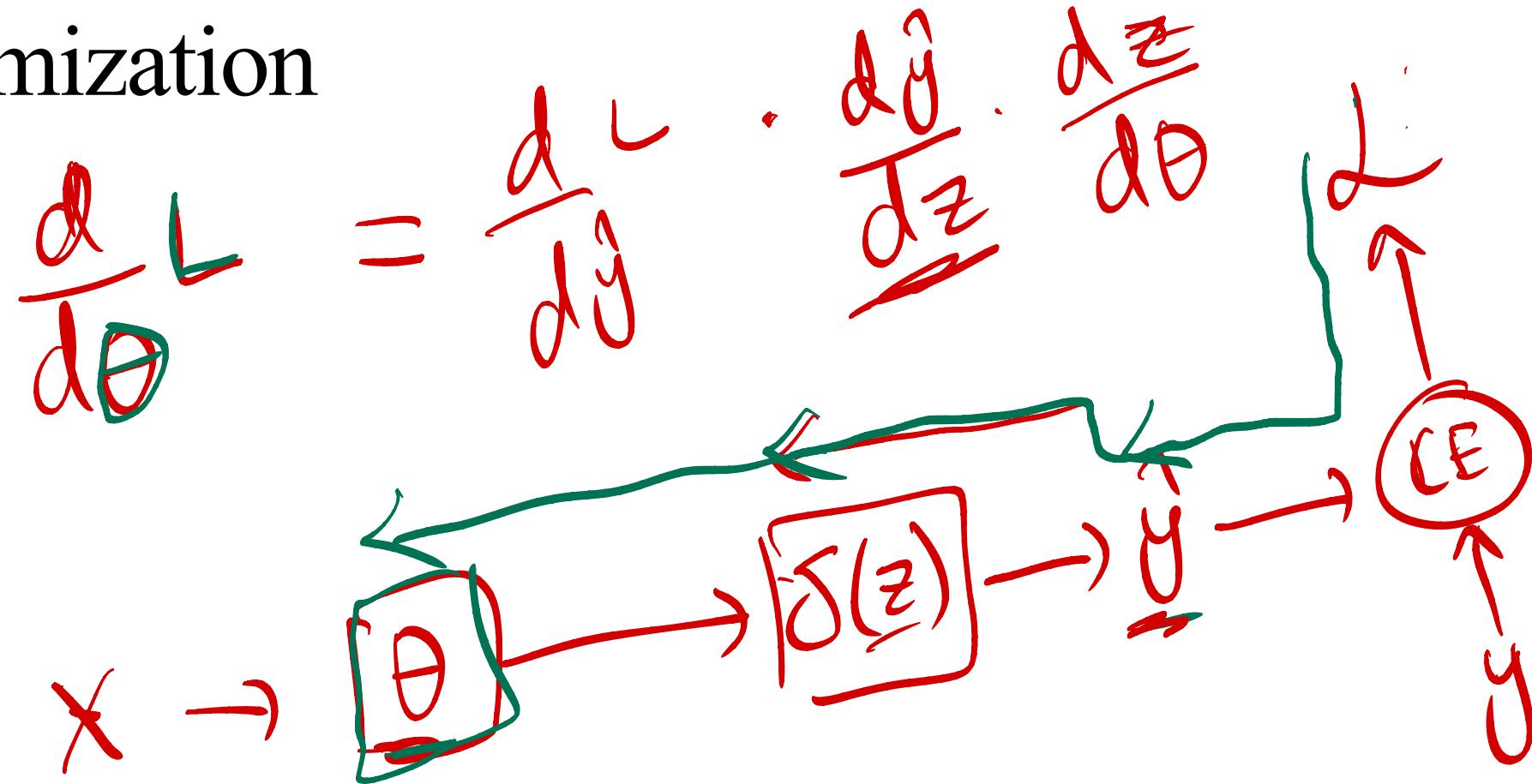
Logistic Regression:

Optimization

Lineal \rightarrow set derivative = 0
non-linear → open system linear equation

Logistic Regression:

Optimization



Logistic Regression: Optimization

$$\frac{\partial L}{\partial \hat{y}} = \frac{d}{d\hat{y}} [y \log \hat{y} - (1-y) \log(1-\hat{y})]$$

$$\frac{\partial L}{\partial y} = \hat{y} - y$$

$$\frac{\partial L}{\partial \hat{y}} = \hat{y}(1-\hat{y})$$

Logistic Regression:

Optimization

$$\frac{d\hat{y}}{dz} = \frac{d}{dz} \left(\frac{1}{1 + \exp(-z)} \right)$$

$$\frac{d\theta}{dz} = \frac{d}{dz} \left(\frac{1}{1 + \exp(-z)} \right) \cdot g'(1 - g)$$

$$\frac{dy}{dz} = g'(1 - g)$$

$$\frac{dz}{d\theta} = \frac{d}{d\theta} (x^\top \theta) = x^\top$$

Logistic Regression:

Optimization

$$\frac{dL}{d\theta} = \frac{dL}{d\hat{y}} \cdot \frac{d\hat{y}}{dz} \cdot \frac{dz}{d\theta}$$

$\frac{\partial L}{\partial \theta} = 0$

$\frac{dL}{d\theta}$ circled.

$X^T [g - \hat{y}] \cdot g(1-\hat{y})$ circled.

linear

Na

Logistic Regression: Optimization

A.G.D

$\theta^0 \rightarrow$ initialize (Randomly)

while until converged.

$\theta^{t+1} = \theta^t - \frac{m}{n} \frac{\sum d\theta}{d\theta}$

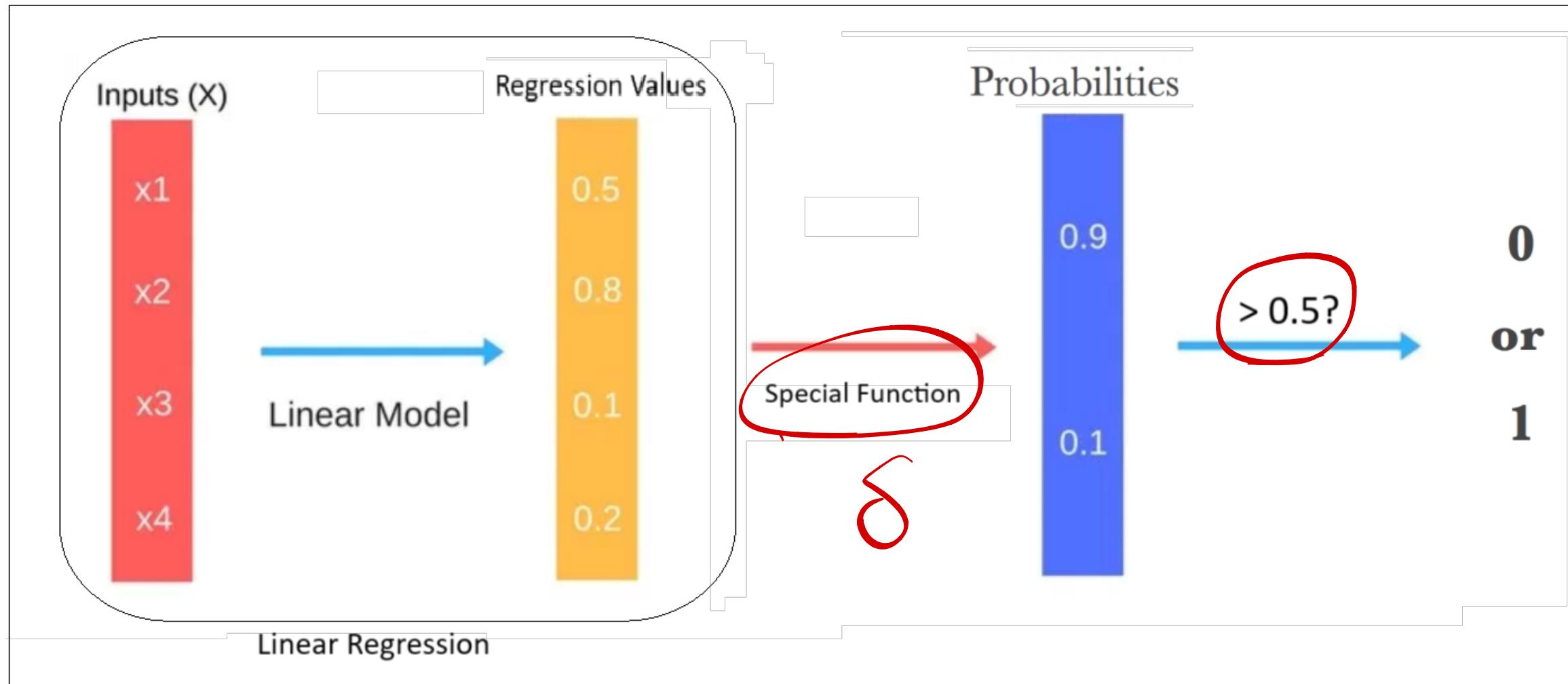
Classifiers

Classifiers are **regression** models with two key modifications:

- Pass the regression outputs through a **special function** to convert them into probabilities.
- Replace the Mean Squared Error loss with a **new loss function** designed for probabilities.

Applying these two to the Linear Regression should give Linear Classifier (called **Logistic Regression**).

Logistic Regression



Special Function

We are searching for a function that have the following characteristics:

1. Could convert any arbitrary input values to **[0,1] range (Probabilities)**.
2. **Smooth and Differentiable**. Because we want to find the minima later, right?

Any ideas?

Sigmoid Function (for Binary Classification)

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

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$

Assuming you have negative and positive classes, its output would be the probability of the positive class.

But what if we have multiclass problem? 🤔



$$\lim_{z \rightarrow -\infty} \sigma(z) = 0$$

$$\lim_{z \rightarrow \infty} \sigma(z) = 1$$

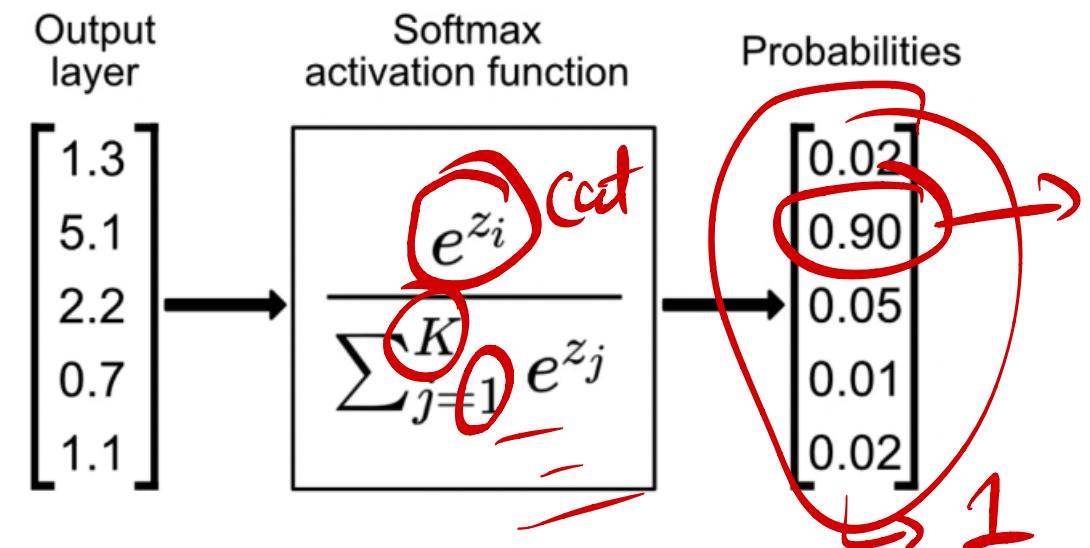
Softmax Function (for Multiclass Classification)

Softmax(\mathbf{z}) = softmax(\mathbf{z}) = $\frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$)

E.g. Divide the cat value by the cat, dog and deer values to get the cat probability.

Assuming you have K classes, its output would be the probability of the ith class. So, you will run it K times to get the probability of each class.

Note: Sigmoid is a special case of Softmax. Optional: Can you prove it?



Classification LOSS

We are searching for a loss that have the following characteristics:

1. **Probabilistic behaviour**: Should work with probabilities, not raw numbers.
2. **Sensitivity**: It should heavily penalize incorrect confident predictions (e.g., predicting 0.9 when the true label is 0).
3. **Differentiable**: Must be smooth and differentiable to enable optimization.

Any ideas?

Classification LOSS

We are searching for a loss that have the following characteristics:

1. **Probabilistic behaviour:** Should work with probabilities, not raw numbers.
2. **Sensitivity:** It should heavily penalize incorrect confident predictions (e.g., predicting 0.9 when the true label is 0).
3. **Differentiable:** Must be smooth and differentiable to enable optimization.

Any ideas?

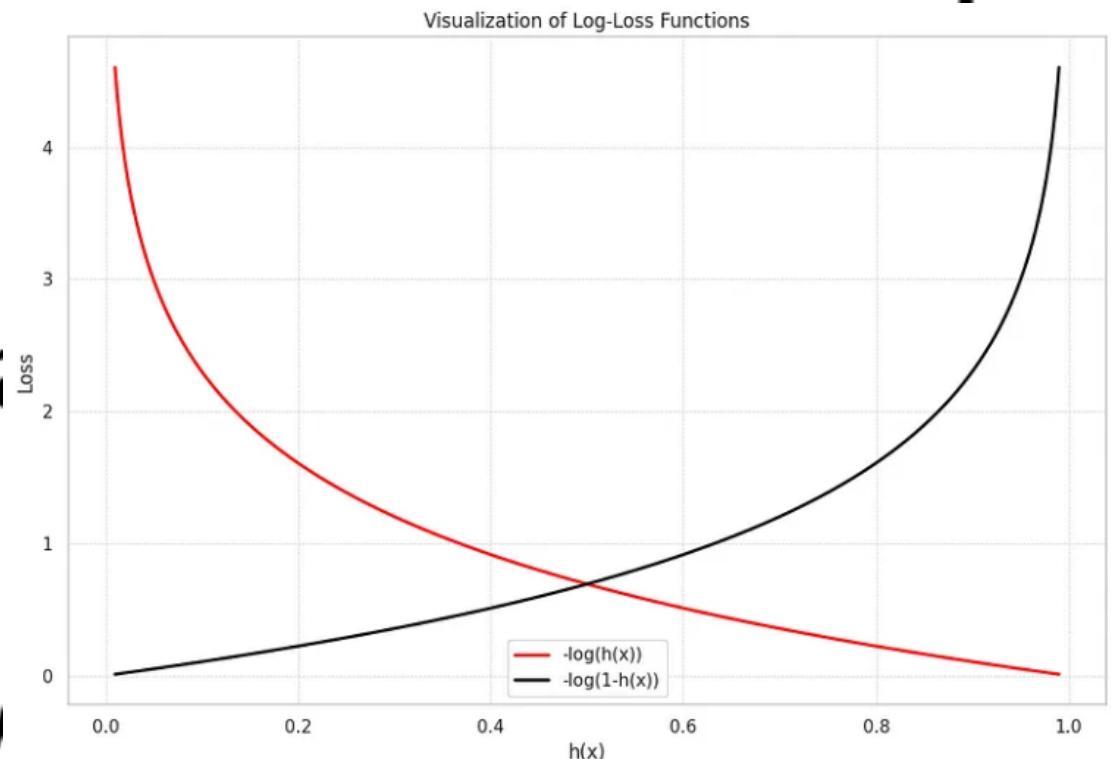
Logarithm is all you need :)

Binary Cross Entropy (LogLoss)

- For one sample, this can be represented as:
$$-\log(h(x))$$

If $y = 1$ and prediction be close to **zero**. (Great loss)

If $y = 1$ and prediction be close to **infinity**. (Very small loss)



Binary Cross Entropy (LogLoss)

- For one sample, this can be represented as:

$$\text{loss}(y, p) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0 \end{cases}$$

- Let's rewrite it in one line:

$$\text{loss}(y, n) = -(y * \log(p) + (1-y) * \log(1-p))$$

Binary Cross Entropy (LogLoss)

- For one sample, this can be represented as:

$$loss(y, p) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1-p) & \text{if } y = 0 \end{cases}$$

- Let's write it in one line:

$$loss(v, p) = -(v * \log(p) + (1-v) * \log(1-p))$$

How to find optimal Parameters?

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

Just like before, simply
take

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$$



However, this does not have a nice closed
solution Just like the MSE case

How to find optimal Parameters?

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

Just like before, simply take

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = 0$$

Cost

However, this does not have a nice closed solution Ju

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

Instead, use gradient descent (optimizer)!

Logistic Regression

$$J(\mathbf{w}) = -\frac{1}{N} \sum_{i=1}^N y_i \log(\sigma(\mathbf{w}^T \mathbf{x}_i)) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i))$$

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

Learning rate

Cost

- We have a linear model for prediction
- For classification, we want to output a probability
- We map the prediction to probabilities with a sigmoid function
- We have a loss function (BCE) to compare models



Logistic Regression

Linear Regression	Logistic Regression
For Regression	For Classification
We predict the target value for any input value	We predict the probability that the input value belongs to the specific target
Target: Real Values	Target: Discrete values
Graph: Straight Line	Graph: S-curve

Classification Metrics

- **Accuracy:**

Accuracy measures the proportion of correctly classified samples out of the total number of samples.

$$\text{Accuracy} = \frac{\text{N. of correct sample}}{\text{Overall samples}}$$

Question: Why not optimizing for the accuracy directly instead of using LogLoss?

Classification Metrics

- **Accuracy:**

Accuracy measures the proportion of correctly classified samples out of the total number of samples.

$$A = \frac{C}{D}$$

Question: Why not optimizing for the accuracy directly instead of using LogLoss?

→ Non-differentiable
Anything else?

Classification Metrics

But accuracy isn't always enough!

Imagine an imbalanced dataset with 95% negative labels (e.g., "not spam").
A model predicting "negative" all the time will be 95% accurate but useless.

Classification Metrics

But accuracy isn't always enough!

Imagine an imbalanced dataset with 95% negative labels (e.g., "not spam").
A model predicting "negative" all the time will be 95% accurate but useless.

Different goals require different metrics

- Do you care more about **catching all positives** (e.g., cancer detection)?
- Or avoiding **false alarms** (e.g., fraud detection)?

Classification Metrics

- **Precision:**

Among all the positive predictions the model made,
how many are actually correct?

important when **minimizing false alarms** is critical,
like in fraud detection or spam filtering.

Tr

Dash-dot-dot

Classification Metrics

- **Recall:**

Among all the actual positives, how many did the model correctly identify?

important when **catching all positives** is vital, such as in cancer detection.

Denominator —

$$\frac{TP}{TP + FN} Tr$$

Classification Metrics

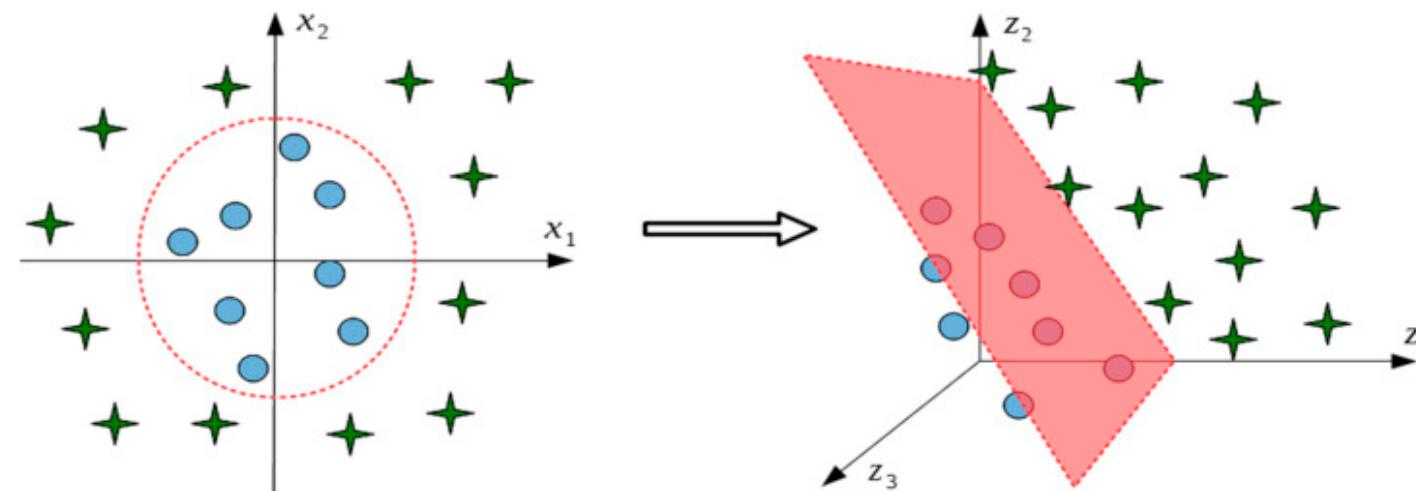
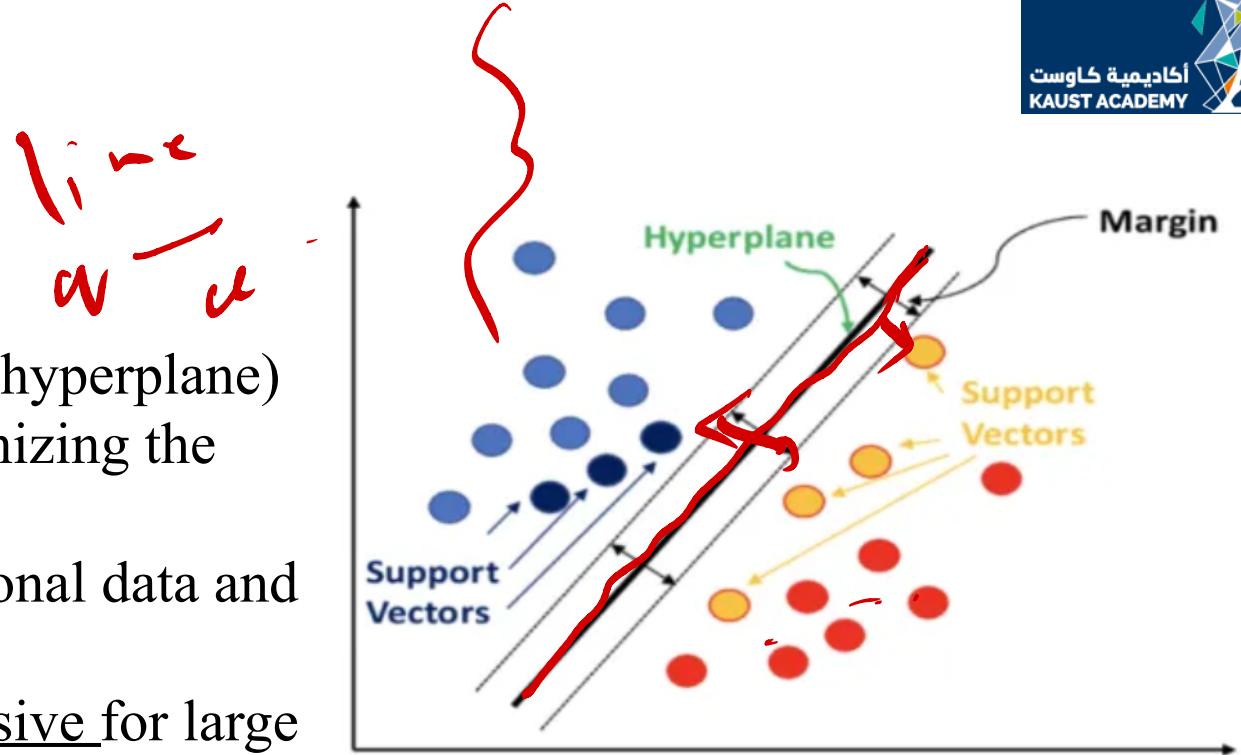
- **F1 Score:**

What if both precision and recall are important?
Take their (harmonic) mean.

$$F1 \ score = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

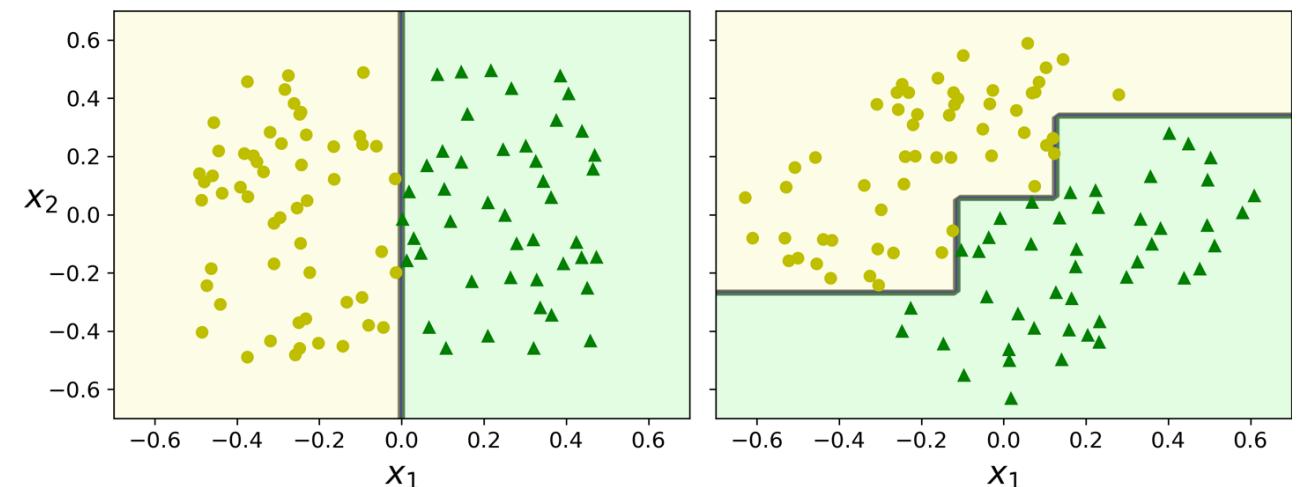
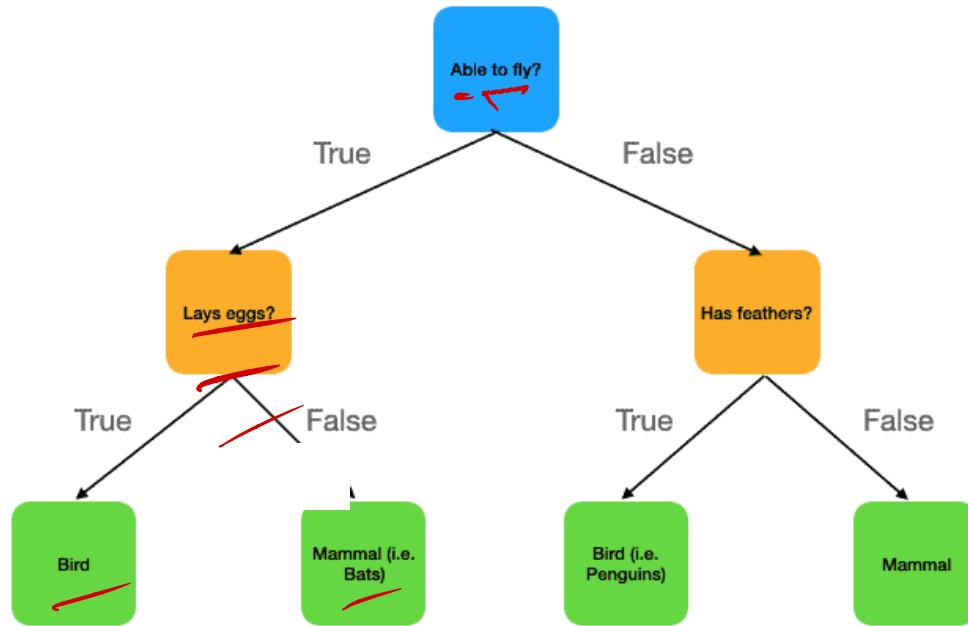
Other ML Models

- **Support Vector Machine (SVM):**
Key Idea:
- SVM finds the best boundary (hyperplane) that separates classes by maximizing the margin between them.
- Works well with high-dimensional data and small datasets.
- Can be computationally expensive for large datasets.



Other ML Models

- **Decision Trees:**
- **Key Idea:**
- A tree-based model splits data into subsets based on feature thresholds, creating a flowchart-like structure for decision-making.
- Overfits very easily.
- could work bad if relation between features and the target is linear. (Check the right figure)



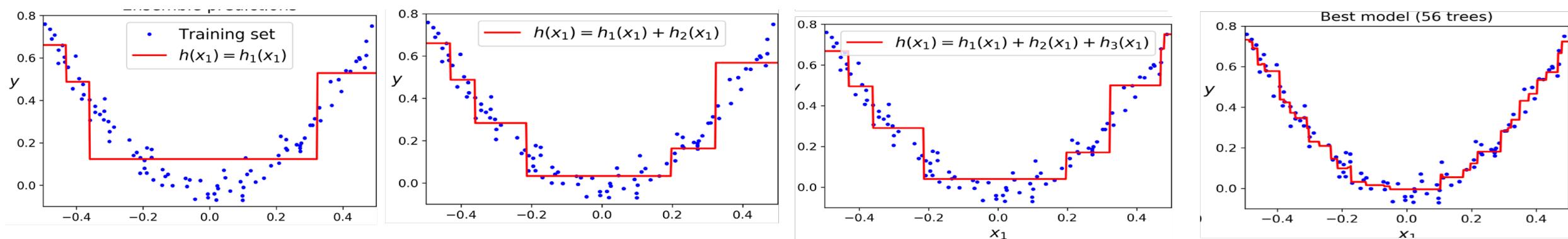
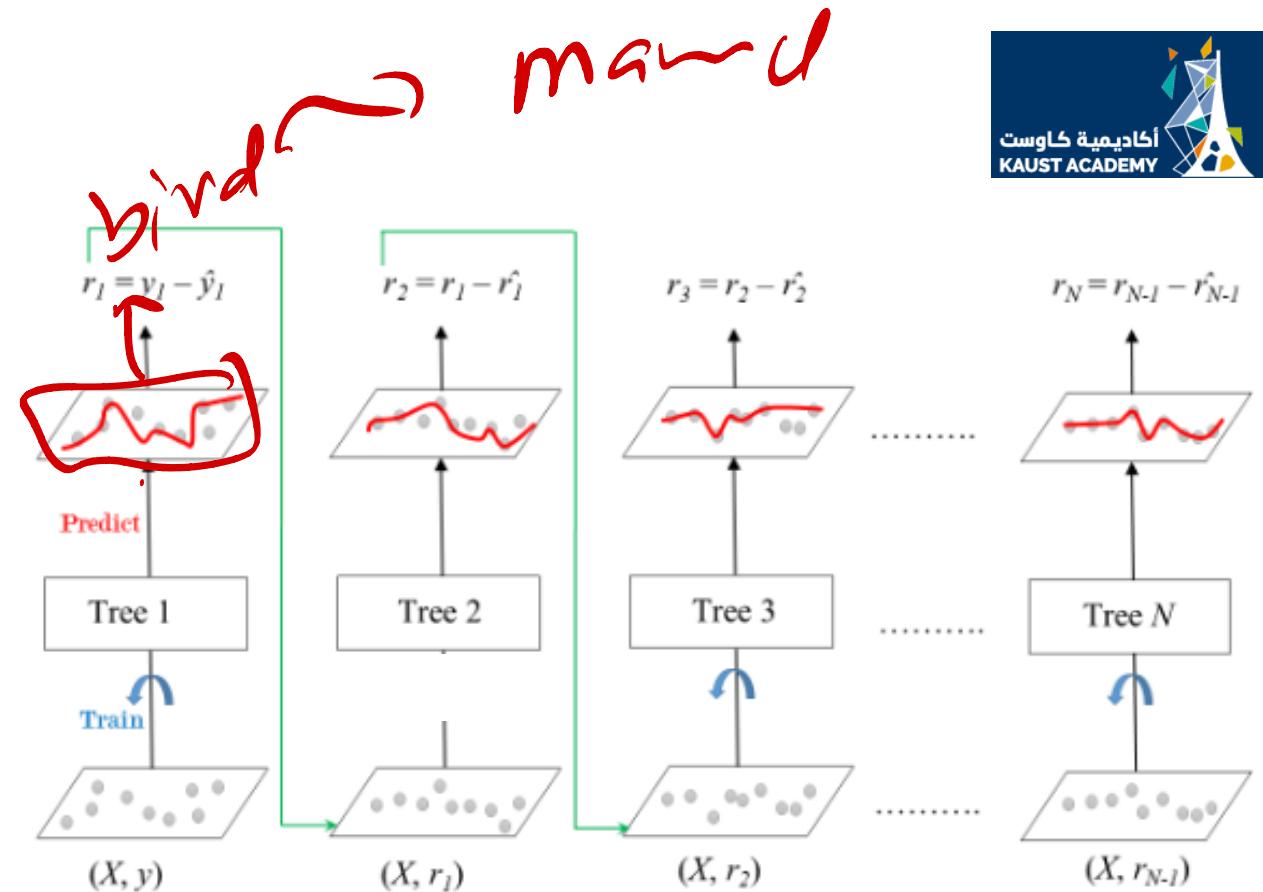
Other ML Models

- **Random Forest:**
Instead of training one tree, let's train a forest :)
- **Key Idea:**
Training K decision trees independently, where each one is fed with a different subset of samples and features.
- Then average the results (Regression) or Take the mode (Classification).
- Powerful model.



Other ML Models

- **Gradient Boosting:**
- **Key Idea:**
- Training K decision trees dependently, where each tree corrects the errors of the previous one.
- Many versions: XGBoost, LightGBM, CatBoost,...etc
- Top ML models.



Artificial Intelligence and Machine Learning

Neural Networks

Lecture Outline

- Logistic Regression Review
- Neural Networks
 - Forward pass
 - Backward pass

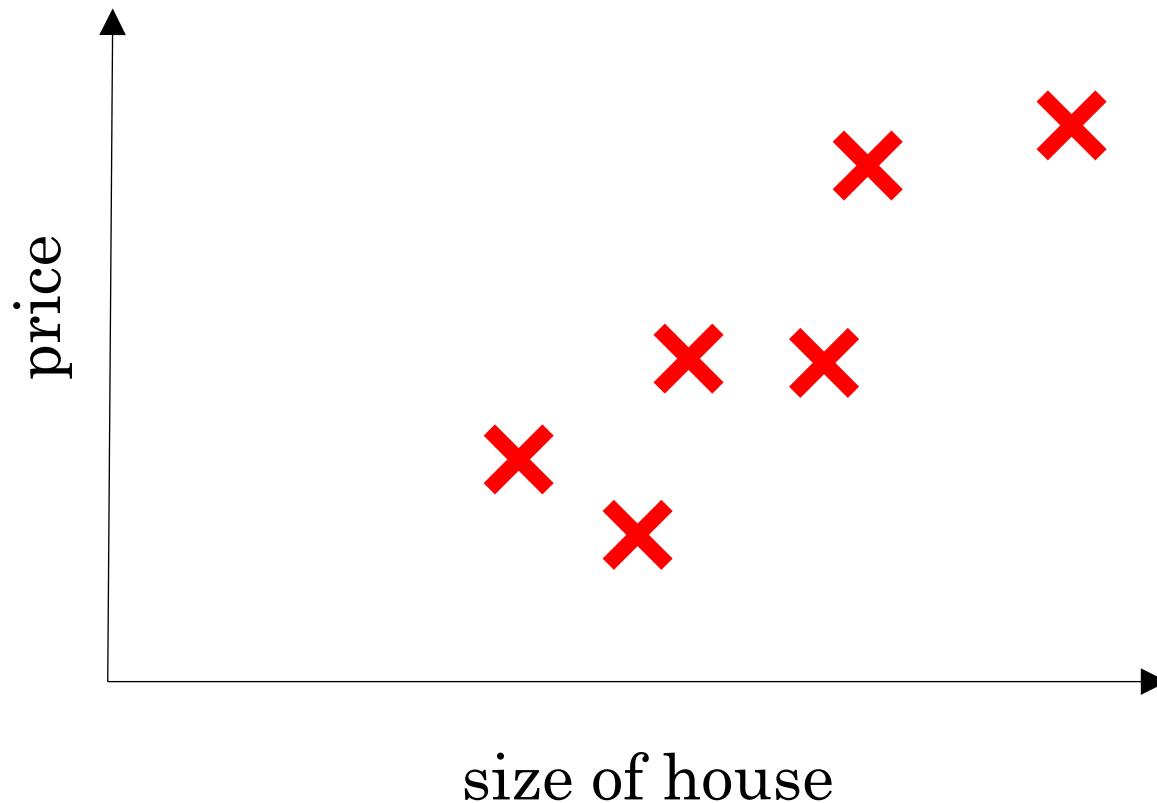


deeplearning.ai

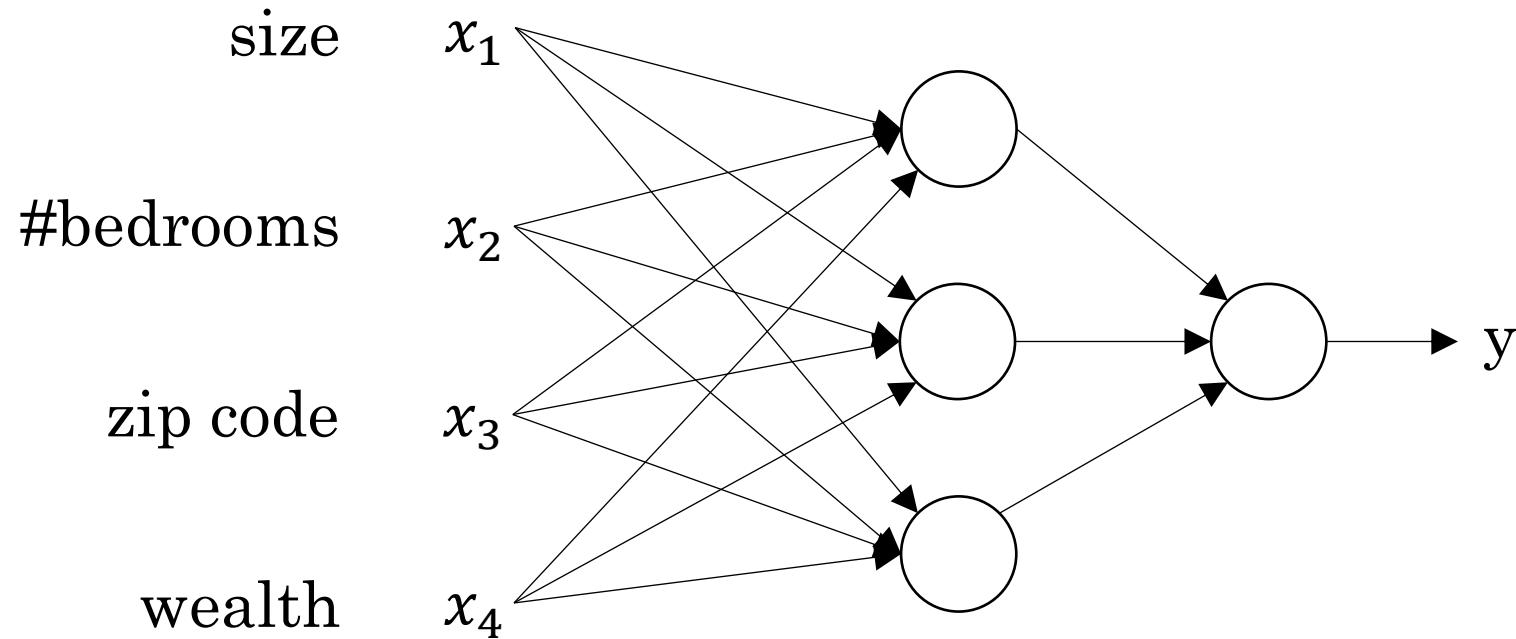
Introduction to Deep Learning

What is a Neural Network?

Housing Price Prediction



Housing Price Prediction





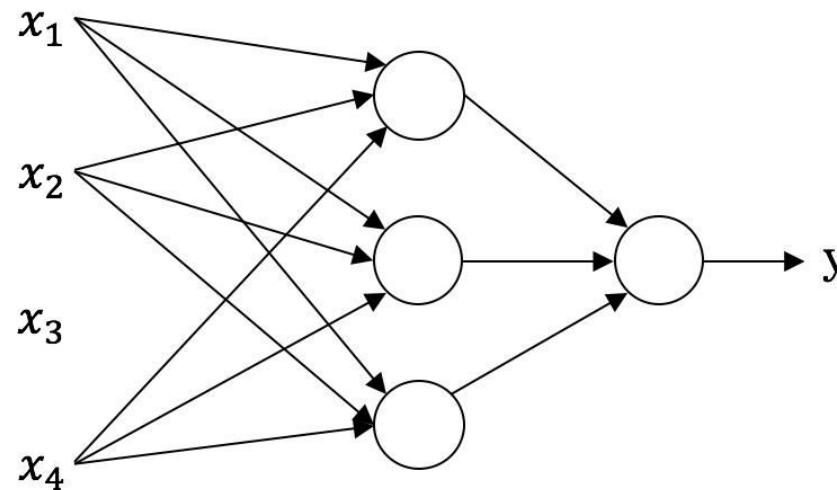
Introduction to Deep Learning

Supervised Learning with Neural Networks

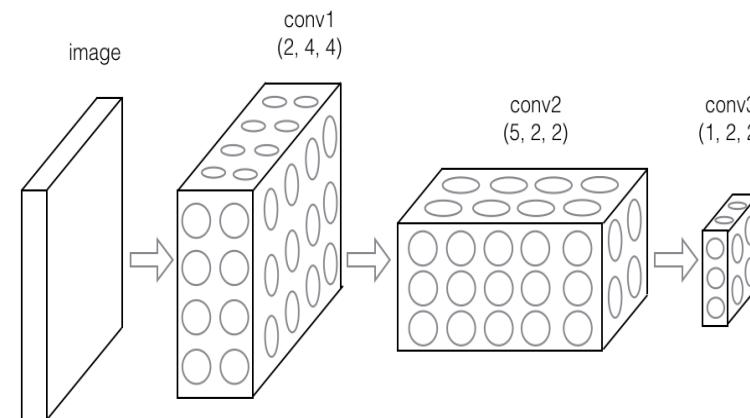
Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

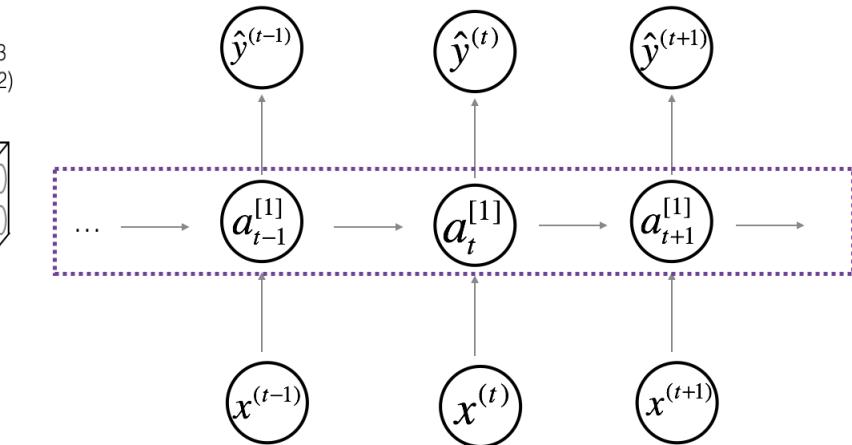
Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

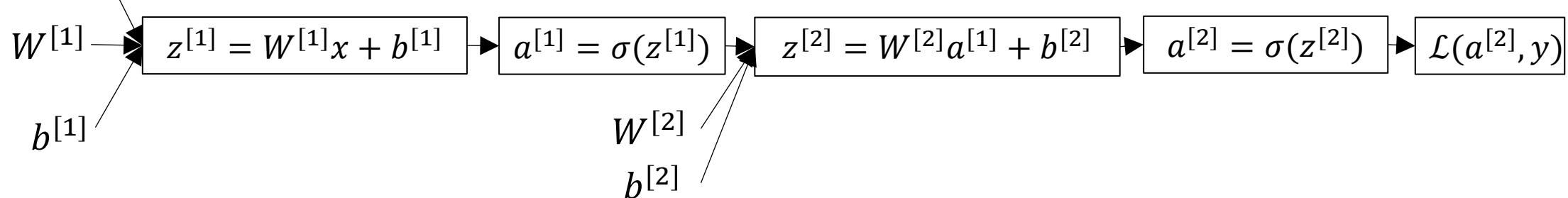
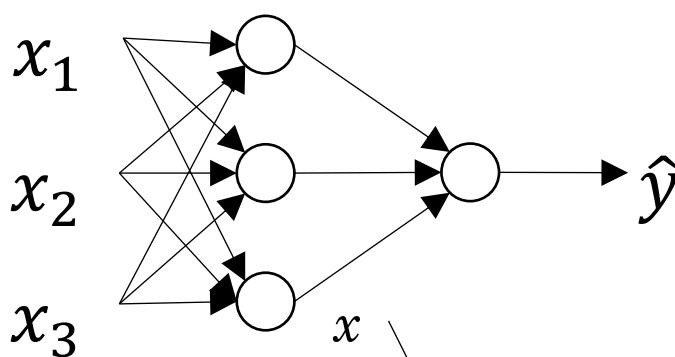
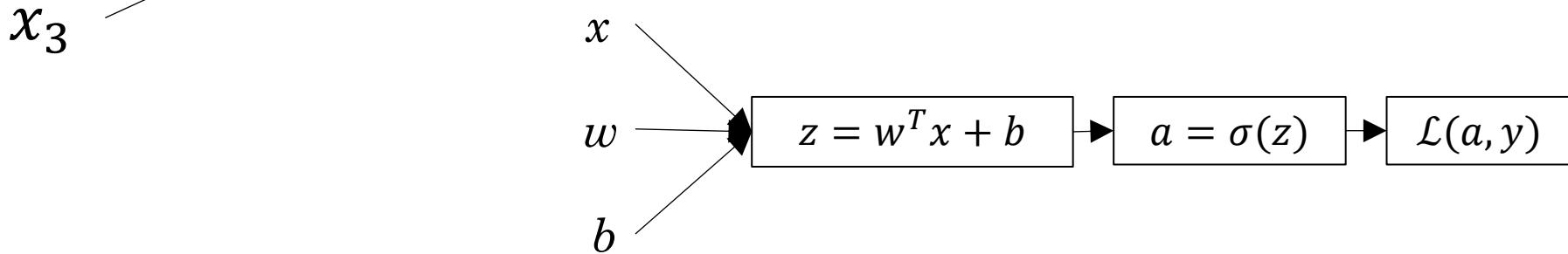
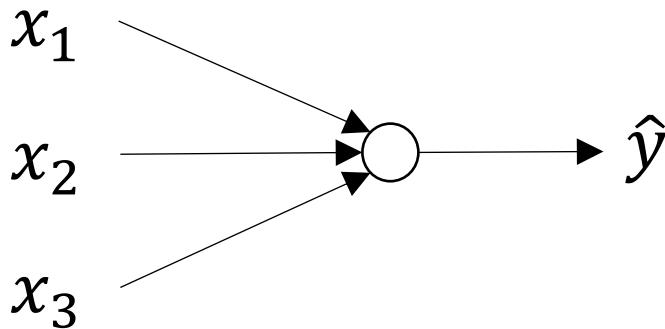


deeplearning.ai

One hidden layer Neural Network

Neural Networks Overview

What is a Neural Network?



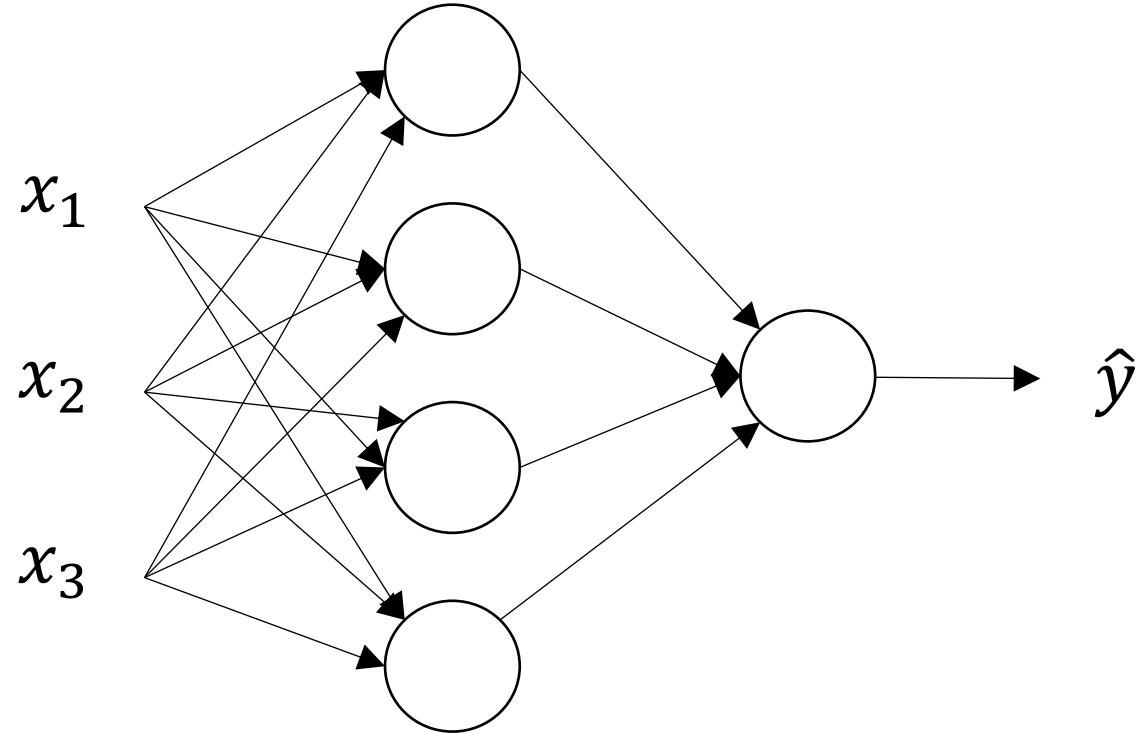


deeplearning.ai

One hidden layer Neural Network

Neural Network Representation

Neural Network Representation



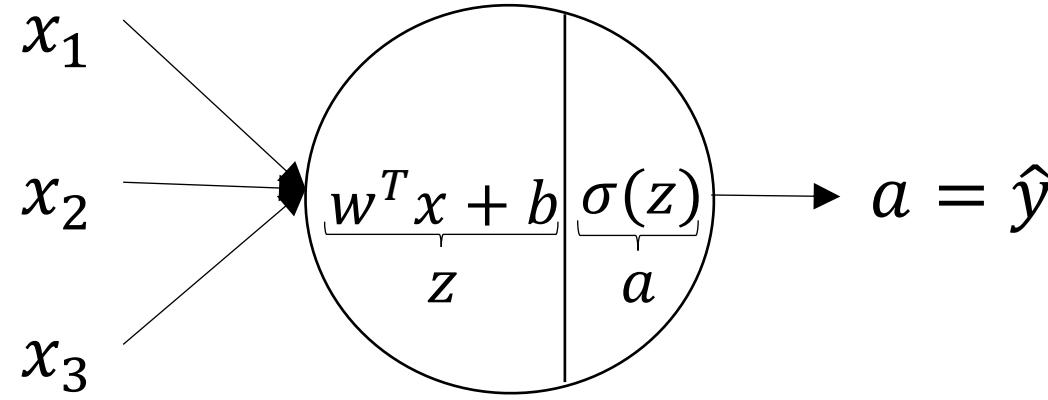


deeplearning.ai

One hidden layer Neural Network

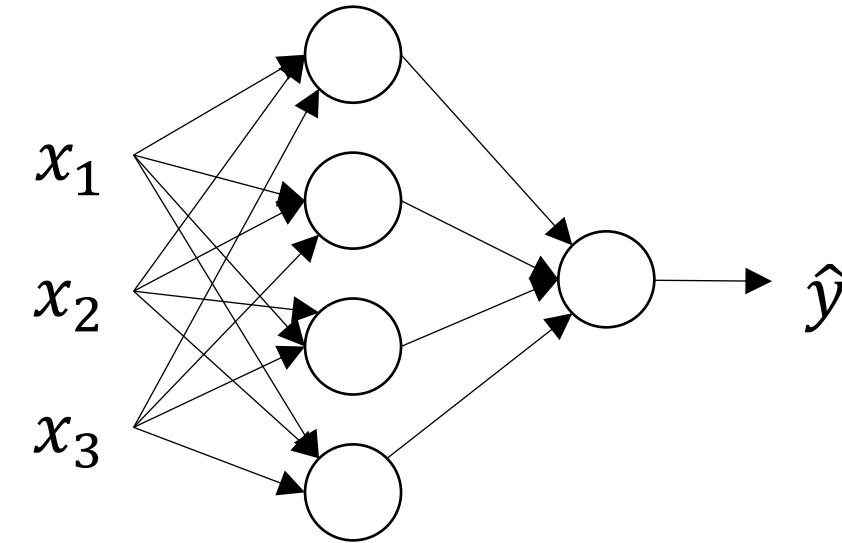
Computing a Neural Network's Output

Neural Network Representation

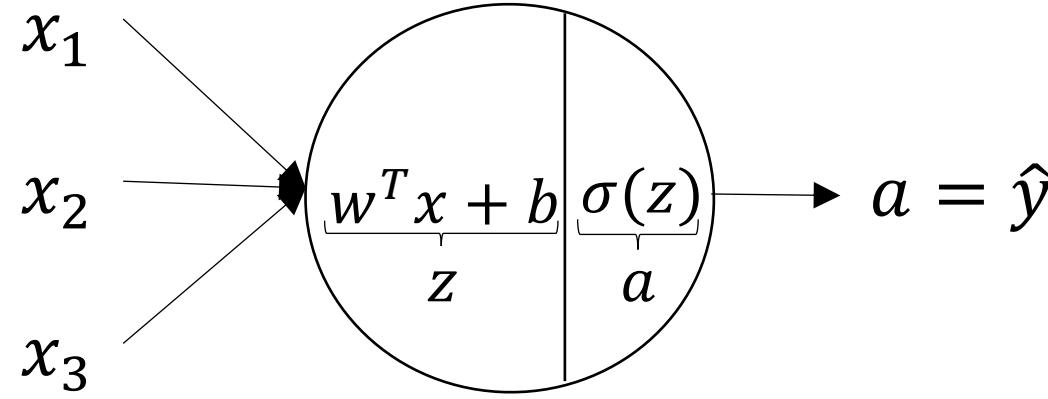


$$z = w^T x + b$$

$$a = \sigma(z)$$

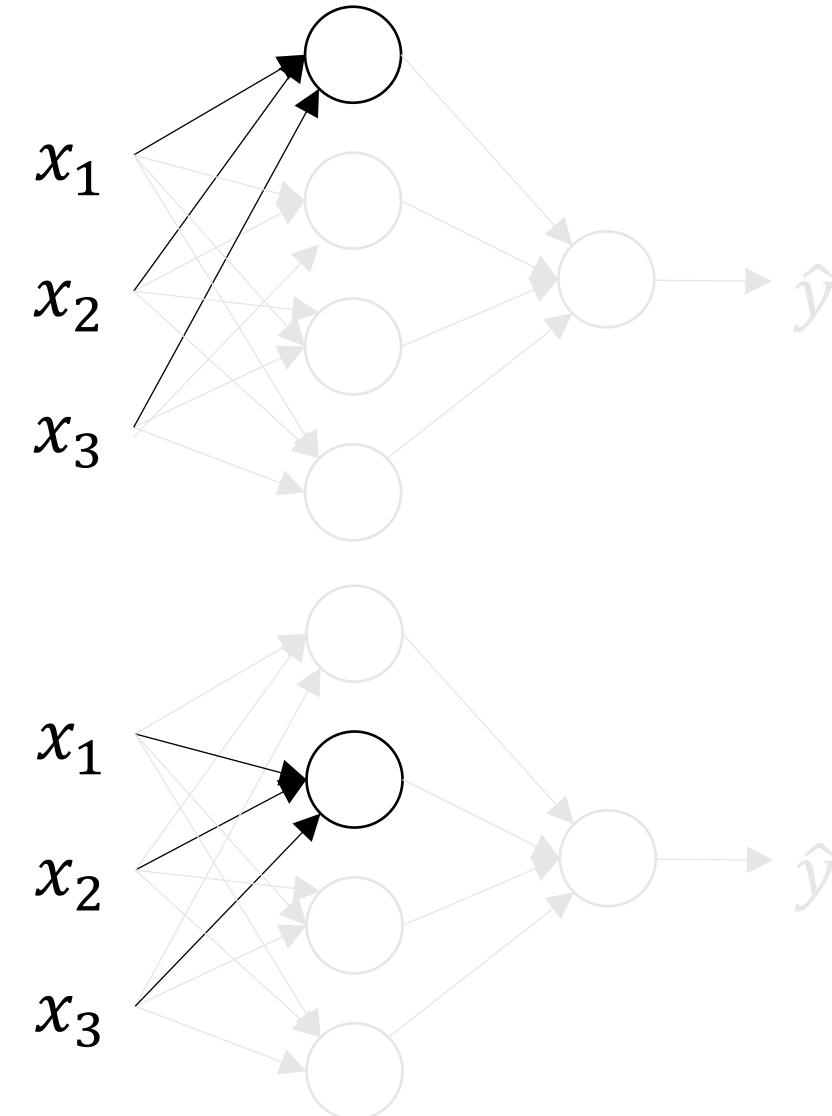


Neural Network Representation

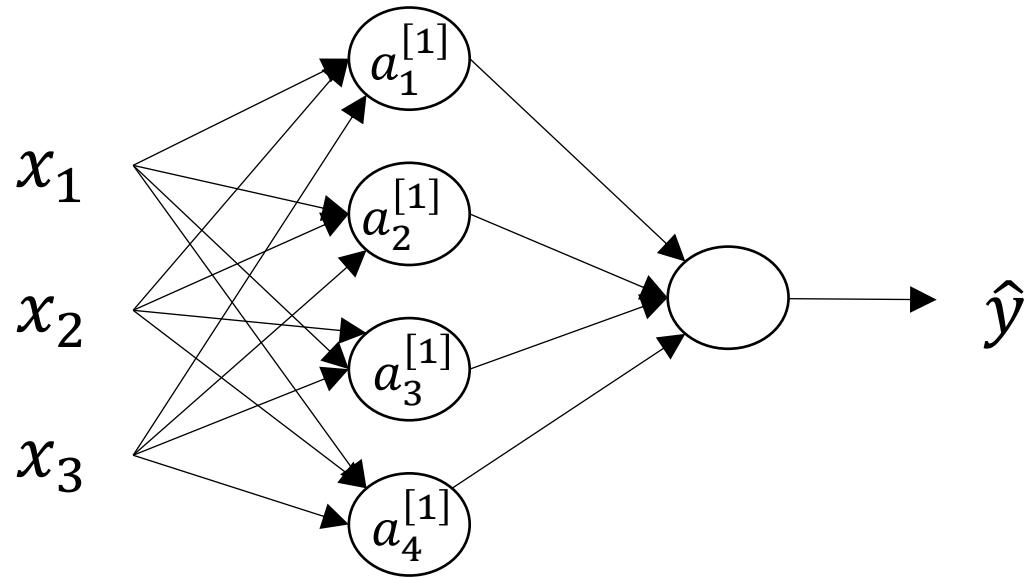


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



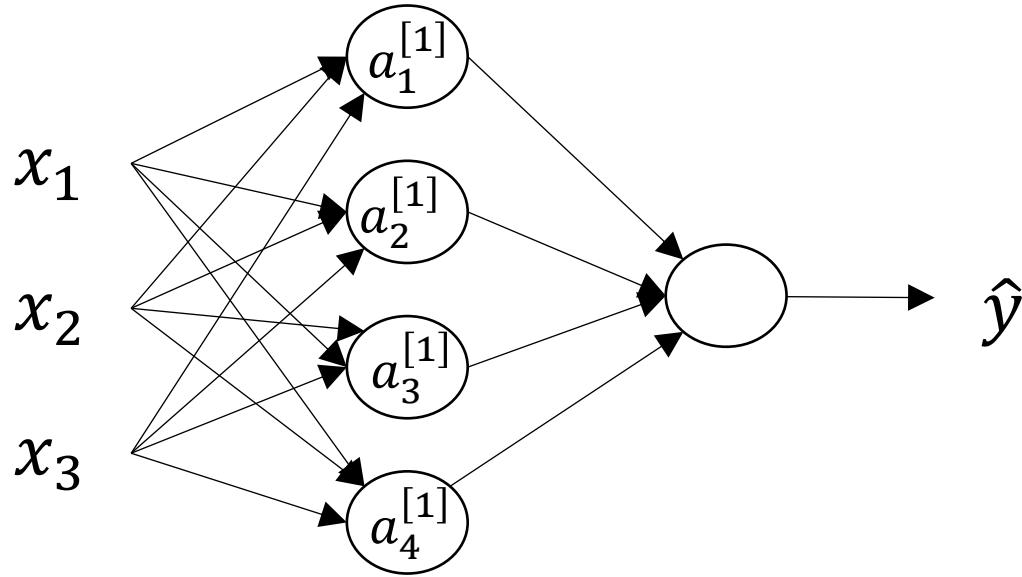
$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]}, \quad a_4^{[1]} = \sigma(z_4^{[1]})$$

Neural Network Representation learning



Given input x :

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

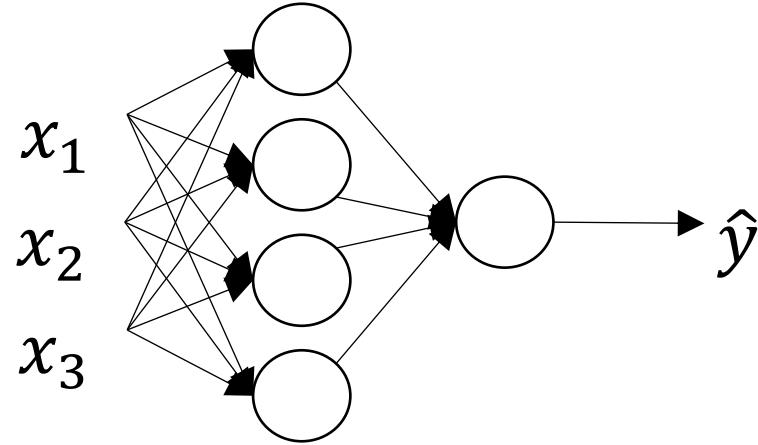


deeplearning.ai

One hidden layer Neural Network

Vectorizing across multiple examples

Vectorizing across multiple examples



$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Vectorizing across multiple examples

for i = 1 to m:

$$z^{[1]}(i) = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1]}(i) = \sigma(z^{[1]}(i))$$

$$z^{[2]}(i) = W^{[2]}a^{[1]}(i) + b^{[2]}$$

$$a^{[2]}(i) = \sigma(z^{[2]}(i))$$

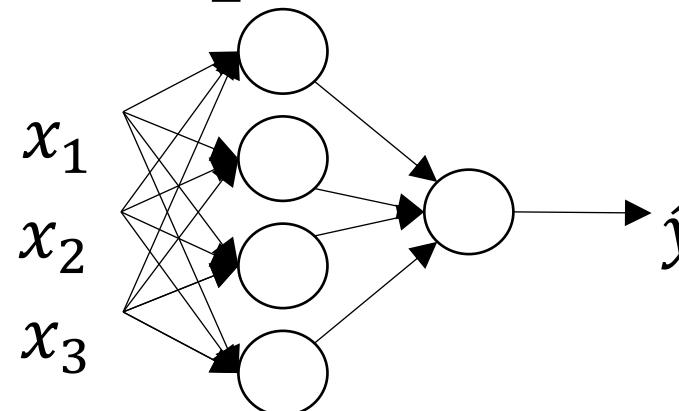


deeplearning.ai

One hidden layer Neural Network

Explanation for vectorized implementation

Recap of vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} | & | & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

for i = 1 to m

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

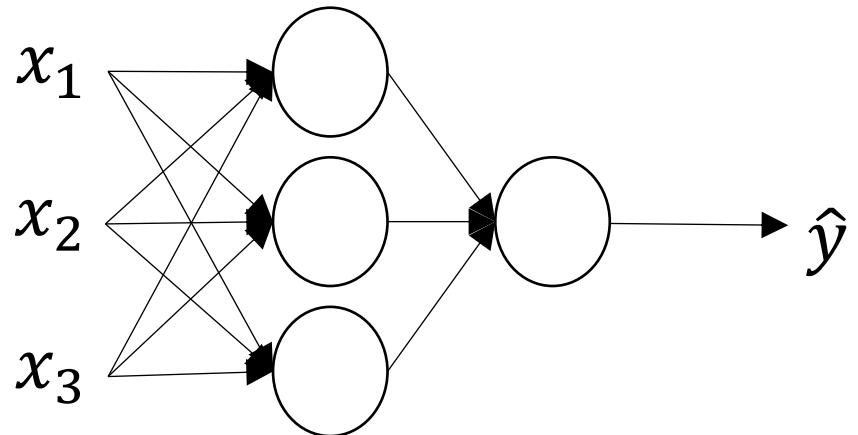


deeplearning.ai

One hidden layer Neural Network

Activation functions

Activation functions



Given x :

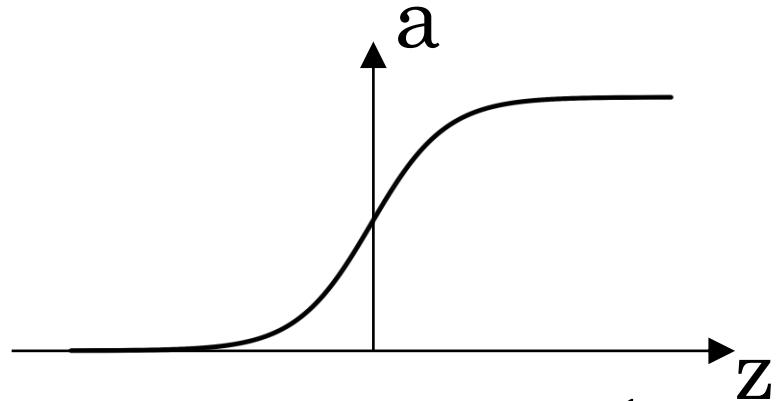
$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = \sigma(z^{[1]})$$

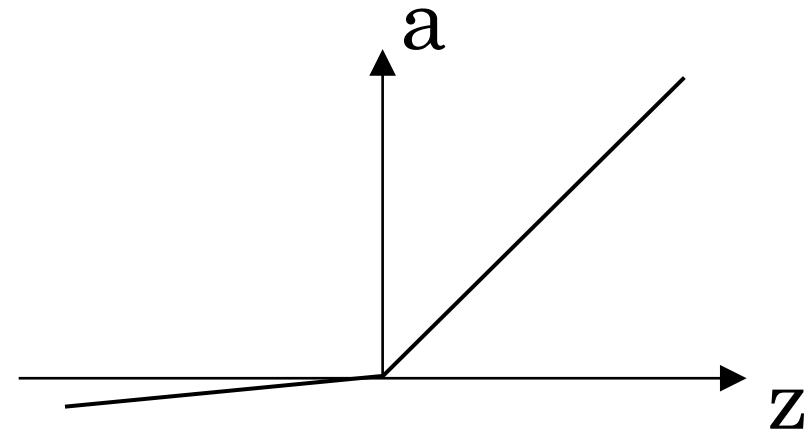
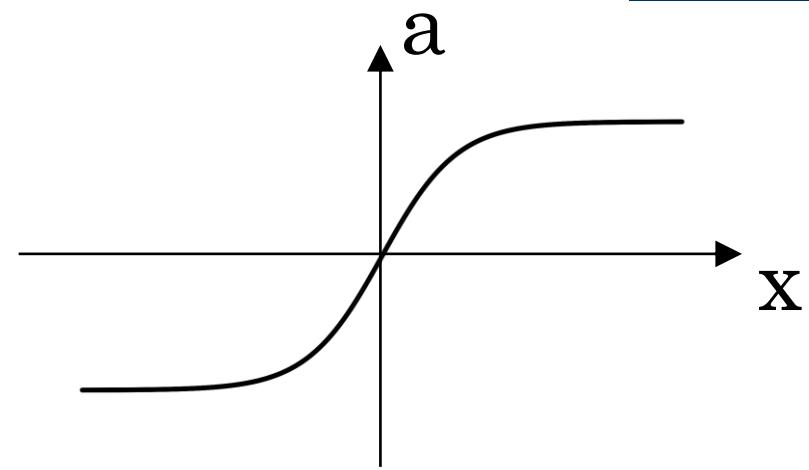
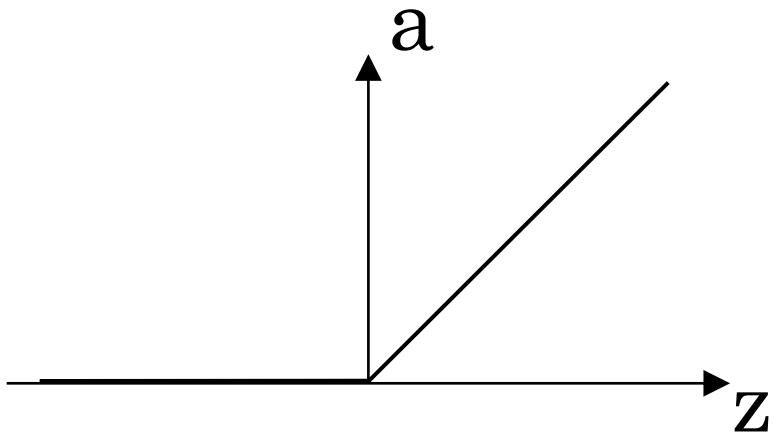
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]})$$

Pros and cons of activation functions



$$\text{sigmoid: } a = \frac{1}{1 + e^{-z}}$$



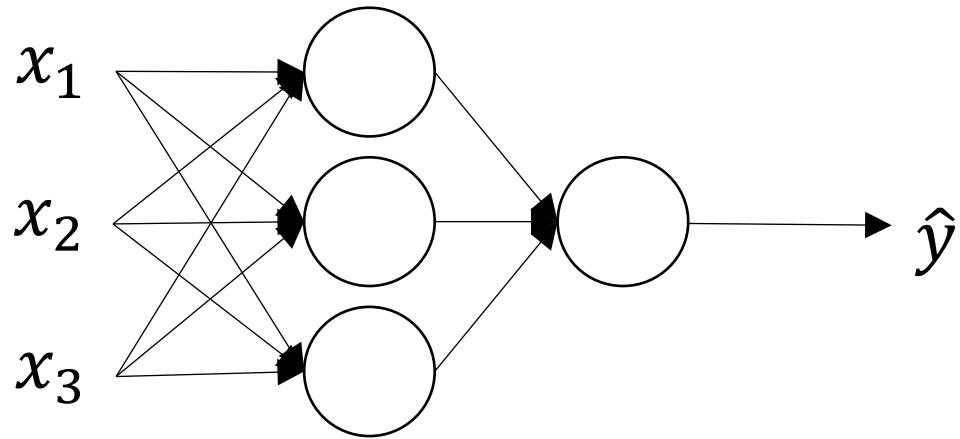


deeplearning.ai

One hidden layer Neural Network

Why do you
need non-linear
activation functions?

Activation function



Given \mathbf{x} :

$$\mathbf{z}^{[1]} = \mathbf{W}^{[1]}\mathbf{x} + \mathbf{b}^{[1]}$$

$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

$$\mathbf{z}^{[2]} = \mathbf{W}^{[2]}\mathbf{a}^{[1]} + \mathbf{b}^{[2]}$$

$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]})$$



deeplearning.ai

One hidden layer Neural Network

Gradient descent for neural networks



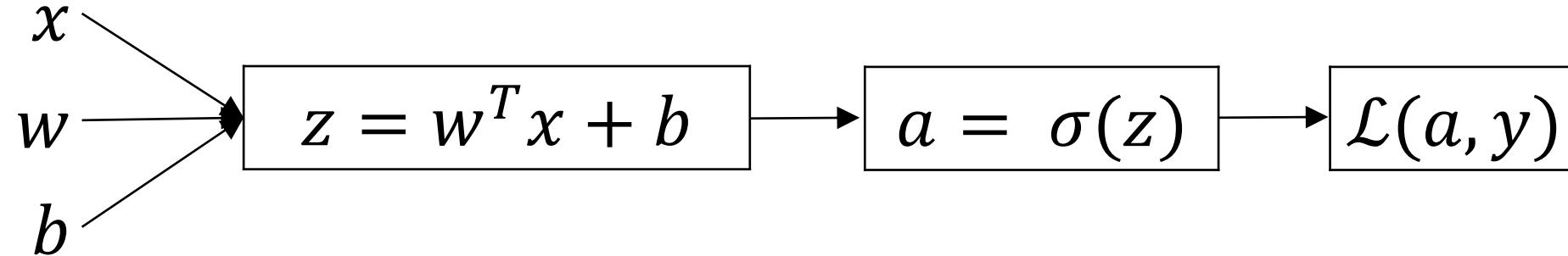
deeplearning.ai

One hidden layer Neural Network

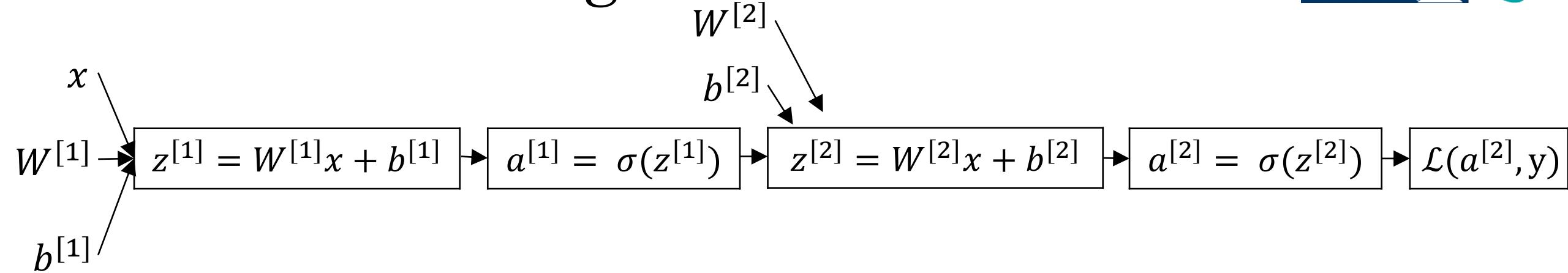
Backpropagation intuition

Computing gradients

Logistic regression



Neural network gradients



Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} \chi^T$$

$$db^{[1]} = dz^{[1]}$$

Summary of gradient descent

$$dz^{[2]} = a^{[2]} - y$$

$$dW^{[2]} = dz^{[2]} a^{[1]T}$$

$$db^{[2]} = dz^{[2]}$$

$$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$$

$$dW^{[1]} = dz^{[1]} \chi^T$$

$$db^{[1]} = dz^{[1]}$$

$$dZ^{[2]} = A^{[2]} - Y$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]T}$$

$$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$$

$$dZ^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(Z^{[1]})$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$$

$$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$$

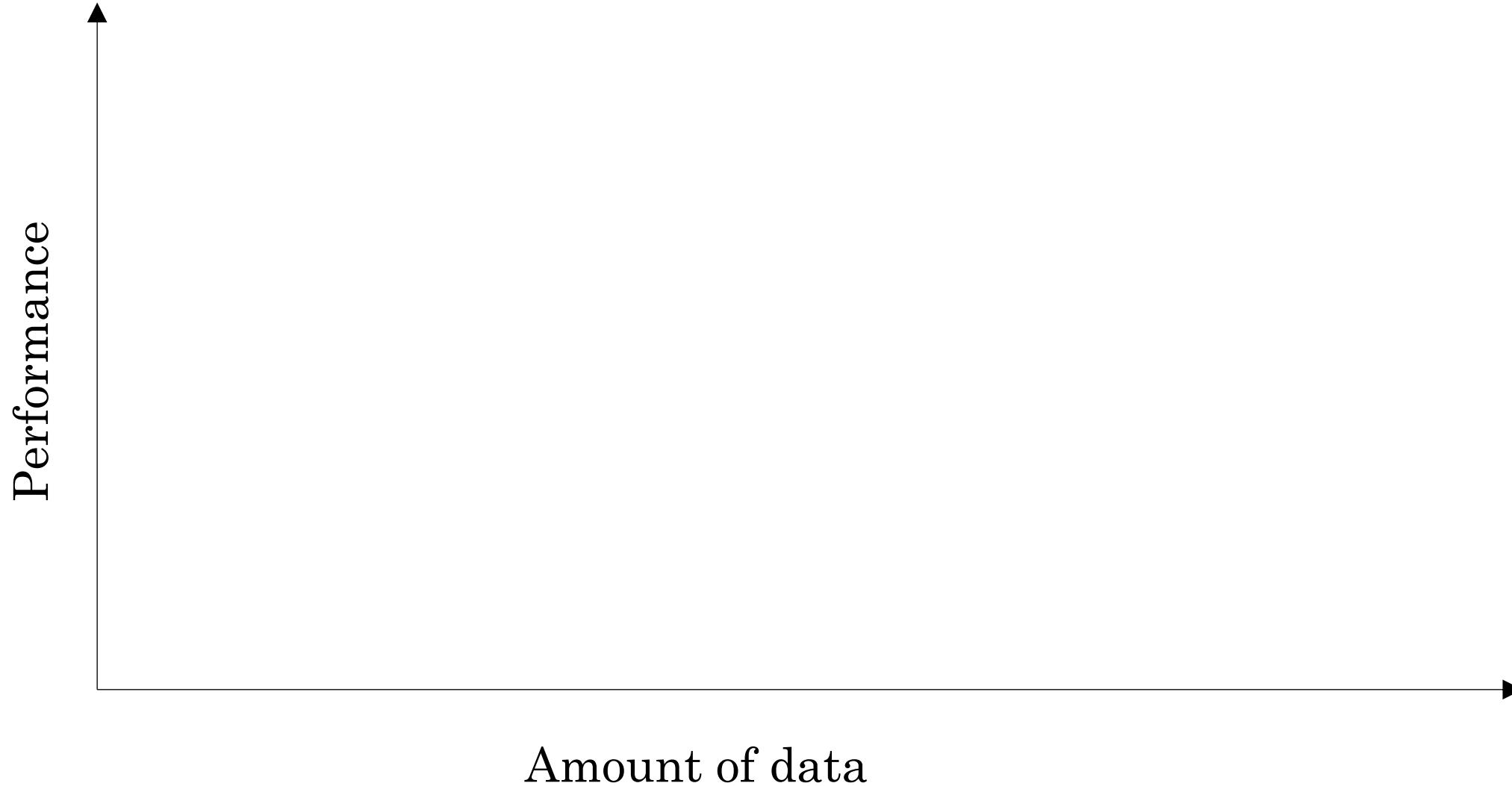


deeplearning.ai

Introduction to Neural Networks

Why is Deep Learning taking off?

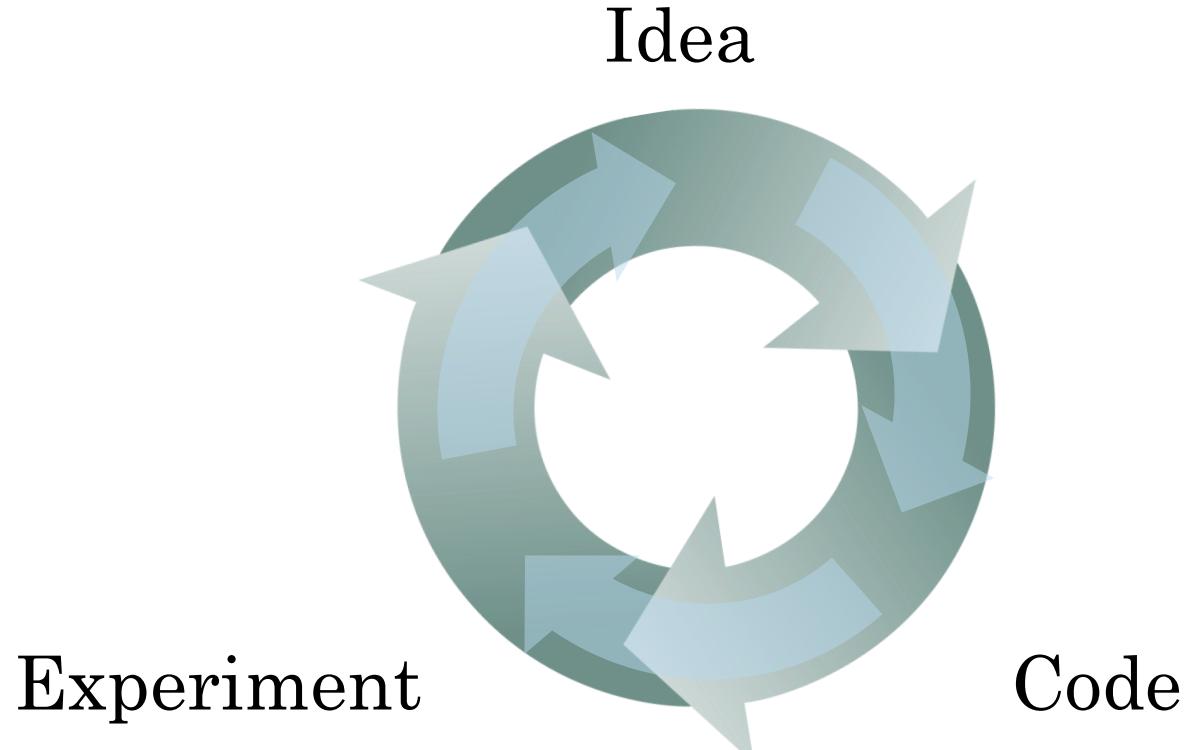
Scale drives deep learning progress



Scale drives deep learning progress



- Data
- Computation
- Algorithms



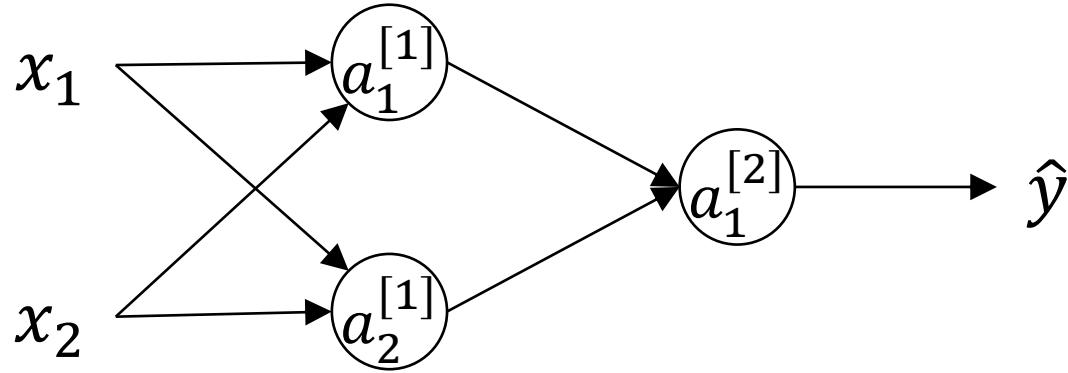


deeplearning.ai

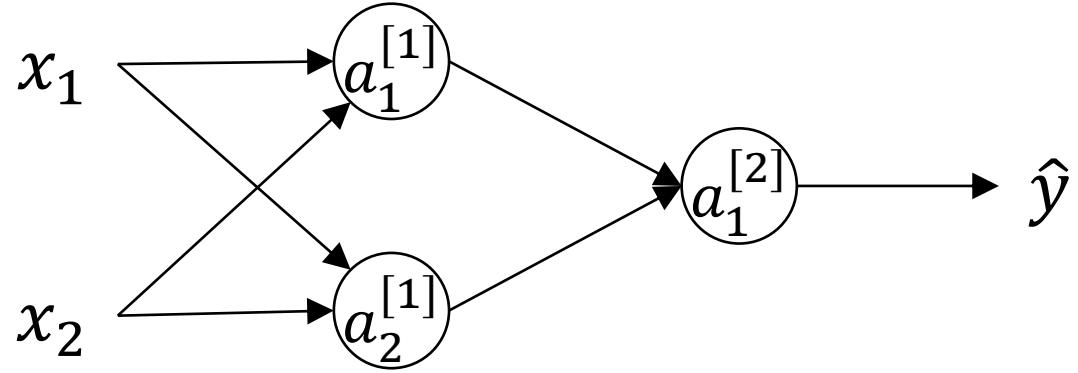
One hidden layer Neural Network

Random Initialization

What happens if you initialize weights to zero?



Random initialization



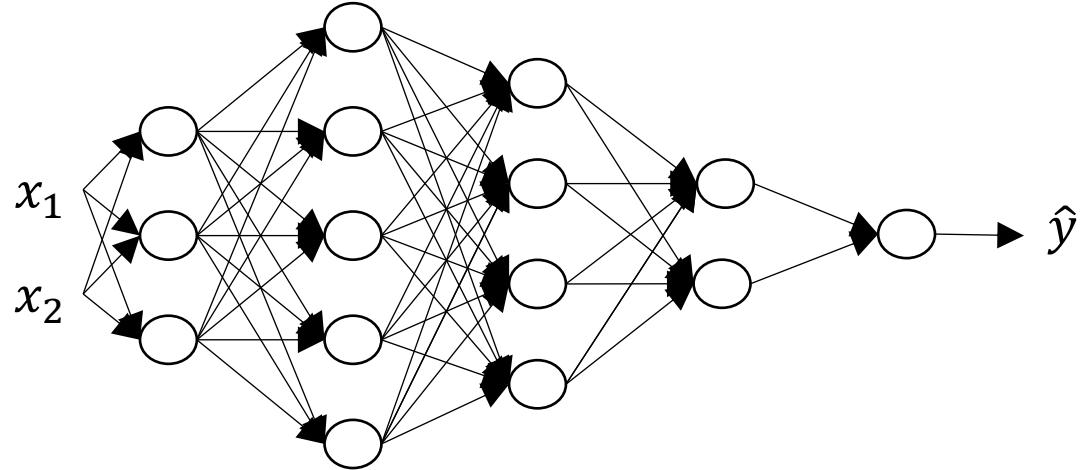


deeplearning.ai

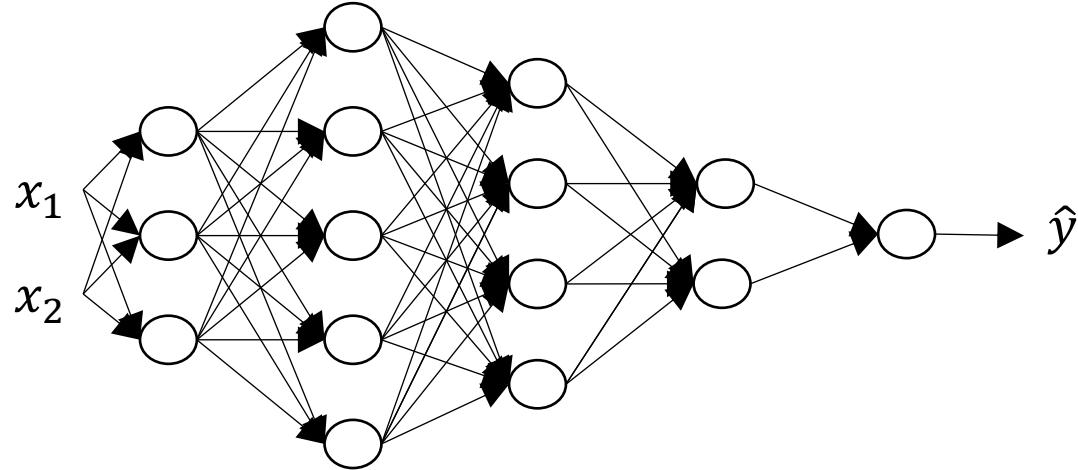
Deep Neural Networks

Getting your matrix dimensions right

Parameters $W^{[l]}$ and $b^{[l]}$



Vectorized implementation



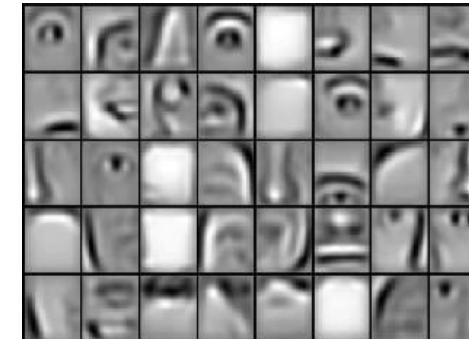
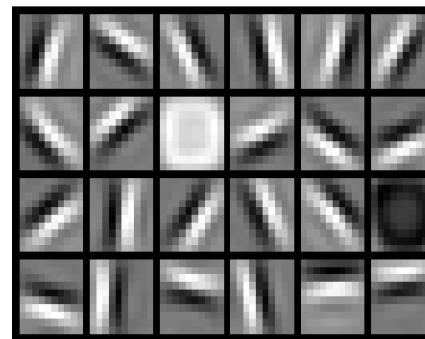
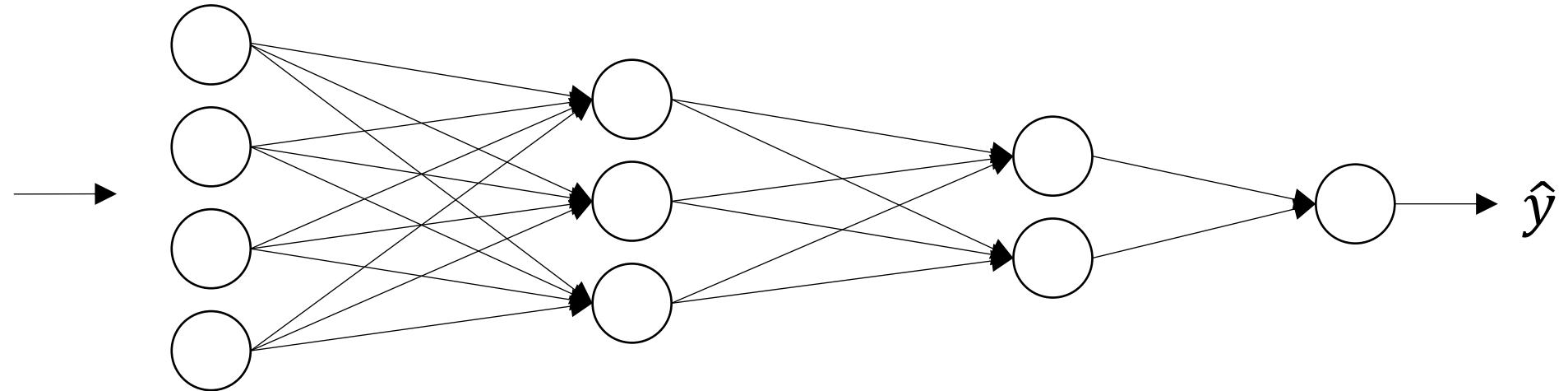


deeplearning.ai

Deep Neural Networks

Why deep representations?

Intuition about deep representation



Circuit theory and deep learning

Informally: There are functions you can compute with a “small” L-layer deep neural network that shallower networks require exponentially more hidden units to compute.

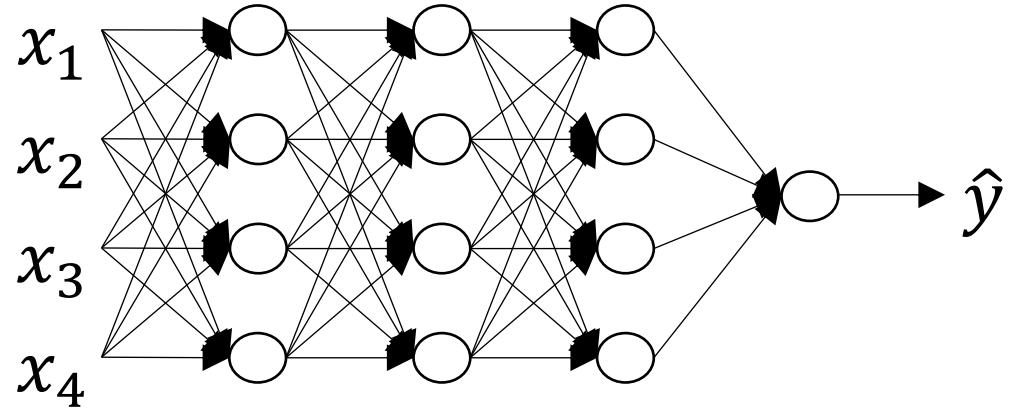


deeplearning.ai

Deep Neural Networks

Building blocks of deep neural networks

Forward and backward functions





deeplearning.ai

Deep Neural Networks

Forward and backward propagation

Forward propagation for layer l

Input $a^{[l-1]}$

Output $a^{[l]}$, cache ($z^{[l]}$)

Backward propagation for layer l

Input $da^{[l]}$

Output $da^{[l-1]}, dW^{[l]}, db^{[l]}$

Artificial Intelligence and Machine Learning

Applications

Lecture Outline

- Linear Regression Tutorial & Playground
- Image Compression
- Auto-Encoders
- Optimizers
- Logistic Regression Interactive Demo
- Neural Network Regression Tutorial & Playground

Image Compression with Linear Regression

10

$$X \rightarrow f(X) = X' \Rightarrow \text{CompFunc}(X') \rightarrow X_{\text{low}}$$

- An image can be transformed into the frequency domain and represented as a combination of some basic components.
- Cosine Basis Functions and Discrete Cosine Transforms (DCT) can enable this.
- The human eye is most sensitive to low frequencies. Therefore, most of the high frequencies can be ignored.
- Principal behind JPEG Image Compression.

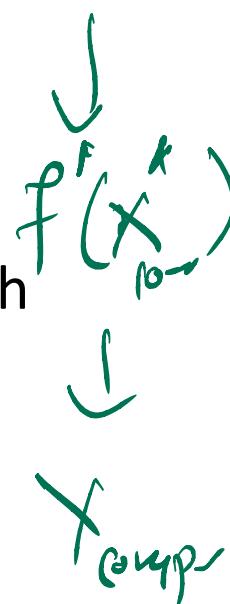
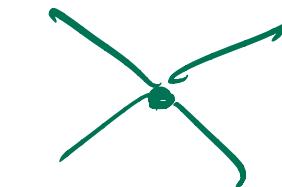
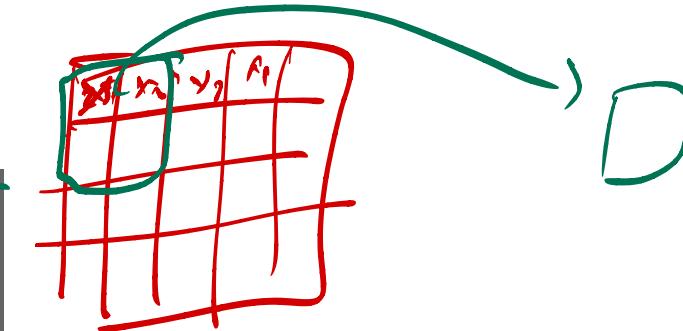
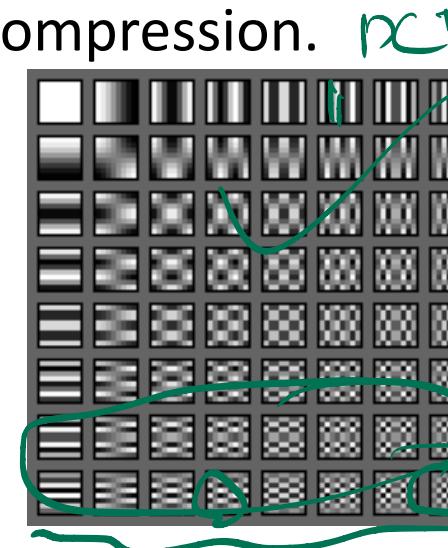
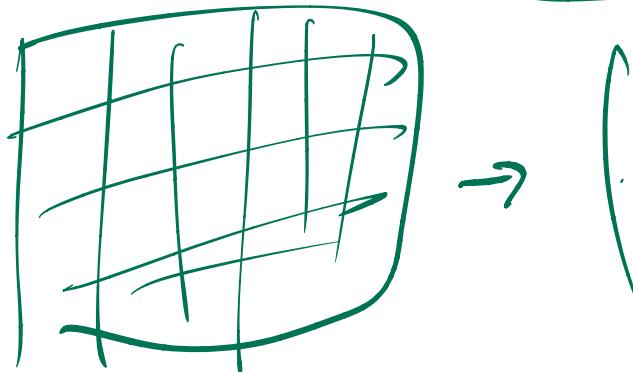


Image Compression with Linear Regression

- Using linear regression, we can learn the weight of each of these cosine basis.
- We only need to store the weights of the basis frequencies.
- The image is reconstructed using these weights.

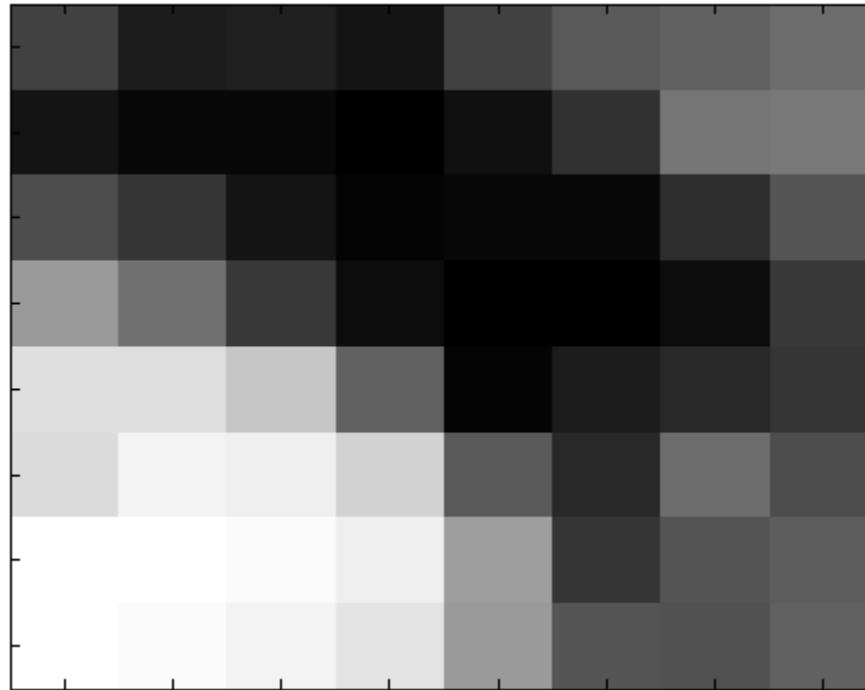
ـ ـ ـ

ـ ـ

ـ

Image Compression

8 x 8 Pixels



Image



Image Compression

- Gray-Scale Example:
- Value Range 0 (black) --- 255 (white)

63	33	36	28	63	81	86	98
27	18	17	11	22	48	104	108
72	52	28	15	17	16	47	77
132	100	56	19	10	9	21	55
187	186	166	88	13	34	43	51
184	203	199	177	82	44	97	73
211	214	208	198	134	52	78	83
211	210	203	191	133	79	74	86

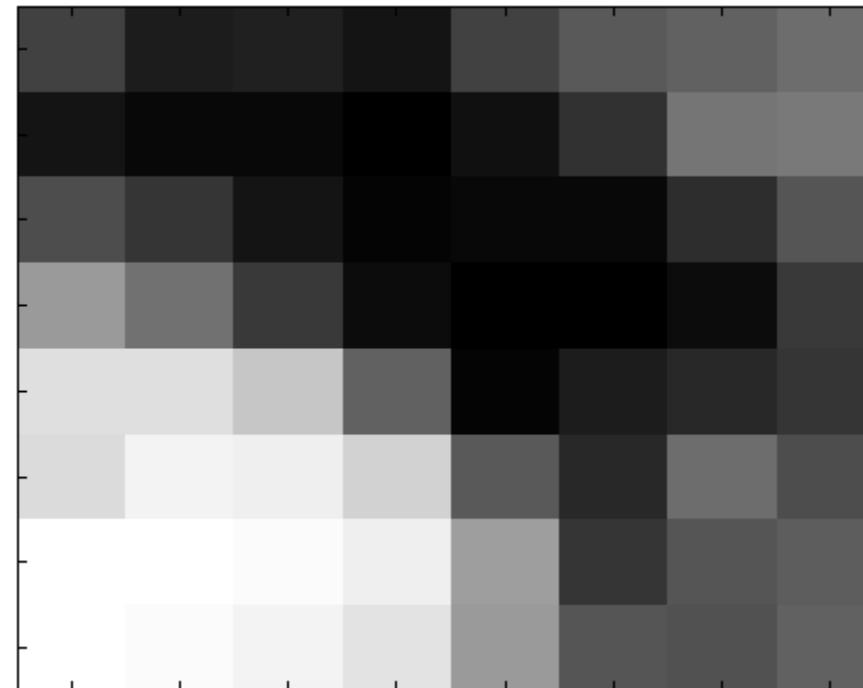


Image Compression

- **2D-DCT of matrix**

Numbers are coefficients
of polynomial

```
-304 210 104 -69 10 20 -12 7
-327 -260 67 70 -10 -15 21 8
 93 -84 -66 16 24 -2 -5 9
 89  33 -19 -20 -26 21 -3 0
 -9  42 18 27 -7 -17 29 -7
 -5  15 -10 17 32 -15 -4 7
 10   3 -12 -1  2  3 -2 -3
 12  30  0 -3 -3 -6 12 -1
```



Image Compression

- Cut the least significant components

-304 210 104 -69 10 20 -12 0
-327 -260 67 70 -10 -15 0 0
93 -84 -66 16 24 0 0 0
89 33 -19 -20 0 0 0 0
-9 42 18 0 0 0 0 0
-5 15 0 0 0 0 0 0
10 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

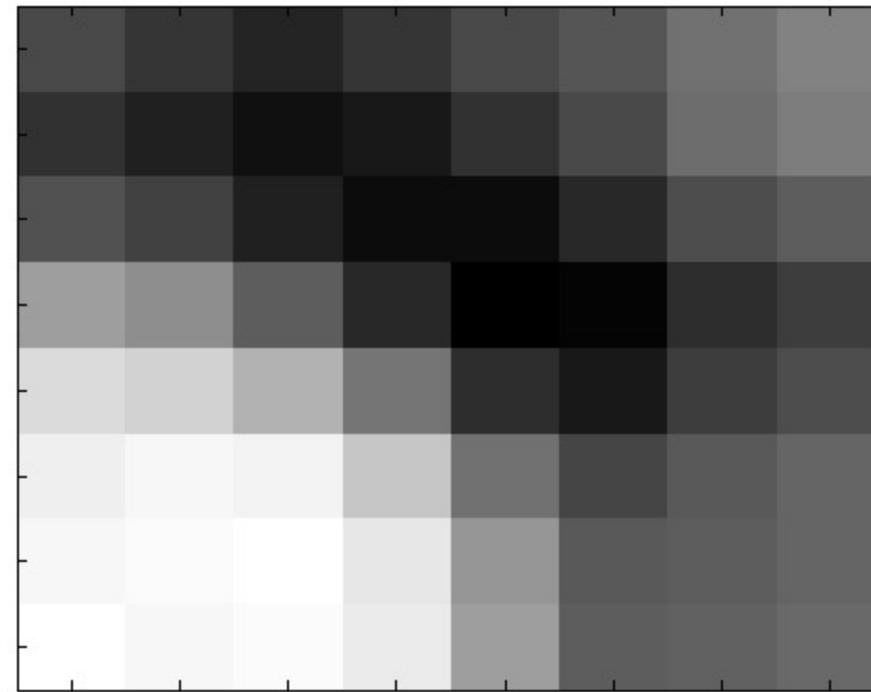


As you can see, we save a little over half the original memory.

Reconstructing the Image

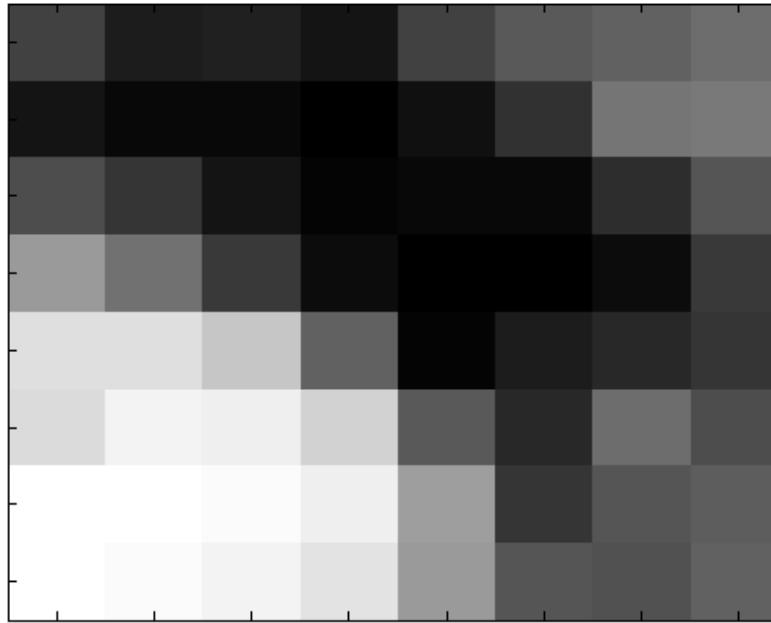
- New Matrix and Compressed Image

55	41	27	39	56	69	92	106
35	22	7	16	35	59	88	101
65	49	21	5	6	28	62	73
130	114	75	28	-7	-1	33	46
180	175	148	95	33	16	45	59
200	206	203	165	92	55	71	82
205	207	214	193	121	70	75	83
214	205	209	196	129	75	78	85



Can You Tell the Difference?

Original



Compressed

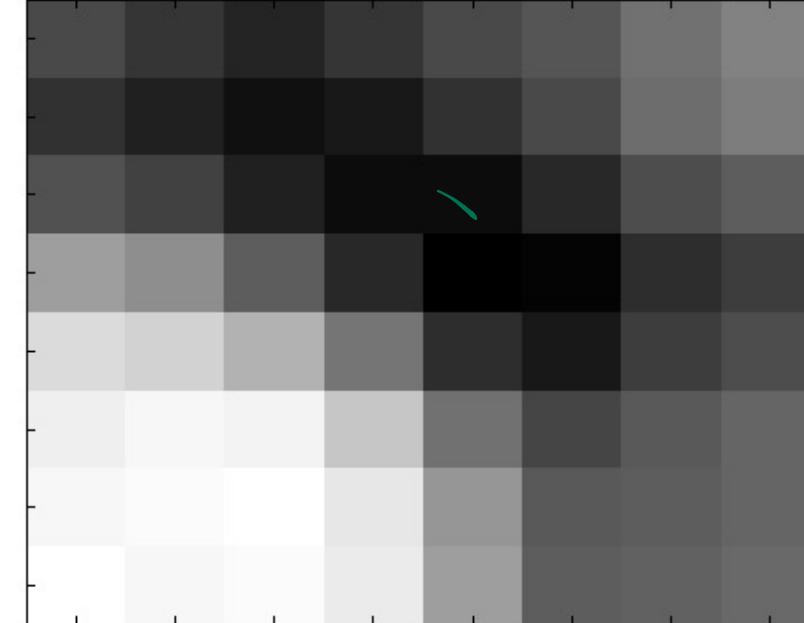


Image Compression

Original



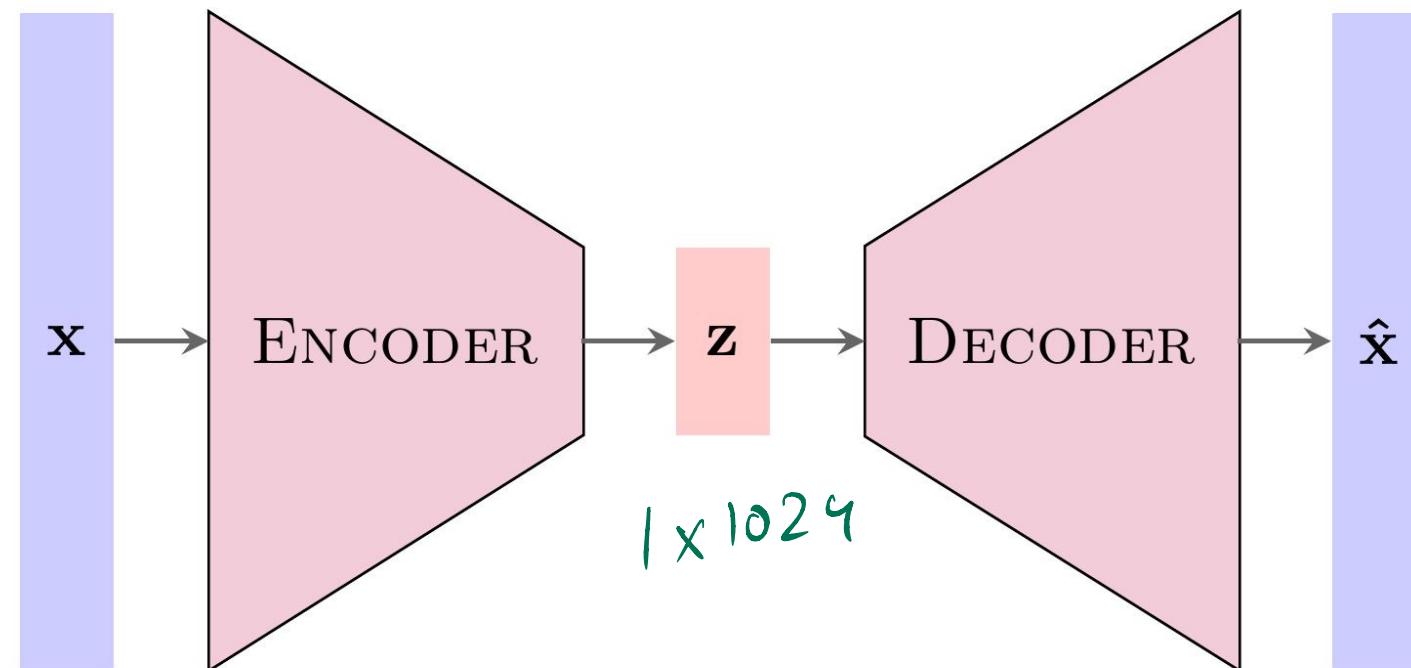
Compressed



Applications – AutoEncoders

- Autoencoders are a type of neural networks where the input is also the output.
- They come under unsupervised learning and there are no labels involved.
- An autoencoder consists of two parts: encoder and decoder.
- The idea here is that you take a higher dimensional input, project it into a lower dimensional space and then project it back into the input space.

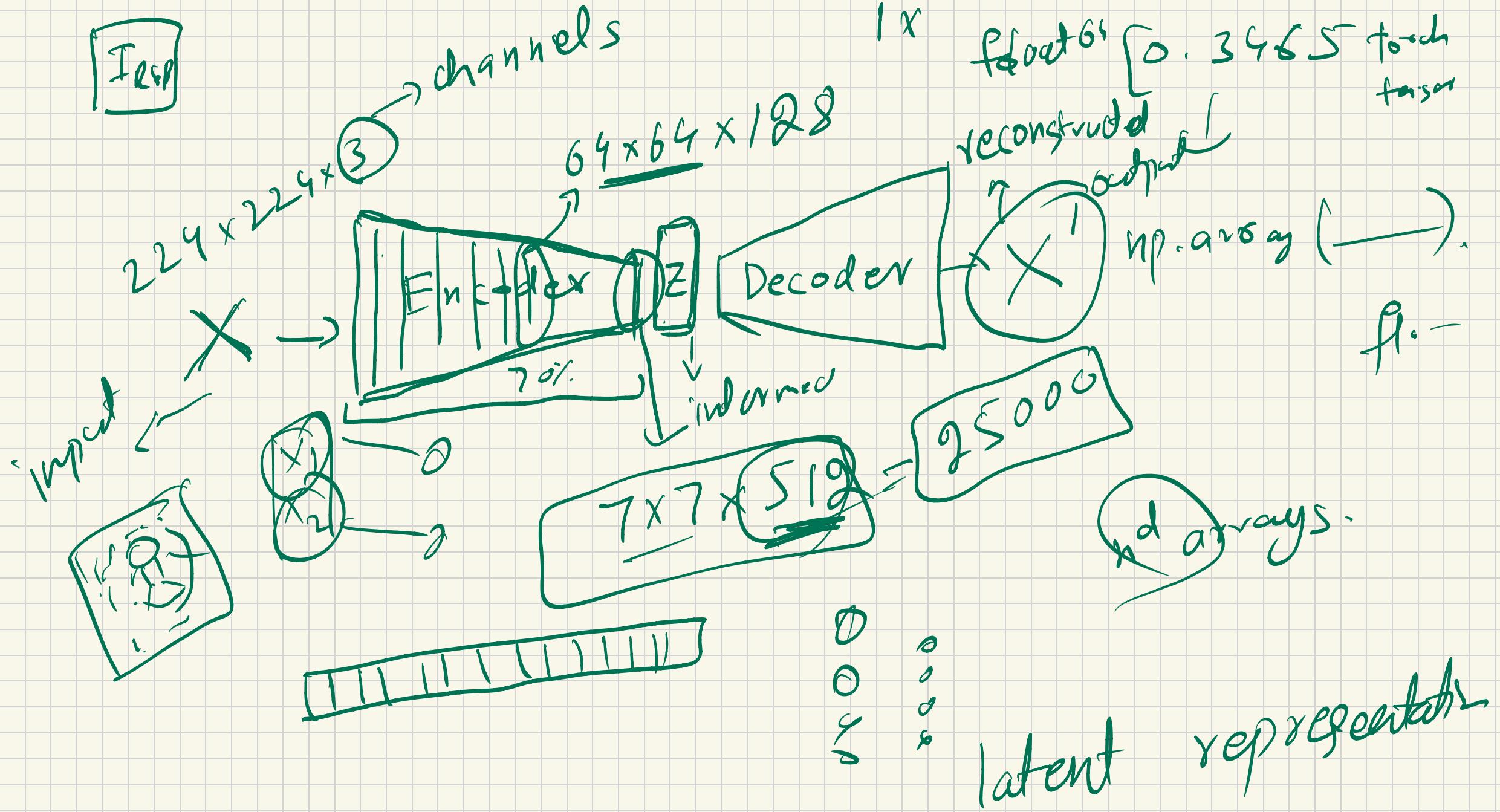
Applications – AutoEncoders

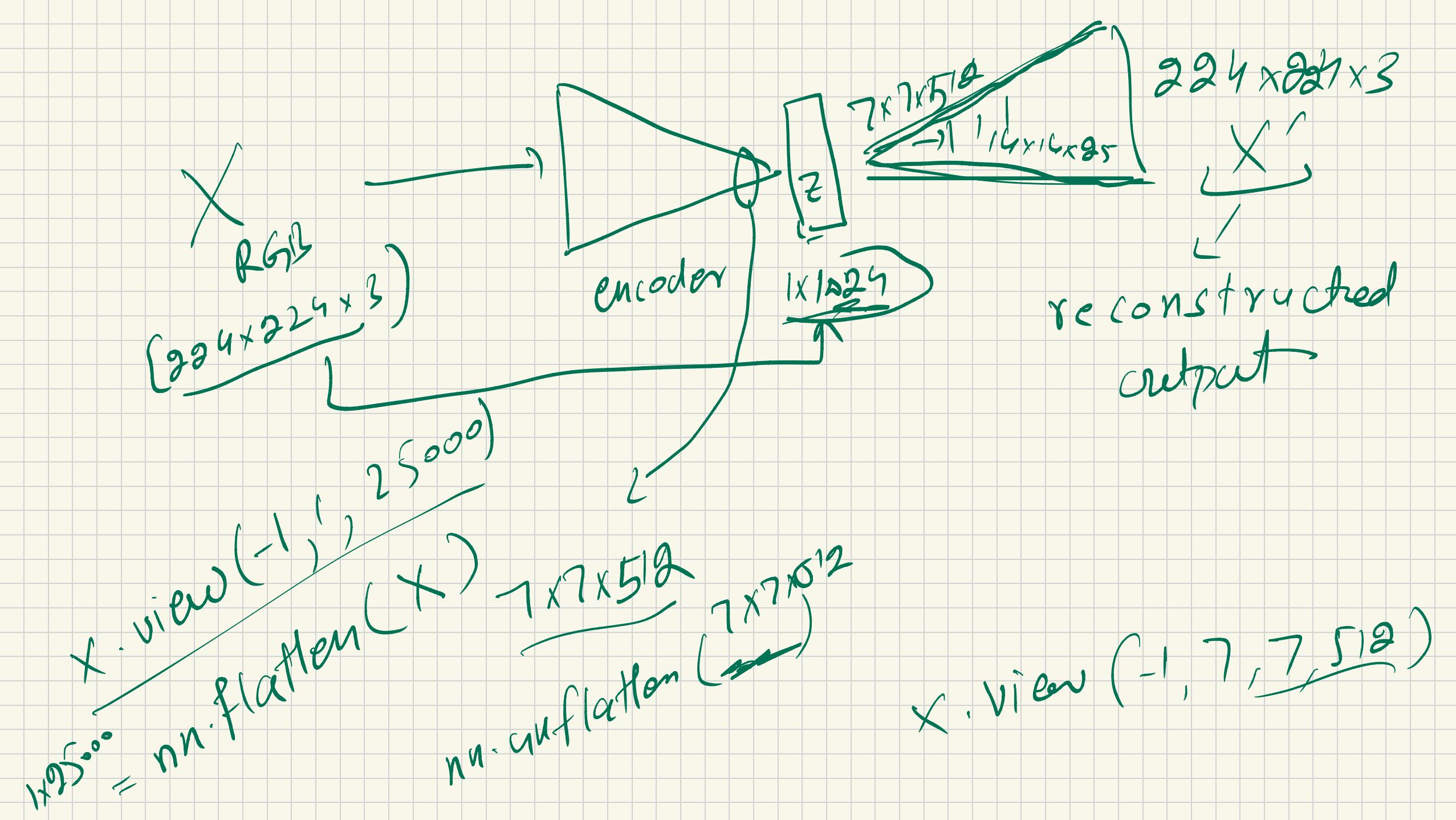


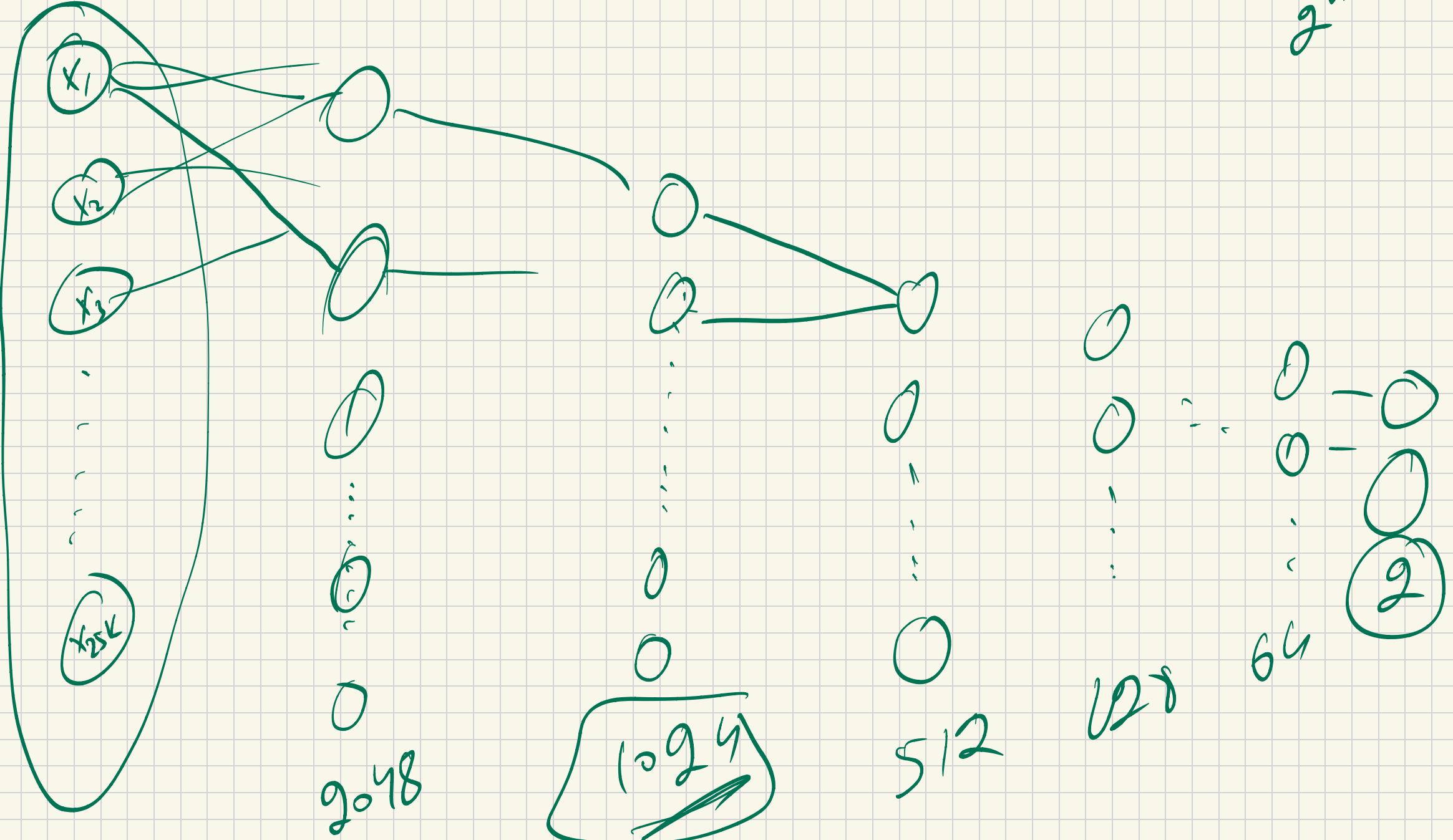
Applications – AutoEncoders

- The autoencoder model tries to minimize the reconstruction error (RE).
- Typically, mean squared is used as the loss function for autoencoders.
- The objective is to minimize the following:

$$L(x, \hat{x}) = \frac{1}{N} \sum_{i=1}^N ||x_i - \hat{x}_i||^2$$

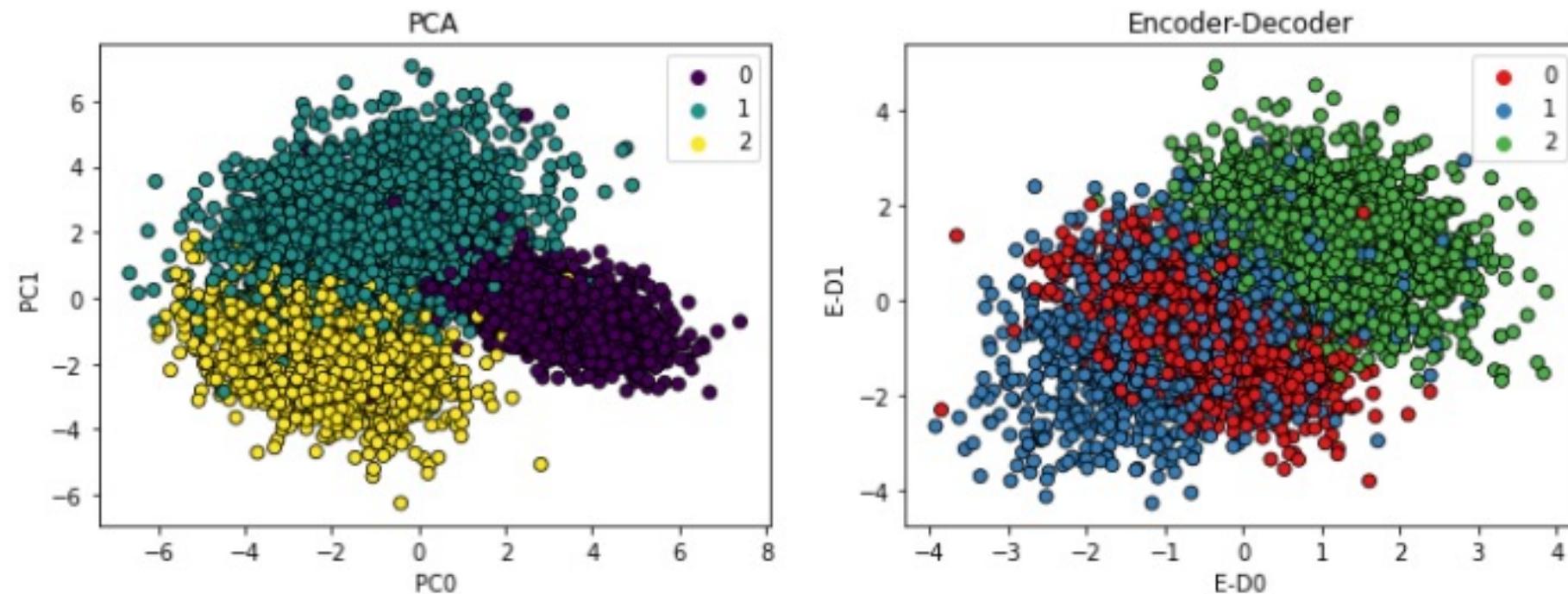






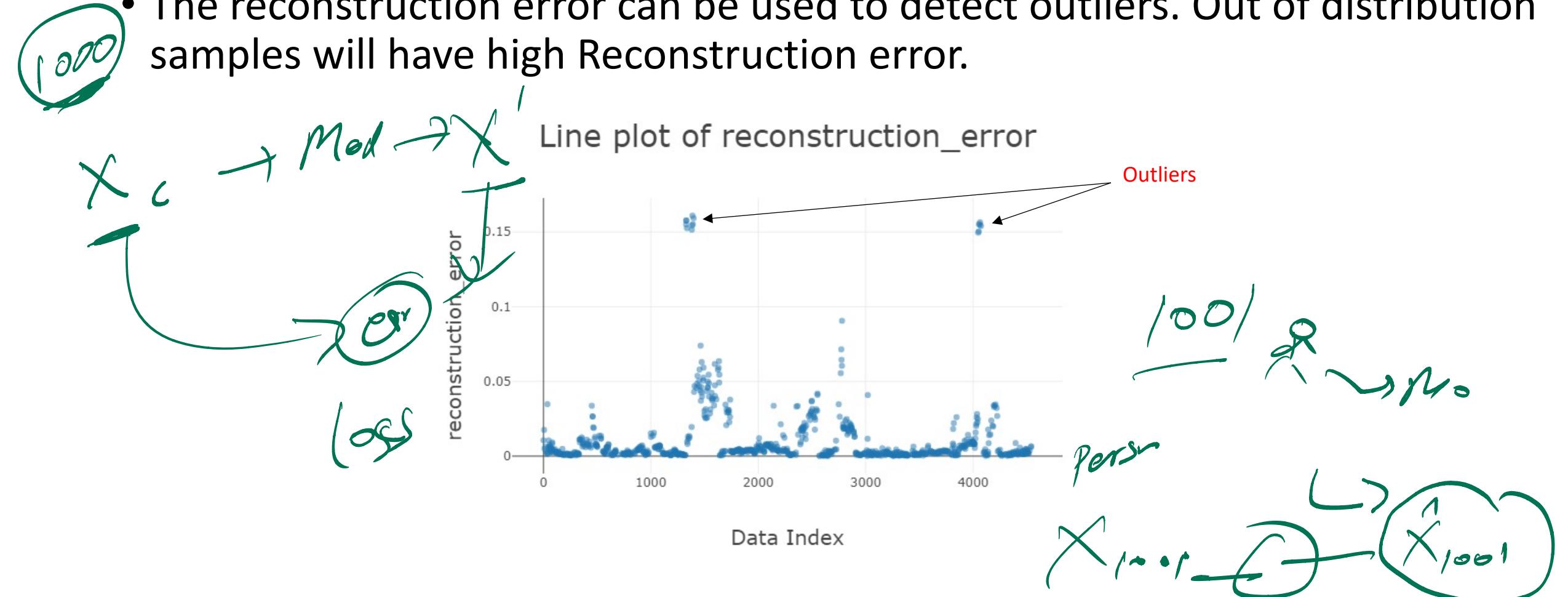
AutoEncoders for Dimensionality Reduction

- The encoder output can be used for dimensionality reduction

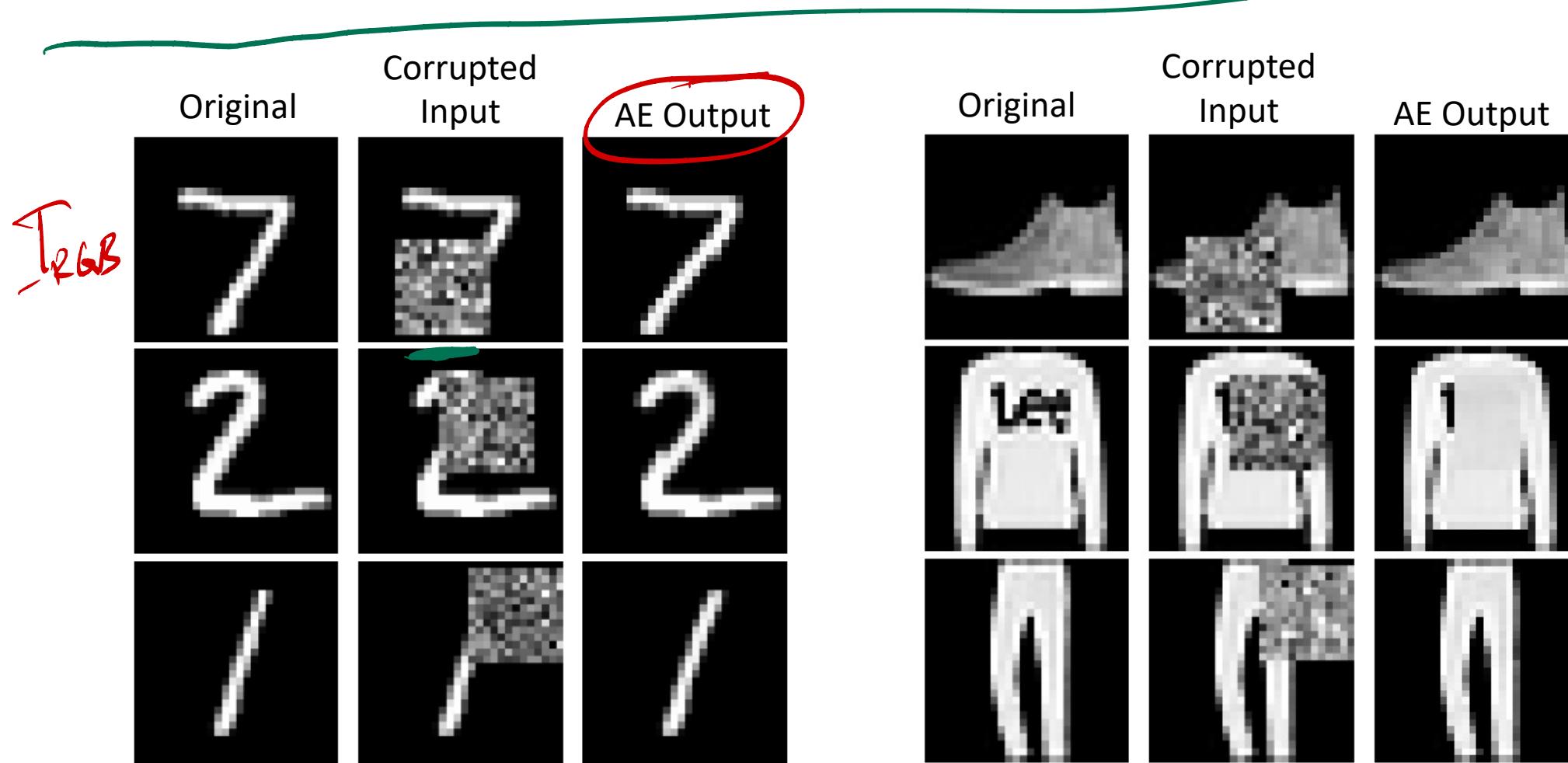


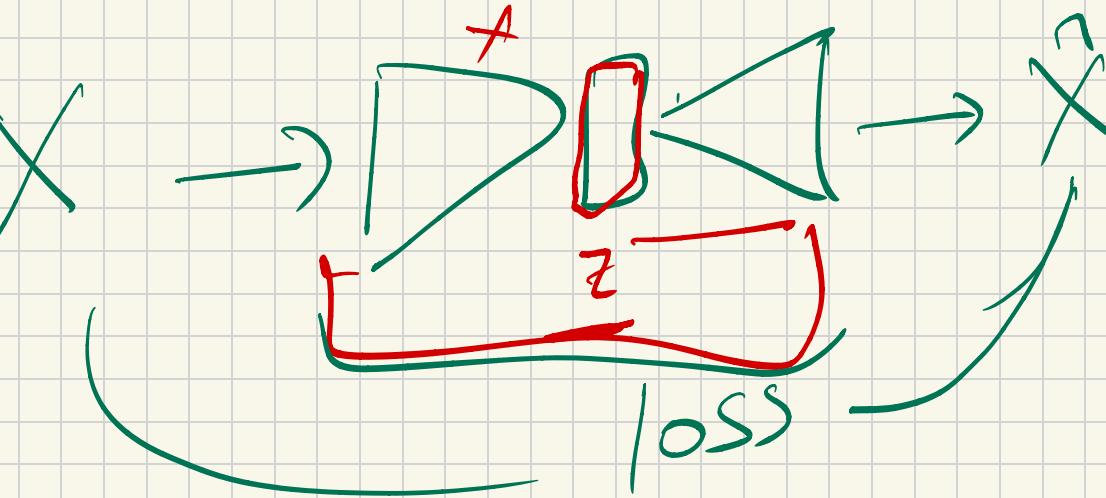
AutoEncoders for Outlier Detection

- The reconstruction error can be used to detect outliers. Out of distribution samples will have high Reconstruction error.

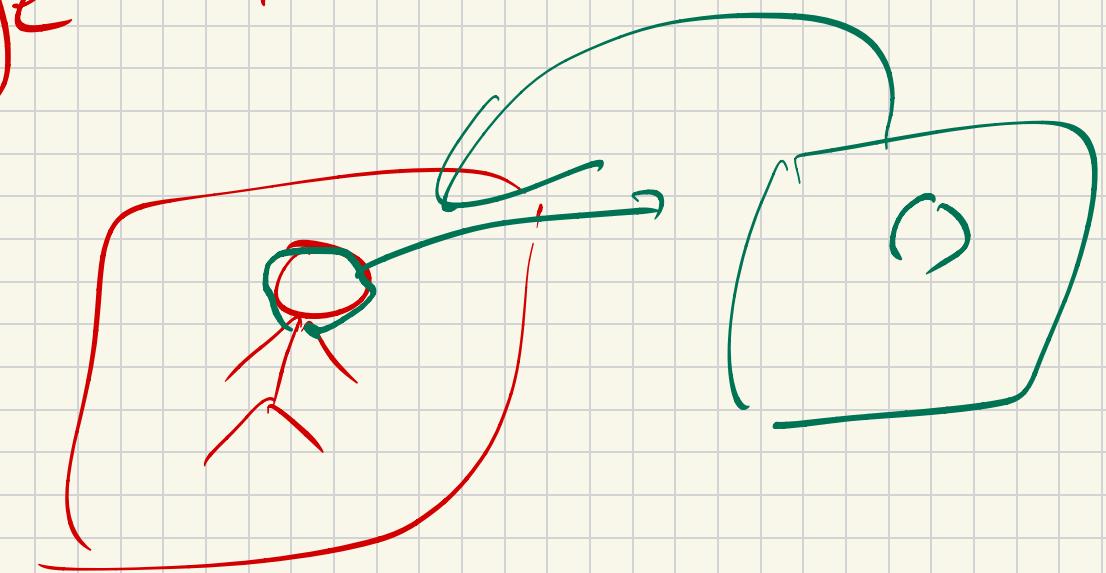
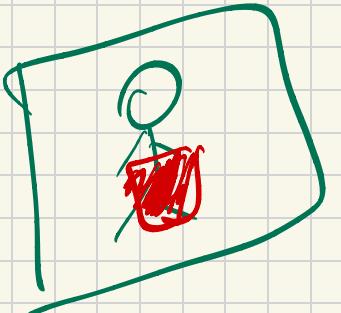


AutoEncoders for Image Completion





Generative
Image



ML Algorithms Perspectives:

- We can look into ML algorithms from two perspective:
 - **Loss Minimization** Problem (like what we did).
 - **Probability Maximization** Problem (using Maximum Likelihood Estimation).
- Both should result in the same solution.

Probalistic Interpretation of Linear Regression and MLE

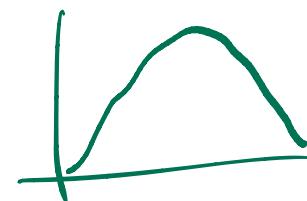
- We can also look at the probalistic Interpretation of Linear Regression.
- Keeping everything else same as the previous formulation

$$y_i = \mathbf{x}_i^T \boldsymbol{\theta} + \epsilon_i$$

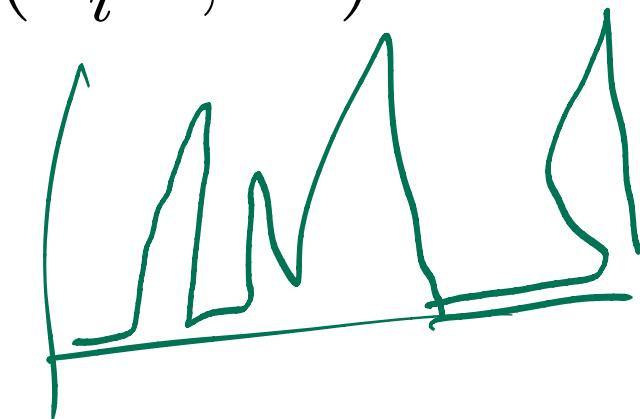
$$P(\hat{y} | x)$$

$$y = n \cdot x + b$$

- Now assume that $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$, then $y_i \sim \mathcal{N}(\mathbf{x}_i^T \boldsymbol{\theta}, \sigma^2)$
- We can write the conditional distribution as :



$$P(y_i | \mathbf{x}_i) \sim \mathcal{N}(0, \sigma^2)$$



Probabilistic Interpretation of LR

- Let's assume that all data points in the dataset are i.i.d. (independent identically distributed). Then we have:

$$\text{IP}(\mathcal{D}) = \prod_{i=1}^N \text{IP}(\underline{x_i}, y_i)$$

- Using Bayes Theorem we can write:

$$\prod_{i=1}^N \text{IP}(\underline{x_i}, y_i) = \prod_{i=1}^N \text{IP}(\underline{x_i}) \text{IP}(y_i | \underline{x_i})$$

$\text{p}(\underline{y} | \underline{x})$

y_i

x_i

P(the teacher drinks water) =

$P(\text{the})P(\text{teacher})P(\text{drinks} \mid \text{the teacher})$.

$P(\text{water} \mid \text{the teacher})$.

Maximum Likelihood Estimator

- In simple words, given the Dataset we want to find the values of the unknown parameters which maximize the probability of the Dataset.
- Using the definition of the conditional distribution we have

$$\text{P}(y_i | \mathbf{x}_i) = \frac{1}{\sigma \sqrt{2\pi}} \exp \left(- \underbrace{(y_i - \mathbf{x}_i^T \boldsymbol{\theta})}_{\text{Error term}} \right)$$

- Using the definition we get

$$\prod_{i=1}^N \text{P}(\mathbf{x}_i, y_i) = \prod_{i=1}^N \text{P}(\mathbf{x}_i) \prod_{i=1}^N \frac{1}{\sigma \sqrt{2\pi}} \exp \left(- \underbrace{(y_i - \mathbf{x}_i^T \boldsymbol{\theta})}_{\text{Error term}} \right)$$

Maximum Likelihood Estimator

- Let's try to maximaize:

$$\prod_{i=1}^N \text{P}(\mathbf{x}_i, y_i) = \prod_{i=1}^N \text{P}(\mathbf{x}_i) \prod_{i=1}^N \frac{1}{\sigma \sqrt{2\pi}} \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta}))$$

- Note that

$$\arg \max_{\boldsymbol{\theta}} \prod_{i=1}^N \text{P}(\mathbf{x}_i, y_i) = \underbrace{\arg \max_{\boldsymbol{\theta}}}_{\text{argmle}} \prod_{i=1}^N \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2)$$

Maximum Likelihood Estimator

- Furthermore, since the right hand side of the above equation is monotonic in \theta the arg max will not change if we take log of the expression

$$\arg \max_{\theta} \prod_{i=1}^N \exp(-(y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2) = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^N (- (y_i - \mathbf{x}_i^T \boldsymbol{\theta})^2)$$

argmin
 in interpretation
 of maximum
 likelihood

- Notice that the right hand side is minimising the MSE.
- Hence solution of minimizing the MSE is equivalent to Maximum Likelihood Estimator for linear regression

A Slight Detour: A Look at Optimization Tools

Introduction

- **Optimizers** are algorithms that adjust the weights of the model to minimize the loss function during training.
- They are one of the main components of Deep Learning models.
- We will go through several types of optimizers in this section.

Direction of maximum increase and decrease for a function

- Gradient direction is the direction of maximum increase for a function
- Negative gradient is the direction of maximum decrease for a function

$$\hat{w} = w - \eta w$$

```
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad # perform parameter update
```

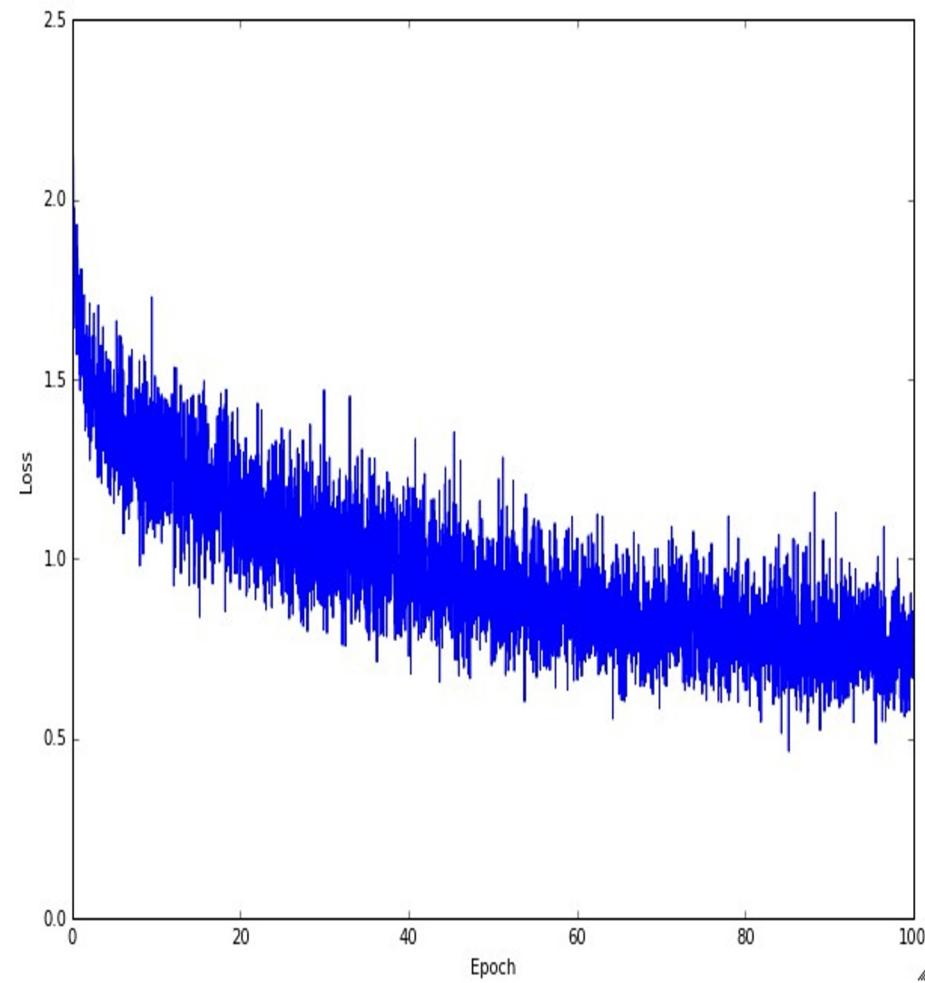
- **Mini batch Gradient Descent**
- only use a small portion of the training set to compute the gradient.

```
# Vanilla Minibatch Gradient Descent

while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

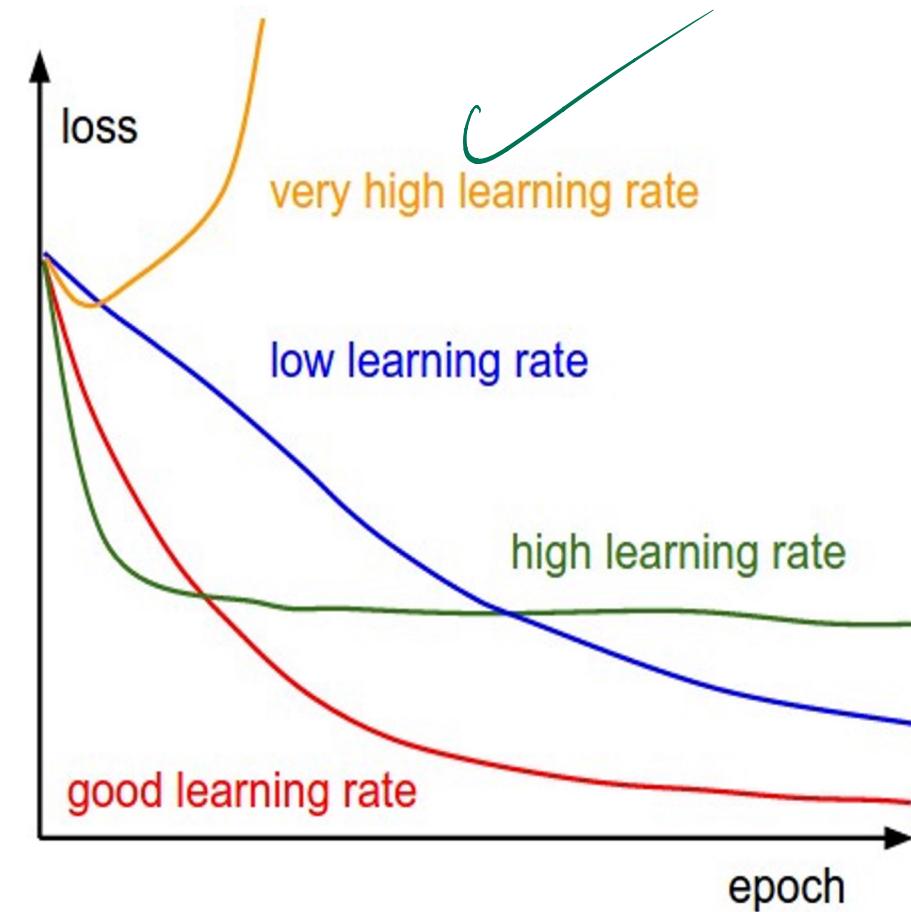
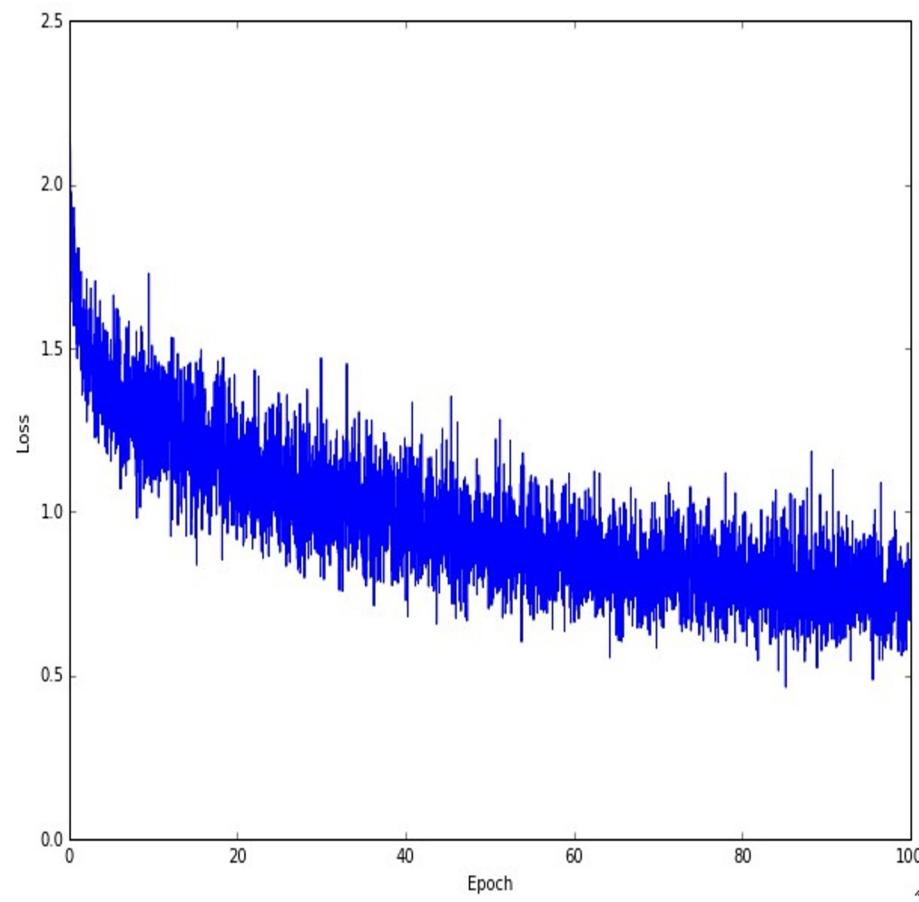
Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

2^5



Example of optimization progress
while training a neural network.

(Loss over mini-batches goes
down over time.)



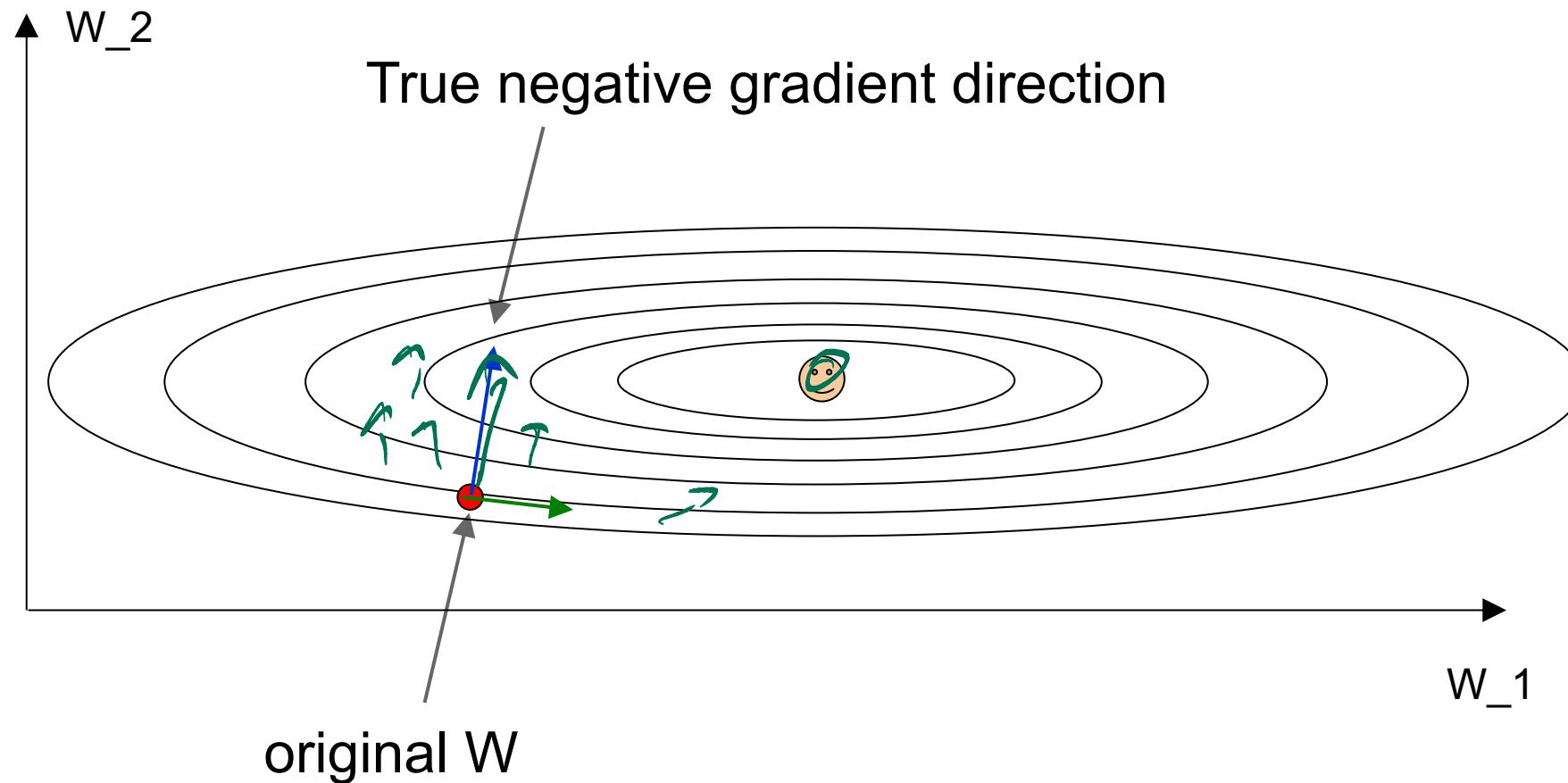
- only use a small portion of the training set to compute the gradient.

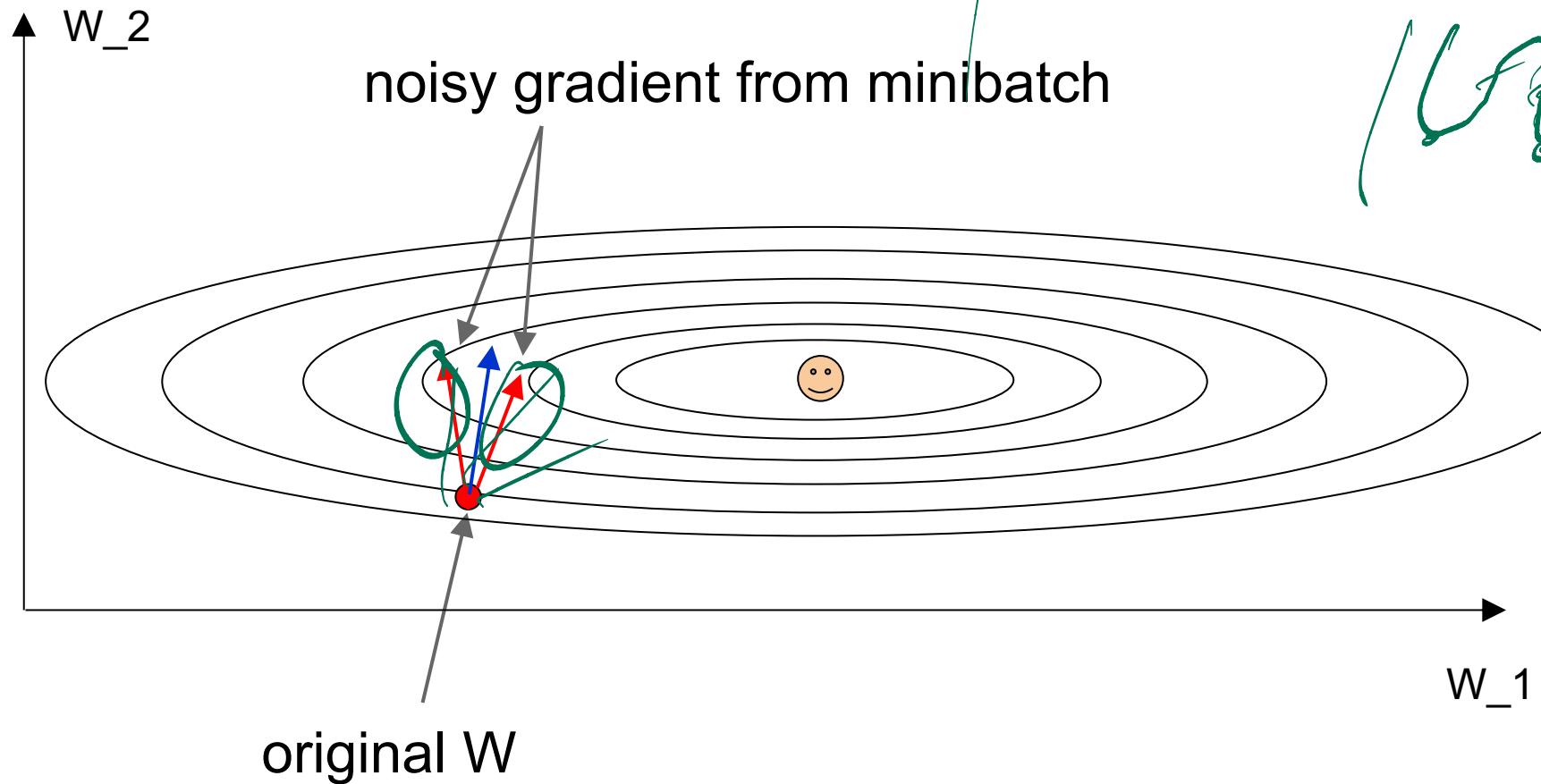
```
# Vanilla Minibatch Gradient Descent

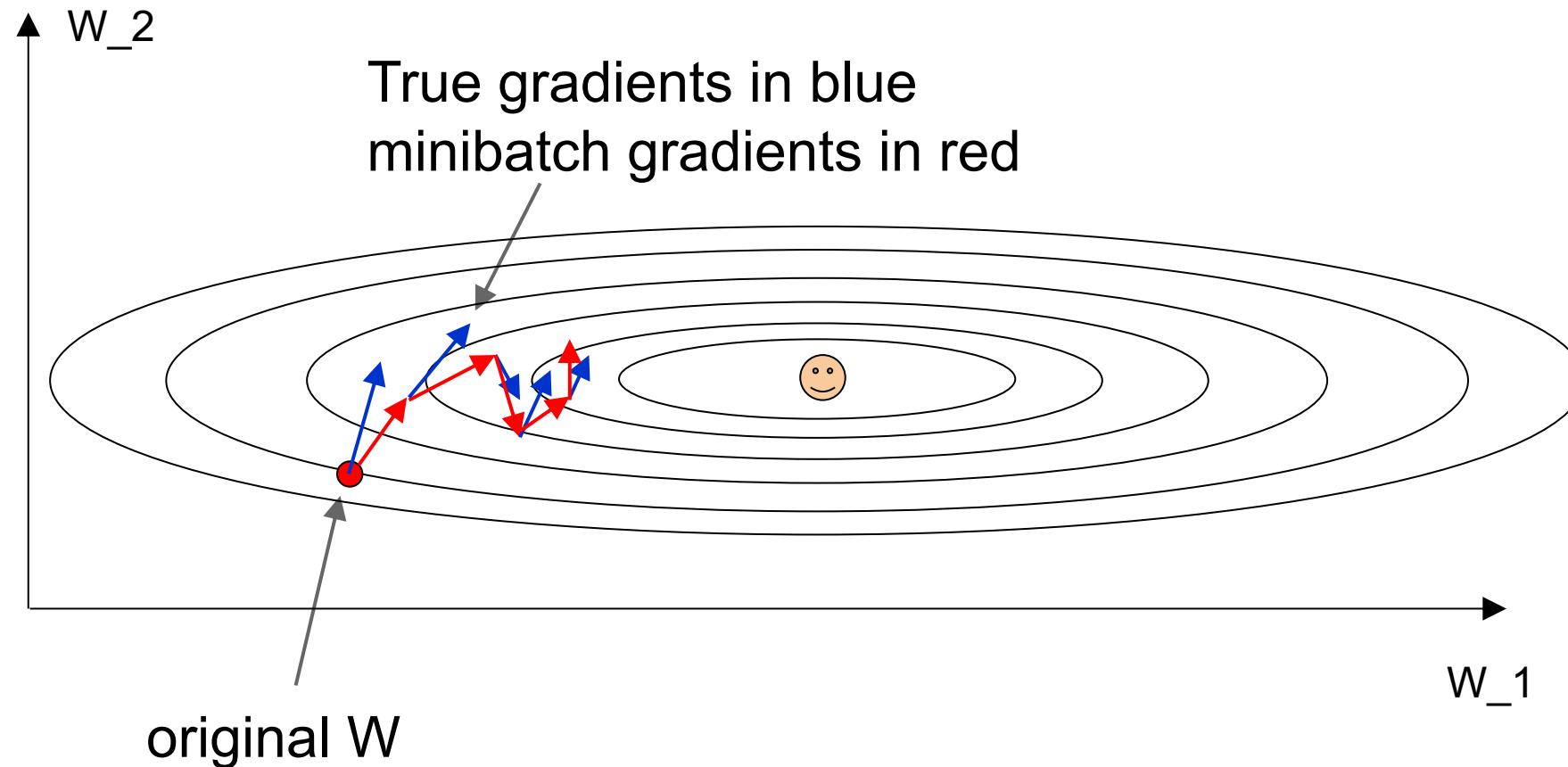
while True:
    data_batch = sample_training_data(data, 256) # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += - step_size * weights_grad # perform parameter update
```

Common mini-batch sizes are 32/64/128 examples
e.g. Krizhevsky ILSVRC ConvNet used 256 examples

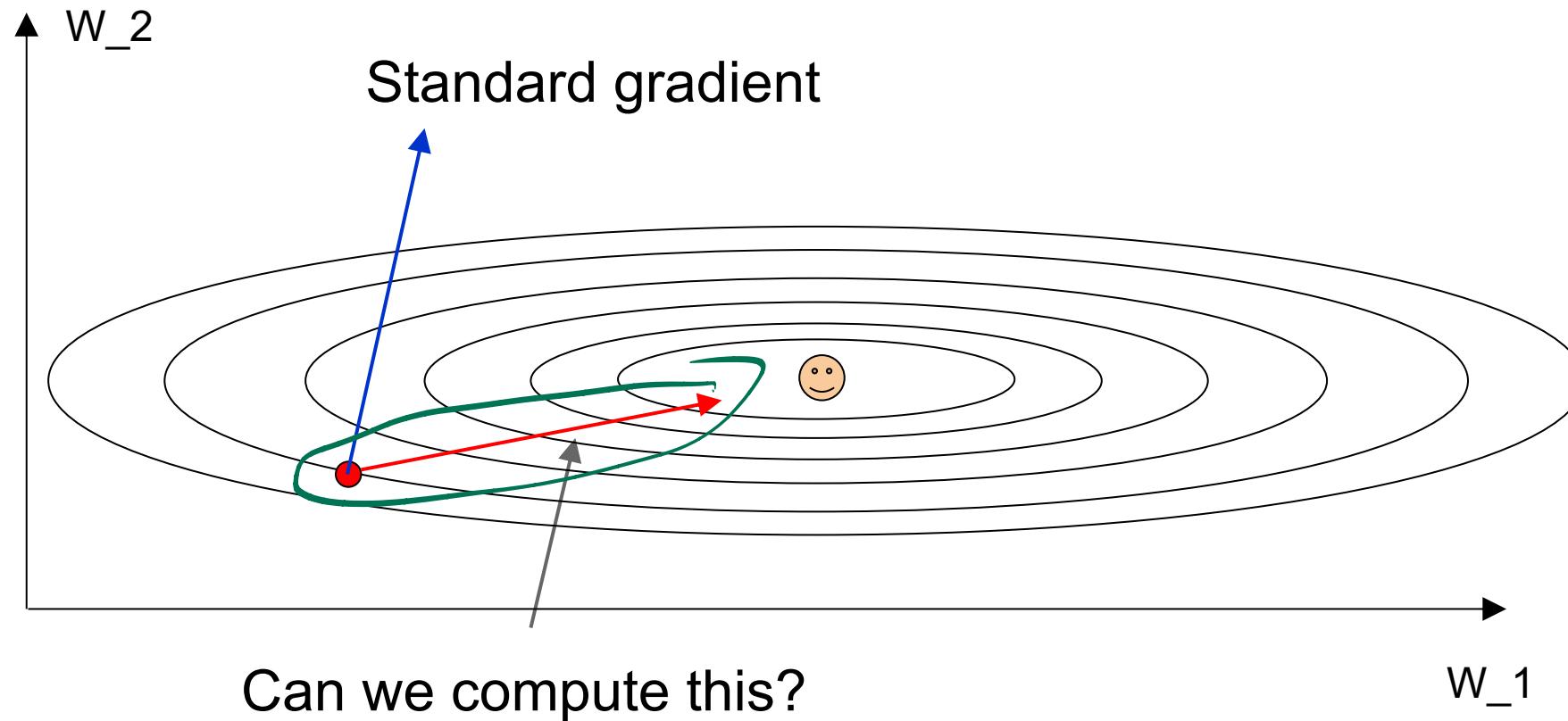
we will look at more fancy update formulas (momentum, Adagrad, RMSProp, Adam, ...)

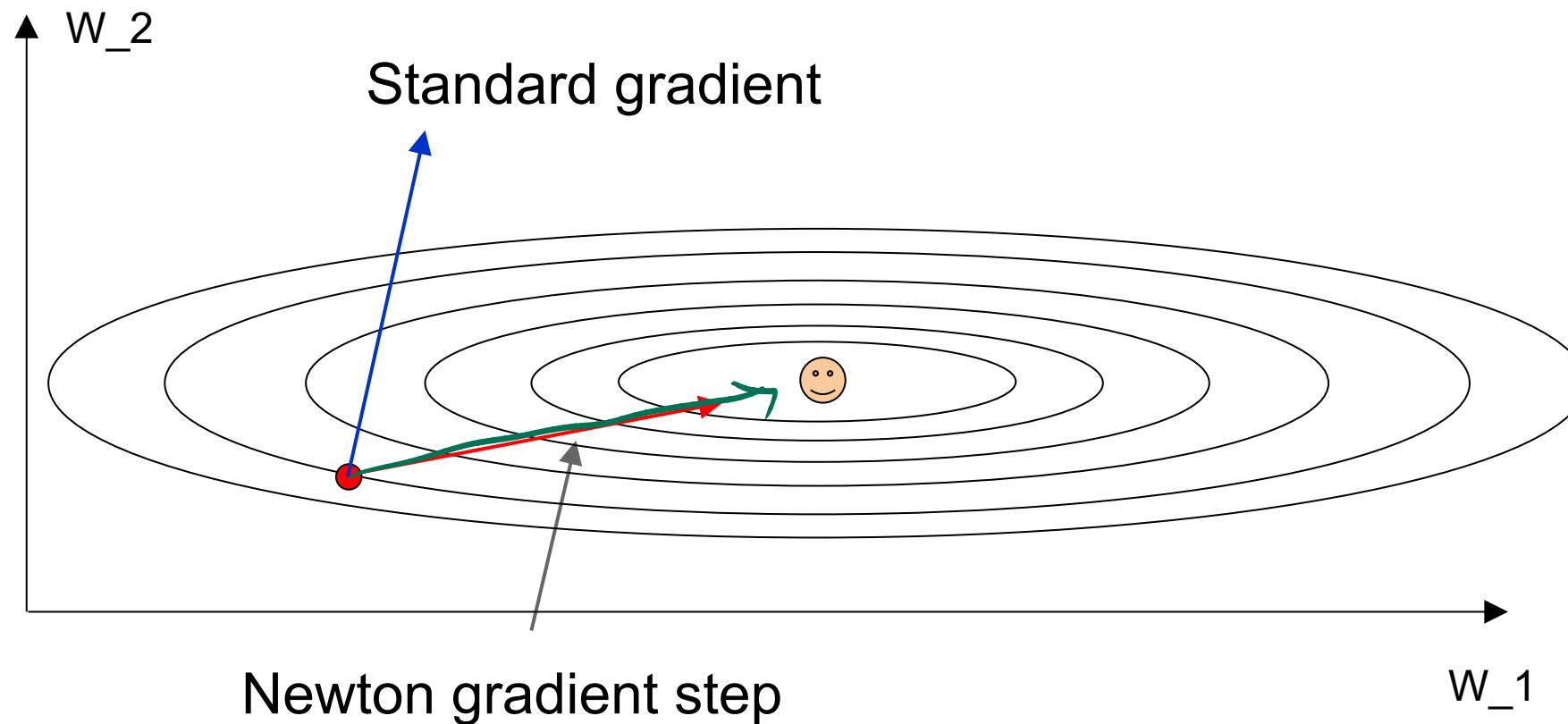






Gradients are noisy but still make good progress on average





higher

The Newton update is:

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

Where H_f is the Hessian matrix of second derivatives of f .

Converges very fast, but rarely used in DL. Why?

The Newton update is:

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

Converges very fast, but rarely used in DL. Why?

Too expensive: if x_n has dimension M, the Hessian $H_f(x_n)$ has dimension M^2 and takes $O(M^3)$ time to invert.

The Newton update is:

$$x_{n+1} = x_n - H_f(x_n)^{-1} \nabla f(x_n)$$

Converges very fast, but rarely used in DL. Why?

Too expensive: if x_n has dimension M, the Hessian $H_f(x_n)$ has dimension M^2 and takes $O(M^3)$ time to invert.

Too unstable: it involves a high-dimensional matrix inverse, which has poor numerical stability. The Hessian may even be singular.

Momentum Optimization

```
# Gradient descent update
x += - learning_rate * dx
```

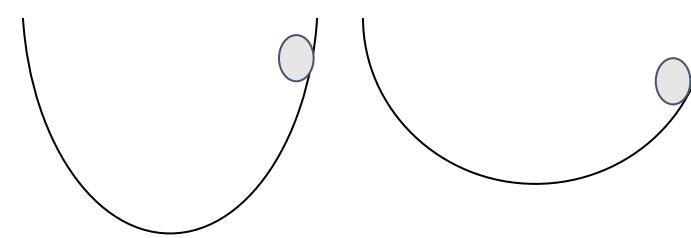


$\frac{1}{2}$

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

- Physical interpretation as ball rolling down the loss function + friction (mu coefficient).
- mu = usually ~0.5, 0.9, or 0.99 (Sometimes annealed over time, e.g. from 0.5 -> 0.99) for adaptation

```
# Gradient descent update  
x += - learning_rate * dx
```



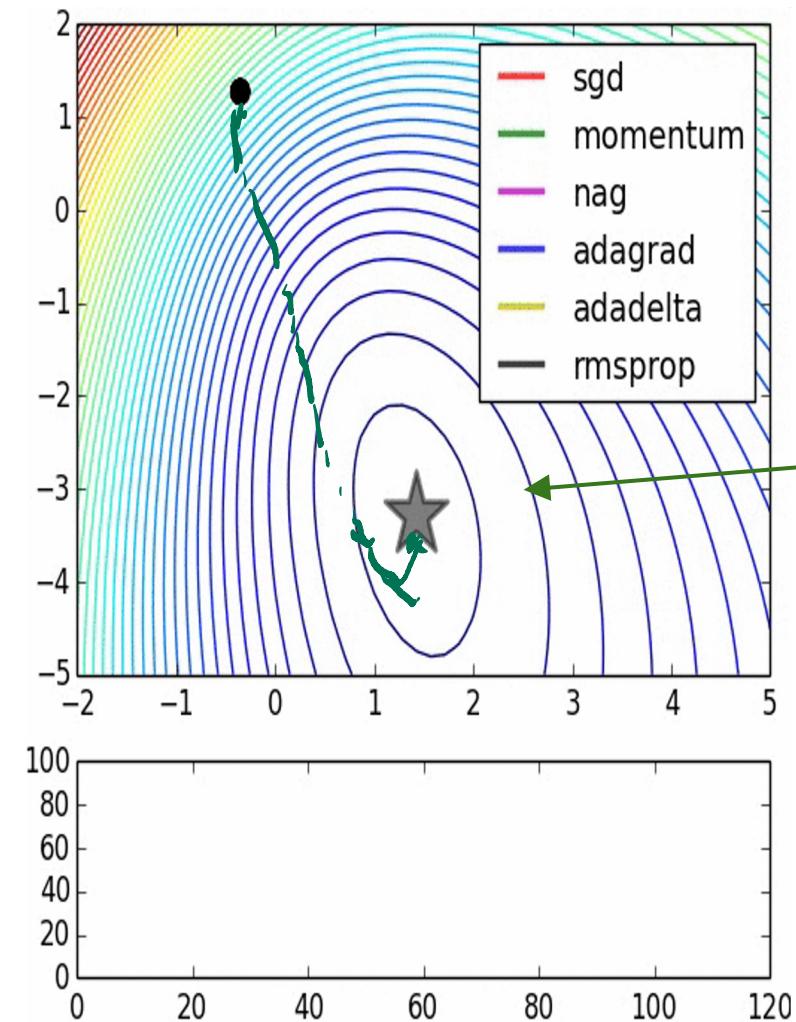
```
# Momentum update  
v = mu * v - learning_rate * dx # integrate velocity  
x += v # integrate position
```

- Allows a velocity to “build up” along shallow directions
- Velocity becomes damped in steep direction due to quickly changing sign



Smoother trajectories in optimization, avoiding zig-zagging behavior.
Faster convergence towards the global minimum.

SGD vs Momentum



notice momentum
overshooting the target,
but overall getting to the
minimum much faster
than vanilla SGD.

```
# Adagrad update  
cache += dx**2  
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```

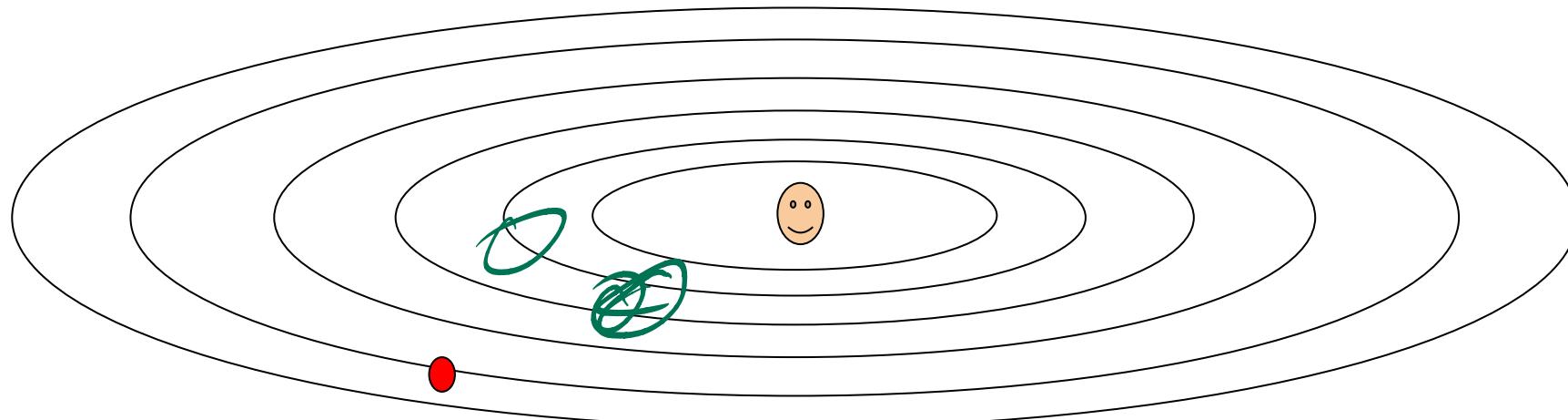
gradient clipping

Added element-wise scaling of the gradient based on the historical sum of squares in each dimension

- The element-wise scaling ensures that parameters corresponding to dimensions with larger cumulative gradients (cache) are updated less aggressively.
- Conversely, parameters with smaller cumulative gradients are updated more, balancing their influence.

making larger adjustments in directions where gradients are smaller (flatter regions) and smaller adjustments in steeper directions.

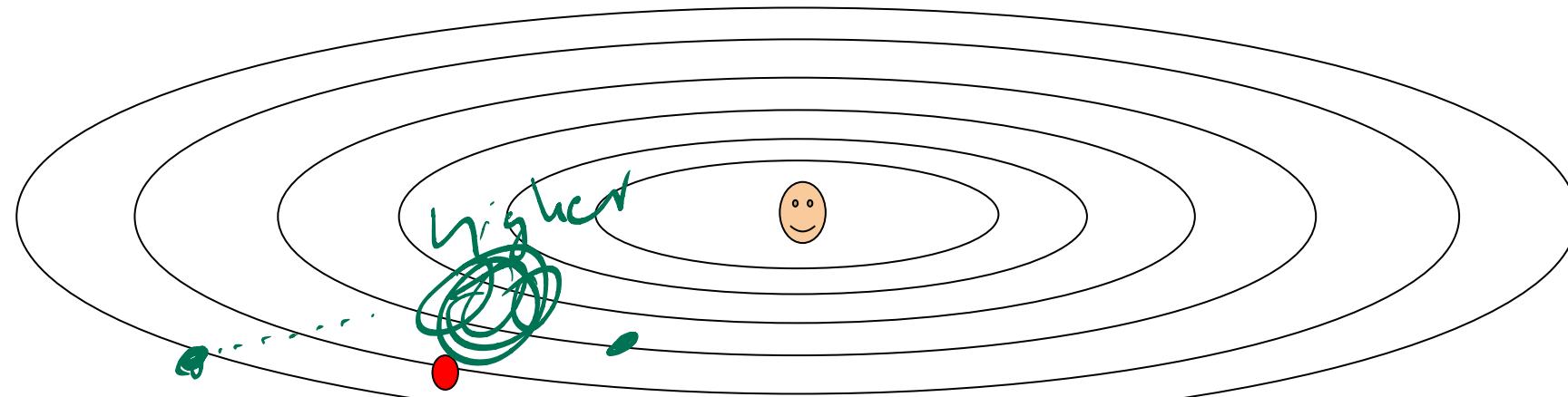
```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



Q: What happens with AdaGrad?

The optimizer takes smaller steps vertically (steep direction) and larger steps horizontally (flat direction).

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



Q2: What happens to the step size over long time?

Extremely small steps as the optimizer nears convergence, effectively "stalling" progress.

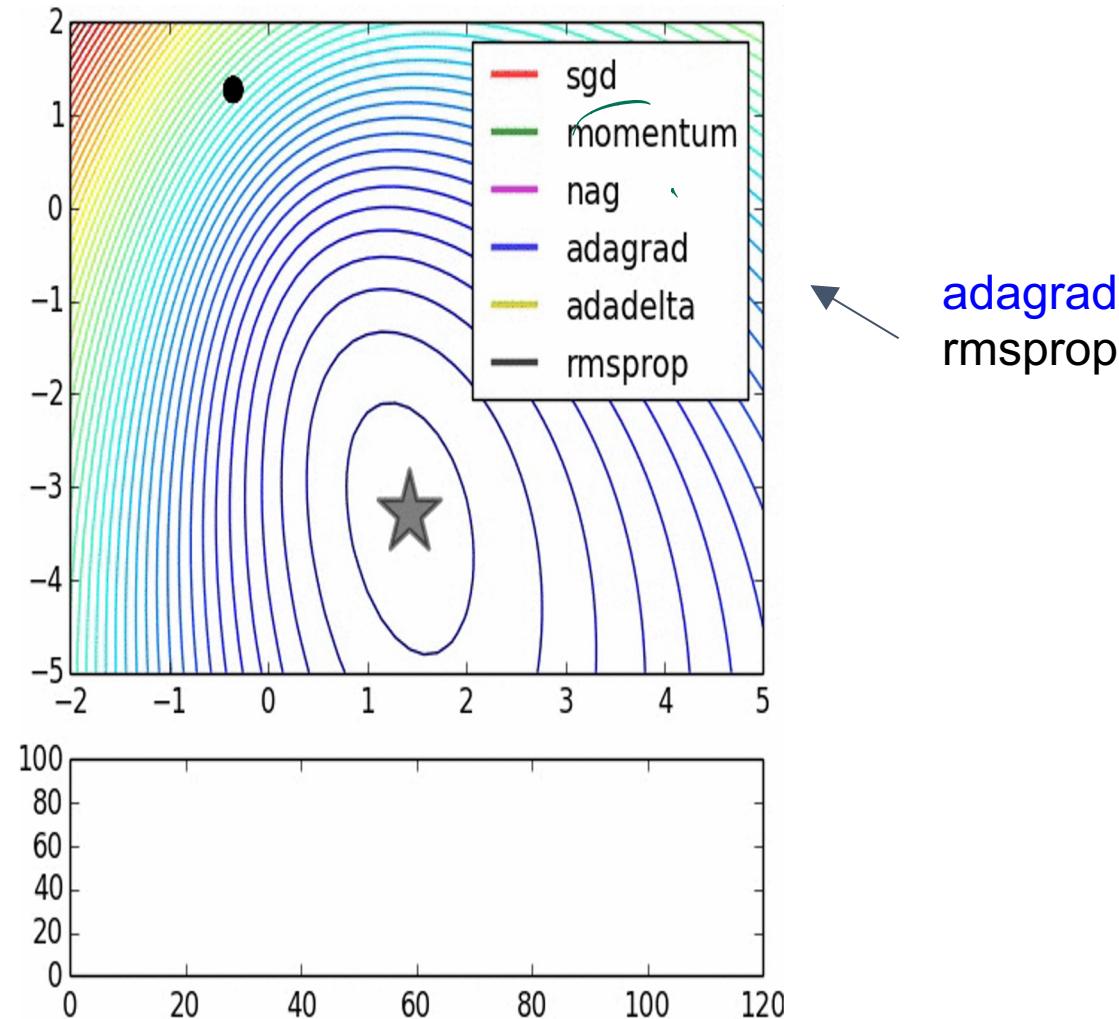
RMSProp

Exponential decay factor for cache, which prevents it from growing indefinitely.

```
# Adagrad update
cache += dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



```
# RMSProp
cache = decay_rate * cache + (1 - decay_rate) * dx**2
x += - learning_rate * dx / (np.sqrt(cache) + 1e-7)
```



Adam optimizer

smooths the gradients over time by keeping a moving average of the gradient.

```
# Adam
m = beta1*m + (1-beta1)*dx # update first moment
v = beta2*v + (1-beta2)*(dx**2) # update second moment
x += - learning_rate * m / (np.sqrt(v) + 1e-7)
```

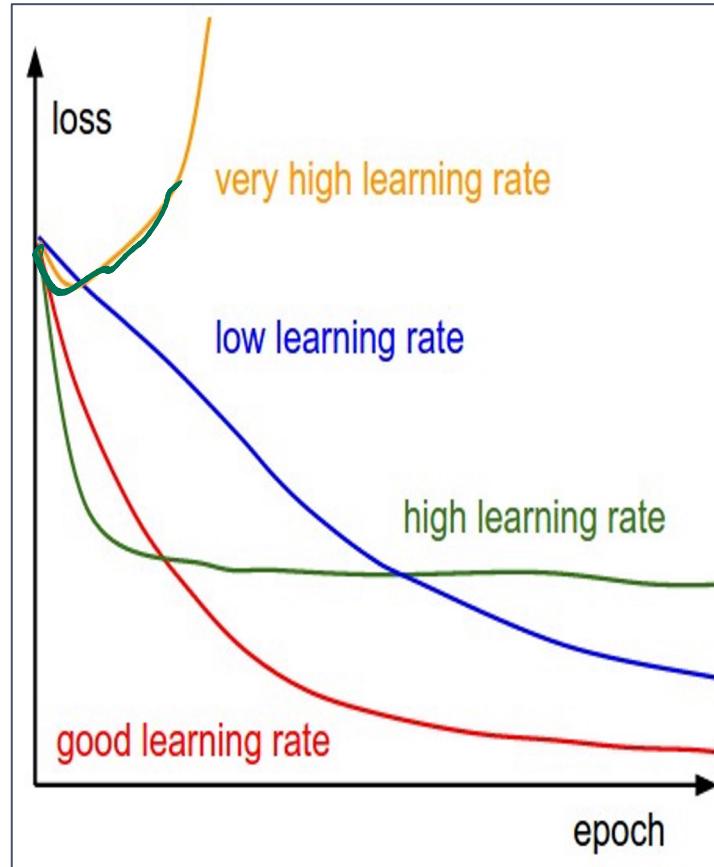
momentum

RMSProp-like

normalizing gradients by scaling them with the square root of their historical squared values.

Looks a bit like RMSProp with momentum

for



Q: Which one of these learning rates is best to use?

Small

model.train()

for i in range(100):

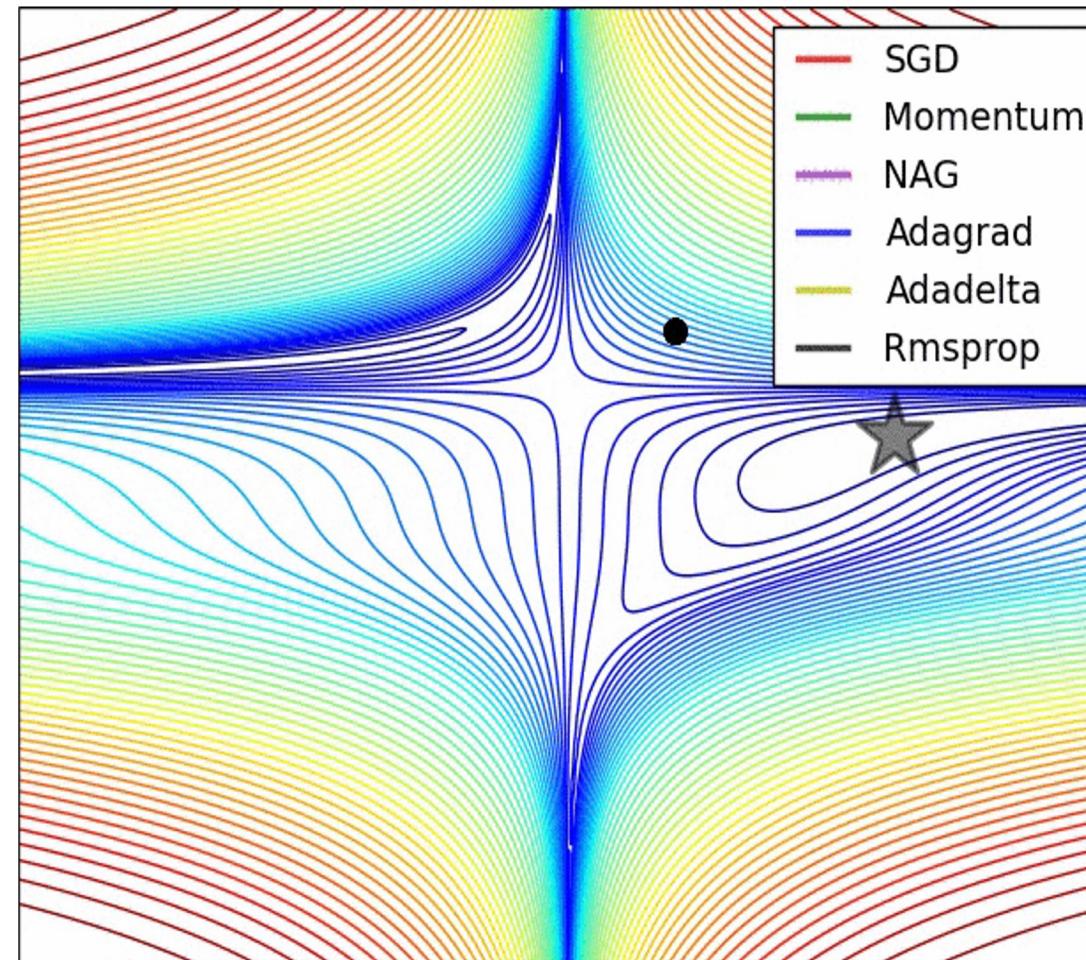
out = model(i)

$$\text{loss} = \text{CE}(out, y)$$

(

)
→ loss + if loss >

If epoch -
adjust Q - .()



(image credits to Alec Radford)

Summary

