MLF Week 1

Aaron Gu, Warrick Tsui

Content for Today

1. Introduction to UTMIST & ML Fundamental Program
2. Introduction to Machine Learning
3. Linear Regression (Your first step in ML!)

UTMIST & the MLF Program

What is UTMIST?

University of Toronto Machine Intelligence Student Team (UTMIST) is the **largest undergraduate ML/AI club in North America.**

8 departments, 24+ design teams, 170+ executives, and a total of **2200+ community members.**

Our goal is to help students of different backgrounds gain experiences to **grow** and **develop** in their **professional** and **academic** careers in AI/ML.

What is Machine Learning Fundamentals (MLF)?

A beginner friendly program that provides you the opportunities to learn ML concepts, build hands on projects, and meet like-minded peers!

MLF is an annual program from **now to March, 2026**. The program is broken down into **2 phases.**

Phase 1 (Fall): Learning through workshops & complete small take home exercises

Phase 2 (Winter): Pitch and build your own project under the guidance and support of UTMIST academic directors

MLF Phase 1 Contents

Week	Date	Topics
1	Sep 27, 2025	Introduction & Linear Regression
2	Oct 4, 2025	Logistic Regression
3	Oct 11, 2025	Neural Network Part 1
4	Oct 18, 2025	Neural Network Part 2
5	Nov 8, 2025	Decision Trees
6	Nov 15, 2025	Naive Bayes & Connection to GenAI
7	Nov 22, 2025	ML Best Practices
8	Nov 29, 2025	Introduction to Deep Learning & Modern ML

MLF Phase 2 Contents

Semester 2 Winter (Jan - March)

Key ideas:

1. Project demo & work through from UTMIST directors
2. Propose your own project through a formal project proposal & gain feedbacks from UTMIST directors
3. Build your own project under the guidance of UTMIST directors

Why MLF?

Get Head: gain experience in ML ahead of course offering!

Meet People: Meet like minded people and upper-year mentors to guide your ML journey!

Beyond Theory: Build hand-on project to apply the knowledge

Engineering projects will prefer hiring candidates who are committed MLF (because they trust us :)

What You Need to Succeed in MLF

1. Passion (Passion is KEY!)
2. Consistent attendance (Get a little better each week)
3. Complete take home exercises (Theory → Practice)
4. Ask questions if you don't understand something (that's how you learn)
5. Learn from peers (Grow and learn together)

Technical:

1. Some knowledge about Python programming (Loop, function, conditional statements, etc)
2. High school math (derivatives, chain-rule, etc)

Meet the Team

Program Directors

Aaron Gu
Program Director

Warrick Tsui
Program Director

Academics Team

Kaden Seto
Academics Team

Oscar Yasunaga
Academics Team

Andrew Magnuson
Academics Team

Jessica Chen
Academics Team

Riyad Valiyev
Academics Team

Chloe Nguyen
Academics Team

Matthew Tamura
Academics Team

Jingmin Wang
Academics Team

Ice Breaker Time!

Turn to the people on your left and your right. Those are the peers who will be on this learning journey with you!

1. Introduce yourself (name, year, program, etc)!

2. Why you are interested in AI?

3. What's your favorite ice-cream flavour?

Machine Learning

2

What is Machine Learning?

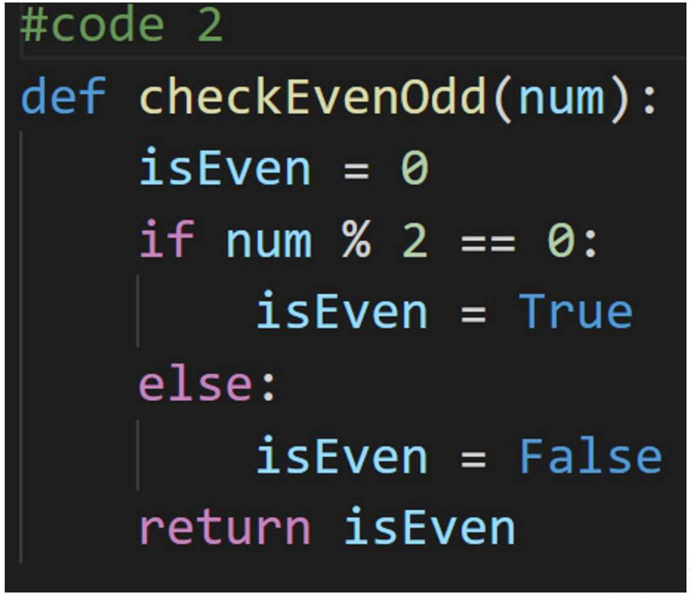
Any ideas?

Machine Learning is...

The study of data, **a lot of data**. We study their **pattern** and underlying **distribution**, and maybe eventually **reproduce** them

Why Not Use Traditional Programming?

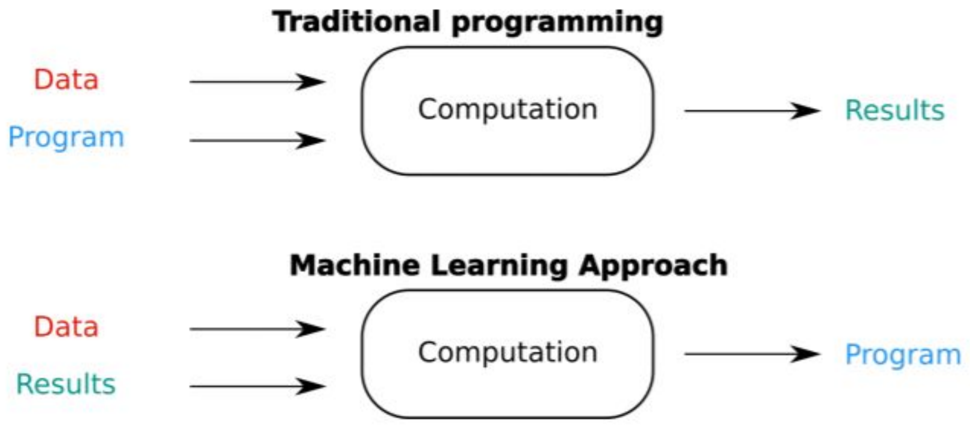
Any ideas?



```
#code 2
def checkEvenOdd(num):
    isEven = 0
    if num % 2 == 0:
        isEven = True
    else:
        isEven = False
    return isEven
```

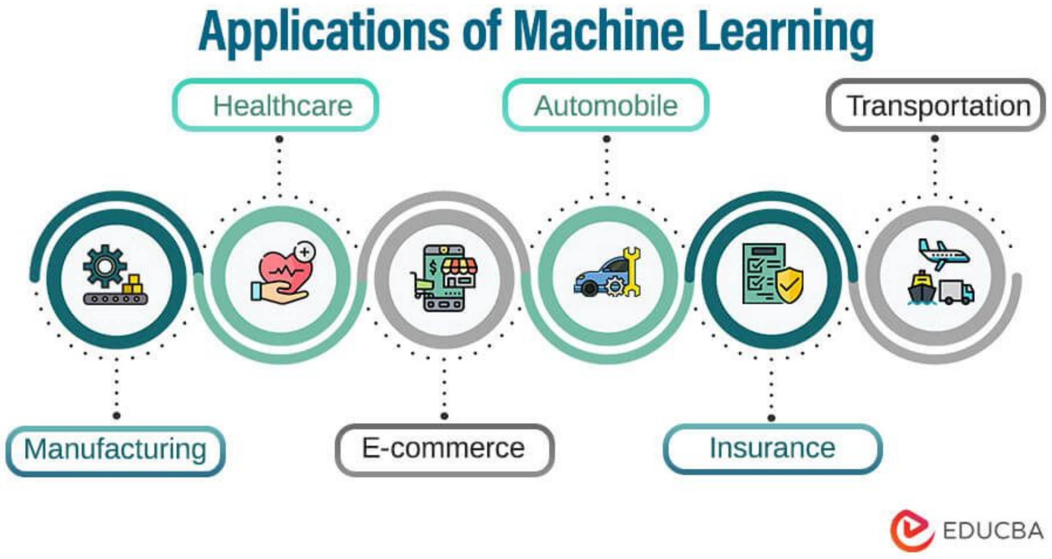
ML vs. Traditional Programming

Traditional Programming	**Machine Learning**
Hand-crafted logic, not derived from data	No clear logic, patterns are derived from data
High transparency	Low transparency
Hard to handle large, complex problems	Easier to adapt to large complex problem



Applications of ML

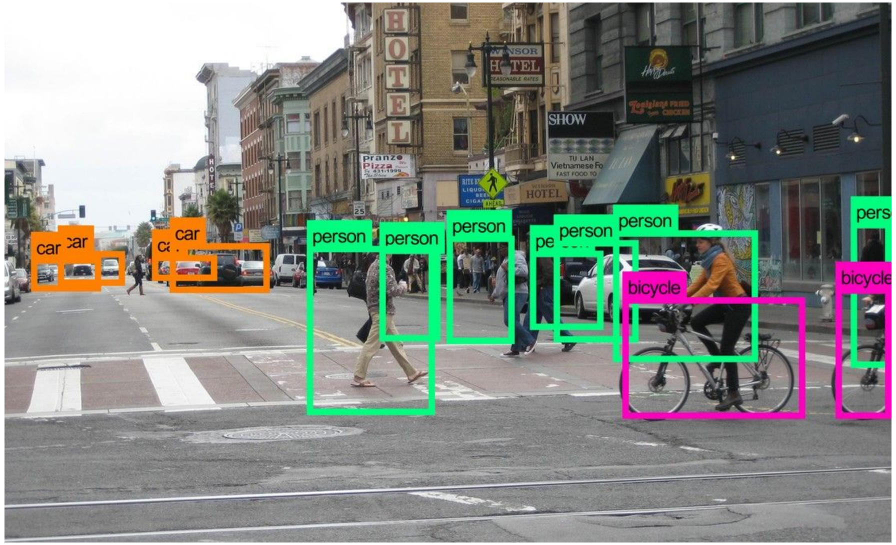
- Finance
- Robotics
- ChatGPT
- Social Media Algorithms
- Health Care
- Self-Driving Cars
- Spam Email Detection
- Way Too Many



Some Major Fields of ML

1. Computer Vision
2. Natural Language Processing
3. Reinforcement Learning
4. Generative AI

Computer Vision



Dealing with what the computer **"sees"**

Relevant Problems:

- Image Classification
- Object Detection
- Face Recognition

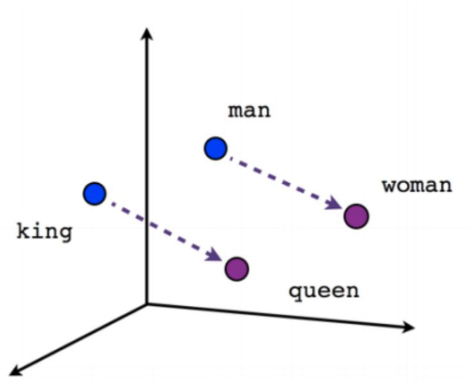
Example: is this an image of a cat or a dog?

Natural Language Processing

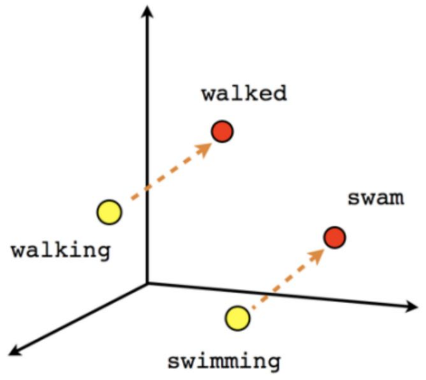
Dealing with human language

Relevant Problems

- Sentiment Classification
- Fraud Detection
- Text Generation
- Language Models



Male-Female



Verb tense

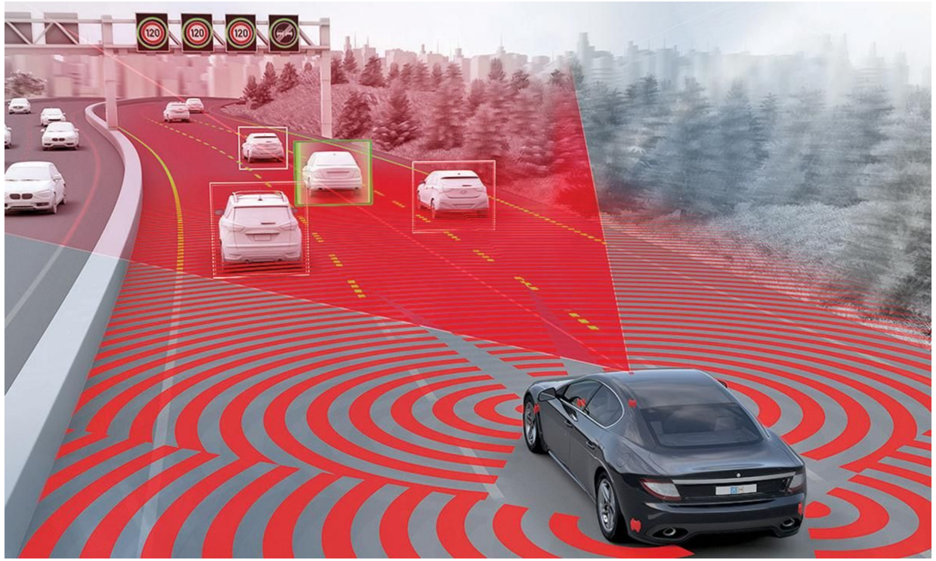
Example: Given an email, is it a spam?

Reinforcement Learning

Learning from trial & error

Relevant Problems

- Self-Driving Car
- Robot learn to Walk
- AI Learn to Play Chess
- Language Model Alignment



Examples: Alpha-go

Generative AI

Generating new data

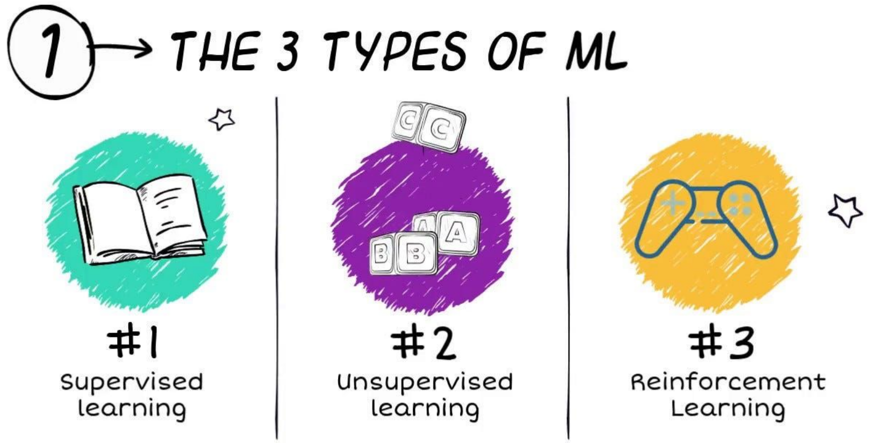
- Image Generation
- Video Generation
- Text Generation (ChatGPT)

Example: Generate an surrealistic planet

Want to Break Into Those Fields?

MLF will help you set the solid fundation to get started!

3 Types of Machine Learning Methods



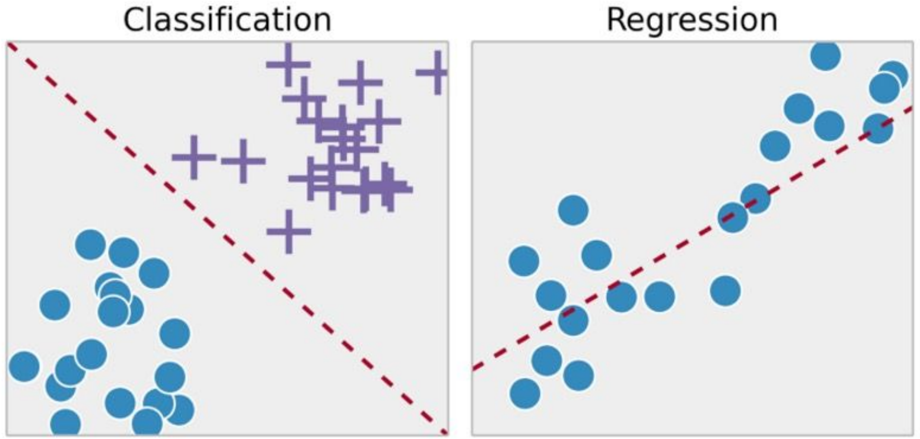
Supervised (MLF Focus): Learn from Answers (Label)

Unsupervised: Find Patterns in Data without Answers (No Label)

Reinforcement Learning (RL Workshop): Trial & Error

Types of Machine Learning Problems

1. Regression (MLF Focus)
2. Classification (MLF Focus)
3. Generation
4. Control



Focus Of Today

Regression - Linear Regression (Regression with a line)

This is the **simplest** algorithm in Machine Learning!

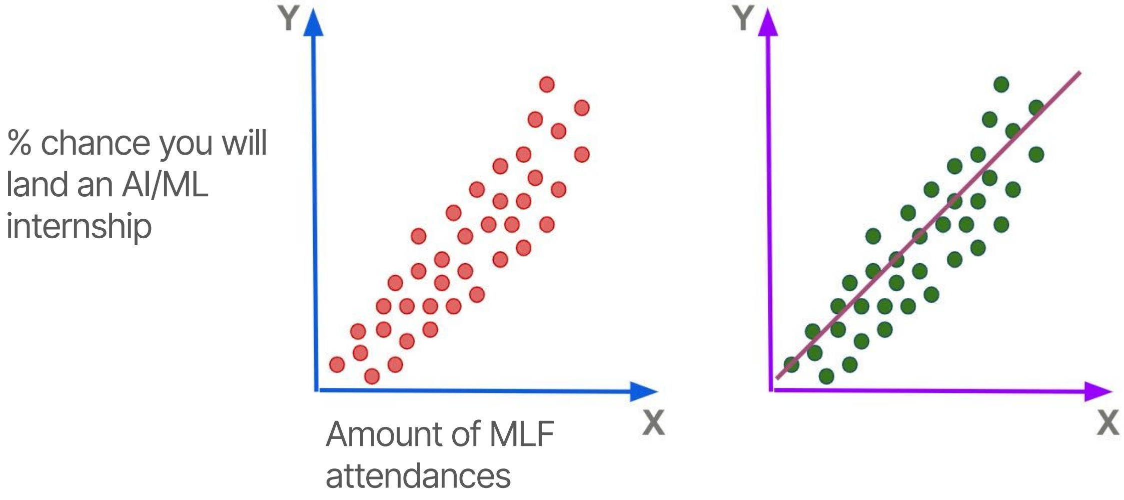
We have to start somewhere.

And **you will already see** the type of **problem solving** required in ML, and complexity scaling **up!**

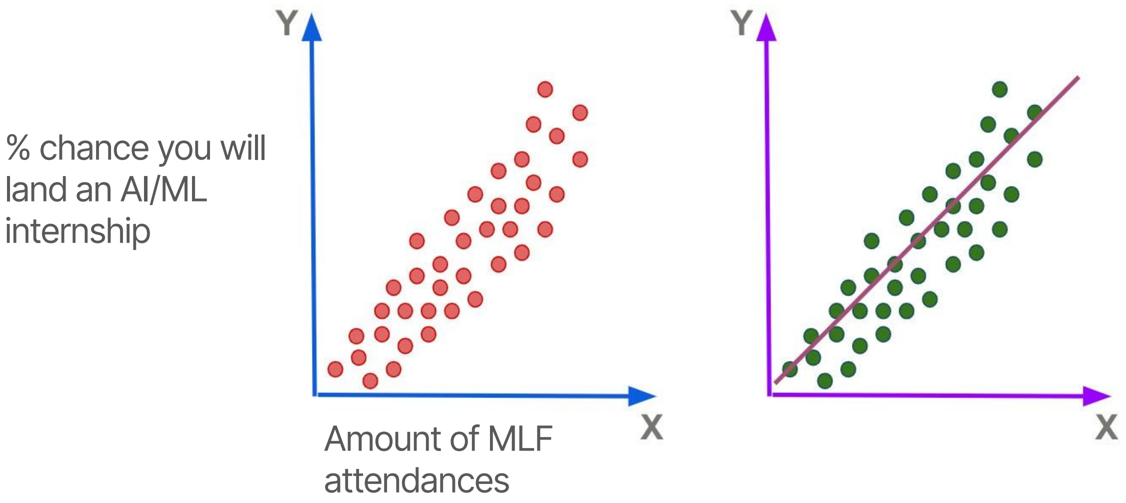
Modeling this (100% real) data...

Let’s say we have this data.

How can I approximate this set of data?



Yes, with a line!



But now let's add some proper ML terminology

I hope you remember this equation.

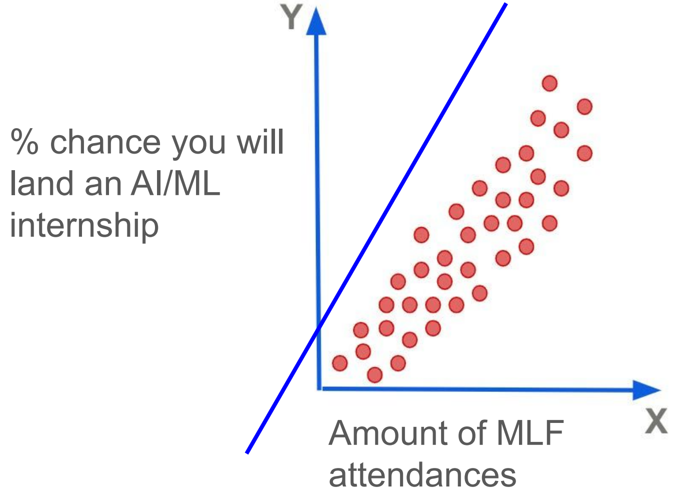
$y = wx + b$

w = **weight**
b = **bias**
x = **input**

And this line is what we call the "model".

Makes a **prediction** based on new data

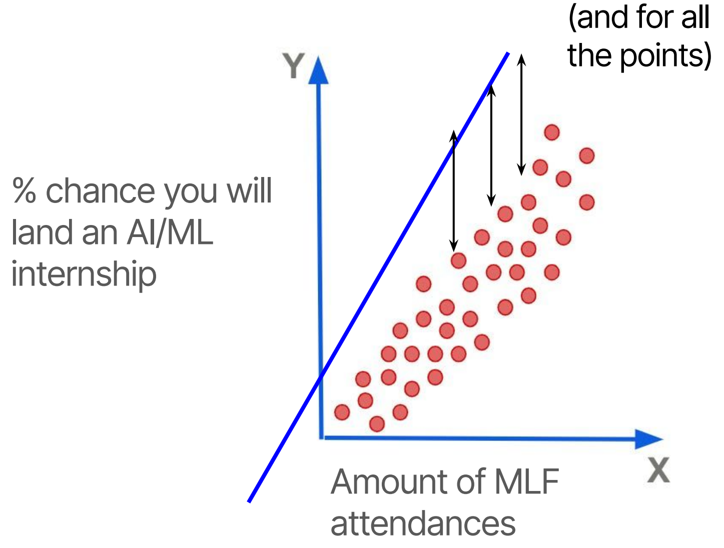
Loss Function...?



Then, how do we indicate how **“good/bad”** the model is?

That is, how good the line is compared to all the dots of data?

How do we "fit" to this line? MSE Loss.



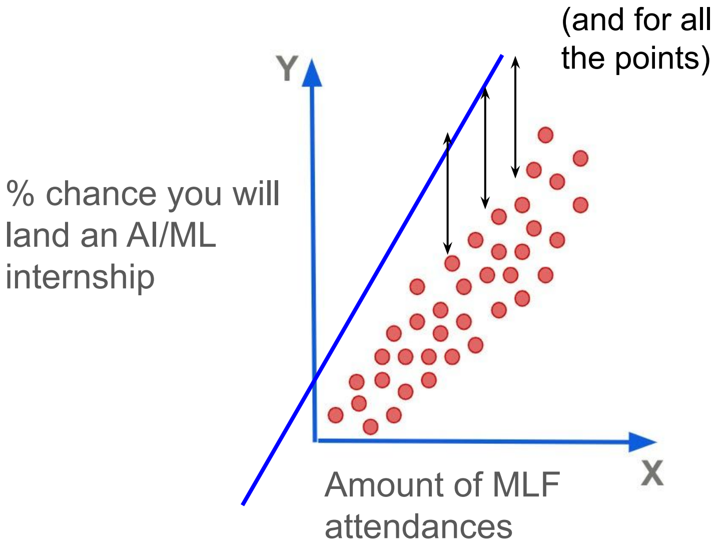
There's a **systematic way!**

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE	= mean squared error
n	= number of data points
Y_i	= observed values
$\hat{Y}_i$	= predicted values

This just means:
"Take the differences between data and line points. Square them, sum them, and take the average"

The line formula comes back!



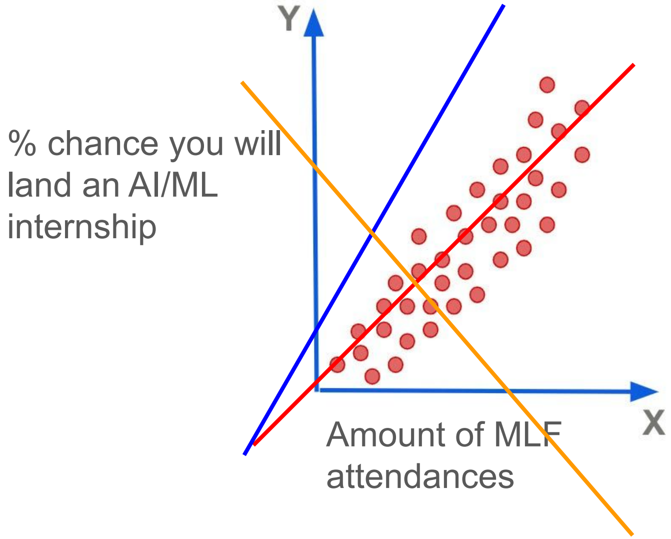
We established earlier that our model predict with **a line!**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)$$

Thus, y hat will be **parameterized** with respect to **the weight** and **the bias!**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (wx + b))$$

“Tweaking” towards the good line

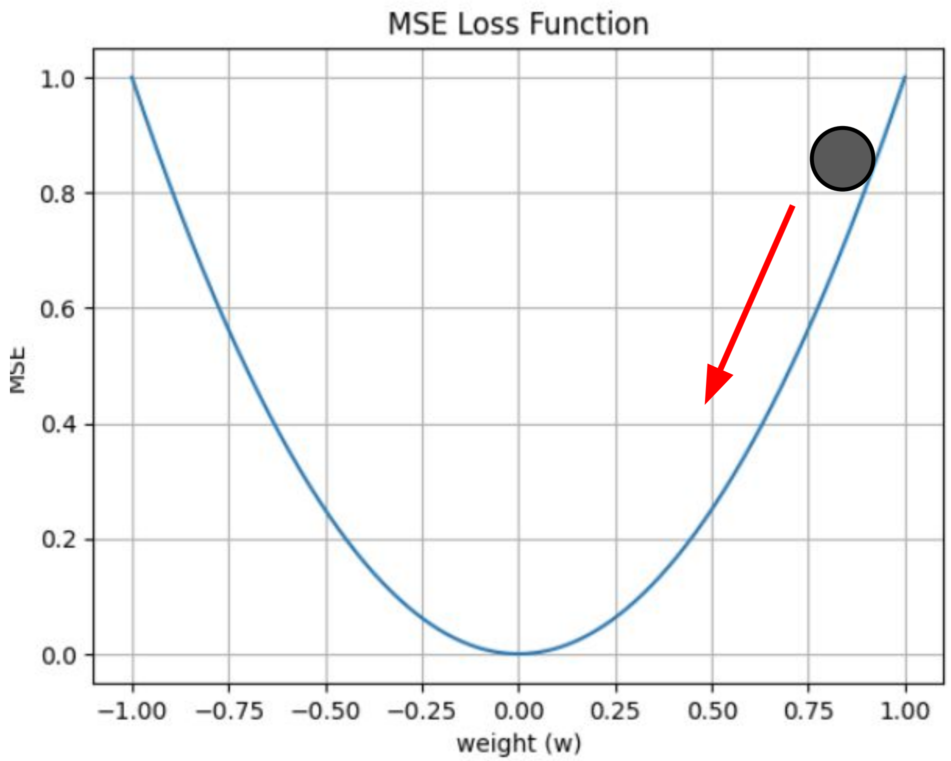


Now, as you can clearly see...

The only variables that really **“changes”** the MSE is the **weight** and **bias!**

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - (wx + b))$$

More with the MSE Function



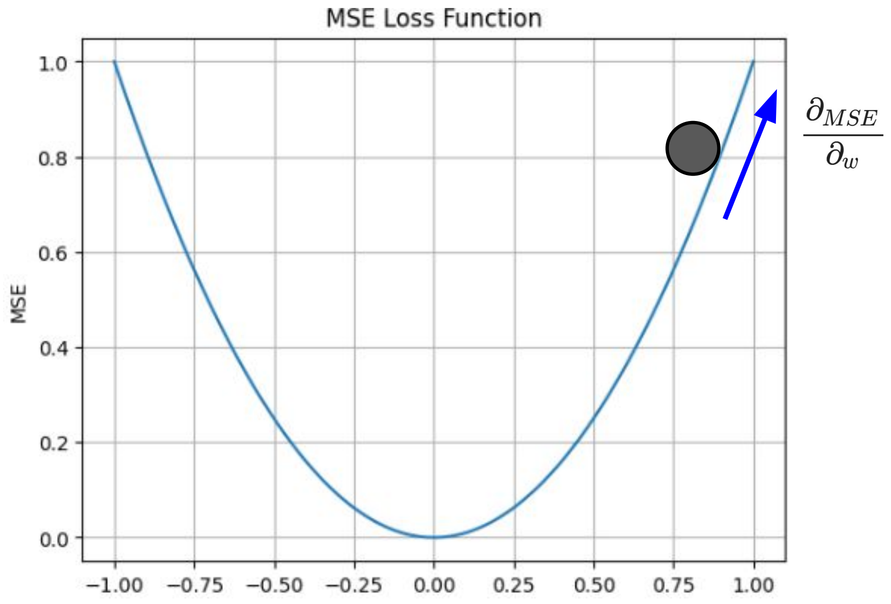
If we only look at MSE loss as a function of the **weight**, it's a **Quadratic Function**

Since you'll start off randomly, let's say you're here with your loss

Our objective is to **minimize** the MSE Loss. This indicates the **minimum average error** between predictions and true values

We want to **nudge the loss downwards** towards the minimum

Introduction to the Gradient



The **Gradient** is made from **partial derivatives.** It gives you the direction that the function **increases** the most.

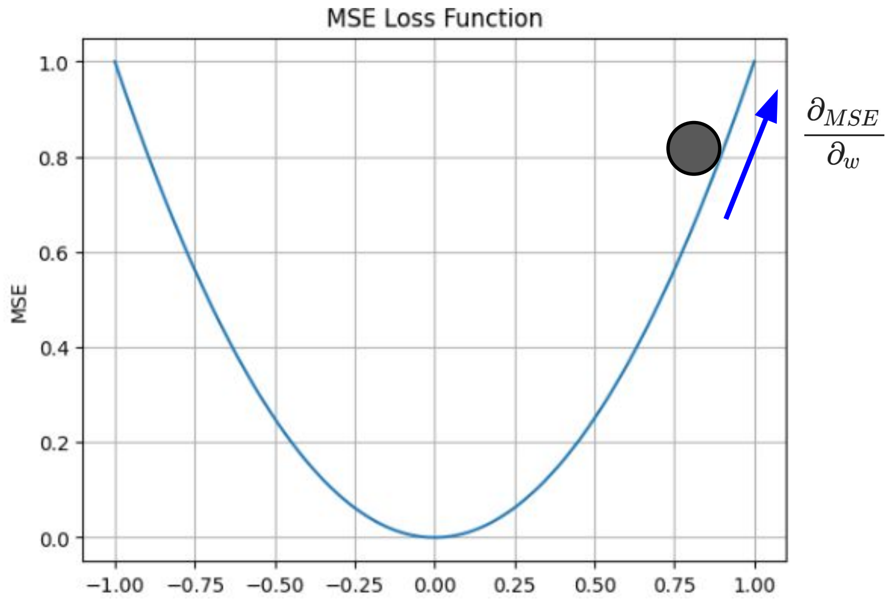
$$\frac{\partial_{MSE}}{\partial_w}$$

So, this is the **partial derivative** of the MSE Loss with respect to the **weight**

$$\frac{\partial_{MSE}}{\partial_b}$$

This is for the **bias.**

Introduction to the Gradient

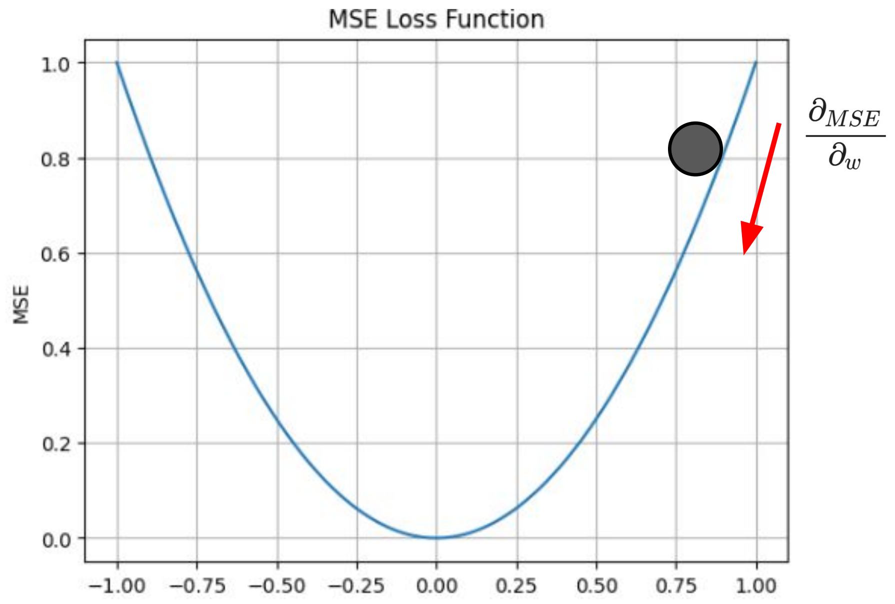


So if the horizontal axis is the **weight**. If you take a small step in this direction, the MSE Loss would go **UP.**

So for our MSE Loss to go **DOWN**

What would you do?

The Update Rule



Take a step in the **opposite direction!**

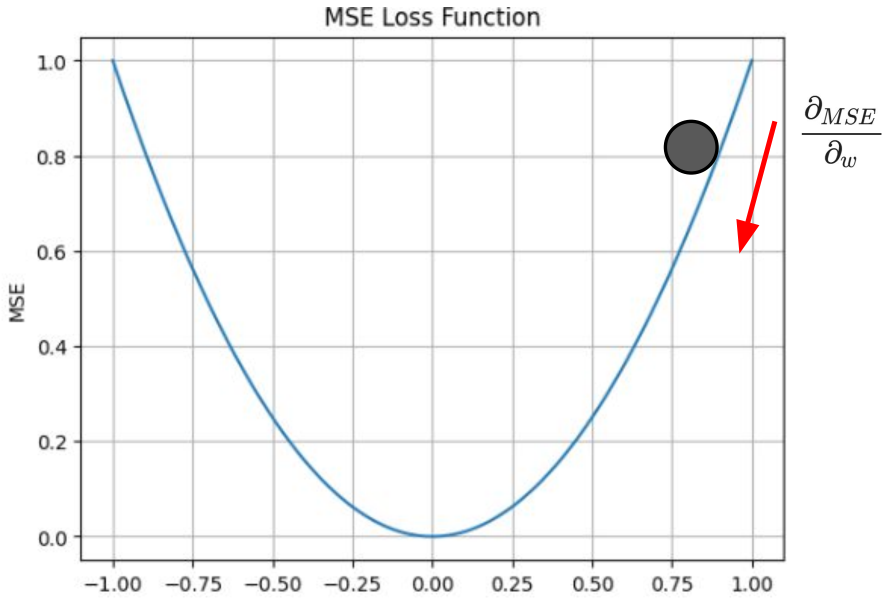
This is just **decreasing** the weight. If the derivative pointed downwards, then you need to **increase** the weight.

To do that mathematically, we use an **update rule!**

$$w := w - \alpha \cdot \frac{\partial \text{MSE}}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

The algorithm we are doing



$$w := w - \alpha \cdot \frac{\partial \text{MSE}}{\partial w}$$

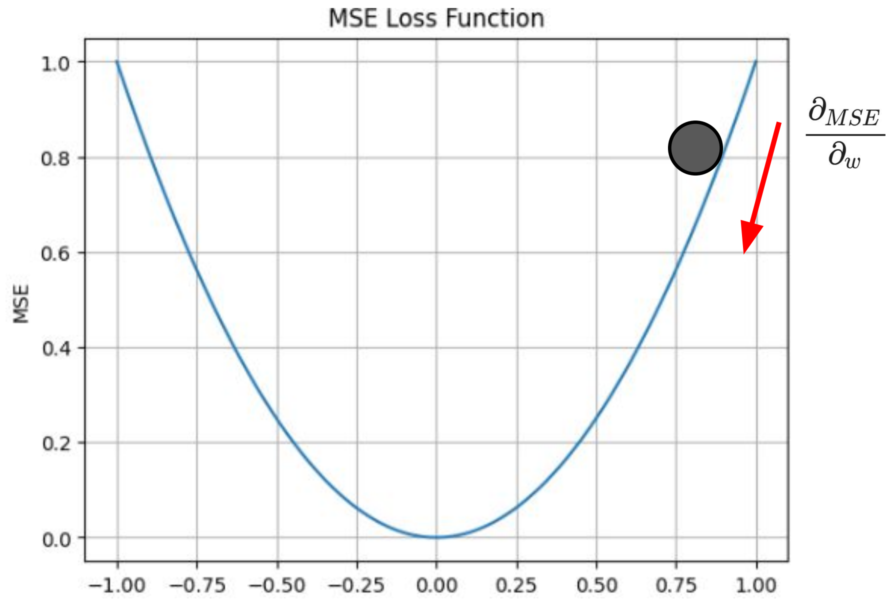
$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

Algorithmically, we:

1. Take **partial derivatives** of MSE wrt to weight and bias
2. Update the based on the formula above!
3. Repeat until we are satisfied with results

But wait...

α, the Learning Rate



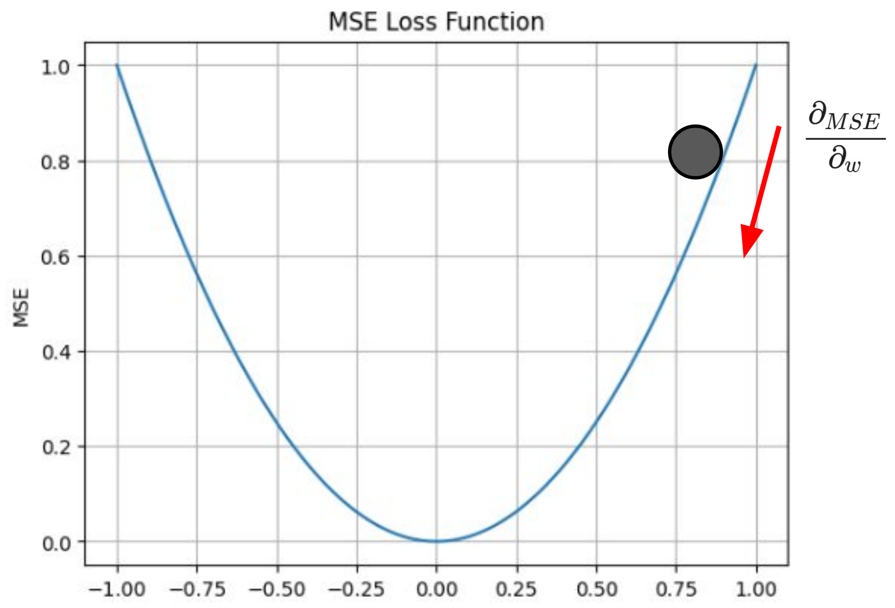
$$w := w - \alpha \cdot \frac{\partial \text{MSE}}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

The α is the **learning rate** — how much we **“nudge”** the MSE loss towards the directions of a lower loss

WE get to choose this number! This is a **hyperparameter** of our model.

The importance of α

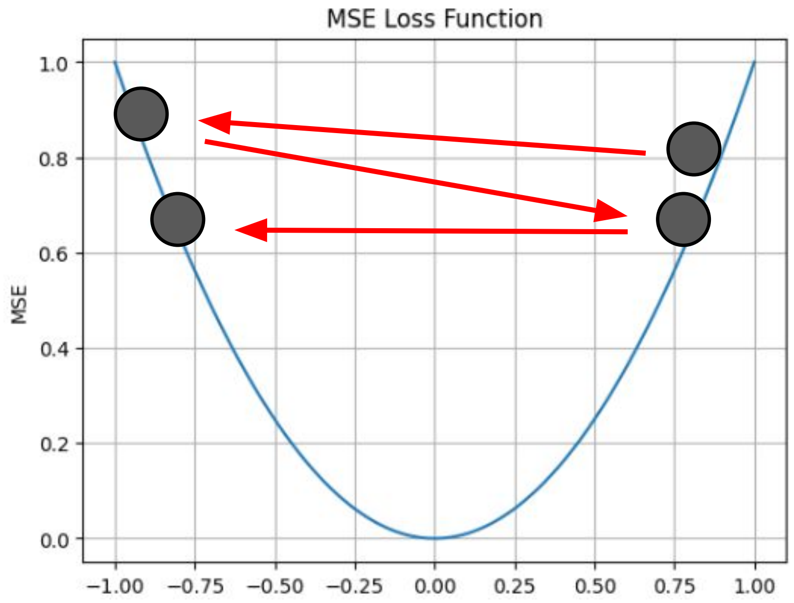


$$w := w - \alpha \cdot \frac{\partial \text{MSE}}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

If α is **too large**, you'd be **overshooting,** you may **never converge** to the minimum!

The importance of α

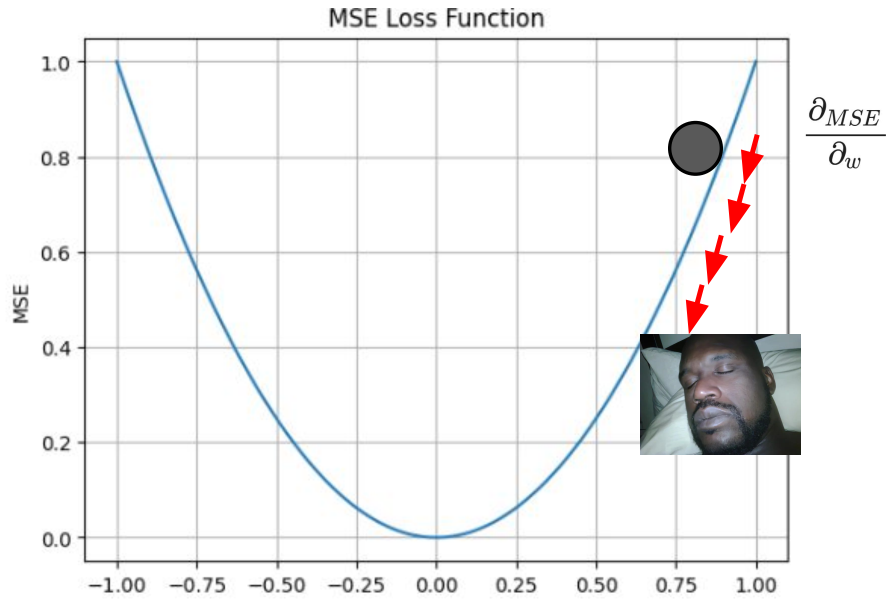


$$w := w - \alpha \cdot \frac{\partial \mathrm{MSE}}{\partial w}$$

$$b := b - \alpha \cdot \frac{\partial \mathrm{MSE}}{\partial b}$$

If α is **too large**, you'd be **overshooting,** you may **never converge** to the minimum!

The importance of α



$$w := w - \alpha \cdot \frac{\partial \text{MSE}}{\partial w}$$

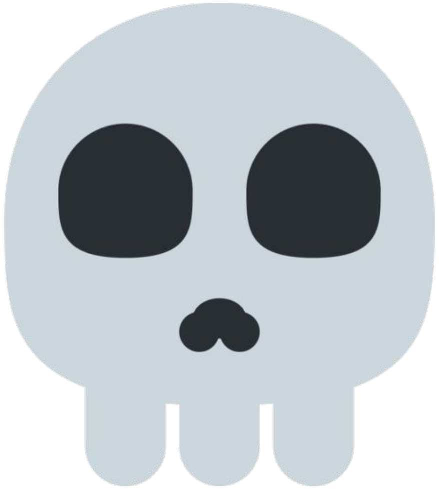
$$b := b - \alpha \cdot \frac{\partial \text{MSE}}{\partial b}$$

If α is **too large**, you'd be **overshooting,** you may **never converge** to the minimum!

If α is **too small,** you will be taking **way too long!**

You need to do lots of testing and tuning!

Also…Why can’t we just mathematically compute the absolute minimum of the MSE??



For a **small amount of parameters**, yes.

But we’ve only been looking at the **ONE-DIMENSIONAL** case of linear regression!

The chances at internship only depended on **MLF attendances.** So just **one variable.**

More variables → **Way too hard to find minimum!**

Multiple Variables….

Now don’t worry this is not a Math/Calculus Course.

YOU CAN EXPERIENCE THE HARDCORE MATH IN SECOND YEAR INSTEAD.

But I need you to conceptually understand.

Fortunately it's the same concept!

$$y = w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

Then, every weight is updated the same!

$$w_i := w_i - \alpha \frac{\partial_{MSE}}{\partial_{w_i}}$$

Now, you can imagine why it's better to do this **"nudging"** instead of finding the minimum from **tons of parameters,** which models can have **thousands of!**

Good News!

You can do this through coding!

No one does this math by hand

Unless this is APS360 :(

PyTorch Time!

Take out your laptops and open the Jupyter Notebook!

The Limitations

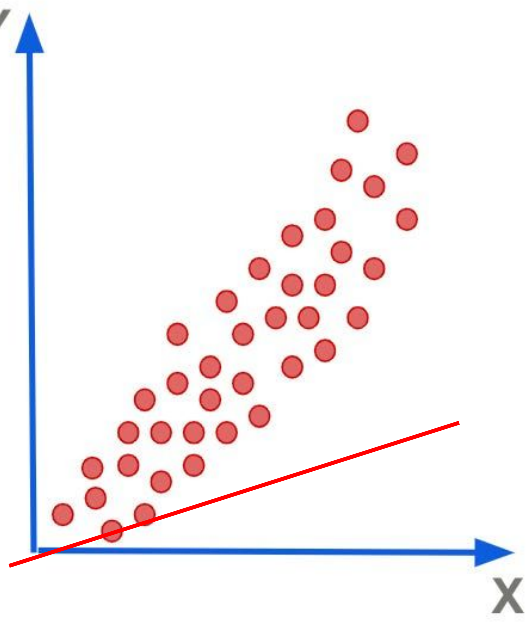
1. Many problems aren’t linear.

(Besides the correlation between landing an internship and going to MLF, of course.)

2. Outliers!

Outliers can heavily skew the results of Gradient Descent.

3. It’s very shallow.



Thank You!