Knowledge Transfer Session - 1

AI
CLUB

What is Machine Learning?

- Machine learning involves building systems that improve their performance by learning from data.
- Instead of hardcoding rules, we train models to recognize patterns and make predictions or decisions.
- These models drive real-world applications, from detecting fraud to translating languages in real time.

The Three Flavors of Learning

Supervised

- The model learns from labeled data, where the correct answers are already known.
- It's used for tasks like predicting house prices or classifying emails as spam.

Unsupervised

- The model explores data without labels, finding hidden patterns or groupings.
- Common use cases include customer segmentation and topic modeling.

Reinforcement

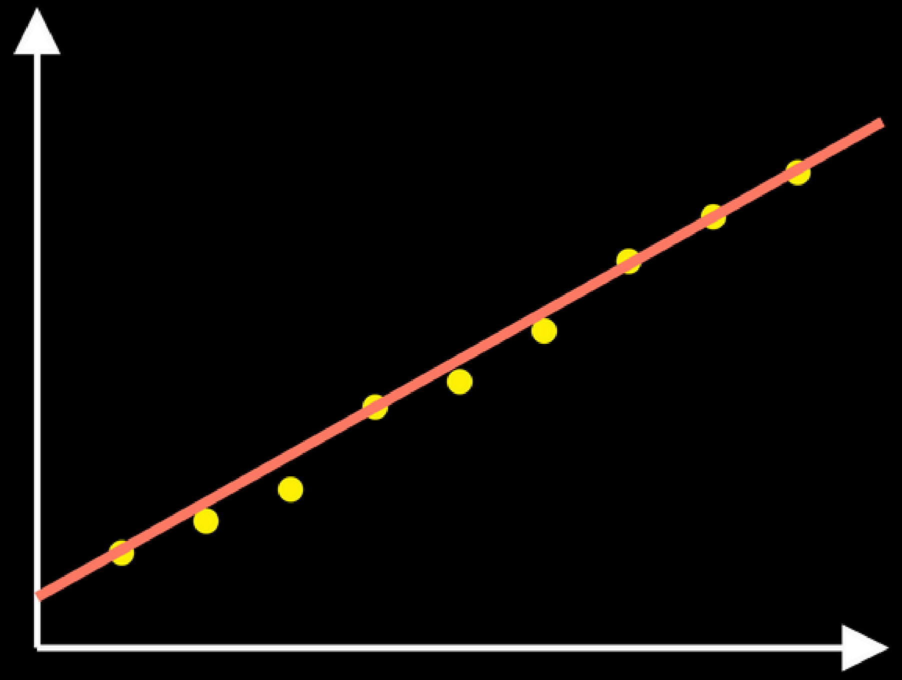
- An agent learns by interacting with an environment, receiving rewards or penalties based on its actions.
- It's how models learn to play games or control robots over time.

Linear Regression, Polynomial Regression and Logistic Regression.

Regression vs Classification

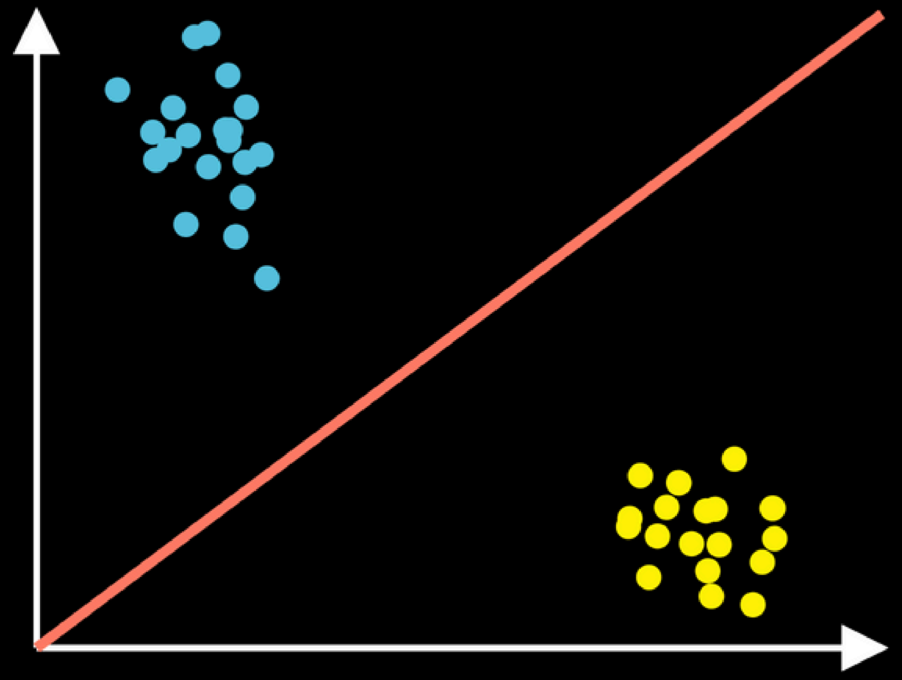
Regression

- Used when the output is a continuous value like predicting temperature, price, or age.
- The model learns to estimate a number based on input features.



Classification

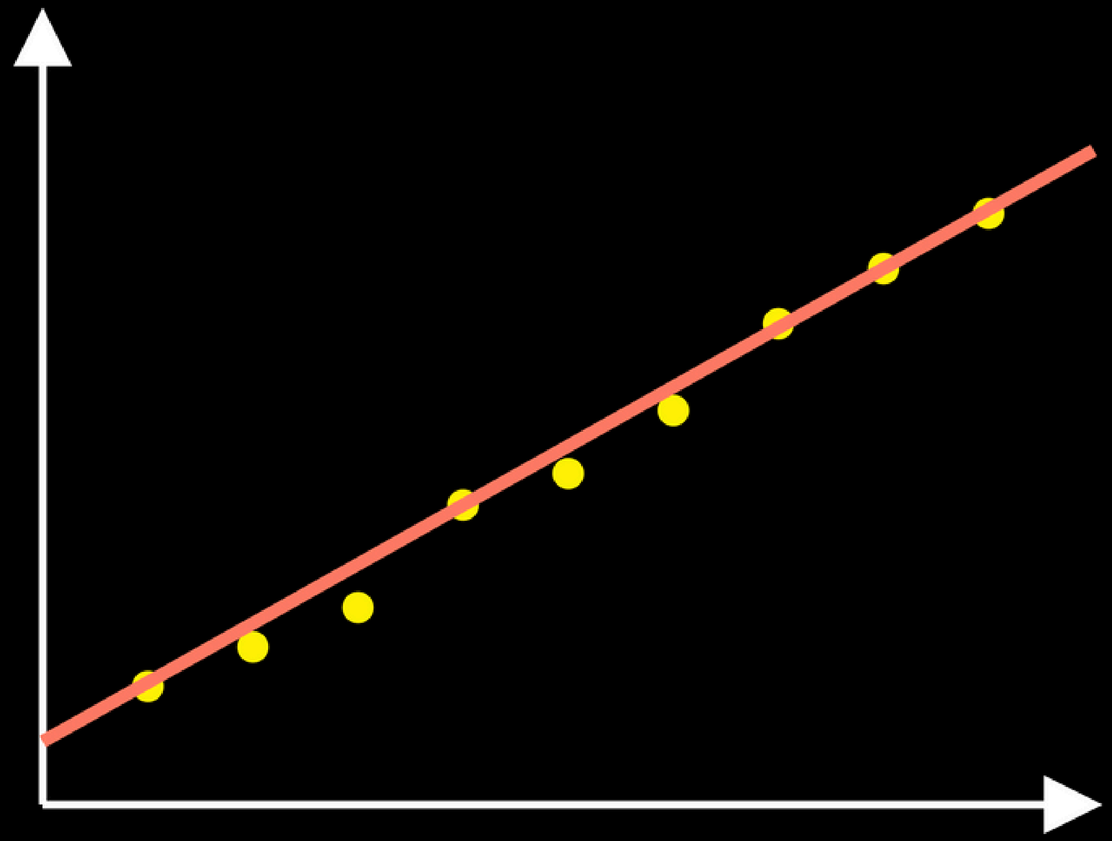
- Used when the output is a category or class like spam vs. not spam, or cat vs. dog.
- The model learns to assign inputs to discrete labels.



Linear Regression

Linear Regression

- Linear regression is a supervised learning algorithm used for predicting continuous outcomes.
- It learns weights θ by minimizing a loss function, typically Mean Squared Error, using optimization methods like Gradient Descent.
- Though simple, it's foundational model that helps build intuition for more complex algorithms in machine learning.



Defining the Model

- We're essentially fitting a line to the data, so we get an equation of the form:

$$\hat{y} = wx + b$$

where:

x : input features

w : slope of the line (also called "weight")

b : y intercept of the line (also called "bias")

$\hat{y}$: value predicted by the model

How do we measure how well the model is performing?

Loss Function

- To measure how poorly the model is performing and guide its improvement, we use a loss function.
- This function, defined in terms of the model's parameters, quantifies the average error between the predicted and actual values.
- The goal of any machine learning algorithm is to minimize the loss function.
- There are various loss functions that can be used depending on the model and data..

- In the case of linear regression, we use the Mean Squared Error (MSE) function to measure the loss (cost)

Mean Squared Error (MSE)

- The Mean Squared Error (MSE) measures the average of the squared differences between predicted and actual values, penalizing larger errors more heavily.
- Mathematically, it is given by:

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

where:

n : number of data points

$x^{(i)}, y^{(i)}$: input and true output of the i th data point

$\hat{y}^{(i)}$: model prediction

$J(w,b)$: the loss function (cost) we want to minimize

Now that we know the loss,
how do we minimize it?

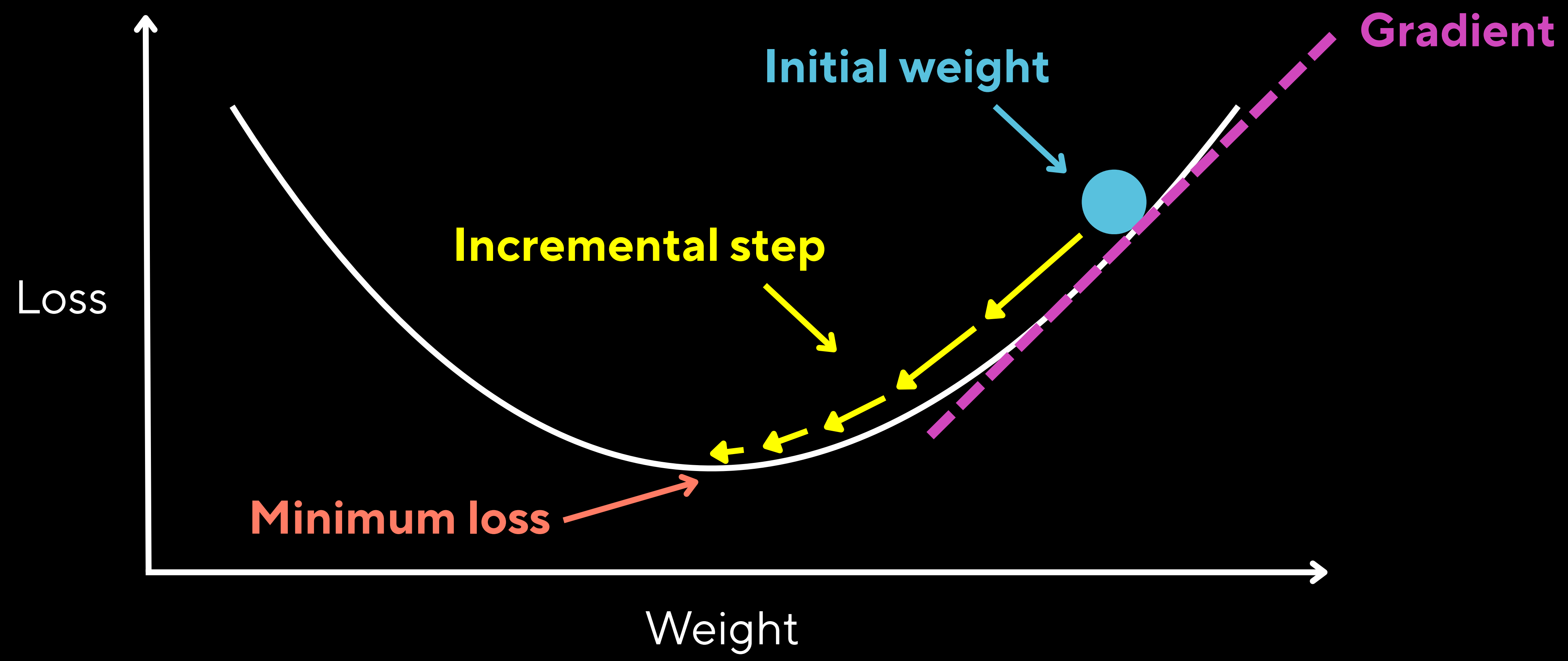
Turning Errors into Learning

- Optimization is the process of adjusting a model's parameters to minimize the loss function.
- It uses mathematical techniques to navigate the error landscape efficiently and acts as a guide by helping the model decide how to update the parameters.
- Essentially, optimization turns feedback (loss) into learning.

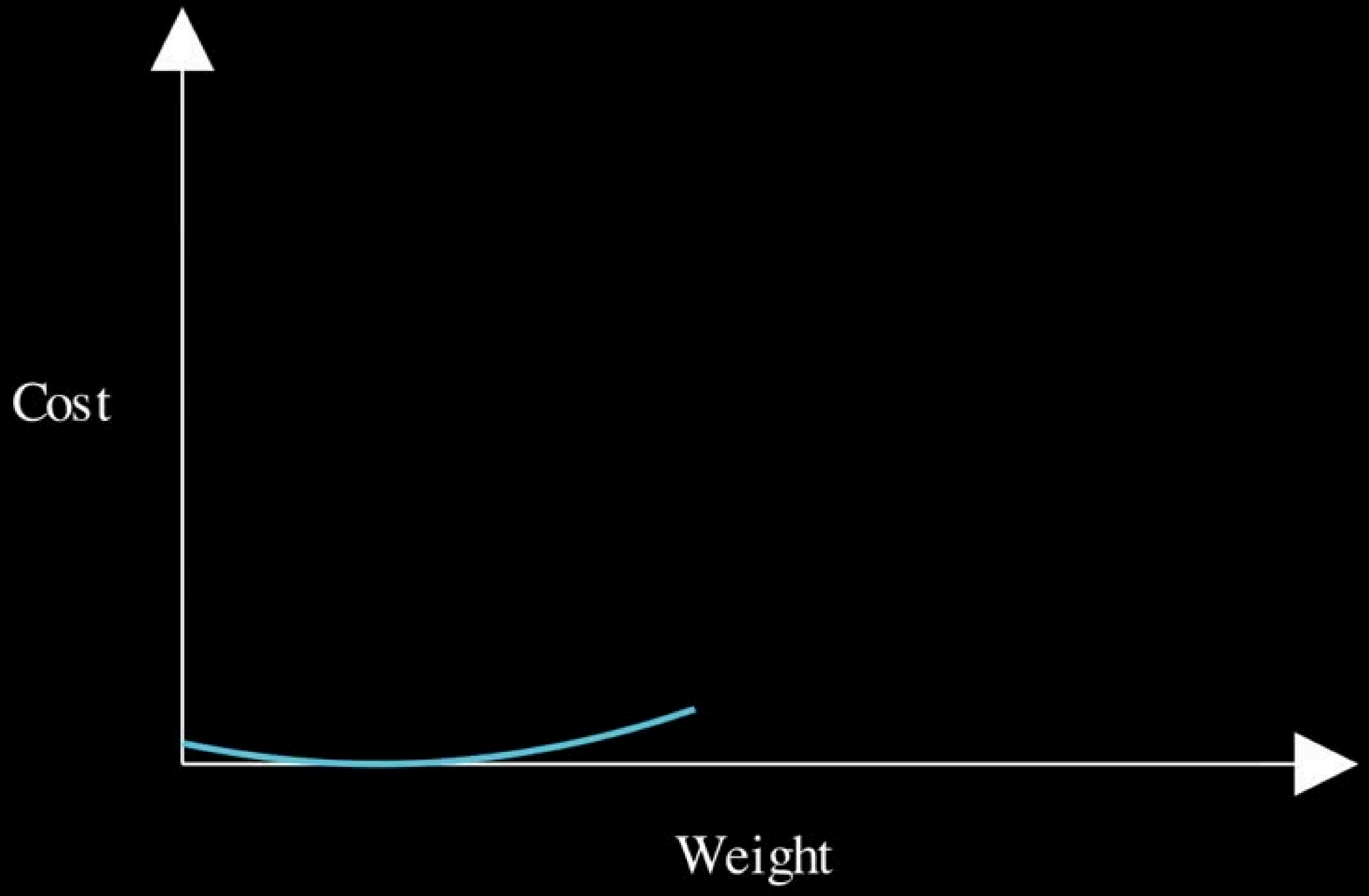
- In case of linear regression, we use the gradient descent method.

Gradient Descent

- Gradient Descent is an optimization algorithm in which we try to reach the minima of the cost function by iteratively moving in the direction of steepest descent.
- During each iteration, we compute the gradient at the current point.
- Since the gradient gives the direction of steepest ascent, we move in the opposite direction with a step size to reach the minimum.
- This process is repeated till we converge to the global minimum.



Gradient Descent in Action



The Math behind Gradient Descent

- The cost function is given by:

$$J(w,b) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)^2$$

- Computing the gradient at the current point:

$$\frac{\partial J}{\partial w} = \frac{2}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)x^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}\left(wx^{(i)} + b - y^{(i)}\right)$$

- Updating the weights and biases:

$$w := w - \eta\frac{\partial J}{\partial w}$$

$$b := b - \eta\frac{\partial J}{\partial b}$$

These steps are sequentially repeated until convergence.

The Learning Rate

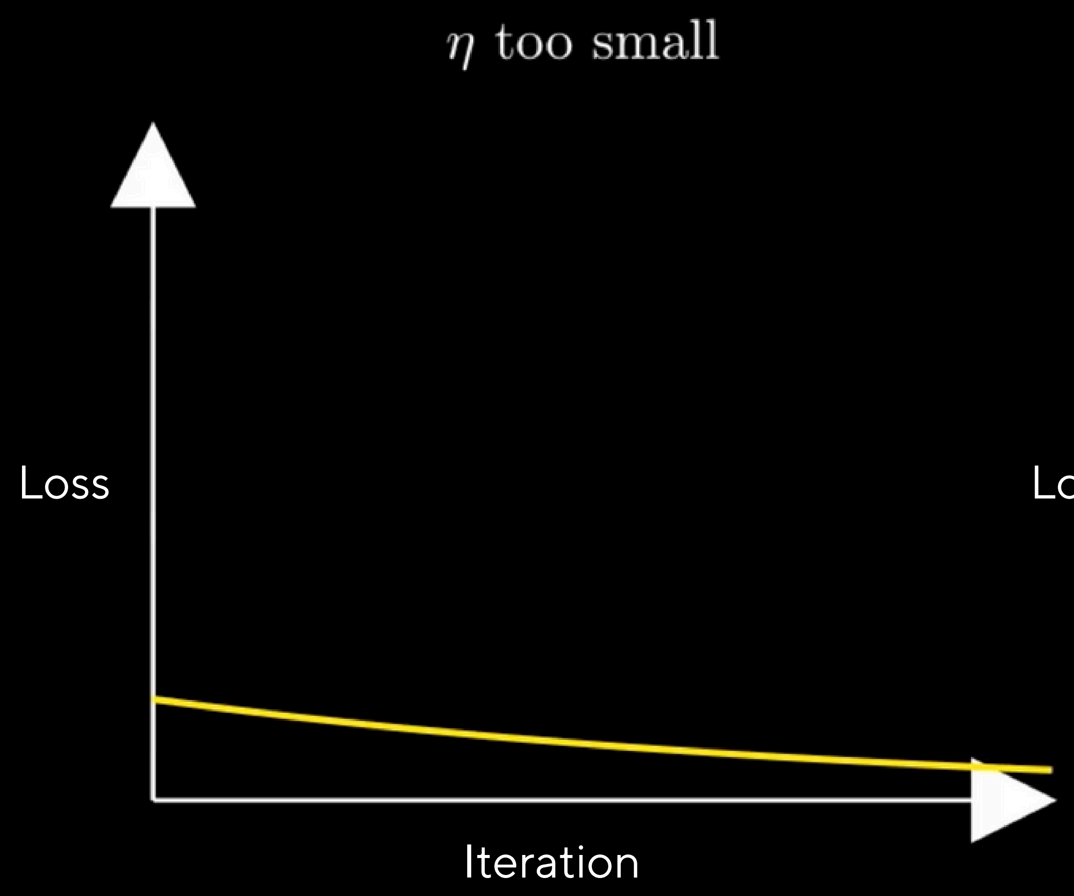
- The learning rate η is a hyper-parameter that determines the size of each step the model takes when minimizing the loss.
- A high learning rate can speed up training but risks overshooting the minimum or diverging entirely.
- A low learning rate ensures more stable updates but can make training very slow or get stuck in local minima.

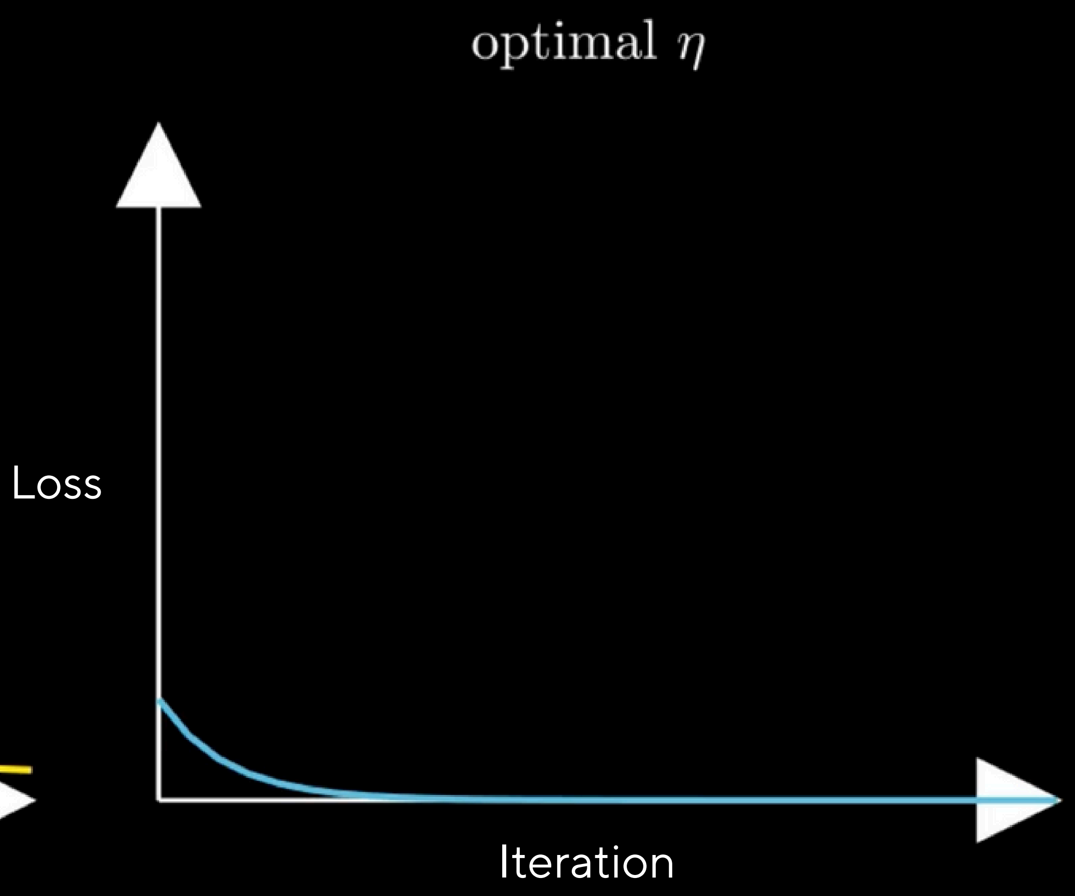
The Learning Rate

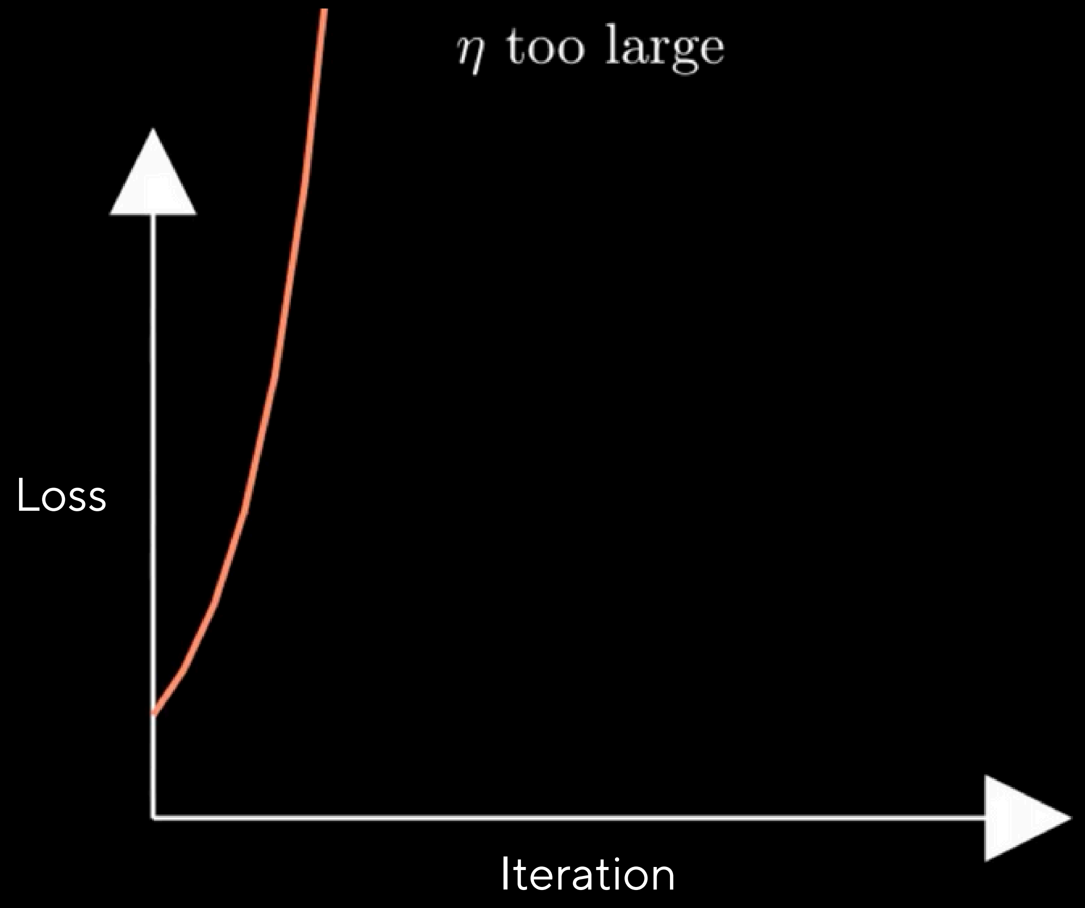
- The learning rate η is a hyper-parameter that determines the size of each step the model takes when minimizing the loss.
- A high learning rate can speed up training but risks overshooting the minimum or diverging entirely.
- A low learning rate ensures more stable updates but can make training very slow or get stuck in local minima.

Finding the Sweet Spot

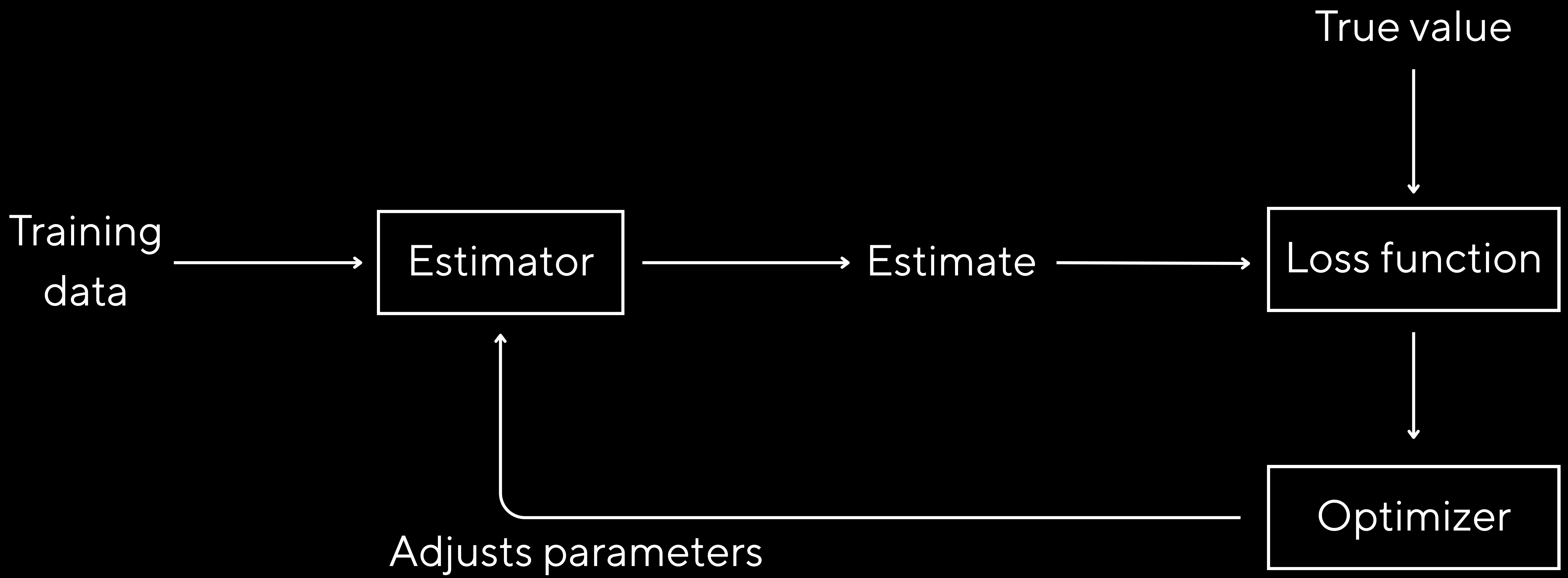
- We plot the Cost function with the number of iterations. This plot is called the learning curve.
- If the chosen learning rate is optimal then the cost function should decrease after every iteration.
- If cost function increases after a single iteration, it means the chosen learning rate is too high

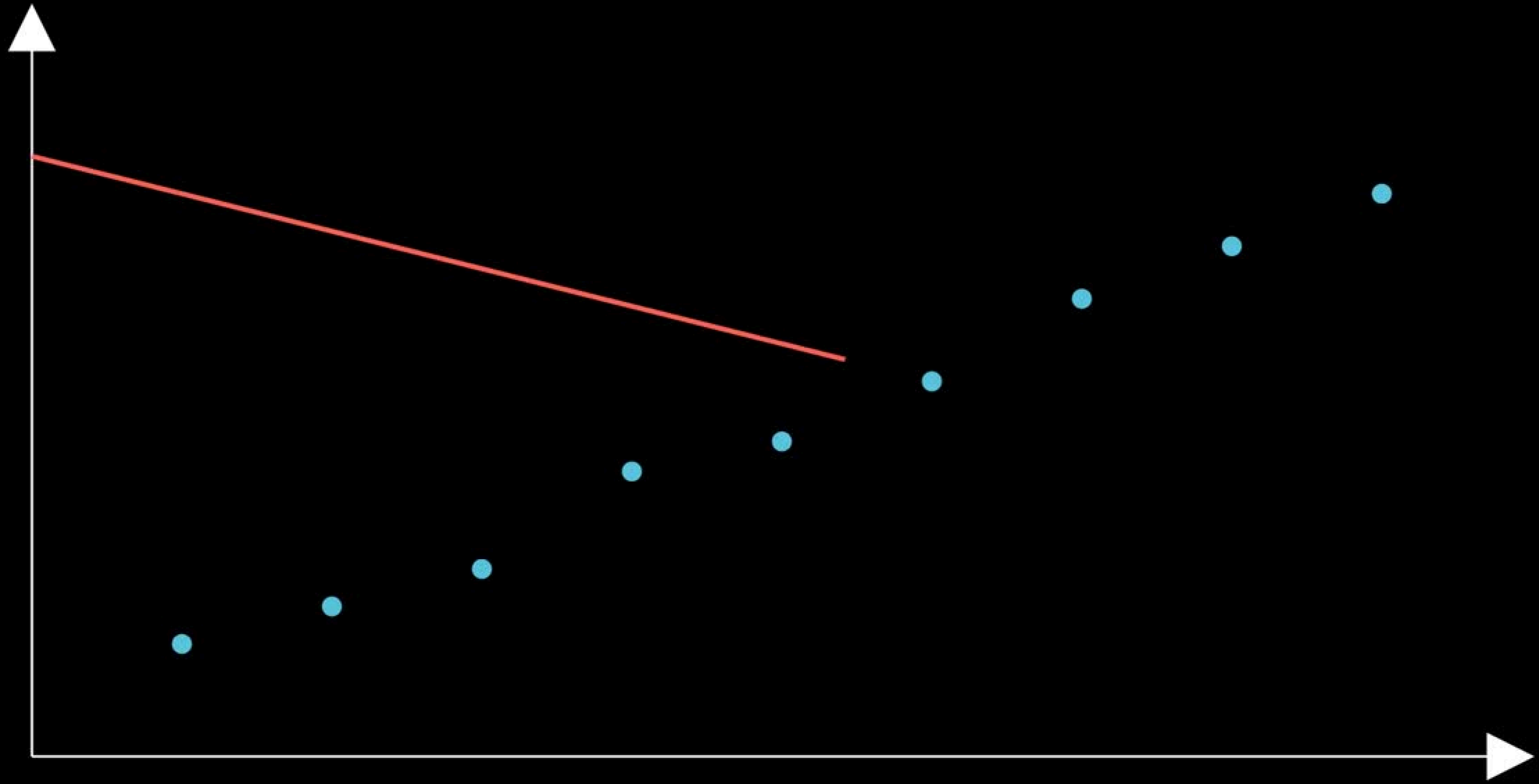






Putting It All Together





But, what if there’s more than one factor?

Multiple Linear Regression

- Multiple linear regression models the relationship between an outcome and several input features.
- Instead of fitting a straight line, it fits a hyperplane that best captures how all the features together influence the prediction.
- The model combines each input with a corresponding weight, adds a bias term, and outputs the result.
- It's used when predictions depend on multiple factors working together.

- Mathematically,

$$\hat{y} = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b$$

Code Implementation

Polynomial Regression

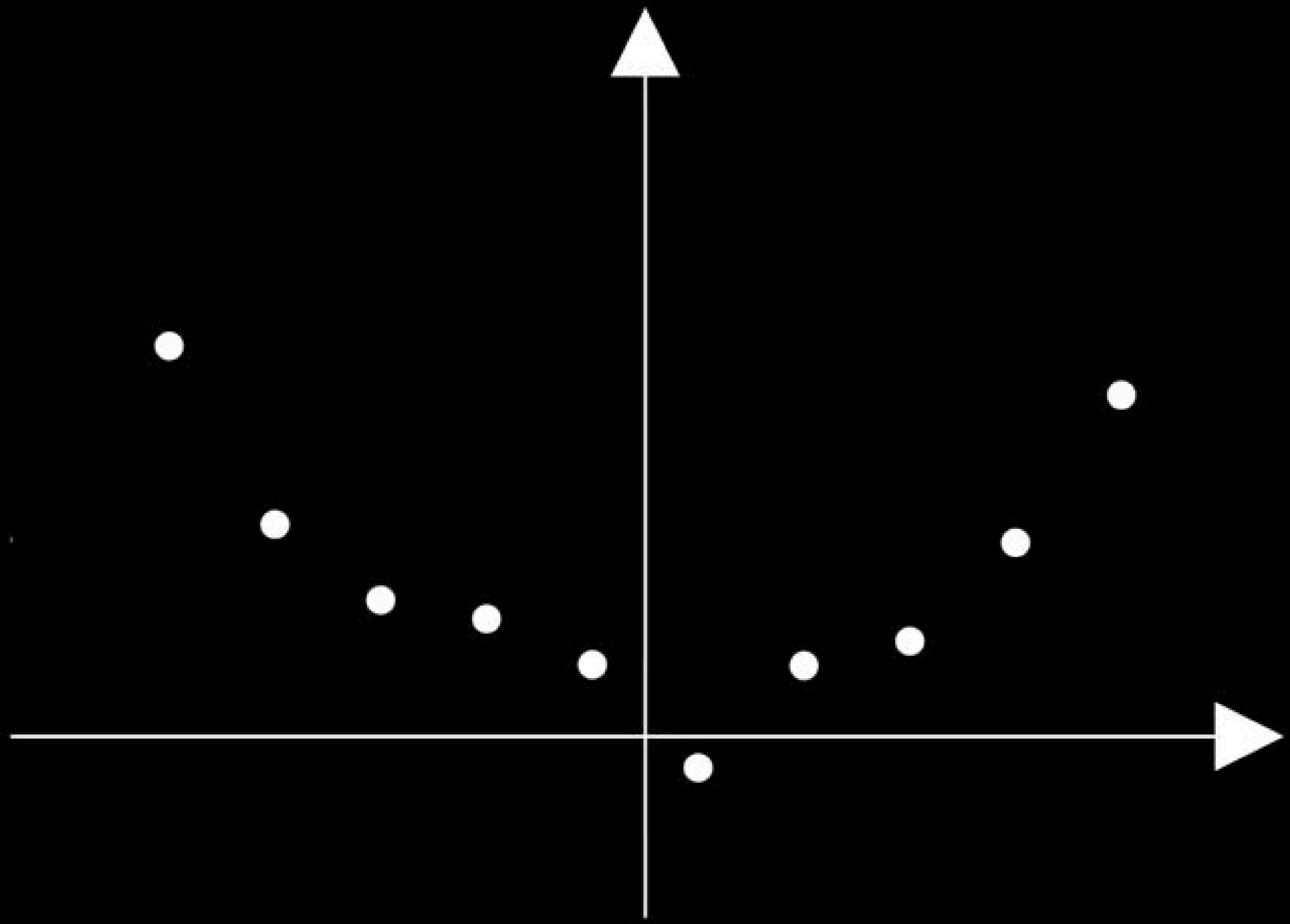
Polynomial Regression

- Polynomial regression builds on linear regression to model non-linear relationships in data.
- Instead of using just the input x , it adds powers like x^2, x^3, and so on, creating a more flexible model.
- This allows it to fit curves that a straight line couldn't capture.

- Mathematically,

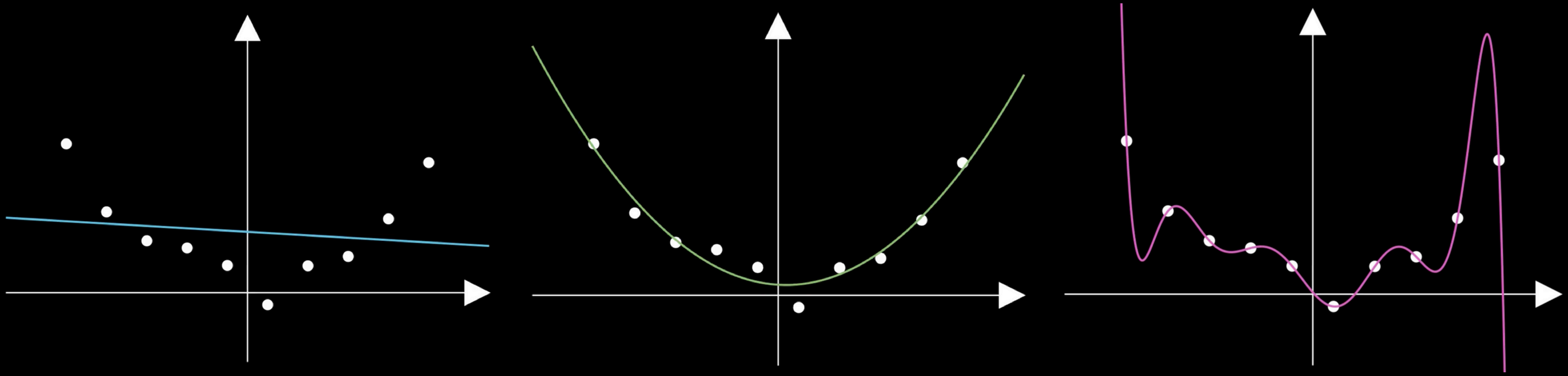
$$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots$$

- Polynomial regression is a special case of multiple linear regression where the input features are nonlinear transformations (powers) of the original input variables.



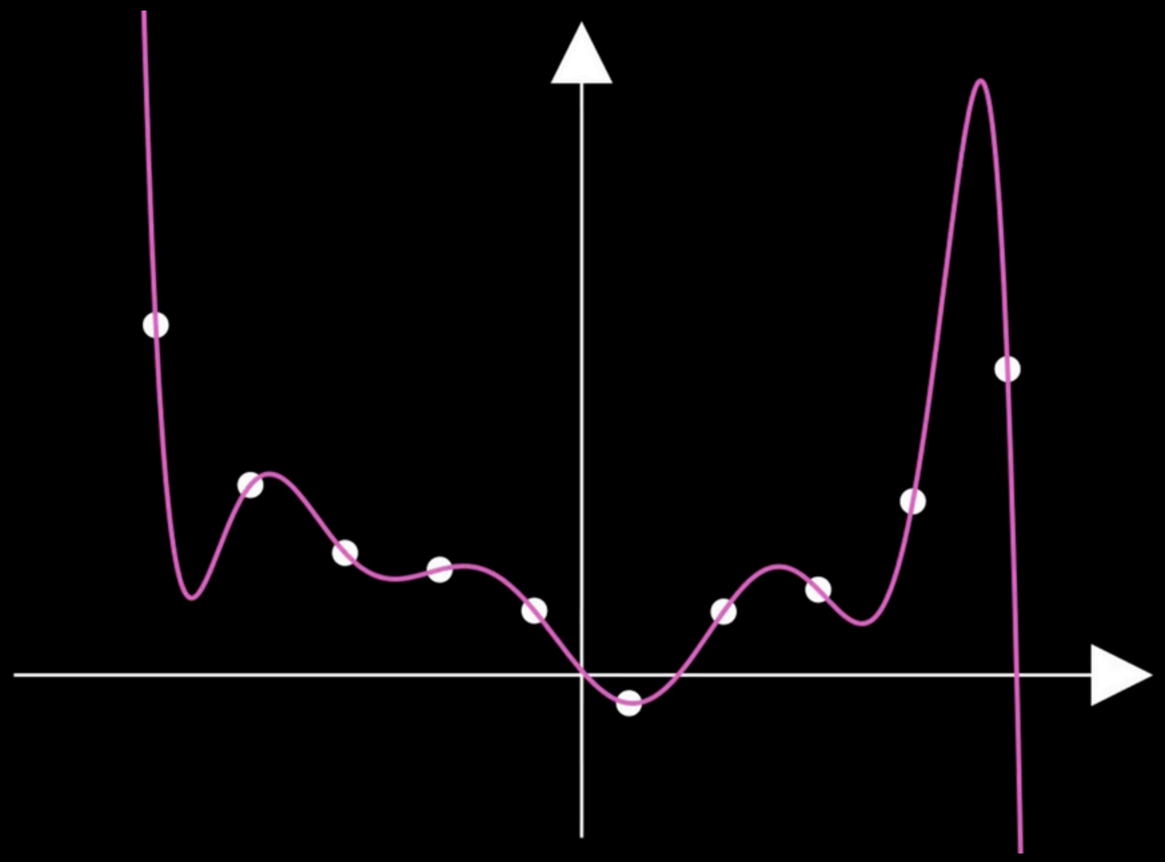
Too Much Flexibility Can Be a Problem

- While polynomial regression can model complex, non-linear patterns, it's highly prone to overfitting, especially with higher-degree terms.
- The model may capture noise in the data rather than true underlying trends.
- It's also sensitive to outliers and can behave erratically when making predictions outside the training range.



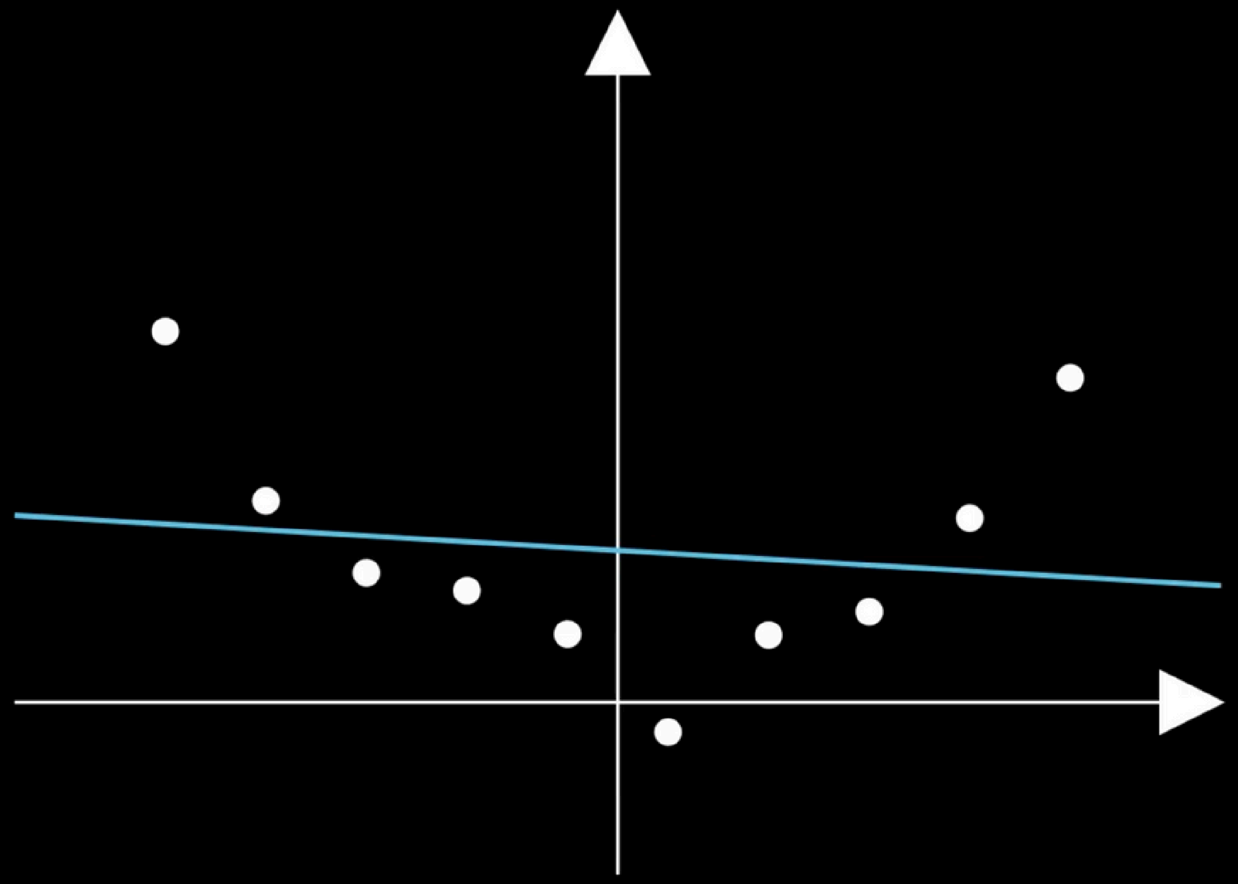
When the Model Tries Too Hard

- Overfitting occurs when a model learns not just the underlying patterns but also the noise in the training data.
- As a result, it performs well on the training set but poorly on unseen data.
- This usually happens when the model is too complex relative to the amount of data available.



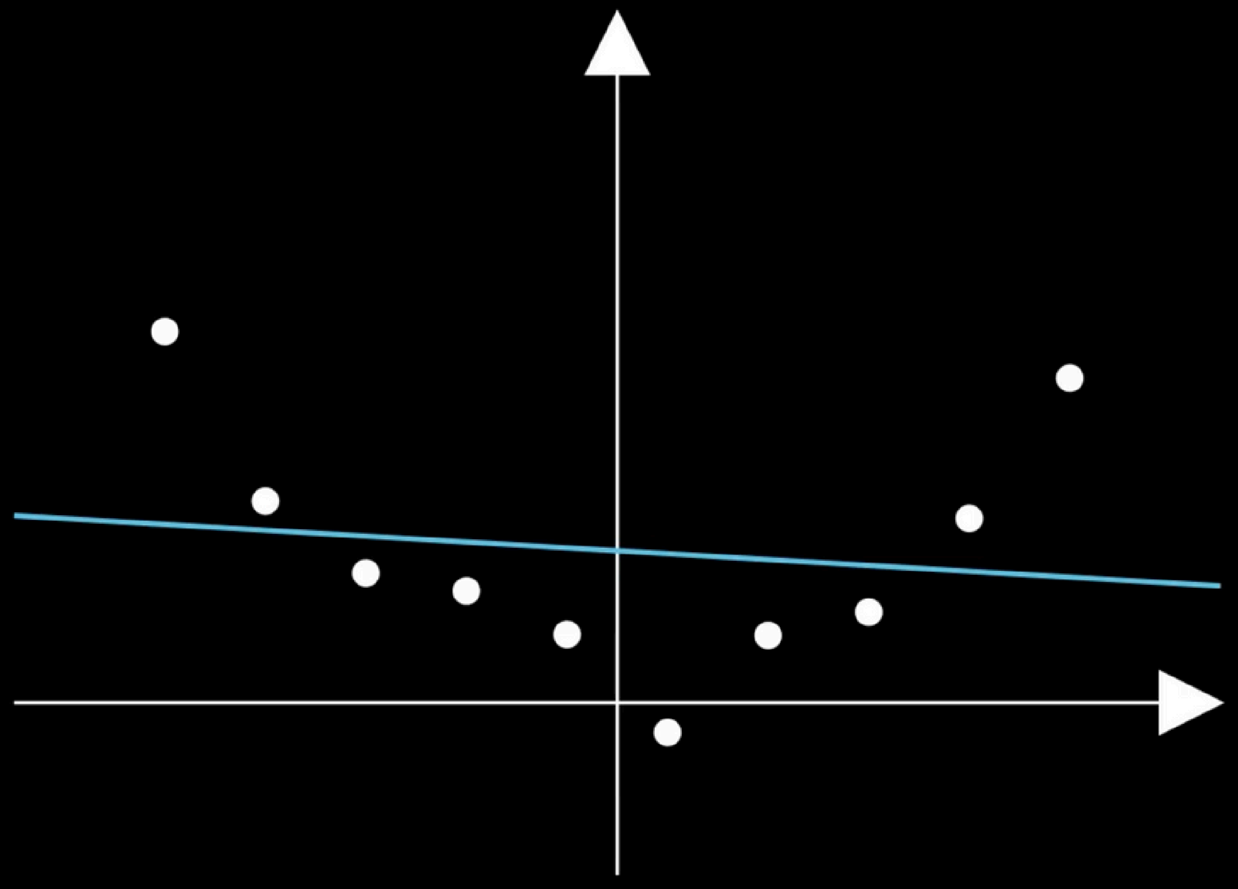
When the Model Doesn't Even Try

- Underfitting happens when a model fails to learn important patterns, resulting in poor performance on both the training and test sets.
- This often occurs when the model lacks sufficient capacity or the features are not informative enough.
- Addressing underfitting typically involves using more complex models or better feature engineering.
- A key goal in machine learning is to find the right balance between underfitting and overfitting for good generalization.



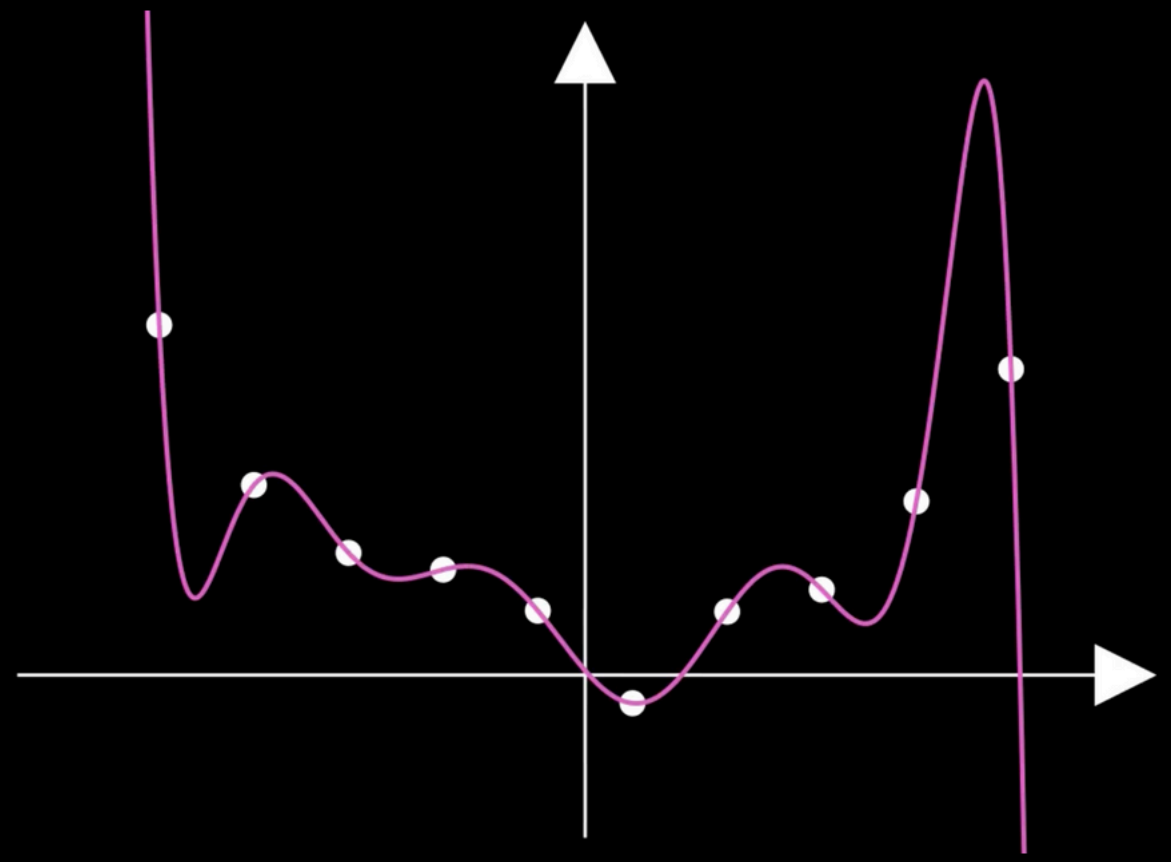
Bias: Oversimplifying the World

- Bias refers to the error introduced by approximating a real-world problem with a simplified model.
- A high bias model makes strong assumptions about the data and often underfits, missing important patterns.
- A high bias means that the model is too simple , hence it is not able to capture important features or patterns from the dataset.
- Reducing bias usually involves choosing more flexible models or improving the feature set.


High bias

Variance: Jumping at Every Noise

- Variance refers to a model's sensitivity to small fluctuations in the training data.
- High variance models capture noise along with the underlying patterns, leading to overfitting.
- They perform well on training data but poorly on unseen data due to lack of generalization.
- Reducing variance often involves simplifying the model or using regularization techniques.


High variance

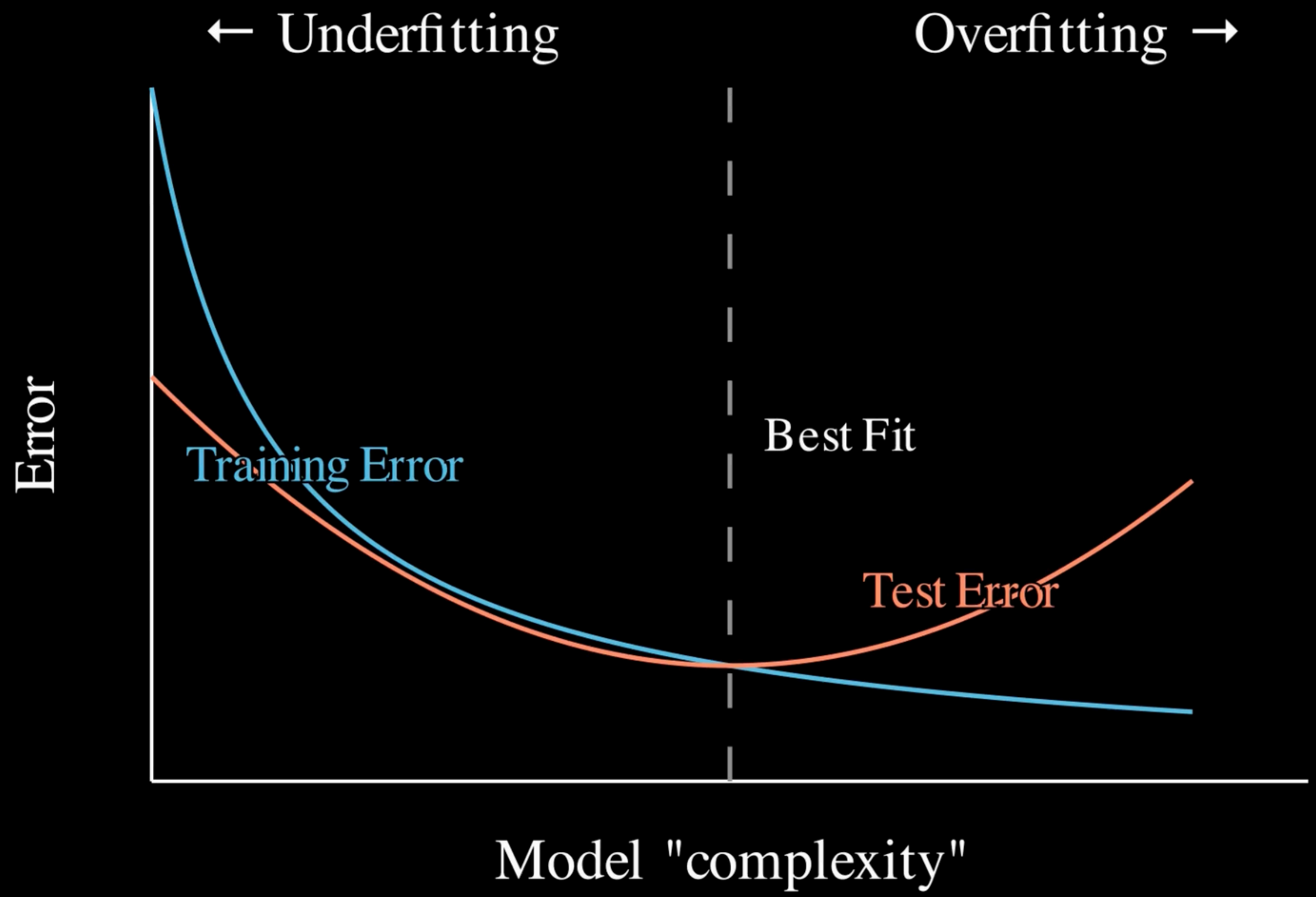
Bias-Variance Tradeoff

- High bias leads to underfitting, where the model is too simple to capture the data's complexity.
- High variance leads to overfitting, where the model captures noise rather than general patterns.
- Improving one often worsens the other, so the goal is to find the sweet spot that minimizes total error.
- Achieving this balance ensures good performance on both training and unseen data.

This brings us to the necessity of optimizing bias and variance.

Optimizing Bias and Variance

- The core idea is to plot the cost function for every degree of x for the testing data. The minima thus found is the optimal order of the polynomial in order to balance bias and variance.



Code Implementation

But Linear Regression Can’t Classify...

- Linear regression works great for predicting continuous values like prices, temperatures, or scores. But what if we want to predict something binary? Like whether an email is spam, or a tumor is malignant.
- Linear regression can’t handle this; it outputs unbounded numbers and doesn't model probability. It might predict 1.4 for a cat and −0.3 for a dog — which makes no sense.
- We need a model that outputs values between 0 and 1, interpretable as probabilities, and separates classes clearly.

That’s where Logistic Regression comes in.

Logistic Regression

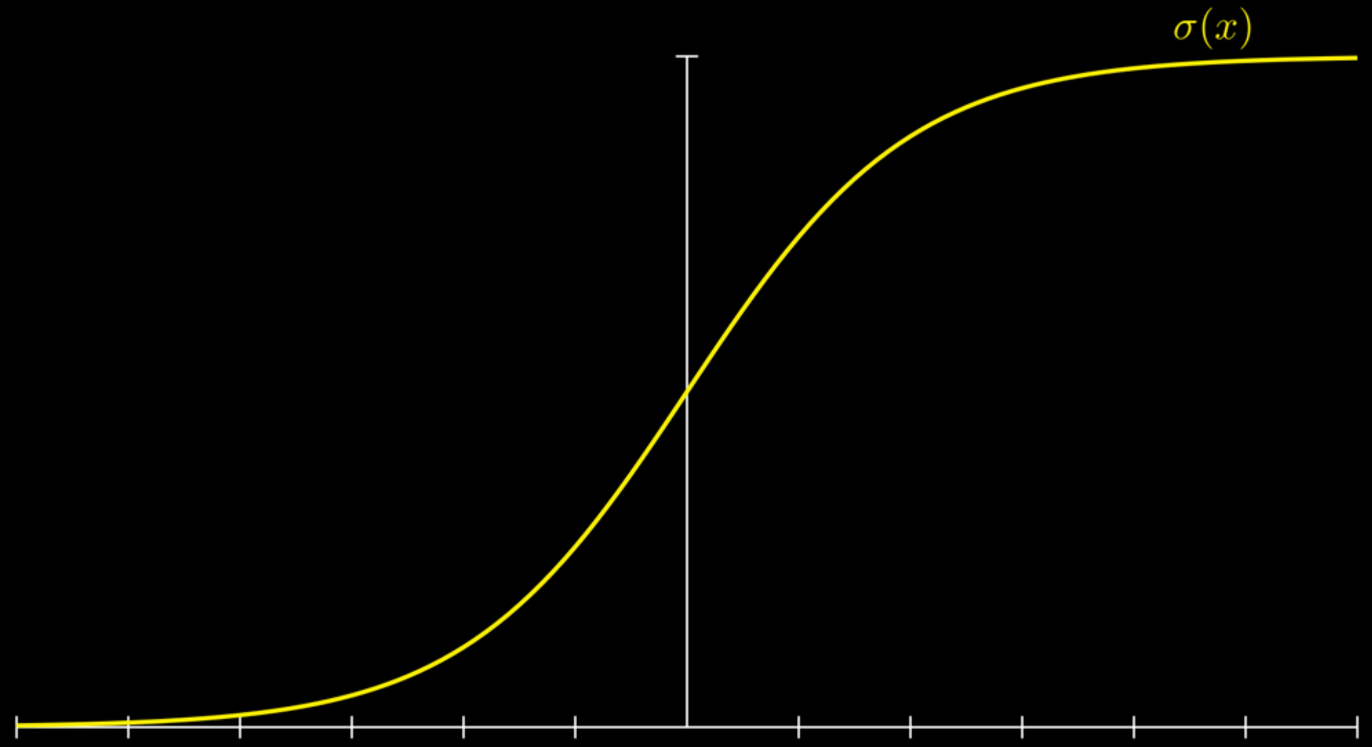
From Lines to Classes

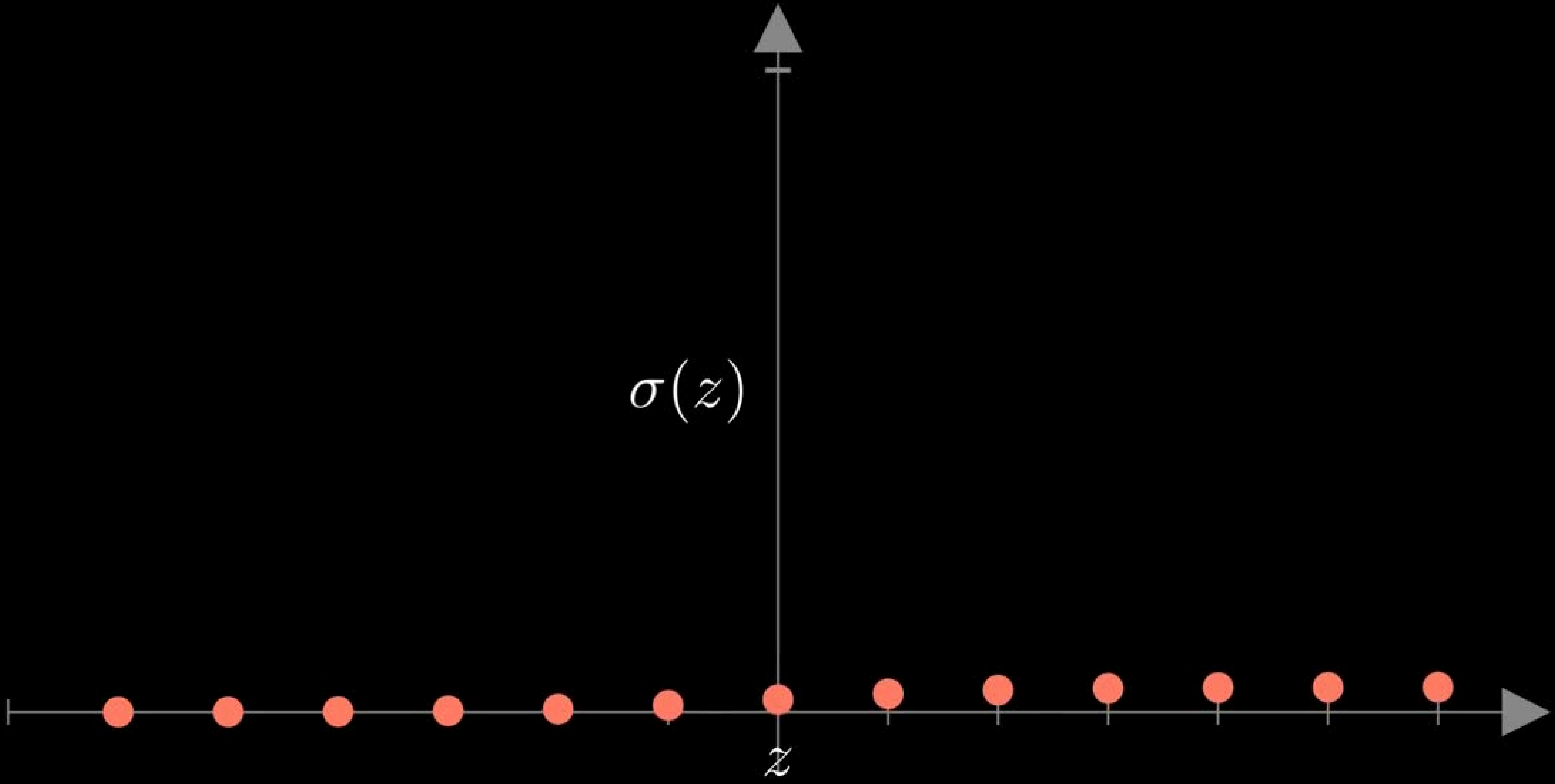
- Logistic Regression is used when the target variable is categorical, typically binary (like yes/no or 0/1).
- Instead of fitting a straight line, it models the probability of a class using the sigmoid function.
- It outputs values between 0 and 1, which can be interpreted as class probabilities.
- Though it’s called "regression," it's actually used for classification tasks.

The Sigmoid or Logistic Function

- The sigmoid function converts the linear output into a probability between 0 and 1.
- It acts like a soft threshold, values far below 0 map near 0, and values far above 0 map near 1.
- This allows logistic regression to make smooth, probabilistic decisions for classification.

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




$\sigma(z)$
z

MSE for Logistic Regression?

- We can’t use MSE for Logistic Regression. The reasons being:

1. Non-convex Loss: MSE + sigmoid makes the loss surface non-convex → gradient descent struggles to find the global minimum.
2. Bad Gradients: Squared error punishes large confident predictions (even if they're correct), leading to inefficient or misleading gradients.
3. Not Probabilistically Sound: Logistic regression models probabilities → MSE doesn’t align with this.

Instead, we use Binary Cross-Entropy, which provides better gradients and a convex loss landscape.

Binary Cross-Entropy

- Binary Cross-Entropy is a loss function that measures the difference between the predicted probability and the actual class label (0 or 1).
- Mathematically, it is given by:

$$\mathcal{L} = -\left[y \log(\hat{y}) + (1-y)\log(1-\hat{y})\right]$$

where:

y : the true label
$\hat{y}$: the predicted probability

Advantages of BCE

- Binary cross-entropy has the following advantages:

1. Probabilistic Fit: Logistic regression predicts probabilities — cross-entropy directly measures how well those match the true labels.
2. Convex Landscape: Leads to a smooth, convex loss surface → gradient descent works reliably.
3. Sharp Penalty for Confident Mistakes: Encourages the model to assign high confidence only when it's right.

Optimization

- Since the loss function is convex, we can use Gradient Descent to update the parameters.

Model:

$$\hat{y} = \sigma(wx + b), \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

Binary Cross-Entropy loss:

$$\mathcal{L} = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right]$$

Gradients:

$$\frac{\partial \mathcal{L}}{\partial w} = (\hat{y} - y)x \qquad \frac{\partial \mathcal{L}}{\partial b} = \hat{y} - y$$

Parameter updates:

$$w := w - \alpha \cdot \frac{\partial \mathcal{L}}{\partial w} \qquad b := b - \alpha \cdot \frac{\partial \mathcal{L}}{\partial b}$$

Why Logistic “Regression”?

- **It's a regression model... technically:** Logistic regression models the probability that an input belongs to a class. Specifically, it predicts a continuous value between 0 and 1 (like 0.93 or 0.12), not a label directly. That’s regression behavior.
- **The classification step is post-processing:** We threshold the predicted probability to get a class label (e.g., if $P>0.5$, predict class 1). But the model itself doesn’t "know" it’s classifying, it’s just fitting probabilities via regression.

But, what if there's more than two output categories?

Multi-Class Regression

- Multiclass logistic regression extends binary logistic regression to handle more than two classes.
- Instead of using a sigmoid, it uses the softmax function to output probabilities across all possible classes, ensuring they sum to 1.
- Each class gets its own set of weights, and the model learns to assign higher probabilities to the correct class.
- The loss function used is categorical cross-entropy, which penalizes incorrect predictions based on probability confidence.

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

$$L = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{K} y_i^{(n)} \log(\hat{y}_i^{(n)})$$

How it Works

$$\begin{tikzcd}
 & \boxed{w1.x + b1} \arrow[r] & z1 \arrow[r, "\text{softmax}"'] & p1 \\
\text{Training data (x)} \arrow[r] & \boxed{w2.x + b2} \arrow[r] & z2 \arrow[r, "\text{softmax}"'] & p2 \\
 & \boxed{w3.x + b3} \arrow[r] & z3 \arrow[r, "\text{softmax}"'] & p3
\end{tikzcd}$$

Code Implementation

Thank you!

AI
CLUB