MACHINE LEARNING

Shoaib Farooq

Department of Computer Science

Disclaimer: This presentation includes contents available online including images copied from Google search and contents of presentations of other professors. I don't claim any image or text to be my own. All the credit goes to the original authors.

Instructor Information

Instructor	Shoaib Farooq	E-mail	m.shoaib1050@gmail.com
 GitHub	https://github.com/SHOAIB1050		
 YouTube	https://www.youtube.com/c/pakithub		
 LinkedIn	https://www.linkedin.com/in/shoaib-farooq-b5b190105/		

Let's Start .....

Lecture #8

GOALS

This Lecture Will Cover:

- Cost Function
- Gradient Descent
- Generalization
  - Overfitting vs Underfitting
  - Features Scaling
  - Train, Test and Evaluate model

COST FUNCTION

- The cost function helps find optimal model parameters
  - Best fit line for the data points.
- Searching for these parameters is a minimization problem
  - Model with minimum error between the predicted value and the actual value.
- One such cost function is:
  - Mean Squared Error(MSE):

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

- $\hat{y}_i$: is predicted label
- y_i: Original label

SURROGATE LOSS FUNCTIONS

0/1 loss: $l(y, y') = 1[yy' \leq 0]$

Hinge: $l(y, y') = \max(0, 1 - yy')$

Exponential: $l(y, y') = \exp(-yy')$

Squared loss: $l(y, y') = (y - y')^2$

GRADIENT DESCENT

- Gradient descent is an optimization algorithm
- It helps for searching for the optimal model parameters
- Update parameters according to the gradient values.
- A gradient measures how much the output of a function changes if you change the parameter values.

GRADIENT DESCENT

- Initialize **w** (e.g., randomly)
- Update the values of ***w*** based on the gradient:

$$w_i = w_i - \lambda \frac{\partial J}{\partial w_i}$$

- *Where λ is learning rate*
- *To find w_0 take derivate of the function with respect to it:*

$$w_0 = w_0 - \lambda \frac{\partial J}{\partial w_0}$$

GRADIENT DESCENT

- To find w_1 take derivate of the function with respect to it:

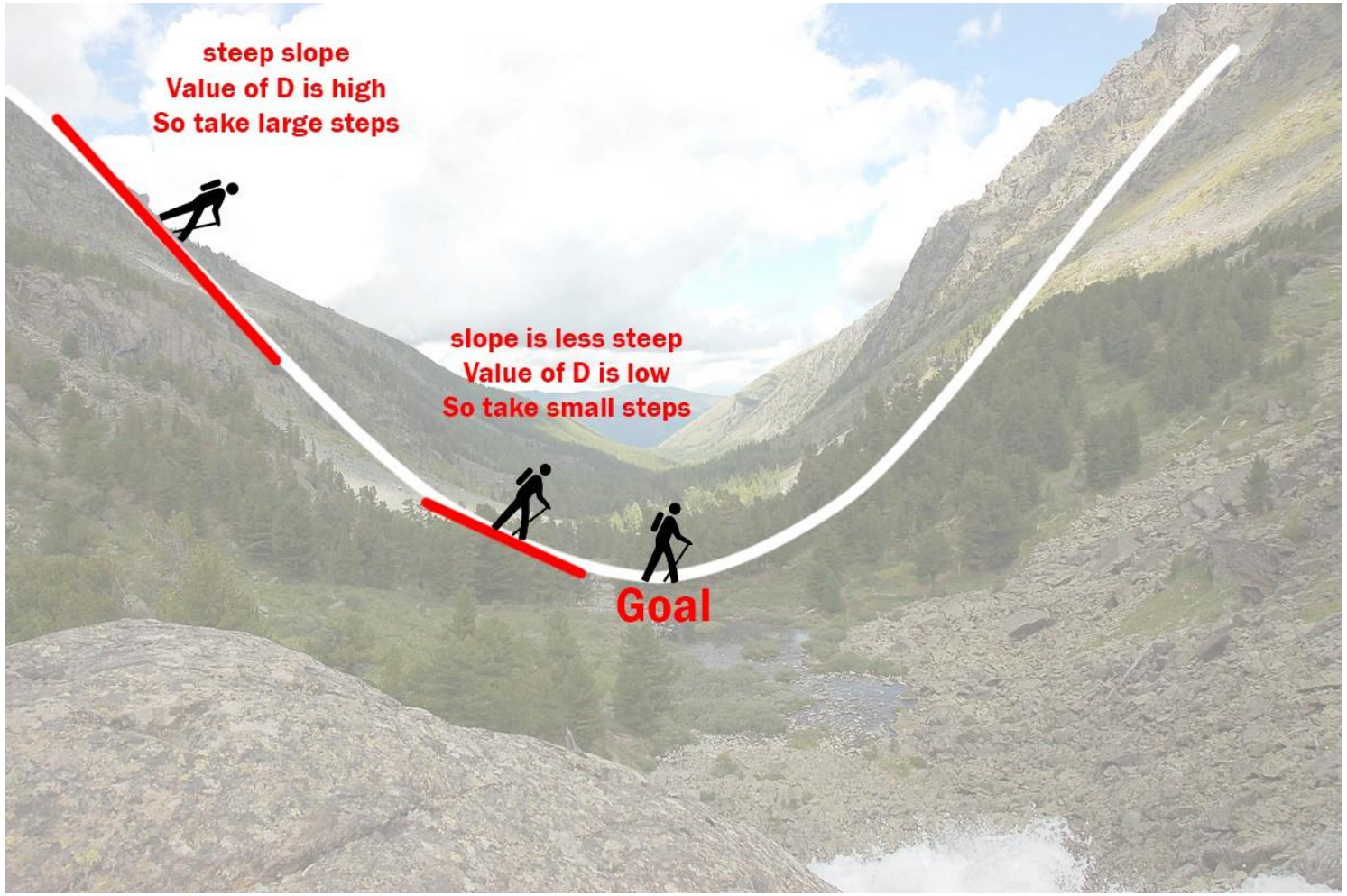
$$w_1 = w_1 - \lambda \frac{\partial J}{\partial w_1}$$

- After solving for the two parameters we get:

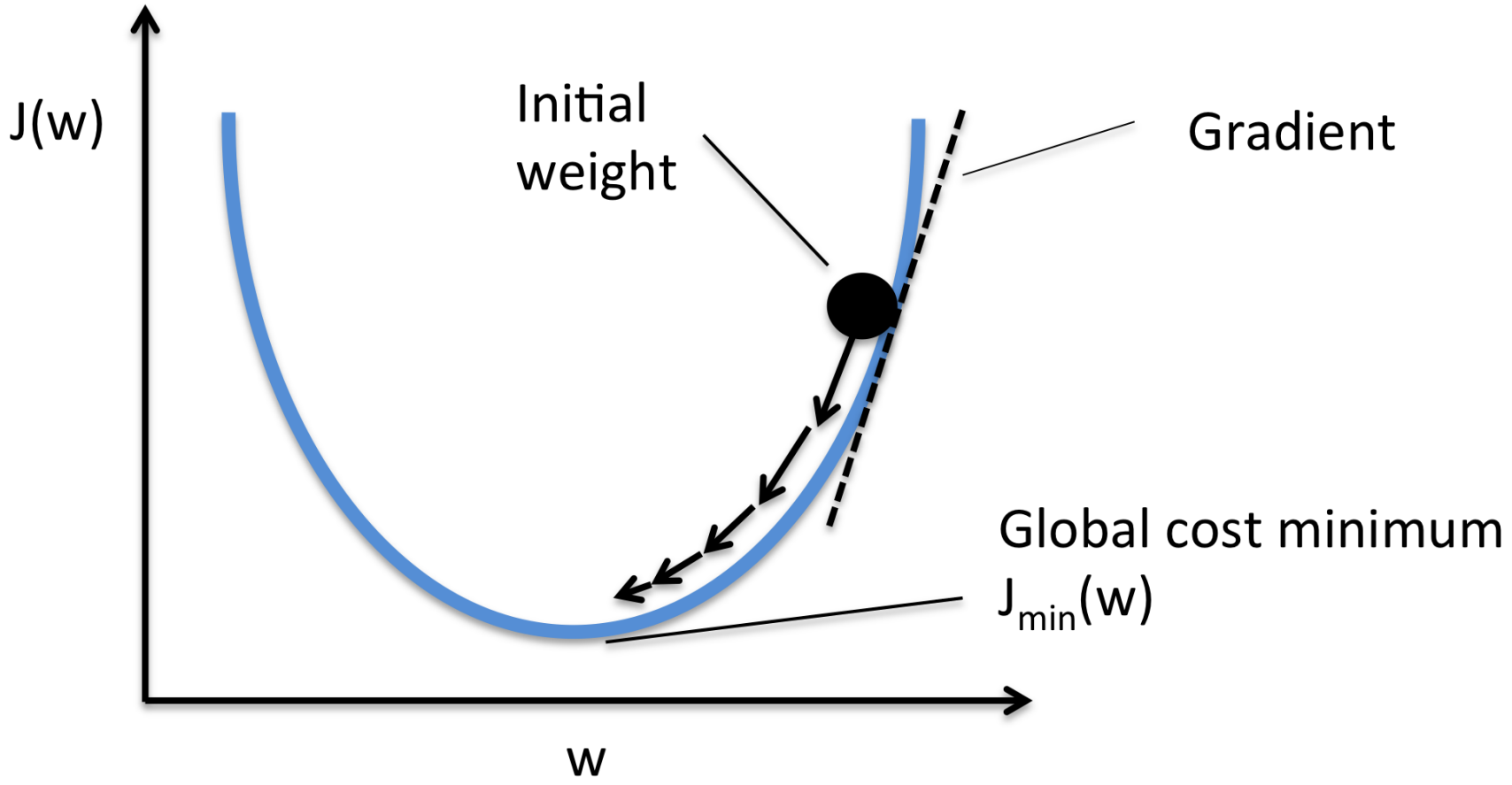
$$\frac{\partial J}{\partial w_1} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})x_i$$

$$\frac{\partial J}{\partial w_0} = -\frac{2}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})$$

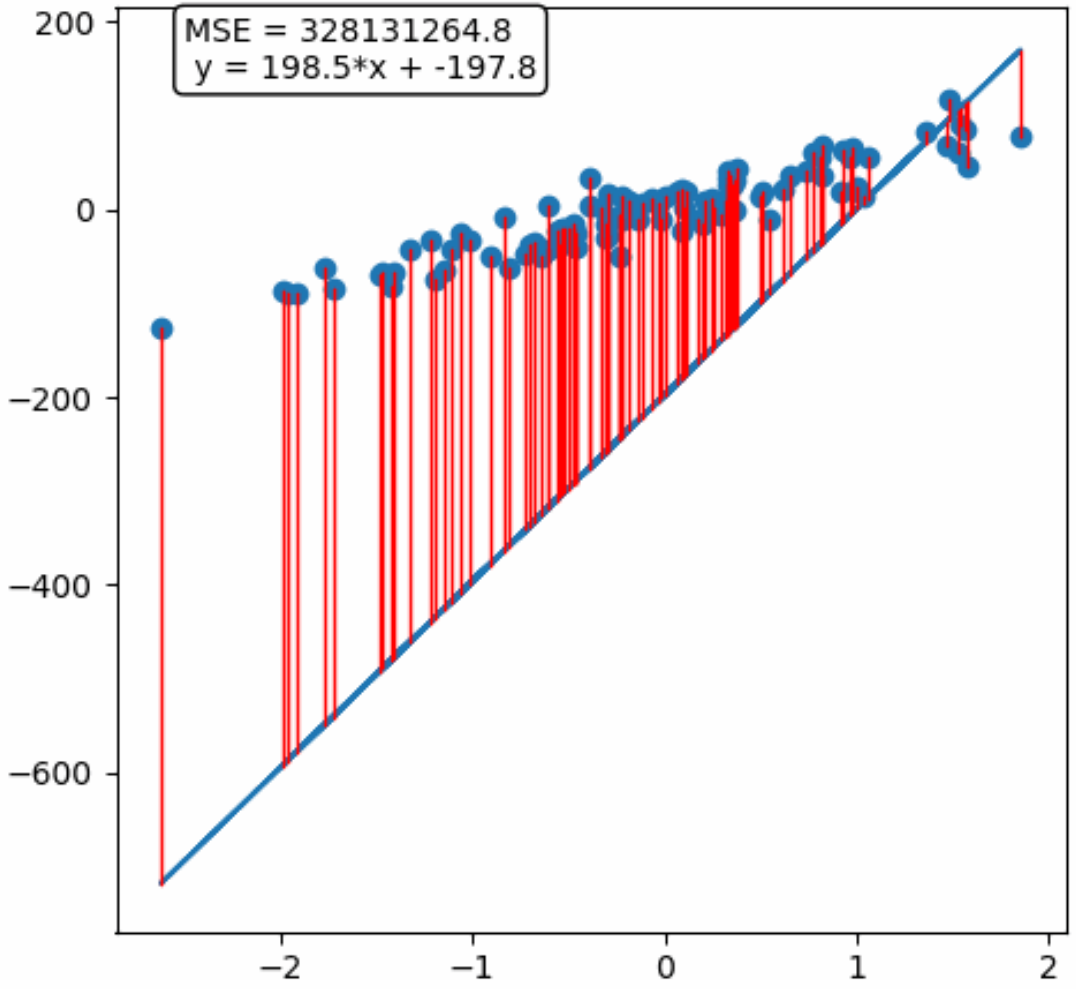
GRADIENT DESCENT

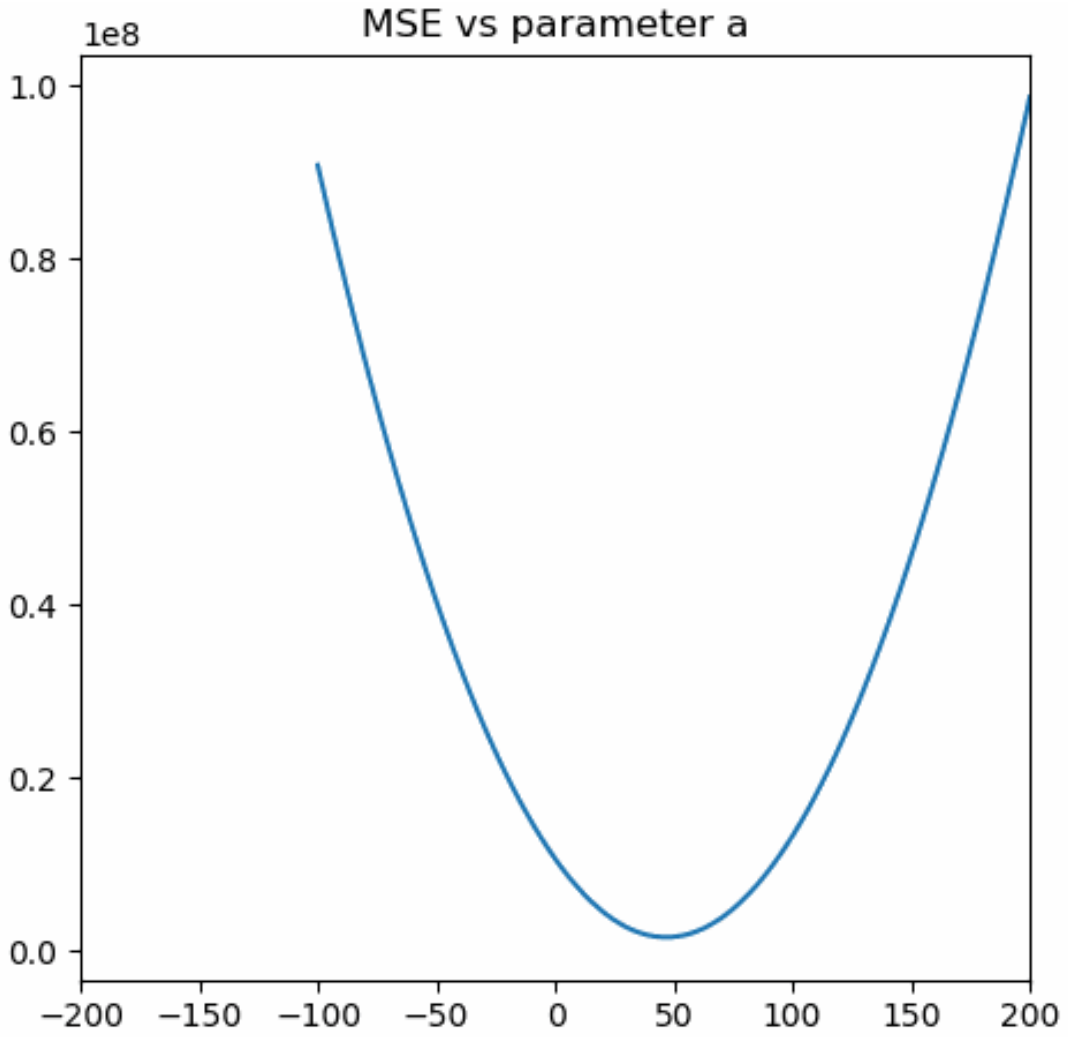


GRADIENT DESCENT



GRADIENT DESCENT





OPTIMIZATION: GRADIENT DESCENT

- To find w_1 take derivate of the function with respect to it:

$$w_1 = w_1 - \lambda \frac{\partial J}{\partial w_1}$$

$$w_n = w_n - \lambda \frac{\partial J}{\partial w_n}$$

OPTIMIZATION: GRADIENT DESCENT

- **Variants of Gradient Descent** are available for optimization in ML
  - Batch Gradient Descent
  - Stochastic Gradient Descent(SGD)
  - Mini-batch Gradient Descent

OPTIMIZATION: GRADIENT DESCENT

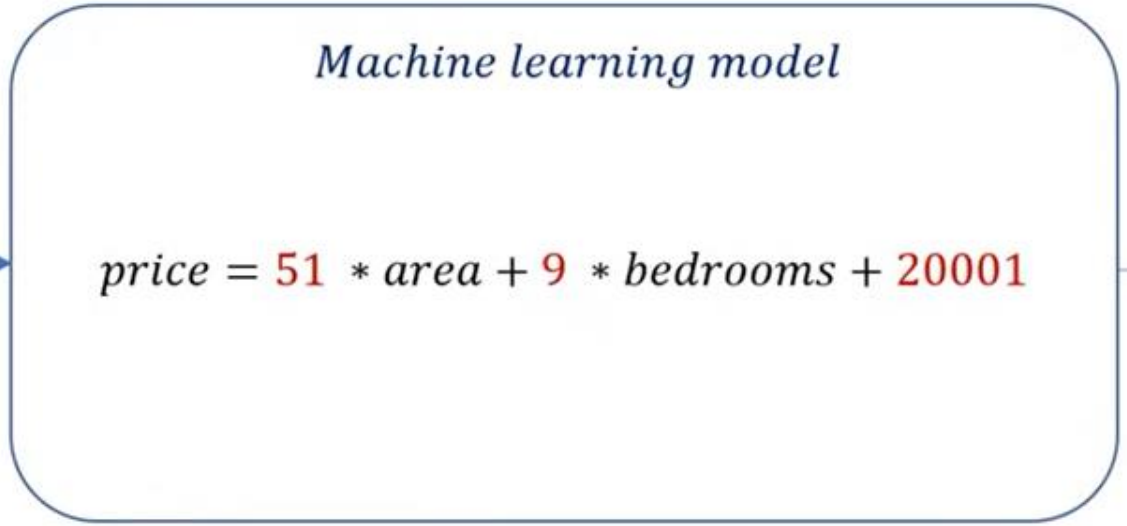
- **Batch Gradient Descent**
  - Updates the weight vector over the full training data
  - It is very slow on very large training data.

$$w_1 = w_1 - \lambda \frac{\partial J}{\partial w_1}$$

$$w_1 = w_1 - \lambda \frac{2}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i) x_i$$

Batch Gradient Descent computes the gradient of the cost function using the entire training dataset for each iteration. This approach ensures that the computed gradient is precise, but it can be computationally expensive when dealing with very large datasets.

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000

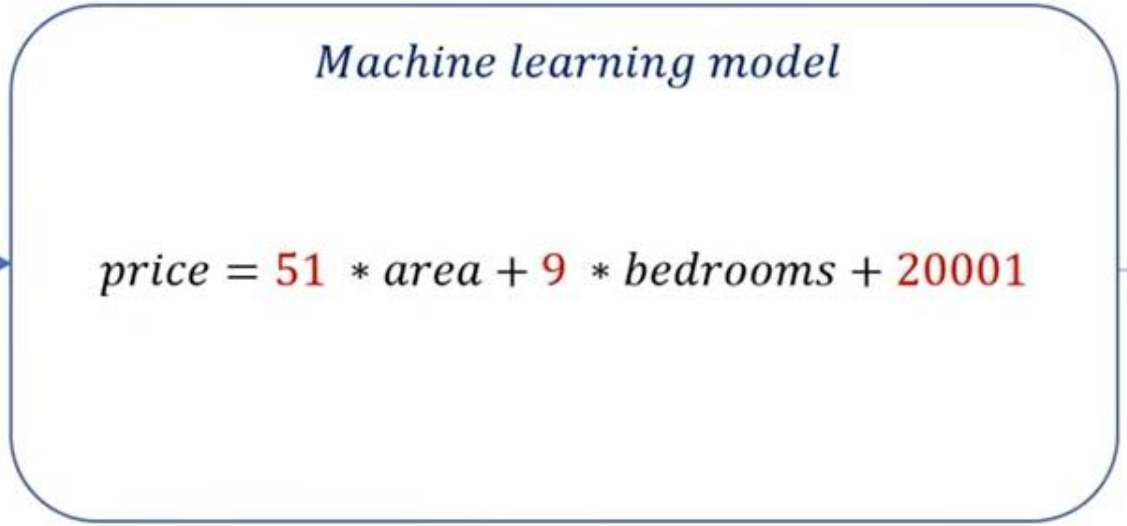


$\widehat{price} = 152610$

$price = 550000$

$error1 = (price - \widehat{price})^2$

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
4000	5	760000
4100	6	810000



$\widehat{price} = 229155$

$price = 550000$

$error6 = (price - \widehat{price})^2$

End of second epoch

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 million samples

1) To find cumulative error for first round (epoch) now we need to do a forward pass for 10 million samples
2) We have 2 features (area and bedroom). This requires finding 20 million derivatives

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

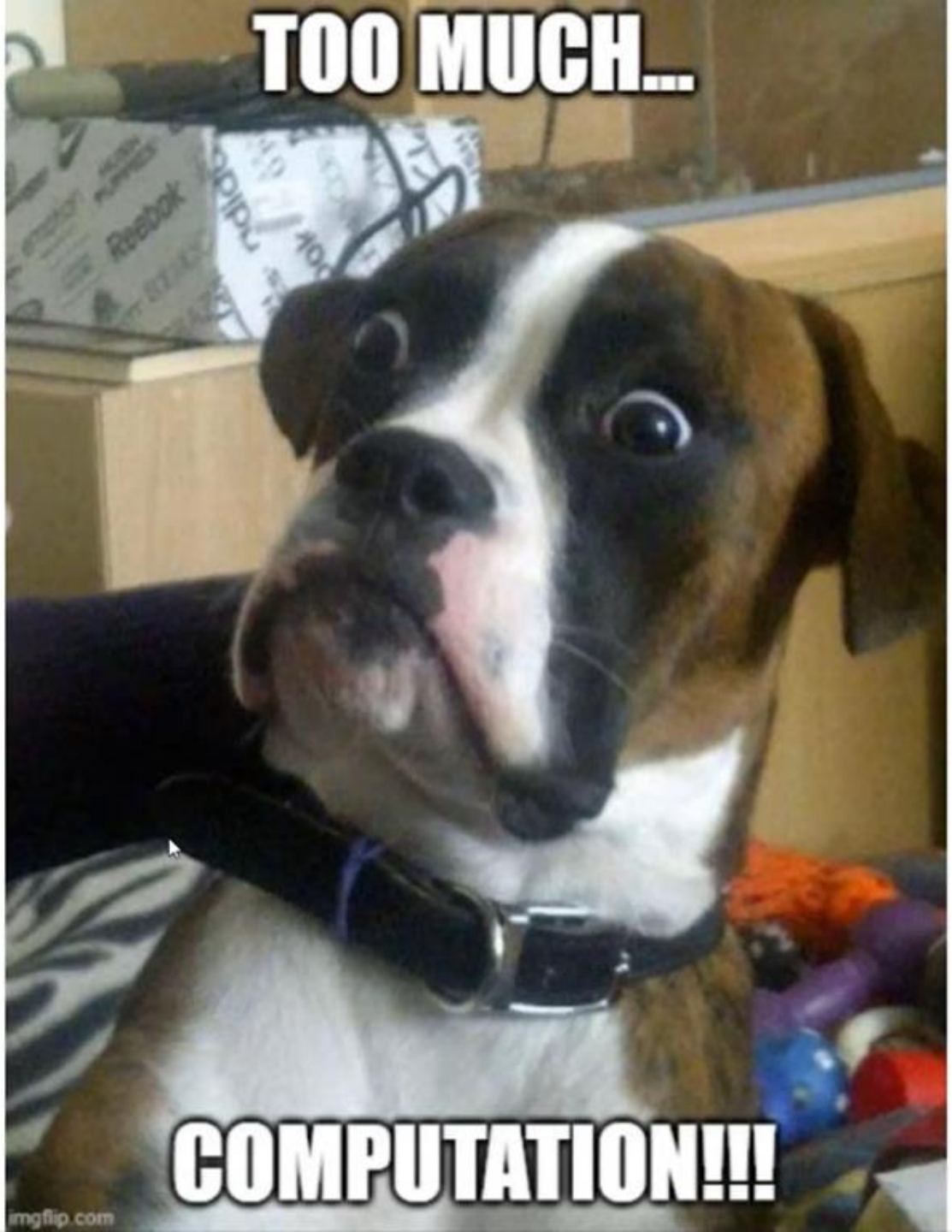
10 million samples

1) To find cumulative error for first round (epoch) now we need to do a forward pass for 10 million samples
2) We have 2 features (area and bedroom). This requires finding 20 million derivatives

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 million samples

1) To find cumulative error for first round (epoch) now we need to do a forward pass for 10 million samples
2) We have 2 features (area and bedroom). This requires finding 20 million derivatives


TOO MUCH...
COMPUTATION!!!
imgflip.com

OPTIMIZATION: GRADIENT DESCENT

Batch Gradient Descent

Advantages

- **Accurate Gradient Estimates**: Since it uses the entire dataset, the gradient estimate is precise.
- **Good for Smooth Error Surfaces**: It works well for convex or relatively smooth error manifolds.

Disadvantages

- **Slow Convergence**: Because the gradient is computed over the entire dataset, it can take a long time to converge, especially with large datasets.
- **High Memory Usage**: Requires significant memory to process the whole dataset in each iteration, making it computationally intensive.
- **Inefficient for Large Datasets**: With large-scale datasets, Batch Gradient Descent becomes impractical due to its high computation and memory requirements.

OPTIMIZATION: GRADIENT DESCENT

- **Stochastic Gradient Descent(SGD)**
  - It updates the parameters for each training data, according to its own gradients:

$$w_1 = w_1 - \lambda \frac{\partial J}{\partial w_1}$$

$$w_1 = w_1 - \lambda (y_i - \hat{y}_i) x_i$$

Stochastic Gradient Descent (SGD) addresses the inefficiencies of Batch Gradient Descent by computing the gradient using only a single training example (or a small subset) in each iteration. This makes the algorithm much faster since only a small fraction of the data is processed at each step.

1. Randomly pick single data training sample

area	bedrooms	price
2600	3	550000
3000	4	565000
3200	3	610000
3600	3	595000
...	...	...
4100	6	810000

10 million samples

Machine learning model

$$price = 1 * area + 1 * bedrooms + 1$$

$\widehat{price} = 3204$

$price = 610000$

$$error = (price - \widehat{price})^2$$

4. Again adjust weights

$$w1 = w1 - learning\ rate * {\partial(\text{error})}/{\partial w1} \qquad w1 = 14 - (-100) = 114$$

$$w2 = w2 - learning\ rate * {\partial(\text{error})}/{\partial w2} \qquad w2 = 4 - (-12) = 16$$

$$b = b - learning\ rate * {\partial(\text{error})}/{\partial b} \qquad \text{bias} = 1501 - (-2001) = 3502$$

4. Again adjust weights

$$w1 = w1 - learning\ rate * {\partial(\text{error})}/{\partial w1} \qquad w1 = 14 - (-100) = 114$$

$$w2 = w2 - learning\ rate * {\partial(\text{error})}/{\partial w2} \qquad w2 = 4 - (-12) = 16$$

$$b = b - learning\ rate * {\partial(\text{error})}/{\partial b} \qquad \text{bias} = 1501 - (-2001) = 3502$$

Stochastic Gradient Descent(SGD)

OPTIMIZATION: GRADIENT DESCENT

Stochastic Gradient Descent(SGD)

Advantages

- **Faster Convergence:** Since the gradient is updated after each individual data point, the algorithm converges much faster than Batch Gradient Descent.
- **Lower Memory Requirements:** As it processes only one data point at a time, it requires significantly less memory, making it suitable for large datasets.
- **Escape Local Minima:** Due to its stochastic nature, SGD can escape local minima and find the global minimum, especially for non-convex functions.

OPTIMIZATION: GRADIENT DESCENT

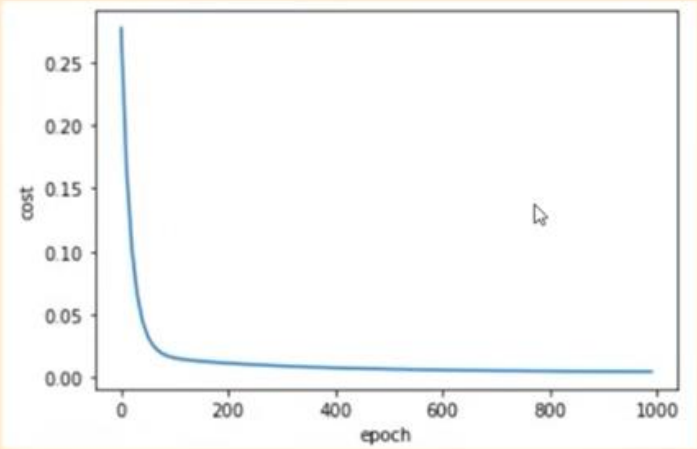
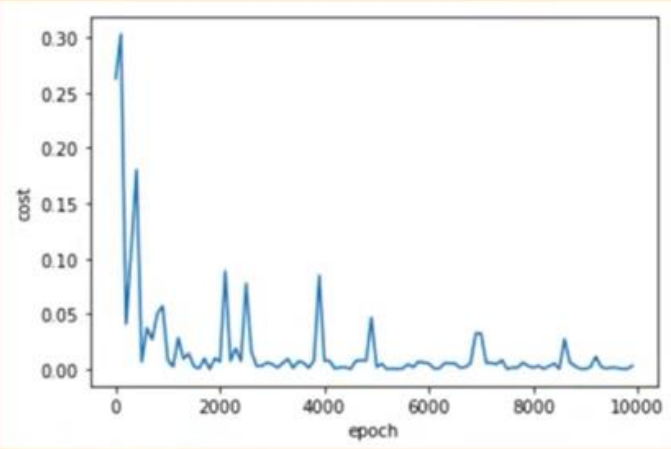
Stochastic Gradient Descent(SGD)

Disadvantages

- **Noisy Gradient Estimates:** Since the gradient is based on a single data point, the estimates can be noisy, leading to less accurate results.

- **Requires Shuffling:** To ensure randomness, the dataset should be shuffled before each epoch.

OPTIMIZATION: GRADIENT DESCENT

Batch Gradient Descent	Stochastic Gradient Descent (SGD)
Use **all** training samples for one forward pass and then adjust weights	Use **one** (randomly picked) sample for a forward pass and then adjust weights
Good for small training set	Good when training set is very big and we don't want too much computation
	

Mini batch is like SGD. Instead of choosing one randomly picked training sample, you will use a batch of randomly picked training samples.

1. For example I have 20 training samples total.
2. Let's say I use 5 random samples for one forward pass to calculate cumulative error
3. After that I adjust weights

OPTIMIZATION: GRADIENT DESCENT

- **Mini-batch Gradient Descent**
  - It computes the gradients on small random sets of instances called mini-batches.
  - It has shown better performance than SGD
  - More robust stable than SGD

OPTIMIZATION: GRADIENT DESCENT

Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini batch gradient descent
Use **all** training samples for one forward pass and then adjust weights	Use **one** (randomly picked) sample for a forward pass and then adjust weights	Use **a batch of** (randomly picked) samples for a forward pass and then adjust weights

OPTIMIZATION: GRADIENT DESCENT

Batch Gradient Descent

- Entire dataset for updation
- Cost function reduces smoothly
- Computation cost is very high

Stochastic Gradient Descent (SGD)

- Single observation for updation
- Lot of variations in cost function
- Computation time is more

Mini-Batch Gradient Descent

- Subset of data for updation
- Smoother cost function as compared to SGD
- Computation time is lesser than SGD
- Computation cost is lesser than Batch Gradient Descent

Thank You ☺