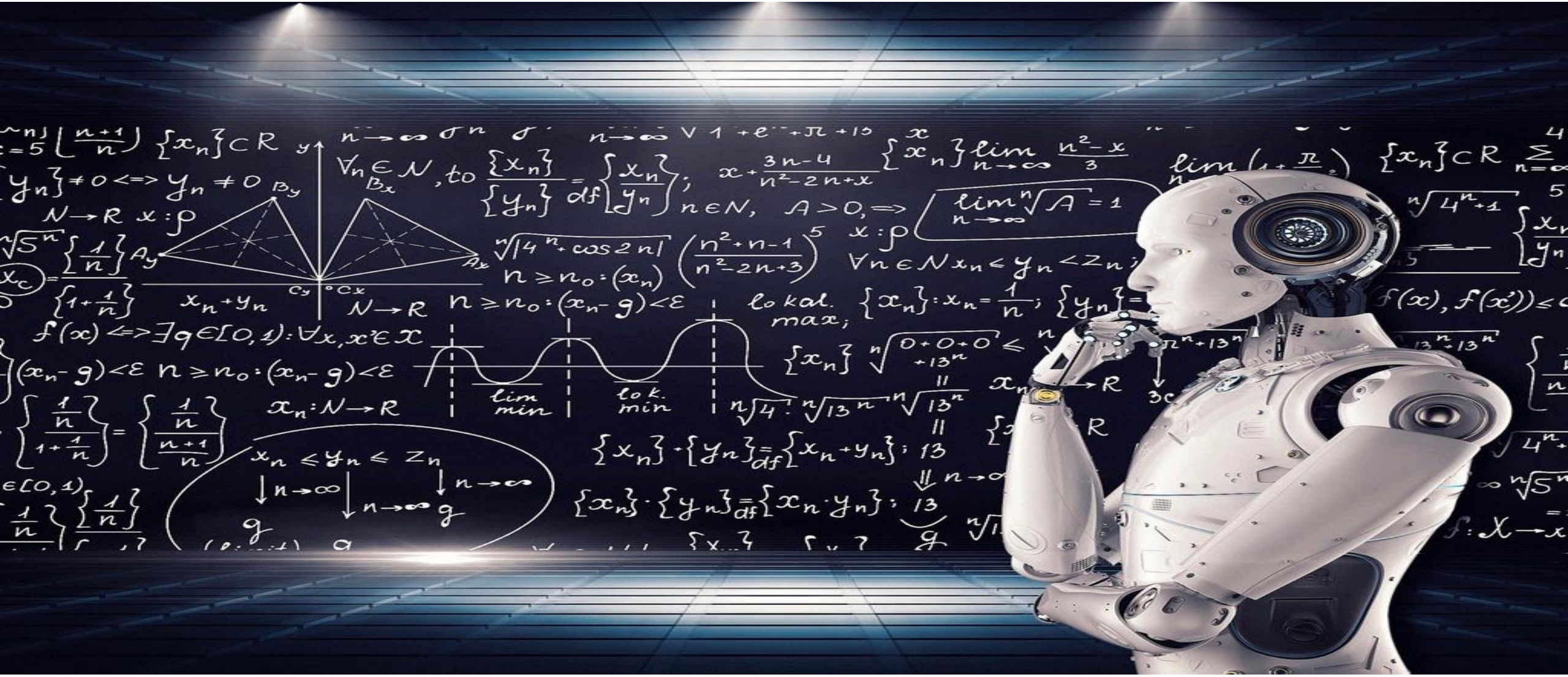
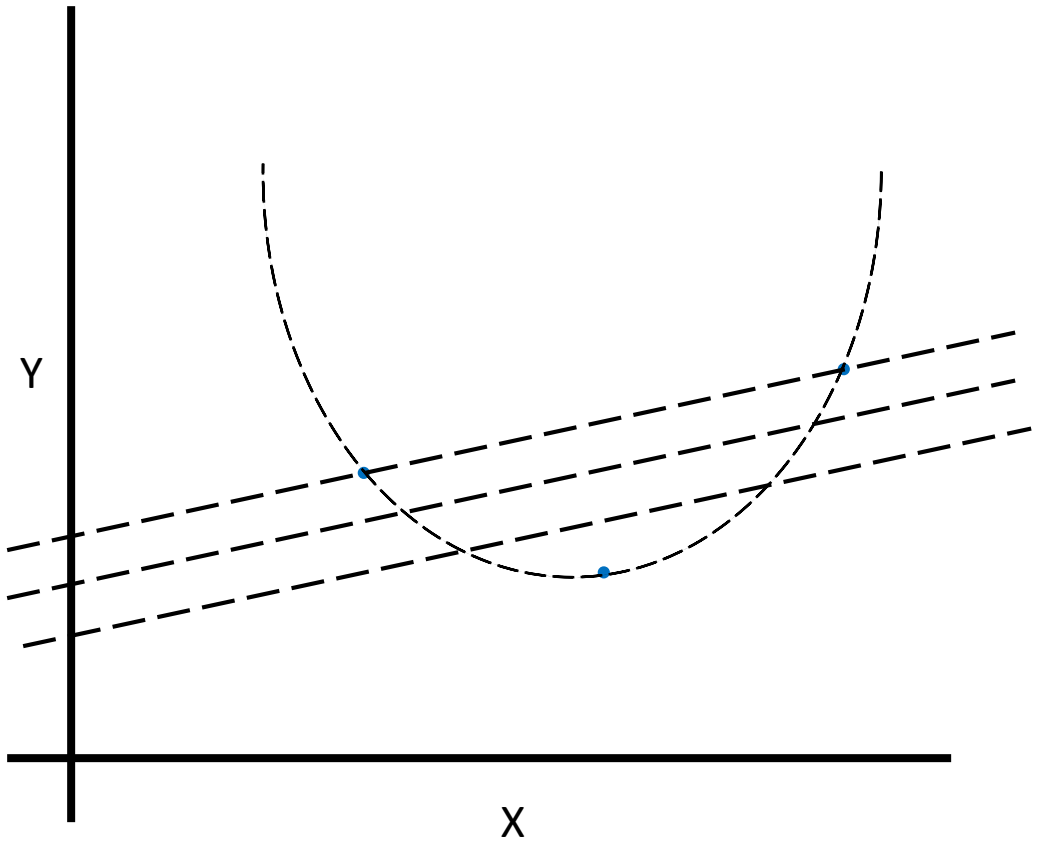
Linear Regression

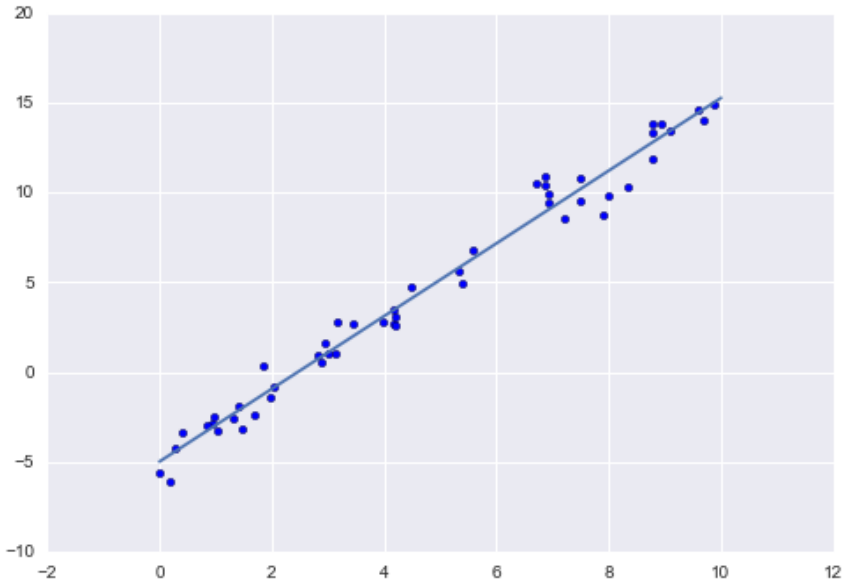


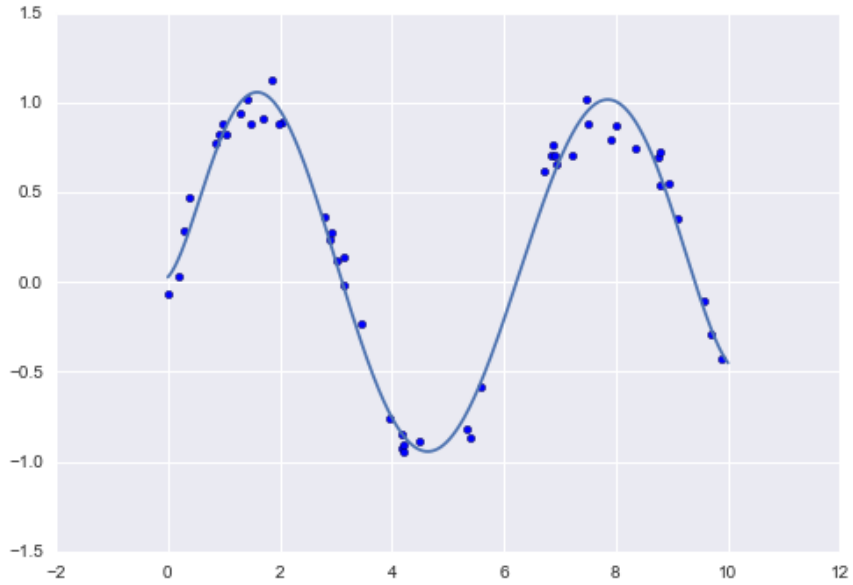
Example



Linear Regression

- We are searching for the function that best describes the data
- What we need in order to find it?
  - Decide what kind of function we are looking for (linear, polynomial, etc.)
  - Quality measure that will tell us if our function is good or not
  - Procedure that will improve our quality measure





IDC HERZLIYA

Linear Regression

- Definitions:
  - Hypothesis function $h_\theta(x) = \theta_0 + \theta_1 x_1$
  - $y^{(i)}$is the true output of the target function on input $x^{(i)}$
    - i.e. $f\left(x^{(i)}\right) = \theta_0 + \theta_1 x^{(i)} = y^{(i)}$
  - Cost function: usually use squared loss which is

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

  - This function called MSE – Mean Square Error

Linear Regression

- We want to find θ_0 and θ_1 that minimize the cost function, i.e.
- $$\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$$
- Algorithm:
  - Guess some value for θ_0 and θ_1
  - Repeat until error is small enough:
    - Update θ_0, θ_1 so that we are closer to the min

Linear Regression

- If we have more than 1 feature?
  - x, θ are vectors now:

$$h_\theta(\bar{x}) = \theta_0 + \bar{\theta} \cdot \bar{x} = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

- So, what is the output of the algorithm?

Gradient Descent

- Gradient definition?
  - The gradient is a multi-variable generalization of the derivative
  - While a derivative can be defined on functions of a single variable, for functions of several variables, the gradient takes its place
  - The gradient is a vector-valued function, as opposed to a derivative, which is scalar-valued
  - Like the derivative, the gradient represents the slope of the tangent of the graph of the function
  - More precisely, the gradient points in the direction of the greatest rate of increase of the function, and its magnitude is the slope of the graph in that direction
  - For an n-dimension function $y = f(x_1, x_2, \dots x_n)$ the gradient is defined as:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right)$$

- Question:

  How can we use the gradient to minimize a function?

Gradient Descent

- Algorithm:
  - Guess some value for θ_0 and θ_1
  - Repeat until error is small enough
    - Update $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ for $j = 0,1$

    // α is a hyperparameter of the algorithm called the learning rate

Gradient Descent Details

- Apply Gradient descent to minimize:

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

Calculating the gradients

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

- Partial derivatives with respect to θ_0:

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

$$= \frac{\partial}{\partial\theta_0}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 {x_1}^i - y^i)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 {x_1}^i - y^i)$$

Calculating the gradients

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

- Partial derivatives with respect to θ_1:

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

$$= \frac{\partial}{\partial\theta_1}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 {x_1}^i - y^i)^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 {x_1}^i - y^i)\cdot {x_1}^i$$

Plug gradients into algorithm

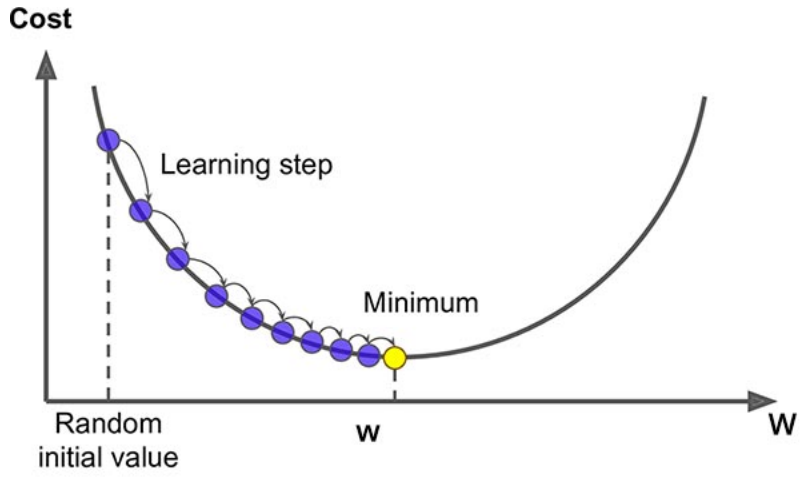
- Algorithm:
  - Guess some value for θ_0 and θ_1
  - Repeat until error is small enough
    - Update $\theta_0 \coloneqq \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) =$
      $$\theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 {x_1}^i - y^i)$$
    - Update $\theta_1 \coloneqq \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\theta_0 + \theta_1 {x_1}^i - y^i\right) \cdot {x_1}^i$

Simultaneous updates

- Correct:
  - $temp0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
  - $temp1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
  - $\theta_0 = temp0;\ \theta_1 = temp1;$
- Incorrect:
  - $\theta_0 = \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
  - $\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
- The cost function will have been updated

Gradient Descent

- Gradient descent is an iterative optimization algorithm for finding the minimum of a function
- You start by initiating the parameters vector θ with random values
- Then, at each step, GD measures the local gradient of the cost function (e.g., MSE) with regard to the vector θ, and it goes in the direction of the greatest descent
- The algorithm continues until it converges to a minimum (where the gradient is zero)

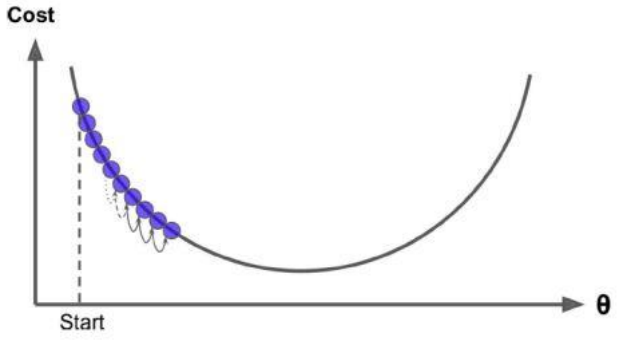


Questions

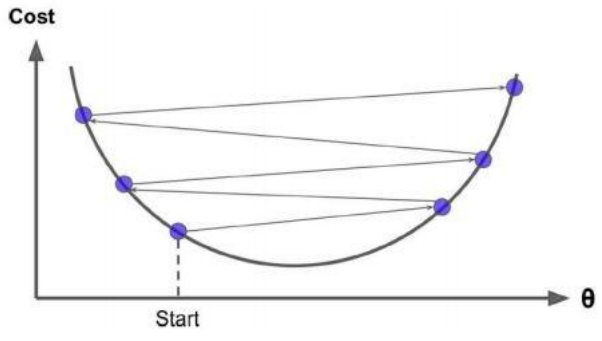
- What if current guess is a minimum?
- What is α ?
- What if α is too small? Too big?
- Do we need to decrease α with time?

Intuition

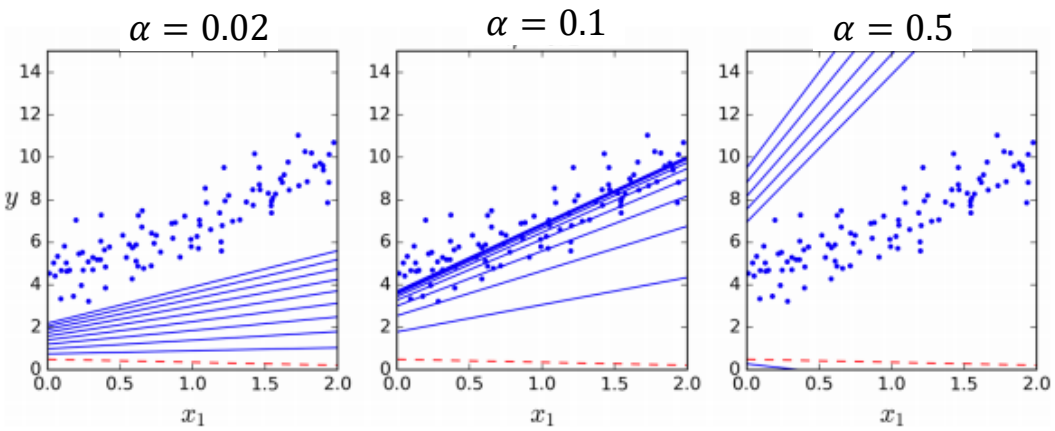
- When we are to the right of the minimum, gradient is positive, so we move left
- When we are to the left of the minimum, gradient is negative, so we move right
- When α is too small, what happens?
- When α is too big, what happens? (possible to diverge)



Slow learning rate: Converges to minimum but very slowly

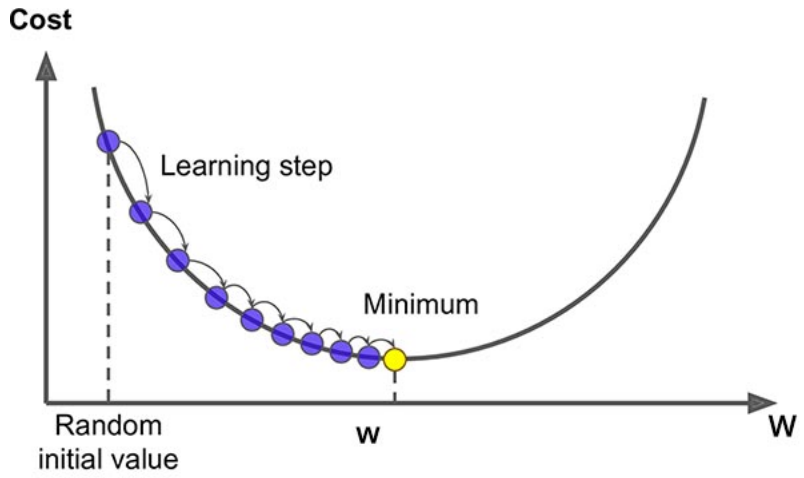


Fast learning rate: May not converge to minimum and error might keep increasing with further epochs



Intuition

- Notice how the steps decrease in size as we get closer to the minimum, why?

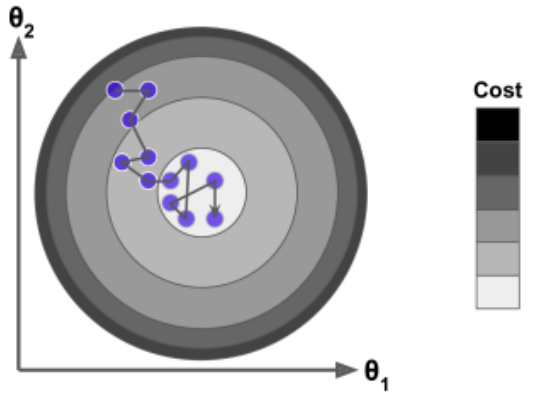


Stochastic \ Batch Gradient Descent

- Till now we considered all the data when we calculated the gradient
- Instead, we can use some of the data – advantages? disadvantages?
- Batch means we are using the whole data (sometimes people say ‘Batch’ but they mean to some of the training examples)
- We can also do stochastic – one example

Stochastic Gradient Descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradient at every step, which makes it very slow when the training set is large
- At the opposite extreme, **Stochastic Gradient Descent** picks a random instance in the training set at every step and computes the gradients based only on that single instance
  - This makes the algorithm much faster and also makes it possible to train on huge training sets, since only one instance needs to be in memory at each iteration
- On the other hand, this algorithm is much less regular than Batch GD
  - Instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average
  - So once the algorithm stops, the final parameter values are good, but not optimal



Stochastic Gradient Descent

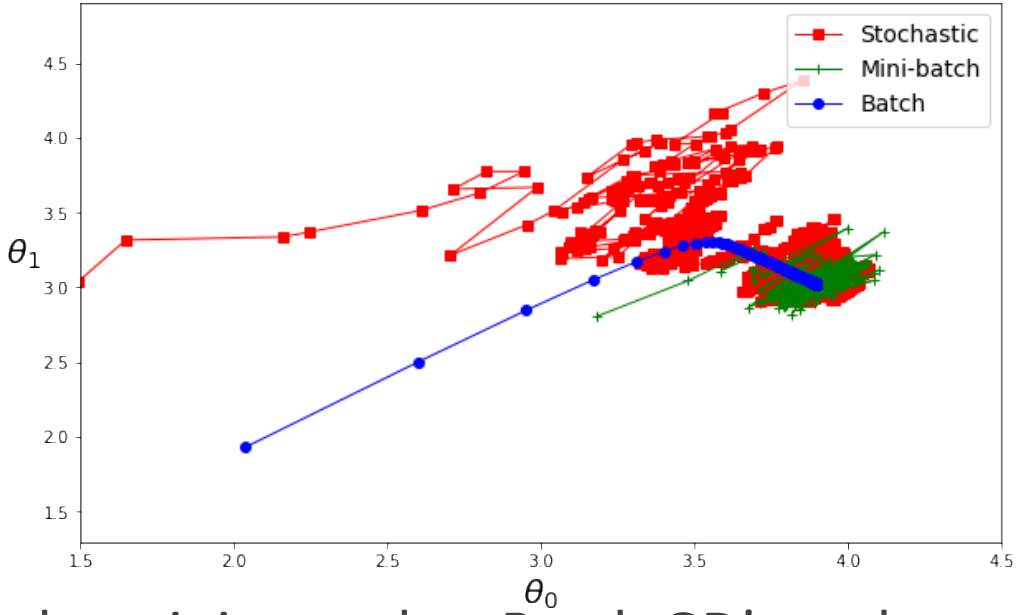
- When the cost function is very irregular, this can actually help the algorithm jump out of local minima, so Stochastic GD has a better chance of finding the global minimum than Batch GD does
- Therefore randomness is good to escape from local optima, but bad because it means that the algorithm can never settle at the minimum

Mini-Batch Gradient Descent

- At each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch GD computes the gradients on small random sets of instances called minibatches
- The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs
- The algorithm's progress in parameter space is less erratic than with SGD, especially with fairly large mini-batches
  - As a result, Mini-batch GD will end up walking around a bit closer to the minimum than SGD
  - But, on the other hand, it may be harder for it to escape from local minima

Mini-Batch Gradient Descent

- The following figure shows the paths taken by the three GD algorithms in parameter space during training



- They all end up near the minimum, but Batch GD's path actually stops at the minimum, while both Stochastic GD and Mini-batch GD continue to walk around
- However, Batch GD takes a lot of time to take each step

Choosing α

- Search method:
  - Try many different values of α and use a ‘held out’ set for testing and measure the error and choose α which had the minimum
  - If cost function is not decreasing or staying the same every round, then gradient descent is not working

Choosing α

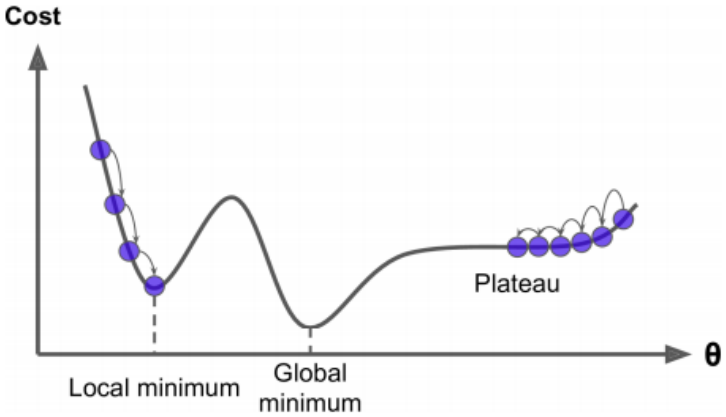
- If cost function is increasing every round, probably α is too big
- Try values $(3)^i$ for $\mathrm{i} = -17, -16, \ldots, 0$ and check training error for each
- Check the error every 100 iteration and stop when the error is getting larger
- Select the alpha with the lowest error
- Another method is to plot the training error and look at the slope
- If training error is increasing, then alpha is too large (probably). If training error slope is not steep enough, then alpha is too small

When to stop iterating?

- We said repeat until convergence or error is small enough. How do we know if this happens?
- Test if $J(\theta)$ decreased from the last iteration less than some small value, like 0.003

Cost Function Shape and Normalization

- Not all cost functions look like a nice regular bowls
- There may be holes, ridges, plateaus, and all sorts of irregular terrains, making convergence to the minimum very difficult
- The following figure shows the two main challenges with Gradient Descent:
  - If the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum
  - If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum

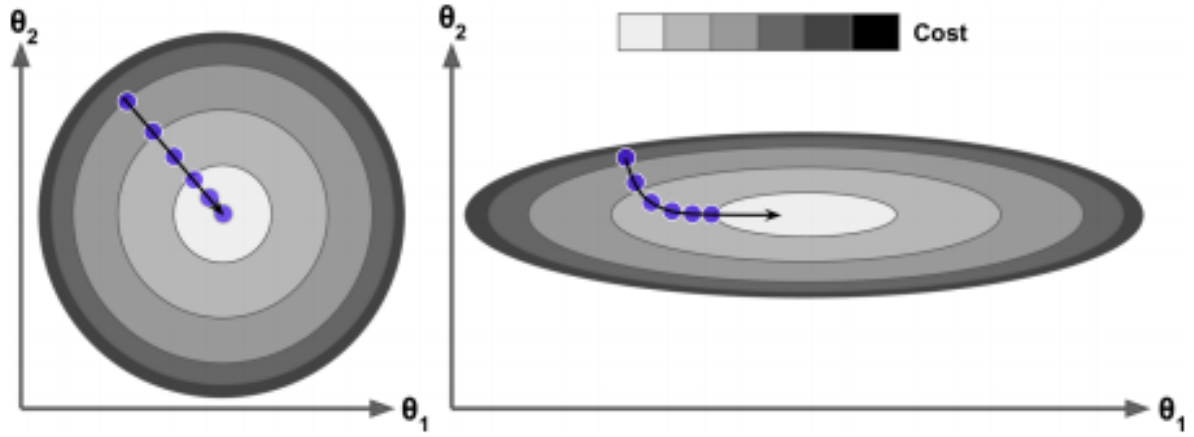


Cost Function Shape and Normalization

- Fortunately, the MSE cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve
- This implies that there are no local minima, just one global minimum
- It is also a continuous function with a slope that never changes abruptly
- These two facts have a great consequence: Gradient Descent is guaranteed to approach arbitrarily close to the global minimum (if you wait long enough and if the learning rate is not too high)

Cost Function Shape and Normalization

- In fact, the cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales
- The following figure shows Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has much smaller values than feature 2 (on the right)



* Since feature 1 is smaller, it takes a larger change in θ_1 to affect the cost function, which is why the bowl is elongated along the θ_1 axis

Feature Scaling

- One of the most important transformations your data needs is *feature scaling*
- With few exceptions, Machine Learning algorithms don't perform well when the input numerical attributes have very different scales
- There are two common ways to get all attributes to have the same scale: *min-max scaling* and *standardization*
- In **min-max scaling** values are shifted and rescaled so that they end up ranging from 0 to 1
  - We do this by subtracting the min value and dividing by the max minus the min
- Scikit-Learn provides a transformer called **MinMaxScaler** for this

Feature Scaling

- Min-max scaling example:

```
from sklearn.preprocessing import MinMaxScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = MinMaxScaler()
scaler.fit_transform(data)
```

```
array([[0.  , 0.  ],
       [0.25, 0.25],
       [0.5 , 0.5 ],
       [1.  , 1.  ]])
```

- The **feature_range** hyperparameter lets you change the range if you don't want 0–1

```
scaler = MinMaxScaler(feature_range=[0, 5])
scaler.fit_transform(data)
```

```
array([[0.  , 0.  ],
       [1.25, 1.25],
       [2.5 , 2.5 ],
       [5.  , 5.  ]])
```

Feature Scaling

- **Standardization** first subtracts the mean value and then it divides by the standard deviation so that the resulting distribution has zero mean and unit variance
- Unlike min-max scaling, standardization does not bound values to a specific range, which may be a problem for some algorithms (e.g., neural networks often expect an input value ranging from 0 to 1)
- However, standardization is much less affected by outliers
- For example, suppose a feature whose normal range is 0-15, has an outlier equal to 100 (by mistake)
- Min-max scaling would then crush all the other values from 0-15 down to 0-0.15, whereas standardization would not be much affected
- Scikit-Learn provides a transformer called **StandardScaler** for standardization

Feature Scaling

- Standardization example:

```
from sklearn.preprocessing import StandardScaler

data = [[-1, 2], [-0.5, 6], [0, 10], [1, 18]]
scaler = StandardScaler()
scaler.fit_transform(data)
```

```
array([[-1.18321596, -1.18321596],
       [-0.50709255, -0.50709255],
       [ 0.16903085,  0.16903085],
       [ 1.52127766,  1.52127766]])
```

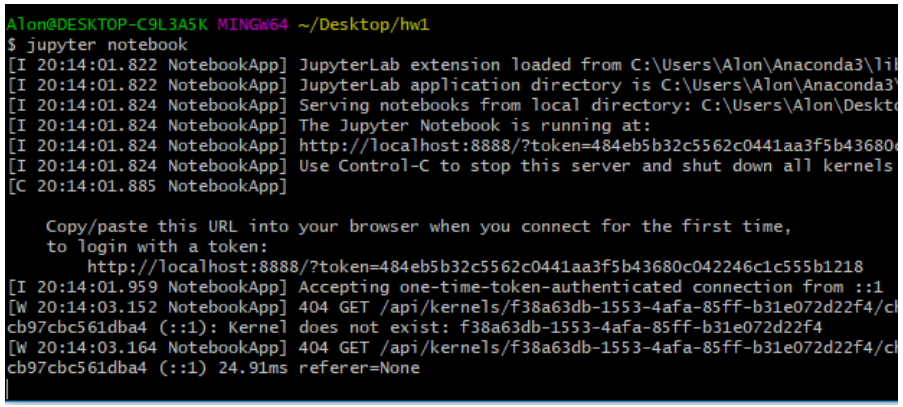
Homework

- This course introduces Jupyter notebook as the standard for homework submission
- Jupyter notebook lets you write and execute Python code in your web browser and make it very easy to tinker with code and execute it in small pieces
- For this reason Jupyter notebooks are widely used in scientific computing
- Every hands-on exercise will include a detailed Jupyter notebook
- You will complete your code in the notebook itself
- Submission will include the notebook

Example – HW1

- The first exercise will include the following
  - Jupyter notebook (hwa1.ipynb)
  - Directory containing the data for the exercise

- The Jupyter notebook includes instructions and some code to get you started. Once you have it installed start it with the command `jupyter notebook`
- Once your notebook server is running, point your web browser at http://localhost:8888 to start using your notebooks



Example – HW1

- Jupyter notebooks are made up of a number of cells. Each cell can contain Python code. You can execute a cell by clicking on it and pressing Shift-Enter. When you do so, the code in the cell will run, and the output of the cell will be displayed beneath the cell

- By convention, Jupyter notebooks are expected to be run from top to bottom. Failing to execute some cells or executing cells out of order can result in errors

In [1]:

```
import numpy as np # used for scientific computing
import pandas as pd # used for data analysis and manipulation
import matplotlib.pyplot as plt # used for visualization and plotting

np.random.seed(42)

# make matplotlib figures appear inline in the notebook
%matplotlib inline
plt.rcParams['figure.figsize'] = (14.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

Part 1: Data Preprocessing (10 Points)

For the following exercise, we will use a dataset containing housing prices in King County, USA. The dataset contains 5,000 observations with 18 features and a single target value - the house price.

First, we will read and explore the data using pandas and the `.read_csv` method. Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

In [2]:

```
# Read comma separated data
df = pd.read_csv('data/train.csv') # Make sure this cell runs regardless of your absolute path.
# df stands for dataframe, which is the default format for datasets in pandas
```

IDC HERZLIYA

Example – HW1

- **Everything** should be written inside the notebook itself

- Do not use additional imports

- Make sure you test your code thoroughly – we provided some basic sanity checks and visualization to make sure you are on the write track, but you should include as many tests as you see fit