CS 146 Discussion Week 1

Linear Algebra

Outline

- Linear regression
- Convex functions

Linear Regression

- Input: $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ input features/attributes for the *i*-th example
- Output: $y^{(i)} \in \mathbb{R}$ corresponding label for the *i*-th example
- Goal: find parameters $\boldsymbol{\theta} \in \mathbb{R}^{d+1}$

  such that least squares loss function is minimized.

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} (h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$$

where $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x = \boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}$

- How do we minimize it? Gradient descent.

Derivation

- Derivation of $J(\boldsymbol{\theta})$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 \\
&= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}
\end{aligned}$$

- Vectorization

$$\tfrac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = \tfrac{1}{n} \textstyle\sum_{i=1}^{n} (\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}^{(i)} - y^{(i)}) \boldsymbol{x}^{(i)}$$

Comparing analytical and numerical solutions

```
x = np.array([[1, 2]]).transpose()
y = np.array([[3]])
def J(theta):
  return (theta.transpose() @ x - y ) ** 2
```

```
theta = np.array([[3, 4]]).transpose()
print(theta)
```

```
[[3]
 [4]]
```

```
print(J(theta))
```

```
[[64]]
```

```
delta = np.array([[0.000001, 0 ]]).transpose()
print(((J(theta + delta) - J(theta)) / delta[0]))
delta = np.array([[0, 0.000001 ]]).transpose()
print(((J(theta + delta) - J(theta)) / delta[1]))
```

```
[[16.00000101]]
[[32.000004]]
```

```
def nabla_J(theta):
  return (theta.transpose() @ x - y ) * x
```

```
print(nabla_J(theta))
```

```
[[ 8]
 [16]]
```

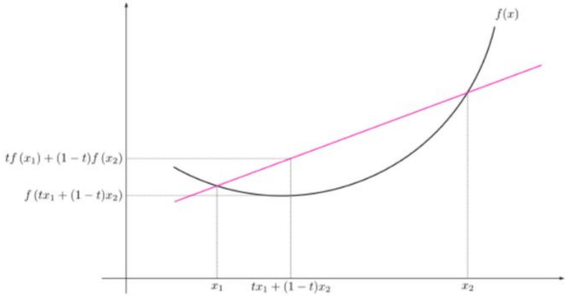
Notebook: https://colab.research.google.com/drive/1LjtgfnjJOYXDStj94HezebKAxTqOIGo4?usp=sharing

Exercise

- For function $J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})^4$, compute first-order derivative for it and rewrite the results in vectorization formulation.

Convex Functions

- **Definition:** A real-valued function f is **convex** if the line segment joining any two points lies above the function



- A function f is convex iff for all $0 \le t \le 1$ and all $x_1, x_2 \in X$:

$$f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2)$$

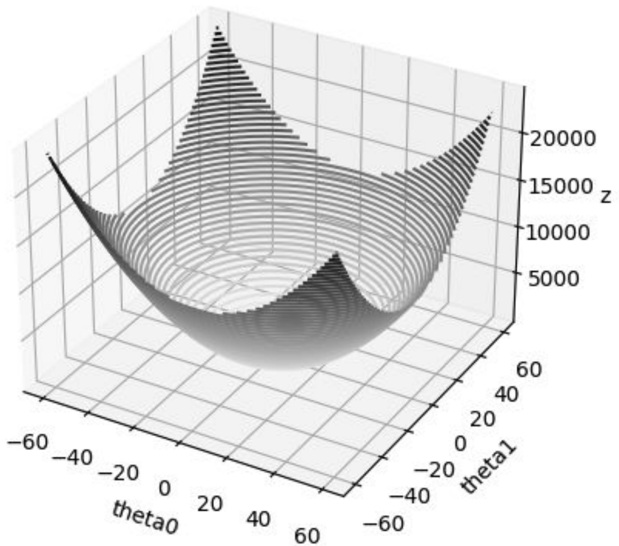
Visualization

```
# suppose we have 3 data points
# x^{(1)} = [1,  1]^T        y^{(1)} = 1
# x^{(2)} = [1,  -1.5]^T     y^{(2)} = 2
# x^{(3)} = [1,  0.5]^T      y^{(3)} = 4
def J(theta0, theta1):
    return ( theta0  + theta1  - 1) ** 2 + \
    ( theta0  - 1.5 * theta1  - 2) ** 2 + \
    ( theta0  + 0.5 * theta1  - 4) ** 2

theta0 = np.linspace(-60, 60, 30)
theta1 = np.linspace(-60, 60, 30)

theta0, theta1 = np.meshgrid(theta0, theta1)
Js = J(theta0, theta1)
```

```
fig = plt.figure()
ax = plt.axes(projection='3d')
ax.contour3D(theta0, theta1, Z, 50, cmap='binary')
ax.set_xlabel('theta0')
ax.set_ylabel('theta1')
ax.set_zlabel('z');
```



Convex Functions

- Prove that $J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) - y^{(i)})^2$ is convex.
  - By showing that the second order derivative is positive semi-definite.