FVE3011

Linear regression / cost function Pytorch Practice

담당교수: 최 학남 xncui@inha.ac.kr

인하대학교

Contents

- Pytorch basic
- Linear regression
- Cost function

Hypothesis and cost function

$$H(x) = Wx + b$$

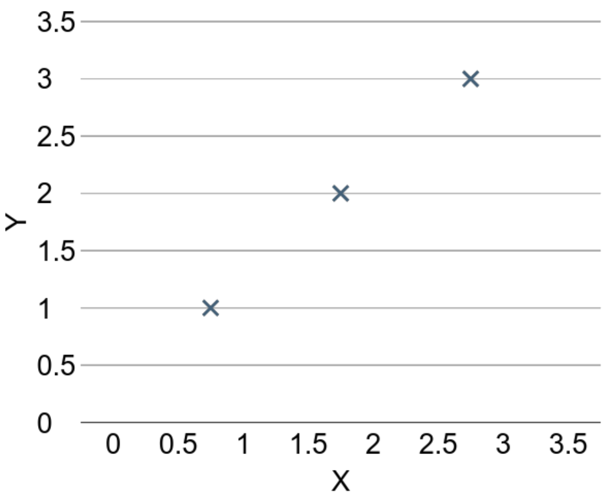
$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} (H(x^{(i)}) - y^{(i)})^2$$

Predicting exam score: regression

x (hours)	y (score)
10	90
9	80
3	50
2	30

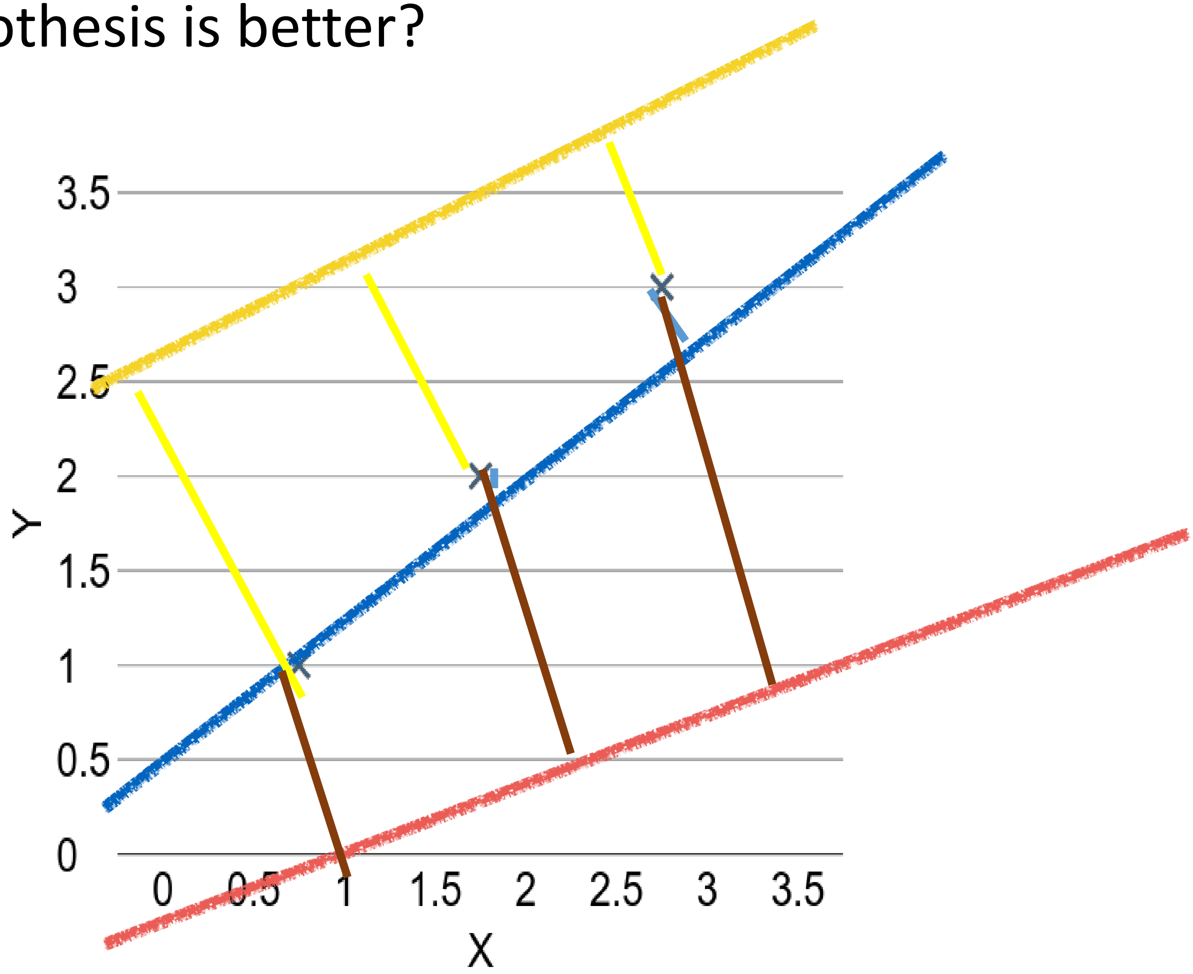
Consider the Simple problem first

x	y
1	1
2	2
3	3



(Linear) Hypothesis $H(x) = Wx + b$

Which hypothesis is better?

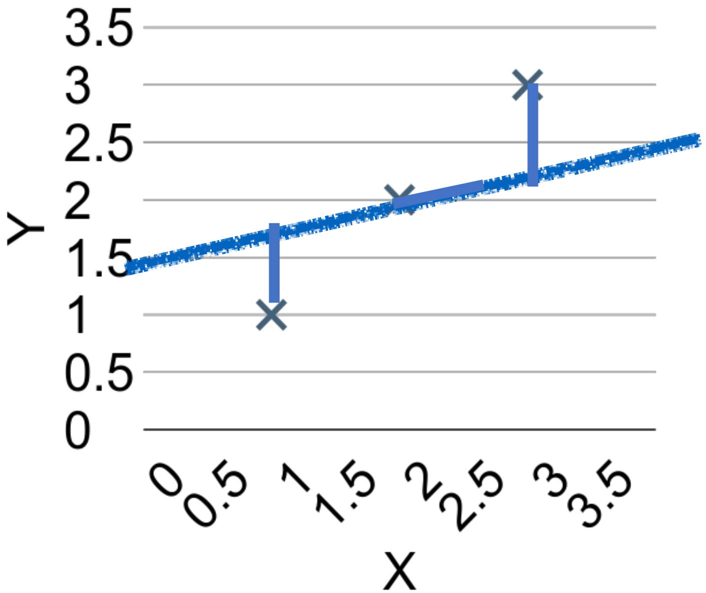


Cost function

- How fit the line to our (training) data

$$H(x) = Wx + b$$

$$H(x) - y$$



$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$

$$\boxed{cost = \frac{1}{m} \sum_{i=1}^{m} (H(x^{(i)}) - y^{(i)})^2}$$

Cost function

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2 \qquad H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

***y* is ground truth**

- Goal: minimize the cost

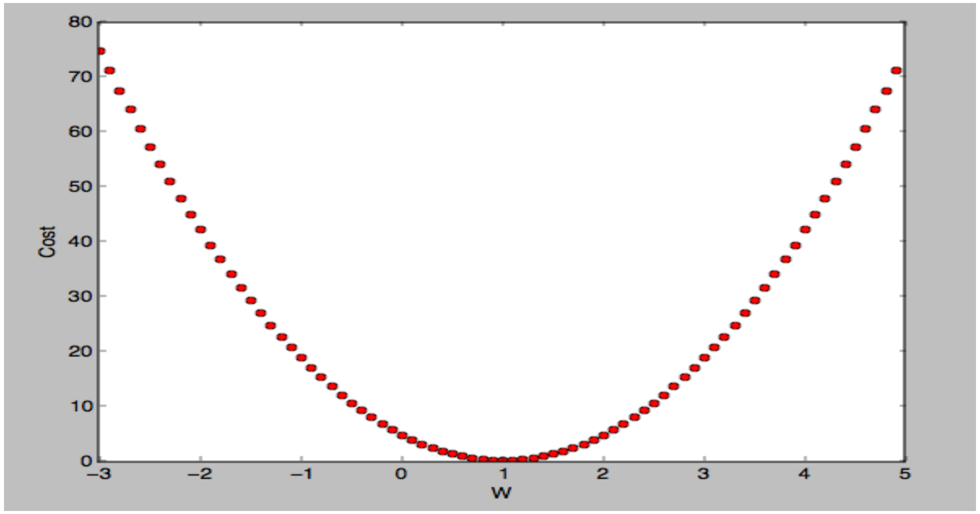
$$\underset{W,b}{\text{minimize}}\, cost(W, b)$$

How to minimize the cost?

x	Y
1	1
2	2
3	3

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

- w = 1 , cost(w) = ?
- w = 0 , cost(w) = ?
- w = 2, cost(w) = ?



How to minimize the cost?

- Gradient descent algorithm(GDA)
  - Minimize cost function
  - Gradient descent is used many minimization problems
  - For a given cost function, ***cost* (*W*, *b*),** it will find ***W*, *b*** to minimize cost
  - It can be applied to more general function: ***cost* (*w1*, *w2*, …)**

How to minimize the cost?

- How it works?
  - Start with initial guesses
  - Start at 0,0 (or any other value)
  - Keeping changing **W** and **b** a little bit to try and reduce **cost(W, b)**
  - Each time you change the parameters, you select the gradient which reduces **cost(W, b)** the most possible
  - Repeat
  - Do so until you converge to a local minimum
  - Has an interesting property
  - Where you start can determine which minimum you end up

Formal definition of GDA

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2 \Rightarrow cost(W) = \frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

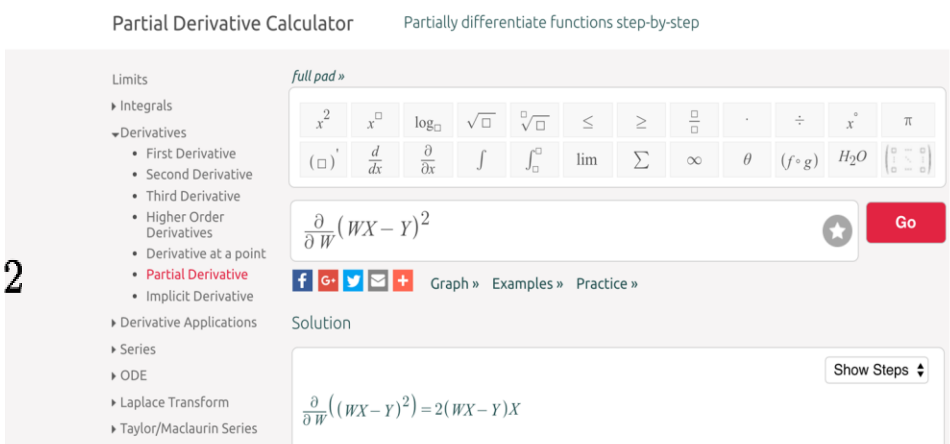
Weight update with gradient

$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

$$W := W - \alpha\frac{\partial}{\partial W}\frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha\frac{1}{2m}\sum_{i=1}^{m}2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$\boxed{W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}}$$

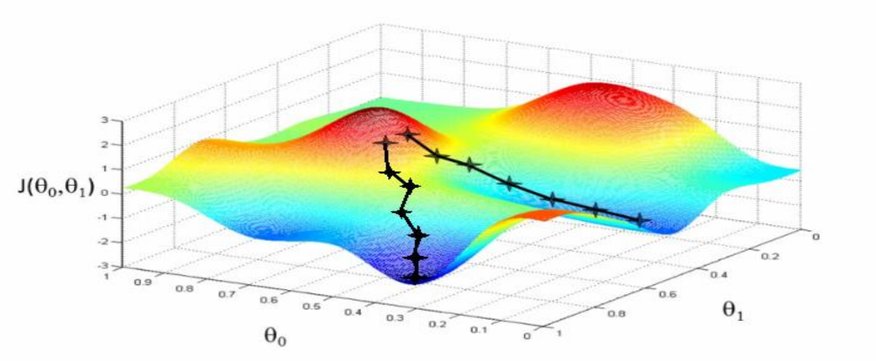


https://www.symbolab.com/solver/partial-derivative-calculator

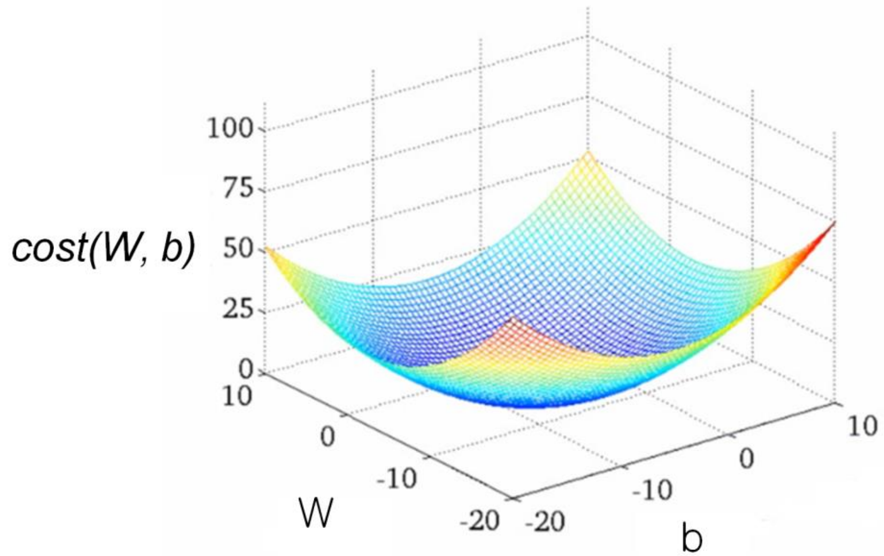
GDA

- GDA works well for the convex function

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)}) x^{(i)}$$



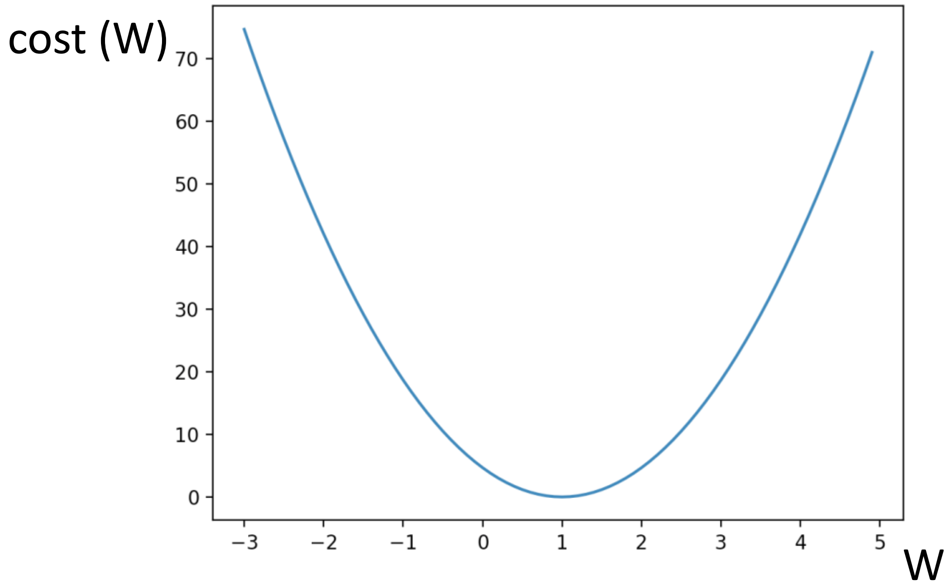
NOT convex function



convex function

TF code with simplified hypothesis

Gradient descent



$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)}) x^{(i)}$$

$$cost(W) = \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)})^2$$

Build graph using PyTorch operations

1

$$H(x) = Wx + b$$

```
# X and Y data
x_train = [[1], [2], [3]]
y_train = [[1], [2], [3]]

X = Variable(torch.Tensor(x_train))
Y = Variable(torch.Tensor(y_train))
# Our hypothesis XW+b
model = nn.Linear(1, 1, bias=True)
```

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

```
# cost criterion
criterion = nn.MSELoss()
```

Build graph using PyTorch operations

1

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

```
# cost criterion
criterion = nn.MSELoss()
```

Optimizer

```
# Minimize
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

Train the model

2

```python
# Train the model
 for step in range(2001):
     optimizer.zero_grad()
     # Our hypothesis
     hypothesis = model(X)
     cost = criterion(hypothesis, Y)
     cost.backward()
     optimizer.step()

     if step % 20 == 0:
         print(step, cost.data.numpy(), model.weight.data.numpy(),
               model.bias.data.numpy())
```

Test the model

3

```
# Testing our model
predicted = model(Variable(torch.Tensor([[5]])))
print(predicted.data.numpy())
predicted = model(Variable(torch.Tensor([[2.5]])))
print(predicted.data.numpy())
predicted = model(Variable(torch.Tensor([[1.5], [3.5]])))
print(predicted.data.numpy())
```

Full code

```python
# lab 2 Linear Regression
import torch
import torch.nn as nn
from torch.autograd import Variable

torch.manual_seed(777)   # for reproducibility

# X and Y data
x_train = [[1], [2], [3]]
y_train = [[1], [2], [3]]

X = Variable(torch.Tensor(x_train))
Y = Variable(torch.Tensor(y_train))

# Our hypothesis XW+b
model = nn.Linear(1, 1, bias=True)

# cost criterion
criterion = nn.MSELoss()

# Minimize
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

```python
# Train the model
for step in range(2001):
    optimizer.zero_grad()
    # Our hypothesis
    hypothesis = model(X)
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 20 == 0:
        print(step, cost.data.numpy(), model.weight.data.numpy(),
              model.bias.data.numpy())

# Testing our model
predicted = model(Variable(torch.Tensor([[5]])))
print(predicted.data.numpy())
predicted = model(Variable(torch.Tensor([[2.5]])))
print(predicted.data.numpy())
predicted = model(Variable(torch.Tensor([[1.5], [3.5]])))
print(predicted.data.numpy())
```

```
'''
0 2.82329 [ 2.12867713] [-0.85235667]
20 0.190351 [ 1.53392804] [-1.05059612]
40 0.151357 [ 1.45725465] [-1.02391243]
...

1920 1.77484e-05 [ 1.00489295] [-0.01112291]
1940 1.61197e-05 [ 1.00466311] [-0.01060018]
1960 1.46397e-05 [ 1.004444] [-0.01010205]
1980 1.32962e-05 [ 1.00423515] [-0.00962736]
2000 1.20761e-05 [ 1.00403607] [-0.00917497]
'''
```

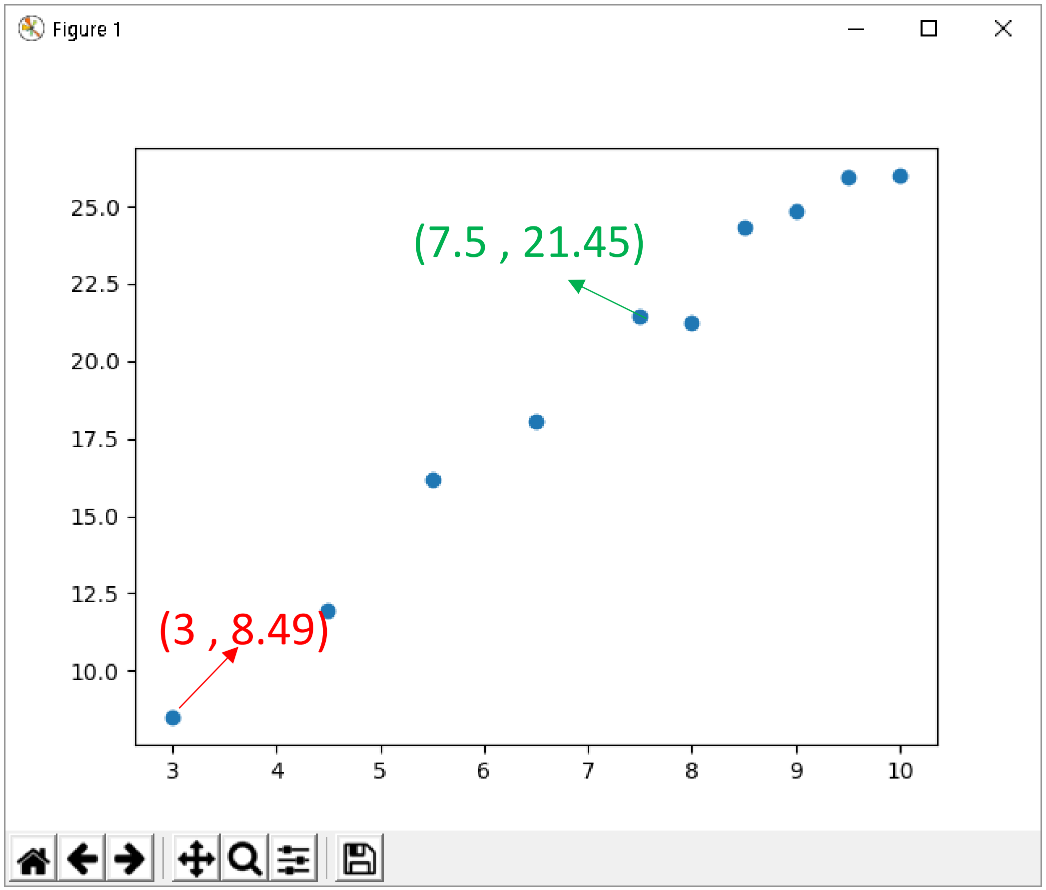
HW-Lienar regression

1. 주어진 데이터에 대한 최적의 w와 b를 구하시오
   - 데이터 설명: x는 학생이 공부한 시간이고, y는 성적이다.
   - 학생이 [3.5, 4, 5]시간 공부했을 때의 예상 점수는 ?

x (hours)	y (score)
10	90
9	80
3	50
2	30

HW-Lienar regression

2. 아래의 점들을 지나는 직선 방정식을 구하시오.



x	3	4.5	5.5	6.5	7.5	8.5	8	9	9.5	10
y	8.49	11.93	16.18	18.08	21.45	24.35	21.24	24.84	25.94	26.02

Reference

- **CS231n:** Convolutional Neural Networks for Visual Recognition 2017
  - http://cs231n.stanford.edu/
- **CS224d:** Deep Learning for Natural Language Processing 2017
  - http://cs224d.stanford.edu/
- **CS20SI:** Tensorflow for Deep Learning Research 2017
  - http://web.stanford.edu/class/cs20si/
- **CS294:** Deep Reinforcement Learning 2017
  - http://rll.berkeley.edu/deeprlcourse/
- 모두를 위한 딥러닝 : https://hunkim.github.io/ml/
- 테리의 딥러닝 토크:
- https://www.youtube.com/playlist?list=PL0oFI08O71gKEXITQ7OG2SCCXkrtid7Fq