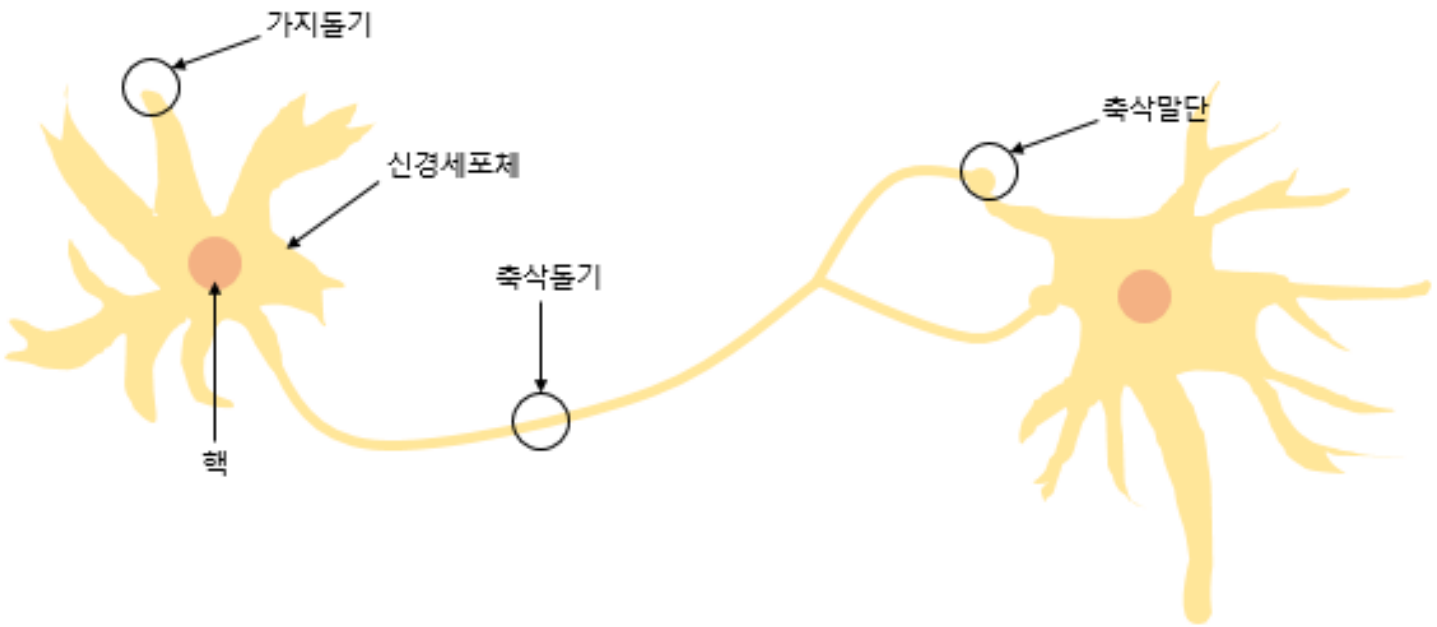
Week03 발표

Perceptron & Multi Layer Perceptron

이경선

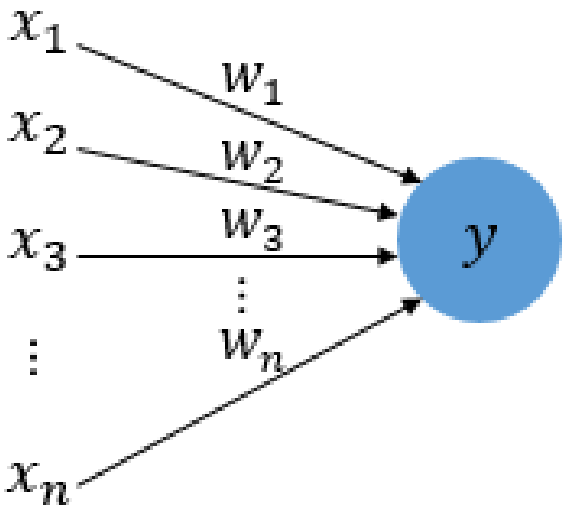
Perceptron



다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
신경 세포 뉴런의 동작과 유사하다.

Perceptron

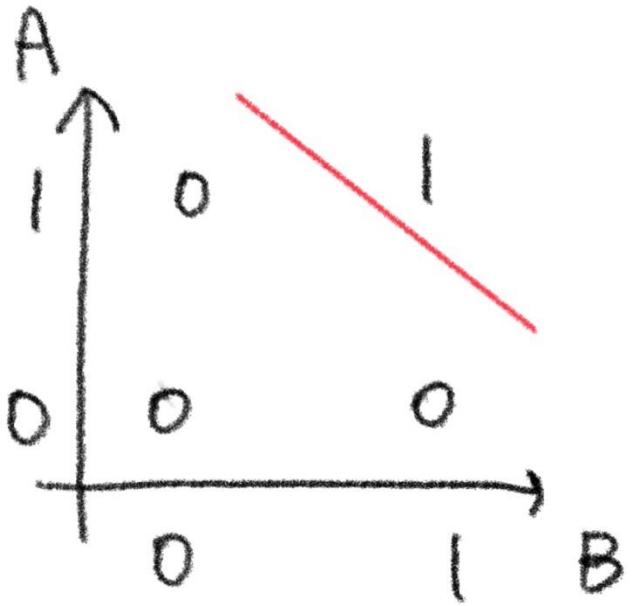
X는 입력값



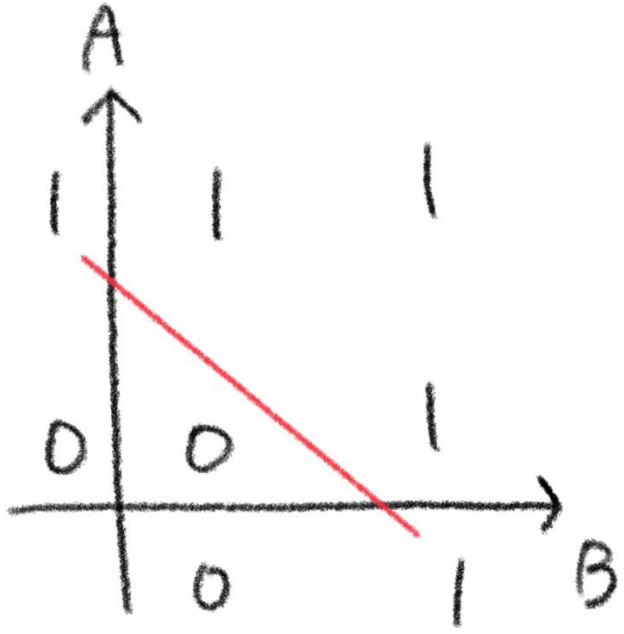
Y는 출력값

단일 인공신경망

AND

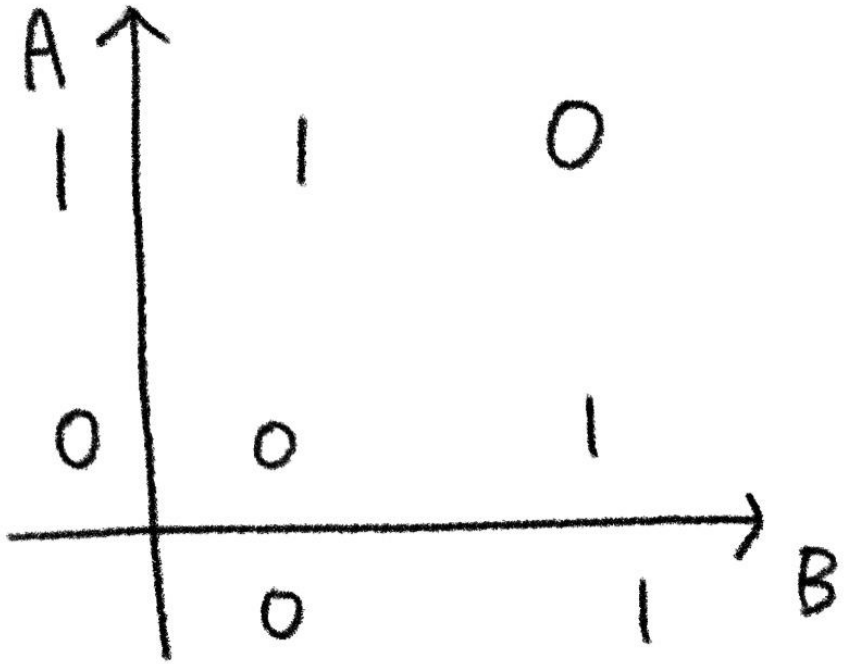


OR



단일 인공신경망

XOR



단일 인공신경망

단일 인공신경망에 XOR을 학습시켜보자!

In [10]:
```
X=torch.FloatTensor([[0,0],[0,1],[1,0],[1,1]]).to(device)
Y=torch.FloatTensor([[0],[1],[1],[0]]).to(device)
# 여기서 .to(device)의 의미가 무엇인가
#CPU 와 GPU 는 서로 혼합해서 계산할수 없기 때문에  CPU 로 계산하면 CPU device 에 쭉
#GPU 면 GPU 로 쭉 계산하기 위해서 to(device)를 씁니다
print(X)
print(Y)

#nn layers
#퍼셉트론의 레이어를 생성한다.
#우선 입력크기가 2이고 출력크기가 1인 linear 레이어를 하나 생성한다.
#그리고 활성화 함수로 이용할 sigmoid 함수를 불러온다.
linear=torch.nn.Linear(2,1,bias=True)
sigmoid=torch.nn.Sigmoid()

#model
#가져온 레이어들을 연결해서 퍼셉트론 모델을 완성한다.
model=torch.nn.Sequential(linear,sigmoid).to(device)

#define cost/loss&optimizer
#cost/loss 와 optimizer를 정의한다.
#이진 분류를 목적으로 하므로 Binary Cross Entropy Loss를 사용한다.
#최적화 방법으로는 변함없이 경사하강법(SGD)를 사용한다.
criterion = torch.nn.BCELoss().to(device)
optimizer=torch.optim.SGD(model.parameters(),lr=1)
```

단일 인공신경망

```
In [10]: X=torch.FloatTensor([[0,0],[0,1],[1,0],[1,1]]).to(device)
         Y=torch.FloatTensor([[0],[1],[1],[0]]).to(device)
         # 여기서 .to(device)의 의미가 무엇인가
         #CPU 와 GPU 는 서로 혼합해서 계산할수 없기 때문에  CPU 로 계산하면 CPU device 에 쭉
         #GPU 면 GPU 로 쭉 계산하기 위해서 to(device)를 씁니다
         print(X)
         print(Y)
         #nn layers
         #퍼셉트론의 레이어를 생성한다.
         #우선 입력크기가 2이고 출력크기가 1인 linear 레이어를 하나 생성한다.
         #그리고 활성화 함수로 이용할 sigmoid 함수를 불러온다.
         linear=torch.nn.Linear(2,1,bias=True)
         sigmoid=torch.nn.Sigmoid()
         #model
         #가져온 레이어를 순서대로 실행하는 모델을 생성한다.
         model=torch.nn.Sequential(linear,sigmoid).to(device)

         #define cost/loss&optimizer
         #cost/loss 와 optimizer를 정의한다.
         #이진 분류를 목적으로 하므로 Binary Cross Entropy Loss를 사용한다.
         #최적화 방법으로는 변함없이 경사하강법(SGD)를 사용한다.
         criterion = torch.nn.BCELoss().to(device)
         optimizer=torch.optim.SGD(model.parameters(),lr=1)
```

.to(device)의 의미

CPU 와 GPU 는 서로 혼합해서 계산할 수 없기 때문에
CPU 로 계산하면 CPU device에 쭉 계산하고
GPU 면 GPU 로 쭉 계산하기 위해 사용.

단일 인공신경망

단일 인공신경망에 XOR을 학습시켜보자!

In [12]:
```
#이제 모델을 학습시킨다.
#학습 10000번 반복하고, 100의 배수번째 학습때마다 cost를 출력한다.

for step in range(10001):
    optimizer.zero_grad()
    hypothesis=model(X)

    #cost/loss function
    cost=criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100==0:
        print(step,cost.item())
```

```
0 0.6931471824645996
100 0.6931471824645996
200 0.6931471824645996
300 0.6931471824645996
400 0.6931471824645996
500 0.6931471824645996
600 0.6931471824645996
700 0.6931471824645996
800 0.6931471824645996
```

단일 인공신경망

단일 인공신경망에 XOR을 학습시켜보자!

In [14]:
```
with torch.no_grad():
    hypothesis=model(X)
    predicted=(hypothesis>0.5).float()
    accuracy=(predicted==Y).float().mean()
    print('₩nHypothesis: ', hypothesis.detach().cpu().numpy(),
          '₩nCorrect: ',  predicted.detach().cpu().numpy(),
          '₩nAccuracy: ', accuracy.item())
```

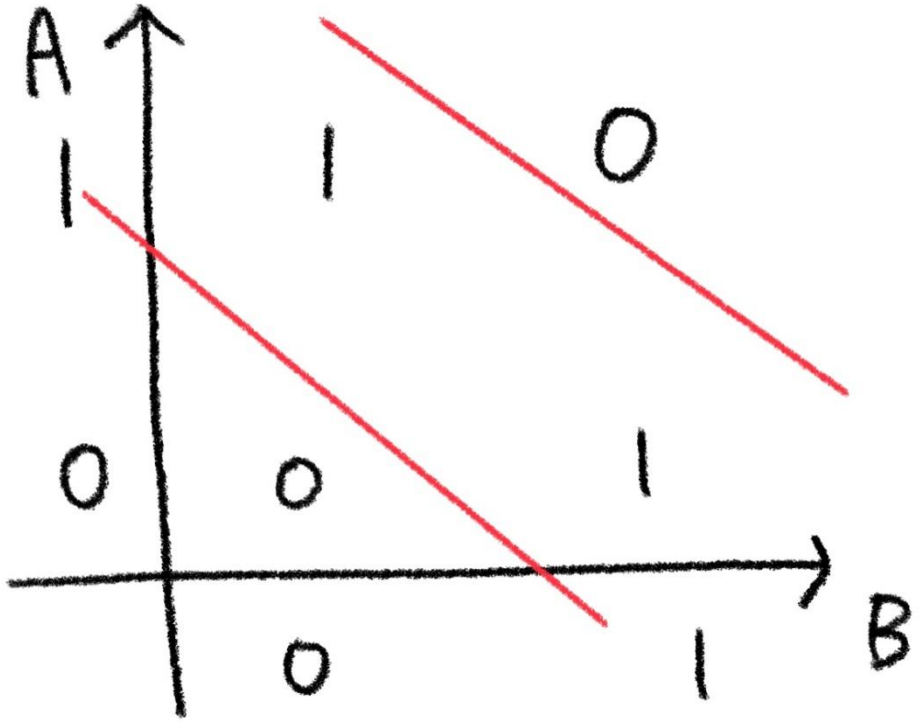
```
Hypothesis:  [[0.5]
 [0.5]
 [0.5]
 [0.5]]
Correct:  [[0.]
 [0.]
 [0.]
 [0.]]
Accuracy:  0.5
```

[0]
[1]
[1]
[0]

따라서 linear 모델로는
XOR문제를 해결할 수 없다!

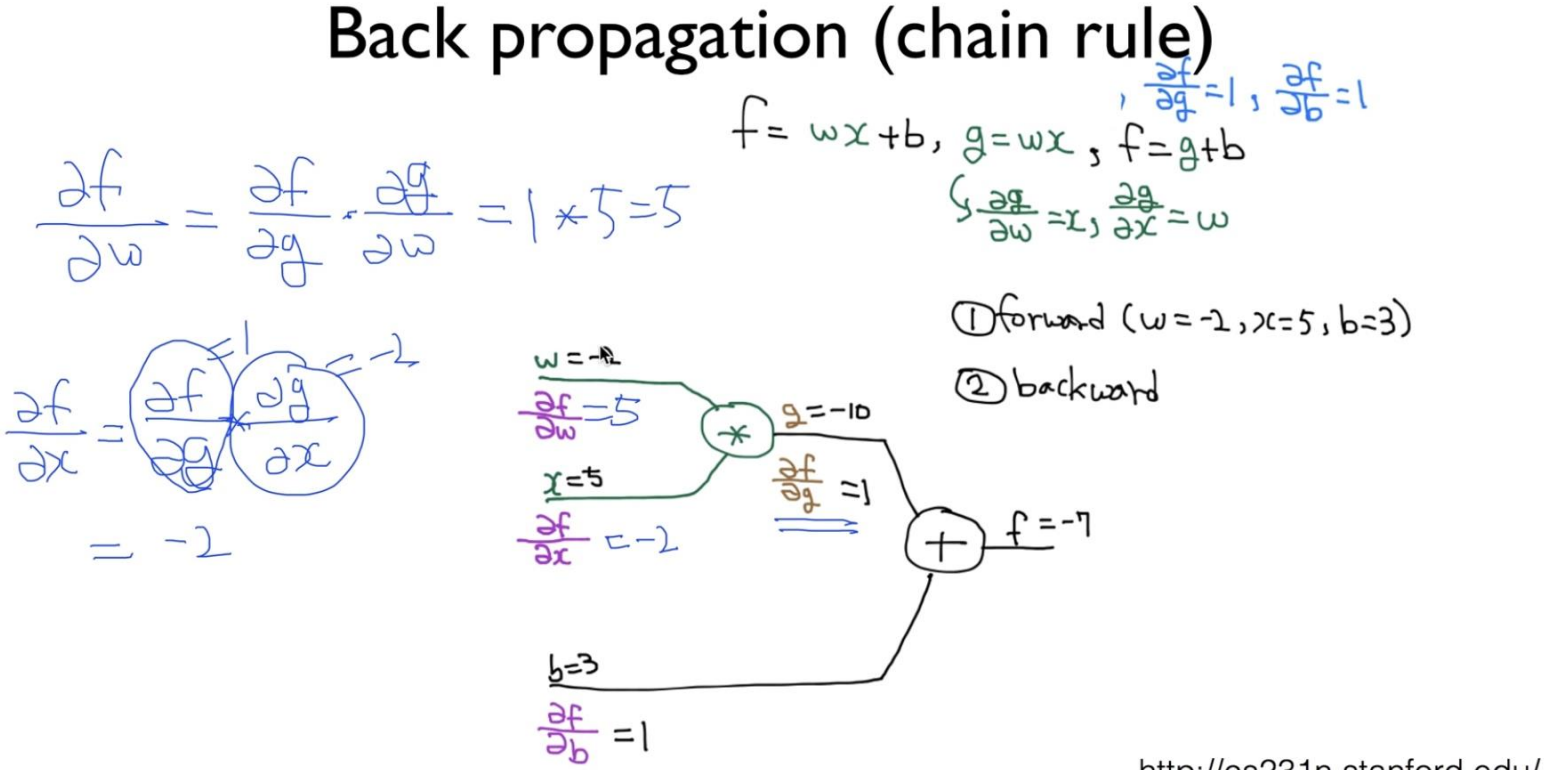
Multi Layer Perceptron

선을 두 개 긋는다고 생각하자!



Multi Layer Perceptron

해결 방법: backpropagation



http://cs231n.stanford.edu/

Multi Layer Perceptron

해결 방법: backpropagation

In [18]:
```
# Backpropagation 예시
X=torch.FloatTensor([[0,0],[0,1],[1,0],[1,1]]).to(device)
Y=torch.FloatTensor([[0],[1],[1],[0]]).to(device)
```

In [19]:
```
w1= torch.Tensor(2,2).to(device)
b1= torch.Tensor(2).to(device)
w2= torch.Tensor(2,1).to(device)
b2= torch.Tensor(1).to(device)

def sigmoid(x):

    return 1.0/(1.0+torch.exp(-x))
# exp 앞에 torch.이 붙은 이유가 뭐지?

def sigmoid_prime(x):
    return sigmoid(x)*(1-sigmoid(x))
```

Multi Layer Perceptron

해결 방법: backpropagation

In [37]:
```
for step in range(10001):
#forward (??????????)
    l1=torch.add(torch.matmul(X,w1),b1)
    a1=sigmoid(l1)
    l2=torch.add(torch.matmul(a1,w2),b2)
    Y_pred=sigmoid(l2)

    cost=-torch.mean(Y*torch.log(Y_pred)+(1-Y)*torch.log(1-Y_pred))
    # backpropagation
    # Loss derivative

    d_Y_pred=(Y_pred-Y)/(Y_pred*(1.0-Y_pred)+1e-7)
    # 1e-7은 아예 0으로 나누어지는 경우를 막기 위해서
    # 이제 점점 앞으로 간다.
    #Layer 2
    d_l2=d_Y_pred*sigmoid_prime(l2)
    d_b2=d_l2
    d_w2=torch.matmul(torch.transpose(a1,0,1),d_b2)
    #transpose는 차원을 반대로 바꿔라. (5,10)>>>(10,5)
    #matmul은 매트릭스 곱

    #Layer 1
    d_a1=torch.matmul(d_b2,torch.transpose(w2,0,1))
    d_l1=d_a1*sigmoid_prime(l1)
    d_b1=d_l1
```

Multi Layer Perceptron

해결 방법: backpropagation

```
    #Layer 1
    d_a1=torch.matmul(d_b2,torch.transpose(w2,0,1))
    d_l1=d_a1*sigmoid_prime(l1)
    d_b1=d_l1
    d_w1=torch.matmul(torch.transpose(X,0,1),d_b1)

    learning_rate=0.2
    w1=w1-learning_rate*d_w1
    b1=b1-learning_rate*torch.mean(d_b1,0)
    w2=w2-learning_rate*d_w2
    b2=b2-learning_rate*torch.mean(d_b2,0)

    if step % 100==0:
        print(step,cost.item())
```

```
0 nan
100 nan
200 nan
300 nan
400 nan
500 nan
600 nan
700 nan
800 nan
```

Multi Layer Perceptron

nn. 코드를 사용하여 조금 더 간편하게 나타내자!

In [3]:
```
#nn layers
linear1=torch.nn.Linear(2,2,bias=True)
#2에서 2로 가는 weight와 bias 자동 설정
linear2=torch.nn.Linear(2,1,bias=True)
#2개의 linear를 가진다!! Multi layer

#2에서 1로 가는 weight와 bias 자동 설정
sigmoid=torch.nn.Sigmoid()
```

In [5]:
```
#모델링
model=torch.nn.Sequential(linear1,sigmoid, linear2, sigmoid).to(device)

# cost/loss/optimizer 정의하기
criterion=torch.nn.BCELoss().to(device)
optimizer=torch.optim.SGD(model.parameters(),lr=1)
```

In [6]:
```
for step in range(10001):
    optimizer.zero_grad()
    hypothesis=model(X)

    #cost loss 함수
    cost=criterion(hypothesis,Y)
    cost.backward()
    optimizer.step()
```

Multi Layer Perceptron

해결 방법: backpropagation

In [7]:
```
# 정확도 계산
# Hypothesis가 0.5보다 크면 true

with torch.no_grad():
    hypothesis=model(X)
    predicted=(hypothesis>0.5).float()
    accuracy=(predicted==Y).float().mean()
    print('₩nHypothesis: ',hypothesis.detach().cpu().numpy(),
          '₩nCorrect: ', predicted.detach().cpu().numpy(),
          '₩nAccuracy: ', accuracy.item())
```

```
Hypothesis:  [[0.00106364]
 [0.99889404]
 [0.99889404]
 [0.00165861]]
Correct:  [[0.]
 [1.]
 [1.]
 [0.]]
Accuracy:  1.0
```

Multi Layer Perceptron

4개의 linear를 이용하자!

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
# nn layers
linear1 = torch.nn.Linear(2, 10, bias=True)
linear2 = torch.nn.Linear(10, 10, bias=True)
linear3=torch.nn.Linear(10,10,bias=True)
linear4=torch.nn.Linear(10,1,bias=True)
sigmoid = torch.nn.Sigmoid()
# model
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid, linear3, sigmoid, line
# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)  # modified learning rate from
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)

    # cost/loss function
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0:
        print(step, cost.item())
```

Multi Layer Perceptron

4개의 linear를 이용하자!

4개의 linear

1600 0.6931051015853882
1700 0.6931012868881226
1800 0.693097054958343 5
1900 0.6930922269821167
2000 0.6930870413780212
2100 0.693081259727478
2200 0.693074643611908

2개의 linear

1600 0.69270324707031 25
1700 0.6923960447311401
1800 0.6917301416397095
1900 0.6899653673171997
2000 0.683831512928009
2100 0.656166017055511 5
2200 0.431095153093338
2300 0.1348916143178939 8
2400 0.06630385667085648