



Type something...

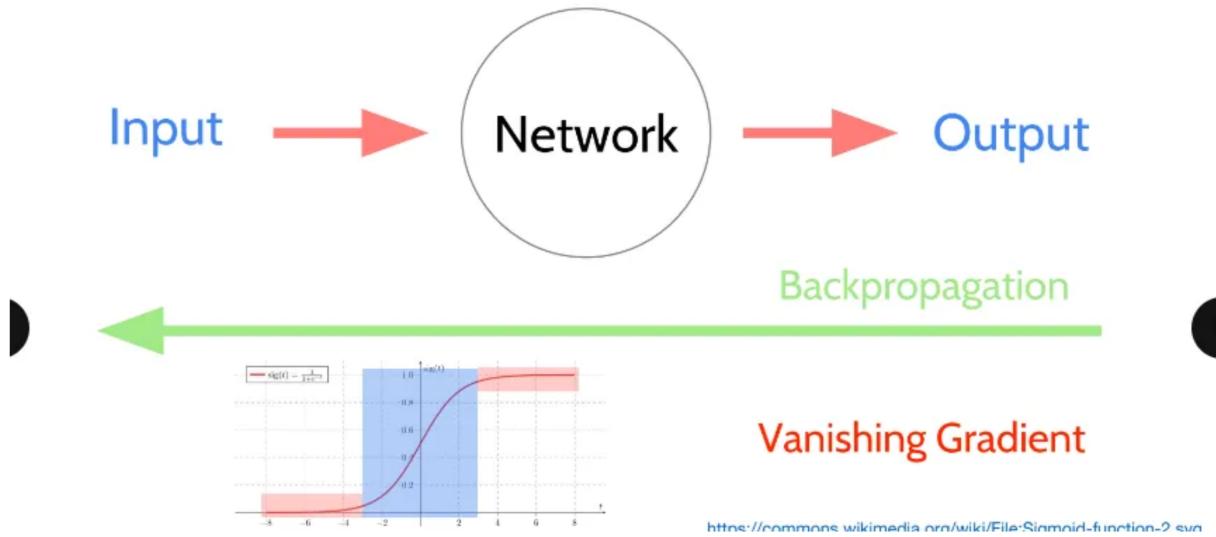
01) ReLU

01\_Problem of Sigmoid

## 01) ReLU

01\_Problem of Sigmoid

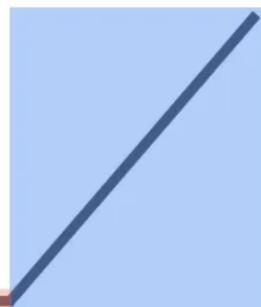
# Problem of Sigmoid



- 0 주변에는 기울기가 잘 존재하지만, 값이 커질수록 기울기가 0으로 수렴하게 된다. 이렇게 되면 backpropagation을 진행할 때 계속 0과 유사한 값이 넘어가져 Vanishing Gradient 즉 기울기 소실문제가 발생하게 된다.

# ReLU

$$f(x) = \max(0, x)$$



`x = torch.nn.sigmoid(x)`

`x = torch.nn.relu(x)`

`torch.nn.sigmoid(x)`  
`torch.nn.tanh(x)`  
`torch.nn.relu(x)`  
`torch.nn.leaky_relu(x, 0.01)`

- 이 문제를 해결하기 위해 나온 함수가 ReLU이다. ReLU는  $\max(0, x)$ 라는 아주 간단한 함수이다. 물론 ReLU도 0이하에서는 기울기가 0이기 때문에 leaky\_relu를 사용하기도 한다.



# Code: mnist\_nn

```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
...

# MNIST data image of shape 28 * 28 = 784
linear1 = torch.nn.Linear(784, 256, bias=True).to(device)
linear2 = torch.nn.Linear(256, 256, bias=True).to(device)
linear3 = torch.nn.Linear(256, 10, bias=True).to(device)
relu = torch.nn.ReLU()

# Initialization
torch.nn.init.normal_(linear1.weight)
torch.nn.init.normal_(linear2.weight)
torch.nn.init.normal_(linear3.weight)

# model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3).to(device)

# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)      # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

- ReLU를 activation function으로 사용하는 모델을 만들어보자. Linear → relu → Linear → relu → linear 를 이용해서 정확도를 구하면 94프로로 아까전 linear 1개 모델보다 6%정도 개선되었다.

```
#parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

#setting
linear1 = torch.nn.Linear(28*28, 256, bias = True).to(device) # 2^8 = 256, 2의 배수
linear2 = torch.nn.Linear(256, 256, bias = True).to(device)
linear3 = torch.nn.Linear(256, 10, bias = True).to(device)
relu = torch.nn.ReLU()

#Initialization
torch.nn.init.normal_(linear1.weight) # 초기화
torch.nn.init.normal_(linear2.weight)
torch.nn.init.normal_(linear3.weight)

#model
model = torch.nn.Sequential(linear1, relu, linear2, relu, linear3).to(device)

#define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device) # Loss = Softmax
optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
```

음수 입력에 대해서는 항상 0을 출력하므로 미분하였을 때 기울기가 0인데 어떻게 기울기 소실 문제를 해결하는걸까? 애초에 해결 할 방법은 있는 것인가?

```
Epoch: 0001 cost = 129.332550049
Epoch: 0002 cost = 36.226955414
Epoch: 0003 cost = 22.935277939
Epoch: 0004 cost = 16.051362991
Epoch: 0005 cost = 11.619537354
Epoch: 0006 cost = 8.622934341
Epoch: 0007 cost = 6.308451176
Epoch: 0008 cost = 4.843585014
Epoch: 0009 cost = 3.543032646
Epoch: 0010 cost = 2.632565498
Epoch: 0011 cost = 2.056926012
Epoch: 0012 cost = 1.580777287
Epoch: 0013 cost = 1.261717200
Epoch: 0014 cost = 1.084918380
Epoch: 0015 cost = 0.804111660
Learning finished
Accuracy: 0.9465000033378601
```



ReLU 함수를 사용하는 유닛이 음수 값을 입력 받게 되면 ReLU의 출력은 0이 되므로 기울기는 0이 된다.

이러한 경우 더이상 해당 유닛은 가중치를 업데이트 하지 못하고 비활성화되어 죽은 유닛이 된다. 이를 **Dying ReLU**라고 부른다.

그렇기에 ReLU의 다른 버전 Leaky ReLU와 같은 함수들이 등장한 것이다.

그럼에도 ReLU는 현재 매우 많은 층을 쌓는 심층 신경망에서 자주 사용되는 활성화 함수로 손에 꼽히며 많은 인기를 받고 있다.

이유가 무엇일까.

- ReLU는 음수 값에서는 가중치를 업데이트 하지 못하고 비활성화 된다. 하지만 ReLU가 인기가 많은 이유는 무엇일까?
- <https://velog.io/@wndudwkd003/딥러닝-활성화-함수와-비선형-활성화-함수-참조를-남긴다>.
- ReLU는 hidden-layer에서는 사용하면 매우 좋지만, output layer에 사용하면 안 좋은 결과를 낼기 때문에 hidden-layer에서만 사용하자.

