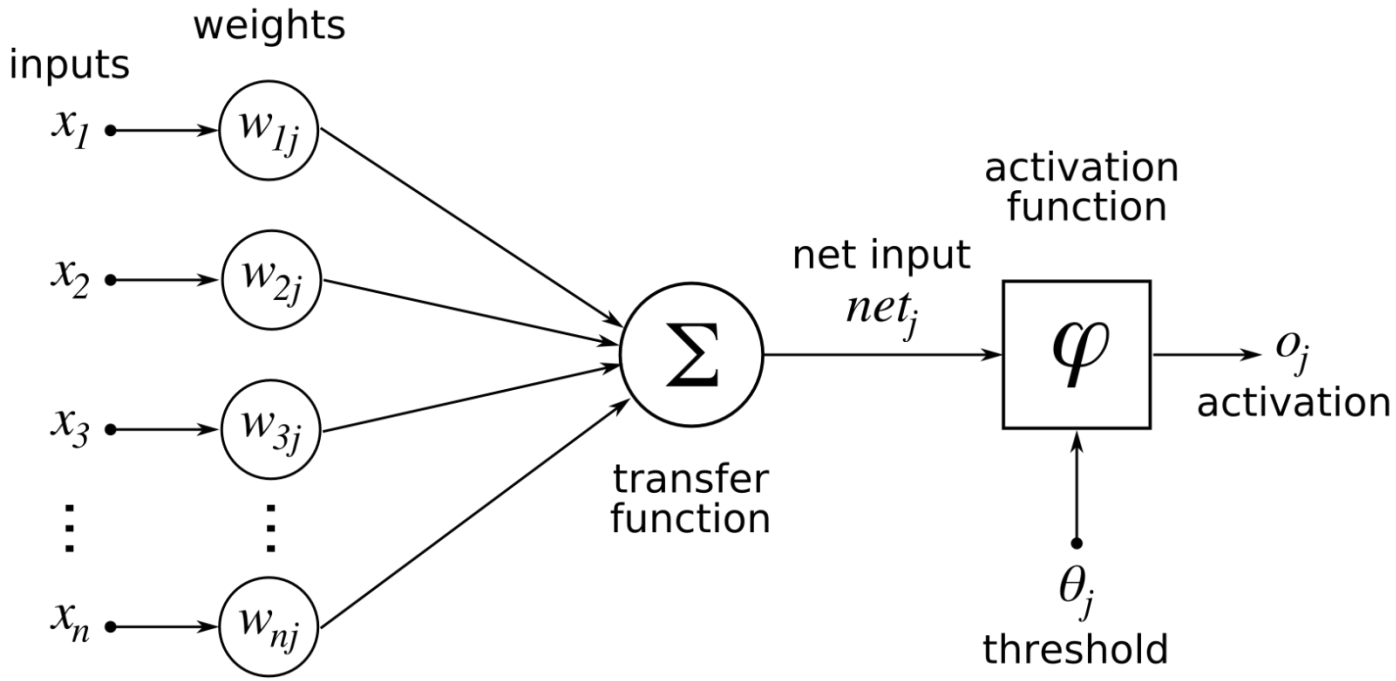
ReLU, Weight Initialization

2022-01-27

Motivation of Activation Function

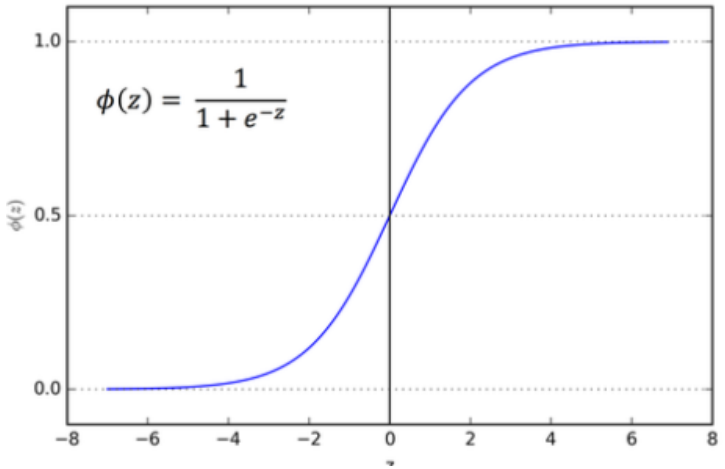
- 입력 신호 → 활성화 함수 → 출력 신호
  - 입력 신호의 총합이 활성화를 일으키는지 결정하는 역할

- $b + w_1 x_1 + w_2 x_2$
- $y = h(a)$



From Sigmoid to ReLU (1)

- Sigmoid의 문제점 : 어느 지점에서 gradient가 거의 0에 가깝다
  - Backpropagation을 하면서 gradient을 전파할 때 activation function의 gradient을 곱하게 됨
  - 아주 작은 값을 계속 곱하게 된다면 앞 단으로 갈수록 gradient가 소멸해버린다. (vanishing gradient)



From Sigmoid to ReLU (2)

- $f(x) = \max(0, x)$
- 최근 신경망 분야에서 많이 사용하는 activation function

ReLU with PyTorch

```
class DNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = torch.nn.Linear(784, 256, bias=True)
        self.linear2 = torch.nn.Linear(256, 256, bias=True)
        self.linear3 = torch.nn.Linear(256, 10, bias=True)
        self.relu = torch.nn.ReLU()
        self.model = torch.nn.Sequential(self.linear1, self.relu,
                                         self.linear2, self.relu,
                                         self.linear3).to(device)

    def forward(self, x):
        return self.model(x)
```

모든 정의

28*28 -> 256 -> 10

포함된 객체를 순차적으로 실행

가져와서 쓰기만 하자!

Weight Initialization

- 가중치를 초기화하는 것이 모델 성능에 중요한 영향을 미친다.

- 그렇다면 가중치를 어떻게 초기화할 것인가?
  - 가중치는 작아야 한다. (학습 모델에 특화되면 안되므로)
  - 하지만 균일해서는 안된다. (모든 가중치의 값이 똑같이 갱신되므로)

  ➔ 작은 가중치를 무작위로 설정하자

Xavier Initialization with sigmoid or tanh (S 모양의 activation function)

```
class dnn_xavier(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = torch.nn.Linear(784, 256, bias=True)
        self.linear2 = torch.nn.Linear(256, 256, bias=True)
        self.linear3 = torch.nn.Linear(256, 10, bias=True)
        self.relu = torch.nn.ReLU()

        torch.nn.init.xavier_uniform_(self.linear1.weight)
        torch.nn.init.xavier_uniform_(self.linear2.weight)
        torch.nn.init.xavier_uniform_(self.linear3.weight)

        self.model = torch.nn.Sequential(self.linear1, self.relu, self.linear2, self.relu, self.linear3)

    def forward(self, x):
        return self.model(x)
```

nn.init.xavier_uniform_(신경망.weight)

He Initialization with ReLU

```
class dnn_he(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = torch.nn.Linear(784, 256, bias=True)
        self.linear2 = torch.nn.Linear(256, 256, bias=True)
        self.linear3 = torch.nn.Linear(256, 10, bias=True)
        self.relu = torch.nn.ReLU()

        torch.nn.init.kaiming_uniform_(self.linear1.weight)
        torch.nn.init.kaiming_uniform_(self.linear2.weight)
        torch.nn.init.kaiming_uniform_(self.linear3.weight)

        self.model = torch.nn.Sequential(self.linear1, self.relu, self.linear2, self.relu, self.linear3)

    def forward(self, x):
        return self.model(x)
```

nn.init.kaiming_uniform_(신경망.weight)

Tips for Weight Initialization

```
def weights_init(m): 레이어가 nn.Linear의 인스턴스이면
    if isinstance(m, nn.Linear): # 모델의 모든 MLP 레이어에 대해서
        nn.init.xavier_normal_(m.weight) # Weight를 xavier_normal로 초기화
        print(m.weight)
```

어떤 레이어(m)가 nn.Linear의 인스턴스이면
레이어 m의 weight을 초기화하는 함수를 만들고

```
model = DNN().to(device)
model.apply(weights_init) # 모델에 weight_init 함수를 적용하여 weight를 초기화
```

모델을 정의한 다음, nn.Linear의 모든 submodule에 함수를 적용할 수 있다.
(torch.nn.module.apply() 함수 이용)