Weight Initialization

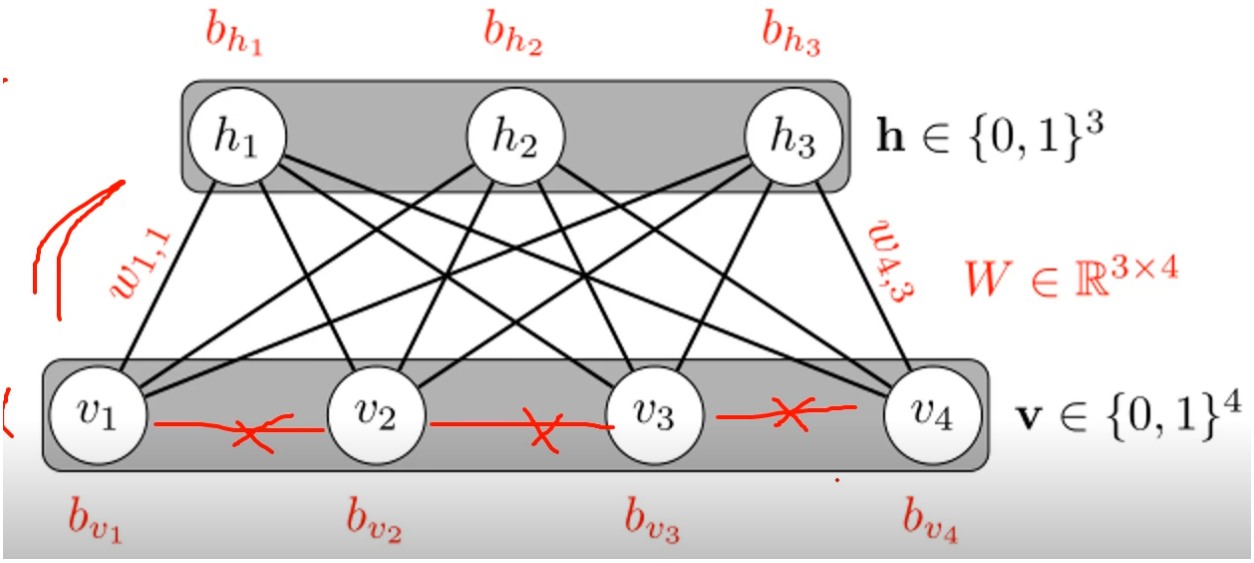
상태	DNN
담당자	

> 💡 Weight을 잘 initialize하는 것이 딥러닝 모델의 성능에 중요한 영향을 미친다.

여기서 말하는 '좋은 모델 성능' 은 overfitting이 억제된 모델을 의미한다. 학습 데이터셋에 특화된 가중치를 작게 만들면 오버피팅이 잘 일어나지 않기 때문에 가중치 값을 작게 하는 것은 오버피팅을 막을 수 있는 방법 중 하나이다.

하지만 가중치의 초깃값을 모두 0으로 해서는 안 되는데(가중치를 균일한 값으로 설정하면 안 되는데), backpropagation에서 모든 가중치의 값이 똑같이 갱신되기 때문이다. 따라서 초깃값은 작되, 무작위로 설정되어야 한다.

Restricted Boltzmann Machine



레이어 간에는 연결이 없지만, 다른 레이어 사이에는 fully-connected 형태의 머신으로, input X → output Y 로의 encoding, output Y → input X' 로의 decoding을 수행할 수 있다.

RBM을 여러 개 쌓아서 weight initalization을 수행할 수 있으나, 요즘은 잘 사용하지 않는 방법이다.

Xavier

> 💡 with sigmoid, tanh

RBM이 무작위로 가중치를 초기화했다면 Xavier는 layer의 특성에 따라 가중치를 초기화한다. Xavier는 sigmoid나 tanh 같은 S자 곡선의 activation function과 궁합이 잘 맞는 가중치 초기화 방법이다.

```
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)
```

Xavier Normal Initialization

$W \sim N(0, Var(W)), Var(W) = \sqrt{\frac{2}{n_{in}+n_{out}}}$

Xavier Uniform Distribution

$W \sim U(-\sqrt{\frac{6}{n_{in}+n_{out}}}, +\sqrt{\frac{6}{n_{in}+n_{out}}})$

여기서 n_{in}은 레이어의 인풋 수, n_{out}은 레이어의 아웃풋 수를 의미한다.

He Initialization

> 💡 with ReLU

Xavier의 변형으로, normal initialization과 uniform initialization이 모두 있고 Xavier와 상당히 유사하다. ReLU와 궁합이 잘 맞는 가중치 초기화 방법이다. (그런데 이 예제에서는 오히려 Xavier에서 accuracy가 더 높게 나왔다.)

```
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)
```

He Normal Initialization

$W \sim N(0, Var(W)), Var(W) = \sqrt{\frac{2}{n_{in}}}$

He Uniform Distribution

$$W \sim U(-\sqrt{\frac{6}{n_{in}}}, +\sqrt{\frac{6}{n_{in}}})$$