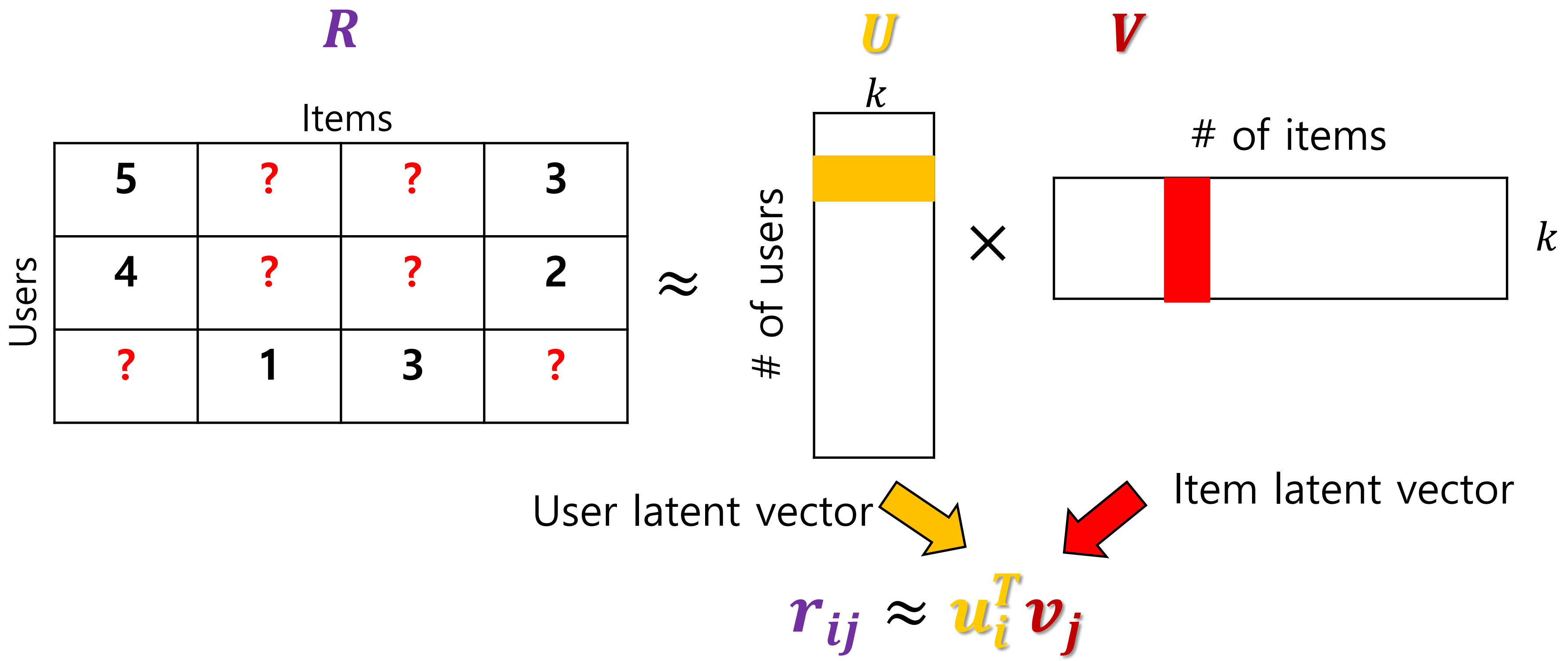
Matrix Factorization

- A popular model-based collaborative filtering
- Matrix factorization is introduced because of sparsity of real data



Matrix Factorization

- Given
  - Sparse rating matrix $R \in \mathbb{R}^{N \times M}$
- Minimize
  - $J(\boldsymbol{U}, \boldsymbol{V}) = \sum_i^N \sum_j^M I_{ij}\left(r_{ij} - \boldsymbol{u}_i^T \boldsymbol{v}_j\right)^2$
  - where
    - Latent user matrix $\boldsymbol{U} = [\boldsymbol{u}_1 \quad \cdots \quad \boldsymbol{u}_N] \in \mathbb{R}^{k \times N}$
    - Latent item matrix $\boldsymbol{V} = [\boldsymbol{v}_1 \quad \cdots \quad \boldsymbol{v}_M] \in \mathbb{R}^{k \times M}$
    - I_{ij} means indicator function s.t
      - $I_{ij} = 1$ if user i rated item j
      - $I_{ij} = 0$, otherwise

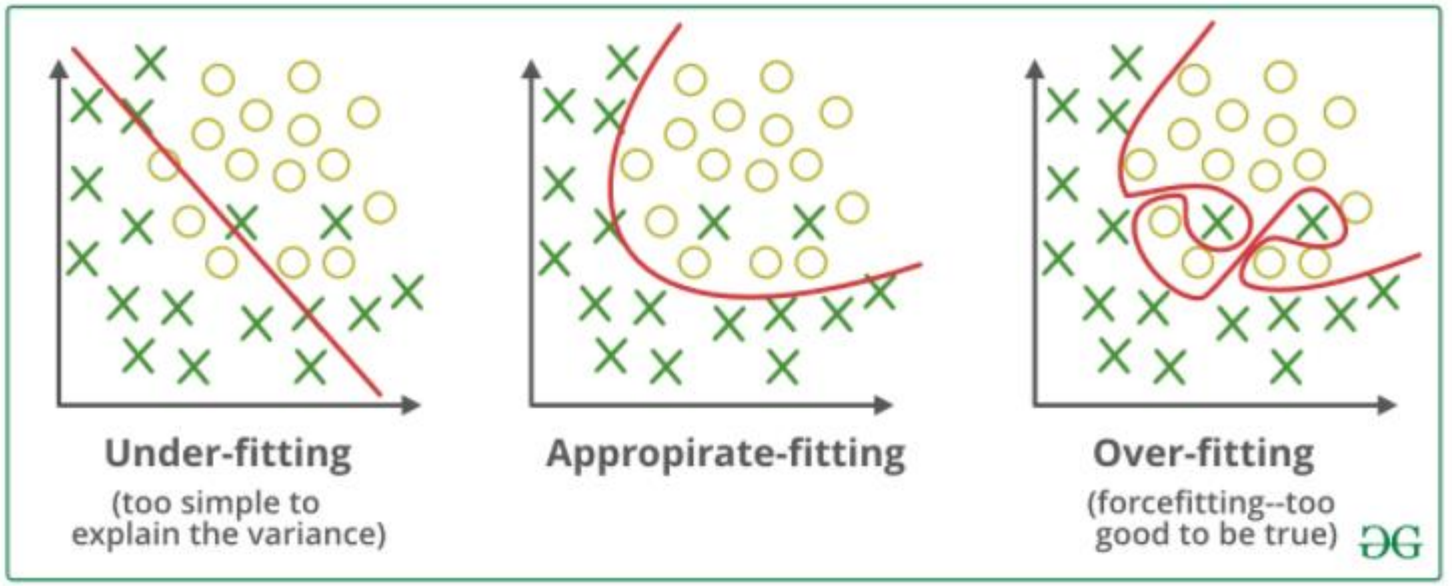
Regularization

한양대학교 ERICA 소프트웨어융합대학 COLLEGE OF COMPUTING | 인공지능학과 Department of Artificial Intelligence

정 우 환 (whjung@hanyang.ac.kr)

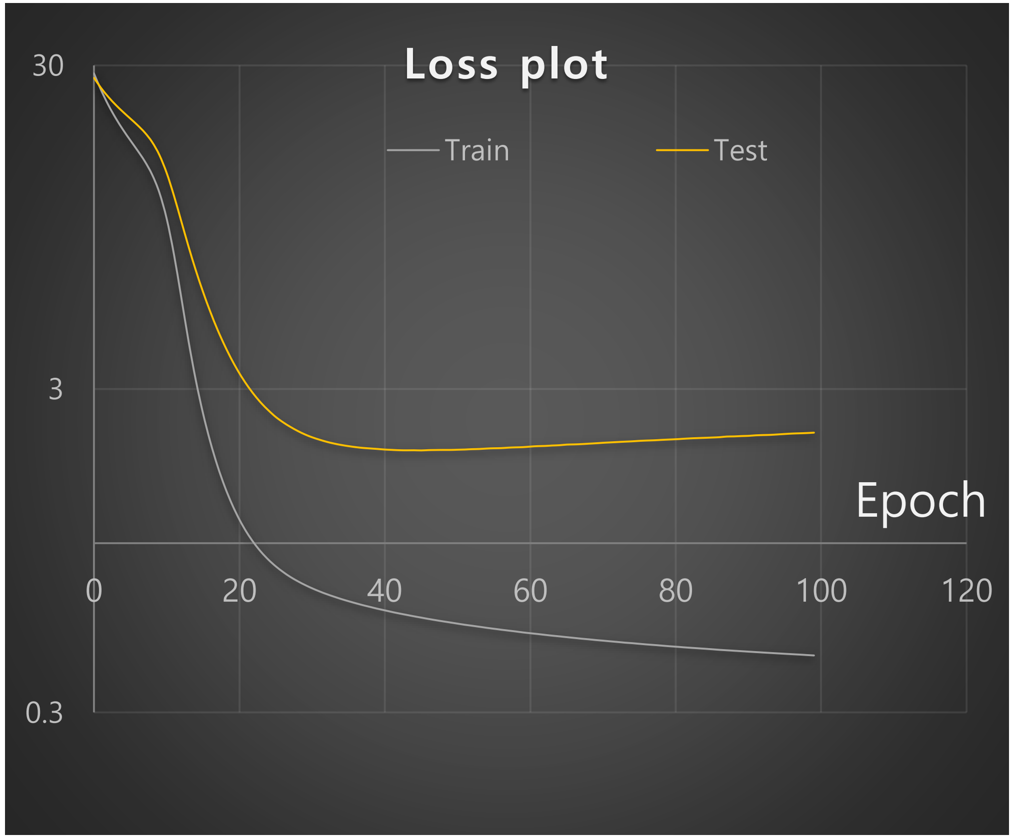
Fall 2022

Underfitting and Overfitting



dewangNautiyal
Geekforgeeks

Overfitting



- Matrix factorization
- Settings
  - Adam(lr = 0.01)
  - K = 16

Regularization

Regularization: Logistic Regression

- $\hat{y} = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$

$$J(\boldsymbol{w}, b) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right)}_{\text{Prediction cost}} \quad \underbrace{+ \frac{1}{m}\frac{\lambda}{2}\|\boldsymbol{w}\|_2^2 \;\; + \frac{1}{m}\frac{\lambda}{2} b^2}_{\text{Regularization}}$$

L2 regularization $\quad \|\boldsymbol{w}\|_2^2 = \sum_{j=1}^{n_x} w_j^2$

L1 regularization $\quad \|\boldsymbol{w}\|_1^1 = \sum_{j=1}^{n_x} |w_j|$

Regularization: Neural Networks

$$J\left(\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}, \boldsymbol{W}^{[2]}, \boldsymbol{b}^{[2]} \ldots \boldsymbol{W}^{[L]}, \boldsymbol{b}^{[L]}\right) = \underbrace{\frac{1}{m}\sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right)}_{\text{Prediction cost}} + \underbrace{\frac{\lambda}{2m}\sum_{l=1}^{L}\left\|\boldsymbol{W}^{[l]}\right\|_F^2}_{\text{Regularization}}$$

Frobenious norm $\left\|\boldsymbol{W}^{[l]}\right\|_F = \sqrt{\sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}\left(W_{i,j}^{[l]}\right)^2}$

$$\frac{\partial\left\|\boldsymbol{W}^{[l]}\right\|_F^2}{\partial W_{i,j}^{[l]}} = W_{i,j}^{[l]}$$

$$\boldsymbol{W}^{[l]} \coloneqq \boldsymbol{W}^{[l]} - \eta\left[\nabla J_{pred} + \frac{\lambda}{2m}\boldsymbol{W}^{[l]}\right]$$

a.k.a. Weight Decay

Regularization: Matrix Factorization

$$J(\boldsymbol{U},\boldsymbol{V}) = \underbrace{\sum_{i}^{N}\sum_{j}^{M} I_{ij}\left(r_{ij} - \boldsymbol{u}_i^T\boldsymbol{v}_j\right)^2}_{\text{Prediction cost}} \underbrace{+\frac{\lambda}{2}\|\boldsymbol{U}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{V}\|_F^2}_{\text{Regularization}}$$

L_2 regularization and probability

L_2 regularization

$$minimize \frac{\lambda}{2m} \sum_{i=1}^{n^{[l]}} \sum_{j=1}^{n^{[l-1]}} \left(W_{i,j}^{[l]}\right)^2$$

Maximum likelihood estimation

$$W_{i,j}^{[l]} \sim N(0, \frac{1}{\lambda})$$

PDF $$f\left(W_{ij}^{[l]}\right) = \sqrt{\frac{\lambda}{2\pi}} \exp\left(-\frac{\lambda}{2} {W_{ij}^{[l]}}^2\right)$$

Log likelihood $$\log f\left(W_{ij}^{[l]}\right) = -\frac{\lambda}{2} {W_{ij}^{[l]}}^2$$

L_2 Regularization in PyTorch

- Directly add to the loss

```
outputs = model(inputs)
loss = criterion(outputs, labels)

for name, param in model.named_parameters():
    if "weight" in name:
        loss = loss + torch.norm(param)
```

```
for name, param in model.named_parameters():
    print(f"{name}  ₩t{param.shape}")
```

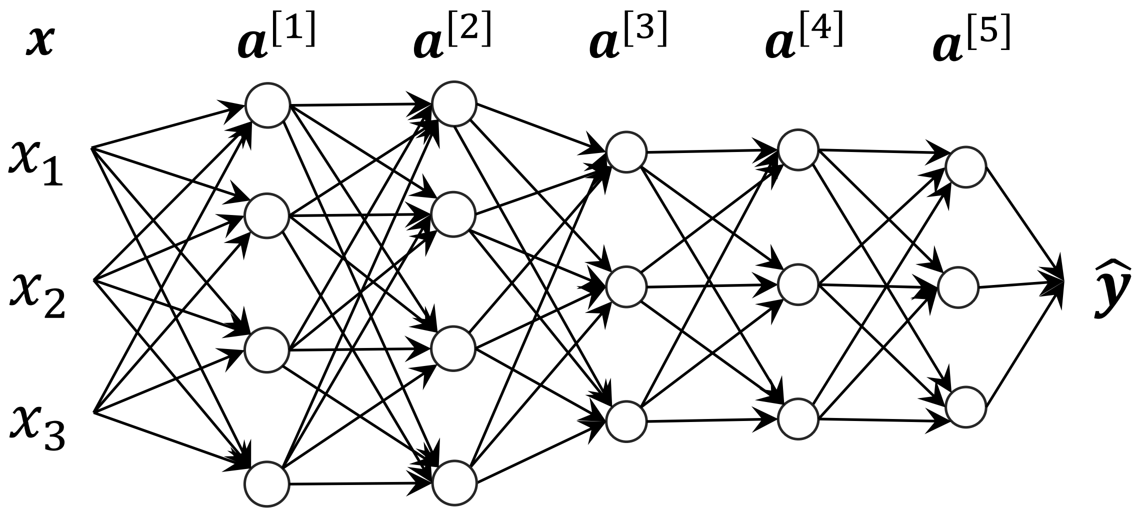
```
l_layers.0.weight      torch.Size([512, 784])
l_layers.0.bias        torch.Size([512])
l_layers.1.weight      torch.Size([256, 512])
l_layers.1.bias        torch.Size([256])
l_layers.2.weight      torch.Size([128, 256])
l_layers.2.bias        torch.Size([128])
l_layers.3.weight      torch.Size([10, 128])
l_layers.3.bias        torch.Size([10])
```

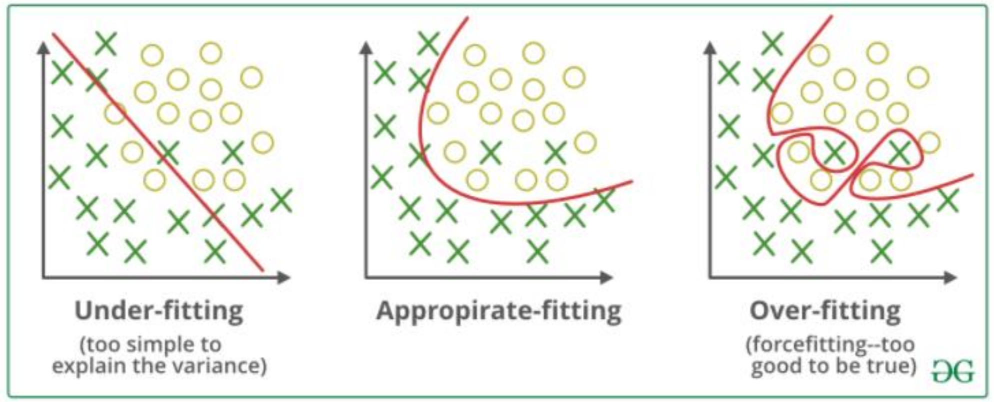
- Specify weight decay when initialize optimizers (recommended)

CLASS `torch.optim.SGD(params, lr=<required parameter>, momentum=0, dampening=0, weight_decay=0, nesterov=False)` [SOURCE]

- **weight_decay** (*float*, *optional*) – weight decay (L2 penalty) (default: 0)

Why regularization reduces overfitting?

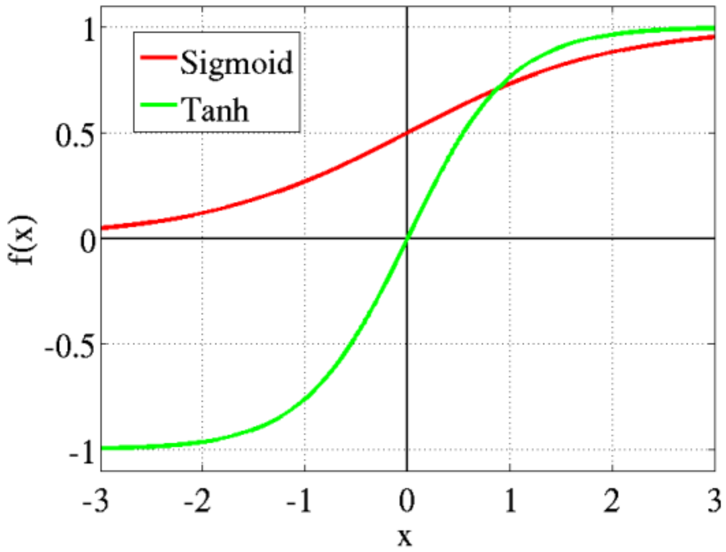




$$J\left(\boldsymbol{W}^{[1]}, \boldsymbol{b}^{[1]}, \boldsymbol{W}^{[2]}, \boldsymbol{b}^{[2]} \ldots \boldsymbol{W}^{[L]}, \boldsymbol{b}^{[L]}\right)$$

$$= \frac{1}{m} \sum_{i=1}^{m} L\left(\hat{y}^{(i)}, y^{(i)}\right) + \frac{\lambda}{2m} \sum_{l=1}^{L} \left\| \boldsymbol{W}^{[l]} \right\|_F$$

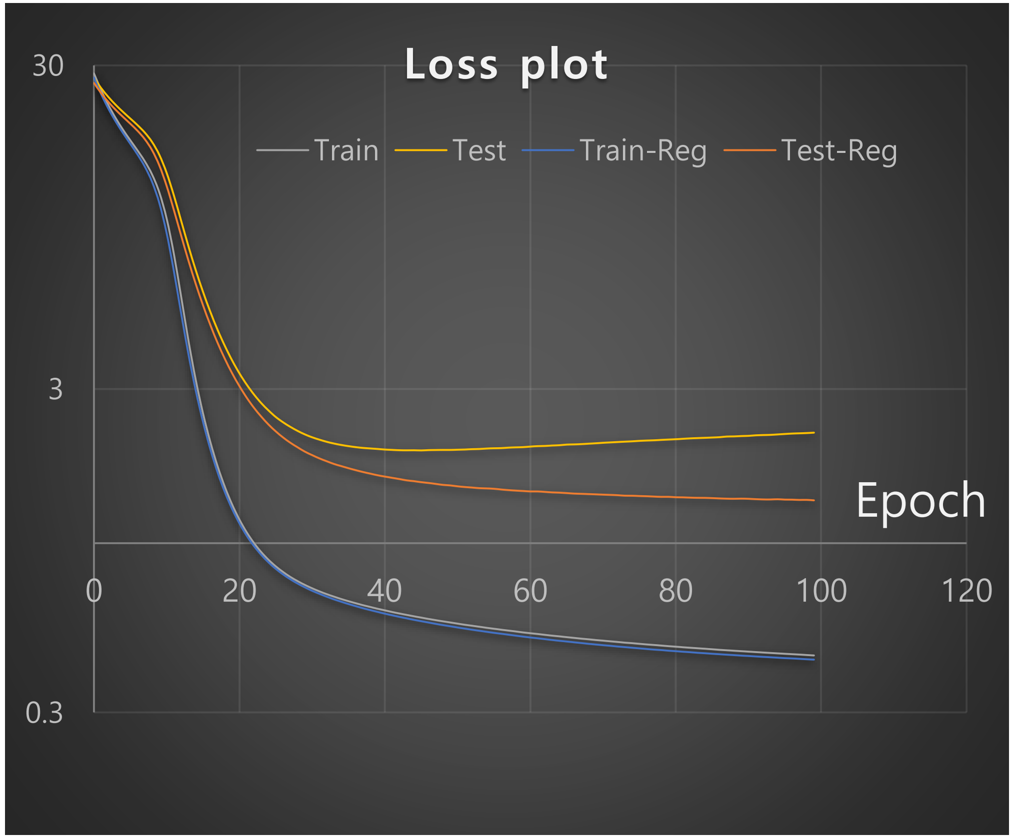
Why regularization reduces overfitting?



$$\boldsymbol{z}^{[i]} = \boldsymbol{W}^{[i]}\boldsymbol{a}^{[i-1]} + \boldsymbol{b}^{[i]}$$

$$\boldsymbol{a}^{[i]} = f\left(\boldsymbol{z}^{[i]}\right)$$

Effect of the regularization



- Matrix factorization
- Settings
  - Adam(lr = 0.01)
  - K = 16
- Regularization
  - Weight decay: 10^{-5}

```
mf_model = MF(n_users, n_items, rank = 16)
optimizer = torch.optim.Adam(mf_model.parameters(), lr= 0.001, weight_decay = 1e-5)
criterion = nn.MSELoss()

result = []
for epoch in range(100):
```