

9 April 2021

Introduction

The code implementation is stored in a Github repository¹. The results are seeded using `torch.manual_seed(0)` to provide reproducible results.

1 Implement a matrix factorisation using gradient descent

1.1 Implement gradient-based factorisation

```
from typing import Tuple
import torch

def sgd_factorise(A: torch.Tensor, rank: int,
                 num_epochs=1000, lr=0.01) ->
    Tuple[torch.Tensor, torch.Tensor]:
    m, n = A.shape
    U = torch.rand((m, rank))
    V = torch.rand((n, rank))

    for epoch in range(num_epochs):
        for r in range(m):
            for c in range(n):
                e = A[r, c] - (U[r] @ V[c].T)
                U[r] += lr * e * V[c]
                V[c] += lr * e * U[r]

    return U, V
```

1.2 Factorise and compute reconstruction error

$$\hat{U} = \begin{bmatrix} 0.6168 & -0.1530 \\ 0.4108 & 1.5961 \\ 1.0798 & 1.1800 \end{bmatrix},$$

$$\hat{V} = \begin{bmatrix} 0.8126 & 1.8290 \\ 0.7836 & -0.2088 \\ 0.8384 & 1.0195 \end{bmatrix},$$

$$\text{Loss} = 0.12197002023458481$$

2 Compare your result to truncated SVD

2.1 Compare to the truncated-SVD

$$U_t = \begin{bmatrix} -0.0801 & -0.7448 & 0.6625 \\ -0.7103 & 0.5090 & 0.4863 \\ -0.6994 & -0.4316 & -0.5697 \end{bmatrix},$$

$$S_t = [5.3339 \quad 0.6959 \quad 0.0000],$$

$$V_t = \begin{bmatrix} -0.8349 & 0.2548 & 0.4879 \\ -0.0851 & -0.9355 & 0.3430 \\ -0.5439 & -0.2448 & -0.8027 \end{bmatrix},$$

$$\text{Loss} = 0.12191088497638702$$

The reconstruction loss of the truncated SVD is almost identical to the results in Section 1.2. This is explained by the Eckart-Young-Mirsky theorem². The Eckart-Young-Mirsky theorem states that the singular value decomposition of a matrix D can be approximated with subject to $\text{rank}(D) \leq r$.

3 Matrix completion

3.1 Implement masked factorisation

```
def sgd_factorise_masked(A: torch.Tensor, M:
                        torch.Tensor, rank: int, num_epochs=1000,
                        lr=0.01) -> Tuple[torch.Tensor,
                        torch.Tensor]:
    m, n = A.shape
    U = torch.rand((m, rank))
    V = torch.rand((n, rank))

    for epoch in range(num_epochs):
        for r in range(m):
            for c in range(n):
                if M[r, c]:
                    e = A[r, c] - (U[r] @
                                    V[c].T)
                    U[r] += lr * e * V[c]
                    V[c] += lr * e * U[r]

    return U, V
```

3.2 Reconstruct a matrix

$$\hat{U} = \begin{bmatrix} 0.6501 & -0.1515 \\ 0.2346 & 1.2954 \\ 1.1472 & 1.2511 \end{bmatrix},$$

$$\hat{V} = \begin{bmatrix} 0.9019 & 1.5197 \\ 0.8868 & -0.1223 \\ 0.5406 & 1.3175 \end{bmatrix},$$

$$\hat{A} = \hat{U}\hat{V}^T = \begin{bmatrix} 0.3561 & 0.5951 & 0.1518 \\ 2.1802 & 0.0496 & 1.8334 \\ 2.9360 & 0.8643 & 2.2685 \end{bmatrix},$$

$$\text{Loss} = 1.4483590126037598$$

Although the matrix completion approximation of \hat{A} is not identical with \hat{A} , the gradient descent-based approach of minimization provides reasonable results in the recovery of the missing values subject to rank r .

¹Eugene Teoh. *COMP6248 Deep Learning*. URL: <https://github.com/eugeneteoh/COMP6248-Deep-Learning>.

²Carl Eckart and Gale Young. "The approximation of one matrix by another of lower rank". In: *Psychometrika* 1.3 (Sept. 1, 1936), pp. 211–218. ISSN: 1860-0980. DOI: 10.1007/BF02288367. URL: <https://doi.org/10.1007/BF02288367> (visited on 04/19/2021).