Deep Learning Report - Lab 1

Chirag Prakash Tubakad(cpt1u21@soton.ac.uk)

I. Task 1 - Matrix factorisation Using Gradient Descent

A. *Gradient based matrix factorisation*

One of the ways to achieve low rank matrix factorisation is by using gradient descent. Given the signature and the parameters that the function accepts, the algorithm was implemented using the pseudo-code in algorithm 1.

Algorithm 1 Matrix factorisation using gradient descent

```
for $epoch = 1, 2, \ldots, N$ do
    for $r = 1, 2, \ldots, m$ do
        for $c = 1, 2, \ldots, n$ do
            e = $A_{rc} - \hat{U}_{r,*}(\hat{V}_{c,*})^T$
            $\hat{U}_{r,*} = \hat{U}_{r,*} + \eta.e.\hat{V}_{c,*}$
            $\hat{V}_{c,*} = \hat{V}_{c,*} + \eta.e.\hat{U}_{r,*}$
        end for
    end for
end for
Return Û,V̂
```

B. *Factorisation and Reconstruction Error*

In this sub task, the rank 2 matrix factorisation had to be computed for a given matrix A = $\left(\begin{smallmatrix} 0.3374 & 0.6005 & 0.1735 \\ 3.3359 & 0.0492 & 1.8374 \\ 2.9407 & 0.5301 & 2.2620 \end{smallmatrix}\right)$ using a learning rate of 0.01 and the number of epochs equal to 1000. The factorised matrices are :

$$\hat{U} = \begin{bmatrix} 0.1611, 0.1563 \\ 1.0847, 1.0529 \\ 1.3816, 1.3410 \end{bmatrix}, \hat{V} = \begin{bmatrix} 0.1260, 0.9404 \\ 0.2848, 0.2934 \\ 1.6537, 1.7038 \end{bmatrix}$$

The reconstruction loss is given by $\|$ A - ÛV̂ $\|^2$.

It basically measures the absolute distance between the original matrix A and the reconstructed matrix ÛV̂. The distance was calculated using PyTorch's inbuilt "torch.nn.functional.mse" loss function with reduction='sum'. The result turned out to be 16.34

II. Task 2 - Comparison of results with truncated SVD

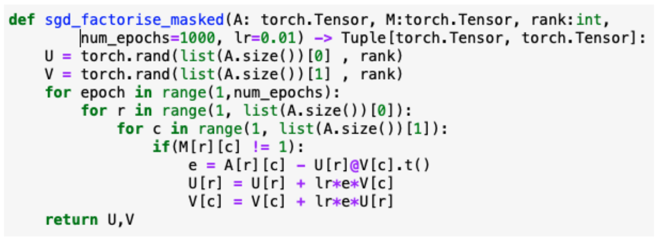
The truncated singular value decomposition is computed as A = $U_t \sigma V_t^T$

It was calculated using PyTorch inbuilt function "torch.svd". The last singular value was set to zero and the reconstruction loss before and after setting the last singular value to 0 was computed.The reconstruction loss before setting last singular value to 0 = 33.971 and the reconstruction loss after setting last singular value to 0 = 33.921

From the results it is evident that our algorithm appears to have performed better in comparison to the randomised SVD implementation provided by PyTorch. The reason for this is because in truncated SVD, it throws out the extra rows/columns which can be expensive if r is much smaller than the rank M. It sacrifices a part of the precision to increase the stability of the solution thereby enhancing its generalisation capability.

III. Task 3 - Matrix Completion

this is one of the neat applications of gradient based approach to matrix factorisation. Data in the real world is not necessarily always complete and in some instances when it is complete it would be noisy. These are characteristics inherent to most real world datasets. Matrix completion in a nutshell helps impute the missing observation and potentially remove some of the noise. The algorithm from the given pseudo-code was implemented as shown in figure 1.



```python
def sgd_factorise_masked(A: torch.Tensor, M:torch.Tensor, rank:int,
        num_epochs=1000, lr=0.01) -> Tuple[torch.Tensor, torch.Tensor]:
    U = torch.rand(list(A.size())[0] , rank)
    V = torch.rand(list(A.size())[1] , rank)
    for epoch in range(1,num_epochs):
        for r in range(1, list(A.size())[0]):
            for c in range(1, list(A.size())[1]):
                if(M[r][c] != 1):
                    e = A[r][c] - U[r]@V[c].t()
                    U[r] = U[r] + lr*e*V[c]
                    V[c] = V[c] + lr*e*U[r]
    return U,V
```

Fig. 1: Implementation of masked factorisation

Given a sample matrix A = $\left(\begin{smallmatrix} 0.3374 & 0.6005 & 0.1735 \\ & 0.0492 & 1.8374 \\ 2.9407 & & 2.2620 \end{smallmatrix}\right)$ the task was perform rank 2 factorisation using the default learning rate and the number of epochs.

The matrix that the algorithm estimated was $\begin{bmatrix} 0.2483 & 0.0167 & 0.6169 \\ 0.7336 & 0.0492 & 1.8374 \\ 0.7350 & 0.0457 & 2.2620 \end{bmatrix}$ This in comparison to the original matrix given had an absolute distance measure of 12.41 which is the least value when compared to all of the above implementations. This just goes to show that the algorithms performs well in estimating missing values and noisy data which is particularly helpful in applications such as recommendation systems.