Lab 2 Exercise – PyTorch AutoGrad

Justin Ugwudike (jknu1g19@soton.ac.uk)

Feb 27, 2023

1.1 Question 1_1

$$\min_{\hat{U},\hat{V}}(\|A - \hat{U}\hat{V}^\top\|_F^2)$$

Figure 1 – The function to optimise.

Using the code shown in Appendix A, the rank-2 factorisation of the matrix is achieved by minimising the function shown in Figure 1. The results are shown in Figure 2.

```
RECONSTRUCTION--------
tensor([[ 0.2193,  0.5037,  0.3697],
        [ 3.2610, -0.0122,  1.9618],
        [ 3.0304,  0.6037,  2.1129]])
MSE LOSS--------------
tensor(0.1224)
```

Figure 2 – The reconstruction of the target matrix using U and V (obtained via gradient descent utilising automatic differentiation). The mean squared error loss is shown.

The code produces and approximation of the target matrix with the MSE loss being 0.1224, this value differs by +0.0004 to the loss achieved using stochastic gradient descent with non-automatic gradients in Lab 1.

1.2 Question 1_2

With the code from Appendix A the rank-2 factorisation of the data from a dataset of 150 instances and 4 features was estimated using gradient descent. The function in Appendix B allowed me to reconstruct the data matrix using Singular Value decomposition (SVD), the losses achieved using both methods is shown in Figure 3.

```
MSE LOSS--------------
tensor(15.2292, dtype=torch.float64)
SVD MSE LOSS----------
tensor(15.2288, dtype=torch.float64)
```

Figure 3 – The reconstruction loss of the data matrix through gradient descent and the rank-2 reconstruction loss through SVD.

The reconstruction loss of the data matrix using the function in Appendix A is 15.2292. The losses achieved when using each method are very close in value (0.0004 difference).

1.3 Question 1_3

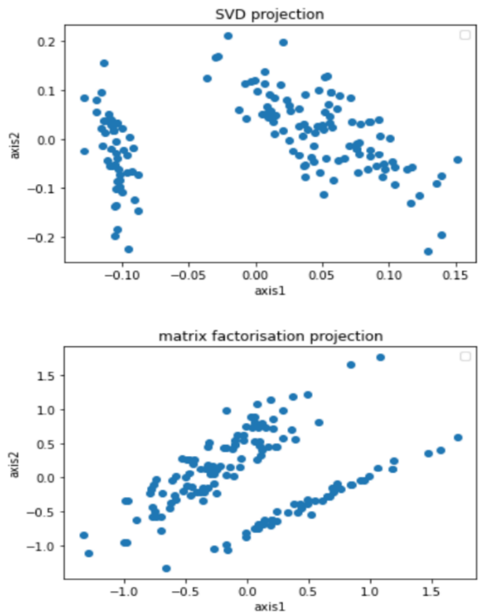

Figure 4 – The first graph represents the data in the U matrix computed by SVD and the second shows the data compute by gradient descent.

By looking a Figure 4 both methods have grouped the data into two classes. The two pairs of classes retain similar shapes and sizes when using gradient descent and SVD however, the orientations of the two pairs of data groups are different.

1.4 Question 2_1

The function used to train an MLP through gradient descent is shown in Appendix C.

1.5 Question 2_2

From Appendix D I can see that either increasing the learning rate and maximum iterations leads to a higher prediction accuracy in both the training and validation data. In every experiment, the accuracy when using the training data is much greater than the accuracy when using the validation data. From these results I can infer that using a iteration count of 1000 and learning rate of 0.01 (to avoid overfitting when using a learning rate of 0.1) would be the best parameters to use in this example as the accuracies for both the training and validation data reach 99% and above.

2 Appendix

Appendix A - Completed gd_factorise_ad() function.

```python
def gd_factorise_ad(A: torch.Tensor, rank : int, num_epochs=1000, lr=0.01):
    U_init = torch.rand(A.shape[0], rank)
    V_init = torch.rand(A.shape[1], rank)
    U = torch.tensor(U_init, requires_grad=True, dtype=torch.float)
    V = torch.tensor(V_init, requires_grad=True, dtype=torch.float)


    for i in range(0, num_epochs):
        # manually dispose of the gradient (in reality it would be better to de
        U.grad=None
        V.grad=None
        # evaluate the function
        J = function(A,U,V)
        # auto-compute the gradients at the previously evaluated point x
        J.backward()
        # compute the update
        U.data = U - U.grad*lr
        V.data = V - V.grad*lr

    return U, V
```

Appendix B – Singular Value Decomposition and reconstruction function.

```python
def SVD_reconstrucion(A):
    SVD = torch.svd(A) #svd
    singular_values = torch.diag(SVD.S) #singular values
    #print("SINGULAR VALUES-------")
    #print(singular_values)
    singular_values[singular_values.shape[0] - 1] = 0 #set last singular value
    singular_values[singular_values.shape[0] - 2] = 0 #set second-to-last singu
    #print("RANK2 SINGULAR VALUES-")
    #print(singular_values)
    reconstruction = SVD.U @ singular_values @ SVD.V.t()
    mse_loss = torch.nn.functional.mse_loss(reconstruction, A, reduction='sum')
    return reconstruction, mse_loss
```

Appendix C – Function to perform MLP gradient descent using automatic differentiation.

```python
def MLP(data_in, targets_in, num_epochs=100, lr=0.01):
    W1_init = torch.rand(4, 12)
    W2_init = torch.rand(12, 3)
    b_init = torch.tensor([0])
    W1 = torch.tensor(W1_init, requires_grad=True, dtype=torch.float)
    W2 = torch.tensor(W2_init, requires_grad=True, dtype=torch.float)
    b1 = torch.tensor(b_init, requires_grad=True, dtype=torch.float)
    b2 = torch.tensor(b_init, requires_grad=True, dtype=torch.float)

    for i in range(0, num_epochs):
        # manually dispose of the gradient (in reality it would be better to de
        W1.grad=None
        W2.grad=None
        b1.grad=None
        b2.grad=None
        # evaluate the function
        logits = torch.relu(data_in @ W1 + b1) @ W2 + b2
        J = torch.nn.functional.cross_entropy(logits, targets_in)
        # auto-compute the gradients at the previously evaluated point x
        J.backward()
        # compute the update
        W1.data = W1 - W1.grad*lr
        W2.data = W2 - W2.grad*lr
        b1.data = b1 - b1.grad*lr
        b2.data = b2 - b2.grad*lr

    return W1,W2,b1,b2,J
```

Appendix D – Three sets of accuracies recorded from an MLP when using different learning rates and maximum iterations.

ACCURACY--------------

LR=0.1	ITER=100
training	99.0%
validation	100.0%
LR=0.01	ITER=100
training	69.0%
validation	100.0%
LR=0.001	ITER=100
training	51.0%
validation	72.0%
LR=0.1	ITER=1000
training	99.0%
validation	100.0%
LR=0.01	ITER=1000
training	99.0%
validation	100.0%
LR=0.001	ITER=1000
training	94.0%
validation	98.0%

ACCURACY--------------

LR=0.1	ITER=100
training	99.0%
validation	100.0%
LR=0.01	ITER=100
training	79.0%
validation	100.0%
LR=0.001	ITER=100
training	41.0%
validation	98.0%
LR=0.1	ITER=1000
training	100.0%
validation	100.0%
LR=0.01	ITER=1000
training	99.0%
validation	100.0%
LR=0.001	ITER=1000
training	80.0%
validation	100.0%

ACCURACY--------------

LR=0.1	ITER=100
training	99.0%
validation	100.0%
LR=0.01	ITER=100
training	79.0%
validation	100.0%
LR=0.001	ITER=100
training	48.0%
validation	72.0%
LR=0.1	ITER=1000
training	100.0%
validation	100.0%
LR=0.01	ITER=1000
training	99.0%
validation	100.0%
LR=0.001	ITER=1000
training	84.0%
validation	98.0%