Lab 2 Exercise - PyTorch Autograd

Amishapriya Singh
Student id: 33333904
as6u21@soton.ac.uk

1. Implement a matrix factorisation using gradient descent

1.1. Implement gradient-based factorisation using PyTorch's AD

```
def gd_factorise_ad(A: torch.Tensor, rank: int, epochs = 1000, lr = 0.01):
  [m, n] = A.shape
  U = torch.rand((m, rank), dtype=float).clone().detach().requires_grad_(True)
  V = torch.rand((rank, n), dtype=float).clone().detach().requires_grad_(True)
  for epoch in range(epochs):
    U.grad, V.grad = None, None
    J = torch.nn.functional.mse_loss(U@V, A, size_average=None, reduce=None, reduction='sum')
    J.backward()
    U.data = U - lr * U.grad
    V.data = V - lr * V.grad
  return U, V
```

1.2. Factorise and compute reconstruction error on real data

Reconstruction loss (squared L2 norm) for gradient-based factorisation = 15.23 whilst that for truncated SVD = 15.23.

1.3. Compare against PCA

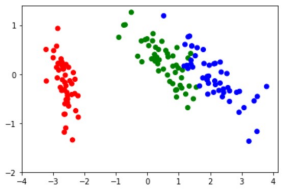

Figure 1. PCA projection

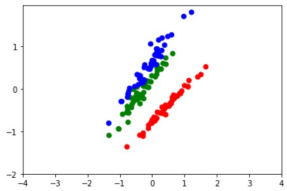

Figure 2. Matrix factorisation projection

The direction of projection in the two cases is different. While SVD projects data along principal components, matrix factorisation projects the data with right rotation and scaling ($U' = U\Sigma^{1/2}$).

2. A Simple MLP

2.1. Implement the MLP

```
def mlp (data, targets, lr = 0.01, epochs = 100):
  W1 = torch.rand((4, 12), dtype=torch.float).clone().detach().requires_grad_(True)
  W2 = torch.rand((12, 3), dtype=torch.float).clone().detach().requires_grad_(True)
  b1 = torch.tensor(0, requires_grad=True, dtype=torch.float)
  b2 = torch.tensor(0, requires_grad=True, dtype=torch.float)
  for epoch in range(epochs):
    W1.grad, W2.grad, b1.grad, b2.grad = None, None, None, None
    logits = torch.relu(data @ W1 + b1) @ W2 + b2
    J = loss_function(logits, targets)
    J.backward()
    W1.data = W1 - lr * W1.grad
    W2.data = W2 - lr * W2.grad
    b1.data = b1 - lr * b1.grad
    b2.data = b2 - lr * b2.grad
  return W1, W2, b1, b2
```

2.2. Test the MLP

If we run the MLP on training data multiple times, we see that in several such iterations, the accuracy for training and test dips to the range of 70-80 indicating the sensitivity of the learning to weight initialisation. It seems that every so often, the weights are poorly initialised leading to the model getting stuck in a local minimum.