1 Gradient Descent MLP

```python
def gd_factorise_ad(A, rank, num_epochs=1000, lr=0.01):
    m, n = A.shape # m x n matrix
    Uh = torch.rand(m, rank, requires_grad=True) # m x r matrix
    Vh = torch.rand(n, rank, requires_grad=True) # n x r matrix

    for epoch in range(num_epochs):
        input = A
        pred = Uh @ Vh.T
        # loss
        loss = torch.nn.functional.mse_loss(input, pred, reduction='sum')
        # backward pass
        loss.backward()
        # gradient descent
        with torch.no_grad():
            Uh.data = Uh.data - lr * Uh.grad.data
            Vh.data = Vh.data - lr * Vh.grad.data

            Uh.grad.data.zero_()
            Vh.grad.data.zero_()

    return Uh, Vh
```

(a) Gradient Factorisation

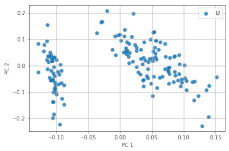
```python
for epoch in tqdm(range(epochs)):
    # first hidden layer
    h1_linear = X_train @ W1 + b1
    h1 = torch.relu(h1_linear)
    # output layer
    h2_linear = h1 @ W2 + b2
    output = torch.relu(h2_linear)
    # loss
    loss = torch.nn.functional.cross_entropy(output, y_train) # softmax_cross_entropy
    # backprop
    loss.backward()
    # prevent pytorch from creating computational graph
    with torch.no_grad():
        # update weights using gradient descent
        W1.data = W1.data - lr * W1.grad.data
        W2.data = W2.data - lr * W2.grad.data
        # manually zero gradients after backward pass
        W1.grad.zero_()
        W2.grad.zero_()
    # track loss over epochs
    losses.append(loss.item())
    # track train accuracy over epochs
    pred = mlp_predict(X_train, W1, W2, b1, b2)
    acc = eval_accuracy(pred, y_train)
    train_accuracy.append(acc)
    pred_v = mlp_predict(X_test, W1, W2, b1, b2)
    acc_v = eval_accuracy(pred_v, y_test)
    valid_accuracy.append(acc_v)
```

(b) MLP

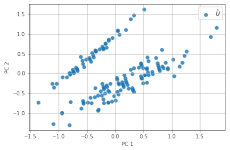
Figure 1: Code snippets (zoom to read).

With learning late = 0.01 and 2000 epochs, reconstruction loss = 0.0254. Performing truncated SVD on the same data gives reconstruction loss = 1.0201.

SVD is a numerical method of computing the PCA. The reconstruction error is If mean-centred datapoint is $\boldsymbol{X}$ and its approximations are $\boldsymbol{X}$ then reconstruction error is $||\boldsymbol{X} - \tilde{\boldsymbol{X}}||_F^2$. With Eckhart-Young Theorem, we know that an optimal choice of $\tilde{\boldsymbol{X}}$ is $\boldsymbol{X}_k$ found via truncated SVD.



(a) From $\boldsymbol{U}$ using truncated SVD



(b) From $\hat{\boldsymbol{U}}$ using factorisation

Figure 2: Projection of data onto principle directions.

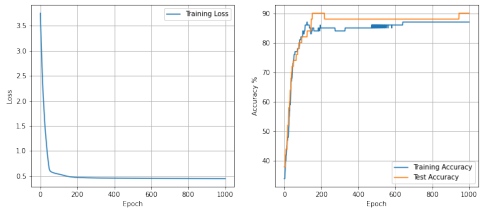


Figure 3: Left plot showing training loss. Right plot showing training accuracy in blue and validation accuracy in orange.