**Akhil Chilumuru**
**UIN: 633002192**
**Nov 2, 2022**

**CSCE 636: Deep Learning**
**Assignment 3**

# 1 Answers to the non-programming part

1. **Given a symmetric matrix $A \in R^{3X3}$ and its Eigen decomposition, find its singular value decomposition.**
As the given matrix **A** is symmetric, its eigen vectors are orthogonal. This implies we just have to transform the eigen values matrix into a singluar value matrix. The singular values must be non-zero and must be in decreasing order. Let's transform the given equation using this objective:

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ u_{21} & u_{22} & u_{23} \\ u_{31} & u_{32} & u_{33} \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \\
&= \begin{bmatrix} u_{11} & u_{12} & -u_{13} \\ u_{21} & u_{22} & -u_{23} \\ u_{31} & u_{32} & -u_{33} \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{bmatrix} \\
&= \begin{bmatrix} u_{11} & -u_{13} & u_{12} \\ u_{21} & -u_{23} & u_{22} \\ u_{31} & -u_{33} & u_{32} \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{21} & u_{31} \\ u_{12} & u_{22} & u_{32} \\ u_{13} & u_{23} & u_{33} \end{bmatrix}
\end{aligned}$$

This is the SVD for the given matrix **A**

2. **Provide a complete proof of the Ky Fan Theorem.**
Since H is a symmetric matrix, its eigen decomposition would look like $H = U\Sigma U^T$. Substituting this in RHS of the equation would result in,

$$A^T H A = A^T U \Sigma U^T A = (U^T A)^T \Sigma (U^T A)$$

Assume $B = U^T A$, then

$$B^T B = (U^T A)^T (U^T A) = A^T U U^T A$$

Because $U$ is orthogonal matrix $U^T U = UU^T = I_n$, we also know that $A^T A = I_k$, then the above equation becomes,

$$\begin{aligned}
B^T B &= A^T (UU^T) A \\
&= A^T A \\
&= I
\end{aligned} \tag{1}$$

Which means, B is a matrix with orthonormal columns. The trace can be written as,

$$trace(A^T H A) = trace(B^T \Sigma B) = trace(\Sigma B B^T) = \sum_{i=1}^{n} \lambda_i ||b_i||^2$$

where $\lambda_i$ is the $i^{th}$ eigenvalue of $H$ and $b_i$ is the $i^{th}$ row of B.
$b_i$ can also be written as $i^{th}$ column of $B^T$. Meaning, $b_i = (u_i^T A)^T = A^T u_i$. Then the above trace becomes:

$$trace(A^T H A) = \sum_{i=1}^{n} \lambda_i ||A^T u_i||^2 \tag{2}$$

We know that $A^TA = I_k$ and $[a_1, a_2...a_k]$ are orthonormal vectors. Another way to look at this is $A$ is first $k$ columns orthogonal matrix of size n x n.

That is, consider $A^c = [a_{k+1}, a_{k+2}, ..., a_k]$ to be another set of orthonormal vectors such that, $[A, A^c]$ is an orthogonal matrix. Then,

$$\begin{aligned} ||A^Tu_i||^2 &\leq ||A^Tu_i||^2 + ||(A^c)^Tu_i||^2 \\ &\leq |||[A, A^c]^Tu_i||^2 \\ &\leq u_i^T[A, A^c][A, A^c]^Tu_i \\ &\leq u_i^Tu_i \text{ (since, } [A, A^c] \text{ is orthorgonal)} \\ &\leq 1 \end{aligned} \tag{3}$$

Given that we already know $|||[A, A^c]^Tu_i|| = 1$, the above expression becomes equal when $||A^Tu_i|| = 1$. This means, $(A^c)^Tu_i = 0$. Therefore, $u_i$ should be in the orthogonal complement space of linear space formed by $[a_{k+1}, a_{k+2}, ..., a_k]$ vectors.

In other words, $u_i$ should be in the column space of $A$. Which means, $u_i$ could be written as a linear combination of columns of $A$. Therefore, $u_i$ could be written as

$$u_i = Aq_i \text{ where } q_i \text{ is a unit vector} \in \mathbb{R} \tag{4}$$

However, we did not yet establish for which $i$ this is true.

Summation over all rows $i$ will be,

$$\begin{aligned} \sum_{i=1}^{n} ||A^Tu_i||^2 &= \sum_{i=1}^{n} u_i^TAA^Tu_i \\ &= trace((U^TA)(A^TU)) \\ &= trace((A^TU)(U^TA)) \\ &= trace(A^TUU^TA) = trace(A^TA) \\ &= trace(I_k) \\ &= k \end{aligned}$$

Now,

$$\begin{aligned} trace(A^THA) &= \sum_{i=1}^{n} \lambda_i||A^Tu_i||^2 \\ &\leq \sum_{i=1}^{k} \lambda_i||A^Tu_i||^2 + \sum_{i=k+1}^{n} \lambda_i||A^Tu_i||^2 \\ &\leq \sum_{i=1}^{k} \lambda_i||A^Tu_i||^2 + \lambda_{k+1} \sum_{i=k+1}^{n} ||A^Tu_i||^2 \\ &\leq \sum_{i=1}^{k} \lambda_i||A^Tu_i||^2 + \lambda_{k+1}(\sum_{i=1}^{n} ||A^Tu_i||^2 - \sum_{i=1}^{k} ||A^Tu_i||^2) \\ &\leq \sum_{i=1}^{k} \lambda_i||A^Tu_i||^2 + \lambda_{k+1}(k - \sum_{i=1}^{k} ||A^Tu_i||^2) \\ &\leq \sum_{i=1}^{k} (\lambda_i - \lambda_{k+1})||A^Tu_i||^2 + k\lambda_{k+1} \end{aligned}$$

$$\leq \sum_{i=1}^{k}(\lambda_i - \lambda_{k+1}) + k\lambda_{k+1}$$
$$\leq \sum_{i=1}^{k}\lambda_i$$

Therefore we can say $||A^T u_i|| = 1$ for $1 \leq i \leq k$ and $||A^T u_i|| = 0$ for $i \geq k$

Finally, from the upper bound derived above we can claim that $max(trace(A^T HA)) = \sum_{i=1}^{k} \lambda_i$. Hence, part 1 is proved.

We also have $u_i = Aq_i$ for $1 \leq i \leq k$. This can be also written in compact form as $U = AQ^T$ where $U = [u_1, u_2, ..., u_k]$ and $Q^T = [q_1, q_2, ...q_k]$. Note that transpose is used for notation convenience.

Now, if $Q^T$ is orthogonal matrix then,

$$U = AQ^T$$
$$UQ = AQ^T Q$$
$$UQ = A$$

Therefore, the optimal $A^*$ is given by $A^* = [u_1, u_2..., u_k]Q$ where $Q$ is an orthogonal matrix. Hence part 2 is also proved.

5. **Bonus question**

Let $K_1$ be the kernel matrix $K(i,j)$ $i = 1, 2, .., n$ and $j = 1, 2, .., n$ i.e., $K_1$ is the kernel for training data. Therefore, $K_1$ can be written as:

$$K_1 = X_{tr} X_{tr}^T$$

Since, $K_1$ is a valid kernel matrix and positive semi definite, it's compact SVD can be written as,

$$\begin{aligned} K_1 &= U\Sigma U^T \\ &= U\Sigma^{\frac{1}{2}}\Sigma^{\frac{1}{2}}U^T \\ &= (U\Sigma^{\frac{1}{2}})(U\Sigma^{\frac{1}{2}})^T \\ &= (U\Sigma^{\frac{1}{2}})RR^T(U\Sigma^{\frac{1}{2}})^T; (R \text{ is any orthogonal matrix } i.e., RR^T = R^T R = I) \\ &= (U\Sigma^{\frac{1}{2}}R)(U\Sigma^{\frac{1}{2}}R)^T \end{aligned} \tag{5}$$

Which is analogous to $K_1 = X_{tr} X_{tr}^T$.
It implies $X_{tr} = U\Sigma^{\frac{1}{2}}R$.
Let $X_{tr}^{'} = X_{tr}R^T$ be an orthogonal transformation on the original training data $X_{tr}$.
Which means $X_{tr}^{'} = U\Sigma^{\frac{1}{2}}RR^T = U\Sigma^{\frac{1}{2}}$ is the new training data after transformation.

We can use this transformed training data, and the kernel will still be same. Even the logistic regression boundary will not be faulted because orthogonal transformation does not change the relative distribution of the training data. Perhaps the weight vectors will also be orthogonally transformed but the magnitude or norm of the weight vectors will not change. This is a well known property of orthogonal matrices:

Let $w$ be a weight vector and $R$ be the orthogonal transformation, then $||Rw|| = ||w||$

This means that even the L2 regularization will remain the same. Overall, logistic regression results will be equivalent even after orthogonal transformation. Therefore, we can use the transformed data $X_{tr}^{'}$ in place of our original training data $X_{tr}$.

Now, let $K_2$ be the kernel matrix $K(i,j)$ $i = 1, 2, .., n$ and $j = n+1, n+2, ..., n+m$ i.e., $K_2$ is the kernel for test data. Therefore, $K_2$ can be written as:

$$\begin{aligned}
K_2 &= X_{tr} X_{test}^T \\
&= X_{tr} R^T R X_{test}^T \\
&= (X_{tr} R^T)(X_{test} R^T)^T \\
&= X_{tr}^{'} (X_{test}^{'})^T \text{ (where } X_{test}^{'} \text{ is the test data after the same orthogonal transformation as train data)} \\
&= U \Sigma^{\frac{1}{2}} (X_{test}^{'})^T \\
U^T K_2 &= U^T U \Sigma^{\frac{1}{2}} (X_{test}^{'})^T = \Sigma^{\frac{1}{2}} (X_{test}^{'})^T \\
(\Sigma^{\frac{1}{2}})^{-1} U^T K_2 &= (\Sigma^{\frac{1}{2}})^{-1} \Sigma^{\frac{1}{2}} (X_{test}^{'})^T \\
\Sigma^{\frac{-1}{2}} U^T K_2 &= (X_{test}^{'})^T
\end{aligned} \tag{6}$$

Therefore, $X_{test}^{'} = K_2^T U (\Sigma^{\frac{-1}{2}})^T$ is the test data after the orthogonal transformation.

In summary $X_{tr}^{'} = U \Sigma^{\frac{1}{2}}$ and $X_{test}^{'} = K_2^T U (\Sigma^{\frac{-1}{2}})^T$ are the transformed training and test data respectively and can be used in the solver for primal form of logistic regression.

# 2 Results and Analysis of the programming part

3. **Kernel vs Neural Network**

   (a) Kernel Logistic Regression
   Hyperparameters: learning_rate=0.01, batch_size=128, max_epochs=50, hidden_dim=$number_of_training_samples$.
   Different values for sigma were explored and the results were as below:

| $\sigma$ | validation accuracy |
|---|---|
| 0.1 | 57.40% |
| 0.5 | 61.90% |
| 1.0 | 99.73% |
| 3.0 | 100% |
| 10.0 | 99.735% |
| 30.0 | 99.47% |

   We can notice that any sigma value above 1.0 achieves almost 100% accuracy. Based on above scores, I took sigma=3.0 for the final value and the resulting test accuracy is 99.13%.

   (b) RBF kernel logistic regression using centroids
   In RBF kernel logistic regression, the kernel is computed using representatives (also called as centroids) of the training samples. These representatives are selected using a clustering algorithm called K-means. Hyperparameters: learning_rate=0.01, batch_size=128, max_epochs=50. Different values for hidden_dim (number of centroids) and sigma were explored and the results were as below:

| hidden_dim | $\sigma$ | validation accuracy |
|---|---|---|
| 32 | 0.1 | 56.87% |
| 32 | 0.5 | 57.14% |
| 32 | 1.0 | 83.06% |
| 32 | 3.0 | 100% |
| 32 | 10.0 | 100% |
| 32 | 30.0 | 94.97% |
| 64 | 0.1 | 52.90% |
| 64 | 0.5 | 58.46% |
| 64 | 1.0 | 81.48% |
| 64 | 3.0 | 100% |
| 64 | 10.0 | 99.73% |
| 64 | 30.0 | 98.94% |
| 128 | 0.1 | 51.32% |
| 128 | 0.5 | 59.25% |
| 128 | 1.0 | 89.15% |
| 128 | 3.0 | 100% |
| 128 | 10.0 | 100% |
| 128 | 30.0 | 99.20% |

From these numbers, we can infer that the hidden_dim size has very little effect on the validation accuarcy. Once again, sigma=3.0 value gave the best accuracy. For the final model, I took hidden_dim=64 and sigma=3.0 and the resulting test accuracy is 98.91%.

(c) Feed forward Network
Hyperparameters: learning_rate=0.01, batch_size=128, max_epochs=50. Different values for hidden_dim were explored and the results were as below:

| hidden_dim | validation accuracy |
|---|---|
| 32 | 99.73% |
| 64 | 100% |
| 128 | 99.73% |

The final model is with 64 hidden dimension, and the resulting test accuracy was 98.70%.

In summary, the performance of the best model from each of the three architectures is:

| Model | Test accuracy |
|---|---|
| Kernel Logistic Regression | 99.13% |
| RBF Kernel Logistic Regression | 98.91% |
| Feed Forward Network | 98.70% |

RBF kernel is almost as good as Kernel Logistic Regression despite having very few number of hidden representations (64 vs 1328 to be precise). Feed forward network (FFN) also performs similar to the RBF kernel model with the same number of hidden representations (64). However, the difference in number of parameters is significant. FFN has about 16384 (input_dim*hidden_dim = 256*64) more parameters than RBF kernel model. The tradeoff is that in RBF kernel model, we had to perform k-means clustering to find the centroids. Despite having fewer parameters than FFN, RBF performs very well achieving 98.91% test accuracy.

4. **PCA vs Autoencoder**

(b) **In the class PCA(), complete the reconstruction() function to perform data reconstruction. Please evaluate your code by testing different numbers of the principal component that p = 32, 64, 128.**
The results were as follows:

| Number of principal components | Reconstruction Error |
|---|---|
| 32 | 134.91 |
| 64 | 87.07 |
| 128 | 46.16 |
| 256 | 6.6e-13 |

(d) **In the class AE(), complete the reconstruction() function to perform data reconstruction. Please test your function using three different dimensions for the hidden representation d that d = 32, 64, 128.**

In this variant of the Autoencoder, we use shared weights between encoder and decoder i.e, decoder weights is the transpose of the encoder weights. The results were as follows:

| hidden_dim (or) d | Reconstruction Error |
|---|---|
| 32 | 129.83 |
| 64 | 86.23 |
| 128 | 46.44 |
| 256 | 3.17 |

(e) **Compare the reconstruction errors from PCA and AE.**

In summary, these are the reconstruction erros when we set p=d and compare both PCA and AE:

| p (or) d | PCA error | AE error |
|---|---|---|
| 32 | 134.91 | 129.83 |
| 64 | 87.07 | 86.23 |
| 128 | 46.16 | 46.44 |
| 256 | 6.6e-13 | 3.17 |

From the above results, we can observe that when p is low (32,64), AE has marginally lower reconstruction error than PCA. When p is high, PCA error is marginally low. If we ignore the minor differences in the errors, overall in conclusion, when the weights are shared between encoder and decoder, AE is equivalent to PCA.

(f) **Experimentally justify the relations between the projection matrix G in PCA and the optimized weight matrix W in AE. Note that you need to set p = d for valid comparisons. Please explore three different cases that p = d = 32, 64, 128.**

| $p = d$ | $\|G - W\|_F$ |
|---|---|
| 32 | 7.86 |
| 64 | 11.22 |
| 128 | 15.98 |
| 256 | 22.58 |

We can deduce that G and W are not same. This is expected because PCA aims to achieve minimal reconstruction error through sigular value decomposition which is not unique for a given matrix. Whereas AE aims to achieve minimal reconstruction error through gradient descent. Therefore, G and W may not be the same.

However, we know that solutions of PCA are related by an orthogonal transformation. And one of them must be W. Let us assume $W = GR_{orth}$ where $R_{orth}$ is some orthogonal transformation.

First let's verify that both are orthonormal matrices. For that we verify $||G^TG - W^TW|| \approx 0$

| $p = d$ | $\|\|G^TG - W^TW\|\|_F$ |
|---|---|
| 32 | 0.0118 |
| 64 | 0.0177 |
| 128 | 0.0302 |
| 256 | 0.8965 |

As we can see, all are close to zero. Which means, both $G$ and $W$ are indeed orthonormal matrices. Now, let's verify the transformation $W = GR_{orth}$. If this were true, then

$$\begin{aligned} G^T W &= G^T G R_{orth} \\ &= R_{orth} \end{aligned} \tag{7}$$

Now as $R_{orth}$ is an orthogonal matrix, $R_{orth}R_{orth}^T = R_{orth}^T R_{orth} = I$ must be true. Let's verify that empirically:

| $p = d$ | $\|\|R_{orth}R_{orth}^T - I\|\|_F$ | $\|\|R_{orth}^T R_{orth} - I\|\|_F$ |
|---|---|---|
| 32 | 0.1014 | 0.1014 |
| 64 | 0.0307 | 0.0307 |
| 128 | 0.0936 | 0.0936 |
| 256 | 0.3753 | 0.3753 |

This proves that $R_{orth}$ is indeed orthogonal matrix, which means our original assumption i.e., $W = GR_{orth}$ must also be true. Therefore, the transformation that takes from $G$ to $W$ is:

$$W = GR_{orth}; \text{ where } R_{orth} = G^T W$$

(g) **Please modify the network() and forward() function so that the weights are not shared between the encoder and the decoder. Report the reconstructions errors for d = 32, 64, 128. Please compare with the sharing weights case and briefly analyze you results.**

| d | error with shared weights | error without shared weights |
|---|---|---|
| 32 | 129.83 | 130.23 |
| 64 | 86.23 | 86.40 |
| 128 | 46.44 | 46.81 |
| 256 | 3.17 | 4.85 |

From the above results, we can observe that both then networks perform almost the same. AE with shared weights may be just a special case of AE without shared weights. Perhaps the gradient descent pushed the decoder weights to be close to the transpose of encoder weights in order to achieve minimal reconstruction error.

(h) **Please modify the network() and forward() function to include more network layers and nonlinear functions. Please set d = 64 and explore different hyper- parameters. Report the hyper parameters of the best model and its reconstruction error. Please analyze and report your conclusions.**

After setting d=64, learning rate =0.001, max_epochs = 300 and tuning rest of the hyper parameters, following were the results:

| Network | Batch Size | Reconstruction Error |
|---|---|---|
| [256, 128, 64, 128, 256] w. ReLU | 32 | 77.84 |
| [256, 128, 64, 128, 256] w. ReLU | 64 | 73.73 |
| [256, 128, 64, 128, 256] w. ReLU | 128 | 72.45 |
| [256, 128, 64, 128, 256] w. tanh | 32 | 76.28 |
| [256, 128, 64, 128, 256] w. tanh | 64 | 71.56 |
| [256, 128, 64, 128, 256] w. tanh | 128 | 69.10 |

As we can notice, AE with more layers and non-linear activation gives lower reconstruction error than both PCA and AE with single layer. This implies that with non-linearity, the autoencoder can learn to make a more accurate reconstruction.