ETH zürich

High Performance Computing for Science and Engineering I

P. Koumoutsakos
S. Martin
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2021

Set 3 - Principal Component Analysis and Sanger's rule

Issued: October 29, 2021
Hand in (optional): November 12, 2021 12:00am

Question 1: Principal Component Analysis with Sanger's Rule (25 points)

In this question we want to investigate methods for computing the Eigen values and vectors of a dataset's co-variance matrix:

$$\mathbf{C} := \frac{1}{N-1}\mathbf{X}^T\mathbf{X}, \text{ with } \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \dots \\ \mathbf{x}_N^T \end{bmatrix} \in \mathbb{R}^{N\times n} \tag{1}$$

All methods are based on the Perceptron, but use different rules for updating the weights. Importantly, the updates strongly depend on the output from the forward-pass of the Perceptron, as defined in equation (2):

$$\mathbf{O} = \mathbf{X}\mathbf{W} \text{ , with } \mathbf{W} \in \mathbb{R}^{n\times k} \tag{2}$$

Here, the weight matrix, $\mathbf{W}$, stores the k weight vectors as column vectors. We want to define our update procedure in such a manner that the i-th column vector, $\mathbf{w}_i$, converges to the i-th Eigen vector with unit length of the co-variance matrix.

a) **[5 points]** Implement the forward pass of the Perceptron in the file *perceptron.cpp*. Use a single call to BLAS in your implementation.
Points are awarded accordingly:
- Correct method gives (**1 point**).
- Operating on the correct matrices and vectors gives (**1 point**).
- Using correct indexing gives (**1 point**).
- Passing the unit test gives (**1 point**).
- Using a single function call as specified in the description gives (**1 point**).

b) **[20 points]** In order to find Eigen values and vectors, we need to consider the evolution of our weights. We define $\mathbf{O}^t$ to be the output matrix from step t.

$$\mathbf{O}^t = \left[\mathbf{o}_1^t \dots \mathbf{o}_n^t\right] \text{ , such that } \mathbf{o}_i^t = \mathbf{X}\mathbf{w}_i^t \tag{3}$$

As indicated, we see that our weights evolve over time and $\mathbf{w}_i^t$ is the i-th weight vector at step t. We then need to define the gradient update rules, which is done in the following manner:

1. Hebb's Rule:

$$\nabla \mathbf{w}_i^t \propto \mathbf{X}^T \mathbf{o}_i^t \tag{4}$$

2. Oja's Rule:

$$\begin{aligned}\nabla \mathbf{w}_i^t &\propto \mathbf{X}^T \mathbf{o}_i^t - \left(\left(\mathbf{o}_i^t\right)^T \mathbf{o}_i^t \right) \mathbf{w}_i \\ &= \mathbf{X}^T \mathbf{o}_i^t - \left\|\mathbf{o}_i^t\right\|^2 \mathbf{w}_i^t\end{aligned} \tag{5}$$

3. Sangers's Rule:

$$\begin{aligned}\nabla \mathbf{w}_i^t &\propto \mathbf{X}^T \mathbf{o}_i^t - \sum_{j=1}^{i} \left(\left(\mathbf{o}_i^t\right)^T \mathbf{o}_j^t \right) \mathbf{w}_j^t \\ &= \mathbf{X}^T \mathbf{o}_i^t - \begin{bmatrix} \mathbf{w}_1 \dots \mathbf{w}_i \end{bmatrix} \begin{bmatrix} \mathbf{o}_1^T \\ \dots \\ \mathbf{o}_i^T \end{bmatrix} \mathbf{o}_i\end{aligned} \tag{6}$$

We only specify the gradient as being proportional to the given formulas. The reason is that the length of our update is equal to a specified learning rate, η. The update step is then given by equation (7):

$$\mathbf{w}_i^{t+1} = \mathbf{w}_i^t + \eta \cdot \frac{\nabla \mathbf{w}_i^t}{\|\nabla \mathbf{w}_i^t\|} \tag{7}$$

The Hebb and Oja rules have already been defined in *hebb.cpp* and *oja.cpp* respectively and can be used for inspiration. Your task is to implement Sanger's update rule in *sanger.cpp* as given by equation (6). Also in this task, each TO DO corresponds to a single function call; either to BLAS or another class method.

For each of the four TO DOs, the points are awarded accordingly:

- Correct method gives (**1 point**).
- Operating on the correct matrices and vectors gives (**1 point**).
- Using correct indexing gives (**1 point**).
- Passing the unit test gives (**1 point**).
- Using a single function call as specified in the description gives (**1 point**).

Hint: In order to test your implementation of each TO DO individually, you can copy the master solution for the other TO DOs and then run the unit test.

Hint: You can test your implementation using the unit test - just run "make unit_test". If you pass all tests, you implementation is very likely to be correct. Also, in order to investigate the different methods closer, you can create plots by running "make plots". The plots display the evolution of the Eigen values and the final Eigen vectors. They can therefore both be used for debugging and gaining more insight into the different methods.

Hint: Reference for BLAS and C-BLAS.

Question 2: Linear Auto Encoders (25 points)

This question will cover the fundamental theory of linear neural networks and more specifically linear auto-encoders.

A linear neural network is, just as any conventional feed-forward neural network, a concatenation of structured mappings between layers; in each layer, we first apply an affine transformation before employing an activation function to each individual output. In contrast to other neural networks, in a linear neural network we can combine the two steps into a single affine transformation. Hence, we can write a general linear neural network as given in equation (8) with the definition of a linear layer as given in equation (9).

$$f : \mathbb{R}^n \longrightarrow \mathbb{R}^m, \mathbf{x} \mapsto \left(\phi^{m,h_L}_{\mathbf{W}_{L+1},\mathbf{b}_{L+1}} \circ \ldots \circ \phi^{h_2,h_1}_{\mathbf{W}_2,\mathbf{b}_2} \circ \phi^{h_1,n}_{\mathbf{W}_1,\mathbf{b}_1} \right)(\mathbf{x}) \tag{8}$$

$$\phi^{k,l}_{W,b} : \mathbb{R}^l \longrightarrow \mathbb{R}^k, \mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b}, \text{ with } \mathbf{W} \in \mathbb{R}^{k\times l},\ \mathbf{b} \in \mathbb{R}^k \tag{9}$$

a) **[10 points]** Prove that any linear neural network is an affine mapping.

Hint: Prove by induction on the number of layers that for any linear neural network $f : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, there exist $\mathbf{W} \in \mathbb{R}^{m\times n}$, $\mathbf{b} \in \mathbb{R}^m$ such that $\forall \mathbf{x} \in \mathbb{R}^n : f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$.

In this sub-question we prove that any linear neural network is an affine transformation. As given in the hint, we can write any affine mapping as $\mathbf{x} \mapsto \mathbf{W}\mathbf{x} + \mathbf{b}$ for some matrix $\mathbf{W}$ and vector $\mathbf{b}$. We prove that any linear neural network can be written in this form by means of induction with respect to the number of layers in the network.

1. The first step is define our predicate, as done in (10):

$$P(k) = 1 :\Longleftrightarrow \text{All linear neural networks with k layers are affine transformations} \tag{10}$$

   Correctly defining a predicate gives (**2 point**).

2. The second step is to prove our base case. One options is to define our base case as $k = 0$ giving us the the identity mapping, which is affine with $\mathbf{W} = \mathbf{I}$ and $\mathbf{b} = \mathbf{0}$. The other option is using $k = 1$ as base case. Here we only have one layer which is affine by definition as given in equation (9).

   We accept both base cases and correctly proving the base case gives (**2 point**).

3. The third step is to assume $P(k) = 1$ holds for some $k \in \mathbb{N}$.

   This step gives (**1 point**).

4. The fourth step is to prove $P(k) = 1 \longrightarrow P(k + 1) = 1$.
   We first assume that f_{k+1} is a linear neural network with $k + 1$ layers. Hence, we can write:

$$\begin{aligned} f_{k+1}(\mathbf{x}) &= \left(\phi^{h_{k+1},h_k}_{\mathbf{W}_{k+1},\mathbf{b}_{k+1}} \circ \phi^{h_k,h_{k-1}}_{\mathbf{W}_k,\mathbf{b}_k} \circ ... \circ \phi^{h_1,h_0}_{\mathbf{W}_1,\mathbf{b}_1}\right)(\mathbf{x}) \\ &= \phi^{h_{k+1},h_k}_{\mathbf{W}_{k+1},\mathbf{b}_{k+1}}\left(\left(\phi^{h_k,h_{k-1}}_{\mathbf{W}_k,\mathbf{b}_k} \circ ... \circ \phi^{h_1,h_0}_{\mathbf{W}_1,\mathbf{b}_1}\right)(\mathbf{x})\right) \\ &= \phi^{h_{k+1},h_k}_{\mathbf{W}_{k+1},\mathbf{b}_{k+1}}(f_k(\mathbf{x})) \end{aligned} \tag{11}$$

We see that $f_k(\mathbf{x}) := \left(\phi^{h_k,h_{k-1}}_{\mathbf{W}_k,\mathbf{b}_k} \circ ... \circ \phi^{h_1,h_0}_{\mathbf{W}_1,\mathbf{b}_1}\right)(\mathbf{x})$ is a linear neural network with k layers and is therefore an affine mapping, according to our assumption in the previous step. Hence, we can write $f_k(\mathbf{x}) = \tilde{\mathbf{W}}\mathbf{x} + \tilde{\mathbf{b}}$. This then gives:

$$\begin{aligned} f_{k+1}(\mathbf{x}) &= \phi^{h_{k+1},h_k}_{\mathbf{W}_{k+1},\mathbf{b}_{k+1}}(f_k(\mathbf{x})) \\ &= \phi^{h_{k+1},h_k}_{\mathbf{W}_{k+1},\mathbf{b}_{k+1}}\left(\tilde{\mathbf{W}}\mathbf{x} + \tilde{\mathbf{b}}\right) \\ &= \mathbf{W}_{k+1}\left(\tilde{\mathbf{W}}\mathbf{x} + \tilde{\mathbf{b}}\right) + \mathbf{b}_{k+1} \\ &= \left(\mathbf{W}_{k+1}\tilde{\mathbf{W}}\right)\mathbf{x} + \left(\mathbf{W}_{k+1}\tilde{\mathbf{b}} + \mathbf{b}_{k+1}\right) \\ &= \hat{\mathbf{W}}\mathbf{x} + \hat{\mathbf{b}} \end{aligned} \tag{12}$$

From equations (11) and (12), we see that f_{k+1} has to be an affine transformation. Hence, we have proved $P(k) = 1 \longrightarrow P(k+1) = 1$.

This step gives (**4 point**).

5. The fifth and last step is to state that from the previous steps we can conclude that $\forall k \in \mathbb{N} : P(k) = 1$. Hence, all linear neural networks are affine transformations.

   The final conclusion gives (**1 point**).

Any correct proof is awarded full points. For questions regarding correctness please contact: jlokna@math.ethz.ch

b) **[15 points]** In the linear auto-encoder we consider a linear neural network with a single hidden layer of size k. Let $\mathbf{X} \in \mathbb{R}^{N \times n}$ be the data matrix with the individual data points as row vectors. Furthermore, we generally have the following decomposition of $\mathbf{X}$, also known as the SVD-decomposition:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ ... \\ \mathbf{x}_N^T \end{bmatrix} = \sum_{i=1}^{min\{N,n\}} \sigma_i^2 \mathbf{u}_i \mathbf{v}_i^T \tag{13}$$

The σ-values are ordered $\sigma_1^2 \geq \sigma_2^2 \geq ... \geq \sigma_{min\{N,n\}}^2 \geq 0$, and the u- and v-vectors are orthonormal, meaning that $\mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}$ and $\mathbf{v}_i^T\mathbf{v}_j = \delta_{ij}$. Lastly, we define $\mathbf{V}_k := [\mathbf{v}_1 ... \mathbf{v}_k]$ with $k \leq min\{N,n\}$.

Based on these definitions prove that $(\mathbf{V}_k, \mathbf{V}_k^T)$ is an optimizer of the objective of the linear auto-encoder as given in equation (14). How does this result relate to the PCA of $\mathbf{X}$?

$$(V_k, V_k^T) \in \underset{U\in\mathbb{R}^{n\times k}, W\in\mathbb{R}^{k\times n}}{\arg\min} \sum_{i=1}^{N} \left\| \left(\phi_{\mathbf{U},\mathbf{0}}^{n,k} \circ \phi_{\mathbf{W},\mathbf{0}}^{k,n}\right)(\mathbf{x}_i) - \mathbf{x}_i \right\|^2 \tag{14}$$

Hint: Use the Eckart-Young-Mirsky theorem which states that $\sum_{i=1}^{k} \sigma_i^2 \mathbf{u}_i \mathbf{v}_i^T$ is the best rank k approximation of $\mathbf{X}$. Also, the following identities might prove useful:

- $A \in \mathbb{R}^{m\times n} \rightarrow Rank(A) \leq min\{m, n\}$
- $Rank(AB) \leq min\{Rank(A), Rank(B)\}$

The first step of the proof is to transform the objective into matrix form:

$$\begin{aligned}
\sum_{i=1}^{N} \left\| \left(\phi_{\mathbf{U},\mathbf{0}}^{n,k} \circ \phi_{\mathbf{W},\mathbf{0}}^{k,n}\right)(\mathbf{x}_i) - \mathbf{x}_i \right\|_2^2 &= \sum_{i=1}^{N} \left\| \mathbf{U}\mathbf{W}\mathbf{x}_i - \mathbf{x}_i \right\|_2^2 \\
&= \left\| \mathbf{X}\mathbf{W}^T\mathbf{U}^T - \mathbf{X} \right\|_F^2 \\
&= \left\| \tilde{\mathbf{X}}_{\mathbf{k}} - \mathbf{X} \right\|_F^2
\end{aligned} \tag{15}$$

Correctly reforming the objective gives (**2 point**).

Then, we compute the rank of $\tilde{\mathbf{X}}_{\mathbf{k}}$:

$$\begin{aligned}
Rank(\tilde{\mathbf{X}}_{\mathbf{k}}) &= Rank(\mathbf{X}\mathbf{U}^T\mathbf{W}^T) \\
&\leq min\{Rank(X), Rank(U), Rank(W)\} \\
&\leq min\{min\{n, N\}, min\{n, k\}, min\{n, k\}\} \\
&= min\{n, k, N\} = k
\end{aligned} \tag{16}$$

Correctly computing an upper bound on the rank gives (**1 point**).

Secondly, We plug in $\mathbf{U} = \mathbf{V}_k$ and $\mathbf{W} = \mathbf{V}_k^T$ and compute the resulting $\tilde{\mathbf{X}}_{\mathbf{k}}$:

$$
\begin{aligned}
\tilde{\mathbf{X}}_{\mathbf{k}} &= \mathbf{X}\mathbf{W}^T\mathbf{U}^T \\
&= \mathbf{X}\mathbf{V}_k\mathbf{V}_k^T \\
&= \left(\sum_{i=1}^{min\{N,n\}} \sigma_i^2 \mathbf{u}_i\mathbf{v}_i^T\right)\mathbf{V}_k\mathbf{V}_k^T \\
&= \left(\sum_{i=1}^{min\{N,n\}} \sigma_i^2 \mathbf{u}_i\left(\mathbf{v}_i^T\mathbf{V}_k\right)\right)\mathbf{V}_k^T \\
&= \left(\sum_{i=1}^{k} \sigma_i^2 \mathbf{u}_i\mathbf{e}_i^T\right)\mathbf{V}_k^T \\
&= \sum_{i=1}^{k} \sigma_i^2 \mathbf{u}_i\left(\mathbf{e}_i^T\mathbf{V}_k^T\right) \\
&= \sum_{i=1}^{k} \sigma_i^2 \mathbf{u}_i\mathbf{v}_i^T
\end{aligned}
\tag{17}
$$

Proving that that using $\mathbf{U} = \mathbf{V}_k$ and $\mathbf{W} = \mathbf{V}_k^T$, results in $\tilde{\mathbf{X}}_{\mathbf{k}} = \sum_{i=1}^k \sigma_i^2\mathbf{u}_i\mathbf{v}_i^T$, gives (**4 point**).

From the Eckart-Young-Mirsky theorem we know that the best rank-k approximation of $\mathbf{X}$ w.r.t. the Frobenius-norm is $\sum_{i=1}^k \sigma_i^2\mathbf{u}_i\mathbf{v}_i^T$. Hence, as we know that $Rank(\tilde{\mathbf{X}}_{\mathbf{k}}) = k$ and that using $\mathbf{U} = \mathbf{V}_k$ and $\mathbf{W} = \mathbf{V}_k^T$, results in $\tilde{\mathbf{X}}_{\mathbf{k}} = \sum_{i=1}^k \sigma_i^2\mathbf{u}_i\mathbf{v}_i^T$, we conclude that $(\mathbf{V}_k, \mathbf{V}_k^T)$ is an optimizer of the objective of the linear auto-encoder as given in equation (14).

Concluding using the Eckart-Young-Mirsky theorem gives (**5 point**).

There are several relations to PCA, here we point out a few:

1. One relation to PCA is that $\mathbf{v}_1, ..., \mathbf{v}_k$ are the first k principal components of the data matrix. So a solution to PCA can be directly used to construct optimal weights to the linear auto-encoder.
2. Another is that the objective in equation (14) equals the minimal reconstruction loss if we allow for arbitrary pairs of linear encoding and decoding mappings.
3. A third relation is that both can be seen as mapping higher dimensional data onto a lower dimensional space while trying to maintain as much information as possible.

Pointing out one of the relations to PCA gives (**3 point**).

Any questions regarding correctness of alternative proofs and relations between PCA and linear auto-encoders can be sent to: jlokna@math.ethz.ch

References

[1] Oja, Erkki Simplified neuron model as a principal component analyzer, Journal of mathematical biology, 1982.

[2] Oja, Erkki, Principal components, minor components, and linear neural networks, Neural networks, 1992.

[3] Terence, D. Sanger, Optimal unsupervised learning in a single-layer linear feedforward neural network, Neural Networks, 1989.

[4] Bruno, A. Olshausen, Linear Hebbian learning and PCA.

[5] Baldi, Pierre and Hornik, Kurt, Neural networks and principal component analysis: Learning from examples without local minima, Neural networks, 1989.