582744 Advanced Course in Machine Learning

Exercise 4 **Due April 19, 10:00 AM**

Rules:

1. Your submission is composed of two parts: (a) A single PDF file containing your answers to all questions, including both the pen and paper questions and the written answers and various plots for the programming questions. (b) a single compressed file (zip or tar.gz) containing your code (and nothing else). If your code is in a single file, it can be sent also a plain source code.
2. The submission should be sent directly to BOTH Arto (`arto.klami@cs.helsinki.fi`) and Aditya (`aditya.jitta@cs.helsinki.fi`).
3. All material must be renamed to your student ID. Always mention your name and student ID also in the written report.
4. The subject field of your email should be [AML][your student ID][exercise 4].
5. Please typeset your work using appropriate software such as LaTeX. However, there is no need to typeset the pen and paper answers – you can also include a scanned hand-written version.
6. The pen and paper exercises can alternatively be returned in paper form during the Tuesday lecture.

This set of exercises is due on Tuesday April 19th, before 10:00 AM.

1 Deflationary orthogonalization (3 points)

Optimization of ICA (and PCA) requires finding a matrix $\mathbf{W}$ that is orthonormal, meaning that $\mathbf{w}_i^T \mathbf{w}_i = 1$ for all i and $\mathbf{w}_i^T \mathbf{w}_j = 0$ for all $i \neq j$. Here $\mathbf{w}_i$ are the columns of $\mathbf{W}$.

Assume we are given an iterative optimization algorithm for minimizing the loss function over a single vector, updating $\mathbf{w}_i^{t+1} = f(\mathbf{w}_{it})$. Assuming we have already found the first k columns of the solution, we can use the following algorithm to find the next solution:

1. Initialize $\mathbf{w}_i$ randomly
2. Perform one iteration of the optimization algorithm to get $\mathbf{w}_i^{t+1}$
3. Normalize the solution vector using

$$\mathbf{w}_i = \mathbf{w}_i - \sum_{j \leq K} (\mathbf{w}_i^T \mathbf{w}_j) \mathbf{w}_j$$
$$\mathbf{w}_i = \frac{\mathbf{w}_i}{\sqrt{\mathbf{w}_i^T \mathbf{w}_i}}.$$

4. Repeat the two previous steps until convergence of the optimization process

Prove that the resulting $\mathbf{W}$ is orthonormal.

2 Independent component analysis (3 points)

Given a collection of K observed signals $\mathbf{X} \in \mathbb{R}^{K \times T} = \mathbf{WS}$, independent component analysis is formulated as learning the mixing matrix $\mathbf{W} \in \mathbb{R}^{K \times K}$ and the latent signals. Here $\mathbf{X}$ and $\mathbf{S}$ are treated as time series of T-points and we observe K signals for each. We can solve the problem if the rows of $\mathbf{S}$ are statistically independent and non-Gaussian. However, the solution is not unique. Which aspects of $\mathbf{S}$ cannot be identified? Why?

Now assume we find the independent components by maximum likelihood estimation of the model

$$p(\mathbf{X}_{.t} | \mathbf{W}) = N(\mathbf{W}\mathbf{S}_{.t}, \sigma^2 \mathbf{I})$$
$$p(\mathbf{S}_{.t}) = \prod_{k=1}^{K} p_k(\mathbf{S}_{kt}),$$

where we assume the densities for the latent signals follow the logistic distribution

$$p_k(\mathbf{S}_{kt}) = -2 \log \cosh(\frac{\pi}{2\sqrt{3}} \mathbf{S}_{kt}) - \log \frac{4\sqrt{3}}{\pi}.$$

Does this change the answer to the previous question? Why?

3 Non-negative matrix factorization (programming, 9 points)

Implement non-negative matrix factorization using the multiplicative update rules presented in the lecture slides. For more information, you can consult the original research article http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf.

Read in the data set https://www.cs.helsinki.fi/u/aklami/teaching/AML/exercise_4.csv that contains small image patches extracted from natural images, represented as row-wise concatenation of 16×16 grayscale pixel representations. Store it as a data matrix $\mathbf{X}$ so that each image patch is represented as one column.

Run NMF on the first 3500 examples in that data set using some reasonable choice for the rank K of the approximation, and plot the training and validation losses as a function of the algorithm iteration. The validation loss is here computed for the last 500 samples, and the loss is given by

$$L(\mathbf{W}, \mathbf{H}) = \|\mathbf{X} - \mathbf{W}\mathbf{H}\|^2,$$

the squared re-construction error.

The updates were given at the lecture notes but the equations were somewhat hastily written, and are hence repeated here for convenience:

$$\mathbf{H}_{t+1} = \mathbf{H}_t \times \frac{\mathbf{W}_t^T \mathbf{X}}{\mathbf{W}_t^T \mathbf{W}_t \mathbf{H}_t}$$
$$\mathbf{W}_{t+1} = \mathbf{W}_t \times \frac{\mathbf{X}\mathbf{H}_{t+1}^T}{\mathbf{W}_t \mathbf{H}_{t+1} \mathbf{H}_{t+1}^T}$$

Here the products in the numerator and denumerator are regular matrix products, but the division and the multplication with $\mathbf{H}_t$ and $\mathbf{W}_t$ are element-wise.

Repeat the same process using PCA (you can use ready-made implementations here), using the same choice for K. Which one reaches better training error? How about validation error? Can you think of an explanation for the result?

Finally, visualize the resulting basis vectors (columns of $\mathbf{W}$) in image format (converting the vectors back to matrices for visualization; the result should look a bit like the example shown during the lecture). For PCA you should order the components according to the variance explained. Can you think of a good ordering for the NMF components?

Hint: To compute validation error for NMF you need to run the same algorithm but using fixed $\mathbf{W}$. This is because there is no closed-form expression for $\mathbf{H}$ even if we know $\mathbf{W}$. A practical way to compute the validation error during the training process is to keep on updating $\mathbf{H}_{train}$ and $\mathbf{H}_{validation}$ during training, remembering to only use $\mathbf{X}_{train}$ and $\mathbf{H}_{train}$ for updating $\mathbf{W}$. Alternatively, you can learn $\mathbf{H}_{validation}$ separately after learning $\mathbf{W}$.

(Note that the dataset is originally from http://mldata.org/repository/data/viewslug/natural-scenes-data/ but was modified slightly for this exercise. Also note that the first release of the exercise has the image patches as rows of $\mathbf{X}$ instead of columns and hence the roles of $\mathbf{W}$ and $\mathbf{H}$ were swapped. The notation was changed to match the original NMF article.)

4 Stochastic neighbor embedding (programming, 9 points)

Implement stochastic neighbor embedding, using the loss function and gradient presented in the lecture slides. For more information, you can consult the original research article http://papers.nips.cc/paper/2276-stochastic-neighbor-embedding.pdf.

In this exercise you will use it for exploring the MNIST data set of handwritten digits, mapping them to a two-dimensional embedding space. Download the data set and useful code snippets for handling it from http://www.cs.helsinki.fi/u/sorkhei/mnist.tar.gz; this is the same package that was used on the "Introduction to ML" course last year.

- Initialize the algorithm using the first two PCA components, using at least 5000 of the examples for computing the principal components. Normalize the scores by their standard deviation, so that the variance for each dimension is one.

- Write a function that computes the neighborhoods using

$$d_{ij}^2 = \sum_d \frac{(\mathbf{x}_{id} - \mathbf{x}_{jd})^2}{2\sigma^2}$$
$$p_{ij} = \frac{e^{-d_{ij}^2}}{\sum_{k \neq j} e^{-d_{ik}^2}},$$

  further assuming $d_{ij} = \infty$ to exlude the point itself from the neighborhood

- Compute the neighborhoods p_{ij} in the original data space using $\sigma^2 = 10000^2$ in the above formula, and the neighborhoods q_{ij} in the embedding space using $\sigma^2 = 1$. How do these two parameters control the algorithm? Why did we need very large value for the former?

- Optimize the loss function

$$L(\mathbf{z}) = \sum_{i,j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

  using gradient descent with the gradient

$$\frac{\partial L}{\partial \mathbf{z}_i} = 2 \sum_j (\mathbf{z}_i - \mathbf{z}_j)(p_{ij} - q_{ij} + p_{ji} - q_{ji}).$$

  Start with standard gradient descent using fixed step size of 0.05, but if you wish you can also try faster alternatives. Plot the training loss as a function of the algorithm iteration.

- Visualize both the initial PCA representation and the final SNE representation with scatter-plots, coloring the samples according to the class labels. Does either one look better?

Note that the algorithm is rather slow, so you probably will not want to learn the embedding for all of the samples. Try making an implementation that works at least for 1000 samples, but if your code is too slow then use a smaller subset. Remember to mention in your report how many samples you used.

If you want, you can also play around with different initializations, optimization algorithms, and different values for σ. The choices above were given to make grading easier (that is, so that the result plots and errors would be comparable across your solutions) and they are definitely not optimal for solving the problem itself.