Machine Learning
Exercise 5

Marc Toussaint
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

May 23, 2014

1 PCA and reconstruction on the Yale face database

On the webpage find and download the Yale face database http://ipvs.informatik.uni-stuttgart.de/mlr/marc/teaching/data/yalefaces.tgz. (Optinally use `yalefaces_cropBackground.tgz`.) The file contains gif images of 165 faces.

a) Write a routine to load all images into a big data matrix $X \in \mathbb{R}^{165\times 77760}$, where each row contains a gray image.

In Octave, images can easily be read using `I=imread("subject01.gif");` and `imagesc(I);`. You can loop over files using `files=dir(".");` and `files(:).name`. Python tips:
`import matplotlib.pyplot as plt import scipy as sp plt.imshow(plt.imread(file_name))`
`u, s, vt = sp.sparse.linalg.svds(X, k=neigenvalues)`

b) Compute the mean face $\mu = 1/n \sum_i x_i$ and center the whole data, $X \leftarrow X - \mathbf{1}_n \mu^\top$.

c) Compute the singular value decomposition $X = UDV^\top$ for the data matrix.[1] In Octave/Matlab, use the command `[U, S, V] = svd(X, "econ")`, where the `econ` ensures we don't run out of memory.

d) Map the whole data set to $\boldsymbol{Z} = \boldsymbol{X} V_p$, where $V_p \in \mathbb{R}^{77760\times p}$ contains only the first p columns of V. Assume $p = 60$. The Z represents each face as a p-dimensional vector, instead of a 77760-dimensional image.

e) Reconstruct all images by computing $\tilde{X} = \mathbf{1}_n \mu^\top + \boldsymbol{Z} V_p^\top$. Display the reconstructed images (by reshaping each row of $\tilde{X}$ to a 320×243-image) – do they look ok? Report the reconstruction error $\|X - \tilde{X}\|^2 = \sum_{i=1}^n \|x_i - \tilde{x}_i\|^2$. Repeat for various PCA-dimensions $p = 1, 2, ...$

2 Neural Networks (optional)

Neural networks (NNs) are an class of parameterized functions $y = f(x, w)$ that map some input $x \in \mathbb{R}^n$ to some output $y \in \mathbb{R}$ depending on parameters w. We've introduced the *logistic sigmoid function* as

$$\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{e^{-z}+1}\,, \quad \sigma'(z) = \sigma(z)(1-\sigma(z))$$

A 1-layer NN, with h_1 neurons in the hidden layer, is defined as

$$f(x, w) = w_1^\top \sigma(W_0 x)\,, \quad w_1 \in \mathbb{R}^{h_1}, W_0 \in \mathbb{R}^{h_1 \times d}$$

with parameters $w = (W_0, w_1)$, where $\sigma(z)$ is applied component-wise.

A 2-layer NN, with h_1, h_2 neurons in the hidden layers, is defined as

$$f(x, w) = w_2^\top \sigma(W_1 \sigma(W_0 x))\,, \quad w_2 \in \mathbb{R}^{h_2}, W_1 \in \mathbb{R}^{h_2 \times h_1}, W_0 \in \mathbb{R}^{h_1 \times d}$$

with parameters $w = (W_0, W_1, w_2)$.

The weights of hidden neurons W_0, W_1 are usually trained using gradient descent. The output weights w_1 or w_2 can be optimized analytically as for linear regression.

[1] This is alternative to what was discussed in the lecture: In the lecture we computed the SVD of $X^\top X = (UDV^\top)^\top (UDV^\top) = VD^2V^\top$, as U is orthonormal and $U^\top U = \mathbf{I}$. Decomposing the covariance matric $X^\top X$ is a bit more intuitive, decomposing X directly is more efficient and amounts to the same V.

a) Derive the gradient $\frac{\partial}{\partial W_0} f(x)$ for the 1-layer NN. (For simplicity, provide the gradient element-wise.)

b) Derive the gradients $\frac{\partial}{\partial W_0} f(x)$ and $\frac{\partial}{\partial W_1} f(x)$ for the 2-layer NN.