Exercises

Latent Variable Models

Maneesh Sahani

1. **Principal Components Analysis**.

   The conventional latent variable model for Probabilistic Principal Components Analysis has a standard normal latent $\mathbf{y}$ and an arbitrary *loading matrix* Λ.

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, I)$$
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\Lambda\mathbf{y}, \psi I).$$

   An alternative model would be to draw $\mathbf{y}$ from a normal with diagonal covariance (say Υ), and then restrict Λ to be orthogonal:

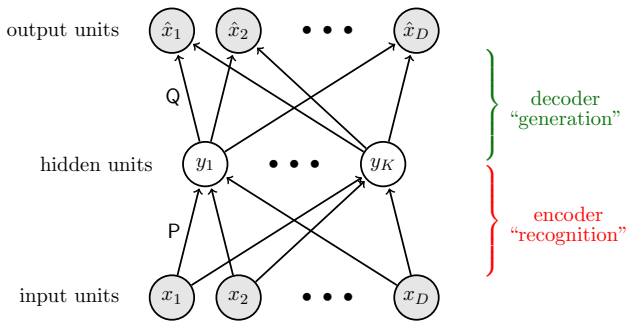
$$p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, \Upsilon); \qquad \Upsilon_{ij} = 0 \text{ for } i \neq j$$
$$p(\mathbf{x}|\mathbf{y}) = \mathcal{N}(\Lambda\mathbf{y}, \psi I); \qquad \Lambda^\mathsf{T}\Lambda = I.$$

   (a) Show that this alternative model is equivalent to the standard one.

   (b) Derive the mean and covariance of $p(\mathbf{y}|\mathbf{x})$ within the alternative model in the non-probabilistic PCA limit, $\psi \to 0$.

2. **Linear Autoencoders**. A linear autoencoder is a "neural network" implementation of PCA. The network has three "layers" of "units" — each unit represents a single scalar variable, and so each layer represents a vector:

$\hat{x}_j$	$j = 1 \ldots D$	output units
y_k	$k = 1 \ldots K < D$	hidden units
x_i	$i = 1 \ldots D$	input units



   The mappings from input to hidden, and hidden to output layers, are linear:

$$y_k = \sum_i P_{ki}\, x_i$$
$$\hat{x}_j = \sum_k Q_{jk}\, y_k$$

   Given a set of N observed "input" vectors $\{\mathbf{x}_n\}$, the weight matrices P and Q are set to minimise the "autoencoding error" $\sum_n \|\hat{\mathbf{x}}_n - \mathbf{x}_n\|^2$, often using iterative gradient descent. Assume the input vector distribution has zero mean.

   (a) Show that after the weights have converged, they obey the identities:

$$P = (Q^\mathsf{T}Q)^{-1}Q^\mathsf{T}$$
$$Q = \Sigma_X P^\mathsf{T}(P\Sigma_X P^\mathsf{T})^{-1}$$

   where $\Sigma_X = \sum_n \mathbf{x}_n\mathbf{x}_n^\mathsf{T}$ and we assume that Q is rank K.

   (b) It is clear that if P and Q minimize the error, then, for any invertible $K \times K$ matrix C the matrices $P_* = CP$ and $Q_* = QC^{-1}$ also achieve the same minimum. Show that you can always find a matrix C such that $Q_*^\mathsf{T}Q_* = I$ and so $P_* = Q_*^\mathsf{T}$.

(c) Show that the minimisation of the error is equivalent to the maximisation of

$$\mathsf{Tr}\left[Q_* Q_*^{\mathsf{T}} \Sigma_X\right]$$

and that this maximum is achieved by choosing the columns of Q_* proportional to the first K eigenvectors of Σ_X. (You may need to look up the relationship between the singular value decomposition—or eigendecomposition of a symmetric matrix—and low-rank approximations).

This demonstrates the assertion made in lecture that the linear autoencoder will find the subspace of the leading K principal components.

(d) How would you adapt the algorithm to implement factor analysis in the case that the uniquenesses (another term for the output noise variances Ψ_{dd}) are known?

(e) Is it possible to use an autoencoder for FA in the case of unknown uniquenesses? How, if so; or why not, if not?

3. **Simulated data**.

Run the MATLAB command `[X,Y,Z,C] = GenerateData;`. This will return simulated spike counts `X` (100 neurons × 50 time bins × 100 trials). Poisson rates `Z` (same size), underlying latent variables `Y` (4 variables × 50 time bins × 100 trials), and the 'loading matrix' `C` (100 × 4).

Look at the trial average values of `X`. Can you see structure?

Implement and run PCA and FA on the pooled single trial counts (that is, on `reshape(X, 100, [])`) and on the trial-averages. How do the four results compare? Why the differences? Is the PCA eigenspectrum helpful? Why does FA not recover the matrix `C` exactly?

Now call `[X,Y,Z,C] = GenerateData('randphase', 1);`. Look at the trial-averaged mean counts, and at individual trials. Can you see what the difference is?

Run PCA and FA on the new data with and without averaging. Interpret your results.

Use the provided code to run GPFA. Try running the cross-validation part – it should correctly indicate a dimensionality of 4. Compare the orthonormalised latents and the regular ones (in `seqTrain(:).xorth` and `seqTrain(:).xsm` respectively. Why are they differnt, and which is more interpretable?