MLPR Tutorial Sheet 5

Reminders: Attempt the tutorial questions, and ideally discuss them, before your tutorial. You can seek clarifications and hints on the class forum. Move on if you're getting stuck on a part for a long time. Groups often won't discuss every part. Full answers will be released after the last group meets.

---

1. **The costs of some classifiers:**

   We have a training set of N examples, with D-dimensional features and binary labels.

   Assume the following computational complexities: matrix-matrix multiplication AB costs $O(LMN)$ for $L \times M$ and $M \times N$ matrices A and B. Inverting an $N \times N$ matrix G and/or finding its determinant costs $O(N^3)$.[1]

   a) What is the computational complexity of training a "Bayes classifier" that models the features of each class by a maximum likelihood Gaussian fit (a Gaussian matching the mean and covariances of the features)?

   b) What is the computational complexity of assigning class probabilities to a test feature vector?

   c) In *linear discriminant analysis* we assume the classes just have different means, and share the same covariance matrix. Show that given its parameters θ, the "log odds" for a binary classifier,

   $$\log \frac{p(\mathbf{x} \,|\, y=1, \theta)}{p(\mathbf{x} \,|\, y=0, \theta)} \;=\; \log p(\mathbf{x} \,|\, y=1, \theta) - \log p(\mathbf{x} \,|\, y=0, \theta),$$

   is a linear function of $\mathbf{x}$ (as opposed to quadratic).

   d) When the log odds are linear, the predictions have the same form as logistic regression: $P(y=1 \,|\, \mathbf{x}, \theta) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$, but the parameters $\mathbf{w}(\theta)$ and $b(\theta)$ are fitted differently.

   How do your answers to a) and b) change when the classes share one covariance? What can you say about the cost of the linear discriminant analysis method compared to logistic regression?

2. **Linear autoencoder**

   We centre our data so it has zero mean and fit a linear autoencoder with no bias parameters. The autoencoder is a D-dimensional vector-valued function $\mathbf{f}$, computed from D-dimensional inputs $\mathbf{x}$, using an intermediate K-dimensional "hidden" vector $\mathbf{h}$:

   $$\mathbf{h} = W^{(1)}\mathbf{x}$$
   $$\mathbf{f} = W^{(2)}\mathbf{h}.$$

   Assume we want to find a setting of the parameters that minimizes the square error $\|\mathbf{f} - \mathbf{x}\|^2 = (\mathbf{f} - \mathbf{x})^\top (\mathbf{f} - \mathbf{x})$, averaged (or summed) over training examples.

   a) What are the sizes of the weight matrices $W^{(1)}$ and $W^{(2)}$? Why is it usually not possible to get zero error for $K < D$?

---

1. In fact, good implementations usually take a factorization of G rather than inverting it, which also costs $O(N^3)$. Given that factorization, we can compute $G^{-1}H$ in $O(N^2K)$, where H is $N \times K$, and find the (log) determinant in $O(N)$. I'm using "big-O notation". A good introduction on this notation in the context of computational complexity are the notes from our second year Inf2b course. Applied areas like machine learning usually use big-O notation more sloppily than in that note; I think I have only once used Ω or Θ.

b) It's common to transform a batch (or "mini-batch") of data at one time. Given an $N \times D$ matrix of inputs X, we set:

$$H = XW^{(1)\top}$$
$$F = HW^{(2)\top}$$

The total square error $E = \sum_{nd}(F_{nd} - X_{nd})^2$, has derivatives with respect to the neural network output

$$\frac{\partial E}{\partial F_{nd}} = 2(F_{nd} - X_{nd}), \quad \text{which we write as } \bar{F} = 2(F - X).$$

Using the backpropagation rule for matrix multiplication,

$$C = AB^\top \quad \Rightarrow \quad \bar{A} = \bar{C}B \;\text{ and }\; \bar{B} = \bar{C}^\top A,$$

write down how to compute derivatives of the cost with respect to $W^{(1)}$ and $W^{(2)}$. If time: you should be able to check numerically whether you are right.

c) The PCA solution sets $W^{(1)} = V^\top$ and $W^{(2)} = V$, where the columns of V contain eigenvectors of the covariance of the inputs. We only really need to fit one matrix to minimize square error.

Tying the weight matrices together: $W^{(1)} = U^\top$ and $W^{(2)} = U$, we can fit one matrix U by giving its gradients $\bar{U} = \bar{W}^{(1)\top} + \bar{W}^{(2)}$ to a gradient-based optimizer. Will we fit the same V matrix as PCA?

3. **Non-linear autoencoders**

Some datapoints lie along the one-dimensional circumference of a semi-circle. You could create such a dataset, by drawing one of the features from a uniform distribution between -1 and $+1$, and setting the other feature based on that:

$$x_1^{(n)} \sim \text{Uniform}[-1, 1]$$
$$x_2^{(n)} = \sqrt{1 - \left(x_1^{(n)}\right)^2}.$$

a) Explain why these points can't be perfectly reconstructed when passed through the linear autoencoder in Q2 with $K = 1$.

b) Explain whether the points could be perfectly reconstructed with $K = 1$ by some non-linear *decoder*: $\mathbf{f} = \mathbf{g}(h)$. Where $\mathbf{g}$ could be an arbitrary function, perhaps represented by multiple neural network layers. Assume the *encoder* is still linear: $h = W^{(1)}\mathbf{x}$.

c) Explain whether the points could be perfectly reconstructed with $K = 1$ by some non-linear *encoder*: $h = g(\mathbf{x})$. Where g could again be an arbitrary function, perhaps represented by multiple neural network layers. Assume the *decoder* is still linear: $\mathbf{f} = W^{(2)}h$.

If you run out of things to do (I think most of the class actually have plenty to do), you could try to implement and fit some of the models mentioned above. For example, can you fit a dataset as well as PCA using the ideas in Q2? Or can you create and fit a dataset lying on a low-dimensional manifold as in Q3? There's probably not time to discuss or debug code in your tutorial groups. However, I will comment on code posted to the forum:

```
```
Put code between three backtics like this.
```
```