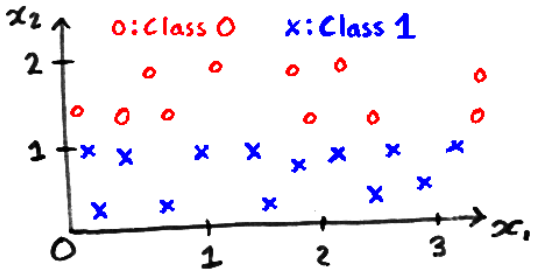
MLPR Tutorial Sheet 7

1. **Learning a transformation:**

   The K-nearest-neighbour (KNN) classifier predicts the label of a feature vector by finding the K nearest feature vectors in the training set. The label predicted is the label shared by the majority of the training neighbours. For binary classification we normally choose K to be an odd number so ties aren't possible.

   KNN is an example of a *non-parametric* method: no fixed-size vector of parameters (like $\mathbf{w}$ in logistic regression) summarizes the training data. Instead, the complexity of the function represented by the classifier grows with the number of training examples N. Often, the whole training set needs to be stored so that it can be consulted at test time. (Gaussian processes are also a non-parametric method.)

   a) How would the predictions from regularized linear logistic regression, with $p(y=1 \mid \mathbf{x}, \mathbf{w}, b) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$, and 1-nearest neighbours compare on the dataset below?



   Non-parametric methods can have parameters. We could modify the KNN classifier by taking a linear transformation of the data $\mathbf{z} = A\mathbf{x}$, and finding the K nearest neighbours using the new $\mathbf{z}$ features. One loss function for evaluating possible transformations A, could be the *leave-one-out* (LOO) classification error, defined as the fraction of errors made on the training set when the K nearest neighbours for a training item may not include the point being classified. (It's an M-fold cross-validation loss with $M=N$.)

   b) Write down a matrix A where the 1-nearest neighbour classifier has lower LOO error than using the identity matrix for the data above, and explain why it works.

   c) Explain whether we can fit the LOO error for a KNN classifier by gradient descent on the matrix A.

   d) Assume that I have implemented some other classification method where I can evaluate a cost function c and its derivatives with respect to feature input locations: $\bar{Z}$, where $\bar{Z}_{nk} = \frac{\partial c}{\partial Z_{nk}}$ and Z is an $N \times K$ matrix of inputs.

   I will use that code by creating the feature input locations from a linear transformation of some original features $Z = XA$. How could I fit the matrix A? If A is a $D \times K$ matrix, with $K < D$, how will the computational cost of this method scale with D?

   You may quote results given in lecture note w7c.

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 $K < D$?

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.
```

Tutorial 7 is the last tutorial, but you can still go to ML-Base to discuss your questions, questions in the notes, or past tutorial questions.