ECE-GY 6143: Introduction to Machine Learning
Final Exam Solutions, Spring 2019

Prof. Sundeep Rangan

1. *SVM.* Consider an SVM classifier,

$$\widehat{y} = \begin{cases} 1 & \text{if } z > 0, \\ -1 & \text{if } z \geq 0, \end{cases} \qquad z = \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + b,$$

for the kernel function,

$$K(\mathbf{x}, \mathbf{x}_i) := \begin{cases} 1 & \text{if } \|\mathbf{x} - \mathbf{x}_i\|^2 \leq r^2, \\ 0 & \text{if } \|\mathbf{x} - \mathbf{x}_i\|^2 > r^2, \end{cases}$$

where $r > 0$ is some parameter and $\|\mathbf{w}\|^2$ denotes the squared norm $\|\mathbf{w}\|^2 = w_1^2 + w_2^2$. You are given four training samples.

i	1	2	3	4
x_{i1}	0	1	0	2
x_{i2}	0	0	2	2
y_i	1	-1	-1	1

(a) Find parameters r, α_i and b such that the SVM classifier makes no errors on the training data. Use at most two non-zero values of α_i.

(b) Given the choice of parameters in (a), draw the region of (x_1, x_2) where the classifier predicts $\widehat{y} = 1$.

(c) Complete the following python function `predict` that computes a vector of outputs `yhat` for a data matrix `X`. You must provide the other arguments needed. For full credit, avoid for loops.

```
def predict(X,...):
    ...
    return yhat
```

Solution:

(15 points total)

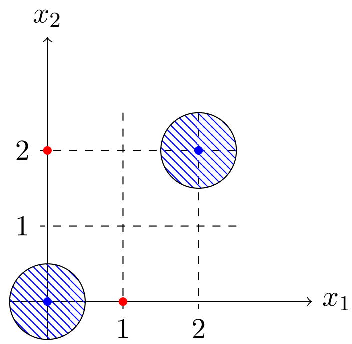

Figure 1: Scatter plot of the data points where the red circles are $y_i = -1$ and blue are $y_i = 1$. The hashed regions are the areas where the classifier will set $\widehat{y} = 1$.

(a) (6 points) Take $\alpha = [1, 0, 0, 1]$, $b = -0.5$ and $r = 0.5$. Then,

$$\begin{aligned} z &= \sum_{i=1}^{N} \alpha_i y_i K(x, x_i) + b \\ &= K(x, x_1) + K(x, x_4) - 0.5. \end{aligned}$$

Since $r = 0.5$, we have $K(x_i, x_j) = 0$ for all training points $x_i \neq \times_j$ since the training points are separated by more than r. Therefore, at $x = x_1$ we have

$$z = K(x_1, x_1) + K(x_1, x_4) - 0.5 = 1 + 0 - 0.5 > 0.$$

Similarly, at $x = x_4$, $z > 0$. But, at $x = x_2$,

$$z = K(x_2, x_1) + K(x_2, x_4) - 0.5 = 0 + 0 - 0.5 < 0.$$

Similarly, at $x = x_3$, $z < 0$. Therefore, the classifier classifies all the points correctly.

(b) (4 points) See Fig. 1.

(c) (5 points, 3 points if for loops were used instead of Python broadcasting) One solution is as follows:

```
def predict(X,Xtr,ytr,b,alpha,r):
    # Compute the distances
    # D[i,j] = ||X[i,:] - Xtr[j,:]||^2
    D = np.sum((X[:,None,:] - Xtr[None,:,:])**2, axis=2)

    # Compute the kernel
    K = (D < r**2)

    # Compute the score
```

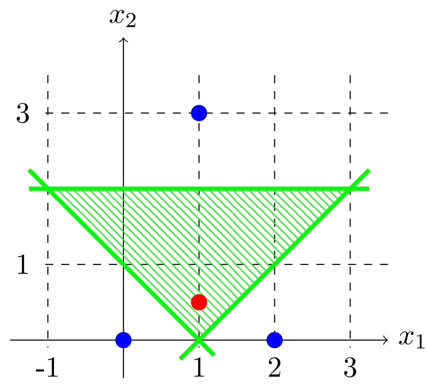

Figure 2: Green lines: Boundaries of the three hidden units. The hidden units $u_j^{\text{H}} = 1$ in the interior of the triangle. Blue circles: Points classified to $\widehat{y} = 0$; Red circle: Point classified to $\widehat{y} = 1$.

```
z = K.dot(ytr*alpha) + b
yhat = 2*(z > 0)-1  # Note:  We convert to +/- one
return yhat
```

2. *Neural Networks.* Consider a neural network, with $N_i = 2$ input units, $\mathbf{x} = (x_1, x_2)$, $N_h = 3$ hidden units and one output unit for binary classification:

$$z_j^{\text{H}} = \sum_{k=1}^{N_i} W_{jk}^{\text{H}} x_k + b_j^{\text{H}}, \quad u_j^{\text{H}} = \begin{cases} 1 & \text{if } z_j^{\text{H}} > 0 \\ 0 & \text{if } z_j^{\text{H}} \leq 0, \end{cases} \quad j = 1, \ldots, N_h$$

$$z^{\text{O}} = \sum_{k=1}^{N_h} W_k^{\text{O}} u_k^{\text{H}} + b^{\text{O}}, \quad \widehat{y} = \begin{cases} 1 & \text{if } z^{\text{O}} > 0 \\ 0 & \text{if } z^{\text{O}} \leq 0. \end{cases}$$

(a) Suppose that

$$\mathbf{W}^{\text{H}} = \begin{bmatrix} 0 & -1 \\ 1 & 1 \\ -1 & 1 \end{bmatrix}, \quad \mathbf{b}^{\text{H}} = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}$$

For each hidden unit $j = 1, 2, 3$, draw the regions of inputs (x_1, x_2) where $u_j^{\text{H}} = 1$.

(b) Find a vector of output weights $\mathbf{W}^{\text{O}}$ and bias b^{O} such that:

- $\mathbf{x} = (0, 0)$, $(2, 0)$, $(1, 3)$ are classified as $\widehat{y} = 0$; and
- $\mathbf{x} = (1, 0.5)$ is classified as $\widehat{y} = 1$.

Solution:

(14 points)

(a) (7 points) We have

$$\mathbf{z}^{\mathrm{H}} = \mathbf{W}^{\mathrm{H}}\mathbf{x} + \mathbf{b}^{\mathrm{H}} = \begin{bmatrix} -x_2 + 2 \\ x_1 + x_2 - 1 \\ -x_1 + x_2 + 1 \end{bmatrix}.$$

Hence,

$$u_1^{\mathrm{H}} = 1 \Longleftrightarrow x_2 \leq 2$$
$$u_2^{\mathrm{H}} = 1 \Longleftrightarrow x_2 \geq 1 - x_1$$
$$u_3^{\mathrm{H}} = 1 \Longleftrightarrow x_2 \geq -1 + x_1$$

The boundaries of the three regions are shown in Fig. 2.

(b) (7 points) The points to be classified as $\widehat{y} = 0$ and $\widehat{y} = 1$ are shown, respectively, in blue and red in Fig. 2. Now, we select $\mathbf{W}^{\mathrm{O}} = [1, 1, 1]$ and $b^{\mathrm{O}} = -2.5$. Then, $z^{\mathrm{O}} > 0$ only when $u_j^{\mathrm{H}} = 1$ for all j. This is the triangular region in the Fig. 2, which contains the red point, but all the blue points are outside this region.

3. *Backpropagation.* Consider the model with D-dimensional inputs $\mathbf{x} = (x_1, \ldots, x_D)$ and M-dimensional outputs $\mathbf{y} = (y_1, \ldots, y_M)$, given by

$$\widehat{y}_m = \sum_{\ell=1}^{L} B_{\ell m} z_\ell, \quad z_\ell = \sum_{j=1}^{D} x_j A_{j\ell}, \quad m = 1, \ldots, M,$$

with parameters $A_{j\ell}$ and $B_{\ell m}$.

(a) You are given training samples $\mathbf{x}_i = (x_{i1}, \ldots, x_{id})$ and $\mathbf{y}_i = (y_{i1}, \ldots, y_{iM})$. Write $\widehat{y}_{im}$ in terms of the inputs x_{ij}.

(b) Draw the computation graph between the data, parameters and loss function using the loss function,

$$J = \sum_{i=1}^{N} \sum_{m=1}^{M} (y_{im} - \widehat{y}_{im})^2.$$

(c) Show how to compute $\partial J / \partial A_{j\ell}$ from $\partial J / \partial z_{i\ell}$.

Solution:

(15 points)

(a) (5 points) We have,

$$\widehat{y}_{im} = \sum_{\ell=1}^{L} B_{\ell m} z_{i\ell}, \quad z_{i\ell} = \sum_{j=1}^{D} x_{ij} A_{j\ell}.$$

(b) (5 points) See Fig. 3.

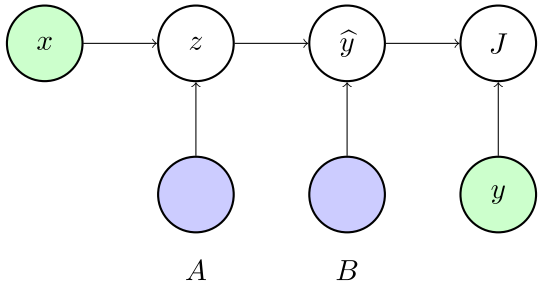

Figure 3: Computation graph for Problem 3. Parameters are shown in light blue and data in light green.

(c) (5 points) We have

$$\partial z_{i\ell}/\partial A_{j\ell} = x_{ij}.$$

By chain rule,

$$\frac{\partial J}{\partial A_{j\ell}} = \sum_i \frac{\partial J}{\partial z_{i\ell}} \frac{\partial z_{i\ell}}{\partial A_{j\ell}} = \sum_i \frac{\partial J}{\partial z_{i\ell}} x_{ij}.$$

4. *CNN kernels*. A systems has five cameras that each take 256 × 192 dimension gray scale images. A mini-batch of training samples consists of 100 samples, with each sample having the five camera images.

(a) What is the shape of a tensor X representing the mini-batch of training data.

(b) A first layer of a CNN performs a convolution with 20 output channels and 3×3 kernels W. Write the equation relating the input X and output Z. What is the shape of W and output Z. Assume the convolution is performed on the valid pixels.

(c) Describe a possible kernel W that detects a difference between camera 2 and camera 3, but ignores cameras 0,1, and 4. The difference output should be in output channel 0.

Solution:

(14 points)

(a) (4 points) Take the shape `(sample,row,height,camera)` for `(100,256,192,5)`.

(b) (5 points) We have

$$Z[i, j_1, j_2, m] = \sum_{k_1,k_2} X[i, j_1 + k_1, j_2 + k_2, n]W[k_1, k_2, n, m].$$

The kernel W will have shape $(3, 3, 5, 20)$. Since the output is computed on the valid pixels, Z, and the kernels are 3×3, will have shape $(100, 256-3+1, 192-3+1, 20) = (100, 254, 190, 20)$.

(c) (5 points) Take

$$W[:,:,n,0] = \begin{cases} \frac{1}{9} & \text{if } n = 2 \\ -\frac{1}{9} & \text{if } n = 3 \\ 0 & \text{else.} \end{cases}$$

In this way, the kernels take the average of cameras 2 and 3 and subtract the two averages.

5. *CNN sub-sampling.* Consider a 1D convolution following by sub-sampling,

$$z[j] = \sum_k w[k]x[j+k], \quad u[m] = z[sm],$$

for some stride parameter $s > 0$. That is, u takes every s samples of z.

(a) If x is 250 milliseconds of audio sampled at 20 kHz, how many samples are in x?

(b) If x has length 1000, w has length 10 and $s = 4$, how many output samples are in u assuming the convolution is only computed on the valid samples.

(c) Given a gradient $\partial J/\partial u[m]$, how do you compute $\partial J/\partial w[k]$?

Solution:

(14 points)

(a) (4 points) The number of samples are $(20)(250) = 5000$.

(b) (4 points) Before sub-sampling, there are $1000 - 10 + 1 = 991$ samples. So, there will be $\lfloor 991/4 \rfloor = 247$ samples

(c) (6 points) Given a gradient $\partial J/\partial u[m]$, how do you compute $\partial J/\partial w[k]$?

We have,

$$u[m] = z[sm] = \sum_k w[k]x[sm+k],$$

so,

$$\frac{\partial u[m]}{\partial w[k]} = x[sm+k].$$

By chain rule,

$$\frac{\partial J}{\partial w[k]} = \frac{\partial J}{\partial u[m]}\frac{\partial u[m]}{\partial w[k]} = \frac{\partial J}{\partial u[m]}x[sm+k].$$

6. *PCA.* You are given python arrays with PCA data:

- `mu`: A `100`-dim vector representing the mean of the data; and
- `V`: A `(100,5)` array with the 5 top PCs of the data.
- `lam`: A `5`-dim vector of eigenvalues.

(a) Given a vector `z` of PC coefficients, write a few lines of python code to reconstruct the data `x`.

(b) Now, suppose you are given only the first 80 of the 100 coefficients of a vector `x`. Write a few lines of python code to estimate the PCA coefficients `z`. You can assume you have a function,

```
w = lstsq(A,b)  # Solves the least-squares solution to b=A.dot(w)
```

(c) Continuing in part (b), how would you estimate the remaining 20 coefficients of `x`?

Solution:

(14 points)

(a) (4 points) One solution is:

```
xhat = mu + V.dot(z)
```

(b) (6 points) We know that $x \approx \mu + Vz$, so the first 80 components are, $x[:80] \approx \mu[:80] + V[:80,:]z$. We can then solve via least squares:

```
z = lstsq(V[:80,:], x[:80]-mu[:80])
```

(c) (4 points) We can recover the last 20 components:

```
xhat[80:] = V[80:,:].dot(z) + mu[80:]
```

7. *K-means.* You are given five data samples:

i	1	2	3	4	5
x_{i1}	0	1	0	2	2
x_{i2}	0	0	1	2	3

(a) Draw the five points.

(b) Starting with $K = 2$ cluster centers at $(0, 0)$ and $(1, 0)$, what are the cluster assignments and new cluster centers after one iteration of K-means?

(c) Suppose you want to use clustering for outlier detection. You find cluster means $\boldsymbol{\mu}_i$, $i = 1, \ldots, K$ on the training data. Then, given a new data $\mathbf{x}$ and a threshold t, you declare $\mathbf{x}$ an outlier if $\|\mathbf{x} - \boldsymbol{\mu}_i\| \geq t$ for all i. Complete the following function to implement the outlier detection on a matrix of data `X`. The output is `out[i]=1` if the sample `X[i,:]` is an outlier, and `out[i]=0` otherwise. You must specify the other inputs of your function. Avoid for loops for full credit.

```
def outlier_detect(X, ...):
    ...
    return out
```

Solution:

(a) (4 points) The five points are shown in Fig. 4.

(b) (4 points) The starting clusters are $\boldsymbol{\mu}_1 = (0, 0)$ and $\boldsymbol{\mu}_2 = (1, 0)$. The points closest

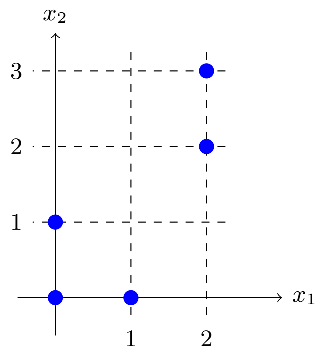

Figure 4: Data points for K-means clustering 7.

to these cluster centers are:

$$C_1 = \{(0,0),(0,1)\}\,,$$
$$C_2 = \{(1,0),(2,2),(2,3)\}\,.$$

The new cluster centers are

$$\boldsymbol{\mu}_1 = \frac{1}{2}\left[(0,0)+(0,1)\right] = (0,\ 0.5),$$
$$\boldsymbol{\mu}_2 = \frac{1}{3}\left[(1,0)+(2,2)+(2,3)\right] = \left(\frac{5}{3},\ \frac{5}{3}\right).$$

(c) (4 points) One solution to this (in python) is as follows:

```
def outlier_detect(x,mu,t):
    # Compute squared distance to the clusters
    # dsq[i,j] = distance from x[i,:] to mu[j,:]
    dsq = np.sum((x[:,None,:] - mu[None,:,:])**2, axis=2)

    # Find the minimium distance
    dmin = np.min(dsq, axis=1)

    # Declare an outlier if minimum distance
    # exceeds threshold
    out = (dmin > t**2)
    return out
```