EE-UY/CS-UY 4563: Introduction to Machine Learning
Midterm 2 Solutions, Fall 2017

1. (33 points)

   (a) (8 points) Let

   $$z_i = \sum_{j=1}^{p} x_{i-j} w_j,$$

   so that $\hat{x}_i = \phi(z_i)$. Also, $\mathbf{z} = \mathbf{A}\mathbf{w}$ if

   $$\mathbf{A} = \begin{bmatrix} x_0 & 0 & \cdots & 0 \\ x_1 & x_0 & \cdots & 0 \\ x_{T-2} & x_{T-2} & \cdots & x_{T-p-1} \end{bmatrix}.$$

   (b) (8 points) The loss function is

   $$J(\mathbf{w}) = \sum_{i=1}^{T-1} g_i(z_i), \quad g_i(z_i) = (\phi(z_i) - x_i)^2.$$

   Using the forward-backward rule,

   $$\nabla J(\mathbf{w}) = \mathbf{A}^\mathsf{T} \nabla_{\mathbf{z}} g(\mathbf{z}),$$

   where

   $$\nabla_{\mathbf{z}} g(\mathbf{z}) = [g_1'(z_i), \cdots, g_{T-1}'(z_{T-1})]^\mathsf{T},$$

   and

   $$g_i'(z_i) = 2(\phi(z_i) - x_i)\phi'(z_i) = 2(\phi(z_i) - x_i)\frac{e^{-z_i}}{1 - e^{-z_i}}.$$

   (c) (9 points) The first order approximation is

   $$\begin{aligned} J(\mathbf{w}^1) - J(\mathbf{w}^0) &\approx \nabla J(\mathbf{w}^0) \cdot (\mathbf{w}^1 - \mathbf{w}^0) \\ &= \sum_{j=1}^{p} \frac{\partial J(\mathbf{w}^0)}{\partial w_j}(w_j^1 - w_j^0) = \delta \frac{\partial J(\mathbf{w}^0)}{\partial w_1}. \end{aligned}$$

   (d) (9 points) One possible implementation is given as:

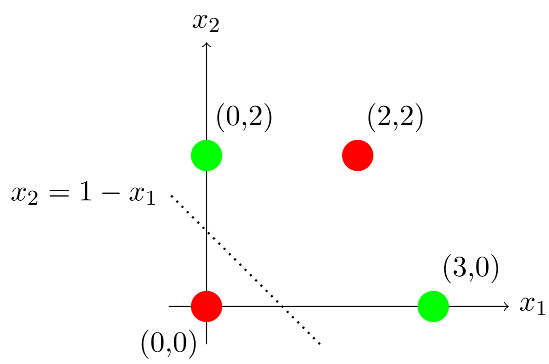

Figure 1: Scatter plot of the data points where the green circles are $y_i = 1$ and the red circles are $y_i = -1$. The dotted line is the boundary of a potential linear classifier.

```
def proj_grad(feval,alpha,nit,winit):
    w = winit
    for it in range(nit):
        J, Jgrad = feval(w)
        v = w - alpha*Jgrad
        w = np.maximum(0, v)
    return w, J
```

2. (33 points total)

   (a) (8 points) See Fig. 1.

   (b) (8 points) The data is not linearly separable. But, if we take the classifier with $z = x_1 + x_2 - 1$, this makes only one error. The weights and bias for this classifier is:

   $$\mathbf{w} = [1, 1], \quad b = -1.$$

   Of course, there are other classifiers that make only one error. Any such choice will receive full marks, but you must specify $\mathbf{w}$ and b and not simply draw the line.

   (c) (8 points) The point at $(2, 2)$ is misclassified by the classifier in part (b). Since this is the only misclassified point, it will have the highest hinge loss. For this point,

   $$z_i = w_1 x_{i1} + w_2 x_{i2} + b = (1)(2) + (1)(2) - 1 = 3.$$

   Hence, the hinge loss is

   $$\epsilon_i = \max\{0, 1 - y_i z_i\} = \max\{0, 1 - (-1)(3)\} = 4.$$

   (d) (9 points) A simple implementation is as follows:

```
def predict(x,xtr,ytr,alpha,gam):
    dsq = (x[0]-xtr[:,0])**2 + (x[1] - xtr[:,1])**2
    z = np.sum( ytr*alpha*np.exp(-gam*dsq) )
    yhat = (z > 0)
```

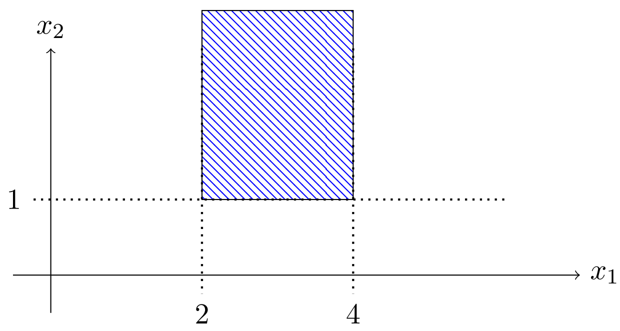

Figure 2: Region where $u^{\text{H}}_j = 1$ for all j is the described by the intersection of the three half-planes.

Note that the function requires the training data `xtr,ytr` as well as `alpha` and `gam`.

You will receive full credit for the above code. But, if you wanted the code to work for an arbitrary dimension d, you could compute the distance `dsq` with the following code which uses python broadcasting.

```
    dsq = np.sum((x[None,:] - xtr)**2, axis=1)
```

The other lines would remain the same.

3. (34 points total)

(a) (9 points) Since $\mathbf{z}^{\text{H}} = \mathbf{W}^{\text{H}}\mathbf{x} + \mathbf{b}^{\text{H}}$,

$$\mathbf{z}^{\text{H}} = \begin{bmatrix} z^{\text{H}}_1 \\ z^{\text{H}}_2 \\ z^{\text{H}}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} -2 \\ 4 \\ -1 \end{bmatrix} = \begin{bmatrix} x_1 - 2 \\ -x_1 + 4 \\ x_2 - 1 \end{bmatrix}.$$

For each hidden output j, $u^{\text{H}}_j = 1$ when $z^{\text{H}}_j > 0$. Therefore,

$$\begin{aligned} u^{\text{H}}_1 = 1 &\iff x_1 - 2 > 0 \iff x_1 > 2 \\ u^{\text{H}}_2 = 1 &\iff -x_1 + 4 > 0 \iff x_1 < 4 \\ u^{\text{H}}_3 = 1 &\iff x_2 - 1 > 0 \iff x_2 > 1. \end{aligned}$$

Thus, each region where $u^{\text{H}}_j = 1$ is a half-plane and their intersection is shown in Fig. 2.

(b) (9 points) Since $\mathbf{x} = (3, 2)$ is in the region in Fig. 2 we have $u^{\text{H}}_j = 1$ for all j, Therefore with $W^{\text{O}} = (1, 1, 1)$,

$$z^{\text{O}} = \sum_{j=1}^{3} W^{\text{O}}_j u^{\text{H}}_j + b^{\text{O}} = (1)(1) + (1)(1) + (1)(1) + b^{\text{O}} = 3 + b^{\text{O}}.$$

In order that $\hat{y} = 1$, we need $z^{\text{O}} > 0$ and therefore,

$$z^{\text{O}} > 0 \iff 3 + b^{\text{O}} > 0 \iff b^{\text{O}} > -3.$$

(c) (8 points) We have

$$z^{\mathrm{H}}_{ij} = \sum_{k=1}^{N_h} W^{\mathrm{H}}_{jk} x_{ik} + b^{\mathrm{H}}_j \Longrightarrow \frac{\partial z^{\mathrm{H}}_{ij}}{\partial W^{\mathrm{H}}_{jk}} = x_{ik}$$

By chain rule,

$$\frac{\partial J}{\partial W^{\mathrm{H}}_{jk}} = \sum_{i=1}^{N} \frac{\partial J}{\partial z^{\mathrm{H}}_{ij}} \frac{\partial z^{\mathrm{H}}_{ij}}{\partial W^{\mathrm{H}}_{jk}} = \sum_{i=1}^{N} \frac{\partial J}{\partial z^{\mathrm{H}}_{ij}} x_{ik}.$$

(d) (8 points) We assume that x_{ij} is represented in a matrix `x` and $\partial J / \partial z^{\mathrm{H}}_{ij}$ is represented in a matrix `Jgrad_zhid`. Then, we can perform the sum in part (c) as:

```
Jgrad_Whid = np.zeros((nhid,nin))
for j in range(nhid):
    for k in range(nin):
        Jgrad_Whid[j,k] = np.sum(zhid[:,j]*x[:,k])
```

This will get full credit. But, you can avoid the for loops using python broadcasting:

```
Jgrad_Whid = np.sum(zhid[:,:,None]*x[:,None,:], axis=0)
```