EL-GY 9123: Introduction to Machine Learning
Final, Fall 2017

Answer all FOUR questions. Exam is closed book. No electronic aids. But, you are permitted a limited number of cheat sheets. Part marks are given. If you do not remember a particular python command or its syntax, use pseudo-code and state what syntax you are assuming.

Best of luck!

1. You are given five data points x_i with binary class labels $y_i = \pm 1$:

x_i	-1	1	3	4	6
y_i	-1	1	-1	1	1

   (a) Find a linear classifier that makes a minimum number of errors on the training data points. Precisely describe how your classifier maps x to a prediction $\hat{y}$. What training points, if any, are incorrectly labeled by the classifier?

   (b) Now consider a classifier of the form,

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

   Using a kernel $K(x, x_i) = x x_i$, find values for $\boldsymbol{\alpha}$ and b such that (1) matches the classifier in part (a). In general, there are multiple solutions. Pick any $\boldsymbol{\alpha}$ and b that works.

   Hint: Select $\boldsymbol{\alpha}$ such that there is only one non-zero coefficient α_i.

   (c) Now consider (1) with a radial basis function kernel, $K(x, x_i) = e^{-\gamma(x - x_i)^2}$. Suppose that γ is very large. Find a vector $\boldsymbol{\alpha}$ and bias b such that this classifier makes no errors on the training data. For this choice of $\boldsymbol{\alpha}$ and b, approximately draw z vs. x over some suitable range of values x.

   (d) Write a python method, `predict`, that outputs $\hat{y}$ in (1) for the RBF kernel in part (c): Assume the input data is a vector of points `x` and the function should return `yhat`, a vector with the corresponding predictions.

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

   You will need to supply your method `predict` any arguments in addition to `x` that you will need. Your function should work for arbitrary training data, $\boldsymbol{\alpha}$, b, and γ.

2. Consider a neural network used for regression with a scalar input x and scalar target y,

$$z_j^{\mathrm{H}} = W_j^{\mathrm{H}} x + b_j^{\mathrm{H}}, \quad u_j^{\mathrm{H}} = \max\{0, z_j^{\mathrm{H}}\}, \quad j = 1, \ldots, N_h$$
$$z^{\mathrm{O}} = \sum_{k=1}^{N_h} W_k^{\mathrm{O}} u_k^{\mathrm{H}} + b^{\mathrm{O}}, \quad \hat{y} = g_{\mathrm{out}}(z^{\mathrm{O}}).$$

The hidden weights and biases are:

$$\mathbf{W}^{\mathrm{H}} = \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{b}^{\mathrm{H}} = \begin{bmatrix} -1 \\ 1 \\ -2 \end{bmatrix}.$$

(a) What is the number N_h of hidden units? For each $j = 1, \ldots, N_h$, draw u_j^{H} as a function of x over some suitable range of values x. You may draw them on one plot, or on multiple plots.

(b) Since the network is for regression, what output activation function $g_{\mathrm{out}}(z^{\mathrm{O}})$ would you choose? Given training data (x_i, y_i), $i = 1, \ldots, N$, what loss function would you use to train the network?

(c) Using the output activation and loss function selected in part (b), find output weights and bias, $\mathbf{W}^{\mathrm{O}}$, b^{O}, for the training data below. For your choice of parameters, draw $\hat{y}$ vs. x over some suitable range of values x. You should not need to do any elaborate minimization.

x_i	-2	-1	0	3	3.5
y_i	0	0	1	3	3

(d) Write a function `predict` to output $\hat{y}$ given a vector of inputs `x`. Assume `x` is a vector representing a batch of samples and `yhat` is a vector with the corresponding outputs. Use the activation function you selected in part (b), but your function should take the weights and biases for both layers as inputs. Clearly state any assumptions on the formats for the weights and biases. Also, to receive full credit, you must not use any for loops.

3. Consider a neural network-like that takes each input $\mathbf{x}$ and produces a prediction $\hat{y}$ given

$$\begin{aligned} z_j &= \sum_{k=1}^{N_i} W_{jk} x_k + b_j, \quad u_j = 1/(1+\exp(-z_j)), \quad j = 1, \ldots, M, \\ \hat{y} &= \frac{\sum_{j=1}^{M} a_j u_j}{\sum_{j=1}^{M} u_j}, \end{aligned} \tag{2}$$

where M is the number of hidden units and is fixed (i.e. not trainable). To train the model, we get training data $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$.

(a) Rewrite the equations (2) for the batch of inputs $\mathbf{x}_i$ from the training data. That is, correctly add the indexing i, to the equations.

(b) If we use a loss function,

$$L = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2,$$

draw the computation graph describing the mapping from $\mathbf{x}_i$ and parameters to L. Indicate which nodes are trainable parameters.

(c) Suppose that, in backpropagation, we have computed $\partial L / \partial \hat{y}_i$ for all i. Describe how to compute the components of the gradient $\partial L / \partial \mathbf{u}$.

(d) Write a few lines of python code to implement the gradients in part (c). Indicate how you represent the gradients. Full credit requires that you avoid for-loops.

4. Consider a convolutional layer of a neural network that takes an input tensor U and computes an output tensor Z via a 2D convolution,

$$Z[\ell, i, j, m] = \sum_{k_1} \sum_{k_2} \sum_{n} W[k_1, k_2, n, m] U[\ell, i + k_1, j + k_2, n] + b[m], \tag{3}$$

for some weight kernel $W[k_1, k_2, n, m]$ and bias $b[m]$. Suppose that:

- The input tensor U has 100 samples, each with size 200×300 pixels and 32 input channels.
- Each kernels in the filter W have shape 3×3.
- There are 64 output channels.
- The convolution is computed only on the *valid* pixels.

(a) What are the dimensions (i.e. shape) of U, Z and W?

(b) Suppose that for the output channel $m = 0$, we want $Z[\ell, i, j, m]$ to be a large positive value whenever there is a increase in input channel $n = 10$ going from left to right near the pixel location (i, j) in the $U[\ell, i, j, n]$. We also want the output channel $m = 0$ to have no dependence on input channels $n \neq 10$. What are possible values for $W[k_1, k_2, n, m]$ for $m = 0$? There is no single correct value, but justify your answer.

(c) Given the gradient components $\partial J / \partial Z[\ell, i, j, m]$, show how to compute the gradient components $\partial J / \partial U[\ell, i, j, n]$. You may use the relation,

$$Z[\ell, i', j', m] = \sum_{i} \sum_{j} \sum_{n} W[i - i', j - j', n, m] U[\ell, i, j, n] + b[m].$$

You do not need to prove this.

(d) Suppose that we are given a python function `convolve2d` that can implement the convolution (3) as,

```
Z = convolve2d(U,W,mode)
```

where `U` and `W` are `numpy` arrays and `mode=='valid'` or `'same'` depending on how we treat the boundary conditions. Also assume you have functions,

```
flip(X,axis)            # Flips X in axis
swapaxes(X,axis1,axis2) # Swaps axes axis1 and axis2
```

Write a few lines of python code to implement the gradient calculation in part (c) using these functions. You must explain your answer.

Hint: Re-index the summation in part (c) to try to rewrite the gradient $\partial J / \partial U$ as the output of a convolution of flips and swaps of W and $\partial J / \partial Z$.