CS5785 Homework 4

The homework is generally split into programming exercises and written exercises. This homework is due on **May 8, 2017 just before class**. Upload your homework to CMS. Please upload code as a single `.zip` file and the writeup as a single `.pdf` file. A complete submission should include:

1. A write-up as a single `.pdf` file
2. Source code and data files for all of your experiments (AND figures) in `.py` files if you use Python or `.ipynb` files if you use the IPython Notebook. If you use some other language, include all build scripts necessary to build and run your project along with instructions on how to compile and run your code.

The write-up should be in **professional lab report format.** It should contain a general summary of what you did, how well your solution works, any insights you found, etc. On the cover page, include the class name, homework number, and team member names. You are responsible for submitting clear, organized answers to the questions.
Please include all relevant information for a question, including text response, equations, figures, graphs, output, etc. If you include graphs, be sure to include the source code that generated them. Please pay attention to the discussion board for relevant information regarding updates, tips, and policy changes. You are encouraged (but not required) to work in groups of 2.

1 Written Exercises

1. **Maximum-margin classifiers** Suppose we are given $n = 7$ observations in $p = 2$ dimensions. For each observation, there is an associated class label.

X_1	X_2	Y
3	4	Red
2	2	Red
4	4	Red
1	4	Red
2	1	Blue
4	3	Blue
4	1	Blue

   a. Sketch the observations and the maximum-margin separating hyperplane.

   b. Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1, and β_2.

   c. On your sketch, indicate the margin for the maximal margin hyperplane.

d. Indicate the support vectors for the maximal margin classifier.

e. Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

f. Sketch a hyperplane that separates the data, but is not the maximum-margin separating hyperplane. Provide the equation for this hyperplane.

g. Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

2. This problem involves the OJ data set which is available at
`https://gist.github.com/gcr/e86ca41c43accbaed32226cc63af14e7`
(click the "Raw" button to download).

a. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations. Report the class fractions in your training and test sets.

b. Fit a support vector classifier to the training data using `C = 0.01`, with `Purchase` as the response and the other variables as predictors. Describe the classifier. What are the learned coefficients?

c. What are the training and test error rates?

d. Use cross validation to select an optimal cost (`C`) and report a CV error plot. Consider values in the range 0.01 to 10. What `C` is the best using the one-stderr rule?

e. Compute the training and test error rates using this new value for cost.

f. For a support vector classifier with a radial bassis function (RBF) kernel $k(x, x') = e^{-\gamma\|x-x'\|}$, why should we consider a *smaller* γ value as corresponding to a simpler model?
(Note that sometimes we write the RBF kernel as $k(x, x') = e^{-\|x-x'\|/\sigma^2}$ using $\sigma = 1/\sqrt{\gamma}$.)

g. Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use cross-validation (with the one-standard-error rule) to select an appropriate `gamma` .

h. Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set `degree=2` and use cross validation again to choose an appropriate `gamma`.

i. Overall, which approach seems to give the best results on this data?

3. **Neural networks as function approximators.** Design a feed-forward neural network to approximate the 1-dimensional function given in Fig. 1 on the following page. The output should match exactly. How many hidden layers do you need? How many units are there within each layer? Show the hidden layers, units, connections, weights, and biases.

   Use the ReLU nonlinearity for every unit. Every possible path from input to output must pass through the same number of layers. This means each layer should have the form

$$Y_i = \sigma(W_i Y_{i-1}^T + \beta_i), \tag{1}$$

   where $Y_i \in \mathbb{R}^{d_i \times 1}$ is the output of the ith layer, $W_i \in \mathbb{R}^{d_i \times d_{i-1}}$ is the weight matrix for that layer, $Y_0 = x \in \mathbb{R}^{1\times 1}$, and the ReLU nonlinearity is defined as

$$\sigma(x) \triangleq \begin{cases} x & x \geq 0, \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

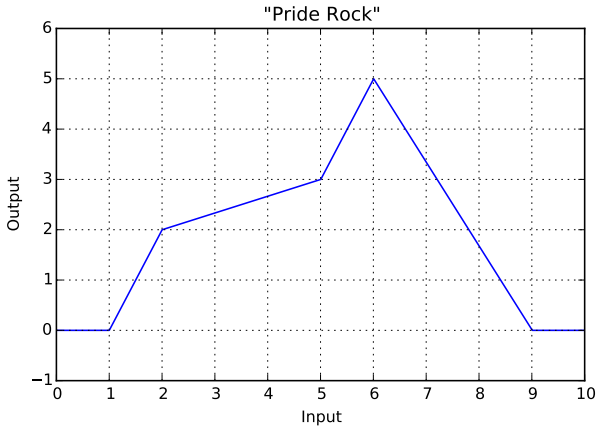

Figure 1: Example function to approximate using a neural network.

Hint 1: Try to express the input function as a sum of weighted and scaled ReLU units. Once you can do this, it should be straightforward to turn your final equation into a series of neural network layers with individual weights and biases.

Hint 2: If you're not convinced neural networks can approximate this function, see `http://neuralnetworksanddeeplearning.com/chap4.html` for an example of the flavor of solution we're looking for. (If you rely on this source, cite it!) However, keep in mind that your output must match the given input *exactly*.

4. **Approximating images with neural networks.** In this question, you will implement your own neural network toolkit. You will be writing your own implementation from scratch, using *C++* and CUDA. You should calculate the derivatives of each layer by hand using pencil and paper. Please attach a scan of your paper notes to the homework.

Just kidding. We're not that mean. There are several good convolutional neural network packages that have done the heavy lifting for us. One of the more interesting (and well-written!) demos is called ConvNetJS. It is implemented in Javascript and runs in a modern web browser without any dependencies.

Take a look at convnet.js's "Image Painting" demo at: `http://cs.stanford.edu/people/karpathy/convnetjs/demo/image_regression.html`

a. **Describe the structure of the network.** How many layers does this network have? What is the purpose of each layer?

b. What does "Loss" mean here? What is the actual loss function? You may need to consult the source code, which is available on Github.

c. Plot the loss over time, after letting it run for 5,000 iterations. How good does the network eventually get?

d. Can you make the network converge to a lower loss function by lowering the learning rate every 1,000 iterations? (Some *learning rate schedules*, for example, halve the learning rate every n iterations. Does this technique let the network converge to a lower training loss?)

e. **Lesion study.** The text box contains a small snippet of Javascript code that initializes the network. You can change the network structure by clicking the "Reload network" button, which simply evaluates the code. Let's perform some brain surgery: Try commenting out each layer, one by one. Report some results: How many layers can you drop before the accuracy drops below a useful value? How few hidden units can you get away with before quality drops noticeably?

f. Try adding a few layers by copy+pasting lines in the network definition. Can you noticeably increase the accuracy of the network?

5. **Random forests for image approximation** In this question, you will use random forest regression to approximate an image by learning a function, $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, that takes *image* (x, y) *coordinates* as input and outputs *pixel brightness*. This way, the function learns to approximate areas of the image that it has not seen before.

a. Start with an image of the Mona Lisa. If you don't like the Mona Lisa, pick another interesting image of your choice.

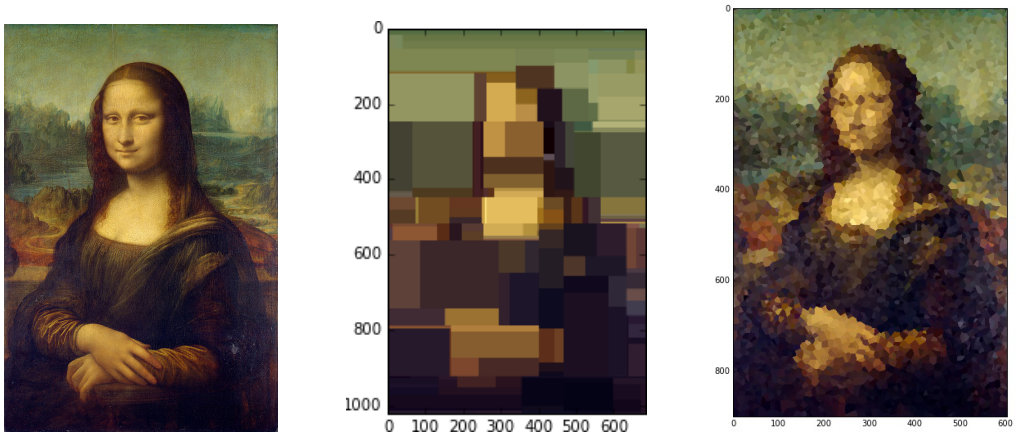

Figure 2: **Left**: `http://tinyurl.com/mona-lisa-small` *Mona Lisa*, Leonardo da Vinci, via Wikipedia. Licensed under Public Domain. **Middle**: Example output of a decision tree regressor. The input is a "feature vector" containing the (x, y) coordinates of the pixel. The output at each point is an (r, g, b) tuple. This tree has a depth of 7. **Right**: Example output of a k-NN regressor, where $k = 1$. The output at each pixel is equal to its closest sample from the training set.

b. **Preprocessing the input.** To build your "training set," uniformly sample 5,000 random (x, y) coordinate locations.

- What other preprocessing steps are necessary for random forests inputs? Describe them, implement them, and justify your decisions. In particular, do you need to perform mean subtraction, standardization, or unit-normalization?

c. **Preprocessing the output.** Sample pixel values at each of the given coordinate locations. Each pixel contains red, green, and blue intensity values, so decide how you want to handle this. There are several options available to you:

- Convert the image to grayscale
- Regress all three values at once, so your function maps (x, y) coordinates to (r, g, b) values: $f : \mathbb{R}^2 \to \mathbb{R}^3$
- Learn a different function for each channel, $f_{Red} : \mathbb{R}^2 \to \mathbb{R}$, and likewise for f_{Green}, f_{Blue}.

d. Rescale the pixel intensities to lie between 0.0 and 1.0. (The default for pixel values may be between 0 and 255, but your image library may have different defaults.)

e. What other preprocessing steps are necessary for random regression forest outputs? Describe them, implement them, and justify your decisions.

f. To build the final image, for each pixel of the output, feed the pixel coordinate through the random forest and color the resulting pixel with the output prediction. You can then use `imshow` to view the result. (If you are using grayscale, try `imshow(Y, cmap='gray')` to avoid fake-coloring). You may use any implementation of random forests, but you should understand the implementation and you must cite your sources.

g. Experimentation.

i. Repeat the experiment for a random forest containing a single decision tree, but with depths 1, 2, 3, 5, 10, and 15. How does depth impact the result? Describe in detail why.

ii. Repeat the experiment for a random forest of depth 7, but with number of trees equal to 1, 3, 5, 10, and 100. How does the number of trees impact the result? Describe in detail why.

iii. As a simple baseline, repeat the experiment using a k-NN regressor, for $k = 1$. This means that every pixel in the output will equal the nearest pixel from the "training set." Compare and contrast the outlook: why does this look the way it does?

iv. Experiment with different pruning strategies of your choice.

h. Analysis.

i. What is the decision rule at each split point? Write down the 1-line formula for the split point at the root node for one the trained decision trees inside the forest. Feel free to define any variables you need.

ii. Why does the resulting image look like the way it does? What shape are the patches of color, and how are they arranged?

iii. *Easy:* How many patches of color may be in the resulting image if the forest contains a single decision tree? Define any variables you need.

iv. *Tricky:* How many patches of color might be in the resulting image if the forest contains n decision trees? Define any variables you need.