New York University Tandon School of Engineering
Computer Science and Engineering

CS-UY 4563: Written Homework 4.
Due Monday, April 20th, 2020, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list the names of any collaborators at the top of your solution set, or write "No Collaborators" if you worked alone.

Problem 1: Kernels for Shifted Images (20pts)

In class we discussed why the Gaussian kernel is a better similarity metric for MNIST digits than the inner product. Here we consider an additional modification to the Gaussian kernel.

For illustration purposes we consider 5x5 black and white images: a pixel has value 1 if it is white and value 0 if it is black. For example, consider the following images of two 0s and two 1s:

$$I_1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad I_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad I_3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad I_4 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(a) Let $\vec{x}_i \in \{0,1\}^{25}$ denote the vectorized version of image I_i, obtained by concatenating the rows of the matrix representation of the image into a vector. Compute a 4×4 kernel matrix $\mathbf{K}$ for images $I_1, \ldots, I_4$ using the standard Gaussian kernel $k_G(I_i, I_j) = e^{-\|\vec{x}_i - \vec{x}_j\|_2^2}$.

Students can compute $\mathbf{K}$ by hand. I used a little MATLAB script, which you can find under `image_calcs.m`. My final result for $\mathbf{K}$ was:

$$\begin{bmatrix} 1 & .0009 & .0003 & .0067 \\ .0009 & 1 & .0067 & .0025 \\ .0003 & .0067 & 1 & .0000 \\ .0067 & .0025 & .0000 & 1 \end{bmatrix}$$

(b) Suppose I_1 and I_2 are in our training data and I_3 and I_4 are in our test data. Which training image is most similar to each of our test images according to Gaussian kernel similarity? Do you expect a kernel classifier (k-NN, kernel logistic regression, etc.) to correctly or incorrectly classify I_3 and I_4?

Based on the kernel matrix above, I_3 is most similar to I_2 and I_4 is most similar to I_1. Therefore we expect most methods would *incorrectly* classify I_3 as a 1 and I_4 as a 0.

(c) Consider a "left-right shift" kernel, which is a similarity measure defined as follows:

For an image I_i, let I_i^{right} be the image with its far right column removed and let I_i^{left} be the image with its far left column removed. Intuitively, I_i^{right} corresponds to the image shifted one pixel to the right and I_i^{left} corresponds to the image shifted one pixel left. Define a new similarity metric k_{shift} as follows:

$$k_{\text{shift}}(I_i, I_j) = k_G(I_i^{\text{right}}, I_j^{\text{right}}) + k_G(I_i^{\text{left}}, I_j^{\text{left}}) + k_G(I_i^{\text{right}}, I_j^{\text{left}}) + k_G(I_i^{\text{left}}, I_j^{\text{right}})$$

Intuitively this kernel captures similarity between images which are similar *after a shift*, something the standard Gaussian kernel does not account for.

Recompute the a 4×4 kernel matrix $\mathbf{K}$ for images $I_1, \ldots, I_4$ using k_{shift}.

Again my code is in `image_calcs.m`. My final result for $\mathbf{K}_{\text{shift}}$ was:

$$\begin{bmatrix} 2.0007 & .0153 & 1.0080 & .0144 \\ .0153 & 2.0050 & .1433 & 1.0074 \\ 1.0080 & .1433 & 2.0135 & .0074 \\ .0144 & 1.0074 & .0074 & 2.0050 \end{bmatrix}$$

(d) Again I_1 and I_2 were in our training data and I_3 and I_4 were in our test data. Now which training image is most similar to each of our test images according to the "left-right shift" kernel? Do you expect a typically kernel classifier to correctly or incorrectly classify I_3 and I_4?

Based on the kernel matrix above, I_3 is most similar to I_1 and I_4 is most similar to I_2. Therefore we expect most methods would *correctly* classify I_3 as a 0 and I_4 as a 1.

(e) Prove that k_{shift} is a positive semi-definite kernel function. **Hint:** Use the fact that k_G is positive semi-definite.

Since k_G is PSD, there exists some feature transformation ϕ such that $k_G(I_a, I_b) = \phi(I_a)^T\phi(I_b)$ for any images I_a, I_b. Now, for any image I, let $\tilde{\phi}(I) = \phi(I^{\text{right}}) + \phi(I^{\text{left}})$. Then:

$$\begin{aligned}\tilde{\phi}(I_a)^T\tilde{\phi}(I_b) &= (\phi(I_a^{\text{right}}) + \phi(I_a^{\text{left}}))^T(\phi(I_b^{\text{right}}) + \phi(I_b^{\text{left}})) \\ &= k_G(I_a^{\text{right}}, I_b^{\text{right}}) + k_G(I_a^{\text{right}}, I_b^{\text{left}} + k_G(I_a^{\text{left}}, I_b^{\text{right}}) + k_G(I_a^{\text{left}}, I_b^{\text{left}}) \\ &= K_{\text{shift}}(I_a, I_b).\end{aligned}$$

By the demonstration above, $K_{\text{shift}}(I_a, I_b)$ can be written as an inner product after feature transformation $\tilde{\phi}$, and thus it's positive semi-definite.

Problem 2: Thinking About Margin (10pts)

Consider the data set of four examples $\vec{x}_1, \ldots \vec{x}_4$ with two features each: $\vec{x}_i = (\vec{x}_i[1], \vec{x}_i[2])$. Each data example has a binary class label $y_i = \pm 1$.

$x_i[1]$	0	1	1	2
$x_i[2]$	0	0.3	0.7	1
y_i	-1	-1	1	1

(a) Find a linear classifier that separates the two classes. Your classifier f should be of the form:

$$f(\vec{x}) = \begin{cases} 1 & \text{if } b + w_1\vec{x}[1] + w_2\vec{x}[2] > 0 \\ -1 & \text{if } b + w_1\vec{x}[1] + w_2\vec{x}[2] \leq 0 \end{cases}$$

State the intercept b and weights w_1 and w_2 for your classifier. Note there is no unique correct answer, as there are multiple linear classifiers that could separate the classes.

Lots of different options here would work. One simple possibility is $w_1 = 0$, $w_2 = 1$, $b = -.5$.

(b) For the classifier you found in part (a), what is the maximum γ such that

$$y_i(b + w_1x_{i1} + w_{i2}x_{i2}) \geq \gamma, \text{ for all } i?$$

For each of our data points, we can explicitly compute $y_i(b + w_1x_{i1} + w_{i2}x_{i2})$. We get: $.5, .2, .2, .5$. So the maximum value of γ is .2.

(c) Compute the margin of your classifier from part (a). The margin m is the minimum perpendicular distance from any point to the separating hyperplane defined by a linear classifier. Mathematically it can be computed as:

$$m = \frac{\gamma}{\|\vec{w}\|_2},$$

where $\|w\|_2 = \sqrt{w_1^2 + w_2^2}$.

For my classifier $\|\vec{w}\|_2 = 1$, so $m = \gamma = .2$.

(d) Which data examples are on the margin for your classifier? I.e. for which i does $\vec{x}_i$ lie exactly distant m from your chosen separating hyperplane?

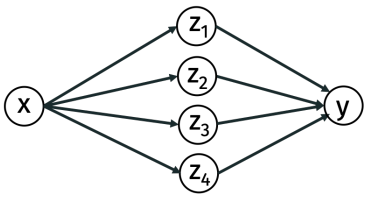
Both $\vec{x}_2$ and $\vec{x}_3$ are on the margin.

(e) Is your classifier a maximum margin classifier, or does there exists a separating hyperplane with larger margin? Justify your answer briefly (in words or math). To answer this question, it might be helpful to plot the dataset and your classification rule.

No. If you look at a plot, it's pretty clear that the margin of my hyperplane could be reduce by rotating the plane slightly around the point $(1, .5)$ in a counter clockwise direction.

Problem 3: Neural Networks for Curve Fitting (15pts)

Consider the following 2-layer, feed forward neural network for single variate regression:

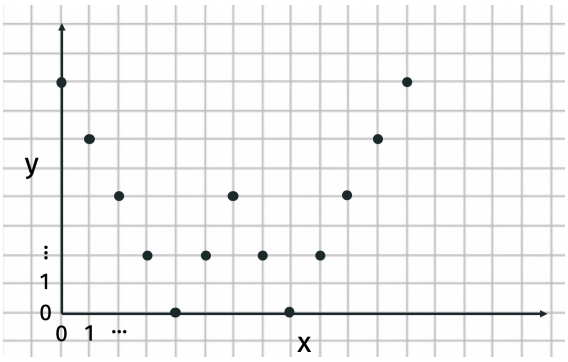


Let $W_{H,1}, W_{H,2}, W_{H,3}, W_{H,4}$ and $b_{H,1}, b_{H,2}, b_{H,3}, b_{H,4}$ be weights and biases for the hidden layer. Let $W_{O,1}, W_{O,2}$, $W_{O,3}, W_{O,4}$ and b_O be weights and bias for the output layer. The hidden layer uses rectified linear unit (ReLU) non-linearities and the output layer uses no non-linearity.

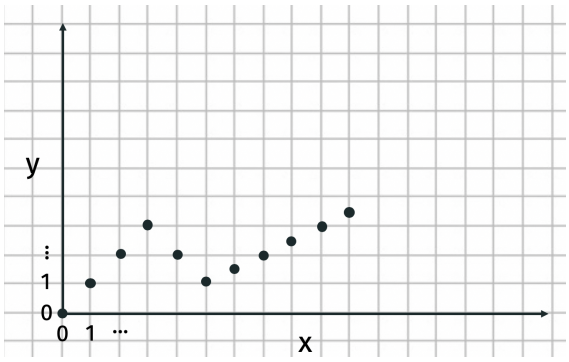
Specifically, for $i = 1, \ldots, 4$, $z_i = \max(0, \bar{z}_i)$ where $\bar{z}_i = W_{H,i}x + b_{H,i}$. And

$$y = b_O + \sum_{i=1}^{4} W_{O,i} z_i.$$

(a) For each of the two datasets below, determine values for weights and biases which would allow this network to perfectly fit the data.



Dataset 1



Dataset 2

Encourage them to think about what features would allow them to fit these datasets. They might want to look back to their previous homework at the piecewise regression example. There are a lot of numbers that work. One set of solutions is as follows:

For Dataset 1:

$W_{H,1} = W_{H,2} = W_{H,3} = W_{H,4} = 1$. $b_{H,1} = 0$, $b_{H,2} = -4$, $b_{H,3} = -6$, $b_{H,4} = -8$.

$W_{O,1} = -2$, $W_{O,2} = 4$, $W_{O,3} = -4$. $W_{O,4} = 4$, $b_O = 8$.

For Dataset 2:

For this dataset 3 hidden features would have sufficed: by setting $W_{O,4} = 0$ we're ignoring z_4.

$W_{H,1} = W_{H,2} = W_{H,3} = 1$. $W_{H,4} =$ arbitrary, $b_{H,1} = 0$, $b_{H,2} = -3$, $b_{H,3} = -5$, $b_{H,4} =$ arbitrary.

$W_{O,1} = 1$, $W_{O,2} = -2$, $W_{O,3} = 1.5$. $W_{O,4} = 0$, $b_O = 0$.

(b) For input parameters $\vec{\theta}$ let $f(x, \vec{\theta})$ denote the output of the neural network for a given input x. We want to train the network under the squared loss. Specifically, given a training dataset $(x_1, y_1), \ldots, (x_n, y_n)$, we want to choose $\vec{\theta}$ to minimize the loss:

$$\mathcal{L}(\vec{\theta}) = \sum_{i=1}^{n} (y_i - f(x_i, \vec{\theta}))^2.$$

Write down an expression for the gradient $\nabla\mathcal{L}(\vec{\theta})$ in terms of $\nabla f(x, \vec{\theta})$. **Hint:** Use chain rule.

$$\begin{aligned}\nabla\mathcal{L}(\vec{\theta}) &= \sum_{i=1}^{n} \nabla(y_i - f(x_i, \vec{\theta}))^2 \\ &= \sum_{i=1}^{n} 2(y_i - f(x_i, \vec{\theta})) \cdot \nabla f(x_i, \vec{\theta})\end{aligned}$$

(c) Suppose we randomly initialize the network with ± 1 random numbers:

$$\begin{aligned} W_{H,1} = -1, W_{H,2} = 1, W_{H,3} = 1, W_{H,4} = -1 \\ b_{H,1} = 1, b_{H,2} = 1, b_{H,3} = -1, b_{H,4} = 1 \\ W_{O,1} = -1, W_{O,2} = -1, W_{O,3} = -1, W_{O,4} = 1 \\ b_O = 1 \end{aligned}$$

Call this initial set of parameter $\vec{\theta}_0$. Use forward-propagation to compute $f(x, \vec{\theta}_0)$ for $x = 2$.

First we compute:

$$\begin{aligned} \bar{z}_1 &= -1 & z_1 &= 0 \\ \bar{z}_2 &= 3 & z_2 &= 3 \\ \bar{z}_3 &= 1 & z_3 &= 1 \\ \bar{z}_4 &= -1 & z_4 &= 0 \end{aligned}$$

And then we see that $y = f(x, \vec{\theta}_0) = -3$.

(d) Use back-propagation to compute $\nabla f(x, \vec{\theta}_0)$ for $x = 2$. To do the computation you will need to use the derivative of the ReLU function, $\max(0, z)$. You can simply use:

$$\frac{\partial}{\partial z} \max(0, z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$$

This derivative is discontinuous, but it turns out that is fine for use in gradient descent.

First we compute derivatives for the last layer of weights:

$$\begin{aligned} \frac{\partial f}{b_O} &= 1 \\ \frac{\partial f}{W_{O,1}} &= z_1 = 0 \\ \frac{\partial f}{W_{O,2}} &= z_2 = 3 \\ \frac{\partial f}{W_{O,3}} &= z_3 = 1 \\ \frac{\partial f}{W_{O,4}} &= z_4 = 0 \end{aligned}$$

Then for the hidden layer of nodes:

$$\frac{\partial f}{\partial z_1} = W_{O,1} = -1 \qquad \frac{\partial f}{\partial \bar{z}_1} = -1 \cdot \frac{\partial z_1}{\partial \bar{z}_1} = 0$$
$$\frac{\partial f}{\partial z_2} = W_{O,2} = -1 \qquad \frac{\partial f}{\partial \bar{z}_2} = -1 \cdot \frac{\partial z_2}{\partial \bar{z}_2} = -1$$
$$\frac{\partial f}{\partial z_3} = W_{O,3} = -1 \qquad \frac{\partial f}{\partial \bar{z}_3} = -1 \cdot \frac{\partial z_3}{\partial \bar{z}_3} = -1$$
$$\frac{\partial f}{\partial z_4} = W_{O,4} = 1 \qquad \frac{\partial f}{\partial \bar{z}_4} = 1 \cdot \frac{\partial z_4}{\partial \bar{z}_4} = 0$$

Then for the first layer of weights:

$$\frac{\partial f}{b_{H,1}} = \frac{\partial f}{\partial \bar{z}_1} = 0 \qquad \frac{\partial f}{W_{H,1}} = x \cdot \frac{\partial f}{\partial \bar{z}_1} = 0$$
$$\frac{\partial f}{b_{H,2}} = \frac{\partial f}{\partial \bar{z}_2} = -1 \qquad \frac{\partial f}{W_{H,2}} = x \cdot \frac{\partial f}{\partial \bar{z}_2} = -2$$
$$\frac{\partial f}{b_{H,3}} = \frac{\partial f}{\partial \bar{z}_3} = -1 \qquad \frac{\partial f}{W_{H,3}} = x \cdot \frac{\partial f}{\partial \bar{z}_3} = -2$$
$$\frac{\partial f}{b_{H,4}} = \frac{\partial f}{\partial \bar{z}_4} = 0 \qquad \frac{\partial f}{W_{H,4}} = x \cdot \frac{\partial f}{\partial \bar{z}_4} = 0$$