University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 411 - Machine Learning and Data Mining, Fall 2018

Assignment 3

Due date: Wednesday December 5, 11:59pm.
No late assignments will be accepted.

As in all work in this course, 20% of your grade is for quality of presentation, including the use of good English, properly commented and easy-to-understand programs, and clear proofs. In general, short, simple answers are worth more than long, complicated ones. Unless stated otherwise, all answers should be justified. The TA has a limited amount of time to devote to each assignment, so what you hand in should be legible (either typed or *neatly* hand-written), well-organized and easy to evaluate. (An employer would demand no less.) All computer problems are to be done in Python with the NumPy, SciPy and scikit-learn libraries.

Hand in five files: The source code of all your programs (functions and script) in a single Python file, a pdf file of figures generated by the progams, a pdf file of all printed output, a pdf file of answers to all the non-programming questions (scanned hand-writing is fine, but *not* photographs), and a scanned, signed copy of the cover sheet at the end of the assignment. Be sure to indicate clearly which question(s) each program and piece of output refers to. All the Python code (functions and script) for a given question should appear in one location in your source file, along with a comment giving the question number. All material in all files should appear in order; *i.e.*, material for Question 1 before Question 2 before Question 3, etc. It should be easy for the TA to identify the material for each question. In particular, all figures should be titled, and all printed output should be identified with the Question number. The five files should be submitted electronically as described on the course web page. In addition, if we run your source file, it should not produce any errors, it should produce all the output that you hand in (figures and print outs), and it should be clear which question each piece of output refers to. *Output that is not labled with the Question number will not be graded. Programs that do not produce output will not be graded.*

Style: Use the solutions to Assignments 1 and 2 as a guide/model for how to present your solutions to Assignment 3.

I don't know policy: If you do not know the answer to a question (or part), and you write "I don't know", you will receive 20% of the marks of that question (or part). If you just leave a question blank with no such statement, you get 0 marks for that question.

No more questions will be added.

Tips on Scientific Programming in Python

If you haven't already done so, please read the NumPy tutorial on the course web page. Be sure to read about indexing and slicing Numpy arrays. First, indexing begins at 0, not 1. Thus, if `A` is a matrix, then `A[7,0]` is the element in row 7 and column 0. Likewise, `A[0,4]` is the element in row 0 and column 4. Slicing allows large segments of an array to be referenced. For example, `A[:,5]` returns column 5 of matrix `A`, and `A[7,[3,6,8]]` returns elements 3, 6 and 8 of row 7. Similarly, if `v` is a vector, then the statement `A[6,:]=v` copies `v` into row 6 of matrix `A`. Note that if `A` and `B` are two-dimensional Numpy arrays, then `A*B` performs *element-wise* multiplication, *not* matrix multiplication. To perform matrix multiplication, you should use `numpy.matmul(A,B)`. Whenever possible, *do not use loops*, which are very slow in Python. In particular, avoid iterating over the elements of a large vector or matrix. Instead, use numpy's vector and matrix operations, which are much faster and can be executed in parallel. For example, if `A` is a matrix and `v` is a column vector, the `A+v` will add `v` to every column of `A`. Likewise for rows and row vectors. Also, the functions `sum` and `mean` in `numpy` are useful for summing or averaging over all or part of an array. Many NumPy functions that are defined for single numbers can be passed lists, vectors and matrices instead. For example, $f([x_1, x_2, ..., x_n])$ returns the list $[f(x_1), f(x_2), ..., f(x_n)]$. The same is true for many user-defined functions. The term `numpy.inf` represents infinity. It results from dividing by 0 in numpy. It can also result from overflow (*i.e.*, from numbers that are too large to represent in the computer, like 10^{1000}). The term `numpy.nan` stands for "not a number", and it results from doing 0/0, inf/inf or inf-inf in numpy. For generating and labelling plots, the following SciPy functions in `matplotlib.pyplot` are needed: `plot`, `xlabel`, `ylabel`, `title`, `suptitle` and `figure`. You can use Google to conveniently look up SciPy functions. e.g., you can google "numpy matmul" and "pyplot suptitle".

1. (38 points total) *Neural Networks: decision boundaries.*

   This warm-up exercise introduces the process of defining and training a neural network and illustrates the non-linear decision boundaries they produce. You will use the two-dimensional data that you generated in Question 1 of Assignment 2. To train a neural net, you will use the class `MLPclassifier` in `sklearn.neural_network`. It generates a Python object, much as the function `LogisticRegression` did in Question 2 of Assignment 2. To plot the decision boundary, you will need the function `dfContour` from `bonnerlib2`, which you can download from the course web site.

   To keep things simple, this assignment will only use neural nets with a single hidden layer. You should also set your neural nets to use stochastic gradient descent (sgd) as the optimization method, and `tanh` for the hidden-layer activation function. Set the initial learning rate to 0.01, the tolerance for optimization to 10^{-10}, and the maximum number of iterations to 10,000.

   In answering the questions below, you will need to use the methods and attributes of *MLPclassifier*.

   (a) (0 points) Using the function `genData` that you wrote in Question 1(a) of Assignment 2, generate a training set with 1,000 points per cluster, and a test set with 10,000 points per cluster. Use the same cluster centres and covariance matrices as in Question 1(b) of Assignment 2.

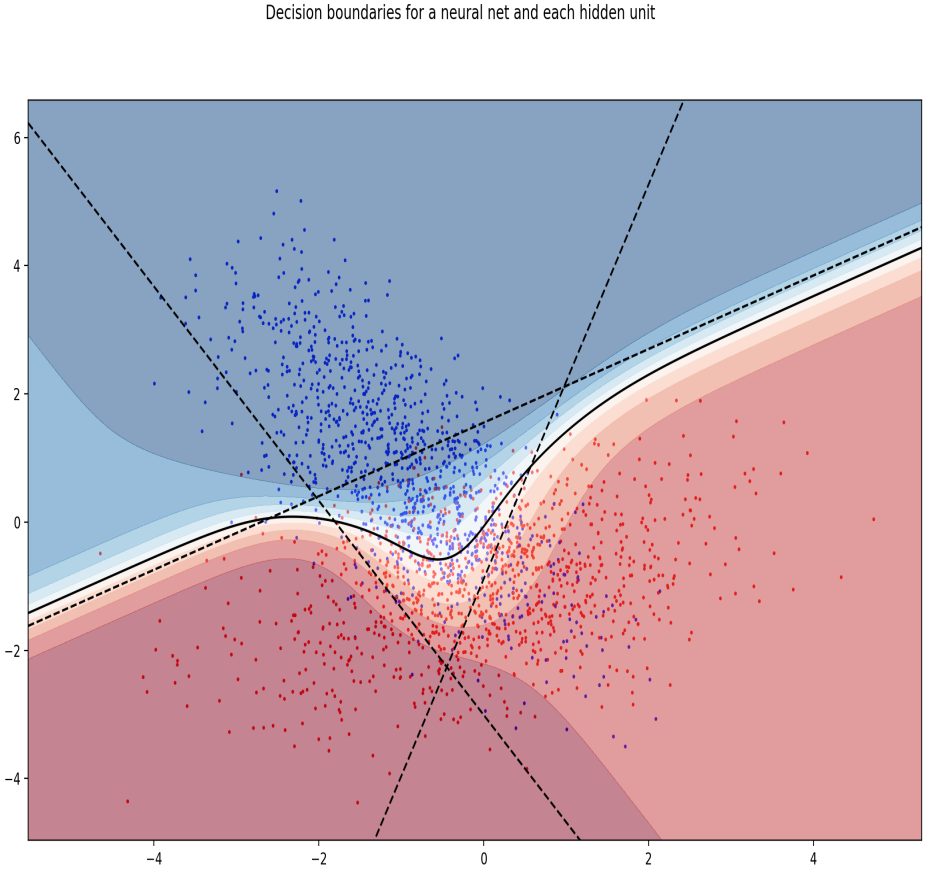
   (b) (5 points) Train a neural net with one unit in the hidden layer. Plot the training data, and draw the decision boundary on top of the training data using `dfContour`.[1] (Be sure to plot the training data *before* you call `dfContour`.) Title the figure, "Question 1(b): Neural net with 1 hidden unit." It should look very similar to the decision boundary in Question 2(c) of Assignment 2. *Explain this similarity.*

   (c) (3 points) Train a neural net with two units in the hidden layer, compute and print the test accuracy, plot the training data, and draw the decision boundary on top of the training data. Do this nine times. You should observe nine different decision boundaries, some straight and some crooked. Arrange the nine plots in a 3×3 grid in a single figure. Title the figure, "Question 1(c): Neural nets with 2 hidden units." In a separate figure, plot the training data and decision boundary of the neural net with the highest test accuracy. Title the figure, "Question 1(c): Best neural net with 2 hidden units." Print the accuracy. It should be greater than the accuracy given in the solutions to Question 2(b) of Assignment 2. (If not, run your procedures again.)

   (d) (3 points) Repeat part (c) for neural nets with three hidden units. The best test accuracy here should be greater than in part (c). (If not, run your procedures again.) Title the figures appropriately.

[1] `dfContour` plots the decision boundary as a black line and plots several other contours of the function defined by the neural net. The colour of a region represents the predicted probability that a point is blue or red. The darker the blue, the greater the probability of blue. The darker the red, the greater the probability of red.

Figure 1:



(e) (3 points) Repeat part (c) for neural nets with four hidden units. The best test accuracy here should be greater than in part (d). (If not, run your procedures again.) Title the figures appropriately.

(f) (3 points) Explain why you get so many different decision boundaries in parts (c) and (d).

(g) (3 points) Recall that each hidden unit in a neural net is a logistic-regression unit. For the best neural net in part (d), plot the decision boundaries for each of these logistic-regression units. Specifically, plot the training data, plot the decision boundary of the whole neural net, and plot the decision boundaries for each hidden unit as a black dashed line. Set limits on the x and y axis so that the training data fills the figure without a lot of empty space. The result should look something like Figure 1. Title the figure, 'Question 1(g): Decision boundaries for 3 hidden units'.

(h) (3 points) Repeat part (g) for the best neural net in part (c). Title the figure, 'Question 1(h): Decision boundaries for 2 hidden units'.

(i) (3 points) Repeat part (g) for the best neural net in part (e). Title the figure,

'Question 1(i): Decision boundaries for 4 hidden units'.

(j) (4 points) Using your figure in part (g), give a geometrical explanation of how the hidden units contribute to the decision boundary of the neural net. Do not give a general explanation. Instead, explain the specific decision boundary generated by the neural net in part (g).

(k) (3 points) Generate a precision/recall curve for the best neural net in part (d). Recall that precision and recall are defined in terms of predicted positives. The input to a neural net is a predicted positive if the output exceeds a given threshold. Specifically, if x is an input and $p(x)$ is the output, then x is a predicted positive if $p(x) > t$, where t is the threshold. Since this is a classification problem, $p(x)$ is a probability (the predicted probability that x is in class 1). Thus $0 \leq p(x) \leq 1$, so you should generate the precision/recall curve using 1,000 values of t equally spaced between 0 and 1. Title the figure. 'Question 1(k): Precision/recall curve'.

You will probably get division-by-zero warnings when trying to compute the precision when t is very close to 1. In this case, use a reasonable value for the precision, *i.e.*, a value that makes the precision/recall curve look reasonable.

(l) (5 points) Compute and print the area under the precision/recall curve in part (k). It should be greater than the area given in the solutions to Question 2(k) of Assignment 2. *Give a simple explanation of why the area here should be greater.*

2. (45 points total) *Neural Networks: theory*

Consider the neural networks of Question 1. They had one hidden layer using a tanh activation function, and a single output using a sigmoid activation function. Using the notation of slide 31 of Lecture 10, the operation of the neural net can be described as follows:

$$o = sigmoid(z) \qquad z = hW + w_0 \qquad h = tanh(u) \qquad u = xV + v_0 \qquad (1)$$

Here x is the input (a row vector) and o is the output (a real number). V is a matrix of weights, v_0 is a row vector of bias terms, W is a column vector of weights and w_0 is a bias term (a real number).

The loss function is given by the cross entropy:

$$C = \sum_n c(t^{(n)}, o^{(n)}) \quad \text{where} \quad c(t, o) = -t \log o - (1 - t) \log(1 - o)$$

where the sum is over training points, $t^{(n)}$ is the target value for input $x^{(n)}$, and $o^{(n)}$ is the output.

In the following questions, you may use the results given on slide 21 of Lecture 10. To make your proofs easier to mark, use the index n to range over training instances, use j to range over hidden units, and use i and k to range over features of an input vector, x.

(a) (10 points) Prove that $\partial c(t, o)/\partial W_j = (o - t)h_j$

(b) (10 points) Prove that $\partial c(t,o)/\partial V_{ij} = (o-t)x_i w_j / \cosh^2 u_j$

(c) (4 points) Give an equation for $\partial c(t,o)/\partial w_0$ and a brief justification.

(d) (4 points) Give an equation for $\partial c(t,o)/\partial v_{0j}$ and a brief justification. Here, v_{0j} is the j^{th} component of v_0.

(e) (5 points) Express $\partial C/\partial W$ as matrix-vector multiplication. Give a proof.

(f) (6 points) Express $\partial C/\partial V$ as matrix-matrix multiplication. Give a proof.

(g) (4 points) Express $\partial C/\partial v_0$ as matrix-vector multiplication. Give a brief justification.

(h) (2 points) Write down a formula for $\partial C/\partial w_0$. Give a brief justification.

3. (42 points total) *Neural Networks: implementation.*

In this question, you will use the theory developed in Question 2 to train a neural net using gradient descent. You will test your implementation on the data from the same distribution as in Question 1. Do not use any loops in computing the gradients.

Using the function `genData` from Question 1(a) of Assignment 2, generate a training set with 10,000 points per cluster, and a test set with 10,000 points per cluster. Use the same cluster centres and covariance matrices as in Question 1(b) of Assignment 2. Use this data in the rest of this question.

(a) (7 points) *Forward Inference.*

Write a Python function `forward(X,V,v0,W,w0)` that implements the forward pass of a neural net. Here `X` is a matrix in which each row is an input to the neural net, and the other arguments are the weights and biases of the neural net given in Equation (1). The function should return four matrices, containing the values of u, h, z and o in Equation (1).

Test your `forward` function on the test data you generated above. Use the weights and bias terms from the neural net with 3 hidden units that you learned in Question (d) in your best training run (call it `nn3`). Compare the output values returned by your `forward` function to the probabilities returned by `nn3` when run on the same input. Specifically, compute and print $\sum_n (o_1^{(n)} - o_2^{(n)})^2$, where the sum is over all test points. Here, $o_1^{(n)}$ is the neural-net output computed by your `forward` function on test point $x^{(n)}$, and $o_2^{(n)}$ is the output computed by `nn3`. The printed value should be very close to 0.

(b) (20 points) *Batch Gradient Descent.*

Write a Python function `bgd(J,K,lrate)` to train the kind of neural networks considered in Question 2 using batch gradient descent (see Question 6 of Assignment 2). Here, `J` is the number of units in the hidden layer, `K` is the number of epochs of training, and `lrate` is the learning rate. Initialize the weights using random numbers from a Gaussian distribution with a mean of 0 and a standard deviation of 1. Initialize the bias terms to be all zero. Every ten epochs (starting

with epoch 10), the function should compute and record the training loss, the mean training accuracy, and the mean test accuracy. Use the training and test data that you generated above.

After training is finished, the function should print the final test accuracy. In addition, plot the recorded histories of training and test accuracy on a single pair of axes using `semilogx`. Label the axes, and title the figure, "Question 3(b): training and test accuracy for bgd". These curves should be jagged, tend to increase, and tend to flatten out at the right end. In a second figure, plot the recorded history of training loss using `semilogx`. Label the axes, and title the figure, "Question 3(b): training loss for bgd". This curve should be smooth and decreasing, flattening out somewhat at the right end. In a third figure, plot the test accuracy for the final K/2 epochs using `plot`. This curve should be very jagged, and there may be no discernable trend. Label the axes, and title the figure, "Question 3(b): final test accuracy for bgd". In a fourth figure, plot the training loss for the final K/2 epochs using `plot`. This curve should be smooth and decreasing. Label the axes, and title the figure, "Question 3(b): final training loss for bgd".

Lastly, as in Question 1(d), generate a scatter plot of the training data, and on top of this, plot the decision boundary and contours of the neural net. Title the figure, "Question 3(b): Decision boundary for my neural net". To generate the plot, you should modify `dfContour` in `bonnerlib2`. Call the modified function `MYdfContour`, and add the comment "# *** modified ***" to the end of every line that you modify. To bring out the detail in the plot, put the figure into full-screen mode before saving it.

Execute your function using $J = 3$ hidden units and $K = 1,000$ epochs. Experiment to find a good learning rate. You should be able to attain a final test accuracy of at least 84.8% on some training run. (If not, you may have an unlucky training set, so try generating a new one.) The decision boundary should be similar to that of the best neural net in Question 1(d). You may have to run your `bgd` funtion several times to get the desired performance. Print the learning rate.

(c) (10 points) *Stochastic Gradient Descent.*

Repeat part (b) using stochastic gradient descent (see Question 7 of Assignment 2). Use mini-batches of 50 training points, train for 20 epochs, and record the training loss and training and test accuracies after every epoch (using the entire training and test sets). Generate the same figures and print the same results as in part (b). Title the figures appropriately. By experimenting with the learning rate, you should be able to attain a final test accuracy of at least 84.8% on some training run. (If not, you may have an unlucky training set, so try generating a new one.) The decision boundary should be similar to that in part (b).

(d) (5 points) Explain why stochastic gradient descent takes only 20 epochs to achieve 84.8% accuracy, while batch gradient descent takes 1,000 epochs.

125 points total

University of Toronto Mississauga
CSC 411 - Machine Learning and Data Mining

Cover sheet for Assignment 3

Complete this page and hand it in with your assignment.

Name: _____
(Underline your last name)

Student number: _____

I declare that the solutions to Assignment 3 that I have handed in are solely my own work, and they are in accordance with the University of Toronto Code of Behavior on Academic Matters.

Signature: _____