University of Toronto Mississauga
Department of Mathematical and Computational Sciences
CSC 411 - Machine Learning and Data Mining, Fall 2019

Assignment 3

Due date: Wednesday December 4, 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".

Because they are very slow, you should avoid the use of loops and iteration in your Python programs, replacing them by Numpy matrix and vector operations wherever possible. For the same reason, you should not use recursion or higher-order functions (such as the python `map` function or any numpy function listed under "functional programming", such as `apply-along-axis`, which are just just loops in disguise), unless otherwise specified.

In addition, if a part of a question prints any output, you should precede all code for that part with lines like the following:

```
print('\n')
print('Question 1(d).')
print('-------------')
```

You do not have to include these lines in your line-counts of code.

1. (30 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 two-dimensional data similar to that of Question 1 of Assignment 2. To train a neural net, you will use the class `MLPclassifier` in `sklearn.neural_network`. As in Assignment 2, to plot the decision boundary, you should use the function `dfContour` from `bonnerlib2`, which you can download from the course web site.

   To keep things simple, this question will only use neural nets with a single hidden layer. You should set your neural nets to use stochastic gradient descent (sgd) as the optimization method, and `logistic` for the hidden-layer activation function. Set the initial learning rate to 0.01, the tolerance for optimization to 10^{-8}, and the maximum number of iterations to 1,000. When calling `MLPclassifier` it may be convenient to put each argument on a separate line, for readability. In such cases, the call to `MLPclassifier` will still count as only a single line of code.

   In answering the questions below, you will need to use the methods and attributes of `MLPclassifier`. You should not use any loops in your programs, except a single loop when generating an array of subplots. You may borrow any code you like from Assignments 1 or 2, but you must include it in your line counts of code.

   (a) (0 points) Using the function `gen_data` that you wrote in Assignment 2, generate a training set like that of Question 4(c) of Assignment 2. That is, `mu0` = (1,1), `mu1` = (2,2), `cov0` = 0, `cov1` = 0.9, `N0` = 1,000 and `N1 = 500`. Generate a test set with `N0` = 10,000 and `N2` = 5,000. Use this data set in the rest of this question.

   (b) (8 points total) 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] The x and y axes should both extend from -3 to 6. (Be sure to plot the training data and set the axes *before* you call `dfContour`.) Title the figure, "Question 1(b): Neural net with 1 hidden unit." The decision boundary should be a straight line. In addition, print the accuracy, precision and recall of the neural net on the test data. You should write your own code to compute the precision and recall using only Numpy functions. You can do this question in at most 26 lines of code. (5 points)

   Explain why the decision boundary is a straight line (3 points).

   (c) (4 points) Train a neural net with two units in the hidden layer, plot the training data, and draw the decision boundary on top of the training data. Do this twelve times. You should observe twelve different decision boundaries. Arrange the twelve plots in a 4×3 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

[1] `dfContour` plots the decision boundary as a black line and plots several other contours of $P(C = 1|x)$ for a classifier. The colour of a region represents the predicted probability that a point in the region is in class 1 or 0. The darker the blue, the greater the probability that $C = 1$. The darker the red, the greater the probability that $C = 0$.

figure, "Question 1(c): Best neural net with 2 hidden units." Print the accuracy, precision and recall of this net on the test data. The accuracy should be greater than in part (b). You can do this question in at most 26 lines of code.

(d) (4 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. You can do this question in at most 4 lines of code by reusing code from part (c), *i.e.*, by calling functions defined in part (c). (Each such function call counts as one line of code.)

(e) (4 points) Repeat part (c) for neural nets with four hidden units. The best test accuracy here should be about the same as in part (d). (If not, run your procedures again.) Title the figures appropriately. You can do this in at most 4 lines of code by reusing code from part (c).

(f) (5 points) The best test accuracies in parts (c), (d) and (e) should all be less than in Question 4(c) of Assignment 2. *Explain why we should expect this.*

(g) (5 points) *Explain why you get so many different decision boundaries in parts (c), (d) and (e).*

2. (50 points total) *Neural Networks: theory*

In this question, you will develop matrix equations needed to implement a neural net in Question 3. The net has a single hidden layer and multiple output units, where the hidden units use a sigmoid activation function. We will use the network for multi-class classification, so the activation function at the output is a softmax. The operation of the neural net can be specified as follows:

$$o = softmax(z) \qquad z = hW + w_0 \qquad h = \sigma(u) \qquad u = xV + v_0 \tag{1}$$

Here x, h and o are row vectors representing the input, the hidden units, and the output, respectively; W and V are weight matrices; w_0 and v_0 are row vectors representing bias terms; and σ is the sigmoid function.

Recall that the softmax function is given by

$$o_i \;=\; \frac{e^{z_i}}{\sum_j e^{z_j}}$$

and the sigmoid function is given by $\sigma(y) = 1/(1+e^{-y})$, for any real number, y. In this question, you may use the following properties of the softmax and sigmoid functions, respectively, without proving them:

$$\frac{\partial o_i}{\partial z_j} \;=\; o_i\delta_{ij} - o_i o_j \tag{2}$$

$$\frac{\partial \sigma(y)}{\partial y} \;=\; \sigma(y)[1-\sigma(y)] \tag{3}$$

where δ_{ij} is the Kronecker delta (*i.e.*, $\delta_{ij} = 0$ if $i \neq j$, and 1 otherwise). You may not use any other results from the lecture slides, although you may use their proofs if

you find them useful. (Note however that the derivatives on slide 38 of Lecture 10 are wrong.)

Since we are using the neural network for classification, the loss function used during training is the cross entropy:

$$C \;=\; \sum_n c(t^{(n)}, o^{(n)}) \qquad \text{where} \qquad c(t, o) \;=\; -\sum_i t_i \log o_i \tag{4}$$

where the left sum is over training points, and the right sum is over output units. Here, $o^{(n)}$ is the output of the neural net on input $x^{(n)}$, and $t^{(n)}$ is a binary vector representing the target class of $x^{(n)}$ using a 1-of-K encoding (see slide 10 of Lecture 7).

Below, you will derive gradient equations for the neural net. Since the inputs and outputs of each layer of the net are vectors, we will need a generalization of the chain rule from real functions to vector functions. For example, let z_j be a component of vector z. As explained in class, z_j affects the loss, c, through each of the output units, o_i. The derivative of c thus has components due to each of the o_i. In particular,

$$\frac{\partial c}{\partial z_j} \;=\; \sum_j \frac{\partial c}{\partial o_i} \frac{\partial o_i}{\partial z_j} \tag{5}$$

Likewise, hidden unit h_k affects c through each component, z_j, of vector z. Thus,

$$\frac{\partial c}{\partial h_k} \;=\; \sum_i \frac{\partial c}{\partial z_j} \frac{\partial z_j}{\partial h_k} \tag{6}$$

In the questions below, X, Z, H, U, O and T are data matrices whose n^{th} rows are $x^{(n)}$, $z^{(n)}$, $h^{(n)}$, $u^{(n)}$, $o^{(n)}$ and $t^{(n)}$, respectively. $\vec{1}$ is a row vector of 1's. To make your proofs easier to mark, use the index n to range over training instances, use i to range over output units, j to range over components of z, k to range over hidden units, and m to range over features of an input vector, x.

Prove each of the following matrix equations from scratch, using only the results stated above, along with basic results from calculus and linear algebra:

(a) (12 points)

$$\frac{\partial C}{\partial Z} \;=\; O - T$$

(b) (7 points)

$$\frac{\partial C}{\partial W} \;=\; H^T \frac{\partial C}{\partial Z}$$

(c) (3 points)

$$\frac{\partial C}{\partial w_0} \;=\; \vec{1} \frac{\partial C}{\partial Z}$$

(d) (9 points)

$$\frac{\partial C}{\partial H} \;=\; \frac{\partial C}{\partial Z} W^T$$

(e) (7 points)

$$\left[\frac{\partial C}{\partial U}\right]_{nk} = H_{nk}(1 - H_{nk}) \left[\frac{\partial C}{\partial H}\right]_{nk}$$

(f) (7 points)

$$\frac{\partial C}{\partial V} = X^T \frac{\partial C}{\partial U}$$

(g) (3 points)

$$\frac{\partial C}{\partial v_0} = \vec{1} \frac{\partial C}{\partial U}$$

(h) (2 points) Let `dCdw0` and `dCdZ` be Numpy arrays representing $\partial C/\partial w_0$ and $\partial C/\partial Z$, respectively. Write a single line of Python code for computing $\partial C/\partial w_0$ using the equation in part (c) *without constructing a vector of 1's or using any multiplication*. Do not execute this code. Hand it in in the same file as your proofs.

3. (60 points total) *Neural Networks: implementation*

In this question, you will use the theory developed in Question 2 to write a Python program that trains neural networks on the MNIST data from Assignment 2. As in Question (2), we will only consider neural nets with one hidden layer and a sigmoid (logistic) activation function. You will implement both batch and stochastic gradient descent. The batch implementation is more straighforward, but as you will see, it takes much longer to converge. For comparison, and to provide some quick results, you will first train neural networks on MNIST using `MLPclassifier`. Do not use any functions in `sklearn` other than `MLPclassifier` and its methods.

In parts (a) to (e), you will use a reduced training set, which will reduce accuracy, but will speed up training and program development. Use the first 10,000 points of the MNIST training data as validation data, and use the next 10,000 points as the reduced training data. (So 40,000 of the 60,000 MNIST training points will be unused.) Use all of the MNIST test data for testing. Once your programs are working, you will use the entire MNIST data set in parts (f) and (g). When you are asked to compute the cross entropy, this refers to the training cross entropy, i.e., cross entropy computed wrt the data the neural net was trained on (whether reduced data or not).

(a) (5 points) *Stochastic Gradient Descent.*

Using `MLPclassifier` in `sklearn`, train a neural net with 30 hidden units on the reduced MNIST data. Do 10 iterations of training. Use stochastic gradient descent as the solver with a batch size of 100. Set the tolerance for optimization to 10^{-8}, a very low value to ensure all 10 iterations are carried out. Do *not* use 1-of-K encodings of class labels with `MLPclassifier`, as it expects integer labels for multi-class classification.[2]

You will have to experiment to find a good learning rate. At this stage, you may want to set the keyword argument `verbose` to True, to print out the training

[2] `MLPclassifier` assumes that class labels encoded as binary arrays are meant for multi-label classification (something we have not studied), which is different from multi-class classification.

error at each iteration. If the training error increases as learning progresses, or bounces around erratically, then the learning rate is too high. If the training error decreases very slowly, then the learning rate is too small. Find a learning rate that maximizes the test accuracy after 10 iterations. Only consider learning rates that are powers of 10, like 100, 10, 1, 0.1, 0.01, etc. You should be able to achieve a test accuracy of about 94% after 10 iterations. Do not hand in any of these experiments. The point is simply to find a good learning rate.

Using the learning rate you have just found, train the neural net 10 times (with `verbose=False`) on the reduced MNIST data set. Compute and print out the validation accuracy of each trained net. Choose the trained net that has the maximum validation accuracy. Print out its validation accuracy, test accuracy and cross entropy. The cross entropy should be between 300 and 500. (If not, try running your program again.) You can use the `score` method to compute the accuracies, but you will have to compute the cross entropy yourself. Finally, print out the learning rate that you used. You may use one loop in your program.

(b) (10 points) *Batch Gradient Descent.*

Repeat part (a) with a batch size equal to the size of the training set. This effectively carries out batch gradient descent. You should find that the same learning rate is still about the best, but the accuracy after 10 iterations is lower than in part (a) and is 75%-80%. *Explain why this is.*

(c) (5 points) Here, we will see how long it takes batch gradient descent to achieve the same level of test accuracy as stochastic gradient descent. Use `MLPclassifier` with the same arguments as in part (b) to define a neural net and train it for 50 iterations. Train the net just once, not 10 times. Print the final training and test accuracies and cross entropy. You should find that the test accuracy is about 92%. Do not use any loops.

Repeat again using 200 iterations. The final test accuracy should now be closer to 94%, i.e., an accuracy that was achieved in part (a) after just 10 iterations.

(d) (20 points total) *Batch Gradient Descent: implementation.*

Use the theory developed in Question 2 to write a Python program that trains a neural net with one hidden layer for multi-class classification using batch gradient descent. Your program may use one loop. Using program comments, clearly indicate what portion of your program implements the forward pass of training, and what portion implements back propagation. Initialize the weight matrices randomly using a standard Gaussian distribution (*i.e.*, mean 0 and variance 1), and initialize the bias terms to 0.

When doing weight updates, use the average gradient, not the gradient itself. For instance, to update the weight matrix W, use the command

$$W \;=\; W - \lambda \frac{\partial C}{\partial W} / N$$

where λ is the learning rate and N is the number of terms in the sum in equation (4). In batch gradient descent, N is the number of points in the training

set, but would be the batch size in stochastic gradient descent. Using the average gradient means that the optimal learning rate does not change much when the size of the training set changes (which is why the same learing rate worked in parts (a) and (b)). *Explain why this is* (5 points). Notice that using the average gradient is equivalent to dividing the learning rate by N.

Run your program on the reduced MNIST data set using 30 hidden units and 100 iterations of gradient descent. Again, you will have to experiment to find a good learning rate. You should be able to achieve a test accuracy of around 85% after 100 iterations. Do not hand in any of these experiments. You will probably find that the optimal learning rate here is different than in part (b), since `MLPclassifier` uses momentum, while your program does not.

As in part (a), use the learning rate you have just found to train a neural net 10 times. (You may use one additional loop for this purpose.) Compute and print out the validation accuracy of each trained net. Choose the trained net that has the maximum validation accuracy. Print out its validation accuracy, test accuracy and cross entropy. Finally, print out the learning rate that you used. (15 points)

(e) (10 points) *Stochastic Gradient Descent: implementation.*

Modify your program in part (d) to peform stochastic gradient descent with mini-batches. That is, instead of computing the gradient of the loss function on the entire training set at once, compute the gradient on a small, random subset of the training data (called a mini-batch), perform weight updates, and then move on to the next mini-batch, and so on. As you saw in parts (a), (b) and (c), this can lead to much faster convergence. As in part (d), your implementation of the weight-update step should use the average gradient (averaged over the current mini-batch).

To produce random mini-batches of the training data, first shuffle the training data randomly, then sweep across the shuffled data from start to finish. For example, if we want mini-batches of size 100, then the first mini-batch is the first 100 points in the training set. The second mini-batch is the next 100 points. The third mini-batch is the next 100 points, etc. (If the number of training points is not a multiple of 100 then the last mini-batch in a sweep will have fewer than 100 points in it.) Each such sweep of the training data is called an epoch. Before each epoch begins, you should reshuffle the entire training set randomly, so that each epoch produces a different, random set of mini-batches.

Your program may use two loops, one nested inside the other. Program comments should clearly indicate where an epoch begins and where mini-batches are created.

Now, repeat the rest of part (d), performing 100 epochs of training with mini-batches of size 100, as in part (a). Because you are using the average gradient to perform weight updates, you should find that the optimal learning rate here is about the same as in part (d). You should be able to achieve a test accuracy of over 90% after 100 epochs.

(f) (5 points) In the rest of this question, you will use the entire MNIST data set, so you may want to reload it from a file. With more data, we can achieve higher test

accuracy. To fully take advantage of this possibility, we will increase the number of hidden units in the neural net, to increase the net's capacity to represent complex functions and decision boundaries. With more data and more hidden units, the neural net will now take much longer to train, so we will forgo the use of validation data, and will train the net just once, not 10 times.

Modify your program for stochastic gradient descent so that it prints out the test accuracy after every ten epochs (with the first print-out being after the first epoch), so we can monitor the progress of training.

Use the modified program to train a neural net with 100 hidden units on the full MNIST data set using mini-batches of size 100. Perform 100 epochs of training. Print out the final training accuracy, test accuracy and cross entropy. You should be able to achieve a final test accuracy of over 96%. In fact, a test accuracy of over 90% can be achieved after just one epoch, and an accuracy of over 96% after 50 epochs. You should also find that the final training accuracy is almost 100%.

(g) (5 points) Repeat part (f) with your program for batch gradient descent. You should find that the test accuracy increases much more slowly during training, and that the final test accuracy is below 90%.

140 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: _____