CS 540: Introduction to Artificial Intelligence
Homework Assignment # 10

Assigned: 4/23
Due: 4/30 before class

Hand in your homework:

If a homework has programming questions, please hand in the Java program. If a homework has written questions, please hand in a PDF file. Regardless, please zip all your files into hwX.zip where X is the homework number. Go to UW Canvas, choose your CS540 course, choose Assignment, click on Homework X: this is where you submit your zip file.

Late Policy:

All assignments are due at the beginning of class on the due date. One (1) day late, defined as a 24-hour period from the deadline (weekday or weekend), will result in 10% of the total points for the assignment deducted. So, for example, if a 100-point assignment is due on a Wednesday 9:30 a.m., and it is handed in between Wednesday 9:30 a.m. and Thursday 9:30 a.m., 10 points will be deducted. Two (2) days late, 25% off; three (3) days late, 50% off. No homework can be turned in more than three (3) days late. Written questions and program submission have the same deadline.

Assignment grading questions must be raised with the instructor within one week after the assignment is returned.

Collaboration Policy:

You are to complete this assignment individually. However, you are encouraged to discuss the general algorithms and ideas with classmates, TAs, and instructor in order to help you answer the questions. You are also welcome to give each other examples that are not on the assignment in order to demonstrate how to solve problems. But we require you to:

- not explicitly tell each other the answers
- not to copy answers or code fragments from anyone or anywhere
- not to allow your answers to be copied
- not to get any code on the Web

In those cases where you work with one or more other people on the general discussion of the assignment and surrounding topics, we suggest that you specifically record on the assignment the names of the people you were in discussion with.

Question 1: Neural Network [60 points, evenly among parts]

We will build a neural network to determine if a bank note is authentic. We have obtained a data set (`https://archive.ics.uci.edu/ml/datasets/banknote+authentication`) with features that are properties of a wave transformed bank note image such as variance, skewness, curtosis and entropy. The data set has already been processed and three csv files (train.csv, eval.csv and test.csv) can be downloaded from the course website. Each line in these files have four real-valued features which we will denote with $x = (x_1, x_2, x_3, x_4)$, and the class label y which is either 0 or 1.

Our neural network architecture is shown below:

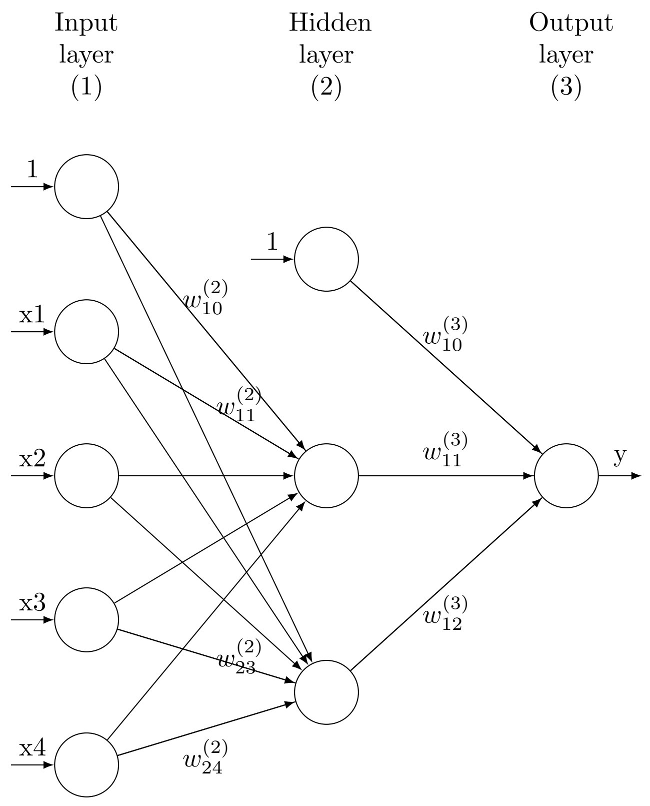

Figure 1: Neural Network Architecture.

There is one hidden layer with two hidden units, and one output layer with a single output unit. The input layer is fully connected to the hidden layer, and the hidden layer is fully connected to the output layer. Each layer also has a constant bias 1 input with the corresponding weight. The weight of the network from unit j in layer $l-1$ to unit k in layer l will be denoted by $w_{kj}^{(l)}$. For bias nodes, j will be denoted by 0 in the weights. This convention is demonstrated for some of the weights in figure 1. We will use *Sigmoid* activation function for all the activation units. Recall that the sigmoid function activation is obtained by,

$$g(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

This neural network is fully defined by the 13 weights $w_{10}^{(2)}, \ldots, w_{14}^{(2)}, w_{20}^{(2)}, \ldots, w_{24}^{(2)}, w_{10}^{(3)}, w_{11}^{(3)}$ and $w_{12}^{(3)}$.

Write a program **NeuralNet.java** with the following command line format:

```
$java NeuralNet FLAG [args]
```

Where the optional arguments are real valued (use 'double' data type for them).

1. **Forward Propagation:**

   We will first focus on making predictions *given* fixed weights.

   Recall in a neural network for any unit j, it first collects input from lower units to produce $z_j^{(l)}$, then applies a non linear activation function to produce an activation, $a_j^{(l)}$:

$$z_j^{(l)} = \sum_{i:i\to j} (a_i^{(l-1)} w_{ji})$$

$$a_j^{(l)} = g(z_j^{(l)}) = \frac{1}{1+e^{-z}}$$

   Note that the activations for the input layer are the input themselves: $a_j^{(1)} = x_j$ and $a_0^{(1)} = 1$ is the bias unit.

   When FLAG=100, arg1 ... arg13 are the weights $w_{10}^{(2)}, \ldots, w_{14}^{(2)}, w_{20}^{(2)}, \ldots, w_{24}^{(2)}, w_{10}^{(3)}, w_{11}^{(3)}$ and $w_{12}^{(3)}$, and arg14=x_1, arg15=x_2, arg16=x_3, arg17=x_4. Print activations of the hidden layer units ($a_1^{(2)}$, $a_2^{(2)}$) on line one separated by a space followed by the activation of the output layer unit ($a_1^{(3)}$) on line two. When printing, show 5 digits after decimal point by rounding (but do not round the actual variables). For example,

```
$java NeuralNet 100 .1 .2 .3 .4 .5 .5 .6 .7 .8 .9 .9 .5 .2 1 0 1 0
0.66819 0.86989
0.80346

$java NeuralNet 100 .021 .247 .35879 .1414 .75 .512 .686 .717 .818 .919 .029 .135 .20701 1 0 1 0
0.60094 0.88247
0.57268

$java NeuralNet 100 0 .2 .3 .4 .5 0 .6 .7 .8 .9 0 .5 .2 0 0 0 0
0.50000 0.50000
0.58662
```

   Tip to handle Commandline arguments:

   Since there are a lot of commandline arguments in this homework, you can use the following trick to avoid typing the arguments each time you run the program and instead load them from a file directly.

```
$java NeuralNet ‘< 100.args‘
```

where 100.args is a file that contains commandline arguments on a single line separated by spaces. Also note that the quote characters are back-ticks. This doesn't work with normal single quotes.

2. **Back Propagation:**

   To learn the weights of the neural network, we first store all the activations computed using forward propagation. Then, we propagate the δ terms in a backwards manner to calculate the gradient of error with respect to each weight. Once we have the gradients, we can use Gradient Descent to update the weights to minimize the error.

   Given a training example $x = (x_1, x_2, x_3, x_4)$ and its label y, the error made by the neural network on the example is defined as

$$E_x = \frac{1}{2}(a_1^{(3)} - y)^2$$

   The partial derivative of error with respect to the output activation $z_1^{(3)}$ from chain rule is

$$\frac{\partial E_x}{\partial z_1^{(3)}} = \delta_1^{(3)} = \frac{\partial E_x}{\partial a_1^{(3)}} \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}$$

   Now,

$$\frac{\partial E_x}{\partial a_1^{(3)}} = \frac{\partial(\frac{1}{2}(a_1^{(3)} - y)^2)}{\partial a_1^{(3)}} = \frac{1}{2}2(a_1^{(3)} - y) = a_1^{(3)} - y$$

   and

$$\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} = \frac{\partial g(z_1^{(3)})}{\partial z_1^{(3)}} = a_1^{(3)}(1 - a_1^{(3)})$$

   So, we have

$$\delta_1^{(3)} = (a_1^{(3)} - y)a_1^{(3)}(1 - a_1^{(3)})$$

   When FLAG=200, arg1 ... arg13 are the weights $w_{10}^{(2)}, \ldots, w_{14}^{(2)}, w_{20}^{(2)}, \ldots, w_{24}^{(2)}, w_{10}^{(3)}, w_{11}^{(3)}$ and $w_{12}^{(3)}$, and arg14=x_1, arg15=x_2, arg16=x_3, arg17=x_4, and arg18 = y. Print a single number, $\delta_1^{(3)}$ with 5 decimals precision. For example,

```
$java NeuralNet 200 .1 .2 .3 .4 .5 .5 .6 .7 .8 .9 .9 .5 .2 1 0 1 0 1
-0.03104

$java NeuralNet 200 -.1 .2 -.3 -.4 .5 -.5 -.6 -.7 .8 -.9 -.9 .5 -.2 -1 0 -1 0 1
-0.14814

$java NeuralNet 200 .101 .809 .31 .9 .13 .55 .66 .12 .31 .1 .92 .05 .22 10 0 11.1 0.01 1
-0.04172
```

3. The partial derivative of error with respect to hidden layer activation units can be similarly computed using Chain Rule. For hidden unit j:

$$\delta_j^{(2)} = \delta_1^{(3)} w_{1j}^{(3)} a_j^{(2)} (1 - a_j^{(2)})$$

FLAG=300 has same arguments as FLAG=200. Print two numbers $\delta_1^{(2)}, \delta_2^{(2)}$ separated by space. For example,

```
$java NeuralNet 300 .1 .2 .3 .4 .5 .5 .6 .7 .8 .9 .9 .5 .2 1 0 1 0 1
-0.00344 -0.00070

$java NeuralNet 300 -.1 .2 -.3 -.4 .5 -.5 -.6 -.7 .8 -.9 -.9 .5 -.2 -1 0 -1 0 1
-0.01847 0.00657

$java NeuralNet 300 .101 .809 .31 .9 .13 .55 .66 .12 .31 .1 .92 .05 .22 10 0 11.1 0.01 1
-0.00000 -0.00000
```

4. We now have all the information that we need to compute the gradient of error with respect to edge weights. To compute the partial derivative with respect to the edge weights:

$$\frac{\partial E_x}{\partial w_{jk}^{(l)}} = \delta_j^{(l)} a_k^{(l-1)}$$

FLAG=400 also has same arguments as FLAG=200. Print $\frac{\partial E_x}{\partial w_{10}^{(2)}} \frac{\partial E_x}{\partial w_{11}^{(2)}} \ldots \frac{\partial E_x}{\partial w_{14}^{(2)}}$ on line 1, $\frac{\partial E_x}{\partial w_{20}^{(2)}} \frac{\partial E_x}{\partial w_{21}^{(2)}} \ldots \frac{\partial E_x}{\partial w_{24}^{(2)}}$ on line 2, and $\frac{\partial E_x}{\partial w_{10}^{(3)}} \frac{\partial E_x}{\partial w_{11}^{(3)}} \frac{\partial E_x}{\partial w_{12}^{(3)}}$ on line 3 each separated by a space. For example,

```
$java NeuralNet 400 .1 .2 .3 .4 .5 .5 .6 .7 .8 .9 .9 .5 .2 1 0 1 0 1
-0.03104 -0.02074 -0.02700
-0.00344 -0.00344 -0.00000 -0.00344 -0.00000
-0.00070 -0.00070 -0.00000 -0.00070 -0.00000

$java NeuralNet 400 -.1 .2 -.3 -.4 .5 -.5 -.6 -.7 .8 -.9 -.9 .5 -.2 -1 0 -1 0 1
-0.14814 -0.07777 -0.04916
-0.01847 0.01847 -0.00000 0.01847 -0.00000
0.00657 -0.00657 0.00000 -0.00657 0.00000

$java NeuralNet 400 .101 .809 .31 .9 .13 .55 .66 .12 .31 .1 .92 .05 .22 10 0 11.1 0.01 1
-0.04172 -0.04172 -0.04172
-0.00000 -0.00000 -0.00000 -0.00000 -0.00000
-0.00000 -0.00000 -0.00000 -0.00000 -0.00000
```

5. Now we perform Stochastic Gradient Descent to train the neural network on the given bank note authentication data set. To do so, for each training example, we first compute activations for all the units using Forward Propagation. We then compute δ terms for each hidden and output unit and

compute gradients of error with respect to each weight using Backward Propagation. We then update the weights as follows:

$$w_{jk}^{(l)} = w_{jk}^{(l)} - \eta \frac{\partial E_x}{\partial w_{jk}^{(l)}}$$

where η is the learning rate chosen.

We will use the training set "train.csv" for training the network. We will compute error on evaluation set "eval.csv" by summing up the error on each example from the set as follows:

$$E_{eval} = \sum_{x \in Eval} E_x = \sum_{x \in Eval} \frac{1}{2}(a_1^{(3)} - y)^2$$

When FLAG=500, arg1 ... arg13 are initial weights $w_{10}^{(2)}, \ldots, w_{14}^{(2)}, w_{20}^{(2)}, \ldots, w_{24}^{(2)}, w_{10}^{(3)}, w_{11}^{(3)}$ and $w_{12}^{(3)}$, and arg14=η. Print two lines for each training example in the order of their appearance (we shall ignore the actual Stochastic Gradient Descent algorithm in which training examples are randomly selected and use the actual file order instead):

(a) the updated weights $w_{10}^{(2)}, \ldots, w_{14}^{(2)}, w_{20}^{(2)}, \ldots, w_{24}^{(2)}, w_{10}^{(3)}, w_{11}^{(3)}, w_{12}^{(3)}$

(b) the evaluation set error E_{eval} after the update

For example, (the first output line for every training example is wrapped as it is too long to fit on a single line. Please refer to the test cases provided for exact outputs)

```
$java NeuralNet 500 .1 .2 .3 .4 .5 .5 .6 .7 .8 .9 .9 .5 .2 .1
0.10020 0.19910 0.29883 0.40218 0.49989 0.50006 0.59973 0.69965
    0.80065 0.89997 0.90277 0.50228 0.20243
38.14430
0.10071 0.19860 0.30025 0.40156 0.49910 0.50026 0.59953 0.70022
    0.80040 0.89965 0.90709 0.50388 0.20403
38.22326
0.09969 0.19466 0.29375 0.40403 0.50010 0.50024 0.59942 0.70005
    0.80047 0.89968 0.89494 0.49427 0.19201
37.84961
0.09986 0.19411 0.29245 0.40606 0.50000 0.50028 0.59928 0.69970
    0.80100 0.89965 0.89770 0.49662 0.19451
37.91446
0.10012 0.19392 0.29335 0.40517 0.49912 0.50032 0.59925 0.69982
    0.80088 0.89953 0.90327 0.49720 0.19469
37.97933
...
0.08324 -0.17301 -0.08162 0.31971 0.54455 -0.05914 1.40048 0.86019
    0.85080 0.71119 0.99056 0.36316 -2.41299
7.88486
0.08293 -0.17443 -0.08069 0.31889 0.54418 -0.05913 1.40053 0.86016
    0.85083 0.71120 0.98618 0.35992 -2.41737
7.88083
```

6. Now, our neural network is trained. We will use the trained neural network to make predictions on the test set "test.csv" and compute the test set accuracy. Accuracy is defined as the fraction of examples correctly predicted. First, using the initial weights and η given, train your neural network.

   FLAG=600 has same arguments as FLAG=500. For each example in "test.csv," use the trained neural network weights and print actual label, predicted label and confidence of prediction on each line. Confidence of prediction is the actual value of the activation of the output unit. Consider predicted label to be 0 when this confidence is less than or equal to 0.5, else predict 1. In the end, print the test set accuracy with 2 decimal precision.

```
$java NeuralNet 600 .1 .2 .3 .4 .5 .5 .6 .7 .8 .9 .9 .5 .2 .1
1 1 0.54785
0 0 0.21332
0 1 0.63250
0 0 0.24230
0 0 0.21203
...
0 0 0.19786
0 0 0.19522
0.93
```