CS 422: Data Mining

Department of Computer Science
Illinois Institute of Technology
Vijay K. Gurbani, Ph.D.

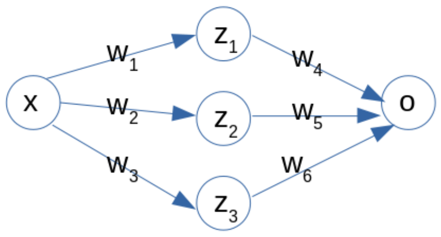
Fall 2021: Homework 7 (10 points)

Due date: Friday, Nov 12, 2021, 11:59:59 PM Chicago Time

Please read all of the parts of the homework carefully before attempting any question. If you detect any ambiguities in the instructions, please let me know right away instead of waiting until after the homework has been graded.

1. Exercises (Please submit a PDF file containing answers to these questions. Any other file format will lead to a loss of 0.5 point. Non-PDF files that cannot be opened by the TAs will lead to a loss of all points.)

1.1 (2 points) Consider the neural network shown in the figure below.



The weight matrix, W, is: $[1, 1, -1, 0.5, 1, 2]^T$. Assume that the hidden layer uses RelU and the output layer uses Sigmoid activation function. Assume squared error. The input x = 4, and the output y = 0.

Recall that RelU is defined as a function, f(x) = max(0, x). Its derivative is $f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$

Squared error is defined as $E(y, \hat{y}) = (y - \hat{y})^2$.

The partial derivative of E with respect to $\hat{y}$ is $-2(y - \hat{y})$.

Using this information, answer the following questions:
(Show all work, and all answers should be rounded to 3 decimal places OR POINTS WILL BE TAKEN OFF!)
(a) Use forward propagation to compute the predicted output.
(b) What is the loss or error value?
(c) Using backpropagation, compute the gradient of the weight vector, that is, compute the partial derivative of the error with respect to all of the weights.
(d) Using a learning rate of 1.0, compute new weights from the gradient. With the new weights, use forward propagation to compute the new predicted output, and the loss (error).
(e) Comment on the difference between the loss values you observe in (b) and (d).

1.2 (2 points) Tan, Ch 7, Ex. 2, 6, 11, 12.

2 Practicum problems

2.1 Feed Forward Neural Networks (6 points)

Please use the template file (HW-7-Problem-2-1-Template.Rmd) provided with the homework to bootstrap your code.

The dataset available in the file activity-small.csv [1] in Blackboard contains data in three dimensions from a single chest-mounted accelerometer. The dataset is intended for activity recognition research purposes. It provides challenges for identification of people using motion patterns. The predictor variables are in three dimensions: x-acceleration, y-acceleration, and z-acceleration. The class label is one of {0,1,2,3}, where 0 implies “Working on the computer,”, 1 implies “Standing up, walking, and going up/down the stairs,”, 2 implies “Walking,” and 3 implies “Talking while standing.”

The dataset contains 1,000 observations, of which 80% should be used for training a neural network, and 20% for testing. The intent of the trained model will be to predict the class label.

To get you started, please use the file HW-7-Problem-2-1-Template.Rmd as it contains starter code to scale the dataset and to set the seed. Add your code to the template file as indicated in the file.

(a) Create a neural network and train it on the training dataset so it predicts one of the four classes: {0, 1, 2, 3}. You can play around with using different activation functions (sigmoid, relu, tanh) in the hidden layesr. Your network must have **only one** hidden layer for this question.

For the output layer, use the softmax activation function. Loss is measured using ‘categorical_crossentropy’ on the ‘adam’ optimizer.

Train the network for 100 epochs, using a batch size of 1 (batch gradient descent).

Fit your trained model to the held-out testing dataset and create a confusion matrix.

(i) What is the overall accuracy of your model on the test dataset?
(ii) What is the per-class sensitivity, specificity, and balanced accuracy on the test dataset?

Output should be of the form:

```
Batch gradient descent
   Overall accuarcy: XX.XX
   Class 0: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
   Class 1: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
   Class 2: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
   Class 3: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
```

Hint: Write a function to create the neural network model, and have the function return the model. You will be reusing this function in (b). The name of this function must be create_model().

(b) Here, we try mini-batch gradient descent.

Using the neural network from (a), train the network for 100 epochs using the following batch sizes: 1, 32, 64, 128, 256, while keeping all other parameters the same as in (a). (You are now doing mini-batch gradient descent.)

Each time you train the network with the batch size, time how long it takes to train the network.

To benchmark a piece of code in R, you can use the following template:

```
> begin <- Sys.time()
> model %>% fit(…)  # Code to be benchmarked
> end <- Sys.time()
> # Predict on held-out testing dataset, and get the conf. matrix (see c below)
```

The time it took to train the network is the difference in time between 'begin' and 'end' timestamps. (Note that this is user time, not system time, but it is close enough for our understanding.)

WARNING: You will have to rebuild the network model from scratch as you move through the batch sizes, using the function to create a neural network you wrote in (a). If you do not re-build the network model from scratch, it continues to keep residual values from previous batch sizes and continues to incrementally train based on the new batch size. So, before every batch size, start with a clean slate:

```
model ← NULL
model ← create_model()
...
```

For each batch size, note the time it took to train the model. Then fit the model on the held out test dataset and derive overall accuracy, and per class sensitivity, specificity, and balanced accuracy. Your output should look like the following:

```
Batch size: 1
  Time taken to train neural network: XX.XX (seconds)
  Overall accuarcy: XX.XX
  Class 0: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
  Class 1: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
  Class 2: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
  Class 3: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX

Batch size: 32
  Time taken to train neural network: XX.XX (seconds)
  Overall accuarcy: XX.XX
  Class 0: Sens. = XX.XX, Spec. = XX.XX, Bal.Acc. = XX.XX
  ...
```

(c) Analyze the output from the mini-batch gradient descent.

(i) Why do you think that the time vary as you increase the batch size?
(ii) Comment on the output from the mini-batch gradient descent. Does overall accuracy, balanced accuracy and per-class statistics remain the same? Change? If change, why?

(d) Starting with the network in (a), add one more hidden layer to your network and re-train your model and see if adding a second hidden layer produces better performance. Note that deciding how many layers to add and how many neurons per layer is more of an art than an exact science. There are some heuristics, but a lot of experience comes from playing with the neural network and observing its performance as layers (and neurons) are added. I will let you experiment with the number of neurons and specific activation function in the new hidden layer.

Take a principled approach and document how you constructed your new hidden layer, and for each such construction, enumerate the:

(i) Overall accuracy of your model on the test dataset.
(ii) Per-class sensitivity, specificity, and balanced accuracy on the test dataset.

Pick the construction that had the best performance results and compare that to the performance you observed in (a). Comment on the changes you observed by adding a new hidden layer. (Does the performance increase? Decrease? Stay the same?)

[1] The entire activity recognition dataset is available at https://archive.ics.uci.edu/ml/datasets/Activity+Recognition+from+Single+Chest-Mounted+Accelerometer, the dataset in activity-small.csv has been curated to reduce it in size and reduce the number of class labels.