HW9: Feedforward NN
Due: 11pm on March 8, 2023
Author: Dhwani Serai

Q1 (15 points): Suppose a feedforward NN (sometimes called multilayer perceptron or MLP) has m layers: the input layer is the 1st layer, the output layer is the last layer, and there are $m-2$ hidden layers in between. The number of neurons in the k^{th} layer is n_k. Each neuron in one layer is connected to every neuron in the next layer and there are no other connections. Let's ignore the bias for Q1. That is, the model only has weight w, and there is no bias b.

(a) 5 pts: How many model parameters (i.e., weights) are there in this network?
(Ans) Number of layers in the kth layer is n_k So for m layers:
Trainable params: $\sum_{k=1}^{m-1} n_k * n_{k+1}$

(b) 10 pts: Let x be a column vector[1] that denotes the values of the input layer. Let M_k denote the weight matrix between layer k and $k+1$; that is, the cell $a_{i,j}$ in M_k stores the weight on the arc from the j^{th} neuron in layer k to the i^{th} neuron in layer $k+1$. Let's assume all the layers use the same activation function g, which is a function from R to R (R is the set of real numbers). If x is a column vector, $g(x)$ will apply the function g to every element in x and return a column vector as the output.

- Given the input x (x is a column vector), what is the formula for calculating the output of the first hidden layer?
  (Ans) For the first hidden layer the output would be:
  $h_1 = \mathrm{g}(\mathrm{M}_1\mathrm{x})$

- Given the input x, what is the formula for calculating the output of the output layer?
  (Ans) For calculating the output of the output layer:
  $\mathrm{y} = \mathrm{g}(\mathrm{M}_n h_{n-1})$ where h_{n-1} is the output of the n-1 layer
  $= \mathrm{g}(\mathrm{M}_n \mathrm{g}(\mathrm{M}_{n-1} h_{n-2}))$
  This will be recursively applied until the first hidden layer

Q2 (30 points): Suppose that you are training a neural network to do text classification, with $n > 2$ classes.

(a) 5 pts: What loss function can you use for the output layer? Can the loss function be the error rate on the training data? Why or why not? Here, let's assume that you need to use backpropagation for training.

[1] A row vector is a $1 \times n$ matrix (e.g., $[a_1, a_2, ..., a_n]$); a column vector is a $n \times 1$ matrix. If you transpose a row vector, you get a column vector. For more info, see `https://en.wikipedia.org/wiki/Row_and_column_vectors`

Ans For training a text classifier with n ¿ 2 classes, we can use categorical cross-entropy loss function for the output layer.
$L_{cce} = \sum_{i=1}^{n} t_i \log(p_i)$, where
n is number of classes,
t_i is the actual label
p_i is the predicted probability distribution

- No the loss function cannot be error rate because error rate is not differentiable like categorical cross entropy so it cannot be optimized using backpropogation

(b) 15 pts: What are the main idea and benefit of stochastic gradient descent?

What is a training epoch?

Let T be the size of the training data (i.e., the number of training instances), m be the size of mini-batch, and your training process contains E training epoches. How many times is each weight in the NN updated?

Ans One training epoch for stochastic gradient descent corresponds to n weight updates, where n is the size of the training data (for each sample). One training epoch corresponds to one weight update in batch gradient descent (gradient computed according to whole training dataset).

- The main goal of stochastic gradient descent is to gradually update weights, with each iteration moving in the direction of the cost function's sharpest reduction. An update for SGD reacts to a single look at the dataset's training sample. Compared to batch gradient descent, stochastic gradient descent offers the advantage of faster convergence. When utilizing an appropriately declining learning rate, both stochastic and batch gradient descent are certain to converge to local minima.

(c) 10 pts: How can one choose the learning rate? What's the risk if the rate is too big? What's the risk if the rate is too small?
Grid search, learning rate schedules, and adjustable learning rates are some of the techniques we can implement. Any update to momentum-based approaches slows the learning rate. This guarantees that first updates are bigger and subsequent updates are smaller and smaller. Other approaches, which dynamically select the learning rate, include Nesterov accelerated gradient, AdaGrad, and AdaDelta. Without a properly decreasing learning rate, too little or too much learning will cause very slow convergence or the gradient descent to overestimate the optimal value.

Q3 (15 points): Run hw9.sh with different config file settings (e.g., **config1.yml** and **config2.yml** are the config files for the first two experiments in Table 1). The **activation** value in the config file should be set to 0 (for sigmoid function). For the learning rate, keep it as 0.5. Fill out Table 1.

Table 1: Classification accuracy with **sigmoid** activation function

Expt id	# of hidden layer	# of neurons in hidden layers	# of epoches	mini-batch size	test accuracy	CPU time (in minutes)
1	1	30	30	10	74.66	1m 7s
2	1	30	30	50	61.61	0m 56s
3	1	30	100	10	81.8	3m 12s
4	1	60	30	10	75.71	4m 8s
5	2	30, 30	30	10	72.47	1m 9s
6	2	40, 20	30	10	72.57	1m 45s
7	3	20, 20, 20	30	10	73.14	0m 42s

Q4 (20 points):

- In the readme.[txt | pdf], explain which functions (or which lines) in which file(s) you have changed.

- In the network file tanh function and tanh_prime function have been changed
  tanh_prime refers to the derivative of tanh for backpropogation
  tanh = (np.exp(z)-np.exp(-z))/(np.exp(z)+np.exp(-z))
  tanh_prime =1- $\tanh^2$

- Submit the modified python code. Please keep the file names unchanged.

Table 2: Classification accuracy with **tanh** activation function

Expt id	# of hidden layer	# of neurons in hidden layers	# of epochs	mini-batch size	test accuracy	CPU time (in minutes)
1	1	30	30	10	71.23	1m 1s
2	1	30	30	50	61.61	0m 50s
3	1	30	100	10	74.47	3m 12s
4	1	60	30	10	62.76	4m 3s
5	2	30, 30	30	10	66.47	1m 15s
6	2	40, 20	30	10	63.42	1m 50s
7	3	20, 20, 20	30	10	57.33	0m 44s

Q5 (20 points): Answer the following questions:

(a) 10 pts: For any experiment in Q3 or Q4, if you run it multiple times, you are likely to get different results. Why is that?

- specify the line numbers in the python code that causes this non-deterministic behavior of the system.
  Ans Line 64 in the network file shows that the training data is randomly shuffled and then divided into minibatches.
  Also the weights and biases are randomly assigned on line 36 and 37

This causes the data to give some variability in training thus changing the test accuracy by a bit.
Stochastic Gradient Descent is known to optimize the convergence process using mini-batches but this gives some change in the training due to the randomness factor.

- Is this kind of behavior desireable? Why or why not?
  This is not desireable where we need reproducibility in the results.
  Otherwise it is good to have because of faster convergence and lower susceptibility to noise in the dataset.

(b) 10 pts: What can you conclude from Tables 1 and 2, and from Q5(a)?

- Number of neurons: As we increase the number of neurons in the hidden layer the accuracy increases.
- Number of Epochs: Accuracy increases from increasing number of epochs from 30 to 100, indicating the 30 epoch results haven't converged yet. Increasing the number of epochs leads to a pronounced effect on increase in training time, and also produces the highest accuracies for both activation functions.
- Initialization: Different runs produce different results as indicated in Q5(a), since the weights are randomly generated with no fixed seeds for the RNGs.
- Activation Functions: The choice of activation functions does not seem to significantly affect either accuracy or training. Neither choice seems to be better than the other.

Submission: Submit the following to Canvas:

- Your note file *readme.(txt | pdf)*, which includes your answers to Q1, Q2, Q4, Q5, and Tables 1-2, and any notes that you want the TA to read.
- hw.tar.gz that includes two python files specified in dropbox/22-23/572/hw9/submit-file-list.
- Make sure that you run **check_hw9.sh** before submitting your hw.tar.gz.