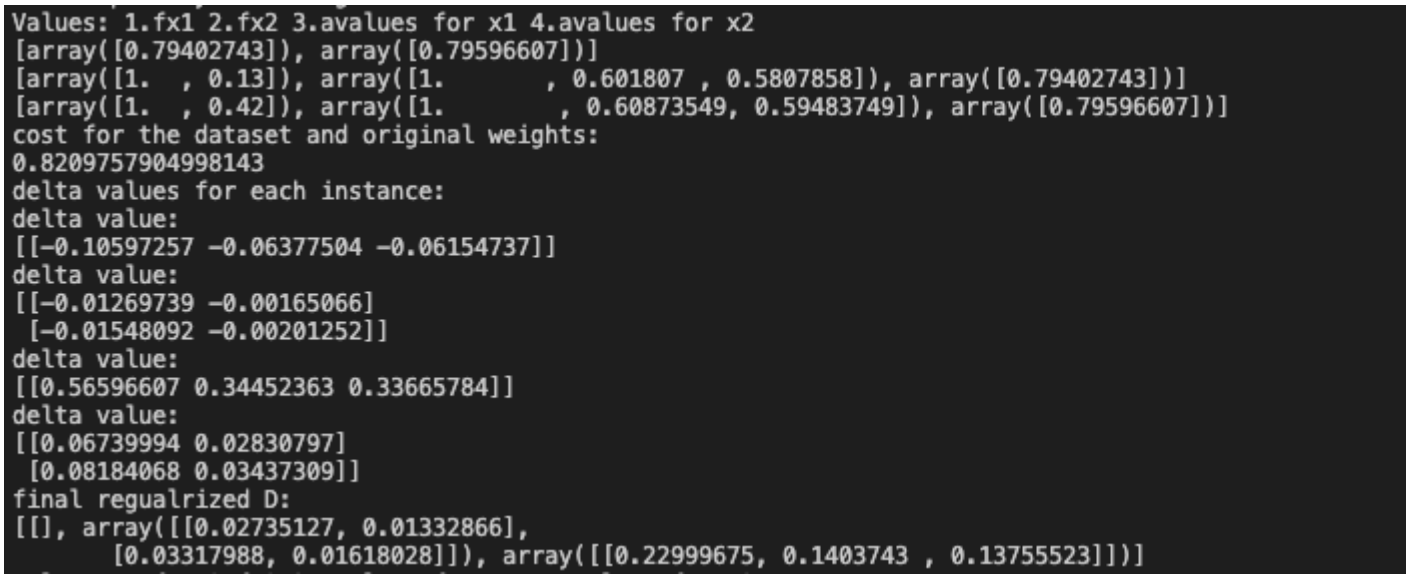


Figure 1: benchmark 1

1 benchmark

When run the main.py, benchmark would print out like above and below.

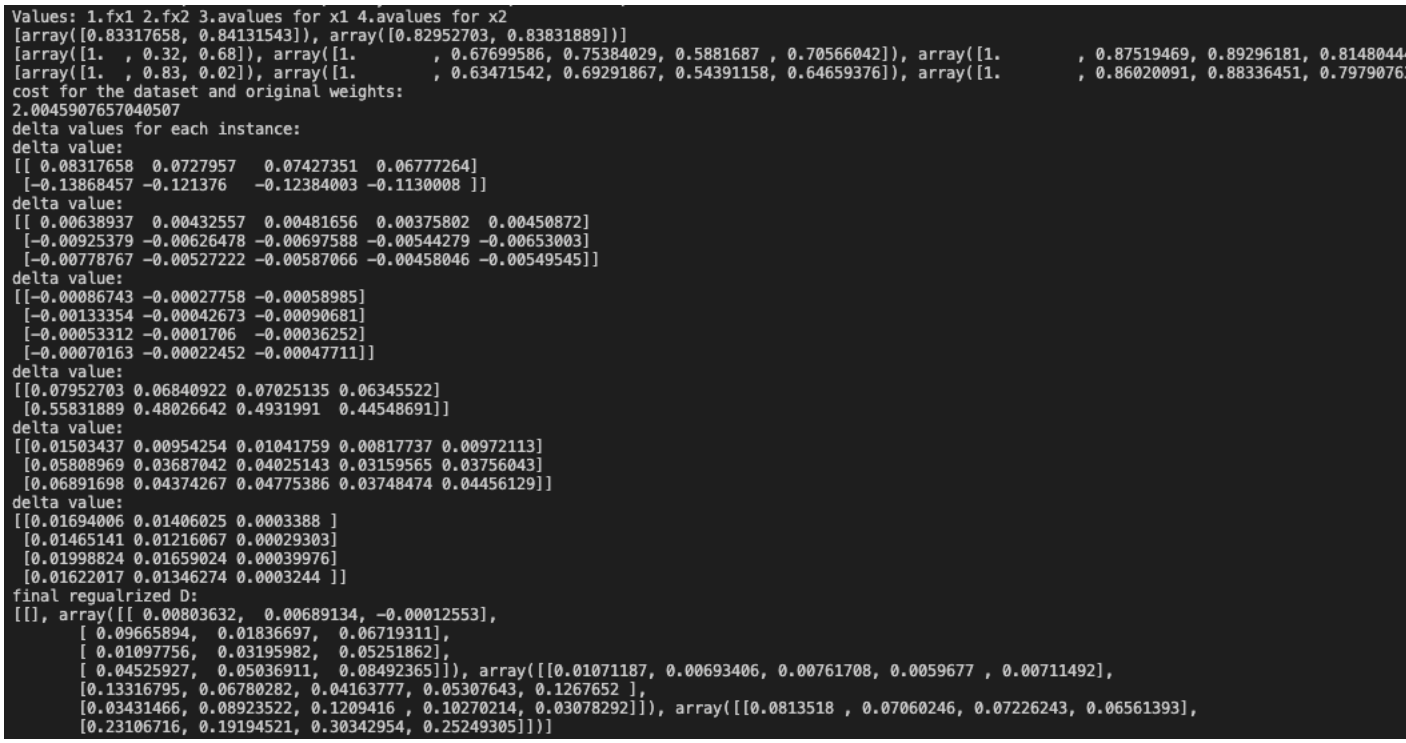


Figure 2: benchmark 2

2 Deliverables and Experiments

The main objective of this homework is to study how the performance of a trained neural network is affected by *(i)* the neural network architecture (i.e., by the number of layers and neurons); and *(ii)* by the regularization parameter.

When implementing your neural network, do not forget to add a *bias* input to each neuron, and to ensure that updates to bias weights are performed correctly if regularization is used. For more information, please see the slides of lecture 19. Also, do not forget to properly normalize the attributes of training and test instances whenever appropriate/necessary.

You are free to choose between two simple stopping criteria. You may, for instance, *(1)* stop if (after presenting all training instances to the network and updating its weights) the cost function J improves by less than some small user-adjusted constant ϵ; or *(2)* stop after a constant, pre-defined number k of iterations—where each iteration consists of presenting all instances of the training set to the network, computing gradients, and updating the network's weights. You are free to choose which criterion you would like to use. *In all questions below, do not forget to mention explicitly which criterion you used and what was its corresponding hyper-parameter (i.e., ϵ or k). We encourage you to try different possibilities to identify the setting that produces the best possible performance for your algorithm.*

After verifying and ensuring the correctness of your solution (by comparing the outputs produced by your backpropagation implementation with the step-by-step examples we provided; please see Section 2.1), you will be analyzing two datasets:

(1) The Wine Dataset The goal, here, is to predict the type of a wine based on its chemical contents. The dataset is composed of 178 instances. Each instance is described by 13 *numerical* attributes, and there are 3 classes.

(2) The 1984 United States Congressional Voting Dataset The goal, here, is to predict the party (Democrat or Republican) of each U.S. House of Representatives Congressperson. The dataset is composed of 435 instances. Each instance is described by 16 *categorical* attributes, and there are 2 classes.

2.1 Correctness Verification

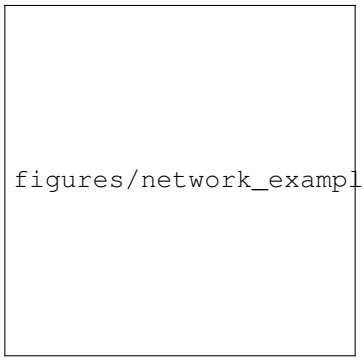
You will first have to verify that your implementation of backpropagation is correct; that is, that the gradients it computes are accurate. To do this, we are providing each student with two files: *backprop_example1.txt* and *backprop_example2.txt*. Each of these files describes a particular neural network architecture and one (minimal) training set. Each file shows, in detail, all intermediate quantities computed by the backpropagation algorithm, which you will then compare with those produced by your algorithm. These quantities include information such as the activation of each neuron when the network is presented with a given input, the final predicted output produced by the network (and the corresponding expected output), the error/cost function, J, of the network, the δ values of each neuron, and the gradients of all weights.

You *must* ensure that your implementation of backpropagation is correct. To do so, you should write a function capable of reproducing the quantities described in each of the two benchmark files. This function should produce textual output (e.g., by using simple *print* statements) indicating all relevant quantities computed by your implementation of backpropagation—similarly to how they are presented in the provided benchmark files. In other words, as part of your solution to this assignment, you *must* allow us to call this function (that is, the function that shows that your code can reproduce the outputs presented in *backprop_example1.txt* and *backprop_example2.txt*), so that we can verify the correctness of your code. The output itself that you generate does not have to be identical to that shown in the benchmark files, but it should include at least, for each training instance and for both datasets: the activation of each neuron; the final predicted output of the network; the expected output; the cost, J, associated with that particular instance; the δ values of each neuron and the gradients of all weights after the network is presented with that training instance; and the final (regularized) gradients after the backpropagation algorithm is done processing all instances in the training set.

You should then *(i)* include in your report instructions describing how we (the instructor, staff, and TA's) should run the function you implemented to demonstrate the correctness of your backpropagation algorithm; i.e., the function that produces all results/quantities described above; and *(ii) present* in your report the textual output produced by this function, for both provided benchmark files. This part of the assignment is extremely important, and a non-trivial amount of your final grade will depend on us being able to verify that you implemented the backpropagation training algorithm correctly. We strongly recommend that you only start working on the next part of this assignment *after* you have verified that your implementation of backpropagation is correct. Otherwise, it may be hard (or impossible) for your neural network to properly learn solutions to the proposed problems.

The two benchmark files we provide (*backprop_example1.txt* and *backprop_example2.txt*) are structured as follows. The first line of each file specifies the regularization parameter, λ, that should be used. After that, you will find a line describing the network structure—the number of neurons in each layer. Recall that the first layer corresponds to the network's input layer and that the last layer corresponds to the output layer. In the notation we adopted here, the bias neuron is *not* included in

the neuron counts shown in this line. As an example, consider a neural network with two inputs, one hidden layer with 3 neurons, and one output. In this case, the second line of the provided files would define the network's structure as being [2 3 1]. Notice, however, that (as always) your implementation *should* include bias neurons/weights. We do not typically count them when defining/specifying the architecture of a network because it is implicit that they are always being used. After this section of the file, you will encounter a description of the initial/current weights of the network. Here, each row represents the weights of a given neuron in one particular layer; the weights of the i-th neuron are stored in the i-th row of the corresponding table/matrix. Recall that the first weight of each neuron corresponds to the bias weight connected to that neuron. Consider the following simple neural network as an example:



The weights $\theta^{l=1}$ (Theta1) would be shown as follows:

```
0.4 0.1
0.3 0.2
```

The weights $\theta^{l=2}$ (Theta2) would be shown as follows:

```
0.7 0.5 0.6
```

After this section of the file, you will find a description of the inputs (x) and expected outputs (y) of each instance in a training set. The subsequent parts of the file show the intermediate quantities computed by the forward propagation process when performed on each of the training instances. In particular, this part of the file shows the z and a values of each neuron (i.e., their activations), as well as the network's final prediction, $f(x)$, and the cost function, J, associated with that particular instance. The final section of each presents all intermediate quantities computed by the algorithm during the backpropagation phase: the δ values of each neuron, after processing a particular training instance, and the gradients of all weights after processing that instance. After that, you will find the final (regularized) gradients of all weights, computed based on all training instances. As discussed in class, these gradients correspond to the gradients based on which we should update the network's weights so that it (on average) makes better predictions for all training examples in the dataset.

2.2 Experiments and Analyses

For each dataset, you should:

1. Train a neural network and evaluate it using the stratified cross-validation technique discussed in class. You should train neural networks using different values for the regularization parameter, λ, and using different architectures. You can decide which architectures you will evaluate. As an example, consider testing networks with 1, 2, 3, and 4 hidden layers, and with various numbers of neurons per layer: e.g., 2, 4, 8, 16.

   House Votes Dataset NeuronNetwork Structure: [16,10,6,3,2],[16,10,8,4,2],[16,10,8,2],[16,8,4,2],[16,10,2],[16,4,2]

   Lamda values: 0.1, 0.2, 0.25, 0.25, 0.2, 0.25

   Wine Dataset NeuronNetwork Structure: [13,10,6,3,2],[13,10,8,4,2],[13,10,8,2],[13,8,4,2],[13,10,2],[13,4,2]

   Lamda values: 0.1, 0.2, 0.25, 0.25, 0.2, 0.25

2. For each trained neural network (i.e., for each combination of architecture and regularization that you tested), you should measure the resulting model's accuracy and F1 score.

3. Based on these experiments, you should create, for each dataset and for each of the metrics described above, a table summarizing the corresponding results (i.e., you should show the value of each performance metric for each type of neural network that you tested, on both datasets). **You should test *at least* 6 neural network architectures on each dataset**.

   For House Vote Dataset:

   1. Structure: [16,10,6,3,2]; lamda=0.10: acc = 0.9427061; f1 = 0.9444301
   2. Structure: [16,10,8,4,2]; lamda=0.20: acc = 0.9455180, f1 = 0.9461446
   3. Structure: [16,10,8,2]; lamda=0.25: acc = 0.9546537, f1 = 0.9571486
   4. Structure: [16,8,4,2]; lamda=0.20: acc = 0.9571812, f1 = 0.9528124
   5. Structure: [16,10,2]; lamda=0.20: acc = 0.9400609, f1 = 0.9420825
   6. Structure: [16,8,2]; lamda=0.25: acc = 0.9418693, f1 = 0.9436791

   For Wine Dataset:

   1. Structure: [13,10,6,3,3]; lamda=0.10: acc = 0.9777778; f1 = 0.9800510
   2. Structure: [13,10,8,4,3]; lamda=0.20: acc = 0.9777778, f1 = 0.9800510
   3. Structure: [13,10,8,3]; lamda=0.25: acc = 0.9833333, f1 = 0.9852115
   4. Structure: [13,8,4,3]; lamda=0.20: acc = 0.9833333, f1 = 0.9852115
   5. Structure: [13,8,3]; lamda=0.20: acc = 0.9888889, f1 = 0.9899734
   6. Structure: [13,6,3]; lamda=0.25: acc = 0.9888889, f1 = 0.9899734

4. Discuss (on a high level) what contributed the most to improving performance: changing the regularization parameter; adding more layers; having deeper networks with many layers but few neurons per layer? designing networks with few layers but many neurons per layer? Discuss any patterns that you may have encountered. Also, discuss whether there is a point where constructing and training more "sophisticated"/complex networks—i.e., larger networks—no longer improves performance (or worsens performance).

   I would say that it would perform better when the network has few layers but many neurons per layer. This is because we are basically using the back propagation to adjust weight connecting each layers. With more neurons in each layer, the weights at that layer would be more and which, as a result, a better adjust of weights during each time of processing data. In this way, it would perform better.

   Also, I think we should not have too many layers for these two datasets since they do not have complicate attributes. But if a network with too many layer be used to train by this dataset, it would have a good performance, but it is due to the over fit which means that the performance

is not actually good. It is kind of like that, with more layer in the network, the network could describe the data better which make it perform well on training instances, but may have a bad performance on new comming data.

5. Based on the analyses above, discuss which neural network architecture you would select if you had to deploy such a classifier in real life. Explain your reasoning.

   For house vote, F1 is a more important scale for this model, in this way, model 3 with structure: [16,10,8,2] and lamda 0.25.

   For wine, accuracy is a more important scale for the model, in this way, model 5 with strucuture: [13,8,3] and lamda 0.2.

6. After identifying the best neural network architecture for each one of the datasets, train it once again on the corresponding dataset and create a learning curve where the y axis shows the network's performance (J) on a test set, and where the x axis indicates the number of training samples given to the network. This graph intuitively represents something like this: after showing 5 training examples to the network, what is its performance J on the test set? After showing 10 training examples to the network, what is its performance J on the test set? If the network's parameters and architecture are well-adjusted, this graph should indicate that the network's performance improves as the number of training examples grows; in particular, that J *decreases* as a function of the number of training instances presented to the network. Please also report the step size value, α, you used when training this network.

7. Although this last step/task is not required, we recommend that you train the network using the mini-batch gradient descent approach. You can manually select and adjust the mini-batch size that you would like to use. Training a neural network in this way significantly accelerates the training process.

3 Some Hints

1. Initialize the network's weights with small random numbers from -1 to +1 or random numbers sampled from a Gaussian distribution with zero mean and variance equal to 1.

2. When trying to identify effective network architectures, start your investigation by testing networks with few layers (e.g., try, first, networks with just one hidden layer). Increase the number of layers—and/or the number of neurons per layer—only when necessary.

3. When training a given neural network architecture, try, first, to use larger step sizes, α. If you set α too small, the training process may become very slow or prematurely converge to a bad local minimum (depending on the stopping criterion you use). By contrast, if the network's performance oscillates significantly during training when using large values of α, you might be overshooting. In this case, decrease the value of α. Repeat this process until you find the largest step size that can be effectively used—i.e., the value that allows the weights to be updated fairly quickly, but that does not cause weight instability or divergence.

4. After you identify a value of α that causes the network's performance to improve in a reasonably consistent way, check the performance (value of J) to which the network converges. If it is not satisfactory, your network architecture might be too simple (i.e., the network might be underfitting). In this case, try to increase the number of layers and/or the number of neurons per layer. Then, repeat the analyses above regarding how to evaluate different step sizes, α.

There are four ways in which we may receive extra credit in this homework.

(Extra Points #1: 13 Points) Implement the vectorized form of backpropagation. As previously mentioned, this is not mandatory, but you can get extra credit if you do it. Most modern implementations of neural networks use vectorization, so it is useful for you to familiarize yourself with this type of approach.

I implement the verctorized form of backpropagation. (self.backP in nn.py).

(Extra Points #2: 13 Points) Analyze a third dataset: the **Breast Cancer Dataset**. The goal, here, is to classify whether tissue removed via a biopsy indicates whether a person may or may not have breast cancer. There are 699 instances in this dataset. Each instance is described by 9 *numerical* attributes, and there are 2 classes. You should present the same analyses and graphs as discussed above. This dataset can be found in the same zip file as the two main datasets.

1. Structure: [9,6,4,2]; lamda=0.10: acc = 0.9615378; f1 = 0.9615350
2. Structure: [9,5,3,2]; lamda=0.20: acc = 0.9566281, f1 = 0.9637510
3. Structure: [9,8,2]; lamda=0.25: acc = 0.9600183, f1 = 0.9752115
4. Structure: [9,4,2]; lamda=0.20: acc = 0.9587343, f1 = 0.9503063
5. Structure: [9,18,6,2]; lamda=0.20: acc = 0.9894389, f1 = 0.9838174
6. Structure: [9,27,9,3,2]; lamda=0.25: acc = 0.9879743, f1 = 0.981772

(Extra Points #3: 13 Points) Analyze a fourth, more challenging dataset: the **Contraceptive Method Choice Dataset**. The goal, here, is to predict the type of contraceptive method used by a person based on many attributes describing that person. This dataset is more challenging because it combines *both numerical and categorical attributes*. There are 1473 instances in this dataset. Each instance is described by 9 attributes, and there are 3 classes. The dataset can be downloaded here. You should present the same analyses and graphs discussed above.

(Extra Points #4: 13 Points) Implement the method (discussed in class) that allows you to *numerically* check the gradients computed by backpropagation. This will allow you to further ensure that all gradients computed by your implementation of backpropagation are correct; that way, you will be confident that you trust your implementation of the neural network training procedure. You should present in your report the estimated gradients for the two neural networks described on the provided benchmark files. First, estimate the gradients using $\epsilon = 0.1$, and then $\epsilon = 0.000001$. If you choose to work on this extra-credit task, please include the corresponding source code along with your Gradescope submission.