CS158 - Assignment 8
Neural Networks
Due: Sunday, April 3rd by 11:59pm

https://www.smbc-comics.com/comic/bayesian

For this assignment we're going to be adding our last classifier to our collection, a two-layer neural network. Before starting this assignment, make sure that you understand how backpropagation works. The notes are probably the best resources, but the book and the alternate reading will also be helpful.

Starter

You can find the starter for this assignment at

`http://www.cs.pomona.edu/classes/cs158/assignments/assign8/assign8-starter.zip`

It is identical to the starter for assignment 6 (and, therefore, also assignment 7) except I have added two methods to the `DataSet` class:

- `getCopyWithBias`: Returns a new copy of the dataset where a "bias" feature has been added to each example, i.e. the feature size is increased by one and each example in the dataset has an additional feature with the value 1.0. Note that this creates a completely new copy of the dataset including the examples and leaves the dataset that you call it on unchanged. This function will be useful during training.
- `addBiasFeature`: If you have a newly created a dataset with the additional bias feature, you may also want to be able to take a new example from the original dataset (without the bias) and add the bias, e.g. if you want to classify a new example. This method is called on a dataset where the bias has been added and will add the bias to the new example.

Take a look at these two methods and make sure you understand what they do. You may want to use them in your neural network code, though there are also other ways to handle the bias.

Specifications

Implement a class called `TwoLayerNN` that implements the `Classifier` interface. This class should implement a two layer network with a variable number of hidden nodes and one output node. Your class should meet the following specifications:

- Have a constructor that takes the number of hidden nodes as input as an `int`.
- Have a method called `setEta` that sets η, the learning rate, for the network. By default, set $\eta = 0.1$.
- Have a method called `setIterations` that sets the number of times to iterate over the training data during training. By default, set the number of iterations to 200.
- You neural network should use the *tanh* activation function for all nodes/neurons and they should all include a bias (both the hidden layer and the output layer).
- You should initialize all of your network weights with random values between -0.1 and 0.1.
- The `classify` method should still output a 1 or -1. Any network where the result of the output node is greater than 0 should be 1 and everything else -1.
- The `confidence` for the classifier should be the absolute value of the output, i.e. how far away from 0 is the prediction.
- You may assume that the dataset has features from 0 up to `getAllFeatureIndices().size()-1`. You should not, however, assume that all examples have all the features specified. If a feature is not available for an example, assume the value is 0.

- Both the dataset given to train and the example given to classify will *not* have an extra bias feature. Your classifier will need to handle this internally.

An Example

Neural networks are notoriously hard to debug (and, therefore, to get right). I strongly encourage you to work through an example by hand. To help you in this endeavor, I provide here the values for one pass through training a network. You can use this data in a few ways:

1. You can use it to work through an example by hand to make sure you understand the algorithm. It may take you an hour, but it will save you hours in debugging time.
2. If you work through this example by hand, you can also modify it to, for example, ignore the bias. This can also help you debug your program incrementally.
3. You can use this example to help debug your code and as *one* check that your code is working correctly. Note, the numbers should be more or less *exactly* the same if you're checking your code. If not, even if they're off by a little bit, it likely means you have a mistake.

Starting network:

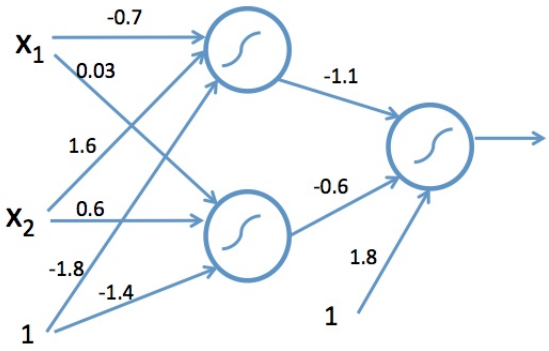


Figure 1: Starting network

If the model is then trained with the example $[1, 1]$ with label 0 with $\eta = 0.5$ (a larger value than the default to see more change), you would get the following output from the nodes:

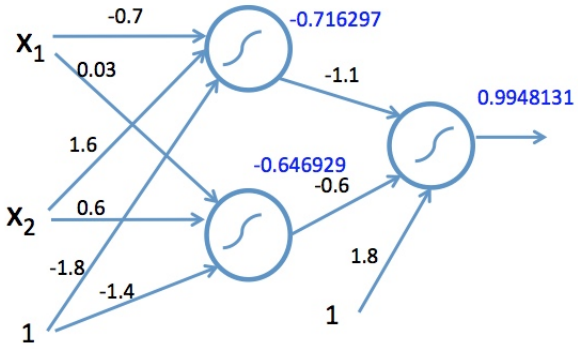


Figure 2: Node outputs

Finally, using these outputs, the weights can all be updated resulting in the new network of:

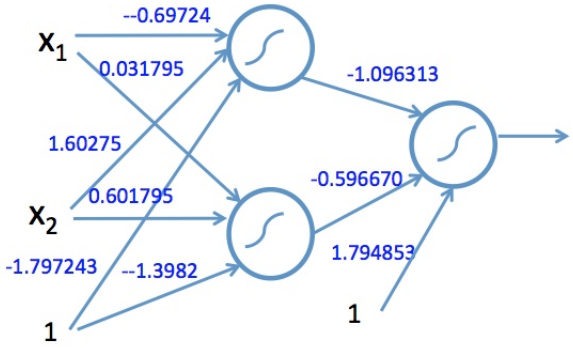


Figure 3: Updated weights after one iteration

Hints/Advice

- On this assignment more than probably any other we've seen so far it is critical that you have a working example where you know what all of the values are supposed to be, ideally multiple examples with increasing complexity. Do not hope to just keep changing things in your code until you get it to work correctly.
- Relatedly, print out *lots* of information as you are developing. Of course, remove these print statements before you turn it in.
- Adding in the bias can add another layer of confusion. My advice is to first work through an example by hand without the bias and get your code working on this example. Then, add in the bias and try and get the full implementation working.

  You may add in the bias however makes the most sense for you, but my advice is:

- Use the new `DataSet` methods to add a bias value to the training set. If you do it right at the beginning of the `train` method you should be able to handle the hidden layer bias without changing the code at all.
- When you initialize the weights, make sure that you add extra weights for the bias. If you initialize the weights *after* you've added the extra bias feature to the training data it should only require adding one more weight to the output layer weights.
- Handle the bias on the output node. It's only one node, so it should only require a few minor changes. This you will have to add programmatically.
- Don't forget that you'll also need to deal with the bias during classification. Again, the `DataSet` methods should help you with this.

Experiments

In a file called `experiments` (pick some reasonable file type) include the analysis below (explicitly number each part in your writeup). For all of the experiments, use the titanic dataset *with binary features*, i.e., `titanic-train.csv`.

1. Above, I gave you an example network and showed what would happen if you trained the network on one example. Starting at the original network (Figure 1) using $\eta = 0.5$, calculate the following *using your code*:

   (a) What are the node outputs if we input an example $[0.5, 0.2]$ with the label -1? You should have three values, two for the hidden node outputs and one for the output node.

   (b) What are the final weights if we train for one iteration on the example $[0.5, 0.2]$ with the label -1? Write your input weights as a matrix, with the first row, the weights into the hidden node, the second row the weights into the second hidden row, etc. For example, the weights for the network in Figure 3 would be:

   ```
   Hidden weights:
    x1         x2        bias
   [-0.69724, 1.60275, -1.797243]
   [0.031795, 0.60179, -1.3982]

   Output weights:
    v1          v2          bias
   [-1.096313, -0.596670, 1.794853]
   ```

2. Create a plot that has the number of training iterations on the x-axis and then plots three things: 1) the sum of the squared errors for the iteration, i.e. $\sum_x (y - \hat{y})^2$ (this is the loss function that you're trying to minimize), 2) the training accuracy, and 3) the testing accuracy. Use the default parameters and three hidden nodes.

   Do this for one random 90/10 split of the data (i.e. generate the values as the model is training). Since the errors and the accuracies are on two different scales you can either have

two y-axes (left and right) or you can normalize the loss values so that they're on a scale from 0 to 1. What we want to visualize is how does the loss correlate with the other two.

Write a few sentences describing this data. Anything interesting? Do you see signs of overfitting?

3. Experiment using 10-fold cross validation on how the number of hidden nodes impacts the performance. Create a table with values from 1 to 10 hidden nodes and include training and test averages with the default parameters. Note this can take a few minutes to do since training the models, particularly as the number of hidden nodes goes up, can be slow.

   Write a few sentences describing the data. What size network would you use?

4. Experiment with some of the model parameters (i.e. number of hidden nodes, η and the number of iterations) to try and find the best model possible. State how you experimented, the final best model and it's 10-fold accuracy, and any other observations that you had.

Extra Credit

For those who would like to experiment (and push themselves) a bit more (and of course, get a bit of extra credit) you can try out some of these extra credit options. If you try out these options, include an extra file called `extra.txt` that describes what extra credit you did.

- Add in the ability to switch between the *tanh* activation function and the sigmoid activation function. Add two methods `setTanhActivation` and `setSigmoidActivation` to switch between them. Do a small experiment to show how their performance differs. *Note:* since the sigmoid ranges from 0 to 1, you'll need to change the negative label to 0 on the Titanic dataset.

- Add momentum to the network. Add a method called `setMomentum` that sets the momentum value. Do a small experiment to show how momentum affects the performance.

- Some other NN feature we talked about in class. If you do these either 1) make sure that your network still meets the specifications above or 2) create a separate copy with your additions. Do a small experiment to highlight the effect of your addition.

When You're Done

Make sure that your code compiles, that your files are named as specified and that you have followed the specifications exactly (i.e. method names, number of parameters, etc.).

Create a directory with your last name, followed by the assignment number, for example, for this assignment mine would be `kauchak8`. If you worked with a partner, put both last names.

Inside this directory, create a `code` directory and copy all of your code into this directory, maintaining the package structure.

Finally, also include your `writeup` file.

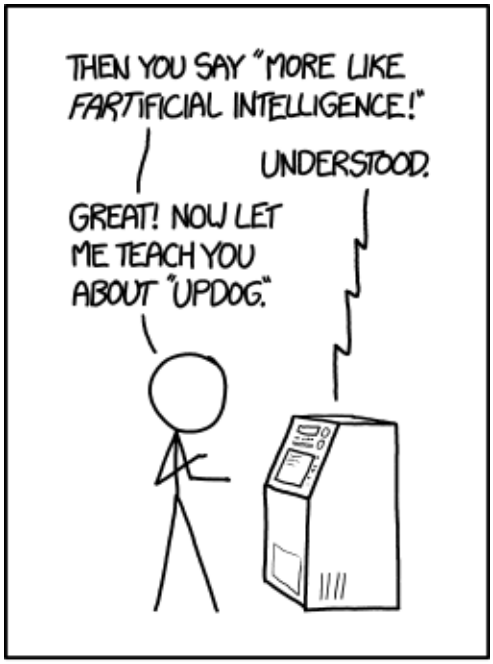
`zip` this folder and submit that file.

Commenting and code style

Your code should be commented appropriately (though you don't need to go overboard). The most important things:

- Your name (or names) and the assignment number should be at the top of each file
- Each class and method should have an appropriate JavaDoc
- If anything is complicated, it should include some comments.

There are many possible ways to approach this problem, which makes code style and comments very important here so that I can understand what you did. For this reason, you will lose points for poorly commented or poorly organized code.



http://xkcd.com/1696/