CSC 380 Assignment 4: Neural Networks (v1)

Learning Objectives

In this project, you will gain hands on experience with neural networks. You will create a network, give it training and test data, train it, and report your results.

After doing this project you should be able to:

- Describe how artificial neural networks learn from data,
- Describe the relationship between the complexity of the learned function and the complexity of the structure of the network,
- Create neural networks to learn various tasks, and
- Analyze the trade-off between performance on the training and test sets.

What to do

Set up

You will use an app developed at the University of British Columbia, as part of their AIspace tools. Follow these setup steps:

1. Install java if you don't already have it. (But if you're not into programming in Java, don't worry – that's not required for this project.)
2. Download the jar version of the Neural Applet, neural.jar.
3. You should be able to start the system from a command line (assuming that your computer knows how to start java, and that you're in the same directory as the jar file) like this:

```
java -jar neural.jar
```

If that didn't work, here's some other things you can try, depending on what type of computer you're using:

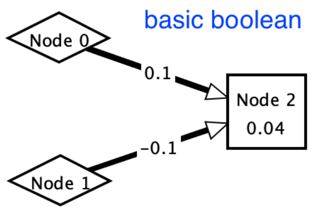
- PC: Download the exe file. Here also, you may have to tell your computer that it's okay to run a file you got from the internet.
- Mac: You may have to give permission for this to run: Go to System Preferences > Security and Privacy > General, and either temporarily "Allow Apps downloaded from anywhere", or if it says that neural.jar was prevented from opening because it came from the web, you can tell it to open anyway.

Starter network for OR and XOR.

First, use the applet to create a network that computes the logical OR function.

1. Network creation

   - The applet starts in "Create" mode, with "Create Node" the default action.
   - Click somewhere on the canvas (large empty rectangle) to create a node. Select OK to choose the default parameters for the node.
   - For OR, you'll need 2 input nodes and one node for output. *For ease of grading, put the inputs on the left and the output on the right. You can move them after creating them with the Select tool.*
   - Click on the "Create Edge" tool.
   - Click on a node that the edge (link) will go from.
   - Click on a node that the edge (link) will go to.
   - Node shapes change automatically to indicate that they are input (diamond), hidden (oval), or output (rectangle) nodes.
   - Connect the two inputs to the output. Your network should look roughly like this:



2. Creating the data

   - Click `View/Edit Examples`, then `Add New` (under Training examples) to make the 4 different examples you need for OR, assuming 0 = False and 1 = True. Each example is essentially a row in the truth table:

Node 0	Node 1	Node 2 (output)
0	0	0
0	1	1
1	0	1
1	1	1

     – *Note that normally you would want to put some of the data into the test set, but since we only have 4 items here, we'll forego that for now.*

3. Training the network

   - Close that window, and click on the `Solve` "tab" (underneath the greyed out Stop sign).
   - Click `Initialize Parameters` to assign random values to the weights and biases (thresholds). (In this case, there's just one bias – on the output node.)
   - Now click `Step to Target Error`. The target error defaults to 0.1 on the training set.
   - Note how many steps it took to finish training.

- Click `Show Plot` to see the "Total Error Plot".
- Take a screenshot of the trained network and the error plot to include in your report.

4. Starting the document
   - Create your document for submission, and include the heading: **Part 1, OR network**
   - Include your above-mentioned screenshots, and answer the following question:
     - How did the combination of the weights on the edges and the bias on the output node ensure that the network would get the right answer (output) for the training examples?

5. Now repeat steps 2-4 for the XOR function: $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. You can go to Create mode, and edit the examples instead of creating new ones.
   - Train the network. (Remember to initialize the Parameters first.) You'll probably want to set the "Auto Step Speed" to "Very Fast" under `Neural Options`. If it doesn't reach the target error within about 30,000 epochs (steps), it's probably not going to, so stop the training, reinitialize, and try again a few times.
   - Add hidden nodes to your network. Go back to Create mode, add another node below your original output node, create a new output node, and connect the nodes appropriately: both inputs to both hidden layer nodes, and both hidden layer nodes to the new output node. Now repeat the training procedure.
   - In your report, under the heading, **Part 2: XOR network**:
     - If it worked (trained successfully), include your network and error screenshots (scaled appropriately – they don't need to be huge). *Ensure that all of the weights are visible by moving the nodes around if necessary.* **Also** answer the same question above for this network.
     - If it never reached the target error, describe in detail what you did, and include the screenshot of your network.

Mirror symmetry

A slightly more complex task for neural networks was described in "Learning Representations by Back-Propagating Errors", by Rumelhart, Hinton, and Williams in Nature (1986). In this task, a string of six bits (0 or 1) is judged for "mirror symmetry" (i.e., a pattern of ABCCBA, or palindrome). An output of 1 indicates that the first three bits, when reversed, match the last three bits. An output of 0 indicates that they do not match. The network structure that they used is shown below, along with a description. Study the network, especially with regards to the weights and the ratios between the weights from top to bottom and left to right.

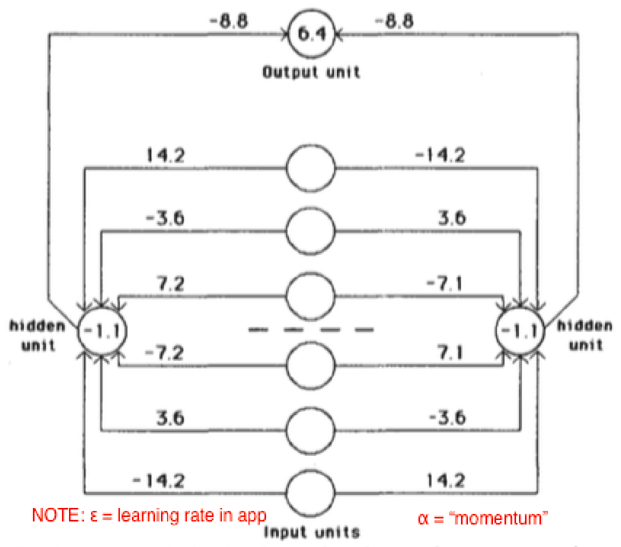


Fig. 1 A network that has learned to detect mirror symmetry in the input vector. The numbers on the arcs are weights and the numbers inside the nodes are biases. The learning required 1,425 sweeps through the set of 64 possible input vectors, with the weights being adjusted on the basis of the accumulated gradient after each sweep. The values of the parameters in equation (9) were $\varepsilon = 0.1$ and $\alpha = 0.9$. The initial weights were random and were uniformly distributed between -0.3 and 0.3. The key property of this solution is that for a given hidden unit, weights that are symmetric about the middle of the input vector are equal in magnitude and opposite in sign. So if a symmetrical pattern is presented, both hidden units will receive a net input of 0 from the input units, and, because the hidden units have a negative bias, both will be off. In this case the output unit, having a positive bias, will be on. Note that the weights on each side of the midpoint are in the ratio 1:2:4. This ensures that each of the eight patterns that can occur above the midpoint sends a unique activation sum to each hidden unit, so the only pattern below the midpoint that can exactly balance this sum is the symmetrical one. For all non-symmetrical patterns, both hidden units will receive non-zero activations from the input units. The two hidden units have identical patterns of weights but with opposite signs, so for every non-symmetric pattern one hidden unit will come on and suppress the output unit.

Create a network (including nodes and edges) that follows the architecture of the Rumelhart et al example. *Note: As you're doing this, make sure you keep careful track of which screenshots go with which tasks. Putting annotation on the files and giving them meaningful names should help.*

1. Use the "File" > "Load data from file" option with the file `symmdata.csv`, which contains the 64 examples for this task.. In the "Data Set Parameter Input" window that pops up, enter your names for the (6) inputs and one output separated by commas, e.g. "i0,i1,i2,i3,i4,i5,o1". In the next window, "Neural Network Construction", tell it how many layers of hidden nodes you want, and how many nodes in each layer. For the first experiment, you should mimic the network from the paper, so you should have one layer of hidden nodes with 2 nodes. Ensure that your output node is selected as output. (None of them should be "ordered".)

2. Arrange the nodes to roughly follow the layout shown in the example above.

3. Go into Solve mode.

4. As discussed in class, with machine learning, a small portion (10% or so) of the training data is reserved (i.e., kept out of the training process by moving it into the test set), in order to

keep the network from "memorizing the training set" and to test the network's generalization. Do this by clicking `View/Edit Examples`, and under Training examples, choose `Select % of Examples` in the choice menu, and `Random`, and enter 10. Then click the right arrow to move the selected examples to the test set.

5. Set some more `Neural Options`
   - `Neural Options > Learning Options`: Set the Learning Rate and Momentum to match those from the Rumelhart et al.\ example.
   - `Neural Options > Parameter Initialization Options`: Same here (Random with Bound: 0.3)
6. Click `Initialize Parameters` to give an initial weight to each edge.
7. Train by clicking Step to Target Error
8. Click on Show plot (even while training). Training should stop when the error on the Training set gets below 0.1.
9. Again, if it doesn't stop by 20,000 steps or so, Stop, Reinitialize, and start again.

After training, analyze your results and include in your report under the heading **Part 3: Mirror Symmetry** answers to the following questions and supporting screenshots:

1. Look at the hidden nodes to view their bias and outgoing weights.
   - How do the weights and biases ensure that the network will get the right answers?
   - How do they compare to those in the Rumelhart et al example? (Don't just look at the absolute numbers, but the relationships between them too.)
2. Click `Show Plot` to see the training and test error. Did the test error and training error both go down?
3. Click on Summary Statistics to see which items from the test set were (un)successfully classified.
   - Were they all correct?
   - What do you think would be an "acceptable" error rate for a task like this?
4. In this task, how many positive and negative items are there? How do you think that affects the training?

Another symmetry

Repeat the same procedure as above using the data file `abcabc.csv`. In your report under the heading **Part 4: Another Symmetry**, describe how this task is different from mirror symmetry and how that is reflected in the learned weights. Also answer the same questions as above and provide supporting screenshots.

Symmetry gone wild

One more task: The file `symmdata-plus.csv` is like the original symmetry data, but it has 8 inputs instead of 6. The first and last inputs are pseudo-random numbers (generated by me). The rest of the input and the output are the same as in symmdata.csv. In principle, a network should be able to learn this task by simply ignoring the first and last inputs (i.e., learning to give them a very low weight), and paying attention to the rest. Let's see what you can do with it, including your results in your report under the heading: **Part 5: Symmetry Gone Wild**

First try the same structure (except for number of input nodes) and procedure as above. You may have to repeat it several times. (You can stop any run over 50,000 steps.) If you find a solution, document it as above, paying special attention to how it performs on the test set.

Regardless of whether or not that succeeds, try some variations on the network and/or settings to see if you can come up with a better solution (as measured by speed of training or accuracy of performance on the test set). For example, you could:

- Change the structure of the network by adding additional hidden nodes or additional hidden layers,
- Change the Parameter Initialization and or Learning Options.

Notes:

- Because you normally start training a neural network with random weights (via the "Initialize Parameters" button here), different runs may produce different results. So it's a very good idea to try any different configuration multiple (at least a few) times.
- It's very important to keep good notes while you're experimenting, and to carefully match up the saved images with the descriptions of your attempts. Science!

In your report, document what you did in your experimentation, but include screenshots only for interesting results. For less interesting results, include a table describing the things you tried and the outcomes.

What to hand in

You should submit a single report that documents your experience and your analyses. Your report should include the labeled sections as described above.

Grading

This project is worth 12 points. It will be graded on how well you follow the instructions above, on the depth of your insights, and on your demonstrated understanding of the mechanics of neural networks.