Connectionism: A Hands-On Approach

Michael R. W. Dawson
Biological Computation Project
Department of Psychology
University of Alberta

Monday, April 5, 2004

Word Count: 80,479

Please don't quote – this is just a draft.

Chapter 27: Creating Your Own Training Sets

27.1 BACKGROUND

In using the software to do the exercises in this book, you have been working on training sets that I have provided you. Of course, the most interesting connectionist research to conduct is to train networks on problems that are of particular interest to you. This can easily be done, by creating your own training sets in a format that can be read by the software that you have been using. This final chapter describes the general properties of the .net files that are used to train a network. We then describe the steps that the user can take to define their own training sets for further study.

27.2 DESIGNING AND BUILDING A TRAINING SET

27.2.1 DESIGN ISSUES

The typical way for a student to start in my lab is for he or she, motivated by some curiosity about neural networks, to drop by my office. Usually, the student has some interest in a core topic in cognition or perception, and they want to explore that topic using connectionism. At some point in our first meeting they will ask me whether it is possible for a PDP network to learn to do *x*, where *x* is the core topic of interest to them. Now, it is well known that the PDP networks have the same in principle power as a universal Turing machine (Dawson, 1998). So, the easy, and short, in principle answer to their question is always 'yes'. Of course, the student is more interested in the longer, and harder, practical answer to their question. This answer involves actually creating a network, which is often the topic of a thesis. To create a network, a number of different design decisions must be explored.

The first design issue requires making a decision about what specific task will be learned by a network. For example, one of my former students was interested in studying the Wason card selection task with a neural network (Leighton & Dawson, 2001). When human subjects perform this task, they are presented four different cards and a logical rule. They have to decide which cards to turn over in order to test the validity of the rule. How could we get a neural network to perform an analogous task? We decided that we would present some representation of cards and a rule to a network, and that we would train the network to show with its output units which of the four cards it would turn over if it could. In general, in order to get a neural network to do some task of interest, you have to translate the task into some sort of pattern classification problem. This is particularly true for a network of value units, because for such a network the output units need to be trained to turn on or off, and cannot be trained easily to generate intermediate values.

This leads immediately to the second design issue. How does one represent network inputs and outputs? Consider one network that was trained to solve Piaget's balance scale problem (Dawson & Zimmerman, 2003). When children are presented this task, they see a real balance scale that has pegs equally spaced along the top of its arms. Different numbers of disks can be placed on the pegs. Children must decide whether the balance scale will tip to the left, tip to the right, or balance. How can this task be converted into a pattern classification problem, and encoded for a network? We decided to represent the balance scale configurations using 20 input units. The first five units represented left weight and were *thermometer coded*. In thermometer coding, a unit was turned on for every weight that was placed on a peg. So, if two weights were on a peg, the first two input units were activated; if four weights were on a peg, then the first four input units were activated. The second set of five units represented left distance and were unary coded. With this coding, only one unit is turned on to represent which peg the weights were placed on. For example, if the third peg were to be used, then only the third input unit in this group would be activated. The same encoding was repeated for the remaining 10 input units to represent right weight (with 5 units) and right distance (with the remaining 5 units). Of course,

other input schemes could be used. We could use thermometer coding for distance as well. We could use unitary coding for the number of pegs. We could use some completely different coding scheme for both properties. One of the important things to remember about representing inputs is that there will always be a variety of different encodings, and some might be easier to use or might generate more interesting results then others.

In terms of the output response for the balance scale network, two output units were used (Dawson & Zimmerman, 2003). If the left output unit was turned on, then this meant that the balance scale would tip to the left. If the right output unit was turned on, then this indicated that the balance scale would tip to the right. Problems in which the scale balanced were represented by zero values on both output units. Again, other output encodings are possible. For instance, we could have used three output units instead of two, where the third output unit would have been used indicate that the scale balanced.

A third, and perhaps subtle, design issue concerns what responses will be made by a network to its stimuli when training is over. For example, consider the Wason card selection network (Leighton & Dawson, 2001). In one version of this network, we trained it to make responses that were logically correct. However, this task is of interest to cognitive psychologists because human subjects usually make systematic errors. In other versions of this network, we trained it to make the responses that humans would make instead of training it to respond correctly. By examining these other versions of the network, we were able to gain some insight into how humans deal with the card selection problem.

A fourth design issue concerns the pattern of connectivity in a network. In particular, how many hidden units should be used? Furthermore, should there be direct connections between the input units and the output units? Sometimes these questions will be answered by having a particular theory about how the network should solve the problem. However, it's usually the case that these questions are answered empirically. Because network interpretation is usually the primary focus of research in my lab, we usually decide to explore a problem to find the smallest number of hidden units that can be used to reliably learn a stimulus response mapping (Berkeley et al., 1995). In order to do this, you have to start out with an educated guess about the number of hidden units. For problem of moderate size, I might start out with the same number of hidden units at there are input units, or perhaps a slightly smaller number of hidden units. I would then train the network. If it learned the problem quickly, then I would reduce the number of hidden units and train it again. I would keep on doing this until I found a number of hidden units such that if I went below this number the network would not converge, but if I used this number of hidden units the network would converge. Unfortunately, there is no principled way to choose the correct number of hidden units before hand.

27.2 .2 Format For A Training Set File

After deciding on a representation for the inputs and outputs of a training set, and after deciding about some issues concerning network connectivity, you are in a position to create a training set that can be read by the software packages that you have been using while going through this book. The training set is simply a text file that provides a little bit of information about network connectivity, that provides the set of input patterns, and that provides the set of output responses. This file can be created with any word processor, and then saved as a text file. After the file has been saved, it needs to be renamed to have the extension .net. The software that you have been using will only read files that have this extension.

All of the files that can be read by the software are organized in the same fashion. The first four lines in the file provide the number of output units, the number of hidden units, the number of input units, and the total number of training patterns. For architectures that do not use hidden units, the number zero must be in that spot in the file. The next set of lines present each of the input patterns in the training set. Each row represents one input pattern. Each number in the row represents a value for an input unit. Importantly, with in a row different input values are sepa-

rated by a single space. The remaining lines in the file represent the desired network response to each input pattern. The first line in this part of the file represents the network's response to the first input pattern, the second line represents the response to the second input pattern, and so on. Within a line, each value represents the desired value for an output unit, and again adjacent values are separated by a single space.

The numbers below provide an example file for the 3-parity problem. In this example, we see that the network will have one output unit, two hidden units, three input units, and there will be eight training patterns in total. The next eight lines present the eight input patterns. Notice that each line has three values in it, one for each input unit. The final eight lines represent the responses to each of the input patterns. Note that because there is only one input unit there is only one response value per line.

```
1
2
3
8
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
0
1
1
0
1
0
0
1
```

All that one needs to do to create their own training set for the software used in this book is to create a text file that has the same general characteristics as those that were just described. The steps for doing this are:

1. Decide on a set of input pattern/output pattern pairs of interest
2. Open a word processor (e.g., the Microsoft Notepad program) to create the file
3. On separate lines, enter the number of output units, hidden units, input units, and training patterns
4. On separate rows, enter each input pattern. Remember to separate each value with a space
5. On separate rows, enter each output pattern. Remember to separate each value with a space
6. Save the file as a text file
7. In Windows, rename the file to end with the extension .net instead of the extension .txt. Remember that the software will only read in files that have the .net extension.
8. Your file can now be read in by one of the three programs that you have been using, and you can use the software to train a network to solve a problem that *you* are interested in.

27.2 .3 EXERCISE 27.1

To complete this Exercise, you will need to define a problem that could be presented to a network. You will have to say exactly how this problem would be represented. ***However, you do***

not have to build the training set for the problem. You only need to describe what it would look like if you did build it. If you wanted to take the next step and build the training set to present to one of my programs, then you could build your training set using the instructions that were given above.

1. **In a short paragraph, describe the general nature of the problem that you would like translate into some form that could be presented to one of the networks that we have dealt with in this book.**
2. **If you were to build this training set, how many output units would you need?**
3. **What would each output unit in your network represent (i.e., what representation would you use to encode network outputs)?**
4. **If you were to build this training set, how many input units would you need?**
5. **What would each input unit represent? (i.e., describe your input encoding)**
6. **How many patterns would be in your training set?**
7. **In a short paragraph, answer the following question: If you were to build this training set, what architecture would you present it to, and why would you choose this architecture?**