Neural Nets and Symbolic Reasoning

Practical exercises using the *tlearn neural network simulator*

1. Read the *tlearn* user manual

2. a. Load the AND project (see chapter 3). Train the network randomly with 100 training sweeps. Set *seed with* = 0. What is the effect of changing the learning rate. Compare learning the AND function with learning the OR function.

   b. Create a new set of nodes for the exclusive OR (you can do that by modifying the files *and.teach, and.data, and.cf* if required. Call the project XOR. Now train the network with 1000 sweeps using the parameters that you found appropriate in learning the AND (or OR) function. If the network didn't solve XOR try *resume training* for further 4000 sweeps. Do you find any differences if you use different training options: *train sequentially* vs. *train randomly*. Make use of the error display.

   c. Load XOR (chapter 4). Try out various training options and train with 10000 sweeps. Find out parameters that produce a good performance. Discuss the error display. If you found a stable solution, find out the weights after learning and draw the network including the weights. Discuss how the network solves the task.

3. a. It has been suggested that momentum can help the learning algorithm solve problems like XOR which would otherwise be very difficult to solve. Do you agree with this suggestion? Under what conditions might it be true? (Hint: try to find out optimal combinations between the learning rat and momentum; fix seed!)

   b. Reconfigure your network to use different numbers of nodes in the hidden layer, say from 1 to 5. Does increasing the number of hidden units assist the network solving the task? Try to determine the role of the individual units in solving the task.

4. a. Open the project *auto1* in chapter 5. Set the learning rate parameter to 0.3 and specify a momentum of 0.9. Specify random seed of 1 and train randomly, updating the weights after every sweep (pattern update). Train the network for 3000 sweeps. Activate the error display to examine the global error. Did the network solve the task? What global level of guarantees that tlearn has found a solution to the encoding problem?

   b. Now modify your auto-association task so that one of the input patterns is distorted. For example, you might change the input pattern 1 0 0 0 to 0.6 0 0 0.2.

In the testing options dialogue box select the *auto1.3000.wts* file which you created earlier. Has the network partially reconstructed the noisy pattern? Experiment with input patterns that have noise in different locations and in differing amounts.

c. You have seen how a network can represent 4 input patterns using 2 bits. What do you think would happen if you modified the network so that it had 5 inputs and 5 outputs, but still only 2 hidden units; and then trained it to encode the following patterns: 10000, 01000, 00100, 00010, 00001. Do you think the network could do the task? Take into account that the hidden units are **not** restricted to binary values! Construct such a network and report your results. So as not to override the work on *auto1*, call this project *auto2*.

5. Open the project *shift* (chapter 7). *Shift.data* contains 32 patterns, the first 12 contain the target string 111 while the last 20 do not. Consequently, the *shift.teach* file will have 1 as output for the first 12 pattern and 0 for the last 20.

   a. Train the network with a learning rate of 0.3 and a momentum of 0.9 for 2000 epochs (= 64000 sweeps). Be sure to choose the *train randomly* option. Has the network learned the training data? If not, start with another seed.

   b. Test the networks ability to generalize. Create a new *data* file containing novel patterns (call it *novshift.data*) using the following eight input patterns:
   0 0 0 0 0 1 1 1 1 1 1 0 0 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 0 0 1 1
   1 0 1 1 0 0 0 1 1 0 0 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1
   Test the network's response to these novel patterns. How well has the network generalized? Use the clustering procedure (in the tlearn menu click *special: cluster analysis*) on the hidden node activation patterns of the training data.

   Remark: for the cluster procedure you need the 32 activation pattern for the hidden nodes (vector file) and the name file (corresponding to *shift.data*). Use the *tlearn* editor to generate the two files (remove superfluous material, especially delete all spaces in the name file). The vector file you get by going to the *Testing options* submenu, select *shift.data* for the testing set. Then, in the network menu, chose *Probe selected nodes*. This will run the network once more sending the hidden unit outputs to the *Output* display.

   Can you tell from the grouping pattern something about the generalization which the network has inferred?

   c. Open the project *shift2*. First compare its configuration with that of the project *shift*. Notice the groupings of weights discussed in the lecture. Train the network with the same parameters as before. Test the network's response to the novel patterns. Did the network better generalize than in the case before. Again, use the clustering analysis to understand the generalization the network has performed.

Further, examine the contents of the weight file. Do you understand the network's solution?

6. Open the *letters* file in the *tlearn* folder for chapter 8. Create a file called *codes* which contains these lines:

```
b 1 1 0 0
d 1 0 1 0
g 1 0 0 1
a 0 1 0 0
i 0 0 1 0
u 0 0 0 1
```

Now with the *letters* file open and active, select the *Translate* option from the *Edit* menu and translate the *letters* file using *codes.* Call the new file *srn.data*. Next, copy this file to a file *srn.teach* and edit the file, moving the first line to the end of the file. The *srn.teach* file is now one step ahead of the *srn.data* file in the sequence. Complete the *teach* and *data* files by including the appropriate header information. Now load the *srn.cf* file who builds a 4x10x4 network with 10 context nodes.

a. Train the network with learning rate parameter to 0.1 and momentum to 0.3. Train the network with 70000 sweeps, using the *Train sequentially* option. Why it is imperative that you train sequentially and not train randomly? To see how the network is progressing, keep track of the RMS error. Why do you think the RMS error is so large?

b. Test the network using the *predtest.data* file. How well has the network learned to predict the next element in the sequence? Given a consonant, does it get the vowel identity and number correctly? (in fact, the letter sequence consisted of only three unique strings: **ba dii guuu**)

c. Investigate the network's solution by examining the hidden node activation patterns associated with each input pattern. Perform a cluster analysis on the test patterns (see the hints in exercise 5). Can you infer something from the cluster analysis regarding the network's solution?

7. Open the *stems* file and the *pasts* file in the *tlearn* folder of chapter 11. The *stems* file contains 500 AE (*Artificial English*) verb stems and the *pasts* file contains the corresponding past tense forms. Each "verb stem" consists of three phonemes. The past tense forms contain a fourth element for coding the suffix (W, X, Y, Z; see the lecture on *past tense acquisition* for explaining the details). The list with the 500 verbs contains 2 arbitrary past tense forms, 410 regulars, 20 no change verbs and 68 vowel change verbs, in that order. A ASCII code is used for encoding the phonemes.

a. Use the translation table *phonemes* →*ASCII* from the lecture, repeated here:

/*b*/ b, /*p*/ p, /*d*/ d, /*t*/ t, /*k*/ k, /*v*/ v, /*f*/ f, /*m*/ m, /*n*/ n,
/*h*/ G, /*d*/ T, /*q*/ H, /z/ z, /s/ s, /*w*/ w, /l/ l, /*r*/ r, /*y*/ y, /*h*/ h,
/*i*/ E (eat), /*I*/ I (bit), /*o*/ O (boat), /*U*/ u (book), /*e*/ A (bait),
/*e*/ e (bet), /*ai*/ I (bite), /*æ*/ @ (bat), /*au*/ # (cow), /*O*/ * (or),
Past tense suffixes: *No suffix* W, *-[d]* X, *-[t]* Y, *-[id]* Z

Translate the first 10 and last 10 examples of the *stems* and *pasts* files from the ASCII code into the phonemic code and pair the stems with the corresponding past tense forms. Discuss whether this pairings give a plausible description of potential English past tense forms.

b. Use the *tlearn* editor and create two sublists *stems_short* and *pasts_short* containing exactly the first 10 & last 10 elements of the files *stem* and *pasts*, respectively. Use the **translate** facility under the **edit** menu to convert *stems_short* to a binary representation of the verb stems that obeys the phonological coding scheme stored in the *phonemes* file. Do the same for *pasts_short*.
And this is the phoneme file (in case you cannot find it or the file is overwritten):

```
E 1 1 1 1 1 1
i 1 1 0 0 1 1
O 1 1 1 0 1 1
^ 1 1 0 1 1 1
U 1 1 1 1 0 1
u 1 1 0 0 0 1
A 1 1 1 1 1 0
e 1 1 0 0 1 0
I 1 1 1 0 0 0
@ 1 1 0 1 0 0
# 1 1 1 1 0 0
* 1 1 0 0 0 0
b 0 1 1 1 1 1
p 0 0 1 1 1 1
d 0 1 1 1 1 0
t 0 0 1 1 1 0
g 0 1 1 1 0 0
k 0 0 1 1 0 0
v 0 1 1 0 1 1
f 0 0 1 0 1 1
m 0 1 0 0 1 1
n 0 1 0 0 1 0
G 0 1 0 0 0 0
T 0 0 1 0 1 0
H 0 1 1 0 1 0
s 0 0 1 0 0 1
z 0 1 1 0 0 1
w 0 1 0 1 1 1
l 0 1 0 1 1 0
r 0 1 0 1 0 1
y 0 1 0 1 0 0
h 0 0 0 1 0 0
W 0 0
X 1 0
Y 0 1
Z 1 1
```

c. Use the network architecture proposed in *tlearn*'s chapter 11. Train the network with a learning rate of 0.3 and a momentum of 0.8 for 50 periods (train randomly; try out various initial seeds). Analyze the network performance using the error

display. What are the values for $R_{oot}M_{ean}S_{quare}= \sqrt{\sum_{k=1}^{N}\frac{(\vec{t}_k-\vec{o}_k)^2}{N}}$? Are the observed errors high or small? Give an interpretation!

d. It should be mentioned that the coding scheme for the suffix doesn't make use of phonological codes. Instead, simple 2-bit pattern are used to encode the absence of suffix (0 0) and the three allomorphs of the /ed/ suffix respectively *-[d]* (0 1), *-[t]* (1 0), *-[id]* (1 1). Note that the choice of (0 0) to represent no suffix and the other three patterns to represent the suffixes brings some structure into the suffix pattern. Do you think this structure is important, or can we use arbitrary coding schemes here? What do you think about a phonological coding of the suffixes (as Rumelhart & McClelland did it)?

```
W 0 0
X 1 0
Y 0 1
Z 1 1
```

8. **This is the exercise that has to be submitted**. It contains 2 parts. The first part addresses standard tasks implemented in *tlearn*. The second part addresses an important research topic: what is the appropriate number of hidden units for a given task. Too many hidden units can lead to the effects of over-learning: learning becomes instant-based, and relevant generalizations are lost. In contrast, too few hidden units can make it completely impossible to solve the learning task. [In this connection there are interesting speculations about the possible biological causes of **autism.** Some authors (e.g. Cohen 1994) claim that autism has to do with structural deficits (too few neurons in some areas, such as the cerebellum, and too many neurons in other areas, such as the amygdala and hippocampus).]

   For both parts give a concise description of the results obtained. Please, include a gallery of relevant graphics displaying the main results and illustrating your main conclusions.

   Part I

   Load the project *phone* of chapter 11. It contains the *phone.data* and *phone.teach* files that are the featural translations of the *stems* file and the *pasts* file, respectively, discussed in exercise 7. The network contains 20 input units, 20 output units and 30 hidden nodes.

   a. Train the network with a learning rate of 0.3 and a momentum of 0.8 for 50 periods (train randomly; try out various initial seeds). Try a variety of other network parameters if the error level remains high. Analyze the network performance using the error display. What are the values for $R_{oot}M_{ean}S_{quare}= \sqrt{\sum_{k=1}^{N}\frac{(\vec{t}_k-\vec{o}_k)^2}{N}}$? Are the observed errors high or small? Give an interpretation!

b. Analyze network performance. One way to get a quick idea of how the network is performing on individual patterns is to calculate the error for individual patterns in the training set. To do this you must check the **calculate error** box in **testing options**… dialogue box. When you attempt to verify that the network has learned, *tlearn* will display the RMS error for each individual pattern, in the order that it appears in the *phone.data* file. These errors are automatically displayed in the **error display**. Since you know the order of the verbs (see exercise 7) it is relatively easy to determine which verbs (verb classes) are performing poorly and which verbs (verb classes) are performing well.

c. Classify the types of errors you find with the 20 *no change verbs* (nr. 413-422). Use the output translation technique to investigate the performance of the network on these particular items.

Output translation: In the **special** menu open **output translation**. Click on the pattern file box and select the *phoneme.out* reference file. Check the **Euclidean distance** box and then click OK. Finally, check the **use output translation** box in the **testing options**. Now, when you attempt to **verify the network has learned**, the **output** display will include an ASCII representation of the output units together with the target output. The output display will continue to include the ASCII output until you uncheck the **use output translation** box.
Due to an error in the pasts`-file you should interpret an X in the output as -t (instead of -d) and a Y as -d (instead of -t)!`

d. In the *tlearn* folder you will find a file called *phone.test* that contains a list of verb stems that the network has not been trained on. Use this list (or part of it) to evaluate how the network generalizes to novel forms. (Hint: use the **output translation** utility)

Part II

Does network performance and generalization alter with the number of hidden units in the network? Use two modifications of the network considered before:

(i) 3 hidden units only
(ii) 100 hidden units.

Compare the performance of these two networks with the network studied before. Especially interesting are the network's abilities to deal with irregular forms - see (b) and (c) - and with generalizations to novel forms - see (d).