Tutorial 6: CSNLP

Simple Recurrent Networks

Date: 12 December 2001
Student's name:

In this tutorial, the aim is to familiarise yourself with the processing of sequences over time, using simple recurrent networks. This tutorial is based on Chapter 8 of Plunkett & Elman, and takes you through the learning of letter sequences: **ba dii guuu** as discussed in the lecture.

To begin, copy the Chapter 8 directory of Plunkett & Elman to your desktop. The encoding scheme used in the this simulation is slightly different from that of Elman's original simulations (this is given in the `codes` file):

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

As with the original encoding, the first bit represents the consonant feature, the other 3 a localist representations of the each consonant and vowel. The `letters` file contains a random sequence of 2993 letters which can be converted to the vector representation using the `Translate` option (while letters is open and active) and selecting the pattern file `codes`. This resulting training file can be saved as srn.data.

Since we are training the network to predict the next letter, create the `srn.teach` file by copying the `srn.data` file, then edit `srn.teach`, moving the first line to the end of the file.

Ex 1: Examine the network architecture. Based on the network configuration `srn.cf`, draw the network as a conventional SRN, indicating which nodes numbers are the inputs, output, hidden, and context units.

Ex 2: Why do the think the recurrent connections from the hidden to the context units are one to one with fixed weights?

Ex 3: Train the network (sequentially!) with learning rate = 0.1 and momentum = 0.3 for 70000 sweeps. How many epochs it this? Monitor the RMS error during training, and explain why it seems to stay so high.

Ex 4: Examine the file `predtest.data`. What letter sequence is being tested by this file? Test the network using this file (make sure the appropriate `predtest.teach` also exists). Also make sure the simulator is set to Calculate error, under Testing options, so you can easily examine the error behaviour for each letter in the sequence. Sketch the error plot, annotated with letters at each point.

Ex 5: How well has the network learned to predict the next element in the sequence? Does is correctly predict the vowel following a consonant? Does is correctly predict the number of vowels?

Ex 6: Does the network correctly predict when a consonant will be the next item in the sequence? Explain why you think it has or hasn't.

Ex 7: Examine the networks solution by examining the hidden node activations associated with each input pattern, by performing a cluster analysis of the hidden units on the test patterns. Sketch the resulting dendrogram, and comment on what it suggests about the networks solution.

Cluster analysis: clear the output display, then in **Network, Probe selected nodes**. Output now contains the hidden unit activations for the patterns in `pretest.data`. Save the Output as `predtest.hid`. Create a Names file called `predtest.lab` which has **b a d i1 i2 g u1 u2 u3**, one per line. Then choose **Special, Cluster Analysis**.