# COSC421 2015 Assignment 2 (15%)
# A simple recurrent network for encoding word forms

There are two options for this assignment. The first involves programming; the second is an essay. Both options are based around Elman's **simple recurrent network** (**SRN**): a network which can learn sequences. In the assignment, you will be dealing with SRNs which learn word forms, represented as sequences of phonemes.

## Option 1: A neural network for learning word forms

In this option, your task is to implement an SRN which receives as input a sequence of words, each of which is represented as a sequence of **phonemes**, and learns to recognise words as frequently occurring patterns of phonemes. Phonemes are sound units (see Lecture 6): a convenient ASCII set to use for programming are the ARPABET phonemes used for US English listed in Appendix 1. In this scheme, the word *yield* is represented as the phoneme sequence *Y,IY,L,D*.[1]

Your SRN will simulate the phonological learning which happens in infants when they hear words in their native language. You must create a vocabulary of English words encoded as phoneme sequences, and present words from this vocabulary randomly to the network, as sequences of phonemes. At each point, the network should be trained to predict the next phoneme in the sequence.

After training, the network should learn three things. Firstly, it should learn general rules about how phonemes can succeed one another (e.g. that the phoneme after *B* can be *AE* or *AY*, but not *P*). Secondly, it should learn the forms of the training words it was shown, so that when shown the start of a word, it can predict the rest of it (provided there are no other words with the same starting sequence). Finally, it should learn to recognise the boundaries between words: assuming that words are presented at random during training, word boundaries will be places where the network is particularly unsure about the phoneme which comes next.

The architecture of your network should look as shown in Figure 1. The network comprises:

- An **input layer**, with one unit for each phoneme;
- An **output layer**, with one unit for each phoneme;
- A **hidden layer** of units in between the input and output layers;
- A **context layer** of units providing additional input to the hidden layer, representing 'the current context'.

All units have real-valued activations in the range 0–1. Hidden layer units use a **sigmoid** activation function, and output units use a **linear** activation function. The input and

[1] You can modify this set to represent NZ pronuncuation if you wish.

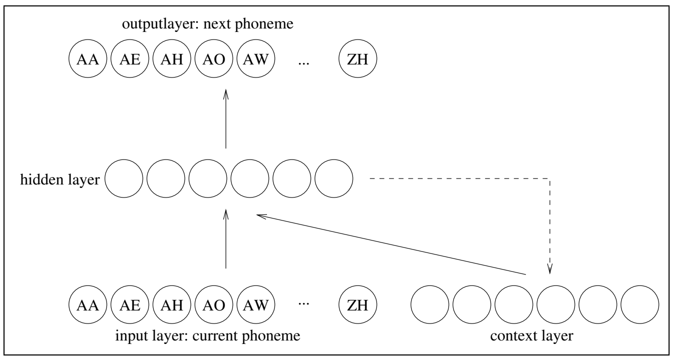

Figure 1: Architecture of the network

output layers use a **localist** encoding scheme: to represent a given phoneme, we set the activation of one unit to 1, and of all other units to 0. The input and context layers are fully connected to the hidden layer, and the hidden layer is fully connected to the output layer. The output layer is passed through a **softmax** function, which constrains the activations of the output units to sum to 1, so they can be interpreted as a probability distribution.

Training happens in a sequence of time steps. At each time step, the network receives a phoneme on its input layer, and makes a prediction about the *next phoneme* on its output layer (through the weights which connect the input to the output layer). The network is trained using the *actual* next phoneme as a training signal. (The training algorithm is **back-propagation**: all you need to know about this algorithm is that it alters the weights in the network to minimise the difference betweeen the predicted next phoneme and the actual next phoneme.)

An SRN can learn to predict more than one time step ahead, because of its context layer. The context layer is a copy of the hidden layer at the previous time step: so it holds information about the previous phoneme—but also, recursively, about the *previous context*. During training, it learns a representation of phonemes which captures the common sequential patterns it was presented during training.

If you did the neural networks course last semester, you can use your own back-propagation code to build the network. Alternatively, you can use some Java code written in-house (by a recent postdoc, Martin Takac). The code is available on the 421 webpage, under 'Resources'. In either case, your task is to take a standard multilayer perceptron trained using backprop, and turn it into a SRN. You will have to create a training regime involving a series of time steps, and add special functionality to the network setting the context layer of the current time step to the same values as the hidden layer at the previous time step. Aside from this, the way training works is the same as in regular backprop: at each time step there's a training input (a vector on the input and context layers), which is propagated forward to the hidden layer and then to the output layer, and then there's an error term created by comparing the actual activations of the output units with the desired output for this input (i.e. the next phoneme).

You will train your word-learning SRN in a series of **epochs**. In each epoch, you will present your SRN with each word in your vocabulary (one phoneme at a time). Words must be presented in random orders, so the network can learn where word boundaries are. In each epoch, you can keep a record of the average size of the error your SRN is making in its prediction of the next phoneme: during training, this error should decrease. (It won't get anywhere near zero, because there are normally several possible next phonemes.) When the error reaches an asymptote, you can stop training.

Here are some notes about the NNpackage code.

- There's a small manual (NNManual.doc).
- MinimalNetwork.java holds the main functions for creating, training and testing a network.
- To create a network, you need to create an object of the `MinimalNetwork` class. You must also create a file holding various parameter values for your network. These parameters are explained in the manual. To create an SRN, there are a couple of ready-made parameters you must use:
  - `NNtype,SRN` (automatically creates the context layer)
  - `useSoftmax,true` (ensures that activations in the output layer sum to 1)
  - `formDim,[val]` (where [val] is the size of the input layer).
- Two ways to train the network are shown in MinimalNetwork.java, illustrated by `exampleOfUse1` and `exampleOfUse2`.
  - `exampleOfUse1` is good for getting started you can just specify the sequence of training inputs and outputs in two files. For instance, you could teach your SRN one particular short sequence of phonemes.
  - `exampleOfUse2` is more configurable: here, the training data are specified directly in the code. This is better for learning a set of words (since the words themselves have to be in random orders).
  - To test the network, set a vector on its input units using `myNet.setInput`, then run propagate to create an output value. To read the output, `myNet.Output` returns an array of the output values.

## Code

You are free to implement the network in any language.

## Behaviour of the trained network

Your network should show three behaviours.

- When given the first few phonemes of a word, it should make sensible predictions about the next phoneme.

- When given an initial sequence which identifies a single word uniquely, it should predict the remaining sequence of phonemes of that word correctly.
- When it reaches the end of a word, there should be a sudden drop in its ability to predict the next phoneme. (This drop should allow it to recognise word boundaries.)

## Report

You should also submit a report about your code. The report should contain four sections:

- How the code implements the network model;
- How to run the code (so I can test it);
- Some examples of its results (including evidence of the three behaviours it must show);
- A brief evaluation of how well it does, and what might be done to improve it.

## Submission and marking

You should submit the code and the assignment by email to me (`alik@cs.otago.ac.nz`) by 5pm on **Monday of Week 9** (i.e. on **Monday September 12th**). 10% of available marks will be deducted for each day late.

I'll give equal weight to the code and the report. (But the 'report mark' is likely to reflect the quality of the code, so in that sense the implementation is the main focus.)

## Appendix 1: a set of phonemes

Here's a set of phonemes in the 'ARPABET' phonetic alphabet for US English. Each phoneme comes with an example of a word it occurs in (along with the full ARPABET encoding of this word).

| Phoneme | Example word | Encoding of example word |
|---|---|---|
| AA | odd | AA, D |
| AE | at | AE, T |
| AH | hut | HH, AH, T |
| AO | ought | AO, T |
| AW | cow | K, AW |
| AY | hide | HH, AY, D |
| B | be | B, IY |
| CH | cheese | CH, IY, Z |
| D | dee | D, IY |
| DH | thee | DH, IY |
| EH | Ed | EH, D |
| ER | hurt | HH, ER, T |
| EY | ate | EY, T |
| F | fee | F, IY |
| G | green | G, R, IY, N |
| HH | he | HH, IY |
| IH | it | IH, T |
| IY | eat | IY, T |
| JH | gee | JH, IY |
| K | key | K, IY |
| L | lee | L, IY |
| M | me | M, IY |
| N | knee | N, IY |
| NG | ping | P, IH, NG |
| OW | oat | OW, T |
| OY | toy | T, OY |
| P | pee | P, IY |
| R | read | R, IY, D |
| S | sea | S, IY |
| SH | she | SH, IY |
| T | tea | T, IY |
| TH | theta | TH, EY, T, AH |
| UH | hood | HH, UH, D |
| UW | two | T, UW |
| V | vee | V, IY |
| W | we | W, IY |
| Y | yield | Y, IY, L, D |
| Z | zee | Z, IY |
| ZH | seizure | S, IY, ZH, ER |

## Option 2: Essay

The second option for the assignment is an essay, with the following title:

> Describe in detail Elman's (1990) simple recurrent network (SRN) model, and explain how it can be trained to learn sequences. Then describe Gaskell and Marslen-Wilson's (1997) adaptation of a SRN to model the perception of words. Your description of should include (i) an account of the input representations used in Gaskell and Marslen-Wilson's network; (ii) the architecture of the network; (iii) how it is trained; and (iv) what behaviour it produces, and how this compares with human performance in the recognition of words.

To answer this question, your main reference should obviously be the two papers referred to. But you can also briefly consult other papers—for instance, papers which are cited by these papers, or which cite them.[2]

### Submission and marking

You should submit the essay by email to me (`alik@cs.otago.ac.nz`) by 5pm on **Monday of Week 9** (i.e. on **Monday 12th September**). 10% of available marks will be deducted for each day late.

You will be marked on the clarity and organisation of the essay, and on evidence that you have read the assigned paper in detail and understood the model described.

## References

Elman, J. (1990). Finding structure in time. *Cognitive Science*, **14**, 179–211.

Gaskell, M. and Marslen-Wilson, W. (1997). Integrating form and meaning: a distributed model of speech perception. *Language and Cognitive Processes*, **12**, 613–656.

[2]You can get these papers online from Web of Science. Go to the Otago Library webpage, then follow links to 'Article Databases' → 'W' → 'Web of Science via Web of Knowledge'. This is a fantastic resource—if you don't already know about it, you should try it!