6a. Recurrent Networks

Textbook, Chapter 10
Outline

- Processing Temporal Sequences
- Sliding Window
- Recurrent Network Architectures
- Hidden Unit Dynamics
- Long Short Term Memory
Processing Temporal Sequences

There are many tasks which require a sequence of inputs to be processed rather than a single input.

- speech recognition
- time series prediction
- machine translation
- handwriting recognition

How can neural network models be adapted for these tasks?
The simplest way to feed temporal input to a neural network is the “sliding window” approach, first used in the NetTalk system (Sejnowski & Rosenberg, 1987).
NetTalk Task

Given a sequence of 7 characters, predict the phonetic pronunciation of the middle character.

For this task, we need to know the characters on both sides.

For example, how are the vowels in these words pronounced?

pa pat pate paternal

mo mod mode modern
NetTalk Architecture

26 output units

/k/

80 hidden units

203 input units (7 groups of 29)
NetTalk

- NETtalk gained a lot of media attention at the time.

- Hooking it up to a speech synthesizer was very cute. In the early stages of training, it sounded like a babbling baby. When fully trained, it pronounced the words mostly correctly (but sounded somewhat robotic).

- Later studies on similar tasks have often found that a decision tree could produce equally good or better accuracy.

- This kind of approach can only learn short term dependencies, not the medium or long term dependencies that are required for some tasks.
Simple Recurrent Network (Elman, 1990)

- at each time step, hidden layer activations are copied to “context” layer
- hidden layer receives connections from input and context layers
- the inputs are fed one at a time to the network, it uses the context layer to “remember” whatever information is required for it to produce the correct output
we can “unroll” a recurrent architecture into an equivalent feedforward architecture, with shared weights

applying backpropagation to the unrolled architecture is referred to as “backpropagation through time”

we can backpropagate just one timestep, or a fixed number of timesteps, or all the way back to beginning of the sequence
Other Recurrent Network Architectures

- it is sometimes beneficial to add “shortcut” connections directly from input to output
- connections from output back to hidden have also been explored (sometimes called “Jordan Networks”)

© Alan Blair, 2017-19
Second Order (or Gated) Networks

\[
x_t^j = \tanh(W_{j0}^{j0} + \sum_{k=1}^{d} W_{jk}^{j0} x_{t-1}^k)
\]

\[
z = \tanh(P_0 + \sum_{j=1}^{d} P_j x_t^j)
\]
Task: Formal Language Recognition

<table>
<thead>
<tr>
<th>Accept</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 1</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 0</td>
</tr>
<tr>
<td>1 1 1 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 1 1 1 1 1</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1</td>
<td>1 1 1 1 1 0</td>
</tr>
<tr>
<td>1 1 1 1 1 1 1 1</td>
<td>1 0 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Scan a sequence of characters one at a time, then classify the sequence as Accept or Reject.
Dynamical Recognizers

- gated network trained by BPTT
- emulates exactly the behaviour of Finite State Automaton

\[ W_0 = \begin{bmatrix} -0.89 & -0.09 & -0.14 \\ -1.13 & -0.09 & -0.14 \end{bmatrix} \]

\[ W_1 = \begin{bmatrix} 0.20 & 0.68 & 0.96 \\ 0.20 & 0.81 & 1.19 \end{bmatrix} \]

\[ P = \begin{bmatrix} -0.07 & 0.66 & 0.75 \end{bmatrix} \]
### Task: Formal Language Recognition

<table>
<thead>
<tr>
<th>Accept</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0</td>
<td>1 1 0 0 0</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0 0 1</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0</td>
<td>1 1 1 1 0 0 0 0 1 1</td>
</tr>
<tr>
<td>1 0 0 1 0 0</td>
<td>1 1 0 1 0 1 0 0 0 0 0 0 1 0 1 1 1</td>
</tr>
<tr>
<td>0 0 1 1 1 1 1 1 0 1 0 0</td>
<td>1 0 1 0 0 1 0 0 0 1</td>
</tr>
<tr>
<td>0 1 0 0 1 0 0 1 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>1 1 1 0 0</td>
<td>0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

Scan a sequence of characters one at a time, then classify the sequence as Accept or Reject.
Dynamical Recognizers

- trained network emulates the behaviour of Finite State Automaton
- training set must include short, medium and long examples
Phase Transition

\[ W_0 = \begin{bmatrix} -0.567 & 1.761 & 0.815 \\ -0.219 & -2.591 & 0.446 \end{bmatrix} \]
\[ W_1 = \begin{bmatrix} 0.752 & 0.548 & -1.071 \\ 0.074 & -0.813 & 1.502 \end{bmatrix} \]
\[ P = \begin{bmatrix} 0.069 & 0.172 & -0.985 \end{bmatrix} \]
# Chomsky Hierarchy

<table>
<thead>
<tr>
<th>Language</th>
<th>Machine</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular</td>
<td>Finite State Automaton</td>
<td>$a^n$ ($n$ odd)</td>
</tr>
<tr>
<td>Context Free</td>
<td>Push Down Automaton</td>
<td>$a^n b^n$</td>
</tr>
<tr>
<td>Context Sensitive</td>
<td>Linear Bounded Automaton</td>
<td>$a^n b^n c^n$</td>
</tr>
<tr>
<td>Recursively Enumerable</td>
<td>Turing Machine</td>
<td>true QBF</td>
</tr>
</tbody>
</table>
Task: Formal Language Prediction

abaabhabaabaabbbbbabaabaabbbabbaaaaaabbbbb…

Scan a sequence of characters one at a time, and try at each step to predict the next character in the sequence.

In some cases, the prediction is probabilistic.

For the $a^n b^n$ task, the first $b$ is not predictable, but subsequent $b$’s and the initial $a$ in the next subsequence are predictable.
Elman Network for predicting $a^n b^n$
Oscillating Solution for $a^n b^n$
Learning to Predict $a^n b^n$

- the network does not implement a Finite State Automaton but instead uses two fixed points in activation space – one attracting, the other repelling (Wiles & Elman, 1995)
- networks trained only up to $a^{10} b^{10}$ could generalize up to $a^{12} b^{12}$
- training the weights by evolution is more stable than by backpropagation
- networks trained by evolution were sometimes monotonic rather than oscillating
Monotonic Solution for $a^n b^n$
Hidden Unit Analysis for $a^n b^n$

hidden unit trajectory

fixed points and eigenvectors
Counting by Spiralling

- for this task, sequence is accepted if the number of $a$'s and $b$'s are equal
- network counts up by spiralling inwards, down by spiralling outwards
SRN with 3 hidden units can learn to predict $a^n b^n c^n$ by counting up and down simultaneously in different directions, thus producing a star shape.
Partly Monotonic Solution for $a^n b^n c^n$
Long Range Dependencies

- Simple Recurrent Networks (SRNs) can learn medium-range dependencies but have difficulty learning long range dependencies.
- Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) can learn long range dependencies better than SRN.
Long Short Term Memory

Two excellent Web resources for LSTM:

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/LSTM.php
Reber Grammar
Embedded Reber Grammar
Simple Recurrent Network

SRN – context layer is combined directly with the input to produce the next hidden layer.

SRN can learn Reber Grammar, but not Embedded Reber Grammar.
Long Short Term Memory

LSTM – context layer is modulated by three gating mechanisms: forget gate, input gate and output gate.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long Short Term Memory

Gates:
\[ f_t = \sigma (W_f x_t + U_f h_{t-1} + b_f) \]
\[ i_t = \sigma (W_i x_t + U_i h_{t-1} + b_i) \]
\[ g_t = \tanh (W_g x_t + U_g h_{t-1} + b_g) \]
\[ o_t = \sigma (W_o x_t + U_o h_{t-1} + b_o) \]

State:
\[ c_t = c_{t-1} \odot f_t + i_t \odot g_t \]

Output:
\[ h_t = \tanh c_t \odot o_t \]
Gated Recurrent Unit

GRU is similar to LSTM but has only two gates instead of three.