

## 31YB: LECTURE 12

Thurs, 13 Nov

### Recurrent Neural-Networks and Learning from Temporal Sequences

Lecture PART I: Learning from Temporal Sequences

- Learning Time Sequences: Problems
- Solution: Treating Time as Space
- Example Application: NETalk
- Limitations

Lecture PART II: Recurrent Neural Networks

- Hopfield Nets as Content Addressable Memory,
- Limitations of Hopfield Nets, Boltzmann Machine
- Partially Recurrent Nets: Elman Nets, Jordan Nets

1

---

---

---

---

---

---

---

---

---

---

### Lecture Part I: Learning Time Sequences

Two types of applications:

- Sequence Recognition
  - produce a particular output in response to an input sequence
  - e.g. speech recognition (output the word spoken)
- Sequence Reproduction (Prediction)
  - from part of a sequence, predict the rest
  - time-series prediction e.g. financial markets

---

---

---

---

---

---

---

---

---

---

### Treating Time as Space

- Turn temporal sequence into spatial pattern at the input
- Use a feed-forward net (e.g. MLP + backprop, or RBF) to learn and to recognise

---

---

---

---

---

---

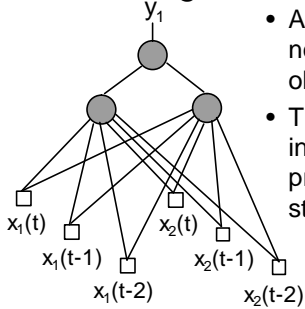
---

---

---

---

## Treating Time as Space



- At each time-step a new input  $\mathbf{x}$  is observed
- This net also receives input from the previous two time-steps

---

---

---

---

---

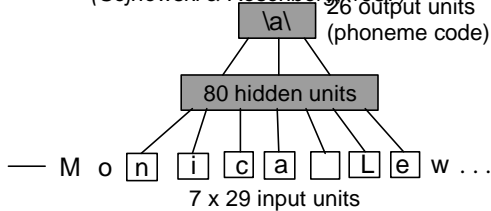
---

---

---

## Example Application: NETtalk

(Sejnowski & Rosenberg, 1987)



- Trained on 1024 English words with phonemes
- Produced intelligible speech from novel text

---

---

---

---

---

---

---

---

## Treating Time as Space: Some Limitations

- Length of temporal window *fixed* in advance
- Large input layer (slow computation)
- Synchronising the input signal can be tricky in certain implementations

---

---

---

---

---

---

---

---

*Lecture PART II:  
Feed-forward or recurrent ?*

- So far, nets are entirely feed-forward
  - NB: don't confuse back-prop with feedback !
- Biological neural nets have a **lot** of feedback connections
- Feedforward + feedback = recurrent

---

---

---

---

---

---

---

---

*Recurrent Nets Background:  
John Hopfield*

- Prof. biology/chemistry, CalTech /AT&T Bell Labs
- Re-energised ANN research in 1982
- Studied systems of interconnected 'neurons'
  - stable states which are entered if started in a nearby state
  - network can learn to set up stable states
- Auto-Association, Content-Addressable Memory

*(ref. Igor Aleksander & Helen Morton, 'Neural Computing')*

---

---

---

---

---

---

---

---

*Content Addressable Memory*

- Suppose we store in memory the item:  
'H.A. Kramers and G.H. Wannier *Phys. Rev.*, 60, 252 (1941)'
- A content addressable memory (CAM) could retrieve it from a partial query such as:  
'and Wannier (1941)'
- and a CAM tolerant of distortions from:  
'Vannier, (1941)'

---

---

---

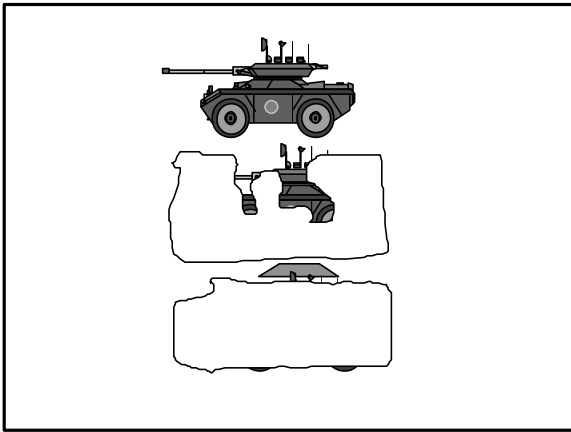
---

---

---

---

---




---

---

---

---

---

---

---

---

### Hopfield Net

- Fully-connected: all neurons interconnected
- **Symmetric**, bi-directional weights ( $w_{ij} = w_{ji}$ )
- Firing rule same as the McCulloch-Pitts model i.e. weighted sum of inputs and threshold at  $T_j$

$$y_j = 1 \text{ if } \sum_{i \neq j} w_{ij} y_i > T_j$$

$$y_j = 0 \text{ if } \sum_{i \neq j} w_{ij} y_i \leq T_j$$

- If  $y_j=1$  we say that neuron  $j$  is 'firing'
- *Asynchronous* operation: only one neuron attempts to fire at a time. All equally likely.

---

---

---

---

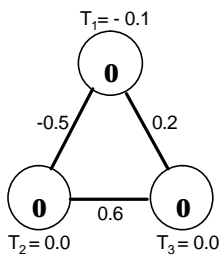
---

---

---

---

### A Simple Hopfield Net (NOT Examinable)



Given the initial state:  
 $y_1, y_2, y_3 = 0, 0, 0$

- What happens if one of the neurons tries to fire ?
- It can be shown that whatever state it starts in, this net always settles in the state 0, 1, 1 (stable state)

---

---

---

---

---

---

---

---

## *Hopfield's Analysis*

- Think of each state as having an *energy level*,  $E$
- Then think of net's behaviour as reducing energy
- Stable states are now '*energy wells*'
  
- Should define  $E$  so that a state change never increases  $E$ , *i.e.*  $\Delta E \leq 0$

---

---

---

---

---

---

---

---

## *Hopfield's Analysis*

- By defining 'energy' for states, Hopfield's analysis proved that state changes in an irreversible manner
  
- A stable state is reached when there are no accessible states with lower energy
  
- Unfortunately, stability can be local

---

---

---

---

---

---

---

---

## *Hopfield Nets as CAMs*

- Storing: (Training)
  - Need a method for computing weights and thresholds so that the patterns to be stored become stable network states
  - This is training the network
- Recall: (Testing)
  - Given a pattern as a starting state, the net will tend to converge to the most similar stored pattern
- This will give a content-addressable memory with tolerance to distortions.

---

---

---

---

---

---

---

---

## Limitations of Hopfield Nets

- Local minima
  - The net may not always reach the lowest state.  
i.e. the net can find intermediate stable 'energy wells'.
- Storage Capacity
  - The number of patterns  $p$  that can be stored without unacceptable errors. If patterns are random then  $p$  is proportional to  $N$  but less than  $0.15N$ .
    - e.g. 10 patterns can required 70 neurons.
  - If patterns are not random but close to 'orthogonal' then  $p$  can be higher.
- Instability
  - Example patterns will produce instability if they share many bits with other example patterns.

---

---

---

---

---

---

---

---

## Limitations of Hopfield Nets

- It can be shown that a 3-neuron Hopfield net cannot learn to produce the following four stable states with equal probability 1/4:  
101, 011, 000, 110
- This is XOR in disguise
- Need 'hidden' units

---

---

---

---

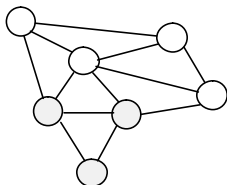
---

---

---

---

## Boltzmann Machines



- Visible units (input, output)
- Hidden units

Hopfield → Boltzmann like SLP → MLP

---

---

---

---

---

---

---

---

## Boltzmann machines

- Developed independently by several groups
  - Ackley, Hinton and Sejnowski (1986)
  - Geman and Geman (1984)
  - Smolensky (1986)
- Units fire probabilistically based on a sigmoid activation function
- Learning adjusts weights to give states of *visible* units a particular desired probability distribution

---

---

---

---

---

---

---

---

## II: Partially Recurrent Networks

- Approach: add carefully chosen feedback connections to a feed-forward network
- Units receiving feedback are '*context units*'
- These units 'remember' aspects of the past
- Feedback connections have fixed weights, so back-prop can still be used for training

---

---

---

---

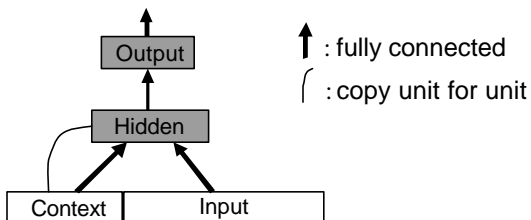
---

---

---

---

## Elman Networks



- ✦ *Context units* hold a copy of the hidden unit activations from the previous time-step
- ✦ Context treated just like inputs for backprop
- ✦ *Elman nets* used for sequence recognition and prediction

---

---

---

---

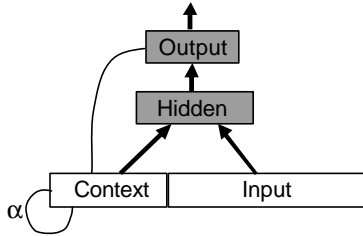
---

---

---

---

## Jordan Networks



- ◆ Context units receive copy of previous *outputs*
- ◆ *Self-connections* give context units individual memory...it can be shown that the smaller  $\alpha$ , the shorter its 'memory'.
- ◆ *Jordan nets used for sequence recognition & generation*

---

---

---

---

---

---

---

---

## Summary

- NN Application: Learning Time Sequences: Treating time as space
- Recurrent Nets:
- Fully Recurrent: Hopfield Nets, Limitations, Boltzmann machines
- Partially Recurrent Nets: Elman & Jordan nets

### *Remaining Topics/Lectures*

- Genetic Algorithms & Neuromorphic Systems
- **Note:** *Today: Extended Lab (3) Session: 1:30-3PM* to help with your (last minute!) Assignment queries

---

---

---

---

---

---

---

---