Recurrent Neural Networks

Brian Hrolenok

George Mason University

CS688 Pattern Recognition - Fall 2009

Outline

Motivation

Why do we need another NN model?

- Sequence prediction
- Temporal input
- Biological Realism

Temporal XOR

"1 0 1 0 0 0 0 1 1 1 1 "

". . 1 . . 0 . . 1 . . ?"

Motivation

Why do we need another NN model?

- Sequence prediction
- Temporal input
- Biological Realism

Temporal XOR

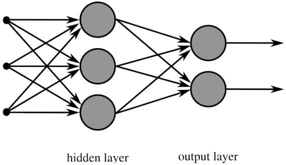
"1 0 1 0 0 0 0 1 1 1 1 "

". . 1 . . 0 . . 1 . . ?"

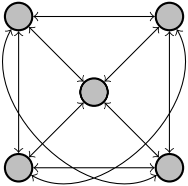
From MFNN to RNN

ANNs represent computation as flowing through a graph.

- Multi-layer Feed-forward Neural Network - *DAG*



- Recurrent Neural Network - *Digraph*



- Running? Training? Input? Output?

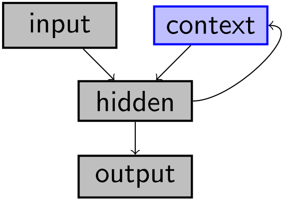
Models

Some RNN models that will be discussed today:

- Elman Networks
- Jordan Networks
- Hopfield Networks
- Liquid State Machines
- Echo State Networks
- Topology & Weight Evolving Neural Networks

Elman Networks

Elman networks are MFNNs with an extra context layer



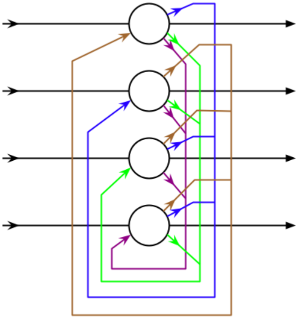
- Synchronous
- Fix recurrent weights
- Training: use backpropegation

Running

1. Input units get their values
2. Hidden units compute their value (weighted sum of inputs and context units)
3. Context units get their value
4. Output units compute their value

Hopfield Networks

Network is defined by its weight matrix, W_{ij}



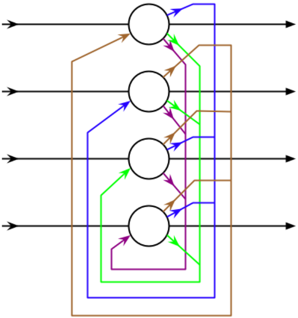
- Fully connected graph
- Asynchronous
- Fixed-points of dynamical system

Running

1. Pick random node p of N
2. Compute sum of incomming links: $x_p = \sum_k W_{kp} V_k + I_p$
3. Compute activation level: $V_p = f(x_p)$
4. Repeat

Hopfield Networks (2)

Hopfield networks will converge to a fixed point if the weight matrix is under certain restrictions.



The Lyapunov function

$E = -\frac{1}{2}\sum_{jk} W_{jk} V_j V_k - \sum_m I_m V_m$

- Weight matrix is symmetric
- No self loops

Training

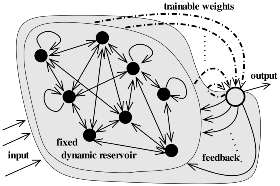
- Associative memory: Hebbian Learning
  $W_{ij} \leftarrow W_{ij} + x_i^k x_j^k$
- Optimization problems: formulate E and solve for W

Backpropagation Through Time

"Unroll" the network in time, then apply backpropagation as normal.

- Only works for synchronous networks
- Activation function should have easily computed higher order derivatives

Backpropagation Decorrelation



- Only output weights are trainable
- Weight update rule
  $\Delta w_{ij}(k+1) = \eta \frac{f(x_j(k))}{\sum_s f(x_s(k))^2+\varepsilon} \gamma_i(k+1)$
  $\gamma_i(k+1) =$
  $\sum_{s\in O} (w_{is} f'(x_s(k))) e_s(k) - e_i(k+1)$

w_{ij}: weight matrix, η: learning rate, f: activation function, ε: regularization constant, O: set of output neurons, e_s: error for s

Evolutionary Computation

Representation

- NEAT - NeuroEvolution of Augmenting Topologies
- Complexification/Simplification
- Competing conventions
- Speciation
- Random initial populations

What I'm Working On

Comparison of RNN training strategies on several test problems.

- Sequence prediction (*temporal XOR, grammars*)
- Double Pole Balancing without Velocity (demo: `http://www.youtube.com/watch?v=fqk2Ve0C8Qs`)
- Utterance Recognition (if I can get the data)

RNN models and training strategies

- SRNs (backprop, fixed-topology EC)
- General RNNs (BPDC, fixed-topology EC, NEAT)