Neural Turing Machine

Author: Alex Graves, Greg Wayne, Ivo Danihelka
Presented By: Tinghui Wang (Steve)
Introduction

• Neural Turning Machine: Couple a Neural Network with external memory resources
• The combined system is analogous to TM, but differentiable from end to end
• NTM Can infer simple algorithms!
In Computation Theory...

Finite State Machine

$$(\Sigma, S, s_0, \delta, F)$$

- $\Sigma$: Alphabet
- $S$: Finite set of states
- $s_0$: Initial state
- $\delta$: state transition function: $S \times \Sigma \rightarrow P(S)$
- $F$: set of Final States
Push Down Automata
\((Q, \Sigma, \Gamma, \delta, q_0, Z, F)\)

- \(\Sigma\): Input Alphabet
- \(\Gamma\): Stack Alphabet
- \(Q\): Finite set of states
- \(q_0\): Initial state
- \(\delta\): State transition function:
  \(Q \times \Sigma \times \Gamma \to P(Q) \times \Gamma\)
- \(F\): Set of Final States
Turing Machine

\((Q, \Gamma, b, \Sigma, \delta, q_0, F)\)

- **\(Q\)**: Finite set of States
- **\(q_0\)**: Initial State
- **\(\Gamma\)**: Tape alphabet (Finite)
- **\(b\)**: Blank Symbol (occurs infinitely on the tape)
- **\(\Sigma\)**: Input Alphabet \((\Sigma = \Gamma \setminus \{b\})\)
- **\(\delta\)**: Transition Function \n  \(\left(\left(Q \setminus F\right) \times \Gamma\right) \to Q \times \Gamma \times \{L, R\}\)
- **\(F\)**: Final State (Accepted)
Neural Turing Machine - Add Learning to TM!!

Finite State Machine (Program)

Neural Network (I Can Learn!!)
A Little History on Neural Network

• 1950s: Frank Rosenblatt, Perceptron – classification based on linear predictor
• 1969: Minsky – Proof that Perceptron Sucks! – Not able to learn XOR
• 1980s: Back propagation
• 1990s: Recurrent Neural Network, Long Short-Term Memory
• Late 2000s – now: Deep Learning, Fast Computer
Feed-Forward Neural Net and Back Propagation

- Total Input of unit $j$:
  \[ x_j = \sum_i y_i w_{ji} \]

- Output of unit $j$ (logistic function)
  \[ y_j = \frac{1}{1 + e^{-x_j}} \]

- Total Error (mean square)
  \[ E_{total} = \frac{1}{2} \sum_c \sum_j (d_{c,j} - y_{c,j})^2 \]

- Gradient Descent
  \[ \frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial x_j} \frac{\partial x_j}{\partial w_{ji}} \]

Recurrent Neural Network

- $y(t)$: n-tuple of outputs at time $t$
- $x^{net}(t)$: m-tuple of external inputs at time $t$
- $U$: set of indices $k$ that $x_k$ is output of unit in network
- $I$: set of indices $k$ that $x_k$ is external input
- $x_k = \begin{cases} x^{net}_k(t), & \text{if } k \in I \\ y_k(t), & \text{if } k \in R \end{cases}$
- $y_k(t + 1) = f_k(s_k(t + 1))$ where $f_k$ is a differential squashing function
- $s_k(t + 1) = \sum_{i \in I \cup U} w_{ki} x_i(t)$

Recurrent Neural Network – Time Unrolling

• Error Function:

\[ J(t) = -\frac{1}{2} \sum_{k \in U} |e_k(t)|^2 = -\frac{1}{2} \sum_{k \in U} |d_k(t) - y_k(t)|^2 \]

\[ J_{Total}(t', t) = \sum_{t'=t+1}^{t} J(t) \]

• Back Propagation through Time:

\[ \varepsilon_k(t) = \frac{\partial J(t)}{\partial y_k(t)} = e_k(t) = d_k(t) - y_k(t) \]

\[ \delta_k(\tau) = \frac{\partial J(t)}{\partial s_k(\tau)} = \frac{\partial J(t)}{\partial y_k(\tau)} \frac{\partial y_k(\tau)}{\partial s_k(\tau)} = f'_k(s_k(\tau)) \varepsilon_k(\tau) \]

\[ \varepsilon_k(\tau - 1) = \frac{\partial J(t)}{\partial y_k(\tau - 1)} = \sum_{l \in U} \frac{\partial J(t)}{\partial s_l(\tau)} \frac{\partial s_l(\tau)}{\partial y_k(\tau - 1)} = \sum_{l \in U} w_{lk} \delta_l(\tau) \]

\[ \frac{\partial J_{Total}}{\partial w_{ji}} = \sum_{t=t'+1}^{t} \frac{\partial J(t)}{\partial s_j(\tau)} \frac{\partial s_j(\tau)}{\partial w_{ji}} \]
RNN & Long Short-Term Memory

• RNN is Turing Complete
  • With Proper Configuration, it can simulate any sequence generated by a Turing Machine

• Error Vanishing and Exploding
  • Consider single unit self loop:
    $$\delta_k(\tau) = \frac{\partial J(t)}{\partial s_k(\tau)} = \frac{\partial J(t)}{\partial y_k(\tau)} \frac{\partial y_k(\tau)}{\partial s_k(\tau)} = f'_k(s_k(\tau)) \varepsilon_k(\tau) = f'_k(s_k(\tau)) w_{kk} \delta_k(\tau + 1)$$

• Long Short-term Memory

Purpose of Neural Turing Machine

• Enrich the capabilities of standard RNN to simplify the solution of algorithmic tasks
**NTM Read/Write Operation**

\[ r_t = \sum_i w_t(i)M_t(i) \]

- \( w_t(i) \): vector of weights over the N locations emitted by a read head at time t

- \( r_t \): vector of weights over the N locations

- \( M_t \): M-Size Vector

- \( M_t(i) \): weight at location i in the M-Size Vector

- \( e_t \): Erase Vector, emitted by the header

- \( a_t \): Add Vector, emitted by the header

**Erase:**
\[ M_t(i) = M_{t-1}(i)[1 - w_t(i)e_t] \]

**Add:**
\[ M_t(i) += w_t(i)a_t \]
Attention & Focus

• Attention and Focus is adjusted by weights vector across whole memory bank!
Content Based Addressing

- Find Similar Data in memory

\[ k_t : \text{Key Vector} \]
\[ \beta_t : \text{Key Strength} \]

\[ w^e_t(i) = \frac{\exp(\beta_t K(k_t, M_t(i)))}{\sum_j \exp(\beta_t K(k_t, M_t(j)))} \]

\[ K(u, v) = \frac{u \cdot v}{\|u\| \cdot \|v\|} \]
May not depend on content only…

• Gating Content Addressing

\[ g_t : \text{Interpolation gate in the range of (0,1)} \]

\[ w_t^g = g_t w_t^c + (1 - g_t) w_{t-1} \]
Convolutional Shift – Location Addressing

$s_t$: Shift weighting – Convolutional Vector (size N)

$$\tilde{w}_t(i) = \sum_{j=0}^{N-1} w_t^g(j) s_t(i-j)$$
Convolutional Shift – Location Addressing

\[ \gamma_t : \text{sharpening factor} \]

\[ w_t(i) = \frac{\tilde{w}_t(i)\gamma_t}{\sum_j \tilde{w}_t(j)\gamma_t} \]
• Goal: to demonstrate NTM is
  ▪ Able to solve the problems
  ▪ By learning compact internal programs

• Three architectures:
  ▪ NTM with a feedforward controller
  ▪ NTM with an LSTM controller
  ▪ Standard LSTM controller

• Applications
  ▪ Copy
  ▪ Repeat Copy
  ▪ Associative Recall
  ▪ Dynamic N-Grams
  ▪ Priority Sort
NTM Experiments: Copy

• Task: Store and recall a long sequence of arbitrary information

• Training: 8-bit random vectors with length 1-20

• No inputs while it generates the targets

4/6/2015
Tinghui Wang, EECS, WSU
NTM Experiments: Copy – Learning Curve

![Learning Curve Graph](image.png)

- **LSTM**
- **NTM with LSTM Controller**
- **NTM with Feedforward Controller**

**Graph Details:**
- **Y-axis:** Cost per sequence (bits)
- **X-axis:** Sequence number (thousands)

4/6/2015

Tinghui Wang, EECS, WSU
NTM Experiments: Repeated Copy

- Task: Repeat a sequence a specified number of times
- Training: random length sequences of random binary vectors followed by a scalar value indicating the desired number of copies
NTM Experiments: Repeated Copy – Learning Curve
NTM Experiments: Associative Recall

- Task: ask the network to produce the next item, given current item after propagating a sequence to network
- Training: each item is composed of three six-bit binary vectors, 2-8 items every episode
NTM Experiments: Dynamic N-Grams

• Task: Learn N-Gram Model – rapidly adapt to new predictive distribution

• Training: 6-Gram distributions over binary sequences. 200 successive bits using look-up table by drawing 32 probabilities from Beta(.5,.5) distribution

• Compare to Optimal Estimator:

\[ P(B = 1|N_1, N_2, c) = \frac{N_1 + 0.5}{N_1 + N_0 + 1} \]

4/6/2015

Tinghui Wang, EECS, WSU
NTM Experiments: Priority Sorting

- Task: Sort Binary Vector based on priority
- Training: 20 binary vectors with corresponding priorities, output 16 highest-priority vectors
NTM Experiments: Learning Curve

Figure 13: Dynamic N-Gram Learning Curves.

Figure 18: Priority Sort Learning Curves.
## Some Detailed Parameters

### NTM with Feed Forward Neural Network

<table>
<thead>
<tr>
<th>Task</th>
<th>#Heads</th>
<th>Controller Size</th>
<th>Memory Size</th>
<th>Learning Rate</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$10^{-4}$</td>
<td>17,162</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$10^{-4}$</td>
<td>16,712</td>
</tr>
<tr>
<td>Associative</td>
<td>4</td>
<td>256</td>
<td>$128 \times 20$</td>
<td>$10^{-4}$</td>
<td>146,845</td>
</tr>
<tr>
<td>N-Grams</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$3 \times 10^{-5}$</td>
<td>14,656</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>8</td>
<td>512</td>
<td>$128 \times 20$</td>
<td>$3 \times 10^{-5}$</td>
<td>508,305</td>
</tr>
</tbody>
</table>

### NTM with LSTM

<table>
<thead>
<tr>
<th>Task</th>
<th>#Heads</th>
<th>Controller Size</th>
<th>Memory Size</th>
<th>Learning Rate</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$10^{-4}$</td>
<td>67,561</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$10^{-4}$</td>
<td>66,111</td>
</tr>
<tr>
<td>Associative</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$10^{-4}$</td>
<td>70,330</td>
</tr>
<tr>
<td>N-Grams</td>
<td>1</td>
<td>100</td>
<td>$128 \times 20$</td>
<td>$3 \times 10^{-5}$</td>
<td>61,749</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>5</td>
<td>$2 \times 100$</td>
<td>$128 \times 20$</td>
<td>$3 \times 10^{-5}$</td>
<td>269,088</td>
</tr>
</tbody>
</table>

### LSTM

<table>
<thead>
<tr>
<th>Task</th>
<th>Network Size</th>
<th>Learning Rate</th>
<th>#Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy</td>
<td>$3 \times 256$</td>
<td>$3 \times 10^{-5}$</td>
<td>1,352,969</td>
</tr>
<tr>
<td>Repeat Copy</td>
<td>$3 \times 512$</td>
<td>$3 \times 10^{-5}$</td>
<td>5,312,007</td>
</tr>
<tr>
<td>Associative</td>
<td>$3 \times 256$</td>
<td>$10^{-4}$</td>
<td>1,344,518</td>
</tr>
<tr>
<td>N-Grams</td>
<td>$3 \times 128$</td>
<td>$10^{-4}$</td>
<td>331,005</td>
</tr>
<tr>
<td>Priority Sort</td>
<td>$3 \times 128$</td>
<td>$3 \times 10^{-5}$</td>
<td>384,424</td>
</tr>
</tbody>
</table>
Additional Information

- Literature Reference:

- NTM Implementation
  - https://github.com/shawntan/neural-turing-machines

- NTM Reddit Discussion
  - https://www.reddit.com/r/MachineLearning/comments/2m9mga/implementation_of_neural_turing_machines/