Differentiable Neural Computers

Hybrid Computing using a neural network with dynamic external memory (Graves et al. 2016)

Konstantinos Kogkalidis

May 28, 2018

Logic and Computation

Cross-domain

- Data Flow Programming
- Bayesian Reasoning
- Machine Learning
- Functional Programming

Overview: Probabilistic Programming

Cross-domain

- Data Flow Programming
- Bayesian Reasoning
- Machine Learning
- Functional Programming

Intuition: *"Rather than explicitly write a program, write some constraints on the behavior of the desired program and use computational tools to search the program space for models satisfying these constraints."*

Overview: Probabilistic Programming

Cross-domain

- Data Flow Programming
- Bayesian Reasoning
- Machine Learning
- Functional Programming

Intuition: *"Rather than explicitly write a program, write some constraints on the behavior of the desired program and use computational tools to search the program space for models satisfying these constraints."*

PROGRAM	MODEL
Discrete	Continuous
Deterministic	Stochastic
Static	Adaptive

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs
  - End-to-end differentiable

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs
  - End-to-end differentiable
  - Auto-associative memory

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs
  - End-to-end differentiable
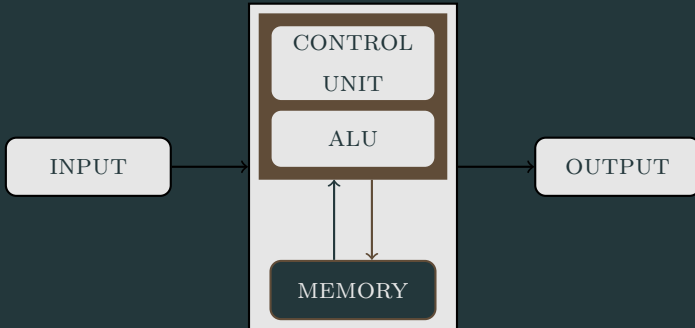 - Auto-associative memory
  - Turing complete

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs
    - End-to-end differentiable
    - Auto-associative memory
    - Turing complete
    + Memory attention mechanisms

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs
    - End-to-end differentiable
    - Auto-associative memory
    - Turing complete
    + Memory attention mechanisms
- Mimic mammalian biological memory
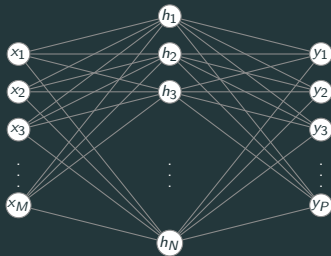- Employ classical concepts of computation

Von Neumann architecture

Simple Neural Net

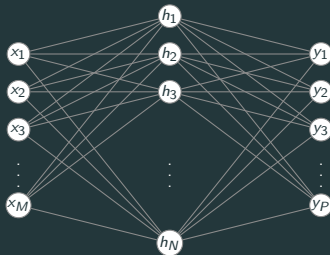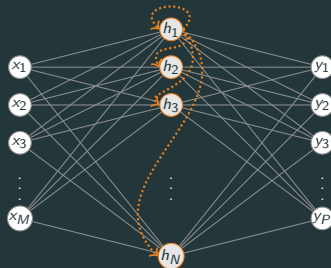

$$y = g(h), \ h = f(x)$$

No memory

Simple Neural Net

$y = g(h), \ h = f(x)$

No memory

Simple Recurrent Net

$h(t) = f([x(t); h(t-1)])$

Finite, non-contiguous memory

Introduction: RNNs

Simple Neural Net

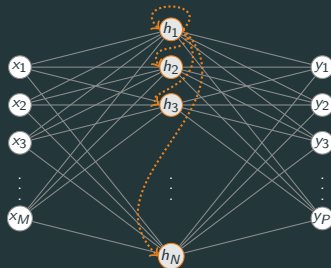## Simple Recurrent Net

$$y = g(h), \ h = f(x)$$

No memory

$$h(t) = f([x(t); h(t-1)])$$

Finite, non-contiguous memory

"If training vanilla neural nets is optimization over functions, training recurrent nets is optimization over programs."

Approach

Train a RNN to act as a controller to interact with a memory matrix of N (arbitrary many) addresses.

Approach

Train a RNN to act as a controller to interact with a memory matrix of N (arbitrary many) addresses.

1. Content Lookup
   - Attention over memory defined by weightings $W \in \mathbb{R}^N$
   - Compare controller output with memory objects (auto-associative memory)
   - Allow partial matches (pattern completion)

Approach

Train a RNN to act as a controller to interact with a memory matrix of N (arbitrary many) addresses.

1. Content Lookup
   - Attention over memory defined by weightings $W \in \mathbb{R}^N$
   - Compare controller output with memory objects (auto-associative memory)
   - Allow partial matches (pattern completion)
2. Sequential Retrieval
   - Fill $L \in \{0,1\}^{2N}$ indexing temporal transitions
   - Shift operations defined by LW, $L^T W$

Approach

Train a RNN to act as a controller to interact with a memory matrix of N (arbitrary many) addresses.

1. Content Lookup
   - Attention over memory defined by weightings $W \in \mathbb{R}^N$
   - Compare controller output with memory objects (auto-associative memory)
   - Allow partial matches (pattern completion)
2. Sequential Retrieval
   - Fill $L \in \{0,1\}^{2N}$ indexing temporal transitions
   - Shift operations defined by LW, $L^T W$
3. Dynamic Allocation
   - Mark memory locations with $\{0,1\}$ to signal usage
   - Manipulate signals during R/W operations to enable reallocation
   - Generalization to unbounded memory

A deep long short-term memory network receiving input:

$$\boldsymbol{\mathcal{X}}_t = [\boldsymbol{x}_t; \boldsymbol{r}_{t-1}^1; \ldots; \boldsymbol{r}_{t-1}^R]$$

and producing output:

$$(\boldsymbol{v}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_T]; \vartheta)$$

where $\mathcal{N}$ a set of state equations and ϑ their trainable parameters.

A more detailed look into $\mathcal{N}$:

$$\boldsymbol{i}_t^l = \sigma(W_i^l[\boldsymbol{\mathcal{X}}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}] + \boldsymbol{b}_i^l) \qquad \text{(input gate)}$$

$$\boldsymbol{f}_t^l = \sigma(W_f^l[\boldsymbol{\mathcal{X}}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}] + \boldsymbol{b}_f^l) \qquad \text{(forget gate)}$$

$$\boldsymbol{s}_t^l = \boldsymbol{f}_t^l \boldsymbol{s}_{t-1}^l + \boldsymbol{i}_t^l tanh(W_s^l[\boldsymbol{\mathcal{X}}_t; \boldsymbol{h}_{t_1}^l; \boldsymbol{h}_t^{l-1}] + \boldsymbol{b}_s^l) \qquad \text{(state)}$$

$$\boldsymbol{o}_t^l = \sigma(W_o^l[\boldsymbol{\mathcal{X}}_t; \boldsymbol{h}_{t-1}^l; \boldsymbol{h}_t^{l-1}] + \boldsymbol{b}_o^l) \qquad \text{(output gate)}$$

$$\boldsymbol{h}_t^l = \boldsymbol{o}_t^l tanh(\boldsymbol{s}_t^l) \qquad \text{(hidden)}$$

$$\boldsymbol{v}_t = W_y[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L] \qquad \text{(output vector)}$$

$$\boldsymbol{\xi}_t = W_\xi[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L] \qquad \text{(interface vector)}$$

Single LSTM layer

$$(\boldsymbol{v}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_T]; \vartheta)$$

$$(\boldsymbol{v}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_T]; \vartheta)$$
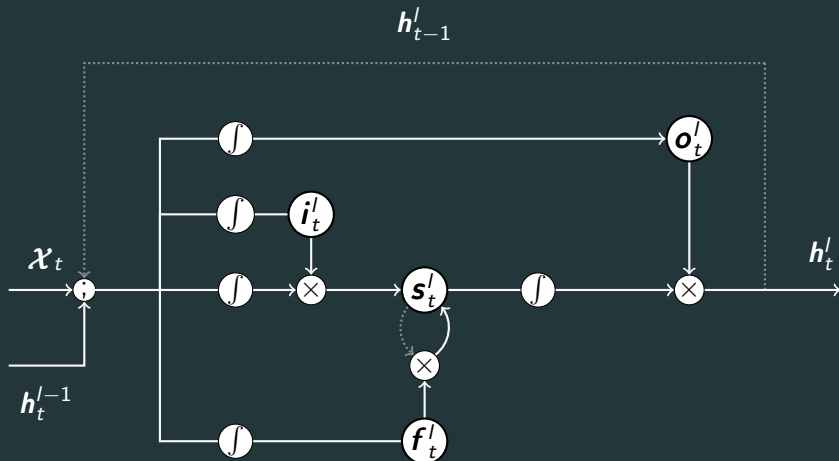
Output given by:

$$\boldsymbol{y}_t = \boldsymbol{v}_t + W_R[\boldsymbol{r}_t^1; \ldots; \boldsymbol{r}_t^R] \qquad \text{(hello)}$$

$$(\boldsymbol{v}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_T]; \vartheta)$$

Controller: Outputs

$$(\boldsymbol{v}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_T]; \vartheta)$$

$$\boldsymbol{\xi}_t = [\boldsymbol{k}_t^{r,1}; \ldots; \boldsymbol{k}_t^{r,R}; \ \hat{\beta}_t^{r,1}; \ldots; \hat{\beta}_t^{r,R}; \ \boldsymbol{k}_t^w; \ \hat{\beta}_t^w;$$
$$\hat{\boldsymbol{e}}_t; \ \boldsymbol{v}_t; \ \hat{\boldsymbol{f}}_t^1; \ldots; \hat{\boldsymbol{f}}_t^R; \ \hat{g}_t^a; \ \hat{g}_t^w; \ \hat{\boldsymbol{\pi}}_t^1; \ldots; \hat{\boldsymbol{\pi}}_t^R;]$$

$$(\boldsymbol{v}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_T]; \vartheta)$$

$$\boldsymbol{\xi}_t = [\overbrace{\boldsymbol{k}_t^{r,1}; \ldots; \boldsymbol{k}_t^{r,R}}^{\text{read keys}}; \overbrace{\hat{\beta}_t^{r,1}; \ldots; \hat{\beta}_t^{r,R}}^{\text{read strengths}}; \overbrace{\boldsymbol{k}_t^{w}; \hat{\beta}_t^{w}}^{\text{write key/strength}}; \overbrace{\hat{\boldsymbol{e}}_t}^{\text{erase vector}};$$

$$\underbrace{\boldsymbol{v}_t}_{\text{write vector}}; \underbrace{\hat{\boldsymbol{f}}_t^{1}; \ldots; \hat{\boldsymbol{f}}_t^{R}}_{\text{free gates}}; \underbrace{\hat{g}_t^{a}; \hat{g}_t^{w}}_{\text{allocation/write gates}}; \underbrace{\hat{\boldsymbol{\pi}}_t^{1}; \ldots; \hat{\boldsymbol{\pi}}_t^{R}}_{\text{read modes}}]$$

- Neural Turing Machines (Graves, Wayne, Danihelka)
- Entity Networks (Henaff, Weston, Szlam, Bordes, LeCun)
- Principles of Probabilistic Programming Languages (Goodman)
- Backprop as a Functor (Fong, Spivak, Tuyras)
- Formal Methods for Probabilistic Programming (Selsam, Liang, Dill)