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

Overview: Probabilistic Programming

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

Overview: DNC

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

Overview: DNC

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs

Overview: DNC

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Extension of NTMs
    - End-to-end differentiable

Overview: DNC

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

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

Overview: DNC

Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

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

Overview: DNC

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

Overview: DNC

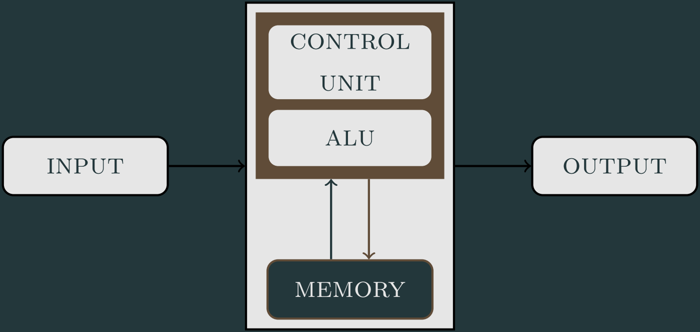
Differentiable Neural Computer

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

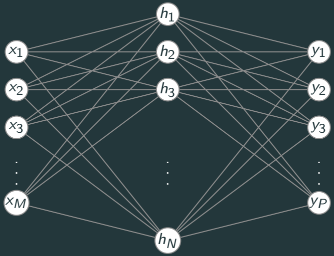
Introduction: Classic Computation

Von Neumann architecture



Introduction: RNNs

Simple Neural Net

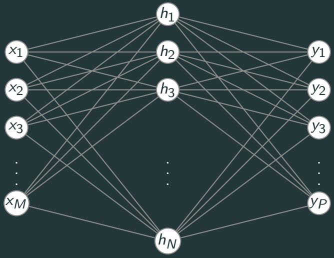


$$NN : \boldsymbol{x}_{t_i} \mapsto \boldsymbol{y}_{t_i}$$

No memory

Introduction: RNNs

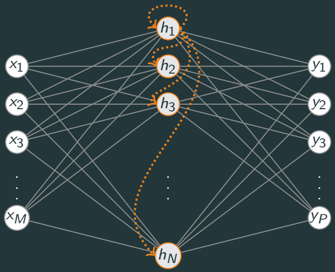
Simple Neural Net



$$NN : \boldsymbol{x}_{t_i} \mapsto \boldsymbol{y}_{t_i}$$

No memory

Simple Recurrent Net

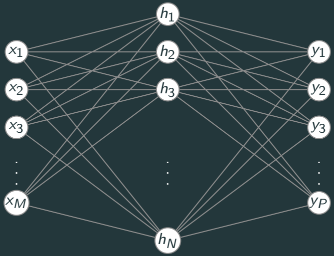


$$RNN : \boldsymbol{x}_{t_0} \otimes \boldsymbol{x}_{t_1} \otimes \cdots \otimes \boldsymbol{x}_{t_i} \mapsto \boldsymbol{y}_{t_i}$$

Finite memory

Introduction: RNNs

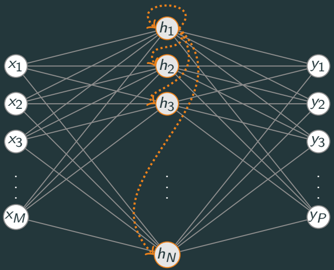
Simple Neural Net



$NN : \boldsymbol{x}_{t_i} \mapsto \boldsymbol{y}_{t_i}$

No memory

Simple Recurrent Net



$RNN : \boldsymbol{x}_{t_0} \otimes \boldsymbol{x}_{t_1} \otimes \cdots \otimes \boldsymbol{x}_{t_i} \mapsto \boldsymbol{y}_{t_i}$

Finite memory

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

Approach

Train a RNN to act as the controller of a memory matrix M of N addresses through R read heads and one write head.

Approach

Train a RNN to act as the controller of a memory matrix M of N addresses through R read heads and one write head.

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 the controller of a memory matrix M of N addresses through R read heads and one write head.

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 the controller of a memory matrix M of N addresses through R read heads and one write head.

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

Controller: Overview

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.

Controller: State Equations

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

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

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

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

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

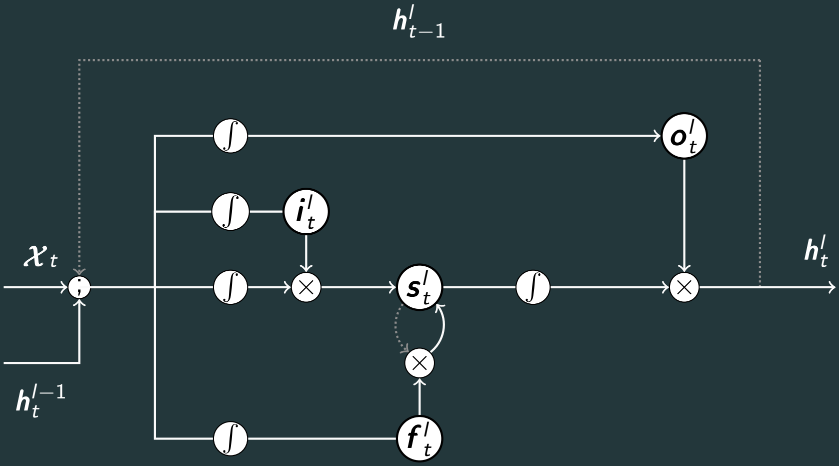
$$\boldsymbol{h}_t^l = \boldsymbol{o}_t^l tanh(\boldsymbol{s}_t^l) \qquad \text{(hidden)}$$

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

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

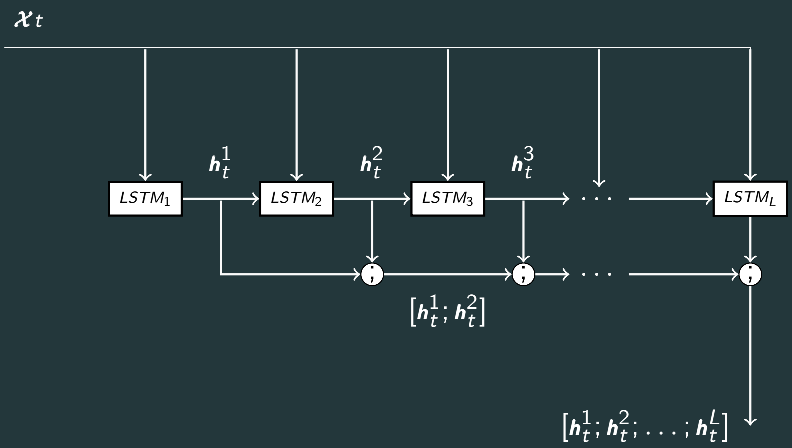
Controller: Signal-Flow (1/2)

Single LSTM layer



Controller: Signal-Flow (2/2)

LSTM Network (multiple layers)



Controller: Outputs

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

Controller: Outputs

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

Intermediate output $\boldsymbol{\upsilon}_t = W_y[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L]$

$$\boldsymbol{y}_t = \boldsymbol{\upsilon}_t + W_R[\boldsymbol{r}_t^1; \ldots; \boldsymbol{r}_t^R] \qquad \text{(Memory-conditioning)}$$

Controller: Outputs

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

Intermediate output $\boldsymbol{\upsilon}_t = W_y[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L]$

$$\boldsymbol{y}_t = \boldsymbol{\upsilon}_t + W_R[\boldsymbol{r}_t^1; \ldots; \boldsymbol{r}_t^R] \qquad \text{(Memory-conditioning)}$$

Interface vector $\boldsymbol{\xi}_t = W_\xi[\boldsymbol{h}_t^1; \ldots; \boldsymbol{h}_t^L]$

- Read keys
- Read strengths
- Write key
- Write strength
- Erase vector
- Write vector
- Free gates
- Allocation gate
- Write gate
- Read modes

Memory Adressing: Content-Lookup

R read keys $\boldsymbol{k}^{r,i} \in \mathbb{R}^W, \; i = 1 \ldots R$

R read strengths $\beta^{r,i} \in [1, \infty), \; i = 1 \ldots R$

Write key $\boldsymbol{k}^w \in \mathbb{R}^W$

Write strength $\beta^w \in [1, \infty)$

Memory Adressing: Content-Lookup

R read keys $\boldsymbol{k}^{r,i} \in \mathbb{R}^W, \; i = 1 \dots R$

R read strengths $\beta^{r,i} \in [1, \infty), \; i = 1 \dots R$

Write key $\boldsymbol{k}^w \in \mathbb{R}^W$

Write strength $\beta^w \in [1, \infty)$

Weightings w given by $\mathcal{C}$

$$\mathcal{C}(M, \boldsymbol{k}, \beta)[i] = \frac{exp\{\mathcal{D}(\boldsymbol{k}, M[i,:])\beta\}}{\sum_j exp\{\mathcal{D}(\boldsymbol{k}, M[j,:])\beta\}}$$

Memory Adressing: Content-Lookup

R read keys $\boldsymbol{k}^{r,i} \in \mathbb{R}^W, \; i = 1 \dots R$

R read strengths $\beta^{r,i} \in [1, \infty), \; i = 1 \dots R$

Write key $\boldsymbol{k}^w \in \mathbb{R}^W$

Write strength $\beta^w \in [1, \infty)$

Weightings w given by $\mathcal{C}$

$$\mathcal{C}(M, \boldsymbol{k}, \beta)[i] = \frac{exp\{\mathcal{D}(\boldsymbol{k}, M[i, :])\beta\}}{\sum_j exp\{\mathcal{D}(\boldsymbol{k}, M[j, :])\beta\}}$$

Read operations

$$\boldsymbol{r}_t^i = M_t^T \boldsymbol{w}_t^{r,i}$$

Write operations

$$M_t = M_{t-1} \circ (\mathbf{1} - \boldsymbol{w}_t^w \boldsymbol{e}_t^T) + \boldsymbol{w}_t^w \boldsymbol{v}_t^T$$

Further Reading

- Neural Turing Machines (Graves, Wayne, Danihelka)
- Entity Networks (Henaff, Weston, Szlam, Bordes, LeCun)
- End-to-End Memory Networks (Sukhbaatar, Szlam, Weston, Fergus)
- Jointly Learning to Align and Translate (Bahdanau, Cho, Bengio)
- Principles of Probabilistic Programming Languages (Goodman)
- Backprop as a Functor (Fong, Spivak, Tuyras)
- Formal Methods for Probabilistic Programming (Selsam, Liang, Dill)