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

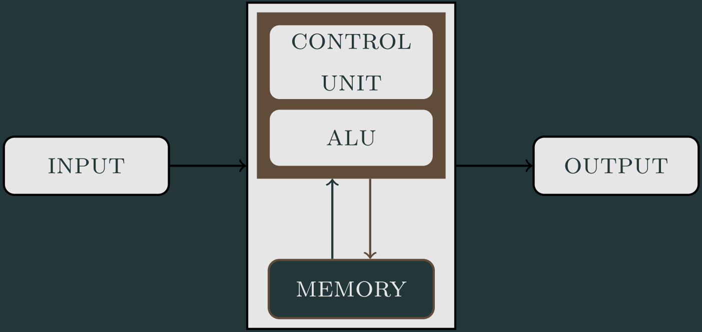
Differentiable Neural Computer

A recurrent neural network coupled with an external memory.

- Functional replication of biological memory
- Reimaging of classical concepts of computation
- Extension of NTMs
  - End-to-end differentiable
  - Auto-associative memory
  - Turing complete
  - \+ Computationally efficient memory management

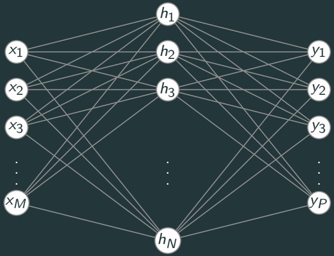
Introduction: Classic Computation

Von Neumann architecture



Introduction: Neural Networks

Simple Neural Net

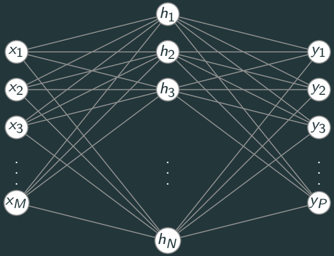


$$NN : \boldsymbol{x}_t \mapsto \boldsymbol{y}_t$$

No memory

Introduction: Neural Networks

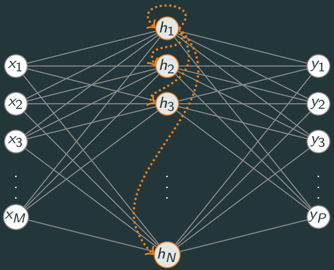
Simple Neural Net



$NN : \boldsymbol{x}_t \mapsto \boldsymbol{y}_t$

No memory

Simple Recurrent Net

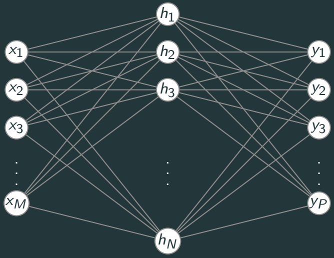


$RNN : \boldsymbol{x}_0 \otimes \boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_t \mapsto \boldsymbol{y}_t$

Finite memory

Introduction: Neural Networks

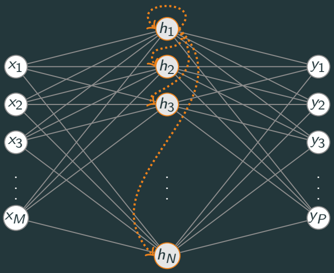
Simple Neural Net



$NN : \boldsymbol{x}_t \mapsto \boldsymbol{y}_t$

No memory

Simple Recurrent Net



$RNN : \boldsymbol{x}_0 \otimes \boldsymbol{x}_1 \otimes \cdots \otimes \boldsymbol{x}_t \mapsto \boldsymbol{y}_t$

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 $M \in \mathbb{R}^{N \times W}$ through R read heads and one write head.

Approach

Train a RNN to act as the controller of a memory matrix $M \in \mathbb{R}^{N \times W}$ through R read heads and one write head.

1. Content Lookup
   - Attention over memory defined by weightings $w \in \mathcal{S}^N$
   - 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 $M \in \mathbb{R}^{N \times W}$ through R read heads and one write head.

1. Content Lookup
    - Attention over memory defined by weightings $w \in \mathcal{S}^N$
    - Compare controller output with memory objects (auto-associative memory)
    - Allow partial matches (pattern completion)
2. Sequential Retrieval
    - Fill $L \in [0, 1]^{N \times N}$ indexing temporal transitions
    - Shift operations defined by $Lw, \; L^\top w$

Approach

Train a RNN to act as the controller of a memory matrix $M \in \mathbb{R}^{N \times W}$ through R read heads and one write head.

1. Content Lookup
   - Attention over memory defined by weightings $w \in \mathcal{S}^N$
   - Compare controller output with memory objects (auto-associative memory)
   - Allow partial matches (pattern completion)
2. Sequential Retrieval
   - Fill $L \in [0,1]^{N \times N}$ indexing temporal transitions
   - Shift operations defined by $Lw, \; L^\top w$
3. Dynamic Allocation
   - Mark memory locations with $\boldsymbol{u} \in [0,1]^N$ 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:

$$\boldsymbol{\mathcal{X}}_t = [\boldsymbol{x}_t; \boldsymbol{r}^1_{t-1}; \ldots; \boldsymbol{r}^R_{t-1}] \qquad \text{(timestep t)}$$

and producing output:

$$(\boldsymbol{y}_t, \boldsymbol{\xi}_t) = \mathcal{N}([\boldsymbol{\mathcal{X}}_1; \ldots; \boldsymbol{\mathcal{X}}_t]; \vartheta) \qquad \text{(entire sequence)}$$

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

LSTM: Overview

LSTM Network

Multiple stacked LSTM units.

LSTM Unit

A RNN that has an intrinsic memory cell $\boldsymbol{c}_t^l$ and three gates.

LSTM: Overview

LSTM Network

Multiple stacked LSTM units.

LSTM Unit

A RNN that has an intrinsic memory cell $\boldsymbol{c}_t^l$ and three gates.

1. Input gate $\boldsymbol{i}_t^l$
2. Forget gate $\boldsymbol{f}_t^l$
3. Output gate $\boldsymbol{o}_t^l$

LSTM: Signal-Flow (1/2)

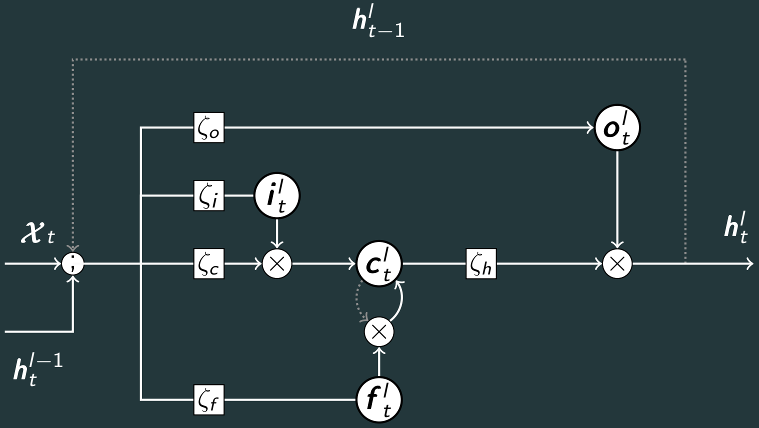
LSTM Unit (single layer)

- Input: $[\mathcal{X}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t]$
- Output: $\boldsymbol{h}^l_t$

LSTM: Signal-Flow (1/2)

LSTM Unit (single layer)

- Input: $[\mathcal{X}_t; \boldsymbol{h}^l_{t-1}; \boldsymbol{h}^{l-1}_t]$
- Output: $\boldsymbol{h}^l_t$



Controller: Signal-Flow (2/2)

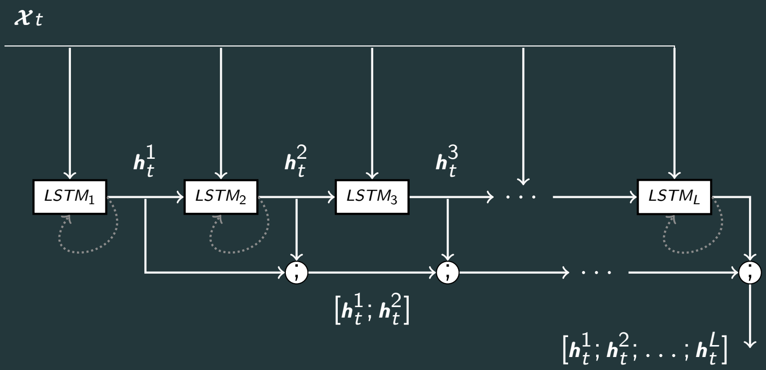
LSTM Network (multiple layers)

- Input: $\boldsymbol{\mathcal{X}}_t$
- Output: $[\boldsymbol{h}_t^1; \boldsymbol{h}_t^2; \ldots \boldsymbol{h}_t^L]$

Controller: Signal-Flow (2/2)

LSTM Network (multiple layers)

- Input: $\mathcal{X}_t$
- Output: $[\boldsymbol{h}_t^1; \boldsymbol{h}_t^2; \ldots \boldsymbol{h}_t^L]$



Controller: Outputs

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

Controller: Outputs

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

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

Controller: Outputs

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

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

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

- Read keys: $\boldsymbol{k}_t^{r,i}$
- Read strengths: $\beta_t^{r,i}$
- Write key: $\boldsymbol{k}_t^w$
- Write strength: β_t^w
- Erase vector: $\boldsymbol{e}_t$
- Write vector: $\boldsymbol{v}_t$
- Free gates: $\boldsymbol{\phi}_t^i$
- Allocation gate: g_t^a
- Write gate: g_t^w
- Read modes: $\boldsymbol{\pi}_t^i$

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

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

Matching function $\mathcal{D}$ comparing memory contents

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

Matching function $\mathcal{D}$ comparing memory contents

Weighting function $\mathcal{C}$ normalizing and sharpening matches

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

Memory Adressing: R/W

Attention dictated by weightings $\boldsymbol{w} \in \mathcal{S}^N$

Erase vector $\boldsymbol{e}_t \in [0,1]^W$

Write vector $\boldsymbol{v}_t \in \mathbb{R}^W$

Read operations

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

Memory Adressing: R/W

Attention dictated by weightings $\boldsymbol{w} \in \mathcal{S}^N$

Erase vector $\boldsymbol{e}_t \in [0,1]^W$

Write vector $\boldsymbol{v}_t \in \mathbb{R}^W$

Read operations

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

Write operations

$$M_t = \underbrace{M_{t-1} \circ (\mathbf{1} - \boldsymbol{w}_t^w \boldsymbol{e}_t^\top)}_{\text{erased memory}} + \underbrace{\boldsymbol{w}_t^w \boldsymbol{v}_t^\top}_{\text{new write}}$$

Memory Adressing: Dynamic Allocation

Free gates $\boldsymbol{\phi}_t^i \in [0,1]^W$
Allocation gate $g_t^a \in [0,1]$
Write gate $g_t^w \in [0,1]$

"Free list" scheme

$$\boldsymbol{\psi}_t = \prod_{i=1}^{R} (\mathbf{1} - \boldsymbol{\phi}_t^i w_{t-1}^{r,i}) \qquad \text{(memory retention)}$$

$$u_t = (\boldsymbol{u}_{t-1} + \boldsymbol{w}_{t-1}^w - \boldsymbol{u}_{t-1} \circ \boldsymbol{w}_{t-1}^w) \circ \boldsymbol{\psi}_t \qquad \text{(usage tracking)}$$

Memory Adressing: Dynamic Allocation

Free gates $\phi_t^i \in [0,1]^W$
Allocation gate $g_t^a \in [0,1]$
Write gate $g_t^w \in [0,1]$

"Free list" scheme

$$\psi_t = \prod_{i=1}^{R} (\mathbf{1} - \phi_t^i w_{t-1}^{r,i}) \qquad \text{(memory retention)}$$

$$u_t = (\boldsymbol{u}_{t-1} + \boldsymbol{w}_{t-1}^w - \boldsymbol{u}_{t-1} \circ \boldsymbol{w}_{t-1}^w) \circ \psi_t \qquad \text{(usage tracking)}$$

Attention shift

- Obtain the allocation vector $\boldsymbol{a}_t$ by normalizing $\boldsymbol{u}_t$
- Shift $\boldsymbol{w}_t$ by $g_t^a \boldsymbol{a}_t$ and scale by g_t^w

Memory Adressing: Temporal Linking

Read modes: $\boldsymbol{\pi}_t^i \in \mathcal{S}_3$

Memory Adressing: Temporal Linking

Read modes: $\boldsymbol{\pi}_t^i \in \mathcal{S}_3$

Temporal Transition $\boldsymbol{L}_t \in [0,1]^{N \times N}$

$$\boldsymbol{L}[i,j] = \underbrace{(1 - w_t^W[i] - w_t^W[j])\boldsymbol{L}_{t-1}[i,j]}_{\text{Part of last transition}} + \underbrace{w_t^w[i] w_{t-1}^w[j]}_{\text{Current transition}}$$

Memory Adressing: Temporal Linking

Read modes: $\boldsymbol{\pi}_t^i \in \mathcal{S}_3$

Temporal Transition $\boldsymbol{L}_t \in [0,1]^{N \times N}$

$$\boldsymbol{L}[i,j] = \underbrace{(1 - w_t^W[i] - w_t^W[j])\boldsymbol{L}_{t-1}[i,j]}_{\text{Part of last transition}} + \underbrace{w_t^w[i] w_{t-1}^w[j]}_{\text{Current transition}}$$

Mode Interpolation

$$\boldsymbol{w}_t^{r,i} = \underbrace{\boldsymbol{\pi}_t^i[1] L \boldsymbol{w}_t^{r,i}}_{\text{Forward shift}} + \underbrace{\boldsymbol{\pi}_t^i[2] \boldsymbol{w}_t^{r,i}}_{\text{No shift}} + \underbrace{\boldsymbol{\pi}_t^i[3] L^\top \boldsymbol{w}_t^{r,i}}_{\text{Backward shift}}$$

Experiments

Experiments

- Language Tasks

  Inference

  Logical Reasoning

Experiments

- Language Tasks
  - Inference
  - Logical Reasoning
- Graph Tasks
  - Network Traversal
  - Policy Learning

Experiments

- Language Tasks
  - Inference
  - Logical Reasoning
- Graph Tasks
  - Network Traversal
  - Policy Learning
- Block Puzzles

Conclusion

Recap

We can simulate computation and expand upon it by using differentiable, continuous operations.

Conclusion

Recap

We can simulate computation and expand upon it by using differentiable, continuous operations.

Takeaway

We can automatically infer simple functions over complex data structures in the form of probability distributions, just by using examples!

Thank you!

Questions?

Appendix: LSTM Equations

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

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

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

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

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

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

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

Appendix: Further Reading

Neural Architectures

- Learning to Forget (Gers, Schmidhuber, Cummins)
- 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)

Probabilistic Programming

- Principles of Probabilistic Programming Languages (Goodman)
- Backprop as a Functor (Fong, Spivak, Tuyras)
- Formal Methods for Probabilistic Programming (Selsam, Liang, Dill)