

# 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
- Fuzzy Logic
- Functional Programming

# Overview: Probabilistic Programming

## Cross-domain

- Data Flow Programming
- Bayesian Reasoning / Machine Learning
- Fuzzy Logic
- 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
- Fuzzy Logic
- 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

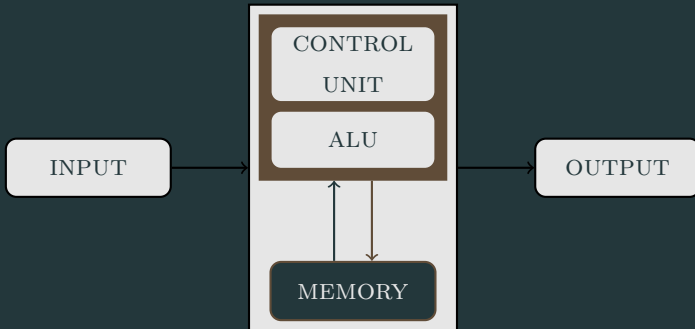
## Differentiable Neural Computer

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

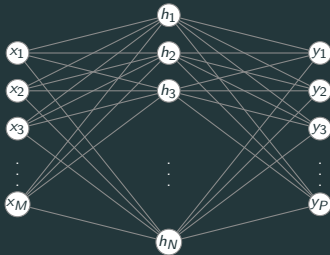
# Introduction: Classic Computation

## Von Neumann architecture



# Introduction: Neural Networks

## Simple Neural Net



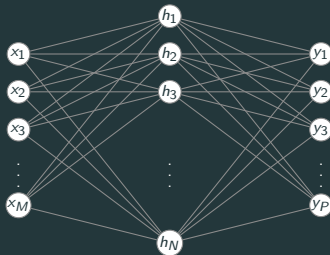
$$NN : \mathbf{x}_t \mapsto \mathbf{y}_t$$

No memory



# Introduction: Neural Networks

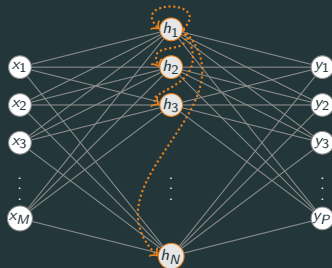
## Simple Neural Net



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

No memory

## Simple Recurrent Net



$$RNN : \mathbf{x}_0 \otimes \mathbf{x}_1 \otimes \dots \otimes \mathbf{x}_t \mapsto \mathbf{y}_t$$

Finite memory

# Introduction: Neural Networks

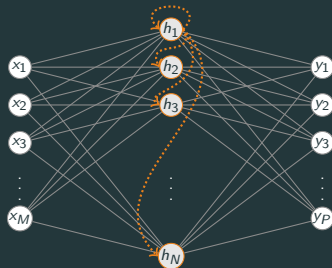
## Simple Neural Net



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

No memory

## Simple Recurrent Net



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

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  $u \in [0, 1]^N$  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:

$$\boldsymbol{\mathcal{X}}_t = [\boldsymbol{x}_t; \boldsymbol{r}_{t-1}^1; \dots; \boldsymbol{r}_{t-1}^R] \quad (\text{timestep } t)$$

and producing output:

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

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

# LSTM: Overview

## LSTM Network

Multiple stacked LSTM units.

## LSTM Unit

A RNN that has an intrinsic memory cell  $c_t^I$  and three gates.



# LSTM: Overview

## LSTM Network

Multiple stacked LSTM units.

## LSTM Unit

A RNN that has an intrinsic memory cell  $c_t^l$  and three gates.

1. Input gate  $i_t^l$
2. Forget gate  $f_t^l$
3. Output gate  $o_t^l$

## LSTM: Signal-Flow (1/2)

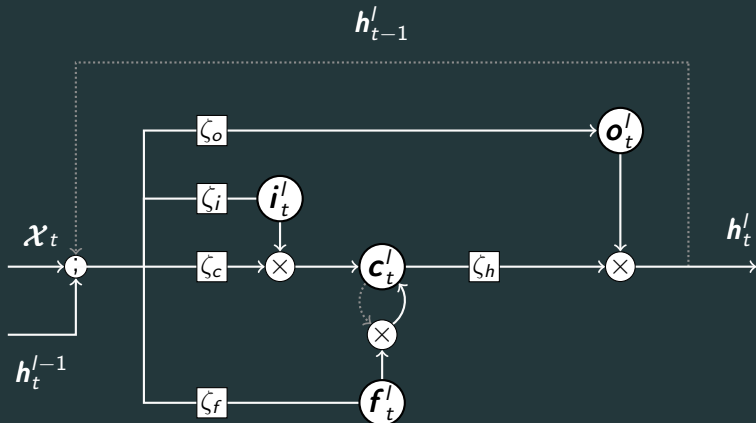
### LSTM Unit (single layer)

- Input:  $[\mathcal{X}_t; h'_{t-1}; h'^{-1}_t]$
- Output:  $h'_t$

## LSTM: Signal-Flow (1/2)

### LSTM Unit (single layer)

- Input:  $[\mathcal{X}_t; h_{t-1}^l; h_t^{l-1}]$
- Output:  $h_t^l$



## Controller: Signal-Flow (2/2)

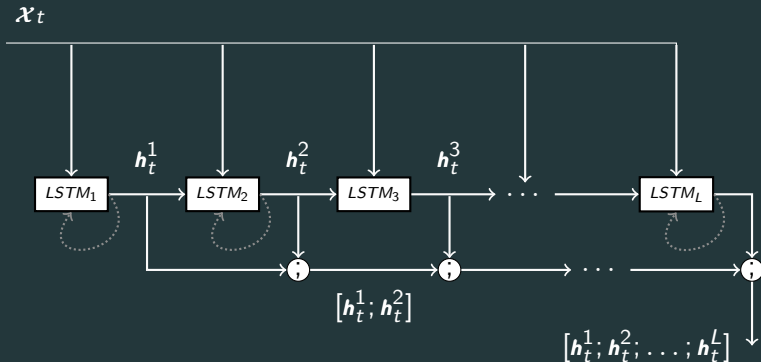
LSTM Network (multiple layers)

- Input:  $\mathcal{X}_t$
- Output:  $[h_t^1; h_t^2; \dots h_t^L]$

## Controller: Signal-Flow (2/2)

### LSTM Network (multiple layers)

- Input:  $\mathcal{X}_t$
- Output:  $[h_t^1; h_t^2; \dots; h_t^L]$



## Controller: Outputs

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

## Controller: Outputs

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

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

## Controller: Outputs

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

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

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

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



## Memory Addressing: Content-Lookup

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

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

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

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

## Memory Addressing: Content-Lookup

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

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

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

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

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

## Memory Addressing: Content-Lookup

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

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

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

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

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

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

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

## Memory Addressing: R/W

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

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

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

Read operations

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

# Memory Addressing: R/W

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

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

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

Read operations

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

Write operations

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

# Memory Addressing: 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

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

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

# Memory Addressing: 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

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

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

Attention shift

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

# Memory Addressing: Temporal Linking

Read modes:  $\pi_t^i \in \mathcal{S}_3$



# Memory Addressing: Temporal Linking

Read modes:  $\pi_t^i \in \mathcal{S}_3$

Temporal Transition  $L_t \in [0, 1]^{N \times N}$

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

# Memory Addressing: Temporal Linking

Read modes:  $\pi_t^i \in \mathcal{S}_3$

Temporal Transition  $L_t \in [0, 1]^{N \times N}$

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

Mode Interpolation

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

# Experiments

- Language Tasks
  - Inference
  - Logical Reasoning

# Experiments

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

# 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

$$i_t' = \sigma(W_i'[\mathbf{x}_t; \mathbf{h}_{t-1}'; \mathbf{h}_t'^{-1}] + \mathbf{b}_i') \quad (\text{input gate})$$

$$f_t' = \sigma(W_f'[\mathbf{x}_t; \mathbf{h}_{t-1}'; \mathbf{h}_t'^{-1}] + \mathbf{b}_f') \quad (\text{forget gate})$$

$$\mathbf{s}_t' = f_t' \mathbf{s}_{t-1}' + i_t' \tanh(W_s'[\mathbf{x}_t; \mathbf{h}_{t-1}'; \mathbf{h}_t'^{-1}] + \mathbf{b}_s') \quad (\text{state})$$

$$\mathbf{o}_t' = \sigma(W_o'[\mathbf{x}_t; \mathbf{h}_{t-1}'; \mathbf{h}_t'^{-1}] + \mathbf{b}_o') \quad (\text{output gate})$$

$$\mathbf{h}_t' = \mathbf{o}_t' \tanh(\mathbf{s}_t') \quad (\text{hidden})$$

$$\mathbf{v}_t = W_y[\mathbf{h}_t^1; \dots; \mathbf{h}_t^L] \quad (\text{output vector})$$

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

## 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)