Lecture 13
Neural Networks with External Memory

CMSC 35246: Deep Learning

Shubhendu Trivedi
&
Risi Kondor

University of Chicago

May 10, 2017

Neural Networks with Explicit Memory

A Drawback

- We have looked at a bunch of supervised neural network models

A Drawback

- We have looked at a bunch of supervised neural network models
- These models (such as for object recognition, machine translation) slowly absorb the examples into their weights to learn the concept over successive gradient descent iterations

A Drawback

- Knowledge about the concepts is held *implicitly* in the weights

A Drawback

- Knowledge about the concepts is held *implicitly* in the weights
- This accords networks with a very limited short term memory

A Drawback

- Knowledge about the concepts is held *implicitly* in the weights
- This accords networks with a very limited short term memory
- In many real tasks of interest we want to be able to explicitly store, and be able to access and manipulate such information

A Drawback

- Knowledge about the concepts is held *implicitly* in the weights
- This accords networks with a very limited short term memory
- In many real tasks of interest we want to be able to explicitly store, and be able to access and manipulate such information
- Traditional frameworks struggle with memorizing facts and being able to manipulate information for some task of interest (such as question answering, programming need longer term memory, out of sequence accesses to information)

A Drawback

- Knowledge about the concepts is held *implicitly* in the weights
- This accords networks with a very limited short term memory
- In many real tasks of interest we want to be able to explicitly store, and be able to access and manipulate such information
- Traditional frameworks struggle with memorizing facts and being able to manipulate information for some task of interest (such as question answering, programming need longer term memory, out of sequence accesses to information)
- Solution: Endow a Neural Network with an external memory that it can read from and write to

A Drawback

- A memory unit has pieces of information stored at different *locations*

A Drawback

- A memory unit has pieces of information stored at different *locations*
- We want to be able to:
  - Access locations

A Drawback

- A memory unit has pieces of information stored at different *locations*
- We want to be able to:
  - Access locations
  - Read/write information from/to these locations

A Drawback

- A memory unit has pieces of information stored at different *locations*
- We want to be able to:
  - Access locations
  - Read/write information from/to these locations
- Attention mechanisms give us a way to point to specific chunks (we saw two examples - Machine Translation and Caption generation)

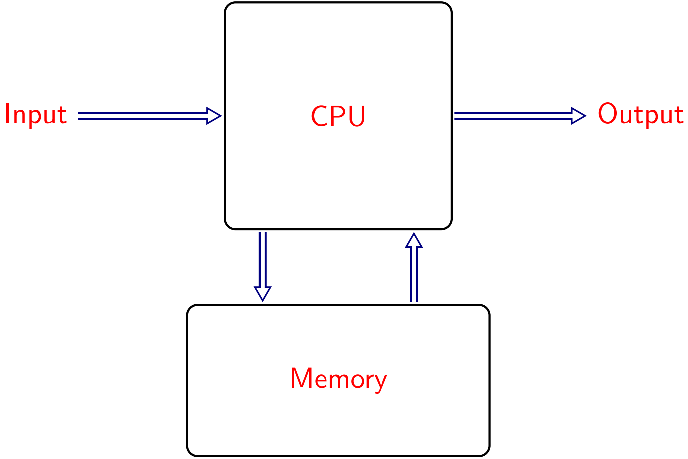
A Drawback

- A memory unit has pieces of information stored at different *locations*
- We want to be able to:
  - Access locations
  - Read/write information from/to these locations
- Attention mechanisms give us a way to point to specific chunks (we saw two examples - Machine Translation and Caption generation)
- Let us now see how can we use these intuition to construct networks with an explicit external memory

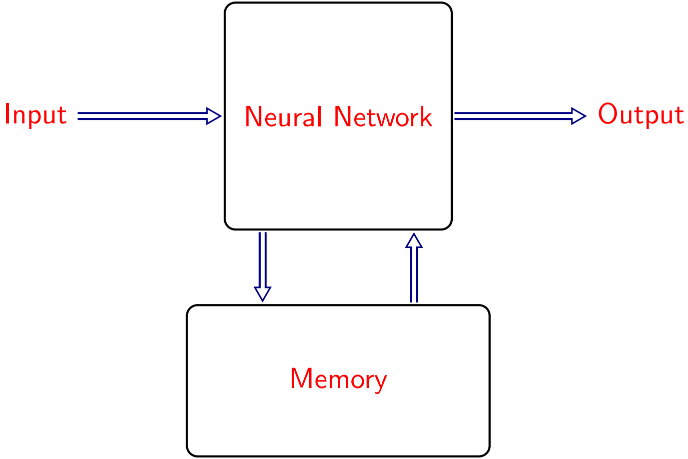
A Drawback

- A memory unit has pieces of information stored at different *locations*
- We want to be able to:
  - Access locations
  - Read/write information from/to these locations
- Attention mechanisms give us a way to point to specific chunks (we saw two examples - Machine Translation and Caption generation)
- Let us now see how can we use these intuition to construct networks with an explicit external memory
- Slightly anachronistic: Will first look at Neural Turing Machines and then Memory Networks

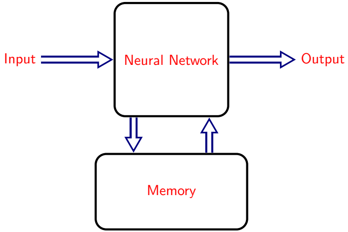
A Primitive Computer Model



A Neural Computer

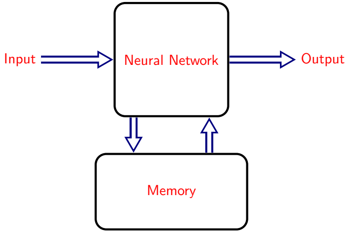


A Neural Computer



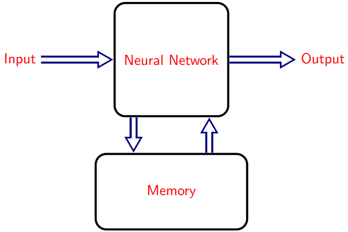
- **Goal:** Turn a Neural Network into a *Differentiable Computer*

A Neural Computer



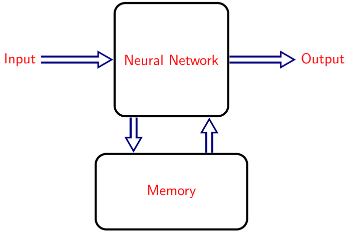
- **Goal:** Turn a Neural Network into a *Differentiable Computer*
- We roughly want the Neural Network to be the CPU

A Neural Computer



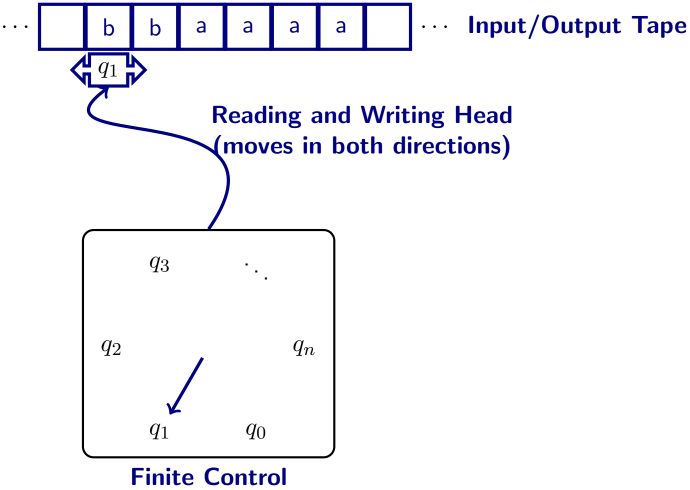
- **Goal:** Turn a Neural Network into a *Differentiable Computer*
- We roughly want the Neural Network to be the CPU
- Give NN read-write access to an external memory unit

A Neural Computer



- **Goal:** Turn a Neural Network into a *Differentiable Computer*
- We roughly want the Neural Network to be the CPU
- Give NN read-write access to an external memory unit
- Want the whole system to trainable by backpropagation

A Turing Machine



Informal Description

- A Turing Machine consists of:

Informal Description

- A Turing Machine consists of:
- A Tape:

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation
- A Head: Can read/write symbols on the tape and move left/right one cell at a time

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation
- A Head: Can read/write symbols on the tape and move left/right one cell at a time

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation
- A Head: Can read/write symbols on the tape and move left/right one cell at a time
- A State Register: Stores the state of the TM ("state of mind" that a person performing the computation)

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation
- A Head: Can read/write symbols on the tape and move left/right one cell at a time
- A State Register: Stores the state of the TM ("state of mind" that a person performing the computation)
- A Finite table of instructions: Given current state and the symbol it is reading on the tape tells the machine to either:
  - Either erase or write a symbole to a cell

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation
- A Head: Can read/write symbols on the tape and move left/right one cell at a time
- A State Register: Stores the state of the TM ("state of mind" that a person performing the computation)
- A Finite table of instructions: Given current state and the symbol it is reading on the tape tells the machine to either:
  - Either erase or write a symbole to a cell
  - Move the head L or R

Informal Description

- A Turing Machine consists of:
- A Tape:
  - Consists of cells next to each other
  - A cell has a symbol from some finite alphabet (which has a special blank symbol)
  - Is unbounded: The TM is always given as much tape as needed for its computation
- A Head: Can read/write symbols on the tape and move left/right one cell at a time
- A State Register: Stores the state of the TM ("state of mind" that a person performing the computation)
- A Finite table of instructions: Given current state and the symbol it is reading on the tape tells the machine to either:
  - Either erase or write a symbole to a cell
  - Move the head L or R
  - Then assume the same or a new state

Informal Description

- A Turing Machine consists of:

Informal Description

- A Turing Machine consists of:
- A State Register: Stores the state of the TM ("state of mind" that a person performing the computation)

Informal Description

- A Turing Machine consists of:
- A State Register: Stores the state of the TM (”state of mind” that a person performing the computation)
- A Finite table of instructions: Given current state and the symbol it is reading on the tape tells the machine to either:
  - Either erase or write a symbole to a cell

Informal Description

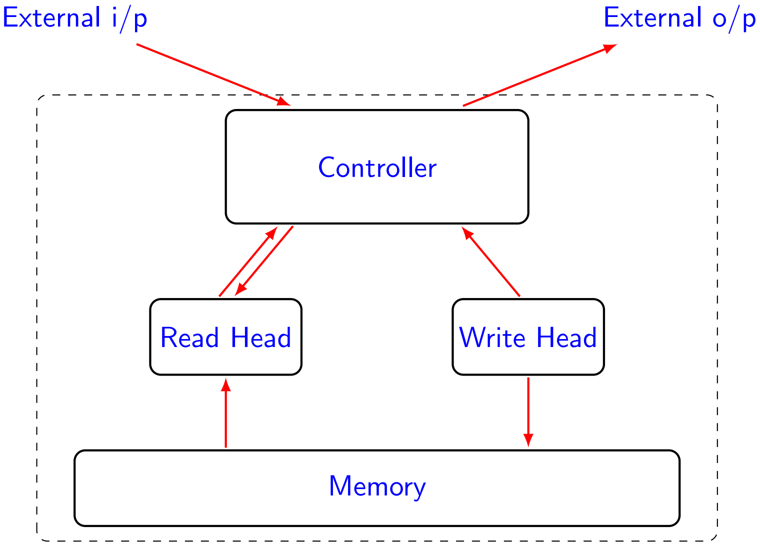
- A Turing Machine consists of:
- A State Register: Stores the state of the TM (''state of mind'' that a person performing the computation)
- A Finite table of instructions: Given current state and the symbol it is reading on the tape tells the machine to either:
  - Either erase or write a symbole to a cell
  - Move the head L or R

Informal Description

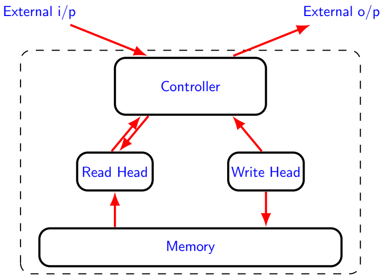
- A Turing Machine consists of:
- A State Register: Stores the state of the TM (”state of mind” that a person performing the computation)
- A Finite table of instructions: Given current state and the symbol it is reading on the tape tells the machine to either:
  - Either erase or write a symbole to a cell
  - Move the head L or R
  - Then assume the same or a new state

Goal: Want to mimic the working of a Turing Machine in a differentiable manner

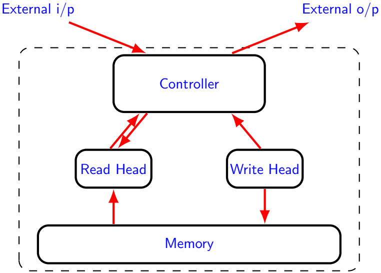
Architecture



Architecture

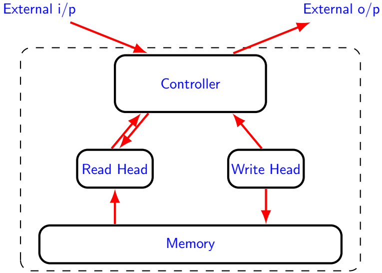


Architecture



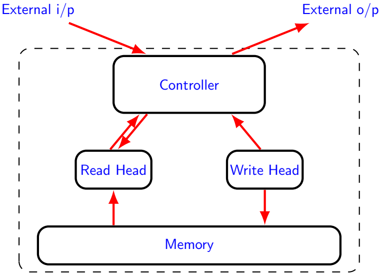
- Controller: Is a RNN or a CNN

Architecture

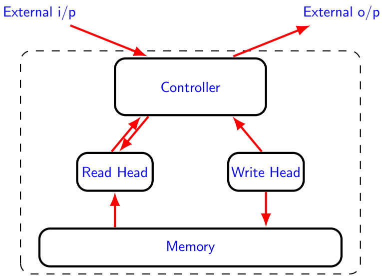


- Controller: Is a RNN or a CNN
- Receives input vectors and outputs vectors just as a normal neural network

Architecture

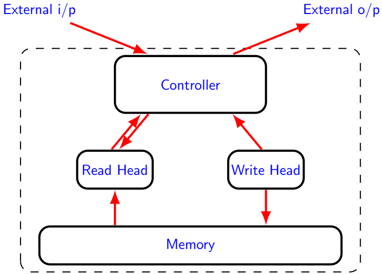


Architecture



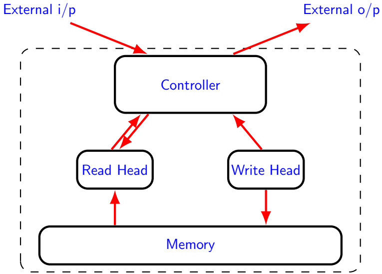
- The controller is connected to a real valued memory matrix which it can read/write to

Architecture



- The controller is connected to a real valued memory matrix which it can read/write to
- Controller interacts with this memory matrix with attentional processes that try to mimic the notion of heads in a TM

Architecture



- The controller is connected to a real valued memory matrix which it can read/write to
- Controller interacts with this memory matrix with attentional processes that try to mimic the notion of heads in a TM
- Main Idea: Keep everything differentiable

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts
- We do this by an attentional model

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts
- We do this by an attentional model
- The controller will output a weight vector – a distribution over the rows of the memory matrix

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts
- We do this by an attentional model
- The controller will output a weight vector – a distribution over the rows of the memory matrix
- We do this in two ways:

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts
- We do this by an attentional model
- The controller will output a weight vector – a distribution over the rows of the memory matrix
- We do this in two ways:
    - Based on location

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts
- We do this by an attentional model
- The controller will output a weight vector – a distribution over the rows of the memory matrix
- We do this in two ways:
  - Based on location
  - Based on content

Read/Write

- As hinted, we don't want to read/write to the whole memory, but want to focus on selective parts
- We do this by an attentional model
- The controller will output a weight vector – a distribution over the rows of the memory matrix
- We do this in two ways:
  - Based on location
  - Based on content
- Let's see both

Addressing by Content

- The controller emits a **key** $\mathbf{k}$ which is compared to the content of each memory location $M[i]$

Addressing by Content

- The controller emits a **key** $\mathbf{k}$ which is compared to the content of each memory location $M[i]$
- This comparison is done using some similarity measure $S[\cdot, \cdot]$ (e.g. cosine similarity)

Addressing by Content

- The controller emits a **key** $\mathbf{k}$ which is compared to the content of each memory location $M[i]$
- This comparison is done using some similarity measure $S[\cdot, \cdot]$ (e.g. cosine similarity)
- The similarities are then normalized using softmax (should remind of earlier lecture)

Addressing by Content

- The controller emits a **key** $\mathbf{k}$ which is compared to the content of each memory location $M[i]$
- This comparison is done using some similarity measure $S[\cdot, \cdot]$ (e.g. cosine similarity)
- The similarities are then normalized using softmax (should remind of earlier lecture)
- Additionally: We define a parameter $\beta \geq 0$ that controls the sharpness of focus

Addressing by Content

- The controller emits a **key** $\mathbf{k}$ which is compared to the content of each memory location $M[i]$
- This comparison is done using some similarity measure $S[\cdot,\cdot]$ (e.g. cosine similarity)
- The similarities are then normalized using softmax (should remind of earlier lecture)
- Additionally: We define a parameter $\beta \geq 0$ that controls the sharpness of focus

$$\mathbf{w}[i] = \frac{\exp(\beta S(\mathbf{k}, M[i]))}{\sum_j \exp(\beta S(\mathbf{k}, M[j]))}$$

- Find memories close to the key

Addressing by Location

- Content based addressing is a form of associative lookup: only cares about what the vectors are and not where they are

Addressing by Location

- Content based addressing is a form of associative lookup: only cares about what the vectors are and not where they are
- Sometimes we want to not care about the contents, but only care about the actual location

Addressing by Location

- Content based addressing is a form of associative lookup: only cares about what the vectors are and not where they are
- Sometimes we want to not care about the contents, but only care about the actual location
- Idea: Controller outputs a shift kernel $\mathbf{s}$ (usually softmax on numbers between $+1$ and -1)

Addressing by Location

- Content based addressing is a form of associative lookup: only cares about what the vectors are and not where they are
- Sometimes we want to not care about the contents, but only care about the actual location
- Idea: Controller outputs a shift kernel $\mathbf{s}$ (usually softmax on numbers between $+1$ and -1)
- This is convolved with a weighing $\mathbf{w}$ to produce a shifted weighing $\tilde{\mathbf{w}}$

Addressing by Location

- Content based addressing is a form of associative lookup: only cares about what the vectors are and not where they are
- Sometimes we want to not care about the contents, but only care about the actual location
- Idea: Controller outputs a shift kernel $\mathbf{s}$ (usually softmax on numbers between $+1$ and -1)
- This is convolved with a weighing $\mathbf{w}$ to produce a shifted weighing $\tilde{\mathbf{w}}$

$$\tilde{\mathbf{w}} = \sum_j \mathbf{w}[j]\mathbf{s}(i-j)$$

Addressing by Location

- Content based addressing is a form of associative lookup: only cares about what the vectors are and not where they are
- Sometimes we want to not care about the contents, but only care about the actual location
- Idea: Controller outputs a shift kernel $\mathbf{s}$ (usually softmax on numbers between $+1$ and -1)
- This is convolved with a weighing $\mathbf{w}$ to produce a shifted weighing $\tilde{\mathbf{w}}$

$$\tilde{\mathbf{w}} = \sum_j \mathbf{w}[j]\mathbf{s}(i-j)$$

- Gives a way to use a weighing already generated and push it up or down

Motivation

- Why do we need these different addressing mechanisms?

Motivation

- Why do we need these different addressing mechanisms?
- Idea is to mimic various data structures and accessors in programming languages

Motivation

- Why do we need these different addressing mechanisms?
- Idea is to mimic various data structures and accessors in programming languages
- **Content key only:** Associative map
- **Content and Location:** $\mathbf{k}$ finds an array in memory, shift indexes in it
- **Location:** Only iterates from the last focus

Reading from Memory

- We've defined weighings: How do we read from the memory?

Reading from Memory

- We've defined weighings: How do we read from the memory?
- Reading is simple: The read head gives a read vector $\mathbf{r}$ to the controller

Reading from Memory

- We've defined weighings: How do we read from the memory?
- Reading is simple: The read head gives a read vector $\mathbf{r}$ to the controller

$$\mathbf{r} = \sum_i \mathbf{w}[i] M[i]$$

Writing to Memory

- Slightly more complicated

Writing to Memory

- Slightly more complicated
- Decompose it into an erase and an add

Writing to Memory

- Slightly more complicated
- Decompose it into an erase and an add
- The controller will generate an erase vector $\mathbf{e}$ and an add vector $\mathbf{a}$ (both between 0 and 1) and sends it to the write vector

Writing to Memory

- Slightly more complicated
- Decompose it into an erase and an add
- The controller will generate an erase vector $\mathbf{e}$ and an add vector $\mathbf{a}$ (both between 0 and 1) and sends it to the write vector
- The write head then resets and writes to the memory

Writing to Memory

- Slightly more complicated
- Decompose it into an erase and an add
- The controller will generate an erase vector $\mathbf{e}$ and an add vector $\mathbf{a}$ (both between 0 and 1) and sends it to the write vector
- The write head then resets and writes to the memory

$$M[i] \leftarrow M[i](1 - \mathbf{w}[i] \odot \mathbf{e}) + \mathbf{w}[i] \odot \mathbf{a}$$

Motivation

- Basic motivation: How to turn Neural Networks into differentiable computers?

Motivation

- Basic motivation: How to turn Neural Networks into differentiable computers?
- Roughly: Want our NN to operate as a CPU that can read/write to an external memory

Motivation

- Basic motivation: How to turn Neural Networks into differentiable computers?
- Roughly: Want our NN to operate as a CPU that can read/write to an external memory
- Idea: The two together should be able to learn to program from input and output examples using backpropagation

Motivation

- Basic motivation: How to turn Neural Networks into differentiable computers?
- Roughly: Want our NN to operate as a CPU that can read/write to an external memory
- Idea: The two together should be able to learn to program from input and output examples using backpropagation
- Separate computation from memory

Task 1: Repeat Copy

- Task: Read a vector and then reproduce the whole vector

Task 1: Repeat Copy

- Task: Read a vector and then reproduce the whole vector
- Implements a simple algorithm. Time goes from left to right. Left column shows the write weighings, right columns shows the read weightings

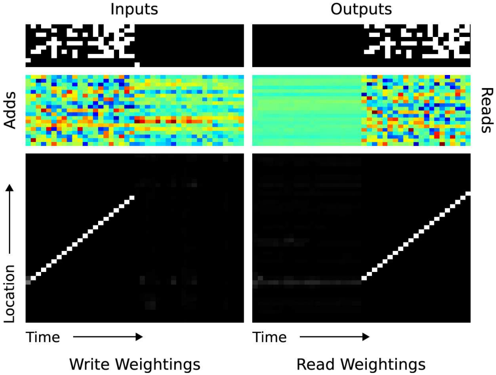


Figure: Neural Turing Machine, Graves *et al.*

Task 1: Repeat Copy

- Interesting part: Trained on sequences of length 10, it generalizes to sequence of length 120

Task 1: Repeat Copy

- Interesting part: Trained on sequences of length 10, it generalizes to sequence of length 120

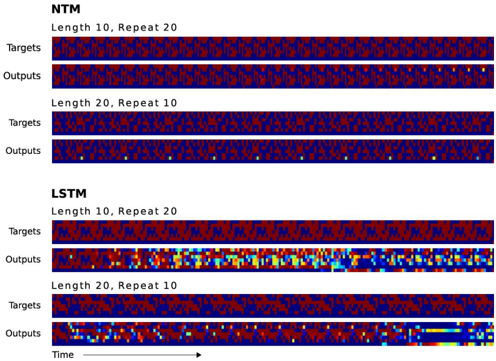


Figure: Neural Turing Machine, Graves *et al.*

Task 2: Copy N times

- For loop: Give it a sequence and number, reproduce it N times

Task 2: Copy N times

- For loop: Give it a sequence and number, reproduce it N times

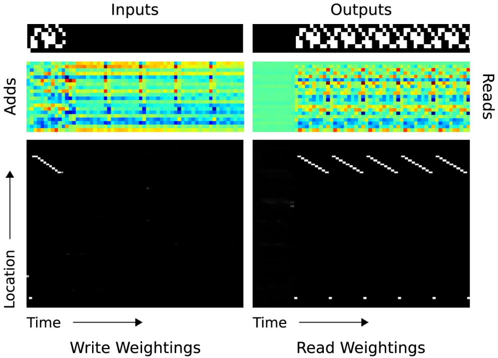
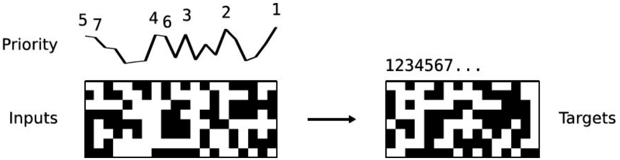


Figure: Neural Turing Machine, Graves *et al.*

Task 3: Priority Sort



Task 3: Priority Sort

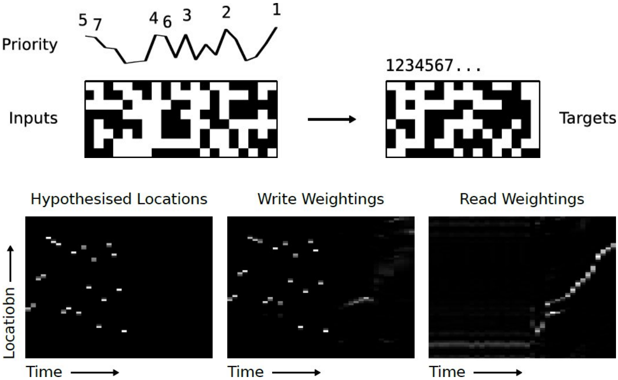
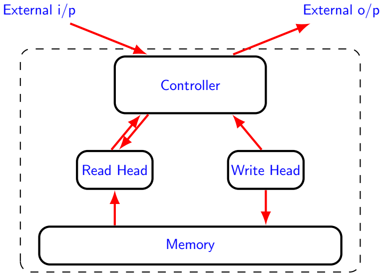


Figure: Neural Turing Machine, Graves *et al.*

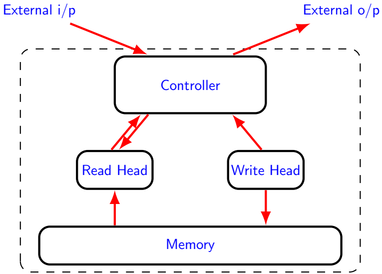
NTM v 2.0

Differentiable Neural Computers

Architecture

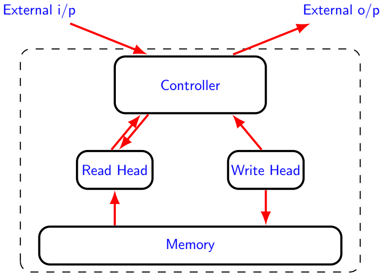


Architecture



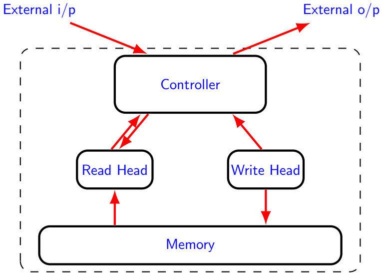
- Remember: The whole architecture is recurrent even if the controller is not recurrent.

Architecture

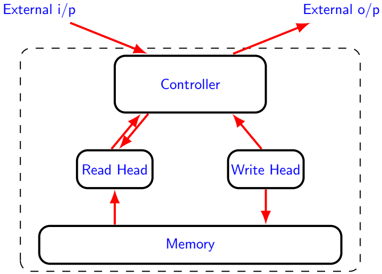


- Remember: The whole architecture is recurrent even if the controller is not recurrent.
- Let us see how far can we push this paradigm of modifying these real numbers i.e. the memory

Architecture

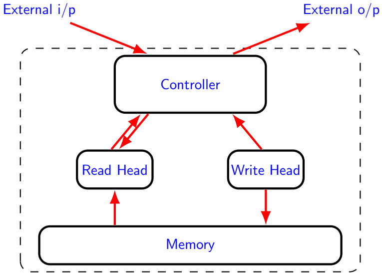


Architecture



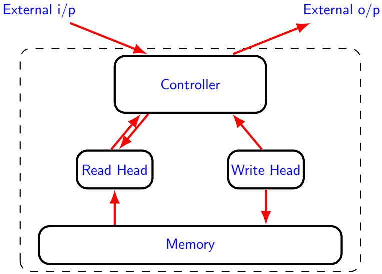
- We now have three attentional processes:
  - Based on content

Architecture



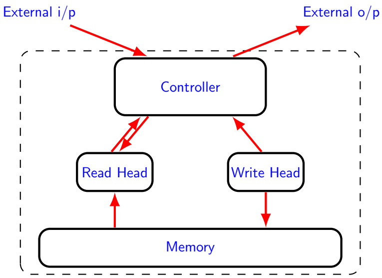
- We now have three attentional processes:
  - Based on content
  - Based on memory allocation

Architecture



- We now have three attentional processes:
  - Based on content
  - Based on memory allocation
  - Based on temporal order

Architecture



- We now have three attentional processes:
  - Based on content
  - Based on memory allocation
  - Based on temporal order
- Based on content, memory allocation and temporal order: The controller will interpolate between these by using scalar gates

Allocating Memory

- NTM could only *allocate* memory in contiguous blocks

Allocating Memory

- NTM could only *allocate* memory in contiguous blocks
- Leads to memory management issues (blocks start to overlap)

Allocating Memory

- NTM could only *allocate* memory in contiguous blocks
- Leads to memory management issues (blocks start to overlap)
- We can define a differentiable free list that book-keeps the memory usage of each location $\mathbf{u}_t$

Allocating Memory

- NTM could only *allocate* memory in contiguous blocks
- Leads to memory management issues (blocks start to overlap)
- We can define a differentiable free list that book-keeps the memory usage of each location $\mathbf{u}_t$
- Usage is increased after a write $\mathbf{w}_t^w$, maybe decreased after each read $\mathbf{w}_t^{r,i}$ by using a free gate f_t^i

Allocating Memory

- NTM could only *allocate* memory in contiguous blocks
- Leads to memory management issues (blocks start to overlap)
- We can define a differentiable free list that book-keeps the memory usage of each location $\mathbf{u}_t$
- Usage is increased after a write $\mathbf{w}_t^w$, maybe decreased after each read $\mathbf{w}_t^{r,i}$ by using a free gate f_t^i

$$\mathbf{u}_t = (\mathbf{u}_{t-1} + \mathbf{w}_{t-1}^w - \mathbf{u}_{t-1} \odot \mathbf{w}_{t-1}^w) \odot \prod_{i=1}^{R} (\mathbf{1} - f_t^i \mathbf{w}_{t-1}^{r,i})$$

Allocating Memory: Test

- Gave a bunch of random sequences and asked the system to reproduce them without resetting the memory:

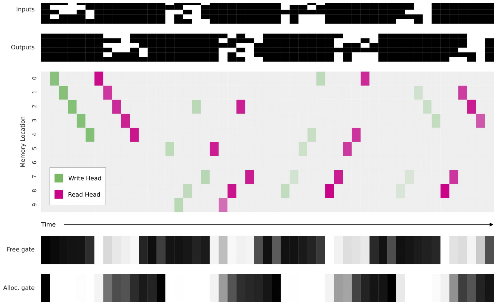


Figure: Hybrid Computing using a Neural Network with Dynamic External Memory, Graves *et al.*

Temporal Attention

- A Neural Turing Machine was able to search by content and index location but not by the order in which memories were written

Temporal Attention

- A Neural Turing Machine was able to search by content and index location but not by the order in which memories were written
- This is essential for certain tasks in which a sequence of sub-tasks have to be remembered in a certain order

Temporal Attention

- A Neural Turing Machine was able to search by content and index location but not by the order in which memories were written
- This is essential for certain tasks in which a sequence of sub-tasks have to be remembered in a certain order
- We can move iterate over the memory in order that they were written by making use of a precedence weighting

Temporal Attention

- A Neural Turing Machine was able to search by content and index location but not by the order in which memories were written
- This is essential for certain tasks in which a sequence of sub-tasks have to be remembered in a certain order
- We can move iterate over the memory in order that they were written by making use of a precedence weighting

$$\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

Temporal Attention

$$\mathbf{p}_t = \Big(1 - \sum_i \mathbf{w}_t^w[i]\Big)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

Temporal Attention

$$\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

- $\mathbf{p}_t$ updates a matrix L_t called the Temporal Link Matrix

Temporal Attention

$$\mathbf{p}_t = \Big(1 - \sum_i \mathbf{w}_t^w[i]\Big)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

- $\mathbf{p}_t$ updates a matrix L_t called the Temporal Link Matrix

$$L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$$

Temporal Attention

$$\mathbf{p}_t = \Big(1 - \sum_i \mathbf{w}_t^w[i]\Big)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

- $\mathbf{p}_t$ updates a matrix L_t called the Temporal Link Matrix

$$L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$$

- The controller can use L_t to retrieve the row that was written immediately before or after the last read

Temporal Attention

$$\mathbf{p}_t = \Big(1 - \sum_i \mathbf{w}_t^w[i]\Big)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

- $\mathbf{p}_t$ updates a matrix L_t called the Temporal Link Matrix

$$L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$$

- The controller can use L_t to retrieve the row that was written immediately before or after the last read
- This allows the controller to iterate in time

Temporal Attention

$$\mathbf{p}_t = \left(1 - \sum_i \mathbf{w}_t^w[i]\right)\mathbf{p}_{t-1} + \mathbf{w}_t^w$$

- $\mathbf{p}_t$ updates a matrix L_t called the Temporal Link Matrix

$$L_t[i,j] = (1 - \mathbf{w}_t^w[i] - \mathbf{w}_t^w[j])L_{t-1}[i,j] + \mathbf{w}_t^w[i]\mathbf{p}_{t-1}[j]$$

- The controller can use L_t to retrieve the row that was written immediately before or after the last read
- This allows the controller to iterate in time
- Three way gates are used to interpolate between the forward and backward iterations as well as content

Architecture

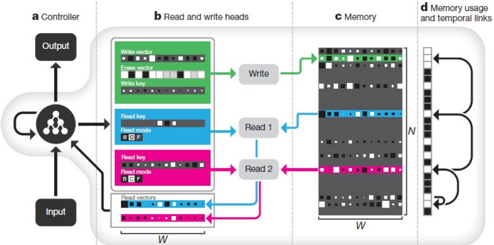
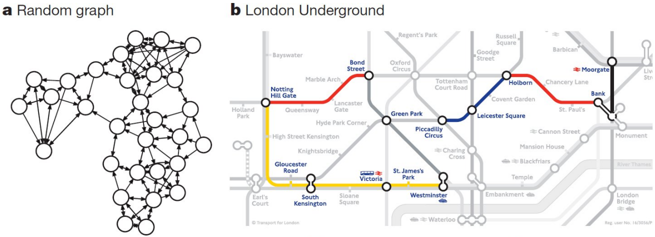


Figure: Hybrid Computing using a Neural Network with Dynamic External Memory, Graves *et al.*

Shortest Paths



Underground input:	Traversal	Shortest-path
(OxfordCircus, TottenhamCtRd, Central) (TottenhamCtRd, OxfordCircus, Central) (BakerSt, Marylebone, Circle) (BakerSt, Marylebone, Bakerloo) (BakerSt, OxfordCircus, Bakerloo) ⋮ (LeicesterSq, CharingCross, Northern) (TottenhamCtRd, LeicesterSq, Northern) (OxfordCircus, PiccadillyCircus, Bakerloo) (OxfordCircus, NottingHillGate, Central) (OxfordCircus, Euston, Victoria) 84 edges in total	Traversal question: (BondSt, _, Central), (_, _, Circle), (_, _, Circle), (_, _, Circle), (_, _, Circle), (_, _, Jubilee), (_, _, Jubilee), Answer: (BondSt, NottingHillGate, Central) (NottingHillGate, GloucesterRd, Circle) ⋮ (Westminster, GreenPark, Jubilee) (GreenPark, BondSt, Jubilee)	Shortest-path question: (Moorgate, PiccadillyCircus, _) Answer: (Moorgate, Bank, Northern) (Bank, Holborn, Central) (Holborn, LeicesterSq, Piccadilly) (LeicesterSq, PiccadillyCircus, Piccadilly)

Figure: Hybrid Computing using a Neural Network with Dynamic External Memory, Graves *et al.*

Shortest Paths

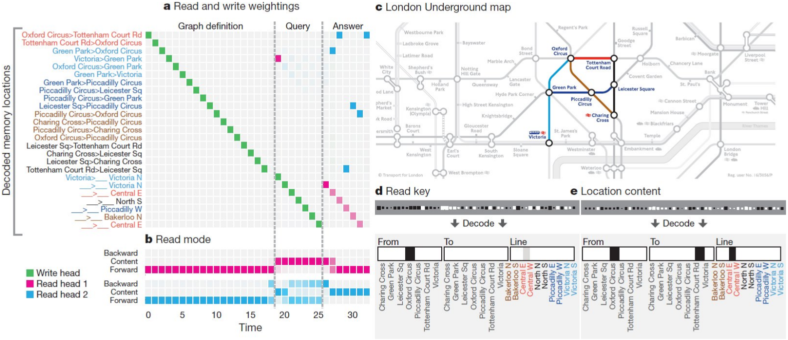


Figure: Hybrid Computing using a Neural Network with Dynamic External Memory, Graves *et al.*

Family Tree

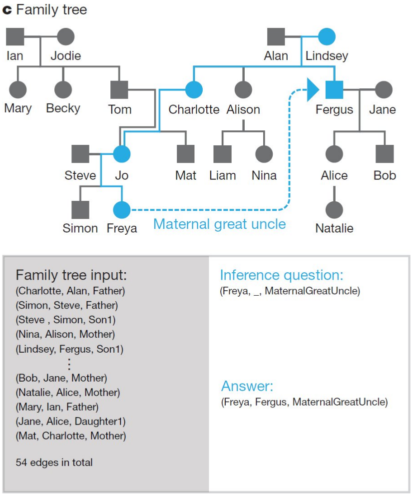


Figure: Hybrid Computing using a Neural Network with Dynamic External Memory, Graves *et al.*
https://www.youtube.com/watch?v=B9U8sI7TcMY

Question Answering (bAbi)

Sample:

Sheep are afraid of wolves
Cats are afraid of dogs
Mice are afraid of cats
Gertrude is a sheep
Question: What is Gertrude afraid of?

Question Answering (bAbi)

Sample:

Sheep are afraid of wolves
Cats are afraid of dogs
Mice are afraid of cats
Gertrude is a sheep
Question: What is Gertrude afraid of?

Question Answering (bAbi)

Sample:

Sheep are afraid of wolves

Cats are afraid of dogs

Mice are afraid of cats

Gertrude is a sheep

Question: What is Gertrude afraid of?

Question Answering (bAbi)

Sample:

Sheep are afraid of wolves
Cats are afraid of dogs
Mice are afraid of cats
Gertrude is a sheep
Question: What is Gertrude afraid of?
Answer: Wolves

Motivation

- Sentences are accessed out of order

Motivation

- Sentences are accessed out of order
- There can be many sentences in between: long term dependencies

Memory Networks (End to End)

- Neural Network model with an external memory

Memory Networks (End to End)

- Neural Network model with an external memory
- Soft attention mechanisms are used to read from memory

Memory Networks (End to End)

- Neural Network model with an external memory
- Soft attention mechanisms are used to read from memory
- Depending on the taak, we can do multiple hops on the memory

Memory Networks (End to End)

- Neural Network model with an external memory
- Soft attention mechanisms are used to read from memory
- Depending on the taak, we can do multiple hops on the memory
- The goal again is to keep the system differentiable end-to-end