Can Active Memory Replace Attention?

Lukasz Kaiser & Samy Bengio

Google Brain

NIPS 2016
Presenter: Chao Jiang

Outline

1. Introduction
2. Active Memory
3. Step by Step to Neural GPU
4. Another two steps: 1. the Markovian Neural GPU
5. Another two steps: 2. The Extended Neural GPU
6. Experiments
   - Machine Translation

Background

- Several mechanism to focus attention of a neural network on selected parts of its input or memory have been used successfully
- Similar improvement have been obtained using **Active Memory** - do not focus on a single part of a memory, but operate on all of it in parallel, in a uniform way
- However, active memory has **not improved** over attention for **NLP** tasks, especially **machine translation**

What does this paper do?

- Analyzing the **shortcoming** of previous active memory model
- Proposed **an extended model of active memory** whose performance matches existing attention model and generalizes better
- Comparing active memory and attention

What is Active Memory?

- Broadly, referring to any model where **every part of the memory** undergoes active **change** at every step
- In contrast to attention models, where **only a small part of the memory change** at every step
- **Exact implementation vary** from model to model

Active Memory in this paper

- In this paper, we rely on the **convolution operator**
- Given a memory tensor s, an active memory model will produce the next memory s' by using a number of convolutions on s and combining them

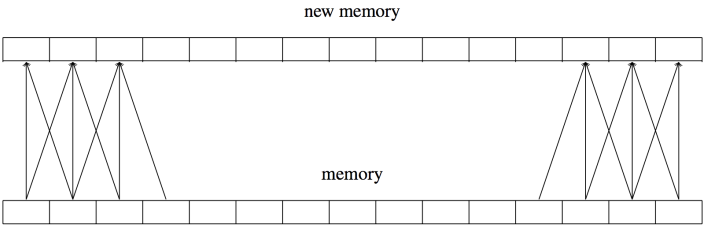


Figure 2: Active memory model. The whole memory takes part in the computation at every step. Each element of memory is active and changes in a uniform way, e.g., using a convolution.

Active Memory in this paper

- The convolution acts on a kernel bank U and a 3-dimensional tensor s
- U shape: $[K_w, k_h, m, m]$, s shape: $[w, h, m]$, output shape: $[w, h, m]$

$$U * s[x, y, i] = \sum_{u=\lfloor -k_w/2 \rfloor}^{\lfloor k_w/2 \rfloor} \sum_{v=\lfloor -k_h/2 \rfloor}^{\lfloor k_h/2 \rfloor} \sum_{c=1}^{m} s[x+u, y+v, c] \cdot U[u, v, c, i] \quad (1)$$

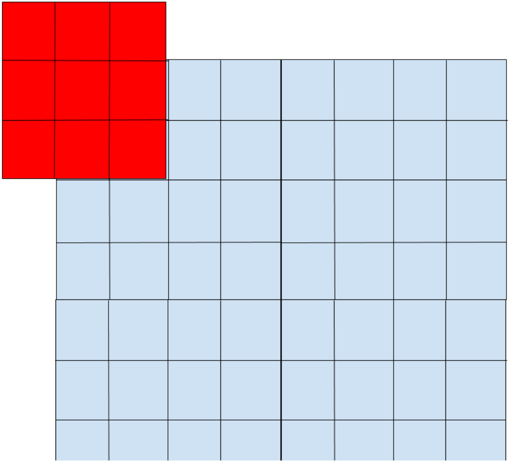
- The index $x + u$ might sometimes be negative or larger than the size of size of s, we assume the value is 0

Dive deeper: how to understand each dimension?

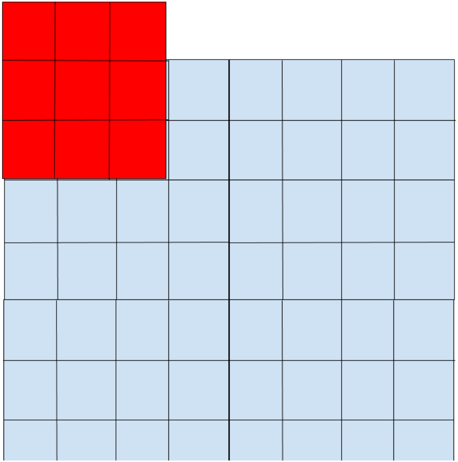
$$U * s[x, y, i] = \sum_{u=\lfloor -k_w/2 \rfloor}^{\lfloor k_w/2 \rfloor} \sum_{v=\lfloor -k_h/2 \rfloor}^{\lfloor k_h/2 \rfloor} \sum_{c=1}^{m} s[x+u, y+v, c] \cdot U[u, v, c, i] \quad (2)$$

- U shape: $[K_w, k_h, m, m]$, s shape: $[w, h, m]$, output shape: $[w, h, \underline{m}]$
- Convolution kernel U: $\underline{m}$ fliters, each of them dimension is $[K_w, k_h, m]$
- s shape: $[\underline{w, h}, m]$, output shape: $[\underline{w, h}, m]$
- The center of the fliter (each fliter is 3-dimensional) slice over the input s (3-dimensional), when exceeds boundary, assume number is 0
- So, output shape $=$ s shape $= [w, h, m]$

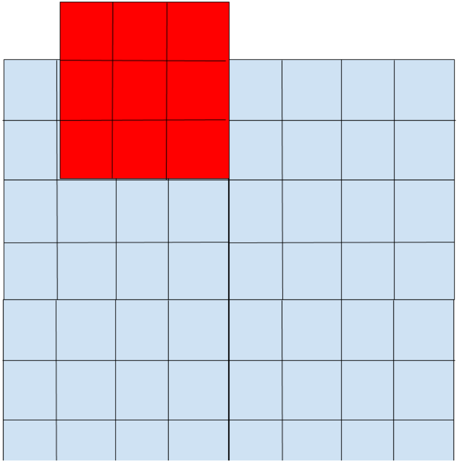
Convolution



Convolution



Convolution



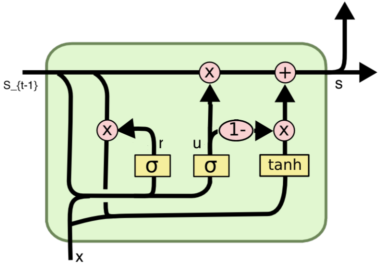
Active Memory recap

- The convolution acts on a kernel bank U and a 3-dimensional tensor s
- U shape: $[K_w, k_h, m, m]$, s shape: $[w, h, m]$, output shape: $[w, h, m]$

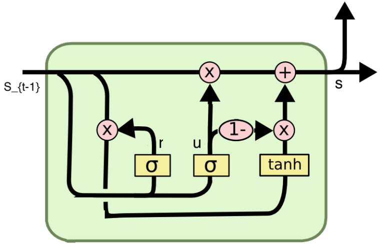
$$U * s[x, y, i] = \sum_{u=\lfloor -k_w/2 \rfloor}^{\lfloor k_w/2 \rfloor} \sum_{v=\lfloor -k_h/2 \rfloor}^{\lfloor k_h/2 \rfloor} \sum_{c=1}^{m} s[x+u, y+v, c] \cdot U[u, v, c, i] \quad (3)$$

- The index $x + u$ might sometimes be negative or larger than the size of size of s, we assume the value is 0

From GRU to Convolutional Gated Recurrent Unit (CGRU)

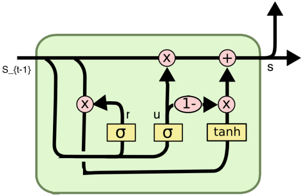


$$\mathrm{GRU}(x, s) \;=\; u \odot s + (1-u) \odot \tanh(Wx + U(r \odot s) + B), \quad \text{where}$$
$$u = \sigma(W'x + U's + B') \quad \text{and} \quad r = \sigma(W''x + U''s + B'').$$



$$\mathrm{CGRU}(s) \;=\; u \odot s + (1-u) \odot \tanh(U * (r \odot s) + B), \quad \text{where}$$
$$u = \sigma(U' * s + B') \quad \text{and} \quad r = \sigma(U'' * s + B'').$$

Convolutional Gated Recurrent Unit (CGRU)



$$\mathrm{CGRU}(s) = u \odot s + (1-u) \odot \tanh(U * (r \odot s) + B), \quad \text{where}$$
$$u = \sigma(U' * s + B') \quad \text{and} \quad r = \sigma(U'' * s + B'').$$

$$\mathrm{CGRU}(s) = u \odot s + (1-u) \odot \tanh(U * (r \odot s) + B), \quad \text{where}$$
$$u = \sigma(U' * s + B') \quad \text{and} \quad r = \sigma(U'' * s + B'').$$

- u, s, r, B shape: $[w, n, m]$, U shape: $[K_w, k_h, m, m]$
- We do not process a new input in every step, all the input are written into the start state s_0
- $U * s$ denotes the convolution of a kernel bank U with s

Neural GPU overview

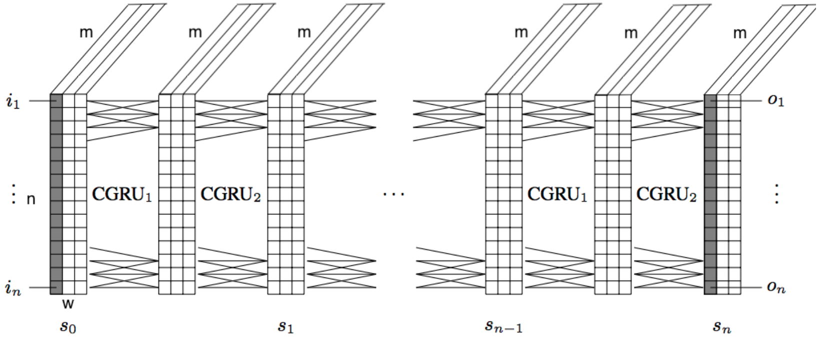
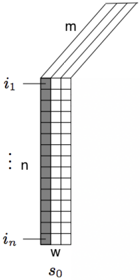


Figure 3: Neural GPU with 2 layers and width $w = 3$ unfolded in time.

Neural GPU input

- The given sequence $i = (i_1, \cdots, i_n)$ of n discrete symbols from $\{0, \cdots, I\}$ is first embedding into the tensor s_0 by concatenating the vectors obtained from an embedding lookup of the input symbols into its first column
- s_0 shape: $[w, n, m]$, an embedding matrix E shape: $[I, m]$
- $s_0[0, k, :] = E[i_k]$ for all $k = 1 \cdots n$ (here $i_1, \cdots, i_n$ is the input)
- All other elements of s_0 are set to 0



Neural GPU transition

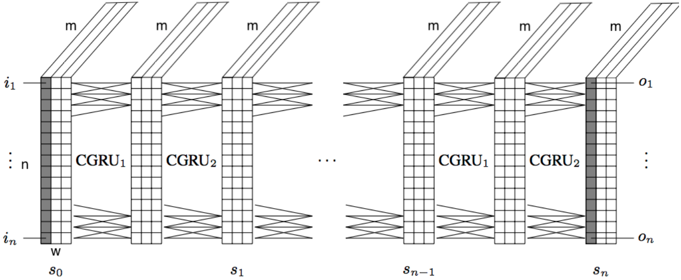


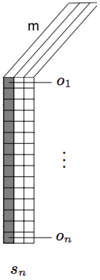
Figure 3: Neural GPU with 2 layers and width $w = 3$ unfolded in time.

- Apply l different CGRU gates in turn for n steps to produce the final tensor s_{fin}

$$s_{t+1} = \mathrm{CGRU}_l(\mathrm{CGRU}_{l-1} \ldots \mathrm{CGRU}_1(s_t) \ldots) \quad \text{and} \quad s_{\text{fin}} = s_n.$$

Neural GPU output

- Result is obtained by multiplying each item in the first column of s_{fin} by an output matrix O to obtain the logits $l_k = Os_{fin}[0, k, :]$
- Output matrix O shape: $[I, m]$ (I is vocabulary size)
- $s_{fin}[0, k, :]$ shape: $[m, 1]$, l_k shape: $[I, 1]$
- And then, select then last one: $o_k = argmax(l_k)$
- During training, they use standard loss function, i.e., compute a softmax over the logits l_k and use the negative log probability of the target as the loss.



Neural GPU recap

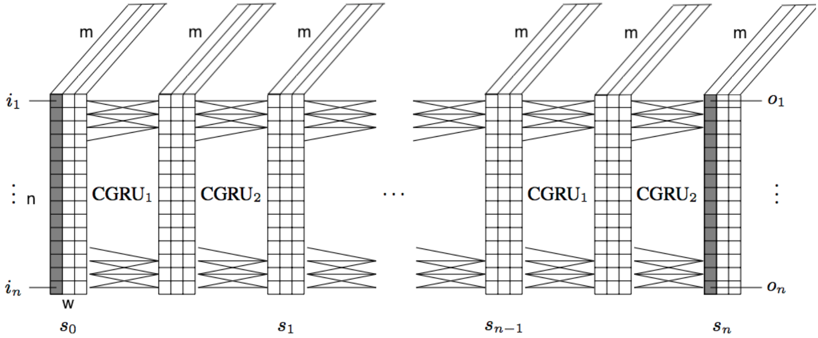


Figure 3: Neural GPU with 2 layers and width $w = 3$ unfolded in time.

How to improve?

- Baseline Neural GPU model performance is very poor
- The main reason is the output generator
- Every output symbol is generated independently of all other outputs symbols, conditionally only on the state s_{fin}
- Introducing the Markov hypothesis: every output symbol depends on the previous output

The Markovian Neural GPU

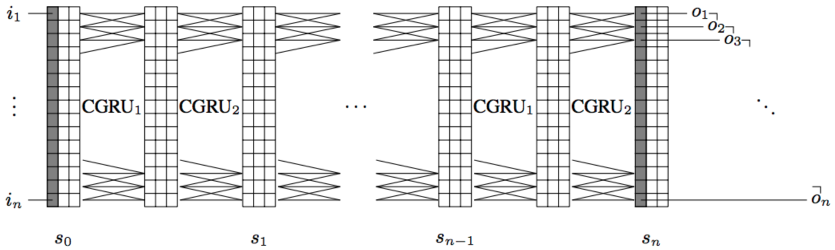


Figure 4: Markovian Neural GPU. Each output o_k is conditionally dependent on the final tensor $s_{\text{fin}} = s_n$ and the previous output symbol o_{k-1}.

$$l_k = O \operatorname{concat}(s_{\text{fin}}[0, k, :], E' o_{k-1})$$

The Markovian Neural GPU

$$l_k = O\,\mathrm{concat}(s_{\mathrm{fin}}[0, k, :], E' o_{k-1})$$

- Concatenate each item from the first column of s_{fin} with the embedding of the previous output generated by another embedding matrix E'
- For $k = 0$, use special symbol $o_{k-1} = GO$
- $o_k = argmax(l_k)$

How to improve?

- Markovian Neural GPU yields much better results on neural machine translation, but still far from those achived by models with attention
- Markovian dependence is too week, a full recurrent dependence of the state is needed
- Extending baseline model with an active memory decoder

The Extended Neural GPU overview

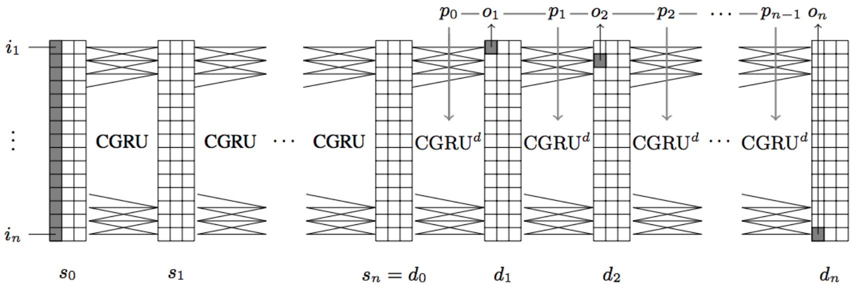


Figure 5: Extended Neural GPU with active memory decoder. See the text below for definition.

- Same as baseline model until $s_{fin} = s_n$
- s_n is the start point for the active memory decoder, i.e., $d_o = s_n$
- In the active memory decoder, use a separate **output tape tensor p** (same shape as d_0, $[w, n, m]$)

The Extended Neural GPU: active memory decoder

- Start p_0 set to all 0, and define the decoder state by:

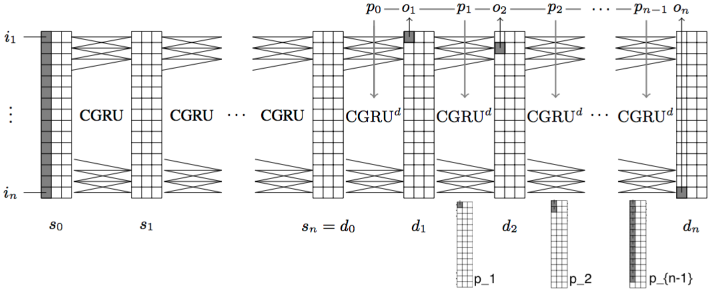
$$d_{t+1} = \mathrm{CGRU}_l^d(\mathrm{CGRU}_{l-1}^d(\ldots \mathrm{CGRU}_1^d(d_t, p_t)\ldots, p_t), p_t),$$

- $CGRU^d$ is defined just like $CGRU$, but difference is highlighted

$$\mathrm{CGRU}^d(s,p) \;=\; u \odot s + (1-u) \odot \tanh(U * (r \odot s) + \boldsymbol{W} * \boldsymbol{p} + B), \;\text{ where}$$
$$u = \sigma(U' * s + \boldsymbol{W'} * \boldsymbol{p} + B') \quad \text{and} \quad r = \sigma(U'' * s + \boldsymbol{W''} * \boldsymbol{p} + B'').$$

- W shape: $[K_w, k_h, m, m]$, p shape: $[w, n, m]$
- $l_k = Od_k[0, k, :]$, $o_k = argmax(l_k)$

Go to detail about p



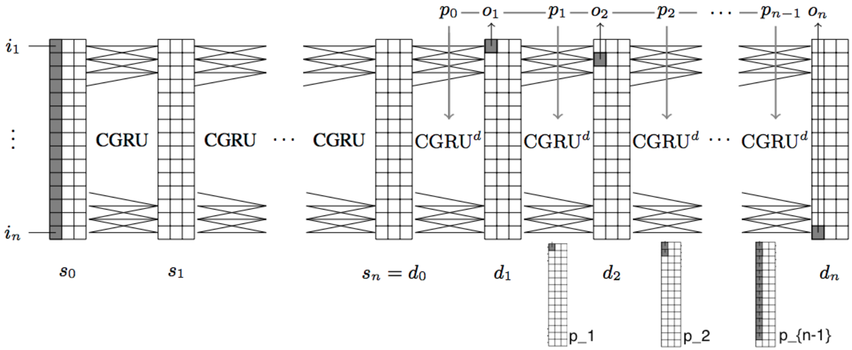
- Generate the k-th output using $l_k = Od_k[0, k, :]$, $o_k = argmax(l_k)$
- Symbol o_k is then embedded back in to a dense representation using another embedding matrix E'
- Put it into the k-th place on the output tape p

Neural GPU output

- In this way, we accumulated outputs step-by-step on the output tape p

$$p_{k+1} = p_k \quad \textbf{with} \quad p_k[0, k, :] \leftarrow E' o_k.$$

Neural GPU recap



Outline

Experiment framework

- All components in out model are differentiable, can be trained using any stochastic gradient descent optimizer
- number of layers $l = 2$, the width of the state tensors $w = 4$, number of maps $m = 512$, the convolution kernels $k_w = k_h = 3$
- WMT' 14 English-French translation task
- Baseline model: GRU with attention model

Result

Model	Perplexity (log)	BLEU
`Neural GPU`	30.1 (3.5)	< 5
`Markovian Neural GPU`	11.8 (2.5)	< 5
`Extended Neural GPU`	3.3 (1.19)	**29.6**
`GRU+Attention`	3.4 (1.22)	26.4

Table 1: Results on the WMT English->French translation task. We provide the average per-word perplexity (and its logarithm in parenthesis) and the BLEU score. Perplexity is computed on the test set with the ground truth provided, so it do not depend on the decoder.

- An active model can indeed match an attention model on the machine translation task

Performance on sentences of different length

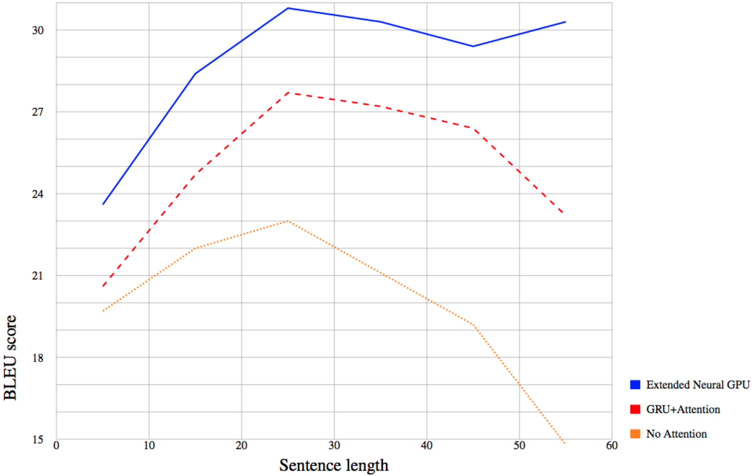


Figure 6: BLEU score (the higher the better) vs source sentence length.

How to decide sentence length?

- In previous paper (same author using neural GPU to learn algorithm), they just use padding symbol on input ot make it match the output length

Input	0	0	1	1				
Output	0	0	1	1	0	0	1	1

- In this paper, they test all sizes between input size and double the input size and report the best one