Neural Combinatorial Optimization Using Neural Turing Machine With Input Pointers

Amey Agrawal
2014A7PS148P

Birla Institute of Technology and Science, Pilani

Final Report submitted in the fulfillment of the course
CS F441 Selected Topics In Computer Science

May 1, 2017

Abstract

Amey Agrawal
Undergraduate Student,
Department of Computer Science
BITS Pilani
2017

Neural Turning Machines (NTMs) are powerful learning model which can exploit external memory to solve complex problems. But NTMs can not process inputs wth variable size dictionary which is essential to solving combinatorial optimisation problems. This project presents a method to overcome this issue.

We describe a new learning model which lets enables processing of inputs with variable size dictionary. And then discuss the possible applications such an algorithm and its implications.

Acknowledgements

TBA

Contents

Chapter 1

Introduction

Neural Turing Machine (NTM) architecture provides neural networks ability to store data over long times periods using dynamic external memory. While drawing a parallel with a conventional computer we can say, the external memory matrix is similar to random-access memory (RAM) and the internal memory of controller Long Short-Term Memory (LSTM) the network is analogous to registers in a central processing unit (CPU). Recent studies show the ability of NTMs to perform structured tasks on complex data structures.

The number of target classes in the output depends on the input in a large class of combinatorial optimisation problems such as sorting variable-sized sequences and Travelling Salesman problem (TSP). Pointer Networks (Ptr-Nets) architecture extends LSTMs to address the problem using neural attention mechanism. Ptr-Nets have demonstrated their ability to solve and generalise combinatorial optimisation problems.

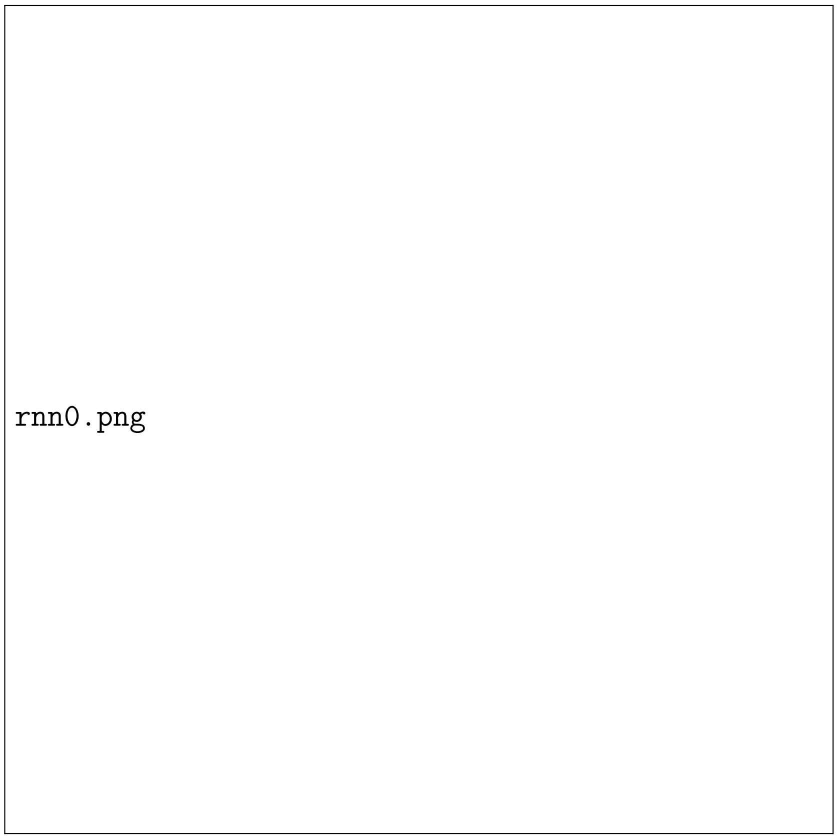
To exploit the power of Neural Turning Machines in solving combinatorial optimisation problem we describe a method inspired by Pointer Networks. This work is further discussed in chapter chapter 4.

Chapter 2

Background

2.1 Reccurent Neural Networks

Recurrent Neural Networks (RNN) adds state memory to the multi-layer perceptron. Output at any point of time depends on both inputs to the network and current state. The ability of RNNs to perform context-aware computations significantly enriched the capabilities of neural networks. [**?**] proved that the finite sized RNN with sigmoidal activation is Turing complete.



Credit: *Christopher Olah, 2015*

Figure 2.1: An unrolled recurrent neural network.

An typical RNN is charecturized mathamatically as follows,

$$h_t = tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$y_t = W_{hy}h_t$$

[**?**] showed that vanilla RNNs face problem of exploding and vanishing gradient problem which makes them hard to train. Vanilla RNNs are hardly ever used in practice.

2.2 Long Short Term Memory (LSTM)

The landmark publication of [?] introduced Long Short-Term Memory (LSTM) which overcomes the vanishing gradient problem. LSTM provides a mechanism to selectively forget the information from the previous state and hence process long sequences of data without any hurdle.

LSTMs have been used for solving challenging problems like speech recognition graves2014, text generation sutskever2011 and protein secondary structure prediction sonderby2014protein.

lstm0.png

Credit: *Christopher Olah, 2015*

Figure 2.2: The repeating module in an LSTM containing four interacting layers.

The above figure is mathamtically denoted as,

$$a = W_x x_t + W_h h_{t-1} + b$$

$$\begin{bmatrix} i \\ f \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma(a_1 : N) \\ \sigma(N : 2N) \\ \sigma(2N : 3N) \\ tanh(3N : 4N) \end{bmatrix}$$

$$c_t = f \cdot c_{t-1} + i \cdot g$$

$$h_t = o \cdot tanh(c_t)$$

2.3 Sequance to sequence learning

Sequence-to-sequence learning model introduced by [?] uses one recurrent a neural network to encode the input and another one to generate an output sequence from the embedding. Sequence to sequence learning enabled researchers to achieve state-of-the-art in many challenging problems like machine translation, interactive responder systems and automated story-telling.

seq2seq0.png

Credit: *Suriyadeepan Ram, 2016*

Figure 2.3: Sequence to sequence learning architecture.

For a training sequance (P, C^P) where, $P = \{P_1, P_2, ..., P_n\}$ is a sequance of n vectors and $C^P = \{C_1, C_2, ..., C_{m(P)}\}$ we can denote sequance to sequance learning model as,

$$p(C^P|P;\theta) = \prod_{i=1}^{m(P)} p_\theta(C_i|C_1, C_2, ...C_{i-1}, P;\theta)$$

$$\theta^* = argmax \sum_{P,C^P} \log p(C^P|P;\theta)$$

The output of the decoder is obtained by applying a softmax over the word encoding space in a typical text-to-text model. This step requires the dictionary size to decided during the training itself.

2.4 Pointer Networks

sec:ptr [**?**] proposed a simple yet elegant solution to the problem of variable size output dictionaries using neural attention. Instead of finding conditional probability over a fixed size output dictionary, in pointer networks, we learn conditional probability of discrete tokens corresponding to positions in the input sequence. That is, it points to a token in input sequence while generating the output sequence.

`ptr0.png`

Credit: *Oriol Vinyals, 2015*

Figure 2.4: Pointer Networks.

Updating the equation from 2.3 we write,

$$u^i_j = v^T tanh(W_1 e_j + W_2 d_i) \quad j \in (1, 2, ..., n)$$

$$p(C_i | C_1, C_2, ...C_{i-1}, P) = softmax(u^i)$$

Where $(u_1, u_2, ..., u_n)$ and $(v_1, v_2, ..., v_{m(P)})$ are encoder and decoder hidden states respectively. While v, W_1 and W_2 are learnable parameters.

`ptr1.png`

Adapted from: *Stephen Merity, 2016*

Figure 2.5: Comparison of pointer network and RNN.

Now we can have sequences with arbitrary output dictionary sizes given that all output sequence tokens are in the input sequence.

2.5 Neural Turing Machines

ntm introduced Neural Turing Machines. The state memory of an RNN is small and hard to update selectively, hence though they are Turing complete in theory, they are unsuitable for solving a wide range of algorithmic problems. Selective portions of NTM's external memory can be addressed for reading and write operations using neural attention. Network parameters which enable this are called "heads". The controller is responsible for processing the input sequence, generating read and write heads and producing the output.

ntm0.png

Credit: *Christopher Olah, 2016*

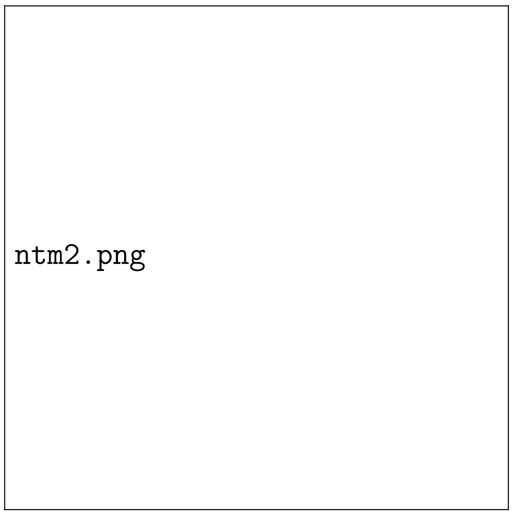
Figure 2.6: Neural Turing Machine.

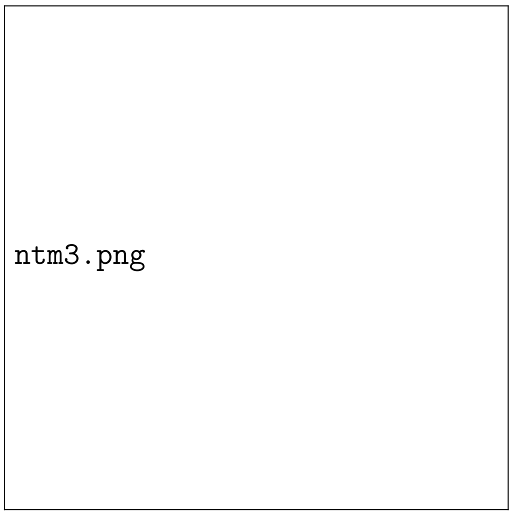
The architecture of controller network is the most crucial hyper-parameter of an NTM. Compared to feed-forward neural networks LSTMs have faster convergence and better test performance. The read and write operations on memory are formilated as,

$$r_t = \sum_i w_t(i) M_t(i)$$

$$M'_t(i) = M_{t-1}(i)[1 - w_t(i)e_t],$$
$$M_t(i) = M'_t(i) + w_t(i)a_t$$

Where M_t is the memory and r_t is the read value at at timestep t. e and a are erase and add vectors respectively.





Credit: *Christopher Olah, 2016*

Figure 2.7: Memory Read and Write in Neural Turing Machine.

NTM has two addressing modes, random access and sequential access. Sequential access is useful when we want to read consecutive blocks of memory sequentially. The w_t vector is calculated using the following pipeline,

ntm5.png

Credit: *Alex Graves, 2015*

Figure 2.8: Calculating attention in NTM.

ntm6.png

Credit: *Tristan Deleu, 2015*

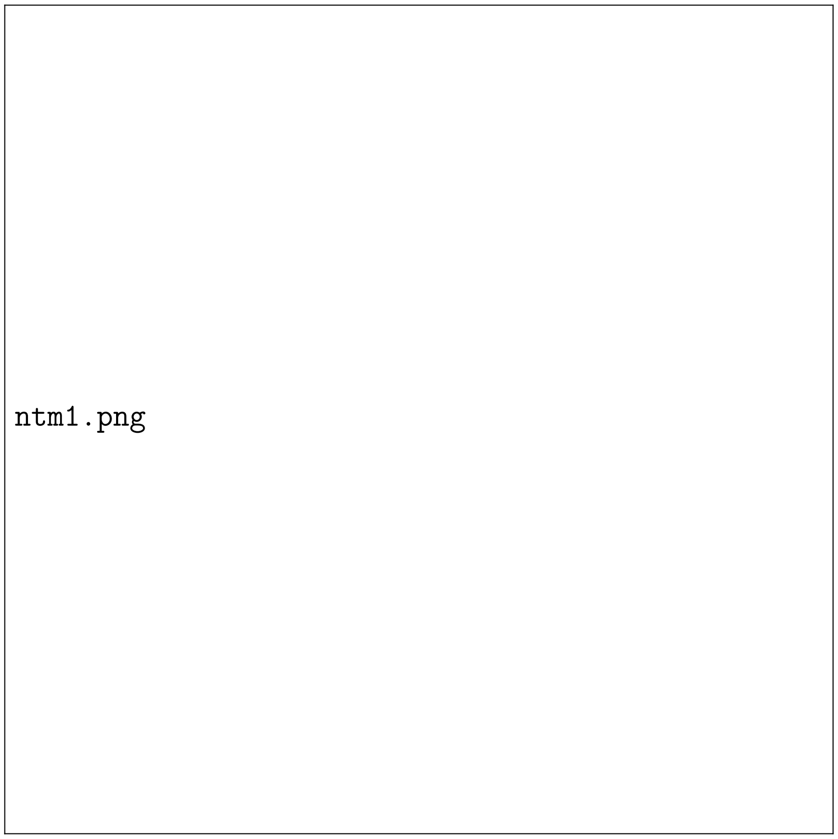
Figure 2.9: Visually understanding attention based addressing in NTM.

In case of LSTM controller, controller outputs are obtained from last output of LSTM as,

$$k_t = tanh(h_t W_k)$$
$$\beta_t = relu(h_t W_\beta)$$

$$g_t = sigmoid(h_t W_g)$$
$$s_t = softmax(h_t W_s)$$
$$\gamma_t = relu(h_t W_\gamma) + 1$$

While performing a task an NTM can be visualised simply as slightly modified a version of a typical sequence to sequence learning.



Credit: *Tristan Deleu, 2015*

Figure 2.10: Nural Turing Machine Unrolled in Time.

Further improvements have been made to the attention mechanism by dnc which lead to the system that can self-organise complex data structures like graphs and trees to perform algorithmic operations. dntm decoupled address and content vectors to learn a wide variety of addressing strategies. ngpu

presented a new architecture called Neural GPU which is a highly parallel variant of NTM and hence easier and faster to train.
Also NTM still requires the size of output directory to be fixed beforehand, which we resolve with our new architecture in 4.

Chapter 3

Combinatorial Optimization Problems

3.1 Introduction

Combinatorial optimisation is a subset of mathematical optimisation and has applications in applied mathematics and software engineering. Combinatorial optimisation is consists of finding an optimal permutation of items from a finite set. Some common combinatorial optimisation problems are the travelling salesman problem ("TSP") , the convex hull problem, the knapsack problem and the minimum spanning tree problem ("MST").

3.2 Complexity Class

Apart from some special cases namely shortest paths and shortest path trees, flows and circulations, spanning trees, matching, and matroid problems most Combinatorial optimisation problems are NP-Complete.
Approximation algorithm exists for many of the popular problems which yield near optimal solution. But there is not a single framework which can be used to solve a general combinatorial optimisation problem.

3.3 Neural Combinatorial Optimization

[?] tried to create a generalised framework to solve neural combinatorial optimisation using pointer networks. They introduced a new algorithm to train the network pointer networks using policy gradient. A new search strategy named The active search was used in the experiments as well.
They benchmarked their results for TSP and Knapsack problems and surpassed the the heuristics on the standard testbench.

Chapter 4

Neural Turing Machines With Input Pointers

4.1 Motivation

Neural Turing machines have been utilised to solve complex problems involving complex operations on input sequences like question answering and repeated copy. [**?**] used differential neural computers to solve graph and tree relation problems where the network learned to self-organize the memory in appropriate data structure. Such behaviour would be highly desirable while solving combinatorial optimisation problems. Hence, we enhance the standard NTM architecture to have input pointers just like pointer networks.

4.2 Algorithum

Typically NTMs output a vector in the same hyperspace as that of input by performing a linear transform on the output of LSTM controller.
The very same equations in **??** can be applied directly to the output of LSTM h_t to obtain conditional probability over the input tokens.

4.3 Standard Testbench

sec:test

Copy

The task is to replicate the input exactly in output. Copy task can be considered as a trivial case of combinatorial optimisation problems and it has been used as a standard benchmark in NTM literature. Random vectors with Bernoulli distribution has been typically used as the testbench for copy task.

Repeat Copy

Repeat copy is a modified version of copy task where the input is replicated n times in the output. The input data for both repeat copy and copy task is same and generated similarly.

Travelling Salesman Problem

Travelling salesman problem has been used to analyse algorithms in literature extensively. TSP is benchmarked against standard TSP20 and TSP50 datasets.

4.4 Implementation

Keras

An implementation of algorithm has been developed based upon SigmaQuan's implementation in Karas. It uses theano backend with keras 1.2.0.

Pytorch

As both the Keras and Theano versions using in SigmaQuan's implementation are out-dated another version was developed based upon bzcheeseman's implementation in Pytorch. Pytroch supports dynamic graphs unlike tensorflow and thenao hence allowing for easier modelling.

Chapter 5

Conclusion and Furthur Direction

A new architecture has been developed to solve complex combinatorial optimisation problems based upon results and experiments in literature. Due to unavailability of enough computation power to train network the architecture could not be validated against standard testbench.
In future whenever GPU are available the architecture has to be benchmarked against the tasks described in **??**. Furthur, the reinforcement learning algorithm proposed in [**?**] can be implemented. Also, the possibilities for extending the architecture to Neural GPU can be considered.

Bibliography