Model-based Reinforcement Learning in Computer Systems

Sean J. Parker
Clare Hall

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: sjp240@cam.ac.uk

April 8, 2021

Declaration

I, Sean J. Parker of Clare Hall, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 0

Signed:

Date:

Acknowledgements

Abstract

Write a summary of the whole thing. Make sure it fits in one page.

Contents

1 Introduction **vii**

2 Background and Related Work **1**

2.1 Introduction to Deep Learning Models 1

2.1.1 Current approaches to optimising deep learning models 2

2.2 Reinforcement Learning 3

2.2.1 Model-Free and Model-Based RL 4

2.2.2 World Models 6

2.3 Graph Neural Networks 6

3 Title **7**

4 Title **9**

5 Title **11**

List of Figures

2.1 Single perceptron as a dataflow (computation) graph 2

List of Tables

Chapter 1

Introduction

Chapter 2

Background and Related Work

2.1 Introduction to Deep Learning Models

This section discusses the way in which machine learning models are represented for efficient execution on physical hardware devices. First, we discuss how the mapping of tensor operations to computation graphs is performed followed by an overview of recent approaches that optimise computation graphs to minimise execution time.

Over the past decade, there has been a rapid development of various deep learning architectures that aim to solve a specific task. Common examples include convolutional networks (popularised by AlexNet then ResNets, etc), transformer networks that have seen use in the modelling and generation of language. Recurrent networks that have shown to excel at learning long and short trends in data.

Importantly, the fundamental building blocks of the networks have largely remained unchanged. As the networks become more complex, it becomes untenable to manually optimise the networks to reduce the execution time on hardware. Therefore, there is extensive work in ways to both automatically optimise the models, or, alternatively apply a set of hand-crafted optimisations.

Computation graphs are a way to graphically represent both the individual tensor operations in a model, and the connections (or data-flow) along the edges between nodes in the graph. Figure 2.1 shows how the expression, $y = \texttt{ReLU}(\mathbf{w} \cdot \mathbf{x} + b)$, can be represented graphically in a computation graph.

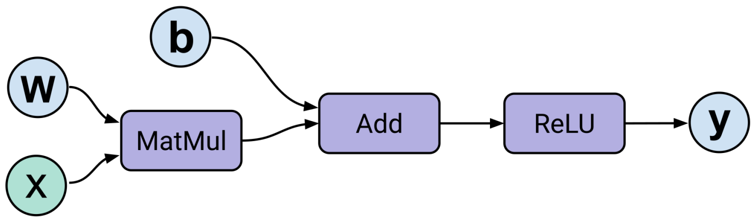

Figure 2.1: The operations shown in purple are the nodes of the computation graph which take an arbitraty number of inputs, performs a computation at the node and produces an output. The blue nodes represent the input nodes for tensors. The directed edges show the flow of tensors through the graph.

Similarly, the whole model can be converted into a stateful dataflow graph in this manner. By using a stateful dataflow (or computation) graph, we can use any optimisation technique for backpropagation of the model loss though the graph [TODO rewrite last sentence]. We consider two key benefits of this representation. First, we can execute the model on any hardware device as the models have a single, uniform representation. Secondly, it allows for pre-execution optimisations based on the host device, for example, we may perform different optimisations for executing on a GPU compared to a TPU.

2.1.1 Current approaches to optimising deep learning models

Due to the prevalence and importance of machine learning, especially deep networks, there is a focus on finding ways decrease the inference runtime and by extension, increasing the model throughput. All major frameworks such as TensorFlow [1], PyTorch [], MXNet, and Caffe have some level of support for performing pre-execution optimisations. However, the process of performing such optimisations is often time-consuming and cannot be completed in real-time. Rather, it is to use a deep learning optimisation library such as cuDNN [2].

TVM

Metaflow and TASO

2.2 Reinforcement Learning

Reinforcement learning (RL) is a sub-field in machine learning, broadly, it aims to compute a control policy such that an agent can maximise its cumulative reward from the environment. It has powerful applications in environments where a model that describes the semantics of the system are not available and the agent must itself discover the optimal strategy via a reward signal.

- Can also mention POMDPs

Formally, RL is a class of learning problem that can be framed as a Markov decision processes (MDP) when the MDP that describes the system is not known [3]; they are represented as a 5-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}_a, \mathcal{R}_a, \rho_0 \rangle$ where:

- $\mathcal{S}$, is a finite set of valid states
- $\mathcal{A}$, is a finite set of valid actions
- $\mathcal{P}_a$, is the transition probability function that an action a in state s_t leads to a state s'_{t+1}
- $\mathcal{R}_a$, is the reward function, it returns the reward from the environment after taking an action a between state s_t and s'_{t+1}
- ρ_0, is the starting state distribution

We aim to compute a policy, denoted by π, that when given a state $s \in \mathcal{S}$, returns an action $a \in \mathcal{A}$ with the optimisation objective being to find a control policy π^* that maximises the *expected reward* from the environment defined by 2.1. Importantly, we can control the 'far-sightedness' of the policy by tuning the discount factor $\gamma \in [0, 1)$. As γ tends to 1, the policy will consider the rewards further in the future but with a lower weight as the distant expected reward may be an imperfect prediction.

$$\pi^* = \arg\max_{\pi} \ \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \ \mathcal{R}_t\right] \tag{2.1}$$

Classic RL problems are formulated as MDPs in which we have a finite state space, however, such methods quickly become inefficient with large state spaces that we consider with applications such as Atari and Go. Therefore, we take advantage of modern deep learning function approximators, such as neural networks, that makes learning the solutions far more efficient in practise. We have seen many successfully applications in a wide range of fields, for example, robotic control tasks [4], datacenter power management, device placement, and, playing both perfect and imperfect information games to a super-human level. Reinforcement learning excels when applied to environments in which actions may have long-term, inter-connected dependencies that are difficult to learn or model with traditional machine learning techniques.

In the following sections we discuss the two key paradigms that exist in reinforcement learning and the current research in both areas and the application to systems tasks.

2.2.1 Model-Free and Model-Based RL

Model-free and model-based are the two main approaches to reinforcement learning, however, with recent work such as [5, 6, 7], the distinction between the two is becoming somewhat nebulous; it is possible to use a hybrid approach that aims to improve the sample efficiency of the agent by training model-free agents directly in the imagined environment.

The notable differences between the two approaches are summarised in Table

[TODO: x].

- Model-based is more sample efficient

- Model-free has had more applications with greater success, e.g. in Atari, Go

- In systems environments, it can be costly for the agent to interact with the environment (which is the opposite of Atari emulators)

- Approaches for overcoming the same inefficiency, by almost brute force, by using massive amounts of compute via distributed training

- Model-based RL is concerned with learning a model of the underlying environment which it learns to simulate its dynamics – thereby agents can be cheaply and quickly trained in this 'imagined/dreamed/hallucinogenic' environment (famous example is AlphaZero where it was given an explicit model of the environment) : two options, learn a model, or be given a model

- Model-free, either policy optimisation (PPO) or Q-learning (and lots in between)

2.2.2 World Models

2.3 Graph Neural Networks

Algorithm 1: Steps of computation in a full GN block

```
for $k \in \{1 \dots N^e\}$ do
    $\mathbf{e}'_k \leftarrow \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$
end
for $i \in \{1 \dots N^n\}$ do
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e \to v}(E'_i)$
    $\bar{\mathbf{v}}'_i \leftarrow \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u})$
end
let $V' = \mathbf{v}'_{i=1:N^v}$
let $E' = (\mathbf{e}'_k, r_k, s_k)_{k=1:n^e}$
$\bar{\mathbf{e}}' \leftarrow \rho^{e \to u}(E')$
$\bar{\mathbf{v}}' \leftarrow \rho^{v \to u}(V')$
$\bar{\mathbf{u}}' \leftarrow \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u})$
return $(E', V', \mathbf{u}')$
```

- Form of neural network that has seen lots of active research recently

- Can be viewed in similar way to CNNs

- Graph network, Edge block -¿ Node block -¿ global block

- We use it to process the graphs and produce a latent tensor (from the global block) which we can learn using

Chapter 3

Title

Chapter 4

Title

Chapter 5

Title

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning, 2014.

[3] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[4] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019.

[5] Harrison Brown, Kai Fricke, and Eiko Yoneki. World-models for bitrate streaming. *Applied Sciences*, 10(19), 2020.

[6] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model-based reinforcement learning for atari, 2020.

[7] Jan Robine, Tobias Uelwer, and Stefan Harmeling. Smaller world models for reinforcement learning, 2021.