University of Cape Town
Faculty of Science

Department of Mathematics and Applied Mathematics

Applied Mathematics Masters Thesis

submitted for the degree of

Master of Science

Evaluating Transformers as Memory Systems in Reinforcement Learning

by

Thomas Makkink
Student Number: MKKTHO001

Submission Date: September 7, 2021
Supervisors: Dr Jonathan Shock and Dr Arnu Pretorius

Declaration of Authorship

I, Thomas Makkink, declare that this thesis titled, Evaluating Transformers as Memory Systems in Reinforcement Learning, and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: September 7, 2021

Abstract

Memory is an important component of effective learning systems and is crucial in non-Markovian as well as partially observable environments. In recent years, Long Short-Term Memory (LSTM) networks have been the dominant mechanism for providing memory in reinforcement learning, however, the success of transformers in natural language processing tasks has highlighted a promising and viable alternative. Memory in reinforcement learning is particularly difficult as rewards are often sparse and distributed over many time steps. Early research into transformers as memory mechanisms for reinforcement learning indicated that the canonical model is not suitable, and that additional gated recurrent units and architectural modifications are necessary to stabilize these models. Several additional improvements to the canonical model have further extended its capabilities, such as increasing the attention span, dynamically selecting the number of per-symbol processing steps and accelerating convergence. It remains unclear, however, whether combining these improvements could provide meaningful performance gains overall. This dissertation examines several extensions to the canonical Transformer as memory mechanisms in reinforcement learning and empirically studies their combination, which we term the Integrated Transformer. Our findings support prior work that suggests gating variants of the Transformer architecture may outperform LSTMs as memory networks in reinforcement learning. However, our results indicate that while gated variants of the Transformer architecture may be able to model dependencies over a longer temporal horizon, these models do not necessarily outperform LSTMs when tasked with retaining increasing quantities of information.

Acknowledgments

I'd like to take this opportunities to thank all those who helped me along this journey:

- To my supervisors, Dr Jonathan Shock and Dr Arnu Pretorius, for the their excellent guidance and unfaltering encouragement. Thank you Dr Shock for always being available and for the moleskin notebook you gave me at the very beginning of this dissertation, it has been well used. Thank you Dr Pretorius for your wisdom, mentorship and showing me the magic of PyTorch. It was been an absolute pleasure working with you and I look forward to collaborating in the future.

- To Shahil Mawjee, a good friend I made along the way. Thank you for helping me run experiments, always being available to chat about the project and for your guidance on reinforcement learning topics.

- To Grisel Pretorius, thank you for your undying enthusiasm, generous spirit and for never failing to make me laugh. Your crash course on how to pull all nighters proved invaluable during the final weeks before hand in.

- To my mom Shelley, my dad Patrick, my sisters Josie and Rose and my brother Francis, thank you for providing all the support and love a son and brother could ever ask for. From the numerous plates of butter chicken, to the off-the-books counseling sessions, boxing in the garage, and swimming in the pool. This dissertation would not be possible without you.

- And last, but not least, to my Rottweiler Malkia, thank you for your ever-patient companionship whilst spending afternoons lying in the sunspot next to my desk.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The introduction of the Transformer in 2017 was a watershed moment for the field of Natural Language Processing (NLP), inspiring new research and innovation in the field. The Transformer network is a powerful sequence-to-sequence architecture that utilizes the attention function to model dependencies. Although the Transformer's most notable achievements to date predominately occurred in an NLP setting, research practitioners have successfully applied the Transformer to domains ranging from computer vision (Dosovitskiy et al., 2020; Carion et al., 2020) to biological sequencing (Cheng et al., 2020).

One such domain relevant to this dissertation is the role of memory in sequential decision making problems. The paradigm of machine learning concerned with these types of problems is reinforcement learning, whereby an agent learns to make intelligent decisions through trial-and-error interactions within an environment. Most classical reinforcement learning problems, such as the game of Go and Checkers, do not require memory as the current configuration of the board is sufficient to determine the optimal next move. In real world tasks, however, this assumption rarely holds. For example, consider the simple task of an agent foraging mushrooms in a naturalistic terrain. To succeed at this task the agent must be able to remember which areas of the map contain mushrooms, what time of the year they are ready to harvest and which areas have been foraged recently. Furthermore, memory is often critical in a multi-agent setting, whereby an agent may for example receive an important message or instructions from another agent and must have the ability to retain this information over time to make informed decisions in the future.

Without the ability to encode, retain and exploit information from past experience, the set of real world tasks a reinforcement agent may be useful for is significantly reduced. Consequently, improving the performance of memory systems in reinforcement learning is critical for broadening the types of tasks that these systems are useful for and their application in the real world.

In recent years, recurrent neural networks such as the Long Short-Term Memory (LSTM) network (Hochreiter and Schmidhuber, 1997) have been the most widely adopted mechanism for providing memory in reinforcement learning. It is well established that the performance of recurrent neural networks diminishes when modeling long range temporal dependencies (Bengio et al., 1994). Despite efforts to design more expressive memory architectures, such as read-write operations with an external memory database (Graves, 2017; Wayne et al., 2018), these models have not been widely adopted, perhaps due to their complex implementation. The repeated success of the Transformer across a wide range of sequential modeling tasks has, however, highlighted a promising and viable alternative.

The Transformer was introduced as the new state-of-the-art sequence model for machine translation in a paper titled "Attention is All You Need" (Vaswani et al., 2017). The Transformer not only outperformed the previous top models, which were predominately based on recurrent or convolutional neural networks, but also required far shorter training times to do so. Since then, Transformer-based models have dominated NLP literature, outperforming their predecessors on a diverse range of tasks such as language modeling (Dai et al., 2019; Yang et al., 2020), machine translation (Vaswani et al., 2017; Edunov et al., 2018), question answering (Yang et al., 2020; Dehghani et al., 2019), multi-task representation learning (Devlin et al., 2019; Yang et al., 2020) and summarization (Liu and Lapata, 2019). For the purpose of this dissertation, the Transformer model introduced by Vaswani et al. (2017) will be referred to as the *Canonical Transformer*, while the term *transformers* will refer to the general class of models that are based on the Transformer architecture.

As the title of the paper that introduced the Canonical Transformer alludes to, this Transformer relies exclusively on the attention function to model dependencies between the input and output sequences, dispensing with recurrent and convolutional layers entirely. The key advantage of the attention function over recurrent

units and convolutions is its ability to model dependencies irrespective of their distance in the sequences (Kim et al., 2017; Bahdanau et al., 2016), which allows the Transformer to scale to much longer-term dependencies. Additionally, the attention function is significantly more parallelizable than recurrent units, lending itself well to modern computing hardware.

The repeated success of the Transformer did not, however, translate to immediate success when utilized as a memory system in reinforcement learning. For example, Mishra et al. (2018) demonstrated that the Canonical Transformer was unable to solve even the simplest bandit tasks. Transformers are difficult to optimize in the supervised case as well, typically requiring a complex learning rate schedule[1] (Vaswani et al., 2017; Dai et al., 2019) or specialized weight initialization schemes (Radford et al., 2019) to train effectively. These measures seem insufficient for reinforcement learning where critical information may be sparsely distributed over the span of an entire episode.

Recent work has empirically demonstrated that the addition of Gated Recurrent Units to the Transformer submodule drastically improves and stabilizes performance in reinforcement learning (Parisotto et al., 2019). This architectural variant of the Transformer, the Gated Transformer-XL, outperformed an LSTM baseline on the challenging DMLab-30 benchmark suite (Beattie et al., 2016). The performance of the Gated Transformer-XL was a powerful demonstration of the potential of Transformer-based architectures in reinforcement learning and made a strong case for their wider adoption.

There have been various other improvements to the Canonical Transformer in recent years which remain unevaluated in a reinforcement learning setting. For example, ReZero improved signal propagation in the canonical model and increased convergence speed (Bachlechner et al., 2020). The Universal Transformer incorporated the inductive bias of recurrent neural networks by applying an in-depth recurrent transition function (Dehghani et al., 2019). Furthermore, the Universal Transformer added a dynamic halting mechanism based on adaptive computation time (Graves, 2017), which allows the model to dynamically select the number of processing steps required for each symbol in the sequence. Given that the Gated Transformer-XL,

[1] e.g. linear warmup or cosine decay.

ReZero and the Universal Transformer improve independent aspects of the Canonical Transformer and build upon a shared framework, it is plausible that they could be combined into a single Transformer model.

There are, however, several key challenges that stand before the mainstream adoption of transformers in reinforcement learning. Firstly, there is no current consensus on how best to adapt these models to a reinforcement learning setting. Unfortunately, the work published by Parisotto et al. (2019) lacked a clear explanation of how the Gated Transformer-XL was integrated as a memory system with the underlying reinforcement learning algorithm. Moreover, the official implementation of this Transformer is not publicly available, which means the experiments conducted in the study cannot be easily replicated nor is it straightforward to apply the model to a different reinforcement learning task. Secondly, research on Transformer-based architectures has been predominately focused on solving NLP tasks, and it is unclear whether many of the most recent innovations in the field will translate to meaningful performance gains in reinforcement learning. An additional consequence of this trend is that the majority of open-source Transformer libraries, such as the excellent HuggingFace API, were designed specifically for NLP tasks and cannot easily be ported to reinforcement learning. Thirdly, further experimentation is required to evaluate the scaling properties of the Transformer on specific aspects of memory. For example, memory length and size, whereby memory length refers to the number of sequential time steps an agent can retain information for and memory size is the quantity of information an agent can retain.

1.1 Research Questions

The primary aim of this dissertation was to investigate the use of transformers as memory systems in reinforcement learning. To address this research aim, the following research questions were declared:

1. Which Transformer-based architectures, if any, are able to outperform an LSTM baseline as a memory system in reinforcement learning in terms of memory length and memory size?

2. Does integrating the various extensions to the Transformer architecture lead

to an overall performance increase?

1.2 Research Hypotheses

The following research hypotheses are declared:

1. The attention function is expected to be insufficient for modeling temporal dependencies in reinforcement learning.

2. The addition of Gated Recurrent Units to the Transformer submodule is expected to stabilize and improve the performance of transformers as memory systems in reinforcement learning.

Chapter 2

Background and Related Work

This chapter investigates several mechanisms that may be useful for providing memory in Reinforcement Learning systems, including the current state-of-the-art Gated Transformer-XL model and the well-established Long Short-Term Memory network. We begin with a brief overview of Reinforcement Learning itself (section 2.1), which is frequently formulated under the mathematical framework of Markov decision processes (MDPs). MDPs represent an idealized form of the reinforcement learning problem and are often insufficient for describing many real world tasks. For these types of tasks, reinforcement learning systems frequently require specialized networks to encode, retain and retrieve information over time. Section 2.2 investigates two types of sequence models as candidates for this task: recurrent neural networks and Transformer-based architectures. Specifically, we consider which elements of each model have empirically been demonstrated to be important for reinforcement learning. Section 2.3 highlights reinforcement learning benchmarks and past integrated approaches relevant to this dissertation.

2.1 Reinforcement Learning

Reinforcement learning is a paradigm of machine learning that is concerned with learning through trial-and-error interaction within an environment. As illustrated by figure 2.1.1, at each time step a decision-making agent receives an observation from its environment and performs an action which influences the environment state. From this interaction, the agent receives feedback via a reward signal and, by way

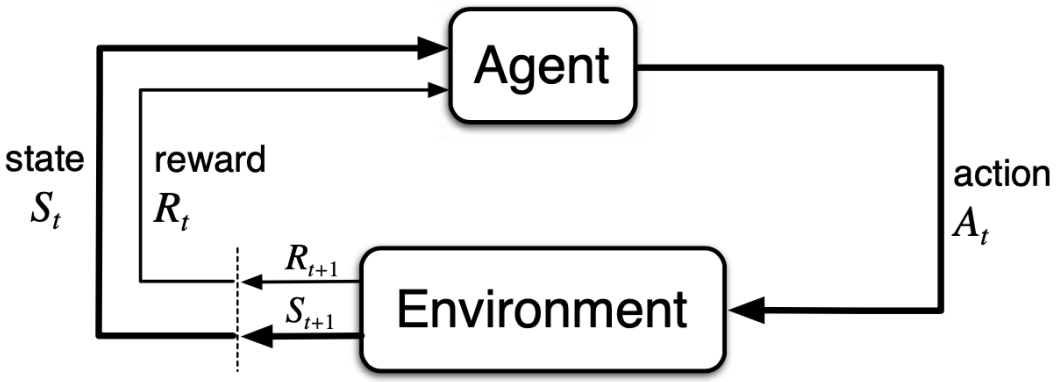


Figure 2.1.1: The agent-environment interaction as illustrated in Sutton and Barto (2018). At each time-step t the agent observes some state and performs an action which influences the environment state. From this interaction, the agent receives feedback via a reward signal and observes the next state.

of some transition function, finds itself in a new state. Reinforcement learning is distinct from other branches of control in that the dynamics of the environment are unknown to the agent. Instead, the agent must explore the environment sufficiently to discover which behaviors yield the most reward, and exploit this experience to make good decisions. The goal of the agent is simply to maximize cumulative reward over time. Consequently, reinforcement learning abstracts the problem of goal-directed learning from interaction into three distinct signals that are passed between the agent and environment: the choices made by the agent (actions), the basis on which the choices are made (states), and the agent's goal (rewards).

This abstraction has several key characteristics that make reinforcement learning an appealing problem solving paradigm. Firstly, the feedback used for learning is a simple scalar reward, which means that no explicit teacher or large labelled dataset is required for training. Secondly, little to no prior knowledge of the environment or task is required. Thirdly, reinforcement learning is valid in non-deterministic environments. Finally, reinforcement learning architectures are highly extensible and can be extended to incorporate aspects of planning (Walsh et al., 2010), intelligent exploration (Burda et al., 2018) and hierarchical control (Li et al., 2019; Nachum et al., 2018). A powerful illustration of this extensibility is the advent of *deep reinforcement learning*, or reinforcement learning combined with large neural network architectures. Many of reinforcement learning's most high profile practical achievements in recent years, such as mastering Chess (Silver et al., 2018), Atari

games (Mnih et al., 2015), to beating world champions at Go (Silver et al., 2016) and beating professional players at StarCraft (Vinyals et al., 2019), were in part only possible due to breakthroughs in modern neural network research. The experiments conducted in this dissertation exploit this same characteristic, experimenting with mechanisms that have proven successful in a natural language processing setting in the domain of reinforcement learning.

2.1.1 Markov Decision Processes

The reinforcement learning problem is often formalized as the "optimal control of incompletely-known Markov decision processes" (Sutton and Barto, 2018). MDPs are a mathematically idealized form of the reinforcement learning problem, whereby the transition function is Markovian - that is, the current state includes all aspects of the past agent-environment interactions that make a difference to the future. A finite MDP is a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is a finite set of states.
- $\mathcal{A}(s)$ is a finite set of actions available at state s.
- $P\left(s' \mid s, a\right)$ is the state-transition model.
- $R\left(s, a, s'\right)$ is the reward function.
- $\gamma \in [0, 1]$ is the discount factor.

The agent interacts with the environment at each of a sequence of discrete time steps, $t = 0, 1, 2, 3, \ldots$. At each time step t, the agent receives observation $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}(s)$ according to the agent's policy, which is a mapping from states to a distribution over actions $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$. At the subsequent time step the agent receives a numerical reward $r_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and transitions to a new state s_{t+1}. Under this formulation, the Markov property can formally be defined as

$$P\left(s_{t+1} \mid s_0, a_0, \ldots, s_{t-1}, a_{t-1}, s_t, a_t\right) = P\left(s_{t+1} \mid s_t, a_t\right). \tag{2.1}$$

The observations, actions, and rewards constitute the agent's experience. The agent uses this experience to update its policy and maximize the total expected reward, given by

$$r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.2}$$

where $0 \leq \gamma \leq 1$ is used to discount future rewards. If $\gamma = 0$, the agent will be myopic and only concerned with maximizing the immediate reward. As γ approaches 1, the agent will take future rewards into account more strongly, thus, becoming more far-sighted.

Ultimately, the aim of the reinforcement learning agent is to learn the optimal policy π^*. To define a partial ordering over policies, the state-value function is required. The value of state s following policy π is given by

$$V_\pi(s) = \sum_{a \in \mathcal{A}(s)} \pi(s,a) \left[\sum_{s'} P\left(s' \mid s, a\right) \left(R\left(s,a,s'\right) + \gamma V_\pi\left(s'\right)\right) \right]. \tag{2.3}$$

A policy π is better or equal to another policy π' if and only if $\forall s \in \mathcal{S}, V_\pi(s) \geq V_{\pi'}(s)$. It follows that all optimal policies share the same optimal value function $\forall s \in \mathcal{S}, V_{\pi^*}(s) = \max_\pi V_\pi(s)$. The optimal policy values are given by the set of solutions to the *Bellman optimality equations*:

$$sV_{\pi^*}(s) = \max_{a \in \mathcal{A}(s)} \sum_{s'} P\left(s' \mid s, a\right) \left[R\left(s,a,s'\right) + \gamma V_{\pi^*}\left(s'\right)\right]. \tag{2.4}$$

Having solved equation 2.4, the optimal policy π^* is greedy with respect to the optimal value function:

$$\pi^*(s,a) = \begin{cases} 1 & \text{if } a = \operatorname{argmax}_{a \in \mathcal{A}(s)} \sum_{s'} P\left(s' \mid s, a\right) V_{\pi^*}\left(s'\right) \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

Alternatively, it may be easier to learn the optimal policy using the action-value function $Q_\pi(s,a)$, which represents the expected return of taking action a in state s and following policy π thereafter. The optimal Q-value function Q_{π^*} can be written in terms of V_{π^*}:

$$Q_{\pi^*}(s,a) = \sum_{s'} P\left(s' \mid s, a\right) \left[R\left(s,a,s'\right) + \gamma V_{\pi^*}\left(s'\right)\right]. \tag{2.6}$$

Having solved equation 2.6, the optimal policy immediately follows.

2.1.2 Partial Observability and Non-Markovian Environments

MDPs are an extremely useful formalization of sequential decision making tasks, from which precise theoretical statements can be made about many common reinforcement learning algorithms.

In real world environments, however, the observations that the agent receives are often only a partial glimpse of the true underlying state of the system. These types of environments are described as Partially Observable Markov Decision Processes (POMDP), and differ from MDPs in that the agent does not receive the true state of the environment[1]. Instead, the agent receives an observation $o \in \Psi$ which is generated from the environment according to the probability distribution $o \sim \mathcal{O}(s)$.

Formally, a POMDP is a 7-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Psi, \mathcal{O}, \gamma)$, where:

- Ψ is a set of observations
- $\mathcal{O}$ is a set of conditional observation probabilities

The other five elements in the tuple remain the same as defined in section 2.1.1.

In POMDPs the agent must make decisions under uncertainty about the environment's true state. Consequently, optimal behavior may include an information gathering phase which is explicitly designed to improve the agent's estimation of the current state (Ha and Schmidhuber, 2018). The agent may also utilize a memory system in order to better understand the environment's underlying state (Hausknecht and Stone, 2017). For example, consider the T-maze experiment that is commonly used to study memory in rodents. At the start of the experiment the rodent is exposed to a stimulus and subsequently must turn down either the right or left branch of the maze (which is shaped like a T). The correct choice will lead to a reward and an incorrect choice a punishment. The rodent begins at the base of the maze and cannot see around the corners of each branch and consequently does not have access to the full state of the environment. Instead, the agent must remember the stimulus it was exposed to at the start of the experiment and use it to make the correct decision.

[1] Real world applications that include the use of POMDPs include management of patients with ischemic heart disease (Hauskrecht and Fraser, 2000), assistive technology for persons with dementia (Hoey et al., 2007) and aircraft collision avoidance (Kochenderfer et al., 2015).

In many real world tasks the Markov property itself seldom holds, or as Whitehead and Lin (1995) stated: "Markov decision tasks are an ideal. Non-Markov tasks are the norm." In these environments the agent cannot make optimal decisions without integrating information across time. Consequently, a large body of work has been developed that explores different mechanisms that can be used to provide memory in reinforcement learning systems. Several of these mechanisms will be detailed in section 2.2.

2.1.3 Function Approximation

For domains with large state-spaces, finding a solution to the optimal value function (equation 2.4) becomes intractable. It is simply not possible to store the value for each state in a table or array in memory, nor is the time or data to accurately fill these tables available. For example, consider the game of Go which is played on a 19x19 board. The state space complexity of Go, which represents the number of legal positions that are reachable from the starting position of the game, is estimated to be 10^{170} (Allis, 1994; Tromp and Farnebäck, 2006), which exceeds the number of atoms in the observable universe. Unfortunately, many real world problems also have extremely large state spaces. To overcome this, the policy or value function can be represented by a parameterized function, whereby the set of parameters ϕ approximate the state-space and have a much smaller dimension. The key strength of function approximation under this approach is its ability to generalize from a limited subset of experiences to states not yet encountered.

Function approximation in reinforcement learning does, however, pose some unique challenges. For example, the objective function is often non-stationary and must be approximated from incrementally acquired data. Furthermore, many reinforcement learning algorithms, such as dynamic programming and temporal difference learning, perform bootstrapping whereby update estimates are made on the basis of other estimates, which introduces bias to the target. Consequently, training methods that are suitable for non-stationarity and non-independent and identically distributed data are required.

In recent years, neural networks trained using gradient-based optimization algorithms have been the most widely adopted function approximation methods for

reinforcement learning. Under these approaches, the parameters ϕ are typically the connection weights in the layers of the neural network, which are updated during training via gradient-descent to minimize an objective function $J(\phi)$:

$$\phi = \phi - \alpha \nabla J(\phi), \tag{2.7}$$

where α is the step-size or learning rate. Conversely, in scenarios whereby we seek to maximize the objective function, such as directly optimizing the agent's policy as outlined in section 2.1.5, gradient-ascent is utilized:

$$\phi = \phi + \alpha \nabla J(\phi). \tag{2.8}$$

A variety of different gradient-based optimization algorithms exist, with two popular approaches being *stochastic gradient descent* and its modern extension *adaptive moment estimation* (adam). Stochastic gradient descent is distinct from batch or vanilla gradient descent in that a parameter update is performed for each training example, rather than on a batch of training examples. The advantages of stochastic gradient descent is that it is usually much faster than batch gradient descent and can be used to learn online (Ruder, 2017). However, stochastic gradient descent updates often have high variance that can cause the objective function to fluctuate, which complicates convergence to the exact minimum as the updates may keep overshooting. Decreasing the learning rate over time[2] is a commonly used remedy for this problem, but the rates and threshold of this decrease need to be predefined and are, thus, unable to adapt to the characteristics of the data.

Choosing the initial learning rate itself is typically a bottleneck for efficient training, whereby a learning rate that is too small will lead to slow convergence, and a learning rate too large can hinder convergence and cause the objective function to fluctuate around the minimum or even diverge. Furthermore, if the data is sparse and the features have different frequencies, which is very often the case in reinforcement learning, it may not be optimal to update the same learning rate for all the parameters. Instead, it may be advantageous to perform larger updates for rarely occurring features and smaller updates for commonly occurring features.

[2]Frequently called annealing.

Adam (Kingma and Ba, 2017) is a popular extension of stochastic gradient descent that addresses these challenges by computing adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients. Adam was designed to combine the advantages of two other gradient descent optimizers: AdaGrad (Duchi et al., 2011), which works well with sparse gradients, and RMSProp (Tieleman and Hinton, 2012), which is well-suited to online and non-stationary settings. Furthermore, Adam has the advantage of typically requiring very little hyper-parameter tuning and is widely considered to be one of the top performing gradient-descent optimization algorithms for problems with sparse input data (Ruder, 2017).

2.1.4 Value Function Estimation

As outlined in the previous section, function approximation is a commonly used strategy in reinforcement learning problems with large state-spaces. In such scenarios, the value function (equation 2.3) is approximated by a parameterized function $\hat{V}_\pi(s, w) \approx V_\pi(s)$, where the parameters $w \in \mathbb{R}^d$ have a much smaller dimension than the size of the state-space[3]. During training, the parameters w are updated via gradient-descent to minimize an objective function, such as the *Mean Squared Value Error*:

$$\text{VE}(w) = \mathbb{E}_\pi[(V_\pi(S) - \hat{V}_\pi(S, w))^2], \tag{2.9}$$

where S is a sequence of states sampled from the environment following policy π. A full update via stochastic gradient-descent is, thus, given by:

$$\Delta w = \alpha \frac{1}{2}(V_\pi(S_t) - \hat{V}_\pi(S_t, w_t))\nabla \hat{V}_\pi(S_t, w_t), \tag{2.10}$$

whereby each update is performed on a single training example. Having estimated $\hat{V}_\pi(S)$, the agent's policy is greedy with respect to the value function (equation 2.5). In practice, however, the policy is typically ϵ-greedy which follows the greedy strategy with probability $1 - \epsilon$ and selects a random action with probability

[3]The state-action value function (equation 2.6) can similarly be parameterized $\hat{Q}_\pi(s, a, w) \approx Q_\pi(s, a)$.

ϵ.

2.1.5 Policy Gradient Methods

The agent's policy itself can be represented as a parameterized function and used to optimize the expected return directly. In such cases, a value function may be useful for learning the policy parameters, but is not required for action selection (Sutton and Barto, 2018). The class of methods that follow this schema are called *policy gradient methods*. Methods that learn approximations to both the policy and value functions are often called *actor-critic methods*, where actor refers to the learned policy and critic the learned value function (Sutton and Barto, 2018).

Policy gradient methods maximize the expected total return (equation 2.2) by repeatedly estimating the gradient of the objective function $J(\theta)$ and updating the policy parameters $\theta \in \mathbb{R}^d$ in the direction that maximizes the objective function via gradient-ascent. There are several different related expressions for the policy gradient, which have the form:

$$\nabla J(\theta_t) = \mathbb{E}_t[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\Phi]. \tag{2.11}$$

where Φ may be one of several options, such as the total discounted reward (equation 2.2), the state-action value function (equation 2.6) or the *advantage function*, given by

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s). \tag{2.12}$$

The choice of $\Phi = A_\pi(s, a)$ yields close to the lowest possible variance (Greensmith et al., 2004; Schulman et al., 2018), although in practice the advantage function is often unknown and must be estimated. Intuitively, the advantage function provides a measure of whether or not the action is better or worse than the policy's expected behavior.

2.2 Sequence Models

In this section, several candidates for providing memory in reinforcement learning are detailed. Unlike traditional neural networks, which assume all inputs and outputs are independent, these networks are specifically designed to process sequential information[4]. These models allow an agent to retain information over time to inform future actions, serving the function of memory. The sequence models are grouped according to two categories: recurrent neural networks (sections 2.2.1 - 2.2.2) and Transformer-based models (sections 2.2.3 - 2.2.7). Of the Transformer-based models, the Gated Transformer-XL is the current state-of-the-art memory network, while the Universal Transformer and ReZero have not been evaluated in a reinforcement learning context.

2.2.1 Long Short-Term Memory Networks

Recurrent Neural Networks (RNNs) are a class of neural networks that process sequences by generating a series of hidden state vectors h_t as a function of the previous hidden state h_{t-1} and the current input x_t. Formally, the hidden state and output is computed as follows:

$$h_t = \tanh\left(b_h + U_h h_{t-1} + W_x x_t\right), \tag{2.13}$$

$$y_t = \text{softmax}\left(b_y + U_y h_t\right), \tag{2.14}$$

where,

- $x_t \in \mathbb{R}^{L \times d}$ is an input sequence vector of length L, where the superscript d refers to the sequence representation dimension and t to time step index.
- $h_t \in \mathbb{R}^h$ is the hidden state vector, where the superscript h indicates the number of hidden units.
- $W \in \mathbb{R}^{h \times d \times L}, U \in \mathbb{R}^{h \times h}$ and $b \in \mathbb{R}^h$ are the weight matrices and bias vectors that are updated during training.

[4]Sequences are defined as an enumerated collection of objects where order matters and repetition is allowed.

- $tanh$ and $softmax$ are activation functions.

A well-documented short coming of recurrent neural networks is their limited ability to model problems with long range temporal dependencies (Bengio et al., 1994). RNNs suffer from what is commonly known as the *vanishing/exploding gradient problem*, whereby the influence of a given input on the hidden layer either decays or blows up exponentially as it cycles around the network's recurrent connections (Bengio et al., 1994). Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997) attempt to remedy this through the addition of three gating layers, known as the *forget gate*, *input gate* and *output gate*, which regulate the flow of information into and out of the LSTM cell. In addition to the three gating mechanisms, the LSTM also incorporates an internal cell state c, which helps preserve information across the network's recurrent connections.

Intuitively, the *forget gate* controls the extent to which we forget values in the cell state. For each element in the previous cell state c_{t-1} a value between 0 and 1 is generated, whereby 1 represents to completely remember and 0 to completely forget. Through a similar mechanism, the *input gate* determines the extent to which new information is written into the internal cell state and the *output gate* controls which values in the cell are used to compute the output activation of the LSTM unit.

Formally, we define the full computation for an LSTM unit by

$$f_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f \right), \tag{2.15}$$

$$i_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i \right), \tag{2.16}$$

$$o_t = \sigma_g \left(W_o x_t + U_o h_{t-1} + b_o \right), \tag{2.17}$$

$$\tilde{c}_t = \tanh \left(W_c x_t + U_c h_{t-1} + b_c \right), \tag{2.18}$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \tag{2.19}$$

$$h_t = o_t \circ \tanh \left(c_t \right), \tag{2.20}$$

where $c_0 = 0$, $h_0 = 0$ and $\circ$ denotes the Hadamard product. The notation is defined as follows:

- $f_t \in \mathbb{R}^h$ is the forget gate activation vector.

- $i_t \in \mathbb{R}^h$ is the input gate activation vector.
- $o_t \in \mathbb{R}^h$ is the output gate activation vector.
- $\tilde{c}_t \in \mathbb{R}^h$ is the candidate cell states.
- $c_t \in \mathbb{R}^h$ is the internal cell state.
- σ_g is the sigmoid activation function.

LSTMs have achieved remarkable success across a wide number of domains, such as robot control (Mayer et al., 2006), speech recognition (Graves and Schmidhuber, 2005; Graves et al., 2013), handwriting recognition (Graves and Schmidhuber, 2008; Graves et al., 2007) time series prediction (Schmidhuber et al., 2005), object segmentation (Wang et al., 2018) and reinforcement learning (Hausknecht and Stone, 2017). When utilized as a memory system in reinforcement learning, the LSTMs hidden and internal cell state are commonly carried forward throughout the episode to preserve information from previous time-steps, as illustrated by figure 2.2.1 (Hausknecht and Stone, 2017). As mentioned in chapter 1, LSTMs have been the most widely adopted mechanism for providing memory in reinforcement learning in recent years.

2.2.2 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) (Cho et al., 2014) is a simpler alternative to the LSTM with fewer parameters. Like the LSTM, the GRU uses gating mechanisms to address the vanishing/exploding gradient problem by adaptively capturing dependencies of different time scales. Unlike the LSTM, the GRU uses two gates instead of three and does not have an additional internal memory cell. The two gates, the *update gate* and the *reset gate*, are given by

$$z_t = \sigma_g \left(W_z x_t + U_z h_{t-1} + b_z \right), \tag{2.21}$$

$$r_t = \sigma_g \left(W_r x_t + U_r h_{t-1} + b_r \right), \tag{2.22}$$

where $z_t \in \mathbb{R}^h$ is the update gate vector, $r_t \in \mathbb{R}^h$ is the reset gate vector and σ_g is the sigmoid activation function as before. The hidden state is then computed using the gates as follows:

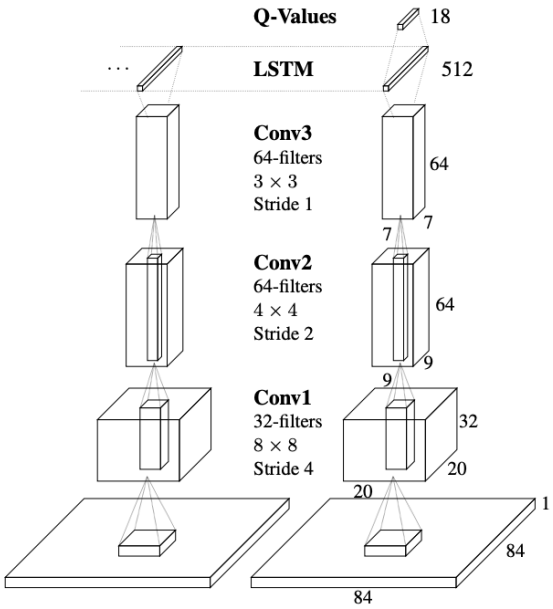


Figure 2.2.1: An LSTM was used on the partially observable Atari games in Hausknecht and Stone (2017). Each observation was a single-channel image of the game screen, which was first processed by three convolutional layers to extract the features in the image. The LSTM processed the output of these layers and passed the hidden and internal cell states forward throughout the episode. The hidden and internal cell state, thus, act as a compressed form of memory, ensuring that each Q-value was informed by both the current observation and the history of observations.

$$\tilde{h}_t = \tanh\left(W_h x_t + U_h\left(r_t \circ h_{t-1}\right) + b_h\right), \tag{2.23}$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h}_t, \tag{2.24}$$

where $h_0 = 0$. A visual representation of the GRU gating system is depicted in figure 2.2.2.

According to the authors, the reset gate r modulates how much information in the previous hidden state is *dropped* or *forgotten*, while the update gate z selects whether the hidden state is to be updated with a new hidden state $\tilde{h}$. As each hidden unit has separate reset and update gates, dependencies will be captured over different time scales. For example, those units that learn short-term dependencies will tend to have frequently active reset gates, while longer-term dependencies will have active update gates.

The GRU has empirically been demonstrated to yield similar performance to an LSTM on polyphonic music modeling, speech signal modeling and natural language

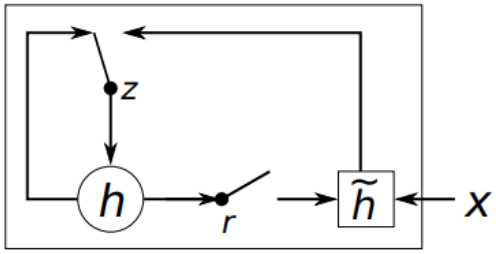


Figure 2.2.2: A visual illustration of the proposed hidden activation function provided by Cho et al. (2014). The update gate z determines where the hidden state is updated with a new hidden state $\tilde{h}$, while the reset gate r selects whether the previous hidden state should be ignored.

processing tasks (Ravanelli et al., 2018; Su and Kuo, 2019). In a much larger study on machine translation, however, the LSTM was found to consistently outperform GRUs (Britz et al., 2017).

2.2.3 The Canonical Transformer

The Canonical Transformer was introduced in the seminal paper "Attention Is All You Need" in 2017, as the new state-of-the-art sequence model for machine translation (Vaswani et al., 2017). As the title makes reference to, the Transformer relies solely on the attention function to model temporal dependencies, dispensing with recurrent units completely.

The attention function has three significant advantages over recurrent units. Firstly, it does not suffer from the vanishing/exploding gradient problem in the same way RNNs do, as the receptive field of the attention function consists of the entire sequence, which allows dependencies to be modeled regardless of their distance in the sequence. Secondly, it is far more parallelizable, as measured by the minimum number of sequential operations required in each layer. The attention function requires $O(1)$ sequential operations, while a recurrent layers requires $O(L)$, where L is the sequence length[5]. Thirdly, the attention function requires significantly less training time[6], which is due to both lower computational complexity per layer

[5]Each recurrent unit relies on the hidden state and output from the previous unit. This inherently sequential computation precludes parallelization within training examples, which becomes critical at longer sequence lengths (Vaswani et al., 2017).

[6]On the WMT 2014 English-to-French translation task, the Canonical Transformer established a new single-model state-of-the-art after training for 3.5 days on eight GPUs, a small fraction of the training costs of the previous best models from the literature.

and more parallelizability. The complexity per layer is $O(L^2 \cdot d)$ for the attention function and $O(n \cdot d^2)$ for recurrent units, where d is the sequence representation dimension. For the majority of modern machine translation models, the sentence representation dimension is far larger than the sequence length (Wu et al., 2016).

Formally, the input to the attention function consists of query and key vectors of dimension d_k, and value vectors of d_v. The output is a weighted sum of the value vectors, where the weight assigned to each value is determined by the dot-product of the query with the corresponding key. The key/value/query concepts originate from retrieval systems, whereby a query is mapped against a set of keys by some similarity function. The value for the key most strongly associated with the query is retrieved.

The Canonical Transformer utilizes a form of attention called *scaled dot-product attention* specifically, which is given by

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V, \tag{2.25}$$

where the query, key and value vectors are packed together into matrices $Q \in \mathbb{R}^{L \times d_k}$, $K \in \mathbb{R}^{L \times d_k}$ and $V \in \mathbb{R}^{L \times d_v}$. $\frac{1}{\sqrt{d_k}}$ is the scaling factor, which the authors hypothesize helps stabilize the gradients of the softmax function when the dot products grow large (Vaswani et al., 2017).

Instead of performing a single attention function, multiple attention heads are computed in parallel; the output values are then concatenated and linearly projected:

$$\text{MultiHeadAttention}(Q, K, V) = \text{Concat}(head_1; ...; head_\eta)W^O \tag{2.26}$$

where $head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$. The weight matrices per attention head are denoted as $W_i^Q, W_i^K \in \mathbb{R}^{d \times d_k/\eta}$, $W_i^V \in \mathbb{R}^{d \times d_v/\eta}$ and the final output weight matrix $W^O \in \mathbb{R}^{hd_v \times d}$. Multi-head attention allows the model to jointly attend to different representational subspaces at different positions in the input sequence (Vaswani et al., 2017). Both scaled dot-product attention and multi-head attention are illustrated by figure 2.2.3.

In addition to the attention mechanism, a fully connected feed-forward network

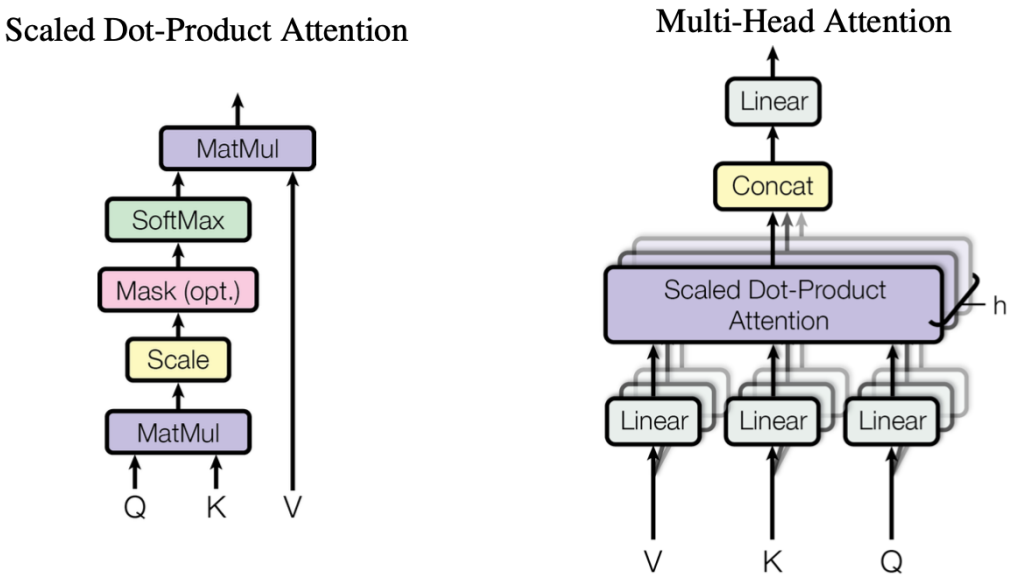


Figure 2.2.3: (left) Scaled dot-product attention attention (equation 2.25), which receives query, key and value vectors Q, K and V as input and outputs a weighted sum of the value vectors. (right) Multi-head attention (equation 2.26) which consists of several attention layers running in parallel. Note that the authors Vaswani et al. (2017) used h to represent the number of attention heads, while we have used η.

is applied to each position separately and identically. The feed-forward network consists of two linear transformations with a ReLU activation in between:

$$\text{FeedForward}(x) = \max\left(0, xW_1 + b_1\right) W_2 + b_2. \tag{2.27}$$

As the Transformer solely relies on the attention mechanism which is permutation-invariant, the network has no mechanism to infer information about the order of the sequence. Consequently, information about the positions of the elements in the sequence is injected into the sequence itself. To this end, positional encodings $P \in \mathbb{R}^{L\times d}$ are summed with the input embedding $X \in \mathbb{R}^{L\times d}$. The form of positional encodings used in (Vaswani et al., 2017) are known as *sinusoidal positional embeddings* and utilizes the sine and cosine function of different frequencies, given by

$$\begin{aligned} \mathrm{P}_{i,2j} &= \sin(i/10000^{2j/d}), \\ \mathrm{P}_{i,2j+1} &= \cos(i/10000^{2j/d}), \end{aligned} \tag{2.28}$$

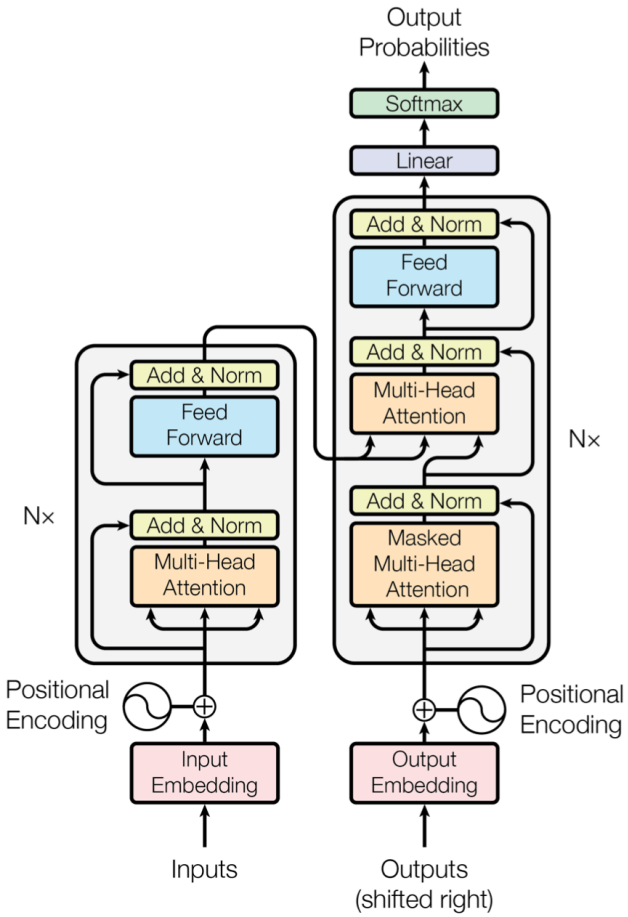


Figure 2.2.4: The full Transformer architecture as depicted in Vaswani et al. (2017).

where $i \leq j \leq d$ is the vector dimension and $1 \leq i \leq L$ is the position.

The full Transformer network consists of a stack of N submodules comprised of multi-head attention and fully connected feed-forward layers, as illustrated in figure 2.2.4. The Transformer submodule also includes a residual connection (He et al., 2015) and a layer normalization layer (Ba et al., 2016), given by

$$x_i = \text{LayerNorm}(x + \text{sublayer}(x)) \tag{2.29}$$

where sublayer $\in$ {MultiHeadAttention, FeedForward}. Residual connections are commonly employed to improve signal propagation in neural networks which enables training at greater depth (more layers). Normalization techniques in contrast accelerate training of deep networks by ensuring that signals at initialization have zero mean and unit variance as they propagate through the network (Ioffe and Szegedy, 2015; Ba et al., 2016).

The Canonical Transformer is arranged in an encoder-decoder structure, which was common among the state-of-the-art sequence models at the time (Cho et al., 2014; Bahdanau et al., 2016). The function of the encoder is to map the input sequence of symbol representations $(x_1, ..., x_L)$ to a fixed fixed length vector representation $z = (z_1, ..., z_L)$. The decoder receives the vector z and generates an output sequence $(y_1, ..., y_L)$ of symbols one element at a time. The model is auto-regressive at each step, consuming the previous output symbol as input when generating the next. The outputs fed into the decoder are masked (set to $-\infty$) to prevent the decoder from attending to subsequent positions.

2.2.4 Transformer-XL

A key strength of the attention mechanism is its global receptive field, which allows the Transformer to model long-term temporal dependencies. The attention function does, however, suffer from the *context fragmentation problem*, whereby attention is restricted to the current segment of the sequence (Dai et al., 2019). For example, if you wish to translate a book from one language to another, the words in the book would typically be broken up into fixed-length segments (e.g. 512 consecutive words) and supplied to the Transformer. The attention mechanism does not, however, have any mechanism to facilitate the flow of information between these segments, as illustrated by figure 2.2.5. Moreover, these fixed-length segments typically do not take into account semantic boundaries, such as whether the segment ends in the middle of a sentence or a paragraph. Consequently, the Canonical Transformer's ability to model temporal dependencies is restricted, which frequently leads to inefficient optimization and inferior performance. The Transformer-XL[7] addresses the context fragmentation problem by reusing hidden states between segments, which effectively enables the flow of information across segments (Dai et al., 2019). Additionally, Dai et al. (2019) introduced a novel relative positional encoding scheme to preserve information about the order of the sequence.

Formally, let two consecutive segments of length L be $s_\tau = [x_{\tau,1}, \cdots, x_{\tau,L}]$ and $s_{\tau+1} = [x_{\tau+1,1}, \cdots, x_{\tau+1,L}]$. The hidden states produced by the Transformer-XL when processing a sequence are *fixed* and *cached* to be reused as an extended context

[7] XL meaning extra long.

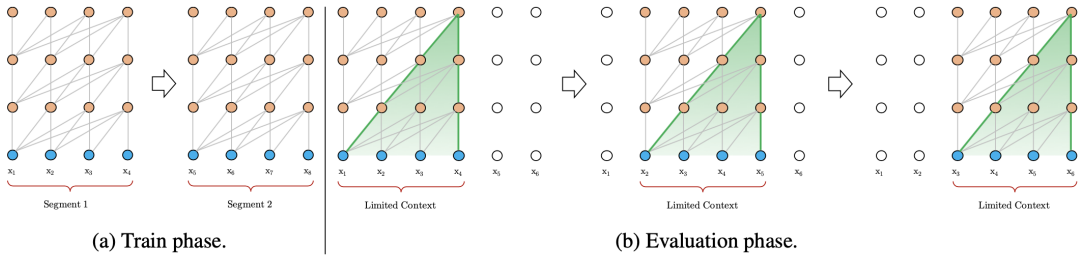


Figure 2.2.5: The context fragmentation problem of the Canonical Transformer model whereby the attention function is restricted to a single segment and cannot exploit information from previous segments. In this illustration each segment is of length 4 (Dai et al., 2019).

when processing the next segment. The n-th layer hidden state sequence produced for the τ-th segment s_τ is denoted as $m_\tau^n \in \mathbb{R}^{L \times d}$. The n-th layer hidden state for segment $s_{\tau+1}$ is produced schematically as follows:

$$\begin{aligned} \tilde{m}_{\tau+1}^{n-1} &= \left[\text{SG}\left(m_\tau^{n-1}\right) \circ m_{\tau+1}^{n-1}\right], \\ q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau+1}^n &= m_{\tau+1}^{n-1} W_q^\top, \tilde{m}_{\tau+1}^{n-1} W_k^\top, \tilde{m}_{\tau+1}^{n-1} W_v^\top, \\ m_{\tau+1}^n &= \text{TransformerLayer}\left(q_{\tau+1}^n, k_{\tau+1}^n, v_{\tau+1}^n\right). \end{aligned} \tag{2.30}$$

$\text{SG}(\cdot)$ is the stop-gradient function, W denotes the model parameters and the notation $[m_u \circ m_v]$ indicates the concatenation of two hidden sequences along the length dimension. In comparison to the Canonical Transformer, the critical difference is that the key $k_{\tau+1}^n$ and value $v_{\tau+1}^n$ are conditioned on the extended context $\tilde{m}_{\tau+1}^{n-1}$, which allows the model to exploit information from previously processed segments and model longer-term dependencies. This extended context is visually illustrated by the green paths in figure 2.2.6. The stop-gradient function stops gradient computation for the previous segment, which ensures that the hidden states from previous segments are fixed and only the attention distribution for the current segment can be updated. The number of cached hidden states M is defined as the memory $\mu_\tau^n \in \mathbb{R}^{\bar{M} \times d}$.

The memory mechanism in the Transformer-XL effectively addresses context fragmentation, but introduces a new problem: how can positional information remain coherent when reusing states? The Canonical Transformer uses a form of positional encoding where the i-th row P_i corresponds to the i-th *absolute* position in the segment. If the same positional encoding mechanism is adopted with hidden

state reuse, each hidden state would be computed by

$$\begin{aligned} m_{\tau+1} &= f\left(m_\tau, X_{s_{\tau+1}} + P_{1:L}\right), \\ m_\tau &= f\left(m_{\tau-1}, X_{s_\tau} + P_{1:L}\right), \end{aligned} \tag{2.31}$$

where f represents a transformation function. In this case, X_{s_τ} and $X_{s_{\tau+1}}$ are associated with the same positional encoding $P_{1:L}$. Consequently, the model will have no information to distinguish the difference between $x_{\tau,j}$ and $x_{\tau+1,j}$ for any $j = 1, \ldots, L$, which will result in substantial performance loss.

To solve this problem, the authors introduced a novel encoding scheme that incorporates the *relative* positional information in the hidden states. The relative positional encodings are denoted as $R \in \mathbb{R}^{L\times d}$, where the i-th row R_i corresponds to the relative distance of i between two positions. By injecting the relative distance dynamically into the attention mechanism, the query vector can distinguish between $x_{\tau,j}$ and $x_{\tau+1,j}$ through the different distances.

The relative positional encoding scheme was formulated by first decomposing the attention score between q_i and k_i within the same segment as:

$$\begin{aligned} P_{i,j} = &\underbrace{X_i^\top W_q^\top W_k X_j}_{(a)} + \underbrace{X_i^\top W_q^\top W_k P_j}_{(b)} \\ &+ \underbrace{P_i^\top W_q^\top W_k X_j}_{(c)} + \underbrace{P_i^\top W_q^\top W_k P_j}_{(d)}. \end{aligned} \tag{2.32}$$

The four terms were reparameterised as follows:

$$\begin{aligned} R_{i,j} = &\underbrace{X_i^\top W_q^\top W_{k,X} X_j}_{(a)} + \underbrace{X_i^\top W_q^\top W_{k,R} R_{i-j}}_{(b)} \\ &+ \underbrace{u^\top W_{k,X} X_j}_{(c)} + \underbrace{v^\top W_{k,R} R_{i-j}}_{(d)} \end{aligned} \tag{2.33}$$

Three major changes were made to the attention mechanism to accommodate the relative positional encoding scheme:

1. The absolute positional embedding P_j were replaced in terms *(b)* and *(d)* with the relative counterpart R_{i-j}, where R is still a sinusoid encoding matrix as in Vaswani et al. (2017).

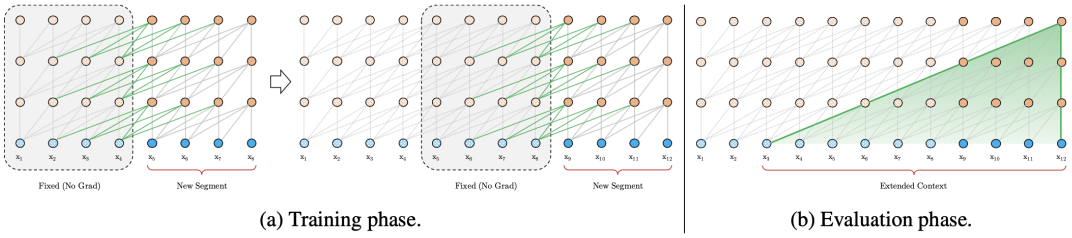


Figure 2.2.6: Illustration of the increased attention span of the Transformer-Xl model, whereby the attention function can exploit information from prior segments (Dai et al., 2019).

2. The trainable parameters $u \in \mathbb{R}^d$ and $v \in \mathbb{R}^d$ replaced query $P_i^\top W_q^\top$ in *(c)* and $P_i^\top W_q^\top$ in *(d)*.

3. The two weight matrices $W_{k,X}$ and $W_{k,R}$ are separated to produce content-based key vectors and location-based key vectors.

Under the new reparameterization, each term has an intuitive meaning: *(a)* represents content-based addressing, *(b)* captures content-dependent positional bias, *(c)* governs global content bias and *(d)* encodes a global positional bias (Dai et al., 2019).

2.2.5 Gated Transformer-XL

Despite the impressive performance of the Canonical Transformer and Transformer-XL across a range of NLP tasks, these models have performed poorly when used as memory systems in reinforcement learning. For example, the canonical model was not even able to solve simple bandit tasks and tabular MDPs (Mishra et al., 2018). Similarly, Parisotto et al. (2019) demonstrated that the performance of the Transformer-XL is often comparable to a random policy. Mishra et al. (2018) hypothesized that despite their global receptive field, pure attention look ups cannot compare two adjacent time steps (such as single state-action-state transition) in the same way a single convolution for example can.

In 2019, however, Parisotto et al. (2019) demonstrated that replacing the residual connections in the Transformer submodule with Gated Recurrent Units (section 2.2.2) and reordering the layer-normalization mechanism stabilizes learning and drastically improves performance in reinforcement learning. The Gated Transformer-

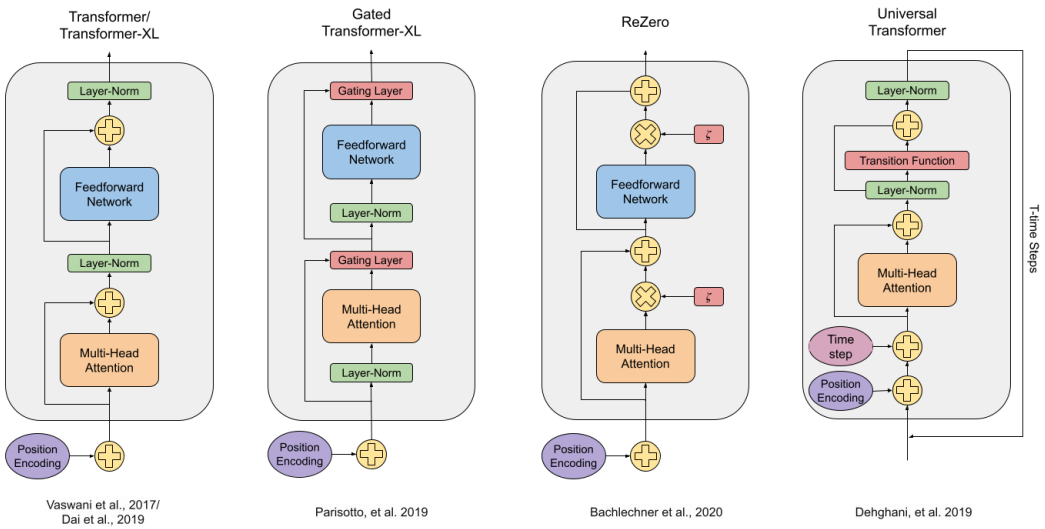


Figure 2.2.7: Overview of the different Transformer submodules. From left to right: the Canonical Transformer (Vaswani et al., 2017) and Transformer-XL (Dai et al., 2019), the Gated Transformer-XL (Parisotto et al., 2019), ReZero (Bachlechner et al., 2020) and the Universal Transformer (Dehghani et al., 2019).

XL (GTrXL) outperformed a competitive LSTM baseline on the challenging DMLab-30 benchmark suite (Beattie et al., 2016; Parisotto et al., 2019). Moreover, the GTrXL was demonstrated to scale better with longer temporal dependencies, learn faster and exhibited comparable robustness to initial seed and hyper-parameter sensitivity.

The GTrXL is primarily based on the Transformer-XL architecture and uses the same attention mechanism with relative positional encoding (equation 2.33) and hidden state reuse between segments (equation 2.30). The decision to replace the residual connection with a gating mechanism was motivated by the success of these mechanisms across a variety of architectures (Hochreiter and Schmidhuber, 1997; Cho et al., 2014). Parisotto et al. (2019) experimented with various different gating mechanisms, such as short-cut-only gating (He et al., 2016), highway connections (Srivastava et al., 2015) and sigmoid-tanh gating (Oord et al., 2016), but ultimately found that gated recurrent units yielded the highest performance.

Additionally, the authors demonstrated that reordering the layer-normalization, as done in several previous studies (He et al., 2015; Baevski and Auli, 2019; Radford et al., 2019), helped stabilize and improve overall performance in both the Transformer-XL and GTrXL. Placing the layer normalization on the input stream

of the multi-head attention mechanism and feedforward network, as illustrated by figure 2.2.7, enables an identity map from the input of the Transformer to its output. Parisotto et al. (2019) hypothesize that assuming the Transformer submodules at initialization produces values that are in expectation near zero, the state encoding is passed untransformed to the policy and value heads, enabling the agent to learn a Markovian policy at the start of training. This may be useful in environments where reactive behaviors need to be learned before memory-based ones, or as the authors put it "an agent needs to learn how to walk before it can learn how to remember where it has walked" (Parisotto et al., 2019).

2.2.6 ReZero

Experimentation with very deep neural networks has been a central driving force in modern machine learning research (He et al., 2015; Brown et al., 2020). The expressivity of neural networks typically grows exponentially with depth (Poole et al., 2016), which enables strong generalization. Deep neural networks are, however, often difficult to train due to poor signal propagation through the network. Recent theoretical work by Pennington et al. (2017) demonstrated that *dynamic isometry* plays a key role in efficiently training deep neural networks. ReZero[8] is a simple architectural modification to deep neural networks that ensures initial dynamic isometry by gating each residual connection with a zero-initialized learnable parameter (Bachlechner et al., 2020). This additional parameter dynamically facilitates well-behaved gradients and arbitrarily deep signal propagation. ReZero was empirically demonstrated to accelerate convergence in Transformer networks and allow training at greater depth.

Formally, we define a deep neural network with a depth of $\mathcal{L}$ layers and width w. An input signal x_0 of width w propagates through $\mathcal{L}$ layers that perform functions $\mathrm{F}\left[\mathcal{W}_i\right] : \mathbb{R}^w \to \mathbb{R}^w$, where W_i denotes all parameters in layer $i = 1, ..., \mathcal{L}$. The signal propagation through the network is given by

$$x_{i+1} = \mathrm{F}\left[\mathcal{W}_i\right](x_i)\,. \tag{2.34}$$

It is well established that a major determinant of stable signal propagation when

[8] ReZero stands for ***re**sidual with **zero** initialization*.

training deep neural networks lies in appropriately selecting the initial weights. For example, seminal work by Glorot and Bengio (2010) demonstrated that appropriately-scaled Gaussian weights can help mitigate the vanishing/exploding gradient problem in deep feedforward networks. These random weight initializations were primarily driven by the principle that the mean squared singular value of the input-output Jacobian (equation 2.35) should remain close to 1. This condition implies that a randomly selected error vector will preserve its norm under backpropagation on *average*. It does not, however, provide any guarantees of the worst case growth or shrinkage of an error vector, and was recently recognized to be insufficient to guarantee trainability (Pennington et al., 2017). The stronger condition of *dynamic isometry* demands that *every* Jacobian singular value in the network remains close to 1. Under this stronger requirement, every error vector will approximately preserve its norm and additionally all angles between different error vectors will be preserved (Pennington et al., 2017).

$$J_{\text{io}} \equiv \frac{\partial x_{\mathcal{L}}}{\partial x_0}. \tag{2.35}$$

Two key components relevant to signal propagation in the Canonical Transformer include layer normalization and the multi-head attention function. Neither component in isolation nor in combination with a residual connection can satisfy dynamic isometry, as was shown in Bachlechner et al. (2020). The authors verified this claim in practice by computing the input-output Jacobian for Canonical Transformer encoder layers of various depths, illustrated in figure 2.2.8. Although the shallow version of the Canonical Transformer exhibited a singular value distribution peaked around unity, the deeper versions were poorly distributed. This finding is consistent with the observation that deep transformers are typically very difficult to train.

In order to facilitate deep signal propagation in transformers, Bachlechner et al. (2020) proposed removing the layer normalization function and rescaling the multi-head attention and position-wise feedforward network with a zero-initialized learnable parameter, given by

$$x_{i+1} = x_i + \zeta_i \,\text{sublayer}\,(x_i)\,, \tag{2.36}$$

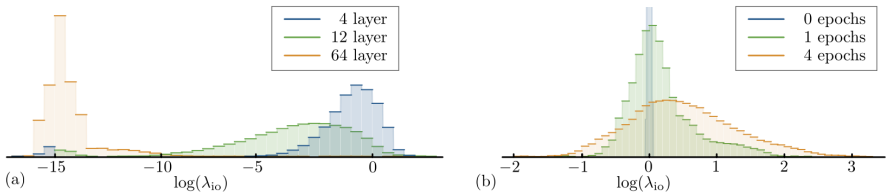


Figure 2.2.8: Histograms depicting the log singular values ($\log(\lambda_{\text{io}})$) for the input-output Jacobian as depicted in Bachlechner et al. (2020). (a) Canonical Transformer encoder at initialization with depths 4, 12 and 64 layers. (b) ReZero Transformer encoder with 64 layers before and during training.

where ζ_i is the learned residual weight parameter and sublayer $\in \{\text{MultiHeadAttention, FeedForward}\}$. At initialization, $\zeta_i = 0$, which allows for unimpeded signal propagation through the network and trivially satisfies dynamic isometry. Bachlechner et al. (2020) verified these claims by plotting the input-output Jacobian for a 64-layer ReZero Transformer, as illustrated in figure 2.2.8. Interestingly, ζ_i serves a similar purpose as reordering the layer normalization mechanism in section 2.2.5, by creating an identity map between the input and output of the Transformer at the start of training. Consequently, in addition to speeding up convergence and allowing training at greater depth, utilizing ReZero in a reinforcement learning setting may enable the agent to learn a Markovian policy at the start of training.

2.2.7 Universal Transformer

Despite the many successes of the Canonical Transformer highlighted in section 2.2.3, these models fail to generalize on a variety of simple tasks which recurrent models handle with ease. For example, copying strings or logical inference when the string or formula lengths exceed those observed at training time (Dehghani et al., 2019). The Universal Transformer (UT) addresses this problem by combining the parallelizability and global receptive field of the attention mechanism with the recurrent inductive bias of RNNs. Additionally, the UT adds a *dynamic halting mechanism* (Graves, 2017) which allows the model to select the number of refinement steps for each input in the sequence dynamically (Dehghani et al., 2019).

The UT is based on the same encoder-decoder architecture as the Canonical Transformer, but applies a recurrent transition function to the representations of each symbol in the input and output sequence. This transition function does not,

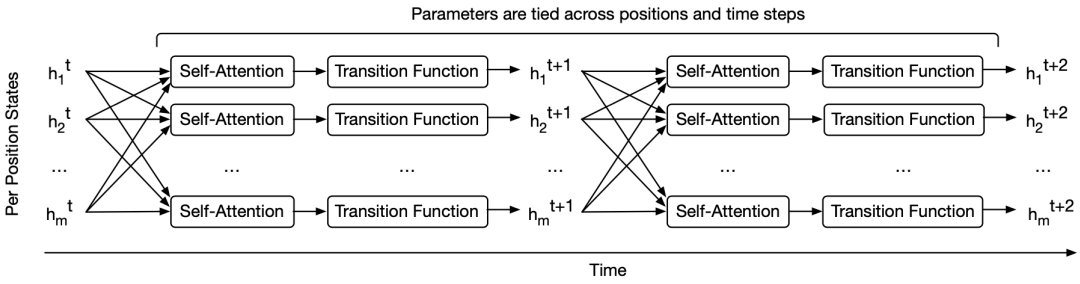


Figure 2.2.9: The Universal Transformer processes the sequence by repeatedly refining each symbol in parallel as illustrated in (Dehghani et al., 2019). Information is combined from different positions using self-attention (equation 2.25) and applying a recurrent transition function across all time steps $1 \leq t \leq T$. h_i^t is the representation for input symbol $1 \leq i \leq L$ for each recurrent time-step t. T is determined with dynamic halting for each symbol in the sequence.

however, recur over positions in the sequence like most applications of RNNs, but rather over consecutive revisions of the vector representations of each position (i.e. over "depth"). In each recurrent time step, each position in the sequence is concurrently revised in two sub-steps. In the first step, the attention function maps the input sequence to a vector representation whereby each position in the sequence is informed by all the other positions at the previous time step. In the second step, the outputs of the attention function are processed by a recurrent transition function which is shared across position and time. The recurrent transition function can be applied any number of times, as UTs can have a variable number of per-symbol processing steps. This is in sharp contrast to the recurrent and Transformer-based models detailed so far, which apply a fixed stack of layers and as a result have a constant depth. This process is illustrated by figure 2.2.9.

The UT submodule is very similar to the Canonical Transformer submodule, with the exception of the recurrent transition replacing the position-wise feed forward network, $LayerNorm(x + transition(x))$. A visual comparison between the canonical and Universal Transformer is depicted in figure 2.2.7. The transition function is either a separable convolution (Chollet, 2017) or a fully-connected neural network consisting of a single rectified-linear activation function between two affine transformations. The UT uses the same scaled dot-product attention (equation 2.25) and multi-head attention mechanism (equation 2.26) as Vaswani et al. (2017). Additionally, the UT uses two-dimensional (position, time) *coordinate em-*

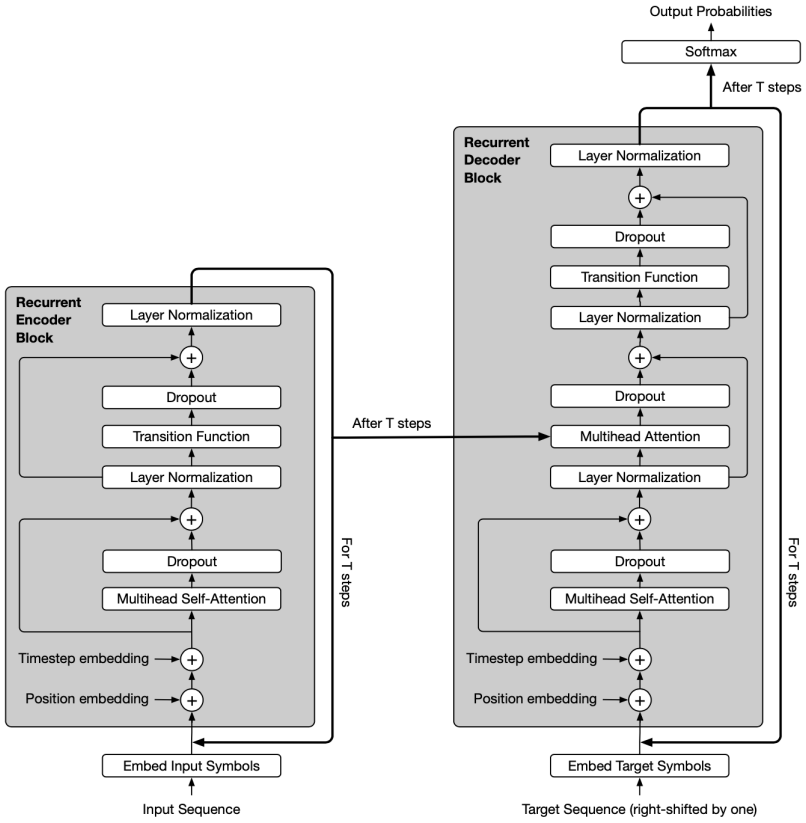


Figure 2.2.10: The full Universal Transformer architecture as depicted in (Dehghani et al., 2019).

beddings. These embeddings are obtained by computing the sinusoidal positional embeddings for positions $1 \leq i \leq L$ and the time step $1 \leq t \leq T$ separately for each vector-dimension $i \leq j \leq d$, and summing:

$$C^t_{i,2j} = \sin\left(i/10000^{2j/d}\right) + \sin\left(t/10000^{2j/d}\right), \tag{2.37}$$

$$C^t_{i,2j+1} = \cos\left(i/10000^{2j/d}\right) + \cos\left(t/10000^{2j/d}\right). \tag{2.38}$$

After updating all positions of the input sequence in parallel for T time steps, the final output of the Universal Transformer encoder is a matrix $H^T \in \mathbb{R}^{L \times d}$ for the L symbols in the input sequence. The decoder shares the basic recurrent structure as the encoder, and additionally attends to the final encoder representation H^T. Like the Canonical Transformer, the UT is autoregressive: the encoder produces an output one symbol at a time and the decoder consumes the previous output positions. The decoder self-attention distributions are masked as in (Vaswani et al., 2017) to prevent the model from attending to positions left of any predicted symbol.

The full UT architecture is illustrated in figure 2.2.10.

One of the most interesting features of the UT which has only been briefly touched upon is the *dynamic halting* mechanism. In sequence processing systems, certain symbols (such as words or states in reinforcement learning) are more ambiguous than others (Dehghani et al., 2019) and, thus, may require more resources to process. Adaptive Computational Time (ACT) (Graves, 2017) is a mechanism that dynamically determines the number of computational steps needed to process each input symbol (called the "ponder time"), based on a scalar halting probability predicted by the model at each step (Dehghani et al., 2019). Inspired by this mechanism, the UT adds a ACT halting mechanism to each symbol in the sequence. Once the per-symbol recurrent transition function halts, its state is simply copied to the next step until all the functions halt or the maximum number of steps is reached. More details are available in Appendix A.

2.3 Benchmarks, Ablation Studies and Integrated Approaches

Many successful modern reinforcement learning algorithms were built upon a strong theoretical foundation, such as temporal difference learning (Sutton, 1988) and other foundational algorithms. Ideally, good theory should provide insights into our algorithms and illuminate the path towards general improvement beyond ad-hoc tinkering. For many difficult problems, however, theory often lags behind practice. Unfortunately, this is the case for many deep reinforcement algorithms (Osband et al., 2020). In such scenarios, empirical research is a powerful framework to make practical progress before reaching full theoretical understanding. The successful development of algorithms and theory usually works in combination, with insights from each side enriching the other.

2.3.1 Benchmarks and Ablation Studies

Two popular methods to conduct empirical research in reinforcement learning, and machine learning more generally, are benchmarks and ablation studies. For example, all of the Transformer models detailed in chapter 2 were evaluated on benchmarks.

Benchmarks provide a shared platform on which to compare different algorithms and evaluate their strengths and weaknesses. Ablation studies in contrast are used to investigate more targeted aspects of the algorithm, such as which gating mechanism is the most effective in GTrXL or where the layer-normalization should be placed (Parisotto et al., 2019).

Despite their popularity, benchmarks are a contentious issue in reinforcement learning research. Unfortunately, the field currently lacks a singular or handful of standardized benchmarks on which research practitioners can evaluate and compare their algorithms. In other words, there is no MNIST or ImageNet for reinforcement learning, as there is in computer vision. Instead, two differing schools of thought have developed. On the one hand, benchmark suites such as DeepMind Lab contain several 3D simulation problems inspired by experiments in neuroscience, which are designed to evaluate whether an agent can solve large, complex tasks (Beattie et al., 2016). Various large-scale learning systems such as MERLIN (Wayne et al., 2018), IMPALA (Espeholt et al., 2018) and GTrXL (Parisotto et al., 2019) were evaluated on this benchmark suite. The drawback of DeepMind Lab, however, is that these experiments typically require several billion training samples and cloud-scale compute, which does not lend itself to hyper-parameter study, quick prototyping or replication. Furthermore, the cost of running these experiments can be viewed as further exacerbating a growing compute divide, whereby research in machine learning is only available to a handful of firms and universities with enormous compute-budgets (Ahmed and Wahed, 2020).

On the other hand, smaller testbeds and toy environments have been developed that are designed for rapid prototyping and statistically significant comparison of new algorithms (Rafiee, 2020). These benchmarks do not, however, typically test how well the algorithm scales, which is crucial for deploying reinforcement learning systems in the real world. Consequently, there is a need for benchmarks that strike a balance between these two schools of thought.

Deepmind's Behaviour Suite for Reinforcement Learning (**bsuite** for short) attempts to provide such a middle ground (Osband et al., 2020). This benchmark suite was designed to test targeted aspects of agent behavior, such as exploration or memory, rather than integrating general learning ability. The advantage of using

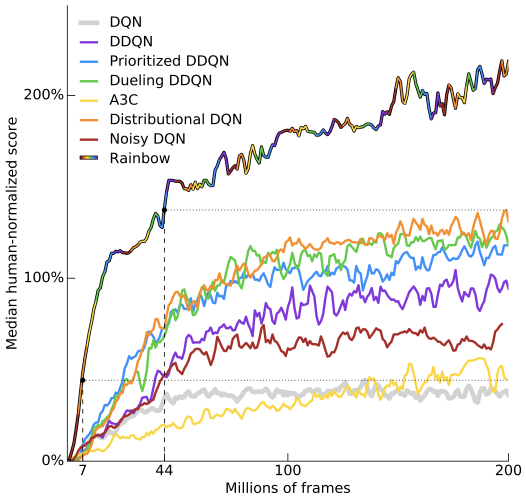


Figure 2.3.1: The median human-normalized performance of the Rainbow algorithm in comparison to six other published baselines on across 57 Atari games (Hessel et al., 2017). Curves are smoothed with a moving average over 5 points.

such targeted and transparent experiments is that confounding factors are stripped away and the performance on tasks directly corresponds to the key aspect of behavior being tested. Furthermore, these experiments were designed with an emphasis on scalability and evaluate agent behavior at increasing levels of complexity. Consequently, **bsuite** provides an ideal set of experiments for rapid prototyping and replication, while still evaluating how well these algorithms scale.

2.3.2 Integrated Approaches

As outlined in chapter 2, the extensibility of the reinforcement learning method is one of its many appealing traits. In line with this observation, it is not uncommon to experiment with combining several different innovations in machine learning research into a single integrated system. The *Rainbow* algorithm is an excellent example of this, whereby several independent improvements to the Deep Q-Learning algorithm (Mnih et al., 2013), such as double q-learning (van Hasselt et al., 2015), prioritized experience replay (Schaul et al., 2016) and dueling networks (Wang et al., 2016) amongst others, were integrated into a single algorithm that achieved state-of-the-art performance on the Atari 2600 benchmark at the time (depicted in figure 2.3.1) (Hessel et al., 2017).

Transformers are currently at a similar point to the Deep Q-Learning algorithm in 2016, whereby a number of independent innovations that address different issues

have been developed. For example, ReZero improved signal propagation through weighted residual connections, while the Universal Transformer incorporated the inductive bias of recurrent neural networks and added adaptive computational time to dynamically determine the number of processing steps required for each symbol. Due to the fact that these innovations build on a shared framework, it is plausible that they could be combined. It remains unclear, however, whether these innovations individually or in combination will provide meaningful performance gains in reinforcement learning.

Chapter 3

Methodology

The research aim of this dissertation was to evaluate transformers as memory systems in reinforcement learning. Specifically, we sought to address the research questions and hypotheses defined in chapter 1. As outlined in chapter 2, recurrent neural networks have in recent history been the most widely adopted mechanism for providing memory in reinforcement learning. However, Transformers present a promising and viable alternative. Although the Canonical Transformer has been shown to perform poorly in a reinforcement learning setting, there have been several extensions to the Canonical Transformer in recent years that remain unevaluated. Furthermore, it is unclear whether these extensions could fruitfully be combined into a single Integrated Transformer model. In this chapter we justify and detail the experiments that were designed to address these research questions and hypotheses.

We begin with a detailed overview of the environments each memory model was trained and evaluated on in section 3.1. We then move on to discuss the advantage actor-critic algorithm in section 3.2, which was used as the baseline reinforcement learning algorithm for all experiments. The proceeding sections discuss how the Transformer-based models were adapted to reinforcement learning (section 3.3) and the hyper-parameters and architecture of each memory model (section 3.4). The penultimate section 3.5 presents the Integrated Transformer, which is a novel combination of the Gated Transformer-XL, ReZero and the Universal Transformer. Finally, the implementation of these models in a single open-source library is discussed in section 3.6.

3.1 Environments

Each memory model was trained and evaluated using the memory chain environment from **bsuite** (Osband et al., 2020), which is a stylized T-Maze problem inspired by similar experiments commonly used to study the function of memory in animals, depicted in figure 3.1.1. The memory chain environment is a POMDP, in which the agent receives and must remember some context to make the correct decision at the end of the episode.

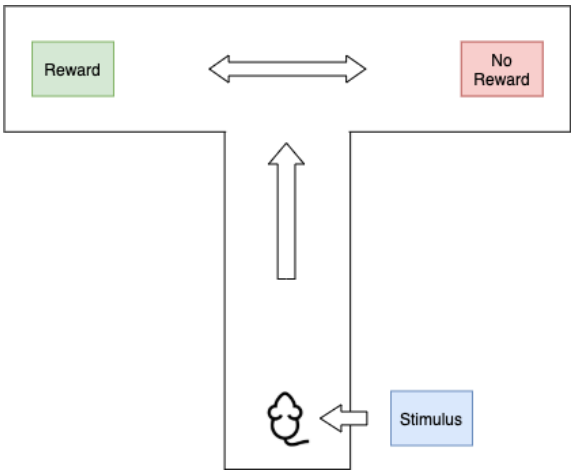


Figure 3.1.1: T-Maze as performed in animal cognition experiments. At the start of the episode the animal is exposed to some stimulus, and based on that stimulus must select whether to turn down the left or right branch of the maze. If the agent selects correctly, it will be rewarded.

Specifically, we ran three types of experiments at increasing levels of complexity: *memory length*, *memory size* and *distributed memory*. In the memory length experiments, the agent is shown a single binary context at the first time step, and must remember the context for a number of steps that is increased in length logarithmically from 1 to 100. In memory size, the length of the maze is kept fixed, but the number of bits the agent must remember is increased logarithmically from 1 to 40. We designed a third experiment which varies both the length of the T-maze and the number of bits in the context, and additionally distributes the bits throughout the episode rather than simply at a single time step. Furthermore, rather than matching the component at the end of the episode as depicted by figure 3.1.2, the agent must calculate the sum of the context bits received during the episode and then determine the correct action based on this calculation. While the memory length and memory size experiments allow a targeted comparison of very specific aspects

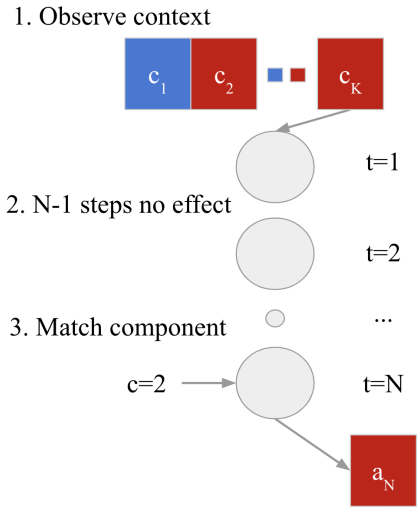


Figure 3.1.2: Visual representation of the memory chain environment as depicted in (Osband et al., 2020). At the start of the episode the agent observes a binary context vector c of length k. During the subsequent time steps, $t = 1...N-1$, the agent's actions have no effect. At the last time step, $t = N$, the agent is supplied with a scalar query $q < K$, which the agent must use as an index $c[q]$ to select the right action $\in \{0, 1\}$.

of memory, we felt the distributed environment was necessary as it mapped closer to the dynamics of many real world settings, whereby important information is often spread throughout the episode. A detailed breakdown of the configuration of each environment is available in Appendix B.

3.2 Advantage Actor-Critic

The advantage actor-critic algorithm (Mnih et al., 2016) was selected as the baseline reinforcement learning method on which to compare the various memory models. As outlined in section 2.1.5, actor-critic algorithms are a class of policy gradients methods that optimize both a policy (actor) and a value function (critic). The learned value function typically operates as a baseline to reduce the variance of the policy gradient update estimates. The algorithm updates the policy $\pi(a_t \mid s_t, \theta)$ and estimate of the value function $V(s_t, w)$ every t_{max} actions or until a terminal state is reached. The update performed by the algorithm can be seen as $\nabla_{\theta'} \log \pi\left(a_t \mid s_t, \theta'\right) A\left(s_t, a_t, \theta, w\right)$, where $A\left(s_t, a_t, \theta, w\right)$ is an estimate of the advantage function (equation 2.12). The pseudocode is given in algorithm 1.

The advantage actor-critic algorithm was designed specifically with modern com-

puting hardware in mind, and facilitates training multiple agents in parallel. There are two versions of the algorithm: asynchronous (A3C) and synchronous (A2C). The asynchronous version is designed to be run on a single multi-core machine, whereby multiple actor-learners are trained in parallel using independent CPU threads. Each thread-specific agent interfaces with a set of global network parameters independently. At the time of publication, A3C achieved state-of-the-art results on the Atari domain (Mnih et al., 2016). A2C[1] in contrast is a synchronous, deterministic version of A3C which is able to utilize GPUs more efficiently while achieving the same if not better performance (Wu et al., 2017). A2C uses a coordinator to wait for all the parallel actor-learners to complete before updating the global parameters synchronously. The advantage of this synchronous update is that all the thread-specific policies operate on the same version of the update, which helps keep training cohesive.

Algorithm 1 Asynchronous Advantage Actor-Critic (Mnih et al., 2016)

// Assume global parameters vectors θ *and* w*, and global shared counter* $T = 0$
// Assume thread-specific parameter vectors θ' *and* w'*.*
Initialize time step $t = 1$.
while $T \leq T_{max}$ **do**
    Reset gradient: $d\theta = 0$ and $dw = 0$.
    $t_{start} = t$
    Sample starting state s_t
    while $st \neq terminal$ and $t - t_{start} \leq t_{max}$ **do**
        Perform actions a_t according to policy $\pi(a_t \mid s_t, \theta')$
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, w') & \text{for non-terminal } s_t \end{cases}$
    for $i = t - 1, ..., t_{start}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients w.r.t $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i \mid s_i, \theta')(R - V(s_i, w'))$
        Accumulate gradients w.r.t $w' : dw \leftarrow dw + \partial (R - V(s_i, w'))^2 / \partial w'$
    Perform asynchronous update of θ using $d\theta$ and w using dw.

A2C was selected for this dissertation as it is lightweight, delivers excellent performance and can easily be adapted to utilize an external memory system such as a LSTM or Transformer. Moreover, **bsuite** includes a implementation of A2C as part of the package, which helped inform hyper-parameter selection and the design of the

[1] A3C stands for asynchronous advantage actor-critic. The name A2C is derived from the fact that the first A for asynchronous is dropped.

neural network architectures (section 3.4). Other deep reinforcement learning algorithms, such as the Deep Q-Learning (Mnih et al., 2015), could similarly be adapted to include a recurrent or transformer-based memory mechanism (Hausknecht and Stone, 2017).

3.3 Adapting Transformers for Reinforcement Learning

The Transformer architecture was adapted for reinforcement learning tasks in several ways. Firstly, the encoder was exclusively used as opposed to the full encoder-decoder architecture. In a language modeling setting, the encoder maps the input sequence to a sequence of continuous representations, which are masked and passed into the decoder. The masking ensures that the decoder can only model dependencies from left-to-right, which is not necessary in an online reinforcement learning setting, whereby the agent receives a single observation at each time step and has no way of looking ahead.

The second modification to the transformers was the addition of memory, inspired by the Transformer-XL. Initially, we attempted to adapt the environments using a sliding window approach, whereby a sequence of observations of *window length* were received by the agent rather than a single observation at each time step. This approach is, however, less generalizable and ideally the memory systems should be adapted to the environments rather than the other way round. Consequently, the same hidden state reuse mechanism and relative positional encoding from the Transformer-XL was added to each Transformer model. As outlined in section 2.2.3, the attention mechanism is permutation-invariant and the positional encoding is used to infer information about the order of the sequence. In a reinforcement learning context, this allows the agent to order past experience along the access of time and distinguish between observations based on the time step in which they occurred. Pseudocode that details how the Transformer and LSTM networks were adapted to A2C is available in algorithm 2.

Algorithm 2 Pseudocode for the forward pass of A2C with a memory network (Transformer/LSTM).

1: $y = MultiLayerPerceptron(x)$
2: $r = y$
3: $y = MemoryNetwork(y) + r$
4: $policy\_logits = PolicyNetwork(y)$
5: $value = ValueNetwork(y)$

3.4 Hyper-Parameter Selection and Memory Architectures

For all experiments, the hyper-parameters and model architectures were modeled as closely to the benchmark algorithms packaged with **bsuite** as possible. This design decision was taken to avoid the immense computational and time costs associated with sophisticated hyper-parameter tuning and architecture search methods. Instead, resources were focussed on exploring a range of different Transformer models and prototyping novel Transformer architectures.

In line with these principles, the advantage actor-critic consisted of a shared network that was 2 layers deep and 64 neurons wide, with a single network head for the value and policy distribution respectively. Adam (Kingma and Ba, 2017) was chosen as the optimization algorithm with the default learning rate 0.001. As outlined in section 2.1.3, Adam is computationally efficient, appropriate for problems with non-stationary objectives and noisy gradients and requires very little hyper-parameter tuning. A discount factor $\gamma = 0.99$ was selected for all environments.

In terms of the memory networks, a single-layer LSTM with 64 features in the hidden state was utilized, which is the same architecture that was used as a baseline in **bsuite**. Each Transformer architecture consisted of a single submodule, with a single attention head and a single feedforward layer with a width of 32 neurons. The memory length was set to a maximum of 100, to ensure the context was in range for all experiments. For the Universal and Integrated Transformer models, the maximum number of adaptive computational time steps was set to 6 and the halting threshold 0.9, which are the default values presented by Dehghani et al. (2019). The hyper-parameters for the memory systems are summarized in table 3.1.

Each algorithm was trained on every experiment for 10000 episodes and then

Model	# Layers	Head Dim.	# Heads	Hidden Dim.	Memory Size	# Param.
LSTM	1	-	-	64	-	42051
Transformer-XL	1	64	1	32	100	33891
GTrXL	1	64	1	32	100	83811
ReZero	1	64	1	32	100	33636
Universal Transformer	1	64	1	32	100	34020
Integrated Transformer	1	64	1	32	100	83685

Table 3.1: Hyper-Parameters for each memory system.

evaluated for 100 episodes on three randomly selected seeds $\in 4, 5, 10$. Additionally, to enable statistical comparison tests, two memory length experiments and two memory size experiments were run using 30 seeds. A total of 1828 experiments were completed in this study. The performance of each model was calculated for both training and evaluation by computing a rolling average of the return received by the agent over the last 100 episodes.

3.5 The Integrated Transformer

As outlined in section 2.3.2, there is a history of combining independent innovations in modern neural network research. Inspired by this fact, an additional novel Transformer architecture was trained and evaluated on **bsuite**. The Transformer combines the Gated Recurrent Units from the gated Transformer-XL (Parisotto et al., 2019), the zero-initialized residual weighting from ReZero (Bachlechner et al., 2020) and the adaptive computational time mechanism from the Universal Transformer (Dehghani et al., 2019; Graves, 2017). The final Integrated Transformer submodule is illustrated by figure 3.5.1.

The pseudocode for the Transformer is presented in algorithm 3. The variable h corresponds to the halting probability of the adaptive computational time mechanism, while t represents the number of processing steps. The variable ζ is the weighted residual connection outlined in section 2.2.6. For the purpose of readability and simplicity, the adaptive computational time mechanism has been presented in a simplified format. More details of this mechanism are available in Appendix B.

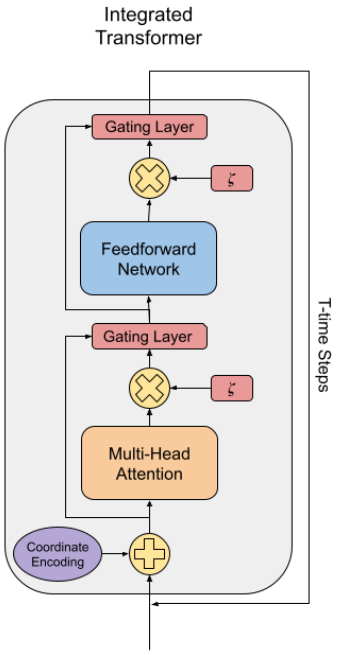


Figure 3.5.1: Visual representation of the Integrated Transformer. The gating layer is derived from Parisotto et al. (2019), the coordinate encodings and adaptive computational time mechanism from Dehghani et al. (2019); Graves (2017) and the residual weighting ζ from Bachlechner et al. (2020).

3.6 Building the Transformer Library

A significant undertaking of this dissertation involved developing a library of transformers designed specifically for reinforcement learning. As outlined in chapter 1, the majority of open source Transformer implementations are developed specifically for Natural Language Processing tasks and cannot easily be ported to reinforcement learning. There are currently very few, if any, reputable open source implementations of transformers suitable for reinforcement learning. Consequently, all five Transformer models examined in this dissertation were developed from scratch.

The Transformer library was implemented in Python and made extensive use of the PyTorch machine learning library. PyTorch was selected for several reasons. Firstly, it is free and open-source. Secondly, it is fast and integrates easily with GPUs. Thirdly, it is intuitive to use and provides powerful debugging features. Finally, many publicly available Transformer implementations were built using PyTorch.

Developing the Transformer library was a demanding software engineering task,

Algorithm 3 Pseudocode for the Integrated Transformer.

$h = 0$
$t = 0$
while $h \leq h_{max}$ and $t \leq t_{max}$ **do**
    $y = \text{RelativeMultiHeadAttention}(x)$
    $y = y \times \zeta$
    $y, hidden\_state\_1 = \text{GRU}(x + y, hidden\_state\_1)$
    $x = y$
    $y = \text{FeedForward}(y)$
    $y = y \times \zeta$
    $y, hidden\_state\_2 = \text{GRU}(x + y, hidden\_state\_2)$
    Compute halting probability h for each symbol in the sequence
    $t = t + 1$

with a strong emphasis on designing each component to be modular and loosely coupled. Designing the library according in this way ensured that the different Transformer models could easily be interchanged and combined into a single Integrated Transformer. In order to encourage replication, continued exploration and transparency, the code and hyperparameters for each model has been made publicly available: https://github.com/TomMakkink/transformers-for-rl.

Chapter 4

Results

This chapter details the performance of the five Transformer models and the LSTM baseline on the memory length, memory size and distributed memory experiments outlined in chapter 3. The memory length experiments (section 4.1) were designed to evaluate the scaling properties of the memory models along the axis of time, whereby the agent must retain information over an increasing number of time steps. The memory size experiments (section 4.2) in contrast evaluated each memory models ability to scale with the quantity of information that needed to be retained. The distributed memory experiments (section 4.3) are a hybrid of the memory length and size experiments, increasing both time and quantity of information retained but distributing the information over a number of time steps rather than at a single one. Additionally, for each experiment the convergence speed, stability and seed sensitivity of each model during training was examined. A summary of the evaluation performance of each model per experiment type is presented in table 4.1 and the full set of evaluation results for all the experiments is available in Appendix C.

4.1 Memory Length

The memory length experiments required the agent to remember a single binary context (received at the first time step) for a number of time steps that was increased logarithmically from 1 to 100. The Gated Transformer-XL, LSTM and the Integrated Transformer were the top three performing models on these experiments, followed by the Universal Transformer, ReZero and finally the Transformer-XL, as

Model	Memory Length	Memory Size	Distributed Memory
No Memory	0.043 ± 0.076	0.015 ± 0.086	0.245 ± 0.025
LSTM	0.832 ± 0.113	$\mathbf{0.310} \pm 0.086$	$\mathbf{0.649} \pm 0.187$
Transformer-XL	0.292 ± 0.079	0.044 ± 0.126	0.381 ± 0.144
GTrXL	$\mathbf{0.856} \pm 0.225$	0.150 ± 0.169	0.443 ± 0.201
ReZero	0.441 ± 0.399	0.061 ± 0.129	0.327 ± 0.128
Universal Transformer	0.645 ± 0.317	0.083 ± 0.147	0.335 ± 0.139
Integrated Transformer	0.721 ± 0.290	0.189 ± 0.133	0.392 ± 0.118

Table 4.1: Mean evaluation return and standard deviation for each model per environment type over three random seeds.

depicted in table 4.1. The high level of variance across the Transformer models, however, makes it difficult to discern whether these results would hold over a larger set of randomly selected seeds. The distribution of evaluation results, illustrated in figure 4.1.1, indicate that this variance was likely caused by each model either learning successfully and solving the environment or failing to learn at all.

Most modern reinforcement learning algorithms are notoriously sensitive to hyperparameter and initial seed selection, which makes it a challenging domain to conduct research in. In ideal circumstances, each experiment would have been repeated using at least 30 different randomly selected seeds to enable statistical comparison testing. Although **bsuite** is far less computationally expensive than benchmarks such as DMLab-30, running the full suite of memory length experiments for every memory model is still prohibitively costly. For example, a single run of either the Universal or Integrated Transformer on all 23 memory length experiments can take up to 3 days to complete on a single GPU. For the same set of experiments the Gated Transformer-XL, ReZero and Transformer-XL average run times of 24 hours. Running 7 memory models on all 23 memory length experiments over 3 randomly selected seeds already entailed running 483 experiments in total. Given the compute and time available, it was simply not feasible to run every experiment using 30 random seeds. Instead, two memory length experiments were selected to be run over 30 random seeds: context lengths 20 and 70.

These context lengths were selected for two reasons. Firstly, lengths 20 and 70 provide good approximate representations of short-term and long-term temporal dependencies, which allows the scaling properties of the models to be examined. Secondly, both these context lengths represented a point of distinct interest in the

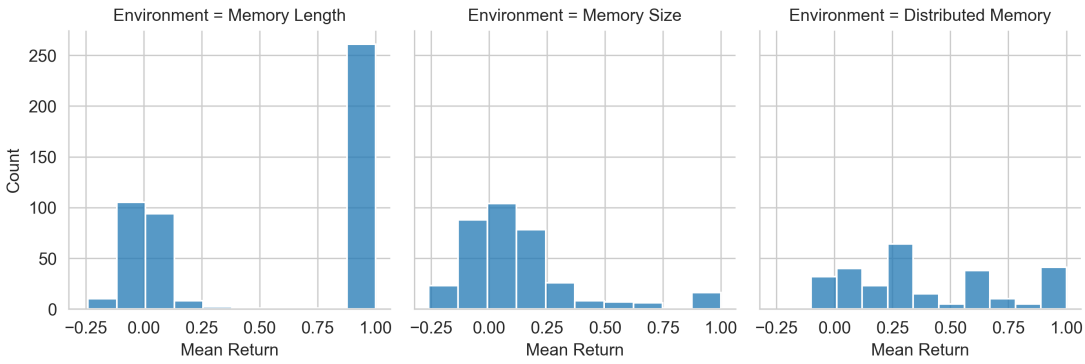


Figure 4.1.1: The distribution of the evaluation results for the memory length, memory size and distributed memory experiments plotted as a histogram with 10 bins. The number of experiments run per experiment type for all 7 memory models over 3 seeds was 483 for memory length, 357 for memory size and 273 for distributed memory. The total number of experiments run was 1113.

results over three seeds gathered thus far, as illustrated by figure 4.1.2. The performance of the LSTM and Transformer models was similar for the shorter context lengths below 20. At context length 20, however, the LSTM began to outperform the Transformer models. Similarly, at context length 70 the Gated Transformer-XL, Universal Transformer and Integrated Transformer outperformed the LSTM.

4.1.1 Scaling with Memory Length

The evaluation results for the memory models on context lengths 20 and 70 are illustrated in figure 4.1.3. For a context length of 20, the LSTM, Gated Transformer-XL and Integrated Transformer were the top three performing models. An independent two-sample t-test was performed to determine whether there was a statistically significant difference between the two Transformer models in comparison to the LSTM baseline. The test yielded a p-value of 0.459 for the Gated Transformer-XL and LSTM comparison. For the standard significance level of $\alpha = 0.05$, the null hypothesis of identical average scores could not be rejected, confirming a statistically insignificant difference between the two models. Similarly, there was a insignificant difference between the LSTM and the Integrated Transformer, with a p-value of 0.733. These results suggest that for reinforcement learning tasks with short-range temporal dependencies, the performance of the LSTM, Gated Transformer-XL and Integrated Transformer is similar.

The Universal Transformer and ReZero performed moderately well on context

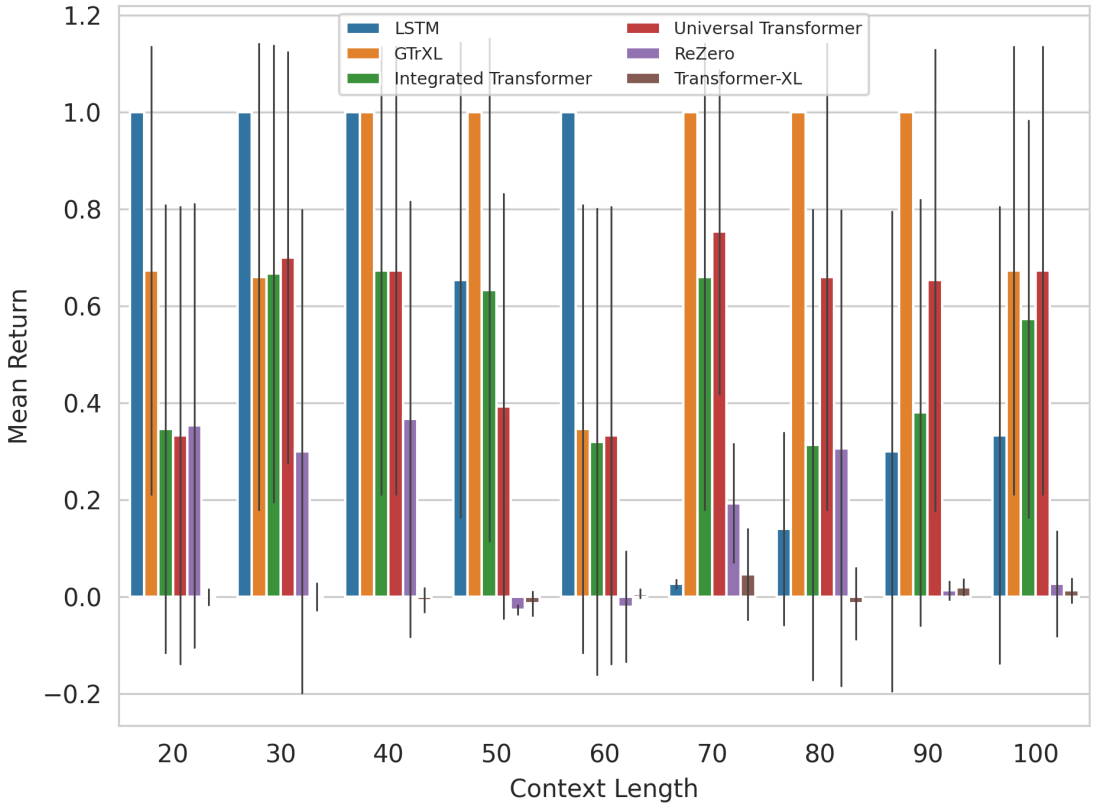


Figure 4.1.2: The mean evaluation scores averaged over 3 seeds for the memory length experiments for context lengths 20 to 100. The error lines on each bar indicate the standard deviation.

length 20, but were still below the LSTM baseline. The Transformer-XL in contrast, failed to learn on both context lengths 20 and 70, with final performance similar to a random policy.

When modeling long-term temporal dependencies on context length 70, the Gated Transformer-XL, Integrated Transformer and Universal Transformer outperformed the LSTM baseline. The performance of the Gated Transformer-XL is consistent with work conducted by Parisotto et al. (2019) that demonstrated the Gated Transformer-XL scaled better than the LSTM with temporal length. Surprisingly, the Universal Transformer performed far better when modeling long-term temporal dependencies as opposed to short-term dependencies. The inverse was true of ReZero, which scaled poorly on the longer context length.

An independent two-sample t-test was performed to determine whether there was a significant difference between the three top performing Transformer models. The comparison between the Universal Transformer and the GTrXL yielded a p-value

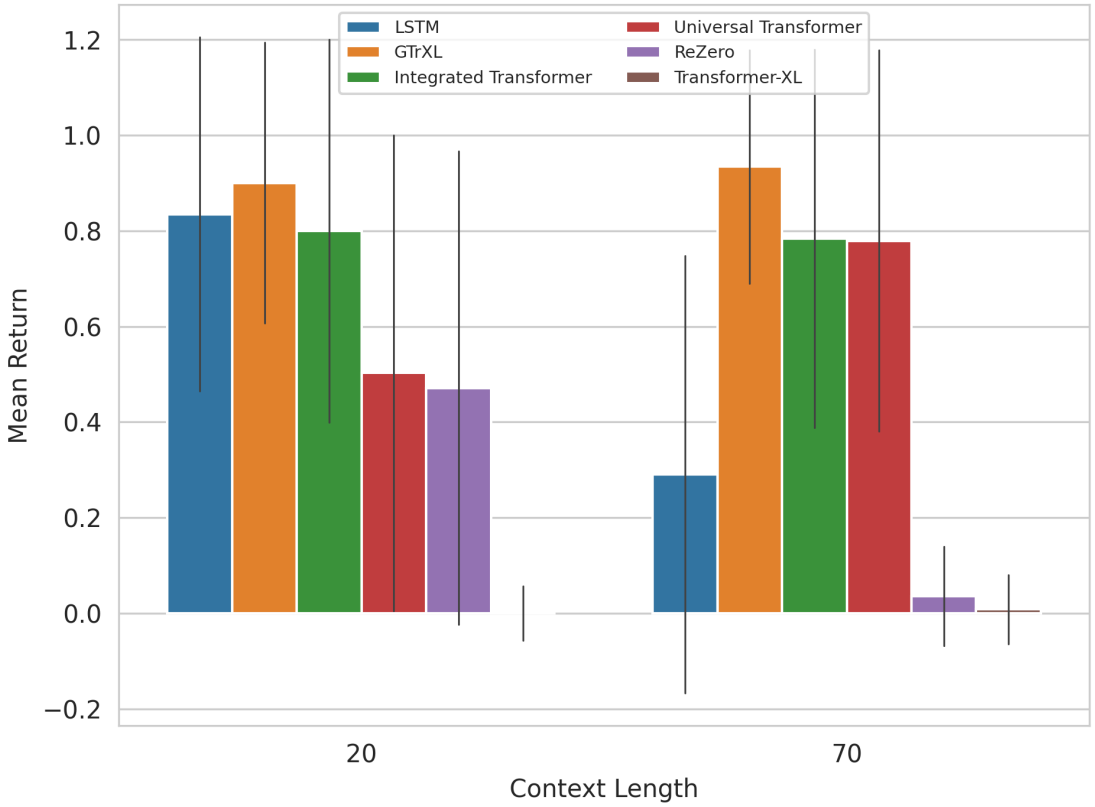


Figure 4.1.3: The mean evaluation scores averaged over 30 seeds for the memory length experiments with context lengths 20 and 70. The error lines on each bar indicate the standard deviation.

of 0.081, which indicated a statistically insignificant difference between these two results. The same conclusion was drawn between the Integrated Transformer and GTrXL, with a p-value of 0.087. Consequently, all three of these models exhibited similar performance when modeling longer-term temporal dependencies.

4.1.2 Training and Seed Sensitivity

The training curve for all memory models on context lengths 20 and 70 is illustrated by figure 4.1.4. For both context lengths, the Gated Transformer-XL converged faster during training than any other model. The performance of the Gated Transformer-XL did show a gradual decrease after converging. The training curve for the Integrated Transformer and LSTM were very similar for context length 20, displaying similar convergence speed and stability. At context length 70, however, the Integrated Transformer learns far more effectively than the LSTM. The Universal Transformer was more unstable on context length 20 in comparison to length 70,

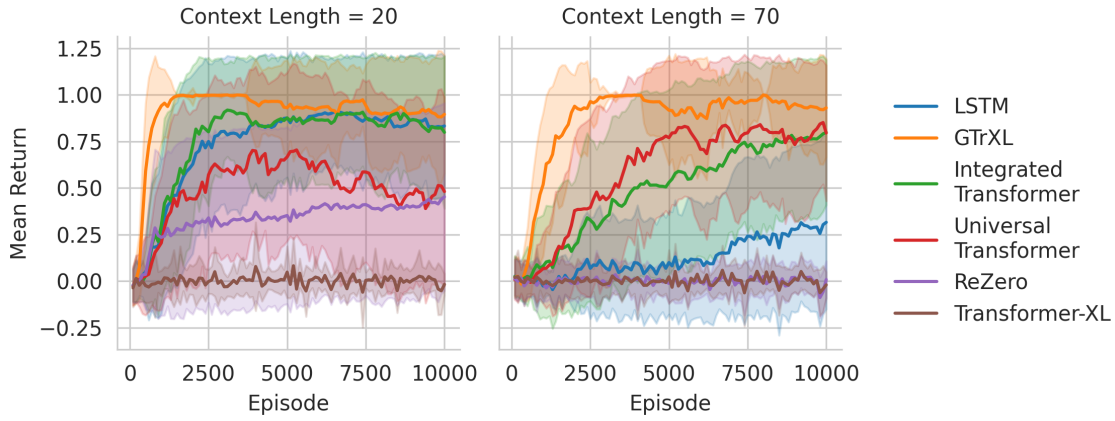


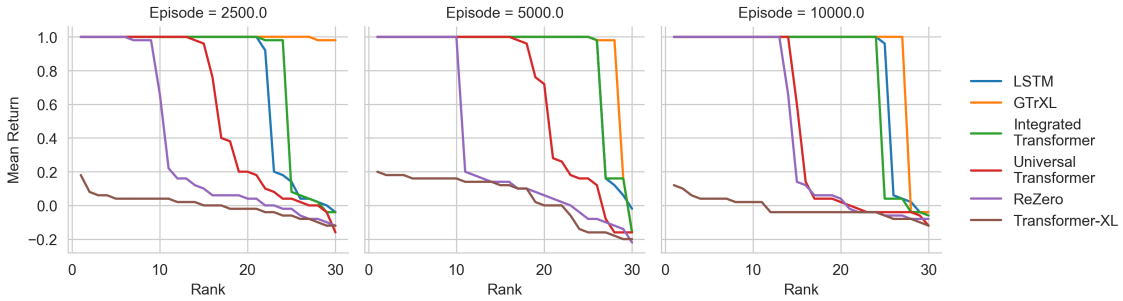
Figure 4.1.4: The training curves for all memory models on memory length experiments with context lengths 20 and 70. The shaded region represents the standard deviation across all 30 seeds.

achieving a maximum score of approximately 0.7 at episode 5000 then dropping off for the remaining training episodes.

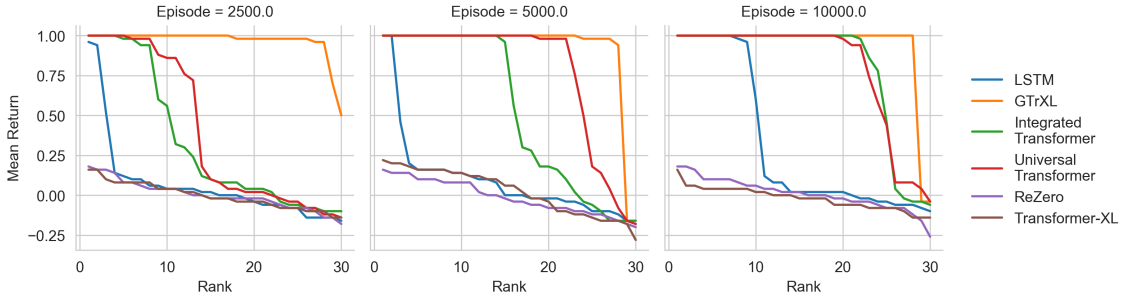
A seed sensitivity analysis, depicted by figure 4.1.5, revealed that the Gated Transformer-XL was the most robust model to initial seed selection. Furthermore, the Gated Transformer-XL was most stable during the early stages of training, such as episode 2500. The Integrated Transformer and LSTM baseline exhibited similar seed sensitivity during training on context length 20, but the Integrated Transformer was far more robust to seed selection on context length 70. The inverse was true of the Universal Transformer, which was less robust to seed selection than the LSTM on the shorter context length but better on the longer context length.

4.2 Memory Size

The memory size experiments required the agent to retain an increasingly large context (scaled from 1 to 40 logarithmically) while keeping the length of the maze fixed. These experiments were designed to evaluate how each memory model scaled with an increasing quantity of information that must be retained. The LSTM was the top performing model on the memory size experiment type, achieving a mean score of 0.310 across all 17 experiments. The top performing Transformer models were the Integrated Transformer and the Gated Transformer-XL with mean scores of 0.189 and 0.150 respectively. The Universal Transformer, ReZero and Transformer-



(a) Context Length 20.



(b) Context Length 70.

Figure 4.1.5: Seed sensitivity analysis for (a) context length 20 and (b) context length 70. The performance of the LSTM, ReZero, the Integrated Transformer, GTrXL and Universal Transformer are depicted during training at three intervals: episodes 2500, 5000 and 10000. The 30 seeds are ranked in descending order according to the models' performance. Higher and flatter lines indicate more robust architectures.

XL in contrast performed extremely poorly, yielding mean returns of less than 0.1 as detailed in table 4.1.

The evaluation results for the top three performing models on the memory size experiments are presented in figure 4.2.1. There is a broad trend of the LSTM scaling better with context size than the Gated Transformer-XL and the Integrated Transformer. However, there is also clearly some noise in these results, which is likely caused by the high levels of variance across the 3 seeds.

In order to further explore this trend and enable statistical comparison tests, the same strategy was employed as detailed in the previous section, whereby two experiments were trained and evaluated over 30 random seeds: context sizes 5 and 10. The criteria for selecting these two experiments was very similar to selecting context lengths 20 and 70, whereby these two sizes provided insights into the scaling properties of the models and represent contentious points in the results averaged over three seeds. For example, for both context sizes 5 and 10 the mean score of the

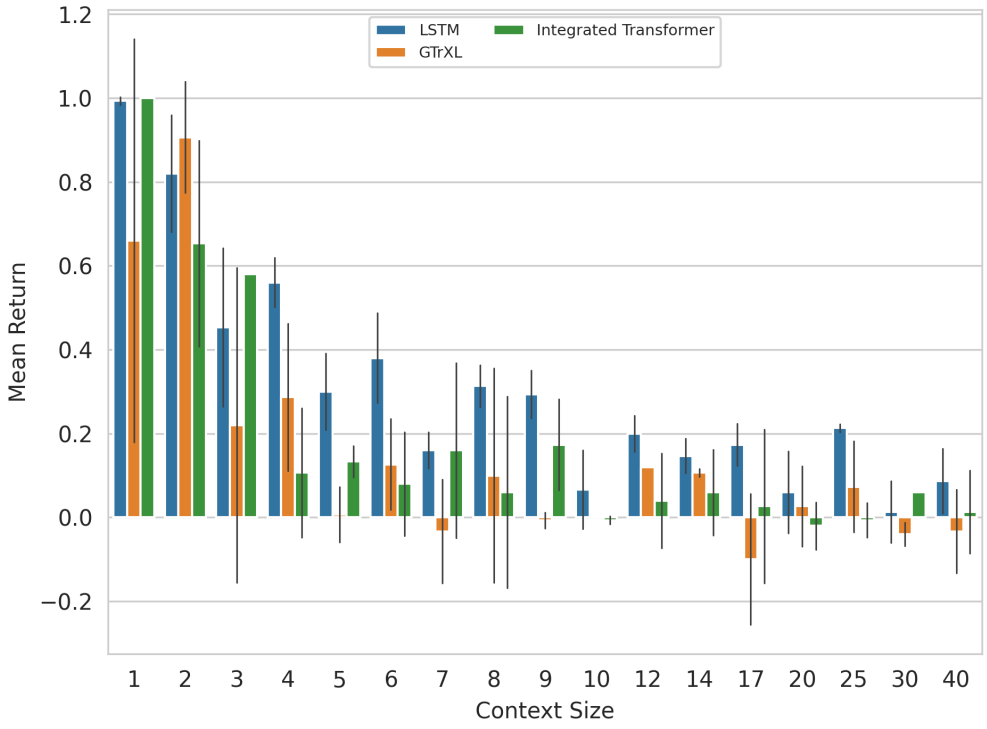


Figure 4.2.1: The mean evaluation return for the LSTM, Gated Transformer-XL and Integrated Transformer on the memory size experiments. The error-bars indicate the standard deviation.

LSTM is greater than the Integrated Transformer and Gated Transformer-XL, as illustrated by figure 4.2.1. Given the standard deviation of these results, however, it is not clear if this trend would translate to a larger pool of seeds.

4.2.1 Scaling with Memory Size

Figure 4.2.2 shows the results for context sizes 5 and 10 averaged over 30 seeds. These results clearly show that the Universal Transformer, ReZero and Transformer-XL perform poorly on both context sizes. An independent two-sample t-test was performed to evaluate whether there was a statistically significant difference between the performance of the LSTM and the Integrated Transformer. For context size 5, the comparison between the LSTM and Integrated Transformer yielded a p-value of 0.002, which means that the null hypothesis was rejected and a significant difference did exist. Similarly, for context size 10 there was a significant difference between the LSTM and Integrated Transformer with the comparison test yielding a p-value of 0.001. Consequently, the results from the memory size experiments demonstrated

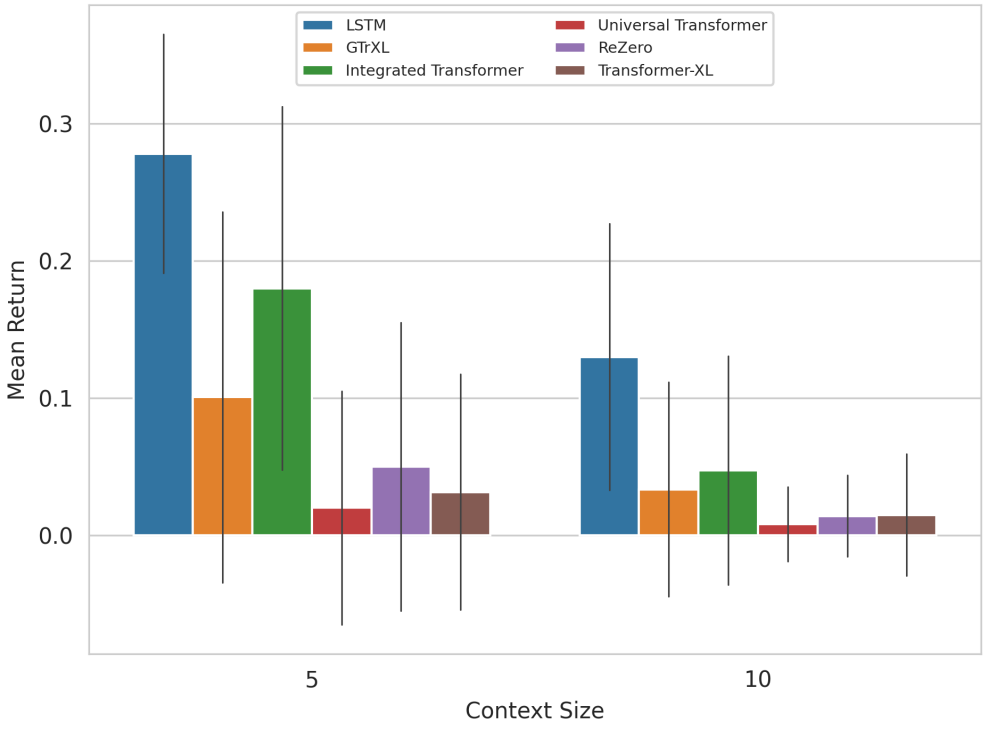


Figure 4.2.2: The mean evaluation scores averaged over 30 seeds for the memory size experiments with context sizes 5 and 10.

that the LSTM scaled better than all the Transformer models with memory size.

4.2.2 Training and Seed Sensitivity

All five Transformer models and the LSTM baseline were unstable during training. As illustrated by figure 4.2.3, the mean return for each model fluctuated throughout the training episodes. From this figure, it was clear that the LSTM learned more effectively than the Transformer models. Furthermore, the LSTM was the most robust to seed selection throughout all stages of training on both context sizes 5 and 10, as depicted in the seed sensitivity analysis in figure 4.2.4. Of the Transformer models, the Integrated Transformer and Gated Transformer-XL were the most robust to seed selection on context length 5, particularly during the later stages of training. The other three Transformer models in contrast were extremely sensitive to seed selection in this experiment. All five Transformer models scaled poorly from context size 5 to 10 in terms of robustness to seed selection.

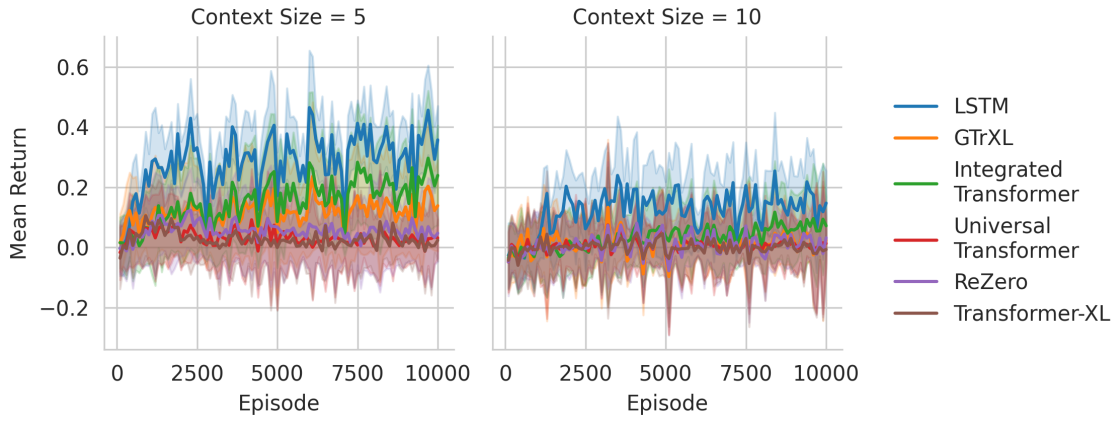
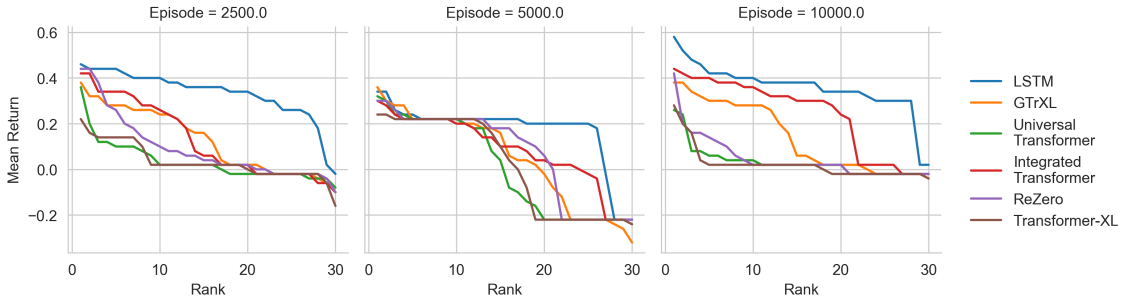


Figure 4.2.3: The training curves for all memory models on memory size experiments with context sizes 5 and 10.

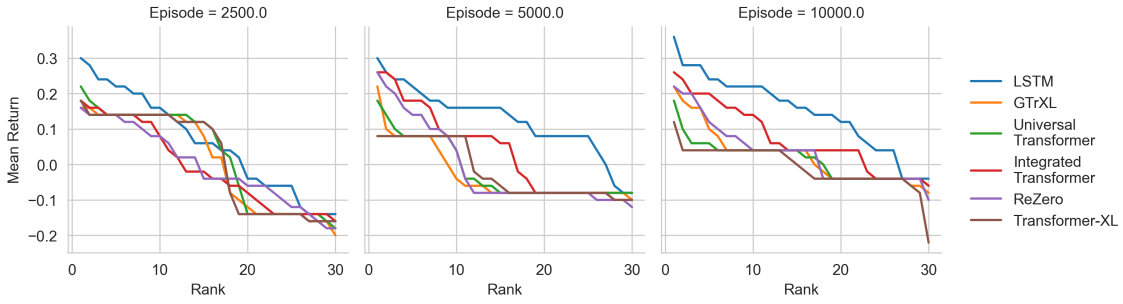
4.3 Distributed Memory

The purpose of the distributed memory environment was to provide a set of experiments that modeled real world problems more closely than the memory size and length experiments. As such, the context received by the agent was distributed throughout the episode rather than at a single time step. The LSTM was the top-performing memory model overall on this environment with a mean evaluation return of 0.649, followed by the Gated Transformer-XL, Integrated Transformer, Transformer-XL, Universal Transformer and finally ReZero. The mean results across all experiments in this environment are included in table 4.1.

The strong performance of the LSTM was due to its ability to scale to larger memory sizes in comparison to the Transformer models. For example, there was a clear trend of the LSTM outperforming the Transformer models on context sizes greater than 5, as illustrated by figure 4.3.1. In contrast, the Gated Transformer-XL and Integrated Transformer performed relatively well on the experiments with smaller context sizes, such as 3 and 5. Given the poor performance of the Transformer-XL, Universal Transformer and ReZero on the memory size tasks in the previous section, it is not surprising these models performed so poorly on the distributed memory experiments.



(a) Context Size 5.



(b) Context Size 10.

Figure 4.2.4: Seed sensitivity analysis for (a) context size 5 and (b) context size 10.

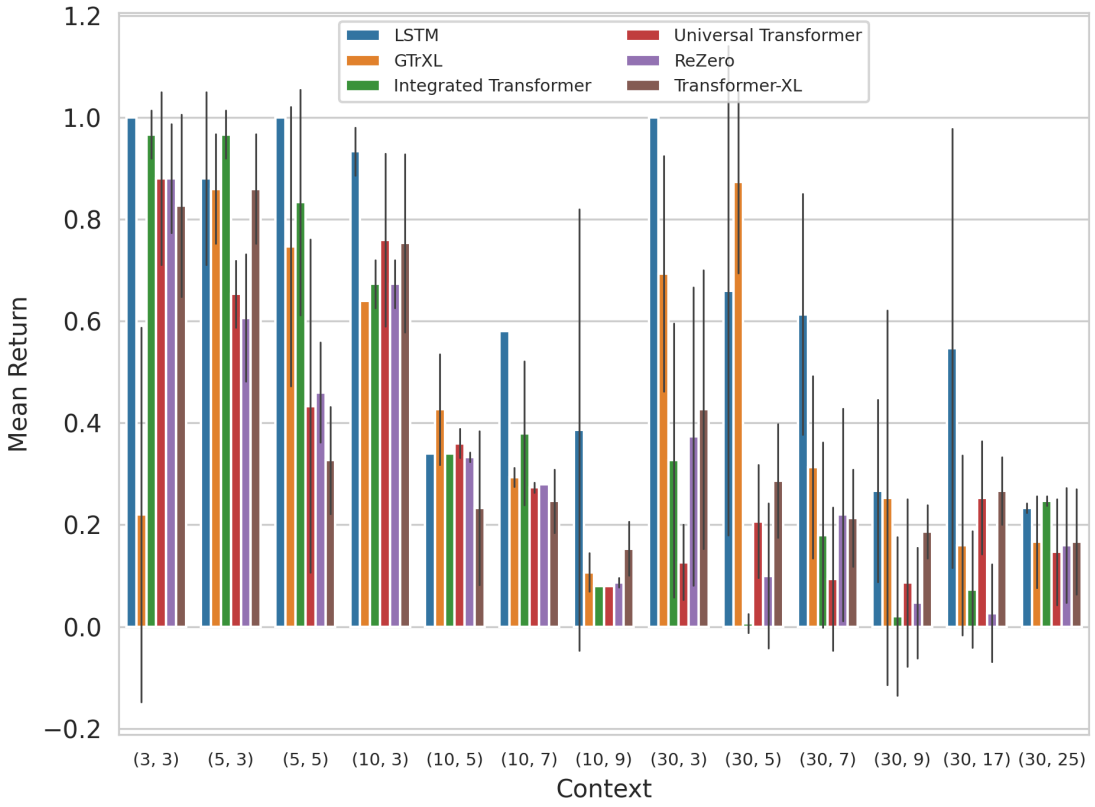


Figure 4.3.1: The mean return of all memory models on the distributed memory experiments. The context is a tuple which represents the context length as the first element and the context size as the second.

Chapter 5

Discussion

This chapter discusses the results presented in chapter 4 in relation to the current literature on memory systems in reinforcement learning as well as the research questions and hypotheses outlined in chapter 1. We begin by discussing the scaling properties of the five Transformer models in relation to the LSTM baseline on the memory length experiments. We then proceed to discuss the results of memory size and distributed memory experiments.

5.1 Scaling with Temporal Dependencies

The results of the memory length experiments detailed in section 4.1 clearly demonstrate that the Transformer-based architectures outperform the LSTM when modeling long-range temporal dependencies. The difficulty of optimizing recurrent neural networks such as the LSTM on problems with long-range dependencies has been well-established for many years (Bengio et al., 1994). The results outlined in section 4.1.1 strongly reflect this, whereby the performance of the LSTM decreases as the context length increases due to the vanishing/exploding gradient problem outlined in section 2.2.1.

The excellent performance of the Gated Transformer-XL and Integrated Transformer on the memory length experiments is consistent with work conducted by Parisotto et al. (2019), which demonstrated that the addition of Gated Recurrent Units to the Transformer submodule drastically improves and stabilizes performance in reinforcement learning. Furthermore, these results support the hypothesis that

gated variants of the Transformer scale to long-term temporal dependencies more effectively than the LSTM (Parisotto et al., 2019). Moreover, the Gated Transformer-XL converged faster than the other memory models during training and was extremely robust to seed selection, particularly in the early stages of training when the model had just converged. In light of this finding, a strategy of early stopping may be useful when training the Gated Transformer-XL in reinforcement learning, whereby training concludes once a certain performance threshold has been reached or once performance has begun to plateau.

Although the Integrated Transformer performed well on the memory length experiments, there are several drawbacks to this model in comparison to the Gated Transformer-XL. Firstly, it requires more compute and takes far longer to train. Secondly, the Integrated Transformer takes longer to converge during training. Finally, the Integrated Transformer is more sensitive to random seed selection than the Gated Transformer-XL. Given these three drawbacks and the similar performance of the Gated Transformer-XL and Integrated Transformer on the memory length tasks, it is unlikely that the Integrated Transformer would be employed over the Gated Transformer-XL in practice.

There are several hypotheses as to why the Integrated Transformer did not perform on par with the Gated Transformer-XL. For example, given the poor performance of ReZero across all experiments it is likely that the weighted residual connection actually detracted from overall performance rather than improving it. Furthermore, the Transformer models used in these experiments were composed of a single layer and a single attention head. One of the major benefit of ReZero is improved signal propagation through the Transformer, particularly at greater depth. It is unclear whether the weighted residual connections may have improved performance with much deeper Transformer models on more complex tasks. It is also possible that more extensive hyper-parameter tuning could have improved the convergence speed of the Integrated Transformer. Given the finite amount of time and compute, a decision was made to use the default set of hyperparameters outlined in section 3.4 and focus on running a large number of experiments to enable statistical testing. The use of more sophisticated hyper-parameter tuning could have a large influence on model performance. Further work is required to validate these

hypotheses.

The poor performance of the Transformer-XL on the memory length experiments is consistent with the studies conducted by Parisotto et al. (2019) and Mishra et al. (2018) that found pure attention look ups to be insufficient for modeling dependencies in reinforcement learning. The fact that ReZero outperforms the Transformer-XL on the memory length experiments does, however, suggest that improved signal propagation in the Transformer can help improve performance. Despite this improvement ReZero was still well below the performance of the LSTM and did not scale to the longer context lengths.

The performance of the Universal Transformer on the memory length experiments was very surprising. Longer context lengths are expected to be more difficult to model than shorter context lengths. However, the performance of the Universal Transformer was the inverse of this, performing poorly on shorter context lengths and well on longer context lengths. It is unclear why this is. Examining the training curves in figure 4.1.4, on context length 20 the performance of the Universal Transformer was very unstable and decreased after episode 5000. On the context length 70, however, the performance stabilized after episode 5000. Given the seed sensitivity of this model and high variance in the results, it may be beneficial to run this model on a larger pool of random seeds. This task would be no small feat as the adaptive computational time mechanism increases the computational demands and length of each experiment immensely. The Universal and Integrated Transformer take approximately 3 times longer to train than the other Transformer models. An additional avenue of further investigation would be to track the number of processing steps per symbol in the adaptive computational time mechanism during training and compare these results across context lengths.

Overall the gated Transformer variants outperformed the LSTM baseline on the memory length experiments while the non-gated Transformers performed poorly. It must be noted that model capacity may have played a role in these results. For example, the Gated Transformer-XL and Integrated Transformer had approximately twice the number of parameters as the LSTM and non-gated Transformers. It is unclear what the relationship between model capacity and performance on memory tasks is in reinforcement learning. On the one hand, simpler architectures such as

the LSTM may be easier to integrate into a reinforcement learning agent and require less optimization. On the other hand, increasing capacity in Transformer models has empirically been demonstrated to lead to better performance and generalization but may require more sophisticated optimization (Devlin et al., 2019; Parisotto et al., 2019). Further work is required to access the role of capacity in an reinforcement learning setting.

5.2 Scaling with Information Quantity

The LSTM was the top performing model on the memory size experiments, outperforming all the Transformer-based models. Given the excellent performance of the Gated Transformer-XL and Integrated Transformer on the memory length experiments, it is unclear why these transformers performed so poorly in this setting. Furthermore, there are no studies available in the reinforcement learning literature that examine the scaling properties of Transformer-based models in terms of memory size specifically.

The ability to remember large quantities of information is an important component of memory in reinforcement learning. For example, consider the results of the distributed memory experiments depicted in figure 4.3.1. The LSTM was able to outperform the Transformer models based on the distributed experiments due to its ability to scale to larger memory sizes compared to the Transformer models. Furthermore, as the maximum context length in this environment was fairly short (30), the LSTM was able to perform approximately on par with increasing memory length.

It remains unclear, however, whether increasing the depth, width or number of attention heads would improve the performance of the transformers on the memory size experiments. In a supervised setting, given an appropriately large training set increasing the size of the model generally leads to better performance overall (Devlin et al., 2019). It is unclear whether this same trend would translate to reinforcement learning. As mentioned prior, more extensive hyper-parameter tuning could also help improve the performance of the transformers on this task.

The results from the memory length, memory size and distributed memory ex-

periments suggest that the type of memory task is a major determinant of whether to use a Transformer-based architecture or LSTM. For tasks that require information to be retained over a long period of time, gated Transformer-based models are the more appropriate choice. For tasks that require large quantities of information to be retained, however, the LSTM is still the most suitable option.

Given these findings, several potential avenues of future work present themselves. Firstly, it is important to verify if the same scaling properties of the transformers observed in these experiments translate to other more complex environments. Secondly, there is the broader question of why the Transformer models do not scale to larger memory sizes. To investigate this question, visualizing the attention distribution may be useful (Vashishth et al., 2019) as well as evaluating the transformers on other tasks that require large quantities of information to be remembered. Thirdly, there is ample opportunity to experiment with other variants of the Transformer, such as the Linformer (Wang et al., 2020) which improved the computational and memory efficiently of the Canonical Transformer. Finally, the question of how best to adapt the Transformer architecture itself to reinforcement learning is still very much open. For example, this dissertation and the work conducted by Parisotto et al. (2019) investigated the Transformer as a memory system in reinforcement learning. There are, however, many other facets of reinforcement learning whereby the Transformer may be useful, such as in an offline or model-based setting.

Chapter 6

Conclusion

This dissertation investigated the performance of transformers as memory systems in reinforcement learning. Each Transformer model was evaluated in relation to an LSTM baseline on highly targeted aspects of memory such as size and length.

The results of the experiments demonstrated the gated variants of the Transformer architecture, the Gated Transformer-XL and the Integrated Transformer, scale more effectively to longer memory lengths than the LSTM. Interestingly, while the Universal Transformer performed poorly on short context lengths, this model was able to scale more effectively to long context lengths. In terms of memory size, the LSTM outperformed all the Transformer-based models. This finding suggests that transformers are ideal for memory tasks with long-range temporal dependencies, but are not suitable for tasks that require large quantities of information to be remembered.

Although the Integrated Transformer and the Gated Transformer-XL performed well on the memory lengths experiments, the Integration Transformer takes far longer to train than the Gated Transformer-XL, takes longer to converge during training and is less robust to seed selection. Given these drawbacks, integrating the components of ReZero, the Universal Transformer and the Gated Transformer-XL does not seem to offer performance gains.

In terms of the research hypotheses of this study, the poor performance of the Transformer-XL and ReZero on both the memory length and memory size experiments suggest that the attention function is insufficient for modeling temporal dependencies in reinforcement learning, supporting research hypothesis 1. Further-

more, the fact that both the Gated Transformer-XL and Integrated Transformer outperformed the Transformer-XL and ReZero indicates that Gated Recurrent Units can improve the performance of transformers in reinforcement learning, supporting research hypothesis 2.

Our recommendations for future areas of research include but are not limited to the following: validating the results of the experiments across a larger number of random seeds, investigating the effect of increasing the number of Transformer layers and attention heads, and lastly exploring the broader question of why the Transformer does not scale to larger memory sizes.

Transformers represent an exciting opportunity to improve memory systems in reinforcement learning and are excellent candidates for tasks with long range temporal dependencies.

Appendix A

Adaptive Computational Time

The dynamic halting mechanism for the Universal Transformer based on adaptive computation time (Graves, 2017) was originally implemented in TensorFlow. We adapted this implementation to PyTorch and present the code below. In each step of the Universal Transformer the halting probabilities, remainders, numbers of updates up to that point, previous state (initialized as zeros) and a scalar hyper-parameter threshold between 0 and 1 are given. The new state of each position is then computed for each position and the new per-position halting probabilities are calculated. The Universal Transformer halts for positions that exceed the threshold, and updates the other positions in the sequence until all positions halt or the predefined maximum number of steps is reached. The full implementation of adaptive computation time is available at the following open source GitHub repository: https://github.com/TomMakkink/transformers-for-rl.

```
    while ((halting_probability < halting_threshold) & (n_updates < max_act_timesteps)):
        # Add coordinate embeddings to the state
        state = CoordinateEncoder(state)

        # Calculate probabilities based on the state
        p = Sigmoid(Linear(state))

        # Mask for inputs which have not halted yet
        still_running = (halting_probability < 1.0).float()

        # Mask of inputs which halted at this step
        new_halted = (halting_probability + p * still_running > halting_threshold).float() *
    still_running

        # Mask of inputs which haven't halted, and didn't halt this step
        still_running = (halting_probability + p * still_running <= halting_threshold).float() *
    still_running

        # Add the halting probability for this step to the halting
        # probabilities for those input which haven't halted yet
        halting_probability = halting_probability + p * still_running

        # Compute remainders for the inputs which halted at this step
        remainders = remainders + new_halted * (1 - halting_probability)

        # Add the remainders to those inputs which halted at this step
        halting_probability = halting_probability + new_halted * remainders

        # Increment n_updates for all inputs which are still running
        n_updates = n_updates + still_running + new_halted

        # Compute the weight to be applied to the new state and output
        # 0 when the input has already halted
        # p when the input hasn't halted yet
        # the remainders when it halted this step
        update_weights = p * still_running + new_halted * remainders

        state = TransformerSubmodule(state)

        # update running part in the weighted state and keep the rest
        previous_state = (state * update_weights) + (previous_state * (1 - update_weights))

        step += 1
```

Listing A.1: Adaptive Computational Time in Pytorch

Appendix B

Experiment Configurations

Experiment	Memory Length	Memory Size	Distributed Memory
1	(1, 1)	(1, 1)	(3, 3)
2	(2, 1)	(1, 2)	(5, 3)
3	(3, 1)	(1, 3)	(5, 5)
4	(4, 1)	(1, 4)	(10, 3)
5	(5, 1)	(1, 5)	(10, 5)
6	(6, 1)	(1, 6)	(10, 7)
7	(7, 1)	(1, 7)	(10, 9)
8	(8, 1)	(1, 8)	(30, 3)
9	(9, 1)	(1, 9)	(30, 5)
10	(10, 1)	(1, 10)	(30, 7)
11	(12, 1)	(1, 12)	(30, 9)
12	(14, 1)	(1, 14)	(30, 17)
13	(17, 1)	(1, 17)	(30, 25)
14	(20, 1)	(1, 20)	-
15	(25, 1)	(1, 25)	-
16	(30, 1)	(1, 30)	-
17	(40, 1)	(1, 40)	-
18	(50, 1)	-	-
19	(60, 1)	-	-
20	(70, 1)	-	-
21	(80, 1)	-	-
22	(90, 1)	-	-
23	(100, 1)	-	-

Table B.1: Configuration for the three memory chain environments: memory length, memory size and distributed memory. Each configuration is represented as a tuple, whereby the first element is the context length and the second the context size. For example, a configuration of (3, 10) translates to a context length of 3 and a context size of 10.

Appendix C

Results per Experiment

Context Length	GTrXL	Integrated	LSTM	No Memory	ReZero	Transformer-XL	Universal
1	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	1.000 ± 0.000	0.673 ± 0.566	1.000 ± 0.000	0.660 ± 0.589
2	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	-0.020 ± 0.035	-0.007 ± 0.023	0.333 ± 0.578	0.007 ± 0.023
3	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.033 ± 0.064	0.673 ± 0.566	-0.020 ± 0.000	-0.020 ± 0.000
4	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	-0.013 ± 0.122	0.673 ± 0.566	1.000 ± 0.000	0.007 ± 0.023
5	0.808 ± 0.411	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.020 ± 0.225	0.660 ± 0.589	1.000 ± 0.000	1.000 ± 0.000
6	0.660 ± 0.589	0.993 ± 0.012	$\mathbf{1.000 \pm 0.000}$	0.013 ± 0.099	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
7	$\mathbf{1.000 \pm 0.000}$	0.347 ± 0.566	$\mathbf{1.000 \pm 0.000}$	-0.100 ± 0.122	0.667 ± 0.577	1.000 ± 0.000	1.000 ± 0.000
8	0.673 ± 0.566	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.000 ± 0.092	0.700 ± 0.520	0.987 ± 0.023	1.000 ± 0.000
9	$\mathbf{1.000 \pm 0.000}$	0.673 ± 0.566	$\mathbf{1.000 \pm 0.000}$	0.060 ± 0.040	0.667 ± 0.577	0.020 ± 0.000	1.000 ± 0.000
10	0.870 ± 0.360	0.660 ± 0.589	$\mathbf{1.000 \pm 0.000}$	0.013 ± 0.070	0.660 ± 0.589	-0.027 ± 0.042	0.673 ± 0.566
12	0.660 ± 0.589	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	-0.007 ± 0.099	0.673 ± 0.566	0.333 ± 0.578	1.000 ± 0.000
14	0.660 ± 0.589	$\mathbf{1.000 \pm 0.000}$	0.673 ± 0.566	-0.040 ± 0.106	0.940 ± 0.087	0.000 ± 0.035	0.673 ± 0.566
17	$\mathbf{1.000 \pm 0.000}$	0.333 ± 0.578	$\mathbf{1.000 \pm 0.000}$	0.013 ± 0.012	0.333 ± 0.577	0.020 ± 0.040	0.660 ± 0.589
20	$\mathbf{0.900 \pm 0.298}$	0.800 ± 0.407	0.835 ± 0.376	0.020 ± 0.053	0.471 ± 0.504	0.001 ± 0.058	0.503 ± 0.506
25	0.673 ± 0.566	$\mathbf{1.000 \pm 0.000}$	$\mathbf{1.000 \pm 0.000}$	0.000 ± 0.040	0.327 ± 0.585	0.027 ± 0.142	1.000 ± 0.000
30	0.660 ± 0.589	0.667 ± 0.577	$\mathbf{1.000 \pm 0.000}$	0.027 ± 0.050	0.300 ± 0.610	0.000 ± 0.035	0.700 ± 0.520
40	$\mathbf{1.000 \pm 0.000}$	0.673 ± 0.566	$\mathbf{1.000 \pm 0.000}$	0.033 ± 0.061	0.367 ± 0.551	-0.007 ± 0.031	0.673 ± 0.566
50	$\mathbf{1.000 \pm 0.000}$	0.633 ± 0.635	0.653 ± 0.600	-0.073 ± 0.133	-0.027 ± 0.012	-0.013 ± 0.031	0.393 ± 0.537
60	0.347 ± 0.566	0.320 ± 0.589	$\mathbf{1.000 \pm 0.000}$	-0.033 ± 0.121	-0.020 ± 0.140	0.007 ± 0.012	0.333 ± 0.578
70	$\mathbf{0.934 \pm 0.248}$	0.783 ± 0.402	0.291 ± 0.465	-0.027 ± 0.012	0.036 ± 0.105	0.009 ± 0.074	0.779 ± 0.405
80	$\mathbf{1.000 \pm 0.000}$	0.313 ± 0.595	0.140 ± 0.243	-0.007 ± 0.023	0.307 ± 0.601	-0.013 ± 0.090	0.660 ± 0.589
90	$\mathbf{1.000 \pm 0.000}$	0.380 ± 0.538	0.300 ± 0.606	0.013 ± 0.121	0.013 ± 0.023	0.020 ± 0.020	0.653 ± 0.583
100	$\mathbf{0.673 \pm 0.566}$	0.573 ± 0.502	0.333 ± 0.577	0.053 ± 0.042	0.027 ± 0.133	0.013 ± 0.031	$\mathbf{0.673 \pm 0.566}$

Table C.1: Evaluation results for the memory length experiments. In all the experiments the context size is 1.

Context Size	GTrXL	Integrated	LSTM	No Memory	ReZero	Transformer-XL	Universal
1	0.660 ± 0.589	$\mathbf{1.000 \pm 0.000}$	0.993 ± 0.012	0.060 ± 0.080	0.333 ± 0.578	0.660 ± 0.589	0.333 ± 0.578
2	$\mathbf{0.907 \pm 0.162}$	0.653 ± 0.301	0.820 ± 0.171	-0.033 ± 0.042	0.100 ± 0.173	0.047 ± 0.081	0.373 ± 0.361
3	0.220 ± 0.460	$\mathbf{0.580 \pm 0.000}$	0.453 ± 0.232	0.120 ± 0.069	0.240 ± 0.000	-0.040 ± 0.280	0.207 ± 0.058
4	0.287 ± 0.216	0.107 ± 0.189	$\mathbf{0.560 \pm 0.072}$	-0.067 ± 0.099	0.033 ± 0.070	0.140 ± 0.171	0.093 ± 0.129
5	0.101 ± 0.138	0.180 ± 0.135	$\mathbf{0.278 \pm 0.089}$	-0.007 ± 0.076	0.050 ± 0.107	0.080 ± 0.000	0.020 ± 0.087
6	0.127 ± 0.133	0.080 ± 0.151	$\mathbf{0.380 \pm 0.131}$	0.047 ± 0.092	0.020 ± 0.069	0.013 ± 0.064	-0.007 ± 0.050
7	-0.033 ± 0.151	$\mathbf{0.160 \pm 0.255}$	$\mathbf{0.160 \pm 0.053}$	-0.053 ± 0.115	0.080 ± 0.183	-0.120 ± 0.000	0.040 ± 0.139
8	0.100 ± 0.312	0.060 ± 0.280	$\mathbf{0.313 \pm 0.061}$	0.213 ± 0.081	0.260 ± 0.000	0.087 ± 0.300	0.087 ± 0.300
9	-0.007 ± 0.023	0.173 ± 0.133	$\mathbf{0.293 \pm 0.070}$	-0.060 ± 0.140	0.080 ± 0.140	0.013 ± 0.031	0.007 ± 0.023
10	0.033 ± 0.080	0.047 ± 0.085	$\mathbf{0.130 \pm 0.099}$	-0.053 ± 0.042	0.014 ± 0.030	0.000 ± 0.000	0.008 ± 0.028
12	0.120 ± 0.000	0.040 ± 0.139	$\mathbf{0.200 \pm 0.053}$	0.073 ± 0.081	-0.027 ± 0.129	-0.120 ± 0.000	0.120 ± 0.000
14	0.107 ± 0.012	0.060 ± 0.125	$\mathbf{0.147 \pm 0.050}$	0.093 ± 0.042	-0.047 ± 0.129	-0.087 ± 0.023	-0.007 ± 0.090
17	-0.100 ± 0.191	0.027 ± 0.225	$\mathbf{0.173 \pm 0.061}$	-0.033 ± 0.194	-0.087 ± 0.266	0.047 ± 0.250	0.073 ± 0.254
20	0.027 ± 0.117	-0.020 ± 0.069	$\mathbf{0.060 \pm 0.120}$	0.007 ± 0.031	-0.047 ± 0.101	-0.020 ± 0.069	0.047 ± 0.151
25	0.073 ± 0.133	-0.007 ± 0.050	$\mathbf{0.213 \pm 0.012}$	0.013 ± 0.076	0.027 ± 0.064	-0.033 ± 0.042	0.000 ± 0.000
30	-0.040 ± 0.035	$\mathbf{0.060 \pm 0.000}$	0.013 ± 0.090	0.033 ± 0.083	-0.020 ± 0.072	0.040 ± 0.087	-0.040 ± 0.092
40	-0.033 ± 0.122	0.013 ± 0.121	$\mathbf{0.087 \pm 0.095}$	-0.093 ± 0.127	0.020 ± 0.087	0.047 ± 0.162	0.060 ± 0.156

Table C.2: Evaluation results for the memory size experiments. In all experiments the length of the maze is kept fixed at 1.

Context	GTrXL	Integrated	LSTM	No Memory	ReZero	Transformer-XL	Universal
(3, 3)	0.220 ± 0.450	0.967 ± 0.058	$\mathbf{1.000 \pm 0.000}$	0.640 ± 0.000	0.880 ± 0.131	0.827 ± 0.219	0.880 ± 0.208
(5, 3)	0.860 ± 0.131	$\mathbf{0.967 \pm 0.058}$	0.880 ± 0.208	0.640 ± 0.000	0.607 ± 0.153	0.860 ± 0.131	0.653 ± 0.081
(5, 5)	0.747 ± 0.335	0.833 ± 0.272	$\mathbf{1.000 \pm 0.000}$	0.340 ± 0.000	0.460 ± 0.120	0.327 ± 0.129	0.433 ± 0.401
(10, 3)	0.640 ± 0.000	0.673 ± 0.058	$\mathbf{0.933 \pm 0.058}$	0.640 ± 0.000	0.673 ± 0.058	0.753 ± 0.214	0.760 ± 0.208
(10, 5)	$\mathbf{0.427 \pm 0.133}$	0.340 ± 0.000	0.340 ± 0.000	0.340 ± 0.000	0.333 ± 0.012	0.233 ± 0.185	0.360 ± 0.035
(10, 7)	0.293 ± 0.023	0.380 ± 0.173	$\mathbf{0.580 \pm 0.000}$	0.280 ± 0.000	0.280 ± 0.000	0.247 ± 0.076	0.273 ± 0.012
(10, 9)	0.107 ± 0.046	0.080 ± 0.000	$\mathbf{0.387 \pm 0.531}$	0.080 ± 0.000	0.087 ± 0.012	0.153 ± 0.064	0.080 ± 0.000
(30, 3)	0.693 ± 0.284	0.327 ± 0.330	$\mathbf{1.000 \pm 0.000}$	0.027 ± 0.042	0.373 ± 0.359	0.427 ± 0.335	0.127 ± 0.090
(30, 5)	$\mathbf{0.873 \pm 0.219}$	0.007 ± 0.023	0.660 ± 0.589	-0.040 ± 0.020	0.100 ± 0.174	0.287 ± 0.136	0.207 ± 0.136
(30, 7)	$\mathbf{0.313 \pm 0.219}$	0.180 ± 0.223	0.613 ± 0.289	-0.040 ± 0.072	0.220 ± 0.255	0.213 ± 0.117	0.093 ± 0.172
(30, 9)	0.253 ± 0.450	0.020 ± 0.191	$\mathbf{0.267 \pm 0.219}$	-0.040 ± 0.053	0.047 ± 0.133	0.187 ± 0.064	0.087 ± 0.201
(30, 17)	0.160 ± 0.216	0.073 ± 0.140	$\mathbf{0.547 \pm 0.528}$	0.080 ± 0.144	0.027 ± 0.117	0.267 ± 0.081	0.253 ± 0.136
(30, 25)	0.167 ± 0.110	$\mathbf{0.247 \pm 0.012}$	0.233 ± 0.012	0.240 ± 0.000	0.160 ± 0.139	0.167 ± 0.127	0.147 ± 0.127

Table C.3: Evaluation results for the distributed memory experiments. The context is represented by a tuple, whereby the first element is the memory length and the second the memory size.

Bibliography

Ahmed, N. and Wahed, M. (2020), 'The De-democratization of AI: Deep Learning and the Compute Divide in Artificial Intelligence Research', *arXiv:2010.15581 [cs]* . arXiv: 2010.15581.
URL: *http://arxiv.org/abs/2010.15581*

Allis, V. (1994), Searching for solutions in games and artificial intelligence, PhD thesis, s.n.], S.l. ISBN: 9789090074887 OCLC: 905509528.

Ba, J. L., Kiros, J. R. and Hinton, G. E. (2016), 'Layer Normalization', *arXiv:1607.06450 [cs, stat]* . arXiv: 1607.06450.
URL: *http://arxiv.org/abs/1607.06450*

Bachlechner, T., Majumder, B. P., Mao, H. H., Cottrell, G. W. and McAuley, J. (2020), 'ReZero is All You Need: Fast Convergence at Large Depth', *arXiv:2003.04887 [cs, stat]* . arXiv: 2003.04887.
URL: *http://arxiv.org/abs/2003.04887*

Baevski, A. and Auli, M. (2019), 'Adaptive Input Representations for Neural Language Modeling', *arXiv:1809.10853 [cs]* . arXiv: 1809.10853.
URL: *http://arxiv.org/abs/1809.10853*

Bahdanau, D., Cho, K. and Bengio, Y. (2016), 'Neural Machine Translation by Jointly Learning to Align and Translate', *arXiv:1409.0473 [cs, stat]* . arXiv: 1409.0473.
URL: *http://arxiv.org/abs/1409.0473*

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., Schrittwieser, J., Anderson, K., York, S., Cant, M., Cain, A., Bolton, A., Gaffney, S., King, H., Hassabis, D.,

Legg, S. and Petersen, S. (2016), 'DeepMind Lab', *arXiv:1612.03801 [cs]* . arXiv: 1612.03801.
URL: *http://arxiv.org/abs/1612.03801*

Bengio, Y., Simard, P. and Frasconi, P. (1994), 'Learning long-term dependencies with gradient descent is difficult', *IEEE Transactions on Neural Networks* **5**(2), 157–166. Conference Name: IEEE Transactions on Neural Networks.

Britz, D., Goldie, A., Luong, M.-T. and Le, Q. (2017), 'Massive Exploration of Neural Machine Translation Architectures', *arXiv:1703.03906 [cs]* . arXiv: 1703.03906.
URL: *http://arxiv.org/abs/1703.03906*

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020), 'Language Models are Few-Shot Learners', *arXiv:2005.14165 [cs]* . arXiv: 2005.14165.
URL: *http://arxiv.org/abs/2005.14165*

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T. and Efros, A. A. (2018), 'Large-Scale Study of Curiosity-Driven Learning', *arXiv:1808.04355 [cs, stat]* . arXiv: 1808.04355.
URL: *http://arxiv.org/abs/1808.04355*

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A. and Zagoruyko, S. (2020), 'End-to-End Object Detection with Transformers', *arXiv:2005.12872 [cs]* . arXiv: 2005.12872.
URL: *http://arxiv.org/abs/2005.12872*

Cheng, J., Bendjama, K., Rittner, K. and Malone, B. (2020), 'BERTMHC: Improves MHC-peptide class II interaction prediction with transformer and multiple instance learning', *bioRxiv* p. 2020.11.24.396101. Publisher: Cold Spring Harbor

Laboratory Section: New Results.
URL: *https://www.biorxiv.org/content/10.1101/2020.11.24.396101v1*

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014), 'Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation', *arXiv:1406.1078 [cs, stat]* . arXiv: 1406.1078 version: 3.
URL: *http://arxiv.org/abs/1406.1078*

Chollet, F. (2017), 'Xception: Deep Learning with Depthwise Separable Convolutions', *arXiv:1610.02357 [cs]* . arXiv: 1610.02357.
URL: *http://arxiv.org/abs/1610.02357*

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V. and Salakhutdinov, R. (2019), 'Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context', *arXiv:1901.02860 [cs, stat]* . arXiv: 1901.02860.
URL: *http://arxiv.org/abs/1901.02860*

Dehghani, M., Gouws, S., Vinyals, O., Uszkoreit, J. and Kaiser, L. (2019), 'Universal Transformers', *arXiv:1807.03819 [cs, stat]* . arXiv: 1807.03819.
URL: *http://arxiv.org/abs/1807.03819*

Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. (2019), 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', *arXiv:1810.04805 [cs]* . arXiv: 1810.04805.
URL: *http://arxiv.org/abs/1810.04805*

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. (2020), 'An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale', *arXiv:2010.11929 [cs]* . arXiv: 2010.11929.
URL: *http://arxiv.org/abs/2010.11929*

Duchi, J., Hazan, E. and Singer, Y. (2011), 'Adaptive Subgradient Methods for Online Learning and Stochastic Optimization', p. 39.

Edunov, S., Ott, M., Auli, M. and Grangier, D. (2018), 'Understanding Back-Translation at Scale', *arXiv:1808.09381 [cs]* . arXiv: 1808.09381.
URL: *http://arxiv.org/abs/1808.09381*

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S. and Kavukcuoglu, K. (2018), 'IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures', *arXiv:1802.01561 [cs]* . arXiv: 1802.01561.
URL: *http://arxiv.org/abs/1802.01561*

Glorot, X. and Bengio, Y. (2010), 'Understanding the difficulty of training deep feedforward neural networks', p. 8.

Graves, A. (2017), 'Adaptive Computation Time for Recurrent Neural Networks', *arXiv:1603.08983 [cs]* . arXiv: 1603.08983.
URL: *http://arxiv.org/abs/1603.08983*

Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J. and Fernández, S. (2007), 'Unconstrained On-line Handwriting Recognition with Recurrent Neural Networks', *Advances in Neural Information Processing Systems* **20**, 577–584.
URL: *https://papers.nips.cc/paper/2007/hash/4b0250793549726d5c1ea3906726ebfe-Abstract.html*

Graves, A., Mohamed, A.-r. and Hinton, G. (2013), 'Speech Recognition with Deep Recurrent Neural Networks', *arXiv:1303.5778 [cs]* . arXiv: 1303.5778.
URL: *http://arxiv.org/abs/1303.5778*

Graves, A. and Schmidhuber, J. (2005), 'Framewise phoneme classification with bidirectional LSTM and other neural network architectures', *Neural Networks* pp. 5–6.

Graves, A. and Schmidhuber, J. (2008), 'Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks', *Advances in Neural Information Processing Systems* **21**, 545–552.
URL: *https://papers.nips.cc/paper/2008/hash/66368270ffd51418ec58bd793f2d9b1b-Abstract.html*

Greensmith, E., Bartlett, P. L. and Baxter, J. (2004), 'Variance Reduction Techniques for Gradient Estimates in Reinforcement Learning', p. 60.

Ha, D. and Schmidhuber, J. (2018), 'World Models', *arXiv:1803.10122 [cs, stat]* . arXiv: 1803.10122.
URL: *http://arxiv.org/abs/1803.10122*

Hausknecht, M. and Stone, P. (2017), 'Deep Recurrent Q-Learning for Partially Observable MDPs', *arXiv:1507.06527 [cs]* . arXiv: 1507.06527.
URL: *http://arxiv.org/abs/1507.06527*

Hauskrecht, M. and Fraser, H. (2000), 'Planning treatment of ischemic heart disease with partially observable Markov decision processes', *Artificial Intelligence in Medicine* **18**(3), 221–244.
URL: *http://www.sciencedirect.com/science/article/pii/S0933365799000421*

He, K., Zhang, X., Ren, S. and Sun, J. (2015), 'Deep Residual Learning for Image Recognition', *arXiv:1512.03385 [cs]* . arXiv: 1512.03385.
URL: *http://arxiv.org/abs/1512.03385*

He, K., Zhang, X., Ren, S. and Sun, J. (2016), 'Identity Mappings in Deep Residual Networks', *arXiv:1603.05027 [cs]* . arXiv: 1603.05027.
URL: *http://arxiv.org/abs/1603.05027*

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M. and Silver, D. (2017), 'Rainbow: Combining Improvements in Deep Reinforcement Learning', *arXiv:1710.02298 [cs]* . arXiv: 1710.02298.
URL: *http://arxiv.org/abs/1710.02298*

Hochreiter, S. and Schmidhuber, J. (1997), 'Long Short-term Memory', *Neural computation* **9**, 1735–80.

Hoey, J., Bertoldi, A. v., Poupart, P. and Mihailidis, A. (2007), 'Assisting persons with dementia during handwashing using a partially observable Markov decision process.', *International Conference on Computer Vision Systems : Proceedings*

(2007) .
URL: *https://biecoll.ub.uni-bielefeld.de/index.php/icvs/article/view/190*

Ioffe, S. and Szegedy, C. (2015), 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', *arXiv:1502.03167 [cs]* . arXiv: 1502.03167.
URL: *http://arxiv.org/abs/1502.03167*

Kim, Y., Denton, C., Hoang, L. and Rush, A. M. (2017), 'Structured Attention Networks', *arXiv:1702.00887 [cs]* . arXiv: 1702.00887.
URL: *http://arxiv.org/abs/1702.00887*

Kingma, D. P. and Ba, J. (2017), 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 [cs]* . arXiv: 1412.6980.
URL: *http://arxiv.org/abs/1412.6980*

Kochenderfer, M. J., Amato, C., Chowdhary, G., How, J. P., Reynolds, H. J. D., Thornton, J. R., Torres-Carrasquillo, P. A., Ure, N. K. and Vian, J. (2015), Optimized Airborne Collision Avoidance, *in* 'Decision Making Under Uncertainty: Theory and Application', MIT Press, pp. 249–276. Conference Name: Decision Making Under Uncertainty: Theory and Application.
URL: *https://ieeexplore.ieee.org/document/7288641*

Li, S., Wang, R., Tang, M. and Zhang, C. (2019), 'Hierarchical Reinforcement Learning with Advantage-Based Auxiliary Rewards', *arXiv:1910.04450 [cs]* . arXiv: 1910.04450.
URL: *http://arxiv.org/abs/1910.04450*

Liu, Y. and Lapata, M. (2019), 'Text Summarization with Pretrained Encoders', *arXiv:1908.08345 [cs]* . arXiv: 1908.08345.
URL: *http://arxiv.org/abs/1908.08345*

Mayer, H., Gomez, F., Wierstra, D., Nagy, I., Knoll, A. and Schmidhuber, J. (2006), A system for robotic heart surgery that learns to tie knots using recurrent neural networks, *in* '2006 IEEE/RSJ international conference on intelligent robots and systems', pp. 543–548. ISSN: 2153-0866.

Mishra, N., Rohaninejad, M., Chen, X. and Abbeel, P. (2018), 'A Simple Neural Attentive Meta-Learner', p. 17.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D. and Kavukcuoglu, K. (2016), 'Asynchronous Methods for Deep Reinforcement Learning', *arXiv:1602.01783 [cs]* . arXiv: 1602.01783.
URL: *http://arxiv.org/abs/1602.01783*

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. (2013), 'Playing Atari with Deep Reinforcement Learning', p. 9.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015), 'Human-level control through deep reinforcement learning', *Nature* **518**(7540), 529–533. Number: 7540 Publisher: Nature Publishing Group.
URL: *https://www.nature.com/articles/nature14236*

Nachum, O., Gu, S., Lee, H. and Levine, S. (2018), 'Data-Efficient Hierarchical Reinforcement Learning', *arXiv:1805.08296 [cs, stat]* . arXiv: 1805.08296.
URL: *http://arxiv.org/abs/1805.08296*

Oord, A. v. d., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A. and Kavukcuoglu, K. (2016), 'Conditional Image Generation with PixelCNN Decoders', *arXiv:1606.05328 [cs]* . arXiv: 1606.05328.
URL: *http://arxiv.org/abs/1606.05328*

Osband, I., Doron, Y., Hessel, M., Aslanides, J., Sezener, E., Saraiva, A., McKinney, K., Lattimore, T., Szepesvari, C., Singh, S., Van Roy, B., Sutton, R., Silver, D. and Van Hasselt, H. (2020), 'Behaviour Suite for Reinforcement Learning', *arXiv:1908.03568 [cs, stat]* . arXiv: 1908.03568.
URL: *http://arxiv.org/abs/1908.03568*

Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gulcehre, C., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M. M., Heess, N. and Hadsell, R. (2019), 'Stabilizing Transformers for Reinforcement Learning',

arXiv:1910.06764 [cs, stat] . arXiv: 1910.06764.
URL: *http://arxiv.org/abs/1910.06764*

Pennington, J., Schoenholz, S. S. and Ganguli, S. (2017), 'Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice', *arXiv:1711.04735 [cs, stat]* . arXiv: 1711.04735.
URL: *http://arxiv.org/abs/1711.04735*

Poole, B., Lahiri, S., Raghu, M., Sohl-Dickstein, J. and Ganguli, S. (2016), 'Exponential expressivity in deep neural networks through transient chaos', *arXiv:1606.05340 [cond-mat, stat]* . arXiv: 1606.05340.
URL: *http://arxiv.org/abs/1606.05340*

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D. and Sutskever, I. (2019), 'Language Models are Unsupervised Multitask Learners', p. 24.

Rafiee, B. (2020), 'Testbeds for Reinforcement Learning', *arXiv:2011.04590 [cs]* . arXiv: 2011.04590.
URL: *http://arxiv.org/abs/2011.04590*

Ravanelli, M., Brakel, P., Omologo, M. and Bengio, Y. (2018), 'Light Gated Recurrent Units for Speech Recognition', *IEEE Transactions on Emerging Topics in Computational Intelligence* **2**(2), 92–102. arXiv: 1803.10225.
URL: *http://arxiv.org/abs/1803.10225*

Ruder, S. (2017), 'An overview of gradient descent optimization algorithms', *arXiv:1609.04747 [cs]* . arXiv: 1609.04747.
URL: *http://arxiv.org/abs/1609.04747*

Schaul, T., Quan, J., Antonoglou, I. and Silver, D. (2016), 'Prioritized Experience Replay', *arXiv:1511.05952 [cs]* . arXiv: 1511.05952.
URL: *http://arxiv.org/abs/1511.05952*

Schmidhuber, J., Wierstra, D. and Gomez, F. (2005), 'Evolino: Hybrid neuroevolution / optimal linear search for sequence prediction'.

Schulman, J., Moritz, P., Levine, S., Jordan, M. and Abbeel, P. (2018), 'High-Dimensional Continuous Control Using Generalized Advantage Estimation',

arXiv:1506.02438 [cs] . arXiv: 1506.02438.
URL: *http://arxiv.org/abs/1506.02438*

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T. and Hassabis, D. (2016), 'Mastering the game of Go with deep neural networks and tree search', *Nature* **529**(7587), 484–489. Number: 7587 Publisher: Nature Publishing Group.
URL: *https://www.nature.com/articles/nature16961*

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D. (2018), 'A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play', *Science* **362**(6419), 1140–1144. Publisher: American Association for the Advancement of Science Section: Report.
URL: *https://science.sciencemag.org/content/362/6419/1140*

Srivastava, R. K., Greff, K. and Schmidhuber, J. (2015), 'Highway Networks', *arXiv:1505.00387 [cs]* . arXiv: 1505.00387.
URL: *http://arxiv.org/abs/1505.00387*

Su, Y. and Kuo, C.-C. J. (2019), 'On Extended Long Short-term Memory and Dependent Bidirectional Recurrent Neural Network', *Neurocomputing* **356**, 151–161. arXiv: 1803.01686.
URL: *http://arxiv.org/abs/1803.01686*

Sutton, R. S. (1988), 'Learning to predict by the methods of temporal differences', *Machine Learning* **3**(1), 9–44.
URL: *https://doi.org/10.1007/BF00115009*

Sutton, R. S. and Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA.

Tieleman, T. and Hinton, G. (2012), COURSERA: Neural Networks for Machine Learning, Technical Report.
URL: *http://www.cs.toronto.edu/ hinton/coursera/lecture6/lec6.pdf*

Tromp, J. and Farnebäck, G. (2006), Combinatorics of Go, Vol. 4630, pp. 84–99.

van Hasselt, H., Guez, A. and Silver, D. (2015), 'Deep Reinforcement Learning with Double Q-learning', *arXiv:1509.06461 [cs]* . arXiv: 1509.06461.
URL: *http://arxiv.org/abs/1509.06461*

Vashishth, S., Upadhyay, S., Tomar, G. S. and Faruqui, M. (2019), 'Attention Interpretability Across NLP Tasks', *arXiv:1909.11218 [cs]* . arXiv: 1909.11218.
URL: *http://arxiv.org/abs/1909.11218*

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017), 'Attention Is All You Need', *arXiv:1706.03762 [cs]* . arXiv: 1706.03762.
URL: *http://arxiv.org/abs/1706.03762*

Vinyals, O., Babuschkin, I., Chung, J., Mathieu, M., Jaderberg, M., Czarnecki, W., Dudzik, A., Huang, A., Georgiev, P., Powell, R., Ewalds, T., Horgan, D., Kroiss, M., Danihelka, I., Agapiou, J., Oh, J., Dalibard, V., Choi, D., Sifre, L., Sulsky, Y., Vezhnevets, S., Molloy, J., Cai, T., Budden, D., Paine, T., Gulcehre, C., Wang, Z., Pfaff, T., Pohlen, T., Yogatama, D., Cohen, J., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Apps, C., Kavukcuoglu, K., Hassabis, D. and Silver, D. (2019), 'AlphaStar: Mastering the real-time strategy game StarCraft II'.
URL: *https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/*

Walsh, T. J., Goschin, S. and Littman, M. L. (2010), *Integrating Sample-based Planning and Model-based Reinforcement Learning.*

Wang, L., Duan, X., Zhang, Q., Niu, Z., Hua, G. and Zheng, N. (2018), 'Segment-Tube: Spatio-Temporal Action Localization in Untrimmed Videos with Per-Frame Segmentation', *Sensors* **18**(5), 1657.
URL: *http://www.mdpi.com/1424-8220/18/5/1657*

Wang, S., Li, B. Z., Khabsa, M., Fang, H. and Ma, H. (2020), 'Linformer: Self-Attention with Linear Complexity', *arXiv:2006.04768 [cs, stat]* . arXiv: 2006.04768.
URL: *http://arxiv.org/abs/2006.04768*

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M. and de Freitas, N. (2016), 'Dueling Network Architectures for Deep Reinforcement Learning', *arXiv:1511.06581 [cs]* . arXiv: 1511.06581.
URL: *http://arxiv.org/abs/1511.06581*

Wayne, G., Hung, C.-C., Amos, D., Mirza, M., Ahuja, A., Grabska-Barwinska, A., Rae, J., Mirowski, P., Leibo, J. Z., Santoro, A., Gemici, M., Reynolds, M., Harley, T., Abramson, J., Mohamed, S., Rezende, D., Saxton, D., Cain, A., Hillier, C., Silver, D., Kavukcuoglu, K., Botvinick, M., Hassabis, D. and Lillicrap, T. (2018), 'Unsupervised Predictive Memory in a Goal-Directed Agent', *arXiv:1803.10760 [cs, stat]* . arXiv: 1803.10760.
URL: *http://arxiv.org/abs/1803.10760*

Whitehead, S. D. and Lin, L.-J. (1995), 'Reinforcement learning of non-Markov decision processes', *Artificial Intelligence* **73**(1), 271–306.
URL: *http://www.sciencedirect.com/science/article/pii/000437029400012P*

Wu, Y., Mansimov, E., Liao, S., Grosse, R. and Ba, J. (2017), 'Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation', *arXiv:1708.05144 [cs]* . arXiv: 1708.05144.
URL: *http://arxiv.org/abs/1708.05144*

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, L., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M. and Dean, J. (2016), 'Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation', *arXiv:1609.08144 [cs]* . arXiv: 1609.08144.
URL: *http://arxiv.org/abs/1609.08144*

Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. and Le, Q. V. (2020), 'XLNet: Generalized Autoregressive Pretraining for Language Understanding', *arXiv:1906.08237 [cs]* . arXiv: 1906.08237.
URL: *http://arxiv.org/abs/1906.08237*