MSc Software Design with AI
Advanced Machine Learning (AL_KSAIM_9_1)
Assignment 1 – option 1

source "Yosh" YouTube channel where he trains an AI to play Trackmania using RL

Survey of Modern Neural Architectures for Game AI

Andrew Nolan

A00325359@student.tus.ie

7th May 2025

Contents

Introduction

Game AI isn't just about reacting anymore. A lot of the newer systems try to plan ahead, learn from experience, and deal with situations where they don’t see the full picture. In this piece, I’m looking at three neural network architectures that’ve been used in games like chess, Atari, and some more complex environments. The idea is to compare how they’re built, what they bring to the table, and where each one fits in.

MuZero - figures out how to play games like chess and Go without even knowing the rules, by learning a model of the game and planning using that.

EfficientZero V2, which builds on similar ideas but makes it work with less data, and in both visual and low-dimensional control tasks.

Temporal Difference Variational Auto-Encoder (*TD-VAE*), is a bit different. It tries to build an internal belief about the world and predict future states by skipping steps, which helps in environments where you can’t always see everything.

Reinforcement learning

Reinforcement learning (RL) is a type of machine learning where an agent learns how to make decisions by interacting with its environment. The agent takes actions and receives feedback in the form of rewards or penalties, and its goal is to maximize the total reward over time. What makes RL interesting is that it allows machines to learn autonomously, figuring out what works and what doesn't through trial and error, much like how humans or animals learn by doing.

When it comes to Game AI, RL is particularly powerful. Games are an ideal test bed for RL because they provide a controlled environment with clear rules and immediate feedback, which makes it easier to track progress. Game AI, like the bots that play chess, Go, or even Grand Theft Auto, is essentially about solving a set of problems within a defined space. This makes it a simplification of real-world situations, where the environment may be less predictable, and the stakes higher.

Why I picked Game AI

In a way, games can be viewed as a simplified version of real life. Just like how real-life problems like driving a car or managing a drone are complex, games provide an simplification where the AI can experiment, fail and improve without any real world consequences. For example, self-driving cars can be trained first in a simulated environment before they're allowed to interact with real traffic. It's the same approach for complex systems like helicopter controls or robotic arms. By learning in a simulation, the AI can safely build up experience before facing the chaos of the real world.

Architecture 1: MuZero

Concise Overview

MuZero is a model-based reinforcement learning method that was built to achieve superhuman performance in a wide variety of domains, ranging from video games like Atari to strategy games like chess, shogi, and Go. What makes MuZero stand out is it doesn't require knowledge of the environment's dynamics (how the environment works, how actions lead to results). This makes it useful in real-world situations where the rules are either too complex or unknown, and it needs to figure things out as it goes.

Key Contributions and innovations

Learned Model

MuZero is pretty unique because it combines tree-based search with a model that learns to predict the future. Instead of relying on pre-set game rules or simulations, it learns how to predict things like the reward, the policy (what action to take), and the value (how good a state is). This allows the system to plan and make decisions.

This is similar to how stockfish (popular chess engine) works. Stockfish uses a tree-based search, optimized using alpha-beta pruning, with some evaluation function, which, if I'm not mistaken doesn't use Deep Learning but just chess theory.

In terms of architecture, MuZero is built from three connected networks: the representation function (which encodes the observation history into a hidden state), the

dynamics function (which predicts the next state and reward given an action), and the prediction function (which outputs the policy and value). These modules are feedforward in nature, and typically include residual connections to help with stability. Activation functions like ReLU are commonly used. MuZero is trained using gradient-based optimization (usually Adam), with losses computed from differences between its predictions and actual outcomes across simulated search paths. The real strength of the system comes from combining this learned model with Monte Carlo Tree Search (MCTS)-this lets it plan out moves several steps ahead, similar to how engines like Stockfish evaluate moves. The difference is that Stockfish uses hardcoded chess knowledge and alpha-beta pruning, whereas MuZero learns the patterns and strategy purely from data.

Generalisation

It builds on AlphaZero's success but pushes the boundaries even further. MuZero can handle more than just games like Go and chess. It can be applied to environments with unknown dynamics and even where rewards aren't directly tied to winning or losing.

Efficiency

In environments like Atari, MuZero's efficiency is impressive. It's able to perform at a state-of-the-art level without needing as many resources or a deep understanding of the rules of the games.

Performance on Benchmarks

Atari Games

MuZero made a significant impact on Atari benchmarks. It outperformed previous methods (like R2D2 and SimPLe) in terms of mean and median normalized scores across 57 games. This shows it can really adapt to a range of different environments and still come out on top.

Board Games

In terms of board games, MuZero surpassed AlphaZero's performance in Go by a small margin. It also held its own in chess and shogi, continuing its streak of performing at or above human expert level.

Strengths and limitations

Strengths

Versatility-One of the big wins for MuZero is how adaptable it is. It doesn't need to know the ins and outs of an environment, which is especially useful when the environment is complex or completely unknown.

MuZero performs well in standard games like chess but also excels in visually complex environments like Atari games, making it one of the most versatile RL models out there.

Efficiency-With the MuZero Reanalyze variant, it does an impressive job of reworking its decisions and making use of past data, improving the way it learns and applies that knowledge with fewer resources

Limitations

Scalability-While it works great in controlled environments like board games, scaling MuZero up to work efficiently in more complex, large-scale environments (like video games) can be a challenge. As the number of search simulations grows, performance improvements start to level off, making it harder to maintain that high level of efficiency.

Architecture 2: EfficientZero v2

Concise Overview

EfficientZero V2 is a reinforcement learning setup aimed at doing more with less data. It builds on the original EfficientZero idea but pushes it further so it works better in situations where actions aren't just simple button presses but might be more complex or continuous. The main aim here is to get good results even when you're limited in how much training data you have, which makes it feel a bit more practical when you think about real world uses, like robots or physical systems where you can't just repeat something a million times.

EfficientZero V2's architecture consists of three main networks: representation, dynamics, and prediction, similar to MuZero but optimised for efficiency. It mainly uses convolutional layers to process visual inputs, combined with fully connected layers for non-visual data. The design employs residual connections to improve training stability and help deeper layers learn better. ReLU activation functions provide the necessary non-linearity throughout the model. For connectivity, it's mostly feedforward with these skip connections to avoid vanishing gradients. It has an action embedding layer, which transforms complex or continuous actions into a latent space, allowing the model to handle more decision-making. The learning mechanism integrates a sampling-based search method for better exploration, and loss functions that strike a balance between accurate environment modelling with efficient learning from fewer samples. Overall, the architecture is structured to maintain strong predictive power while minimising data requirements.

Key Contributions and innovations

Discrete *and* Continuous control

One of the biggest improvements here is that EfficientZero V2 works well across different types of control tasks. It works well with continuous inputs such as a joystick and it works well with discrete inputs like buttons.

Smart tree search using sampling

V2 has a sampling-based version of Gumbel search that helps it deal with larger/more complex action spaces without impacting compute too much.

Search-based value estimation

This uses already collected data which helps the algorithm learn more from less.

Performance on Benchmarks

Atari 100k

It got a normalized mean score of 2.428, which was better than anything before it.

Proprio Control

Across 20 tasks, it averaged 723.2. It also ran faster than TD-MPC2 which is important for lag or resource usage.

Vision Control

Scored a mean of 726.1, beating DreamerV3 by a margin and topping 16 out of 20 tasks.

Strengths and limitations

Strengths

High Sample Efficiency-Achieves high performance with limited data

Versatile-Capable of handling both discrete and continuous control tasks, as well as visual and low-dimensional inputs.

Computational Efficiency-Reduces computation with action planning and value estimation.

Limitations

Complexity during training-using the sampling-based Gumbel search and search-based value estimation introduces additional complexity during training.

Early Training Stages-The model's accuracy in the early stages of training can impact the effectiveness of the search-based value estimation.

Architecture 3: Temporal Difference Variational Auto-Encoder (*TD-VAE*)

Concise Overview

TD-VAE is a type of model that tries to predict the future in a smart way by skipping over the in-between steps. It builds a memory that includes its uncertainty about what could happen. This is especially useful in environments where the agent doesn't have access to all the information at once, like if parts of the world are hidden or things only become clear after a while. Instead of going frame by frame, it jumps ahead and makes educated predictions about what might happen in a few steps, based on what it's seen so far.

TD-VAE's architecture is built around a few key components designed to handle this kind of temporal reasoning. It uses an encoder and decoder like a typical variational autoencoder, but here they operate on sequences of observations rather than single data points. The encoder compresses past observations into a latent state that captures both what's known and the uncertainty about the future. The model then uses a transition network to predict how this latent state evolves over time, skipping intermediate steps instead of predicting every frame. Residual connections help maintain stable information flow through these layers. For activation functions, it mostly relies on standard nonlinearities like ReLU to keep things efficient. The learning mechanism combines temporal difference learning with variational inference, optimizing a loss that balances reconstructing future states and accurately modelling uncertainty. Specialized modules like the transition network and the latent state sampler make TD-VAE unique, allowing it to predict multiple steps ahead while managing uncertainty in partially observed environments.

Key Contributions and innovations

Abstract State Representation

Rather than recreating every little detail of what the agent sees, TD-VAE focuses on the bigger picture. It builds an internal version of the world that's simplified but still useful, letting it predict what's likely to happen without wasting effort on stuff that doesn't matter.

Belief State

It holds onto what it knows so far in the form of a “belief state”-a mix of memory and probability. This is useful in cases where the environment hides information, so the agent can still make decent decisions based on what it thinks might be true.

Temporal Abstraction

Probably the biggest difference here is that it doesn’t need to simulate things step-by-step. It learns to jump ahead and still keep its predictions consistent. This means it can think further ahead without needing to process every tiny transition in between.

Performance on Benchmarks

MiniPacman

TD-VAE handled the uncertainty in this environment better than other models. It got lower error when trying to predict future sequences, and just seemed to "get" what might be happening off-screen.

Moving MNIST

It was able to skip forward in time and still correctly track the digit, including its direction and speed, even though it wasn’t checking every frame.

Noisy Harmonic Oscillator

Despite all the noise in the signal, the model was able to pick up on the actual frequency and pattern, showing it can filter out unimportant noise and still track what matters.

DeepMind Lab

Even in more complex 3D setups, it could hold onto possible futures and roll forward in time, not just randomly but in a way that lined up with its beliefs. This was impressive given the visual complexity and uncertainty.

Strengths and limitations

Strengths

Efficient Learning - It learns good mental shortcuts. Instead of wasting time on visual details, it builds an internal state that’s lean and fast.

Temporal Abstraction - The jumpy predictions help it plan faster and not get bogged down by irrelevant steps.

Versatility - It didn’t just work on one kind of task. It handled different types of environments pretty well, from simple toy data to full 3D simulations.

Limitations

Complexity-The model's structure isn't exactly lightweight. Training it and getting the settings right takes more work compared to other options.

Generalization-Even though it did well on the tested tasks, it's hard to say how it'll perform in really different situations without more testing. It might need some tweaking depending on the domain.

Comparative Summary

All three models; MuZero, EfficientZero V2, and TD-VAE, focus on decision-making in complex environments, but they approach the problem differently depending on the goals.

MuZero stands out for its planning capabilities. It learns a model of the environment that doesn't try to recreate the raw observations but instead focuses on what's relevant-rewards, value estimates, and action policies. It uses this learned model inside a tree search to plan ahead. One of its most distinct strengths is that it doesn't need the actual rules of the environment, it figures things out just by interacting with it. That's part of why it performs so well across games like Go, chess, and Atari.

EfficientZero V2 builds on that idea but aims to improve sample efficiency. It uses imagined rollouts, value bootstrapping, and sampling-based planning methods to reduce how much real data is needed. It also brings in more modern architectural choices, like transformer-based components, which help it generalise across both discrete and continuous action spaces. So, while MuZero is more focused on performance in fixed environments, EfficientZero V2 is more concerned with learning faster from less data.

TD-VAE takes a different direction. Instead of planning in the traditional sense, it builds a belief about how the world might change over time. It doesn't simulate every step; instead, it learns to predict across time gaps, capturing information about the future in a compact latent space. The model focuses on learning structure in sequences, both to predict observations, and to reason about possible outcomes. This makes it better suited to environments with partial observability or where exact transitions aren't necessary for planning.

Some trends start to emerge. There's a clear shift toward models that avoid modelling everything in full detail. Instead, they focus on what's useful-such as value estimates, belief states, or compressed future summaries. Another common idea is temporal abstraction. All three models find ways to move beyond step-by-step predictions and start making higher-level inferences.

As for open questions, generalisation remains a challenge-these models still struggle in more chaotic or less structured settings. Also, handling uncertainty more consistently is an area where TD-VAE is ahead, while the other two are still evolving. Finally, integrating the benefits of each-like combining MuZero's planning with TD-VAE's time-jumping belief states-could lead to even stronger models in the future.

References

Wang, S., Liu, S., Ye, W., You, J., & Gao, Y. (2024). *EfficientZero V2: Mastering discrete and continuous control with limited data*.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., & Silver, D. (2020). **Mastering Atari, Go, chess and shogi by planning with a learned model**. *Nature*, 588(7839), 604–609. https://doi.org/10.1038/s41586-020-03051-4 <- MuZero

Gregor, K., Papamakarios, G., Besse, F., Buesing, L., & Weber, T. (2019). *Temporal difference variational auto-encoder*. In *Proceedings of the 7th International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1806.03107 <-TD-VAE

* These paper's can also be found in the github repo