Deep Cognitive Transformers For Deep Reinforcement Learning Agent Network

Yudo Nidlom Firmansyah
yudonidlomfirmansyah@gmail.com

Abstract

Deep reinforcement learning is an algorithm widely used as an optimization algorithm or an algorithm for making decisions, the ability of deep reinforcement learning to adapt to new environments is the main advantage of deep reinforcement learning. This study aims to create an agent that can remember and learn patterns in the environment by taking inspiration from the concept of cognitive psychology and combined with some parts of the transformer model, where agents can focus, remember, and choose the information needed to make the best choice in taking action. This agent will be tested using the Space Invaders game environment environment and another three Atari environments. The results of this study indicate that the Agent can learn complex environments and memorize patterns in this space-invader environment and another three Atari environments, this shows that agents can adapt well if they are in a new environment.

Keywords: Deep Reinforcement Learning, Artificial Neural Network, Memory, Cognitive Psychology, Transformers Model, Decision Making, Space-Invaders, Agent, Atari.

1. Introduction

Reinforcement learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic environment (Pack Kaelbling et al., 1996) . (Nussenbaum & Hartley, 2019) explain that conducting reinforcement learning tests has several common features, such as in a slot machine environment or a deck of cards where each choice has a reward or positive, negative, or neutral outcome, and participants are given instructions to choose the option that provides the most reward, by studying the results of their choices, participants will be more likely to choose the possibility with the highest reward. These probabilistic options are probabilities that sum up the available actions generated by an agent in the form of an artificial neural network model, with outputs generated from probability formulas such as softmax, sigmoid, log softmax, and log sigmoid.

But just using a regular artificial neural network is not enough for agents to be able to understand the environment quickly, so agents must be able to remember and learn patterns in the environment, with the above problems, and take inspiration from how humans think cognitively. The set of models used in cognitive science is diverse, and DNNs are one particular, useful kind(Cichy & Kaiser, 2019). DNN is not enough to meet the requirements of cognitive thinking, what is needed for cognitive thinking includes Attention, Resonating, and decision-making. So it is necessary to take inspiration from several models, such as Transformers and others to build a neural network model with the concept of cognitive thinking.

Model-free deep reinforcement learning (RL) algorithms have been applied in a range of challenging domains, from games (Mnih et al., 2013; Silver et al., 2016 ; Haarnoja et al., 2018). This research aims

to build a neural network model based on cognitive thinking so that agents or models can learn and adapt to complex environments.

2. Relate Work

Previous research on cognitive models has been conducted and has been applied to robots or self-driving cars.

Figure 1 , The illustration of deep cognitive networks (DCNs)(Huang & Wang, n.d.)

(Huang & Wang, n.d.) Stated that key cognitive mechanisms are increasingly being modeled using deep learning models, specifically Deep Cognitive Networks (DCNs), in an effort to address this problem. Through the implementation of many cognitive functions, such as dynamic reasoning, knowledge reuse, and selective information extraction, DCNs can outperform traditional deep learning models in numerous real-world scenarios.

3. Model Architecture

The deep cognitive model has several components: reasoning, attention, memory, and decision. From the four elements, I replaced attention with the Encoder and Decoder of the transformer model, the Encoder and Decoder of the transformer model because there is already an attention model in it, so the model becomes Deep Cognitive Transformer. figure 2 is the Deep Cognitive Transformers architecture, and Figure 3 is environments for a state.

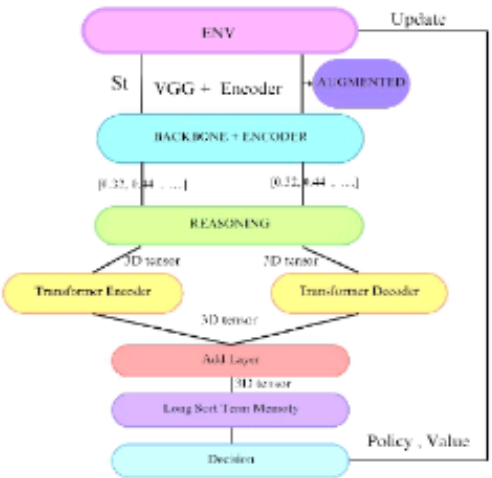


Figure 2, Deep Cognitive Transformers Network Architecture, for test on Atari game

Figure 3, environments space invaders, From this environment, the state is then taken to be processed by the Agent.

The flowchart above is the architecture of the Deep Cognitive Transformers (DeepCTNet) model, where the model above is tested on the Atari game environment (Space Invader) and Encoder Transformers and Decoder Transformers as a substitute for Attention. Encoder maps an input sequence of symbol

representations (x1, . . . , xn) to a sequence of continuous representations z = (z1, . . . , zn).Given z, the decoder then generates an output sequence (y1, . . . , ym) of symbols one element at a time(Vaswani et al., 2017a).

3.1. Image Augmentation

Data augmentation is another way we can reduce overfitting on models, where we increase the amount of training data using information only in our training data (Perez & Wang, 2017). In the architecture above, only one right side has Data augmentation, this is because the left side is the original state image if both sides are only the original state image without augmentation, then the information on both sides will be the same and will not provide new information by making additions in the form of rotation, the image will provide information on the right side so that the model can learn the state with more information. The data augmentation method used is rotation. Techniques for effecting a 2-D rotation of a discrete image are very important in many different disciplines. Examples include medical imaging, digital photography, and computer graphics (Park et al., 2009). The direct determination of the rotated points is a one-pass method (Ashtari et al., 2015).

$$\begin{pmatrix} x_r \\ y_r \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_t \\ y_t \end{pmatrix}. \quad (1)$$

The formula (1) is a one pass method formula that can be used to rotate images. In figure 4 there is a way to use the one pass method.

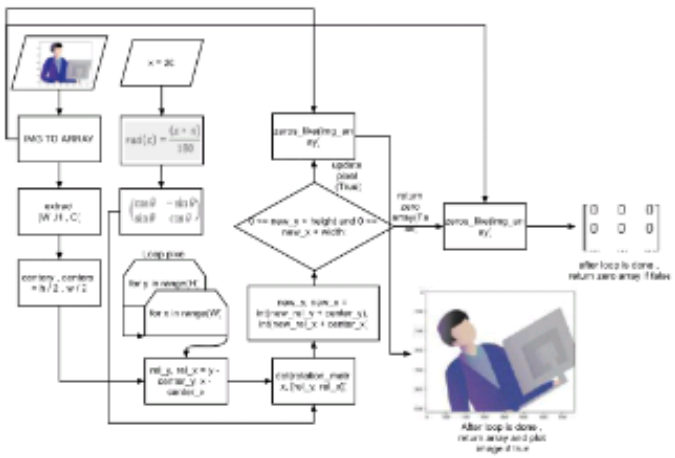


Figure 4, Flowchart data augmentation process to perform rotation (30 degrees)

3.2.Backbone(feature extraction) + Encoder

Now, with the development of convolution neural networks (CNNs), the feature extraction operation has become more automatic and easier. CNNs allow to work on large-scale size of data, as well as cover different scenarios for a specific task (Elharrouss et al., 2022). These networks extract feature from the input image used by the model (Zaidi et al., 2021). with a CNN employed within a recognition model (encoder) for the posterior distribution of the parameters(Pu et al., n.d.). The posterior distribution is a representation of the image or LaTex vector that has been processed through the encoder and then goes to the next block.

3.2.1. Backbone(feature extraction)

VGGNet will be used as a model for feature extraction. The visual geometry group network (VGGNet) is used widely for image classification and has proven to be very effective method (Muhammad et al.,

2018). Figure 4 is a block of VGGNet architecture but I have modified it by adding skip connections. The skip connections introduce complex interactions among layers, and thus the resulting dynamics are more intricate (Xu et al., 2021).

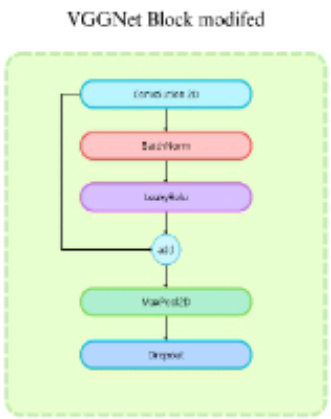


Figure 5, The image above is a block of the VGGNet architecture that I have edited.

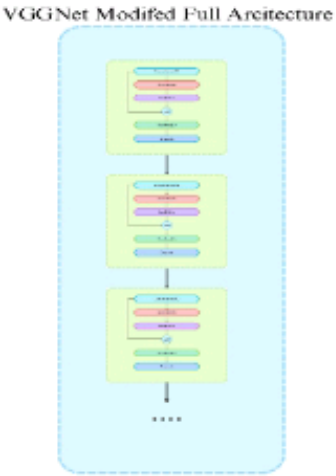


Figure 6, The image above is the VGGNet architecture.

The original blocks on VGGNet have no skip connections and use ReLu $x(x > 0)$,instead of LeakyReLu $x(x\alpha > 0)$. One drawback of the ReLU is that it does not activate for non-positive inputs, causing the deactivation of several neurons during training, which can be viewed again as a vanishing gradient problem for negative values(Maniatopoulos & Mitianoudis, 2021).

3.2.2. Encoder

In the part for the encoder architecture, it will be quite simple, for the Image encoder it only uses Flatten and 2 linear layers as in Figure 6.

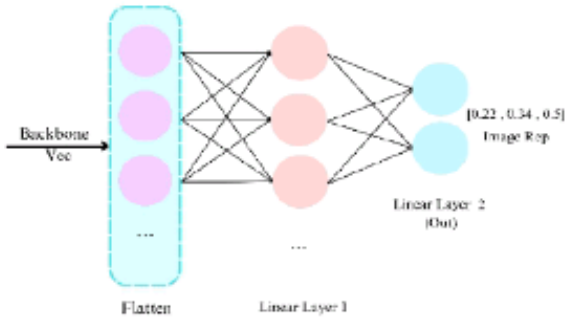


Figure 7, Image Encoder Arcitecture

3.3. Reasoning

Variety of inductive reasoning that is critical for learning. The most classic aspect of induction is the way in which humans and other creatures acquire knowledge about causal relations, which is critical for predicting the consequences of actions and event(Holyoak & Morrison, 2005). in the DeepCTNet model that used memory reasoning or End-to-end Memory Networks. Memory networks reason with inference components combined with a long-term memory component; they learn how to use these jointly (Weston et al., 2014). The continuity of the model we present here means that it can be trained end-to-end from input-output pairs(Sukhbaatar et al., n.d.).

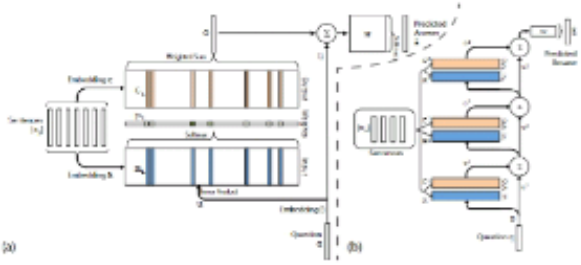


Figure 8,(a): A single layer version of our model. (b): A three layer version of our model(Sukhbaatar et al., n.d.).

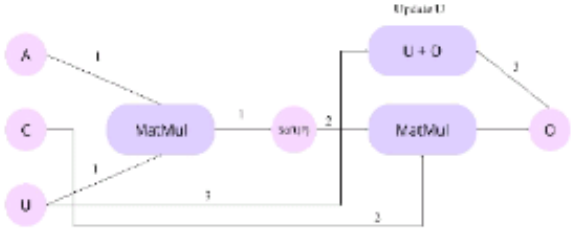


Figure 9, single layer architecture End to End Memory Networks model, simpler illustrator version , Block A , C , U is embending Block

The End to End Memory Network model used is a multi-layer End to End Memory Network where the number of layers used is two layers because Resonating is the part for making considerations before being processed by the decision model. The input to layers above the first is the sum of the output o^k and the input u^k from layer k(Sukhbaatar et al., n.d.).

$$u^{k+1} = o^k + u^k \qquad (2)$$

The formula above is how to update u^k to get the value of o^k you can see formulas 3 to 4.

$$p^k = \frac{e^{(a^k.\ u^k)}}{\sum e^{(a^k.\ u^k)}} \qquad (3)$$

$$o^k = p^k .\ c^k \qquad (4)$$

In formula 3, to get p^k the initial multiplication results $(a^k.\ u^k)$ are entered into softmax to get the score, then p^k will be multiplied by c^k ($o^k = p^k .\ c^k$) to get the value o^k, then continue to formula 2 to update u^{k+1} and the loop will continue until the number of K layers.

3.4. Transformers

The Transformer follows this overall architecture using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in the left and right halves of Figure 1, respectively (Vaswani et al., 2017b).

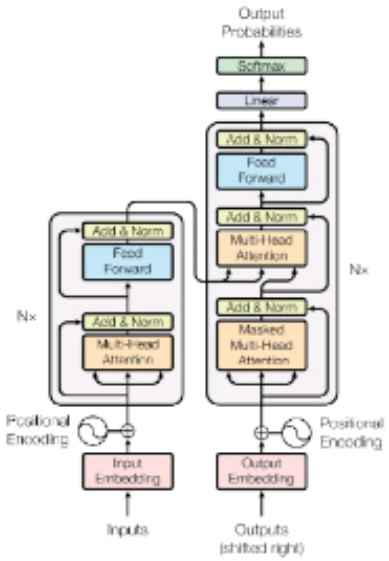


Figure 10, (Figure 1: The Transformer - model architecture(Vaswani et al., 2017b).)

3.4.1. Transformers Encoder(left side)

(Vaswani et al., 2017b) Exsplain, N = 6 identical layers make up the stack that makes up the encoder. Every layer comprises two sub-layers. A feed-forward network that is simple and fully connected based on position is the second mechanism, while a multi-head self-attention mechanism is the first. In other words, each sub-layer's output is represented as LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function that the sub-layer itself implements. In order to support these residual connections, the model's embedding layers and all of its sub-layers generate outputs with a dimension of dmodel = 512.

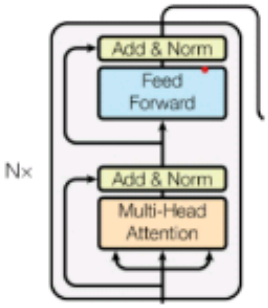


Figure 11, Transformer Encoder

The Transformers Encoder model in the Deep Cognitive Transformers architecture is used for attention gates combined with feed-forward networks. In Figure 11 before entering the encoder it will enter positional embedding, but in the Deep Cognitive Transformers Model positional is replaced with the Reasoning Model.

3.4.2. Transformers Decoder(right side)

(Vaswani et al., 2017b) Exsplain, The decoder also consists of a stack of N = 6 identical layers. In addition to two sub-layers in each encoder layer, the decoder adds a third sub-layer to perform multi-head attention on the encoder stack's output. Similar to the encoder, we use residual connections surrounding each sub-layer, followed by layer normalization. We change the self-attention sub-layer in the decoder stack to prevent positions from attending to other positions. By masking and offsetting the output embeddings by one place, predictions for position i are exclusively based on known outputs at positions less than i.

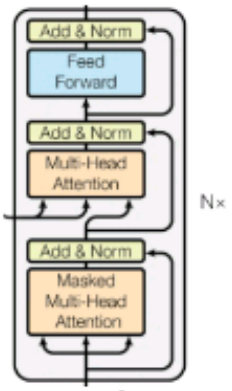


Figure 12, Transformers Decoder

Transformers Decoder Model in Deep Cognitive Transformers architecture is used for attention gates combined with feed-forward networks but added one stack and mask on the second attention, this is useful for increasing computational efficiency and distinguishing 2 paths so that they have different information extraction and provide potential new information. In Figure 12 before entering the Decoder, Data will enter the positional embedding, but in the Deep Cognitive Transformers Model, positional is replaced with the Reasoning Model.

3.4.3. Self-Attention

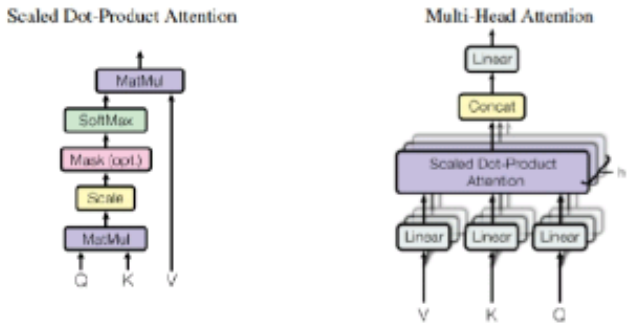


Figure 13, (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several(Vaswani et al., 2017b)

From (Peter Shaw et al., n.d.) explain that self-attention sublayers use h attention heads. The sublayer output is formed by concatenating results from each head and applying a parameterized linear transformation. Each attention head processes an input sequence $(x1, \ldots, xn)$ of n elements $(xi \in R^{dx})$ and generates a new sequence $(z1, \ldots, zn)$ of the same length $(z^i \in R^{dz})$. Each output element, z^i, is calculated as the weighted sum of linearly modified input components.

$$z^i = \sum_{j=1}^{n} \alpha_{ij} \left(x_j W^v\right) \qquad (5)$$

Formula 5 above is the calculation for the Scale Dot Product output where W^v is the input for Scale Dot Product(W^q, W^k, W^v).These parameter matrices are unique per layer and attention head(Peter Shaw et al., n.d.). The weight α_{ij} of each annotation hj is computed by(Bahdanau et al., 2014).

$$\alpha_{ij} = \frac{\exp^{(e_{ij})}}{\sum_{k=1}^{n} \exp^{(e_{ik})}} \qquad (6)$$

Formula 6 above is the softmax formula to find the weight α_{ij}, and in formula 6 there is e_{ij} which can be calculated using formula 7. e_{ij} is computed using a compatibility function that compares two input elements(Peter Shaw et al., n.d.).

$$e_{ij} = \frac{\left(\left(x_i W^q\right)\left(x_i W^k\right)^T\right)}{\sqrt{d_z}} \qquad (7)$$

formula 5 - 7 is the single-head attention formula (Figure 14, (left) Scaled Dot-Product Attention).

3.4.3.1. Multi-Head Attention

Instead of performing a single attention function with d-model-dimensional keys, values, and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively(Vaswani et al., 2017b). (Figure 15(right) Multi-Head Attention consists of several). n practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V (Vaswani et al., 2017b).

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (8)$$

Formula 8 is the complete formula for Scale Dot Product where in Multi-Head Attention it will be stacked and entered into concatenate. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this(Vaswani et al., 2017b).

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Figure 16 , (Vaswani et al., 2017b)

3.4.4. Feed Forward Network

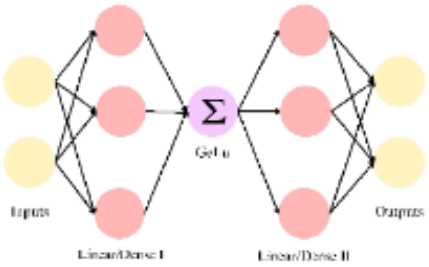


Figure 17, Feed Forward Network

(Tunstall et al., 2022) Exsplain ,According to the literature, the hidden size of the first layer should be four times the size of the embeddings, and the most popular activation function is GELU, This is where most of the capacity and memory is theorized to happen and is commonly scaled when scaling up the models.

$$\text{GELU}(x) = 0.5x\left(1 + \tanh\left[\sqrt{2/\pi}\,(x + 0.044715x^3)\right]\right) \qquad (9)$$

Since the cumulative distribution function of a Gaussian is often computed with the error function, we define the Gaussian Error Linear Unit (GELU) as(Hendrycks & Gimpel, 2016) . Formula 9 (approximate Version).

3.5. Add Layers

Add layers is a block to combine two vector paths, the Encoder and Decoder, where this is useful for combining two pieces of information from the Encoder and Decoder results in such a way that the model can remember and learn and remember information from the two paths.

3.6. Long Short-Term Memory(LSTM)

Inspired by the memory in human brains, many memory-based DCNs design various memory modules to store useful or historical information that can be later reused(Huang & Wang, n.d.). In this case, we will use LSTM as Memory for the Cognitive Transformers model.

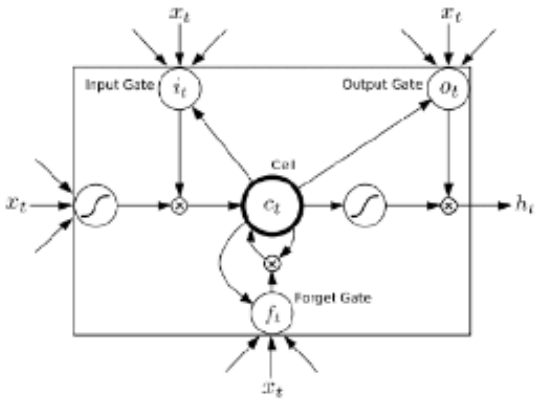


Figure 18, Long Short-term Memory Cell(Graves, 2013)

(Graves, 2013) Exsplain , RNNs often use a sigmoid function as the hidden layer function (H), However, we discovered that the Long Short-Term Memory, which uses purpose-built memory cells to retain information, is superior at discovering and exploiting long-range dependencies.

3.7. DecisionNet

In this decision model, I use the Actor Critic Network model, where this model has 2 of its own models (ActorNet and CriticNet).

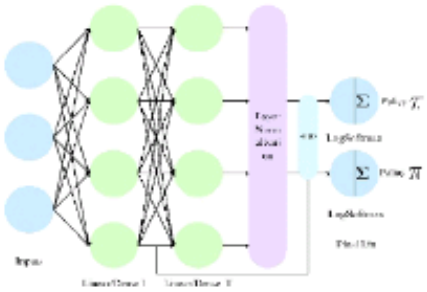


Figure 19, ActorNet

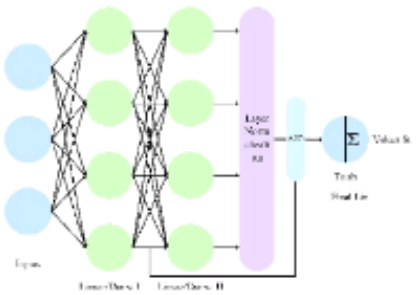


Figure 20, CriticNet

Critic network is in charge of value estimation. When agents select an action and encounter a new state, critic network estimates possible action values conditional on current state to help agent make the action in the next step(Liu et al., 2020). In figure 19 is a model for ActorNet where the model is tasked with generating probabilities (LogSoftmax, in formula 10) for each action.

$$LogSoftmax(x) = Log\left(\frac{\exp^{(x)}}{\sum_{k-1}^{n}\exp^{(x_k)}}\right) \qquad (10)$$

In figure 20 is a model for CriticNet where the model is tasked with generating the Value of the state (Tanh, in formula 11) in the state.

$$Tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})} \qquad (11)$$

4. Train Model

This Deep Cognitive Transformers model will be trained using Reinforcement learning, this is done to prove that the model can understand the environment/data being trained. The Reinforcement learning

algorithm used is the Advantage Actor-Critic algorithm where the Model will be tasked with producing Policy and Value from the state

4.1. Environment

The ultimate challenge of reinforcement learning research is to train real agents to operate in the real environment, but there is no common real-world benchmark to track the progress of RL on physical robotic systems(Kumar et al., 2019). In this study, the model will be trained in the Atari game environment, namely Space Invaders(Figure 3), Boxing(Figure 21), LaserGates(Figure 22) , BattleZone(Figure 23).

Figure 21, Boxing , From Gymnasium

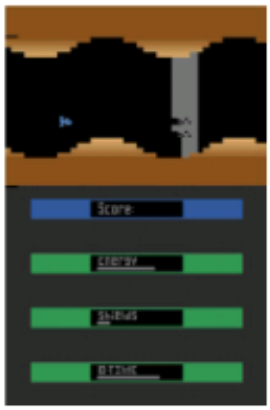


Figure 22, LaserGates from , Gymnasium

Figure 23 , BattleZone from Gymnasium

4.2. Policy

The policy used is Stochastic Policy Gradient. The basic idea is to represent the policy by a parametric probability distribution $\pi\theta\ (a|s) = P\ [a|s;\ \theta]$ that stochastically selects action α in state s according to parameter vector θ(Silver et al., n.d.). The policy selects actions stochastically based on a probability distribution, rather than deterministically (as in deterministic policies). This means that the same state can result in different actions across episodes, allowing for exploration.

4.3. Reward

A reward is an important component for agents to learn the environment, different environments have different amounts and ways to get rewards, in the Actor-Critic algorithm the reward will be processed in a mathematical function, namely the Return function.

$$R_t = 0$$

$$R_t = r_t + \gamma R_t \qquad (12)$$

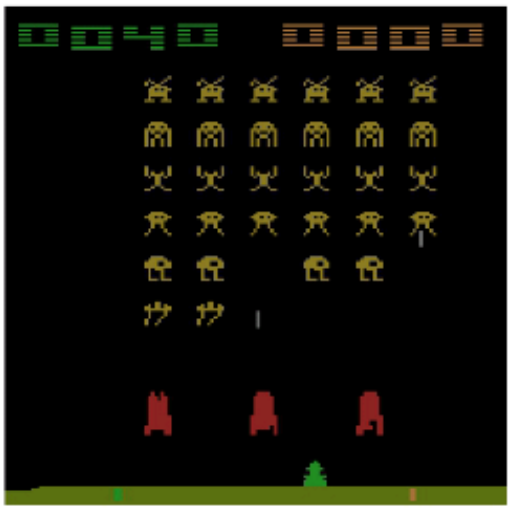
4.4. Advantage

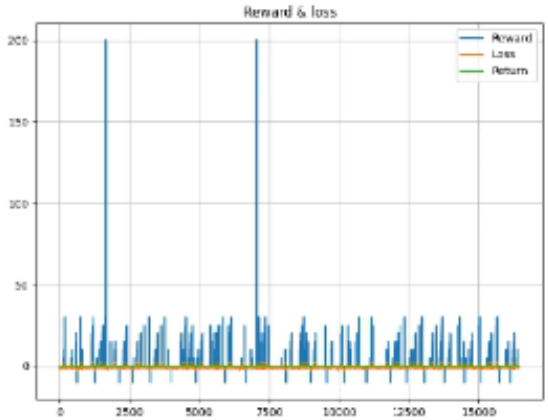
In order to reduce the number of required parameters and improve stability, temporal difference (TD) error is employed as an unbiased estimate of the advantage function(Peng et al., 2017),

$$\delta^{\pi\theta} = r + \gamma V^{\pi\theta}(s') - V^{\pi\theta}(s'). \quad (13)$$

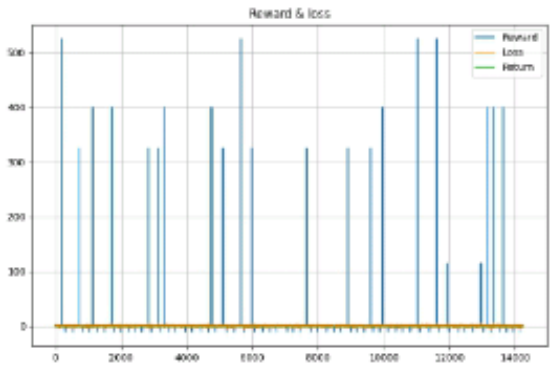
4.5. Train Result

In each environment, the agent will be trained for two hours, which means the agent has no episode limit and will run continuously for two hours non-stop.

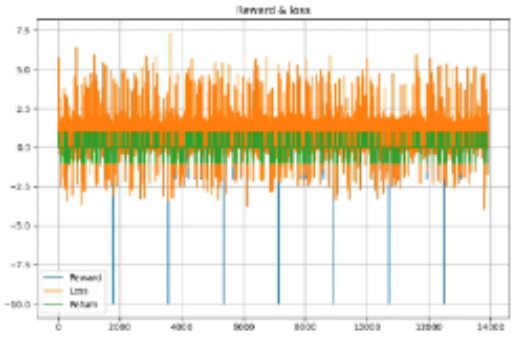




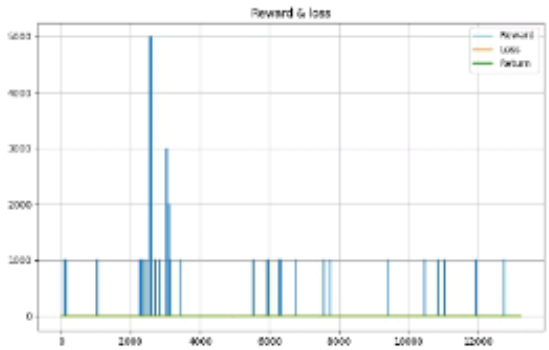
Tabel 1 , the goal is to destroy enemies by firing laser cannons at them before they reach Earth. The game ends when all your lives are lost after being hit by enemy fire, or when they reach Earth. Agents earn points by destroying enemies. Enemies in the back row have more points.The diagram above is a diagram of the results of model training in the space invaders environment. The diagram above shows good results where the Agent can understand how to get a high score and create a strategy to avoid defeat, where you can see that there is a gap in the diagram.



Tabel 2 , Laser gate game is a game where players are required to survive, destroy or pass through gates and kill enemies to survive and get rewards and optimize policies to take better actions where if they succeed in destroying the enemy, the Agent will get various rewards depending on the enemy killed, in the diagram above you can see how the Agent can understand how to get better rewards.



Tabel 3 , boxing environment is a game where the player will act as a boxer where the player will get a score if he successfully hits the enemy's head, where the agent will get a reward of 1.5 to 7.5 (max) and if the enemy successfully attacks the agent, then the agent will get a negative score (-1.5 to -7.5). The diagram above shows that the agent can understand well how to get a reward, the orange color on the diagram is the loss where the loss value will adjust the reward that the agent gets from his environment.



Tabel 4 , the Agent controls a tank and must destroy enemy vehicles. The game is played in first person and creates a 3D illusion. The radar screen shows enemies around the Agent. The Agent starts with 5 lives and gains up to 2 additional lives if you achieve a high enough score. The diagram above shows the Agent can understand how to control the tank to destroy enemy tanks and get high rewards.

5. Conclusion

The model used for Agents to process information from the environment is very efficient in creating strategies to get higher rewards by combining the concept of cognitive psychology and transformers to produce a powerful model to handle complex environments and interactive rewards where interactive rewards are the key to optimizing models and agents to better understand and make more optimal policies. And this DeepCT model, in addition to being used for deep reinforcement learning, can also

be used for several other tasks such as image classification, text classification and for recommendation systems for marketing, storage optimization and so on, in the future the model can be updated for a shorter and faster architecture.

References

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. Journal of artificial intelligence research, 4, 237-285.

Nussenbaum, K., & Hartley, C. A. (2019). Reinforcement learning across development: What insights can we draw from a decade of research?. Developmental cognitive neuroscience, 40, 100733.

Cichy, R. M., & Kaiser, D. (2019). Deep neural networks as scientific models. Trends in cognitive sciences, 23(4), 305-317.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018, July). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In International conference on machine learning (pp. 1861-1870). PMLR.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937). PMLR.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.

Perez, L., & Wang, J. (2017). The effectiveness of data augmentation in image classification using deep learning. arXiv preprint arXiv:1712.04621.

Park, W., Leibon, G., Rockmore, D. N., & Chirikjian, G. S. (2009). Accurate image rotation using Hermite expansions. IEEE Transactions on Image Processing, 18(9), 1988-2003.

Elharrouss, O., Akbari, Y., Almaadeed, N., & Al-Maadeed, S. (2022). Backbones-review: Feature extraction networks for deep learning and deep reinforcement learning approaches. arXiv preprint arXiv:2206.08016.

Zaidi, S. S. A., Ansari, M. S., Aslam, A., Kanwal, N., Asghar, M., & Lee, B. (2022). A survey of modern deep learning based object detection models. Digital Signal Processing, 126, 103514.

Muhammad, U., Wang, W., Chattha, S. P., & Ali, S. (2018, August). Pre-trained VGGNet architecture for remote-sensing image scene classification. In 2018 24th International Conference on Pattern Recognition (ICPR) (pp. 1622-1627). IEEE.

Xu, K., Zhang, M., Jegelka, S., & Kawaguchi, K. (2021, July). Optimization of graph neural networks: Implicit acceleration by skip connections and more depth. In International Conference on Machine Learning (pp. 11592-11602). PMLR.

Maniatopoulos, A., & Mitianoudis, N. (2021). Learnable leaky ReLU (LeLeLU): An alternative accuracy-optimized activation function. Information, 12(12), 513.

Sloman, S. A., & Lagnado, D. (2005). The problem of induction. The Cambridge handbook of thinking and reasoning, 95-116..

Sukhbaatar, S., Weston, J., & Fergus, R. (2015). End-to-end memory networks. Advances in neural information processing systems, 28.

Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. arXiv preprint arXiv:1410.3916.

Huang, Y., & Wang, L. (2023). Deep Cognitive Networks: Enhance Deep Learning by Modeling Human Cognitive Mechanism. Springer.

Shaw, P., Uszkoreit, J., & Vaswani, A. (2018). Self-attention with relative position representations. arXiv preprint arXiv:1803.02155.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

Tunstall, L., Von Werra, L., & Wolf, T. (2022). Natural language processing with transformers. " O'Reilly Media, Inc.".

Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Graves, A. (2013). Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Liu, C. L., Chang, C. C., & Tseng, C. J. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *Ieee Access*, *8*, 71752-71762.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014, January). Deterministic policy gradient algorithms. In *International conference on machine learning* (pp. 387-395). Pmlr.

Peng, B., Li, X., Gao, J., Liu, J., Chen, Y. N., & Wong, K. F. (2018, April). Adversarial advantage actor-critic model for task-completion dialogue policy learning. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 6149-6153). IEEE.