國立成功大學

資訊工程學系

深度學習

期末專題報告

***Dream a Fighting Game with Attention***

組員：P76094169 黃仁鴻

授課老師： 陳奇業 教授

中華民國110年1月

# 目錄

[目錄 i](#_Toc60705229)

[表目錄 ii](#_Toc60705230)

[圖目錄 iii](#_Toc60705231)

[一、簡介 1](#_Toc60705232)

[二、問題描述 1](#_Toc60705233)

[三、研究方法 2](#_Toc60705234)

[3.1 Renderer Module 3](#_Toc60705235)

[3.2 Driver Module 5](#_Toc60705236)

[四、實驗成果與討論 6](#_Toc60705237)

[5.1實驗環境 6](#_Toc60705238)

[5.2成果與討論 6](#_Toc60705239)

[五、結論 7](#_Toc60705240)

[參考文獻 8](#_Toc60705241)

# 表目錄

[表 1環境配置 6](#_Toc60707037)

# 圖目錄

[圖 1各模組間的運作方式 2](#_Toc60707270)

[圖 2 Renderer Module 3](#_Toc60707271)

[圖 3 因應量化問題的分區優化 4](#_Toc60707272)

[圖 4 使用Transformer Decoder作為Driver Module 5](#_Toc60707273)

[圖 5 強化學習架構 7](#_Toc60707274)

# 一、簡介

人類具有優秀的推理及預測能力，即使將過去曾經遊玩過的遊戲於腦海中重建亦非難事，甚至於可以於夢中再現遊玩的情境。

那如果用類神經網路來重現這個能力，就代表著我們可以此方式，在不對程式進行的修改的情況做到跨平台執行，進而降低許多維護成本。

為此，本次實驗將會使用類似 World Model [1]的結構，嘗試將無隨機系統的自製小型格鬥遊戲複製出來，讓其並於網頁上執行。

# 二、問題描述

本次研究將會利用類神經網路重現出實驗所給定的格鬥遊戲，在完成此目標的研究過程中會遇到以下兩個主要問題：

1. 如何讓模型依據過去的狀態與玩家的輸入生成出下一幀的狀態。
2. 模型將實時生成遊戲畫面，為了保證達成此目的需要控制模型的計算量。

# 三、研究方法

本次實驗分成由Auto Encoder構成的Renderer Module與Transformer [2]構成的Driver Module兩個模塊。如圖1所示，Driver Module會利用自回歸的方式不斷生成出下一幀的隱藏狀態，並交由Renderer Module將其轉變成遊戲畫面。

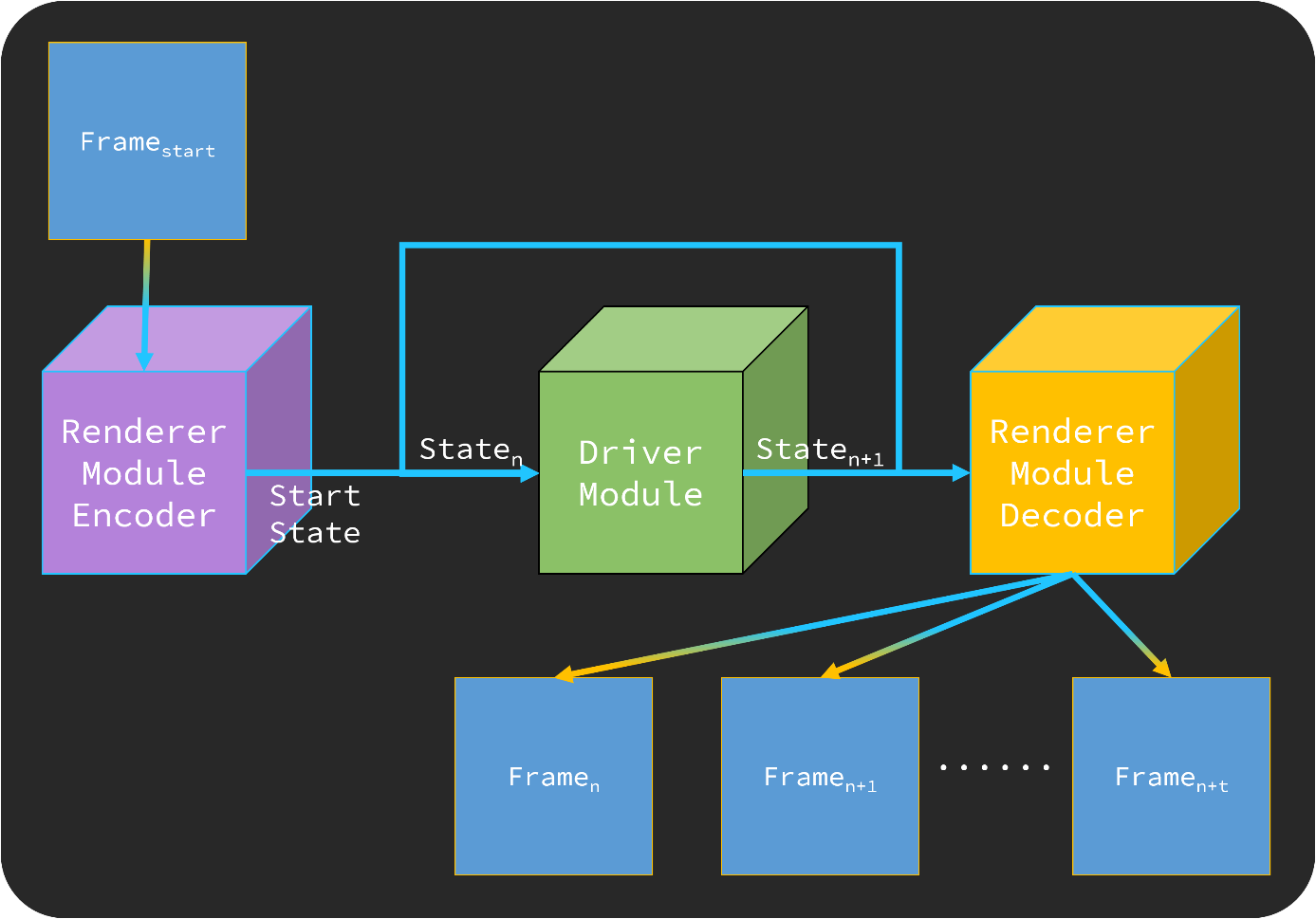


圖 1各模組間的運作方式

### 3.1 Renderer Module

如圖2，Renderer Model分為把遊戲畫面壓縮成狀態編碼的Encoder，以及將狀態編碼轉回遊戲畫面的Decoder。在研究過程中發現如果將狀態編碼量化，在後續Driver Module生成時，具有比連續的狀態編碼更加穩定的結果。

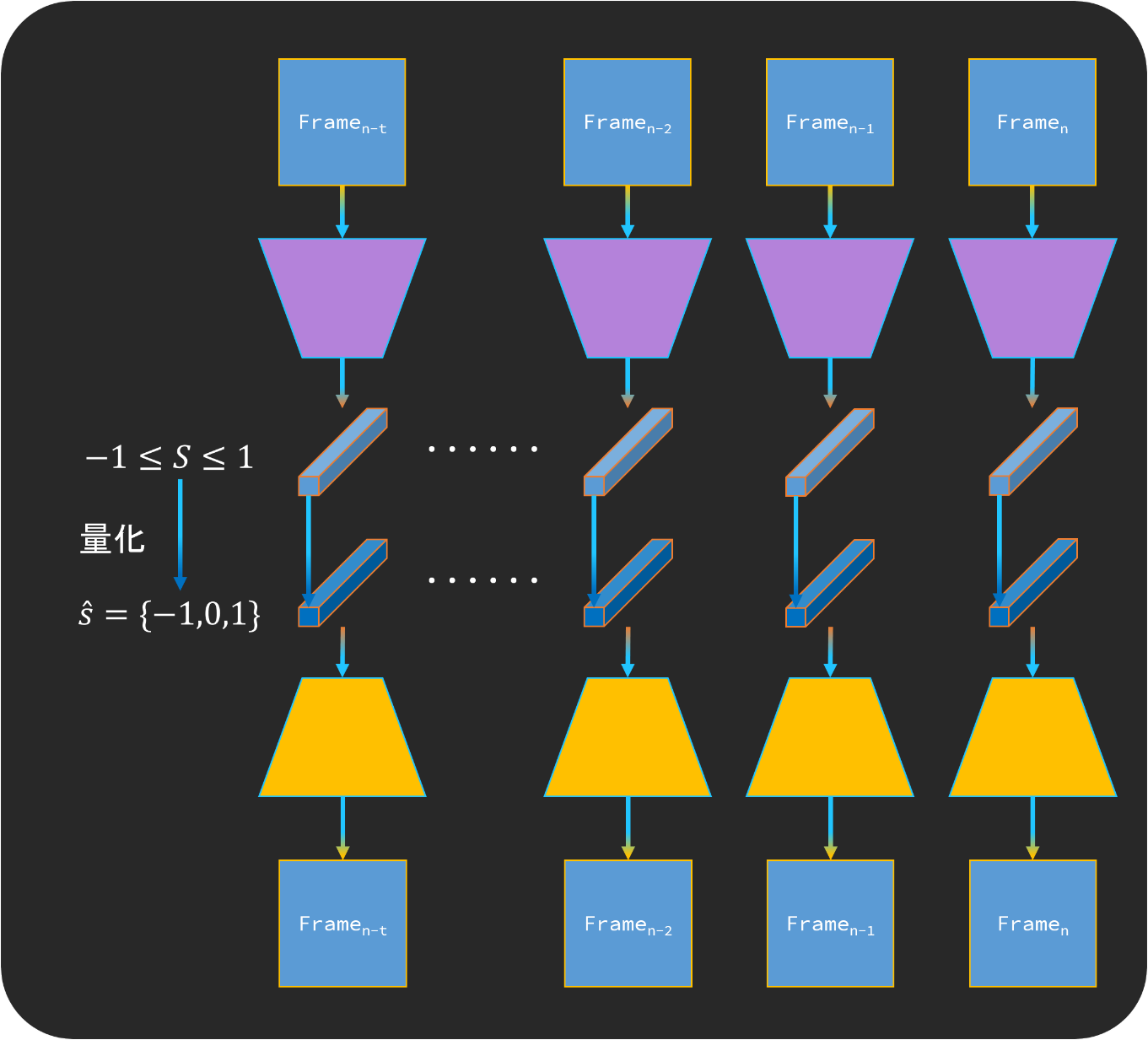


圖 2 Renderer Module

這邊所使用的量化方式是先對Encoder的輸出執行tanh，使其值域介於-1~1之間，再將其四捨五入得到{-1,0,1}的量化結果，但這種方式無法直接使用反傳遞演算法來優化模型。

為了對應此問題，使用圖3的方法將Encoder與Decoder的梯度分開計算。Encoder會將連續的State Embedding傳給Decoder生成輸出畫面，但只會對Encoder執行優化。而Decoder則是使用量化後的State Embedding作為輸入並執行優化。

另外，如果直接使用MSE計算連續的State Embedding與量化的State Embedding的Loss，或是在Decoder計算Loss時將連續的State Embedding與量化的State Embedding一起使用，都會使Renderer Module嚴重毀損。

為了增加Renderer Module的容錯能力與編碼能力，除了使用到量化的方法外，還會髓機把State Embedding部分的值以{-1,0,1}替換掉。

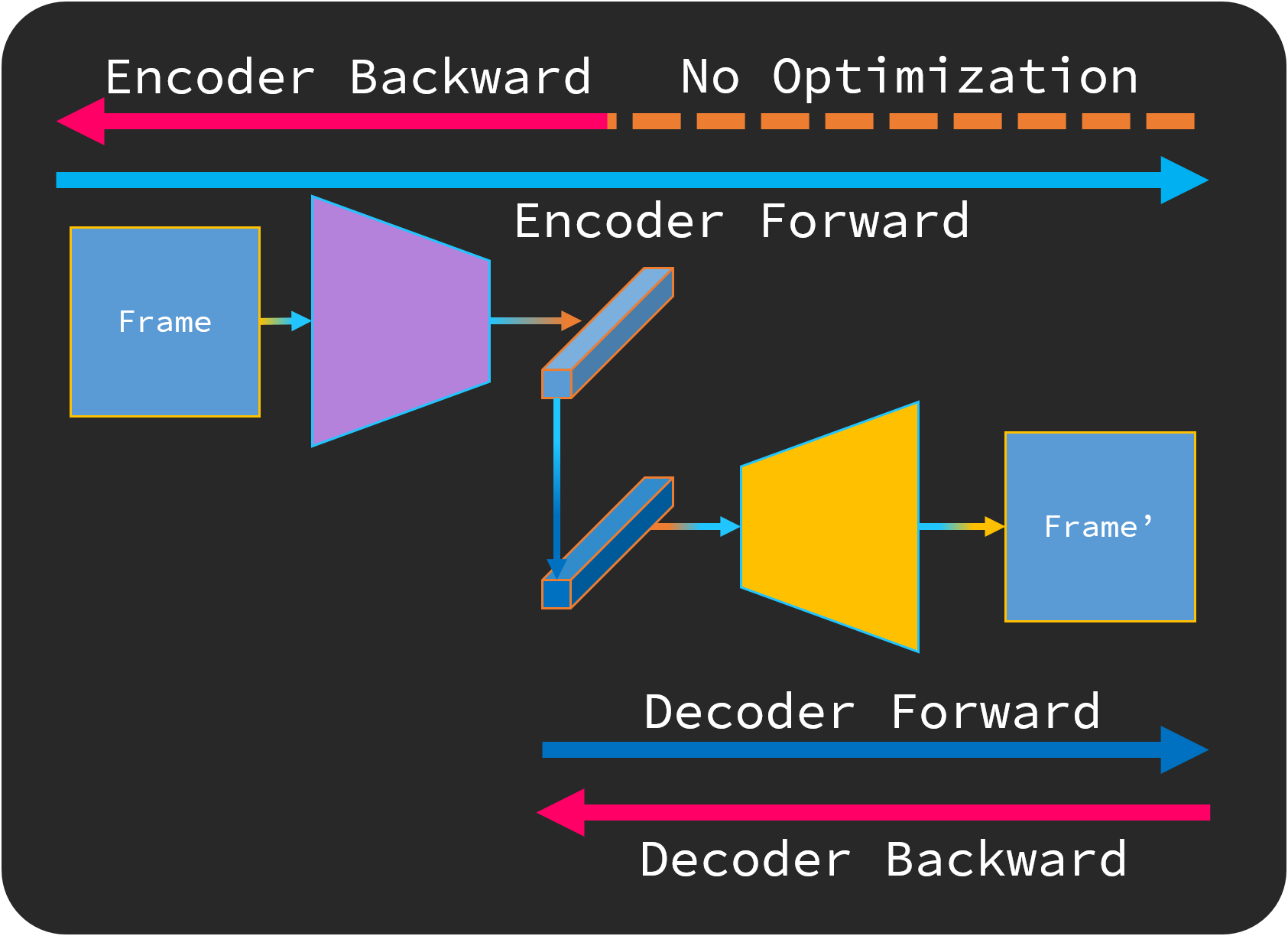


圖 3 因應量化問題的分區優化

### 3.2 Driver Module

Driver Module的目標是利用過往的狀態與使用者的操作，生成出下一幀的State Embedding (如圖4)，因此在訓練過程中要先依靠Renderer Module的Decoder將圖片轉為State Embedding，並由訓練資料中紀錄的使用者操作提取對應的Act Embedding。而作為主結構的Transformer Decoder本身不具備分辨位置的能力，因此還需另外加上式(1)的Position Encoding。

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

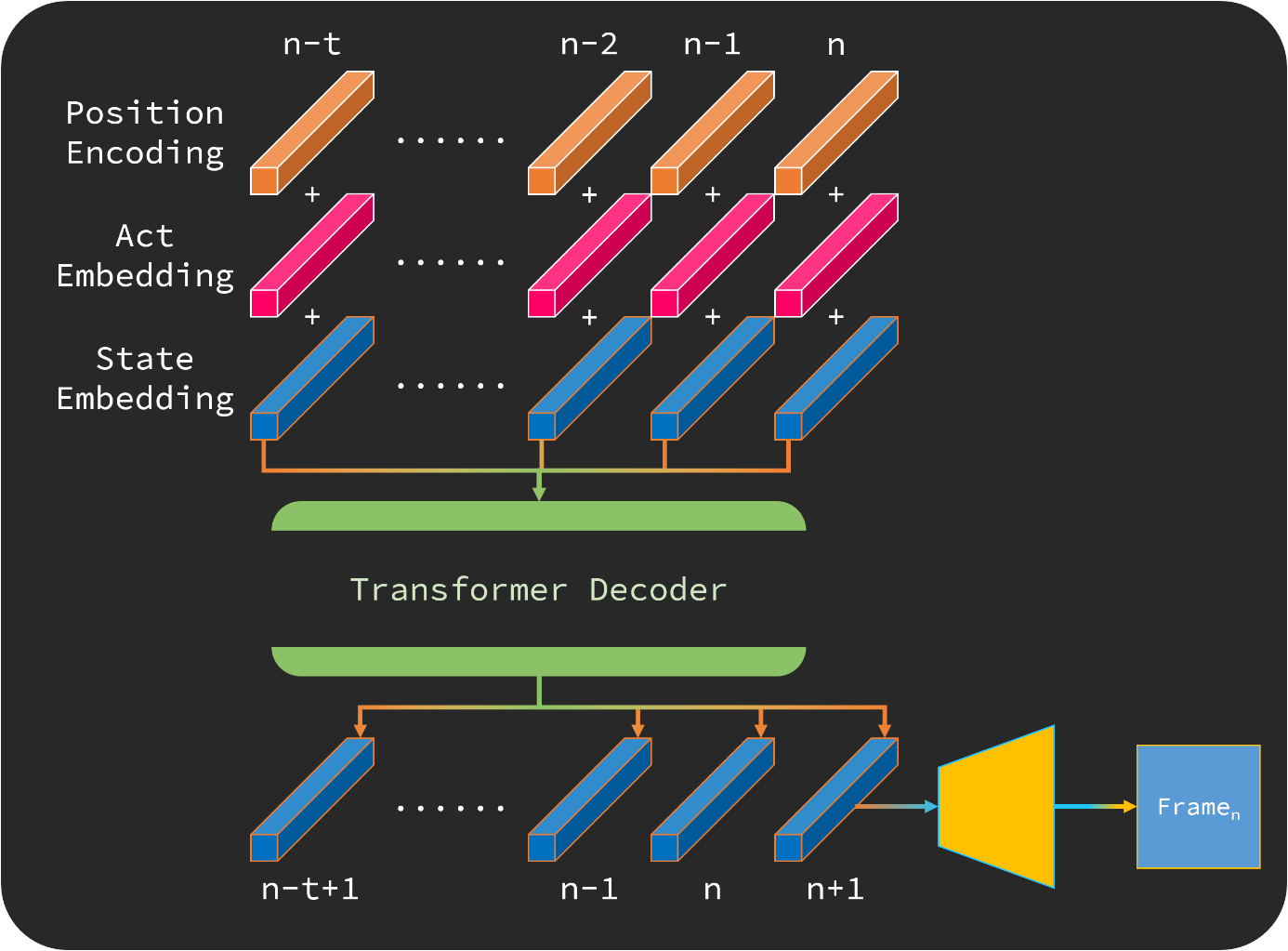


圖 4 使用Transformer Decoder作為Driver Module

# 四、實驗成果與討論

### 4.1實驗環境

表 1環境配置

|  |  |
| --- | --- |
| **處理器** | Intel(R) Core(TM) i5-7400 CPU @ 3.00GHz |
| **記憶體** | 16.00GB |
| **圖形處理器** | NVIDIA Geforce GTX 1050 Ti |
| **作業系統** | Windows 10 64 位元作業系統、Linux Mint 64 位元作業系統 |
| **開發環境** | Chrome 87版以上、Node.js |
| **使用語言** | TypeScript |
| **函式庫** | Tensorflow.js、Babylon.js |

表1是本次實驗的環境配置。運用TypeScript跨平台的優勢使其能於各作業系統上快速建置開發環境，此外Tensorflow.js可利用webGL API調用GPU的平行運算能力加速計算速，並且不必像Tensorflow.py還需要另外安裝CUDA。

### 4.2成果與討論

在實驗過程中發現如果使用一般連續型Auto Encoder生成出的狀態編碼，會造成Driver Module的生成結果變得混亂且無法控制。改用量化型 Auto Encoder後則大幅的改善了這個問題。

而整個模型是否能成功學習如何重現目標任務，則與訓練資料的穩定度有很大的關係，最開始實驗中嘗試使用隨機操作的遊戲紀錄做為訓練資料，但是每幀都進行隨機操作大大的干擾了Driver Module的預測能力，使其生成結果常常與輸入的指令無關。

# 五、結論

雖然目前經過大量調整後取得了比一開始較為穩定的結果，但是Driver Module的能力依舊有待加強。在最終的實驗裡只學會了跳躍、蹲下與不完整的攻擊。即便是只學習特定動作情況下，對於左右位移還有延遲性觸發的操作，其生成效果都不是很理想。

此外，可能是因為使用了量化State Embedding的緣故，雖然穩定卻使得模型缺乏自主泛化不同狀態的能力，可能還需要配合像是DQN [3]等等強化學習的方法(如圖5)來廣泛探索各類狀態，並使State Embedding更加貼近當下的狀態。

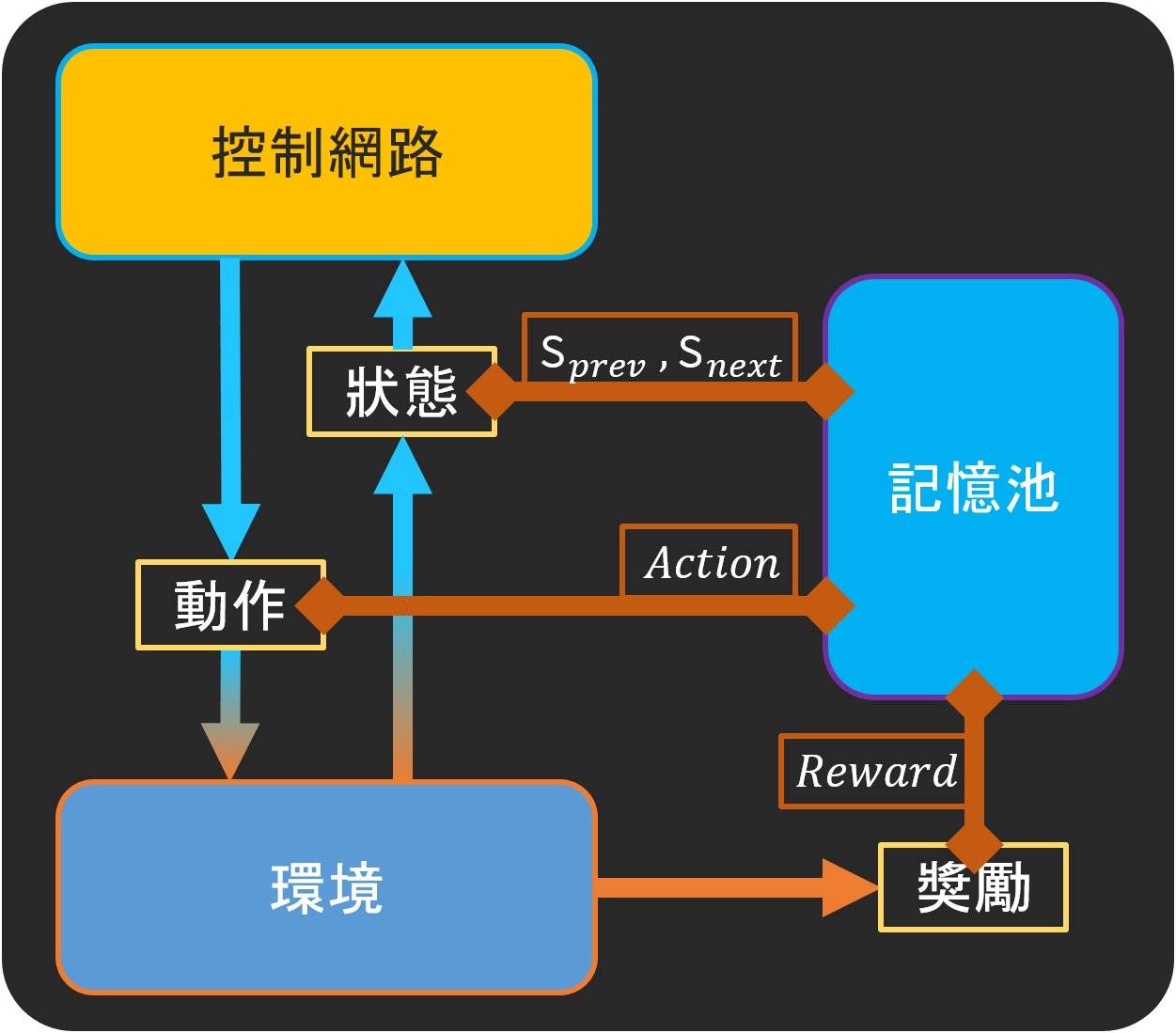


圖 5 強化學習架構

# 參考文獻

1. David Ha, Jürgen Schmidhuber, “World Models,”

arXiv preprint arXiv:1803.10122, 2017.

1. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, “Attention Is All You Need,”

arXiv preprint arXiv:1706.03762, 2017.

1. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller, “Playing Atari with Deep Reinforcement Learning,” arXiv preprint arXiv:1312.5602, 2013.