Playing Space Invaders with Deep Learning

Devin Muzzy dmuzzy@ucsd.edu
Aidan Bell abell@ucsd.edu
Isaiah Dailey idailey@ucsd.edu
Aditya Melkote avmelkote@ucsd.edu

Abstract

In this study, we investigate multiple neural network training methodologies for autonomously playing the Atari game Space Invaders. Initially, we attempt to replicate the seminal Deep Q-Learning (DQL) approach ([6]), laying the groundwork for our experimentation with alternative strategies aimed at enhancing game performance. These explorations include a supervised approach using a Vision Transformer trained on human gameplay data to predict in-game actions; and an implementation of a Dueling Network Architecture integrated with DQL (as described in [12]). Our results reveal varied success across methodologies, offering insights into the potential and limitations of each approach. Key Findings include identifying hyperparamters that train faster than [6], and the effectiveness of Vision Transformers in imitation learning over a baseline CNN.

1 Introduction

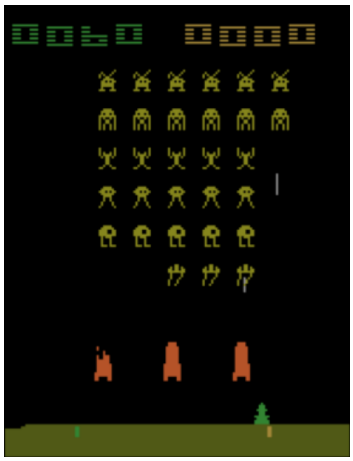

Figure 1: A standard frame of the Atari game *Space Invaders*

The original *Playing Atari with Deep Reinforcement Learning* paper [6] explores the possibility of training agents to play various Atari games using Deep Q-learning. This training algorithm saw some success in playing Atari games, but was not able to match human performance on several games, including Space Invaders. Their later Nature paper released in 2015 [7] had much better performance on all games, and featured a network with more parameters in CNN and FC layers. In this paper, we experiment with several methods for improving these results for Space Invaders.

To start, we reproduced the results from [6] by implementing their architecture. We originally ran into some difficulties given our performance and compute limitations (training was too slow), but we were able to overcome these by using the implementation in the reinforcement learning library

Stable-Baselines3 ([11]). SB3 allowed for parallelization while training so we could train with multiple environments running at once, and allowed for larger replay buffer sizes with more efficient memory use, given our limited VRAM on our personal machines. We were able to reproduce the results in [6], and created an agent that could competently[1] play Space Invaders.

Next, we experimented with training an image classifier to play. The Atari Grand Challenge Dataset [5] has created the opportunity to build models for traditional reinforcement learning tasks without the need to interact with the environment. This dataset contains 1,766,078 frames of Space Invaders spread across 423 different episodes of game play. To take advantage of this dataset we trained a model to and predict human actions. For this model we used a Vision Transformer architecture based on the An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale paper [3]. With this we were able to achieve near-human accuracy on Space Invaders without the model ever actually playing or simulating the game.

As a baseline for our transformer model, we train an offline model to predict expert actions with the DQN architecture decribed in [6]. We additionally fine-tune this model with online DQN after training, to test if DQN can benefit from transfer learning.

Next, we reproduced the *dueling network* model architecture described in [12], in which two output layers with two different loss functions are trained simultaneously: One aims to predict the state value function, and one predicts the state-dependent action advantage function. We were not able to reproduce the significant performance gains found in *ibid.*, and we suspect there is an implementation error (we implemented it by modifying the standard Deep Q-network implementation found in Stable-Baselines3 ([11])). Unfortunately, due to time constraints, we were not able to adequately troubleshoot the issue, and more work should be done to try and get a better implementation. Nevertheless, we were still able to achieve good performance, and the model even achieved scores as high as the mid $2,000$'s on several playthroughs (although of course, the average score was much lower). Here is a link to a recording of the model playing such a game.

Along the way, we also tried reproducing the results in the paper *Fully Online Decision Transformer for Reinforcement Learning* ([9]), but decided to abandon this approach after we were unable to reproduce the results therewithin. The approach taken in this paper is a modification of the decision transformer architecture, originally outlined in [1].

2 Related Work

Q-learning was originally introduced in 1989 as an algorithm for "learning from delayed rewards" by Chris Watkins in his Ph.D. thesis ([13]), and was later expanded on by Watkins & Dayan in [14], where it was first called Q-learning. Watkins' algorithm worked by learning the reward obtained by performing an action a in a state s. Unfortunately, as a training strategy, Q-learning is infeasible for problems with a very large number of states and actions, as in standard Q-learning one needs to learn a table of rewards for every possible state-action pair. In 2014, Google DeepMind applied Q-learning to create "Deep Q-learning" ([6]), in which rather than learning a table of reward values for each state-action pair, a neural network is trained to predict reward values. This allowed the Google team to perform reinforcement learning for problems which were previously intractable, for example, in order to train a CNN to play Atari games, such as Space Invaders.

The introduction of transformers allow for modeling the reinforcement learning problem framed as a Markov Decision Process into a sequence learning problem, to which transformers can be used to predict optimal actions. Some intermediary advancements have been made, such as Deep Transformer Q Networks, which utilize self attention to solve RL problems better then their conventional counterparts. A transformer learns the positional encoding to represent an agent's history, which is used to predict Q values at each time step that are fed into the rest of the reinforcement learning model ([4]). While we have decided against implementing decision transformer architecture, we have still decided to use transformer architecture in our work, due to the astounding results transformers have been able to achieve in machine learning subfields besides reinforcement learning.

Quantile Regression Deep-Q Learning (QRDQN)[2] is the current top performer of Stable-Baselines3 model performance at 10M training frames in Space Invaders on the Stable-Baselines3 zoo[11], achieving an avg score of 1899. QRDQN is able to learn much faster than other Deep-Q learning

[1] Depending on your definition of "competent".

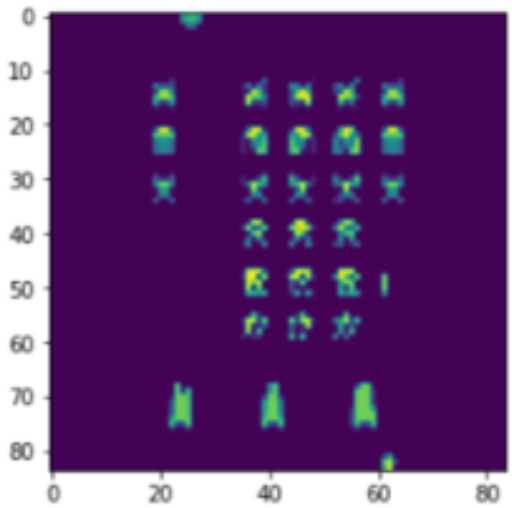

Figure 2: Example 84x84 input frame to our DQN network

variants. Instead of having random variation in interactions with the environment "average out" over time to result in an accurate value function, QRDQN explicitly estimates the distribution as it is learning.

3 Methods

3.1 Deep-Q Network Baseline

For our baseline, we sought to replicate the reults of "Playing Atari with Deep Reinforcement Learning"[6]. We implemented the experience replay deep Q learning algorithm with the hyperparameters detailed in the paper. The baseline model takes the last 4 frames of game data as input, where each frame is of size 84x84 (grayscale). The baseline DQN architecture is comprised of 2 convolutional layers followed by 2 fully connected layers. The first convolutional layer has 32 8x8 filters with stride 4 (with RelU activation function[8]). The second convolutional layer has 64 4x4 filters with stride 2, followed by RelU. The last hidden layer has 256 fully connected rectifier units, and is fully connected to the the output layer with 6 outputs. The outputs from the network predict the Q-value for each of the 6 valid Space Invader actions given the input state (last 4 frames of gameplay). To implement the deep-q learning network (DQN) in PyTorch, we modified the example DQN implementation for the CartPole-v1 environment on the PyTorch website [10] to match the DQN implementation details of [6]. We had to decrease the experience replay memory from 1,000,000 to 50,000 due to memory constraints. After 8 hours of training, DQN with Mnih et al. (2013)[6] hyperparameters did not significantly outperform the baseline (Figure 4). Because we wanted to test modifications to this DQN, we reasoned that this slow training procedure would not work for the limited time window we had for this research. In order to achieve a high scoring policy more quickly, we modified the hyper-parameters of [6] (Figure 5). We increased batch size from 32 to 128, and updated policy weights based on memories 3x as often. Performing more updates only increased our time per episode by 25%, but significantly increases reward propagation through Q states, making average Q value for actions increase much more quickly. We additionally allow the network to choose an action in response to every frame during training, while the original paper only allowed agents to choose an action every 3 frames.

Our modified DQN achieved much better results within 20,000 episodes (24hrs training), but it was still unable to match Mnih et al. (2013)[6] performance. We pivoted to use StableBaselines3's[11] DQN with the parameters from the later DQN Nature paper Mnih et al. published in 2015 [7].We trained in parallel with 4 workers for 36hrs on a NVIDIA GTX 1080 Graphics card, and thanks to StableBaselines3 memory optimizations, we were able to use a replay buffer length of 250,000.

3.2 Dueling Network for DQN

To improve upon our baseline DQN, we implemented the *dueling network* architecture as described in [12] within our DQN, which uses two separate streams that estimate a state value as well as the advantages of taking each action. Both streams share the same convolutional feature learning module, which outputs learned features used by the action-advantage function estimator stream as well as the state value function estimator. Both are then aggregated together and an estimate is produced for the state-action function Q. See Figure 3 for a diagram comparing the standard Q-network architecture and the dueling Q-network model architecture we used.

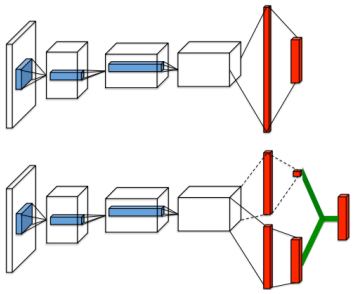

Figure 3: The single-stream Q-network (**top**) and the dueling Q-network architecture (**bottom**) (figure taken from [12, Figure 1])

The terminology "dueling networks" can be potentially misleading, especially in the context of a single-player game like Space Invaders. The network is not playing itself in any sense, rather, two different networks are being trained to perform two different functions. For this reason, we believe the terminology "dueling networks" is rather unfortunate, but it is standard now, so we continue to use it.

The state-action value function Q is as following:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \max_{a'} A(s,a';\theta,\alpha))$$

For those not familiar with the standard meanings of the relevant variables in this equation, we refer the reader to [12], where it may be found as Equation (8).

Our architecture from the DQN model has the following changes to support the dueling network. We introduce two branches after the convolutional layers, each of which takes in the flattened output from the last convolutional layer to which ReLU is applied. Then, using the state-action value function, the value and the advantage are aggregated together and returned in the forward function of our network. Once again, to train on UCSD's datahub with 8gb of vram, we made modifications to the experience memory, as well as made modifications to the parameters in the network.

This provides our Q-learning algorithm with the foundation for learning action policies from its environment. For our implementation, we modified the implementation of Deep Q-networks in Stable-Baselines3 ([11]) to add the custom loss functions and updated architecture, so as to implement our dueling Q-networks, as this architecture is not currently natively implemented in Stable-Baselines3.

3.3 Vision Transformer

Another method we tried to use to have a model play space invaders was to do imitation learning on a dataset of expert game play. To do this we trained an image classifier to classify a screen from the dataset as the action the human player took at that screen. Then for testing on our gym environment we fed the image classifier the start frame and then took the action that was the argmax of the logits.

As a baseline, we fit the DQN model architecture (2 CNN layers, 2 FC layers)[6] to our training data. However for our primary image classifier architecture we focused on a Vision Transformer copying the architecture in the paper by Dosovitskiy et al[3]. We made two variants of the vision transformer, one that took a 190x190 resolution screen in RGB, and another that took 3 84x84 screens in grayscale.

To gather the expert gameplay we parsed through the Atari Grand Challenge dataset to select only games where the player achieved an excellent score. In doing this we were able to reduce the size of the dataset to around 300k screens with the corresponding expert action for each. Many of the actions that the users took were NOOP actions, choosing to do nothing for that exact screen. This lead to a large class imbalance problem that we addressed by doing multiple attempts of class reweighing. Additionally the dataset contained some actions that were not available in our gym environment, this includes moving while shooting left and right. However our model vary rarly predicted these actions. In the cases it did predict one of these invalid actions during inference we replaced it with a random action from the valid set.

4 Results

Method	Average Space Invaders Score
Random[6]	179
1) DQN (reference)[7]	1976
2) DQN (our replication)	431
3) DQN (using SB3[11] and settings from [7])	763
4) DQN with dueling networks	769
5) Offline with transformer	295 (validation acc: 0.35)
6) Offline with DQN architecture	195 (validation acc: 0.32)
7) Shoot Only in OpenAI Gym	285
Human [6]	3690

Table 1: Approaches & Results.

In Table 1, 1) was trained for 50 million frames. 3) and 4) were evaluated after 20 million frames of training, and reported scores are the average scores of 5,000 validation game playthroughs. 2) was the best result attained from our modified DQN, and was trained for 17,000 episodes and validated over 200 test games. 5) and 6) are the average scores from 200 test games.

4.1 RL approaches to Space Invaders

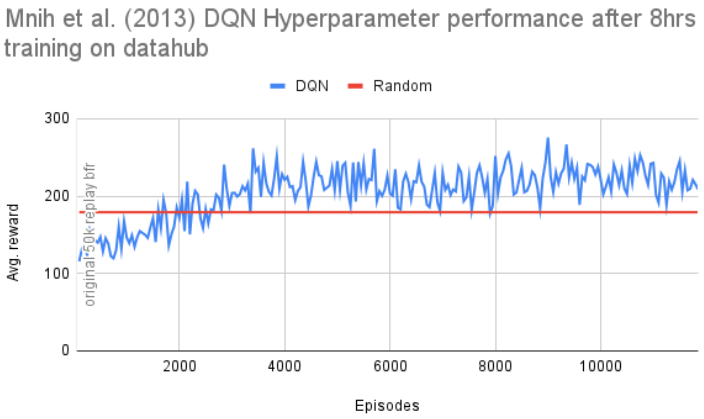

Figure 4: Our initial replication of the original DQN architecture[6] was not much better than random chance after 8hrs of training.

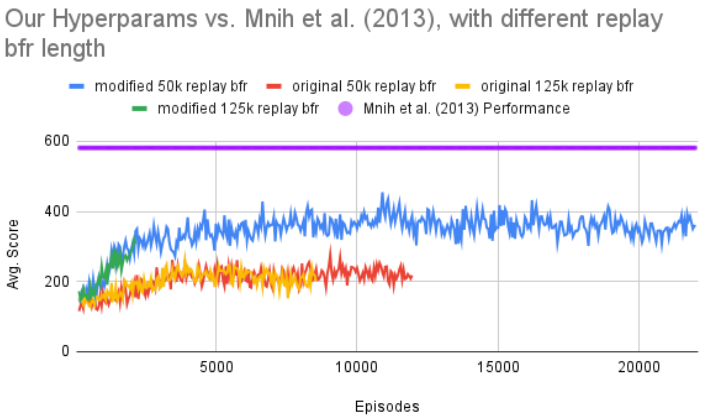

Figure 5: Performance of DQN with "modified" hyperparameters vs. the "original" hyperparams[6]. We hypothesized that our much lower replay buffer (50K vs the original 1M) was the cause of our poor performance, but after increasing memory to 125K by running locally, performance did not significantly improve.

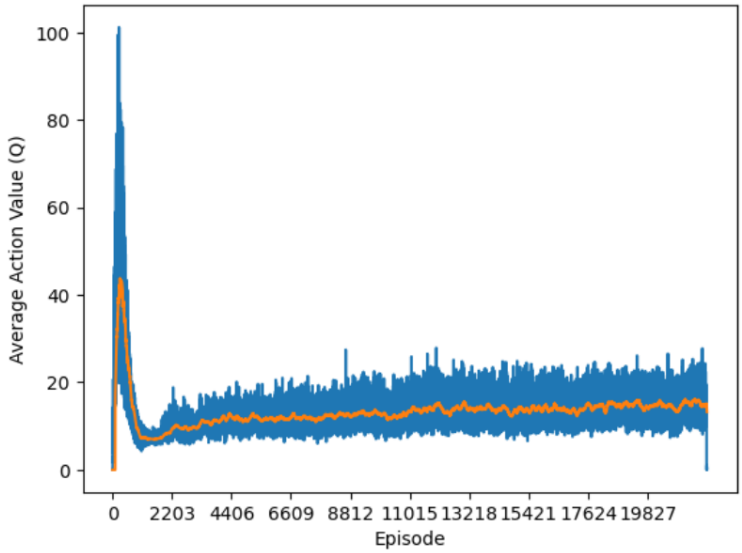

Figure 6: Average chosen action Q-value for DQN with our modified hyperparamaters over 20,000 episodes. Note the large initial spike in Q-Values that stabilizes after 1000 episodes. This spike was not present when using default hyperparameters, and was rather a gradual increase[6]

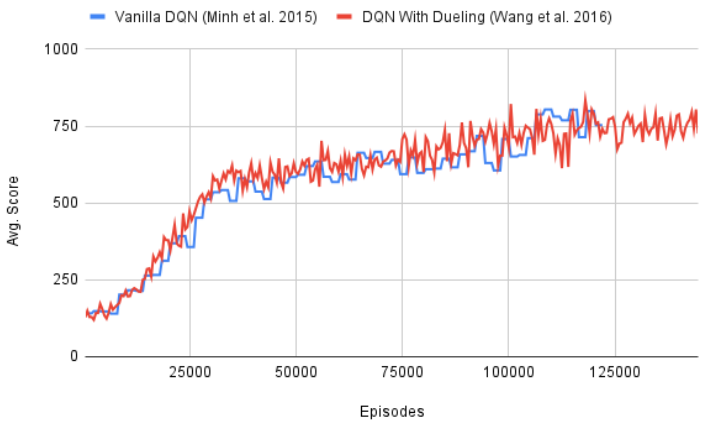

Figure 7: Blue is our replication of [6] using StableBaselines3, trained over 21M frames. Red is a modification of this baseline that implements Dueling[12]. Adding dueling very slightly improved performance relative to the baseline, but this improvement is certainly not statistically significant. Training the dueling networks architecture took a bit over eleven hours on an Nvidia RTX 3080 Ti, and we used a replay buffer size of 250,000 with four simultaneous agents.

4.2 Supervised approaches to Space Invaders

Figure 8: Score for the 190x190 cropped single image full color Vision Transformer

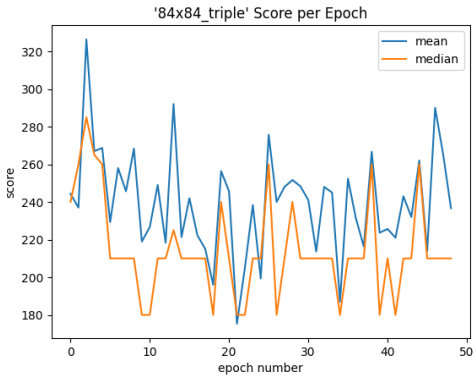

Figure 9: Score for the 84x84 multi frame gray-scale Vision Transformer

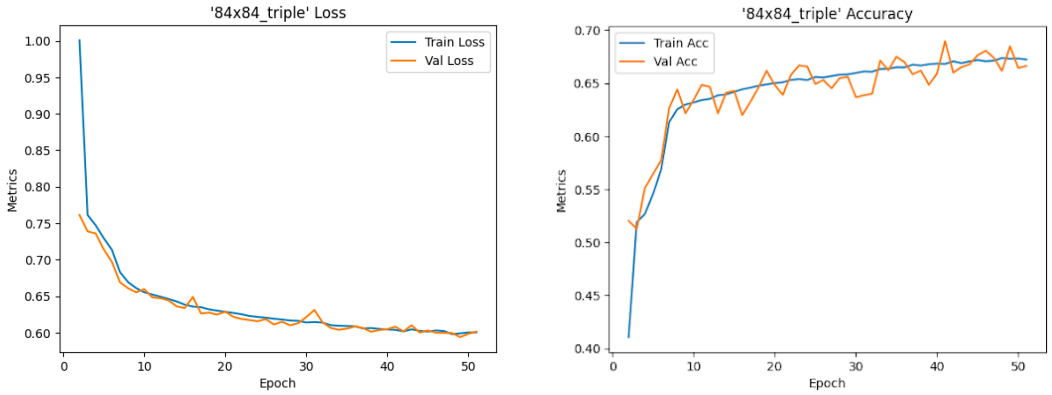

Figure 10: Vision Transformer with 84x84 grayscale image inputs training and validation plots. left: Loss, right: accuracy

5 Discussion

5.1 Replication of DQN performance on Space Invaders[7]

In order to achieve results comparable to [6] using DQN experience replay, we had to increase batch size to 128 and update weights 3x as often. Training time took 2 seconds per episode. In our testing, we discovered that the most up-to-date OpenAI Gym environment for SpaceInvaders (v5) makes enemy bullets completely invisible in the RGB input given to the model as an observation. Our best model after 17,000 episodes (Fig. 2) was trained on SpaceInvaders (v5) and learned to hide behind the barriers, but could not dodge enemy bullets. To our surprise, we were not able to achieve better results when switching to SpaceInvaders (v2) where bullets were visible. None of our custom models were able to successfully learn to dodge enemy bullets. We hypothesize that bullet dodging would have been learned if we were able to train for more than 20K epochs. but due to compute and time constraints we were unable to do this.

In retrospect, we should have started with StableBaselines3's[11] DQN archictecure and made modifications to this stable baseline from there. We got caught up attempting to replicate the results of Mnih et al. 2013[6] using limited compute resources, when a stable (and importantly, optimized) replication of the results already existed. We were not able to replicate the final average score of 1976 achieved by [7]. However, this reported score was achieved after training for 50 million frames, while our network was only trained for 20 million frames.

5.2 Dueling DQN

Our addition of Dueling[12] to DQN failed to significantly improve Space Invaders performance over 20M frames of training relative to the baseline. This contrasts with the original paper *ibid.*, in which a statistically significant improvement was seen by implementing the dueling architecture over the standard Q-learning architecture from [7]. At this time, we are not quite sure what the issue is with our implementation, assuming there is one. It is also possible that our model would have benefited from more training time, although this is unlikely given better results were seen sooner when using the dueling architecture in [12].

5.3 Vision Transformer

Another of our approaches was to train a model using Vision Transformer [3] to do imitation learning from the expert game play. We ended up with two primary models, one that took a single frame in RGB as input in 190x190 resolution, and a second model that took 3 gray scale images with 84x84 resolution in as its input. Training the models was much slower than we anticipated on the hardware we had access to. However training overnight seemed to work well and the model learned the dataset as shown in Figure 6. Our higher resolution model demonstrate better than random and better than shoot only with some noticeable improvement as training went on. Additionally, our

vision transformer significantly outperformed a CNN (with DQN architecute[6]) in actually playing space invaders(Table 1), despite achieving similar validation accuracies for next action prediction (DQN val acc: 0.32, Vision transformer val acc: 0.35). It is possible that our CNN did not perform well because its hyper-parameters were tuned for 84x84 images, and we fed it 190x190 images.

On the other hand our triple input model did not seem to perform well at all. Additionally the triple input model showed no sign of improvement with further training, and remained worse than than only shooting.

A third model was briefly trained with the same architecture as the triple image model to take a single $84x84$ image with RGB color. However for this training we were only able to run a few epochs and we were not able to get reasonable loss metrics. In the future, training a model with the same size on a single image could provide insight into if it is the downsizing, gray scale, or some other issue that causes the triple image model to perform so much worse.

6 Contributions

Devin: Worked on the Vision Transformer model with for the Space Invaders dataset. Wrote about it in report, generated graphs, helped write introduction.

Aidan:Worked on baseline models, including modifying hyperparameters and eventually implementing stablebaselines3, generated graphs for baselines and worked on methods, results, and discussion for baseline models.

Isaiah: Helped write document, perform literature search. Helped run and train models. Wrote code to generate graphs, record videos, train & test model.

Aditya: Worked on initial DQN implementation and DQN-transformer implementation, as well as on dueling network implementation. Wrote parts of methods, results related work and discussion.

References

[1] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling, 2021.

[2] Will Dabney, Mark Rowland, Marc G. Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression, 2017.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[4] Kevin Esslinger, Robert Platt, and Christopher Amato. Deep transformer q-networks for partially observable reinforcement learning, 2022.

[5] Vitaly Kurin, Sebastian Nowozin, Katja Hofmann, Lucas Beyer, and Bastian Leibe. The atari grand challenge dataset, 2017.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015.

[8] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[9] Austin Anhkhoi Nguyen and Creighton Glasscock. Fully online decision transformer for reinforcement learning. https://ngaustin.github.io/images/fully_online_dt_report.pdf, 2023. Manuscript written for a class, not formally published.

[10] PyTorch. Reinforcement learning (dqn) tutorial. https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html, 2024. Accessed: 2024-03-17.

[11] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[12] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning, 2016.

[13] Christopher Watkins. Learning from delayed rewards. 01 1989.

[14] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.