# Improving Deep Q-Learning for Pac-Man: ResNet-Based DQN with Tabular TD-Target Pretraining, Prioritized Replay, and Curriculum Learning

Keke Abai
Boston University
kekeabai@bu.edu

***Abstract*—Deep Reinforcement Learning (DRL) methods such as the Deep Q-Network (DQN) have achieved human-level performance on Atari environments, yet remain difficult to optimize in sparse reward, high-dimensional tasks like Pac-Man. In this work, we explore a combination of architectural improvements (ResNet DQN), supervised pre-train via tabular Q-learning targets, multi-layout generalization, cosine-annealing learning rate scheduling, DDQN, prioritized replay, distributional RL, and noisy networks. Our staged curriculum—classic-only pretraining followed by brief multi-layout finetuning—achieves a win rate of 58% on the (`spiral_harder`), outperforming all single-stage baselines. We compare CNN vs. ResNet architectures, quantify the effects of tabular TD-target initialization, evaluate multi-layout vs. curriculum training, and outline how Rainbow DQN (Hessel et al., 2018) can further improve performance.**

## I. Introduction

Pac-Man presents a challenging reinforcement learning setting with:

- high-dimensional visual observations,
- sparse and delayed rewards (pellets, win, loss),
- complex multi-agent dynamics (ghost pursuit behavior),
- multiple environment layouts of varying difficulty.

Standard DQN struggles under these conditions. This motivates the development of a more robust pipeline combining:

1) a ResNet-based Deep Q-Network architecture,
2) supervised pretraining using tabular Q-learning targets,
3) a staged curriculum moving from simple to complex layouts,
4) cosine annealing for stable optimization.
5) advanced DQN components including prioritized replay, Double DQN, and NoisyNet-based exploration.

This paper records the process and presents experimental results showing strong gains from curriculum learning drop offs from others.

## II. Background

### A. Deep Q-Learning

DQN learns a parameterized Q-value function $Q_\theta(s, a)$ using the Bellman target:

$$y = r + \gamma \max_{a'} Q_{\theta^-}(s', a'),$$

and optimizes:

$$L(\theta) = \left(Q_\theta(s, a) - y\right)^2.$$

Experience replay and a target network reduce correlation and stabilize updates [1].

### B. ResNets in RL

Residual networks enable deeper architectures without vanishing gradients by using skip connections:

$$\text{ResBlock}(x) = F(x) + x,$$

allowing more complex feature extraction and maintaining a manageable amount of parameters.

### C. Cosine Annealing

Learning rate is updated via:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min})\left(1 + \cos\left(\frac{\pi t}{T}\right)\right),$$

providing smooth decay and stable learning dynamics.

### D. Prioritized Experience Replay (PER)

Standard experience replay in DQN samples past transitions uniformly, which can result in the agent wasting time on transitions that are not as helpful. Prioritized Experience Replay (PER) [3] samples transitions with probability proportional to their temporal-difference (TD) error:

$$P(i) = \frac{p_i^\alpha}{\sum_j p_j^\alpha},$$

where $p_i = |\delta_i| + \varepsilon$ represents the priority of transition $i$ and $\alpha$ controls prioritization. PER increases sample efficiency by focusing updates on transitions where the agent's predictions are most incorrect. To correct sampling bias, PER applies weights:

$$w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta,$$

ensuring unbiased convergence as $\beta \to 1$. In Pac-Man, PER accelerates learning by emphasizing rare events such as near-death states or beneficial moves that result in better chance of survival.

### E. Double DQN (DDQN)

The original DQN algorithm suffers from overestimation of action Q values due to the use of a maximization operator inside the TD target. Double DQN [4] helps to neutralize this by decoupling action selection from action evaluation:

$$y = r + \gamma\, Q_{\theta^-}\left(s', \arg\max_{a'} Q_\theta(s', a')\right).$$

The online network $Q_\theta$ selects the maximizing action, while the target network $Q_{\theta^-}$ evaluates it. This modification reduces overestimation and yields more stable training. DDQN improves how trustworthy the decisions are and prevents the policy from becoming overly optimistic.

### F. Noisy Networks for Exploration

Traditional DQN relies on an externally defined exploration schedule (e.g., $\epsilon$-greedy). Noisy Networks (NoisyNet) [5] incorporate trainable parameter noise directly into the linear layers of the network:

$$W = W_\mu + W_\sigma \odot \epsilon, \qquad b = b_\mu + b_\sigma \odot \epsilon,$$

where $\epsilon$ is sampled noise and $(W_\mu, W_\sigma, b_\mu, b_\sigma)$ are learnable parameters. The agent automatically adjusts how much noise to inject into Q-value estimates based on training progress. Unlike $\epsilon$-greedy, NoisyNet exploration is differentiable, learned, and does not require hand-tuned decay schedules. In Pac-Man, this results in more directed better exploration, improving performance in mazes where eating all the pellets results in a win, and exploration needs to adapt to ghosts and pellet in the surrounding area.

## III. Method

### A. Environment

We evaluate on four layouts:

- `spiral`
- `spiral_harder`
- `empty`
- `classic`

These increase in difficulty and lead to the choice of creating four separately deployable agents for each layout, or one generalized agent. I chose the path of one generalized architecture with custom layout pre-training.

### B. Architecture Overview

I started by evaluating three architectures:

*1) Baseline CNN DQN:* A simple convolutional stack with a fully connected head. Fast to train but underfits complex layouts.

*2) Custom ResNet DQN:* Includes:

- Conv stem (64 filters, kernel 8x8),
- Two residual blocks (Res1, Res2),
- FC head for Q-values.

Skip connections improve gradient flow and feature reuse.

*3) Pretrained ResNet via Tabular Q Targets:* Tabular Q-learning provides the optimal $Q^*(s, a)$ for early states. These serve as supervised regression targets before RL training.

### C. Tabular Q-Target Pretraining

Given tabular targets $Q_{\text{tab}}(s)$, we minimize:

$$L_{\text{pre}}(\theta) = \|Q_\theta(s) - Q_{\text{tab}}(s)\|^2.$$

This initializes the ResNet with meaningful value structure, improving convergence stability.

### D. Multi-Layout Pretraining vs. Curriculum

We test two strategies:

*a) Direct Multi-Layout Pretraining:* Tabular data collected from random layouts. This leads to over-generalization: the model learns survival but not winning (0% win rate).

*b) Staged Curriculum (Proposed):*

1) Pretrain & finetune on `classic`,
2) Brief finetuning on {`spiral`, `spiral_harder`, `empty`}.

This preserves strong foundational policies while adding robustness. Using this architecture is where the model seemed to have its best performance. But I believe a few changes, although damaging to results currently, with fine tuning can produce much higher accuracy going forward. To create a more "Rainbow like" model I then implemented the following changes into the architecture of the agent:

### E. Additional Adaptations

1) Prioritzed Replay (PER)
2) Double DQN
3) Noisy Net with and without small $\varepsilon$
4) Simplifying DQN and increasing memory

## IV. Experimental Results

### A. Performance Summary

Table I summarizes the main variants evaluated on the most efficient layout for testing, `spiral_harder`, using the win rate as the primary metric. Note that some ablations where tested using classic but not enough to form a good comparison. It also needs to be considered that in classic pellets collected would have been used as a metric rather than win rate due to constrains of the environment. Going forward more training and testing in classic layout would be helpful as well as in empty and spiral.

### B. Key Observations

- Tabular pretraining significantly stabilizes early learning for the ResNet DQN and improves convergence compared to randomly initialized ResNets.
- Direct multi-layout pretraining from scratch leads to over-generalization: the agent survives longer but rarely wins, resulting in a 0% win rate on `spiral_harder`.
- A staged curriculum—first pretraining and finetuning on `classic`, then briefly finetuning on multiple layouts—produces the most robust agent, achieving 58–66% win rates depending on the configuration. The inflated win rate here comes from testing on less episodes due to computing constraints. Once I gained access to more

| Variant | Win Rate | Notes |
|---|---|---|
| CNN DQN | 20% | simple single-CNN baseline |
| ResNet DQN | 10% | deeper conv, no pretraining |
| Pretrained ResNet (classic) | 30% | tabular TD warm-start |
| Multi-layout Pretrain | 0% | over-generalized, no wins |
| Staged classic-only pretrain | 66% | best single-stage ResNet+tabular model |
| **Staged multi-layout curriculum** | **58%** | best robust curriculum model |
| Prioritized Replay (PER) | 40% | faster early learning on `spiral_harder` |
| Double DQN | 54% | reduced Q overestimation, higher stability |
| Noisy Net + small $\varepsilon$ | 59% | strongest exploration but less stable |
| Noisy Net only | 9% | over-exploration, poor control |
| Noisy Net (patched) | 32% | partial recovery, suggests overfitting/memory issues |
| Increased memory, 1 ResNet block | 17% | more stable but underperforms staged setup |

TABLE I
WIN RATES ON `SPIRAL_HARDER` FOR DIFFERENT ARCHITECTURAL AND ALGORITHMIC VARIANTS.

compute again win rate seems to drop when in reality they are more accurate.

- Prioritized replay and Double DQN further improve performance (up to 54% wins), confirming that overestimation bias is non-trivial in this domain.
- Noisy Networks combined with a small $\varepsilon$-greedy component can reach the highest raw win rate (59%), but are more sensitive to replay buffer size and can overfit or destabilize when used without additional regularization.
- I then went to increase data but realized a sharp decrease in win rate as training data increased. I decided the model may be to complex and prone to overfitting. Maybe simplifying the model would increase performance?
- Increasing replay capacity and simplifying the ResNet (dropping one block) improved stability, but by itself does not match the staged curriculum performance, resulting in a sharp decline in performance.

## V. TOWARD RAINBOW DQN

Following Hessel et al. (2018), a natural next step is to integrate the full Rainbow DQN stack. In this project, several Rainbow components have already been explored empirically, including Double DQN, prioritized replay, Noisy Networks, and an initial (unsuccessful) attempt at distributional Q-learning. Future extensions include:

1) Incorporating a dueling network architecture to disentangle state-value and advantage.
2) Adding multi-step returns to improve long-horizon credit assignment on complex layouts.
3) Properly tuning and stabilizing distributional RL (C51) for Pac-Man, rather than the configuration that collapsed to 0% wins.
4) Combining all tested components—ResNet backbone, tabular pretraining, staged curriculum, Double DQN, PER, NoisyNet, and distributional value estimates—into a unified Rainbow-style agent.

Each of these can be integrated on top of the current ResNet+tabular+curriculum baseline which still needs ample hyperparameter tuning to reach its best performance.

## VI. CONCLUSION

I present a systematic study of architectural and training improvements, and deprovements, for DQN in Pac-Man. A simple CNN DQN achieves only around 20% wins on `spiral_harder`, and some ResNet variants or multi-layout pretraining can perform even worse. By contrast, tabular Q-target initialization, a ResNet backbone, and a staged curriculum from `classic` to harder layouts dramatically improve sample efficiency and final performance, reaching 58–66% win rates on `spiral_harder`. Additional components such as Double DQN, prioritized replay, and NoisyNet-based exploration further validate the importance of stable value estimation and directed exploration as small fluctuations can either improve or derail performance. With a full Rainbow-style integration and more careful distributional RL tuning, this line of work could hopefully approach state-of-the-art Atari-level performance on challenging Pac-Man variants.

## REFERENCES


[1] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.

[2] M. Hessel *et al.*, "Rainbow: Combining improvements in deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[4] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2016.

[5] M. Fortunato *et al.*, "Noisy networks for exploration," *arXiv preprint arXiv:1706.10295*, 2017.


*Use of ChatGPT for formatting and typesetting equations*