

Comparison of Two-Layer DQN Architectures: Linear, Linear with Clipping, ReLU, and LeakyReLU

Technion NeuroAI Lab

February 23, 2026

1 Introduction

This document provides a comprehensive comparison of four Deep Q-Network (DQN) architectures trained on the CartPole-v1 environment. All networks share a two-layer architecture but differ in their activation functions and training stability mechanisms. The four variants are:

1. **Linear**: Fully linear network with no activation functions
2. **Linear with Clipping**: Linear network with gradient clipping and target Q-value clipping
3. **ReLU**: Network with Rectified Linear Unit activation
4. **LeakyReLU**: Network with Leaky Rectified Linear Unit activation

2 Network Architecture

All four networks share the same basic two-layer architecture:

$$Q(s, a; \theta) = W_2 \cdot h(s; W_1) + b_2 \quad (1)$$

where $s \in \mathbb{R}^4$ is the state vector (CartPole observations), $a \in \{0, 1\}$ is the action, and $\theta = \{W_1, b_1, W_2, b_2\}$ are the network parameters.

The hidden layer computation differs based on the activation function:

$$h(s; W_1) = \sigma(W_1 \cdot s + b_1) \quad (2)$$

where $\sigma(\cdot)$ is the activation function, which varies across architectures.

2.1 Linear Network

For the linear network, σ is the identity function:

$$\sigma_{\text{linear}}(z) = z \quad (3)$$

Therefore, the forward pass simplifies to:

$$Q(s, a; \theta) = W_2 \cdot (W_1 \cdot s + b_1) + b_2 = W_2 W_1 \cdot s + W_2 b_1 + b_2 \quad (4)$$

This can be rewritten as a single linear transformation:

$$Q(s, a; \theta) = W_{\text{eff}} \cdot s + b_{\text{eff}} \quad (5)$$

where $W_{\text{eff}} = W_2 W_1$ and $b_{\text{eff}} = W_2 b_1 + b_2$.

2.2 ReLU Network

For the ReLU network, the activation function is:

$$\sigma_{\text{ReLU}}(z) = \max(0, z) = \begin{cases} z & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \quad (6)$$

The forward pass becomes:

$$Q(s, a; \theta) = W_2 \cdot \text{ReLU}(W_1 \cdot s + b_1) + b_2 \quad (7)$$

2.3 LeakyReLU Network

For the LeakyReLU network, the activation function is:

$$\sigma_{\text{LeakyReLU}}(z) = \max(\alpha z, z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases} \quad (8)$$

where $\alpha = 0.01$ is the negative slope coefficient. The forward pass is:

$$Q(s, a; \theta) = W_2 \cdot \text{LeakyReLU}(W_1 \cdot s + b_1) + b_2 \quad (9)$$

3 Feature Contribution Analysis

The feature contribution analysis reveals how each input feature contributes to the Q-value for each action. This is computed differently for each architecture.

3.1 Linear Network Feature Contributions

For the linear network, the effective weight matrix is:

$$W_{\text{eff}} = W_2 W_1 \in \mathbb{R}^{2 \times 4} \quad (10)$$

The contribution of feature i to action j is:

$$\text{contrib}_{j,i} = (W_{\text{eff}})_{j,i} \cdot s_i \quad (11)$$

3.2 ReLU Network Feature Contributions

For the ReLU network, the feature contributions account for the activation mask:

$$m = \mathbb{I}[W_1 \cdot s + b_1 > 0] \in \{0, 1\}^{64} \quad (12)$$

where $\mathbb{I}[\cdot]$ is the indicator function. The effective weight matrix becomes:

$$W_{\text{eff}} = (W_2 \odot m) W_1 \quad (13)$$

where \odot denotes element-wise multiplication (broadcasted appropriately). The contributions are:

$$\text{contrib}_{j,i} = (W_{\text{eff}})_{j,i} \cdot s_i \quad (14)$$

3.3 LeakyReLU Network Feature Contributions

For the LeakyReLU network, the hidden layer activation is:

$$h = \text{LeakyReLU}(W_1 \cdot s + b_1) \quad (15)$$

The contributions are computed directly from the activated hidden layer:

$$\text{contrib}_{j,i} = (W_2)_{j,:} \odot h \cdot (W_1)_{:,i} \cdot s_i \quad (16)$$

4 Deep Q-Learning Algorithm

All networks use the standard DQN algorithm with the following components.

4.1 Bellman Equation

The Q-function is updated using the Bellman equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot|s, a)} \left[r + \gamma \max_{a'} Q^*(s', a') \right] \quad (17)$$

where r is the immediate reward, $\gamma = 0.99$ is the discount factor, and $p(\cdot|s, a)$ is the transition probability.

4.2 Temporal Difference (TD) Target

The TD target for training is:

$$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-) \quad (18)$$

where θ^- are the parameters of the target network, which is updated periodically.

4.3 Loss Function

The loss function is the mean squared error between the current Q-values and the TD targets:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s, a, r, s') \sim \mathcal{D}} \left[(y_t - Q(s, a; \theta))^2 \right] \quad (19)$$

where \mathcal{D} is the replay buffer.

4.4 Gradient Descent Update

The parameters are updated using gradient descent:

$$\theta \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta) \quad (20)$$

where α is the learning rate.

5 Training Stability Mechanisms

5.1 Linear with Clipping Variant

The linear network with clipping includes several stability mechanisms:

5.1.1 Gradient Clipping

Gradients are clipped to prevent explosion:

$$\nabla_{\theta}\mathcal{L} \leftarrow \text{clip}(\nabla_{\theta}\mathcal{L}, -\text{max_norm}, \text{max_norm}) \quad (21)$$

where $\text{max_norm} = 1.0$ and the clipping is applied using the gradient norm:

$$\nabla_{\theta}\mathcal{L} \leftarrow \frac{\nabla_{\theta}\mathcal{L}}{\max(1, \|\nabla_{\theta}\mathcal{L}\|/\text{max_norm})} \quad (22)$$

5.1.2 Target Q-Value Clipping

Target Q-values are clipped to prevent unbounded growth:

$$y_t \leftarrow \text{clip}(y_t, -Q_{\max}, Q_{\max}) \quad (23)$$

where $Q_{\max} = 100.0$.

5.1.3 Weight Initialization

Weights are initialized using Xavier uniform initialization:

$$W \sim \mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}, \frac{\sqrt{6}}{\sqrt{n_{\text{in}} + n_{\text{out}}}}\right) \quad (24)$$

where n_{in} and n_{out} are the input and output dimensions of the layer.

5.1.4 Target Network Update Frequency

The target network is updated every $N_{\text{update}} = 100$ steps (instead of every 500 episodes) for more frequent updates:

$$\theta^- \leftarrow \theta \quad \text{if } t \bmod N_{\text{update}} = 0 \quad (25)$$

6 Hyperparameters Comparison

Table 1 summarizes the hyperparameters for each variant.

7 Key Differences Summary

7.1 Expressiveness

- **Linear**: Limited to linear decision boundaries. Cannot represent non-linear Q-functions.
- **Linear+Clip**: Same expressiveness as linear, but with improved training stability.

Table 1: Hyperparameter Comparison

Hyperparameter	Linear	Linear+Clip	ReLU	LeakyReLU
Learning Rate	10^{-3}	5×10^{-4}	10^{-3}	10^{-3}
Discount Factor (γ)	0.99	0.99	0.99	0.99
Batch Size	64	64	64	64
Replay Buffer Size	10,000	10,000	10,000	10,000
ϵ Start	1.0	1.0	1.0	1.0
ϵ End	0.05	0.05	0.05	0.05
ϵ Decay	0.999	0.999	0.999	0.999
Target Update (steps)	500 episodes	100 steps	500 episodes	500 episodes
Gradient Clipping	No	Yes (1.0)	No	No
Target Q Clipping	No	Yes (± 100)	No	No
Weight Init	Default	Xavier	Default	Default

- **ReLU**: Can represent piecewise linear functions. Enables non-linear decision boundaries.
- **LeakyReLU**: Similar to ReLU but allows small negative gradients, preventing "dying ReLU" problem.

7.2 Training Dynamics

- **Linear**: Prone to gradient explosion and weight growth without bounds.
- **Linear+Clip**: Stabilized training through gradient and target clipping, but still limited expressiveness.
- **ReLU**: Stable training, but can suffer from "dying ReLU" where neurons become inactive.
- **LeakyReLU**: More stable than ReLU, maintains gradient flow for negative inputs.

7.3 Feature Contribution Computation

- **Linear**: Direct matrix multiplication $W_2 W_1$.
- **Linear+Clip**: Same as linear.
- **ReLU**: Requires masking based on activation: $(W_2 \odot m) W_1$.
- **LeakyReLU**: Computed from activated hidden layer: $W_2 \odot h \cdot W_1$.

8 Expected Performance

Based on the architecture differences:

- **Linear**: Poor performance (average reward $\approx 10 - 11$ steps) due to limited expressiveness.
- **Linear+Clip**: Similar poor performance but with more stable training (weight norms controlled).
- **ReLU**: Good performance (can reach 200+ steps) due to non-linear expressiveness.

- **LeakyReLU**: Good to excellent performance, potentially better than ReLU due to improved gradient flow.

9 Mathematical Properties

9.1 Composition of Linear Layers

For a linear network with two layers, the composition is equivalent to a single linear layer:

$$f_2 \circ f_1(x) = W_2(W_1x + b_1) + b_2 = (W_2W_1)x + (W_2b_1 + b_2) \quad (26)$$

This means a two-layer linear network has the same representational capacity as a single-layer network.

9.2 ReLU Non-Linearity

ReLU introduces non-linearity through the piecewise linear function:

$$\text{ReLU}(z) = \max(0, z) = \frac{z + |z|}{2} \quad (27)$$

The derivative is:

$$\frac{d}{dz}\text{ReLU}(z) = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases} \quad (28)$$

9.3 LeakyReLU Non-Linearity

LeakyReLU maintains a small gradient for negative inputs:

$$\text{LeakyReLU}(z) = \max(\alpha z, z) = \begin{cases} z & \text{if } z > 0 \\ \alpha z & \text{if } z \leq 0 \end{cases} \quad (29)$$

The derivative is:

$$\frac{d}{dz}\text{LeakyReLU}(z) = \begin{cases} 1 & \text{if } z > 0 \\ \alpha & \text{if } z \leq 0 \end{cases} \quad (30)$$

where typically $\alpha = 0.01$.

10 Conclusion

The comparison reveals fundamental trade-offs between expressiveness and training stability:

1. **Linear networks** are limited in expressiveness but can be stabilized with clipping mechanisms.
2. **ReLU networks** provide non-linear expressiveness essential for complex Q-functions.
3. **LeakyReLU networks** offer similar expressiveness to ReLU with improved gradient flow.

4. **Stability mechanisms** (gradient clipping, target clipping) are crucial for linear networks but less critical for ReLU-based networks.

The choice of activation function fundamentally determines the network's ability to learn effective policies in reinforcement learning tasks.