Improving Neural Network Plasticity in Reinforcement Learning with Plasticity Injection and Plasticity Metric Regularization

Ryan Cheng Ethan Chen Leon Ma

1. Extended Abstract

Neural network plasticity describes the ability of a trained neural network to learn new tasks. Past experimental and theoretical studies have shown that plasticity tends to decrease as training continues (Lyle et al., 2023) (Dohare et al., 2023) (Berariu et al., 2021). This phenomenon is especially problematic in cases where the target of the model changes during the training epoch such as in the fields of continual learning and reinforcement learning. Neural network plasticity loss has been previously studied for artificially structured supervised reinforcement learning tasks such as continual classification for MNIST and ImageNet (Dohare et al., 2023). These supervised tasks are generally based on problems of moving regression. Such problems have changing target variables that test the ability of the neural network to adapt to new information. This concept of a changing environment is also important in reinforcement learning tasks where it is not the regression variable that is changing but the environment of the agent itself. Thus, high plasticity is essential to reinforcement learning algorithms as the environment can change quickly and unexpectedly.

In this paper, we apply various strategies to maintain network plasticity on a standard online implementation of DQN on the Atari MsPacman environment, as well as on the implementation of offline-to-online fine-tuning for IQL learning on the Pointmass Hard environment. We evaluate two metrics that have been proposed to correlate with plasticity, (Dohare et al., 2023) the average effective rank (Roy & Vetterli, 2007) and the weight magnitude of the layers of the network, as well as a metric that we posit would also correlate with plasticity: the percentage of units that are updated for any given batch (Lyle et al., 2023). By studying these metrics we hoped to better understand plasticity from a statistical point of view.

Motivated by our quantitative metrics of plasticity, we applied a method of plasticity injection (Nikishin et al., 2023) to retain plasticity during the training process. This method works by freezing the original critic network and training a new network during the learning process. This in theory allows the neural network to remain "fresh" and retain its qualities that allow it to learn. We then tracked our metrics and performance before and after the plasticity injection on the DQN agent in LunarLander and MsPacMan and the IQL agent in PointMassHard.

We also attempted loss function regularization. Motivated by classical methods such as LASSO and RIDGE. By adding regularization terms in our loss function and adjusting them to better fit the challenge of combating plasticity loss, we hope to encourage metrics of plasticity such as weight magnitude to remain high during the training process. We also examine the technique of layer normalization with these metrics, which has been proposed to limit the magnitude of the Hessian of the loss function (Ghorbani et al., 2019). This hypothetically smooths our loss landscape and reduces internal covariate shift allowing our neural network to learn new tasks efficiently.

2. Metrics Tracked

We studied how three primary metrics relate to the evaluation return and the plasticity of the critic network. These metrics are the average effective rank of the layers of the network, the average weight magnitude of the network, as well as the percentage of units that are updated in a given training step. Two of these metrics have been previously studied in (Dohare et al., 2023) and (Lyle et al., 2023) (effective rank and weight magnitude), and the third metric (percentage of updated units) is an original metric. Each of these metrics is logged every 1000 environment steps to save computational resources.

Our study slightly differs from previous studies on plasticity loss in that other studies demonstrate plasticity loss directly by training the same network on sets of distributions deliberately shifted from the original, usually in sets of "epochs". These shifted distributions can be completely different environments than the original environment the network is trained on. However, in our study, we study the plasticity and performance of a network over a large number of environment steps, for which the distribution shift is within the same environment and is steady over time. This is closer to the settings for which reinforcement learning agents are applied. To determine whether plasticity loss occurs and quantify the relative plasticity of a network, we characterize empirically the point at which the evaluation returns of the

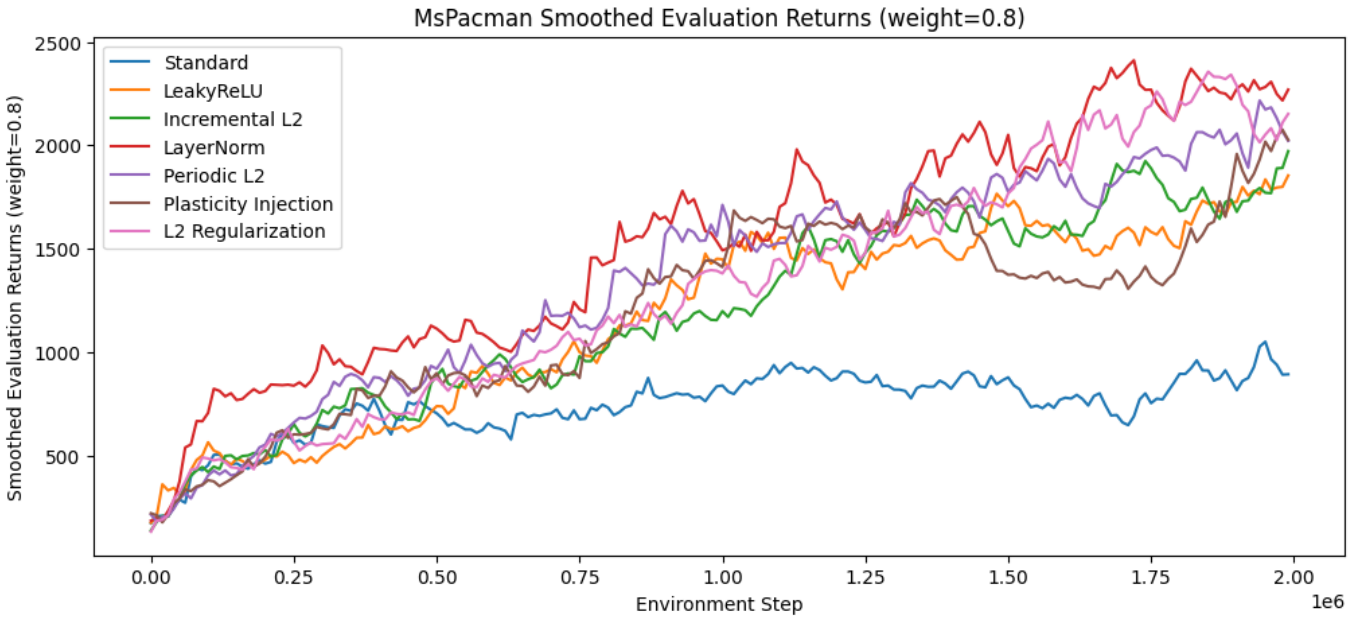

Figure 1. Performance results for each algorithm implemented on the Atari MsPacman environment. These are single runs with values that have been smoothed using an exponential moving average with a weight value of 0.8 for ease of visualization. All have similar performance except for the standard implementation, with LayerNorm performing slightly better than the rest and Plasticity Injection performing slightly worse.

network start to stabilize, and analyze how the metrics relate to the returns up to and after this point.

The effective rank of a matrix is similar to the rank of a matrix in that it quantifies how strongly each dimension influences the transformation induced by a matrix, and has been shown to decrease over time with plasticity loss. It is a continuous measure that ranges from 1 to the rank of the matrix. A low effective rank shows that a small number of dimensions have a significant effect on a transformation, while a high effective rank shows that most of the dimensions contribute equally. For a matrix $A \in \mathbb{R}^{n \times m}$ with singular values σ_k for $k = 1, 2, ..., \max(n, m)$, we define "probabilities" $p_k = \frac{\sigma_k}{||\boldsymbol{\sigma}||_1}$ where $||\boldsymbol{\sigma}||_1$ is the L1 norm of the vector $\boldsymbol{\sigma}$ containing all the singular values of A. The effective rank of A is defined as

$$\text{erank}(A) = \exp\left(-\sum_{k=1}^{\max(n,m)} p_k \log p_k\right).$$

For neural networks, effective rank represents the number of units of a hidden layer that can affect the output of the layer. Low effective rank means that most units do not provide useful information. This can be a good or a bad thing when it comes to plasticity, as low effective rank means that some of the units could be repurposed to learn new information without sacrificing the current complexity of the model. It might also be an indicator that most of the units could be dead and have low to zero gradients when trained.

The average weight magnitude of a network has been shown to increase over time with plasticity loss. It is measured by adding up the absolute value of the weights of a network and dividing it by the total number of weights. High-weight magnitudes have been associated with learning instability, such as the exploding gradient problem in recurrent neural networks, and can lead stochastic approximation algorithms used to train networks to diverge as they rely on the gradients remaining bounded.

Finally, we chose to track the percentage of updated units in a network by counting the number of units in each layer with a nonzero gradient after each training batch. This is similar to the idea of tracking dead units - units for which there is zero gradient for all training points, rendering them immutable. The metric we propose is better suited for online reinforcement learning as in online reinforcement learning there does not exist a fixed training set. We hope to see a consistently high percentage of updated units throughout training, as this is an indicator that the algorithm is continuing to update itself and maintain plasticity. However, we would not want the percentage of updated units to be too high too often throughout training, as this could be an indicator that the model is forgetting information that it learned

previously.

3. Algorithms and Techniques

3.1. Plasticity Injection

To propel agents to keep learning and exploring their environments, we implemented plasticity injection on two pairs of agents with their respective tasks and environments: first, a DQN agent in LunarLander and MsPacman, and second, an IQL agent with online fine-tuning on the PointMass Hard environment.

We denote $h_\theta(x)$ as the original critic network; upon injecting plasticity, we will first freeze $h_\theta(x)$, then initialize two new networks with the same layers and dimensions as $h_\theta(x)$: $h_{\theta'_1}(x)$ is now our critic that will start to train from that step onward and $h_{\theta'_2}(x)$ contains the same weights as $h_{\theta'_1}(x)$, but stays frozen so its gradients are not updated (Nikishin et al., 2023).

To implement the code of plasticity injection, we specified a step in which we inject plasticity. To update the critic, we will compute all possible Q-values from the current state (or observation) with the formula $h_\theta(x) + \left(h_{\theta'_1}(x) - h_{\theta'_2}(x)\right)$. The difference of the latter two terms serve as the residual so we maintain a baseline for future action evaluations.

In our experiments, we manually choose the steps for plasticity injection by first running the baseline algorithm to observe the performance of evaluation return. From this initial graph, we can determine where the return starts to stabilize, which we hypothesize is due to plasticity loss - the agent is not incentivized to explore new actions.

We also propose a new algorithm of injecting plasticity twice to attempt to further boost exploration.

$$\left(h_\theta(x) + \left(h_{\theta'_1}(x) - h_{\theta'_2}(x)\right)\right) + \left(h_{\theta''_1}(x) - h_{\theta''_2}(x)\right)$$

$h_{\theta''_1}(x)$ is our new critic in training and $h_{\theta''_2}(x)$ serves as the critic with a frozen identical copy of the parameters randomly initialized in the critic being trained. We treat the first term as our previous critic that is now frozen. This algorithm operates in the same way as single plasticity injection up until the step at which plasticity is injected a second time. This allows us to encourage the agent to more dynamically adapt to the new environment transitions. From the second injection point onward, we calculate all possible Q values of the current observation with the above formula.

3.2. Regularization

We implemented several variations of the standard L^2 Regularization to the DQN critic loss, replacing the critic update with:

$$\phi_{k+1} \leftarrow \arg\min_{\phi \in \Phi} \left[\sum_j \left(y_j - Q_\phi(\mathbf{s}_j, \mathbf{a}_j)\right)^2 + \lambda(k)||\phi||^2 \right]$$

where y_j is the target value and λ, what is traditionally a hyperparameter used in regularization, is modified to vary according to a pre-specified schedule based on the environment step. Our goal with using regularization is to ensure that the weight magnitude of the critic network remains low, which is desired since weight magnitude is associated with plasticity loss. We include two original variants (as far as we are aware) of L^2 Regularization which we dub Incremental L^2 Regularization and Periodic L^2 Regularization.

For Incremental L^2 Regularization, we steadily increase the value of λ linearly as training continues, from a predefined start value to a predefined end value. The idea behind this is that plasticity loss is more likely to occur late into training, and so we only want to start regularizing the weights of the network late into training. Otherwise, normal L^2 Regularization could interfere with training early on and in the worst case result in loss of information that the network learned earlier.

For Periodic L^2 Regularization, we train the network with L^2 regularization for $N = 500$k steps, then proceed to train the network normally for $M = 100$k steps (with fixed λ) and repeat for as long as training occurs. This is similar in idea to the shrink-and-perturb trick (Dohare et al., 2023) which shrinks the weights of the network directly and adds random noise. However, our method is implicitly applied through the loss. Like in Incremental L^2 Regularization, we hope to prevent early interference with the network's training but ideally allow the network to grow periodically before trimming the weights slightly.

3.3. Layer Normalization and LeakyReLU

Another technique we investigated was applying layer normalization after every convectional and fully-connected layer in the model. This has been proposed to improve network performance by regularizing the loss landscape, providing more benefit to environments that tend to have ill-conditioned Hessians of the loss function or the gradient covariance of the standard model is degenerate (Lyle et al., 2023). We seek to understand how this technique affects our proposed metrics and whether performance gains can be attributed to changes in these metrics.

We also experimented with using LeakyReLU activations in place of standard ReLU activations. This is to prevent the formation of dead units that might occur if standard ReLU was used, as standard ReLU sets the output to zero of any unit that has a value of less than one and provides a large flat region where the gradient is zero.

4. MsPacman Results and Analysis

We first ran the standard implementation of DQN on MsPacman for 4.5 million environment steps (returns and metrics can be provided upon request). Since the returns seemed to stabilize starting around 2 million steps, we chose to perform all our runs up to that point due to computation and memory constraints. To determine the scale at which plasticity loss occurs and measure the network's relative plasticity, we empirically assess the point at which the network's evaluation returns stabilize. Subsequently, we analyze the relationship between metrics and returns both leading up to and following this stabilization point.

Overall, we find that all the techniques generally outperform the standard DQN implementation to around the same degree (see Figure 1). Layer normalization seems to generally perform the best, while plasticity injection generally does worse up to near the end, where LeakyReLU starts doing worse. None of the techniques' returns seem to start converging in this time frame except for Leaky ReLU, which starts seeing diminishing returns after around 1.5 million environment steps. Of note is that plasticity injection after injection at 1 million steps initially starts performing better, then dips around 1.5 million steps and proceeds to rise again at 1.75 million steps. This is likely due to the relative instability of training the new injected network.

To be certain of the veracity of our results, we would ideally perform multiple runs for a longer set of environment steps to see where each technique's returns start to stabilize and/or decrease, with multiple seeds after performing hyperparameter searches for each hyperparameter. However, we only had access to a single laptop with a GPU to perform these runs and did not have the computational power nor computer memory to be able to perform longer runs.

4.1. Effective Rank

For the standard implementation, it appears that the effective rank (Figure 2) initially starts high, decreases for the first 300k environment steps, then starts increasing again. This is likely because all of the networks start with randomly initialized weights within the same bounds and the explanatory power of each unit is around the same, and when we first start training the algorithm we only need a small amount of units to represent the complexity of the data. This causes the initial decrease from the maximum value. Afterwards, as data is introduced to the network, more units are repurposed to represent the data, increasing the overall effective rank of the network. We expect that over a larger number of environment steps, the average effective rank would start to stabilize or decrease once again due to the network being over-saturated with data, as is seen in (Dohare et al., 2023). For these reasons, we believe that effective rank would be a good indicator of whether a network is saturated and plasticity loss starts to occur.

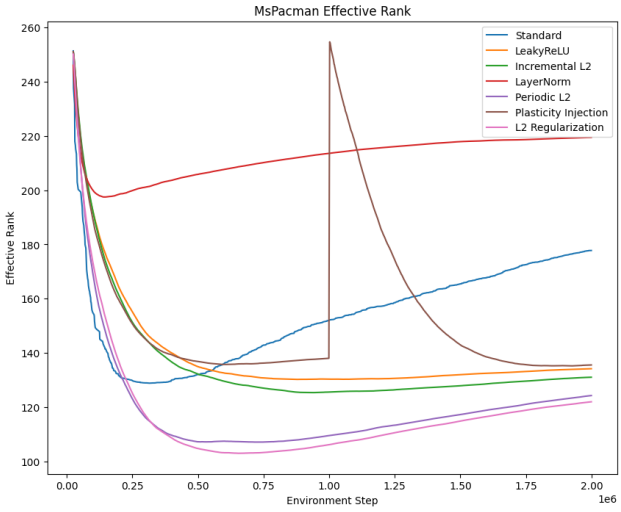

Figure 2. Average effective rank of each algorithm's layers during training on the MsPacman environment. The spike in the middle of the Plasticity Injection run happens because we start to track the new network, which is initialized with random weights. The old network is frozen for all subsequent environment steps after injection, and so the metrics are likewise constant afterwards.

It is interesting to note that regularization tends to result in the lowest effective ranks, and layer normalization results in the highest. The reason regularization results in lower effective ranks is that regularization initially punishes extraneous weights that have little explanatory power for the model. We are not entirely sure why layer normalization results in such high effective rank, we believe this might be because the normalized outputs make each unit have a higher influence on the outputs of the previous layers and thus representational complexity is spread to all of the units rather than to a few. In the long run, this is likely what we want to prevent plasticity loss, as information is likely more efficiently represented since each unit takes the duty of having multiple possible roles in forming the output of a layer. Plasticity injection seems to converge to a relatively low effective rank before and after injection, which is likely an indicator that the network learns a representation of the distribution that is inefficient in storing representational complexity and that the main benefit of injection is providing more memory to represent complexity rather than a more efficient use of memory.

4.2. Weight Magnitude

The weight magnitude of each technique generally increases for all models, although at slower rates for variants of L^2 regularization, as is expected. Weight magnitude does not

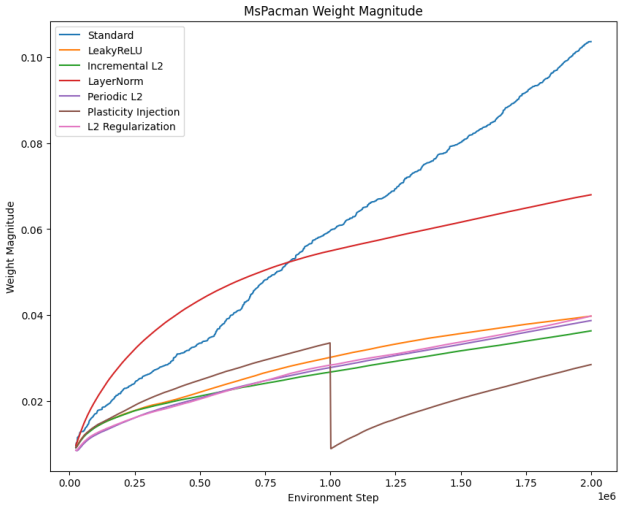

Figure 3. Weight magnitude of each algorithm's layers during training on the MsPacman environment. The drop in the middle of the Plasticity Injection run happens because we start to track the new network, which is initialized with random weights. The old network is frozen for all subsequent environment steps after injection, and so the metrics are likewise constant afterwards.

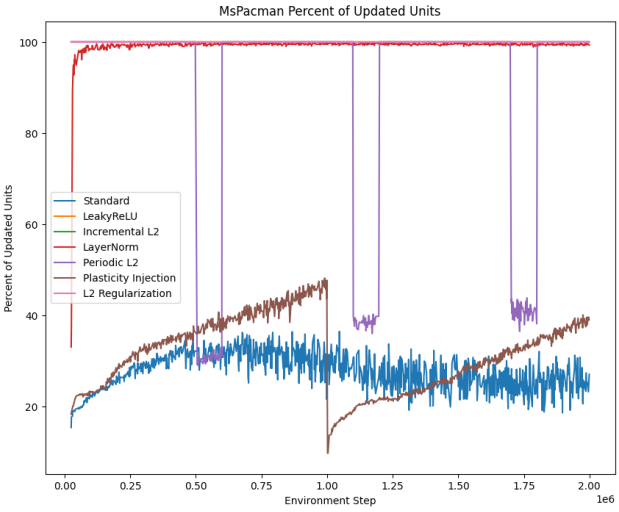

Figure 4. Percentage of updated units of each algorithm's layers during training on the MsPacman environment. For LeakyReLU, Incremental L^2 Regularization, and L^2 Regularization the percentage of updated units is a constant 100%. The drop in the middle of the Plasticity Injection run happens because we start to track the new network, which is initialized with random weights. The old network is frozen for all subsequent environment steps after injection, and so the metrics are likewise constant afterward.

seem to affect the performance of the networks with weight magnitude lower than the standard algorithm much at this stage as layer normalization has a relatively high weight magnitude (due to relatively larger weights being needed for smaller normalized inputs into layers) but has the best performance in the studied time frame. We can, however, observe that the extreme average weight magnitude of the standard DQN algorithm is likely a large contributing factor to the relative instability and underperformance of the standard DQN algorithm compared to the other techniques used on top of it. In this sense the weight magnitude when observed at the extremes is also a good metric for the performance and plasticity of the algorithm.

4.3. Updated Units

Out of the other metrics examined, the percentage of updated units has the most variability based on the algorithm used and thus also provides useful information to consider in the execution of the techniques. The standard algorithm starts off with a low, increases to around 500k environment steps, and then starts decreasing again. The cause of this is likely similar to why the evaluation returns are shaped the way they are - the network first starts updating a few units at a time to learn a less complex representation, and over time includes more units into its representation as new data is fed in. From periodic regularization, we can see that the network tends to include more units in periods of training without regularization (which is desired to spread representational complexity around units).

5. IQL Online Fine-tuning Results and Analysis

5.1. Evaluation Return

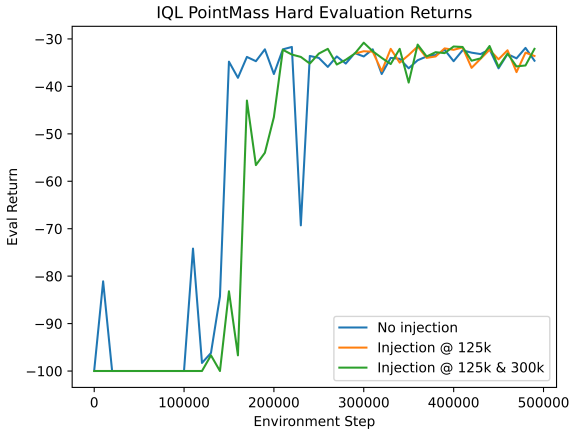

Figure 5. Performance results for the algorithms of base IQL, IQL with plasticity injection once, and IQL with plasticity injection twice implemented on the PointMass Hard environment.

From Figure 5, we can see that both algorithms with plasticity injection once and twice still lead to convergence to the same return. When compared with the base algorithm, after injecting, it took around 50,000 more steps for the agent to spike in reward. Double plasticity injection does not cause much of a difference in reward. Possible explanations include the algorithm taking the same actions given the same states for injections of one time and two times. In other words, $h_{\theta'_1}(x) - h_{\theta'_2}(x)$ and $h_{\theta''_1}(x) - h_{\theta''_2}(x)$ produce similar results, thus not allowing the algorithm to discover even higher-rewarding transitions.

6. Conclusion

6.1. Limitations and Future Directions

One limitation we faced was compute power. Ryan ran all of our MsPacman experiments since he was the only one with a machine powerful enough to handle the computational intensiveness.

Future directions include making modifications to the way the step is chosen for plasticity injection, e.g. tracking the moving average of a specific or a combination of metrics and choosing the step when this value exceeds a threshold. This would require running the baseline algorithm once to determine the optimal bounds on the metrics.

Furthermore, (Nikishin et al., 2023) suggested that injecting plasticity improved computational efficiency. The idea is that the randomized initial weights of a fresh neural network provide a friendlier loss landscape that allows for fast training. However, we did not observe such phenomena in any of our experiments. Trying to replicate this phenomenon would be an interesting topic of further work. Considering our lack of computing resources, a way to both increase plasticity and decrease training time would give us a large improvement in efficiency and performance.

6.2. Each Member's Contributions

Ethan set up codebase, did implementation and writeup of Plasticity Injection for DQN and IQL Online Fine-tuning, implementation of basic and periodic regularization schedules for DQN, some implementation of Metrics Tracked, and ran runs and plotted data for IQL.

Ryan: Implementation, writeup, and analysis of metrics tracked, layer normalization, and LeakyReLU; ideas, analysis, and some implementation for the regularization techniques in the DQN of the MsPacman environment. Also ran MsPacman runs and plotted data.

Leon: MNIST toy MDP and regularization. Write up for the introduction and citations.

6.3. Link to the GitHub Repository

https://anonymous.4open.science/r/cs285_final_proj-B25B/

References

Berariu, T., Czarnecki, W., De, S., Bornschein, J., Smith, S., Pascanu, R., and Clopath, C. A study on the plasticity of neural networks. *arXiv preprint arXiv:2106.00042*, 2021.

Dohare, S., Hernandez-Garcia, J. F., Rahman, P., Sutton, R. S., and Mahmood, A. R. Loss of plasticity in deep continual learning. *arXiv preprint arXiv:2306.13812*, 2023.

Ghorbani, B., Krishnan, S., and Xiao, Y. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pp. 2232–2241. PMLR, 2019.

Lyle, C., Zheng, Z., Nikishin, E., Avila Pires, B., Pascanu, R., and Dabney, W. Understanding plasticity in neural networks. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 23190–23211. PMLR, 2023. URL https://proceedings.mlr.press/v202/lyle23b.html.

Nikishin, E., Oh, J., Ostrovski, G., Lyle, C., Pascanu, R., Dabney, W., and Barreto, A. Deep reinforcement learning with plasticity injection. *NeurIPS 2022*, 2023.

Roy, O. and Vetterli, M. The effective rank: A measure of effective dimensionality. In *2007 15th European signal processing conference*, pp. 606–610. IEEE, 2007.