The Comparison Between Environment Stochastic Independent Representation, Decision Transformer, and Conservative Q-Learning in Arcade Learning Environment

Takuya Yahagi s1290042 Supervised by Prof. Maxim Mozgovoy

Abstract

Offline RL has garnered significant attention for its stability and sampling efficiency in various fields including the gaming industry. While methods such as DT and CQL excel in deterministic environments, their applicability to stochastic settings remains limited. To address this, ESPER method was introduced, demonstrating superior performance in stochastic environments. However, it is not investigated in deterministic settings. This study compares the performance of ESPER, DT and CQL in a deterministic environment using MinAtar's Breakout. Models were trained using datasets generated by a DQN agent and evaluated over one million steps. Results show that ESPER outperforms DT and CQL in terms of average score, demonstrating its adaptability to deterministic environments. However, discrepancies between target returns and actual scores highlight potential limitations in training steps and parameter settings. These findings indicate that ESPER is a versatile approach capable of achieving robust performance in deterministic and stochastic environments. Future work will focus on optimizing training conditions and expanding evaluations to diverse environments to validate its generalizability.

1 Introduction

Recently, offline machine learning methods such as Decision-Transformer (DT) [1], and Conservative Q-Learning (CQL) [2] have received much attention due to their stability of evaluation, and efficiency of sampling in the game industry. However, these methods can be adopted only in the deterministic environment and not in the stochastic environment.

Paster et al. introduced environment-stochasticity-independent representation (ESPER) [3]. This method is intended for use in environments that include stochasticity. ESPER got a higher score than other methods such as DT and CQL in some stochastic environments. However, these methods have not been compared in a non-stochastic environment.

The aim of this study is to compare ESPER, DT, and CQL by training them in a deterministic environment and investigate whether ESPER is capable of using a deterministic environment.

2 Method

2.1 Environments

2.1.1 Deterministic or Stochastic [4]

In reinforcement learning tasks, environments are often classified as either deterministic or stochastic, based on the predictability of their outcomes.

Deterministic environments are those where the outcomes of actions are entirely predictable. If the same state and action are given, the resulting state and reward will always be identical. For example, chess, puzzle solving, and so on.

Stochastic environments introduce uncertainty and randomness in outcomes. In these environments, the same state-action pair can lead to different outcomes depending on factors beyond the agent's control. For example, stock market analysis, autonomous driving, and so on.

2.1.2 Arcade Learning Environments

Arcade Learning Environment (ALE), developed by Bellemare et al [5], is widely used for research in game and reinforcement learning (RL). In this study, we have also attempted to use ALE, however, we noticed that the computation resource for it is considerably high in advancing research. So, we used another environment, MinAtar, as a deterministic environment.

2.1.3 Breakout in MinAtar

Breakout is one of the ALE games where the player controls a paddle at the bottom of the screen to bounce a ball and break rows of bricks positioned at the top of the screen. Scores are added for each brick broken, and the game continues with new rows of bricks added after clearing the existing ones. The game terminates when the ball falls below the paddle and cannot be returned. The deterministic mechanics of Breakout, such as predictable ball reflection angles and consistent reward structures, make it an ideal environment for RL studies focusing on policy optimization in controlled settings.

In this study, we used the light version of Breakout, which is implemented in the MinAtar framework. MinAtar is a simplified testbed for RL research inspired by ALE, developed by Young and Tian [6]. This environment needs lower computational resources than ALE in that its lower spatial dimension (10 × 10 grid compared to 64 × 64 in ALE), its lower action space (6 compared to 18 in ALE), and so on. It also maintains the mechanics of ALE as much as possible. For example, some environments in MinAtar have implemented a mechanism like ALE that dynamically adjusts the difficulty level based on the player's skill level or time. By using MinAtar's version of Breakout, we leverage its efficient resource requirements while preserving the game's core dynamics for our experimental objectives.

In MinAtar's Breakout, the primary optimization goal is to maximize the cumulative reward per episode. Here, an episode refers to a single game session that begins when the game starts and ends when the ball is lost, resulting in the termination of the game. The player controls a paddle at the bottom of the screen and must bounce a ball to break 3 rows × 10 columns of bricks along the top of the screen. A reward of +1 is given for each brick broken by the ball. When all bricks are cleared another 3 rows are added. The ball travels only along diagonals. When the ball hits the paddle, it is bounced either to the left or right depending on the side of the paddle hit. When the ball hits a wall or brick, it is reflected. Termination occurs when the ball hits the bottom of the screen. The visualization is shown in Figure 1.

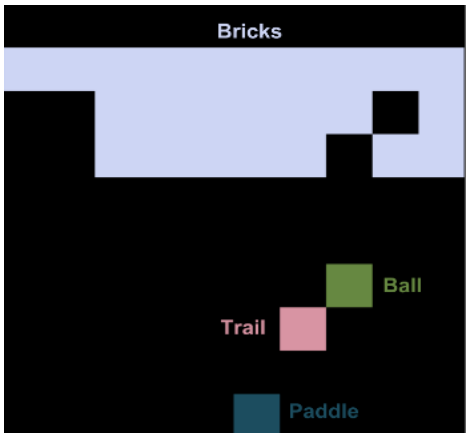

Figure 1 The screenshot of MinAtar's breakout

2.2 Experiments

We trained each model by a million steps.

2.2.1 Dataset

There are no datasets for MinAtar. So, we created a dataset based on DQN Replay Dataset [7]. We adopted Deep Q-Network (DQN) [8] agents in MinAtar's breakout for collecting data. It is trained by about a million steps, and we got about 17500 episodes in this training. Parameters are shown in Table 1. We set no sticky action, the parameter of probability that repeats the agent's previous action. We also did not set frame skipping, which is the parameter to decide the number of steps the same action continues.

Figure 2 shows that DQN average returns per 100 steps in training for collecting datasets.

Hyperparameters	Value
Batch size	32
Learning Rate	2.5e-4
Activation Function	ReLU [9]
Optimizer	RMSProp[10]
Buffer size	100,000
Discount factor	0.99

Table 1 DQN hyperparameters

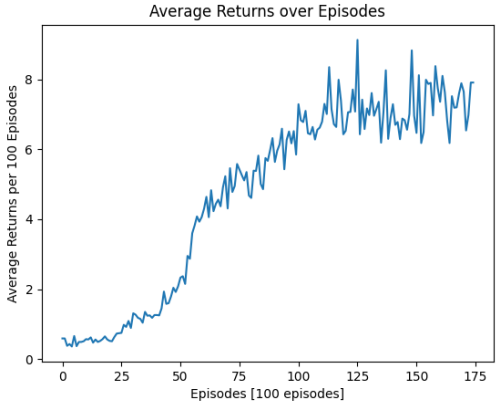

Figure 2 Average returns per 100 episodes. Returns are the cumulative return of each episode. In this graph, they are averaged and plotted every 100 episodes.

2.2.2 Deep Q Network (DQN)

Q-learning is a model-free RL algorithm that aims to learn an optimal policy by iteratively updating a Q-value function. Specifically, the Q-value is updated for each action, and based on its action is selected.

In DQN, the action Q-value is estimated by a neural network. For updating the network, experience replay buffers are used. It is a transition (state, action, reward, and next state). There are two networks, the Q network and the target network. The Q network is updated by batch randomly selected from experience replay buffers. Then the target network is updated by the Q network.

The network parameter θ is updated to minimize the following loss function.

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'|\theta^{-}) - Q(s, a|\theta)\right)^2\right]$$

Here, s, a is the current state and action, respectively, r is a reward obtained after taking action a in state s, s' is next state, γ is a discount factor and θ^{-} is a target network parameter.

2.2.3 Conservative Q-Learning (CQL)

CQL is a value-based offline RL algorithm. Unlike standard Q-learning approaches, CQL incorporates a penalty term into the Q-function objective to enforce conservatism, which ensures that the learned policy avoids overestimating actions that are not well supported by the offline dataset.

In CQL, the following loss function is commonly used.

$$L_{CQL}(Q) = \mathbb{E}_{(s,a)\sim D}[Q(s,a)] - \mathbb{E}_{(s,a)\sim \pi}[Q(s,a)] + \alpha \mathbb{E}_{(s,a)\sim \pi}[log\pi(a|s)]$$

Here, D is the replay buffer of the dataset, and π is the learning policy.

Our CQN architecture consisted of a single convolutional layer, followed by a fully connected hidden layer. Our convolutional layer used 16 3*3 convolutions with stride 1 and the fully connected layer had 128 units. We decided on these conditions based on DQN settings in Kenny's research [6]. Another parameter is shown in Table 2.

Hyperparameters	Value
Batch size	128
Learning Rate	5e-4
Target Update Rate	1e-3
Activation Function	ReLU [9]
Optimizer	Adam [11]
Discount Factor	0.99

Table 2 CQL hyperparameters

2.2.4 Decision-Transformer (DT)

DT is an RL approach that reframes the RL problem as a sequence modeling task, enabling the use of transformers to predict actions based on past trajectories and desired future outcomes. Decision Transformer is categorized as a return conditioned supervised learning (RvS) method.

Unlike traditional RL methods that rely on value functions or policy optimization, DT conditions directly on desired returns. It models the trajectory as a sequence of tokens consisting of states s_t, actions a_t, and returns-to-go R_t. The transformer predicts actions by attending to the entire sequence and considering the relationships between past states, actions, and future rewards.

The input sequence to the transformer is structured as:

$$\tau = (R_t, s_t, a_t, R_{t+1}, s_{t+1}, a_{t+1}, \dots)$$

The model is trained to minimize the negative log-likelihood of actions:

$$L = -\mathbb{E}_{\tau\sim D}[logp(a_t|\tau)]$$

Here, R_t is a return-to-go, representing the sum of future rewards starting from the time step t, s_t is a current state at time t and a_t is an action taken at time t.

We used almost same DT architecture that is implemented in Paster et al. research [3].

Hyperparameter	Value
Batch size	128
Learning Rate	1e-4
Embed dim	128
Num layer	3
Num head	1
Activation Function	ReLU [9]
Dropout	0.1
Weight decay	1e-4
Warmup steps	10000
Discount Factor	1
Target return	1, 2,…,8

Table 3 DT hyperparameters

2.2.5 ESPER

ESPER is a method designed to address the limitations of RvS approaches in stochastic environments. Unlike traditional RvS methods, which condition policies on trajectory returns that may be influenced by environmental stochasticity, ESPER ensures that policies are conditioned on representations independent of such stochasticity. This independence enables ESPER to produce consistent and reliable behaviors even in stochastic settings.

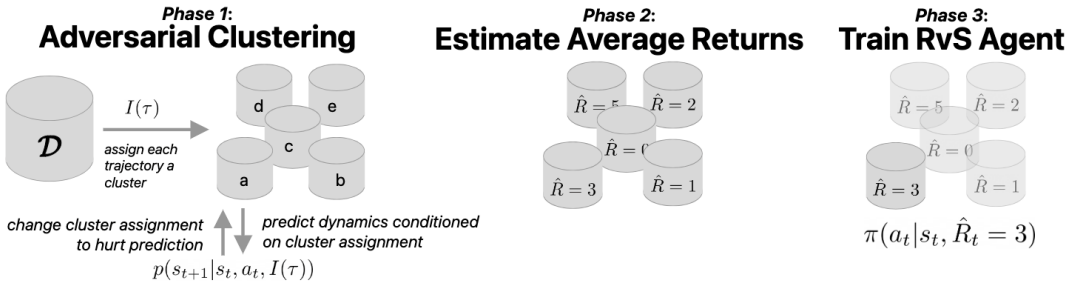

Figure 3 Architecture of ESPER [3]

The implementation of ESPER involves three main phases:

1. Adversarial Clustering

In the first phase, ESPER employs a neural network to assign each trajectory to a cluster based on features independent of environmental stochasticity. This clustering process is adversarial, aiming to minimize the dependency between the cluster assignments and the stochastic outcomes of the environment. The clustering model is trained using two losses, policy reconstruction loss and adversarial loss. Policy reconstruction loss is the loss that encourages the clustering model to capture information about the behavior policy. Adversarial loss is the loss that penalizes cluster assignments that enable the dynamics model to predict next states based on environmental stochasticity.

For the clustering model, a Long Short-Term Memory (LSTM) [12] network is used for trajectory encoding.

2. Estimating Cluster Returns

After clustering, the second phase involves computing the average return for each cluster. The return predictor is trained to estimate these cluster-specific average returns using the following loss function:

$$L(\psi) = \mathbb{E}_{I(\tau)\sim p_\theta(I(\tau)|\tau)}\left[\left|\left|R - f_\psi\left(I(\tau)\right)\right|\right|_2^2\right]$$

Here, R represents the cumulative discounted return for a trajectory and $f_\psi\left(I(\tau)\right)$ predicts the cluster's average return.

For action and return predictors, fully connected Multi-Layer Perceptrons (MLPS) [13] are used.

3. Training the RvS Agent

In this phase, an RvS agent is trained using the clustered average returns as conditioning targets. The policy is learned by minimizing the negative log-likelihood of the actions conditioned on the states and estimated cluster returns:

$$L(\xi) = \mathbb{E}_{s_t,a_t,\hat{R}_t\sim D}\left[-log\pi_\xi\left(a_t|s_t,\hat{R}_t\right)\right]$$

RvS policy is a transformer-based sequence model, following the architecture used in DT. The parameters of DT are same as DT one and are shown in Table 3.

Finally, the architecture of ESPER is shown in Figure 3 and parameter settings are shown in Table 4.

Hyperparameters	Values
Batch size	128
Learning Rate	5e-4
Hidden size	128
Policy hidden layers	2
Clustering model hidden layers	2
Clustering model lstm layers	1
Action predictor hidden layers	2
Return predictor hidden layers	2
Activation Function	ReLU
Optimizer	AdamW [14]
Normalization	Batch norm

Table 4 ESPER Hyperparameters

2.2.6 Evaluation

To evaluate the performance of each method, we implemented a standardized evaluation protocol. Specifically, the evaluation process was conducted every 2,500 training steps. The evaluation process is to conduct only an episode per 2,500 steps. Then the summation of returns is used as a performance value.

3 Results

Our results are represented as discrete values ranging from 0 to 4. Due to the inherently high variance and jagged nature of the raw data, we judged interpreting the trends directly from the original graphs is challenging. So, we applied a 100-step moving average to smooth the results for clearness of visualization and reliable analysis.

For DT and ESPER's results, since we set the target return values to range from 1 to 8, there are eight patterns of results for each method. Figures 4 and 5 show the eight patterns of score transition graphs for DT and ESPER, respectively. From these graphs, we judged that target 1 is the best for DT and target 5 is the best for ESPER. So, we use these as the result of each method.

3.1 Score Transition Graph

We present a graph illustrating the score transitions over training for each method. The X-axis represents the number of evaluations. In the training loops, we evaluate each method every 2500 steps, so evaluation is conducted 400 times. The Y-axis indicates the smoothed scores, calculated using a 100-step moving average.

The results of the transition score are shown in Figure 6. From it, we found that ESPER hit the best score of the other methods. However, many of ESPER's results are staying constant. DT's results also remain constant over time likewise ESPER's one. On the other hand, CQL's results steadily increase after about 200 steps.

Moreover, ESPER looks steadier than DT, because many DT's results finished closing 0 score. (Figures 4 and 5.)

3.2 Comparison Table

We compare these methods by maximum and average scores. The results are shown in Table 5.

From this table, we found that in average score, DT and ESPER is twice as big as CQL. Also, ESPER is a little better than DT. On the other hand, in the maximum score, there are almost no difference in the three methods.

	CQL	DT	ESPER
Max	3	4	4
Average	0.2	0.41	0.55

Table 5 Comparison performance

4 Discussions

From the score transition graphs, ESPER demonstrates a higher average performance compared to both CQL and DT. It achieved relatively higher stability in its results. DT also performed consistently, many of its results converged near a score of 0. It shows DT's limitations in certain target returns. Conversely, CQL tends to increase performance after approximately 200 evaluation steps. It shows its potential for gradual improvement with extended training.

The comparison table reveals that in terms of average score, ESPER outperforms both CQL and DT. DT achieves similar average performance but exhibits more variability, as suggested by its lower minimum scores in the transition graphs.

In maximum scores, all three methods reach comparable performance, suggesting that when optimal trajectories exist, all algorithms can identify and exploit them.

Overall, ESPER achieved equal or better performance than both CQL and DT. So, we can say it ESPER can be utilized in deterministic environment.

However, if we focus on the relationship between the target return and the actual score in the transition graph of DT and ESPER, there are many results that end up not being close to the target return. It shows the possibility of insufficient training steps or suboptimal parameter settings.

5 Conclusion

This study demonstrates that ESPER consistently outperforms both CQL and DT, and it can be utilized in a deterministic environment.

The findings highlight the potential of ESPER for tasks requiring reliable and stable performance in stochastic environments, such as autonomous decision-making and real-world offline RL applications.

However, the discrepancies between the target return and actual performance in both DT and ESPER suggest that further tuning of training steps and parameter settings is necessary to achieve optimal results. Additionally, the current evaluation focuses on discrete scores, which may limit the granularity of performance insights.

Future work could explore more comprehensive parameter optimization strategies and extend the evaluation to include continuous performance metrics for finer-grained analysis. Furthermore, applying these methods to a wider range of environments could validate their generalizability and effectiveness in diverse settings.

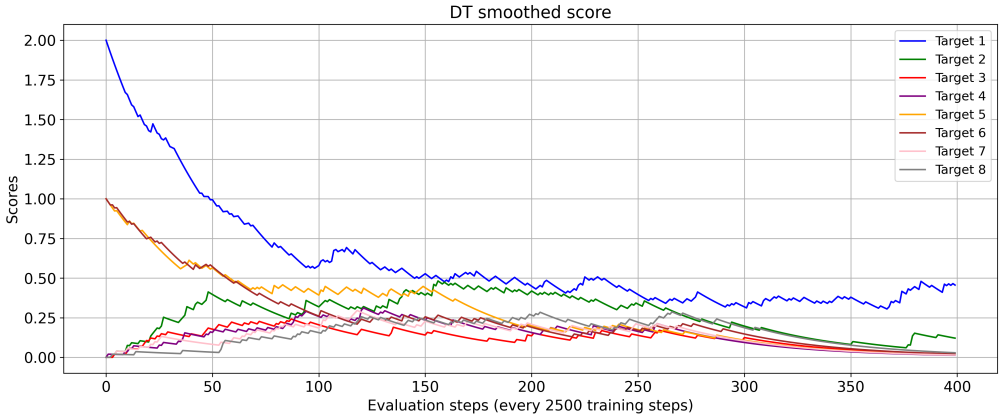

Figure 4 DT smoothed score for each target

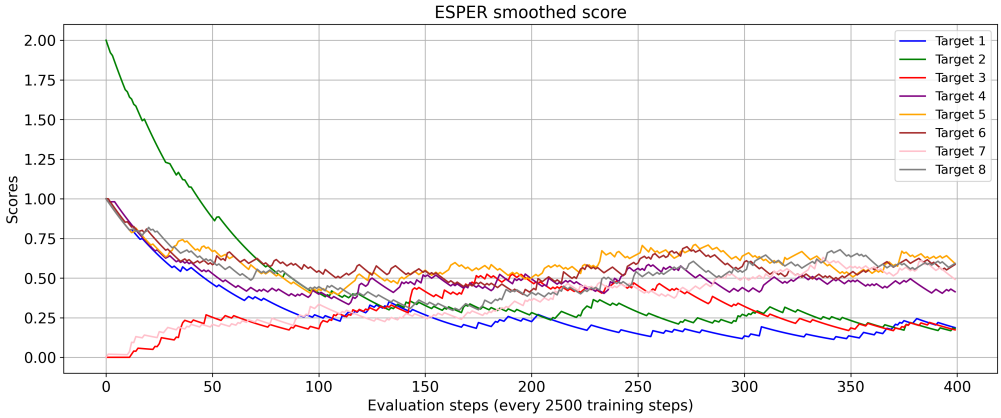

Figure 5 ESPER smoothed score for each target

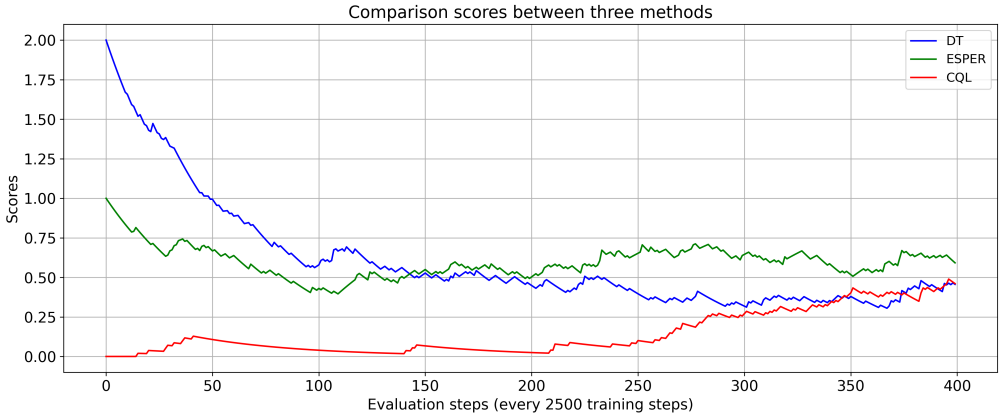

Figure 6 Comparison scores between three methods

Acknowledgment

I would like to thank Prof. Maxim Mozgovoy for his advice on this research, and everyone who cooperated.

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. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual, pages 15084–15097, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/7f489f642a0ddb10272b5c31057f0663-Abstract.html.

[2] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative Q-Learning for Offline Reinforcement Learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS2020, December 6-12, 2020, virtual, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/0d2b2061826a5df3221116a5085a6052-Abstract.html.

[3] Keiran Paster, Shelia A. McIlraith, and Jimmy Ba. You can't count on luck: Why decision transformers fail in stochastic environments. arXiv preprint arXiv:2205.15967, 2022.

[4] GeeksforGeeks, "Deterministic vs Stochastic Environment in AI," [Online]. Available: https://www.geeksforgeeks.org/deterministic-vs-stochastic-environment-in-ai/. [Accessed: Jan. 15, 2025].

[5] Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. J. Artif. Intell. Res., 47:253–279, 2013. doi: 10.1613/jair.3912.

[6] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. arXiv preprint arXiv:1903.03176, 2019.

[7] Papers with Code, "DQN Replay Dataset," [Online]. Available: https://paperswithcode.com/dataset/dqn-replay-dataset. [Accessed: Jan. 15, 2025].

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[9] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In Proceedings of the 27th International Conference on Machine Learning, pages 807–814. Omnipress, 2010.

[10] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. page 14, 2012.

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. Neural computation, 9(8): 1735–1780, 1997. Publisher: MIT Press.

[13] Almeida, L.B. Multilayer perceptrons, in Handbook of Neural Computation, IOP Publishing Ltd and Oxford University Press, 1997.

[14] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.