Appendix

Proof of Theorem 1

Proof. Given the true values of the variances $\mathbf{v} = [\sigma_1^2, \dots \sigma_N^2, \sigma_{target}^2]$, and given $\mathscr{D}$, the vectors $\mathbf{w}_1, \dots \mathbf{w}_N, \mathbf{w}_{target}$ are normally distributed with respective means $\boldsymbol{\mu}_1 \dots \boldsymbol{\mu}_N, \boldsymbol{\mu}_{target}$ and covariances $\sigma_1^2\boldsymbol{\Sigma}_1, \dots \sigma_N^2\boldsymbol{\Sigma}_N, \sigma_{target}^2\boldsymbol{\Sigma}_{target}$ (equation (3)). Then, for any $\mathbf{a} \in \mathbb{R}^N$,

$$\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i \,|\, \mathbf{v}, \mathscr{D} \sim \mathcal{N}(\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i, \sigma_{target}^2 \boldsymbol{\Sigma}_{target} + \sum_i a_i^2 \sigma_i^2 \boldsymbol{\Sigma}_i)$$

and ignoring terms independent of $\mathbf{a}$,

$$\begin{aligned}\mathbb{E}\Big[\|\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i\|_2^2 \,|\, \mathbf{v}, \mathscr{D}\Big] &= \mathrm{tr}(\sigma_{target}^2 \boldsymbol{\Sigma}_{target} + \sum_i a_i^2 \sigma_i^2 \boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i\|_2^2 \\ &\propto \sum_i a_i^2 \sigma_i^2 \mathrm{tr}(\boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i\|_2^2.\end{aligned}$$

Then, taking expectation with respect to $\mathbf{v}$,

$$\begin{aligned}\mathbb{E}\Big[\|\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i\|_2^2 \,|\, \mathscr{D}\Big] &= \mathbb{E}\left[\mathbb{E}\Big[\|\mathbf{w}_{target} - \sum_i a_i \mathbf{w}_i\|_2^2 \,|\, \mathbf{v}, \mathscr{D}\Big] \,\Big|\, \mathscr{D}\right] \\ &\propto \sum_i a_i^2 \, \mathbb{E}[\sigma_i^2 | \mathcal{D}_i] \, \mathrm{tr}(\boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_i a_i \boldsymbol{\mu}_i\|_2^2 \\ &= \sum_{i=1}^N a_i^2 \left(\frac{\beta_i}{\alpha_i - 1}\right) \mathrm{tr}(\boldsymbol{\Sigma}_i) + \|\boldsymbol{\mu}_{target} - \sum_{i=1}^N a_i \boldsymbol{\mu}_i\|_2^2,\end{aligned}$$

where in the last step we used (3) again and the expectation of $\mathrm{InvGamma}(\alpha_i, \beta_i)$. □

Hyper-Parameters for the Neural-Linear Model

Architecture: We set $\mathbf{w}_i \sim \mathcal{N}(0, \mathbf{I})$, $\sigma_i^2 \sim \mathrm{InvGamma}(1, 1)$. For the encoder, we set $d = 20$, use L2 regularization with penalty $\lambda = 10^{-4}$, a learning rate of 10^{-4}, and initialize weights using Glorot uniform. The encoder has two hidden layers with 200 ReLU units each (300 in the first layer for the supply chain problem), and $\tanh$ outputs. We also include a constant bias term in the feature map ϕ when learning $\mathbf{w}$.

Training: Prior to transfer, we first train the neural linear model on 4000 batches of size 64 sampled uniformly from source data. During target training, we train on batches of size 64 sampled from source and target data after each iteration (generation for optimization, episode for supply chain) to avoid catastrophic forgetting of features (one batch for optimization and 20 for supply chain). QPs are solved from scratch at the end of each iteration using the `cvxopt` package [1].

Details for the Static Function Optimization Benchmark

Functions: We consider both transfer and multi-task learning settings. For the transfer experiment, we use the Rosenbrock, Ackley and Sphere functions as the source tasks,

$$\begin{aligned}f_{Rosenbrock}(\mathbf{x}) &= \sum_{i=1}^{9} 100[(x_{i+1} - x_i^2)^2 + (1 - x_i)^2] \\ f_{Ackley}(\mathbf{x}) &= -20 \exp\left(-0.2\sqrt{\frac{1}{10}\sum_{i=1}^{10} x_i^2}\right) - \exp\left(\frac{1}{10}\sum_{i=1}^{10} \cos(2\pi x_i)\right) + 20 + \exp(1) \\ f_{Sphere}(\mathbf{x}) &= \sum_{i=1}^{10} (x_i + 2)^2,\end{aligned}$$

and the Rastrigin function

$$f_{Rastrigin} = 100 + \sum_{i=1}^{10} [(x_i + 2)^2 - 10\cos(2\pi(x_i + 2))],$$

as the target task. We also consider each of the source tasks as the ground truth to see whether we can identify the correct source task in an ideal controlled setting. In the multi-task setting, we did not observe any advantages of our algorithm nor the baselines when solving the Rosenbrock, Ackley or Sphere functions, so we focus on the quality of solution obtained for the Rastrigin function only in the main paper.

Data Processing: For Rosenbrock, sphere and Rastrigin functions, we transform outputs using $y \mapsto \sqrt{y}$ so they become approximately equally scaled. For Rosenbrock, we subsequently also divide outputs by 10. This ensures that the outputs of all functions are approximately equally-scaled, to demonstrate whether we can actually "learn" each function from the data rather than distinguish them according to scale alone.

Solver Settings: To optimize all functions, we use the Differential Evolution (DE) algorithm [2]. The pseudo-code of this algorithm is outlined in Algorithm 2. Here, CR is the crossover probability and denotes the probability of replacing a coordinate in an existing solution with the candidates (crossover), F is the differential weight and controls the strength of the crossover, and NP is the size of the population (larger implies a more global search). We use standard values $CR = 0.7, F = 0.5$ and $NP = 32$ for reporting experimental results, unless indicated otherwise. The search bounds are also set to $[-4, +4]$ for all coordinates; all points that are generated during the initialization of the population and the crossover are clipped to this range.

Algorithm 2 Differential Evolution (DE)

```
Require: $f : \mathbb{R}^D \to \mathbb{R}, CR \in [0, 1], F \in [0, 2], NP \geq 4$
  initialize $NP$ points $\mathcal{P}$ uniformly at random in the search space
  for generation $m = 1, 2, \ldots$ do
    for agent $\mathbf{x} \in \mathcal{P}$ do
      randomly select three candidates $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{P}$ that are distinct from each other and from $\mathbf{x}$
      pick a random index $R \in \{1, 2 \ldots D\}$
      for $i = 1, 2 \ldots D$ do
        pick $r_i \sim U(0, 1)$
        if $r_i < CR$ or $i = R$ then
          set $y_i = a_i + F(b_i - c_i)$
        else
          set $y_i = x_i$
        end if
        if $f(\mathbf{y}) \leq f(\mathbf{x})$ then
          replace $\mathbf{x}$ in $\mathcal{P}$ with $\mathbf{y}$
        end if
      end for
    end for
  end for
  return the agent in $\mathcal{P}$ with the least function value
```

Transfer Learning: We first generate demonstrations $(\mathbf{x}_i, f(\mathbf{x}_i))$ from each source function f using DE (Algorithm 2) and configuration above until a fitness of 0.15 is achieved. Respectively, these have sizes 17693, 6452 and 5853 for each source task. We further transform outputs for training and prediction with the neural-linear model using a log-transform $y \mapsto \log(1 + y)$ to limit the effects of outliers and stabilize the training of the model. When solving the target task, the batch $\mathcal{B}$ is a singleton set containing the best solution from the source data, and O_{base} is trained by replacing the first of the three sampled candidates prior to crossover with probability $p_m = 0.99^m$, where m is the index of the current generation (following the structure of Algorithm 1). This guarantees that the new swarm will possess the traits of the source solutions with high probability, but still maintain diversity so the solution can be improved further.

Multi-Task Learning: We solve all source and target functions simultaneously, sharing the best solutions between them using the mechanisms outlined above for the transfer learning experiment. One QP is maintained for each task to learn weightings over the other tasks excluding itself. Here, we set $p_m = 0.3$, so that the best obtained solutions can be consistently shared between tasks over time.

Details for the Supply Chain Benchmark

Solver Settings: We use the Deep Deterministic Policy Gradient (DDPG) Algorithm [3] to solve this problem. The critic network has 300 hidden units per layer. We also use L2 penalty $\lambda = 10^{-4}$, learning rates 10^{-4} and 2×10^{-4} for actor and critic respectively, $U(-3 \times 10^{-3}, 3 \times 10^{-3})$ initialization for weights in output layers, discount factor $\gamma = 0.96$, horizon $T = 200$, randomized replay with capacity 10000, batch size of 32, and explore using independent Gaussian noise $\mathcal{N}(0, \sigma_t^2)$, where σ_t is annealed from 0.15 to 0 linearly over 50000 training steps.

Transfer Learning: We considered the transfer from multiple source tasks (Scenarios 1, 2 and 3) to a single target task (Target Task) as indicated in Figure 3. We also consider each source task as a ground truth. To collect source data, we train DDPG with the above hyper-parameters on each source task for 30000 time steps, then randomly sub-sample 10000 observations. This last step demonstrates whether our approach can learn stable representations with limited data and incomplete exploration

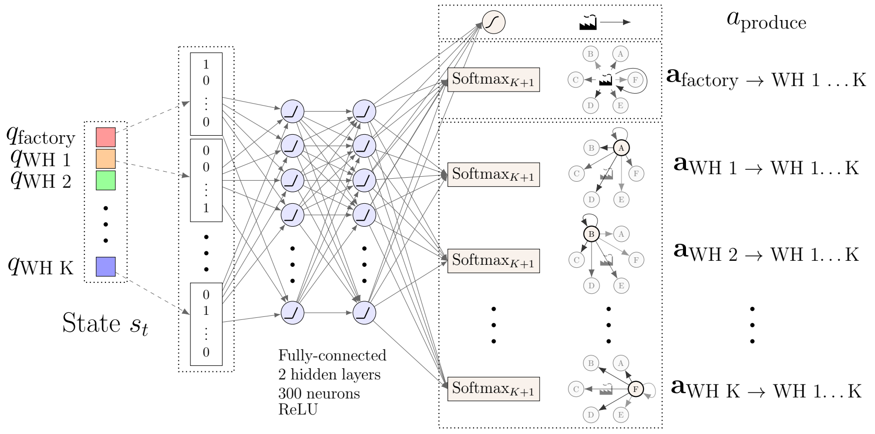


Figure 5: Actor network.

trajectories. Since regression is sensitive to outliers, when training the neural linear model, we remove 2.5% of the samples with the highest and lowest rewards (we also exclude observations collected from the target task that lie outside any of these bounds when training the neural-linear model). In order to implement transfer, we set $p_m = 0.95^m$, where m is the current episode number. This provides a reasonable balance between exploration of source data and exploitation of target data. In accordance with Algorithm 1, we sample batches $\mathcal{B}$ of size 32 uniformly at random from the source data.

Prioritized Experience Replay: We evaluated the performance of Prioritized Experience Replay (PER) [4] on the Supply Chain benchmark. In summary, PER is a replay buffer that ranks experiences using the Bellman error, defined for the DDPG algorithm for a transition (s, a, r, s') as

$$\delta = r + \gamma Q'(s', \mu'(s')) - Q(s, a),$$

where Q is the critic network, and Q' and μ' are the target critic and target actor networks, respectively.

The source demonstrations are combined into a single data set and shuffled. The capacity of the buffer is also fixed to the total number of source demonstrations from all tasks (around 28,000 examples). We then load all source demonstrations into the replay buffer prior to target task training with initial Bellman error $\delta = 1$. During target training, observed transitions are immediately added to the replay buffer and override the oldest experience. In this way, the agent is able to maximize the use of the source data at the beginning of training but eventually shift emphasis to target task data.

Figure 6 demonstrates the composition of each batch (of size 32) according to source (Scenario 1, Scenario 2, Scenario 3, Target Task) for each ground truth. As illustrated in all four plots, in early stages of training, batches are predominantly composed of source data, while in later stages, they consist entirely of target data. Figure 6 shows that PER is unable to favor demonstrations from the source task that correspond to the ground truth, in all four problem settings. One possible explanation of this is that an implicit assumption of PER is violated, namely that experiences are drawn from the same distribution of rewards, whereas in our experiment, experiences can come from tasks with contradictory goals. In this case, experiences corresponding to the ground truth do not necessarily have larger Bellman error (in fact, the opposite may be true).

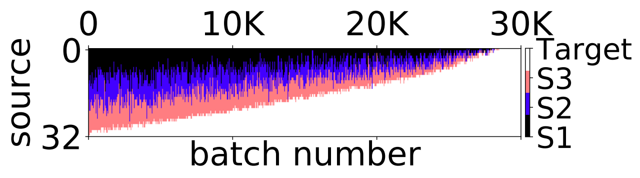
Latent Space Analysis

We include several plots that couldn't go in the main paper due to space limitations. In particular, we analyze the learned latent reward functions $\mathbf{w}$ for source and target tasks for both the function optimization and supply chain problems.

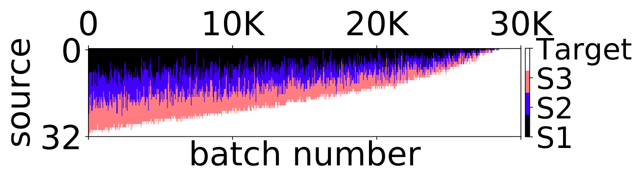
The analysis for the function optimization problem is illustrated in Figure 7 in which we set the latent dimension $d = 2$ for ease of illustration. Similarly, the analysis for the supply chain problem using the original value of $d = 20$ (we tried smaller values of d, but did not obtain satisfactory results due to the complexity of the true reward function for this problem) is illustrated in Figure 8. In both problem settings, BERS is able to learn a meaningful latent representation of the demonstrators' goals.

Additional Experiment for the Reacher Domain

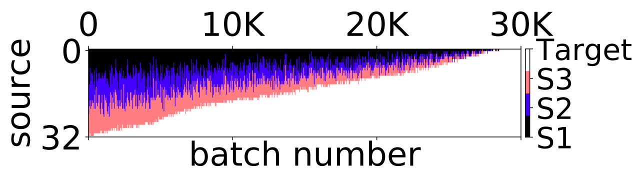
Domain Description: The reacher domain consists of a two-jointed robotic arm that must be controlled to hover above a fixed target location. The state is 4-dimensional and consists of the angle and angular velocities of the two joints. At the beginning of each episode, the angular velocities are set to zero, while the central joint angle is sampled uniformly from $[-\pi, +\pi]$ and the outer joint angle is sampled uniformly from $[-\pi/2, +\pi/2]$. The 2-dimensional continuous action space represents the possible torques that can be applied to each joint, and are discretized into 3 possible actions per dimension (corresponding to minimum, maximum, and zero torques) for a total of 9 possible actions. Following [Barreto *et al.*, 2017], we initialize 4 source



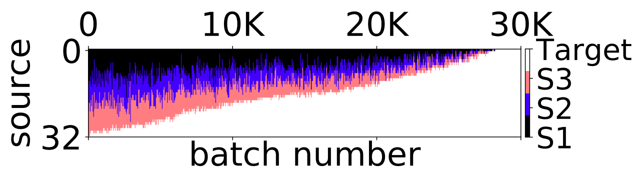
(a) Scenario 1



(b) Scenario 2

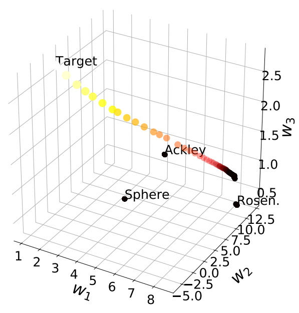


(c) Scenario 3

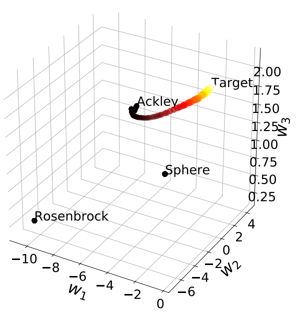


(d) Scenario 4

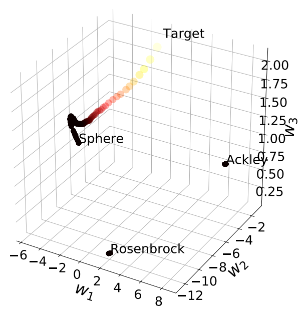
Figure 6: The composition of each batch over time sampled from the prioritized replay (PER) according to whether the sample came from the source or target task data, for each ground truth. PER is unable to rank experiences correctly to match the context.



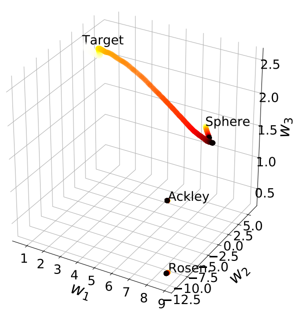
(a) Rosenbrock



(b) Ackley

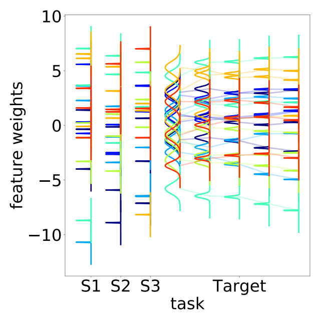


(c) Sphere

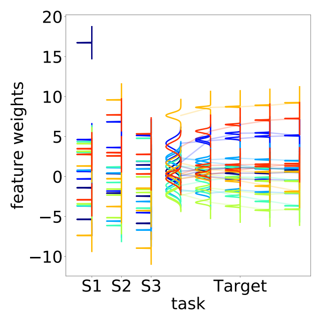


(d) Rastrigin

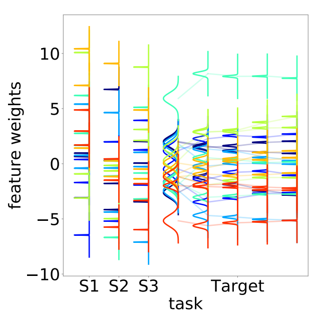
Figure 7: For function optimization with each source and target task as the ground truth, shows the evolution of the posterior mean of each $\mathbf{w}_i$ and $\mathbf{w}_{target}$ during training for the simplified experiment with latent dimension $d = 2$. As shown here, the target mean eventually converges to the correct source task mean.



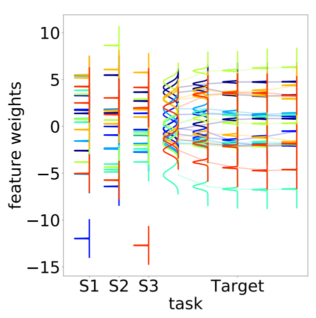
(a) Scenario 1



(b) Scenario 2



(c) Scenario 3



(d) Target

Figure 8: For Supply Chain control with each source and target task as the ground truth, shows the posterior marginal distributions of $\mathbf{w}_i$ and $\mathbf{w}_{target}$ during training on the target task (after 0, 10, 50, 100 and 200 episodes). Over time, the target features begin to concentrate on the corresponding values for the correct source task.

task instances whose target locations are indicated as colored circles in Figure 9, that are used to train each demonstrator. We define 2 additional task instances as target tasks whose target locations are indicated as gray circles. The target task whose goal position is located close to the top-right corner of the map is similar to the demonstrator associated with the yellow goal position, whereas the target task whose goal position is in the bottom-left is considerably different from any of the source tasks. The reward is defined as $1 - 4\delta$, where δ is the Euclidean distance between the arm tip and the target location.

Solver Settings: All tasks are solved using the DQN algorithm [5]. The Q-value function is parameterized as a feedforward neural network containing two hidden layers of size 256 with $\tanh$ activation functions, followed by a linear output layer of size 9 to represent the Q-value of each action. The Adam optimizer with a learning rate of 10^{-3} is used to train the network, on batches of size 32 sampled at random from a replay buffer of infinite capacity. We use a discount factor $\gamma = 0.9$ and horizon $T = 500$ during training. The epsilon-greedy policy with $\varepsilon = 0.1$ is used for training and $\varepsilon = 0.03$ for testing. The neural-linear model for BERS uses the same parameters indicated above, except that the activation function for hidden units is replaced by $\tanh$ and the learning rate is changed to 5×10^{-4}.

Transfer Learning: The demonstrators are trained on each of the 4 training target locations for 30000 time steps. The training data is saved, and a random sample of 3200 transitions is used to pre-train BERS and PPR. The neural-linear models are trained on these subsets of demonstrations by sampling 20000 batches of size 32. The policy network for PPR has the same structure as the Q-value network described above, except the output layer uses a $\mathrm{softmax}$ activation function. The Adam optimizer is used to train the policy networks using each of the 4 training task demonstrations data sets, and uses a learning rate of 10^{-3}. These networks are pre-trained on batches of size 32 for a total of 1000 epochs. Similar to the supply chain problem, we set $p_m = 0.95^m$, where m is the current episode number. A budget of 30000 time steps is allocated for training on each of the target task locations for BERS and PPR.

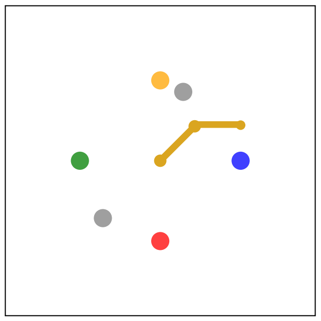
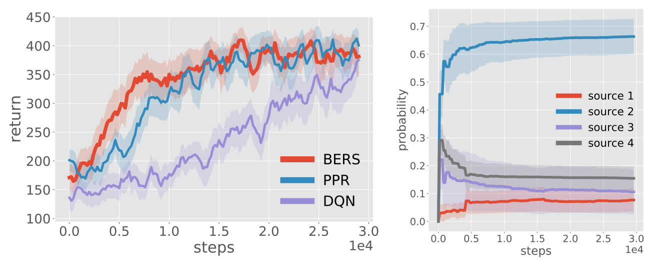
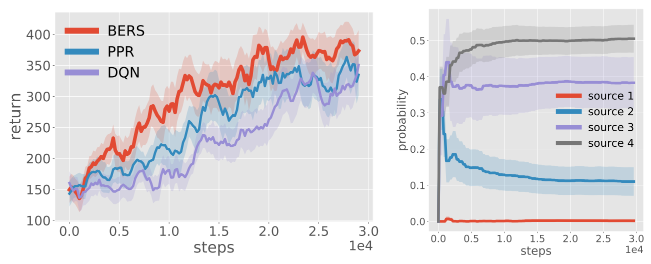


Figure 9: The reacher domain. The 4 colored circles indicate target locations used to train the demonstrators. The target locations for the target tasks are indicated in gray circles.

We compare BERS to PPR and standard DQN, and report the results in Figure 10. Here, we see that the performance of BERS is better than PPR on both test tasks. Interestingly, while the performance improvement on the top-right target location is relatively insignificant, the benefits of BERS are quite substantial on the more difficult target task with the bottom-left target. Looking at the plot of the posterior weights $\mathbf{w}$, we see that BERS was able to identify two demonstrators – with target locations located at the bottom and the left of the space – as being the most relevant. As previously demonstrated on the supply chain domain, BERS is once again able to select a *combination* of two relevant demonstrators. By mixing the demonstrations from a subset of the most relevant demonstrators, it is possible to obtain better performance than training on a single policy or demonstrator.



(a) Top-right target.



(b) Bottom-left target.

Figure 10: Total testing reward per episode (left) and weights assigned to source tasks (right) over epochs using DQN for the reacher domain, for the two target positions. Averaged over 5 trials with shaded standard error bars.

References

[1] Andersen, Martin S., Joachim Dahl, and Lieven Vandenberghe. ”CVXOPT: A Python package for convex optimization.” abel. ee. ucla. edu/cvxopt (2013).
[2] Storn, Rainer, and Kenneth Price. ”Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces.” Journal of global optimization 11.4 (1997): 341-359.
[3] Lillicrap, Timothy P., et al. ”Continuous control with deep reinforcement learning.” ICLR (Poster). 2016.
[4] Wang, Ziyu, et al. ”Dueling network architectures for deep reinforcement learning.” International conference on machine learning. PMLR, 2016.
[5] Mnih, Volodymyr, et al. ”Human-level control through deep reinforcement learning.” nature 518.7540 (2015): 529-533.