Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Long-Term Motion Prediction in Traffic

Katharina Hengel

Course of Study:	Computer Science
Examiner:	Ph. D. Jim Mainprice
Supervisor:	Dr. Luigi Palmieri
Commenced:	May 27, 2020
Completed:	November 27, 2020

Abstract

The field of Inverse Reinforcement Learning (IRL) addresses the task of finding a cost function which describes expert behavior. Since the cost function is solely computed from expert demonstrations, the sample complexity exerts influence on the performance of these algorithms. In this thesis we study the Learning to Search (LEARCH) and the maximum entropy IRL framework as example IRL techniques. Based on these two algorithms we develop a variation of the LEARCH algorithm using the idea of maximum entropy IRL. In the next step we extend LEARCH to Deep-LEARCH as well as the newly developed LEARCH variation to a equivalent Deep-LEARCH variation. Thereby we generalize the cost function to function space using Convolutional Neural Networks (CNNs). Including maximum entropy inside the Deep-LEARCH variation increases the density of the target maps of the CNN.

We discover that LEARCH shows the lowest sample complexity among the investigated algorithms, while maximum entropy shows the highest sample complexity. In the deep learning setting the increased density of the CNN target maps did not improve the performance. Hence, the performance does not change, if the algorithms are extended by CNNs to the function space.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

CNN Convolutional Neural Network. 3, 17

GAN Generative Adversarial Network. 17

IRL Inverse Reinforcement Learning. 3, 15

LEARCH Learning to Search. 3, 17

MDP Markov Decision Process. 19

ReLU Rectified Linear Unit. 71

1 Introduction

Machine Learning has changed the research in the traffic system over the last decades. The goal of self driving cars encourages a multitude of new discoveries, many of which have the goal to predict paths. Whether it is the path of a car parking itself into a parking spot, the fastest route between two cities or the path of surrounding road users, that has to be evaded. The problem of trajectory prediction, i.e. the task of predicting the trajectory from a given start location to a given target location, has many different use cases. Since there can be a lot of influencing factors, like pavement, obstacles and traffic infrastructure, the task of path planning can get complex very easily.

Thus, one technique, which can be used, is imitation learning. In this field the predicted trajectory is inferred only from expert behavior. Imitation learning is best used to learn a complex task, when a lot of expert demonstrations are available. It can be modeled as part of Inverse Reinforcement Learning (IRL), which means that a cost function is derived from the expert demonstration. The predicted trajectory which is described by the optimal policy is inferred from this cost function afterwards. Thereby the cost function does not need to be hand designed, which can get difficult for complex tasks.

2 Related Work

Since IRL is an ill-posed problem, a policy can be described by multiple cost functions. Hence, a further objective is necessary to obtain a unique solution. We distinguish between model-based and model-free IRL methods. While the system dynamics are known for the former, this does not apply to the later.

In model-based IRL there are two common techniques which solve the ambiguity between cost functions by maximizing either a margin or the entropy. Ratliff et al. [RBZ06] introduced the idea of maximizing a margin in the context of route planning. This technique was developed further in the Learning to Search (LEARCH) framework [RSB09]. The principle of maximum entropy on the other hand was applied to IRL by Ziebart et al. [ZMBD08].

In both techniques the cost function is modeled as a linear combination of features. Although Ratliff et al. [RSB09] starts with a linear cost function approximation, they already extend the LEARCH framework to function space using boosting. The maximum margin as well as the maximum entropy technique were further developed using Convolutional Neural Networks (CNNs) to model non-linear cost functions [MBK+16] [WOP16].

Finn et al. [FLA16] presented a model-free IRL approach for non-linear function approximation. Furthermore, Ho and Ermon [HE16] introduce a model-free imitation learning approach, combining Generative Adversarial Networks (GANs) and IRL. However, it does not recover the cost function explicitly. Thus, it is not an IRL algorithm in a narrower sense. In the thesis we will focus on model-based IRL methods.

3 Inverse Reinforcement Learning

The goal of the IRL problem is to find a suitable cost function for a set of expert demonstrations [NR00]. Taking always the least cost action, we obtain a policy which describes the observed behavior of the expert.

The problem is modeled as a Markov Decision Process (MDP) $\mathcal{M} = \{\mathcal{X}, \mathcal{A}, \mathcal{T}, c\}$, where $\mathcal{X}$ denotes the state space, $\mathcal{A}$ the set of possible actions and $\mathcal{T}$ the transition function. The cost function c is the function of interest. It is often assumed to be a linear combination of features $c(x, a) = w^T \cdot \phi(x, a)$.

Furthermore let $\Xi = \{\xi_0, ..., \xi_N\}$ be the set of expert demonstrations. A demonstration is denoted by a set of state-action pairs $\xi_i = \{(x_0, a_0), (x_1, a_1), ..., (x_K, a_K)\}$. We assume a deterministic policy. Hence, the action is implicitly determined by the subsequent state sequence. Therefore, we write $\xi_i = \{x_0, x_1, ..., x_K\}$ instead of $\xi_i = \{(x_0, a_0), (x_1, a_1), ..., (x_K, a_K)\}$ and $c(x) = w^T \cdot \phi(x)$ instead of $c(x, a) = w^T \cdot \phi(x, a)$.

There are several methods to solve the IRL problem, two of which we will introduce in the following sections. While the first approach extends the Maximum Margin Planning framework of Ratliff et al. [RBZ06], the second technique is based on the maximum entropy principle. In Chapter 4 we will then introduce a newly developed algorithm which incorporates ideas of both algorithms explained beforehand.

3.1 LEARCH

The following chapter introduces the LEARCH framework. Here we model the cost function as linear combination of features $c(x) = w^T \cdot \phi(x)$. Afterwards we introduce Deep-LEARCH, which uses the afore mentioned LEARCH framework in combination with CNNs to model non-linear cost functions.

3.1.1 LEARCH as Linear Function Approximation

The LEARCH framework was presented by Ratliff et al. [RSB09] in "Learning to search: Functional gradient techniques for imitation learning". It is based on the Maximum Margin Planning framework [RBZ06].

Let the cost be a linear combination of features $c(x) = w^T \cdot \phi(x)$ with w being the weights and $\phi(x)$ the features. Starting with the weights initialized to one, we use gradient descent steps to improve the current solution step-by-step.

The idea of the LEARCH algorithm is to adjust the cost function in a way, that the expert demonstrations are the minimum cost paths on the solution costmap. For each demonstration ξ, we plan the minimum cost path between its start and target state on the current costmap. We denote this example path as ξ^*. Iteratively increasing the costs along the example path ξ^* and decreasing the costs along the demonstrations ξ, leads to ξ being the least cost path. If both paths match, the shifting of the costs counterbalance. Hence, the costs converge.

Adding a loss onto the costmap before planning the example path ξ^* includes a maximum margin objective in the algorithm. In the end of the algorithm the costs of the demonstration ξ don't only have to be smaller than the costs of any other path between its start and target states, but the costs have to be smaller by a margin which scales with the loss. This technique improves generalization.

Like Ratliff et al. [RSB09] we used a scaled version of the hamming loss in our implementation of the LEARCH algorithm. The hamming loss assigns every state-action pair a loss of 1, if it is not part of the demonstration ξ_i and 0 otherwise.

$$\text{(3.1)}\quad \text{hamming loss: } l_i(x,a) = \begin{cases} 1, & \text{if } x \notin \xi_i \\ 0, & \text{if } x \in \xi_i \end{cases}$$

The scaled hamming loss assigns state-action pairs in the demonstrations ξ_i the lowest loss values while states further away are gradually assigned higher loss values.

$$\text{(3.2)}\quad \text{scaled hamming loss: } l_i(x) = a - (a \cdot \exp(-\frac{1}{2} \cdot (\frac{edt(x,\xi_i)}{s})^2))$$

Here, $edt(x,\xi_i)$ denotes the euclidean distance transform of state x to the demonstration, i.e. the length of the shortest path from state x to any state in the demonstration. The scalar a determines the highest loss value any state x can achieve, whereas s gradually scales the amount of loss for states nearby the trajectory ξ_i. For a small s the area, which is impacted, is more narrow, while for a larger s the area is wider. Figure 3.1 presents an example map of the scaled hamming loss.

To improve robustness, we use exponentiated gradient descent having the following update rule:

$$\text{(3.3)}\quad w_{t+1} = w_t \cdot \exp(\alpha_t \cdot w_h)$$

Similar to Ratliff et al. [RBZ06] we choose the step size α_t to be $\frac{r}{t+m}$ with r being the learning rate, t the iteration count and m a scalar regulating the amount of decrease between two steps.

The informally described procedure results in the Algorithm 3.1. The data set $\mathcal{D}$ is initialized in every iteration step with the empty set. For every start and corresponding target state of a sample trajectory the least cost path ξ^* in the loss-augmented costmap is computed. We use the Dijkstra algorithm in our implementation to compute this path. Afterwards, all states of the demonstration ξ and the example path ξ^* are added to the data set $\mathcal{D}$ with its tendency whether to increase or decrease the cost in this state. More precisely, for each state x in the demonstration ξ we add the tuple $(x,-1)$ and for each state x in the example path ξ^* we add the tuple $(x,-1)$ to $\mathcal{D}$.

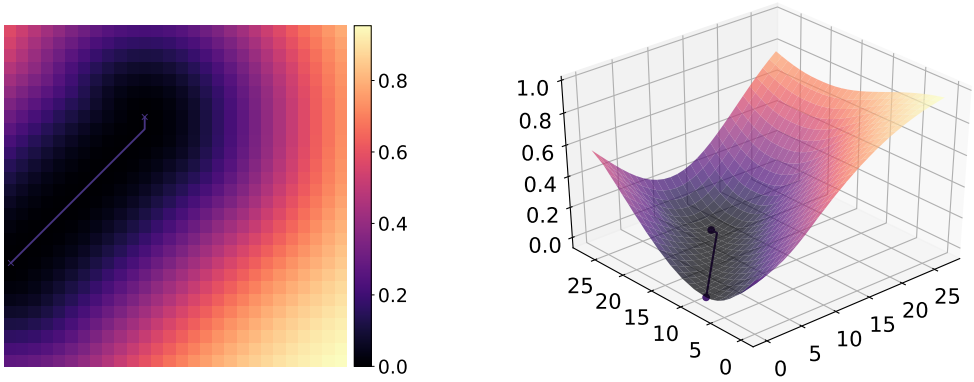


Figure 3.1: Scaled hamming loss map with $a = 1$ and $s = 10$

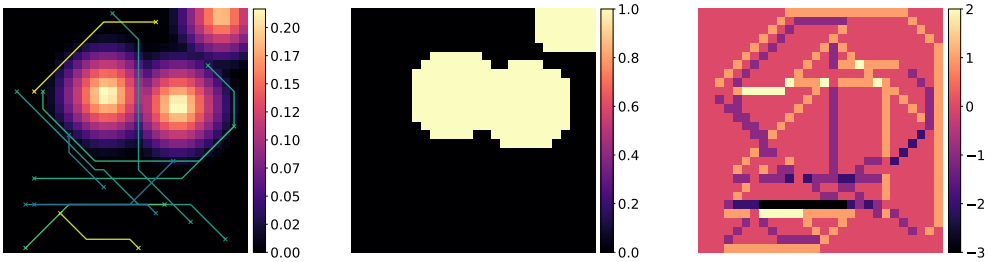
Algorithm 3.1 LEARCH algorithm

procedure LEARCH($\{\xi_i\}_{i=0}^{N}$)
  $w_0 = (1, ..., 1)^T$
  while $\|w_{t+1} - w_t\| < \epsilon$ **do**
    $\mathcal{D} = \emptyset$
    for all example trajectories ξ_i **do**
      $\xi_i^* = \underset{\xi_i^*}{\text{argmin}} \sum_{x \in \xi_i^*} (c(x, w_t) - l_i(x))$
      $\mathcal{D} = \mathcal{D} \cup \{(x, 1) | x \in \xi_i^*\}$
      $\mathcal{D} = \mathcal{D} \cup \{(x, -1) | x \in \xi_i\}$
    end for
    $w_h = \text{argmin}_{w_h} \frac{1}{2} \sum_{(x,y) \in \mathcal{D}} (y - c(x, w_h))^2 + \frac{\lambda_1}{2} \|w_h - w_t\|^2 + \frac{\lambda_2}{2} w_h$
    $w_{t+1} = w_t \cdot \exp(\alpha_t \cdot w_h)$
  end while
  return w_{t+1}
end procedure

Now let X be a vector of all the states where the costs have to be adjusted, and Y the vector encoding the corresponding tendency to increase or decrease the cost in that state. Let w_h be the weight vector computed by linear regression of X and Y with proximal regularization. Hence, w_h is the vector indicating the direction in which the weights w have to be adjusted. Thus, it is the gradient of the LEARCH loss, which we use to update our current solution w_t.

To improve generalization of the algorithm, we extend the LEARCH algorithm to learn on multiple environments simultaneously. This means, we average the gradient w_h over multiple environments before updating the weights with the next gradient step.

All in all we get the following loss and gradient for the LEARCH algorithm:



(a) Ground truth costmap with ten demonstrations

(b) Occupancy map

(c) Gradient map

Figure 3.2: Ground truth costmap and corresponding input and target maps for the CNN in Deep-LEARCH using ten demonstrations

$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\Big(\sum_{x\in\xi_i} w^T \cdot \phi(x) - min_{\xi_i^*}\Big(\sum_{x\in\xi_i^*} w^T \cdot \phi(x) - l_i(x)\Big)\Big) \tag{3.4}$$

$$\nabla\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\Big(\sum_{x\in\xi_i} \phi(x) - \sum_{x\in\xi_i^*} \phi(x)\Big) \tag{3.5}$$

3.1.2 Deep-LEARCH

Deep-LEARCH [MBK+16] extends the previously described algorithm with CNNs, which allows for non-linear approximation of the cost function. Hence, we get the following loss of the cost function c:

$$\mathcal{L}(c) = \frac{1}{N}\sum_{i=1}^{N}\Big(\sum_{x\in\xi_i} c(x) - min_{\xi_i^*}\Big(\sum_{x\in\xi_i^*} c(x) - l_i(x)\Big)\Big) \tag{3.6}$$

The difference between LEARCH and Deep-LEARCH is in the computation of the gradient w_h. Instead of using a linear regression in line 10 of Algorithm 3.1, a CNN is used to compute the functional gradient of the loss. For every training environment a tuple of the occupancy map and the gradient map is created. The occupancy map is created by setting every state to one which is occupied by an obstacle in the ground truth costmap. All other states are zero. An example occupancy map is presented in Figure 3.2b. The creation of the gradient maps follows the idea of the LEARCH algorithm. It is computed by increasing the value of each state on the example paths by one and decreasing the value of each state on the demonstration by one. The example path is hereby computed on the loss augmented costmap. The exact algorithm is described in Algorithm 3.2. Figure 3.2c shows such a target map of Deep-LEARCH. We also call the occupancy map the input map of the CNN, and the gradient map the target map.

Algorithm 3.2 Update of a target map of the CNN in Deep-LEARCH

```
1:  procedure UpdateTargetMap(c, {ξ_i}_{i=0}^N)
2:      D = (0, ..., 0)
3:      for all example trajectories ξ_i do
4:          ξ_i^* = argmin_{ξ_i^*} Σ_{x∈ξ_i^*} (c(x) − l_i(x))
5:          for all x ∈ ξ_i^* do
6:              D(x) = D(x) + 1
7:          end for
8:          for all x ∈ ξ_i do
9:              D(x) = D(x) − 1
10:         end for
11:     end for
12:     return D
13: end procedure
```

The result c_h^j of the CNN for an occupancy map of the environment j is used to update the current costmap of the environment j.

$$c_{t+1}^j = c_t^j \cdot \exp(\alpha_t \cdot c_h^j) \tag{3.7}$$

We use the same step size $\alpha_t = \frac{r}{t+m}$ as in the linear case.

3.2 Maximum Entropy

Like before, we differentiate two approaches. The first one models a linear cost function while the second approach models a non-linear cost function.

3.2.1 Maximum Entropy as Linear Function Approximation

Since the inverse reinforcement learning problem is an ill-posed problem, many reward functions can be optimal for a given set of demonstrations. Ziebart et al. [ZMBD08] resolves this ambiguity by choosing the solution which matches feature counts of the expert demonstrations and the learner's demonstrations while maximizing entropy of the distribution over paths. The latter is done by

maximizing the likelihood of the observed expert demonstrations. Following is the corresponding loss formalized:

$$\begin{aligned}\mathcal{L}(w) &= \sum_{i=1}^{N} \log P(\xi_i|w) \\ &= \sum_{i=1}^{N} \log \frac{1}{Z(w)} \exp(\sum_{x\in\xi_i} w^T \cdot \phi(x)) \\ &= \sum_{i=1}^{N} \sum_{x\in\xi_i} w^T \cdot \phi(x) - N \log Z(w) \\ &= \sum_{i=1}^{N} \sum_{x\in\xi_i} w^T \cdot \phi(x) - N \log \sum_{\xi} \exp(\sum_{x\in\xi} w^T \cdot \phi(x))\end{aligned} \tag{3.8}$$

$P(\xi_i|w)$ denotes the likelihood of the expert demonstration ξ_i in the current costmap using the weight w and $Z(w)$ the partition function. The gradient of the loss is:

$$\begin{aligned}\nabla\mathcal{L}(w) &= \sum_{i=1}^{N} \sum_{x\in\xi_i} \phi(x) - N \frac{1}{\sum_{\xi} \exp(\sum_{x\in\xi} w^T \cdot \phi(x))} \sum_{\xi} \exp(\sum_{x\in\xi} w^T \cdot \phi(x)) \sum_{x\in\xi} \phi(x) \\ &= \sum_{i=1}^{N} \sum_{x\in\xi_i} \phi(x) - N \sum_{\xi} P(\xi|w) \sum_{x\in\xi} \phi(x) \\ &= \sum_{i=1}^{N} \sum_{x\in\xi_i} \phi(x) - N \sum_{x} P(x|w)\phi(x) \\ &= \sum_{i=1}^{N} \sum_{x\in\xi_i} \phi(x) - N \sum_{x} D_x\phi(x)\end{aligned} \tag{3.9}$$

Dividing the gradient by the number of expert demonstrations gives:

$$\frac{1}{N}\nabla\mathcal{L}(w) = \frac{1}{N} \sum_{i=1}^{N} \sum_{x\in\xi_i} \phi(x) - \sum_{x} D_x\phi(x) \tag{3.10}$$

with $\frac{1}{N} \sum_{i=1}^{N} \sum_{x\in\xi_i} \phi(x)$ being the empirical feature count of the expert and $\sum_x D_x\phi(x)$ being the expected feature count of the learner. Let D_x denote the expected state frequency of state x. We compute D_x using the forward-backward algorithm proposed by Ziebart et al. [ZMBD08]. It is presented in Algorithm 3.3. The scalar N determines the number of iterations done in the expected state frequency calculation. The starting states of the demonstrations are used to calculate the initial state probabilities. The target states of the demonstrations are used as terminal states.

Figure 3.3a shows a costmap, where the empirical feature count of one demonstration is used as weight vector of the costmap. The ground truth of the underlaying costmap is presented in Figure 3.3b. An example of the expected state frequencies of three demonstrations is presented in

Algorithm 3.3 Expected state frequency calculation

procedure EXPECTED STATE FREQUENCY CALCULATION(N, starts, targets)
    1. Backward pass
    if $x_i \in$ targets **then**
        $Z_{x_i} = 1$
    else
        $Z_{x_i} = 0$
    end if
    for 1 to N **do**
        $Z_{a_{i,j}} = \sum_k P(x_k|x_i, a_{i,j}) \exp(-c(x_i, w)) Z_k$
        $Z_{x_i} = \sum_{a_{i,j}} Z_{a_{i,j}}$
    end for
    2. Local action probability computation
    $P(a_{i,j}|x_i) = \frac{Z_{a_{i,j}}}{Z_{x_i}}$
    3. Forward pass
    if $x_i \in$ starts **then**
        $D_{x_i,0} = 1$
    else
        $D_{x_i,0} = 0$
    end if
    $D_{x_i,0} = \frac{D_{x_i,0}}{\sum_{x_i} D_{x_i,0}}$
    for t = 1 to N **do**
        $D_{x_i,t+1} = \sum_{a_{i,j}} \sum_k D_{x_k,t} P(a_{i,j}|x_i) P(x_k|s_i, a_{i,j})$
    end for
    4. Summing frequencies
    $D_{x_i} = \sum_t D_{x_i,t}$
    return D_{x_i}
end procedure

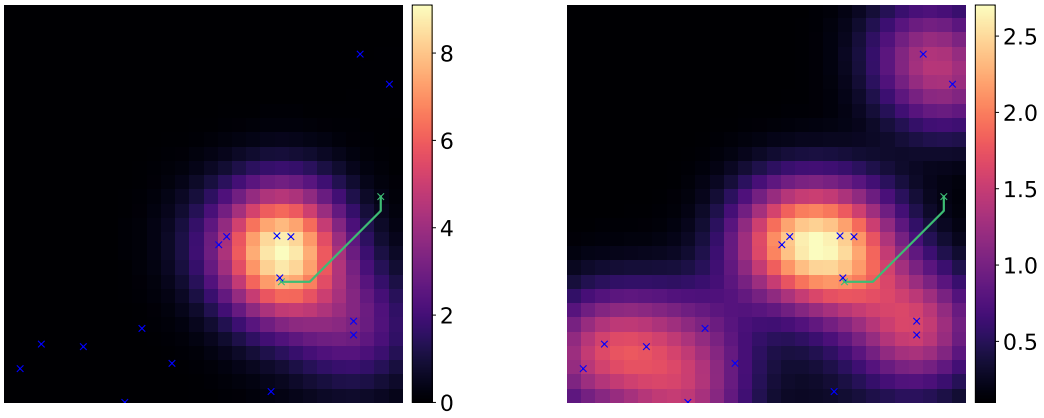
Figure 3.4a. The associated costmap is illustrated in Figure 3.4b. The centers of the radial basis functions are marked with blue crosses. Since we work with costs and not rewards, we inverted the weights before computing the costmap, such that states with a high visitation count have low cost.

We iteratively update the weights w using gradient descent:

$$w_{t+1} = w_t + \alpha_t \nabla \mathcal{L}(w) \tag{3.11}$$

We use the same step size $\frac{r}{t+m}$ as in the LEARCH algorithm.

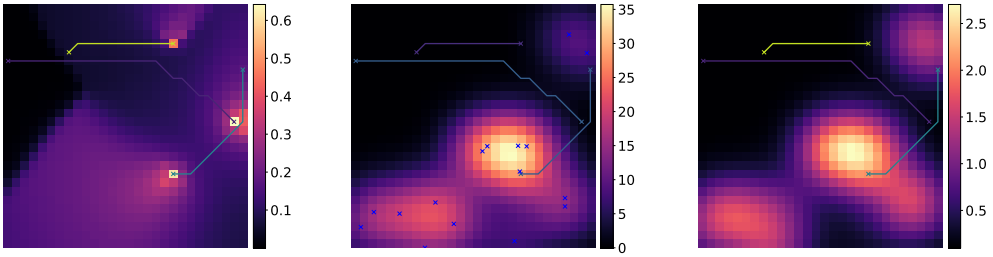
Furthermore, we improve generalization by averaging over multiple environments. Like in the LEARCH algorithm we average the gradient of multiple environments before updating the weights.



(a) Costmap with the empirical feature count of one demonstration as weight vector

(b) Ground truth costmap

Figure 3.3: Empirical feature count and ground truth costmap



(a) Expected state frequencies of three demonstrations

(b) Costmap with the expected state frequencies of three demonstrations as weights

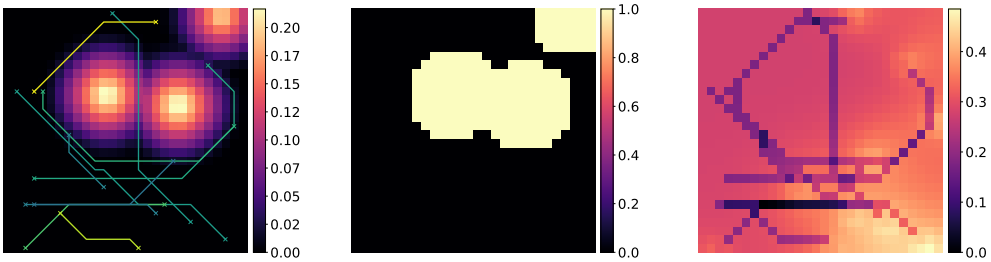
(c) Ground truth costmap

Figure 3.4: Expected state frequencies, costmap of the expected state frequencies and the ground truth costmap

3.2.2 Maximum Entropy using Convolutional Neural Networks

The idea is to extend the maximum entropy IRL framework by a CNN similar to Deep-LEARCH and LEARCH. Hereby the gradient of the maximum entropy loss is included into the target maps of the CNN. The maximum entropy loss is now formalized by:

$$\mathcal{L}(c) = \sum_{i=1}^{N} \sum_{x \in \xi_i} c(x) - N \log \sum_{\xi} \exp\Big(\sum_{x \in \xi_i} c(x) \Big) \tag{3.12}$$



(a) Ground truth costmap with ten demonstrations

(b) Occupancy map

(c) Gradient map

Figure 3.5: Ground truth costmap and corresponding input and target maps for the CNN in maximum entropy using ten demonstrations

Algorithm 3.4 Update of a target map of the CNN using maximum entropy

procedure UPDATETARGETMAP(c, $\{\xi_i\}_{i=0}^{N}$)
  $\mathcal{D} = (0, ..., 0)$
  for all example trajectories ξ_i **do**
    for all $x \in \xi_i$ **do**
      $\mathcal{D}(x) = \mathcal{D}(x) - 1$
    end for
  end for
  for all $x \in \mathcal{X}$ **do**
    $\mathcal{D}(x) = \mathcal{D}(x) + D_x$
  end for
  return $\mathcal{D}$
end procedure

For a number of environments occupancy maps and corresponding target maps are created. We use the ground truth costmap to compute the occupancy map like we explained in Section 3.1.2. The creation of the target maps starts with a zero map. The value of states with high empirical feature count is decreased. We calculate the empirical feature counts as before using:

$$\frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\xi_i}\phi(x) \tag{3.13}$$

The only difference is the feature map. Here, we use a map with dimension $s^2 \times s \times s$ where s is the height and width of the cost grid. The value of an element e^{ijk} in the feature map is one, if $i = j * s + k$. Decreasing the target map on the empirical feature count has the same effect as decreasing the map on the states of the demonstrations. Hence, this step equals the Deep-LEARCH algorithm. While Deep-LEARCH now increases the value of states on the example paths, we increase the value by the expected state frequency. The computation of the target maps is described in Algorithm 3.4. Figure 3.5c shows an example target map.

The CNN returns the functional gradient, which we use to update the costs using exponentiated functional gradient descent. Let c_h^j be the result of the CNN for an occupancy map of the environment j. Furthermore let α_t be the step size computed by $\frac{r}{t+m}$ with r being the learning rate and m a scalar. We get the following gradient decent update rule:

$$c_{t+1}^j = c_t^j \cdot \exp(\alpha_t \cdot c_h^j) \tag{3.14}$$

4 Development of a variation of LEARCH

In the following section we will develop a new algorithm. It is based on the LEARCH algorithm, but includes ideas of maximum entropy. The gradient of LEARCH is created from the demonstrations and example paths. In case of Deep-LEARCH this gives a rather sparse target map for the CNN. In contrast to that maximum entropy obtains a more dense gradient representation by feature matching. We want to exploit maximum entropy by its dense gradient representation to densify the target CNN maps of Deep-LEARCH. In Chapter 5 we will then examine the impact of the matrix density on the learning performance

4.1 Variation of LEARCH as Linear Function Approximation

Fist we will introduce a variant of the LEARCH algorithm which approximates the cost function as a linear combination of features.

$$\mathcal{L}^{Learch}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\xi_i} w^T\cdot\phi(x) - \frac{1}{N}\sum_{i=1}^{N} min_{\xi_i^*}(\sum_{x\in\xi_i^*} w^T\cdot\phi(x) - l_i(x)) \tag{4.1}$$

$$\nabla\mathcal{L}^{Learch}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\xi_i} \phi(x) - \frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\xi_i^*} \phi(x) \tag{4.2}$$

$$\mathcal{L}^{maxEnt}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\xi_i} w^T\cdot\phi(x) - \log\sum_{\xi}\exp(\sum_{x\in\xi_i} w^T\cdot\phi(x)) \tag{4.3}$$

$$\nabla\mathcal{L}^{maxEnt}(w) = \frac{1}{N}\sum_{i=1}^{N}\sum_{x\in\xi_i} \phi(x) - \sum_{x} D_x\phi(x) \tag{4.4}$$

Equations (4.1) to (4.4) show both losses and corresponding gradients of the LEARCH and maximum entropy IRL framework. For better comparison we divided the maximum entropy loss and gradient by N. Both losses are subtractions with the same minuend. They differ only in the subtrahend. Now taking the LEARCH algorithm as basic principle, we want to change it in a way that we optimize the maximum entropy loss. Let $\mathcal{D}$ still be a data set of states and its indication to decrease or increase the costs at this state. Since the minuend is still the same in the loss of the new algorithm, we add the states of the demonstrations with an indication to decrease the costs. For the subtrahend we no longer compute the least cost path ξ^* on the loss-augmented costmap and add it to the data set $\mathcal{D}$. Instead we add every state $x \in \mathcal{X}$ with its corresponding state frequency value D_x to $\mathcal{D}$.

With the omission of the least cost path ξ^* on the loss augmented costmap, the new algorithm does no longer include any notion of margin. The intention now is to incorporate the margin into the expected state frequencies D. Therefore, we compute the expected state frequencies D for

Algorithm 4.1 Variation of the LEARCH algorithm

procedure LEARCH VARIATION($\{\xi_i\}_{i=0}^{N}$)
  $w_0 = (1, ..., 1)^T$
  while $\|w_{t+1} - w_t\| < \epsilon$ **do**
    $\mathcal{D} = \emptyset$
    for all example trajectories ξ_i **do**
      $\mathcal{D} = \mathcal{D} \cup \{(x, \gamma D_x^i) | x \in \mathcal{X}\}$
      $\mathcal{D} = \mathcal{D} \cup \{(x, -1) | x \in \xi_i\}$
    end for
    $w_h = \text{argmin}_{w_h} \frac{1}{2} \sum_{(x,y)\in\mathcal{D}} (y - c(x, w_h))^2 + \frac{\lambda_1}{2} \|w_h - w_t\|^2 + \frac{\lambda_2}{2} w_h$
    $w_{t+1} = w_t \cdot \exp(\alpha_t \cdot w_h)$
  end while
  return w_{t+1}
end procedure

every demonstration on the loss augmented costmap. To differentiate the state frequency D from the loss augmented state frequencies of the trajectory ξ_i, we denote the later by D^i. Hence, this newly developed algorithm includes both ideas of the maximum entropy framework by Ziebart et al. [ZMBD08] and the maximum margin framework by Ratliff et al. [RBZ06]. The final algorithm is presented in Algorithm 4.1. We include the hyperparameter γ as scaling factor of D_x, such that the magnitudes of increase and decrease counterbalance.

Generalization is improved by learning over multiple environments. w_h is calculated for multiple environments and averaged, before updating the weights w_{t+1} in the next gradient descent step. We use the same step size α_t as in LEARCH.

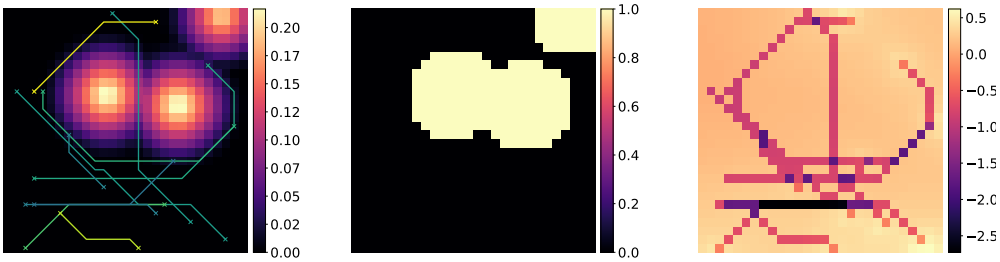
4.2 Variation of Deep-LEARCH

Next, we extend the above algorithm using deep learning. We use exponentiated functional gradient descent to update the cost functional like we did in Section 3.1.2. Let $\alpha_t = \frac{r}{t+m}$ be the step size. We get:

$$c_{t+1}^j = c_t^j \cdot \exp(\alpha_t \cdot c_h^j) \tag{4.5}$$

The target maps of the CNN are created using the idea of the LEARCH variation. Starting with a zero map we decrease the value of the states along the expert demonstrations by one. For every demonstration ξ_i we calculate the corresponding loss augmented expected state frequency D^i and increase the value of the target map by it. Like before, we use γ for scaling. The computation of the target maps is presented in Algorithm 4.2. An example target map is shown in Figure 4.1c.

The approach to Deep-LEARCH, maximum entropy using CNNs and the Deep-LEARCH variation is in all three cases the same. Exponentiated functional gradient descent is used to update the cost function c. The difference in these algorithms is in the computation of the target map of the CNN. Deep-LEARCH uses the demonstrations and the example paths of the loss augmented costmap to compute the gradient map while the other two use the demonstrations and the expected



(a) Ground truth costmap with ten demonstrations

(b) Occupancy map

(c) Gradient map

Figure 4.1: Ground truth costmap and corresponding input and target maps for the CNN in the Deep-LEARCH variation using ten demonstrations

Algorithm 4.2 Update of a target map of the CNN in the Deep-LEARCH variation

procedure UPDATETARGETMAP(c, $\{\xi_i\}_{i=0}^{N}$)
  $\mathcal{D} = (0, ..., 0)$
  for all example trajectories ξ_i **do**
    for all $x \in \mathcal{X}$ **do**
      $\mathcal{D}(x) = \mathcal{D}(x) + \gamma D_x^i$
    end for
    for all $x \in \xi_i$ **do**
      $\mathcal{D}(x) = \mathcal{D}(x) - 1$
    end for
  end for
  return $\mathcal{D}$
end procedure

state frequencies D. The difference in the target maps of maximum entropy using CNNs and the Deep-LEARCH variant is the loss augmentation. In the later the expected state frequency is computed for every demonstration separately. To do this the loss augmented current costmap is used. Contrary to that, maximum entropy computes the expected state frequency over all demonstrations on the current costmap without loss augmentation.

5 Experiment

In the evaluation we compare the IRL algorithms introduced in Chapter 3 and 4. We collect different kind of measurements over a varying number of demonstrations and environments used during training.

5.1 Experiment Setup

We evaluate the algorithms on sample environments. We use two different environment setups. In the first environment setup we use explicit features for the generation of the costmap. Hence, it is used in the evaluation of the algorithms assuming a linear cost function. In contrast the second environment setup is not build by the linear combination of features. It is used in the algorithms using deep learning. Further implementation details are described in Appendix B.

5.1.1 Radial Basis Function Environment

Each environment is a squared grid with 28 x 28 states. The costmaps are computed by the weighted linear combination of features. One radial basis function corresponds to one feature ϕ_i. We use 16 radial basis functions for one environment. The radial basis functions are multiplied by the weight w. Hence, the costmap is build by the inference of multiple gaussian radial basis functions.

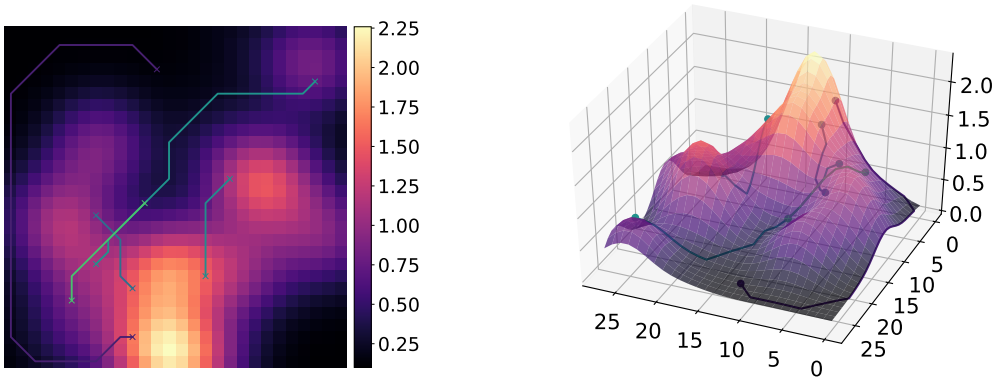


Figure 5.1: Costmap created from four radial basis functions, and five demonstrations

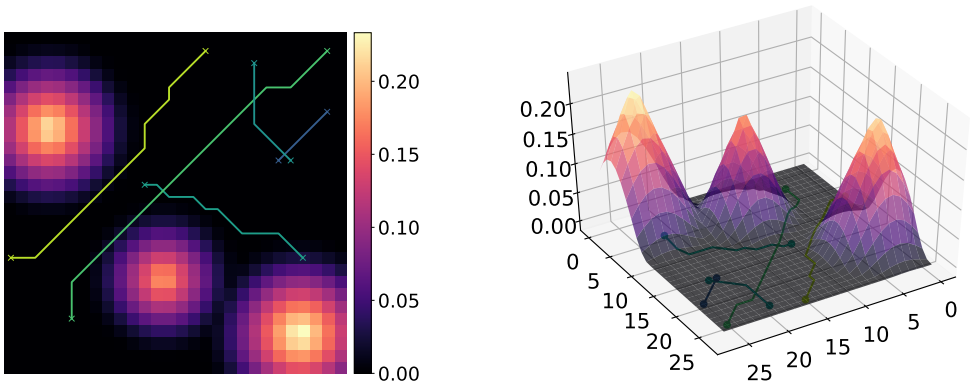


Figure 5.2: Costmap created from a signed distance function of three circular obstacles, and five demonstrations

(5.1) gaussian radial basis function: $\phi_i(x) = \exp(\frac{|x - c_i|^2}{\sigma^2})$

(5.2) tensor of radial basis functions: $\phi(x) = (\phi_1(x), ..., \phi_{16}(x))^T$

(5.3) costmap function: $c(x, w) = w^T \phi(x)$

The centers c_i of the radial basis functions are sampled randomly over the grid. Let σ be the variance of the gaussian function and fixed. Furthermore, let w be random, but fixed for a set of environments. Hence, a set of sample environments only differ in the position of the centers of the radial basis functions, but not in the value of the weight w. Thus, the introduced algorithms can learn one single weight vector over multiple environments.

The expert demonstrations are computed on the above costmap by randomly choosing a start and target state for each demonstration. The least cost path between these states on the ground truth costmap is the demonstration. We used the Dijkstra algorithm [Dij59] to compute this path. Figure 5.1 shows an example costmap created from four radial basis functions and five demonstrations.

5.1.2 Signed Distance Function Environment

The second environment setup is used in all deep learning algorithms. Each environment is again a squared grid with 28 x 28 states. The costmap is build by the signed distance function of multiple obstacles. There are at most three circular obstacles in each environment. The center and the radius of each obstacle is chosen randomly.

$$\text{(5.4)}\quad \text{costmap function: } c(x) = \begin{cases} -d(x) + \frac{1}{2}\epsilon, & \text{if } d(x) < 0 \\ \frac{1}{2\epsilon}(d(x) - \epsilon)^2, & \text{if } 0 \leq d(x) \leq \epsilon \\ 0, & \text{otherwise} \end{cases}$$

We compute the costmap from the signed distance function of the obstacles using the cost function proposed by Ratliff et al. [RZBS09]. Let d(x) be the signed distance function, i.e. the distance from state x to the boundary of the nearest obstacle. d(x) is negative inside the obstacle and positive outside the obstacles. If x is on the boundary of an obstacle it is 0. Let ϵ be the distance from an obstacle at which the obstacle cost zeroes out. The cost function is described in Equation (5.4).

The expert demonstrations are computed with Dijkstra [Dij59] between two randomly sampled start and target states. Due to runtime reasons we did not optimize the trajectories like Toussaint [Tou17] describes. An example of a signed distance function environment with five demonstrations is shown in Figure 5.2.

5.2 Measurements

In the analysis we calculate different measurements to evaluate and compare the performance of the three introduced algorithms. The following notation assumes the cost function to be linear in features. Hence, we compute the loss of the weights w. In the functional case the loss is computed of the cost function $c(x)$. This case follows accordingly.

The LEARCH loss is the sum of the difference between the costs of the demonstrations ξ and example paths ξ^* on the ground truth costmap normalized by the number of demonstrations. We use the ground truth weights $\tilde{w}$ instead of the learned weights to compute this loss. The demonstration is the least cost path on the ground truth costmap. Its cost is smaller than the costs of the example path. Hence, the loss is negative. Thus, we want to maximize it.

$$\text{(5.5)}\quad \text{LEARCH loss: } \mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\Big(\sum_{x\in\xi_i} c(x,\tilde{w}) - \sum_{x\in\xi_i^*} c(x,\tilde{w})\Big)$$

The maximum entropy loss is the log likelihood of the demonstration ξ given the learned weight vector w. $\log Z(w)$ is independent of the expert demonstrations. Hence, $N \log Z(w)$ only increases the convergence behavior. Therefore, we neglect it in the calculation of the maximum entropy loss.

$$\begin{aligned} \text{maximum entropy loss: } \mathcal{L}(w) &= \sum_{i=1}^{N} \log P(\xi_i|w) \\ &= \sum_{i=1}^{N}\sum_{x\in\xi_i} w^T \cdot \phi(x) - N \log Z(w) \\ &\approx \sum_{i=1}^{N}\sum_{x\in\xi_i} w^T \cdot \phi(x) \end{aligned} \tag{5.6}$$

The euclidean distance transform loss quantifies the distance between the demonstrations and the example paths. We normalize by the number of demonstrations and the length of the example paths. Hence, it expresses the average distance of one state in a example path to the demonstration. $|\xi^*|$ denotes the length of the example path ξ^*.

(5.7) euclidean distance transform loss: $\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|\xi_i^*|}\sum_{x\in\xi_i^*} edt(x,\xi_i)$

Furthermore, we calculate the negative log likelihood using the log_loss function of the scikit-learn library [PVG+11].

(5.8) negative log likelihood loss: $\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N} nll(\xi_i,\xi_i^*)$

The costmap difference is the sum of the absolute difference of the ground truth and the learned costmap. Before calculating the costmap difference loss, we normalize both costmaps using $c(x,w) = \frac{c(x,w)-\min_{x'\in\mathcal{X}} c(x',w)}{\sum_{x\in\mathcal{X}}(c(x,w)-\min_{x'\in\mathcal{X}} c(x',w))}$, i.e. we center the map at zero and normalize the resulting vector. $\tilde{w}$ denotes the ground truth weights in contrast to the learned weights w. The costmap difference expresses the average difference of one state in the costmap.

(5.9) costmap difference loss: $\mathcal{L}(w) = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}} |c(x,\tilde{w}) - c(x,w)|$

5.3 Evaluation of the Inverse Reinforcement Learning Algorithms

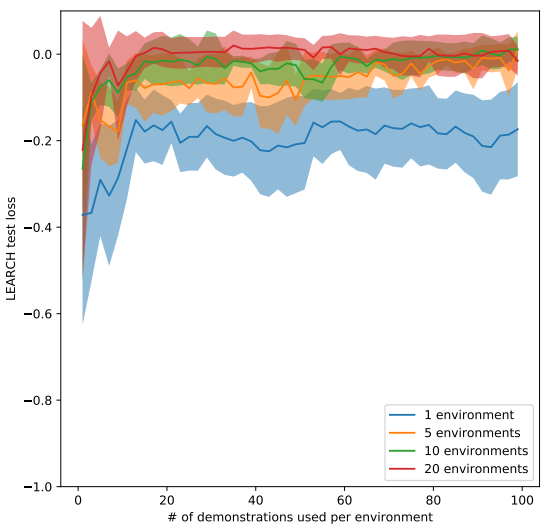
In the following evaluation we compare the performance of the IRL algorithms of Chapter 3 and 4. For this purpose we compute the different loss measurements introduced in Section 5.2 over different amounts of demonstrations. In doing so, we average over multiple environments. In addition we also take the standard deviation of the loss over different environments into account. Furthermore, we compare the three algorithms using linear function approximation against two baseline solutions. Let 'random' denote the solution which always returns a random weight vector with elements in $[0.0, 1.0)$ and 'one vector' denote the solution which always returns $(1, ..., 1)^T$, i.e. a vector created only by ones. These vectors are used as weight vectors in the cost function. A selection of the results is presented in the following subsections. A representation of all results can be found in Appendix A.

5.3.1 Evaluation of the Linear Models

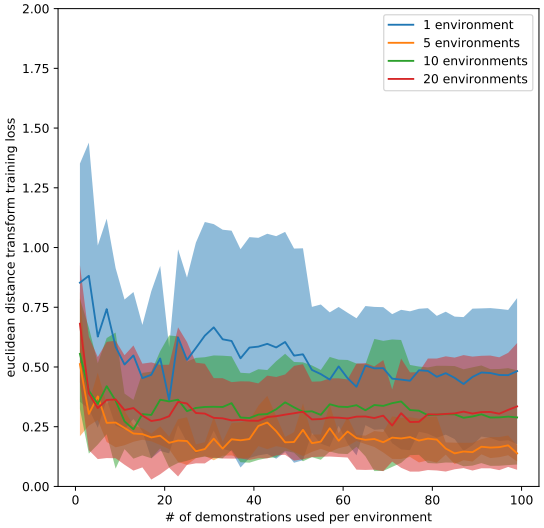
First, we look at the case of approximating the cost function linearly. Here we look at the loss over the range of one to 100 demonstrations per environment. Only every second step in the range of one to 100 is computed. Furthermore, we evaluate the algorithms by learning over one, five, ten and twenty environments. In case the algorithm only uses one environment in the learning process, the result is averaged over ten runs. The training loss is not computed over the number of demonstrations used in the algorithms, but always over 20 demonstrations. So even if the algorithm takes into account only five demonstrations per environment in the training process, the training loss is computed over 20 demonstrations. We also compute the test loss always over 20 demonstrations. Some demonstrations are easier to fit than others, e. g., shorter demonstrations are usually easier to fit than longer demonstrations. Using only the demonstrations of the training process would give inaccurate loss measurements, if only a few demonstrations are used. In case of learning from one demonstration, this single demonstration might not represent the total set of demonstrations well. Hence, the loss is influenced by the properties of this demonstration. To avoid this effect we always use a fixed number of demonstrations to compute the training and test loss. In addition the test loss is averaged over five different environments, except in the case of the baseline methods 'random' and 'one vector'. Here we use twenty environments to get a representative solution, since the standard deviation is especially high in case of the 'random' solution.

Figure 5.3a shows the LEARCH test loss for the LEARCH algorithm. The loss is computed for a varying number of environments which have been used in the training process to average the gradient. The LEARCH test loss improves with increasing amounts of environments. This loss is already quite low with five environments. Hence, averaging the gradient over more than five environments, archives only slightly improved results. The LEARCH test loss is relatively high for less than 15 demonstrations. After this point the loss flattens out. For one environment the curve stagnates after 15 environments. These results can be verified by the euclidean distance transform test loss presented in Figure 5.3c. For more than 15 demonstrations the results using five, ten and twenty environments are rather similar. The improvement over increasing demonstrations per environment is little for more than 15 demonstrations in comparison to the improvement using fewer than 15 demonstrations per environment. We can even see a slight worsening for 25 - 45 demonstrations using 20 environments. This is confirmed by the euclidean distance transform training loss in Figure 5.3b. The results using only five environments are better than using ten or twenty environments. Hence, LEARCH can not adjust well to the influences of multiple different environments. The costmap difference is illustrated in Figure 5.3d. While the costmap difference loss for five environments converges to zero, both the costmap difference for ten and twenty environments remain constant for an increasing number of demonstrations. However, the loss decreases significantly with the number of environments. Contrary to the examined loss measurements above the standard deviation intervals do not overlap anymore. Since the costmaps are normalized before the loss computation, the standard deviation is per se not as large as in case of the other losses. We can conclude that the algorithm needs a reasonable amount of samples for learning. However, more demonstrations or environments do not always achieve better results. There is a point at which the loss stagnates with increasing number of samples.

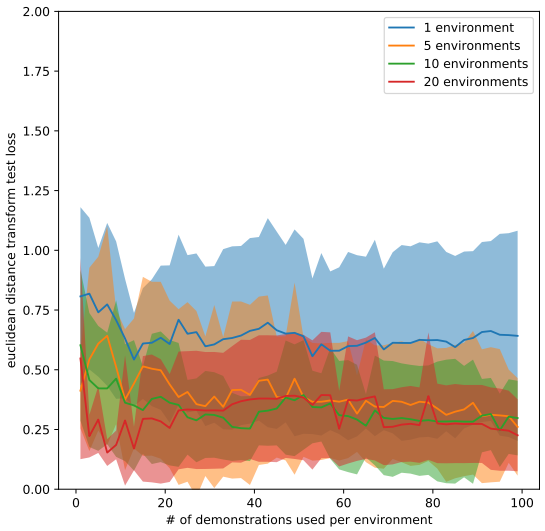
The loss of the maximum entropy algorithm is illustrated in Figure 5.4. Even through the maximum entropy loss does not noticeably improve with an increasing number of environments, the euclidean distance transform, the neg-log likelihood and the costmap difference does. The largest improvement is between one and five environments. Like the LEARCH algorithm we get a satiation for 20



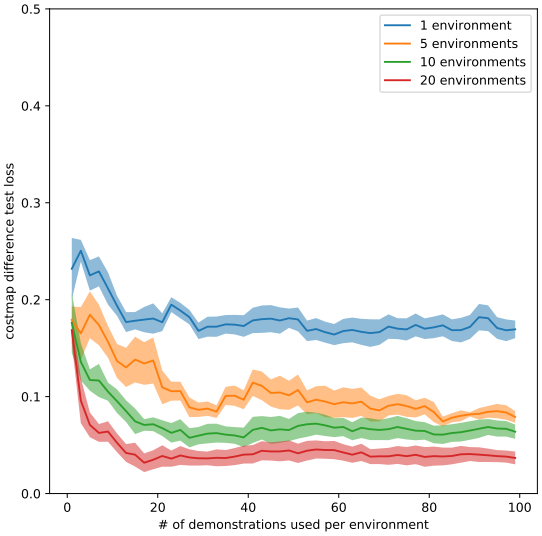
(a) LEARCH test loss



(b) Euclidean distance transform training loss

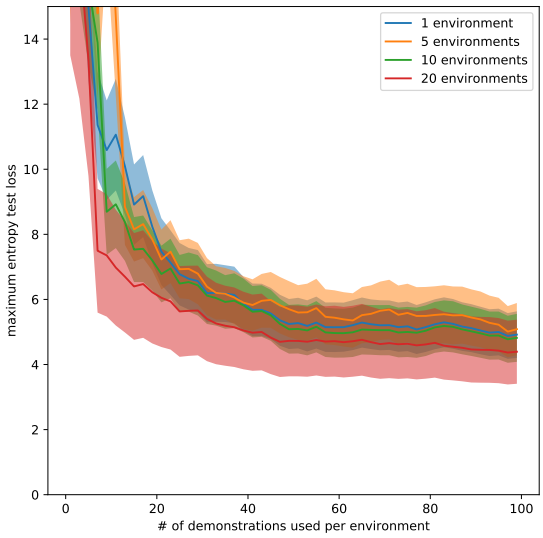


(c) Euclidean distance transform test loss

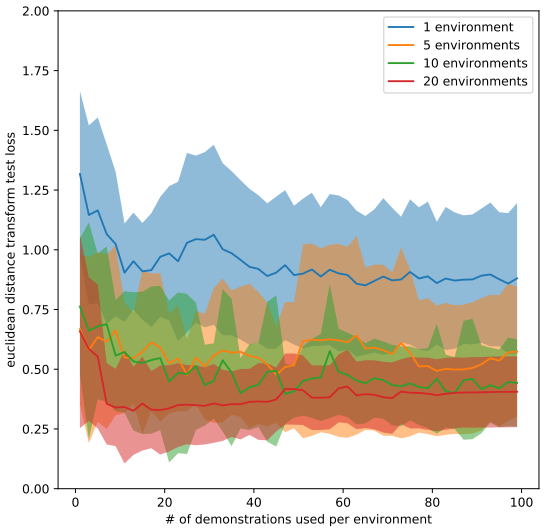


(d) Costmap difference test loss

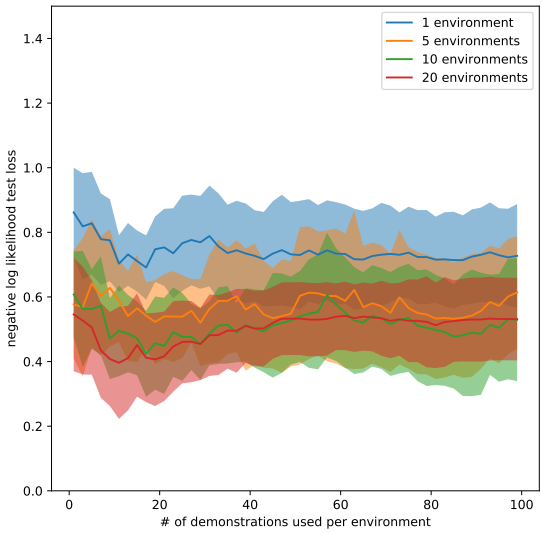
Figure 5.3: Loss over one to 100 demonstrations of the LEARCH algorithm using a varying number of environments



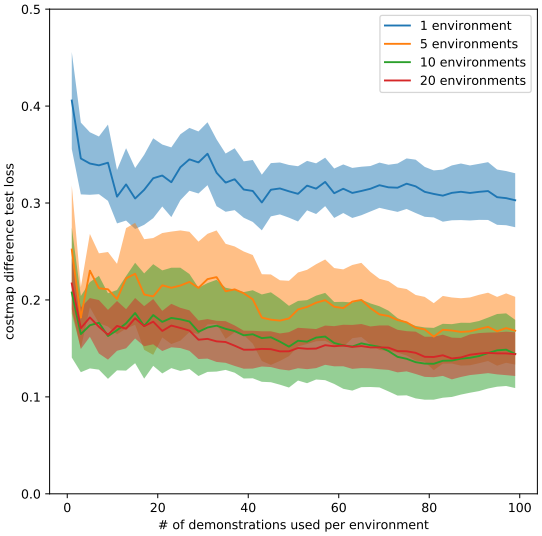
(a) Maximum entropy test loss



(b) Euclidean distance transform test loss

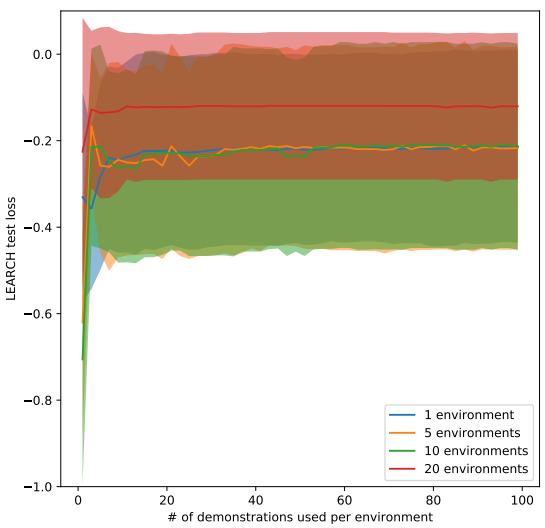


(c) Neg-log-likelihood test loss

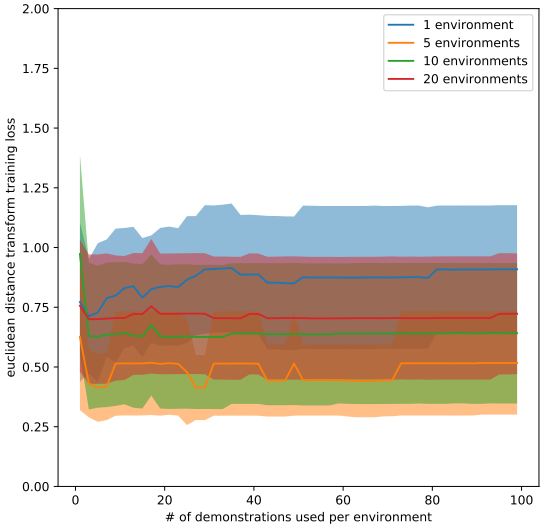


(d) Costmap difference test loss

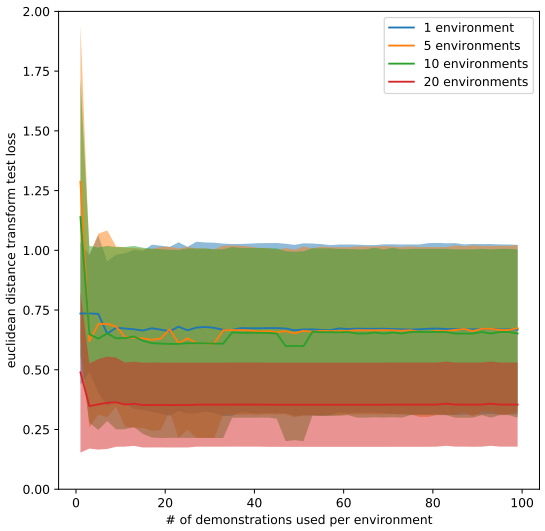
Figure 5.4: Loss over one to 100 demonstrations of the maximum entropy algorithm over a varying number of environments



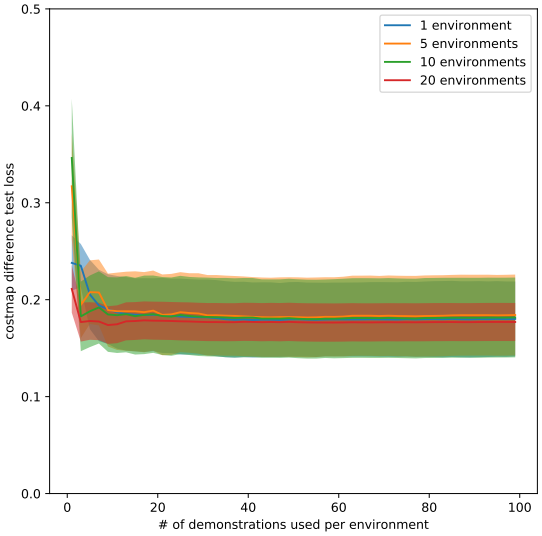
(a) LEARCH test loss



(b) Euclidean distance transform training loss



(c) Euclidean distance transform test loss

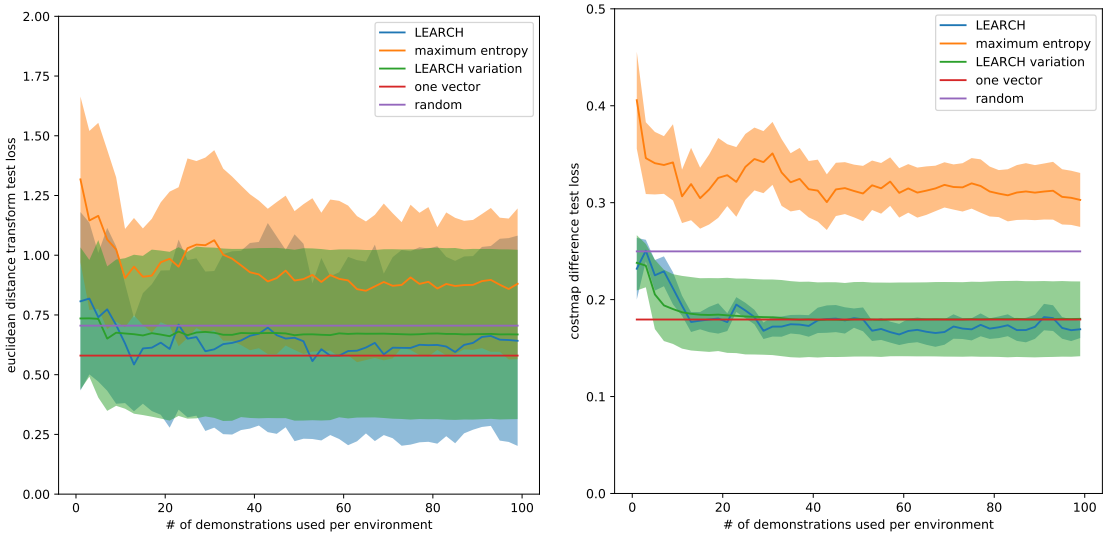


(d) Costmap difference test loss

Figure 5.5: Loss over one to 100 demonstrations of the LEARCH variation over a varying number of environments

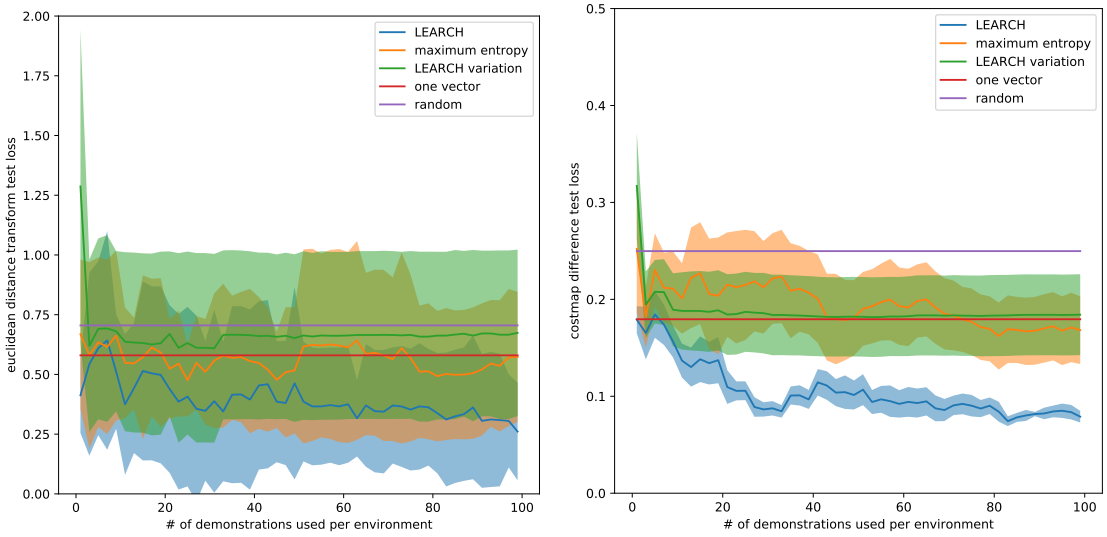
environments. Looking at the loss in the course of increasing demonstrations we see the maximum entropy loss exponentially declining. The other loss measurements decrease only slightly with increasing number of demonstrations. Hence, maximum entropy IRL improves with increasing number of samples until the point of satiation is reached.

The results of the LEARCH variation in Figure 5.5 stand out, since the loss is, unlike before, almost constant in every loss measurement. By contrast, before, where there is at first an improvement in the loss for an increasing amount of environments, which later bottoms out, it is vice versa in case of the LEARCH variant. We see no enhancement for five or ten environments, but with twenty environments. The LEARCH variation shows a good ability to generalize with increased number



(a) Euclidean distance transform test loss **(b)** Costmap difference test loss

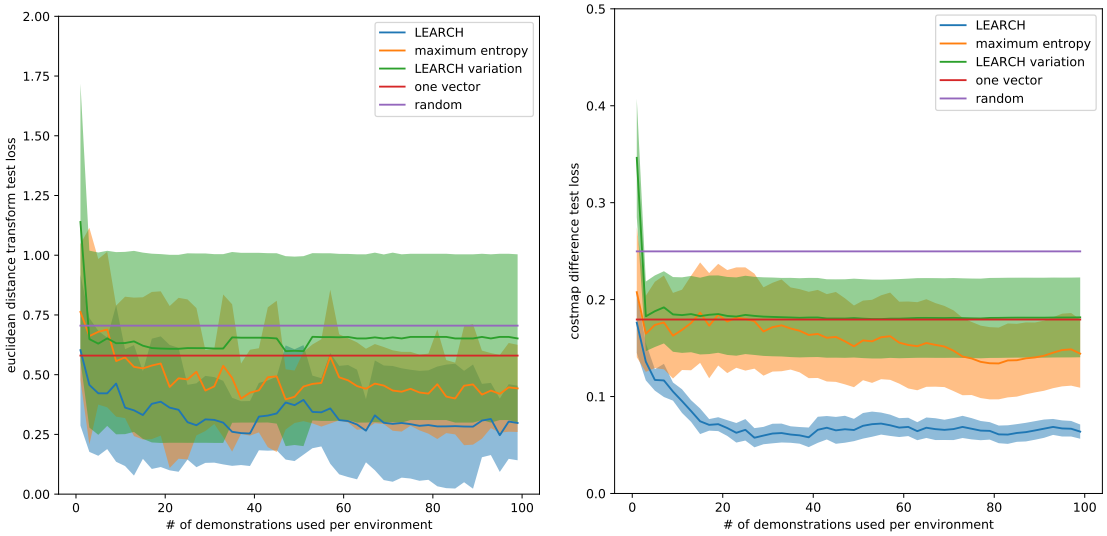
Figure 5.6: Loss over one to 100 demonstrations and one environment



(a) Euclidean distance transform test loss **(b)** Costmap difference test loss

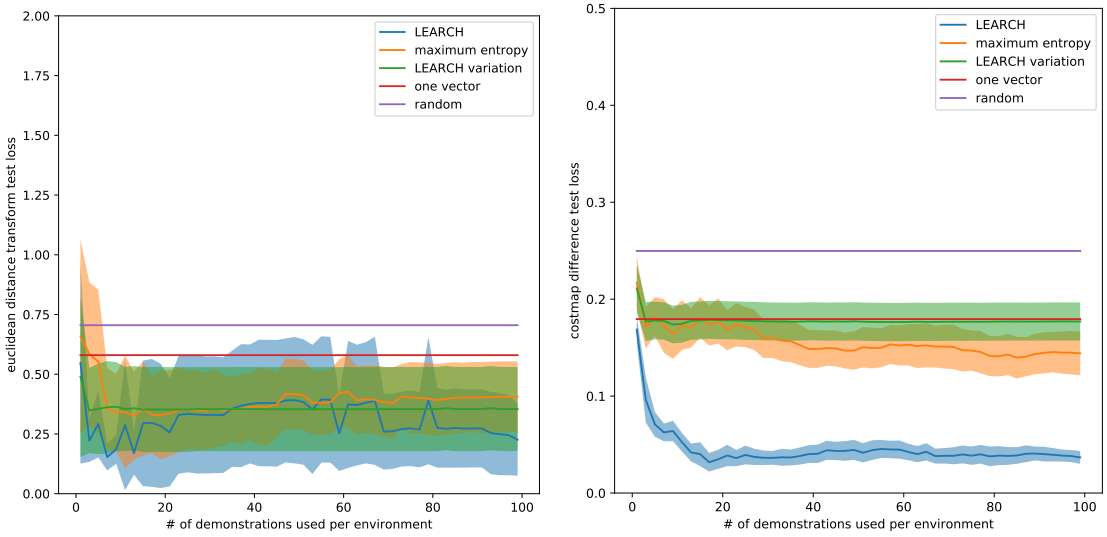
Figure 5.7: Loss over one to 100 demonstrations and five environments

of environments. While the euclidean distance transform training loss is for 20 environments higher than for five or 10 environments, the euclidean distance transform test loss is lower for 20 environments. In the costmap difference test loss the LEARCH variation performs for all number of environments equal. The only difference is in the standard deviation. It is for 20 environments the smallest. The LEARCH variant shows for 20 environments the lowest loss.



(a) Euclidean distance transform test loss **(b)** Costmap difference test loss

Figure 5.8: Loss over one to 100 demonstrations and ten environments



(a) Euclidean distance transform test loss **(b)** Costmap difference test loss

Figure 5.9: Loss over one to 100 demonstrations and twenty environments

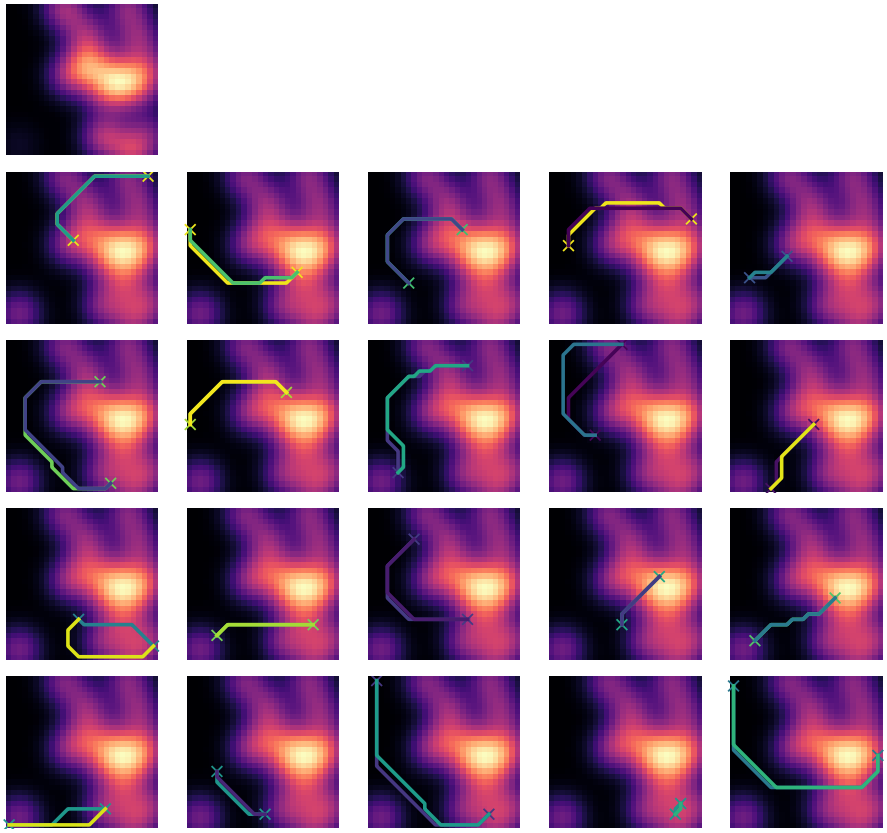


Figure 5.10: Costmap with 20 demonstrations and corresponding example paths using a one vector as weight. The first costmap in the upper left corner represents the ground truth costmap

Now we compare the performance of the algorithms among each other and the baseline solutions dependent on the number of environments. For clarity reasons we omit the standard deviation of the baseline solutions in the following graphs. The graphs are presented in Figures 5.6 to 5.9.

The algorithms perform with one exception always better than the 'random' baseline solution. The loss of maximum entropy is above the 'random' solution as shown in Figure 5.6. 'random' chooses the elements of the weight vector in the interval $[0.0, 1.0)$. In the same range are also the ground truth values. Maximum entropy on the other hand can achieve weights higher than 1. Hence, its performance can be worse. LEARCH and the LEARCH variation are always around or below the 'one vector' solution. Since both algorithms start with the 'one vector' solution and improve it gradually, this behavior is expected.

With an increasing number of environments maximum entropy catches up on LEARCH and achieves the same performance for the euclidean distance transform loss. Looking at the costmap difference loss, the LEARCH algorithm outperforms all other solutions.

In our experiment LEARCH achieves overall the best performance. It shows lower sample complexity than maximum entropy. The LEARCH variation can only improve the 'one-vector' baseline solution using a large number of environments.

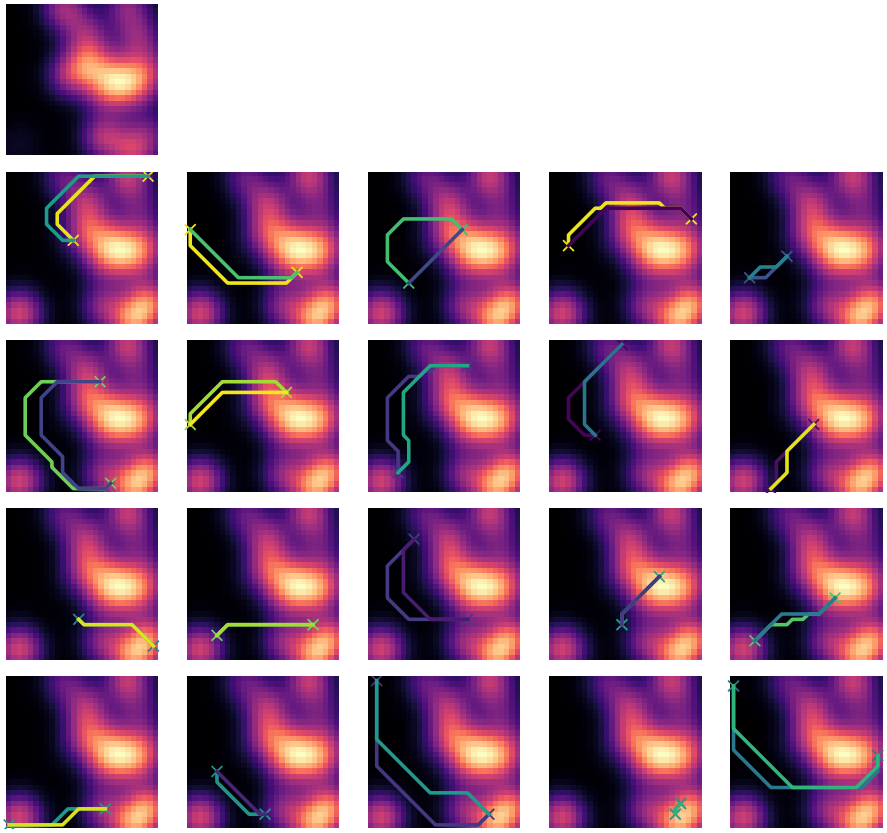


Figure 5.11: Costmap with 20 demonstrations and corresponding example paths using a random vector as weight. The first costmap in the upper left corner represents the ground truth costmap

Looking at the one vector solution, we see that it achieves already a good performance. With an euclidean distance transform loss of 0.58, one state of the example path is in average less than one state from the ground truth demonstration away. This implies that nearly half of the paths segments already match. The example paths in Figure 5.10 verify this insight. The figure shows twenty demonstrations and their corresponding example paths for a costmap using the 'one vector' solution. In only three cases the example path deviates far from the demonstration. In the most cases the example paths are already nearby or even matching the demonstration. Since the 'one vector' solution performs already very well, there is unfortunately not that much room for improvement. In comparison, the example paths of the 'random' solution, shown in Figure 5.11, deviate in more cases further from the demonstrations.

Figure 5.12 show the same demonstrations with example paths for a costmap learned by LEARCH over five environments with 20 demonstrations each. The algorithms accomplish to match the paths quite well except for the first path. The example paths learned by maximum entropy deviate in more cases from the demonstration. The paths and the costmap are illustrated in Figure 5.13. Figure 5.14 shows the results using the LEARCH variant. The example paths resemble the paths using the 'one vector' solution in Figure 5.10.

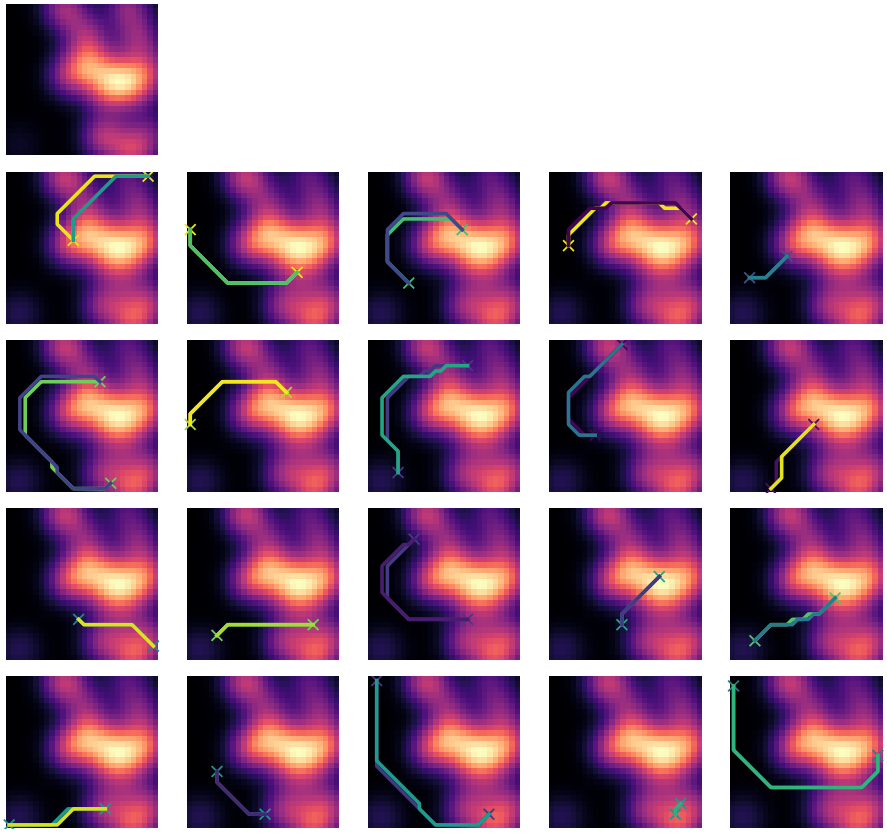


Figure 5.12: Costmap with 20 demonstrations and corresponding example paths learned with the LEARCH algorithm on five environments with each 20 demonstrations. The first costmap in the upper left corner represents the ground truth costmap

5.3.2 Evaluation of the Deep-Learning Algorithms

In this section we evaluate Deep-LEARCH, the variant of Deep-LEARCH and maximum entropy extended by CNNs. We look at the results using one to 25 demonstrations. Like before, we only compute the results for every second number in this range. We evaluate each algorithm using 200, 400 and 800 training environments in the CNN. As before, the loss is computed over a fixed set of demonstrations. Here, we use ten environments with 20 demonstrations each. The CNN performs in each iteration of gradient descent 100 training steps.

The results of the Deep-LEARCH algorithm are presented in Figure 5.15. The LEARCH loss decreases with increasing number of demonstrations and training environments. This applies also to the standard deviation of the LEARCH loss. The standard deviation is for less than ten environments rather large, but it decreases with increasing number of demonstrations. Overall is the loss, especially the euclidean distance transform loss and the neg-log likelihood loss, very close for the different number of training environments. The costmap difference loss stands out, since the loss improves with less sample environments.

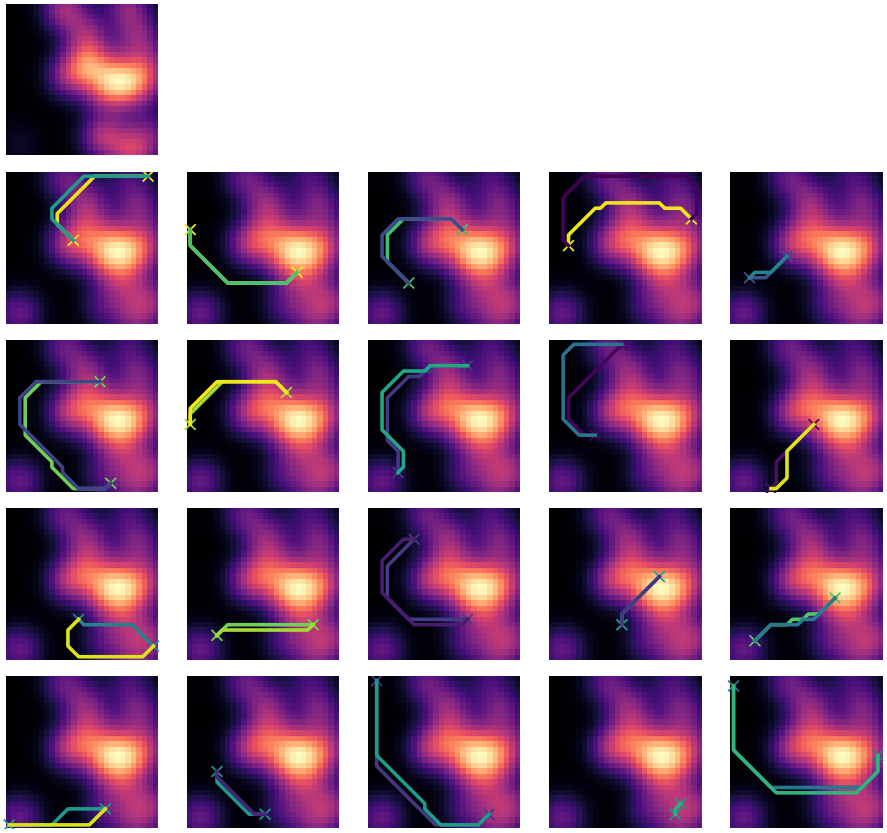


Figure 5.13: Costmap with 20 demonstrations and corresponding example paths learned with maximum entropy on five environments with each 20 demonstrations. The first costmap in the upper left corner represents the ground truth costmap

The Deep-LEARCH variation performs similar to Deep-LEARCH. The results are presented in Figure 5.16. The LEARCH loss decreases with increasing number of demonstrations and environments. The results using 200 and 400 environments are closer to each other than the results using 400 and 800 environments. Hence, the Deep-LEARCH variation can improve more with more sample data.

In the maximum entropy algorithm of Section 3.2.2 we can see the same characteristics as in the Deep-LEARCH variation. Figure 5.17b shows no improvement of the euclidean distance transform loss between 200 and 400 environments. Through, there is an improvement for 800 environments. Not only the average LEARCH loss, but also the standard deviation decreases with an increasing number of samples. The standard deviation is conspicuously large in Figure 5.17a.

Figure 5.18 shows the comparison of the three deep-learning algorithms for 200 environments. Deep-LEARCH outperforms the other two algorithms in all loss measurements. Even through Deep-LEARCH and the Deep-LEARCH variant show the same performance in the LEARCH training loss, Deep-LEARCH outperforms its variant using the test data set. Hence, Deep-LEARCH shows a larger ability to generalize. The standard deviation of the LEARCH test loss decreases from maximum entropy over the Deep-LEARCH variation to Deep-LEARCH. Thus, the precision increases in this order of the algorithms. As we already mentioned before the Deep-LEARCH

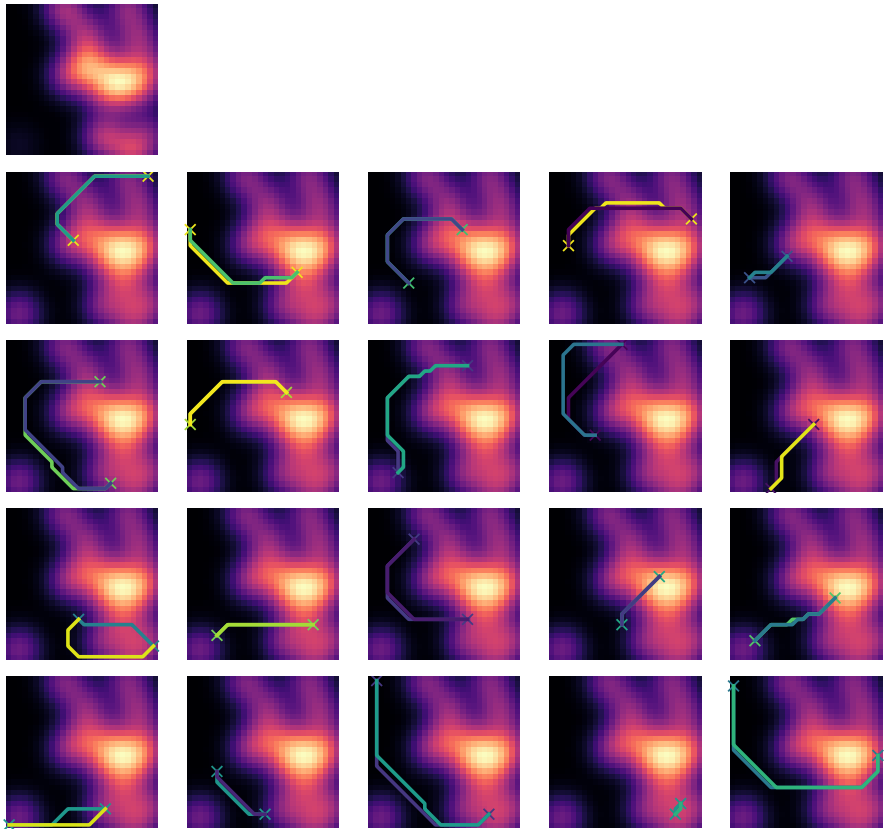


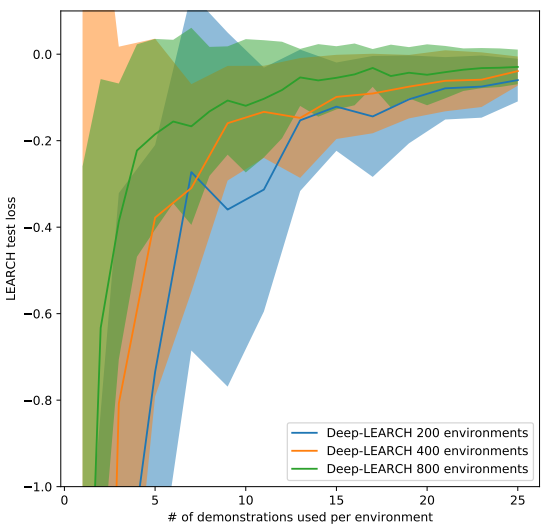
Figure 5.14: Costmap with 20 demonstrations and corresponding example paths learned with the LEARCH variation on five environments with each 20 demonstrations. The first costmap in the upper left corner represents the ground truth costmap

variation and maximum entropy using CNNs only differ in the loss augmentation of the costmap used in the calculation of the expected state frequencies. Since the Deep-LEARCH variation outperforms maximum entropy using CNNs, the loss augmentation does indeed improve the algorithm.

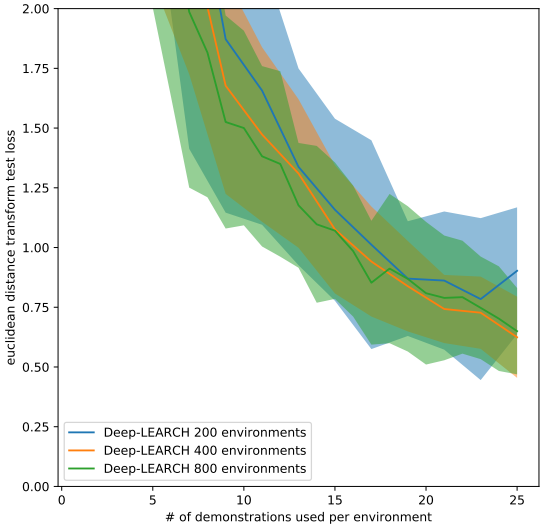
The results using 800 environments are presented in Figure 5.19. Again, Deep-LEARCH performs best, followed by the Deep-LEARCH variation. The difference between the performance of the algorithms is now with 800 environments not as large anymore as for 200 environments.

We can conclude that the results improve for all the three algorithms with increasing number of samples. Deep-LEARCH outperforms the other two algorithms, although the distance to the performance of the other two algorithms decreases with increasing number of samples. Hence, Deep-LEARCH shows the smallest sample complexity. The increased density of the target map of the CNN in the Deep-LEARCH variant does not compensate for the higher sample complexity maximum entropy needs.

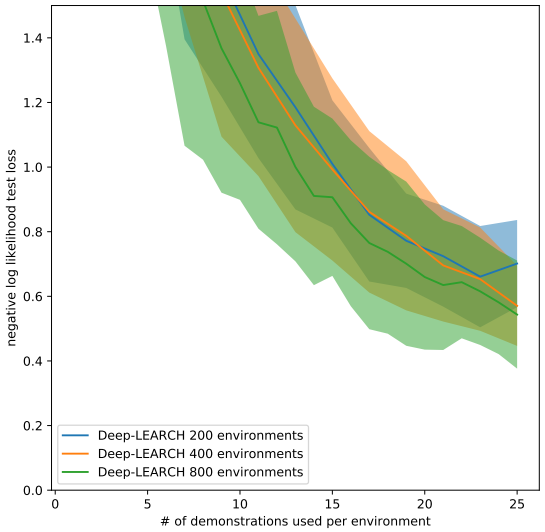
Figures 5.20 to 5.21 show examples of the learned costmaps over the range from one to 23 demonstrations. With the increasing amount of demonstrations the costmaps get more precise. The costmap of the Deep-LEARCH variation is lighter, i.e. the costs are higher. This might be due to the increased density of the target CNN map. The expected state frequencies increase the gradient map in all states. It is striking that the border has relatively high costs. This effect can also be



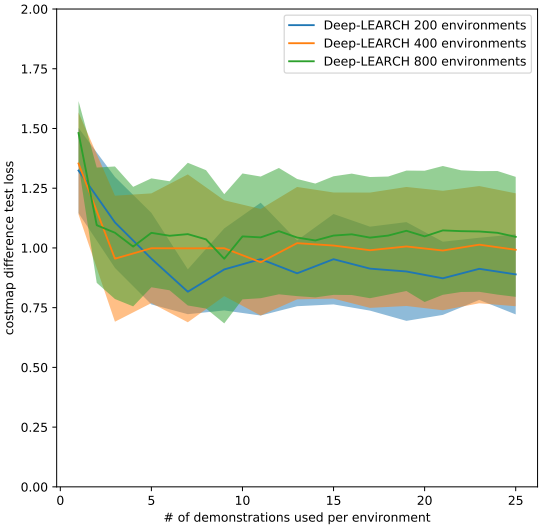
(a) LEARCH test loss



(b) Euclidean distance transform test loss



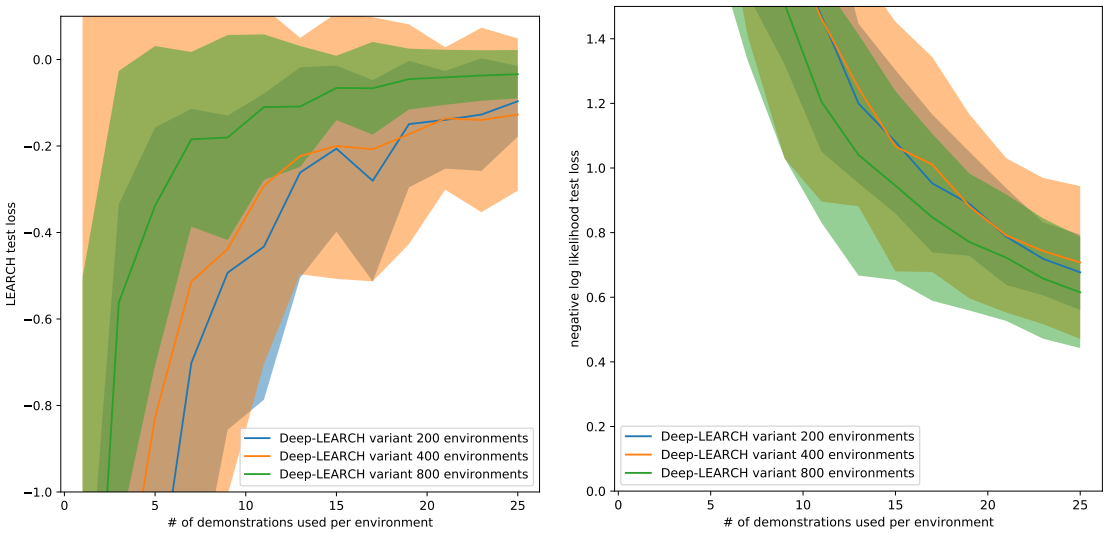
(c) Neg-log likelihood test loss



(d) Costmap difference test loss

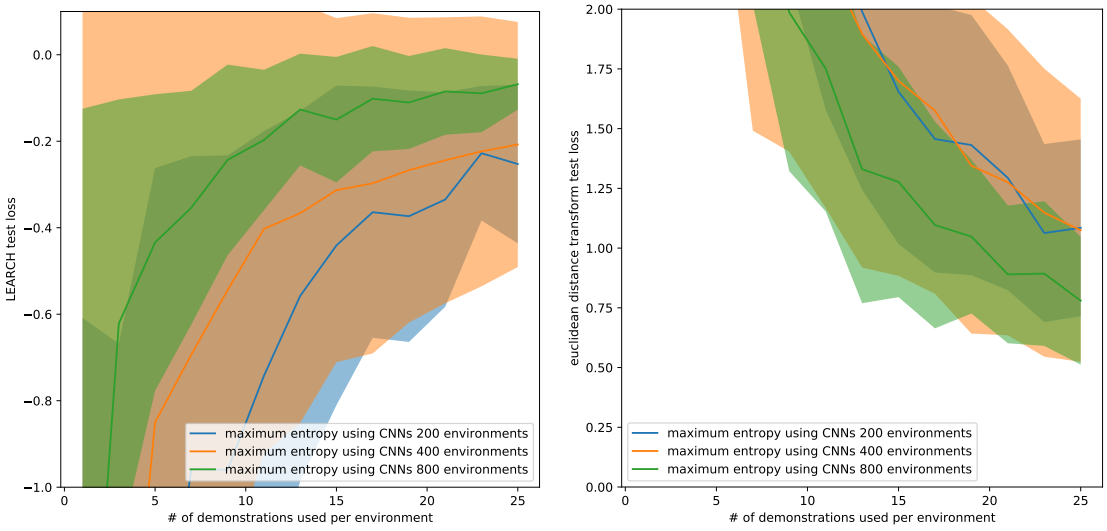
Figure 5.15: Loss over 1 to 25 demonstrations using Deep-LEARCH

seen in Deep-LEARCH, but not in the same extent. We assume that this is due to the architecture of the CNN. The improvement over an increasing number of demonstrations can be verified in the Figures 5.20 to 5.21. The gap between the demonstration and example path gets gradually smaller. Hence, the euclidean distance transform loss decreases. Overall the resulting costmaps are very imprecise. Deep-LEARCH recognizes only one obstacle and not three. The learning of the costmaps can be refined by increasing the number of CNN steps and the number of gradient descent iterations.



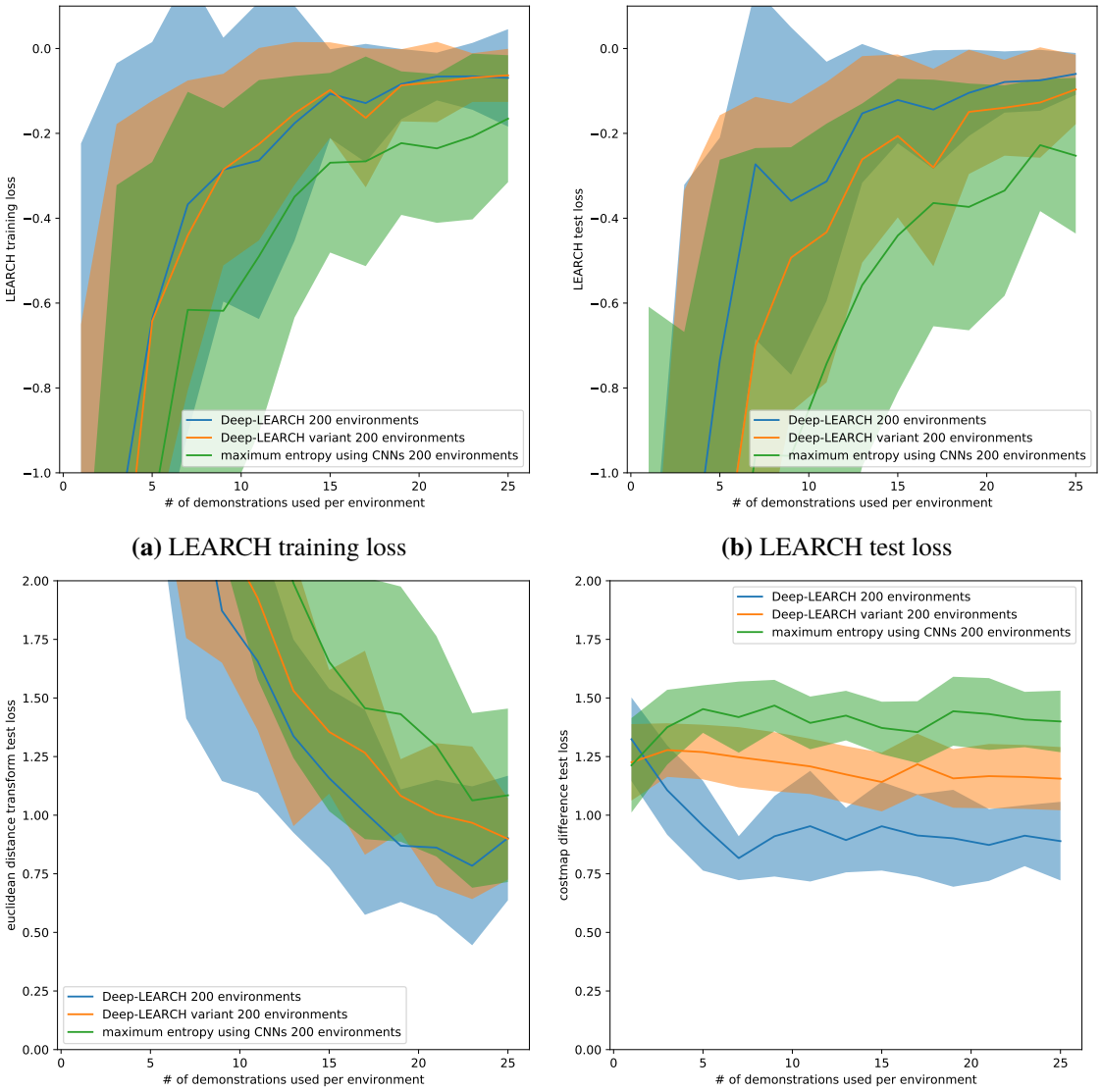
(a) LEARCH test loss **(b)** Neg-log likelihood test loss

Figure 5.16: Loss over 1 to 25 demonstrations using the Deep-LEARCH variation



(a) LEARCH test loss **(b)** Euclidean distance transform test loss

Figure 5.17: Loss over 1 to 25 demonstrations using maximum entropy extended by CNNs



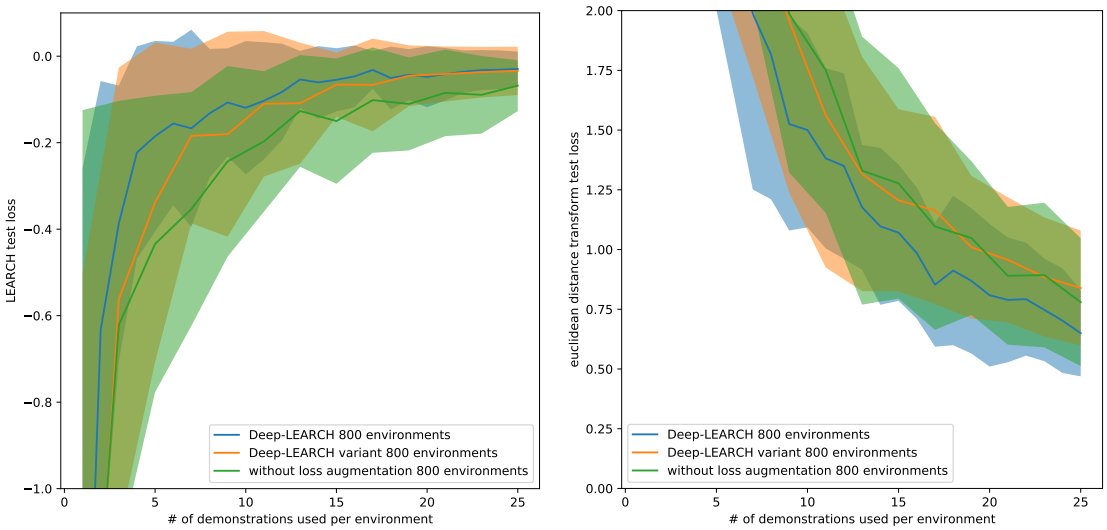
(a) LEARCH training loss

(b) LEARCH test loss

(c) Euclidean distance transform test loss

(d) Costmap difference test loss

Figure 5.18: Loss over 1 to 25 demonstrations for 200 training environments



(a) LEARCH test loss **(b)** Euclidean distance transform test loss

Figure 5.19: Loss over 1 to 25 demonstrations for 800 training environments

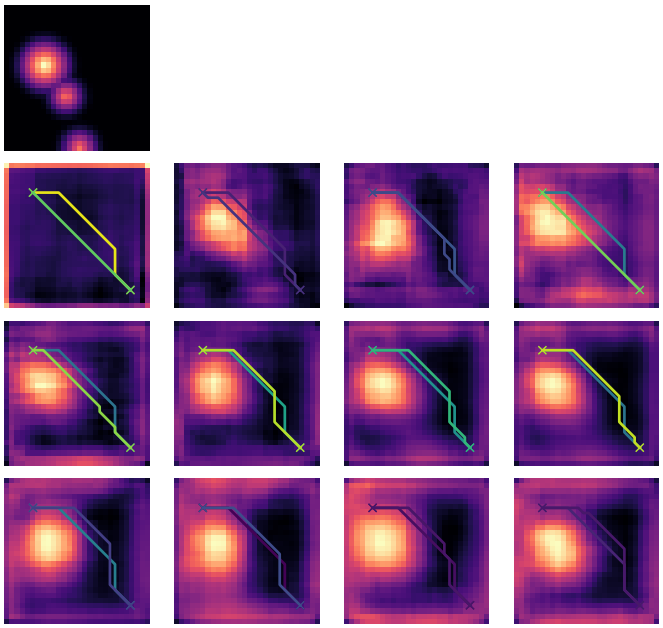


Figure 5.20: Costmap and one demonstration using Deep-LEARCH and 800 training environments. The first costmap in the upper left corner represents the ground truth costmap. The costmaps below show the learned costmaps using 1, 3, 5, ... demonstrations

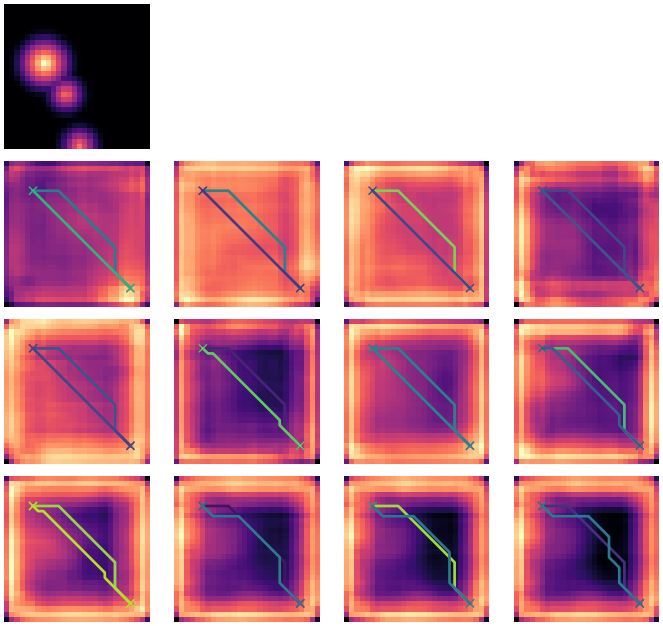


Figure 5.21: Costmap and one demonstration using the Deep-LEARCH variation and 800 training environments. The first costmap in the upper left corner represents the ground truth costmap. The costmaps below show the learned costmaps using 1, 3, 5, ... demonstrations

6 Conclusion and Outlook

Based on the LEARCH and the maximum entropy IRL learning algorithm we developed a new algorithm, a variation of LEARCH. In all three algorithms the cost function is approximated as a linear combination of features. We evaluated all of the algorithms and two constant baseline solutions over a varying number of demonstrations and environments for different kind of losses. The LEARCH algorithm outperforms all the other algorithms and shows the lowest sample complexity. The LEARCH variant does not improve with increasing number of demonstrations. It only improves for a large number of environments. Thus it shows a high sample complexity. The constant 'one vector' baseline solution already achieves a relatively good performance. Many of the example paths already match the demonstration, if a one vector is used as a weight vector. Hence, the results of LEARCH, maximum entropy and the LEARCH variation should be verified by a different environment setup.

We extended the three algorithms with CNNs. This facilitates a non-linear approximation of the cost function. The target maps in the Deep-LEARCH variation have an increased density in comparison to the target maps in the Deep-LEARCH algorithm, since the expected state frequency is used in the computation of the target map. The Deep-LEARCH variation and maximum entropy using CNNs only differ in the loss augmentation of the costmap before computing the expected state frequency. As before we evaluate the algorithms over a varying number of demonstrations and environments for different kind of losses. Again, LEARCH shows the best performance among the evaluated algorithms. It is followed by the LEARCH variation. The difference between the performance of the algorithms decreases with increasing number of samples. Increasing the density of the CNN target maps did not enhance the performance of Deep-LEARCH. Hence, the sample complexity of LEARCH could not be reduced. Future work can evaluate Deep-LEARCH and the Deep-LEARCH variation for more than 25 demonstrations. Since the results did get close to each other for an increasing amount of demonstrations, the Deep-LEARCH variation might be able to outperform Deep-LEARCH for higher amounts of samples. Due to limited computation capacity this was not possible to analyze in this thesis. Furthermore, instead of applying the algorithms to simulated environments, it would be interesting to evaluate their performance on a data set, collected from real life data, like the Stanford Drone Dataset [RSAS16].

Bibliography

[Cha17] M. Chablani. "Autoencoders - Introduction and Implementation in TF." June 26, 2017 (cit. on p. 71).

[Dij59] E. W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390 (cit. on pp. 34, 35, 71).

[FLA16] C. Finn, S. Levine, P. Abbeel. "Guided Cost Learning: Deep Inverse Optimal Control via Policy Optimization". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by M. Balcan, K. Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 49–58. URL: http://proceedings.mlr.press/v48/finn16.html (cit. on p. 17).

[HE16] J. Ho, S. Ermon. "Generative Adversarial Imitation Learning". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, R. Garnett. Curran Associates, Inc., 2016, pp. 4565–4573. URL: http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf (cit. on p. 17).

[MBK+16] J. Mainprice, A. Byravan, D. Kappler, D. Fox, S. Schaal, N. Ratliff. "Functional manifold projections in Deep-LEARCH". In: Dec. 2016 (cit. on pp. 17, 22, 71).

[NR00] A. Y. Ng, S. J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. Ed. by P. Langley. Morgan Kaufmann, 2000, pp. 663–670 (cit. on p. 19).

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830 (cit. on p. 36).

[RBZ06] N. D. Ratliff, J. A. Bagnell, M. Zinkevich. "Maximum margin planning". In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. Ed. by W. W. Cohen, A. W. Moore. Vol. 148. ACM International Conference Proceeding Series. ACM, 2006, pp. 729–736. DOI: 10.1145/1143844.1143936. URL: https://doi.org/10.1145/1143844.1143936 (cit. on pp. 17, 19, 20, 30).

[RSAS16] A. Robicquet, A. Sadeghian, A. Alahi, S. Savarese. "Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes". In: *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*. Ed. by B. Leibe, J. Matas, N. Sebe, M. Welling. Vol. 9912. Lecture Notes in Computer Science. Springer, 2016, pp. 549–565. DOI: 10.1007/978-3-319-46484-8_33. URL: https://doi.org/10.1007/978-3-319-46484-8%5C_33 (cit. on p. 53).

[RSB09] N. D. Ratliff, D. Silver, J. A. Bagnell. "Learning to search: Functional gradient techniques for imitation learning". In: *Auton. Robots* 27.1 (2009), pp. 25–53. DOI: 10.1007/s10514-009-9121-3. URL: https://doi.org/10.1007/s10514-009-9121-3 (cit. on pp. 17, 19, 20).

[RZBS09] N. D. Ratliff, M. Zucker, J. A. Bagnell, S. S. Srinivasa. "CHOMP: Gradient optimization techniques for efficient motion planning". In: *2009 IEEE International Conference on Robotics and Automation, ICRA 2009, Kobe, Japan, May 12-17, 2009*. IEEE, 2009, pp. 489–494. DOI: 10.1109/ROBOT.2009.5152817. URL: https://doi.org/10.1109/ROBOT.2009.5152817 (cit. on p. 35).

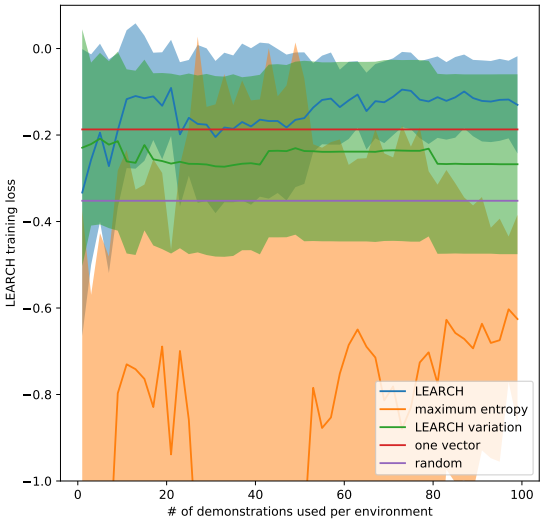
[Tou17] M. Toussaint. "A tutorial on Newton methods for constrained trajectory optimization and relations to SLAM, Gaussian Process smoothing, optimal control, and probabilistic inference". In: *Geometric and numerical foundations of movements*. Springer, 2017, pp. 361–392 (cit. on pp. 35, 71).

[WOP16] M. Wulfmeier, P. Ondruska, I. Posner. *Maximum Entropy Deep Inverse Reinforcement Learning*. 2016. arXiv: 1507.04888 [cs.LG] (cit. on p. 17).

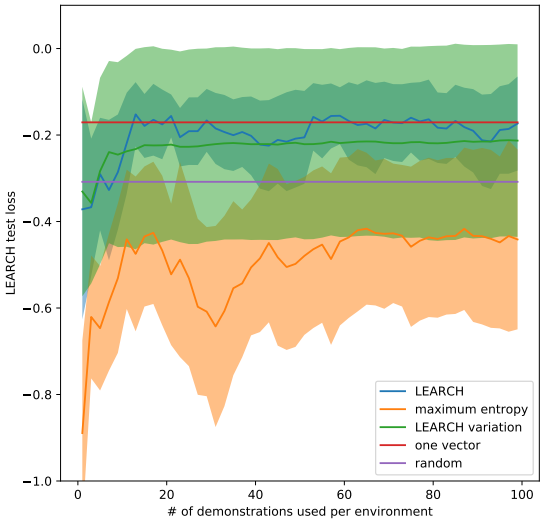
[ZMBD08] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey. "Maximum Entropy Inverse Reinforcement Learning". In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. Ed. by D. Fox, C. P. Gomes. AAAI Press, 2008, pp. 1433–1438. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-227.php (cit. on pp. 17, 23, 24, 30).

All links were last followed on November 25, 2020.

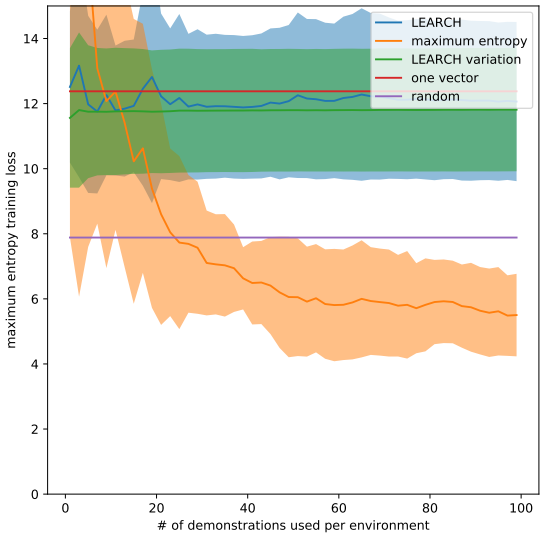
A Graphs of the Evaluation



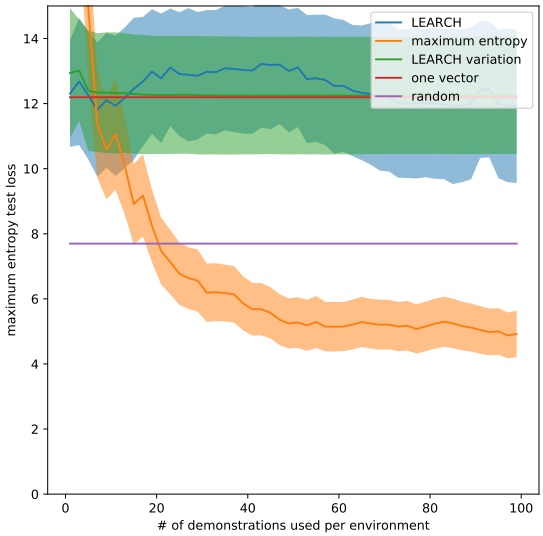
(a) LEARCH training loss



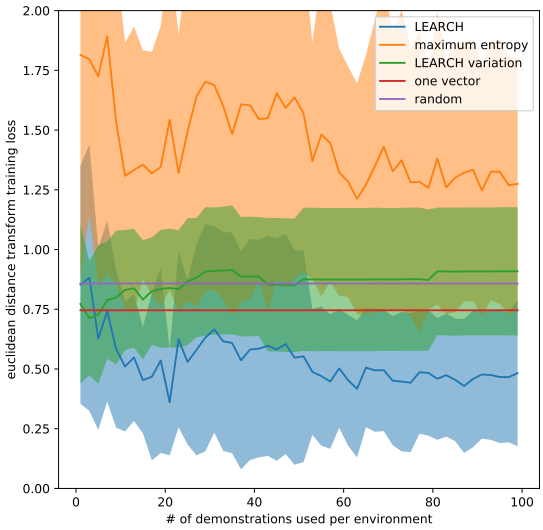
(b) LEARCH test loss



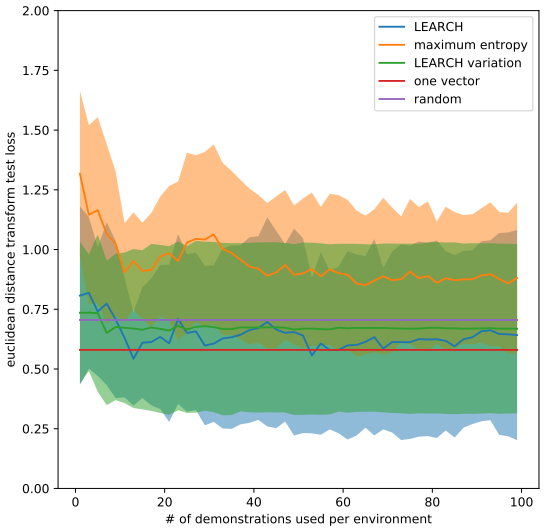
(c) Maximum entropy training loss



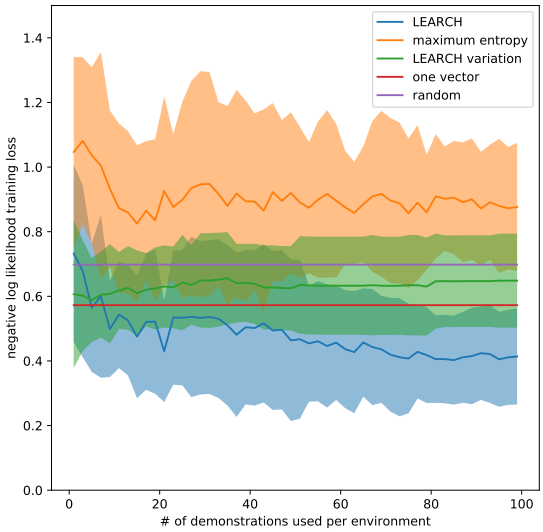
(d) Maximum entropy test loss



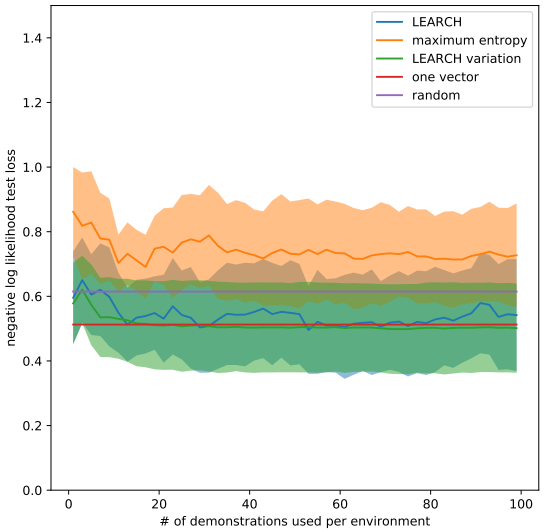
(e) Euclidean distance transform training loss



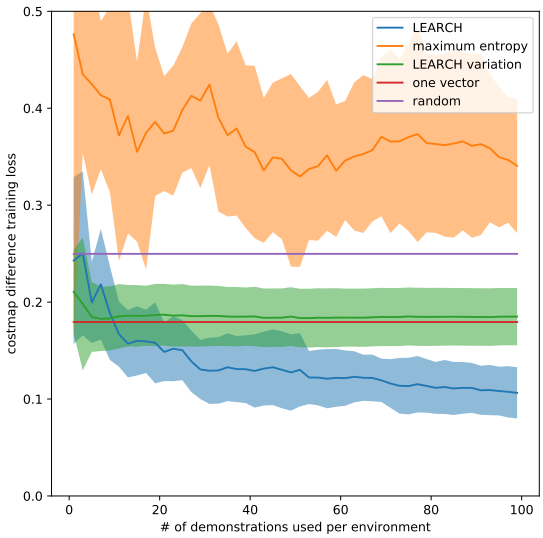
(f) Euclidean distance transform test loss



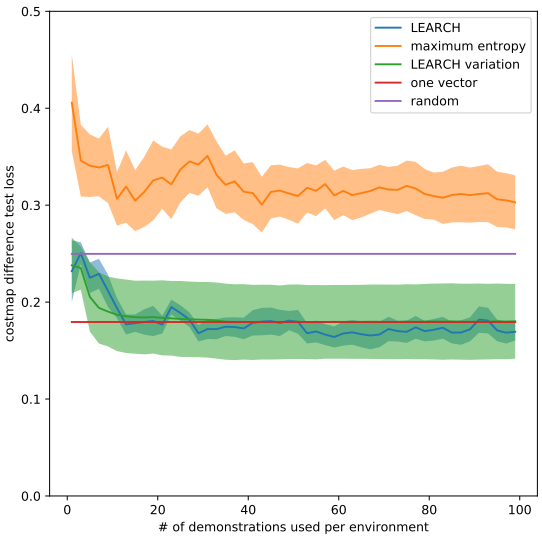
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss

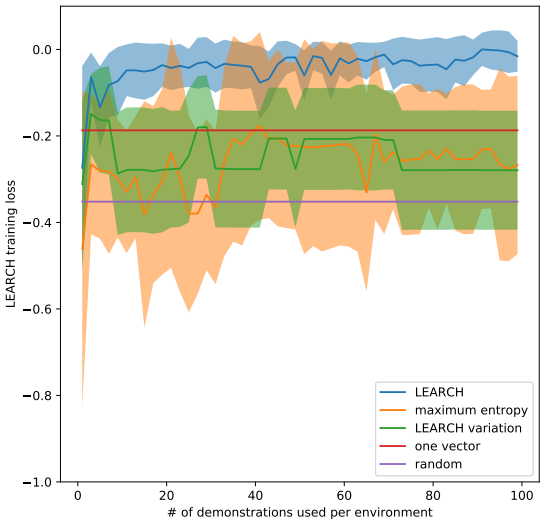


(i) Costmap difference training loss

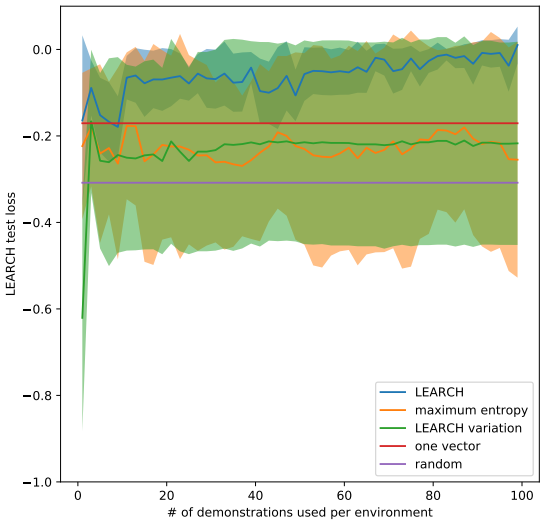


(j) Costmap difference test loss

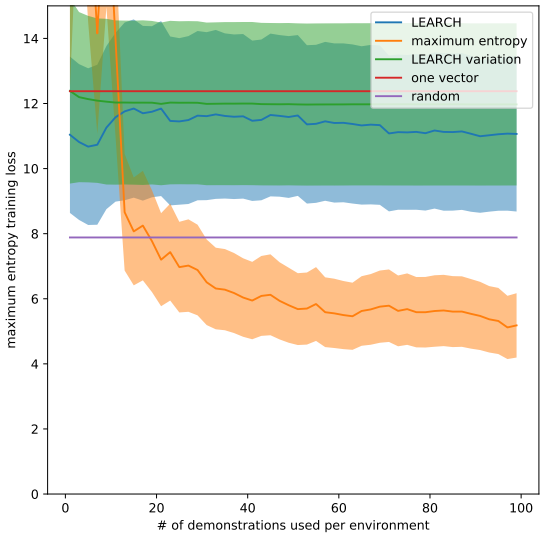
Figure A.1: Training and validation losses for one to 100 demonstrations. Learning is done on one environment. The result is averaged over ten environments



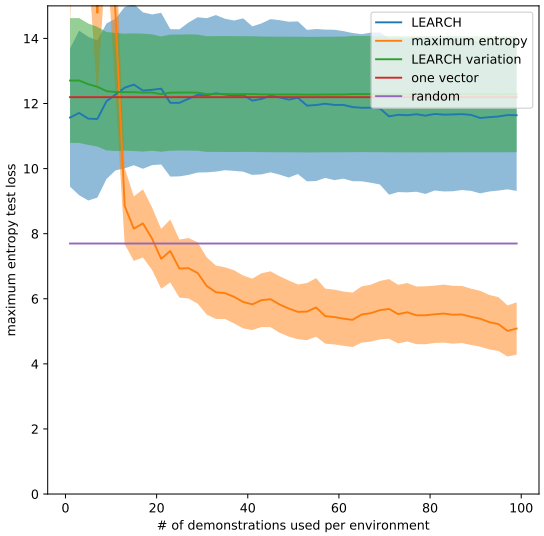
(a) LEARCH training loss



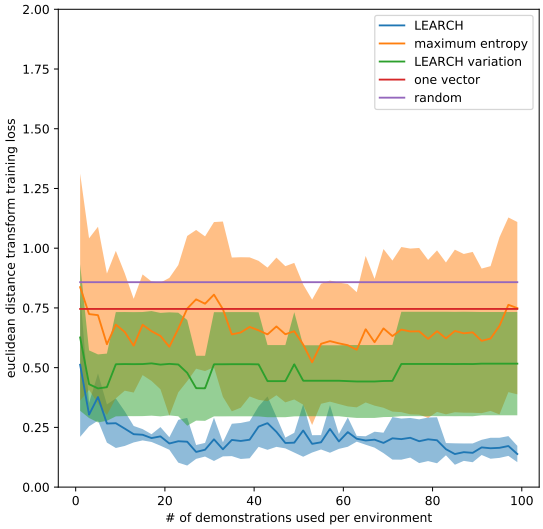
(b) LEARCH test loss



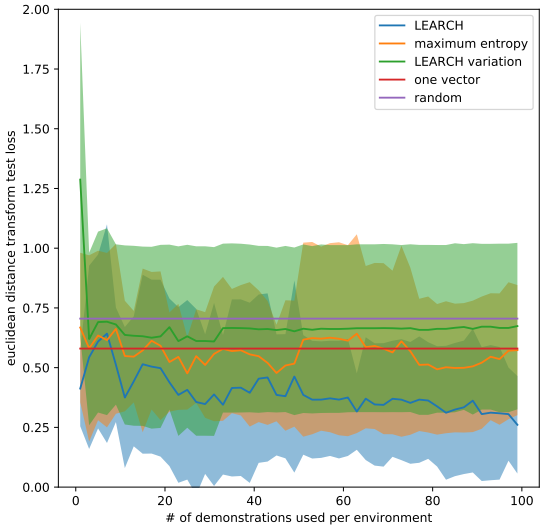
(c) Maximum entropy training loss



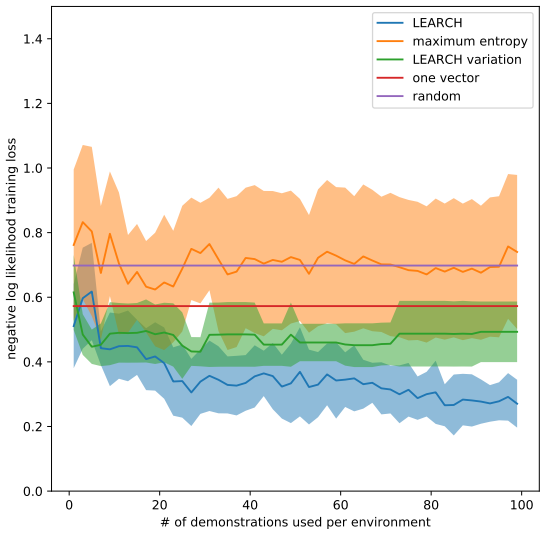
(d) Maximum entropy test loss



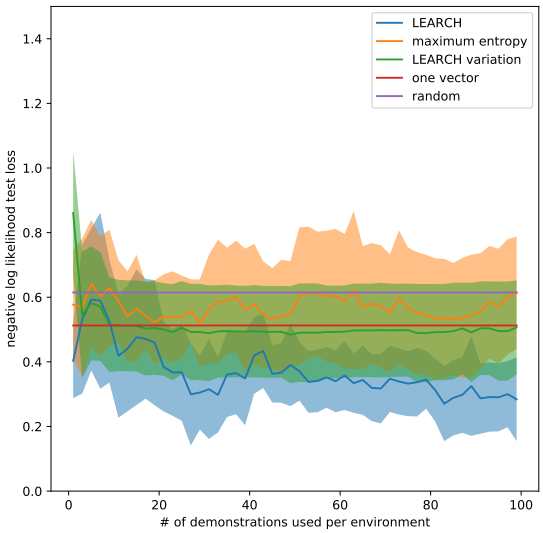
(e) Euclidean distance transform training loss



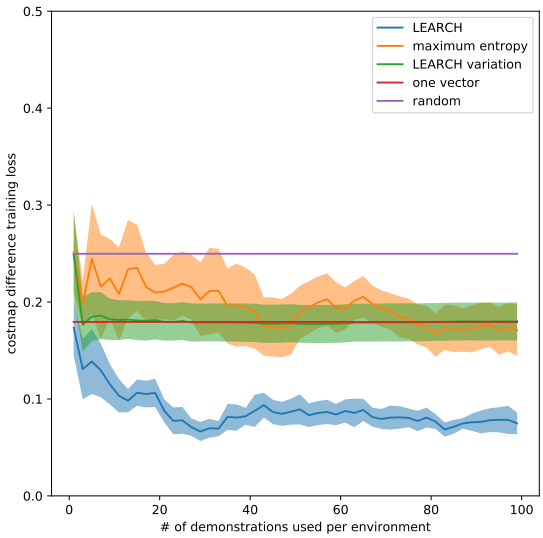
(f) Euclidean distance transform test loss



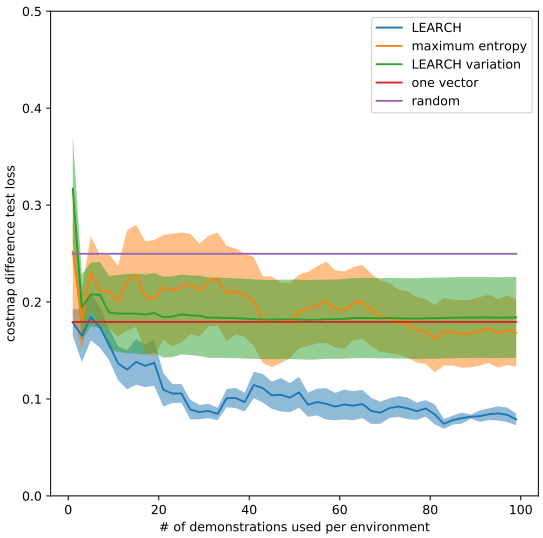
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss

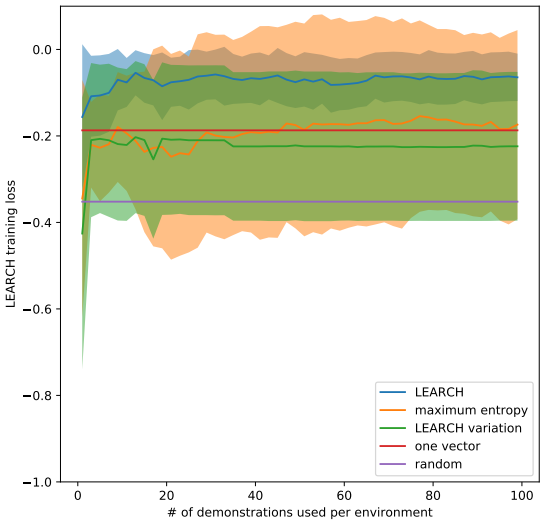


(i) Costmap difference training loss

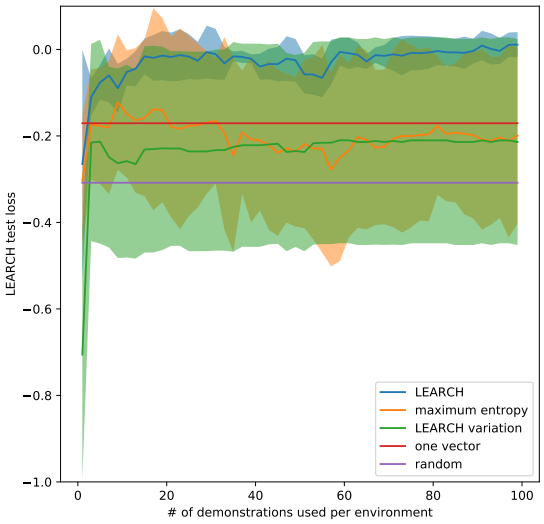


(j) Costmap difference test loss

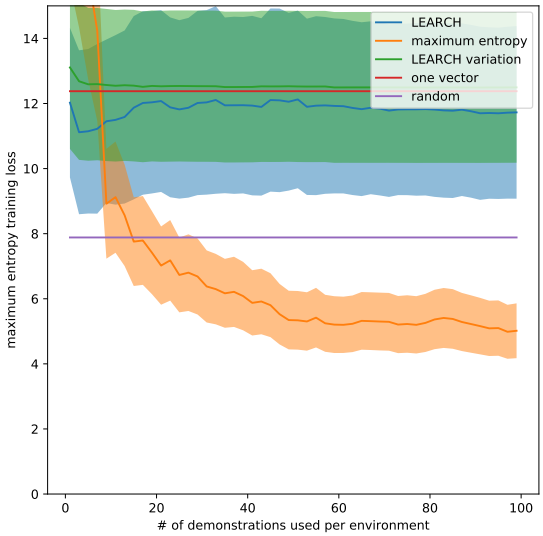
Figure A.2: Training and validation losses for one to 100 demonstrations. Learning is done on five environments simultaneously



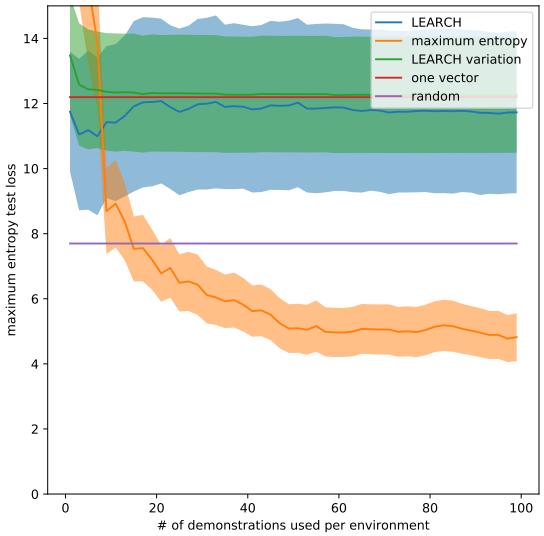
(a) LEARCH training loss



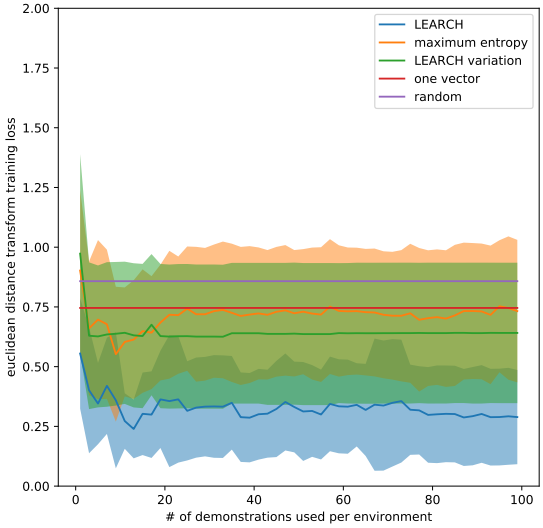
(b) LEARCH test loss



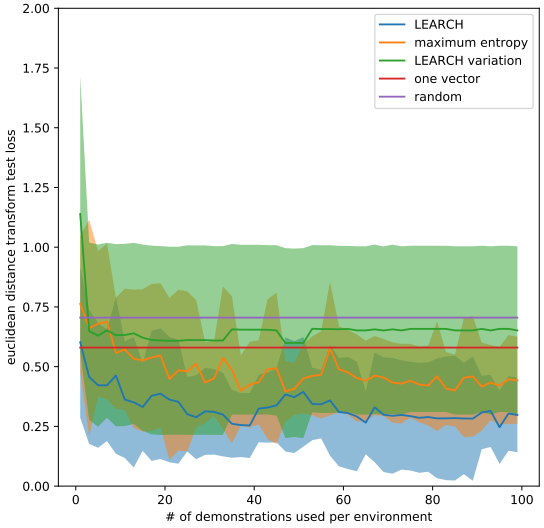
(c) Maximum entropy training loss



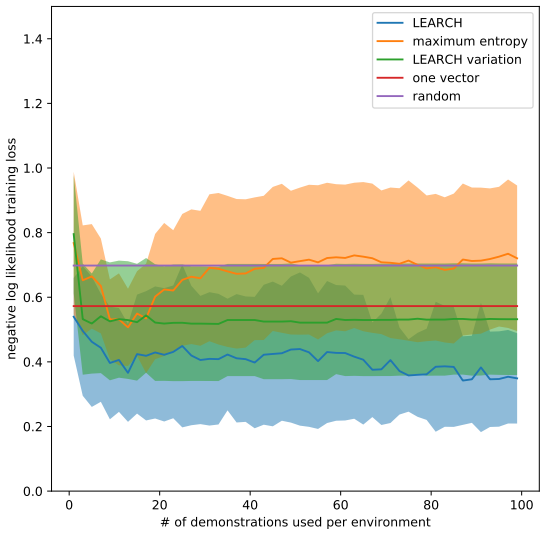
(d) Maximum entropy test loss



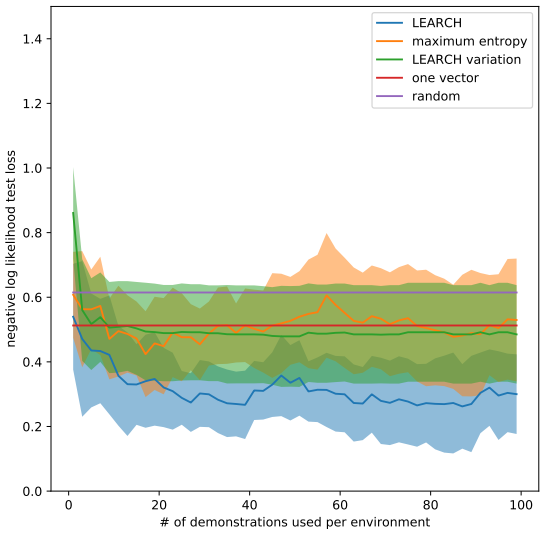
(e) Euclidean distance transform training loss



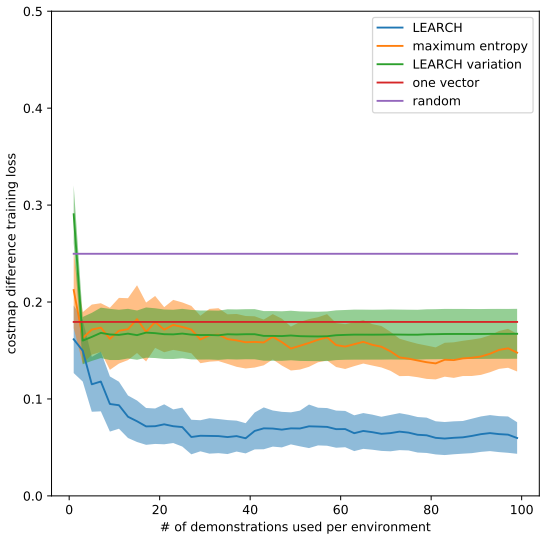
(f) Euclidean distance transform test loss



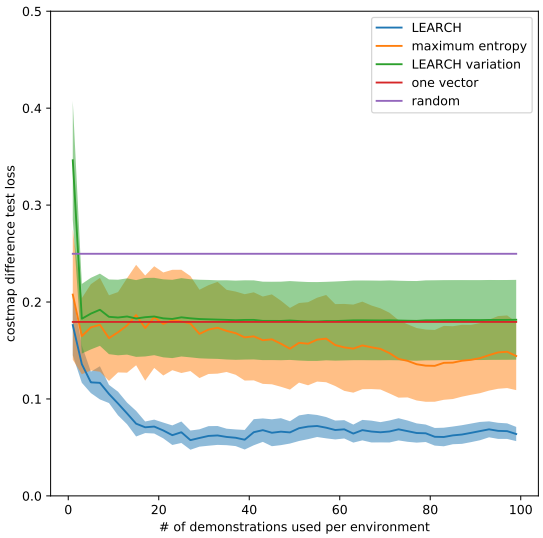
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss

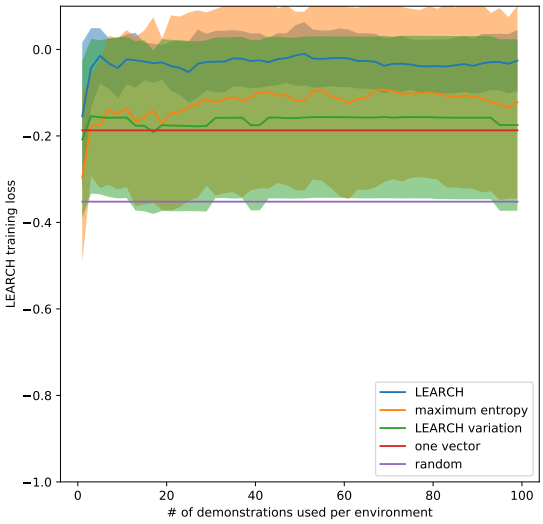


(i) Costmap difference training loss

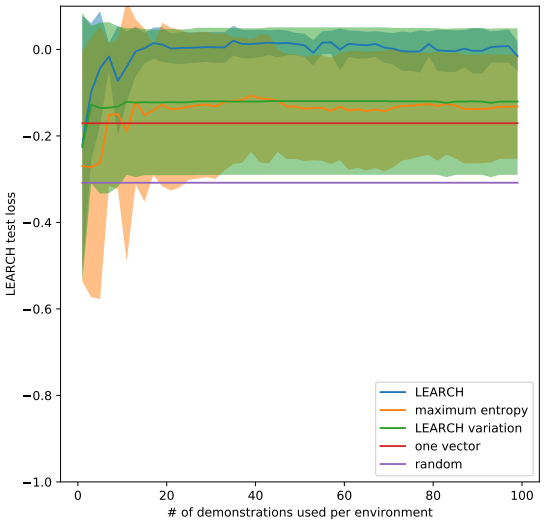


(j) Costmap difference test loss

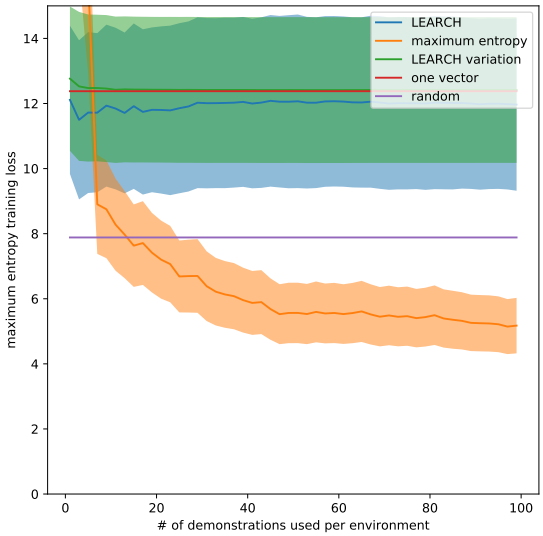
Figure A.3: Training and validation losses for one to 100 demonstrations. Learning is done on ten environments simultaneously



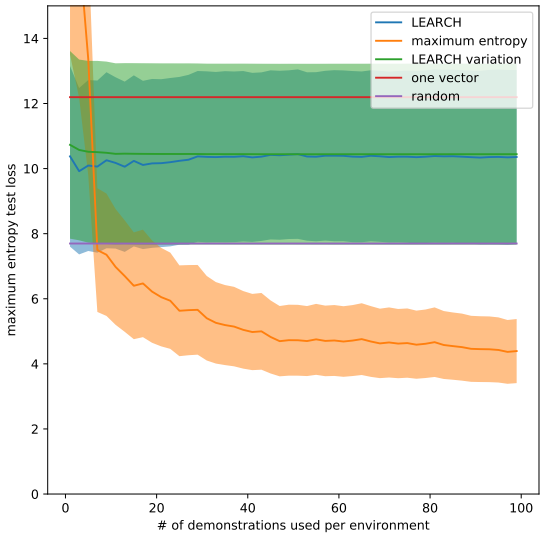
(a) LEARCH training loss



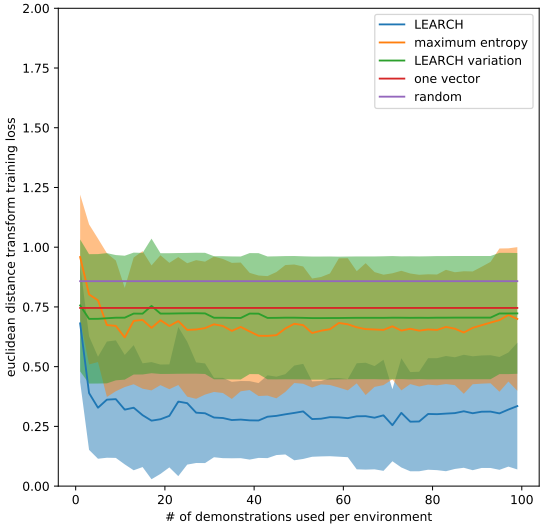
(b) LEARCH test loss



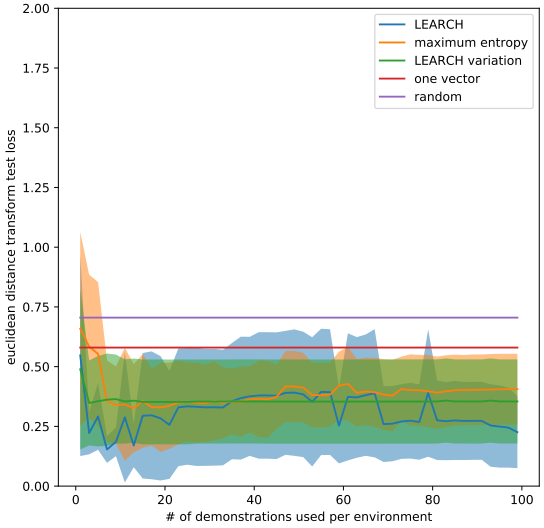
(c) Maximum entropy training loss



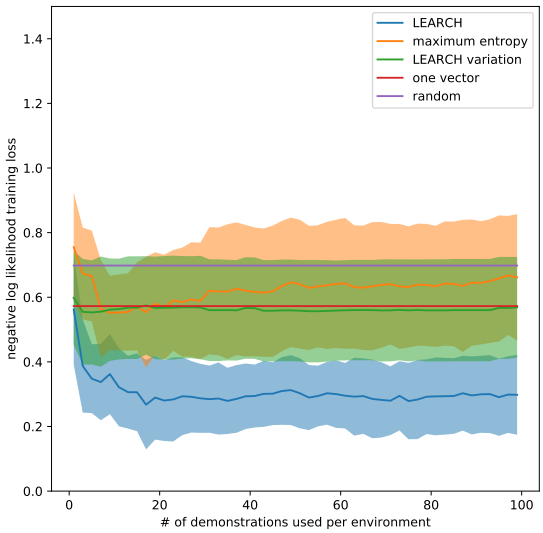
(d) Maximum entropy test loss



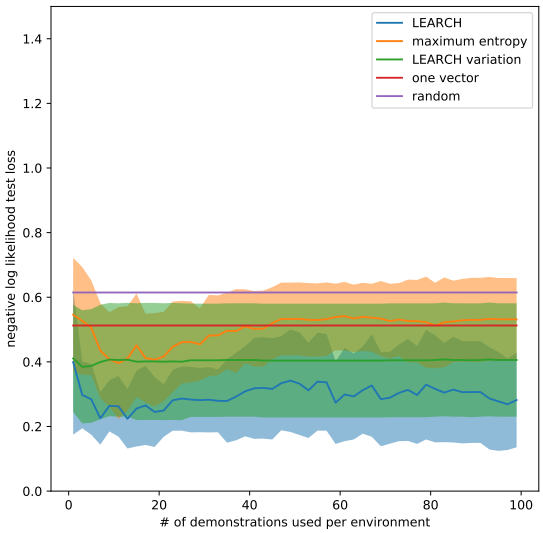
(e) Euclidean distance transform training loss



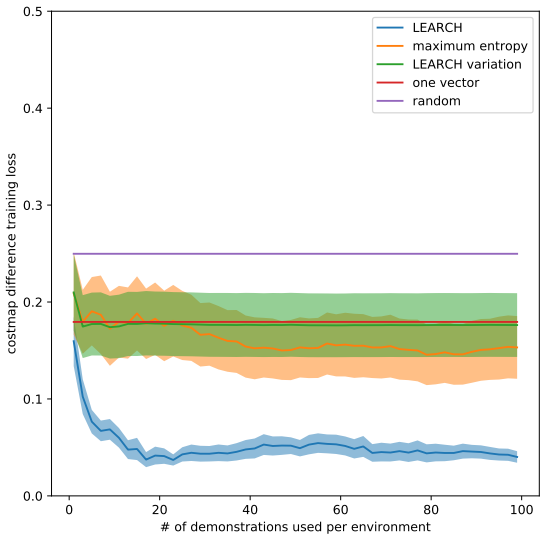
(f) Euclidean distance transform test loss



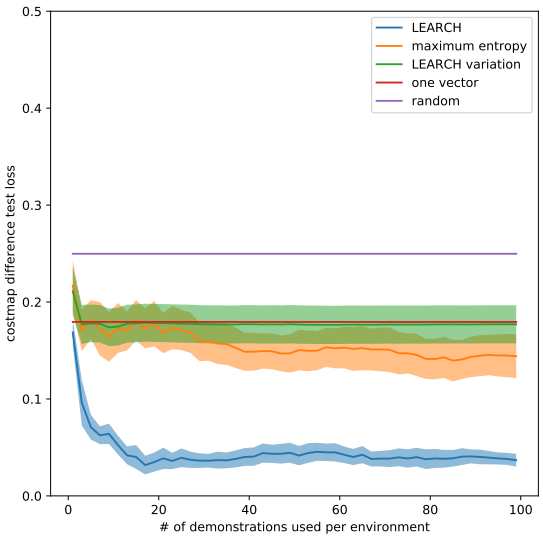
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss

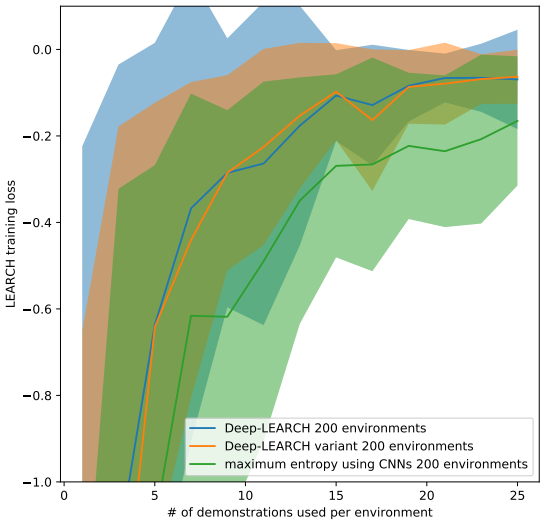


(i) Costmap difference training loss

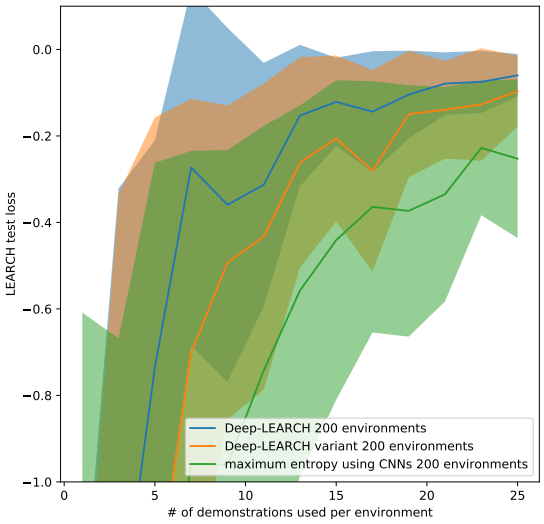


(j) Costmap difference test loss

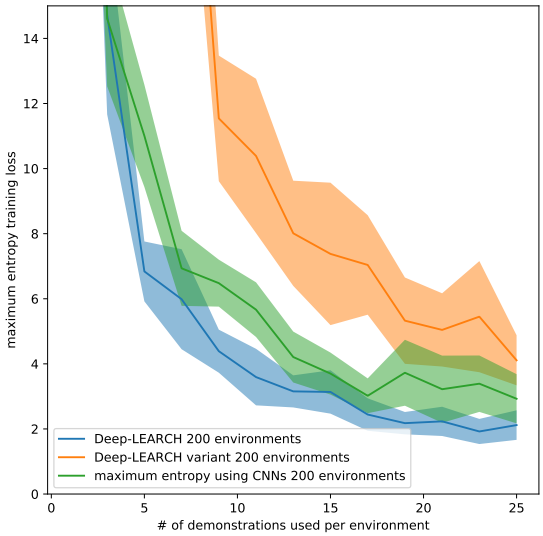
Figure A.4: Training and validation losses for one to 100 demonstrations. Learning is done on 20 environments simultaneously



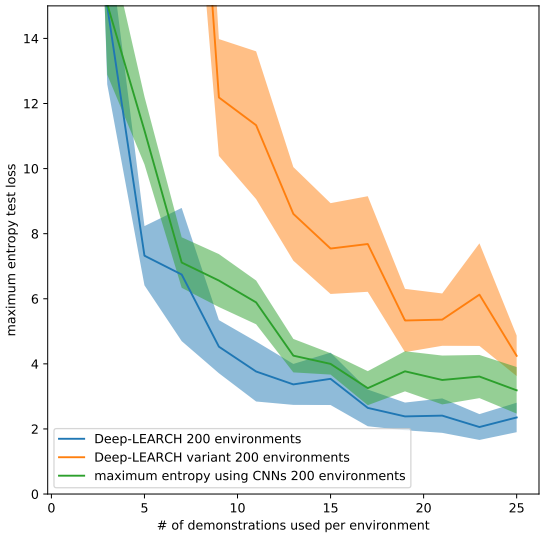
(a) LEARCH training loss



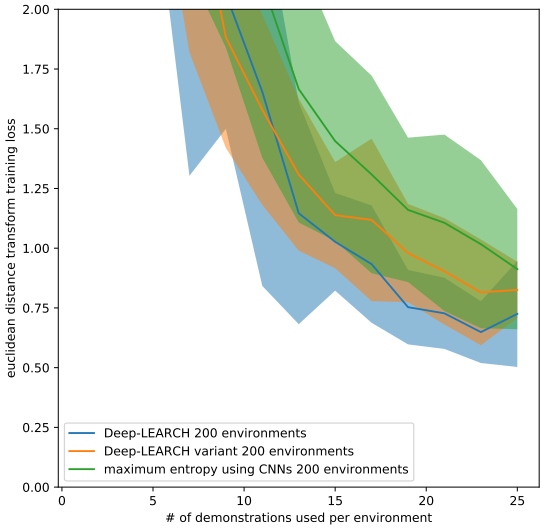
(b) LEARCH test loss



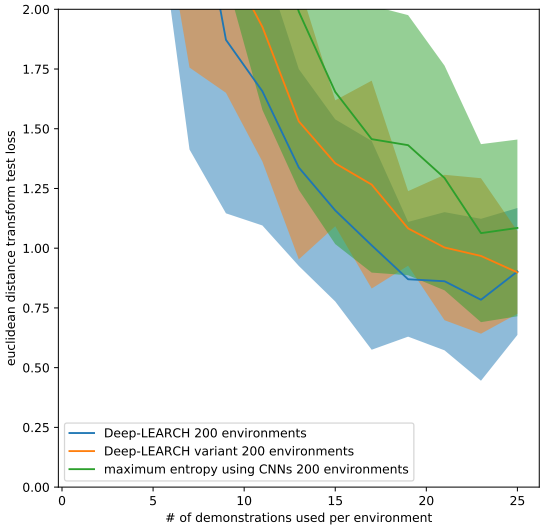
(c) Maximum entropy training loss



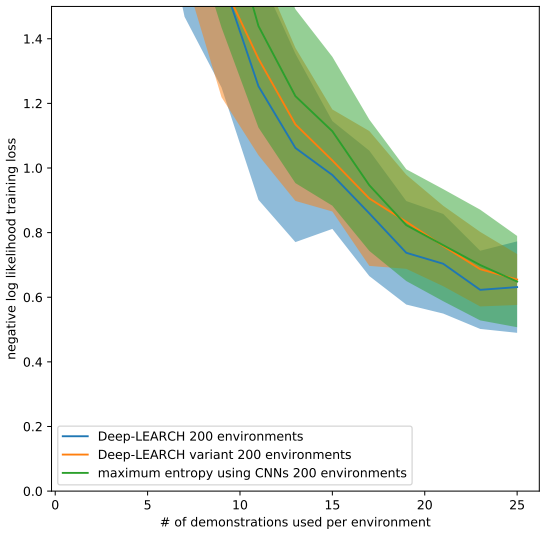
(d) Maximum entropy test loss



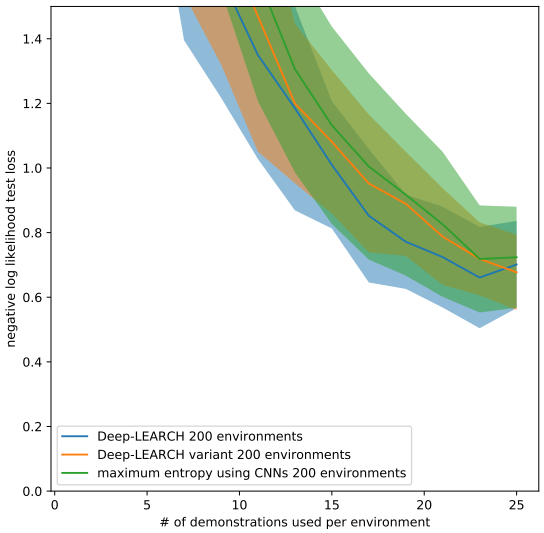
(e) Euclidean distance transform training loss



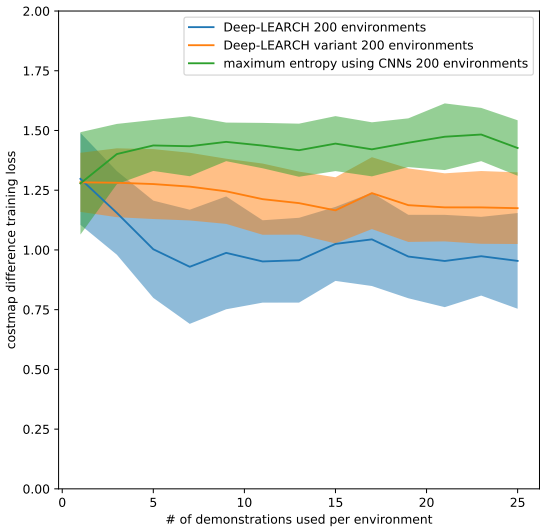
(f) Euclidean distance transform test loss



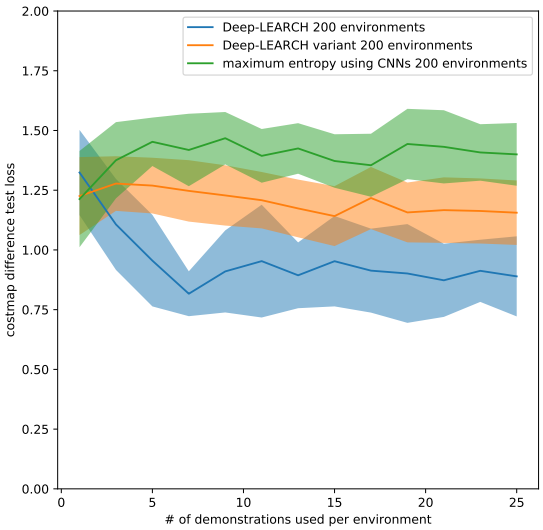
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss

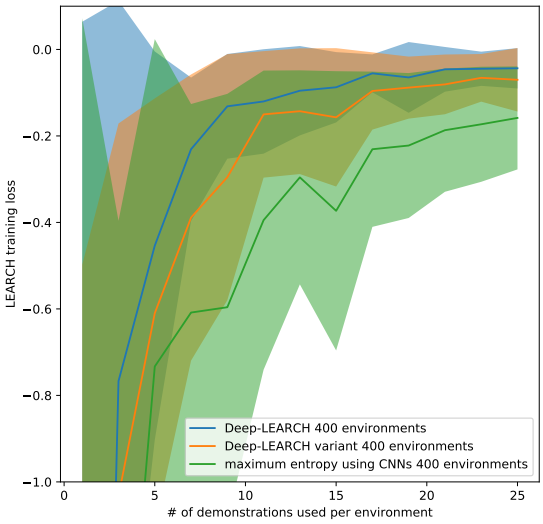


(i) Costmap difference training loss

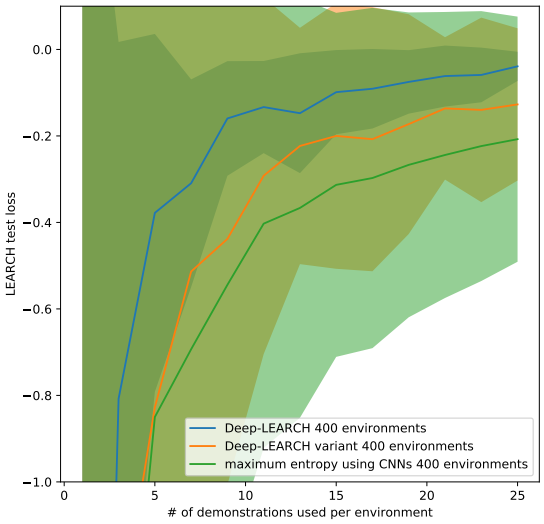


(j) Costmap difference test loss

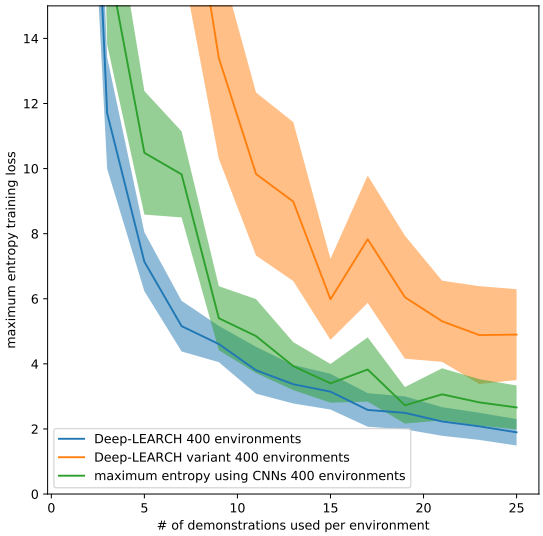
Figure A.5: Training and validation losses for one to 25 demonstrations and 200 training environments



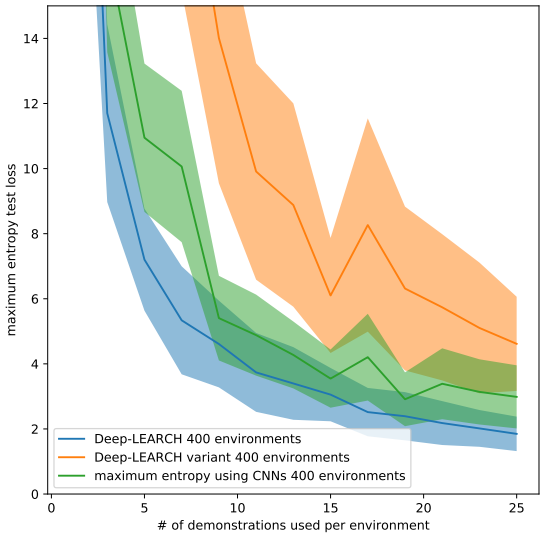
(a) LEARCH training loss



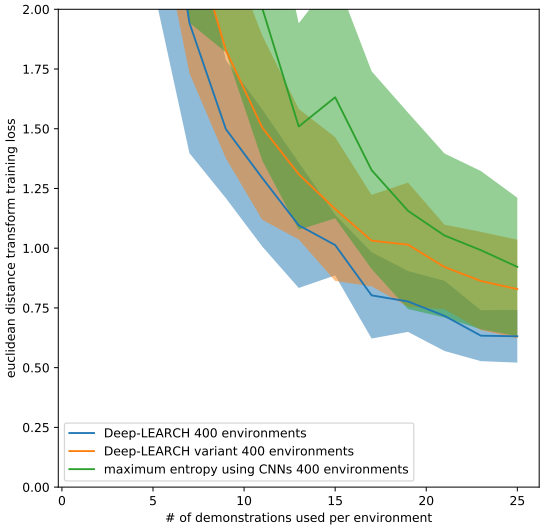
(b) LEARCH test loss



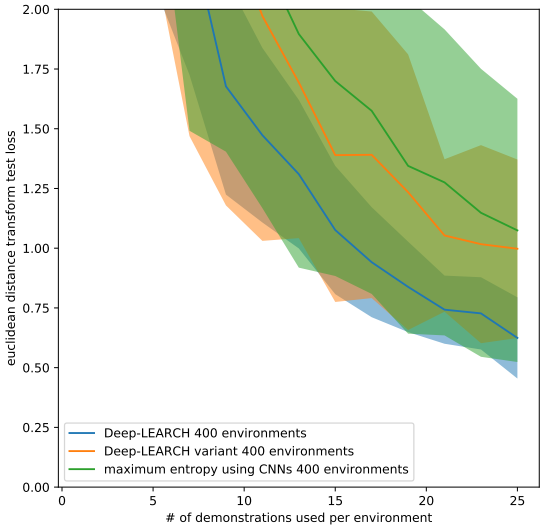
(c) Maximum entropy training loss



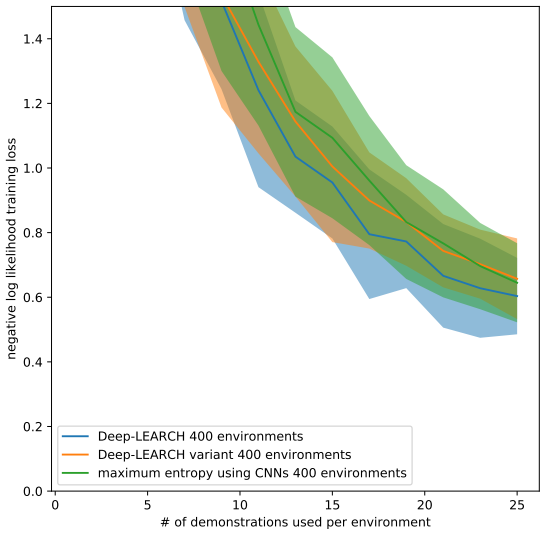
(d) Maximum entropy test loss



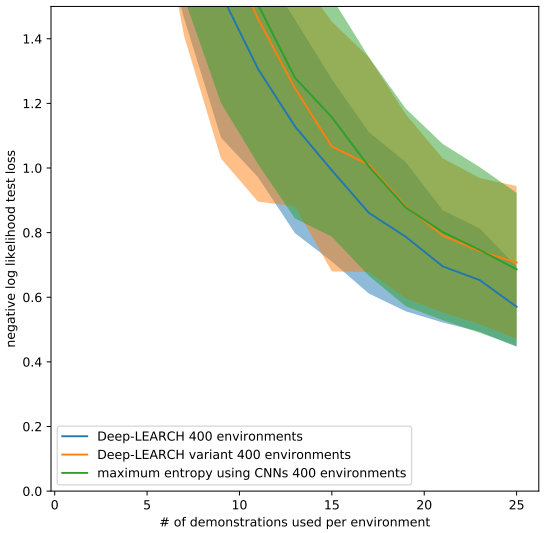
(e) Euclidean distance transform training loss



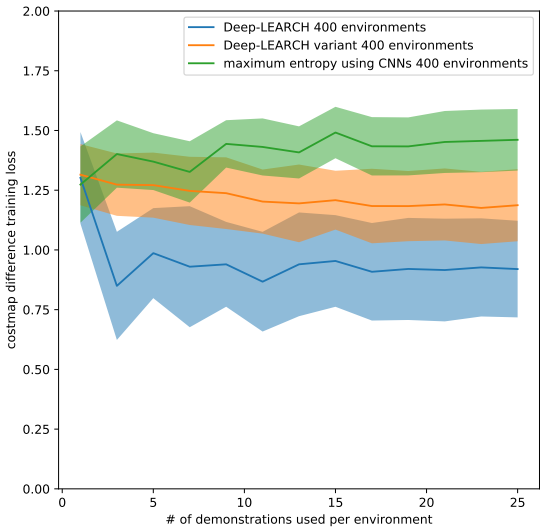
(f) Euclidean distance transform test loss



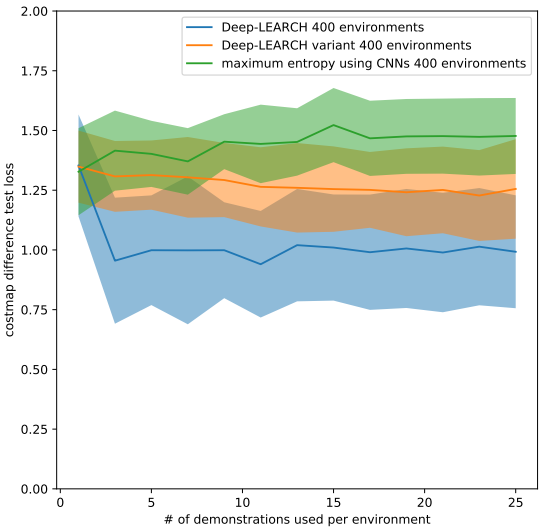
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss

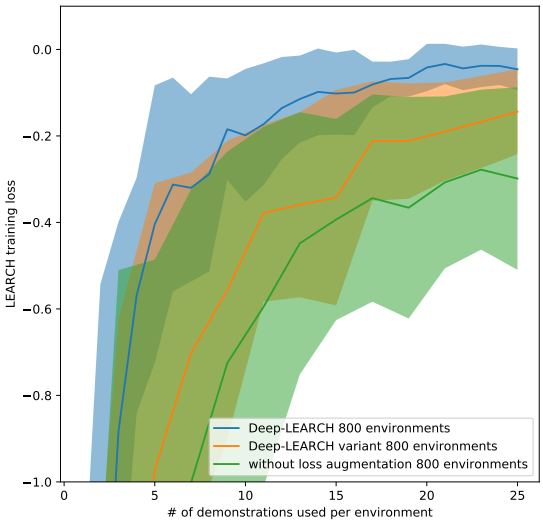


(i) Costmap difference training loss

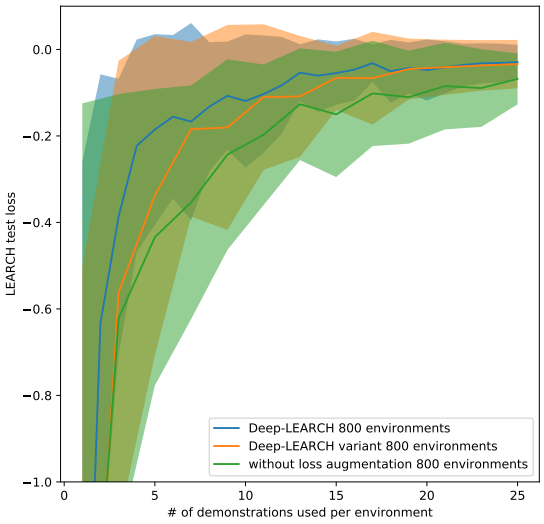


(j) Costmap difference test loss

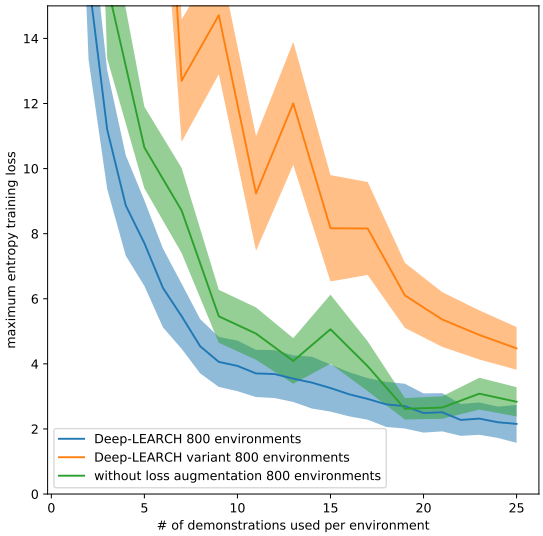
Figure A.6: Training and validation losses for one to 25 demonstrations and 400 training environments



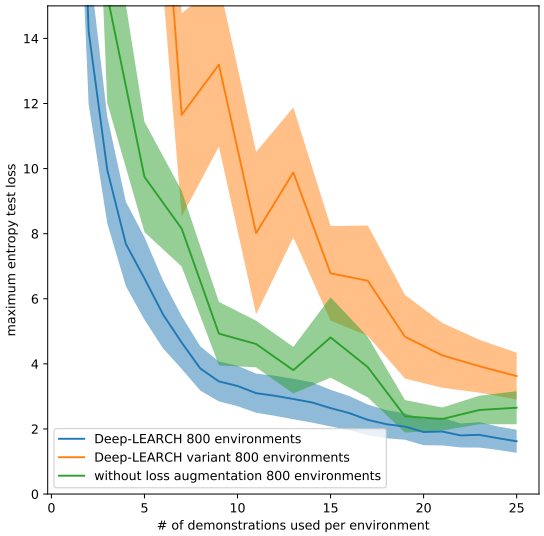
(a) LEARCH training loss



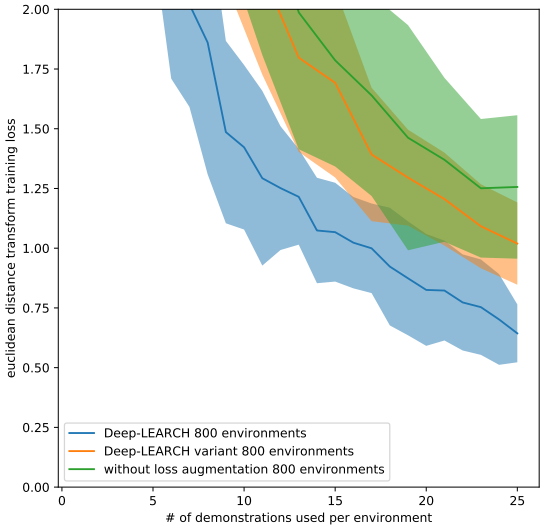
(b) LEARCH test loss



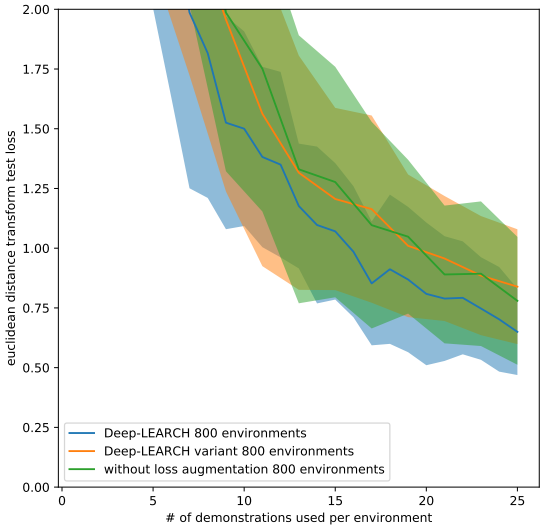
(c) Maximum entropy training loss



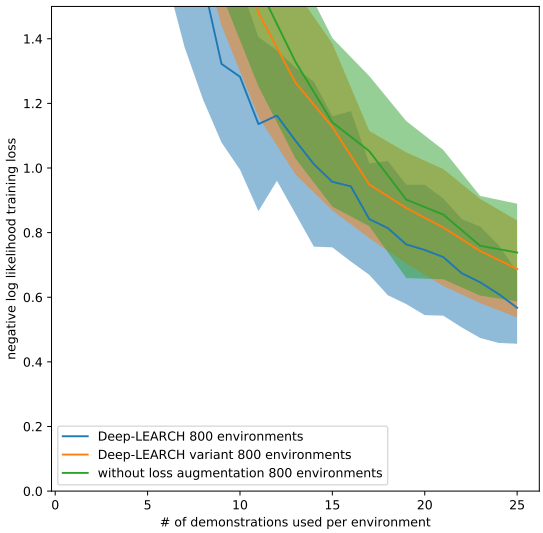
(d) Maximum entropy test loss



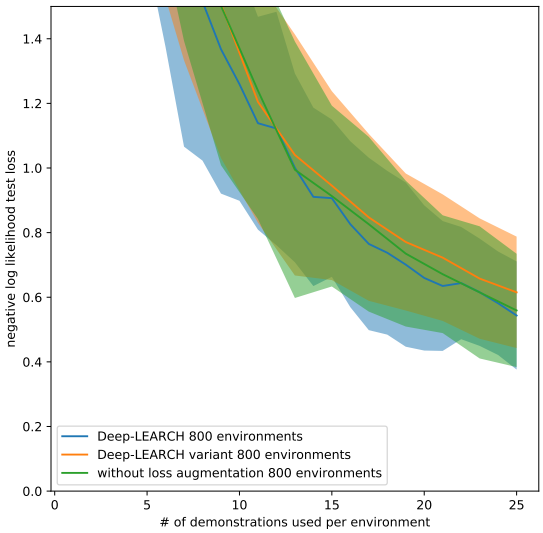
(e) Euclidean distance transform training loss



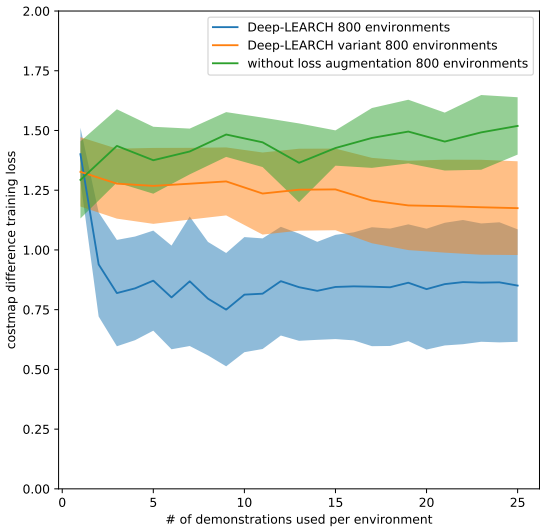
(f) Euclidean distance transform test loss



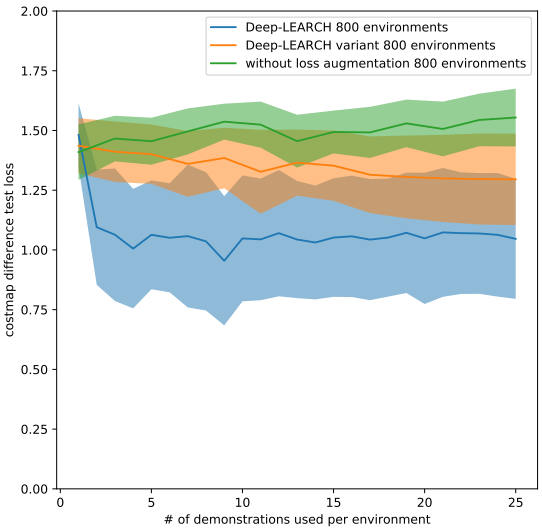
(g) Neg-log-likelihood training loss



(h) Neg-log-likelihood test loss



(i) Costmap difference training loss



(j) Costmap difference test loss

Figure A.7: Training and validation losses for one to 25 demonstrations and 800 training environments

B Implementation Details

The parameter choices are depicted in Table B.1 and Table B.2.

In the environment setup using radial basis functions, σ is fixed to 0.15. In the signed distance function environment ϵ is fixed to 0.1. The radius of the obstacles is chosen randomly from the interval $[0.05, 0.3)$. This forces the obstacles to have a reasonable size. Furthermore, the centers of the obstacles are chosen such, that the obstacles do not overlap. With the trajectory optimization described by Toussaint [Tou17] the trajectories get smoothed. This has the advantage that the trajectories do not overlap as much as if only the Dijkstra [Dij59] algorithm is used. We decided against doing so, since this improved the runtime by the factor 100. While computing 100 demonstrations with Dijkstra on a costmap took about 3.548 s, it take 293.580 s for the same amount of demonstrations, if optimized trajectories are used. With the limited computation capacity this decision sped up the evaluation process.

The implementation of the CNN follows [Cha17] and [MBK+16]. The network consists of a sequence of three convolutions, three deconvolutions and one output layer. Each convolution is followed by a max pooling operation. Before every deconvolution an upsampling operation is applied. Rectified Linear Unit (ReLU) is used as activation function.

Parameter	**LEARCH**	**maximum entropy**	**LEARCH variation**
learning rate r	1	0.4	1
step size scalar m	1	1	1
loss deviation s	10	–	10
loss scalar a	1	–	1
l_2 regularizer	1	–	1
proximal regularizer	0	–	0
iteration count N	–	45	45
convergence rate	0.1	1	0.1

Table B.1: Parameters of LEARCH, maximum entropy IRL and the LEARCH variation

Parameter	Deep-LEARCH	maximum entropy with CNNs	Deep-LEARCH variation
learning rate r	1	1	1
step size scalar m	1	1	1
loss deviation s	10	–	10
loss scalar a	1	–	1
iteration count N	–	35	35
convergence rate	1	1	1
CNN training steps	100	100	100
batch size	64	64	64
learning rate of CNN	0.002	0.002	0.002

Table B.2: Parameters of Deep-LEARCH, maximum entropy using CNNs and the Deep-LEARCH variation

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature