Improving the Generalization of Visual Navigation Policies using Invariance Regularization

Anonymous Authors

Abstract

Training agents to operate in one environment often yields overfitted models that are unable to generalize to the changes in that environment. However, due to the numerous variations that can occur in the real-world, the agent is often required to be robust in order to be useful. This has not been the case for agents trained with reinforcement learning (RL) algorithms. In this paper, we investigate the overfitting of RL agents to the training environments in visual navigation tasks. Our experiments show that deep RL agents can overfit even when trained on multiple environments simultaneously. We propose a regularization method which combines RL with supervised learning methods by adding a term to the RL objective that would encourage the invariance of a policy to variations in the observations that ought not to affect the action taken. The results of this method, called Invariance Regularization, show an improvement in the generalization of policies to environments not seen during training. The experimentation is done on the VizDoom environment which contains hundreds of textures, so allowing us to investigate generalization to changes in the visual observation.

1. Introduction

Learning control policies from high-dimensional sensory input has been gaining more traction lately due to the popularity of deep reinforcement learning (DRL) (Mnih et al., 2015; Levine et al., 2015), which enables learning the perception and control modules simultaneously. However, most of the work done in RL chooses to perform the evaluation of the learned policies in the same environment in which training occurred (Cobbe et al., 2018). DRL agents have been notorious for overfitting to their training environments (Cobbe et al., 2018). An agent could have drastically different performance on testing environments even if it manages to maximize the reward during training (Zhang et al., 2018). Supervised learning algorithms have been shown to have some generalization guarantees when adding proper regularization to the

Figure 1. The figure shows how environments may differ in their visual aspects, like textures of the surfaces. The textures provide a differentiator for each environment, where without them the environments would have shared the same state space.

Using the same environments to train and test agents does not give any insight about the generalization abilities of the learned policy. There could be a number of changes in the environment at test time that would degrade the agent’s performance. Variations could appear in the visual aspects that determine the agent’s observation, the physical structure that determines the agent’s state and even some aspects that are related to the agent’s goal (Figure 1). For example, different observations of the same room are encountered at different times of the day (different lighting conditions). New obstacles could be present. Levels of a game could be different, yet playing a few levels should often be enough to figure out how to play the rest. Such variations might result in a new environment where the control model that defined the training environment has changed. A robust policy should generalize from its experience and perform the same skills in the presence of these variations.
In this paper we study the notion of generalization that is appropriate for visual navigation control policies that are learned with DRL. We present: (1) a study of the generalization of visual control policies to certain changes in the underlying dynamical system; (2) an alternative training method that combines DRL with supervised learning, thus using DRL to learn a controller while leveraging the generalization properties of supervised learning. In our experiments we use the VizDoom platform (Kempka et al., 2016) which is easily customizable and enables the generation of numerous variants of a given environment.

2. Preliminaries

Visual navigation for mobile robots combines the domains of vision and control. Navigation can be described as finding a suitable and safe path between a starting state and a goal state (Bonin-Font et al., 2008). Classical approaches split the problem to a sequence of sub-tasks, such as map construction, localization, planning and path following (Bonin-Font et al., 2008). However, each sub-task requires some hand-engineering that is specific to the environment and task which makes it hard to adapt it to different scenarios without performing some tuning. Deep learning approaches enable the use of highly non-linear classifiers that can adapt their inner representations to learn to robustly solve complicated tasks (Goodfellow et al., 2016).

In this work, we use reinforcement learning algorithms coupled with deep learning approaches to solve the task of navigating an agent towards a goal object using only its visual observations as input. The field of view of the agent is limited, i.e. it does not observe the full environment, and we do not provide an explicit map of the environment to that agent.

2.1. Problem Statement

We model the problem of visual navigation as a partially observed Markov decision process (POMDP) (Spaan, 2012). A POMDP is given by a tuple

\[ \mathcal{P} := (\mathcal{S}, \mathcal{A}, \Omega, R, T, O, \mathbf{P}) \]

where \( \mathcal{S} \) is the set of states, \( \mathcal{A} \) is the set of actions and \( \Omega \) is the set of observations, all which are assumed to be finite sets. The reward function is \( R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R} \).

The conditional transition probability mass function is \( T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1] \), with the interpretation that \( T(s, a, s') = p(s_{t+1} = s' | s_t = s, a_t = a) \) is the probability that the next state is \( s' \) given that the current state is \( s \) and that action \( a \) is taken. The conditional observation probability mass function is \( O : \mathcal{S} \times \mathcal{A} \times \Omega \rightarrow [0, 1] \), with the interpretation that \( O(s, a, o) = p(o_t = o | s_t = s, a_{t-1} = a) \) is the probability of observing \( o \) in state \( s \) when the last action taken was \( a \), and we allow for a special observation probability \( O(s, o) = p(o_0 = o | s_0 = s) \) when in the initial state \( s \) and no action has yet been taken. Finally, \( P_0 \) is the initial state probability mass function, so that \( P_0(s) = p(s_0 = s) \) is the probability that the initial state is \( s \).

In DRL, we work with a parameterized policy \( \pi_{\theta}(h, a) = p_{\theta}(a_t = a | h_t = h) \) with parameters \( \theta \in \Theta \), giving the probability of taking action \( a \) given observation-action history \( h_t := (o_0, a_0, o_1, a_1, \ldots, o_{t-1}, a_{t-1}) \). The objective is to adjust the parameters \( \theta \) to attain a high value for the discounted reward

\[ J_{\mathcal{P}}(\theta) := \mathbb{E}_{\pi_{\theta}} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \right] \]

with discount factor \( \gamma \in [0, 1) \). The expectation is over state-observation-action sequences \((s_t, o_t, a_t)_{t \geq 0}^n\) where the initial state \( s_0 \) is drawn from \( P_0 \) and other elements of such a sequence are drawn from \( T, O \) and \( \pi_0 \) respectively (Sutton & Barto, 1998).

Many methods for attempting to approximate optimal policies have been proposed. For instance, policy gradient methods perform gradient ascent on estimates of the expected discounted reward. In this work we use the proximal policy optimization (PPO) algorithm, which arguably shows relatively robust performance on a wide range of different tasks (Schulman et al., 2017).

2.2. Formalizing Generalization

As in classification, we wish to learn from a finite training set but still perform well on previously-unseen examples from a test set. To formalize this, we have a distribution \( \mathcal{D} \) over POMDPs, representing multiple environments or tasks, and we sample \( \mathcal{P}^{\text{train}} \) POMDPs from this distribution \( \mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\mathcal{D}} \). In the context of navigation, these POMDPs might differ in terms of their observation distributions, perhaps representing views of the same environment at different times of day or year, in terms of their transition distributions, perhaps representing maps with different geometries, or in terms of their reward distributions, perhaps corresponding to the specification of different goal states. Given this sample, we then learn a policy \( \pi_{\theta} \) from a finite collection of
Training in synthetic environments enables the simulation of huge amounts of experience in a span of a few hours. Simulations are convenient to use when training reinforcement learning agents that are often highly sample inefficient (Sutton & Barto, 1998). There is, frequently, a gap between the synthetic world and the real-world, mainly due to the manner in which the simulators depict the real-world dynamics and visual appearances. Often, these simulated worlds capture the richness and noise of the real-world with low-fidelity (Tobin et al., 2017). Many have tried to propose transfer learning techniques to bridge the reality gap in order to still make use of fast simulators for training (Taylor & Stone, 2009).

One popular method to bridge the reality gap is by randomizing some aspects of the training environment. This domain randomization technique has been shown to be successful for the transfer of grasping policies from simulated training environments to the real-world (Tobin et al., 2017). However, the learned models resulting from that work are not control policies, but perception modules. Previous work has showed some success in transferring the perception module learned in simulation to the real world, but not the controller.

Cobbe et al. (2018) conduct a large scale study on generalization using a new environment, that resembles an arcade game, which they call CoinRun. They experiment by training on different background images and different level structures. They test with different regularization strategies and network architectures finding that the RL agent has a surprising tendency to overfit even to large training sets. Zhang et al. (2018) reach a similar conclusion, when learning in grid-world environments, and state that the agents have a tendency to memorize levels of the training set. Unlike Cobbe et al. (2018), however, they argue that the methods that inject stochasticity into the dynamics of the system to prevent memorization, such as sticky actions (Machado et al., 2017) and random initializations (Hausknecht & Stone, 2015), often do not help. In our work we are interested in generalization when navigating under partial observability unlike the fully observable CoinRun or grid-world environments.

Domain adaptation methods have also been used for simulated to real transfer. They allow models trained on a source domain to generalize to a target domain. Boussmalis et al. (2017) train a generative model to adapt the synthetic images of the simulator to appear like the real environment. It was shown to successfully transfer a grasping policy trained in simulation to the real world. However, they do not discuss whether the policy generalizes when variations happen in the target domain.

Another aspect of generalization is the transfer of learned skills to solve different tasks. In other words, generalization to the goal of the trained agent $g$. Achieving different tasks would require the agent to have the ability to maximize different reward functions. Schaul et al. (2015) consider working with value functions that contain the goal $g$ as part of the agent’s state. They call them universal value functions. The reward will then become a function of a state-action-goal tuple $(s, a, g)$ instead of a classical state-action pair. In the paper, the authors present universal value function approximators (UVFA). A method that attempts to learn a universal value function estimate $V_0(s, g)$. They show that UVFA’s can generalize for unseen...
state-goal pairs in grid-world setup.

Deep reinforcement learning has been used to train control policies. These DRL based methods generally propose to learn motor control commands from raw camera images, thus mapping pixels to commands that control the robot’s motors (Levine et al., 2015). DRL algorithms have been used for various navigation tasks such as goal conditioned navigation (Mirowski et al., 2016; Zhu et al., 2016) and mapless navigation (Mirowski et al., 2018).

In the next section we will discuss how to explore the effectiveness of domain randomization techniques for the generalization of visual navigation policies.

4. Generalization in Visual Control

The motivation behind domain randomization is that it is assumed to be an effective technique to provide a policy that is invariant to the changes that would appear in the observations. We explore the problem of navigating the agent towards a goal object with random noise added to the agent’s observations. If the agent is able to perform the task in an environment defined by a POMDP $P_1$ then it should still be able to perform the task in another POMDP $P_2$, if certain features $f$ of the environment that are specific to successfully achieving the task exist and are invariant to these variations, i.e., $f(P_1) = f(P_2)$. Domain randomization is typically used to train policies that can generalize to variations and noise in the observations. It is done by training on several POMDP’s that share the same $S, A, \Omega$ spaces, however each POMDP has its own unique identifier which modifies the state, therefore presenting several variations of the observation of the same state.

We present in the experimentation, Section 6.1, a study on domain randomization when added to RL training and the ability of resulting policies to generalize in unseen POMDPs. We want to investigate if the policy does in fact overfit to the training POMDPs and whether we mitigate that overfitting by training the policies on multiple POMDPs. We also discuss the role that invariant channels might play in the success of domain randomization techniques in achieving a policy that is robust to changes in observations.

5. Invariance Regularization

In the previous sections, we discussed how overfitting to the training environment can be a big problem in RL. Furthermore, we should be careful not to jump to the conclusion that training on different environments will ensure policies that generalize well to new environments. It is merely an assumption that has been shown to empirically hold up when used in a supervised learning context. However, we show in this work that this assumption might not hold for reinforcement learning techniques. This is compatible with the findings in Cobbe et al. (2018) and Zhang et al. (2018).

We reason that in order to generalize well, the training objective should include a term that encourages policy generalization. Therefore, putting the weight of the problem of generalizing explicitly in the objective function itself. Formally, a function $h$ of variable $x$ is invariant to a transformation $\phi$ of $x$ if $h(x) = h(\phi(x))$. We can deduce the same definition for the invariance of a policy $\pi$ to changes in the observation given by some transformation $T$, $\pi(o) = \pi(T(o))$. We add this regularization penalty term to the RL objective as shown in Equation (1):

$$\max_\theta L_{ppo}(o; \pi_\theta) - \frac{\lambda}{N} \sum_i^N d(\pi_\theta(o_i), \pi_\theta(T(o_i))),$$ (1)

where $L_{ppo}$ is the PPO objective (Schulman et al., 2017), $\theta$ is the set of parameters that define the policy $\pi_\theta$, $d$ is a distance function between the two conditional distributions, and $\lambda$ is a weighting coefficient of the penalty.

$T$ is a transformation of the observations. Given an observation $o$ and a transformation on that observation $T$ where the transformation still holds the semantic context of the underlying state, but with added visual variations. We can think of the difference between observing a room with observation $o$ and observing the same room with observation $T(o)$ as the color of the wall for example. Therefore, let us say that we observe $o$ in POMDP $P$ and observe $T(o)$ in POMDP $P^T$ then $f(P) = f(P^T)$, where $f(P)$ is the set of invariant features of the environment defined by the POMDP $P$. We further discuss the nature of $T$ in the experiments section.

The penalty $d$ in Equation 1 resembles adding a constraint on the PPO objective, where the new objective dictates that the policy should simultaneously obtain a high reward while behaving similarly for the observations $o$ and $T(o)$. The idea is similar, in spirit, to trust region policy optimization (Schulman et al., 2015) where a penalty term, resembling that which would result from imposing a trust-region constraint, is added to ensure monotonic improvement of the average return with each policy update. We call this method in Equation 1 invariance regularization (IR) since the regularization term indicates the invariance of the learned policy to a transformation of given observations. We tried to solve the RL problem in Equation 1 in several ways. We found that the most successful way is to do this in an iterative manner as shown in Algorithm 1. We found that splitting the training process to two stages of training...
We leverage the customizability of VizDoom maps and perform these experiments because we are interested in finding if we can train an actor-critic style agent (Konda & Tsitsiklis, 1999) to solve the task. The network consists of three convolutional layers and 2 fully connected layers, followed by the policy and value function estimator layers. The policy output is a four dimensional fully connected layer, where the four dimensions correspond to four actions; move forward, turn right, turn left and do nothing. The output of the policy layer is a log probability of each action. The value layer is a single unit that predicts the value function. This network architecture is proposed by Mnih et al. (2015), ReLUs are used as the non-linear operations in all layers (Nair & Hinton, 2010). As mentioned, we optimize the PPO objective (Schulman et al., 2017) with a binary reward function (+1 if goal is reached, 0 otherwise).

We generate the variations of the training environment by changing the textures on the surfaces using the numerous textures provided by VizDoom (Kempka et al., 2016). We train agents on a subset of 1, 10, 50, 100 and 500 rooms from the generated environments and test on 50 rooms with textures from a held-out set which are different from the ones used to generate the training environments. During training we run several agents in parallel to quickly collect observation-action-reward data in multiple environments. Another advantage is the ability to run one parallel agent on a single training environment. Due to hardware limitations, we cannot run an agent for each environment, at least not when we have a large number of training environments, i.e., 100 or 500. Therefore, each agent will sample one environment from the training set and run on it for some $n$ episodes before sampling another one. We keep a set of the already-seen environments which some agents can sample from. This sampling strategy ensures that the agents are running on a mixture of new environments and environments that have already been sampled and ensure that all environments get sampled with a good amount of exposure to the model while training. We experiment with different types of visual input RGB, RGB-D, Grayscale and Grayscale-D.

6. Experiments

In this section we present the results of two experiments. The first, is about training RL with domain randomization. We discuss the ability of the learned policies to generalize to unseen environments when trained on variations of the training environment. The next part presents the results obtained when using IR with domain randomization and shows that it improves the success rate considerably.

We performed these experiments because we are interested in the following questions: (1) Does training on environments with random variations (as domain randomization suggests) learn a representation of the invariant $f$ with which the policy can generalize to any environment that shares the same invariant features? (2) Can we find a training algorithm that would empirically guarantee finding these invariant features $f$?

6.1. Domain Randomization

We leverage the customizability of VizDoom maps (Kempka et al., 2016) with hundreds of unique textures to generate train/test scenarios. The agent is required to reach an object in order to get a reward. We train an actor-critic style agent (Konda & Tsitsiklis, 1999) to solve the task. The network consists of three convolutional layers and 2 fully connected layers, followed by the policy and value function estimator layers. The policy output is a four dimensional fully connected layer, where the four dimensions correspond to four actions; move forward, turn right, turn left and do nothing. The output of the policy layer is a log probability of each action. The value layer is a single unit that predicts the value function. This network architecture is proposed by Mnih et al. (2015), ReLUs are used as the non-linear operations in all layers (Nair & Hinton, 2010). As mentioned, we optimize the PPO objective (Schulman et al., 2017) with a binary reward function (+1 if goal is reached, 0 otherwise).

We generate the variations of the training environment by changing the textures on the surfaces using the numerous textures provided by VizDoom (Kempka et al., 2016). We train agents on a subset of 1, 10, 50, 100 and 500 rooms from the generated environments and test on 50 rooms with textures from a held-out set which are different from the ones used to generate the training environments. During training we run several agents in parallel to quickly collect observation-action-reward data in multiple environments. Another advantage is the ability to run one parallel agent on a single training environment. Due to hardware limitations, we cannot run an agent for each environment, at least not when we have a large number of training environments, i.e., 100 or 500. Therefore, each agent will sample one environment from the training set and run on it for some $n$ episodes before sampling another one. We keep a set of the already-seen environments which some agents can sample from. This sampling strategy ensures that the agents are running on a mixture of new environments and environments that have already been sampled and ensure that all environments get sampled with a good amount of exposure to the model while training. We experiment with different types of visual input RGB, RGB-D, Grayscale and Grayscale-D.

The role of depth. Adding a depth channel to the observation plays a significant role in generalization. Depth is invariant to many changes in the visible spectrum of the observations. This might lead the training agent to partly find an invariance in observations in its implicit perception model, which in this case can be as simple as focusing of the depth channel only. Therefore, it was not surprising to see, in Table 1, the depth agents (RGB-D, Grayscale-D) are generalizing better than the agents without any depth information.

Table 1 shows the success rate of the PPO models with respect to the number of training environments used and the input type (RGB, RGB-D). The results are averaged over 5 seeds, a standard practice in most literature today. We notice the superior performance of the agent with depth.

### Algorithm 1: RL with iterative supervision

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>210</td>
<td>Initialize $k_1, k_2, \theta_0, T_i = (1...,N), env$</td>
</tr>
<tr>
<td>211</td>
<td>while not converged do</td>
</tr>
<tr>
<td>212</td>
<td>for $i = 1..., k_1$ do</td>
</tr>
<tr>
<td>213</td>
<td>// Train $\pi_{\theta_i}$ on $env_i$ on the RL objective</td>
</tr>
<tr>
<td>214</td>
<td>$\theta_i \leftarrow \max_{\theta} \ L_{ppo}(o^{env}; \pi_{\theta_{i-1}})$</td>
</tr>
<tr>
<td>215</td>
<td>end for</td>
</tr>
<tr>
<td>216</td>
<td>for $j = 1..., k_2$ do</td>
</tr>
<tr>
<td>217</td>
<td>// Train $\pi$ on env and $T_i(env)$</td>
</tr>
<tr>
<td>218</td>
<td>Sample ${o^{env}<em>{t, \pi</em>{\theta_k}}, o^{env}_{t,i}}$</td>
</tr>
<tr>
<td>219</td>
<td>Generate ${o^{T_i(env)}<em>{t, \pi</em>{\theta_k}}, o^{T_i(env)}_{t,i}}$ for $t=1...N$</td>
</tr>
<tr>
<td>220</td>
<td>$\theta_j \leftarrow \min_{\theta} d(\pi_{\theta_{i-1}}(o_{env}), \pi_{\theta_j}(o_{T_i(env)}))$</td>
</tr>
<tr>
<td>221</td>
<td>end for</td>
</tr>
<tr>
<td>222</td>
<td>end while</td>
</tr>
<tr>
<td>223</td>
<td>return $\pi_{\theta_j}$</td>
</tr>
</tbody>
</table>
than the agent without depth. The fact that the RGB agent is not able to generalize well even when exposed to numerous environments tells us that it might not be learning to catch the invariance relating the environments. On the other hand, the RGB-D agents achieve a good performance on the testing environments even when the agents are only exposed to 10 random training environments.

Looking at the RGB RGB-D experiments, the agents trained on 100 and 500 environments generalize worse on average than the ones trained on 10 and 50, which indicates that some agents might be overfitting. Looking at the max statistic of these results we see that the 100 and the 500 outperform the rest. However, they have a higher variance in the success rates between different seeds. It is worth noticing that these agents are able to maximize the reward in the training set regardless of the set size. High variance in the test results of the 100/500 RGB-D experiments show that some seeds are able to achieve a near perfect score on the testing environment and others completely fail, thus indicating the lack of an empirical guarantee that RL agents will generalize when exposed to numerous environments.

Since we do not see the same variance in the Grayscale-D results, we argue that the RGB-D input is more complicated and therefore when presented with a large complicated set, the agent is more prone to overfit to that type of input that to the Grayscale-D one. The average success rate for the RGB input without the depth shows that domain randomization alone might not be an effective method to adapt the policy to variations in the observations, at least not in the context of RL. In fact, it shows little progress, e.g., the RGB agent exposed to one environment achieves around a 20% success on the testing environments and the agents exposed to 50+ environments achieve less than 40% success. These results are consistent when running with a grayscale channel.

While training by randomizing the environment did show some success in making supervised learning models generalize better, it fails to do so in RL policies. It is clear from these results, that adding random variations and relying solely on the RL objective is not enough to ensure generalization. Much of the success of domain randomization in previous works (Tobin et al., 2017) was reported using supervised learning. Also, the generalization abilities of machine learning algorithms have been linked to supervised learning setups. Therefore, it would make sense to adapt supervised learning techniques to regularize the models trained with DRL.

6.2. Invariance Regularization Experiments

In this section we will discuss the results obtained from training the agent using the method displayed in Section 5.

As mentioned in Section 5, we split the objective into two parts: the first will be for training RL on the observations of the original training environment, the second will be a part of a supervised learning step on the transformed observations. This worked better when done iteratively as described in Algorithm 1. In the following experiments, we trained the model with one iteration of the algorithm. Therefore, the training process has two stages, train RL then train with a supervised learning setup, without iterating between both.

As for the nature of transformation $\mathcal{T}$ of the observations, we tested with the same randomly textured environments from VizDoom, that were used in the previous section, in order to be able to make good comparison with the pure RL and domain randomization agents. We train RL on one environment and then use the actions that the trained policy would have taken in that environment to tune the model with supervised learning on the textured environments. Regarding the distance penalty term $d$ in Equation 1, we tested with the KL divergence, $L1, L2$ and cross entropy terms, the KL divergence returned the best results. When performing the step to minimize the KL, we copy the layers of the policy learned with RL to a separate network $\pi_{\text{label}}$ to provide supervision with fixed targets that we can use to train the policy $\pi$ on $\mathcal{T}(o)$ to minimize the objective function $KL(\pi(T(o)) \| \pi_{\text{label}}(o))$.

Table 1 shows the results for combining PPO with our proposed regularization penalty. We see that the success rates for the RGB-D input is close to that of the vanilla PPO agents. However, we see noticeable improvement in the 100/500 agents. The success across different seeds is more consistent and therefore the average performance is better and the variance is lower. Along the RGB input our agent outperforms the agent trained just on domain randomization and RL. This shows us a clear advantage that adding a term on the generalization of the policy has. We also notice similar results for the Grayscale input which further shows that this method is helping the policy generalize even when the input doesn’t contain explicit invariant channels. As for the Grayscale-D results, our method was not able to outperform the RL models on average, however it still achieves a good average success rate.

6.2.1. Comparisons with other regularization techniques

Regularization has been shown to help in the generalization of supervised learning models (Srivastava et al., 2014). Using regularization in supervised learning often improves in the performance of the trained models on test sets. However, they have not been frequently used in deep reinforcement learning setups, possibly due to the fact that...
We hypothesize that the high entropy policies are able to generalize by acting randomly in some instances and this makes them more robust in certain situations. This was backed by the resulted success weighted by shortest path length (SPL) of the same experiments shown in Figure 2 (right). SPL is proposed by Anderson et al. (2018) as a way to measure the navigation agent’s success rate while taking into account the time it takes the agent to be successful. A random behavior that displays robustness and returns a high success would return a relatively lower SPL due to the fact that random behavior will most probably not take the optimal shortest possible path to the goal.\footnote{\textit{SPL} formula suggested by Anderson et al. (2018) contains a $\max(p_i, l_i)$ at the denominator instead of $p_i$, as we wrote in Equation 2. We removed this term as we found the max to be redundant since $p_i \geq l_i$.}

\[ SPL = \frac{1}{N} \sum_{i=1}^{N} \frac{S_i l_i}{p_i}, \quad (2) \]

where $N$ is the number of runs, $S_i$ is the binary indicator of the success of episode $i$, $l_i$ is the shortest path possible and $p_i$ is the length of the path taken by the agent. Figure 2 (right) shows that the dropout and L2 agents have a lower SPL than our agent indicating that these policies with the higher entropy are learning to behave with some randomness in order to be successful unlike our method which tend to respect the efficiency that the original pure RL policy converges to when trained on one environment.

### 7. Discussion and Conclusions

We present a study of the generalization capabilities of visual navigation agents trained with deep reinforcement learning algorithms. We formalize what it means to generalize in the context of the POMDP. We find similar conclusion as Cobbe et al. (2018), Zhang et al. (2018), where the tendency of RL agent to overfit even when exposed to large training sets is quite visible. We show

<table>
<thead>
<tr>
<th>Num training envs:</th>
<th>1</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPO</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RGB</td>
<td>0.21 ± 0.04</td>
<td>0.17 ± 0.04</td>
<td>0.35 ± 0.13</td>
<td>0.35 ± 0.16</td>
<td>0.34 ± 0.14</td>
</tr>
<tr>
<td>RGB-D</td>
<td>0.05 ± 0.04</td>
<td>\textbf{0.89 ± 0.05}</td>
<td>0.9 ± 0.05</td>
<td>0.61 ± 0.37</td>
<td>0.77 ± 0.33</td>
</tr>
<tr>
<td>Grayscale</td>
<td>0.36 ± 0.04</td>
<td>0.33 ± 0.13</td>
<td>0.37 ± 0.04</td>
<td>0.47 ± 0.14</td>
<td>0.41 ± 0.22</td>
</tr>
<tr>
<td>Grayscale-D</td>
<td>0.48 ± 0.07</td>
<td>\textbf{0.88 ± 0.11}</td>
<td>\textbf{0.96 ± 0.02}</td>
<td>\textbf{0.97 ± 0.02}</td>
<td>\textbf{0.96 ± 0.01}</td>
</tr>
</tbody>
</table>

\begin{tabular}{l|c|c|c|c|c}

Table 1. Average success rate and standard deviation of agents, that are trained on a different number of randomly environments, when tested on 50 test environments whose textures are not seen during training. The bold values represent the algorithm that resulted in the best average success rate according to an amount of training environments and an input type. We see that our method brings stability to the average results and improves generalization even when no depth is added.

"Training and testing is occurring in the same environment and the generalization gap is absent (Cobbe et al., 2018)."

We compare our method with some regularization techniques that are frequently used. We particularly test three regularization techniques separately: dropout, batchnorm and L2. The first experiment has a dropout layer added after each convolutional layer (Srivastava et al., 2014), the second has a batchnorm layer added after every convolutional layer (Ioffe & Szegedy, 2015) and the last uses L2 regularization.

We choose the dropout probability to be 0.1 and the L2 weight to be $10^{-4}$, the same values that were proposed by Cobbe et al. (2018). We run the same previous setup, train five models (different seeds) for each technique and evaluate on 50 environments whose textures are sampled from a held-out set. We report the experiments done with RGB input only as it poses a harder problem and a larger gap than RGB-D.

Figure 2 (left) shows the average success rate over 5 seeds for the four methods. We see that our proposed method is the only one that is steadily improving when more environments are added. The batchnorm models performed the worse while the dropout and L2 where achieving similar success rates with our methods in the 50 and 500 training environments. However, when running the model, we noticed that the policies learned with dropout and L2 has a certain randomness to their behavior. This is also backed by the fact that the entropy of the learned policies is substantially higher when dropout and L2 are added to the model.

We hypothesize that the high entropy policies are added to the model.
that using domain randomization with RL, without adding invariant features to the input such as the depth maps, is not enough to generalize. Even with added invariance, the agents showed high variance in its success rate on the testing environments. In the second part, we proposed Invariance Regularization, a method that combines supervised learning with RL to leverage the generalization ability of supervised learning techniques and regularizes the RL model. This algorithm improved the generalization success even with no added depth and displayed stability in performance across different seeds.

In this work, we focused our experimentation on the generalization to changes in the input observation because for visual navigation agents deployed in the real-world it would be very difficult to guarantee a stationary observation set. However, another aspect is to generalize the learned skills to different architectural designs of the environment, as different levels of the game as proposed in the retro competition (Nichol et al., 2018).

Another avenue of future work is to explore the appropriate transformation function $\pi$ of the observations, that would be useful to expose the agent to in order to help in robustness and generalization to other environments. One might consider generating adversarial examples as a means to increase the difficulty of the learning problem (Goodfellow et al., 2015). Another way is to use an adaptive form of $\pi$ coupled with data augmentation techniques. One might learn a model to find the augmentation strategy that makes the task harder on $\pi$ which may help it find a generalized representation, as done by Cubuk et al. (2018).

References


Improving Generalization in Visual Navigation Policies using Invariance Regularization


