Semi-Supervised Policy Initialization for Playing Games with Language Hints

Tsu-Jui Fu and William Yang Wang
UC Santa Barbara
{tsu-juifu, william}@cs.ucsb.edu

Abstract

Using natural language as a hint can supply an additional reward for playing sparse-reward games. Achieving a goal should involve several different hints, while the given hints are usually incomplete. Those unmentioned latent hints still rely on the sparse reward signal, and make the learning process difficult. In this paper, we propose semi-supervised initialization (SSI) that allows the agent to learn from various possible hints before training under different tasks. Experiments show that SSI not only helps to learn faster (**1.2x**) but also has a higher success rate (**11%** relative improvement) of the final policy.

1 Introduction

Most Reinforcement Learning (RL) methods (Mnih et al., 2013, 2016) rely on an agent to explore and maximize the feedback reward. Since designing a reward for each step is impractical, a common setting is only to give out the achieved signal. In Atari Grand Challenge (Kurin et al., 2017), only if achieving the goal, the environmental reward is 1. However, this sparse-reward setting makes the agent difficult to learn (Vecerik et al., 2017).

ExtLang (Goyal et al., 2019) incorporates language hints as an additional reward to overcome the sparse-reward issue. They first build 45 different tasks under Montezuma's Revenge game, where each task consists of a starting state, an unknown goal position, and a given hint. They also collect demo clips, which are partial playing records (states and actions) that each corresponds to a hint, as shown in Fig. 1. To provide an additional reward, ExtLang pre-trains a reward module to reflect the relevance between agent actions and the given hint when exploring a task. In this way, they can supply a hint reward instead of only the sparse environmental reward to make the agent easier to explore.

Though providing an additional reward, the hints are usually incomplete (Kuhlmann et al., 2004). Considering a task in Fig. 1, to achieve the goal, the agent should "jump over the skull," "climb up the ladder," and "jump to get the key." However, the given hint only contains the first one, and learning those latent hints still relies on the sparse environmental reward. To deal with this issue, we propose semi-supervised initialization (SSI) that enables the agent to experience various possible hints in advance. We adopt a hint module to generate possible hints for random states and allow the agent to learn from them. With SSI, agents have a better-initialized policy during training for each task.

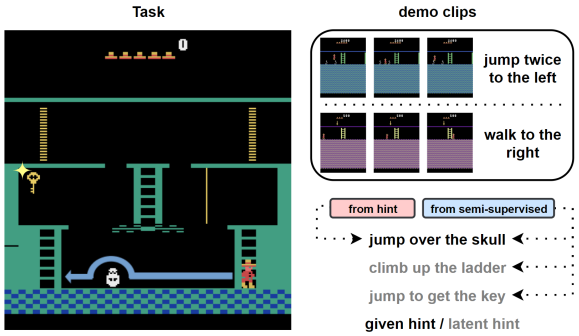

Figure 1: For a task with hint "jump over the skull", both "climb up the ladder" and "jump to get the key" are useful latent hints and can be learned during SSI.

From another point of view, in this paper, we propose a semi-supervised initialization method and investigate the abilities of NLP on controlling complex actions in game environments (Narasimhan et al., 2015; Ammanabrolu and Riedl, 2019).

We perform SSI first and train-evaluate on those tasks built from ExtLang. Experimental results show that with SSI, better-initialized policy not only learns faster but also has a higher success rate.

2 Approach

2.1 Architecture

Fig. 2 illustrates our semi-supervised initialization (SSI). First, the hint module H generates possible hints l for random states s. With s, the policy module P rollouts and step actions a. Then, the

Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 3112–3116
June 6–11, 2021.

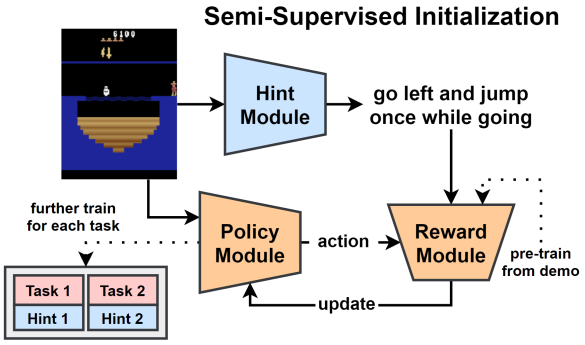

Figure 2: Overview of our semi-supervised initialization (SSI). Random states, along with the hints generated from the hint module, update the policy module by the reward module, and get a better-initialized policy.

reward module R updates P based on the relevance between a and l. With different s, P has the opportunity to learn from various possible hints, and finally serves as a better-initialized policy.

Hint Module (H) H generates a possible hint l for a state s. H adopts CNN to extract the visual feature v of s and attention-based (Bahdanau et al., 2015) GRU (Chung et al., 2014) as the decoder to produce a series of words w as a hint l:

$$\begin{aligned} v &= \text{CNN}(s), h_t = \text{GRU}(w_{t-1}, h_{t-1}), \\ w_t &\sim \text{FC}([h_t, \sum \text{softmax}(h_t W v^T) v]), \\ l &= \{w_1, w_2, ..., w_L\}, \end{aligned} \tag{1}$$

where W is a learnable attention matrix.

Each example in demo clips $\mathcal{D}$ consists of a hint l and a playing record $\{(s_1, a_1), (s_2, a_2), ...\}$. We randomly select s and pre-train H with (s, l) pair.

Policy Module (P) The policy module P is a recurrent action selector which steps a_t for a state s_t at time step t. P applies CNN (Krizhevsky et al., 2012) to extract the visual feature v_t of s_t, GRU to model previous history of s as h, and fully connected layer (FC) to decide which action to step. By rollout, we get $a_{1:T}$:

$$\begin{aligned} v_t &= \text{CNN}(s_t), h_t = \text{GRU}(v_t, h_t - 1), \\ a_t &\sim \text{FC}(h_t). \end{aligned} \tag{2}$$

Reward Module (R) R is a binary classifier[1] which reflects the relevance between l and a, as Fig. 3. Similar to ExtLang (Goyal et al., 2019), we first transform the actions $a_{1:T}$ into action frequency vector f where each value is the ratio of

[1]Though more input (*e.g.*, state frames) may make R more robust, to compare with ExtLang fairly, we use the same setting (the action frequency vector f) as the input.

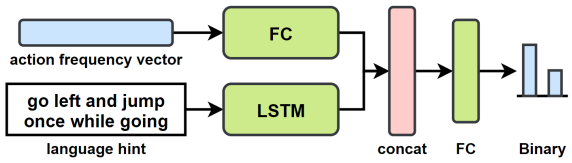

Figure 3: Reward module is a binary classifier which reflects the relevance between the hint and actions.

that action in $a_{1:T}$. R utilizes LSTM (Hochreiter and Schmidhuber, 1997) to encode l as e_l and FC to extract e_f for f. Then, another FC serves as the binary classifier according to e_l and e_f:

$$\begin{aligned} f &= \text{Frequency}(a_{1:T}), \\ e_l &= \text{LSTM}(l), e_f = \text{FC}(f), \\ r &= \text{FC}([e_l, e_f]), \end{aligned} \tag{3}$$

where r is the output of the binary classification and represents the relevance between l and a.

2.2 Semi-Supervised Initialization (SSI)

For a random state s, we adopt H to generate a possible hint l. With starting state s, the agent rollouts and steps actions $a_{1:T}$ by P. Then, R provides the hint reward r_{l_t} for time step t as following:

$$r_{l_t} = \gamma \cdot R(l, a_{1:t}) - R(l, a_{1:t-1}), \tag{4}$$

where γ is a discount factor. This hint reward motivates P to step relevant actions with l.

To update P, we adopt widely used Proximal Policy Optimization (PPO) (Schulman et al., 2017) to maximize r_l during SSI. In this way, the agent learns from various possible hints under different states in advance and has better-initialization for following task-training.

2.3 Task-Training

A task consists of a starting state s, an unknown goal position g, and a given hint l. The agent explores in the environment, starting from s and receives the environmental reward r_E. When achieving g, r_E is 1; otherwise, it is 0 for all other steps.

With better-initialization, we further train P for each task. Similar to our SSI, during task-training, we also have r_l to reflect how relevant of a from P and the given l for this task. Therefore, during task-training, there are 2 kinds of reward, the sparse environmental reward r_E and the hint reward r_l.:

$$\begin{aligned} a_t &= P(s_t), \\ s_{t+1}, r_{E_t} &= \text{Env}(a_t), \\ r_{l_t} &= \gamma \cdot R(l, a_{1:t}) - R(l, a_{1:t-1}). \end{aligned} \tag{5}$$

Finally, we optimize P by maximizing r_E+r_l for this task also using PPO.

Algorithm 1 Learning of SSI and Task-Training

Env: the environment
P: policy module, R: reward module, H: hint module

while DO_SSI **do**
  $s \leftarrow$ Env ▷ random starting state
  $l \leftarrow H(s)$ ▷ generate a hint by H
  $a_{1:T} \leftarrow P(s)$ ▷ rollout s by P
  $r_{l_t} \leftarrow$ hint reward ▷ Eq. 4
  Update P by maximizing r_l using PPO
end while

while DO_Task-Training **do**
  $s, l \leftarrow$ Task ▷ starting state and hint of the task
  $a_{1:T} \leftarrow P(s)$ ▷ rollout s by P
  $r_{l_t} \leftarrow$ hint reward ▷ Eq. 5
  $r_{E_t} \leftarrow$ environmental reward ▷ Eq. 5
  Update P by maximizing $(r_l + r_E)$ using PPO
end while

Learning Process of SSI and Task-Training Alg. 1 describes the learning process of our SSI and task-training. During SSI, P updates to step relevant actions to the generated l from H. Thus, P can consider different hints in advance. During task-training, with better-initialization, P is optimized by both the environmental reward r_E and the hint reward r_l to achieve the final goal.

3 Experiments

Experimental Settings To fairly compare with the baseline ExtLang (Goyal et al., 2019), we conduct the experiments on the same 45 tasks they build under Montezuma's Revenge environment. H is pre-trained by the same demo clips $\mathcal{D}$. We collect the same 160,000 (f, l) pairs as ExtLang to pre-train R. Then, H and R are fixed during SSI and task-training. A task consists of a starting state and a hint, and the agent explores the environment to achieve the unknown goal.

We apply 3-layer CNN to extract the visual feature of a state. Both LSTM and GRU contain 128 hidden units. We utilize PPO to optimize during SSI and task-training with learning rate 7e-4.

As a baseline, ExtLang consists of the same P and R to provide an additional hint reward during task-training. However, without H and SSI, ExtLang explores with a random-initialized policy. We compare ExtLang with our ExtLang-SSI (ExtLang with semi-supervised initialization). All results are averaged from 45 tasks and 5 times experiments.

Quantitative Results Fig. 4 demonstrates the learning curve of ExtLang and our ExtLang-SSI. The x-axis is the training steps of PPO. The upper figure is about the success rate, and the downer one is for accumulated successful episodes[2].

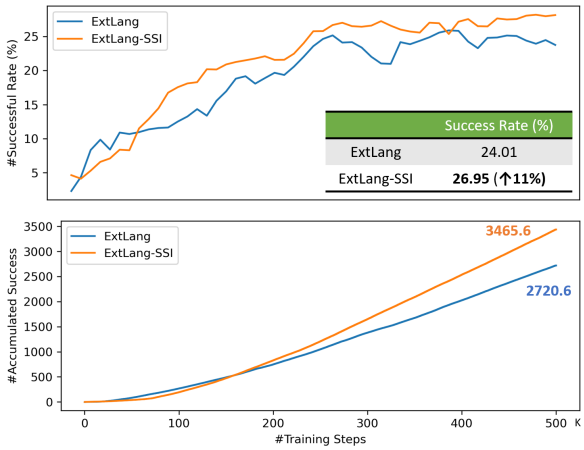

Figure 4: Comparison between learning curves of ExtLang and our ExtLang-SSI.

The results show that under the same training step, ExtLang-SSI can succeed in more episodes than ExtLang. With SSI to learn from possible latent hints in advance, ExtLang-SSI can learn faster than a random-initialized policy. In detail, ExtLang-SSI succeeds 2720 episodes using only 420K training steps where ExtLang requires a total 500K. With better-initialized policy, ExtLang-SSI brings out 1.2x speedup during task-training and success higher 3465.6 episodes in total.

A similar tendency can be found for the success rate. ExtLang-SSI has a higher success rate than ExtLang under the same training step. Apart from the learning curve, we also evaluate the final policy for both ExtLang and ExtLang-SSI. The final success rate is shown in the chart where ExtLang-SSI has a higher 26.95% and outperforms ExtLang with 11% relative improvement. With a better initialization, ExtLang-SSI can lead to a better final policy. The results of both accumulated successful episodes and success rates show that our proposed SSI not only accelerates the learning process but also helps to achieve a higher final success rates.

An interesting insight is that during the early training (before 100K training steps), ExtLang is slightly better than our ExtLang-SSI. Because of learning from various hints in advance, ExtLang-SSI explores the environment based on different latent hints at first. Then, ExtLang-SSI can train faster with experiencing those useful latent hints for this task, and finally, achieve more successful episodes and higher success rate.

[2]Since it is an "accumulated" number, it will keep increasing with more training steps. Note that the training for ExtLang and ExtLang-SSI are both converged.

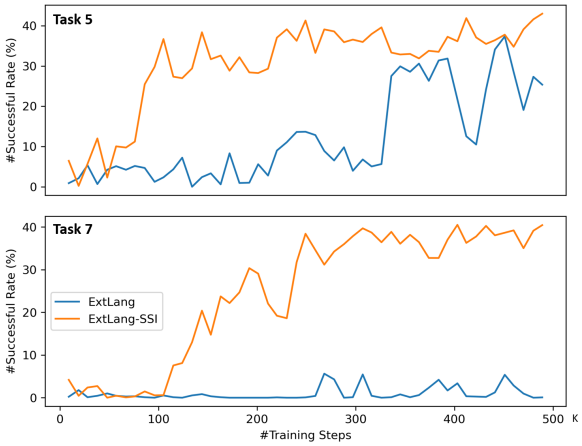

Figure 5: The learning curve for task 5 and 7.

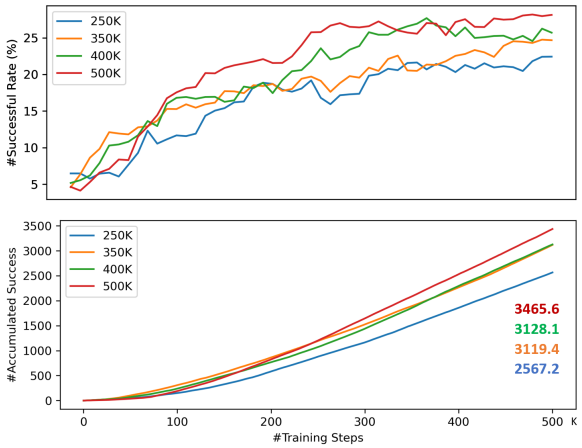

Figure 6: Comparison between learning curves of ExtLang-SSI under different iterations during SSI.

Fig, 5 presents the learning curve about the success rate for Task 5 and 7[3]. For task 5, ExtLang has about 35% success rate at the end, but our ExtLang-SSI outperforms 35% when the very early of training, which means SSI helps to learn faster. Task 7 is more difficult that ExtLang almost fails even with the hint reward. While, with learning from various latent hints, ExtLang-SSI can finally achieve a 40% success rate.

Analysis of SSI To investigate our proposed SSI, We analyze the detailed effectiveness of ExtLang-SSI. Fig. 6 illustrates the learning curves under different iterations during SSI. Similar to Fig. 4, the x-axis is the training step of task-training, and each line is for each iteration number during SSI (250K-500K). We can see that when using 350K iterations to perform SSI, ExtLang-SSI can succeed more than 3000 episodes in 500K training steps. In general, more iterations during SSI enables the agent to access more latent hints with different starting states and helps the agent to learn faster. Besides, SSI also benefits the policy by providing better initialization. Thus, more SSI also makes a higher success rate under task-training.

[3]Task 5 requires the agent to get down and jump over a spider; task 7 needs the agent to turn left, jump, and get a key.

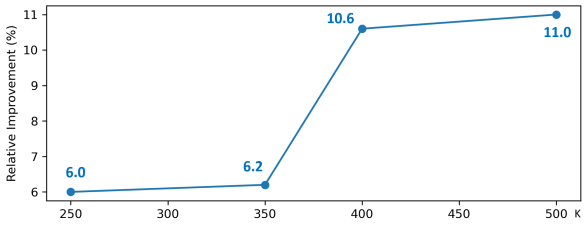

Figure 7: The relative improvement of ExtLang-SSI's final policy under different iterations during SSI.

Noise Rate	0%	10%	30%	50%
Suc. Rate	26.95%	26.38%	24.52%	23.91%

Table 1: The success rate under different noise rates of SSI hints (baseline: 24.01%).

We also evaluate the final policy. Fig. 7 shows the relative improvement of ExtLang-SSI's final policy under different iterations during SSI. Note that the x-axis in Fig. 7 represents the number of iterations during SSI. ExtLang-SSI has a 6.0% relative improvement when applying SSI for 250K iterations. Similar to the learning curves, more SSI brings out a more massive relative improvement and achieve 11% under 500K SSI iterations.

Analysis of Generated Hints We randomly select 100 generated hints and ask people to check if they are relevant to the state. The result shows that 73 are totally corresponding, 21 are relatively corresponding, and only 6 are not corresponding. Our H can actually generate an appropriate hint for a given state so that SSI can help P for better initialization.

We make the noise hints during SSI by randomly pairing a state with any other generated hint. The success rate under different noise rates is shown in Table 1. We can see that a high noise rate will make SSI not that robust. Moreover, if the hints are too noisy, it will even hurt the performance (24.01 down to 23.91). While, we have verified that our H can provide accurate hints. Therefore, SSI benefits the initialization, leading to a better success rate.

Qualitative Results Fig. 8 demonstrates some examples of hint l generated by our H. By updating with hints like "climb down the ladder" or "wait at the bridge appears" , P can learn those latent but useful hints before task-training in a semi-supervised scenario.

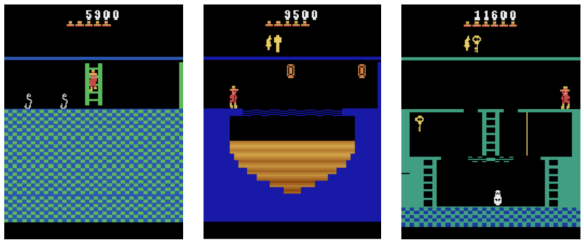

Figure 8: Examples of hint l generated by H.

4 Conclusion and Ethical Considerations

In this paper, we propose semi-supervised initialization that makes the agent learn from various possible hints in advance before play games with language hint. By semi-supervised initialization, the agent can have a better-initialization policy, which benefits further task-training. The experiments show that semi-supervised initialization not only helps the agent to learn faster but also has a higher success rate of the final policy. Our presented SSI can benefit future vision-and-language research for practical applications. In terms of negative impact, since the initialization is learned from those instructions, if there is bias in the original dataset, it may have some potential issues.

Acknowledgments. Research was sponsored by the U.S. Army Research Office and was accomplished under Contract Number W911NF-19-D-0001 for the Institute for Collaborative Biotechnologies. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Prithviraj Ammanabrolu and Mark O. Riedl. 2019. Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning. In *NAACL*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *ICLR*.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In *NIPS Workshop*.

Prasoon Goyal, Scott Niekum, and Raymond J. Mooney. 2019. Using Natural Language for Reward Shaping in Reinforcement Learning. In *IJCAI*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-term Memory. In *Neural Computation*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*.

Gregory Kuhlmann, Peter Stone, Raymond Mooney, and Jude Shavlik. 2004. Guiding a Reinforcement Learner with Natural Language Advice: Initial Results in RoboCup Soccer. In *AAAI Workshop*.

Vitaly Kurin, Sebastian Nowozin, Katja Hofmann, Lucas Beyer, and Bastian Leibe. 2017. The Atari Grand Challenge Dataset. In *arXiv:1705.10998*.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous Methods for Deep Reinforcement Learning. In *ICML*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. In *NIPS Workshop*.

Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. 2015. Language Understanding for Text-based Games using Deep Reinforcement Learning. In *EMNLP*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. In *arXiv:1707.06347*.

Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. 2017. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. In *arXiv:1707.08817*.