# Learning To Learn Hangman

**Sayem Khan,**
sayem.eee.kuet@gmail.com

## Introduction

In this project, we utilize a dataset of 250,000 words to train a reinforcement learning (RL) agent, supplemented by a separate set of testing words provided via an API. We aim to develop an RL-driven strategy that enables the agent to exceed a baseline win rate in the game of Hangman. This challenge leverages RL's capabilities in decision-making under uncertainty Sutton & Barto (2018), equipping the agent to handle both familiar training data and novel words during testing. The ultimate goal is to demonstrate that with advanced RL techniques, the agent can significantly outperform traditional approaches, illustrating the adaptability and effectiveness of RL in complex decision-making scenarios.

## Hangman Game as a Reinforcement Learning Problem

### Single-Word Scenario:

In the context of Hangman, each game involves guessing a word by selecting letters one at a time. We define a reinforcement learning problem for a single word as follows:

- **State Space** $\mathcal{S}$: The state at any given time $t$ is defined by $s_t = (W_t, G_t, A_t)$:
  - $W_t$: The current masked word, with correctly guessed letters revealed and others hidden.
  - $G_t$: The letters already guessed.
  - $A_t$: The number of incorrect guesses remaining before the game is lost, with a maximum of 6 attempts.
- **Action Space** $\mathcal{A}$: The set of all letters not yet guessed:
$$\mathcal{A} = \{a \mid a \notin G_t\}$$
- **Transition Dynamics**: The state transition depends on the guessed letter's presence in the word:
$$s_{t+1} = f(s_t, a_t)$$
where $f$ updates $W_t$ and $A_t$ based on whether $a_t$ is correct.
- **Reward Function**: The agent receives a reward based on its guess:
$$R(s_t, a_t) = \begin{cases} +1 & \text{if } a_t \text{ is correct and the game is won} \\ -1 & \text{if } a_t \text{ is incorrect} \end{cases}$$

**Learning Dynamics in RL for Hangman:** Because RL algorithms can learn from low inductive bias Kirk et al. (2023), they can learn effective policies for complex environments like Hangman with minimal prior assumptions about the task structure. The mathematical formulation of the learning process is as follows:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s_0 \sim \mathcal{D}} \left[ \sum_{t=0}^{\infty} \gamma^t R(s_t, \pi(s_t)) \right]$$

Where:

- $\pi^*$ is the optimal policy.
- $\mathbb{E}$ denotes the expectation over the initial state distribution $\mathcal{D}$.
- $\gamma$ is the discount factor, emphasizing the importance of immediate versus future rewards.
- $R(s_t, \pi(s_t))$ is the reward received after taking action $\pi(s_t)$ in state $s_t$.

This framework allows the RL agent to adaptively learn the optimal strategy for guessing the word by minimizing,

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t, a_t} \left[ R(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t | s_t) \right]$$

where $\theta$ represents the parameters of the policy $\pi$.

Ultimately, we aim for the agent to perform well with an entirely new set of words. Embedding task-specific inductive biases into the model achieves better generalization across diverse scenarios. The meta-learning algorithms, like the $RL^2$ algorithm Duan et al. (2016), are well-suited for learning and generalizing the Hangman game across a wide range of words. Due to the absence of a standard implementation, we have utilized the PyTorch implementation from bay3s (2022) for our work.

## Meta Reinforcement Learning

A meta-learning model is designed to adapt to new tasks from potentially unfamiliar environments it has yet to see during training. This adaptation occurs during testing and involves a brief learning phase with minimal exposure to new settings. Remarkably, this process does not require explicit fine-tuning; there's no need for gradient backpropagation on trainable variables. Instead, the model autonomously modifies its internal hidden states to learn. In essence, Meta Reinforcement Learning applies these meta-learning principles within the realm of reinforcement learning Kirk et al. (2023), Weng (2019).

## Formulation of $RL^2$ for Meta-Learning

In the $RL^2$ framework, we enhance traditional reinforcement learning by addressing learning processes across a diverse set of Markov Decision Processes (MDPs). Let $\mathcal{M}$ represent the set of all possible MDPs, where each $M \in \mathcal{M}$ is sampled according to a distribution $\rho_M(M)$.

An MDP defines each task within this framework, noted as $M_i \in \mathcal{M}$, and is characterized by a 4-tuple:

$$M_i = \langle S, A, P_i, R_i \rangle$$

Where:

- $S$ denotes the set of all possible states,
- $A$ represents the set of feasible actions,
- $P_i : S \times A \times S \to \mathbb{R}^+$ defines the transition probability function,
- $R_i : S \times A \to \mathbb{R}$ specifies the reward function.

Building upon the foundation laid by previous research Wang et al. (2016), the $RL^2$ introduces an additional component to the MDP structure to accommodate what is referred to as a finite horizon $\tau$:

$$\tau_t = \langle s_t, a_{t-1}, r_{t-1} \rangle$$

Within which:

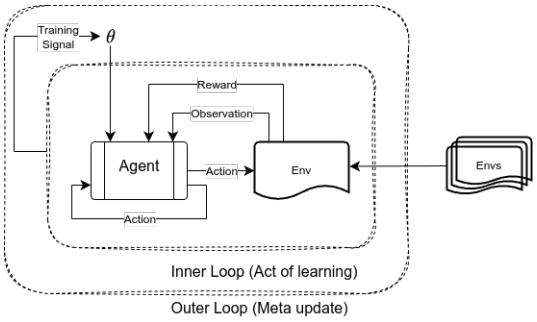


Figure 1: $RL^2$ segments the learning process into two distinct stages: the inner and outer loop. The inner loop focuses on the agent's active learning through direct interactions with the environment, where it updates only the hidden state of the RNN, allowing the RNN to serve as a transient memory that captures context without undergoing gradient updates. Conversely, the outer loop is dedicated to refining the agent's parameters, with gradient updates being conducted using well-established algorithms such as Proximal Policy Optimization (PPO).

- $s_t$ is the observation of the task space at time $t$,
- $a_{t-1}$ is the action taken at the previous timestep,
- $r_{t-1}$ is the reward received from the preceding action.

This schema utilizes a consistent state space $S$ and action space $A$ to facilitate a stochastic policy $\pi_\theta : S \times A \to \mathbb{R}^+$, ensuring that input parameters are uniform across varied tasks. Test tasks are selected from either the original distribution $\mathcal{M}$ or a modified version.

**Meta Episodes and Trials**

- Learning occurs across a series of trials. Each trial $i$ consists of multiple episodes interacting with an MDP $M_i \sim \rho_M(M)$.
- For each trial, the agent is presented with an MDP $M_i = (S_i, A_i, P_i, r_i, \rho_{0,i}, \gamma, T)$, and for each episode within the trial, an initial state $s_0$ is sampled from $\rho_{0,i}$.

**Meta-Learning Objective**

The meta-learning objective in $\text{RL}^2$ is to optimize the policy parameters $\theta$ such that the policy generalizes across the distribution of MDPs. The objective function becomes:

$$\min_\theta \mathbb{E}_{M\sim\rho_M}\left[\mathbb{E}_{\tau\sim\pi_\theta(M)}\left[\sum_{t=0}^{T}\gamma^t r_M(s_t,a_t)\right]\right]$$

Here, $\pi_\theta(M)$ represents the trajectory distribution induced by the policy $\pi_\theta$ on MDP $M$, and the inner expectation captures the expected return within an individual MDP, while the outer expectation averages over the distribution of MDPs.

**Meta-Policy Update**

The policy update for $\text{RL}^2$ is based on the total cumulative return over multiple episodes within a trial. For each trial, the policy update rule is:

$$\theta \leftarrow \theta + \alpha \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t, h_t) R_t$$

Where:

- $h_t$ is the hidden state of the recurrent neural network (RNN) used by the policy to maintain memory over episodes.
- $R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ is the discounted return at time $t$.

**Hidden State Dynamics in $\text{RL}^2$**

The $\text{RL}^2$ algorithm uses an RNN to process sequences of states, actions, and rewards across multiple episodes. The hidden state $h_t$ is updated based on the current state-action pair and the reward signal:

$$h_{t+1} = f(h_t, s_t, a_t, r_t, d_t)$$

Where $d_t$ is a termination flag (1 if the episode terminates, 0 otherwise). The hidden state $h_t$ enables the policy to maintain an internal memory across episodes, allowing for faster adaptation in future episodes within the same trial.

## Experimentation

We trained the meta RL agent on approximately 250,000 words and tested it on a new set of words accessed through an API. For a novel dataset of 1,000 words, the agent achieved a performance rate of 45.9%.

## Future Work

In our future work, we aim to explore sophisticated meta-learning strategies such as VariBAD Zintgraf et al. (2019), which integrates a prior belief about tasks into the learning process.

## References


bay3s (2022), '**RL$^2$: Fast Reinforcement Learning via Slow Reinforcement Learning**', https://github.com/bay3s/rl-squared. Accessed: 2024-09-07. 2

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I. & Abbeel, P. (2016), 'Rl $^2$: Fast reinforcement learning via slow reinforcement learning', *arXiv preprint arXiv:1611.02779* . 2

Kirk, R., Zhang, A., Grefenstette, E. & Rocktäschel, T. (2023), 'A survey of zero-shot generalisation in deep reinforcement learning', *Journal of Artificial Intelligence Research* **76**, 201–264. 1, 2

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*, second edn, The MIT Press.
**URL:** *http://incompleteideas.net/book/the-book-2nd.html* 1

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D. & Botvinick, M. (2016), 'Learning to reinforcement learn', *arXiv preprint arXiv:1611.05763* . 2

Weng, L. (2019), 'Meta reinforcement learning'.
**URL:** *https://lilianweng.github.io/posts/2019-06-23-meta-rl/* 2

Zintgraf, L., Shiarlis, K., Igl, M., Schulze, S., Gal, Y., Hofmann, K. & Whiteson, S. (2019), 'Varibad: A very good method for bayes-adaptive deep rl via meta-learning', *arXiv preprint arXiv:1910.08348* . 4