Taxonomy Construction with Multi-Critic Reinforcement Learning

Bendeguz Toth
6977936

master Artificial Intelligence
Utrecht University

July 14, 2023

Shihan Wang
Injy Sharhan
Supervisor

Pablo Mosteiro Romero
Examiner

Abstract

This thesis investigates the application of multi-critic reinforcement learning to taxonomy construction. Using multiple critics in complicated environments has been shown to improve overall performance on a plethora of reinforcement learning tasks. Guiding the learning process of the critic by explicitly constraining it to a small subset of the task is an effective way to speed up learning and ensure stability during training. Combining several of these constrained critics into a centralized critic outperforms single critic methods on a variety of different tasks. Multi-critic algorithms are especially effective when there is an underlying structure of the task with clearly defined sub-tasks that can be evaluated independently. In this work, we introduce a multi-critic algorithm for taxonomy construction, where we use two critics to evaluate the choice of the parent and child words at each step. Through a series of experiments, we demonstrate that the critics have learned to effectively identify the source of error in incorrect actions, which was not possible with previous methods. We also demonstrate the robustness of our model by analyzing the consistency of the structure of its generated taxonomies.

Contents

1 Introduction

The taxonomy of a domain presents a categorization of the most important concepts and terms within the domain by organizing them according to their *"is-a"* relations [Brachman, 1983]. *P is-a Q* holds for two terms P and Q if P in a given context is a subclass (or a more specific instance) of Q. In this context, P is called the *hyponym*, while Q is the *hypernym*. For example, if we take the domain of our taxonomy to be vehicles, the relation *Ferrari "is-a" car* holds as a Ferrari is a type of car. A full taxonomy of a domain is an acyclic graph where nodes represent terms, and directed edges between them represent *"is-a"* relations.

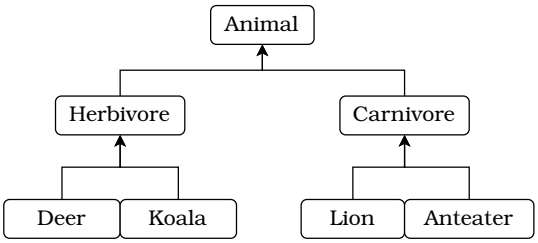

Figure 1: An example (partial) taxonomy.

A well-known use of taxonomies is in biology, where it is used to represent the relations of different species (an example is shown in Figure 1). It is a powerful tool to summarize knowledge and present it in an easy-to-understand way, and as such, it is of great practical importance in its own right. Additionally, it is often used by downstream processes that rely on a background resource for their task. For example, some question-answering methods make use of domain-specific taxonomies for query understanding.

It is, however, not a trivial task to construct high-quality taxonomies from text, especially for domains with a huge number of terms and concepts that need to be incorporated. Manual construction of taxonomies requires a lot of time and manpower and needs to be created by experts in the field to be of tangible value. There have been efforts to hand-craft large taxonomies for a variety of domains, such as WordNet [Miller, 1995]. Even so, it still does not cover every topic that might be of interest.

Because of the challenges of manual taxonomy construction, automating the process has received a lot of attention from researchers. The goal of automatic taxonomy induction is to infer an accurate taxonomy graph from a set of background resources, such as, in the case of biology, a set of articles about different living creatures.

Reinforcement learning has only been applied to taxonomy construction in a limited number of cases [Mao et al., 2018], [Han et al., 2021], and without its full potential. Most notably, there have been no efforts to employ a multi-critic algorithm that can take advantage of the hierarchical nature of the task. The goal of this thesis is to investigate the viability of multi-critic agents and demonstrate that there is potential to improve upon existing models by improved credit assignment.

In Section 2, we give an overview of the current state of the field of taxonomy

construction, as well as introduce some relevant reinforcement learning concepts and present a literature overview of multi-critic methods. In Section 3, we analyze and give a formal definition of the problem. In Section 4, we motivate and define our solution to the proposed problem. Section 5 showcases the experiments and results of our model.

2 Background and related work

In this section, we give an overview of the current state of taxonomy induction, as well as introduce basic concepts of reinforcement learning, followed by an explanation of actor-critic methods.

2.1 Taxonomy construction

In this section, we present an overview of different approaches and methods to taxonomy construction. The task is usually divided into two subtasks [Wang et al., 2017] that are performed independently.

- **Determining *"is-a"* relations** The first task is determining *"is-a"* relations between pairs of terms. Different combinations of candidate words are tested to determine the relations between them. This process might be aided by a background corpus. A background corpus is a large collection of texts with the same domain as the taxonomy. This allows the model to look for domain-specific relations between different terms. For example, if we are making a taxonomy of cooking ingredients, *ginger* might fall under the category *condiment*. However, if we are cataloging living organisms, it would be a type of *plant* (yet there is no *is-a* relation between plants and condiments). As the example above illustrates, the specific domain has a great influence on what relations might be considered valid.
- **Hierarchy construction** The second subtask is equally challenging: It is concerned with assembling a valid taxonomy from the pairs of terms found during phase one. The difficulty of this task does not only come from the fact that the generated nodes and edges need to respect the transitive property (if a "is-a" b and b "is-a" c then also a "is-a" c), but it is also important to make sure that the final graph is acyclic and has a single root node that is reachable from all nodes of the graph. To correctly predict the structure of the target taxonomy, it is again important to have access to a background corpus which dependencies between words can be extracted from.

In the remainder of this section, we present an organized overview of methods for both of these subtasks, briefly discussing their strengths and weaknesses.

2.1.1 Term-pair extraction

Methods of term-pair extraction (finding pairs of words related by "is-a" relation) can generally be divided into two groups based on their approach: Pattern-based and statistical methods.

Pattern-based methods Pattern-based methods rely on matching a large number of hand-crafted syntax-level patterns to extract word relations from text. The first pattern-based method made use of the Hearst patterns [Hearst, 1992], which is a collection of

lexico-syntactic patterns for recognizing hyponymy relations in text. An example of such a pattern is

$$NP_0 \text{ such as } \{NP_1, NP_2, ...(\text{and}|\text{or})\}NP_n$$

This pattern describes the syntax of an expression. Here NP_0 is the hypernym (NP stands for *noun phrase*, which is either a noun, or a phrase that has the same grammatical role as a noun, for example *the red dog*), and NP_1 to NP_n are its hyponyms. The pattern asserts that if in the source text there is a sequence of a noun phrase followed by "*such as*", and then a list of other noun phrases, possibly with an "*and*" or an "*or*" before the last one, we conclude that the noun phrase at the beginning is the hypernym of all the other phrases. For example, the sentence *"Trees such as oak and pine..."* matches this pattern. With a large collection of these patterns, it is possible to mine *"is-a"* relations from background corpora. Hearst patterns have been successfully used to construct taxonomies from websites, such as *Probase* [Wu et al., 2012], which extracts term pairs from web texts.

Pattern-based approaches are, however, extremely inefficient at finding hypernym relations while also being time-consuming. Since the patterns are fully syntactic, they only produce results if there is a perfect syntax level match. Due to the versatile and hard-to-capture nature of human languages, most of the sentences semantically expressing *"is-a"* relations have different syntactic forms, resulting in a low recall [Wu et al., 2012] due to missing out on a lot of valid relations just because they happened to not have the exact form prescribed by the patterns. An additional drawback of the purely pattern-based approaches is that it takes a lot of time and manpower to manually create the patterns.

Over the years, there have been several attempts at mitigating the drawbacks of pattern-based methods by making them more general and less reliant on syntax-level features. [Luu et al., 2014] enables the identification of term pair relations based on the context of the words, allowing for the detection of pairs that do not appear in the same sentence and, as such, would not have been picked up by traditional pattern matching. [Snow et al., 2004] enriches the patterns with dependency path information between the two terms. In their method, the model starts with a number of known hypernym pairs, which are used to extract dependency paths from the background text, and then applies these when detecting new, unknown pairs.

Statistical methods The second class of methods for term pair extraction is statistical-based. Instead of manually specifying the syntactical properties of such relations, those are derived from the statistics of the background text instead.

A simple statistical-based approach is to train a classifier on a set of candidate hypernym-hyponym pairs, for which it is known whether the relation holds. After training, the classifier is able to generalize and can be used to find pairs among previously unseen words. In the most basic case, the feature representation of the two terms is the word vector, such as GloVe [Pennington et al., 2014], or Word2vec [Mikolov et al., 2013]. In some variations, the vector difference of the embeddings is used instead to abstract away from the actual words and focus more on how they relate to each other instead.

A different approach for finding hypernym/hyponym pairs is to look at the context in which the words occur and infer relations based on how similar those contexts are [Geffet and Dagan, 2005]. The underlying assumption here is that a hyponym appears in a subset of the context of the hypernym, while the hypernym appears in all contexts of the hyponym (e.g., the hypernym is more general). Similar methods have also been developed where not only the context words are included but also their syntactic relation [Lin et al., 1998].

[Fu et al., 2014] rely explicitly on the spatial properties of pre-trained word embeddings. Dense word embedding such as Word2vec or GloVe have been shown to retain semantic relations of words in the vector space [Church, 2017]. The authors try to take advantage of this to find hypernym/hyponym relations by learning a projection matrix from hyponyms to hypernyms. Not all pairs share the same linear relationship, but they generally tend to cluster. As such, a separate projection matrix is learned for each cluster.

2.1.2 Taxonomy induction

The second phase of taxonomy construction is building up the graph from a set of given hypernym/hyponym pairs. This can either mean fully constructing a taxonomy from scratch or extending an already existing taxonomy with new terms.

Iterative construction The iterative approach is by far the most popular method that is used almost exclusively in prior work. The general framework starts from some seed taxonomy, which can be either an *empty* or some pre-existing taxonomy, then iteratively extends it, appending new child terms to its nodes where applicable. When inserting new terms, those, of course, need to be related to a term that is already in the taxonomy. In other words, a pair (w_1, w_2) from the set of hypernym/hyponym pairs (which is the input data for the second phase) needs to have either w_1 or w_2 in the taxonomy already in order to be applicable. Since the hypernym relation is not symmetrical, it leads to two possibilities: Either the parent (hypernym) node is already in the taxonomy, in which case its pair can be added as a child node, or the hyponym (child) node is already contained within the taxonomy, in which case its parent need to be added. The second case, however, is a lot more complicated to handle due to the graph structure of the taxonomy. As it is a tree with a single *root*, all nodes that are part of the tree by definition already have a parent node, making it not trivial how a new parent node can be added for an arbitrary child node. Because of this limitation, most methods only allow for the insertion of a child node into an existing parent node and not the other way around. Revisiting the previous example from above, Figure 2 shows how such an extension step is done on a partial taxonomy.

Other than direct *"is-a"* relations, other kinds of indirect relations can also be used to determine whether a given term can be inserted into a taxonomy. [Snow et al., 2006] iteratively extend WordNet taxonomies by considering both *"is-a"* and *cousin* relations. Two terms w_1 and w_2 are (n, m)-cousins if their closest common ancestor is within n and m edges, respectively. For example, if *Lion* were to be added as a child to *Carnivore*, *Koala* and *Lion* would be (2, 2) cousins, as their closest common ancestor (*Animal*)

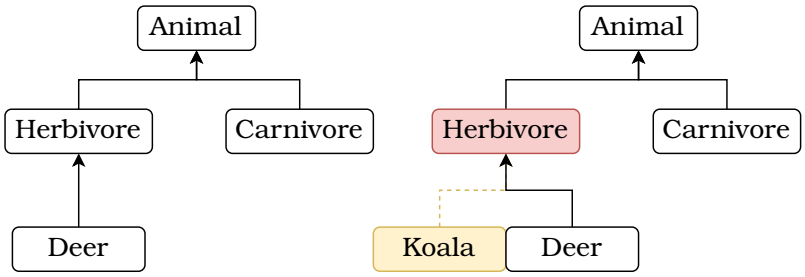

Figure 2: Iterative taxonomy extension. **Left**: The partial taxonomy at arbitrary time step t during the construction process. **Right**: The taxonomy is extended by first selecting a parent node (red), then adding a child node from the term set (yellow). The taxonomy at time step $t+1$ will look like the second taxonomy on the figure.

is 2 edges away from both. By not only considering 1-step relations but also longer dependencies, their method helps to keep the overall structure of the taxonomy more consistent.

For certain more specific domains, it might be difficult to find a suitable root taxonomy to extend, either because the quality is not up to standards or because no taxonomy has been constructed yet. In such cases, construction from scratch is the only option. [Kozareva and Hovy, 2010] start induction from a single *root* concept, which is a general, high-level concept of the given field that gets iteratively expanded into sub-concepts as the taxonomy is being built up.

2.2 Reinforcement learning

In this section, we introduce the preliminaries of reinforcement learning techniques, as well as several classical types of reinforcement learning algorithms.

2.2.1 Preliminaries

Reinforcement learning (RL) [Sutton and Barto, 2018] is a field of machine learning studying the sequential decision-making of an agent in an environment that may be altered by the actions of the agent. A reinforcement learning task can be formally described as a Markov decision process (MDP) [Bellman, 1957]. An MDP is a tuple (S, A, R, P) where S is a set of possible states, A is the set of possible actions, $R : S \times A \times S \to \mathbb{R}$ is the expected immediate reward when transitioning from state s to state s' when taking action a. P described the dynamics of the environment, with $P : S \times A \times S \to \mathbb{R}$ specifying the probability of transitioning from state s to state s' when taking action a. For example, $p(s', r|s, a)$ denotes the probability of transitioning to state s' and receiving reward r after taking action a in state s.

The interaction between the agent and the environment is as follows: At each time step, the agent is presented with the current state $s \in S$, based on which an action $a \in A$ is chosen. Then the next state $s' \in S$ is sampled according to the internal dynamics of

the environment $P(s'|s,a)$, along with a reward for the last time step $r \in \mathbb{R}$. There are two fundamentally different flavors of environments. If there is a final time step, after which no action can be chosen, the task is said to be episodic. If there is no final state, and the agent can continue indefinitely, the environment is said to be continuing.

A *policy* is a (possibly probabilistic) mapping from state to action: $\pi(a|s)$ is the probability of taking action a in state s under policy π. The goal of the learning algorithm is to find the optimal policy π^* that maximizes the (discounted) cumulative reward. Since the cumulative reward might be infinite in the case of a continuing task, a discount factor (commonly denoted as γ) is applied to make sure it is bounded [Sutton and Barto, 2018].

2.2.2 Tabular methods

To learn the optimal policy, value-based methods estimate the expected cumulative reward for each action in all states. The optimal policy will then be to pick the highest-value action in each state. The problem then effectively becomes learning expected return values for each possible state-action pair. The value $q(s,a)$ of action a in state s is the expected discounted return $R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$ where R_t is the expected reward after time step t. It can be rewritten in a recursive form, where the value of an action in a state is expressed as the intermediate reward and the discounted expected reward of the remaining steps.

$$q_\pi(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \sum_{a'} \pi(a'|s) q_\pi(s',a')) \tag{1}$$

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)(r + \gamma \max_{a'} q_*(s',a')) \tag{2}$$

$q(s,a)$ is the action-value function, and Equation 1 is called the Bellman estimation equation, and it describes the value of an action under policy π in a recursive fashion. Equation 2 is the Bellman optimality equation [Bellman, 1957]. It describes the value of an action under the optimal policy [Sutton and Barto, 2018].

One of the most popular algorithms that take advantage of this relationship is q-learning [Watkins and Dayan, 1992]. The agent approximates the optimal action-value function by using Equation 3 as an update rule:

$$q_t(s,a) = q_{t-1}(s,a) + \alpha(r_t + \gamma \max_{a'}[q_t(s',a')] - q_{t-1}(s,a)) \tag{3}$$

Updating the action-value estimates at each time step using this rule eventually converges to the true values. Hence acting greedily with respect to Q is the optimal policy.

2.2.3 Value function approximation

There are some practical problems with q-learning, however. Most notably, it needs to maintain an estimation for each state-action pair separately, increasing memory requirements. In fact, most real-life environments are way too large to be learned this

way. Another problem with this approach is that there is no generalization between states. Each state has to be learned independently, resulting in slow learning as the state space gets bigger. In practice it is often the case that the action-values of similar states are highly correlated. For example, if the goal is for the robot to move to a target location, then displacing it one millimeter will most likely have only a minor effect on its optimal behavior, yet q-learning would need to learn the optimal action for the new state from scratch.

To mitigate the problems outlined above, a technique called value function approximation is used. Instead of maintaining the estimated action values for each state independently, a function is learned that produces those values. This simultaneously solves the memory problem (the value function tends to have a lot fewer parameters than there are state-action pairs) and also enables generalization between similar states. The general form of the action-value function is $q_\mu(s, a)$, which returns the value of action a in state s with learnable parameters μ. The most popular choice of function approximators is neural networks.

DQN Deep q-learning (DQN) [Mnih et al., 2015] is an extension of q-learning where instead of storing the values of each state-action pair individually, the lookup table is approximated by a neural network. The value of action a in state s under a function with parameters μ is denoted as

$$q_\mu(s, a) \tag{4}$$

A pseudo-code of this algorithm is shown on Algorithm 1.

Algorithm 1: DQN

Initialize μ randomly;
Set the learning rate α to an appropriate value;
for *each episode in epoch* **do**
  for *each step in episode* **do**
    Choose action a based on values of q_μ;
    Observe next state s', get reward r;
    target value $\leftarrow r + \gamma \max_{a'}[q_\mu(s', a')]$;
    $L \leftarrow (\text{target value} - q_\mu(s, a))^2$; `/* Calculate loss */`
    $\mu \leftarrow \mu + \alpha \nabla_\mu L$; `/* Update the parameters */`
  end
end

2.2.4 Policy gradient methods

While the previously discussed methods obtain their policy by learning the action value function, there is also a different family of algorithms called the policy gradient (PG) methods [Sutton et al., 1999]. The main idea of policy gradient methods is that it is possible to represent and thus learn the policy explicitly instead of implicitly deriving it

from the action values. While value-based methods learn to evaluate each action, then choose the one with the highest value, PG methods learn a policy function that outputs probabilities for each action.

$$\begin{aligned} &\pi(s) \text{ is a probability distribution over all possible actions in state } s \\ &\pi(a|s) \text{ denotes the probability of choosing action } a \text{ in state } s \end{aligned} \tag{5}$$

One advantage of this approach is that, in practice, it is often easier to learn a good policy than it is to learn an accurate value function. For example, if an agent is learning to play *pacman*, learning to go left when there is a ghost on the right might be easier to understand for the model than calculating the exact score in case it decides to go in that direction. Another benefit of representing a policy directly instead of implicitly choosing actions based on value estimates is that it makes it possible to represent stochastic policies. In some cases, it might not be beneficial to deterministically choose actions. For example, a deterministic policy would not work very well for playing *rock-paper-scissors*, where the optimal policy is, in fact, stochastic.

Neural networks are a popular choice for policy functions, as they are universal function approximators [Hornik et al., 1989]. If a policy is expressed by a neural network with learnable parameters θ, the probability of action a in state s is denoted as

$$\pi_\theta(a|s) \tag{6}$$

REINFORCE REINFORCE [Williams, 1992] is the most simplistic policy gradient algorithm. During training, it simulates an episode to the end (this method is only applicable for episodic environments) and stores all the transitions ((s, a, r, s') (state, action, reward, new state) tuples) in a buffer. The training takes place after the episode finishes. During training, for the transition at time step t, it calculates v_t, which is the discounted return starting from t.

$$v_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} ...$$

Then for each step, the policy parameters θ are updated in proportion to the log value of the network output for the state-action pair times v_t.

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta log \pi_\theta(s_t, a_t) v_t \tag{7}$$

This is a robust but slow learning algorithm, not commonly used in practice because of its low speed of convergence.

Actor-Critic While REINFORCE is a stable algorithm that is guaranteed to converge, it is, in most cases, not applicable to real-world scenarios because it is extremely sample inefficient and takes an excessive amount of time and data to train. The reason for this is that the return values v_t that are used at each time step have an extremely high variance. In more complex environments where episodes last several hundreds or even thousands of time steps, the sample returns can vary significantly between runs. This high variance causes the algorithm to converge very slowly. A solution to this problem is *actor-critic* methods [Konda and Tsitsiklis, 1999]. An actor-critic algorithm combines

the ideas of function approximation and policy gradients. Instead of training the policy towards an empirical sample of the returns, a function approximator is used to learn the expected value of each action. Using this estimated value instead of the true returns reduces variance significantly (because the value of an action does not vary for each run), which leads to much quicker convergence compared to algorithms that do not take advantage of value function approximation, such as REINFORCE.

An actor-critic agent consists of two parts: the policy network (the actor) π parameterized by θ and an action value network (the critic) q parameterized by μ. These are trained in parallel. The training process is illustrated by Algorithm 2.

Algorithm 2: Actor-Critic

Initialize θ and μ randomly;
Set the learning rates of the actor (α_a) and critic (α_c) to an appropriate value;
for *each episode in epoch* **do**
  for *each step in episode* **do**
    Sample action $a \sim \pi_\theta(a|s)$;
    Observe next state s', get reward r;
    Sample action $a' \sim \pi_\theta(a'|s')$;
    $L_c = (r + \gamma q_\mu(s', a') - q_\mu(s, a))^2$;
    $L_a = \ln(\pi_\theta(s, a))q_\mu(s, a)$;
    $\mu \leftarrow \mu + \alpha_c \nabla_\mu L_c$;
    $\theta \leftarrow \theta + \alpha_a \nabla_\theta L_a$;
  end
end

Multi-critic algorithms In complicated environments, it can be difficult for a single critic to quickly and effectively learn to estimate action values. In these cases, it can be beneficial to introduce multiple critics and split up the responsibilities between them [Martinez-Piazuelo et al., 2020], [Yang et al., 2018], [Mysore et al., 2021]. Introducing a smaller area of focus eases the load on any individual critics.

The family of multi-critic algorithms is a diverse collection of algorithms centered around the idea of introducing some kind of structure into the learning procedure to help guide the attention of critics toward certain parts of the feature space. There are multiple ways explored for structuring the critics. [Mysore et al., 2021] propose the use of different critics for different tasks in the same environment. Sometimes an agent needs to be able to behave in different ways in one environment depending on its instructions. In their example, there is a modified *pong* game, where the paddle can also be rotated in addition to moving up and down. In this environment, the agent is tasked with learning two different styles of play. It can play either aggressively, trying to hit the ball with as much power as possible, or defensively, where it slows the ball down. Their results demonstrate that using 2 distinct critics for the different tasks outperforms the use of a single critic that has to be able to tell which mode the agent is

in. This result demonstrates that splitting up the responsibilities of critics is an effective way to improve overall performance.

FACMAC [Peng et al., 2021] is a multi-agent actor-critic system that introduces factorized critics, with their output being combined into a single final value by a centralized critic. The contribution of their work is that combining the critics allows for a centralized policy update rather than having to optimize the policy of each agent independently. As a result, policies where the individual agents effectively work together are more emergent compared to when the update happened in isolation. [Peng et al., 2021] demonstrate that having a potentially large number of independent critics can still lead to overall improvement.

[Wu et al., 2018] analyze the strengths and weakness of DDPG[Lillicrap et al., 2015], and conclude that the method suffers from a sensitive critic, which harms its learning capabilities. To solve this problem, they propose using multiple independent critics and averaging their outputs to obtain a more stable score. This application of multiple critics is different from the previous ones in that the critics do not have distinct roles.

[Martinez-Piazuelo et al., 2020] explore the multi-critic idea for controlling multi-tank water systems. They propose the partitioning of the loss function. If the loss is linearly separable, it can be broken down into several loss partitions, where each partition represents some well-defined part of the cost. For example, a cost partition might be responsible for keeping the temperature of water in a given tank within some defined range. Another partition might pay attention to large switches in control input and gives a penalty if the values of certain inputs change too rapidly. Combining all these different cost partitions results in the final cost. They show that using multiple critics where each critic is assigned to a cost partition leads to a better performance on the task than using only a single critic.

Integrating multiple critics into the algorithm is a powerful tool that can lead to substantial improvements on a variety of different tasks with different model architectures. Taking advantage of the natural structure of the task in a way to limit the scope of a critic leads to more efficient learning on the simpler task. [Peng et al., 2021] also demonstrated that this performance gain is not lost in cases where the final value is a combination of a large number of individual critics. Multi-critic learning is a versatile and powerful paradigm that can benefit an array of different tasks.

2.3 Reinforcement learning for taxonomy construction

There is very limited work in applying reinforcement learning for taxonomy construction. In this section, we describe 2 reinforcement learning agents that managed to achieve a competitive performance on the benchmarks, TaxoRL [Mao et al., 2018] and built upon it DTaxa [Han et al., 2021].

Most algorithms for taxonomy induction see the task as a two-phase process: Finding related term pairs is handled separately from building up a tree from those pairs. The idea behind TaxoRL is that this setup is inherently suboptimal, as it results in a one-directional information flow between these two stages. While during phase 2, the algorithm is able to rely on the result of the first phase, it is not the case the other way around. This means that term pairs are identified independently from the (preliminary)

structure of the taxonomy, which leads to worse performance. The main contribution of the paper by [Mao et al., 2018] is to introduce a method that unifies the two distinct phases into a single one, allowing for hypernym/hyponym pairs to be found with a preliminary taxonomy in mind.

The algorithm starts out with an empty taxonomy containing only a *root* node. Then at each time step, a word is chosen from the provided term set and is attached to one of the nodes in the taxonomy as a child. When choosing the word to add at time $t+1$, the current taxonomy at time t is available to aid decision-making. A reinforcement learning agent is trained to learn to choose the optimal words and parents at each time step.

More formally, the state representation contains the current taxonomy, as well as the set of remaining words that still need to be added. Taking an action at time t involves selecting a node from the taxonomy at time $t-1$ and a new term to append from the set of remaining words. Therefore an action is a tuple $(parent, children)$ with $(|\text{TAXONOMY}_{t-1}| \times |\text{REMAINING WORDS}_{t-1}|)$ number of actions at time t (meaning that the number of actions differs at each time step). An illustration of this process is shown in Figure 3.

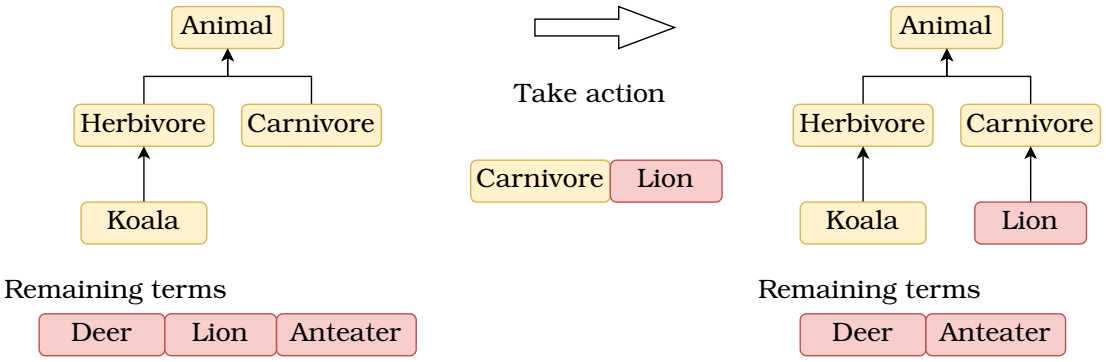

Figure 3: An illustrated example of taking an action. **Left**: The taxonomy at time step t, along with the set of terms that are not yet included (in red). The action is a tuple of a word from the taxonomy (yellow), and a word from the set of remaining terms. **Middle**: The chosen action in this case is the tuple (Carnivore, Lion). The first word will be used as the parent node of the second. **Right**: The updated taxonomy at time $t+1$, with the word from the action attached to its chosen parent node.

During training, the goal of the RL agent is to reproduce the golden taxonomy. Such a binary condition, however (where the reward is 1 for successfully completing the task and 0 otherwise), is generally bad for learning, as it can take a very long time until the agent happens to precisely reconstruct the exact target taxonomy for the first time. To mitigate this, a smooth metric of policy quality is employed in the form of the *Edge-F1* score. It is the F1-score of the *"is-a"* relations that are present in the taxonomy. It is calculated much like the traditional F1-score, except using *edge precision* and *edge recall* where edge precision is the number of edges correctly predicted divided by the number of total predicted edges, and recall is the number of edges correctly predicted

divided by the number of total correct edges. To measure improvement at each time step, the reward at time t is defined as the difference in F1 score between the previous and the current time step: $R_t = F1_{t-1} - F1_t$

Using the state, action, and reward definitions mentioned above, TaxoRL trained a REINFORCE agent. The first task it was evaluated on is end-to-end taxonomy construction, with the dataset containing over 700 examples extracted from WordNet [Bansal et al., 2014]. TaxoRL outperforms the current state of the art on this benchmark. The second experiment is about hypernym organization, using the SemEval-2016 task 13 dataset. TaxoRL significantly outperforms the previous state-of-the-art method on this task.

DTaxa takes the idea of using a reinforcement learning agent to unify the two phases of taxonomy construction one step further. They propose replacing the REINFORCE agent of TaxoRL with a variant of the DDPG agent [Lillicrap et al., 2015], which reduces the variance under training, and leads to faster learning and a better policy.

3 Problem statement

In this section, we describe and analyze the problem statement and give a motivation for the choice of the proposed algorithm.

3.1 Problem analysis

A weakness of the methods discussed in Section 2, a characteristic shared by both TaxoRL and DTaxa, is the structure of their action representations. To elaborate, when an action is made, it is decided which term from the remaining term set shall be added to the taxonomy and also at which position. In essence, an action consists of choosing 2 nodes to create an edge between them. Such an action is built up from 2 inherently different parts, yet it is handled as a single action by both models. We hypothesize that performance improvements can be achieved by modifying the way actions are handled to better match the structure and semantics of the underlying problem.

In order for a single action to be correct, or at least to make sense to an extent where meaningful learning is possible, the two parts of the action (the newly chosen child, as well as the parent node) need to be simultaneously correct. The action can either be correct or not as a whole, regardless of individual components. In some cases, however, one of the sub-actions (we refer to the choice of either one of the terms as a 'sub-action', while the complete action refers to the choice of both the child and parent terms) might be correct considering the context of the taxonomy being built. This leads us to the problem of credit assignment. Credit assignment refers to the process of identifying the cause of a certain outcome [Minsky, 1961]. For example, if the chosen action is incorrect, we can pinpoint the exact component of the system or model that the error originates from. Without proper credit assignment, the model is left in the dark about what exactly went wrong. With proper credit assignment, however, the root cause of errors can be found and corrected more quickly. This, in turn leads to faster learning and better final performance. To illustrate how credit assignment is relevant for our case, we refer to Figure 4. A small hypothetical taxonomy on the domain of

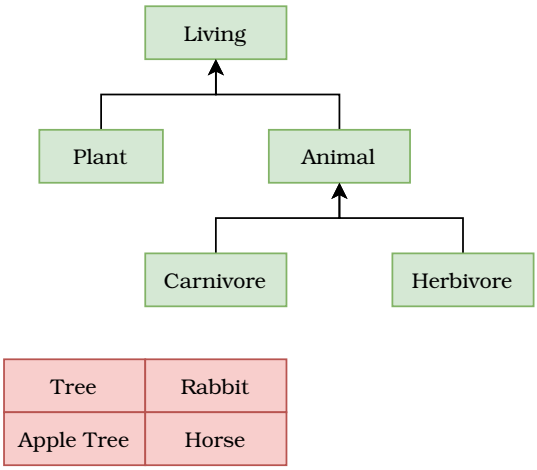

Figure 4: Example taxonomy construction task. The already constructed taxonomy (green) and the elements of the remaining term set (red).

biology is shown, with 5 nodes being already included in the taxonomy (in green) and the set of remaining 4 terms that need to be added in subsequent time steps (in red). We illustrate the problems related to the lack of proper credit assignment by discussing two possible actions chosen at this time step. In Figure 5, one of the potential actions

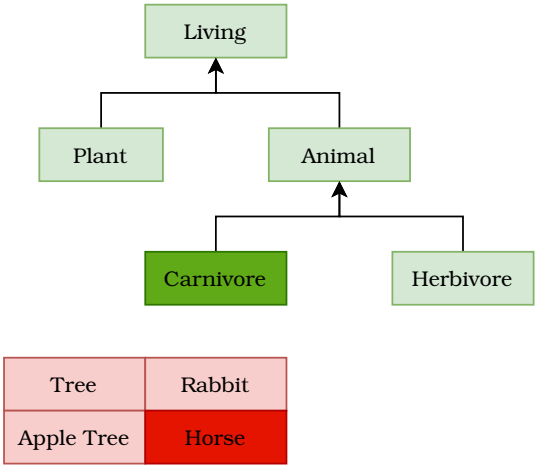

Figure 5: Example of action (*horse*, *carnivore*)

(*horse*, *carnivore*) is shown, with the relevant nodes shaded darker. Of course, the given action is incorrect, as horses are not carnivores, so it is safe to mark the action as a whole wrong. However, without proper credit assignment, the model will not be able to

tell the difference between the effect of each of the nodes on the outcome of the action as a whole. Where does the error come from? Is it caused primarily by choosing the wrong parent, or by choosing the wrong child, or maybe both? By inspecting the taxonomy and the term set of the example, we can see that among the four possible child nodes there are two that belong in the category *plant* and another two that belong in the category *herbivore*, that there is no possible child node for the category *carnivore*. Based on this insight, it can be concluded that the choice of parent (*carnivore*) is part of the action that should take the blame for the incorrect action, as there are no candidate child nodes that result in a correct action with that parent node. The child choice, however is not necessarily incorrect, as there is a node in the taxonomy that can act as its parent. Another example is shown in Figure 6, where the two sub-actions are *plant* and *horse*

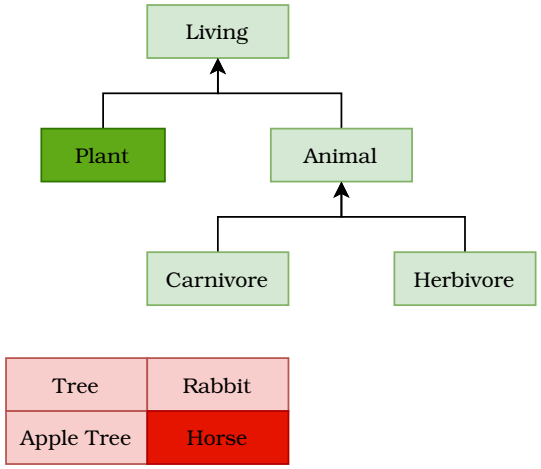

Figure 6: Example of action (*horse, plant*)

instead. In this particular case, the blame cannot be assigned exclusively to a single party, as both sub-actions could make sense in theory, in combination with a matching node selection for other sub-action.

A major drawback of previous designs, including TaxoRL and DTaxa, is that there is no proper distinction between the different ways actions can be incorrect. Conversely, it is also possible that from the remaining term set, such a child node is chosen for which the parent is not yet added to the taxonomy tree, therefore making all possible parent choices incorrect. Figure 7 depicts such a situation. Here the choice of *apple tree* as the child should get most of the blame, as its parent, *tree*, can not be chosen.

The parent choice of *plant* was in fact not even a bad decision considering the possibilities, yet without proper credit assignment, both parts of the action would be seen as equally bad.

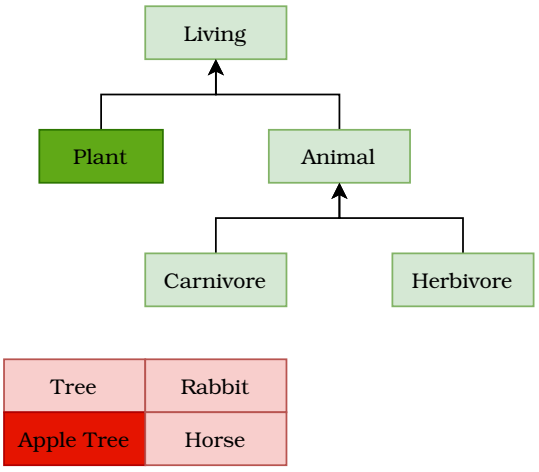

Figure 7: Example of action (*apple tree, plant*)

3.2 Problem formalization

In this section, we formalize the taxonomy construction task as a reinforcement learning problem. To this end, we first give a high-level description of the iterative taxonomy construction process, followed by a detailed definition of the corresponding MDP.

Given a set of terms to be organized, the goal is to create a taxonomy containing words from the term set that matches the *golden taxonomy*, which is the ground truth, a taxonomy that is known to be correct. The model is provided with a large background corpus that can be used to extract information about the relations of words and incorporate that information as features in its action representation.

We take an iterative approach to taxonomy construction, where at time t_0 we start with a taxonomy tree containing only a single term, that we then iteratively extend at each time step by appending a word from the term set as a child to one of its nodes until all the terms are added. Below we give a formal definition of the MDP.

Action At each time step t_n, there is a set of words that are nodes in the taxonomy tree U_t, a set of remaining terms that are not yet part of the tree V_t and a set of edge E where the edges connect two nodes in the taxonomy: $E_t : \{(V \times V)\}$. Furthermore, there is a root node $\text{ROOT}_t \in U$, which is the root of the taxonomy tree.

To give a concrete example of the taxonomy shown in Figure 7, we have the following values:

$$\begin{aligned} U_t &= \{\text{Living, Plant, Animal, Carnivore, Herbivore}\} \\ V_t &= \{\text{Tree, Rabbit, Apple Tree, Horse}\} \\ E_t &= \{\text{(Plant, Living), (Animal, Living), (Carnivore, Animal), (Herbivore, Animal)}\} \\ \text{ROOT}_t &= \text{Living} \end{aligned} \tag{8}$$

An action can be one of two types:

- **Adding a new node as a child** In this case the action a_t has the form $(v, u) \in (V_t \times U_t)$. The new root v is added to the taxonomy as a child node to u. The update to the taxonomy at time step $t+1$ is given by

$$\begin{aligned} U_{t+1} &= U_t \cup \{v\} \\ V_{t+1} &= V_t \setminus \{v\} \\ E_{t+1} &= E_t \cup \{(v, u)\} \\ \text{ROOT}_{t+1} &= \text{ROOT}_t \end{aligned} \tag{9}$$

- **Adding a new node as root** It is also possible to choose the current root as the child and append a new term as its parent (with it becoming the new root). This action a_t has the form (ROOT_t, v) where $v \in V_t$. The update to the taxonomy at time step $t+1$ is given by

$$\begin{aligned} U_{t+1} &= U_t \cup \{v\} \\ V_{t+1} &= V_t \setminus \{v\} \\ E_{t+1} &= E_t \cup \{(\text{ROOT}_t, v)\} \\ \text{ROOT}_{t+1} &= v \end{aligned} \tag{10}$$

Combining those two action possibilities, an action has the following form:

$$a_t \in (V_t \times U_t) \cup (\{\text{ROOT}_t\} \times V_t) \tag{11}$$

State For this MDP, the state s_t at any time step t represents the taxonomy at time t. A taxonomy tree in this task is represented as a collection of edges, as well as the remaining term set V. Note that there is no need to explicitly include the terms that are already part of the tree (the nodes) as they are implicitly given by the edges. The definition of the state is given below:

$$s_t = (E_t, V_t) \tag{12}$$

To give a concrete example of the notion of a state, we refer to Figure 7. The current state of the taxonomy is given by

$$\begin{aligned} E &= \{\text{(Plant, Living), (Animal, Living), (Carnivore, Animal), (Herbivore, Animal)}\} \\ V &= \{\text{Tree, Rabbit, Apple Tree, Horse}\} \\ s &= (E, V) \end{aligned} \tag{13}$$

Reward Similarly to [Mao et al., 2018] and [Han et al., 2021], we use the difference in *Edge-F1* at each time step as the reward signal. *Edge-F1* is defined as

$$
\begin{aligned}
P &= \frac{|\overline{E} \cap E^*|}{|\overline{E}|} \\
R &= \frac{|\overline{E} \cap E^*|}{|E^*|} \\
F_1 &= \frac{2 \cdot P \cdot R}{P + R}
\end{aligned}
\tag{14}
$$

where E^* is the set of edges present in the golden taxonomy and $\overline{E}$ is the set of edges predicted by the model. The reward at time step t is then $F_1^t - F_1^{t-1}$.

4 Methodology

Our proposed solution to mitigate the lack of proper credit assignment in previous methods is to introduce a multi-critic algorithm that separately evaluates both sub-actions. Instead of having a single critic to predict the value of an action, in this thesis we propose an algorithm that makes use of 2 different critics, one for each sub-action. This approach allows for assigning blame to sub-actions independent from each other, leading to better credit assignment.

In our design, we aim to integrate the idea of using multiple critics into the domain of taxonomy construction. In our model, there are 2 sub-critics, each looking at only a part of the features, with their output being combined by a top-level critic into a single estimate. More specifically, one of the critics is responsible for rating the choice of the parent node independently of the child node, while the other rates the choice of the child node independently of the parent node. This design allows for the sub-critics to be independent and makes it possible to easily optimize the model by backpropagating only a single time from the final value of the combined critic. The actor, on the other hand, is not split similarly but is represented by a single network that works on the whole feature vector. An overview of the whole model is shown in Figure 8.

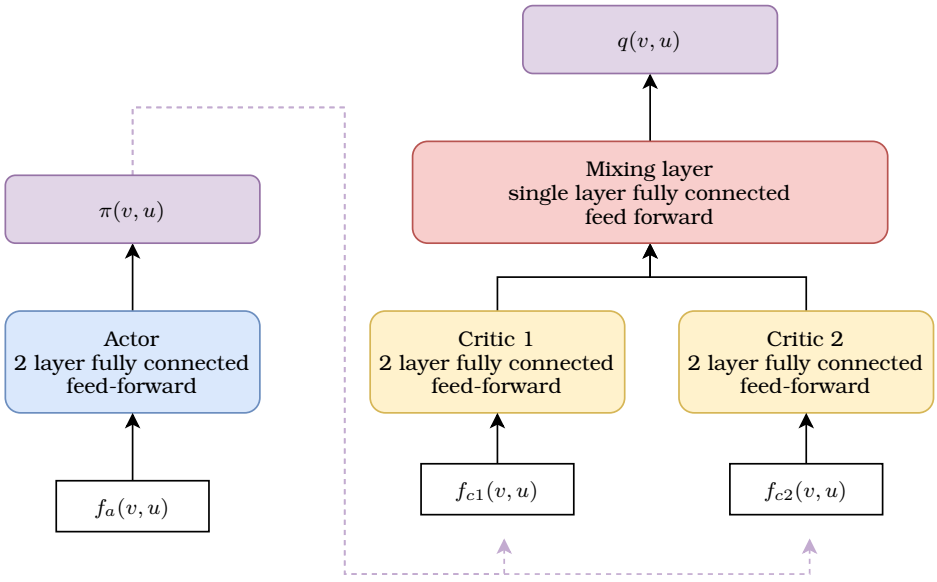

Figure 8: An overview of the model. f_a and f_c are the actor and critic feature vectors respectively, defined in Section 4.1.1 and 4.2.1. **Left** is the actor, a 2-layer fully connected feed-forward neural network that takes the state and action encodings as input and outputs the log probability of taking the action. **Right** is the design of the critic network, with 2 sub-critics and a mixing layer. One of the critics evaluates the child choice, while the other the parent choice of the action. The mixing layer combines those results and produces the final action value.

In the following section, we describe the design of both the actor and the critic component in more detail.

4.1 Actor

The policy network is a fully-connected feed-forward neural network. One of the challenges of designing the actor architecture for the taxonomy construction task is that it has to be able to handle a dynamic actions space. Agents that are trained to play Atari games [Mnih et al., 2015] or to control a robot arm [Franceschetti et al., 2021] have one key property in common: The number of actions remains unchanged throughout the completion of their task. This is, however, not the case for taxonomy construction. The actions are defined by how many terms there are left to be added to the tree, as well as how many nodes there are currently in the taxonomy. These are dynamic quantities that change for each time step. Because of this, the standard architecture where the input to the network is the state representation and the output is a probability distribution over the possible actions is not possible. Instead, the input will be the state and the action, and the output is the probability of taking that action in that state. This way, the network can be used for an arbitrary number of actions.

In addition to this, there is also the challenge of change in action semantics between different episodes, which is normally not the case in most domains. For example, if a robot is trained to walk by rotating its joints by some degree, then each action has a constant role across multiple episodes. If the action corresponding to the first output value of the network rotates a certain joint in the leg in one episode, it will also rotate the same joint in the next run. For the domain of taxonomy construction, this consistency does not apply. If we first build a taxonomy for the animal kingdom, then another one for different kinds of mining equipment, all the actions would have a completely different semantic meaning, even if the size of the action space is the same. In other words, the action corresponding to the first output value will probably have an entirely different meaning in taxonomy a than in taxonomy b. Therefore the actions themselves need to be explicitly encoded; it is no longer enough to rely on the fact that positions remain constant. By using a network that takes in action embeddings, the semantic meaning of each action can be clearly communicated.

4.1.1 Feature representation

In Section 3.2, we defined the action as a tuple (v, u) of 2 words, where the first word will be assigned as a child to the second word. To give the action as an input to the policy network, it needs to be represented by an action embedding. In this section, we describe how an action embedding is created.

In our algorithm, the policy network takes the state and an action representation as input and outputs the log probability of taking the action in the given state. The critic network takes the state and the action representation and outputs the value of that action. Both states and actions are defined as tuples of words. For the networks to be able to work with them, they must be converted into a suitable feature vector. In this section, we discuss how to obtain the feature representations of states and actions.

For every possible action (v, u), a feature vector is constructed, consisting of the following parts:

- **Word embeddings** The word embeddings of v and u. For our algorithm, we used GloVe [Pennington et al., 2014] embeddings.
- **Dependency path embedding** A dependency path represents information about how the two words occur in relation to each other in the background corpus. To obtain the dependency path of two words in a sentence, we first need the parse tree of the sentence. The shortest path between the two terms is the dependency path, which is represented as a sequence of edges starting from the first word to the second. The action representation of (v, u) contains all the dependency paths between v and u in the background corpus. To obtain a fixed-size representation, a path LSTM is used. It is a single-layer LSTM network with a hidden size of 60 that takes a sequence of paths and reduces them to a single vector of dimensionality 60. This LSTM is trained in conjunction with the rest of the model.
- **Syntax level features** The last part of the action feature representation includes a number of different syntax level features. These features are purely derived from the way the two words are written and do not include any additional statistics from the background corpus. Below we provide an overview of the syntax-level features. All these features are concatenated together to form the final vector.

  1. **Capitalization**: Whether any (or both) of the words are capitalized.
  2. **Endswith**: if the second word ends with the first word (for example, for the pair (bear, polar bear), this would fire.)
  3. **Contains**: If the second word contains the first word.
  4. **Suffix match**: The number of matching trailing letters.
  5. **LCS**: The length of the longest continuous substring contained by both words.
  6. **LD**: Length difference between the words. $10 * \frac{|w_1|-|w_2|}{|w_1|+|w_2|}$
  7. **Normalized frequency difference** The ratio between the frequency of pair (v, u) and the the most frequent parent of v, u': $\frac{\text{freq}(v,u)}{\max_{u'} \text{freq}(v,u')}$.
  8. **Generality difference** The generality $g(v)$ of term v is the logarithm of the number of its distinct hyponyms. The generality difference of the pair (v, u) is defined as $g(u) - g(v)$.

We used the same feature representation as DTaxa. More details can be found in Section 2.2 of [Mao et al., 2018]. For an overview of a constructed feature vector for action (v, u), see Figure 9.

As outlined in Section 3.2, a state of the MDP has the form (E, V). However, including the remaining term set V as part of the state feature is not necessary, as the action encoding already contains this information. Therefore we truncate the state representation to only E. Notice that the edges of a taxonomy at time t correspond to actions

Figure 9: An overview of the feature vector representation of action (v, u). It is a concatenation of word embeddings of the 2 terms, the dependency path of these terms based on the background corpus, and their syntax level features into a single vector.

taken up to time t, as each action effectively adds a new edge to the tree. The state at time step t, therefore, is the sequence of actions taken up to t:

$$s_t = (a_1, a_t, ..., a_{t-1}) \tag{15}$$

Its representation is obtained by feeding all the action embeddings $(a_1, a_2, ..., a_{t-1})$ to a single layer LSTM network that combines those values into a fixed-size vector of length 60. This LSTM layer is trained in conjunction with the rest of the model. As the model is fully differentiable, it is possible to simply backpropagate into the LSTM parameters.

4.1.2 Network architecture

The architecture of the policy network is shown in Figure 10.

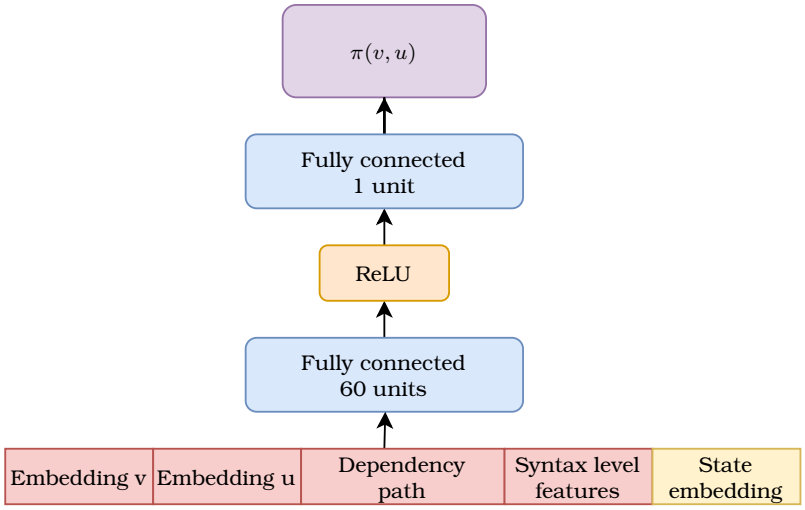

Figure 10: The architecture of the policy network. $\pi(v, u)$ gives the log probability of action (v, u).

The policy network is a simple, 2-layer feed-forward, fully connected neural network. Its input is the action encoding for action (v, u) and the state representation concatenated into a single vector of size 230. The first layer has 60 neurons, followed by a ReLU activation. Its second and final layer only has a single node with no activation. The output is a real-valued scalar that represents the log probability of choosing action (v, u).

During the construction of a taxonomy, all possible action pairs are fed through the network, with the outputs being concatenated to a vector that contains the log probability for each action. A Softmax is applied to this vector to convert the values into a valid probability distribution over the action space. We then sample from this distribution to obtain the action to take.

4.2 Critic

For our model, the critic is divided into two distinct sub-critics. Each of these critics can only observe a part of the feature space and, as such, are responsible for rating different components of the action. The output of these sub-critics is then combined with a single feed-forward layer neural network to obtain the final value estimate that can be used for training the actor. As the action is expressed as the edge (u, v) that is to be added to the taxonomy graph, it can be split naturally into two sub-actions. One part of the action, u, denotes the new term that shall be added to the taxonomy as a child node, while v denotes the parent node to connect the child node to. This clear division between the two parts of the action allows us to employ two distinct critics for rating each sub-action independently. Both critics use a slightly different feature vector, depending on which part of the action they focus on.

4.2.1 Feature representation

Both critics use a different feature vector, however those two vectors are similar, essentially the mirrors of each other. The motivation is that when evaluating one sub-action, no assumptions are to be made about the other part of the action. This leads to two changes in feature representation with respect to the one used by the actor:

- **Word embeddings** The word vector of one of the terms is left out, as for the sub-critic there is no information available about it. This means that one of the actors only uses the word vector v_v while the other only uses the word vector v_u.
- **Relational features** Another difference is that the dependency path and syntax level features can no longer be used, as they rely on knowing both words of the action. Instead of leaving those features out, they are modified in a way to not require knowledge of both terms, only a single one of them. This is achieved by taking a summary of the relations of the known word with all its possible pairs. As an example, let's say that the action chosen by the policy network is the tuple (v_i, u_j). Then the critic responsible for the choice of the child node would take the relations of v_i with every possible u and average them to obtain an approximation. This averaged feature is called the *average shared feature* of critic 1. The average shared feature of critic 2 is constructed in a similar way, except the choice of u_j is known, and the average is taken over all possible choices of v. This process is illustrated in Figures 11 and 12 for both sub-critics.

More formally, the shared features for the sub-critics are defined as:

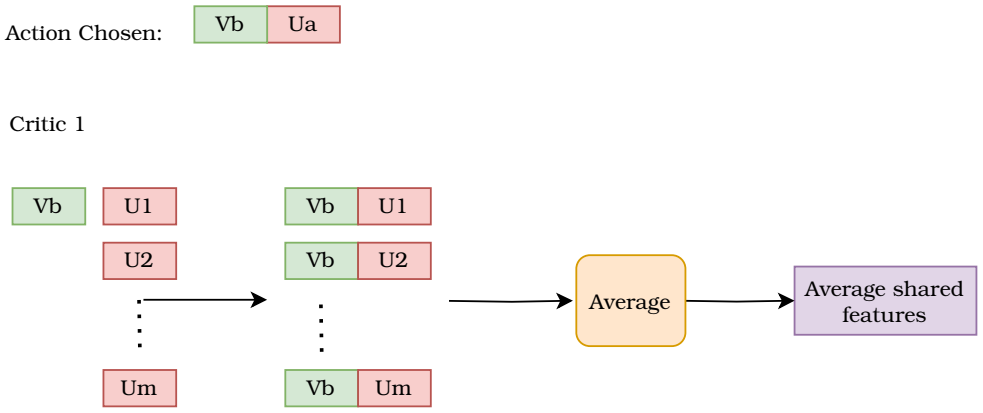

Figure 11: Shared feature summary of sub-critic 1. This sub-critic is only aware of the choice of the child term. To obtain the dependency path and syntax level features, it takes the features with all possible parent terms, then averages them.

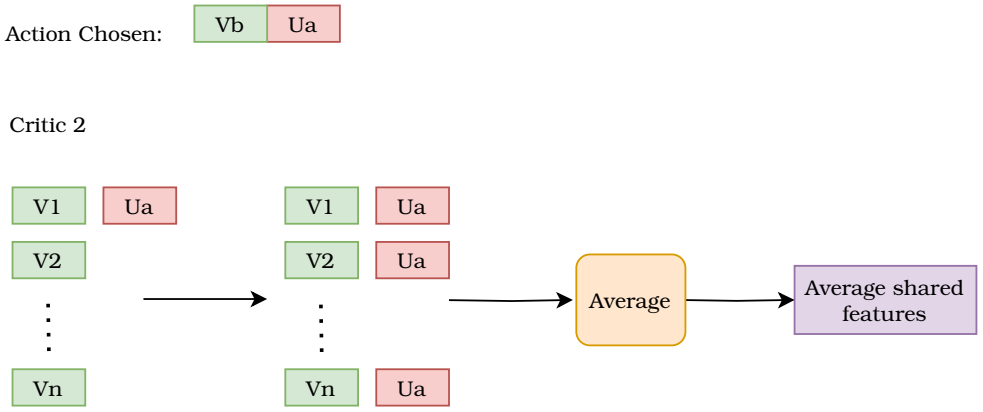

Figure 12: Shared feature summary of sub-critic 2. This sub-critic is only aware of the choice of the parent term. To obtain the dependency path and syntax level features, it takes the features with all possible child terms, then averages them.

$$
\begin{aligned}
&f(v,u): \text{dependency path and syntax features of the term pair } (v,u)\\
&f_{c1}(v): \text{The average shared feature of critic 1, where the child is } v.\\
&\text{The mean is taken of all feature vectors with } v \text{ as child.}\\
&f_{c2}(u): \text{The average shared feature of critic 2, where the parent is } u.\\
&\text{The mean is taken of all feature vectors with } u \text{ as parent.}\\
&f_{c1}(v) = \frac{\sum_{u \in U} f(v,u)}{|U|}\\
&f_{c2}(u) = \frac{\sum_{v \in V} f(v,u)}{|V|}
\end{aligned}
\tag{16}
$$

4.2.2 Network architecture

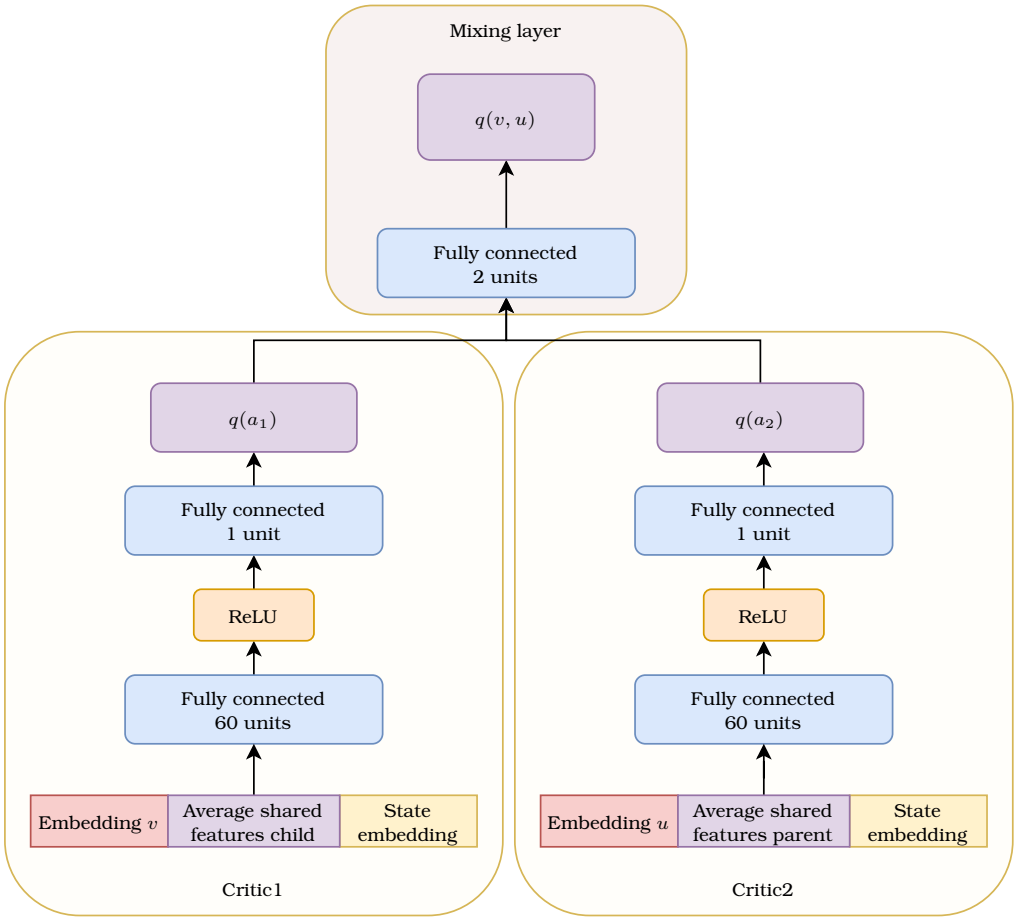

Figure 13: The architecture of the critic. $q(v, u)$ is the action-value of the action (v, u), and $q(a_i)$ is the value of sub-action i

The value network is built up of 3 distinct parts. This is illustrated in Figure 13. There is one network for both critics. Those share the same architecture, with two fully connected layers and a ReLU in between. The input vector contains a word embedding (the embedding of v in the case of critic1 and the embedding of u in the case of critic2), the appropriate average shared features, and the state representation. The input size is 140. The first fully connected layer consists of 64 neurons, while the second layer is just a single neuron. The output is interpreted as the value of the sub-action. The last part of the critic is the mixing layer. It is a simple, single-layer feed-forward neural network that takes the 2 sub-action values and combines them into the final action value. We experimented with different mixing functions, most notably a QMIX-like ar-

chitecture [Rashid et al., 2020], but we found a simple fully-connected layer to be more performant.

4.3 Training

The two critic networks and the policy network are trained jointly in our model. Algorithm 3 describes the joint training process of the agent in detail. The two critics are trained jointly, with the gradient being distributed by the mixing function. The loss is calculated based on the output of the mixing function, that combines the output values of both sub-critics. In the pseudo-code below we refer to the entire value network (both sub-critics and the mixing layer) as *combined critic*.

5 Experiments

To assess the performance of our model and compare it with previous methods, we set up several experiments. Our goal is to gain an insight into the characteristics, strengths and weaknesses of the algorithms by not only looking at the final performance, but also perform qualitative analysis on the structure of the constructed taxonomy. Below we provide a brief overview of our experiments:

1. **Ablation analysis and hyperparameter optimization** Before arriving at the final model structure, we performed a series of experiments to find the optimal network parameters, hyperparameters, and feature representation.

2. **Model performance** In this experiment, we tested the performance of our final model, as well as two of the baseline models (TaxoRL and DTaxa), on a set of taxonomy construction problems and compared their results.

3. **Robustness analysis** In this qualitative analysis, we looked at the structures of the constructed taxonomies by each model. Whether the outputs of a network are robust (the final structure generated remains constant across multiple runs with different starting words) or it generates completely different taxonomies for each run has a large effect on its applicability in practice. In this analysis, we focus on examining the robustness of each of the methods.

4. **Credit assignment** One of the motivations for choosing a multi-critic approach for our model was, as explained in Section 4, to improve credit assignment in the critic. In this qualitative analysis, we showcase how our critics have effectively learned the behavior patterns we outlined above.

5.1 Experiment setup

In this section, we describe our experimental setup in detail. Some parts of the setup remain the same for all 4 of the experiment, such as the model structure, training dataset, and hardware specifications. Different baselines and evaluation metrics, however, might

Algorithm 3: Joint training of the actor and the combined critic

DEFINITIONS;
D : Training dataset;
α_c, α_a : Critic and actor learning rate;
γ : Rewards discount rate;
τ : Target update rate;
μ : Actor parameters;
θ, θ' : Combined critic and combined critic target parameters;
π, q : Policy and value functions;
s, r : State and reward representations;
buff: Replay buffer;
INITIALIZATION;
buff $\leftarrow \emptyset$;
Initialize θ, μ randomly;
$\theta' \leftarrow \theta$;
for $(V, U, E) \in D$; /* For each taxonomy in the training set. */
do
  while $|V| > 0$; /* Repeat until the remaining term set is empty */
  do
    $A = (V \times U) \cup (\{\text{ROOT}\} \times V)$; /* 'A' is the set of all actions. */
    LP $\leftarrow \{\pi_\mu(s, a)$ for all $a \in A\}$; /* 'LP' is the vector of log probabilities of all actions. */
    $(v, u) \leftarrow$ sample(Softmax(LP)); /* Sample action */
    $V \leftarrow V \setminus \{v\}$; /* Update the taxonomy with the selected action */
    $U \leftarrow U \cup \{v\}$;
    $E \leftarrow E \cup \{(v, u)\}$;
    buff.add$(s, (v, u), r, s')$; /* Add (state, action, reward, next state) to buffer */
  end
  ; /* After the episode ends train on all transitions. */
  for $(s, a, r, s') \in$ *buff;* /* For each transition in buffer */
  do
    target $= r + \gamma q_{\theta'}(s', a)$; /* Calculate the critic target */
    $L_c = (\text{target} - q_\theta(s, a))^2$; /* Combined critic loss */
    $L_a = \ln(\pi_\mu(s, a)) \cdot q_\theta(s, a)$; /* Actor loss */
    $\theta \leftarrow \theta + \alpha_c * \nabla_\theta L_c$; /* Updating the parameters */
    $\mu \leftarrow \mu + \alpha_a * \nabla_\mu L_a$;
  end
  $\theta' \leftarrow \tau \cdot \theta + (1 - \tau) \cdot \theta'$; /* Updating target params */
  buff $\leftarrow \emptyset$;
end

be used for the different experiments. In the following, we explicitly mention any experimental setting that is only relevant for a subset of the experiments, while everything else implicitly holds for all experiments.

5.1.1 Baselines

Throughout the experiments, we compare the results of our method with two baseline models: TaxoRL [Mao et al., 2018] and DTaxa [Han et al., 2021] are the best-performing reinforcement learning algorithms on taxonomy construction. TaxoRL was the first paper to employ reinforcement learning to the task, significantly improving the state of the art, while DTaxa was a later iteration on it, replacing the REINFORCE training algorithm with an actor-critic architecture. Both of these methods were evaluated on the same dataset and also used the same background data to generate the feature vectors for the actions. This makes it straightforward to compare their results, which otherwise would be quite difficult, as, for example, using feature vectors extracted from a different background corpus could substantially influence the performance of the model.

TaxoRL The authors of TaxoRL made all their data (along with their code) publicly available. However, after running their training code with the default parameters (some of the hyperparameters were not specified in the paper), we discovered that the results do not correspond to those claimed in their paper. In particular, we discovered that the model generalizes very little to the evaluation set, with most of the improvements being visible on the training set. First, we tried several runs with different parameter settings in hopes of attaining a better performance comparable to their original claims. After we failed to do so, we tried contacting the authors for clarification. We reached out to the corresponding authors and the rest of the team and even raised an issue on their GitHub page. Regrettably, our efforts yielded no response from any of these attempts. For this reason, we decided to do our analysis on the training results, as those seem to improve significantly during training, while for the sake of completeness, we also report the evaluation results for all models. Our set of experiments analyzes the stability of the model during training, as well as the output structure of the fully trained model. As such, simply taking the final performance numbers reported in the TaxoRL paper is not sufficient for our purposes. Therefore we decided to proceed with the code provided and take that as the TaxoRL baseline. In order to match the setting of our model, TaxoRL was trained with a rollout of 2, meaning that it simulates each episode twice before moving on to the next episode.

DTaxa* We also reached out to the authors of the DTaxa paper, as they did not provide a publicly available link to their code. Unfortunately, we did not hear anything back from them either. For the sake of the experiments, we reproduced their model to the best of our abilities based on the limited description that was provided in the paper. We will refer to this reconstructed model as DTaxa*.

DTaxa* is an actor-critic algorithm with a single critic. For the policy network, we used the same architecture as TaxoRL, a two-layer fully connected neural network with

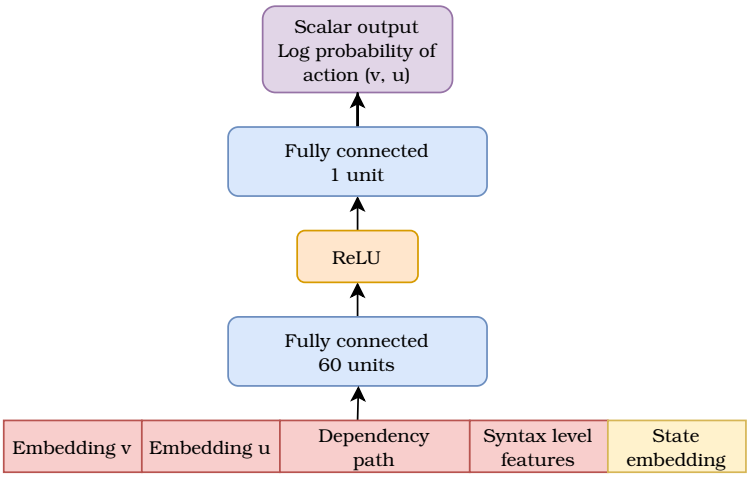

Figure 14: The architecture of the policy network in DTaxa*

60 neurons in the first layer and a single neuron in the second layer, with a ReLU activation in between. The network structure is shown in Figure 14. The length of the feature vector is 230. The input to the policy network has the same structure as our model described in Section 4: The first 50 nodes represent the word embedding of *word1*, the next 50 represent the embedding of *word2*. The path embedding of the 2 words has dimensionality 60, and the syntax level features (detailed in Section 3) 70. In total, the size of the input vector adds up to 230. The output of the network is a scalar that represents the log probability of choosing the action (*word1, word2*). After passing each possible (*word1, word2*) action pair through the network and obtaining their log probabilities, all outputs are arranged in a single vector and fed into a Softmax layer. The output of the Softmax is a probability distribution over the possible actions that will be sampled from during training, and the maximum will be taken during evaluation. The architecture of the policy network is illustrated in Figure 14.

The critic network has the same architecture as the policy network outlined above. The input feature vector is the same for both networks. The main difference lies in the interpretation of the output node. While for the actor, the output represents the log probability of the action, the output of the critic gives the value of the action, which is the expected return after taking said action. The architecture of the critic network is illustrated in Figure 15.

5.1.2 Dataset

For the experiments we use the WordNet taxonomy dataset[Bansal et al., 2014], which was also used by TaxoRL and DTaxa. It contains a set of 761 taxonomies sampled from WordNet [Miller, 1995], each with a depth of 3, built up from 10-50 nodes. This data set only provides the set of words and the corresponding target taxonomies but leaves the background corpus unspecified. The performance of the agent on the benchmark is, of

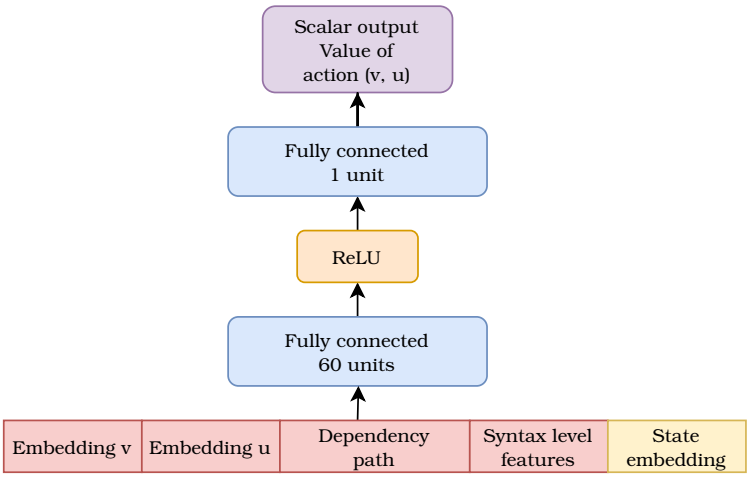

Figure 15: The architecture of the critic network in DTaxa*

course, largely dependent on the background corpus, as it is used to extract statistical information about relations of different terms, which would be used as part of the feature representation during training. In order to make the comparison with previous methods meaningful, we decided to use the same background text as TaxoRL, which is an aggregation of Wikipedia dump, the UMBC web-based corpus [Han et al., 2013] and the One Billion Language Modelling Benchmark [Chelba et al., 2013].

5.1.3 Experimental environment

In our experiments, we train our model, TaxoRL and DTaxa*, on the dataset outlined above. The *ablation analysis* and *performance analysis* experiments focus on how the performance changes during training, while the qualitative analysis (*robustness analysis* and *credit assignment*) will be conducted on the already trained models.

Each model in the experiments is trained for 300 epochs. The weights are then saved and later used in the qualitative analysis. During an epoch, a single training episode is run for each of the taxonomies in the training set. In our training set there are 533 taxonomies, therefore each epoch is 533 episodes long. An episode comprises of a construction of a single taxonomy. At the start of each episode, a set of terms are provided, and the goal is to build up a taxonomy from said terms that match the target *golden taxonomy* as close as possible. The *golden taxonomy* is the correct solution provided as part of the dataset, similar to a *label* in the case of supervised learning.

The agent's action interface only allows for the extending of an already existing taxonomy, but not for creating a new taxonomy from scratch, as the output action is a tuple of (*child*, *parent*). Practically speaking, this restriction requires at least one node (the *root*) to already be present at the start of construction. Providing the correct root at the start of each construction can have a major effect on the performance. To mitigate that, similarly to TaxoRL, we chose to start each episode with a randomly selected root

node. The agent can then attach a new root node on top of it by selecting the node to be added as parent and the current node as child. Starting with a randomly chosen root node also makes the algorithm more robust, as it cannot simply overfit to a simple construction sequence for each taxonomy, but has to adapt to each possible starting point.

5.1.4 Metrics

At the end of the episode (when all terms are added to the taxonomy tree), the final construction is evaluated against the gold taxonomy. The measure we use is the edge F1 score:

$$\begin{aligned} \text{Set of edges present in the constructed taxonomy: } & E_{pred} \\ \text{Set of edges present in the gold taxonomy: } & E_{gold} \\ \text{Edge precision: } & P_e = \frac{|E_{pred} \cap E_{gold}|}{|E_{pred}|} \\ \text{Edge recall: } & R_e = \frac{|E_{pred} \cap E_{gold}|}{|E_{gold}|} \\ \text{Edge F1: } & F1_e = 2 \cdot \frac{P_e \cdot R_e}{P_e + R_e} \end{aligned} \tag{17}$$

5.1.5 Hyperparameter setting

All the hyperparameter values used in later experiments are based on the results of our ablation analysis and hyperparameters experiments. In those experiments we investigate the effect of different learning rates of both the actor and the critic, different layer sizes, and feature representations (for more details, see Section 5.2.1).

The architecture of our actor and critic networks can be found in Section 4. Both critic networks and the policy net are two-layer feed-forward neural networks with a ReLU activation after the first layer. The input size of the actor is 230, the first layer consists of 60 neurons, while the output is a single node. Both sub-critic networks have the same architecture with a two-layer feed-forward neural network, with a ReLU activation between those 2 layers. The critic input size is 180, with the first layer having 64 neurons and the second layer consisting of a single neuron. The results of the two sub-critics are combined in a final, single-layer neural network that takes the two values from each sub-critic as input, and outputs a single final value for the action.

For the training of both the actor and the critic network, Adam optimizer[Kingma and Ba, 2014] was used, with a learning rate of 0.0005 for the actor and 0.0001 for the critic.

5.1.6 Hardware usage

All the experiments were run on a Linux virtual machine running Ubuntu 18.04 with an Intel Xeon Platinum CPU with 2 cores and 32 GB ram. With this setup, running a

single epoch took 50-60 minutes on average. Running the full experiment up to 300 epochs took over 11 days.

5.2 Results

In this section, we present the results of all 4 of our experiments. We conducted the overall performance and robustness experiments for both TaxoRL and DTaxa* as well to compare the results and learn about differences between the models.

5.2.1 Ablation analysis and hyperparameter optimization

In this section, we present the results of our preliminary experiments. The aim of these experiments is to find the best-performing feature representations and hyperparameter settings.

To assess the importance of different parts of the feature space used to represent the taxonomy and the actions to be taken, we run a series of experiments with different subsets of the feature space masked out. Particularly we tested the effect of 2 features, sibling embeddings and using history. Node m is a *sibling* of node n if they share the same parent, that is, there exists a node k such that $(m, k) \in E$ and $(n, k) \in E$. When sibling embeddings are used, the feature vector of an action (v, u) gets modified by adding the average embeddings of the siblings of v to it. The history feature refers to including the summary of past actions in the feature vector.

The results of this analysis can be seen in Table 1. For this analysis, we focused on testing the effect of leaving out parts of the feature representation. Interestingly, we found that leaving out the history representation actually improves the overall performance. We tested this on TaxoRL, where we observed similar behavior. Making use of sibling embeddings, on the other hand, positively impacts the performance. Based on this analysis, we decided to leave out the history representation from both our model and TaxoRL for the main experiment. Note that this analysis was run with an earlier version of the model before it was fully optimized, therefore the results are slightly lower than in the final experiment.

Model	**use history**	**use sibling**	**edge-F1 after 150 epochs**	**edge-F1 after 200 epochs**
Ours	No	No	0.3233	0.3328
Ours	No	Yes	0.3301	0.3434
Ours	Yes	No	0.1649	0.1724
Ours	Yes	Yes	0.2506	0.2596

Table 1: The performance of our model when some of the features are not utilized.

In addition to finding the best feature representation, we also looked at the effect of other hyperparameters. We run experiments to select the learning rate pairs for the actor and critic networks. The results of this analysis can be seen in Table 2.

Model	Actor learning rate	Critic learning rate	Edge F1 at 150 epochs
Ours	0.001	0.001	0.2816
Ours	0.005	0.001	0.3301
Ours	0.01	0.001	0.2041

Table 2: The analysis of different learning rate ratios between the actor and the critic network

In this experiment, we found the best-performing learning rates, however, these are not the same that were used in later experiments. During the preliminary experiments, the model had a slightly different update rule, which meant it was updated much less frequently, but with more stable gradients. To counteract the effects of a slightly less stable gradient in the final version, both learning rates have been multiplied by $\frac{1}{10}$. This did not result in slower convergence as the training steps happened more frequently.

The path LSTM is an important part of the model, as it is responsible for summarizing the information about the relation of two words into a fixed-size representation. The size of this LSTM layer has a significant impact on the final performance. To find the optimal value, we run an experiment, the results of which can be found in Table 3.

Model	path LSTM dimension	Edge F1 at 150 epochs	Edge F1 at 200 epochs
Ours	60	0.3301	0.3434
Ours	128	0.3353	0.3354
Ours	256	0.3208	0.3303

Table 3: Performance of our model with different path LSTM dimensionality.

In this section, we described 3 preliminary experiments with the goal of optimizing different aspects of our model. The final model structure is constructed based on these experiments.

5.2.2 Model performance

For this experiment, we trained our model, as well as TaxoRL and DTaxa*, for 300 epochs on the dataset described in Section 5.1.2. The training results are shown in Figure 16. For the training procedure, we followed the pseudo-code given by Algorithm 3. The graph shows that despite its initial slow start, our method eventually outperforms TaxoRL in the experiment. The slower convergence at the beginning shows a characteristic difference between the two algorithms. TaxoRL starts training the policy network towards the sampled returns at each transition, which provides a noisy but unbiased estimate of the true return. At the same time, our agent updates its policy network toward the output of the critic. The critic network is initialized randomly at the beginning of training, which means that its output is also random. Therefore during the first part of the training, there is no meaningful learning going on in the policy. However, after the

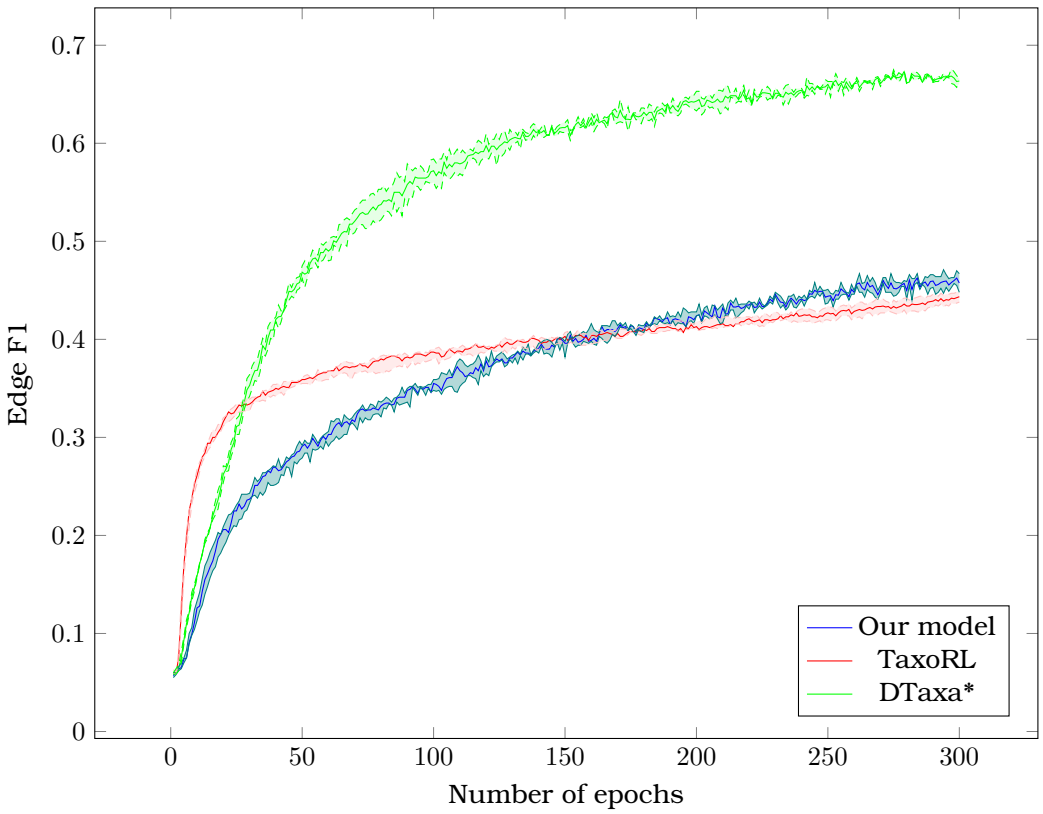

Figure 16: The training graph of TaxoRL, our model and DTaxa. The darker line in the middle shows the average performance, while the lighter lines on the top and bottom give the minimum and maximum values over the runs.

critic is sufficiently trained to provide reasonable estimates for action values, the speed of convergence of the policy picks up and even surpasses that of TaxoRL, because it is now training towards stable targets with low variance (the output of the critic), while TaxoRL works with high-variance empirical return. The graph also shows that DTaxa* significantly outperforms both other methods.

To assess the stability of algorithms, we run each algorithm 3 times and average their results. The dark-colored lines in Figure 16 indicate the median value of each model, while in lighter color around it is the highest and lowest value shown. The graph shows that all 3 models are stable across different training runs, with a low variance in performance.

Table 4 shows the results of all discussed methods after a certain number of epochs. Table 5 shows the evaluation results, which are similar across the algorithms. For the trained models we also report the precision and recall results. For those results see Table 6.

Model	100 epochs	150 epochs	200 epochs	250 epochs	300 epochs
TaxoRL	0.386	0.400	0.413	0.426	0.443
DTaxa*	0.571	0.616	0.643	0.66	0.664
Ours	0.349	0.399	0.421	0.443	0.459

Table 4: The Edge-F1 score on the training set performance of the algorithms after given number of epochs. DTaxa* refers to our DTaxa implementation based on the limited information in their publication.

Model	100 epochs	150 epochs	200 epochs	250 epochs	300 epochs
TaxoRL	0.063	0.066	0.069	0.068	0.065
DTaxa	0.065	0.063	0.067	0.068	0.053
Ours	0.063	0.062	0.062	0.067	0.058

Table 5: The Edge-F1 score on the evaluation set performance of the algorithms after given number of epochs. DTaxa* refers to our DTaxa implementation based on the limited information in their publication.

5.2.3 Robustness analysis

Another important characteristic of a model is its robustness across several runs on the same domain. In addition to the performance, in practical applications, it is often crucial that the produced taxonomies are consistent. To investigate the robustness of our method, we analyze the final constructed structure in a series of runs on the example taxonomy (Figure 18) with different initial root words. The goal of this analysis is to show to what degree our model is able to arrive at the correct final structure regardless of the starting point.

We let the model run with 5 different starting words. Our first observation is that the correct root (*bedroom*) is correctly identified in 3 out of the 5 cases. In the remaining 2 cases, *dormitory* and *guestroom* were chosen as roots. Out of the 55 total edges of the 5 runs, 31 were identified correctly (it was present both in the predicted as well as the gold taxonomy), while 24 were incorrect. Among the incorrect edges a pattern can be observed. 11 of the 24 wrong edges go to *guestroom* as their parent, suggesting that there is a systematical bias in the model, rather than missing completely arbitrarily. This makes it easier to mitigate its flaws in a practical use case, where after construction, a domain expert can quickly check only parts of the final taxonomy that are most likely to

Model	Edge Precision	Edge Recall	Edge F1
TaxoRL	0.23	0.427	0.299
DTaxa*	0.55	0.662	0.601
Ours	0.321	0.443	0.372

Table 6: The final precision, recall and F1 scores after training.

contain errors instead of having to check the entire tree. This is especially useful in the case of much larger trees in more complicated domains, but for the sake of readability, we chose to showcase it on a smaller example. The 5 generated structures are shown in Figure 17.

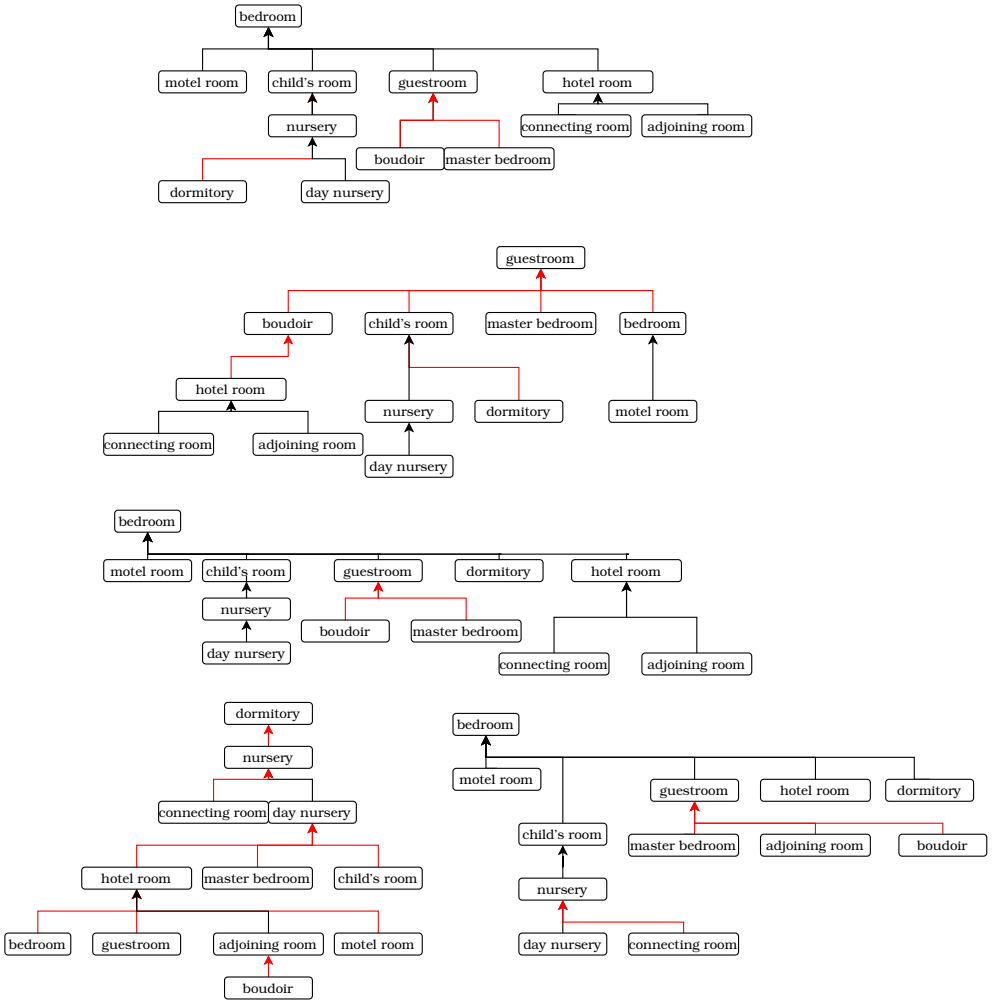

Figure 17: The trees generated by our model. The red arrows indicate the wrong edges. The black edges are correctly found.

The same experiment was also conducted with the benchmark models. We used the same taxonomy for all methods, with 5 runs using a randomly generated starting word for each. DTaxa* had a similar performance in terms of overall correctness, with 29 correct and 26 incorrect edges predicted. The main difference was the ability to identify the root node. DTaxa* failed to find the correct root node in all 5 runs. After manually

inspecting the generated structures, we observed that similarly to our model, DTaxa* also seems to have a bias towards a single term, *connecting room*, which frequently has a lot of children assigned, despite being a leaf node in the golden taxonomy. 11 of the 26 incorrect edges have *connecting room* as their parent. TaxoRL identified 29 edges incorrectly and 26 correctly, which is the weakest performance of the 3 models. However, interestingly it managed to identify the correct root in all 5 cases. The generated taxonomies can be found in Section B.

5.2.4 Credit assignment

One of the motivations for the use of a multi-critic architecture as described in Section 4 would improve convergence speed by enabling credit assignment by the critic. This idea was illustrated with a hypothetical example that showed that in certain scenarios an incorrect action (p, c) might be blamed entirely on one of the sub-actions, while the other might be an objectively good choice. To demonstrate that our model possesses this property we analyzed the construction process of a small example taxonomy.

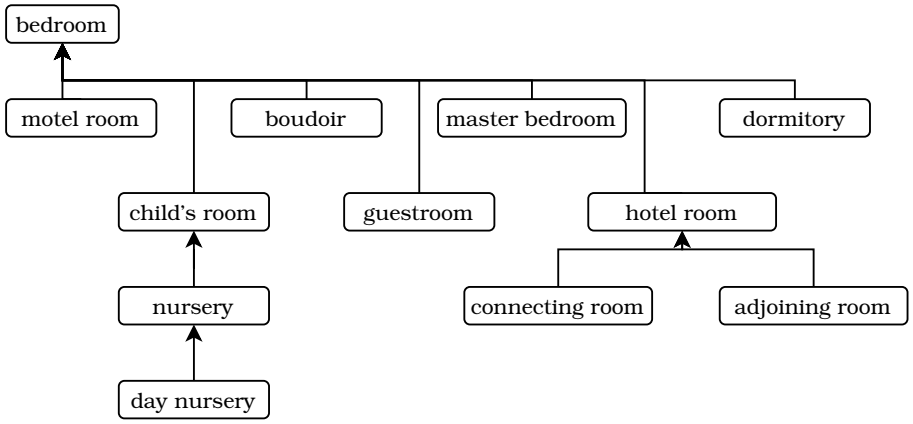

Figure 18: The example taxonomy.

The example taxonomy, which is a golden taxonomy from the dataset, is shown on Figure 18. It is a simple hierarchy of bedrooms. To show the desired properties, we will consider it in a partially built state, where a number of terms still need to be assigned. Figure 19 shows the state of the tree at the point of interest. All the first and second level nodes (in yellow and blue) are already added to the tree, while the green and red nodes are in the remaining term set (their target position is marked on the figure to make it easier to see where each node belongs, but they are not part of the taxonomy at this point in time). For this analysis we will look at the output values of the sub-critics for all possible actions with the partially constructed taxonomy. Since $V = \{\text{nursery, day nursery, connecting room, adjoining room}\}$ the valid actions at this time step are the following: Each of these nodes in V can be either attached as a child node to any node already in the taxonomy (yellow and blue), or alternatively any of the

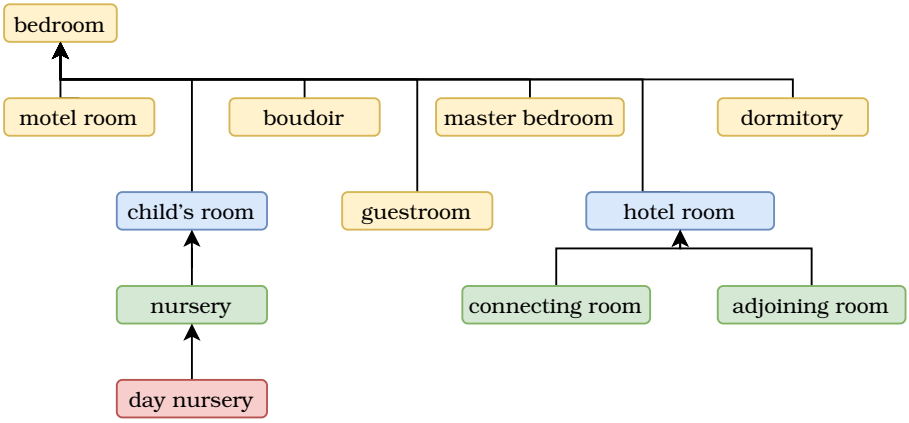

Figure 19: The example taxonomy after all the yellow and blue nodes have been correctly added to the tree. The green and red nodes are still in the remaining term set, with there target position marked on the figure.

terms in V can be used as a new node, by making it a parent of the current root node *bedroom*.

To analyze whether our approach can correctly learn to assign credit for two sub-actions, we investigate the intermediate output values from two critics. Especially, in the given example, we print the average output of the child critic (which determines the estimated value for selecting each possible v in V as the child node).

In this example, choosing either one of the blue terms as child is a good choice, without further information about the parent, as for each of these terms there exists a node in the tree that is its correct parent. However for the red node *day nursery* there is no such parent present. Intuitively, we would expect that the sub-critic responsible for the choice of the child node would give a lower value to actions that choose *day nursery* as the child node compared to others. To test whether our model is able to learn to effectively assign blame in these situations, we investigate the intermediate values assigned to every possible actions by the critic responsible for the choice of the term. For each of the child candidate terms there are in total 8 possible parents. Table 7 shows the average action value for each candidate child. As we can see, the value of *day nursery* is the lowest with -5.14, while the average of the green terms (the ones which have a valid parent in the current tree) is -4.32, meaning that our critic prefers the choice of any of the green nodes above the red one as a new child, which is in line with the expectations. *day nursery* can not be correctly attached as a child to any current nodes in the tree, therefore it is rated lower by the critic.

A similar analysis can be done on the choice of the parent node as well. After looking at Figure 19, we find that only the choice of one of the blue nodes (*child's room*, *hotel room*) as parents leads to a correct action, because none of the yellow terms is a parent to any of the possible children. Therefore we expect the blue terms to have a higher action value than the yellow ones. Table 8 shows the values of the parent candidates.

	nursery	connecting room	adjoining room	day nursery
Action value	-3.72	-4.76	-4.47	-5.14

Table 7: This table shows the inverse actions values of choosing each of the nodes as a child

The average value of the incorrect parents (yellow) is -6.92, while the average value of the correct parent choices (blue) is -4.04. Again we can see the same phenomenon as previously, the sub-critic has successfully learned to prioritize those choices that make sense in isolation, even without further information about the other part of the action (the choice of child in this case).

	bedroom	boudoir	motel room	guestroom
Action value	-5.76	-4.87	-5.87	-5.24

	master bedroom	dormetry	child's room	hotel room
Action value	-9.00	-10.76	-4.71	-3.37

Table 8: This table shows the inverse actions values of choosing each of the nodes as the parent

This analysis demonstrates that our multi-critic algorithm learned to correctly assign the blame in cases where one sub-action is clearly responsible for the choice of incorrect action and does not penalize sub-actions that are in principle correct, and only fail because of the wrong choice of the other. This property can help during training to keep the action value estimates consistent, thus leading to faster convergence.

6 Discussion and future work

In this thesis, we explored the viability of using a multi-critic reinforcement learning algorithm for taxonomy construction. Our original motivation for this choice was that we saw an inherent structure in the action space of the task that was not taken advantage of with previous methods. With the success of the different types of multi-critic algorithms, applying it to a structured problem such as taxonomy construction was a natural idea. The results seem to suggest that it is indeed a promising direction, however, more research is needed to unlock the full potential of this method. One of the key characteristics that we expected from our model, based on the results of previous work on multi-critic agents, is that the critics would easily adapt to their more constrained role and would be able to work together efficiently. Our credit assignment analysis shows that this happened according to expectations, as both the critics were able to recognize correct and incorrect sub-actions. This is a promising sign, however, it largely failed to shine through in the overall performance. While our model outperformed TaxoRL, its performance was significantly weaker than DTaxa*, a method with a single critic. This is a surprising result considering the 2 sub-critics performed as

intended. Trying to pinpoint the reasons behind this performance deficit might be an interesting follow-up research. We believe it is worth investigating the effects of using a different mixing function to effectively combine the insights from both sub-critics into the final value. Another point of interest is the robustness of the algorithm, that is, how reliable it is when constructing the taxonomies. Our experiments showed that the robustness of our method was similar to that of DTaxa*, despite the overall performance deficit. This is another sign that suggests it might be worth it to continue research on this topic. If the performance of the model improves, there is a chance the robustness will get even better, yielding an algorithm that is superior to DTaxa* in practice by virtue of being more stable. In conclusion, we think that our method showcases an interesting idea, and it is a good foundation for further research in a promising direction, with the potential of improving current state-of-the-art results.

7 Conclusion

In our thesis, we demonstrated the advantages of using a multi-critic algorithm for taxonomy construction. Previous methods treat all actions as independent, meaning that adding the edge (*rabbit, animal*) or (*tree, animal*) are seen as completely different actions despite the fact that in both cases the parent node is the same (*animal*). In this thesis, we showed the benefits of decoupling the action into 2 sub-parts, and assigning a critic for each of those parts, instead of employing a single, centralized critic. We demonstrated that this method leads to better credit assignment, because the critics are able to effectively assign blame to the part of the action that is responsible for it being incorrect. While our method did not outperform DTaxa*, the improved credit assignment, together with its overall robustness, suggests that multi-critic approaches for taxonomy construction are viable. With future research, there is the potential for such methods to outperform previous models that do not take advantage of the action structure of the task and are not explicitly designed for proper credit assignment.

References

[Bansal et al., 2014] Bansal, M., Burkett, D., De Melo, G., and Klein, D. (2014). Structured learning for taxonomy induction with belief propagation. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1041–1051.

[Bellman, 1957] Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684.

[Brachman, 1983] Brachman, R. J. (1983). What is-a is and isn't: An analysis of taxonomic links in semantic networks. *Computer*, 16(10):30–36.

[Chelba et al., 2013] Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. (2013). One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

[Church, 2017] Church, K. W. (2017). Word2vec. *Natural Language Engineering*, 23(1):155–162.

[Franceschetti et al., 2021] Franceschetti, A., Tosello, E., Castaman, N., and Ghidoni, S. (2021). Robotic arm control and task training through deep reinforcement learning. In *International Conference on Intelligent Autonomous Systems*, pages 532–550. Springer.

[Fu et al., 2014] Fu, R., Guo, J., Qin, B., Che, W., Wang, H., and Liu, T. (2014). Learning semantic hierarchies via word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1199–1209.

[Geffet and Dagan, 2005] Geffet, M. and Dagan, I. (2005). The distributional inclusion hypotheses and lexical entailment. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 107–114.

[Han et al., 2013] Han, L., Kashyap, A. L., Finin, T., Mayfield, J., and Weese, J. (2013). Umbc_ebiquity-core: Semantic textual similarity systems. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 44–52.

[Han et al., 2021] Han, Y., Lang, Y., Cheng, M., Geng, Z., Chen, G., and Xia, T. (2021). Dtaxa: An actor–critic for automatic taxonomy induction. *Engineering Applications of Artificial Intelligence*, 106:104501.

[Hearst, 1992] Hearst, M. A. (1992). Automatic acquisition of hyponyms from large text corpora. In *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Konda and Tsitsiklis, 1999] Konda, V. and Tsitsiklis, J. (1999). Actor-critic algorithms. *Advances in neural information processing systems*, 12.

[Kozareva and Hovy, 2010] Kozareva, Z. and Hovy, E. (2010). A semi-supervised method to learn and construct taxonomies using the web. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, pages 1110–1118.

[Lillicrap et al., 2015] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

[Lin et al., 1998] Lin, D. et al. (1998). An information-theoretic definition of similarity. In *Icml*, number 1998 in 98, pages 296–304.

[Luu et al., 2014] Luu, A. T., Kim, J.-j., and Ng, S. K. (2014). Taxonomy construction using syntactic contextual evidence. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 810–819.

[Mao et al., 2018] Mao, Y., Ren, X., Shen, J., Gu, X., and Han, J. (2018). End-to-end reinforcement learning for automatic taxonomy induction. *arXiv preprint arXiv:1805.04044*.

[Martinez-Piazuelo et al., 2020] Martinez-Piazuelo, J., Ochoa, D. E., Quijano, N., and Giraldo, L. F. (2020). A multi-critic reinforcement learning method: An application to multi-tank water systems. *IEEE Access*, 8:173227–173238.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26.

[Miller, 1995] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

[Minsky, 1961] Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

[Mysore et al., 2021] Mysore, S., Cheng, G., Zhao, Y., Saenko, K., and Wu, M. (2021). Multi-critic actor learning: Teaching rl policies to act with style. In *International Conference on Learning Representations*.

[Peng et al., 2021] Peng, B., Rashid, T., de Witt, C. A. S., Kamienny, P.-A., Torr, P. H. S., Böhmer, W., and Whiteson, S. (2021). Facmac: Factored multi-agent centralised policy gradients.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

[Rashid et al., 2020] Rashid, T., Samvelyan, M., De Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2020). Monotonic value function factorisation for deep multi-agent reinforcement learning. *The Journal of Machine Learning Research*, 21(1):7234–7284.

[Snow et al., 2004] Snow, R., Jurafsky, D., and Ng, A. (2004). Learning syntactic patterns for automatic hypernym discovery. *Advances in neural information processing systems*, 17.

[Snow et al., 2006] Snow, R., Jurafsky, D., and Ng, A. Y. (2006). Semantic taxonomy induction from heterogenous evidence. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 801–808.

[Sutton and Barto, 2018] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

[Sutton et al., 1999] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

[Wang et al., 2017] Wang, C., He, X., and Zhou, A. (2017). A short survey on taxonomy learning from text corpora: Issues, resources and recent advances. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1190–1203.

[Watkins and Dayan, 1992] Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.

[Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

[Wu et al., 2018] Wu, J., Wang, R., Li, R., Zhang, H., and Hu, X. (2018). Multi-critic ddpg method and double experience replay. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 165–171. IEEE.

[Wu et al., 2012] Wu, W., Li, H., Wang, H., and Zhu, K. Q. (2012). Probase: A probabilistic taxonomy for text understanding. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, pages 481–492.

[Yang et al., 2018] Yang, Z., Merrick, K., Jin, L., and Abbass, H. A. (2018). Hierarchical deep reinforcement learning for continuous action control. *IEEE transactions on neural networks and learning systems*, 29(11):5174–5184.

A DTaxa* robustness

Figure 20 shows the taxonomy trees generated by DTaxa* during the robustness analysis. Only 3 different trees were generated, 2 of the trees occurring twice each.

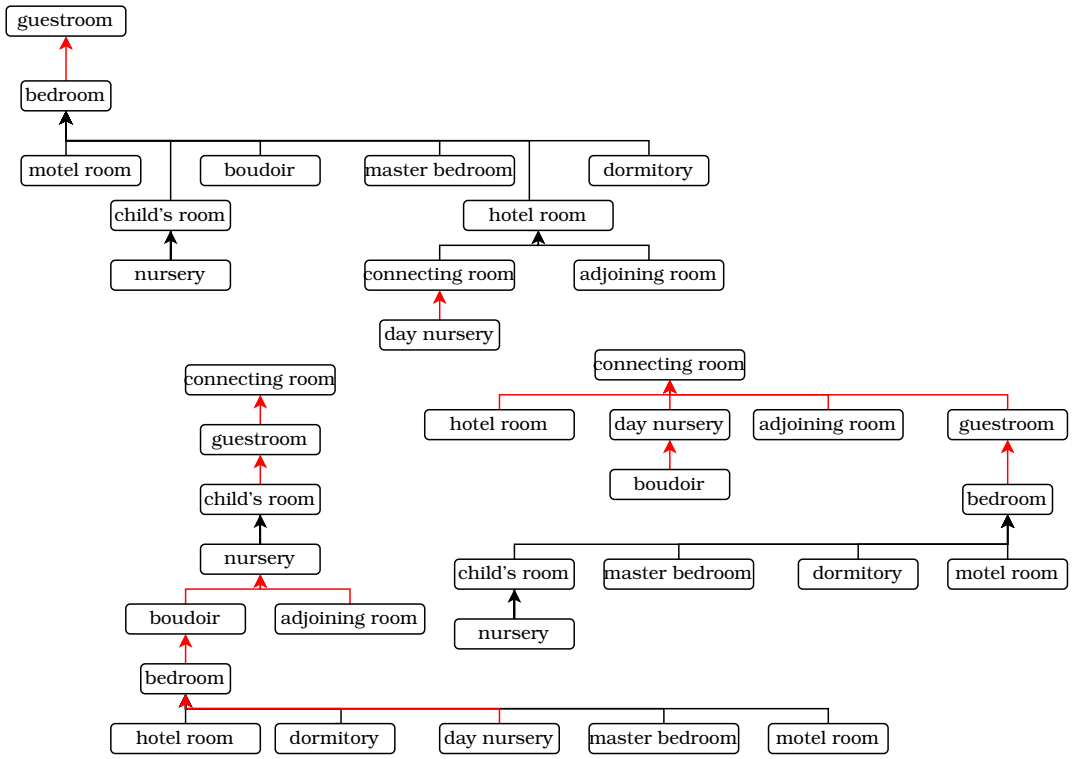

Figure 20: The trees generated by DTaxa* in 5 tries. Two of the trees were created twice. The red arrows indicate wrong edges. The black edges are correctly found.

B TaxoRL robustness

Figure 21 shows the taxonomy trees generated by TaxoRL during the robustness analysis. Only 3 different trees were generated, with the first one occurring 3 times.

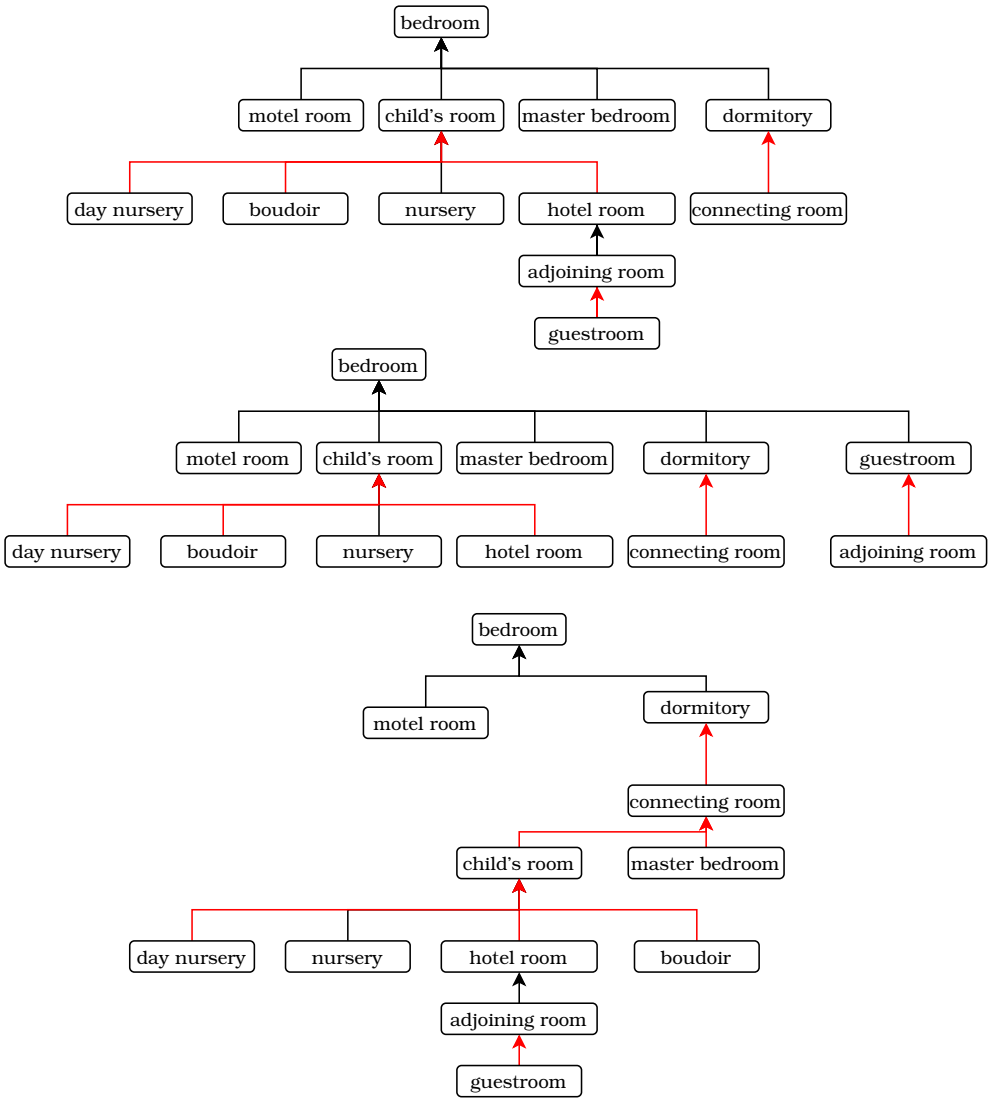

Figure 21: The trees generated by TaxoRL in 5 tries. The first tree occurred 3 times. The red arrows indicate wrong edges. The black edges are correctly found.