Hierarchical Reinforcement Learning Software Project Report Summer Semester 2020

Benjamin Beilharz and **Blanca Birn** and **Leander Girrbach** and **Tai Mai** and **Shaptarshi Roy**
{beilharz, birn, girrbach, mai, roy}@cl.uni-heidelberg.de

Supervised by **Prof. Dr. Stefan Riezler**

Abstract

This report describes our efforts to re-implement the methods described in (Yao et al., 2019). We re-implement the supervised baselines proposed in the paper and the reinforcement learning method (Hierarchical Reinforcement Learning, HRL). As a major part of our project, we integrate reinforcement learning functionality into JoeyNMT (Kreutzer et al., 2019).

1 Introduction

With the advent of digital assistants, the field of semantic parsing has gained yet another use case to be explored. Yao et al. (2019) introduced an approach to semantic parsing that handles user queries similar to those typically sent to digital assistants. In this scenario, the model is equipped with the ability to ask the user follow-up questions if the initial parse of the query is not sufficient for a confident prediction. In order to minimise the number of questions to the user, Yao et al. (2019) employ (hierarchical) reinforcement learning. This approach is considered *hierarchical* because the queries are regarded to be subdivided into different subtasks, which gives the model the opportunity to optimise the order in which it processes the subtasks and minimise user interaction. In this project, we made efforts to re-implement this approach proposed by Yao et al. (2019) and integrate it into the JoeyNMT deep learning framework (Kreutzer et al., 2019).

2 Integrating RL into JoeyNMT

As part of this project, the JoeyNMT framework (Kreutzer et al., 2019) has been extended with reinforcement learning functionalities that allow the implementation of both Classic Control reinforcement learning scenarios such as Acrobot[1] and Cartpole[2], as well as hierarchical reinforcement learning scenarios such as the one presented in Yao et al. (2019). To accomodate this reinforcement learning functionality, a new training procedure has been implemented which can be called from the command line just like the classical JoeyNMT training procedure. Furthermore, the reinforcement testing procedure can be called in just the same way. In order to implement custom environments, following the OpenAI Gym API[3] will ensure compatibility with the rest of the framework. To implement one's own agent network, an abstract baseclass interface is provided as a guideline of what's expected by the rest of the framework. Specifics to the implemented reinforcement learning features and methods are listed in the following two subsections.

2.1 Implemented features

We add a reinforcement learning interface to JoeyNMT for tasks with discrete action space. The implemented features include:

- RL optimisation algorithms (see Sec. 2.2)
- Sampling methods (for sampling actions from the categorical distribution defined by the policy)
- Seamless integration of configuration options into JoeyNMT
- Environments follow the OpenAI Gym (Brockman et al., 2016) environment API
- Test procedure with functionality to calculate the accuracies per subtask in the case of hierarchical reinforcement learning

To run a new reinforcement learning task in JoeyNMT, providing an environment which implements the OpenAI Gym environment and an

[1] https://gym.openai.com/envs/Acrobot-v1/

[2] https://gym.openai.com/envs/CartPole-v1/

[3] https://gym.openai.com/docs/#observations

agent is all that is necessary. We provide the API any agent has to implement as an abstract base class. Optionally, a new value network architecture can be specified as well. All other functionality of JoeyNMT, including the semantics of the configuration files, remains unchanged.

As sampling strategies, we provide:

greedy Samples the action with the highest probability according to the policy

ϵ-greedy Like `greedy`, but with a probability of ϵ, samples an action from the categorical distribution defined by the policy

multinomial Samples one action from the categorical distribution defined by the policy

2.2 Implemented RL algorithms

We implement the following reinforcement learning optimisation algorithms:

REINFORCE (Williams, 1992) For implementing REINFORCE, we use the formulation in (Sutton and Barto, 2018). Given policy π, discount factor γ and episode $s_t, r_t, a_t;\ 0 \leq t \leq T$, we perform gradient ascent on:

$$\nabla \log \pi(a_t \mid s_t) \sum_{k=t}^{T} \gamma^{k-t} r_k \quad \forall t$$

We enable to optionally integrate a baseline $b(s)$. We implement the average reward baseline, which substracts the average reward received so far from each discounted reward. Also, we implement a trainable state-value baseline.

(TD) Actor Critic Given transitions s, a, r, s', discount factor γ, critic network V, and policy π, let $\delta = r + \gamma V(s') - V(s)$. We perform simultaneous gradient ascent on:

$$\delta \nabla \log \pi(a|s) \qquad \text{(Actor Loss)}$$
$$\delta \nabla V(s) \qquad \text{(Critic Loss)}$$

Advantage Actor-Critic (A2C) Lastly, we also implement the algorithm (S3) proposed in (Mnih et al., 2016). Furthermore, our implementation enables the (optional) use of target networks. Please note that we do not implement the asynchronous update features of the algorithm.

All 3 implemented algorithms share as configurable features:

- Entropy loss (maximised)
- Normalisation of discounted rewards to 0 mean and 1 variance
- Multi-agent setting. In this case, rewards, states, and actions are grouped by agent. Optimisation is performed for each agent individually.

2.3 Working examples

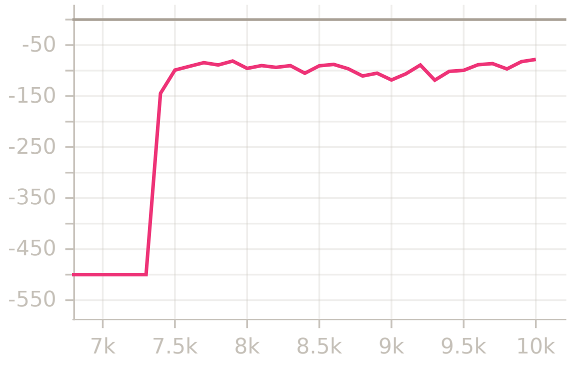

Figure 1: Validation reward during training of the OpenAI (Brockman et al., 2016) Acrobot-v1 task for 10,000 episodes. For every validation, the rewards from 10 episodes were averaged.

The OpenAI Gym (Brockman et al., 2016) integration into JoeyNMT has been tested using environments such as the Acrobot-v1 environment. Fig. 1 shows the validation reward obtained during training. Validations were performed every 100 training episodes by averaging the reward from 10 validation episodes. After stagnating at the minimum reward of -500, the model rapidly jumps in reward and seems to stagnate at a reward of around -100.

3 Reimplementation of HRL

3.1 Reimplementation of baselines

Yao et al. (2019) describe 3 baselines. The baselines do not use reinforcement learning. For each subtask, an independent predictor is trained. Predictors do not interact. The baseline training methods are:

(standard) Supervised training only with recipe descriptions. User answers are not used. For each subtask, a different predictor is trained to predict the subtask label from the recipe description.

rule If the prediction probability (*confidence*) is lower than a certain threshold (the authors propose 0.85), a user answer is appended to the recipe description. Then, another prediction is made. This is repeated until a prediction is made that is confident enough (or until a maximum number of attempts is reached)

sup Based on the `rule`-training, the authors create a custom dataset. For some recipe descriptions and subtasks, they add 2 datapoints to the dataset. One datapoint contains the unaltered recipe description and ”ask user” as label. The other datapoint contains the recipe description together with a user answer and the true label. All other recipe descriptions are included unaltered with their true labels.

Additionally, we train the `standard` baseline with user answers concatenated to the recipe descriptions. This means that user answers are available to the model during training. We do this in order to see what the maximum results are that we could reach. This is similar to the use of an oracle.

We train a model that is similar to LAM (Liu et al., 2016) (the model used in the paper). However, we replace the latent attention by a transformer encoder (taken from JoeyNMT). Precisely, the model consists of the following components: First, the embedded input passes through a transformer encoder. On the output of the transformer encoder we run a bidirectional LSTM. We use the concatenated last hidden states of each direction as input to a 2 hidden layer feed-forward neural network. From this, we obtain prediction probabilities for each label.

For training, we set hyperparameters as follows: All hyperparameters not listed in the table are the standard parameters in PyTorch and JoeyNMT.

Our results are given in Table 2. For the `standard` method, we achieve comparable results to Yao et al. (2019). For `rule` and `sup` methods, our results are much worse than those given in the paper. Note also that the number of questions per recipe description (# Asks) is roughly similar. Finally, we show that theoretically achieving results as given in Yao et al. (2019) is possible with our additional baseline that has access to 1 user answer for all recipe descriptions and subtasks.

Hyperparameter	Value
Embedding dim	50
Transformer layers	2
Self-Attention heads	2
LSTM hidden size	512
LSTM layers	1
MLP hidden size	512
MLP activation	ReLU
Optimiser	AdamW
Epochs	30

Table 1: Hyperparameters for training baselines. The AdamW optimiser was introduced in Loshchilov and Hutter (2018).

	Accuracy	Accuracy C+F	# Asks
standard	0.63	0.36	-
rule	0.74	0.34	4.42
sup	0.78	0.41	2.15
oracle	0.94	-	-

Table 2: Scores on the test set for baselines (including the oracle, which as access to 1 user answer per recipe

3.2 Subtask pretraining

We implemented supervised pretraining methods for the four low-level agents. This step does not implement Reinforcement Learning. The predictions are randomly calculated which prevents a fixed order that may contradict the training’s order of predictions.

We built custom agents for this step. These agents support mini-batching and have the same parameters as the modules that implement Reinforcement Learning. As we are training more than one agent, we decided to use mini-batching as it is a fast and efficient way to train models on a large data set.

The **Ask_User_Ratio** is a customizable parameter that determines how many labels are randomly replaced by the Ask-User label. A recipe description is provided for the replaced label and the prediction is recalculated. This option ensures that the agents learn from the user answers as well as the provided recipe descriptions. This subset of provided recipe descriptions is not static.

If this second prediction is true, the Ask-User label is seen as the true label for the recipe description. If it is false, the true target is regarded as the true label. This means that the correct label for the recipe description is predicted not the Ask-User label.

After each batch, the ratio is multiplied by 0.99 until it reaches a minimum. This results in an agent that asks many questions at the early stages of training and fewer questions with increasing accuracy of the predictions.

Accuracy	0.75
Accuracy C+F	0.37
Average # of Ask-User-Pred per Recipe Description	1.98
Average Reward	1.435

Table 3: Results for Pretraining

3.3 HRL training

3.3.1 Task environment definition

At the beginning of each episode, the environment provides a recipe description. The actions performed by the agents consist in choosing the next subtask (high level policy) or predicting a subtask label (low level policy). Accordingly, the environment returns the current subtask and, optionally, a user answer for the current subtask if requested by a low level policy. When predictions for each subtask have been made, the environment informs the agent about the end of the current episode.

For all training and test runs, we use 5 as the maximum number of times the user can be asked per subtask.

3.3.2 Simulated user definition

At initialization the user simulator generates an answer pool out of the test and train data by searching for certain templates in the recipe descriptions. Then, it saves them according to either their trigger channel and function or their action channel and function. Additionally, those saved descriptions get paraphrased, so the answer pool is more diverse.

If it gets asked, the user simulator's actions depend on the subtask. If the subtask is the trigger/action channel, it will simply return the ground truth which is an approach to a human's natural response (Q: "What is the trigger channel?" A: "Twitter"). If it is asked for trigger/action function it will check if it has the combination of channel and function that is asked in its dictionary and return a randomly chosen answer from it (Q: "What is the trigger function?" A: "If I like a picture on Instagram").

The user simulator has two modes: with dynamic answers and without. If the subtasks' ground truths are not in the answer pool and dynamic answers are off, it will return the original recipe description. If dynamic answers are on, it will try to recognize the relevant part of the description for the subtask and return a paraphrase of it. If it fails to recognize the specific parts it will return a paraphrase of the whole description.

3.3.3 Agent model

Both, the recipe understanding module and the user understanding module, consist of bidirectional LSTMs. Differently from our baseline model and the LAM model suggested in the paper, we do not use any attention mechanism. For both low-level-agents and high-level-agents, predictions are calculated by one hidden layer feed-forward neural network with $\tanh$-activation. All other details are the same as described in the paper.

In case we use a value network, it always is a MLP with 3 hidden layers of 512 hidden units.

We list hyperparameters for training in Tab. 4.

3.4 Results

In Tab. 5, we give results for multiple combinations of hyperparameters. We primarily change the setting of ϵ, γ and β for our ablation studies, where ϵ is for ϵ-greedy sampling, γ is the reward discount factor and β is the reward for asking the user. Tab. 5 shows the results of these ablations studies. These results were obtained with the best performing checkpoint of the model during training, as is typically done during testing in JoeyNMT. Many of these ablation models decreased in performance over time, as can be seen in Fig. 2. Because of this, the best performing checkpoints are those at the beginning of the training, where all ablation models are still quite similar, which leads to the similar results in Tab. 5.

REINFORCE algorithm For the REINFORCE algorithm, we additionally report the validation re-

Hyperparameter	Value
Embedding dim	50
LSTM hidden size	512
LSTM layers	2
MLP hidden size	512
MLP activation	Tanh
Training Epochs	6
Pretraining Epochs	15
Optimizer	SGD
Learning rate	0.001
Clip grad value	1.0
Interpolation w_d	0.5

Table 4: Hyperparamters for training the HRL model. During pretraining, we use the AdamW optimiser instead of SGD. The interpolation factor is the factor that determines how much weight is given to the recipe encoding and how much weight is given to the user answer encoding (see the paper for details).

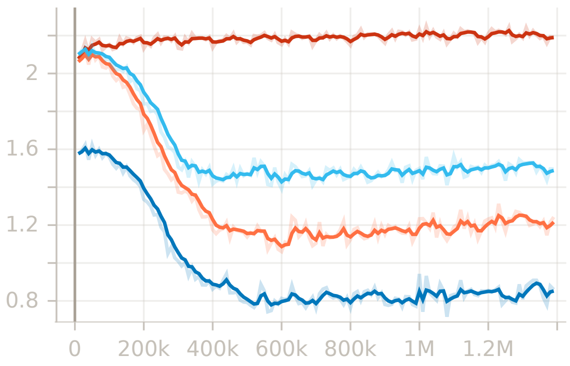

Figure 2: Validation reward during training of the HRL task for about $1.39 \cdot 10^6$ episodes using the REINFORCE algorithm. These curves represent different departures from the default hyperparameter setup. Curves from top to bottom: REINFORCE with multinomial sampler, REINFORCE with ϵ-greedy sampler ($\epsilon = 0.10$), REINFORCE with $\gamma = 0.5$, REINFORCE with $\beta = 0.8$. For every validation, the rewards from 6000 episodes are averaged.

ward during training in Fig. 2. We perform four different studies by altering one of the following default hyperparameters at a time: $\gamma = 0.99$, $\beta = 0.3$, sampler: ϵ-greedy ($\epsilon = 0.05$). As can be seen in Fig. 2, replacing the ϵ-greedy sampler with the multinomial sampler is the only setup to not decrease in reward. This is reflected in slightly higher accuracies (see Tab. 5). Note that Tab. 5 reports the results obtained with the highest performing checkpoint of the model. For those setups where the validation reward decreases, this checkpoint is one saved early during training before the decline in reward.

REINFORCE algorithm with BASELINE We also report the validation reward for the same experiments but with the addition of a VALUE BASELINE in table 5. The results are very similar to just the REINFORCE algorithm and therefore will not be discussed in detail. The same is true for the REINFORCE algorithm with an AVERAGE BASELINE.

Advantage Actor Critic – *A2C* We perform ablation studies for different settings in our training. Overall we observe that *optimizers* other than *stochastic gradient descent* (such as *ADAM*) are not working as well for all settings with *A2C*. The best result (see Tab. 5) is achieved with the setting of using $\epsilon - greedy$ with $\epsilon = 0.1$, discount factor of $\gamma = 0.5$ and a penalty parameter β of 0.8. We observe severe problems with multinomial sampling.

TD Actor-Critic Unfortunately, this algorithm does not seem to work for the HRL task. This is due to divergence of the value network for the high-level policy in early training. We could not definitely locate the source of the problem, but assume that large differences in rewards of subsequent subtasks and greatly differing state-value estimates of the yet untrained value network start self-reinforcing oscillation cycles, always increasing the absolute values predicted by the value network of the high level policy.

4 Conclusion and further work

JoeyNMT now implements reinforcement learning and several algorithms such as REINFORCE and *Advantage Actor Critic*. We present settings for hierarchical reinforcement learning and ablation studies. Further work could include implementation of module tests and implementation of recent policy gradient methods for reinforcement learning such as *Trusted Region Policy Optimization* and *Deep Deterministic Policy Gradients*. The implementation enables further contribution to add functionalities as mentioned above.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym.

Julia Kreutzer, Jasmijn Bastings, and Stefan Riezler. 2019. Joey NMT: A minimalist NMT toolkit for novices. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP): System Demonstrations*, pages 109–114, Hong Kong, China. Association for Computational Linguistics.

Chang Liu, Xinyun Chen, Eui Chul Shin, Mingcheng Chen, and Dawn Song. 2016. Latent attention for if-then program synthesis. In *Advances in Neural Information Processing Systems*, pages 4574–4582.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937.

Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Ziyu Yao, Xiujun Li, Jianfeng Gao, Brian Sadler, and Huan Sun. 2019. Interactive semantic parsing for if-then recipes via hierarchical reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 2547–2554.

RL algorithm	γ (discount factor)	β	sampler	Accuracy	Accuracy C+F	# Asks
A2C	0.99	0.8	ϵ-greedy	0.79	0.49	1.0
A2C	0.5	0.3	ϵ-greedy ($\epsilon = 0.10$)	0.79	0.49	1.0
A2C	0.5	0.8	ϵ-greedy	0.78	0.48	1.0
A2C	0.99	0.8	multinomial	0.52	0.0078	1.0
REINFORCE	0.99	0.8	ϵ-greedy	0.78	0.46	1.0
REINFORCE	0.99	0.3	ϵ-greedy ($\epsilon = 0.10$)	0.78	0.47	1.0
REINFORCE	0.5	0.3	ϵ-greedy	0.77	0.45	1.0
REINFORCE	0.99	0.3	multinomial	0.79	0.49	1.0
Value Baseline	0.99	0.8	ϵ-greedy	0.78	0.46	1.0
Value Baseline	0.99	0.3	ϵ-greedy ($\epsilon = 0.10$)	0.78	0.46	1.0
Value Baseline	0.5	0.3	ϵ-greedy	0.78	0.47	1.0
Value Baseline	0.99	0.3	multinominal	0.79	0.49	1.0
Avg. Baseline	0.99	0.8	ϵ-greedy	0.77	0.45	1.0
Avg. Baseline	0.99	0.3	ϵ-greedy ($\epsilon = 0.10$)	0.78	0.47	1.0
Avg. Baseline	0.5	0.3	ϵ-greedy	0.78	0.47	1.0
Avg. Baseline	0.99	0.3	multinominal	0.79	0.49	1.0
Avg. Baseline	0.99	0.3	ϵ-greedy	0.78	0.47	1.0

Table 5: Results (on test set) for different combinations of hyperparameters. Avg. Baseline is REINFORCE with average-reward baseline. These results are obtained with the best performing checkpoint of the model. γ is the factor used for discounting future rewards, β is the negative reward given when asking the user, sampler is the sampling method used for sampling an action from the categorical distribution returned by the agent. If sampler is epsilon-greedy, $\epsilon = 0.05$ if not further specified. Accuracy is the overall accuracy measured across all predictions independently. Accuracy C+F is the accuracy of recipe descriptions where all 4 predictions are correct. # Asks is the average number of questions per recipe description (not subtask).