

Deep Reinforcement Learning Boosted by External Knowledge

Nicolas Bougie

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda
Tokyo, Japan 101-8430
nicolas-bougie@nii.ac.jp

Ryutaro Ichise

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda
Tokyo, Japan 101-8430
ichise@nii.ac.jp

ABSTRACT

Recent improvements in deep reinforcement learning have allowed to solve problems in many 2D domains such as Atari games. However, in complex 3D environments, numerous learning episodes are required which may be too time consuming or even impossible especially in real-world scenarios. We present a new architecture to combine external knowledge and deep reinforcement learning using only visual input. A key concept of our system is augmenting image input by adding environment feature information and combining two sources of decision. We evaluate the performances of our method in a 3D partially-observable environment from the Microsoft Malmo platform. Experimental evaluation exhibits higher performance and faster learning compared to a single reinforcement learning model.

CCS CONCEPTS

•Computing methodologies → Reasoning about belief and knowledge; Sequential decision making; Neural networks; Partially-observable Markov decision processes;

KEYWORDS

Reinforcement Learning, Object Recognition, External Knowledge, Deep Learning, Knowledge Reasoning

ACM Reference format:

Nicolas Bougie and Ryutaro Ichise. 2018. Deep Reinforcement Learning Boosted by External Knowledge. In *Proceedings of SAC 2018: Symposium on Applied Computing*, Pau, France, April 9–13, 2018 (SAC 2018), 8 pages. DOI: 10.1145/3167132.3167165

1 INTRODUCTION

Reinforcement learning is a technique which automatically learns a strategy to solve a task by interacting with the environment and learning from its mistakes. By combining reinforcement learning and deep learning to extract features from the input, a wide variety of tasks such as Atari 2600 games [14] are efficiently solved. However, these techniques applied to 2D domains struggle in complex environments such as three-dimensional virtual worlds resulting a prohibitive training time and an inefficient learned policy.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SAC 2018, Pau, France

© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM. 978-1-4503-5191-1/18/04...\$15.00
DOI: 10.1145/3167132.3167165

A powerful recent idea to tackle the problem of computational expenses is to modularise the models into an ensemble of experts [10]. Since each expert focuses on learning a stage of the task, the reduction of the actions to consider leads to a shorter learning period. Although this approach is conceptually simple, it does not handle very complex environments and environments with a large set of actions.

A similar idea of extending the information extracted from low-level architectural modules [11] with high-level ones have been previously used in the area of cognitive systems [15] but does not directly relies on RL and was limited to a supervised classification problem. The idea was to leverage information about videos with external ontologies to detect events in videos.

Another technique is called *Hierarchical Learning* [20][1] and is used to solve complex tasks, such as "simulating human brain" [9]. It is inspired by human learning which uses previous experiences to face new situations. Instead of learning directly the entire task, different sub-tasks are learned by the agent. By reusing knowledge acquired from the previous sub-tasks, the learning is faster and easier. Some limitations are the necessity to re-train the model which is time consuming and problems related to catastrophic forgetting of knowledge on previous tasks.

In this paper, our approach focuses on combining deep reinforcement learning and external knowledge. Using external knowledge is a way to supervise the learning and enhance information given to the agent by introducing human expertise. We augment the input of a reinforcement learning model whose input is raw pixels by adding high-level information created from simple knowledge about the task and recognized objects. We combine this model with a knowledge based decision algorithm using Q-learning [24]. In our experiments, we demonstrate that our framework successfully learns in real time to solve a food gathering task in a 3D partially observable environment by only using visual inputs. We evaluate our technique on the Malmo platform built on top of a 3D virtual environment, Minecraft. Our model is especially suitable for tasks involving navigation, orientation or exploration, in which we can easily provide external knowledge.

The paper is organized as follows. Section 2 gives an overview of reinforcement learning and most recent models. The environment is presented in Section 3. The main contribution of the paper is described in Sections 4. Results are presented in Section 5. Section 6 presents the main conclusions drawn from the work.

2 RELATED WORK

Below we give a brief introduction to reinforcement learning and the models used into our system architecture.

2.1 Reinforcement Learning

Reinforcement learning consists of an agent learning a policy by interacting with an environment. At each time-step the agent receives an observation s_t and choose an action a_t . The agent gets a feedback from the environment called a reward r_t . Given this reward and the observation, the agent can update its policy to improve the future rewards.

Given a discount factor γ , the future discounted reward, called return R_t , is defined as follows :

$$R_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'} \quad (1)$$

The goal of reinforcement learning is to learn to select the action with the maximum return R_t achievable for a given observation [19]. From Equation (1), we can define the action value Q^π at a time t as the expected reward for selecting an action a for a given state s_t and following a policy π .

$$Q^\pi(s, a) = \mathbb{E}[R_t \mid s_t = s, a] \quad (2)$$

The optimal policy is defined as selecting the action with the optimal Q-value, the highest expected return, followed by an optimal sequence of actions. This obeys the Bellman optimality equation:

$$Q^*(s, a) = \mathbb{E} \left[r + \gamma \max_{a'} Q^*(s', a') \mid s, a \right] \quad (3)$$

When the state space or the action space is too large to be represented, it is possible to use an approximator to estimate the action-value $Q^*(s, a)$:

$$Q^*(s, a) \approx Q(s, a; \theta) \quad (4)$$

Neural networks are a common way to approximate the action-value. The parameters of the neural network θ can be optimized to minimize a loss function L_i defined as the expected temporal difference error of Equation (3):

$$L_i(\theta_i) = \mathbb{E}_{s, a, r, s'} [(y_i - Q(s, a; \theta_i))^2] \quad (5)$$

where $y_t = r_t + \gamma \max_{a'} Q_{\theta_{target}}(s_{t+1}, a')$

The gradient of the loss function with respect to the weights is the following :

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s, a, r, s'} [(r + \gamma \max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)) \nabla_{\theta_i} Q(s, a; \theta_i)] \quad (6)$$

Mnih *et al.* (2013) used this idea and created the famous method called Deep Q-learning (DQN) [14]. However, the learning may be slow due to the propagation of the reward to the previous states and actions.

Similarly, the value function $V^\pi(s)$ which represents the expected return for a state s following a policy π is defined as follows:

$$V^\pi(s) = \mathbb{E}[R_t \mid s_t = s] \quad (7)$$

Some reinforcement learning models such as *Actor-Critic* or *Dueling Network* decompose the Q-values $Q(s, a)$ into two more fundamental values, the value function $V(s)$ and the advantage function $A(s, a)$ which is the benefit of taking an action compared to the others.

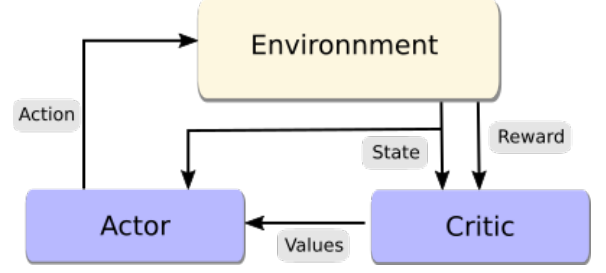


Figure 1: Actor-critic model

$$A(s, a) = Q(s, a) - V(s) \quad (8)$$

2.2 Asynchronous Advantage Actor-Critic (A3C)

It was shown that combining methods of deep learning and reinforcement learning is very unstable. To deal with this challenge, many solutions store the agent's data into a memory, then the data can be batched from the memory. It is done because sequences of data are highly correlated and can lead to learn from its mistakes resulting in a worse and worse policy. A3C [13] avoids computational and memory problems by using asynchronous learning. It allows the usage of on-policy reinforcement learning algorithms such as *Q-learning* [24] or *advantage actor-critic*. The learning is stabilized without using experience replay and the training time is reduced linearly in the number of learners.

The learners of A3C which use their own copy of the environment are trained in parallel. Each process will learn a different policy and hence will explore the environment in a different way leading to a much more efficient exploration of the environment than with a replay memory. A process updates its own policy based on an advantage actor-critic model [8] (Figure 1). The actor-critic model is composed by an actor which acts out a policy and a critic which evaluates the policy. The main thread is updated periodically using the accumulated gradients of the different processes.

The critic takes as input the state and the reward and outputs a score to criticize the current policy. In the case of advantage actor-critic model, the critic estimates the advantage function which requires to estimate V and Q .

The actor does not have access to the reward but only to the state and the advantage value outputted by the critic. Contrary to the critic which is value based, the actor directly works into the policy space and changes the policy towards the best direction estimated by the critic. Optimization techniques such as stochastic gradient descent are used to find θ that maximizes the policy objective function $J(\theta)$. The policy gradient objective function $\nabla_{\theta} J(\theta)$ is defined as follows:

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{\pi, \theta} [\nabla_{\theta} \log \pi_{\theta}(s, a) A^w(s, a)] \quad (9)$$

where $A^w(s, a)$ is a long term estimation of the reward to allow the actor to go in the direction that the critic considers the best.

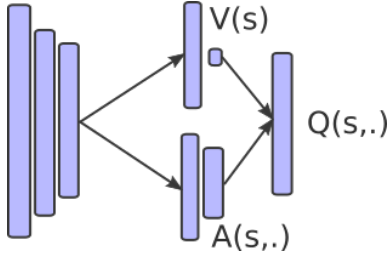


Figure 2: Dueling network architecture

2.3 Dueling Network

The idea is to separately compute the advantage function and the value function and combine these two values at the final layer (Figure 2). The dueling network [23] may not need to care about both values and the advantage at any given time. The estimation of a state value is more robust by decoupling it from the necessity of being attached to a specific action. This is particularly useful in states where its actions do not affect the environment in any relevant way. For example, moving left or right only matters when a collision is upcoming. The second stream, which estimates the advantage function values, is relevant when the model needs to make a choice over the actions in a state. The Bellman's equation (3) becomes now:

$$Q(s, a; \theta, \alpha, \beta) = V(s, \theta, \beta) + (A(s, a; \theta, \alpha) - \max_{a' \in \mathcal{A}} A(s, a'; \theta, \alpha)) \quad (10)$$

And by changing the max by a mean:

$$Q(s, a; \theta, \alpha, \beta) = V(s, \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \alpha)) \quad (11)$$

With θ the shared parameters of the neural network, α the parameters of the stream of the advantage function \mathcal{A} and β the parameters of the stream of the value function V . Since the output of the two streams produces a Q function, it can be trained with many existing algorithms such as Double Deep Q-learning (DDQN) [22] or SARSA [18]. The main advantage is that for each update of the Q -values, the value function is updated whereas with traditional Q-learning only one action-value is updated.

3 TASK & ENVIRONMENT

We built an environment on the top of the Malmo platform [5] to evaluate our idea. Malmo is an open-source platform that allows us to create scenarios with Minecraft engine. To test our model, we trained an agent to collect foods in a field with obstacles. The agent can only receive partial information of the environment from his viewpoint. We only use image frames to solve the scenario. An example of screenshot with the object recognition results is shown in Figure 3.

The goal of the agent is to learn to have a healthy diet. It involves to recognize the objects and learn to navigate into a 3D environment. The task consists in picking up food from the ground for 30 seconds. Food is randomly spread across the environment and four obstacles are randomly generated. Each of the 20 kinds of food

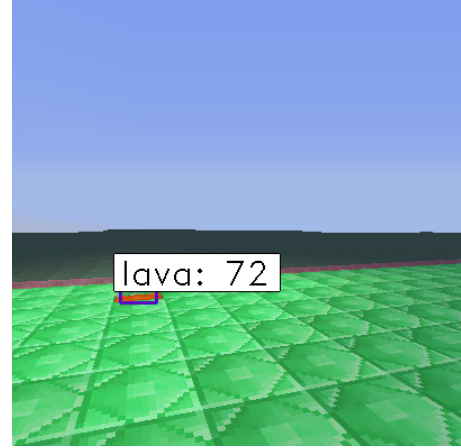


Figure 3: Screenshot of the environment

has an associated reward when the agent picks it up. This reward is a number between +2 (healthy) and -2 (unhealthy). They are distributed equitably, meaning that a random agent should get a reward of 0.

The settings were: window size: 400×400 pixels, actions: *turn left*, *turn right*, *crouch*, *jump*, *move straight* and *move back*, number of objects: 200, number of obstacles: 4. The actions *turn left* and *turn right* are continuous actions to make the learning smoother as consecutive frames are more similar.

4 SYSTEM ARCHITECTURE

4.1 General Idea

Figure 4 describes the global architecture of our new framework called DRL-EK. It consists of four modules: an Object Recognition Module, a Reinforcement Learning Module, a Knowledge Based Decision Module, and an Action Selection Module.

The object recognition module identifies the objects within the current image and generates high-level features. These features of the environment are then used to augment the raw image input to the reinforcement learning module. In parallel, the knowledge based decision module selects another action by combining external knowledge and the object recognition module outputs. To manage the trade-off between these two sources of decision we use an action selection module. The chosen action is then acted by the agent and the modules are updated from the obtained reward.

4.2 Object Recognition Module

Injecting external knowledge requires to understand the scene at a high-level in order to be interpreted by a human. The easiest way to understand an image is to identify the objects. For example, it is intuitive to give more importance to the actions *turn* or *jump* than the action *move straight* when an obstacle is in front of the agent. To recognize the objects, the module uses You Only Look Once (YOLO) [16][17] library which is based on a deep convolutional neural network. As input, we use an RGB image of size 400×400 pixels. YOLO predicts in real time the bounding boxes, the labels and confidence scores between 0 and 100 of the objects. An example

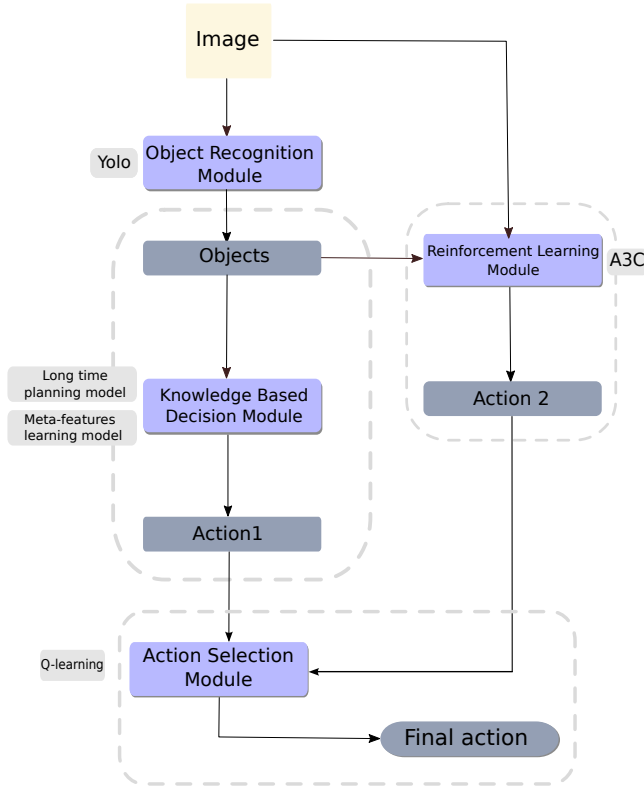


Figure 4: Global architecture of DRL-EK

is shown in Figure 3. We trained YOLO on a dataset of 25 000 images with twenty different classes corresponding to the food that is presented in the environment.

The model is trained off-line before starting the learning into the environment. The neural network architecture is adapted from the one proposed by Redmon *et al.* (2016) for the Pascal VOC dataset [17]. In order to recognize small objects, the size of cells is decreased from 7 to 5 pixels and the number of bounding boxes for each cell is increased from 2 to 4.

In addition to the identified objects, the module creates feature information about the current frame. To generate these high-level abstraction features we combine the recognized objects and external knowledge. They are then used as input by the reinforcement learning module and the knowledge based decision module. We designed two types of features *presence of objects* and *important area*.

4.2.1 Presence Of Objects Features. The first type of features is a vector of booleans which indicates whether an object appears or not within the current image. The size of this vector is the number of different objects in the environment. Since some objects are not helpful to solve the task, we can decide to only take some of the objects into account based on our knowledge about the task.

4.2.2 Important Area Features. As the position of objects is important, we encode information about objects within each area of

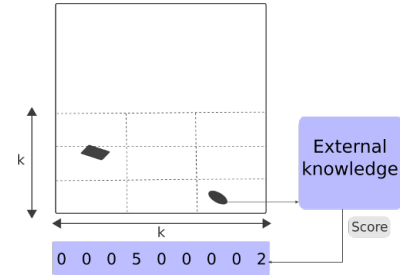


Figure 5: Important areas of an image

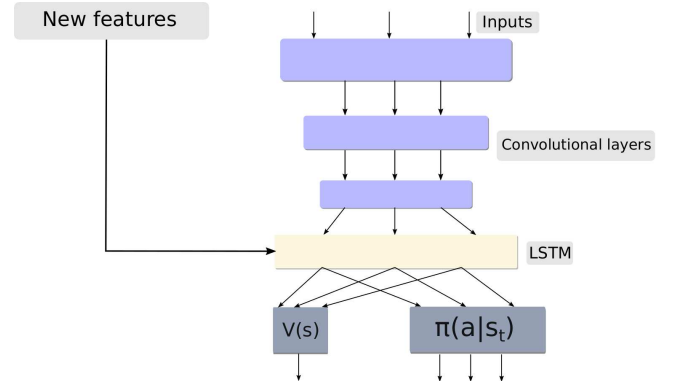


Figure 6: Injection of new features into the reinforcement learning module (A3C)

the image. We split the image into k rectangles vertically and horizontally. So, the number of areas is k^2 and for each one we compute a score (Figure 5). The score of an area is the sum of the score of the objects within this area. External knowledge can be introduced by shaping the score of the objects. From our knowledge about the task, we manually defined the scores to indicate whether or not an object is important to solve the task. To tackle problems with partially observable environments, we keep track of recent information by concatenating the array of scores of the current frame with the arrays of the two previous frames.

In our experiments, the top half of the images only contains the sky so we computed the important area features on the half bottom of the images. We gave a score of -15/+5 to foods we think is unhealthy (cake, cookie) / healthy (meat, fruit) and 0 for the others. That way, if an area contains a healthy food such as a fruit and a sweet food, then the score of the area will be lower than an area containing only a fruit or no object. We set the number of rectangles to 3 (9 areas in total: 3×3). We found that with a higher number of areas the amount of encoded information is bigger but information quality of each area is worse than with 3 areas.

4.3 Reinforcement Learning Module

For a computer, learning from an image is difficult and requires a lot of training steps. To deal with it, the entry point of most of the reinforcement learning models is a recurrent convolutional neural network [3] to extract temporal and spatial features of the image.

We trained a deep reinforcement learning model to perform policy learning and we modified the neural network structure to incorporate external knowledge. In addition to the image input, we injected *presence of objects* or *important area* features which are created by the object recognition module. In the neural network, we give to a Long Short Term Memory (LSTM) [4] the output of the last convolutional layer concatenated with the new features (Figure 6). The next layers of the neural network are two separated fully-connected layers to estimate the value function $V(s)$ and the policy $\pi(a|s_t)$. The purpose is to help the model at the beginning of the training to recognize and focus on objects. The new features augment the raw image input to the reinforcement learning model by adding high-level information. For example, from presence of objects features the model can decide which actions are allowed or not. If a *door* is detected some of the actions may become irrelevant such as *jumping*.

The choice of the reinforcement learning model highly depends on the environment. Since the model at each time-step takes an input and outputs an action, we can easily substitute most of the reinforcement learning techniques such as Deep Q-learning (DQN) [14], Deep Deterministic Gradient Policy (DDPG) [12], Dueling Network [23] or Asynchronous Actor-Critic Agents (A3C) [13] by using a recurrent convolutional neural network as state approximator.

A3C is the most suitable model to solve our task. We tested and empirically searched the best parameters such as a good convolutional neural network architecture and the choice of the optimizer of this model. It provides a baseline to evaluate the importance of each module of our architecture on the final policy.

Working directly with 400×400 pixel images is too computationally demanding. We apply image preprocessing before training A3C. The raw frame is resized to 200×200 pixels. To decrease the storage cost of the images we convert the image scale from 0 – 255 to 0 – 1.

We set the number of workers of A3C to 3 and a convolutional recurrent neural network is used to approximate the states. The reason why we use a recurrent neural network is because the environment is partially observable. The input of the neural network of A3C estimator consists in a $200 \times 200 \times 3$ image. The 4 first layers convolve with the following parameters (filter: 32,32,32,32, kernel size: $8 \times 8, 4 \times 4, 3 \times 3, 2 \times 2$, stride size: 2,2,2,1) and apply a rectifier non-linearity. It is followed by a LSTM layer of size 128 to incorporate the temporal features of the environment. Two separate fully connected layers predict the value function and a policy function, a distribution of probability over the actions. We use RMSProp [21] as optimization technique with $\epsilon = 10^{-6}$ and minibatches of size 32 for training.

4.4 Knowledge Based Decision Module

We believe that the agent is not able to accurately understand and take into account the objects of the environment. A human can easily understand and make a decision from high-level features such as the utility or name of an object. Getting this level of abstraction is difficult but we can help the machine by giving it less low-level information such as color of pixels but more high-level information such as the importance of an area of the image.

Moreover, when the reinforcement learning module is fed with the images and the presence of objects or important areas features, the training time is long due to the size of state space. The knowledge based decision module is able to select an action using external knowledge and high-level features generated by the object recognition module and without direct access to the image. We propose two different approaches, a long time planning model or a meta-feature learning model.

4.4.1 Long Time Planning Model. Our approach to solving the task is based on planning a sequence of actions. We designed and developed a long time planning model without learning which combines traditional test-case algorithms and planning. The model takes as input the probabilities and the bounding boxes of the objects detected within the current frame.

We store in an array the sequence of planned actions. At each time-step, the model checks if the previously planned sequence of actions is still the optimal one and if it is not the case (for example the next action is *jump* but there is no obstacle) the algorithm updates it, otherwise the first action in the array is returned.

To update or plan a sequence of actions, the model uses the information about the objects and manually created rules. First, a test-case verification selects the possible actions. An example of simple rule is, if the object *cookie* is on the left of the image then the action *turn left* is forbidden. Then, to decide of the action among the remaining actions we use a priority list. If the selected action is related to the movement, the model estimates the best angle and the necessary number of steps to perform it. Finally, the first of the planned actions is returned.

To decrease the number of rules we discretized the image space into four areas: *center*, *left*, *right*, *other*. We designed 43 rules to prevent the agent from going in the direction of the food we think is dangerous. To avoid static behaviour, we give more priority to the actions *turn left*, *turn right*, *move straight* than the others in the priority list.

4.4.2 Meta-feature Learning Model. In our previous approach, we manually create rules to reason on high-level features. To automatically learn the rules and select the optimal action from them, we use a deep reinforcement learning model such as DQN or dueling network. Unlike the reinforcement learning module which uses the image, the only input is high-level features such as *important areas* or *presence of objects*. As the input is much smaller than an image, a simple neural network can be trained to approximate the states. The smaller number of parameters leads to a faster learning than a model trained from visual information.

In experiments, we trained a dueling network combined with a double deep Q-learning (DDQN). It empirically gives a smoother learning than most of the other reinforcement learning models. A neural network approximates the states. It consists in 3 fully connected layers of size 100 with a rectifier nonlinearity activation function. Network was trained using the *Adam* algorithm [7], learning rate of 10^{-3} and minibatches of size 32. As input, we used a slightly modified version of the important area features outputted by the object recognition module. To create important area features, we filtered the objects too far and the objects with a confidence score less than 0.25. Taking into account an object such as *grass* is

irrelevant and makes the learning more difficult. We only used dangerous or very healthy (10 objects out of 20) objects and removed 2 objects that we know are difficult to distinguish.

4.5 Action Selection Module

The module aggregates the actions proposed by the reinforcement learning module and the knowledge based decision module to select the action that the agent will perform in the environment. The goal is to take advantage of the fast learning of the knowledge based decision module and the quality of the policy learned by the reinforcement learning module. An important aspect of the action selection, is selecting an action with the highest expected return but also detecting error patterns to correct them. An error must be detected when an action which has not been proposed could offer a higher *return*.

This is achieved by training a Deep Q-learning model to select the best action, detect and correct the error patterns. There is no restriction on the possible actions meaning that the final action may be different from the two proposed actions if an error is detected. We encode the proposed action by the two modules into two indicator vectors. An indicator vector is a binary vector with only one unit turned on to indicate the recommended action. The neural network input is the concatenation of these two vectors.

The Q-learning algorithm is trained using a Boltzmann distribution (Equation 12) as explorer and experience replay (each experience is stored into memory and the algorithm is run on randomly sampled batch) with a memory of size 10^6 . Equation 12 gives the probability of selecting an action in a given state s .

$$P_s(a) = \frac{\exp(Q(s,a)/\tau)}{\sum_{a' \in A} \exp(Q(s,a')/\tau)} \quad (12)$$

The Q-network is composed of 2 hidden fully connected layers of size 50 and are followed by rectified linear units.

5 EXPERIMENTS

We conducted several experiments for evaluating our architecture. In all our experiments, we set the discount factor to 1.0. According to our different tests, on average the best reward that a perfect agent can get in 30 seconds is 9.

5.1 Object Recognition

We evaluated our object recognition module for understanding the correctness of obtained object information in the environment. Figure 7 reports the object recognition module performance. In this experiment, we measured the mean average precision (mAP) as the error metric. The results are similar to the results presented by the authors (Redmon *et al.*, 2017) [17] on the Pascal VOC. dataset [2]. We obtained a mean average precision of 53.47. Although other libraries could offer higher performance, the real-time detection was the main criterion for selecting YOLO.

We noticed that most of the errors are false positives (68.3%) whereas the false negatives (31.7%) are uncommon. It leads to an agent with a policy more greedy and safer. As shown in the figure, the average precision is similar for every class. The performance of YOLO is slightly affected by the complexity of the objects such as their shape, color or size.

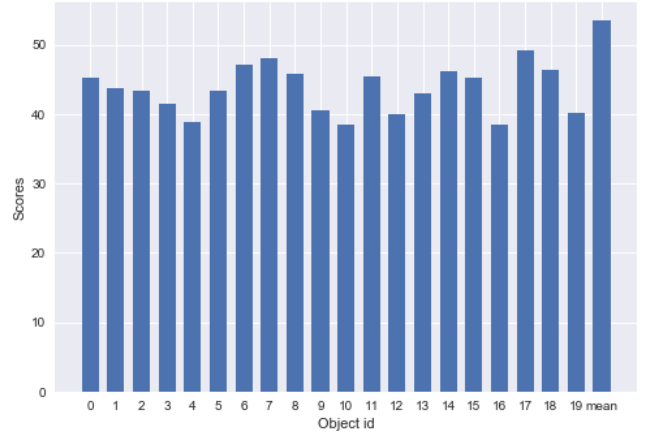


Figure 7: Average precision over all the classes obtained by the object recognition module

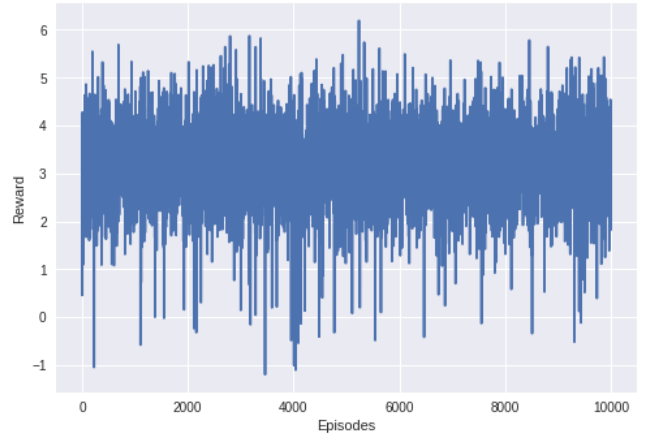


Figure 8: Evolution of the reward of the long time planning model

5.2 Long Time Planning Model

Next, we tested the long time planning model in the knowledge based decision module to evaluate the effectiveness of this approach. We only utilized the object recognition module and the knowledge based decision module in our framework. The long time planning model performs much better than a random agent with an average reward of 3.3 (Figure 8), since expected reward of random agent is 0.

The most likely cause of the wide variance is the difficulty to handle all possible cases with manually created rules. For instance, the agent has difficulty in gathering food near obstacles. Since there is no learning, the quality of the agent only depends of the quality of the rules and is not able to converge. On the other hand, from the first episode the average reward is much higher than any other learning based models.

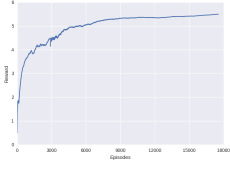


Figure 9: Average rewards of the meta-feature learning model with different parameters. The rewards were averaged over 200 episodes after 5000 training episodes

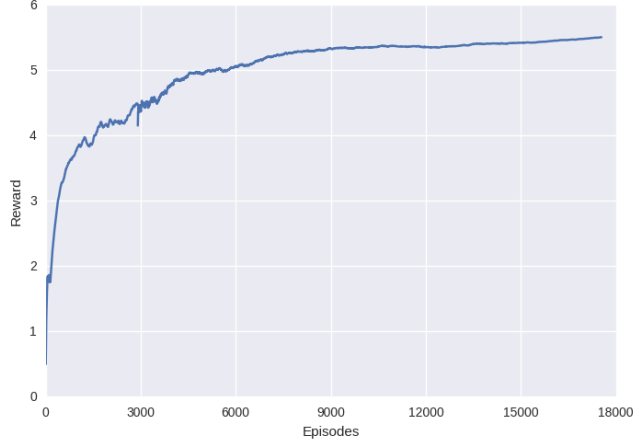


Figure 10: Average reward using meta-feature learning

5.3 Meta-feature Learning Model

To evaluate performance of the meta-feature learning model we use as baseline the long time planning model. We optimized its parameters, by sampling hyper-parameters from categorical distributions:

- Number of areas sampled from {4, 9, 16, 25}
- Number of hidden layers from {1, 5}
- Size of hidden layers sampled from {25, 50, 100, 200, 300}

Figure 9 reports an example of hyper-parameter optimization results. Each cell corresponds to a configuration of parameters. As can be seen on the figure, a number of hidden layers larger than two or a large number of areas results in lower performance. The best hyper-parameters are 9 areas, and a neural network with 3 hidden fully-connected layers of size 100. Training time is about 4 hours for each configuration on a Nvidia Titan-X GPU.

Figure ?? shows how the average total reward of the meta-feature learning model evolves during training with the optimal settings. The dueling network architecture effectively learns to solve the task from the important area features. The learning is fast during the first 3000 episodes and the average reward quickly converges around 5.4. It is also interesting to note that this approach rapidly achieves higher performance than the long time planning model. Automatic rule learning is more effective than manual rule construction. Unfortunately, the rules cannot be represented in a human-interpretable way.

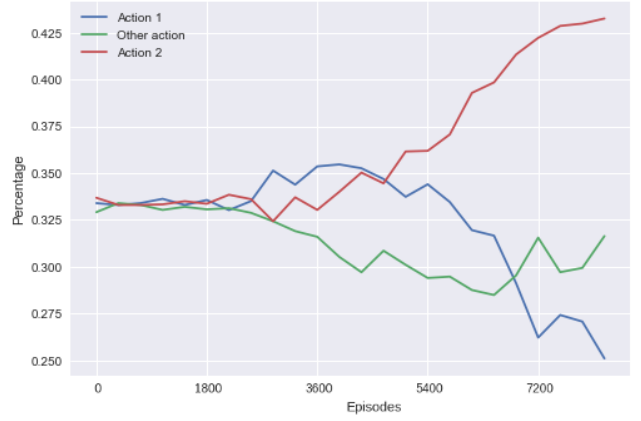


Figure 11: Frequency of selection of each action

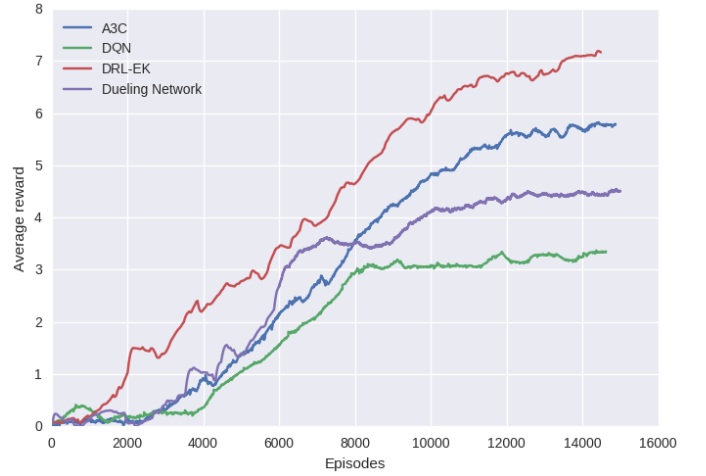


Figure 12: Performance of DRL-EK comparing to DQN, Dueling Network and A3C

5.4 Action Selection

We also evaluated the characteristics of the action selection module. We report the percentage of actions which is selected from the knowledge based decision module (action 1) and from the reinforcement learning module (action 2) against other actions. We measured the frequency of selection of each action every 350 episodes. As shown in Figure 11 the action selection module at the beginning selects equally the actions then more the action 1 and gradually give more importance to the action 2. The results confirm our intuition, the module selects the action of the most efficient module and adapts over time the trade-off between the sources of decision to always select the best one.

5.5 Global Model Evaluation

Finally, we report the average reward of our whole framework trained using the injection of important areas features into the reinforcement learning module and a meta-feature learning model

Table 1: The table compares average reward for various features injected into A3C. The reinforcement learning module was evaluated alone for 12 000 episodes.

Settings	Rewards
A3C	5.6
A3C + <i>presence of objects features</i>	5.8
A3C + <i>important area features</i>	6.1

as knowledge based decision module. Figure 12 compares our proposed method with the best performing reinforcement learning methods. These models learn the policy only using raw pixels.

DRL-EK boosts A3C by injecting important area features. To select these features, we compared *A3C+presence of objects features* and *A3C+important area features* (Table 1). In both cases, the results show that adding a new input to the reinforcement learning module improves the quality of the policy.

As can be seen, DQN gives the worst results with an average reward of 3.2, $\approx 40\%$ less than A3C after converging. After 12 000 episodes, the average reward of the dueling network architecture trained with a double deep Q-learning is around 4.4 while A3C is able to achieve an average reward of 5.6. Surprisingly the meta-feature learning model trained alone (Figure 10) achieves higher performance than learning only from the image with a dueling network or a DQN model.

Asynchronous advantage actor-critic tends to learn faster than any other reinforcement learning based models. We believe this is due to the 3 parallel workers of A3C which offer a nonlinear significant speedup.

These results show that our architecture, DRL-EK, outperforms the baselines. Its average reward is around 15% better than A3C after 14 000 episodes. Moreover, the performance at the beginning of the training and the learned policy of DRL-EK is significantly better than all other models. One thing to note is that the action selection module tends to select an action different from action 1 and 2 (Figure 11). The continuous increase of the average reward of DRL-EK and this observation indicates that the action selection module is partially able to learn to correct the errors.

The experiments demonstrate the importance of each module of our system. With an average time for one step of 0.43 seconds on a Nvidia Titan-X (Pascal) GPU, DRL-EK can be trained in real time.

6 CONCLUSION

We proposed a new architecture to combine reinforcement learning with external knowledge. We demonstrated its ability to solve complex tasks in 3D partially observable environments with image as input. Our central thesis is enhancing the image by generating high-level features of the environment. Further benefits stem from efficiently combining two sources of decision. Moreover, our approach can be easily adapted to solve new tasks with a very limited amount of human work. We have demonstrated the efficacy of our architecture to decrease the training time and to learn a better and more efficient policy.

In the future, a promising research area is building an agent that incorporates human feedback. Another challenge is how to

integrate complex and structured external knowledge such as ontologies or textual data into our model. Finally, we are interested in extending our experiments to new environments such as VizDoom [6].

REFERENCES

- [1] Andrew G. Barto and Sridhar Mahadevan. 2003. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems* 13, 4 (2003), 341–379.
- [2] Mark Everingham, Luc J. Van Gool, Christopher K. I. Williams, John M. Winn, and Andrew Zisserman. 2010. The Pascal Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision* 88, 2 (June 2010), 303–338.
- [3] Matthew J. Hausknecht and Peter Stone. 2015. Deep Recurrent Q-Learning for Partially Observable MDPs. *CoRR abs/1507.06527* (2015).
- [4] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.
- [5] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. 2016. The Malmo Platform for Artificial Intelligence Experimentation. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, NY, USA, 9-15 July 2016*, Subbarao Kambhampati (Ed.). IJCAI/AAAI Press, 4246–4247.
- [6] Michal Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaskowski. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *Proceedings of IEEE conference on Computational Intelligence and Games (CIG)*. IEEE, 1–8.
- [7] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* (2014).
- [8] Vijay R. Konda and John N. Tsitsiklis. 1999. Actor-Critic Algorithms. In *Proceedings of Neural Information Processing Systems*, Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller (Eds.). The MIT Press, 1008–1014.
- [9] Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. 2016. Building Machines That Learn and Think Like People. *Behavioral and Brain Sciences* (2016), 1–101.
- [10] Guillaume Lample and Devendra Singh Chaplot. 2017. Playing FPS Games with Deep Reinforcement Learning. In *Proceedings of AAAI*. 2140–2146.
- [11] Antonio Lieto, Christian Lebiere, and Alessandro Oltramari. 2017. The knowledge level in cognitive architectures: Current limitations and possible developments. *Cognitive Systems Research* (2017).
- [12] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *ArXiv e-prints* (Sept. 2015). arXiv:cs.LG/1509.02971
- [13] 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 *Proceedings of International Conference on Machine Learning*. 1928–1937.
- [14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. *ArXiv e-prints* (Dec. 2013). arXiv:cs.LG/1312.5602
- [15] Oltramari and Lebiere. 2012. Using Ontologies in a Cognitive-Grounded System: Automatic Action Recognition in Video-Surveillance. (2012).
- [16] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. 2015. You Only Look Once: Unified, Real-Time Object Detection. *CoRR abs/1506.02640*. <http://arxiv.org/abs/1506.02640>
- [17] Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. *CoRR abs/1612.08242* (2016).
- [18] G. A. Rummery and M. Niranjan. 1994. *On-Line Q-Learning Using Connectionist Systems*. Technical Report. University of Cambridge.
- [19] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. MIT press Cambridge.
- [20] Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J. Mankowitz, and Shie Mannor. 2017. A Deep Hierarchical Approach to Lifelong Learning in Minecraft. In *Proceedings of AAAI Conference on Artificial Intelligence*. 1553–1561.
- [21] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning* (April 2012).
- [22] Hado van Hasselt, Arthur Guez, and David Silver. 2015. Deep Reinforcement Learning with Double Q-learning. *CoRR abs/1509.06461* (2015).
- [23] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, Maria-Florina Balcan and Kilian Q. Weinberger (Eds.). 1995–2003.
- [24] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292.