AA228 Project Report
Transfer of Q values across tasks in Reinforcement Learning

Sebastien DUBOIS
sdubois@stanford.edu

Guillaume GENTHIAL
genthial@stanford.edu

Abstract

This project investigates methods to transfer knowledge between *source* and *target* tasks within the framework of Q-learning with global approximation. Given an environment, a task is defined as a sub-domain of states-actions, altered dynamics and a specialized objective (reward function). First, we train agents on simple *source* tasks. Then, we use this knowledge to train an agent on a more complicated *target* task. Our models learn the *target* Q-function using the *source* Q-functions as features and some (light) interaction with the environment. We tested our algorithms against the mountain-car benchmark and observed general improvement in both speed and performance. Application to Deep Q Learning is also discussed.

Introduction

The idea of transfer learning is to build knowledge from a set of similar *source tasks* and apply it to a *target task*. The benefits of transfer include improved initial or asymptotic performance, reduced learning time... For instance, we can imagine that an autonomous car could benefit from the knowledge of different tasks, such as avoiding obstacles, driving efficiently, etc., instead of having to learn every task from scratch at the same time.

As stated in (Lazaric, 2012), we can distinguish 3 categories of transfer learning: 1. Inductive transfer learning: transfer from a single source task to a target task with a fixed domain (state and action space). Source and target tasks can differ by their transition and reward functions. 2. *Transfer across tasks with fixed domain*: generalize from a set of source tasks to improve target task performance. 3. *Transfer across tasks with different domains*.

We want to address the case where our target task is a difficult, real-world problem for which we can't afford to learn directly in this environment (eg driving a plane), but for which we can design imperfect models and simulations (source tasks). We restrict our work to source tasks for which the state-action domain is a subset of the target state-action domain. As we can interpret a subset of state-action as changing the dynamics, we fall under the scope of *transfer across tasks with fixed domain*

Related Work

(Lazaric, 2012) and (Taylor and Stone, 2009) provide an overview of transfer learning in reinforcement learning. (Lazaric et al., 2008) studies transfer of samples in Batch Reinforcement Learning, evaluating task compliance and sample relevance. (Taylor et al., 2007) propose a method to map states and action between source and target tasks for temporal difference learning. On the related issue of multi-task reinforcement learning, (Borsa et al., 2016) synthesize methods to learn shared representation for value functions, exploiting the shared structure between tasks.

More recent work makes use of deep learning to estimate the Q function of the target task. A starting point on transfer of parameters in deep neural network is (Yosinski et al., 2014). Actor-Critic method (Parisotto et al., 2015) and policy distillation (Rusu et al., 2015) use model compression techniques. (Rusu et al., 2016) introduce progressive networks, an architecture for neural networks that adds layers as we keep discovering new tasks, while making use of the former learned tasks.

Models

Background and notation

We model an environment as a Markov decision process $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ where $\mathcal{S}$ are the states, $\mathcal{A}$ the actions, $\mathcal{T} : s', (s, a) \mapsto \mathbb{P}(s'|s, a)$ the transition function, $\mathcal{R} : (s, a) \mapsto R(s, a)$ the reward function and γ a discount factor.

Given a policy $\pi : s \mapsto a$, we can define a Q value function $Q_\pi : (s, a) \mapsto Q(s, a)$ as the expected discounted reward over all paths starting in s while taking action a,

$$Q^\pi(s, a) = \mathbb{E}_{\pi,\mathcal{T}}[\sum_{t=0}^{\infty} \gamma^t r_t]$$

The best policy is the one that maximizes our expected reward. The idea of Q-learning is that we can iteratively compute the optimal Q-value function thanks to the Bellman equation. On a transition (s, a, r, s'),

$$Q(s, a) = r + \gamma * \max_{a'} Q(s', a')$$

Then, we have the following update rule used while interacting with the environment, using a learning rate η :

$$Q(s,a) \leftarrow Q(s,a) + \eta(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

Approach

Assuming that we can easily compute (or we are given) the Q functions for the source tasks, our goal is to learn a Q function for the target task, using the source Q functions and some (light) interaction with the target environment.

First, we consider that the Q function of the source tasks can be written using global approximation. For a source s_i we will write $Q_i(s,a) = \theta_i \cdot \beta(s,a)$, where θ_i is a weight vector and $\beta(s,a)$ is a feature vector. The feature function β is shared across the different sources and target tasks, even though the state-action spaces may not be the same.

Within this framework, we study different methods Φ to estimate $Q_T(s,a) = \Phi(Q_i(s,a), s, a)$. Then, we investigate Deep Q learning.

Fixed linear combination

Our first idea was to learn the target Q function as a linear combination of the Q_i:

$$Q_T = \sum_i \alpha_i Q_i \iff \theta_T = \sum_i \alpha_i \theta_i \tag{1}$$

We see three advantages of this model

1. the representation is very compact: we only need to store the sources' θ_i at learning time and can compute a single θ for deployment
2. if knowledge is transferable with such a model, then the transfer should be learned quickly since we have few parameters to learn: it depends only on the number of sources (versus the dimension of the state-action space in classic Q-learning approaches)
3. we can easily interpret the α_i as a measure of task compliance.

We implemented this method via stochastic gradient descent updates on the coefficient vector α. Precisely, while the standard update when observing (s,a,r,s') is

$$\theta \leftarrow \theta - \eta \left(Q(s,a) - (r + \max_{a'} Q(s',a')) \right) \beta(s,a) \tag{2}$$

our update is

$$\alpha_i \leftarrow \alpha_i - \eta \left(Q(s,a) - (r + \max_{a'} Q(s',a')) \right) Q_i(s,a) \tag{3}$$

where Q denotes the current estimate for the target's Q function.

Adaptive linear combination

The problem with the previous model is its lack of expressiveness since it computes a Q function as a weighted sum of the sources' Q functions - *with fixed weights* (once learned). However, consider the case where we can split our target domain into sub-domains. On each sub-domains, the target task is very similar to one of the source tasks (not always the same). In this situation, the specific source task conveys a lot of information that should be transferred to the target, and in particular we would like $Q_T(s,a)$ to select the relevant Q_i.

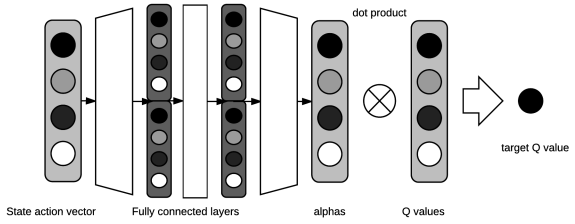

Figure 1: Adaptive linear combination network

Motivated by this example, we considered Q_T of the form

$$Q_T(s,a) = \sum_{i=1}^{n} \alpha_i(s,a) Q_i(s,a) \tag{4}$$

We decided to model these functions α_i with fully connected neural networks. We obtained good performance with a shared fully connected neural network with 2 hidden layers of size $2n$ with $sigmoid$ activation. The output of the network is a vector of size n corresponding to the α_i . It allows us to model complex functions while using the same gradient descent framework based on the temporal difference squared error. See Figure 1.

More specifically, when observing a transition (s,a,r,s'), we use the network to compute $y = (r + \max_{a'} Q_T(s',a')$. Then, we use y to update our network using a gradient descent algorithm on the squared loss $(y - Q_T(s,a))^2$. We used the double-Q learning trick (use old weights to compute y) and both SGD and RMSProp for gradient descent.

In the SGD case, our update rule for the weights w of the network is

$$w \leftarrow w - \eta \left(Q_T(s,a,\tilde{\mathbf{w}}) - (r + \max_{a'} Q_T(s',a',\mathbf{w})) \right) \frac{\partial Q_T(s,a,\mathbf{w})}{w} \tag{5}$$

Such a model has the advantage of being more *selective* on the source task via its adaptive linear combination. However, it requires to store all the θ_i because we cannot compute explicitly a θ_{target} as a function of the θ_i.

Using Q_i as features

The last model has the advantage of keeping the number of parameters reasonably low while providing good expressiveness by selecting relevant source tasks. A natural idea to go a step further is to consider the Q_i as features to compute Q_T, and consider more general functions than adaptive linear combination.

$$Q_T(s,a) = \Phi(Q_1(s,a), ..., Q_n(s,a), s, a) \tag{6}$$

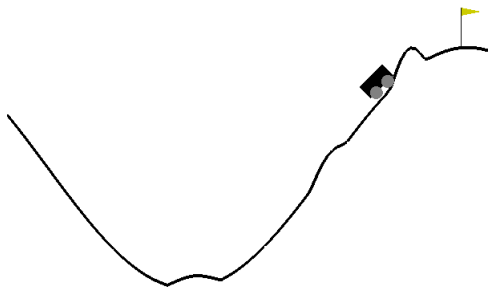

Figure 2: Screenshot of the mountain car environment

where Φ can be any function. Note that this formulation include the two previous models.

We considered a variant of the adaptive linear combination. With the same architecture, we compute $\alpha_i(s,a)$. Then, we consider the weighted vector $(\alpha_1(s,a)Q_1(s,a), ..., \alpha_n(s,a)Q_n(s,a))$ (this weighted vector echoes attention-based models). We then use this vector as input of a 2-layer fully connected neural network whose output is the desired Q-value. We tested two variants of the architecture, changing the activation from *sigmoid* to *relu*.

Deep Q-learning

These methods can also be applied to the case where we write the source Q functions Q_i as deep Q networks. We use the Q_i as features like the previous section. We can also consider a more powerful feature extraction, using the last hidden layer of the Q_i networks as input features of the target network. At training time on the target, we can also choose to back-propagate the gradients to the source networks. This approach echoes (Parisotto et al., 2015) who use compression techniques to link the target network to the source networks. The difference here with their approach is that we don't look for network compression and plug every source network into a bigger target network.

Experiment

We used python and Theano to implement our models on a modified mountain car environment from OpenAI Gym. The code can be found on github[1].

Test environment

We used the mountain car environment from OpenAi's Gym (see Figure 2), to which we made the following changes:

- possibility to control the slope, maximum speed, power of actions (ie relation between action value and speed up), actions (from a_0 to a_n instead of -1/0/1)
- rewards can be based on: *time*, *energy* (penalty proportional to the action's square), *distance* to the goal along x-axis, *center* (distance to the center), *height*, *still* (penalty proportional to the velocity's square).

[1]https://github.com/GuillaumeGenthial/Q-transfer

Note that some tasks, like *still*, do not encourage the car to go to the flag. In addition we modified the dynamics of the environment by adding *obstacles* and *bananas*:

- *obstacles* are bumps or holes with pre-defined positions, and are characterized by their width and magnitude. These impact the trajectory of the car as it would in real life.
- *bananas* refer to random slowdowns which can arise on the car's trajectory. Those are set for a task by probability of slipping at each time step and magnitude of the slowdown.

An episode ends after we reach the goal (the flag) or after 1000 timesteps.

Tasks

We defined the following tasks, see Table 1.

Task	Reward	Actions	Obst.	Bana.
time	time	[-1, 1]	Yes	No
more_actions	time	[-2,2]	Yes	No
more_actions1	time	[-2,2]	No	No
more_actions2	height	[-2,2]	No	No
energy	energy	[-1, 1]	No	No
distance	distance	[-1, 1]	No	No
height	height	[-1, 1]	No	No
still	still	[-1, 1]	No	No
bumps	time	[-1, 1]	Yes	No
bananas	time	[-1, 1]	No	Yes
full	all	[-2, 2]	Yes	No

Table 1: Task description

Evaluation

To evaluate performance of a policy, we used Monte-Carlo policy evaluation with 1000 starts at random. We report the mean and standard deviation. We use a discount factor of 1.

Implementation

Global approximation We update the weight vector θ as described in the Fixed linear combination section.

For our environment, the states s are a pair of position and velocity (pos, vel), which are both floating point numbers. There is a finite number of possible actions a. We consider a grid of size $(100, 100)$ of the position-velocity space. For a given state, we consider i_{pos} and i_{vel} the coordinates in the grid. We construct the feature vector $\beta(s,a)$ as an indicator vector whose entries are

$$(i_{pos}, i_{vel}, a), (i_{pos}, a), (i_{vel}, a), a$$

We used ϵ greedy exploration strategy with an exponentially decreasing ϵ, from 1 to 0.2 and a constant learning rate $\eta = 0.1$.

Adaptive and Q_i as features

We initialized the matrices of the different networks using Glorot Normal method and trained them using RMSProp with learning rate 0.001. Bias vectors were set to zero. We normalized each Q-value between -1 and 1 to prevent scaling issues.

Deep Q Networks We also implemented a deep Q network that takes as input a completed state (position, velocity, height and acceleration) and returns the Q-values for each action. We obtained good performance for the source agents using 2 hidden layers of size 512 and 256 with leaky activation $(\max(x, 0.01x))$. We trained the network with RMSProp with learning rate 0.001 and a memory replay of size 100000. We stopped training after 10 minutes. For technical details about deep Q learning, read (Mnih et al., 2013).

Metrics

In theory, given a budget of trials on the target task, we would evaluate the performance of the transfer through the following metrics:

- *jumpstart* measures the initial performance compared to the non-transfer agent.
- *asymptotic performance* measures the final performance compared to the non-transfer agent.

However in practice, we report the average performance of a policy obtained after N_t learning trials , for different values of N_t. Thus we can estimate *jumpstart* for small values of N_t and compare what the *asymptotic performance* would be for different budget N_t.

Results

Performance on source tasks

We report the performance of both global approximation and deep Q network on some of the source tasks. Theses score are obtained for one training and are indicative.

Task	Global	Deep
bananas	**-157**(1)	-189 (3)
energy	0 (0)	0 (0)
height	**562** (2)	71
bumps	-117 (1)	**-99** (2)
more power	-50 (1)	**23** (0)
standard	-142 (1)	**-117** (1)
distance	-125 (1)	**-104** (3)

Table 2: Score on tasks for Global Approximation and Q-networks $(mean(std))$. Bold is best.

As we can see, global approximation and Deep Q networks achieve similar performance. For speed reasons, we decided to test our transfer methods on global approximation.

Jumpstart performance

Using target task *full* and all other defined tasks as sources, we observe the initial performance (after 2 trials), for the different techniques. We average the performance over 6 experiments.

Transfer method	performance
Linear	-407
Adaptive linear	**-238**
Q_i features, *sigmoid*	-449
Q_i features, *relu*	-340
direct learning	unsuccessful

Table 3: Jump start performance after 2 episodes

As we could expect, learning the target task from scratch is not achieved in 2 episodes, whereas transfer methods achieve reasonable score. We observe that the more reliable method for jumpstart performance is the **adaptive linear** method, which combines flexibility and a low number of parameters. Linear method often performs very well, but seems to suffer from negative coefficients, an event which sometimes can occur. Further investigation is needed to understand and resolve this issue. Surprisingly, the method using the Q_i as feature and $relu$ activation performs very well in some cases (-135 !!) explaining its good score.

Task compliance

We conducted the following experiment

- sources: bumps, energy, distance, height, still, more_actions1
- target: more_actions

The source task `more_actions1` is really close to the target task `more_actions` and this was indeed transferred to the coefficients learned by the fixed combination method. These were as follows: *more_actions1*: 0.9048, *energy*: 0.0151, *height*: -0.0399, *bumps*: 0.0396, *distance*: 0.0396, *still*: 0.0407.

We could also plot the adaptive coefficients and observe a similar behavior. This analysis proves that our framework is able to understand task compliance *without relying on expert knowledge*, just by interacting with the environment.

Asymptotic performance

With the same source and target as before, we plot the performance of the transfer methods and the direct methods on figure 3.

On this special case we notice, as expected, that the transfer methods are very effective at transferring knowledge efficiently in a small amount of steps. However, after a few number of trials, the direct method outperforms the transfer agent.

We also conducted the following experiment:

- Sources: bananas, bumps, still
- Target: time

Here, the target task is the standard one, and the sources one are either not relevant (still encourages the car to stay in the valley), or more complicated than the target. We plot the

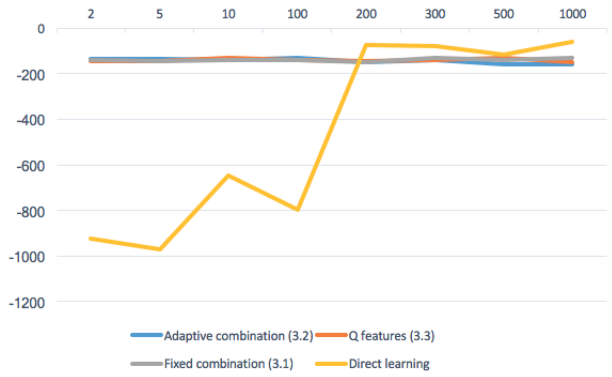

Figure 3: Performance over time

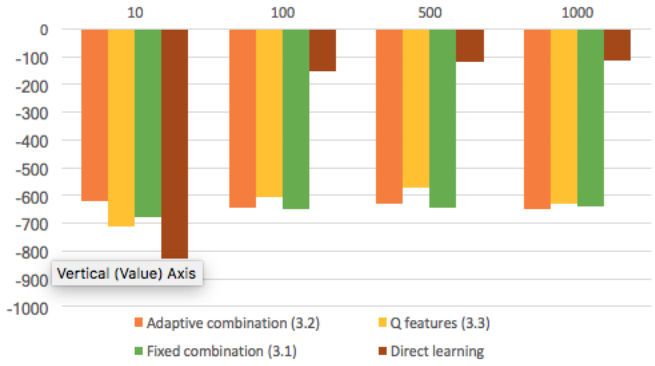

Figure 4: Performance evaluated at different steps

policy evaluation at different time step of training on Figure 4.

As we would expect, we obtain much better performance with the direct learning in this case. We observe that the different transfer methods achieve similar performance on this experiment. Other experiments on different sources and targets seem to show that the most reliable method is still the adaptive combination methods, which combines flexibility and efficiency.

Deep Q networks

As we were able to train deep Q networks on the source tasks, we experimented transfer. We plug the last hidden layers of the source network into a fully connected network with one hidden layer of size 512 and relu activation. We tried two approaches. One, to only train the parameters of the last layer. Two, to retrain all parameters, of both the source and the target networks. At the time of writing this report, experiments are still running, but we were not able to reproduce the transfer performance achieved with the previous methods. Then, the question is still open and could lead to further work.

Discussion

First we observe that the transfer methods described in this project are able to learn useful Q functions for the target task. In most cases, the adaptive combination approach almost always outperformed the simple fixed combination, even after a single learning trial.

Surprisingly, all our methods converge after a couple learning trials which means that there is opportunity for much more complex forms of transfers. On the other hand, this means that our transfer methods can be used as Q function initialization for a standard Q-learning method.

As we noticed that the direct method, if trained long enough, outperforms the transfer method, a compromise would be to jump start the agent with transfer and after a few iterations switch back to standard training. Our methods, which combine global approximation and neural networks could be applied to this idea. For the linear method, we only need to switch back to global approximation with the estimated θ parameter. For the adaptive and Q-features method, more work would be required to back-propagate the gradients.

Having this in mind, using a stack of neural networks for both the input Q-value functions and the transfer would make joint training easier to implement, by simply back-propagating the gradients to the source networks. We could also imagine hybrid methods : use the linear combination to estimate task compliance, then select the most relevant source task and initialize a deep Q network with this source's weights and finally train this network on the target task.

This project also underlines the lack of a standardized test framework for transfer learning. The reinforcement learning community would benefit from such a framework, as there are obvious advantages in transfer learning, as this project investigates.

References

[Borsa et al., 2016] Borsa, D., Graepel, T., and Shawe-Taylor, J. (2016). Learning shared representations in multi-task reinforcement learning. *CoRR*, abs/1603.02041.

[Du et al.,] Du, Y., Gabriel, V., Irwin, J., and Taylor, M. E. Initial progress in transfer for deep reinforcement learning algorithms.

[Lazaric, 2012] Lazaric, A. (2012). *Transfer in Reinforcement Learning: A Framework and a Survey*, pages 143–173. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Lazaric et al., 2008] Lazaric, A., Restelli, M., and Bonarini, A. (2008). Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 544–551, New York, NY, USA. ACM.

[Mnih et al., 2013] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

[Parisotto et al., 2015] Parisotto, E., Ba, L. J., and Salakhutdinov, R. (2015). Actor-mimic: Deep multitask and transfer reinforcement learning. *CoRR*, abs/1511.06342.

[Rusu et al., 2015] Rusu, A. A., Colmenarejo, S. G., Gülçehre, Ç., Desjardins, G., Kirkpatrick, J., Pascanu, R., Mnih, V., Kavukcuoglu, K., and Hadsell, R. (2015). Policy distillation. *CoRR*, abs/1511.06295.

[Rusu et al., 2016] Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. (2016). Progressive neural networks. *CoRR*, abs/1606.04671.

[Taylor and Stone, 2009] Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(1):1633–1685.

[Taylor et al., 2007] Taylor, M. E., Stone, P., and Liu, Y. (2007). Transfer learning via inter-task mappings for temporal difference learning. *Journal of Machine Learning Research*, 8(1):2125–2167.

[Yosinski et al., 2014] Yosinski, J., Clune, J., Bengio, Y., and Lipson, H. (2014). How transferable are features in deep neural networks? *CoRR*, abs/1411.1792.