Research proposal

> Efficiently learning models (for planning) via decomposition and transfer

Introduction

Popular opinion within the machine learning community is that transfer reinforcement learning (TRL) is the most likely path to achieving general artificial intelligence [@LeggAGI] [@Higgins2017DARLAIZ]. Is this really true? And if it is, how can we do it? Let's explore the claim and some approaches.

What is reinforcement learning (RL)?

RL is often associated with the carrot and the stick. Where, good behaviour is rewarded with carrots as food, while bad behaviour is punished with sticks as food. There are two main features of RL that make it hard: "trial-and-error search and delayed rewards" [@Sutton1998ReinforcementLA]. Unlike supervised learning, which gives the learner feedback (*I think that digit is a 5 -> no, it's a 6*), in RL the learner only receives evaluations (*I think that digit is a 5 -> wrong*). Ontop of terse teachers, many actions may be taken before any evaluation is received, thus requiring credit to be assigned to actions, often leaving the learner wondering: "what did I do to deserve this?" (see pigeon superstition for an amusing example of credit assignment gone wrong). An easy way to understand the formal RL setting is as a Markov decision process (MDP).

A MDP is defined as a tuple, $\{\mathcal{S}, \mathcal{A}, P(s_{t+1} \mid s_t, a_t), R(s_t, a_t, s_{t+1})\}$. Where $s \in \mathcal{S}$ is the set of possible states (*for example arrangements of chess pieces*), $a \in \mathcal{A}$ is the set of actions (*the different possible moves, left, right, diagonal, weird L-shaped thing, …*), $P(s_{t+1} \mid s_t, a_t)$ is the transition function which describes how the environment acts in response to the past (s_t) and to your actions (a_t) (*in this case, your opponent's moves, taking one of your pieces, and the results of your actions*), and finally, $r(s_t, a_t, s_{t+1})$ is the reward function, (*whether you won (+1) or lost (-1) the game*) and $R = \sum_{t=0}^{T} r(s_t, a_t, s_{t+1})$ is the return. The player's goal, is to learn a policy π, (which chooses actions, $a_t = \pi(s_t)$) that yields the largest return (max R).

This setting is easily generalised to other, more interesting, cases. For example, if we weaken the requirement on that the learners observation x_t is fully describes the state s_t, then we could apply this framework to partially-observable decision processes, such as StarCraft II or self-driving cars, where we see only a small portion of the truly complex environment.

What is transfer learning (TL)?

TL is closely related to the notion of generalisation, when we have some existing knowledge that we repurpose (or generalise) for another use.

> A hallmark of human cognition is learning from just a few examples. For instance, a person only needs to see one Segway to acquire the concept and be able to discriminate future Segways from other vehicles like scooters and unicycles. Similarly, children can acquire a new word from one encounter (Carey & Bartlett, 1978). How is one shot learning possible? [@Lake2011OneSL]

A familiar example to some is, learning to eat with chopsticks (after already knowing how to use cutlery). We do not need to relearn to use our arms, we remember where our mouths are, and how to chew (we transfer that knowledge). We also remember that the goal of eating is to put food, not too much, into our mouths (we transfer that knowledge). We also transfer the knowledge of "how one might pick something up". But in this case, we need to learn some new grips and fine motor skills that will allow us to pick up the tempting food in front of us.

An easy setting to understand TL is in the multi-task setting (but there are others, few-shot learning, continual learning and even distribution shift could be viewed as transfer learning between the past and present)

Imagine we have two classification tasks, A, B, each of which consist of pairs of observations and targets. Task $= \{(x_i, t_i) : i \in [1 : N]\}$. Now, a learner, f, trained on task A and achieves a loss, $\mathcal{L}$, when evaluated on task A. We denote this as: $\mathcal{L}^A(f_A)$ (subscript denotes training task, superscript denotes evaluation task).

We say transfer has occurred when $L^A(f_A) > L^A(f_{B\to A})$ (the goal is to minimse loss). That is to say, that training on B, and then training on A imporves performance on A (aka pre-training [@Erhan2010WhyDU]). Similarly, we could transfer in the opposite direction, $L^A(f_A) > L^A(f_{A\to B})$. Training on A and then training on B improves performance on the original task, A (note this is actually quite hard, for example see EWC in [@KirkpatrickPRVD16]).

Why do we care?

Whether we imagine a production robot that must package an oddly shaped present, a self-driving car that must react to a clown suit running across the road, an algorithm managing an electrical grid when a storm hits, … transfer allows these agents to generalise from past experience and reliably achieve our goals. The robot "reuses" wrapping strategies from many differently shaped objects to construct a new policy. The self-driving car "guesses" that the clown suit might have a person in it since it has "similar" proportions.

Without the transfer of knowledge between domains, the robots, algorithms and automaton we design would each have to start from nothing, no prior knowledge, or from human provided domain knowledge. Learning with no priors is expensive (it requires a lot of experience and supervision) and writing down priors that work is expensive (humans are not cheap, and often we are not capable of writing down the "right" policy).

Thus, to *efficiently* scale artificial intelligence to the real world and all of its complexity, it will be necessary for knowledge to be transferred between domains.

Transfer as decomposition

One way to achieve transfer learning is to decompose complex observations into a set of modules/"atomic"-factors. Or in other words, to disentangle independent factors.

For example;

- we could take a symmetry group, and rewrite it as the composition of a single atom with various transformations. Thus we have decomposed the group into is parts.
- Aka finding the basis.
- Decomposition of objects and relations?
- (WILL) more examples

The belief that complexity can be decomposed, that observations are built from smaller/few parts, is at the heart of unsupervised learning, and of science. For example, it is common to assume that there are a set of latent variables that combine (non)linearly to generate observations. And within science, reductionism and mediation analysis are key tools for producing understanding. They allow us to break down complexity into simple parts we can study and understand.

Although, it should be noted that, even if we can decompose a complex phenomena, there is a lot of meta information required to actually use this decomposition for planning. Which modules do what? How are they related? When should I apply a given module? How do they combine?

Note: There are other approaches to transfer learning, for example; learning a metric, distillation, the freezing of parameters. But maybe underneath their superficial differences they are all doing the same thing, decomposing the problem into distinct parts?

How to decompose?

As far as I know, there are two ways to build priors into a learner; explicitly or implicitly. Explicit priors normally mean structural constraints and regularisers. Implicit priors can mean the (unintended) priors lurking within an optimisation algorithm (for example SGDs bias towards flat minima [@Poggio2017TheoryOD]).

We could build a decomposition into the structure of our model, for example, by specifying that different loss functions apply to subsets of parameters (like how the generator and discriminator in a GAN learn from different losses). Another example could be an ensemble: we feed the participants in the ensemble certain (different) features, forcing them to do different things. Often, these structural choices come from domain knowledge, beliefs we have, as people, about how the task should be done.

Alternatively, we can regularise our model to decompose its inputs. For this we need a differentiable measure of disentanglement and a flexible representation. This approach is appealing as it should require less domain knowledge.

Decompositions in RL

(Let's explore some popular decompositions in RL.)

As note above, a fundamental decomposition used in RL is the model-based RL framework. Where a reinforcement learner is decomposed into a transition function (a model of the environment) and a policy (how to act in the environment). This facilitates transfer as, the learner can be given a new task, requiring a new policy, (while remaining in the same environment) and simply reuse its model. For example, asking a learned-chess-master to now lose a game of chess on purpose (notably model-free learners like DQN [@Mnih2015HumanlevelCT] cannot do this).

There are other examples of existing work attempting to use a decomposition to facilitate transfer, for example;

- Feudal networks [@Vezhnevets2017FeUdalNF] decompose the learner's policy into a manager and a worker who work a different temporal scales. This could allow management strageties be to shared between different worker's domains.
- Learning to learn by gradient descent by gradient descent [@Andrychowicz2016LearningTL] decomposes learning into, learning the teacher (or optimiser) and the student.
- Modular meta-learning [@Alet2018ModularM] constructs a single function from many modules and must learn the modules and when to use them.

It is possible to structure decompositions in many ways; hierarchically, as an ensemble, as some sort of graph structure, … As of yet, there is no general formula that tells us when a decomposition of a learning problem is going to be optimal or even "good".

Proposed research

Ideally, this masters would provide an understanding of generalisation and transfer. I think that an understanding of, or theory of, generalisation and transfer would answer questions such as;

- what, when and how can knowledge be transferred from A to B?
- is decomposition necessary for transfer? Is it sufficient?
- how (computationally) hard is it to find symmetries between two domains?
- what problems can a "decomposer" solve that another cannot?
- which representation of knowledge is optimal for my setting?
- how can we learn "algebras" for composing modular systems?

But I am unsure how to approach these questions and which tools are the best for thinking about them. While mulling these over, I would like to explore decompositions and model-based learning. My hope is to see patterns and find interesting problems for understanding transfer.

Benchmarks

One of the reasons, in my opinion, that the machine learning field has progressed quickly is its use of shared benchmarks on currently out-of-reach problems. ImageNet [@imagenet_cvpr09] is a great example of this for the computer vision community, and the Atari ALE [@machado17arcade] is currently the canonical benchmark for the RL community.

By definition, TRL requires the testing of learners on many different tasks. But this can take thousands of CPU/GPU hours to evaluate… I think there is a need for cleverly designed benchmarks that don't require thousands of dollars. Currently TRL is a game for players with resources, which means that only Google and Google-Deepmind get to play.

I am imagining a set of as-simple-as-possible settings where we can explore the limits of transfer and decomposition (similar to ai-safety-worlds for the AI safety community).

Setting

Where TRL can be applied is dependent on where RL can be applied. Thus if we want to scale TRL to the "real world", we first need to extend RL to the real world. Some of the key differences between the places RL has been successful so far (for example; Go, Atari ALE [@machado17arcade]) and the real world are;

- continuous action spaces (especially important for robotics),
- resource constraints (memory - online learning, speed - real-time machine learning, …),
- partial information (where model-based RL becomes necessary),
- long-term dependencies

Finally, the real world is a lot more complex than these games we test our algorithms on. If we want to scale to the real world, we need to make our algorithms more data efficient. Transfer learning is a solution.

Specific questions to explore

I hope that the motivations above; transfer learning, model-based RL, testing on benchmarks, and extending RL to various settings, have all made sense. The directions and questions set out below are my attempt to combine these motivations into some actionable research directions and questions.

(these may be ill-posed, trivial, or solved, but hopefully I will find out soon…)

Decompositions

- Can we formalise what we mean by "decompose"? Can we differentiate it? What is its relationship to independence criterion and ICA?
- Meta-RL [@Wang2017LearningTR] trains a learner on the aggregated return over many episodes (a larger time scale). If we construct a temporal decomposition (moving averages at different time-scales) of rewards and approximate them with a set of value functions, does this produce a hierarchy of meta learners?
- Imagine you are given two models f, g that might predict pedestrian and traffic behaviour respectively. How can you safely/sensibly combine their predictions? If the models were provided as densities we could evaluate the next step as $s_{t+1} = \text{argmax}_{s_{t+1}} \prod_i p_i(s_{t+1} \mid s_t)$.
- How can we train modular systems when there is often no gradient defined (as the module was not used)? Using "counterfactual" credit assignment of what might have happened?

Model-based learning (with partial information)

- Build a differentiable neural computer [@Graves2016HybridCU] with locally structured memory (start with 1d and then generalise to higher dimensions). Is the ability to localise oneself necessary to efficiently solve partial information decision problems? Under which conditions does the learned index to a locally structured memory approximate the position of the agent in its environment.
- When attempting to learn a model, the agent uses an exploration policy. This policy may influence the dynamics observed, thus we need to use off-policy methods to correct for the effects of exploration actions. (The model must somehow disentangle the agents policy, and its effects, from the dynamics of the system)
- Inverse energy learning. Inspired to inverse reinforcement learning, what if we assume that the observations we make are the results of some optimal action, in the case of an energy being minimised, $\Delta x = -\eta \frac{\partial E}{\partial x}$.
- While learning a model $s_{t+1} = \tau(s_t, a_t)$ is useful. It is more useful to know how to get around using that model. For example, I want to get to s^k, how can I do that considering I am in another

state, s^i? Want a function $f(s^i, s^k) \rightarrow \{a1, a_2, \dots, a_n\}$ that outputs a sequence of actions. You need to know how to get around… (HER?)

Planning with a learned model (with continuous actions)

(Unfortunately I am not going to make it to these problems in this masters. But I think it is important to remember the ultimate goals of TRL. We don't just care about learning models, additionally, they need to useable for efficient planning.)

- If a model is being learned online how can we efficiently update value estimates computed using the old model?
- How can you backpropagate gradients through the argmax functions required for planning?
- If I am using an imperfect learned model to generate plans, how can I ensure that I do not plan for "fantastic" outcomes (aka they are fantasies). (note, this is closely related to reward hacking)
- Construct a planner that can control the models computational complexity by asking it to provide more approximate solutions (via accuracy masks, for the size of time-steps, or …?).

Timeline

I have allocated time for 8 "sprints" (the bullet points above; decompositions and model-based learning), each of 2 weeks. The goal of each sprint will be to;

- motivate the idea as a solution to an existing problem,
- prove that the "existing" problem really exists,
- generate alternative solutions and a suitable baseline,
- design the minimal viable experiment to falsify the proposed solution,
- implement the experiment if feasible.

Future work will depend on the results of the sprints.

Proposed deliverables

- Tutorial(s) on "core" RL,
- A learned model that benefits from transfer between environments (on Atari ALE or similar),
- An essay on the future of model-based RL,
- The definition and construction of a new benchmark for TL,
- A thesis documenting my work.

Hopefully a few papers, but that is conditional on making discoveries.

References