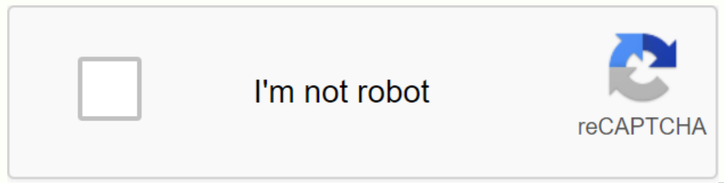

Continue

Meta reinforcement learning pdf

Download PDF Abstract: This paper releases the offline meta-enhancing leather (offline meta-RL) problem setting and introduces an algorithm that performs well in this environment. Offline meta-RL is analoged to the widely successful study strategy of pre-training a model on a large group of fixed, pre-collected data (possibly from various tasks) and refines the model to a new task with relatively little data. That is, in the right meta-RL, our meta-train fixed, advanced data from various tasks and adjusts to a new task with a very small amount (less than 5 trajectory) of data from the new task. Nature offline, algorithms for the right meta-RL can use the largest possible pool of training data available and eliminate potentially unsafe or expensive data collection during meta-training. This setting inherits the challenges of the right RL, but it differs significantly because offline RL does not generally consider a) transfer to new tasks or b) limited data from the test task, both of which we face in the right meta-RL. Target the offline meta-RL environment, we introduce Meta-Actor Critic with benefit weight (MACAW). MACAW is an optimization-based meta-learning algorithm that uses simple, surveillance regression goals for both the inner and outer loop of meta-training. On offline variants of common meta-RL benchmarks, we find empirically that enables this approach to fully offline meta-reinforcement learning and achieve significant gains over previous methods. From: Eric Mitchell [see email] [v1] Thu, 13 August 2020 17:57:14 UTC (8,960 KB)[v2] Tuesday, 6 Oct 2020 17:35:24 UTC (9,924 KB) we should stop trying to find simple ways to think about the content of the mind, such as simple ways to think about space, objects, various agents, or symmetries. All of these are part of the arbitrary, intrinsic complex, outside world. They are not what needs to be built, since their complexity is endless; Instead, we need to build in only the meta-methods that can find and capture this arbitrary complexity. - Richard Sutton, The Bitter Lesson (March 13, 2019)The general trend in machine learning research is to stop fine-tuning models, and instead use a meta-leather algorithm that automatically finds the best architecture and hyperparameters. What about meta-reinforcement learning (meta-RL)?meta-reinforcement learning is just meta-learning applied to reinforcement learning However, in this blogpost I will call meta-RL the special category of meta-learning that uses recurring models, applied to RL, as described in (Wang et al., 2016 arXiv) and (Wang et al, 2018 Nature Neuros). Let's get started. In reinforcement learning, an agent output actions at every step, such as moving left, moving ahead, etc. At each step it receives observations (such as the frames of a video game) and rewards (e.g. r=+1 if it is a correct r=0 different). The best way to understand meta-RL is to see how it works in practice. If you want to skip the examples, here's a short version: Train: Throw a lot of problems with the same core structure to your model. Test: Your meta-RL agent is able to quickly identify the key parameters of any new problem you throw at him, ultimately achieving optimal performance on this new problem.tl;dr: learn your meta-RL agent adding and negative numbers, then it will learn deduction in a breeze because he understands the abstraction for manipulative numbers. A simple example Immagine getting into an empty room with only two levers (facing what is called a multi-armed bandit problem). What are you doing? Subtractor the left lever (action A1), receive a reward r = 1 with probability p and no reward (r = 0) different? Pull the right lever (action A2), receive a reward r = 1 with probability 1-p and no reward (r = 0) differently? Don't think too hard about this one: to answer correctly, you need to know the value of p. For example, p may be equal to 92% (from the chance of reward after doing A1), or p = 0.92 for short. And that's exactly why meta-RL is interesting. If you throw enough two-levers problem (with different values of p) to your meta-RL model, it will become increasingly better to use the couples (action, reward) it has received from interacting with the environment. For example, if after repeating A1 for 100 trials you will get 20 rewards, but you get 80 rewards by doing the same with A2, then it is likely that the likelihood of reward from doing A1 (p) is inferior to the likelihood of reward of doing A2 (1-p), because A1 has led to rewards 20 out of 100 times, while A2 has led to rewards 80 out of 100 times. In practice, the meta-RL agent trained on a bunch of two leverage problems is able to identify which of the two levers leads to the highest reward using only the data points of a very small number of interactions. See below a comparison of a meta-RL agent before training (left) on a case where p=0.92 and a trained meta-RL agent (right) have been tested on a case where p=0.76.before training, p=0.92 (left) vs. after training, p=0.76 (right); SourceThe trained meta-RL agent (right) is able to quickly identify the high-probability-of-reward leverage, accumulation a lot of reward, while the unconsolated meta-RL agent acts randomly, receives little reward. Key takeaways: if we throw a bunch of two levers problem cases with different values of p at our meta-RL model, it will be able to quickly estimate the p of a case, thus acting optimally on new cases. Now that we have an intuitive understanding of meta-RL, we will go through two simulations described in Wang et al, 2018 Nature what I reimplemented:The Two-Step Task Resetting: In RL, an agent interacts with its environment according to the diagram below. The agent environment agency environment in RL (source)For each time t, an agent must perform an action a_t using as input:the state of the environment s_t.the reward r_t (generated by the environment after the action a_-t-1}). We'll call this loop (happens at every time t) a trial. To get a first grip of how the two-step task environment works, here is a poison of a meta-RL agent communicated with the environment for 100 trials (which we will later call an episode). More precisely: the agent takes the different (perceptually distinguishable) states: S1, S2 and S3 (upper nodes of the venum above). For each trial, the meta-RL agent between actions a1 and a2 in an initial state selectS1 (the green arrow under S1 represents the chosen action). If it helps, you can think of a1 and a2 as the two levers of the simple example above. When the agent does a certain action, says a1, it will likely (80% of the time) end up in a certain condition (eg. S2 if action a1 is chosen). At the end of a trial, it gets either a positive reward (r = 1) or it doesn't get reward (r = 0). Depending on the state, the probability of getting a positive reward may be high (p = 0.9) or low (p = 0.1). However, the probability of getting a reward is not fundamentally attached to a state. Indeed, every trial, there is a one in a forty chance (p_switch =0.025) that the often rewarded state and rarely-rewarded state will switch state (e.g. if the often-rewarded state was S2 before the switch, the often rewarded state will be S3 to the switch). This switch actually matters in this setup because it changes slightly the chance to receive a reward for a given action. Indeed, for each switch we exchange only the probabilitys of receiving a reward for S2 and S3. Therefore, the agent switches between two tasks that have the same core structure (the one where S2 is the often rewarded state, and the one where S3 is the often rewarded state). For example, in the poison below, S3 as the often rewarded state for trials 7-9. Then, at trial 10, the probability of reward switch, so S3 is rarely rewarded until a last switch at trial 26.Trials 7-29; probabilities call to 10&26 IMPORTANT NOTE: the agent who chooses between action at each trial between action a1 and a2 does not know what the often rewarded state is. The only way to access this information is the result of his actions at each hearing. For example, if after consistently choosing a1 (resulting in the end of S2 with probability 80%) For 10 trials in row, the agent receives 10 positive rewards in a row (r = 1 at the end of each trial), then S2 is very likely to be the often rewarded state. For this task, the main variable (i.e. the analog for p of the simple example) comes from the uncertainty associated with the of the following question: what is the often rewarded state? In short, if an agent is able to correctly

determine which state is the often rewarded state (for example (for example, Estimate that the often rewarded state is S2), then the best policy is trivial: just choose the action that often goes to this often-rewarded state (if the often rewarded state is S2, then do a1). Model-based vs. model-freeIntuitively, an agent for the two-step task should be able to:understand that there may be a likelihood that the often rewarded state can be either S2 or S3, and that it may at any moment.update its beliefs at each trial regarding which state has the highest likelihood (between S2 and S3), shows adjustments to put it in a nutshell , to solve the two-step task, it will need a model of the environment. In RL, algorithms that do not build a model of the environment are called model-free, and are often referred to other algorithms that do require a model of the environment, or model-based [in practice, the distinction between model-free and model-based is very subtle, especially when the system has memory]. For example, your brain uses model-free RL when you respond to (predictive) value of stimulus (e.g. Pavlov's dogs) and model-based RL when playing video games (because it must communicate with a model of the video) Game in your head before any automotive decisions).source This distinction between model-free and model-based is central to meta-RL: meta-RL uses a model-free algorithm and displays model-based behavior (!) The accommodation probably We've previously spoken about the important distinction between model-free and model-based. To understand the results of the two-step task, it is essential to measure or at least improve the differences between these two. But first, let's refresh our reminders about what the two-step task setting looks like by looking at the diagram below: Two-step task. Adapted from Wang et al, 2018 Nature Neuroscience, Fig. 5aThe key metrics for the two-step task is the stay probability, i.e. the likelihood of repeating the same action after receiving some reward. A useful way to look at this stay probability is by observing the difference between the likelihood when the last transition was common (e.g. arrival in S2 to a1, normal) or unusual (e.g. our chosen a1 and, surprise, we ended up in S3). To better understand this accommodation, let's try to interpret the diagrams below: canonical model-free (left) vs. Canonical Model-Based (right)(From Figure 5 in Wang et al, 2018 Nature Neuroscience)'an intuitive way to look at those diagrams it to think about what we do as people: The standard behavior (when we get tired, in automatic mode) is to repeat the same action as it leads to some positive reward (e.g. if I ate sugar and my brain was like 'it's like 'it's Pretty good', I'll to eat sugar). This corresponds to the diagram on the left (see above), and is related to what we mentioned previously called model-free (i.e. just repeat the without building any model of how the world works). If your brain is able to distinguish between common and unusual transitions (for example by repeating reward actions as the reward arises from a common transition), then it implements some model-based algorithm. For example, if a good night of rest often (80% of the time) brings you joy, you will often repeat it (stay probability > 50%). If you watch Youtube videos, you bring happiness just 20% of the time, the happiness comes from an unusual transition so you'll avoid it. Now let's see how we can apply this knowledge to reproduce the two-step task results of Wang et al, 2018 Nature Neuroscience.Lessons From Reimplement the Two-Step TaskGoalTeach to reproduce a meta-RL agent to solve the two-step task. Code We will use this code. Check out the README.md for specific instructions. The main notebook is the biorxiv-final.ipab jupyter notebook. Please select Tensorflow 1.12 and CPU. Architecture Before I unveil the full meta-RL API, a quick reminder about what the agent environment interface looks like in RL:At the beginning of the two-step task section, I already have a very similar agent environment interface. The only difference is in wording: from here we will talk about the observation (o_t) of the state, instead of the state (s_t) itself. And now, the showstopper: Meta-RL architecture. Adapted from Wang et al, 2018 Nature Neuroscience, Fig. 1aIf you look at it for the first time, it seems hard to understand. Let's expand it with the key below:Inputso_t: observation for the current timestep.a_{t-1}: action from the previous timestep.r_-t-1}: reward from the previous time. Outputsa_t: action for the current timestep.v_t: estimate of the accumulated reward given that the agent is in this state at timetep t.LSTMIt is the type of our neural network. In TensorFlow it has already been implemented (and super optimized), so we'll use it as a black box without thinking too hard about it. However, if you want to learn how it works under I strongly recommend you to read this article. A2CShort name for Advantage Actor Critic, who will use us to train the meta-RL model using previous observations, actions and rewards. A useful way to think about A2C is as part of a broader class of algorithms in RL, called Actor Critical Algorithms, where a neural network outputs both an actor and a critic:The actor outputs the action for the current timestep (a_t). The critic critic criticises what the actor does, by carrying out the value of the states in which he ends up after his actions, an estimated value of the current state (v_t). ExampleLet's assumption that, last test, the agent chose the action a_{t-1} =a1, ended up in S2 and received a reward of r_{t-1} = +1 (S2 was the often rewarded Therefore, we will move the network to a_{t-1}, a_{t-1}, and o_t=S2, because the network needs to understand that he received this particular reward (r_{t-1}) because of his actions (a_{t-1}) and the state in which it ended up to a_{t-1} (S2). Using those inputs and the hidden statements/weights of the LSTM, the actor (in A2C) estimates that the best action (for the current trial) is to do a_t=a1 and the critic estimates the situation with a value (for example 3.4). Mapping statements and actions to integers (in our code, o_t=S2=1 and a_{t-1}=a1=0), we will get the following chart: concrete example for the two-step taskResultsStay Probability See below the Email evolution of the stay probability during 20k episodes of training (code): The stay probability starts at 0.5 for all transitions (see the poison when the pigs are all flat). At the end of his training, the agent displays a model-based behavior (cf. the canonical model-based conspiracy on the right). For reference: canonical model-free (left) vs. canonical model-based (right) (From Figure 5 in Wang et al, 2018 Nature Neuroscience)Short, our meta-RL agent has learned the strategy leading to the highest expected reward. Reward Curve The most important thing to watch when introducing RL training is your reward curve. With TensorFlow you can plot it with Tensorboard. FloydHub automatically discovers your Tensorboard data, so to plot your reward curve, you just need to click on the Tensorboard link (inside the blue dashboard bar at the bottom of the screen)!click the Tensorboard link (inside the blue dashboard bar at the bottom of the screen). We have 8 training (one for each seed (= random number generator)) to test for robustness.the reward the agent gets the task to do, in function of the seed (one color = one seed) and the number of training episodes (the horizontal axe represents the number of episodes); CodeLessonsUnlike traditional machine leather algorithms, with meta-reinforcing Leather the algorithm is stuck on 0% performance for the first 8-12k episodes. Therefore, you can spend countless hours (> 4 hours for the two-step task on the CP) stuck at 0% performance, without knowing whether the model learns or not. My advice: a) ask yourself if my experiment went wrong, what was my mistake? b) try to report as exactly as possible the details of your experiments (the date, the time, the purpose, the reason it might not work, the estimated training time, etc.) Meta-RL seems to be similar to how kids learn things: we don't know how to do something at all, or we know exactly how to do it. In the above chart we take the same dynamic through an S-shaped/sigmoid curve: the meta-RL model understands the task at some point between 8-12k episodes, and from that point on it keeps progress on the task. Let's try to apply meta-RL to a classic psychology experiment, the At each step (or trial), an agent must choose between two randomly placed in left-right positions, but only one is rewarded. Every six tests (which we will call an episode), we replace the two objects with two new objects. We measure how many rewards the agent receives per episode. This experiment was originally tried on monkeys:In the beginning monkeys act indifferent to rewarded and unrewarded objects. After experimenting with several hundred few objects (i.e. after several hundred episodes), monkeys finally understood the underlying structure of the task. Henceforth they are optimally on the task, always selecting the reward object after the second trial (for the first trial they are bound to guess what the reward object is, because they still have no reward).Source:Supplied So the 'rule' to learn is that position is meaningless, it's just the identity of the object associated with reward. - Jane Wang, co-author of Prefrontal Cortex as a Meta-Enhancement Learning System, private communication. For this task, we will use DeepMind's compatible 3D platform for agent-based AI research: DeepMind Lab. In particular, we will use the Harlow Task environment, which is part of a larger open source set of environments to repeat psychology experiments: Psychlab.Down, which sees the agent in DeepMind's environment for the Harlow Task (source). The agent is shaking because for each trial it should: target the red fixation crosstarget the middle of an image. Lessons From Reimplementing the Harlow TaskGoalReproduce the results of harlow task simulations using DeepMind Lab.CodeWe's will be used to this code. Check out the README.md for specific instructions. The code specifically for meta-RL can be found here (this is a sub-module of the main reward). Please select Tensorflow 1.12 and CPU. Input The beginning of each episode, we sample two images (with replacement) of a set of 42 images of students from 42 (to customize the data set to your own needs, see here). Michael Trazzi (=me) (left) and my partner KevinCosta (right)We changed the images to 256x256, and feed them to the Harlow task environment, using an 84x84 window size for the agent's input. Every trial, the agent successfully chooses my face (reward image)In addition we process the observations to make the input as small as possible, using rowing and gray scale. See below the preproulted inputs: the same 6 trials, but now the inputs are pre-processed, we calculate the average value of the pixels of the left image (idem with the right image). If it is the first time this average has been encountered, we select a new unattainted id (0<= id < DATASET_SIZE = 42) for this average. one average matching one image, image, get a unique id for each image (we handle the fact that since the agent envelopes the pixels change by only considering the first frame of a trial for our id). Summary of the various processing strip articles We used the same architecture as in the two-step task, the only difference bees in the observation o_t. Let's make it more clear with an example. Let's suppose that: The current observation is o_t=[id_kevin, id_michael], for example o_t=[12, 35] (in practice we have fed one-hot representations of that id, but let's hold [12, 35] for simplicity). Last trial, the meta-RL model entered the action a_{t-1}=RIGHT (coded by the intellectured 1). In other words, the meta-RL chose the right image. [Note: the agent does only one action per trial, the reverse action to come back to the middle of the screen is automatically in our wrapped environment.] Last trial, the image on the right was the rewarded one. From there, last trial, the agent received the r_ {t-1}=+5 after his action OK. with those inputs, and the current statements and weights of the LSTM, Estimate our model that the best action (for this observation) a_t=1 (ie. choosing action RIGHT) and that the expected value of the current condition is v_t =10.concrete example for the Harlow tasksHere is the reward curve for 5 different seeds (one color per seed) to ~10k episodes (which took approximately 3 days to train) on FloydHub (one CPU per seed) : reward curve forms the Tensorboard, with smooth = 0.9; For reference, random baseline is 0 and optimal performance is +25(the first test is needed random, and the 5 last trials can generate on most 5*5 = 25 rewards)As with the two-step task, all seeds spend the first ~ 1k episodes at 0% performance. The green seed remained at 0% performance. The orange, red and blue seeds achieved peak performance of 6, 8 and 10 rewards per episode respectively, but the training was really unstable. However, the pink seed generated increasing reward, until it finally reached a threshold (around 40% performance for the pink seed) after ~7k episodes. It seems the performance may still increase for some of the seeds. We didn't try to run it longer because the as symptoms at r=10 were challenging. LesseMulti wire. Before we launched our training, we thought that, becauseTensorflow was built for multi-threading (and our code supported multi-wire), everything would go smoothly. It seems that DeepMind Lab's Python API uses CPython, so we don't use multi-thread using (normal) tensorflow, due to the global interpretlot lock. You can learn more about multi-threading, here. Multi-processing. After learning about this global interpreter, I implemented multiprosproutation by Python's library use. However, it seems that Tensorflow does not allow to use multiprocessing after imported tensor flow, tensor flow, that multi-processing branch came to a dead end. We also tried multiprocessing with raying another multiprocessing library. It didn't work out, however, because DeepMind was not pickable, it could not be saved with pickle. You can learn more about multi-processing, here. Adjusting hyperparameters. After giving up on multi-wire/processing, we decided to see if our model would learn something if we drastically reduced the action space and fed only very simple inputs (cf. the Input section above). So, we ended up running experiments with just one thread, a 48 units LSTM and a very simple input. Finally we have achieved something that has reached an average reward of ~10 (approximately half the maximum performance, cf. Results). Our best guess is that 48 units weren't enough to learn the task, but maybe it was the mono wire or something else in our code. In short, always change one hyperparameter at a time, otherwise you don't know why you model isn't learning at all. For a detailed version of the lingering projects, make the tensor flow code or just more reward curves sure you'll check meta_rl the country! To wrap it up, our meta-RL model achieved optimal performance on the two-step task and 40% performance on a simplified version of the Harlow task. In the following section, we will see how our brains can implement meta-enhancement learning. Machine learning in the BrainMulfle models of how the brain reinforcement learning develops. For example: see this paper for an exact version of integrating deep learning and neuroscience.see this paper providing a normative framework within which decision-making can be analysed. To get a sense of brain architecture, let's look at the diagram below.sourceThe relevant parts for meta-RL are: The Crucial Role of DopamineThe brain maximizes some reward function, thus implementing something close to reinforcement learning. However, let's not forget that these rewards should be implemented on biological hardware. In practice, dopamine neurons carry reward forecast errors by special highways called dopaminergic pathways. The four largest dopamine pathways, adjusted here, consist of the differences between received and predicted rewards. They are crucial to basic forms of learning about rewards and making us strive for more rewards – an evolutionary beneficial at alliance. Most dopamine neurons in the middle of humans, monkeys, and rodents signal a reward prediction error; they are activated by more reward than predicted (positive forecast error), staying at baseline activity for fully predicted rewards, and showing depressed activity with less reward than predicted (negative forecast error). - Wolfram Schultz, Dopamine prediction error coding prediction errors inspired Learn algorithms called Temporary Difference methods, one of them is A2C! Model-free vs. Model-based Dopaminergic processes have traditionally only slow, model-free learning, while the prefrontal cortech is often associated with model-based learning. However, the prefrontal cortex uses a history of all previous rewards (stored as long-term memory) to postpone decisions, much like an LSTM. So, if we replace meta-RL's LSTM with the prefontal network (PFN) - matching prefrontal cortex + basal ganglia + thalamus - and A2C with dopamine (DA on the diagram below), we can model the interactions PFN-DA with the same diagram we used for the meta-RL API Agent: architecture. The prefrontal network (PFN), including sectors of the basal ganglia and the thalamus that connect directly to [the prefrontal cortex], is modeled as a recurring neural network, with synoptic weights adapted by an RL algorithm driven by [dopamine];o are perceptual input, an action, r is reward, v is state value, t is time-step and δ is [reward forecast error]. The central box indicates a single, fully connected set of LSTM units. - Wang et al, 2018 Nature Neuroscience, Fig. 1aTherefore, the model-based RL in the PFN can only be the observed behavior of a model-free temporary difference method that uses reward forecast errors (e.g. A2C), performed by dopamine. In fact, this is exactly the model-based behavior we observed when we trained our meta-RL model on the two-step task using A2C. In other words, high-level decision making can only happen an emerging phenomenon due to some model-free dopamineergic sub-processes. Thanks to Yasmine Hamdani (resp. Kevin Costa) to help me debug the code for the two-step task (resp. Harlow task), Jane Wang for all the additional details about the Harlow task and the valuable neuroscience feedback, Arthur Juliani for his code and blog post on meta-RL, Emil Wallner for the numerous Charlie Harrington for his off-to-earth Jean-Stanislas Denain for making sure everything was scientifically sound, Maxime Choulika, Tristan Deborde and Igor Garbuz for making sure I was extra clear and all the FloydHub team (Sai, Naren and Alessio) for their support and to let me run if you want to name an example from the post, please name the resource that example from. If you want to name the post as a whole, you can use the following BibTeX: @misc-metarlblogpost, title={Meta-Reinforcement Learning}, author={Trazzi, Michaël}, howpublished={\url{ }, year={2019} }