25/12/2024

Ben-Gurion University of the Negev
Faculty of Engineering Sciences
Department of Software and Information systems

Deep Reinforcement Learning

Assignment 3

Meta and Transfer Learning

General Instructions

1. The assignment should be submitted in pairs at the Moodle course site.
2. The submission will include a zip file containing:
   - A report in PDF format with answers to the questions appearing in the assignment, and the requested outputs of the codes. Explanations and analysis are expected to be **detailed.**
   - Short instruction for running the scripts.
   - The scripts of your solutions.
3. The scripts should be written in Python. Use the TensorFlow library for Neural Networks. Use TensorBoard for the visualization and graphs.
4. The report can be written either in English or Hebrew.
5. Write your names and ID's in the report.
6. Due date: 15/01/2025

Introduction

Deep reinforcement learning algorithms usually require a large number of trials. So far with the tools we have learned in this course, learning a new task entails re-collecting this large dataset and training from scratch. Intuitively, knowledge gained in learning one task should help to learn new, related tasks more quickly. Humans and animals are able to learn new tasks in just a few trials. In this assignment, we design a reinforcement learning algorithm that leverages prior experience to figure out how to solve new tasks quickly. In recent literature, such methods are referred to as meta-reinforcement learning Mishra et al. [2018], Finn et al. [2017], Wang et al. [2016], Duan et al. [2016].

Section 1 – Training individual networks (25%)

In this section you will implement the actor-critic you implemented in HW2 to two additional small control problems: Acrobot-v1 and MountainCarContinuous-v0. Your goal is to achieve the respective goals of the two problems: reaching the mountain top and bringing the acrobot to a pre-specified height.

To make the transfer learning task easier, the size of the input and output for all tasks needs to be identical (for problems with smaller inputs, use 0 to pad the input. For problems with smaller output, have "empty" actions that are never used).

Please note:

1) This means you will need to retrain the architecture for the cartpole problem.
2) You can use a different architecture for each problem, but each architecture has to include at least one hidden layer (i.e. a network has to have at least an input layer, a hidden layer and an output layer).

- Provide statistics for the running time and number of training iterations required for each task to converge

Section 2 – Fine-tune an existing model (25%)

In this section you will fine-tune a model trained on one problem and attempt to apply it to another. Apply the following for two pairs of tasks (they are refered as source and target, respectively): acrobot -> cartpole, cartploe -> mountainCar:

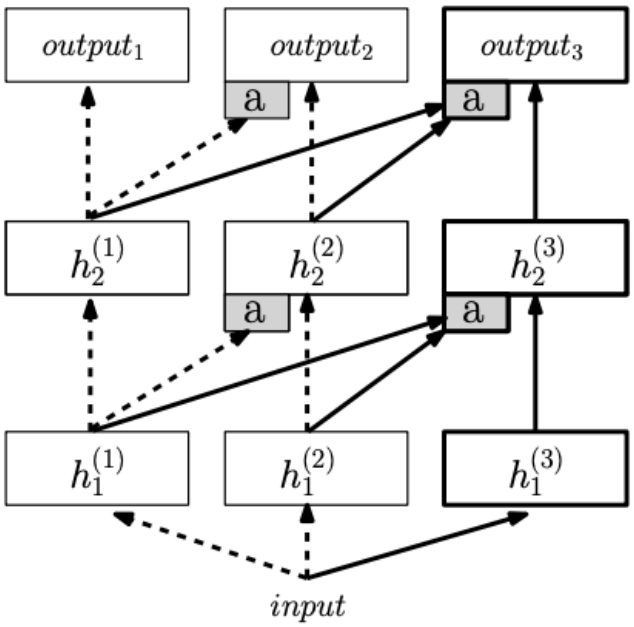
- Take the model that was fully trained on the source and re-initilaize the weights of the output layer

- Train the new network on the target.
- Provide statistics on running time and the number of training iterations required. Compare the results to those is Section 1. Did the fine-tuning lead to faster convergence?

Section 3 – Transfer learning (50%)

In this section you will implement a simplified version of the Progressive Networks that were presented in class. Do the following for the setting {acrobot, mountainCar}->cartpole and {cartpole, acrobot}-> mountainCar (again, sources and target respectively):

- Use the *fully-trained source networks* created in Section 1 and connect them to the *un-trained target network*. Remember that the source netowrks remain frozen throughout the process (Implementing the adapters – the boxes marked with *a* in the diagram – is optional).
- In case you are using a single hidden layer in all architectures, connect the hidden layers of the sources to the output of the target network. In case of multiple hidden layers, connect the top hidden layer *l* in the source to the target output *m*, then connect *l-1* layer to *m-1* until you run out of hidden layers in one of the architectures.
- Train the architecture until convergence. Did the transfer learning improve the training? Provide statistics on the running time and training iterations required.



Important note: transfer learning is tricky, and based on previous years not all will succeed in getting the models to show an improvement (although it worked really well for some). It's OK not to succeed, but it is important that you document your efforts and explain how you went about in getting the architectures to work.