What is Transfer Learning?

• Typical machine learning assumes training and test are matched:
  • Supervised learning: Training and test sets are drawn from same distribution
  • Reinforcement learning: Assumes we learn and test on the same MDP

• Transfer learning:
  • Large family of possible departures from this assumption
  • Need to be careful to make sure the evaluation metric is well defined
Why study transfer learning?

• More realistic – training and testing rarely perfectly matched
  • Learn on simulator
  • Test on real robot

• Avoid learning from scratch every time we see a similar problem
  • Ikea furniture assembly problem
  • Should get easier after you’ve assembled a few pieces
  • [https://jobmob.co.il/blog/funny-ikea-job-interview-cartoon/](https://jobmob.co.il/blog/funny-ikea-job-interview-cartoon/)

Categorizing Transfer Learning Methods

• What are the goals? (How do we measure success?)

• What assumptions are made about similarity between tasks?

• How is the transfer opportunity recognized & implemented?

• What is transferred?
Steps possibly involved in transfer

• Given target task select source task from which to transfer

• Learn relationship between source and target

• Transfer knowledge from source to target

Evaluation: Cost of learning the source

• How do we count cost of learning the source task?

• Could view as “sunk cost”, which means we don’t count it (target task time scenario)

• Could view as part of total cost of learning (total time scenario) – perhaps amortized over many targets
Evaluation: What is measured

- **Jumpstart**: How good is initial performance on a task given the benefit of transfer learning?
- **Asymptotic**: Is the final performance improved from transfer?
- **Total reward**: Is the total reward accumulated by the agent during learning higher (long term benefit of jump start)
- **Transfer ratio**: Ratio of total reward w/transfer to total reward w/o (advantage of being scale invariant)
- **Time to threshold**: Time needed to hit a given performance level

Some theoretic results on sample complexity. See, e.g., Brunskill & Li

What is measure (in a figure)

![Diagram showing performance over training time with labels for Jumpstart, Asymptotic Performance, Time to Threshold, and Total reward. Total reward can be viewed as area under curve.](Figure from Taylor & Stone 09)
Limitations of metrics

- **Jumpstart**: Captures initial boost, but ignores everything thereafter
- **Asymptotic**: How do you know where/when to measure?
- **Total reward**: Same as asymptotic
- **Transfer ratio**: Scale invariant, but not offset invariant
- **Time to threshold**: How to set threshold?

General evaluation difficulties

- Metrics themselves don’t transfer across domains, so impossible to compare different algorithms on different domains (general issue for RL?)
- Lack of standardized transfer benchmarks
Negative transfer

- Transfer learning isn’t guaranteed to help

- Negative transfer occurs when performance is worse than learning target task from scratch

- Example: Ron learning to ski (negative transfer from bicycle riding)

Task Differences

- Different dynamics
- Different reward
- Different actions

- How much are these allowed to differ?
- Are differences drawn from some distribution?
Source task selection

• Human guided

• Automated from library of potential source tasks (possibility of negative transfer)

Task mapping

• How is knowledge from the source task mapped to the target task?

• Human provided

• Learned
  • Ex nihilo (hard)
  • Partially learned (parameter tuning)
What is transferred & how it is used

• Any combination of:
  • Q/Value function
  • Features
  • (Partial) Model (P,R)
  • Shaping reward
  • Policy – as policy or as prior distribution used for exploration
  • samples
  • Other things built on top of the above (e.g., task hierarch)

• Interaction with learning algorithm
  • Policy search (policy gradient) would naturally transfer a policy
  • Methods that use models would transfer models
  • etc.

Example Mountain Car (possible task diffs)

• t: changing the motor or friction
• s: changing the range of the state variables
• si: changing start location
• sf: changing goal position
• v: hiding the position and showing only the velocity
• r: changing the reward function
• a: disabling the do-nothing action
• p: changing state representation
• # changing the number of cars(!)

See tables 1 & 2 from paper
Clustering transfer methods

• Fixed state variables and actions

• Multi-task learning

• Transferring task-invariant knowledge

• Learning task mappings

Fixed state variables and actions

• Simplest case: Concept drift
  • Assume world is changing smoothly/slowly
  • Never threshold the learning rate
  • Never really converges but does the right thing if you are learning faster than the world is changing

• Changing reward function (if changes are cued)
  • Good case for learning transition model
  • Use learned transition model to (re)solve problem when reward changes
  • Could also replay cached experiences if you have a reward oracle
Fixed state variable and actions

• Hierarchy
  • Temporally extended actions (SMDPs, HAMs, MAXQ, etc.) can still be used
  • Might still be useful to try them

• Transferred features
  • Features for one task may still be useful in new tasks
  • Caution about this: Usefulness of features can depend heavily on structure of state space and reward function

Fixed state variables and actions

• Direct transfer of Q/value function
  • These still make sense when applied to the new problem
  • If value function is similar can be good initialization
  • Can also use as a shaping reward

• (In)direct transfer of policy
  • Initialize new problem with policy from old one
  • Use old policy to bias exploration/reduce action choices
Fixed state variables and actions

• Sample transfer
  • Could do “experience replay” with samples from source problem applied to new problem
  • How to decide which samples to use?
    • Reuse transitions if reward changes in known way?
    • Reuse transitions in parts of state space that are “similar”?

• Region based sample transfer
  • Identify regions of the state space that are known to be the same a priori
  • Transfer samples from source to target
  • Example: Move goal in robot navigation/control – samples in non-goal states can be reused

Multi-task learning

• Similar to previous case: State variables and actions are same

• Problem is explicitly multi-task
  • Assumes that task boundaries are clear
  • Assumes some library of solved tasks

• Some tasks could be completely disjoint (need to figure out which are relevant)
• Could assume a distribution over similar tasks and hope to learn a single policy that works for all tasks
• Everything in between
Multi-task learning

• Example: Agent is told new reward function

• What to do?
  • Consult library of solved problems with different reward functions
  • Retrieve solution with most similar reward function
  • Use this to initialize new learner in some way (policy, value function, etc.)

Multi-task learning

• Example: Is this MDP in the (convex hull of the) set of previous MDPs I've seen?

• What to do:
  • Collect data on effects of (temporally extended) actions
  • Compare to known MDPs & solutions in library
  • Retrieve solution as a (combination of) known MDP solutions, or
  • Approach as a totally new problem

• This can potentially include abstraction: MDPs may differ at low level, but could be very similar at a coarse level – use coarse level similarities to initialize solution to target MDP
Transferring task-invariant knowledge

• States and variables may change!
• How do you this?
• Learner reasons about abstractions that are invariant to changes in the underlying problem
• Example: Learning skills
  • Robot can learn to open doors
  • Navigate through door
  • Push buttons
  • These tasks may be embedded in a family of larger tasks with other state variables

Transferring task-invariant knowledge

• Example: Parameterized value functions
• Suppose you have a family of tasks that involve arranging variable numbers of objects, e.g., winning at an RTS (warcraft, etc.)
• Try to learn value functions (and possibly features) that automatically scale with changes in the problem:
  • Features that count the number of desirable objects
  • Features that count the number of threatening objects.
  • etc.
• Goal: Value function that scales automatically as the task changes
• Assumes that changes in the task are expressed in a way that is captured by the features
Mapping provided case

- State variables and actions are different
- Mapping is provided by human

- Not that different from previous cases, but a transformation is required to apply those techniques

Learning task mappings

- Most interesting but also hardest case
- Not known how (some of) S, A, T, or R map between source, target
- Easier special case: Figure out which from a predetermined set of possible mappings applies

- General case:
  - Hard because you are searching the space of functions between MDPs
  - Not obvious this is easier than doing RL on the target from scratch
Manifold transfer in RL

• Ammar et al. 2015 (subsequent to survey paper)
• Uses manifold alignment between source and target
  • Make transition graphs for both source and target
  • Apply manifold learning to both to find a common manifold for both
  • (soft) constraints and objectives:
    • Preserve local distances (standard constraints)
    • Reduce distance between explicitly paired states
• Applied to policy gradient RL – transferred policy and used to initialize PG learner
• Worked weirdly well (see paper)

Conclusions

• Transfer learning is a big and messy subfield
• Feels intuitively right, but
  • Lots of ways to do it
  • Lots of ways to evaluate it

• No dominant way to think about and evaluate transfer in RL - yet