Advanced Artificial Intelligence

Lab 12

Outline

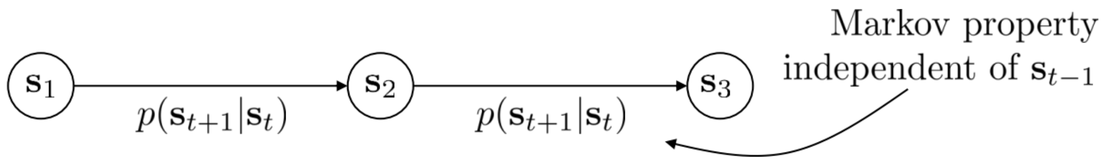
- Introduce a concrete RL problem which uses MDP as a basic model
- Exercise

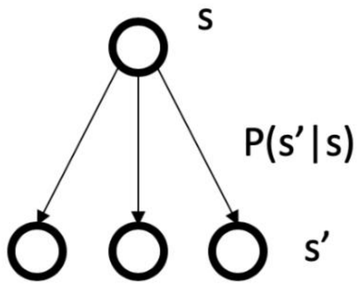
Markov Decision Process (MDP)

Markov Property: memoryless property of a stochastic process

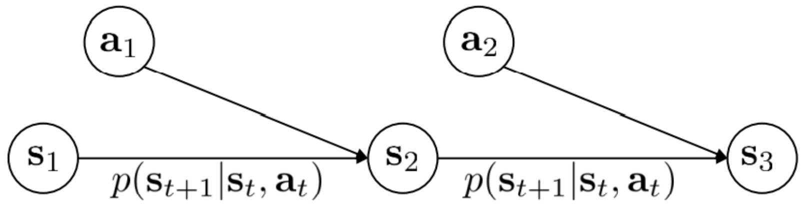
$$P(X_n = x_n \mid X_{n-1} = x_{n-1}, \dots, X_0 = x_0) = P(X_n = x_n \mid X_{n-1} = x_{n-1})$$

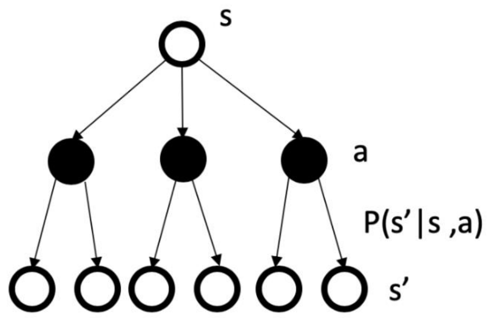
Markov Process/Markov chain: stochastic process with Markov property


$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t)$
Markov property independent of $\mathbf{s}_{t-1}$



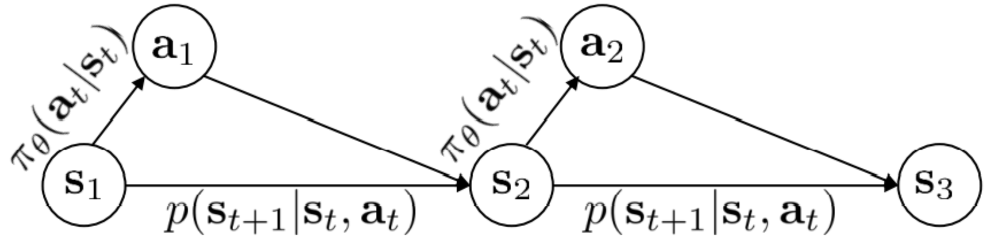
Markov Decision Process: Markov Process with action/decision


$\mathbf{a}_1$
$\mathbf{a}_2$
$\mathbf{s}_1$
$\mathbf{s}_2$
$\mathbf{s}_3$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$



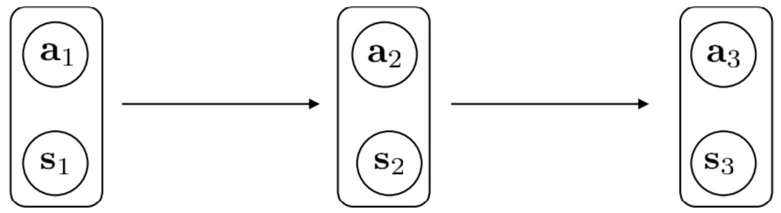
Reinforcement Learning(RL)

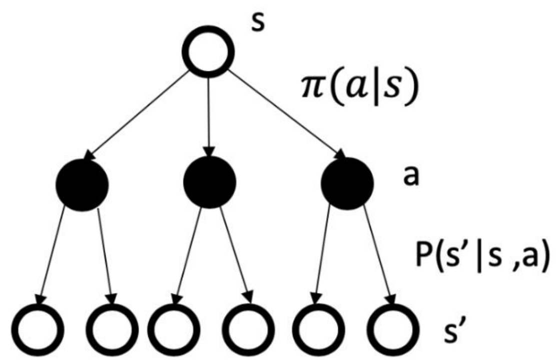
Find a map/distribution of action based on observation


$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{a}_1$
$\mathbf{s}_1$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
$\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
$\mathbf{a}_2$
$\mathbf{s}_2$
$p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$
$\mathbf{s}_3$

$$p((\mathbf{s}_{t+1},\mathbf{a}_{t+1})|(\mathbf{s}_t,\mathbf{a}_t)) = p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)\pi_\theta(\mathbf{a}_{t+1}|\mathbf{s}_{t+1})$$

Somehow still Markov chain on (s, a)

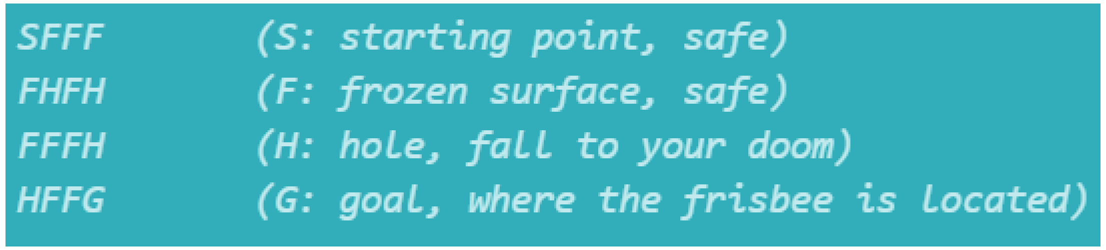




- MDP can formulate lots of environments if define well
- RL find solution based on MDP model
- RL should make sense in lots of well-defined environments

Example: FrozenLake-v0(Gym)

- 4*4 grid map(or 8*8) with frozen surface and hole, detailed map like below
- Hole causes death, frozen surface may cause slip
- Control agent to move from start place to goal place



```
SFFF       (S: starting point, safe)
FHFH       (F: frozen surface, safe)
FFFH       (H: hole, fall to your doom)
HFFG       (G: goal, where the frisbee is located)
```

This is a so simplified environment but somehow shows classic problem.

Formulating it as MDP model is a start to use RL to solve it, which may be a start to solve more complex problems.

But how to do?

Example: FrozenLake-v0(Gym)

Observation: Agent Position, a discrete scalar from 0 to 15 is enough (since map never change, adjacent situation is unnecessary)

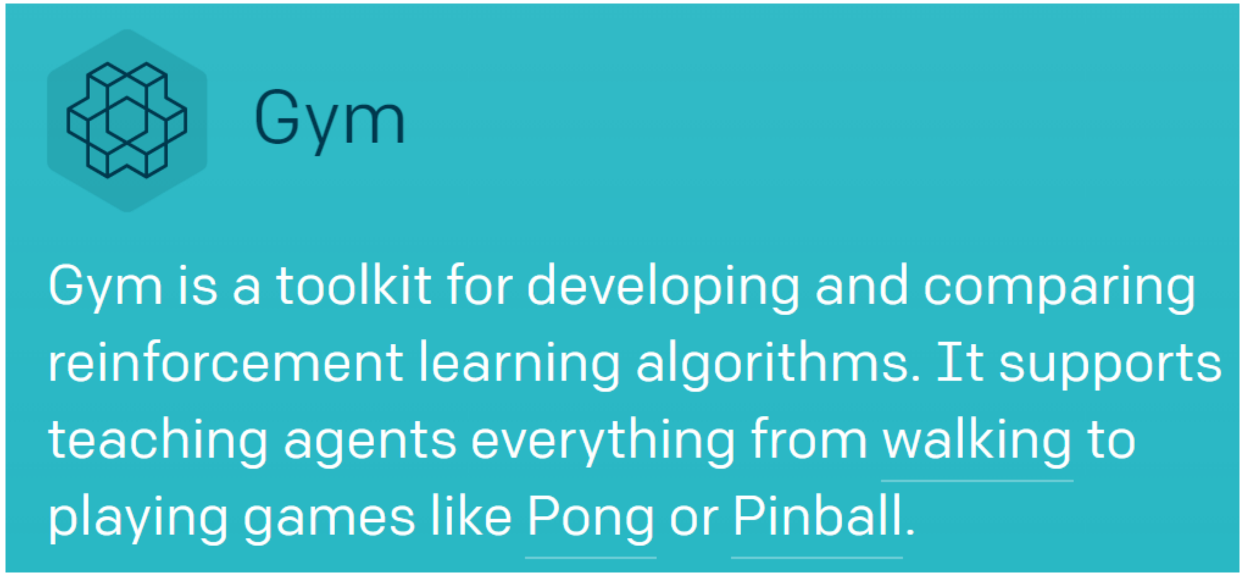
Action: Move Direction, a discrete scalar from 0 to 3 is enough

Reward: design according to aim, 1 if reach goal, else 0, is enough

Transition probability: 3-D array with shape 16*4*16, element is probability by which can imply slip

With all elements above, we come to a standard MDP shown in lecture.
Policy *Iteration*(PI) and Value *Iteration*(VI) should make sense.

Example: FrozenLake-v0(Gym)

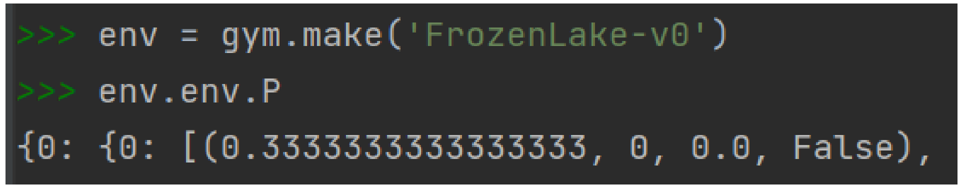

Gym

Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball.

Gym (openai.com) is useful to test RL algorithm, and its interface style is widely-used

FrozenLake-v0 is one of simple environments in Gym, and you can study its dynamics from source code on Github

For FrozenLake-v0 in Gym, transition probability can easily get by:



```
>>> env = gym.make('FrozenLake-v0')
>>> env.env.P
{0: {0: [(0.3333333333333333, 0, 0.0, False),
```

Try to use it to implement PI/VI to train a policy

Complex Example: SC2(StarCraft II)

StarCraft II(SC II) is widely considered as one of the most challenging Real Time Strategy (RTS) games.

- resource gathering and allocation
- building constructions
- training units
- researching technologies
- controlling army to fight with enemy

However, it still can be formulated in RL setting, then be solved by RL tech, and finally achieve grandmaster level AI.

Complex Example: SC2(StarCraft II)

A python RL environment for SC2 from DeepMind can be found here.
Lots of companies try to solve it, like AlphaStar from DeepMind and TStarBot-X from Tencent.
Both of them achieve excellent performance, but not completely solve the problem, for example, not human-level control and not train from scratch.

SC2 is a very complex environment, somehow much more intractable than go. (much larger observation space and action space, complex and long-term reward)
It is hard to train a well-performance agent by RL tech intuitively, current solution covers some advanced topics like multi-agent RL, partial observed RL, self-play, population-based training and so on.

Compared with AlphaStar, code for TStarBot is available on Github, and we may have a look at its formulation.

Complex Example: SC2(StarCraft II)

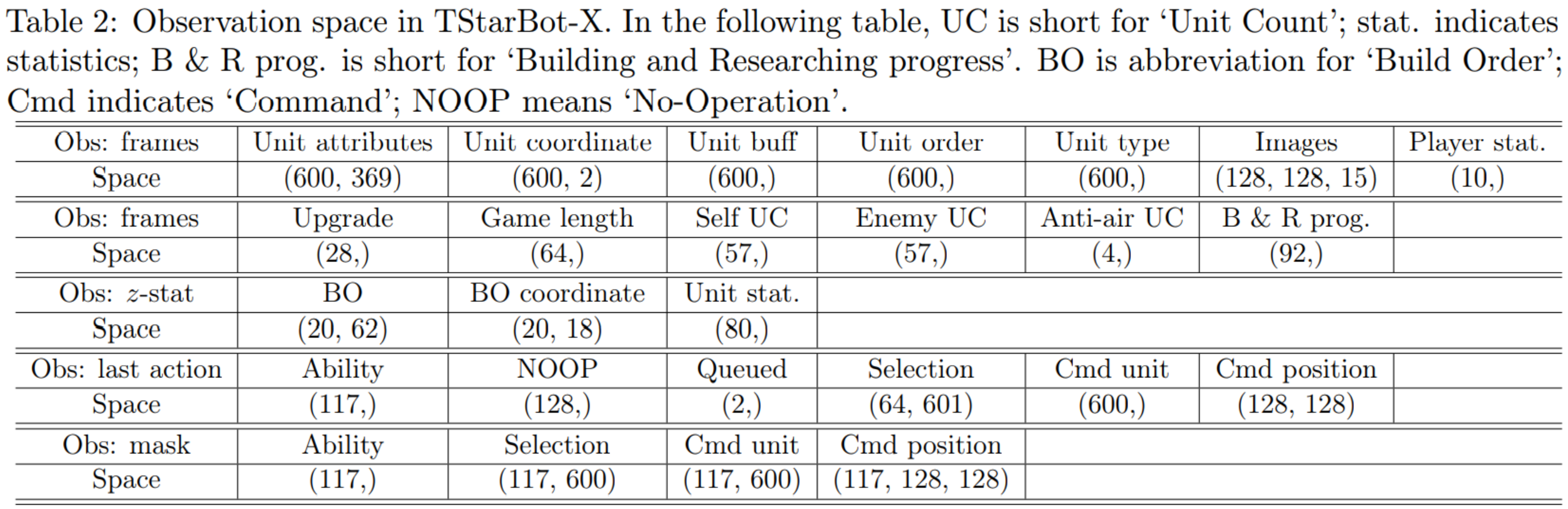


Table 2: Observation space in TStarBot-X. In the following table, UC is short for 'Unit Count'; stat. indicates statistics; B & R prog. is short for 'Building and Researching progress'. BO is abbreviation for 'Build Order'; Cmd indicates 'Command'; NOOP means 'No-Operation'.

Obs: frames	Unit attributes	Unit coordinate	Unit buff	Unit order	Unit type	Images	Player stat.
Space	(600, 369)	(600, 2)	(600,)	(600,)	(600,)	(128, 128, 15)	(10,)
Obs: frames	Upgrade	Game length	Self UC	Enemy UC	Anti-air UC	B & R prog.	
Space	(28,)	(64,)	(57,)	(57,)	(4,)	(92,)	
Obs: z-stat	BO	BO coordinate	Unit stat.				
Space	(20, 62)	(20, 18)	(80,)				
Obs: last action	Ability	NOOP	Queued	Selection	Cmd unit	Cmd position	
Space	(117,)	(128,)	(2,)	(64, 601)	(600,)	(128, 128)	
Obs: mask	Ability	Selection	Cmd unit	Cmd position			
Space	(117,)	(117, 600)	(117, 600)	(117, 128, 128)			

This is not human-level observation (raw image), but raw features which are comparable to humans (human can somehow achieve). (Human-level observation is much harder)

z-stat is a high-level feature introduced and highlighted in AlphaStar.

Complex Example: SC2(StarCraft II)

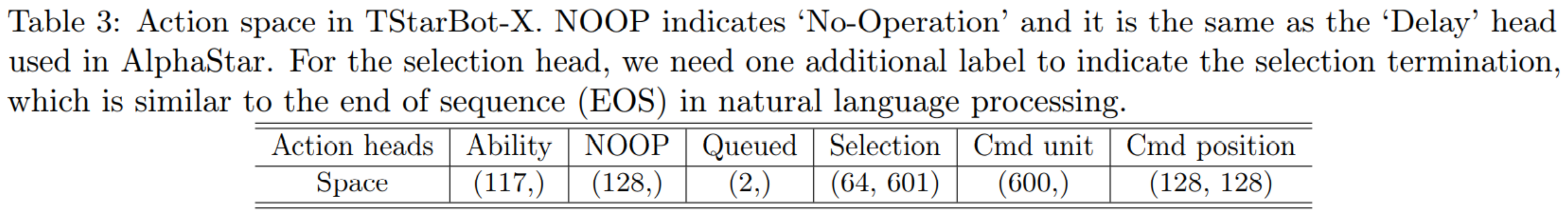


Table 3: Action space in TStarBot-X. NOOP indicates ‘No-Operation’ and it is the same as the ‘Delay’ head used in AlphaStar. For the selection head, we need one additional label to indicate the selection termination, which is similar to the end of sequence (EOS) in natural language processing.

Action heads	Ability	NOOP	Queued	Selection	Cmd unit	Cmd position
Space	(117,)	(128,)	(2,)	(64, 601)	(600,)	(128, 128)

Reward

- Win-loss outcome (main)
- The Edit distance between the build order in z-stat and the immediate build order,
- The Manhattan distances between the build units
- ……

For different intuitive goals, may need to design different rewards, and how to use these rewards also need thinking.

Complex Example: SC2(StarCraft II)

The net work of TStarBot-X

Action

Observation

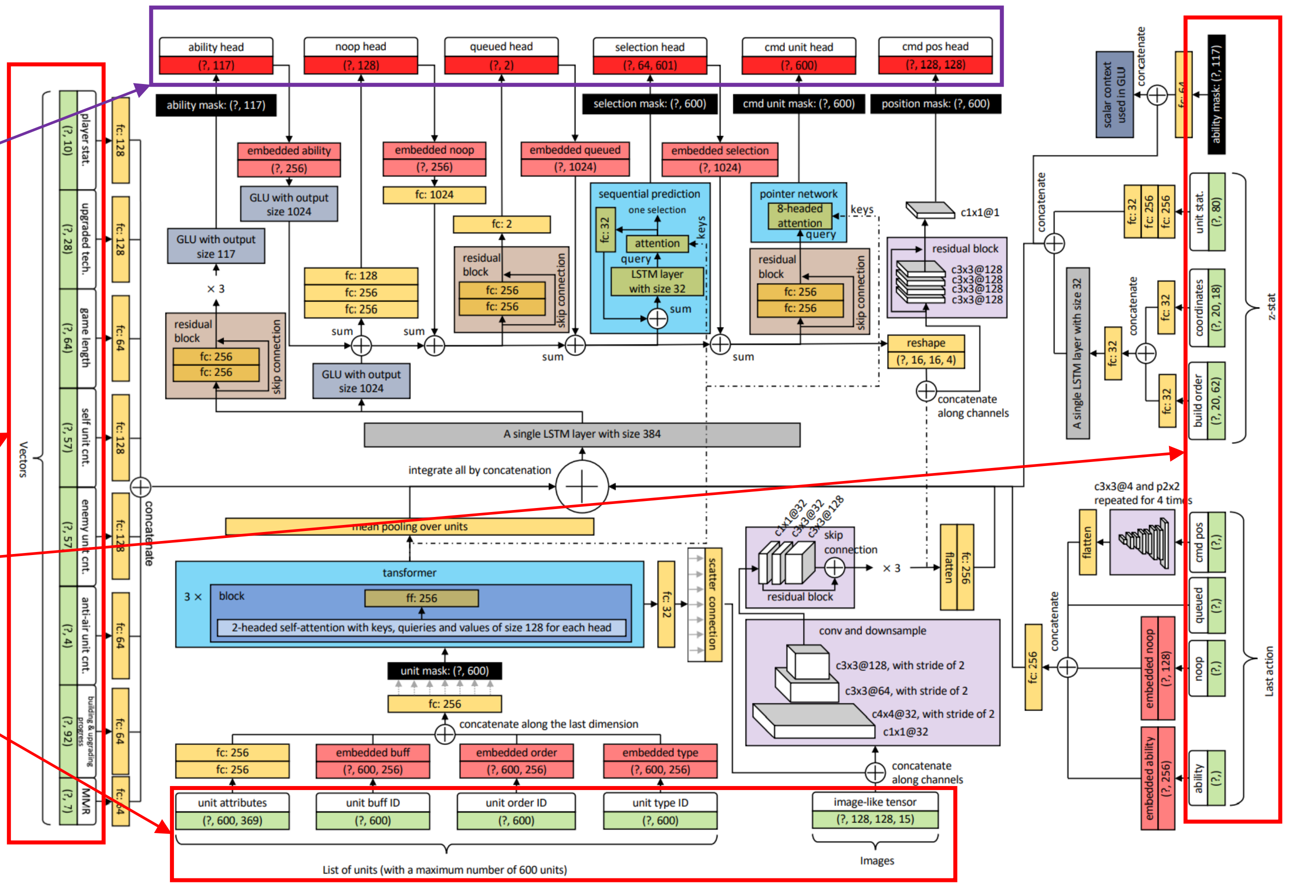


Figure 1: The detailed policy architecture. '?' indicates the batch dimension. All green rectangles are inputs. Dark red rectangles are output multi-head actions. Black rectangles denote masks relying on the game logic.

Another Example: Routing network

Original paper is here, and further study is here.

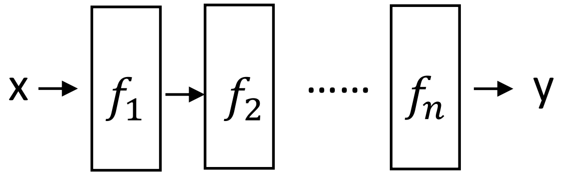
Routing network aims to use RL to partially solve network architecture search problem in multi-task learning.
Multi-task learning aims to train a model for multiple tasks by leverage data from all tasks.
Lots of MTL methods need share parameters of network among tasks, but which parameters should share among which tasks? (Neural Architecture Search)

Use RL to train a policy to determine network architecture by data instead of hand-crafted maybe a promising solution.

So how to formulate such a NAS problem into RL setting? what is observation, action and reward?
This may still be an open problem, but we can have a look at this paper's answer.

Another Example: Routing network

For a network,



So we can train a policy to determine f_1, f_2 …, f_n by some observation
Arbitrary function block maybe too complex, we can set candidate ones in advance.

$$y = f_n(\dots f_2(f_1(x)))$$

Routing network
- Router (determine block by data)
- Function blocks (candidate ones)

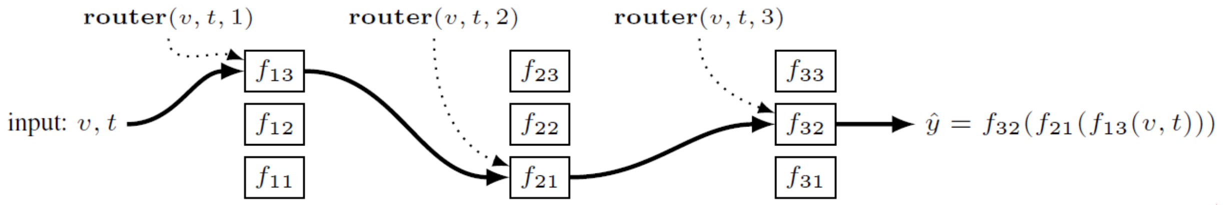

$\mathbf{router}(v, t, 1)$
$\mathbf{router}(v, t, 2)$
$\mathbf{router}(v, t, 3)$
input: v, t
f_{13}
f_{12}
f_{11}
f_{23}
f_{22}
f_{21}
f_{33}
f_{32}
f_{31}
$\hat{y} = f_{32}(f_{21}(f_{13}(v, t)))$

Figure 1: Routing (forward) Example

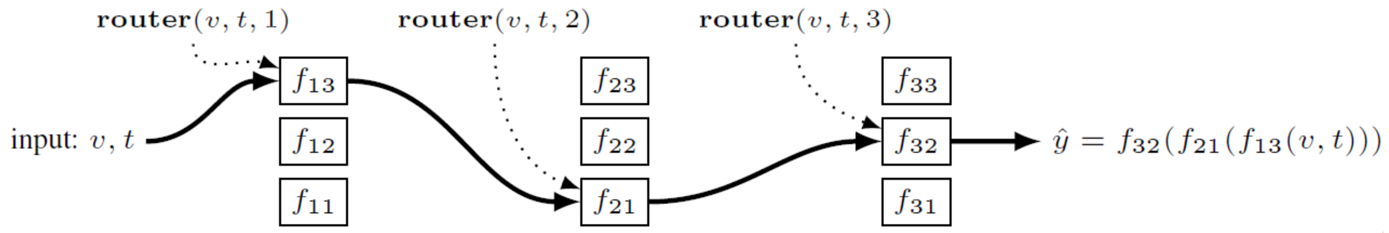
Another Example: Routing network

Observation

Input data (v) (for layer n, it is output of layer n-1)
Task label (t)
Layer index

Action

Function block index or no block



$\mathbf{router}(v, t, 1)$
$\mathbf{router}(v, t, 2)$
$\mathbf{router}(v, t, 3)$
input: v, t
f_{13}
f_{12}
f_{11}
f_{23}
f_{22}
f_{21}
f_{33}
f_{32}
f_{31}
$\hat{y} = f_{32}(f_{21}(f_{13}(v, t)))$

Figure 1: Routing (forward) Example

Reward

A final reward at the end of routing indicate model performance (For classification, 1 if predict correct, else -1)
Maybe some immediate reward at each route time, and for example, in this paper, authors design one to encourage fewer function blocks.

Exercise

- Play with Gym, have a watch to benchmarks on Gym, and some much more complex third-party environment

- Try PI/VI to train policy for FrozenLake-v0 in Gym (Available code can be found here)

- What if continuous observation? (it is easy to find such env in Gym)
  Can simply quantify the continuous space into discrete by equally divide into n parts make sense? (A relative code can be found here)