Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2024

Contents

CHAPTER 10

Markov Decision Processes 1 - Value Functions, Policies

We've developed a notion of a **state machine (SM)**: a system for keeping track of our *current* situation, using a "state".

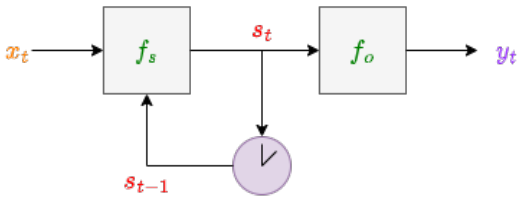
- This state also contains information about the **past**: our **present** state is influenced by past data.

In particular, our focus is on **finite state machines**.

Clarification 1

For the rest of this chapter, we'll assume that our state machines are finite:

- Our set of **states** and set of **inputs** are finite.



A reminder of what our state machine looks like.

10.0.1 A new perspective: the "outside world"

Now, we'll build on our state machine, to create something more specialized for what we need: this will require a new perspective.

- Our "state" has been referred to as our "current situation". This could suggest that it's **representing** something about the **world**.
- In this perspective, our state machine is representing how the world **changes** over time.
- In this case, the state of the world is what we're **interested** in: that's going to be our new **output**.

Concept 2

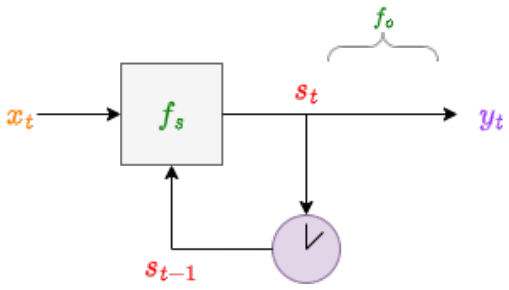
We can view the **state** of our state machine as the current state of the **outside world** that we're modeling.

If we're interested in this "state" of our world, that is the **output** we want:

$$y_t = s_t$$

- f_s is the identity function $f_s(z) = z$: our state is returned, the same way it entered.

Example: In a game, you might want to know **where** you are: so, we keep track of "position" as a state, and return it as an output (on screen).



We can basically remove f_o: it has no effect.

10.0.2 Making "decisions"

This is already interesting, but now, we'll build on this perspective:

- Often, we don't just want to simply **observe** the world, we want to **interact** with it.

- We might want to experiment with different ways to interact with, and change, our **model world**.
- Our state machine modifies its state ("world") through the **input**. We'll use this input to interact with our world: we'll call it an **action**.

Concept 3

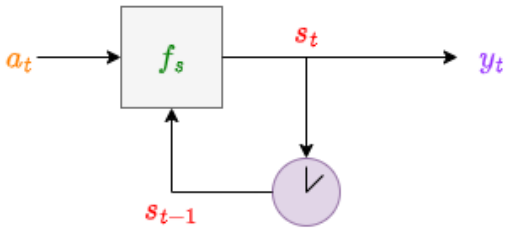
We want to be able to deliberately **modify** the state of the **outside world** though our interactions.

- With an SM, **modify** the state of the world through our **inputs**.

Thus, we'll replace our input x_t with an **action** a_t.

- Our set of actions $\mathcal{A}$ will replace $\mathcal{X}$.

- **Example:** If you were playing a game, your actions might be "move up", "move down", "move left", or "move right".
- These would affect your **position**, which we could encode in our state.



Structurally, nothing has really changed. x_t has become a_t.

10.0.3 Transitioning between States

One limitation of our state machines so far is that they're **deterministic**: the same inputs will always lead to the same outputs.

Definition 4

A **deterministic state machine** is one where the transitions between states are **deterministic**: given the same inputs, we always get the same output.

- In a realistic setting, the same actions won't always have the same effect: we might end up with **different** states, even if we take the same action.

Thus, we'll use **probabilistic** state transitions. Instead of outputting the same result every time, there will be a certain **probability** of a given outcome.

Example: You have a plant you want to keep healthy. It's currently dry, so you choose the action "**water**".

- 95% of the time, the plant becomes "healthy": it's been watered, and has what it needs to grow.
- 5% of the time, the plant becomes "sick": you just got unlucky, and the plant is sick now.

Maybe you watered more than it needed, or maybe something about the environment changed... it doesn't really matter.

This model doesn't require giving up on state machines: our function f_s has just changed, returning a **random variable**.

Concept 5

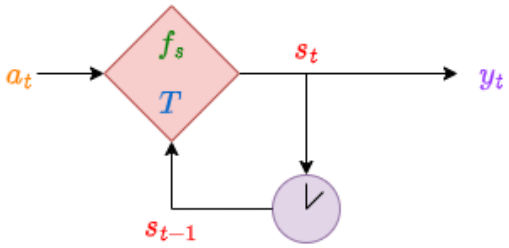
We want to be able to represent the **stochastic** (randomized) nature of our world.

So, we include some randomness in our **state transitions**:

- Given a particular state s_{t-1} and action a_t, **you don't know** exactly what state you'll get for s_t.
- Instead, we have a **distribution**, which assigns a **probability** to each possible next state.

We have a **probabilistic state machine**.

A probabilistic state machine is one type of "non-deterministic" state machine.



T represents the probability distribution of possible output states.

This kind of state machine is very similar to something called a **markov chain**.

Remark (Optional) 6

Our **probabilistic finite state machine (PFSM)** is roughly equivalent to an important mathematical model: a **markov chain**.

In order to be a markov chain, however, it must fulfill the **Markov Property**:

- The state transition is **memoryless**: it only depends on our most recent state s_{t-1}, not any earlier states.

This requirement is already met by our PFSM, but some more complex models may not.

This remark doesn't fully, rigorously define markov chains. However, we've already built a model that behaves very similarly.

The main difference from a markov chain is that, instead of having inputs, we have actions: a mechanism for making **decisions**.

10.0.4 Introducing Rewards

Fundamentally, all we've done so far is choose a particular type of state machine. But now, there's something we'd like to add:

- We have introduced the idea of an "action", but currently, we have reason to choose one action over another.
- To resolve this, we'll introduce an idea of which actions are "good": we'll give a **reward** based on your state, and action.

Concept 7

In addition to our markov chain/PFSM, we'll include a **reward function**, which tells us which situations are more or less desirable.

This depends on both your **action** and current **state**.

- These **state-action pairs** can be compared to each each other using the reward function.

Example: A "reward" in a game might be represented by a change in your score.

This reward creates two kinds of decision-making:

Concept 8

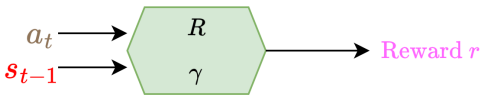
Our reward is determined by the **state-action pair** it receives. This creates two different aspects that weigh into our decision:

- **State**: how do we transition into the state(s) that give us the highest reward?
- **Action**: which actions give us the highest reward?

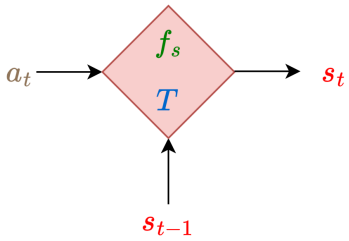
Often, our model must choose between an action that moves you into a "more rewarding" **state**, or an action that gives you higher **immediate reward**.

Example: Do you spend your time getting the easy reward in one area, or try to go to a different, possibly more profitable area?

Later, we'll introduce a **discount factor** γ, which influences this balance between immediate and long-term rewards.



R is a function computing rewards, γ is our discount factor we'll discuss later.



We can compare to the state machine, if we ignore the circular component: they take the same inputs, with different outputs.

Concept 9

Our **reward function** and **state machine** take the same inputs:

- The current state s_{t-1}
- The next action a_t

Based on this information, they tell us two different things:

- The **state machine** tells us how this action affects the world, in this situation.
- The **reward function** tells us how "good" this action is, in this situation.

The former tells us how the world has **changed**, while the latter tells us how **immediately** desirable this action was.

Together, they give us a more complete understanding of this state-action pair.

10.0.5 Markov Decisions Processes

Taking all of these modifications together, we create a new model: the **Markov Decision Process**.

Definition 10

A **Markov Decision Process (MDP)** is a model building upon **state machines**.

First, we make one labelling change:

- Our **inputs** $x_t \in \mathcal{X}$ are replaced with **actions** $a_t \in \mathcal{A}$.

Then, we select a particular variation of state machine:

- Our **state** is returned as the **output**: $f_o(z) = z$
- Our **transition** between states is now **stochastic**: we have a certain probability of ending up in each new state.

Finally, we add one new structure outside the state machine:

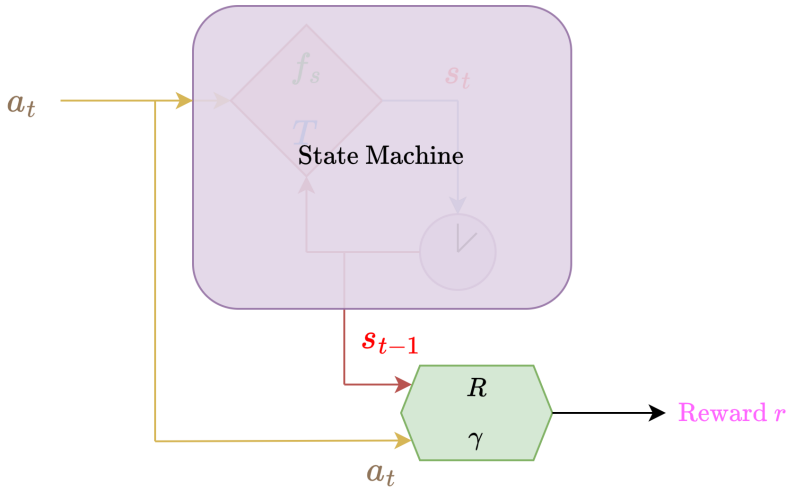
- We include a **reward** function to evaluate the quality of these decisions, based on the **state-action** pair.

Remark (Optional) 11

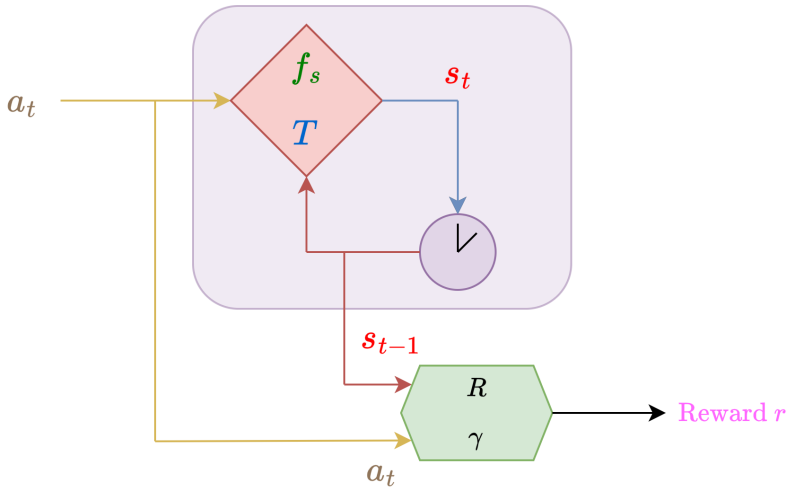
Alternatively, we can view an MDP as a **markov chain**, with two modifications:

- **Actions** replacing inputs, allowing for **decisions**
- A **reward function**, that allows us to evaluate those decisions.

Above, we depicted our state machine and our reward function separately. Here, we'll combine them:



This represents our complete MDP, albeit simplified. We see our real goal: to figure out the relationship between action a_t, and our reward r.



This view shows all the working parts: it's more complex, but more complete.

Our eventual goal is to find out how to **maximize** our reward: we find out which actions provide the most reward.

10.1 Definition and Value Functions

10.1.1 States and Actions in our MDP

We've laid out the general structure of our MDP, but now, we formalize each object.

First, the familiar parts:

Definition 12

For our **MDP**, we have a **finite** action space $\mathcal{A}$ and state space $\mathcal{S}$.

Thus, every action $a \in \mathcal{A}$, and every state $s \in \mathcal{S}$.

Remember: $a \in \mathcal{A}$ means "object a is in the set $\mathcal{A}$.

- Reminder that a "space" is just a set, with some extra structure.
- So, our action space is our set of actions, and our state space is our set of states.

The "structure" depends on what set we choose.

10.1.2 Transition Model

Now, we need to represent the **transition** between states.

- Each *possible* state has a probability p of being our *next* state.
- We'll compute the probability with our **transition model**.

Our transition model with give us a probability. But in order to know the probability, we need three piece of information:

- s: What is our current state? (Previously s_{t-1})
- a: What action did we take? (Previously a_t)
- s' What is the **possible next state** we want to get the **probability** of? (Possible s_t)

Our transition function T takes these three pieces of information, and gives us the probability:

$$T(s, a, s') = \text{Probability that, in state } s\text{, action } a \text{ results in new state } s' \tag{10.1}$$

In more mathematical terms:

Because our state S_t is now a random variable, we'll represent it with a capital letter.

$$T(s, a, s') = \mathbf{P}\left\{S_t = s' \;\middle|\; S_{t-1} = s, \quad A_t = a\right\} \tag{10.2}$$

Definition 13

The **transition function** T gives the probability of

- Entering state s',
- Given that we chose action a in state s

$$T(s, a, s') = \mathbf{P}\left\{S_t = s' \;\middle|\; S_{t-1} = s, \quad A_t = a\right\}$$

After a transition, we will be in **exactly one** new state s'.

We can represent it using function notation by considering the following:

- T has input (state, action, state): $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$
- T returns a probability: a real number between 0 and 1: $[0, 1]$

$$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$$

It would also be valid to write $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, because $\Big[0, 1\Big]$ is part of the real numbers $\mathbb{R}$.

Example: We'll return to the example of our plant.

- Its current state is $s = \text{Dry}$.
- We choose action $a = \text{Water}$.

We have two outcomes:

- 95% chance of becoming healthy.

$$T\Big(\text{Dry}, \text{Water}, \text{Healthy}\Big) = 0.95 \tag{10.3}$$

- 5% chance of becoming sick.

$$T\Big(\textbf{Dry}, \text{Water}, \text{Sick}\Big) = 0.05 \tag{10.4}$$

10.1.3 Comments on our Transition Function

Note that we said that we transition to **exactly one** new state. This means two things:

- Each new state s' is **disjoint**.
- We will definitely end up in **one** of those sets.

Combined, we can say that the probability of all of our states s' adds to 1.

Concept 14

Given a particular **state** s and **action** a, the probabilities for all new sets $\mathbf{s'}$ adds to 1:

$$\sum_{s' \in \mathcal{S}} T\Big(s, a, s'\Big) = 1$$

One more comment: we use our transition to determine the probability of state transitions.

However, T is **not** our state transition function f_s.

Clarification 15

While T and f_s are both involved in **state transitions**, they serve different functions:

- T gives the **probability** of entering a new state, based on our old states.
- f_s actually **gives us** the new state, according to those probabilities.

In other words, T is a function which **describes** how f_s behaves.

They even have different inputs/output sets:

$$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \left[0, 1\right] \qquad f_s : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$$
$$T\left(s, a, s'\right) = p \qquad f_s\left(s, a\right) = s' \tag{10.5}$$

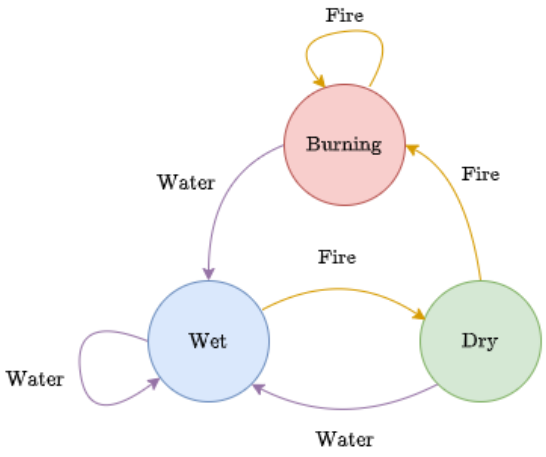
10.1.4 State-Transition Diagram: Review

Our state-transition diagram needs an upgrade, now that our transitions can be probabilistic.

First, we'll review our example from the **RNN chapter**.

Example: We have a blanket. It can be in three states: either wet, dry, or burning. We can represent each state as a "node".

- To change its state, we can either add "water" or "fire".



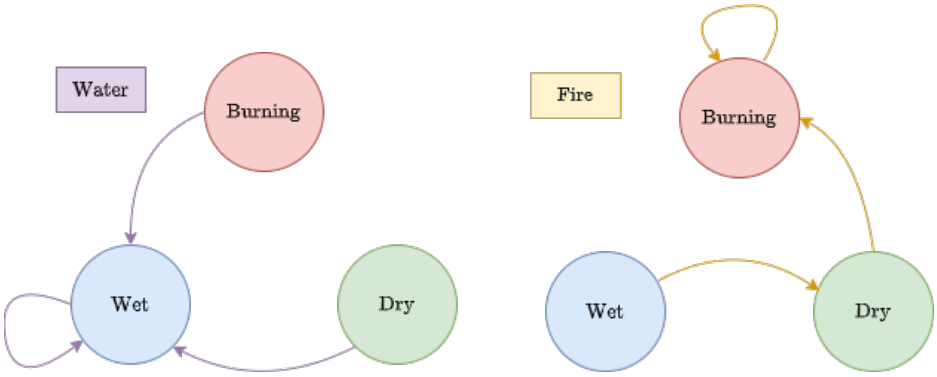
We want to update this to include transition probabilities.

- But this would get pretty dense and **complex**: each arrow would require a **state**, and a **probability**.

- Even worse: right now, we only have one possible outcome for each state-action pair.
  - But our probabilistic version allows for multiple outcomes: more arrows, more complexity.

There could be 2 or more outcomes in the same situation, based on probability.

So, we'll split up our diagram based on the **action**, like we did in the RNN chapter:



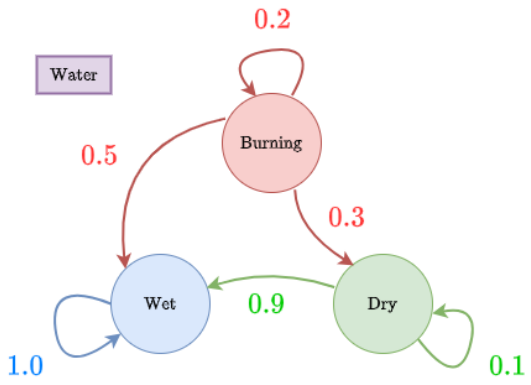
The left diagram uses **water** as an input, while the right diagram uses **fire** as an input.

10.1.5 State-Transition Diagram: Probabilistic

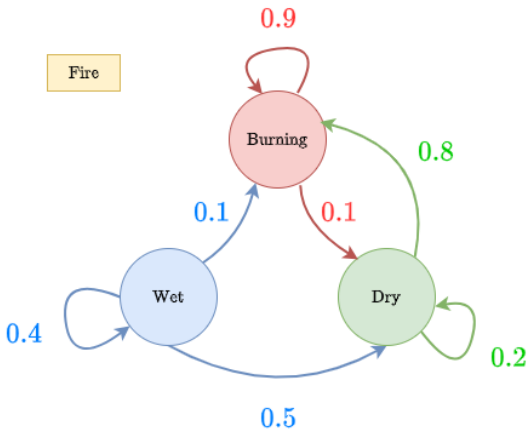
Now, we can extend these diagrams with probabilities.

We'll use the following:

- Add water
  - **Burning** blanket: 30% dry, 50% wet, 20% burn.
  - **Wet** blanket: 100% wet.
  - **Dry** blanket: 10% dry, 90% wet.



- Add fire
  - **Burning** blanket: 10% dry, 90% burn.
  - **Wet** blanket: 50% dry, 40% wet, 10% burn.
  - **Dry** blanket: 20% dry, 80% burn.



These would be almost impossible to represent in a readable way if we include every action on the same graph. So, we create a separate graph for each action.

- If we add more actions, our graph becomes no more complex: we just create more graphs.

Concept 16
For MDPs, we usually have a separate **state-transition diagram** for each **action**.

A second comment: Notice that, when adding water to a wet blanket, it has a 100% chance to stay wet.

- This example is **equivalent** to the deterministic state machine from the RNN chapter: based on our state and action, we know exactly what state we end up with next.

Concept 17
Our **MDP** can reproduce a **deterministic** state machine by setting the probability for every outcome to 0 or 1.

Of course, we still need 1 valid action for each state-action pair.

10.1.6 Transition Matrix

Representing our transitions is made complicated by the fact that we have three parameters: $T(s, a, s')$.

- If we wanted to represent the outputs, with each parameter on one axis, we'd need a 3-tensor to depict the whole thing.

But above, for graphing purposes, we found a solution: separating our transitions based on our **action** a.

- If we only consider one action a, we only have two parameters: s and s'.
- We can represent this with a **matrix** $\mathcal{T}$.

One axis will indicate the previous state s, and the other axis will represent the new state s'.

- We'll use rows for s (input state), and columns for s' (output state).

$$\mathcal{T}(a) = \begin{matrix} \\ \text{Input state } s \end{matrix} \overbrace{\begin{Bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{Bmatrix}}^{\text{Output state } s'} \tag{10.6}$$

- The element in our matrix will represent the **probability** of this transition.

$$\mathcal{T}(a)_{ij} = T\left(s_i, a, s_j\right) \tag{10.7}$$

Notation 18

One way to represent our **transition** T is to create a separate **matrix** $\mathcal{T}$ for each action a where

- Row i starts in state s_i
- Column j moves us to state s_j

In this cell, we have:

$$\mathcal{T}(a)_{ij} = T\left(s_i, a, s_j\right)$$

- The probability that, in state s_i, action a takes us to state s_j.

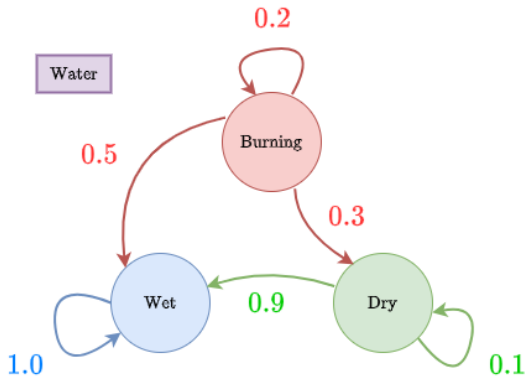
Example: Suppose we have 3 states: s_1, s_2, s_3. Our matrix for action a looks like:

$$\mathcal{T}(a) = \begin{bmatrix} T(s_1, a, s_1) & T(s_1, a, s_2) & T(s_1, a, s_3) \\ T(s_2, a, s_1) & T(s_2, a, s_2) & T(s_2, a, s_3) \\ T(s_3, a, s_1) & T(s_3, a, s_2) & T(s_3, a, s_3) \end{bmatrix} \tag{10.8}$$

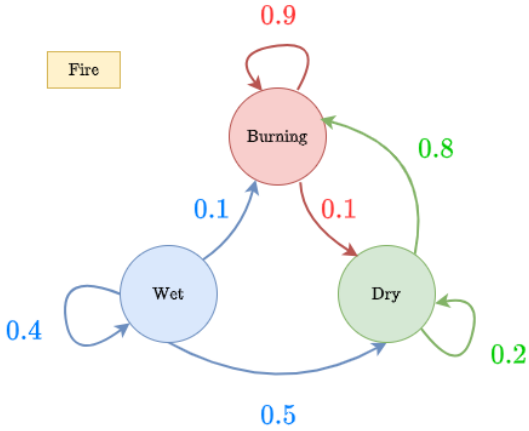
We'll practice on our usual blanket example. We label each of our states with an index:

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} \text{Burning} \\ \text{Dry} \\ \text{Wet} \end{bmatrix} \tag{10.9}$$

With this, we can create a matrix for each action.



$$\mathcal{T}(\mathit{Water}) = \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0 & 0.1 & 0.9 \\ 0 & 0 & 1.0 \end{bmatrix} \tag{10.10}$$



$$\mathcal{T}(\text{Fire}) = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0.8 & 0.2 & 0 \\ 0.1 & 0.5 & 0.4 \end{bmatrix} \tag{10.11}$$

10.1.7 Reward Function

We'll represent our **reward function**:

- We compute the **reward** of each state-action pair as with a number: $r \in \mathbb{R}$.

Definition 19

Our **reward function** R gives the **reward** of a particular **state-action** pair.

This indicates how desirable it is to

- Choose action a
- From state s

$$R\left(s, a\right) = r$$

We determine our function notation by analyzing the input/output pair.

- R has input (state, action): $\mathcal{S} \times \mathcal{A}$
- R returns a "reward" as a real number: $R\left(s, a\right) \in \mathbb{R}$.

$$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

Example: In our blanket example, we may only care about the state, not the action.

Maybe we're trying to use the blanket. A dry blanket can be used, a wet blanket cannot, and a burning blanket is an active problem.

$$R\left(s, a\right) = \begin{cases} 10 & s = \text{Dry} \\ 0 & s = \text{Wet} \\ -20 & s = \text{Burning} \end{cases} \tag{10.12}$$

Concept 20

Sometimes, our **reward function** may only depend on the **state** we are in.

For consistency, we still use the notation $R\left(s, a\right)$.

One thing to be careful of:

We'll procrastinate the discussion of our discount factor γ to our discussion of **infinite horizon**.

In the meantime, we'll include it in our formal definition, but we won't discuss it.

10.1.8 MDP Formalized

Finally, we've built all the pieces we need for a mathematical definition of our MDP.

Definition 21

We formally define **Markov Decision Process (MDP)** as a list of 5 objects: $\left(\mathcal{S}, \mathcal{A}, T, R, \gamma\right)$

- $\mathcal{S}$ is our **state space**, and $\mathcal{A}$ is our **action space**.

$$s \in \mathcal{S} \qquad a \in \mathcal{A}$$

- $T\left(s, a, s'\right)$ is our **transition function**, which gives us the **probability** of transitioning from state s to state s', if we take action a.

$$T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \left[0, 1\right]$$

$$T(s, a, s') \quad = \quad \mathbf{P}\Big\{S_t = s' \;\Big|\; S_{t-1} = s, \quad A_t = a\Big\}$$

- $R\left(s, a\right)$ is our **reward function**, which tells how **desirable** a particular state-action pair is.

$$R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$$

Lastly, we have our discount factor:

Some definitions treat the discount factor as separate from the MDP. We do not.

Definition 22

γ is our **discount factor**, which tells us how much we value future rewards.

$$\gamma \in \left[0, 1\right]$$

The **higher** γ is, the **more** we value future rewards.

- A reward t timesteps in the future, is worth γ^t times as much.

Because γ is never larger than 1, our discount factor can only:

- Treat future rewards as **equal** to current rewards ($\gamma = 1$) or
- Treat future rewards as **lesser** than current rewards ($\gamma < 1$)

10.1.9 Policies

We've discussed taking **actions** a: these are the **decisions** we want to model with our MDP.

- But we haven't actually created a model for **choosing** these actions.

At best, we could select a **sequence** of actions, and then see what reward we get on average. That way, we can compare them.

This strategy is based on our "sequence of inputs" from the RNN chapter.

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \tag{10.13}$$

But this approach doesn't make use of the **current state**:

- In an MDP, our state transitions are **stochastic**: we don't know exactly what the next state is going to be.
- Based on what state we get, the "best" sequence would change.

Concept 23

One weakness of using a pre-planned sequence of actions is that our model is **stochastic**:

- Depending on how our state **randomly** changes, the "best next action" changes.

Example: Imagine we model poker as an MDP. If you used the **exact same strategy** every time, while ignoring your cards (your "state"), you'd be a pretty terrible player.

It would be better if we could **adapt** our choice of **action**, based on the **state** we ended up in.

We'll construct a new function that does just that, called a **policy** π.

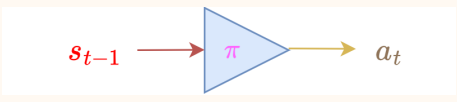
Definition 24

Our **policy** π is a function used to compute **our next action** based on our **current state**.

$$\pi(s) = a$$

One policy represents one particular **strategy** for interacting with the MDP.

$$\pi : \mathcal{S} \to \mathcal{A}$$



Example: Here's one possible policy for our blanket example.

To be fair, we should probably add a third action that is neither "add water", nor "add fire". But this is an example policy: it isn't necessary the best policy.

$$\pi(s) = \begin{cases} \textbf{Add fire} & s = \texttt{Wet} \\ \textbf{Add water} & s = \texttt{Dry} \\ \textbf{Add water} & s = \texttt{Burning} \end{cases} \tag{10.14}$$

This system allows our policy to "react" to changes in our state.

- If we run our MDP twice, and our state changes are **different**, our **policy** will choose different actions accordingly.
- **Example:** In our poker example, our policy chooses actions based on the cards in your hand, and on the table.

Note that a policy π is not necessary **optimal**:

Clarification 25

As long as you provide an **action** for every **state**, you have created a valid **policy**.

- This means that a policy doesn't have to make sense, or even be a **good** policy.

Which policy is "good" depends on entirely on our **reward function**.

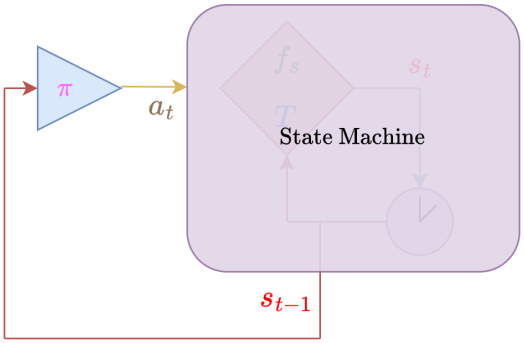
Example: Here's our "pyromaniac" policy. This is probably not optimal unless you really like setting things on fire.

But it could be, depending on the reward function.

$$\pi_p(s) = \begin{cases} \textbf{Add fire} & s = \texttt{Wet} \\ \textbf{Add fire} & s = \texttt{Dry} \\ \textbf{Add fire} & s = \texttt{Burning} \end{cases} \tag{10.15}$$

Our policy is used by our MDP to process through time, generating actions.

- So, let's add it to our MDP diagram.



Our policy chooses an action based on the state, and the state machine answers with our new state. And so, the cycle continues.

In the coming sections, we'll consider two different kinds of problems for our "game" (MDP):

- Finite horizon – we have exactly T timesteps remaining before we finish our "game".
- Infinite horizon – we have no upper limit on when the "game" ends.

But it might end eventually.

10.1.10 Value Functions

We've built up our MDP, a model to help us making **decisions**. Now, we want the **best policy**.

Concept 26

Typically, the **optimal policy** is the one that gives us the **highest rewards**, on average.

- Averaged over our **stochastic**, random state transitions.

Now, we need a way to directly **compute** this average reward, for each policy: that way, we can compare them.

We'll create a function for this purpose. It'll depend on two things:

- Our chosen **policy**
- Our current **state**

Different states earn **different** rewards, even with the **same** policy:

A perfect robot (same policy) could do better in the stock market if they had **more money** (different state).

With this information, we'll compute the average reward: this represents the worth of our policy, so we call it our **value function**.

Definition 27

A **value function** $V_\pi(s)$ gives us the **average reward** of our policy π, starting from state s.

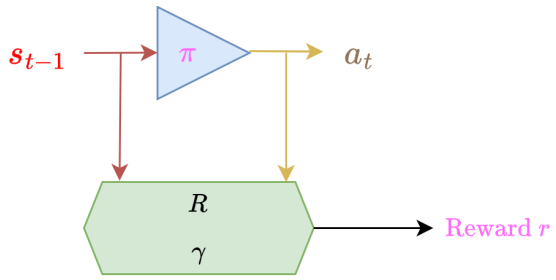
This allows us to compute which policies are "**more valuable**" in a particular **state**.

- Let's use **function** notation. Reward $r \in \mathbb{R}$ is a real number. So, the same is true for our average reward $V_\pi(s)$.

$$V_\pi : \mathcal{S} \to \mathbb{R}$$

Now, we have a general idea of what a value function is. But there's plenty of details we left out.

- For example: if we're averaging over our rewards, we need our **reward function**.
- This function is specifically designed to **evaluate** our policy π:



Our reward is based on the input/output of our policy: it can be seen as directly "observing" our policy function.

Concept 28

While our value function $V_\pi(s)$ only directly depends on π and s, there's information we need from the MDP itself:

- The **reward function** $R(s, a)$: our average reward is made up of many individual rewards, for different situations.
- The **transition function** $T(s, a, s')$: we need to know how likely different outcomes are, for our average.

In other words: our value function is specific to our MDP: two different MDPs will have completely separate value functions.

Lastly, how we compute our value function depends on whether we're using **finite** or **infinite** horizon.

10.1.11 Finite Horizon

For our first setting, we'll assume that we are going to run our MDP for a precise length of **time**, h.

- This is our **horizon**: it tells us how "far away" the end of our MDP run is.
- After that time, we don't do anything with our MDP: no actions, no state changes, no rewards.

Definition 29

The **horizon** h of our MDP is the number of timesteps until it **terminates**.

Upon each **timestep** t, our policy π chooses an **action** a_t, and our state machine transitions to **state** s_t. Then, t increases to $t+1$, and h **decreases** to $h-1$.

- When our MDP terminates, we **stop** taking actions, or receiving any reward.
- If $h = 0$, our MDP has already terminated: we can't take any more actions.

We can run our MDP as pseudocode, with MDP functions already defined:

Meaning, we've defined f_s and R already.

MDP_RUN(H, s_0, π)

```
h = H                    #H is the initial horizon
t = 0

while h > 0              #Not yet terminated
    a_t = π(s_t)         #Choose action
    s_{t+1} = f_s(s_t, a_t)      #Next state based on f_s, described by T

    yield R(s_t, a_t)    #New reward

    h = h − 1            #Update horizon
    t = t + 1

yield None               #h = 0, Program terminated
```

Note that horizon goes down, and time goes up.

Clarification 30

t indicates the **number of timesteps** since **initialization** (the start of the MDP).

h indicates the **number of timesteps** until **termination** (the end of the MDP).

- t increases, while h decreases, as our MDP runs.

Because our horizon is a finite, natural number $h \in \mathbb{N}$, we call this the **finite horizon** case.

Definition 31

If $h \in \mathbb{N}$, then we are dealing with a **finite horizon problem**.

10.1.12 Finite Horizon Value Function

Now, we've set up the problem. In this **finite-horizon** setting, we need to compute the value of different policies.

The first thing to address: we now have a third variable in our value function: **horizon**.

- The amount of **time** remaining affects the total amount of rewards you can receive.
- **Example:** If a button gives you 10 dollars, it's more valuable to be able to press it 5 times ($h = 5$) than only 2 times ($h = 2$).

Definition 32

Your **finite-horizon value function** depends on three factors: **current state**, **policy**, and **horizon**.

We notate this combination as $V_{\text{policy}}^{\text{horizon}}(\text{state})$. Typically, we write

$$V_{\pi}^{h}(s)$$

This is our function is V_{π}^{h}, taking s as an input.

- Each horizon-policy combination has its own function.

We'll use **function** notation again: it's the same as without horizon.

$$V_{\pi}^{h} : \mathcal{S} \to \mathbb{R}$$

How do we get started? This is actually pretty complicated:

- Suppose you have $h = 5$, and there are **three** possible options for your next state.
- Then, there are three possible options for the state after that. This **repeats** until $h = 0$.
- We have $3^5 = 243$ different possible outcomes, each with their own **rewards**, and **probability** of occurring.

Needless to say, it would be a massive headache to compute all of these possible outcomes at once, especially as h gets larger.

Concept 33

Because MDPs are **stochastic**, they can take many possible paths, even with the same starting state s, and same policy π.

Computing every possible outcome directly can become very **expensive**.

Instead, let's try a different tactic: we'll think of the simplest example possible. What would that be? $h = 0$.

- When $h = 0$, there are no further actions to be taken, and no more rewards.
- So, the expected reward is 0.

Key Equation 34

The total reward for a horizon-0 MDP is always 0, no matter what.

$$V_\pi^0(s) = 0$$

10.1.13 Finite Horizon, $H = 1$

We've gotten a start. Now let's try the *second* simplest example, $H = 1$. We have two stages:

- When $h = 1$, we take exactly **one** action, and get **one** reward.
- After that, $h = 0$, and we're in the same situation as before: **no more** rewards.

We'll break our problem into these two parts:

$$V_\pi^1(s) \quad = \quad \Big(h = 1 \text{ reward}\Big) + \Big(h = 0 \text{ reward}\Big) \tag{10.16}$$

Concept 35

After taking our first action at horizon h, we're in the **same situation** as if we started in horizon $(h - 1)$, with a new state s'.

Thus, we can **separate** the rewards for each timestep.

We'll find the $h = 1$ reward first.

- Our reward for each step is given by $R\Big(\texttt{state}, \texttt{action}\Big)$.
- Our state is given by s. What about our action?

- Our **policy** determines our action, based on our state: $\pi(s)$.

$$\left(h = 1 \text{ reward}\right) \quad = \quad R\left(s, \pi(s)\right) \tag{10.17}$$

This same reasoning works no matter which timestep you're starting on.

Concept 36

In our MDP, our **action** will always be determined by our **policy**:

$$a = \pi(s)$$

Thus, our reward $R\left(s, a\right)$ will always take the form

$$R\left(s, \pi(s)\right)$$

Now, the $h = 0$ reward. We've moved into state s', but that doesn't matter: our reward is always 0.

$$\left(h = 0 \text{ reward}\right) \quad = \quad V_\pi^0(s') \quad = \quad 0 \tag{10.18}$$

At $h = 1$, we gather our reward $R\left(s, \pi(s)\right)$, but we also change states from s to s'.

Taken together, we get:

$$V_\pi^1(s) = \overbrace{R\left(s, \pi(s)\right)}^{h=1 \text{ reward}} + \overbrace{V_\pi^0(s')}^{h=0 \text{ reward}} \tag{10.19}$$

Or,

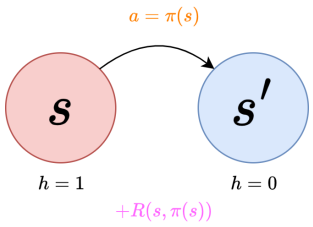
$$V_\pi^1(s) = R\left(s, \pi(s)\right) + 0 \tag{10.20}$$

Key Equation 37

The total reward for a horizon-1 MDP equals the reward for taking one **action**, according to **policy**.

$$V_\pi^1(s) = R\left(s, \pi(s)\right)$$

We can represent this similar to a state-transition diagram:



We take one action $\pi(s)$, and get a reward $R(s, a)$.

Note that, while we show only one future state s', this is just one *possible* outcome.

- You could think of this as one run of our MDP.

10.1.14 Finite Horizon, $H = 2$

Next, we'll tackle $H = 2$.

- When $h = 2$, we take action $\pi(s)$, moving from s to s'.
- When $h = 1$, we take action $\pi(s')$, moving from s' to s''.

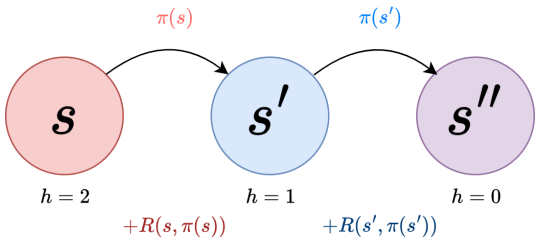
We already know $h = 0$ gives **no reward**.

We can separate out the reward for each timestep:

$$V_\pi^2(s) = \Big(h = 2 \text{ reward}\Big) + \Big(h = 1 \text{ reward}\Big) + \Big(h = 0 \text{ reward}\Big)$$

And remove $h = 0$:

$$V_\pi^2(s) = \Big(h = 2 \text{ reward}\Big) + \Big(h = 1 \text{ reward}\Big)$$



Two actions, two rewards.

First, we'll compute the $h = 2$ reward.

- This is the same as the the $H = 1$ problem: we start in state s, and take action $\pi(s)$.

$$\Big(h = 2 \text{ reward}\Big) \quad = \quad R\Big(s, \pi(s)\Big) \tag{10.21}$$

Now, we want to compute the $h = 1$ reward.

- After $h = 2$, we've moved to state s'. We try to do what we did before:

$$\Big(h = 1 \text{ reward}\Big) \quad = \quad R\Big(s', \pi(s')\Big) \qquad \text{...Maybe?} \tag{10.22}$$

But we have a serious problem: which state is s'?

Concept 38

Our **state transition** is **probabilistic**.

- That means that there are **several new states** s' we could transition to.

So, we can't just directly use $R\Big(s', \pi(s')\Big)$: we don't know which s' will be, and thus our reward.

In other words, the reward for $h = 1$ will depend on the random outcome of our **transition function** T.

How do we resolve this? The key is to remember that V is computing **average reward**: meaning, we'll average our reward over all possible transitions.

A quick review on how to do that:

Concept 39

Suppose we have a **random variable** X with multiple possible outcomes x_i. We want to get the average, or **expected value** $\mathbb{E}[X]$.

- The more **likely** an outcome, the more it **contributes** to the average.

Thus, $\mathbb{E}[X]$ is a **weighted average** of each outcome, **times** its probability.

$$\mathbb{E}[X] = \overbrace{\sum_i}^{\text{All outcomes}} x_i \cdot P(x_i)$$

Example: Suppose you're betting on something: there's a 60% chance you make \$100, and a 40% chance you lose \$40. The average earnings (average **reward**) is:

$$\mathbb{E}[X] = \sum_i x_i \cdot P(x_i) \quad = \quad 0.6 * 100 - 0.4 * 40 = 44 \tag{10.23}$$

Now, we have the tools we need to proceed.

Let's apply this averaging to our current situation:

- We want to get the average reward, $R\left(s', \pi(s')\right)$.
- We'll average over possible states s'.

$$\left(h = 1 \text{ reward}\right) = \underbrace{\sum_{s_i \in \mathcal{S}} R\left(s_i, \pi(s_i)\right) \cdot P(s' = s_i)}_{\text{Averaging over possible transitions to } s_i} \tag{10.24}$$

What's our probability $P(s' = s_i)$?

- Probabilities are determined by our transition function $T\left(s, a, s'\right)$.
- $T\left(s, \pi(s), s'\right)$ represents our transition from $h = 2$ to $h = 1$.

In this situation, $T\left(s, \pi(s), s_i\right)$ is "the probability of moving to state s_i".

Because we already know our initial state s and our policy π.

$$P(s' = s_i) = T\left(s, \pi(s), s_i\right) \tag{10.25}$$

$$\left(h = 1 \text{ reward}\right) = \sum_{s_i \in \mathcal{S}} \underbrace{T\left(s, \pi(s), s_i\right)}_{\text{Chance of state } s_i} \cdot \underbrace{R\left(s_i, \pi(s_i)\right)}_{\text{Reward for state } s_i} \tag{10.26}$$

Key Equation 40

Suppose you know the **state** s of your MDP at horizon H.

- You can compute the reward for the next timestep, horizon $H - 1$, by **averaging over** all of the possible state transitions, to each state s'.

$$\left((h = H - 1) \text{ reward}\right) = \sum_{s'} T\left(s, \pi(s), s'\right) \cdot R\left(s', \pi(s')\right)$$

We can assemble our full solution:

Key Equation 41

The total reward for a horizon-2 MDP adds two rewards:

- The reward for taking one **action**, according to **policy**.
- **Average** of all possible outcomes, based on the result of the **state transition**.

$$V_\pi^2(s) = \overbrace{R\Big(s,\pi(s)\Big)}^{h=2 \text{ reward}} + \overbrace{\sum_{s'} T\Big(s,\pi(s),s'\Big) \cdot R\Big(s',\pi(s')\Big)}^{h=1 \text{ reward}}$$

Without the annotations, we see:

$$V_\pi^2(s) = R\Big(s,\pi(s)\Big) + \sum_{s'} T\Big(s,\pi(s),s'\Big) \cdot R\Big(s',\pi(s')\Big)$$

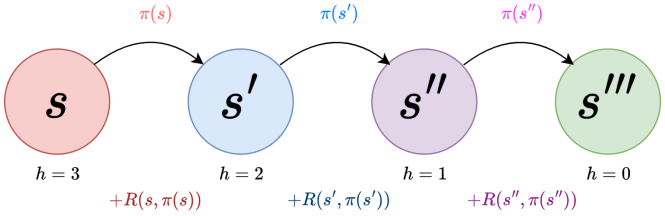
10.1.15 Finite Horizon, $H = 3$ and beyond

Finally, we tackle $H = 3$. And with this step, we come up with a strategy for all finite horizons.

- For $h = 3$, we start in state s, and use our policy to act.
- For $h = 2$, our action depends on the state s' we end up in, **stochastically**.
  - The same is true for $h = 1$: our action depends on the state s''.

Remember: "stochastic" describes our random state transition, with known probabilities.

$h = 0$ gives **no reward**.



Three actions: s' and s'' can vary based on our random transitions.

The reward for $h = 3$ is the simplest: we know what **state** we're in, and our policy's action.

What about $h = 2$? Well, we could **separately** get the reward for $h = 2$, and $h = 1$. But *instead*, we'll consider the following:

- Suppose that we've already selected our new state, s'. If we know our new state, this is **identical** to the $H = 2$ problem.

Concept 42

If we know our **current state**, we actually **don't care** about what happened during earlier timesteps.

- That means, if our current horizon is $h = 2$, we can pretend as if we're starting a new MDP run with $H = 2$.

Let's forget $h = 3$ for a moment:

- We're starting with $H = 2$, from state s'.
- We want to compute the total reward from this state, using our policy π.
- We **already computed** this above: the result is $V_\pi^2(s')$.

Concept 43

$V_\pi^H(s')$ includes all the rewards for all timesteps between $h = H$ and $h = 0$.

$$V_\pi^H(s) = \sum_{h=0}^{H} \Big(\text{Immediate reward for horizon } h\Big)$$

Or,

$$V_\pi^H(s) = \Big(\text{Rewards for } h \leqslant H\Big)$$

That means, if we've already computed $V_\pi^H(s)$, it can handle the last H steps of our MDP:

$$V_\pi^{H+1}(s) = \Big(\text{Reward for horizon } H+1\Big) + \Big(\text{Rewards for } h \leqslant H\Big)$$

Remember to keep V and R separate in your mind:

Clarification 44

Our value function $V_\pi^h(s)$ gives us the **expected reward for all current and future steps**, given a **state** and **policy**.

Our reward function $R\Big(s, \pi(s)\Big)$ gives us the **reward for one step**, given a **state** and **action**.

There's one important detail missing, though: this all assumes that we know s'. But we don't, because of our random transitions.

- So, once again, we have to **average** over all possible s'.

$$\Big((h \leqslant H-1) \text{ rewards}\Big) = \sum_{s_i \in \mathcal{S}} P(s' = s_i) \cdot \overbrace{V_\pi^{H-1}(s_i)}^{\text{Future rewards if } s'=s_i} \tag{10.27}$$

Again, we use our **transition function**:

$$\Big((h \leqslant H-1) \text{ rewards}\Big) = \sum_{s_i \in \mathcal{S}} T\Big(s, \pi(s), s'\Big) \cdot V_\pi^{H-1}(s_i) \tag{10.28}$$

Concept 45

Often, we **don't know** our next state, but we know the **probability distribution** of states.

In this situation, we can:

- Use T to average all of our value functions, to get the **average future rewards**.

Now, we combine this with our first reward, for the initial H step. Because we **know** our state and policy, we don't have to use the transition function for our **first** step.

This is always true for the $h = H$ step.

$$\Big((h \leqslant H) \text{ rewards}\Big) = \Big((h = H) \text{ reward}\Big) + \Big((h \leqslant H-1) \text{ rewards}\Big) \tag{10.29}$$

$$V_\pi^H(s) = R\Big(s, \pi(s)\Big) + \sum_{s_i \in \mathcal{S}} T\Big(s, \pi(s), s_i\Big) \cdot V_\pi^{H-1}(s_i) \tag{10.30}$$

If we apply this to our $H = 3$ example, we get:

$$V_\pi^3(s) = R\Big(s, \pi(s)\Big) + \sum_{s_i \in \mathcal{S}} T\Big(s, \pi(s), s_i\Big) \cdot V_\pi^2(s_i) \tag{10.31}$$

Key Equation 46

The total reward for a horizon-H MDP $V_\pi^H(s_i)$ is broken into two parts:

- The reward for our **first action**.
- The reward for **all future steps**, based on the **first transition** $s \to s'$.
  - The "all future steps" reward has already been computed for the **horizon-$(H-1)$** MDP, $V_\pi^{H-1}(s')$.
  - We'll average over each possible "first transition".

$$V_\pi^H(s) = \overbrace{R\Big(s, \pi(s)\Big)}^{\text{First reward}} + \overbrace{\sum_{s'} \underbrace{T\Big(s, \pi(s), s'\Big)}_{\text{Chance of } s \to s'} \cdot \underbrace{V_\pi^{H-1}(s')}_{\text{Reward if } s \to s'}}^{\text{Future rewards}}$$

If we remove the clutter, our equation is:

$$V_\pi^H(s) = R\Big(s, \pi(s)\Big) + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot V_\pi^{H-1}(s') \tag{10.32}$$

10.1.16 Finite Horizon MDP Solution

We've created a general solution:

- If $H = 0$, the reward is 0.
- If $H > 0$, then we **break** the reward into two parts:
  - The **immediate** reward
  - The **future** rewards, based on the $H - 1$ case.

Since horizon H depends on horizon $H - 1$, we typically "**build up**" our solution, starting from $H = 1$:

- We compute V_π^H, and then V_π^{H+1}, gradually increasing horizon H.
- That way, when we do horizon H, we already have the **value functions** we need (from $H - 1$).

Let's show this gradual progression:

$$V_\pi^0(s) = 0 \tag{10.33}$$

$$V_\pi^1(s) = R\Big(s, \pi(s)\Big) + 0 \tag{10.34}$$

$$V_\pi^2(s) = R\Big(s, \pi(s)\Big) + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot V_\pi^1(s') \tag{10.35}$$

And if we keep going...

$$V_\pi^H(s) = R\Big(s, \pi(s)\Big) + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot V_\pi^{H-1}(s') \tag{10.36}$$

Definition 47

Here, we present the equations for computing the **value functions** for a **finite-horizon MDP**:

$$V_\pi^0(s) = 0$$

$$V_\pi^H(s) = R\Big(s, \pi(s)\Big) + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot V_\pi^{H-1}(s')$$

We start by computing $V_\pi^1(s)$, then $V_\pi^2(s)$, and so on, incrementing our horizon each time.

- And at each step, we use the previous value function.

Once we know how to compute the value function for horizon h, we can compare different policies:

Concept 48

Suppose we have a given horizon h.

If policy π_1 is **better than** π_2, then:

- For every state s, π_1 is at least as good as π_2.

$$\forall s \in \mathcal{S}: \qquad V_{\pi_1}^h(s) \geqslant V_{\pi_2}^h(s)$$

- There is at least one state $\hat{s}$ where π_1 is better than π_2.

$$\exists \hat{s} \in \mathcal{S}: \qquad V_{\pi_1}^h(\hat{s}) > V_{\pi_2}^h(\hat{s})$$

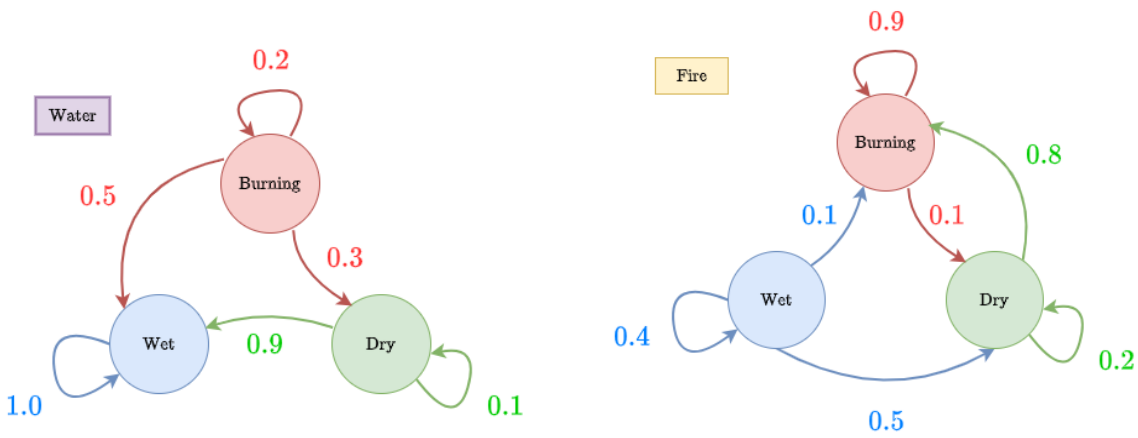
In other words: "π_1 is always at least as good as π_2, and sometimes better".

$\forall$ just means "for all / every", while $\exists$ means "there is at least one".

10.1.17 Finite-Horizon, using our Blanket Example (Optional)

Example: We can apply this to our blanket problem.

First, our transitions:



Now, our rewards:

$$R(s,a) = \begin{cases} 10 & s = \text{Dry} \\ 0 & s = \text{Wet} \\ -20 & s = \text{Burning} \end{cases} \tag{10.37}$$

Finally, our policy:

$$\pi(s) = \begin{cases} \textbf{Add fire} & s = \text{Wet} \\ \textbf{Add water} & s = \text{Dry} \\ \textbf{Add water} & s = \text{Burning} \end{cases} \tag{10.38}$$

We have 3 states: 3 value functions for each horizon h.

Notation 49

Often, it's easier to write each **value function** in a **condensed** form, to make equations easier to read.

- This is best used for your personal calculations: you need to be clear about what system you're using to **abbreviate**.

In this case, we'll use:

$$V_\pi^h(\text{Dry}) = d_h \qquad V_\pi^h(\text{Wet}) = w_h \qquad V_\pi^h(\text{Burning}) = b_h$$

The reward for $h = 0$ is 0, no matter our state.

$$d_0 = 0 \qquad w_0 = 0 \qquad b_0 = 0 \tag{10.39}$$

The reward for $h = 1$ uses the formula

This is a simplified version of our general equation: because $V_\pi^0(s) = 0$, the future steps can be skipped.

$$V_\pi^1(s) = R\Big(s, \pi(s)\Big)$$

Our immediate reward is only based on our current state, according to $R\Big(s, a\Big)$.

$$d_1 = 10 \qquad w_1 = 0 \qquad b_1 = -20 \tag{10.40}$$

For $h = 2$, we have to use our general formula

$$V_\pi^H(s) \;=\; R\Big(s, \pi(s)\Big) + \sum_{s'} \; T\Big(s, \pi(s), s'\Big) \cdot V_\pi^{H-1}(s')$$

- If it's dry, add water.

$$d_2 = \overbrace{10}^{R_d} + \overbrace{\Big(0.1d_1 + 0.9w_1\Big)}^{\sum_{s'} T(\ldots) V_\pi^1(s')} \;=\; 11 \tag{10.41}$$

- If it's wet, add fire.

$$w_2 = \overbrace{0}^{R_w} + \overbrace{\Big(0.5d_1 + 0.4w_1 + 0.1b_1\Big)}^{\sum_{s'} T(\ldots) V_\pi^1(s')} \;=\; 3 \tag{10.42}$$

- If it's burning, add water.

$$b_2 = \overbrace{-20}^{R_b} + \overbrace{\Big(0.3d_1 + 0.5w_1 + 0.2b_1\Big)}^{\sum_{s'} T(\ldots) V_\pi^1(s')} \;=\; -21 \tag{10.43}$$

We can repeat this process, with the same equations, for **larger and larger** horizons, up to any particular h.

10.1.18 Infinite Horizon

In finite horizon, we knew exactly how long our MDP would run for.

- But often, we don't know how long it'll run for: or at the very least, it'll be a **very long time**.

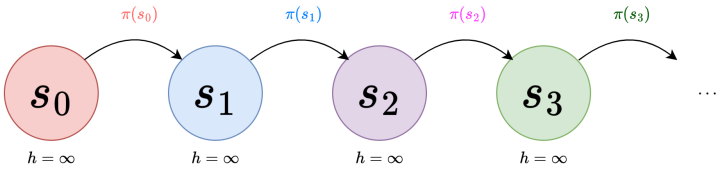
So, we'll consider an **infinite horizon**:

Definition 50

In the **infinite horizon problem**, we **don't know** how long our MDP will run: it can run for a long time.

In this situation, our horizon **never changes**: if you take n steps, you still don't know when the MDP will end.

We call this "infinite" because we've moved our 100% certain **horizon** infinitely far away.



No matter how many steps we take, the horizon is always the same.

So, we don't use it as a variable.

Definition 51

Our **infinite-horizon value function** V_π^∞ doesn't require a horizon, because it's always the same.

This value function behaves the same as before: input is state s, output is **average reward** r.

$$V_\pi^\infty : \mathcal{S} \to \mathbb{R}$$

Note:

Notation 52

We can notate our **infinite-horizon value function** two ways:

$$V_\pi^\infty(s) = V_\pi(s)$$

For readability, we'll use $V_\pi(s)$ in following sections.

10.1.19 Discounting

This model has a major bug we need to work out: infinite rewards.

Concept 53

If we run our MDP for an "infinite" amount of time, it can get an **infinite** amount of **rewards**.

This makes it difficult to **compare** two different policies that both have infinite value.

Example: Imagine you have two policies: one presses the "earn \$10" button, and the other presses the "earn \$20" button.

- Over an infinite timescale, both policies have a value of ∞, even though one is obviously better.

Our solution is to consider the **discounted infinite horizon**: we treat future rewards as less valuable ("discounted") than present rewards.

This brings us back to our γ factor from the beginning of the chapter:

Definition 54

γ is our **discount factor**, which tells us how much we value future rewards.

$$\gamma \in [0, 1]$$

The **higher** γ is, the **more** we value future rewards.

- A reward t timesteps in the future, is worth γ^t times as much.

Because γ is never larger than 1, our discount factor can only either:

- Treat future rewards as **equal** to current rewards ($\gamma = 1$, **finite horizon**)
- Treat future rewards as **lesser** than current rewards ($\gamma < 1$, **infinite horizon**)

Example: We'll re-use the above example, and use $\gamma = 0.9$.

- So, for each timestep, we scale down the reward by 0.9^t.

For simplicity, our first timestep is $t = 0$, not $t = 1$.

If we have the same reward r for every time step, we get:

This is a geometric sum!

$$V_\pi(s) = r + r\gamma + r\gamma^2 + \cdots \quad = \quad r\sum_{t=0}^{\infty}\gamma^t = \frac{r}{1-\gamma} \tag{10.44}$$

Let's compare our \$20 button to our \$10 button.

$$V_{\$10}(s) = 100 \qquad V_{\$20}(s) = 200 \tag{10.45}$$

Now, it's clear which policy is better.

Definition 55

For finite horizon, our value function is given by the **average total rewards**.

$$V_\pi^h(s) = \mathbb{E}\left[\sum_{t=1}^{h} R_t\right]$$

For infinite horizon, our value function is given by the **average discounted rewards**.

$$V_\pi(s) = \mathbb{E}\left[\sum_{t=1}^{h} \gamma^t R_t\right]$$

In both cases, R_t is based on our **state** and **policy**.

Remember that $\mathbb{E}\left[\cdot\right]$ gives our average, or **expected value**.

There are a few justifications for why we could discount future value:

- Economics: **money** is worth more now than in the future.
- Predictability: in the real world, it's harder to accurately predict the distant future than near future: distant rewards are thus less reliable/valuable.

10.1.20 Discount factor: Termination

One of the most useful interpretations is **termination**:

We call our horizon "infinite", meaning that we don't have a **definite stopping point** $h = 0$ for our MDP.

- But, that *doesn't* mean it never terminates: we just don't know **when**.

Instead, consider this: at every timestep, our MDP has a $(1-\gamma)$ chance of terminating. We can only get our **reward** if it **doesn't terminate**: probability γ.

- The chance of our model surviving for t timesteps, and thus receiving reward R_t, is γ^t.

Concept 56

We can interpret our infinite-horizon MDP having an "opportunity" to fail("terminate"), after every timestep.

- Our **discount factor** γ is the chance of our MDP **continuing**.
- $1-\gamma$ is the chance of our MDP **terminating**.

So, while our MDP is unlikely to run "forever", there's no fixed horizon h that it will *definitely* terminate at.

This is why we scale down our reward for R_t by a factor of γ^t: because we're computing **expected reward**.

$$\mathbb{E}\left[r_t\right] = \overbrace{\gamma^t \cdot R_t}^{\text{Model continues}} + \overbrace{(1-\gamma^t)\cdot 0}^{\text{Model terminates}} \tag{10.46}$$

- Note that we can "miss" the reward for R_t by failing at **any step** before t.

Make sure not to confuse $(1-\gamma^t)$ with $(1-\gamma)^t$: compare **exponents**.

If the model terminates, we get 0 reward. So, we can ignore that part of the reward.

$$\mathbb{E}\left[r_t\right] = \gamma^t \cdot R_t \tag{10.47}$$

And this is how we get our $V_\pi(s)$ expression above.

10.1.21 Lifespan of our MDP

As we mentioned, the horizon of our infinite-horizon model never changes.

But we also find something more surprising: **average future lifespan** (average number of timesteps until termination) is the **exact same**, before and after one step.

We could think of this as the "average horizon": on average, how long does our MDP have left?

Let's find out why.

Example: Suppose you're repeatedly flipping a fair coin. You count the number of tails you get. If you ever get heads, you **stop playing**.

A "fair coin" is 50% heads, 50% tails.

Suppose you flip your coin once, and get tails. What are the odds of your next coin being heads?

- They're **the same** as they were before that first coin toss: 50%.

What if you flip your coin, and get 10 tails in a row?

- The odds of your next coin being heads is **still** 50%.

Concept 57

In an MDP, the **odds of termination**, $(1-\gamma)$, are **constant**.

We can apply this to the **average amount of time left** until the model terminates.

- **Example:** Suppose you just got 10 tails in a row. Technically speaking, the **average remaining time** until the game ends is unchanged.
- The coin doesn't "**remember**" that it just won 10 times: you have exactly the same coin as if you just started playing.

Consider the opposite case: if "winning 10 times" decreased the remaining time, that means that our coin is suddenly more likely to lose. Why would the coin change?

We can describe this argument for all MDPs:

- This "average future lifespan" is based on the odds of termination at each turn.
- Since the odds of termination γ are the same, the **average future lifespan** is unchanged.

Clarification 58

The fact that the **average future lifespan** of our MDP is **unchanged**, is often counter-intuitive.

We'll address two points of confusion:

- As a real machine gets older, it's **more likely** to break.
  - But that's because the odds of it breaking **each day** are going up, because it wears out.
  - Nothing in our MDP is "**wearing out**": it just might spontaneously stop.
- It feels like the model should've "**lost**" some of its life, after time passes.
  - This "lost" life is accounted for by the situation where the model immediately **failed**.
  - What's changed about the situation is we know the model model **succeeded**: we're biasing towards a longer total lifetime.

So, your "average future lifespan" is the same, while your "average total lifespan" has increased by 1.

The "average future lifespan", by the way, is always $\frac{1}{1-\gamma}$. Why?

- $(1-\gamma)$ gives our failure odds.

- If the odds of something happening are 1/4, you would **expect** to wait 4 times for it to happen.
- So, you just take the **reciprocal**.

It happens 1 in 4 times, after all.

Key Equation 59

The **average time until termination** ("average future lifespan") of our model is always

$$\frac{1}{1-\gamma}$$

We always have the same horizon, and the same properties. This is a result of the **markov property**:

Definition 60

MDPs are **memoryless**.

- **Future states** are **only** affected by the **current state and action**.
- Information in the **past** has no effect: thus, our MDP doesn't have "memory" of these past events.

This is also called the **markov property**.

One effect of this property is that the **average future lifespan** is **always the same**.

One more note:

Remark (Optional) 61

Often, in probability, we're referring to a "**weaker**" (having less requirements) version of **memorylessness**:

- It **doesn't matter** how long we've been waiting for an event: the time until that event (waiting time) **does not change**.

This is equivalent to the idea that "the **average future lifespan** never changes".

10.1.22 Infinite Horizon Value Function

Let's figure out our value function, starting from the finite horizon one:

$$V_\pi^H(s) = R\Big(s, \pi(s)\Big) + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot V_\pi^{H-1}(s') \tag{10.48}$$

First of all: our **horizon never changes**: if our model didn't terminate, we're in the same situation after each timestep.

Concept 62

Past events do not matter to our **infinite-horizon MDP**.

- If we already survived t timesteps in the **past**, that has no effect on the odds of surviving in the **future**.
- We're in the same situation as when we started.

Example: Suppose that every round of a game, you **flip a coin** to decide whether you'll continue.

- Knowing that you've already survived 5 rounds has **no effect** on how likely you are to survive the next round.
- We can **ignore** those earlier rounds, and pretend as if we've just started playing.

This simplifies our work: we just remove the horizon.

$$V_\pi(s) = R\Big(s, \pi(s)\Big) + \sum_{s'} T\Big(s, \pi(s), s'\Big) \cdot V_\pi(s') \qquad (\text{if } \gamma = 1) \tag{10.49}$$

$V_\pi(s')$ doesn't care how many rounds it's been since we started.

Now, we need to account for γ.

Originally, we added T to represent the **odds** of ending up in state s', and thus getting the reward from $V_\pi(s')$.

$$\mathbf{P}\Big\{s' \text{ reward}\Big\} = T\Big(s, \pi(s), s'\Big) \tag{10.50}$$

But that was when we knew that our model would continue. Now, we have to include two **independent** events:

- We end up in state s': $T\Big(s, \pi(s), s'\Big)$
- Our model doesn't **terminate** immediately: γ.

We just have to multiply these odds together.

$$\mathbf{P}\left\{s' \text{ reward}\right\} = \overbrace{\gamma}^{\text{MDP doesn't terminate}} \cdot \overbrace{T\left(s, \pi(s), s'\right)}^{\text{End up in state } s'} \tag{10.51}$$

Concept 63

In our **infinite MDP**, we have to include our γ factor, **diminishing** the value of future rewards by that factor.

We get:

$$V_\pi(s) = R\left(s, \pi(s)\right) + \sum_{s'} \gamma \cdot T\left(s, \pi(s), s'\right) \cdot V_\pi(s') \tag{10.52}$$

Or, we can even pull γ out of the sum: all future rewards assumes the model doesn't terminate.

Key Equation 64

We can compute the **infinite-horizon value function** using the following, **recursive** equation.

$$V_\pi(s) = R\left(s, \pi(s)\right) + \gamma \sum_{s'} T\left(s, \pi(s), s'\right) \cdot V_\pi(s')$$

We call it **recursive** because V_π references itself in this equation.

10.1.23 Solving the Infinite-Horizon Value Function

We can't solve this version like how we'd solve the **finite-horizon** case: it doesn't make as much sense to do $h = 1$, $h = 2$, etc.

But this version is, in some ways, **simpler**:

- The equation for $V_\pi(s)$ contains other V_π functions.
- And we have one equation for each state s.

So, if we gather all of the equations for every $s \in \mathcal{S}$, we can **solve** for each value function.

We're also in luck: our value function equations are **linear!**

Concept 65

In order to compute the **infinite-horizon value function** for every state, we can follow these steps:

- Write out the equation for value function $V_\pi(s)$ for **every state**, using the other value functions $V_\pi(s')$.
- If we have n states, we now have n equations.
- Solve our n **equations** for our n **variables**, as a **linear system**.

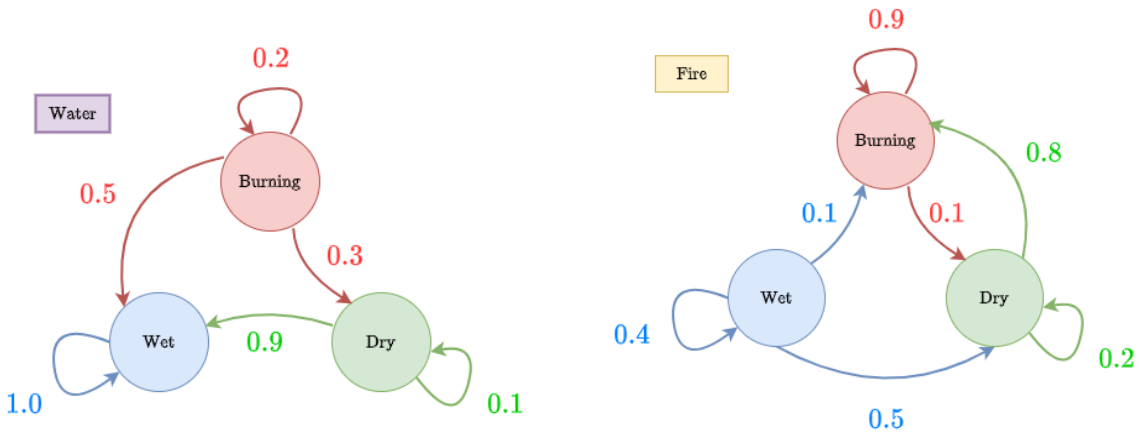
The number of states is the same as the size of of our set $\mathcal{S}$. So, we can also write it as $n = |\mathcal{S}|$.

You can use any of our known tactics for solving linear systems.

10.1.24 Infinite-Horizon, using our Blanket Example (Optional)

Example: We can apply this to our blanket problem. Let's set our discount factor at $\gamma = 0.8$.

First, our transitions:



Now, our rewards:

$$R(s, a) = \begin{cases} 10 & s = \text{Dry} \\ 0 & s = \text{Wet} \\ -20 & s = \text{Burning} \end{cases} \tag{10.53}$$

Finally, our policy:

$$\pi(s) = \begin{cases} \textbf{Add fire} & s = \text{Wet} \\ \textbf{Add water} & s = \text{Dry} \\ \textbf{Add water} & s = \text{Burning} \end{cases} \tag{10.54}$$

We want three value functions.

Notation 66

Often, it's easier to write each **value function** in a **condensed** form, to make equations easier to read.

- This is best used for your personal calculations: you need to be clear about what system you're using to **abbreviate**.

In this case, we'll use:

$$V_\pi(\text{Dry}) = d \qquad V_\pi(\text{Wet}) = w \qquad V_\pi(\text{Burning}) = b$$

Let's get the equations for each value function.

- If it's dry, add water.

$$\mathrm{d} = \overbrace{10}^{R_d} + \underbrace{0.8}_{\gamma} \cdot \overbrace{\left(0.1\mathrm{d} + 0.9w\right)}^{\sum_{s'} T(\ldots) V_\pi(s')} \tag{10.55}$$

- If it's wet, add fire.

$$w = \overbrace{0}^{R_w} + \underbrace{0.8}_{\gamma} \cdot \overbrace{\left(0.5\mathrm{d} + 0.4w + 0.1\mathrm{b}\right)}^{\sum_{s'} T(\ldots) V_\pi(s')} \tag{10.56}$$

- If it's burning, add water.

$$\mathrm{b} = \overbrace{-20}^{R_b} + \underbrace{0.8}_{\gamma} \cdot \overbrace{\left(0.3\mathrm{d} + 0.5w + 0.2\mathrm{b}\right)}^{\sum_{s'} T(\ldots) V_\pi(s')} \tag{10.57}$$

If we solve this **linear system**, we get:

$$\begin{aligned} \mathrm{d} &= V_\pi(\mathrm{Dry}) \approx 17.6 \\ w &= V_\pi(\mathrm{Wet}) \approx 8.7 \\ \mathrm{b} &= V_\pi(\mathrm{Burning}) \approx -14.6 \end{aligned} \tag{10.58}$$