Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Spring 2023

Contents

CHAPTER 12

Markov Decision Processes

In the RNN chapter, we introduced **state machines (SMs)**: a system for keeping track of our **current situation**, using a "state".

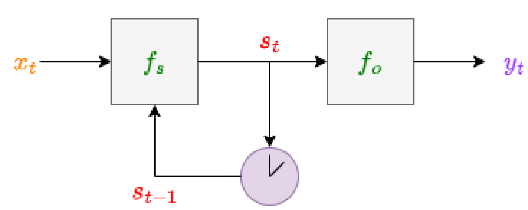
- This state also contains information about the **past**: our **present** state is influenced by past data.

In particular, we used **finite state machines**.

Clarification 1

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

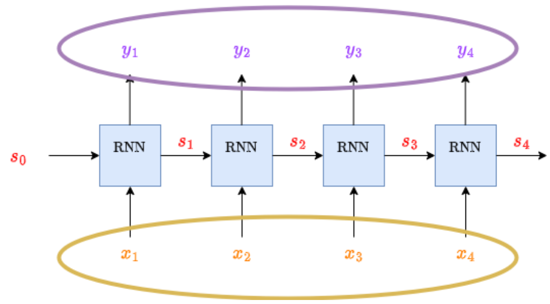
- Our set of **states** and set of **inputs** are finite.



A reminder of what our state machine looks like.

12.0.1 Sequence-to-sequence perspective

At the time, this was used in RNNs, to process **sequences** of information:



Using our state to keep information over "time", we turned one time-sequence into another.

- Our state could be seen as "what we know", updated with new information x_t.
- While our output is **based on** that new information.

12.0.2 A new perspective: the "outside world"

This time, we want to choose a *different* way to view our state machine.

- 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

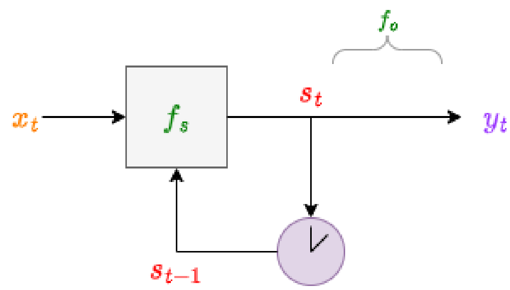
We can view the **state** of our state machine as the current state of the **outside world** that we're modeling.

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.

12.0.3 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**.

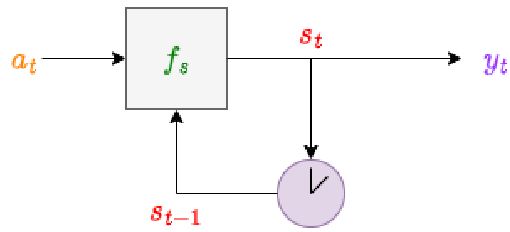
Concept 3
We want to be able to deliberately **modify** the state of the **outside world** though our interactions.

- 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.

12.0.4 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

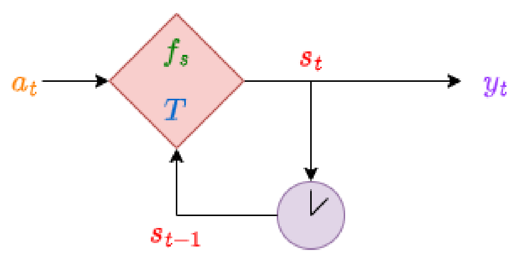
We want to be able to represent the **stochastic** (randomized) nature of our world.

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**.

12.0.5 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:

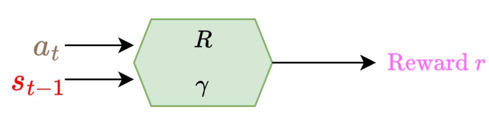
Concept 8
Our reward is determined by the **state-action pair** it receives. This creates two different aspects that weigh into our decision:

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

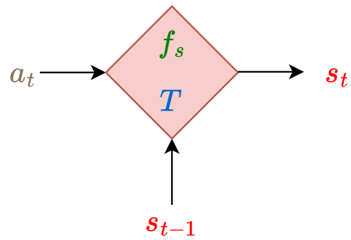
Often, our model must choose between an action that moves you into a "more rewarding" **state**, or an action that gives you higher **immediate reward**.

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.

12.0.6 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:

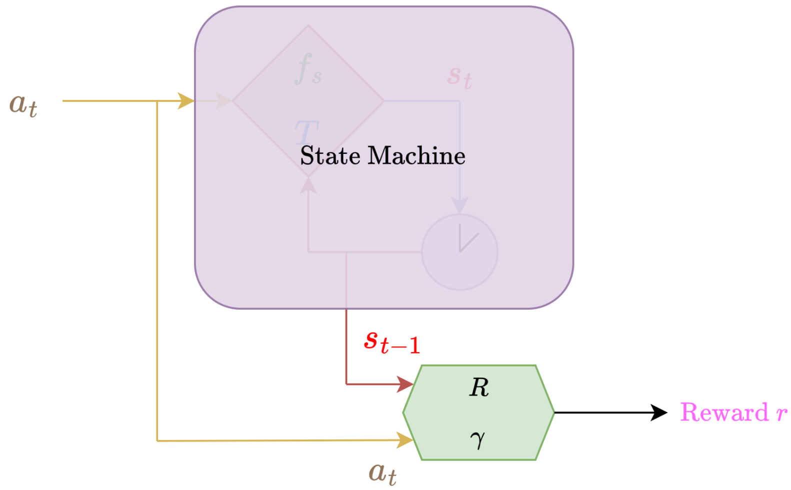
- We include a **reward** function to evaluate the quality of these decisions, based on the **state-action** pair.

Remark (Optional) 11

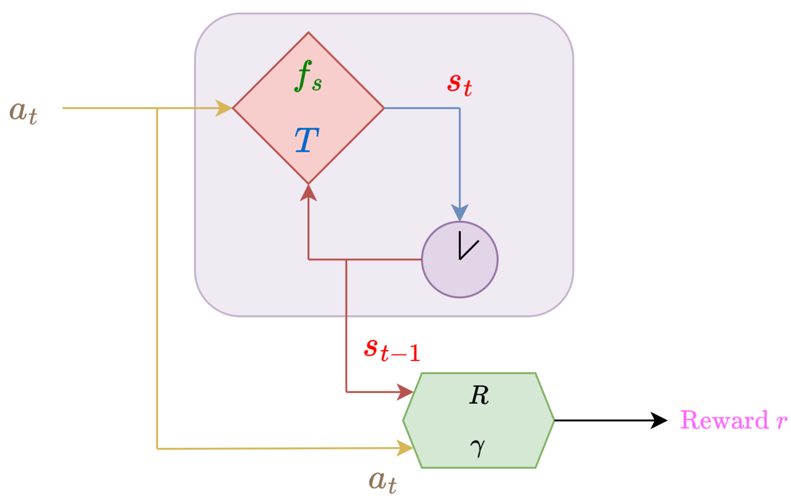
Alternatively, we can view an MDP as a **markov chain**, with two modifications:

- **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.

12.1 Definition and Value Functions

12.1.1 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.

12.1.2 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:

$$T(s, a, s') \quad = \quad \text{Probability that, in } \textbf{state } s\text{, } \textbf{action } a \text{ results in } \textbf{new state } s' \tag{12.1}$$

In more mathematical terms:

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

$$T(s, a, s') \quad = \quad \mathbf{P}\left\{ S_t = s' \;\middle|\; S_{t-1} = s, \quad A_t = a \right\} \tag{12.2}$$

Definition 13

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

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

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

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: $\left[0, 1\right]$

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

It would also be valid to write $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, because $\left[0,1\right]$ 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.

$$T\Big(\text{Dry}, \text{Water}, \text{Healthy}\Big) = 0.95 \tag{12.3}$$

- 5% chance of becoming sick.

$$T\Big(\textbf{Dry}, \text{Water}, \text{Sick}\Big) = 0.05 \tag{12.4}$$

12.1.3 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 **s'** adds to 1:

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

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:

$$\begin{aligned} &T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \left[0,1\right] \qquad && f_s : \mathcal{S} \times \mathcal{A} \to \mathcal{S} \\ &T(s,a,s') = p && f_s(s,a) = s' \end{aligned} \tag{12.5}$$

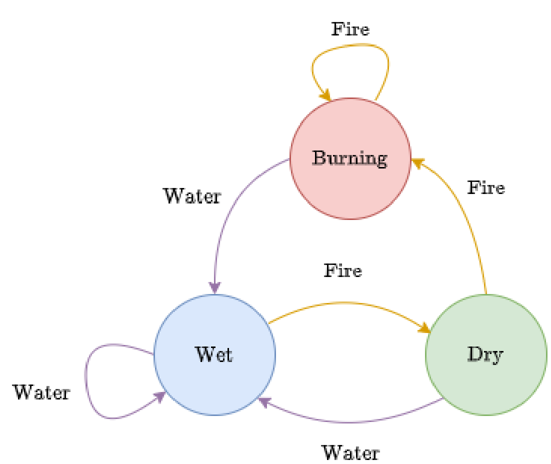
12.1.4 State-Transition Diagram: Review

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".



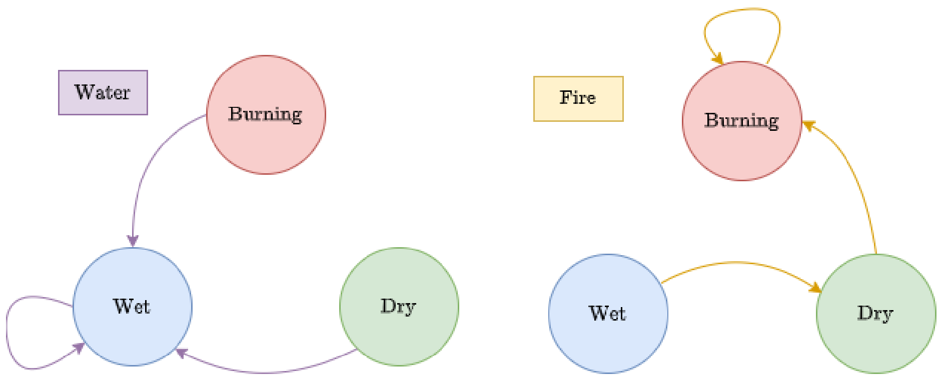
We want to update this to include transition probabilities.

- 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:



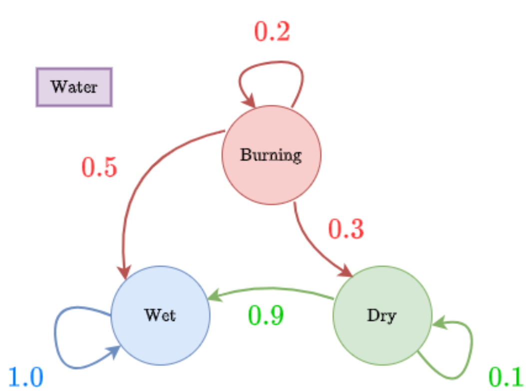
The left diagram uses **water** as an input, while the right diagram uses **fire** as an input.

12.1.5 State-Transition Diagram: Probabilistic

Now, we can extend these diagrams with probabilities.

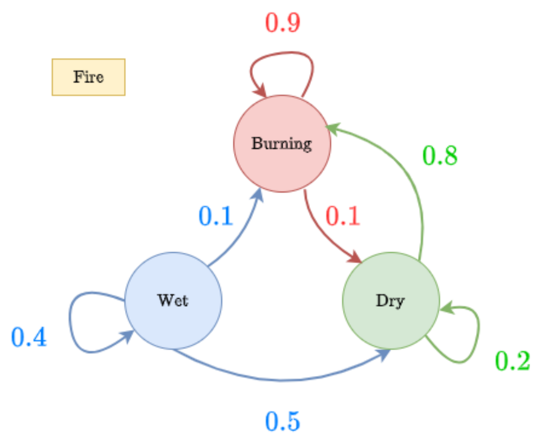
We'll use the following:

- 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.

12.1.6 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).

$$\mathcal{T}(a) \quad = \quad \begin{array}{c} \text{Input state} \\ s \end{array} \left\{ \quad \overbrace{\begin{bmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}}^{\text{Output state } s'} \right. \tag{12.6}$$

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

$$\mathcal{T}(a)_{ij} = T\Big(s_i, a, s_j\Big) \tag{12.7}$$

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:

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

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

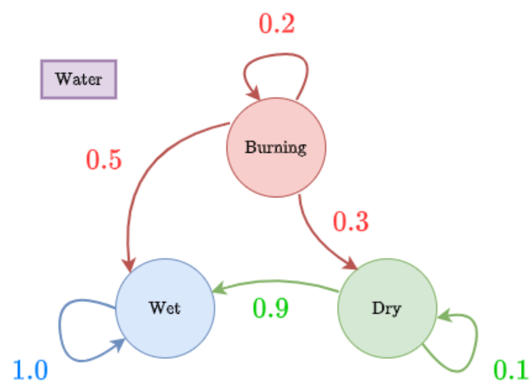
Example: Suppose we have 3 states: s_1, s_2, s_3. Our matrix for action a looks like:

$$\mathcal{T}(a) \quad = \quad \begin{bmatrix} T\Big(s_1, a, s_1\Big) & T\Big(s_1, a, s_2\Big) & T\Big(s_1, a, s_3\Big) \\ T\Big(s_2, a, s_1\Big) & T\Big(s_2, a, s_2\Big) & T\Big(s_2, a, s_3\Big) \\ T\Big(s_3, a, s_1\Big) & T\Big(s_3, a, s_2\Big) & T\Big(s_3, a, s_3\Big) \end{bmatrix} \tag{12.8}$$

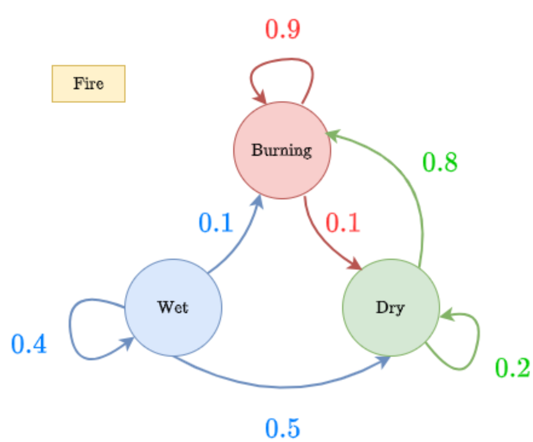
We'll practice on our usual blanket example. We label each of our states with an index:

$$\begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} \text{Burning} \\ \text{Dry} \\ \text{Wet} \end{bmatrix} \tag{12.9}$$

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



$$\mathcal{T}(\text{Water}) = \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0 & 0.1 & 0.9 \\ 0 & 0 & 1.0 \end{bmatrix} \tag{12.10}$$



$$\mathcal{T}(\text{Fire}) = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0.8 & 0.2 & 0 \\ 0.1 & 0.5 & 0.4 \end{bmatrix} \tag{12.11}$$

12.1.7 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.

$$R\left(s, a\right) = \begin{cases} 10 & s = \text{Dry} \\ 0 & s = \text{Wet} \\ -20 & s = \text{Burning} \end{cases} \tag{12.12}$$

Concept 20

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

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

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.

12.1.8 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**.

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

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

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

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

- $R\left(s, a\right)$ 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.

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

- A reward t timesteps in the future, is worth γ^t times as much.
- The higher γ is, the more we value future rewards.