Module 5: Introduction to Machine Learning; Reinforcement Learning

This is a single, concatenated file, suitable for printing or saving as a PDF for offline viewing. Please note that some animations or images may not work.

Module 5 Study Guide and Deliverables

Module Theme:	Introduction to Machine Learning; Reinforcement Learning
Readings:	• Module 5 online content • Russell & Norvig Chapter 21 (Deep Learning), concentrate on Section 21.1 • Russell & Norvig Chapter 22 (Reinforcement Learning), concentrate on Section 22.1
Assignments:	• Lab 5 due Sunday, October 10 at 6:00 AM ET • Assignment 5, due Wednesday, October 13, at 6:00 AM ET
Live Classrooms:	• Wednesday, October 6, from 8:00 PM to 9:00 PM ET • Thursday, October 7, from 8:00 PM to 9:00 PM ET • Live Office: Wednesday and Thursday after Live Classroom, for as long as there are questions

Introduction to Machine Learning

Learning Objectives

By covering the basics of various techniques, we will compare how they can be applied. We will restrict this to two major learning approaches.

After successfully completing this part of the module, you will be able to do the following:

1. Distinguish between learning and non-learning in AI.
2. Make decision when to apply neural nets.

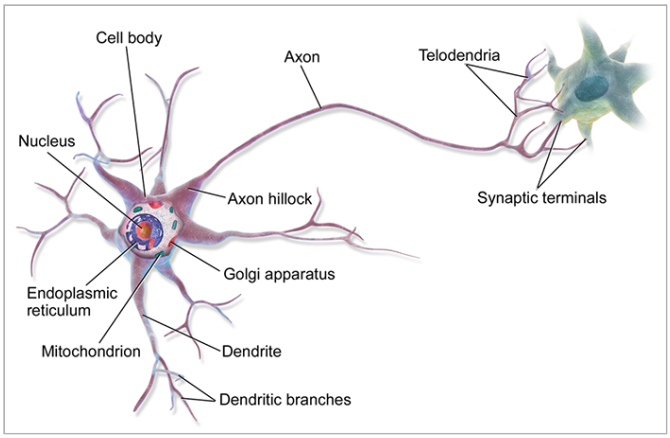
3. Desribe and use “genetic algorithm”.

Introduction to Machine Learning

Neural Nets: What?

Neural nets are based on components of the brain, *neurons*—cells that take input from and provide output to other neurons. A single neuron does not seem to encode knowledge as we understand it; knowledge is encoded by the set of *connections* between neurons.

Figure: Neurons



Source: https://en.wikipedia.org/wiki/Neuron

Application Examples of Neural Nets

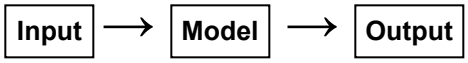
The idea of simulating interconnected neurons has led to many applications, some of which are listed here. The rate and scale of applications have become immense—too long to list, really.

- Check loan applications (Chase)
  - Input: past application/success pairs
  - Result: net usable for new applications
- Recognize handwriting (Apple)
  - Input: sample pairs of script/print
- Detect credit card fraud
- Predict investments (Nikko; Fidelity)
- Appraise real estate
- Predict hospital stays
- Discriminate sounds

- Perform diagnostics
    - Medicine
    - Autos
    - Plant repair
    - …

Neural Nets: Use When…

We typically use neural nets when we don't have a model of how a process actually works. But we must have a set of actual input/outputs. These are mostly from the real world. The input and output of a neural net must each be a vector—an array of numbers.

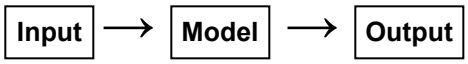


Summary: Use Neural Nets When…

- No model or expertise is known
- Input/output examples are known
- Input and output can be encoded as vectors

Neural Nets: Input

It is actually possible to represent input as an array of numbers, in many cases.

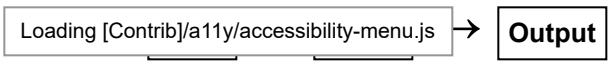


Summary: Neural Nets Input

- Input: Vector of real numbers (or convert to this form)

Neural Nets: Output

Once the architecture of a neural net (i.e., the interconnection pattern and the nature of each neuron's data processing) has been decided, it is fed a set of input/output pairs. These are the *training* set. When successfully trained, the net gives the output for each input to within a tolerable margin of error. It is then ready to be tested and used on any input.



Summary: Neural Nets Output

- Vector of real values
- Giving answers
- Consistent with performance on examples

Neural Nets: Model

Although they are modeled on brains, neural nets are "dumb." Once we set one up, it learns from input/output data by making many numerical adjustments. We don't model how neural nets actually work beyond that: We often can't form much of a vision about why exactly they end up the way they do but the ways to get an idea of this are improving.

An important skill is in how we architect (or "wire," so to speak) each neural-net application in the first place—i.e., prior to training it. This will be explored later in the course, when we focus on neural nets.

Summary: Neural Nets Model

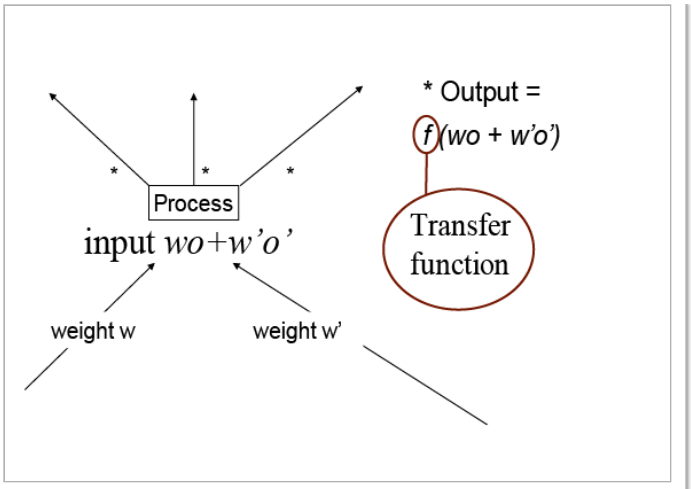
Set up processing elements ("neurons" or "nodes") that behave like human brain neurons, with connections.
Adapt the weightings of the connections to adapt the neural net to the problem.

Modeling Neuronal I/O

In software, we model each connection with a number (the *weight*) that reflects its relative strength.

Each neuron in a neural net takes as input the sum of outputs of other neurons, weighted by the connection strength. It then applies a function (a *transfer* function) to this quantity. The output of this function becomes an input to other neurons (after weighting), or else it is the output to the whole neural net.

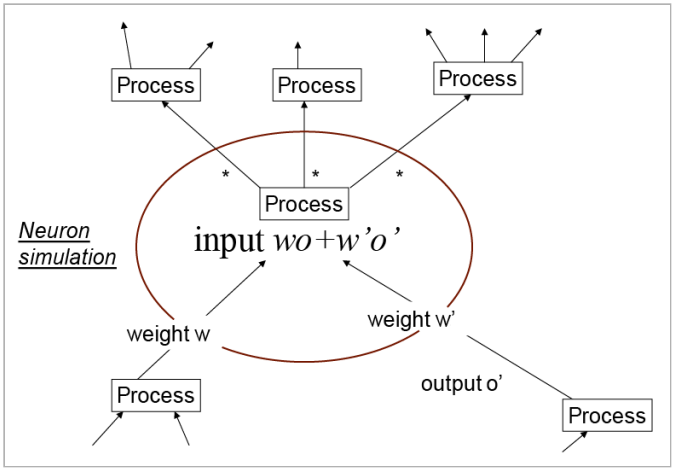
Figure: Modeling Neuronal I/O



This figure below shows how a (simulated) neuron interacts with other neurons.

As with (what we know about) biological neural networks, learning consists mainly of modifying weights.

Figure: Neuron Simulation



TensorFlow Playground Demo

The purpose here is to show how a neural net can recognize whether a point in the unit square belongs to the blue set or the orange set. This is based on a given ("training") set of blue and orange points. The "Playgound" allows you to try various training sets and neural net architectures.

Figure: TensorFlow Playground Demo

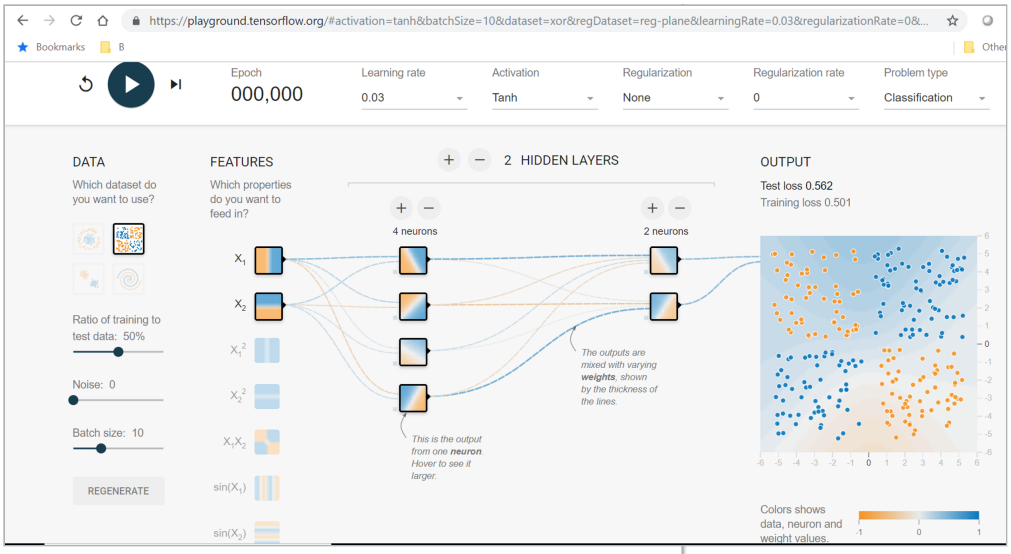
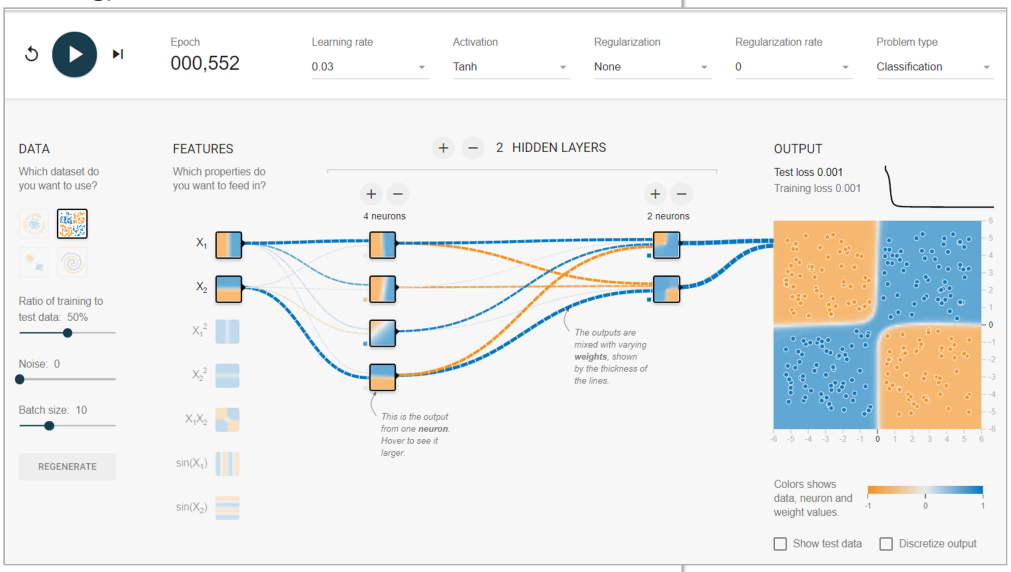


Figure: TensorFlow Playground Demo (After Training)



Check out the TensorFlow Playground Demo.

Deep Learning

A major reason that neural nets have been so successful in the second decade of the 21st century is *deep learning*. This is characterized by multiple layers of neurons and massive amounts of training data. These ingredients were tried for many years without much success; however, persistence, various new techniques, the availability of massive data sets, and increased computation power, which we will describe later in the course (in the neural-net modules), resulted in remarkable success.

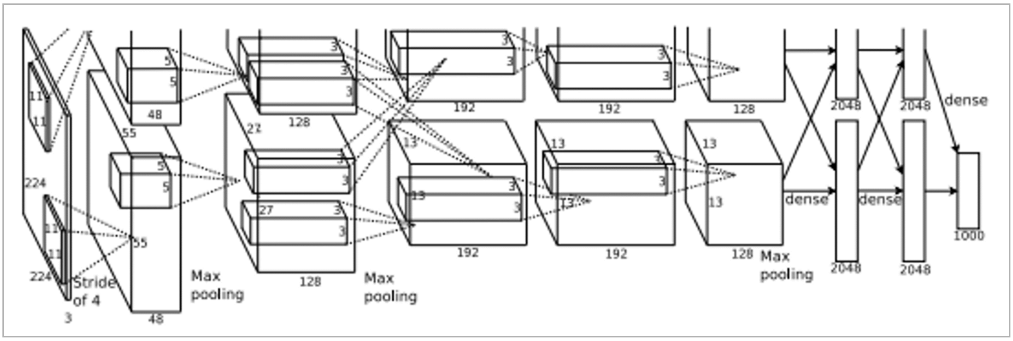
Loading [Contrib]/a11y/accessibility-menu.js

- Neural-net architecture
- Organized in layers
- Each of higher order
    - Example: text summary
        - Sentence syntax
        - Sentence sense
        - Story sense

Architecture (Whole … Parts to Follow)

What brought about the dramatic improvement, and the use of hidden layers are suggested by the figure (from the paper).

Figure: TensorFlow Architecture



Demonstration: MNIST

MNIST—from the National Institute of Stadards and Technology—is a standard example for neural nets. The input is a grid of grey-scale numbers and the output is 0, 1, …, or 9.

Figure: MNIST Demo

Loading [Contrib]/a11y/accessibility-menu.js

Summary

- Machine Learning: applications that learn from data/environment/behavior of agents
- Neural Nets learn from voluminous data, typically input/output.

Reinforcement Learning

Learning Objectives

In reinforcement learning (RL), a random action is taken. If it leads to a favorable outcome, it is strengthened.

After successfully completing this part of the module, you will be able to do the following:

1. Recognize Reinforcement Learning potential.
2. Use basic RL techniques.
3. Apply Monte Carlo decision processes to ML.
4. Compare end-state with discounting.

Reinforcement Learning Definition

Reinforcement learning is often applicable in less-than-certain situations. (If an environment is certain, it is much rarer to be able to learn anything.) There must be a reward of some kind so that actions can be evaluated.

Definition: Context

Learning what to do in an uncertain environment

… with a clear goal

so as to maximize … a (numerical) reward.

Source: Sutton & Barto, p. 3

It is also assumed that the effects and reward of an action can only be determined by actually taking the action.

Definition: Context

Learning what to do in an uncertain environment **(i.e., map situations to actions)** with a clear goal so as to maximize **(possibly delayed)** a (numerical) reward.

Learner must discover what to do by trials (interacting with the environment).

Source: Sutton & Barto, p. 3

What makes reinforcement learning (RL) particularly interesting is that actions may influence, not jus the agent, but the environment itself.

Definition: Context

Learning what to do in an uncertain environment (i.e., map situations to actions) with a clear goal so as to maximize (possibly delayed) a (numerical) reward.

Learner must discover what to do by trials.

$\underbrace{\text{Trade-off of exploitation}}_{\text{What it knows }}$ and $\underbrace{\text{exploration of an agent}}_{\text{What it tries to find out}}$

Source: Sutton & Barto, p. 3

Benefit of Reinforcement Learning

RL has the benefit of operating without a model, somewhat like genetic algorithms, and **is not fundamentally a search technique**.

- No model needed
  - Don't need to understand the whole
- No explicit search performed
  - Not a search method
- Analogous to GA's

Examples of Reinforcement Learning

The following are a few examples. "Playing chess" refers not just generically, but in the context of a particular opponent.

Examples

- Playing chess
- Controlling chemical process
- Simulation
  - E.g., newborn gazelle
- Trash-collecting robot
- Person making a meal

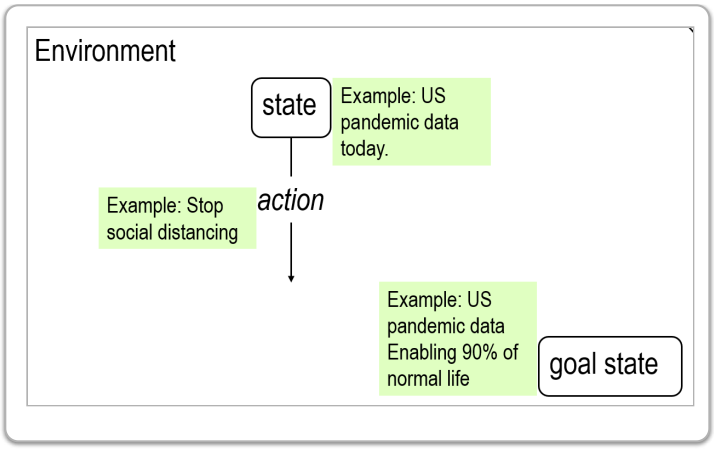
Basic Technique

In this section, we describe the basic reinforcement algorithm.

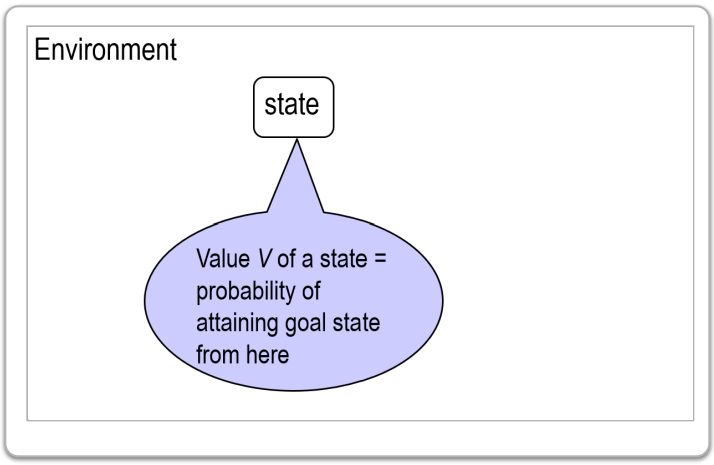
Architecture of Reinforcement Learning

Reinforcement learning takes place in some environment.

State reflects the circumstances under which an action is taken. This is what's strengthened: in other words, we improve then environment in order to make better decisions.

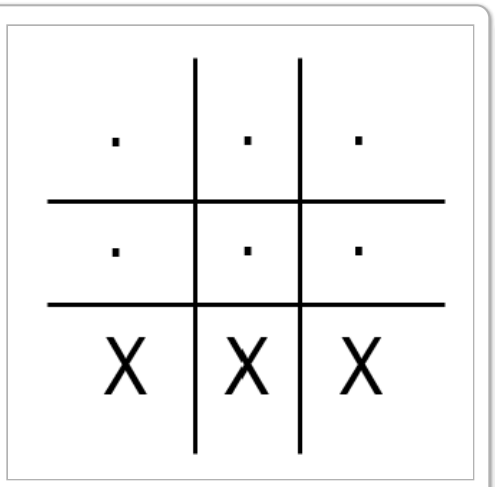


Technically, a state is defined by particular values of particular variables. For example, the state of a `Person` object may be defined as `sick if temperature > 100.4` and `bathing = False`.

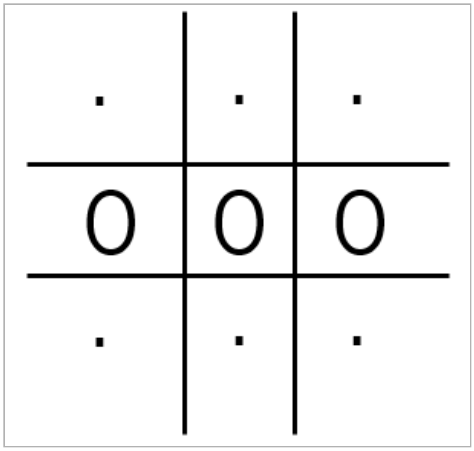


Example: Tic-Tac-Toe

Consider, as an example, playing tic-tac-toe as player X.



Value of such a state: 1

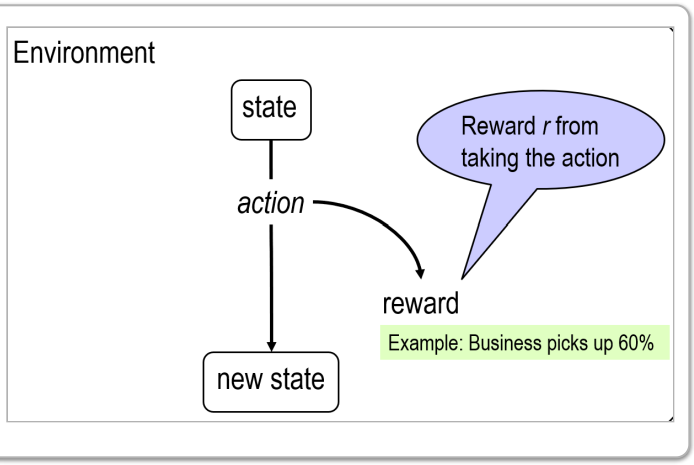


Value of such a state: 0

Immediate and End-Point ***Reward Components***: Reinforcement and reward is either immediate (i.e., visible) or else measured by end-state reward. The latter is more valuable in theory but its estimation is less reliable.

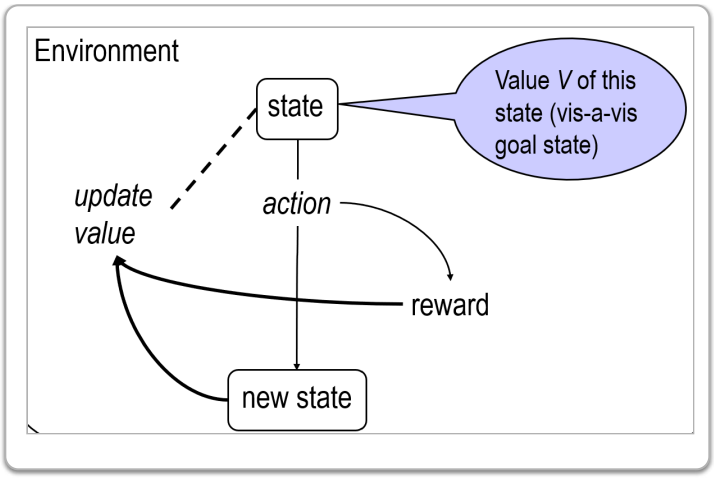
Learn via …

- immediately visible reward
  and / or
- payoff in the end



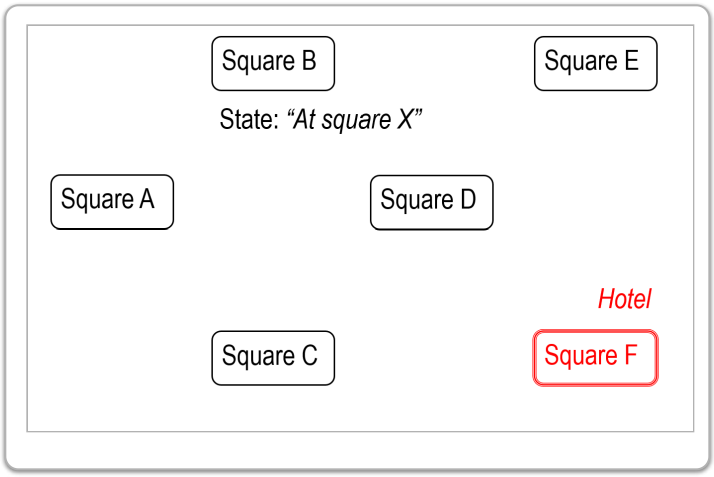
States have ***values***, and when an action is taken with positive reward, the state's value is increased. For example, if your state is "daydreaming" and the action is to wake from your reverie, which results in a dreaming up an improvement in your business (your new state), then you would increase the value of the daydreaming state.

Loading [Contrib]/a11y/accessibility-menu.js



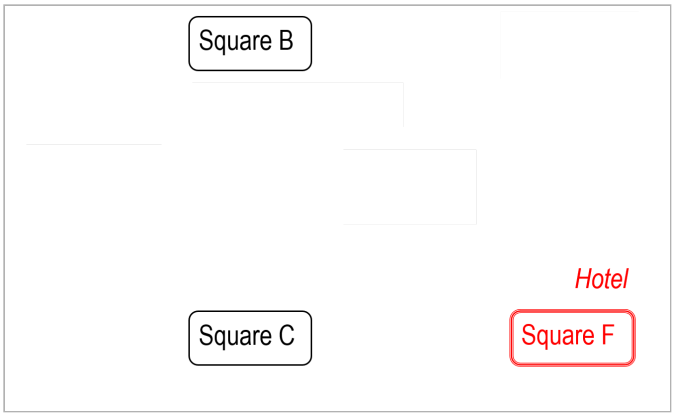
Example: Learning Routes to Hotel in Old City

Marsland takes as an example learning to navigate in an old city. The squares are shown in the figure. It assumes that the squares are well-marked. In this example, a state corresponds to a location. The goal is to learn by reinforcement how to get to our hotel.



It is assumed that we will recognize our hotel (in square F) when we see it.

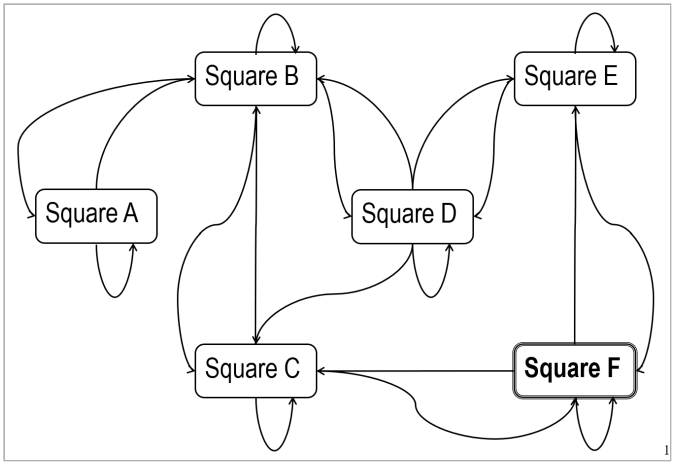
Recall that we learn in RL only via actions, so we begin by being randomly placed, and take random actions, reinforcing beneficial results as we go.



- Start at random
- Will recognize goal when arrive

The “GIven map” figure shows the actual map of connections among squares.

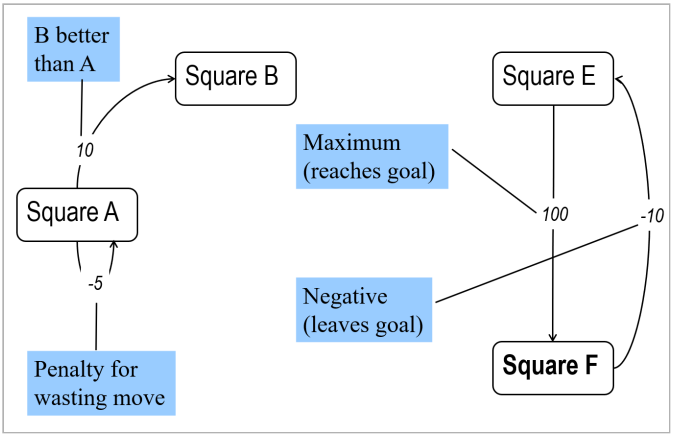
Figure: Given Map



This “Reward of State/Actions” figure shows a selection of rewards:

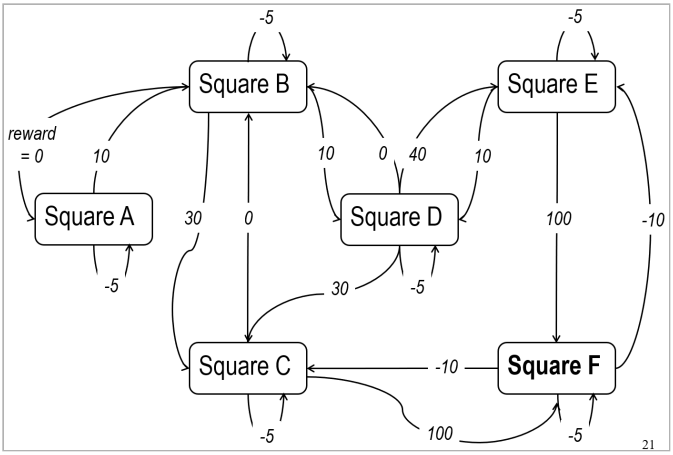
- 100 when a connection yields the hotel,
- -10 when we leave our hotel,
- -5 when we simply start and end at the same place, and
- 10 when we know that our destination is better than our source.

Figure: Reward of State/Actions



This following figure shows what an RL outcome could look like. It would enable us to find a route to our hotel from anywhere. We discuss next how to create this.

Figure: Reward of State/Actions

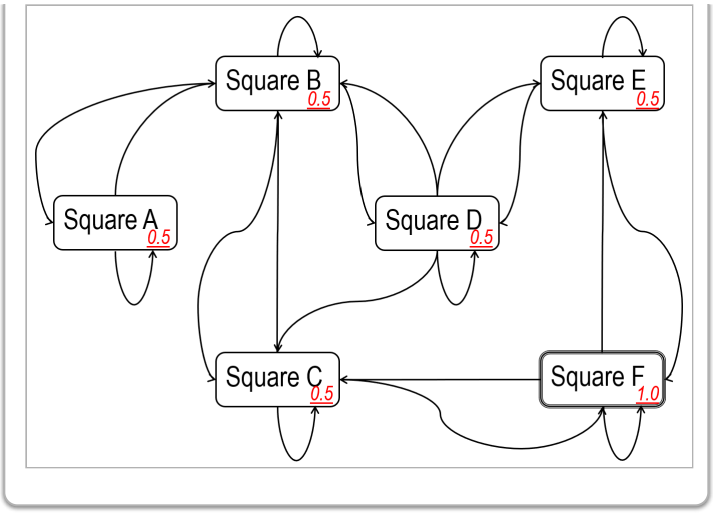


Initialization of Values (Little Knowledge)

Since we have little knowledge, we can begin by giving the goal state the value 1 and all others significantly less—0.5, say.

- Goal State: 1
- Other states: Typically: 0.5

Figure: Initialization



To Move

Now we need to take an action (move, good or bad) in order to make progress. Most of the time, we move rationally i.e., in accordance with the best knowledge (this is “exploitation”) but we know that this needs some shake-up because our knowledge itself is imperfect. As a result, we occasionally make a random move (“exploration”).

- Mostly: Greedily (exploitation)
- Occasionally: Randomly (exploration)

“Policy” (for Selecting Actions) Options

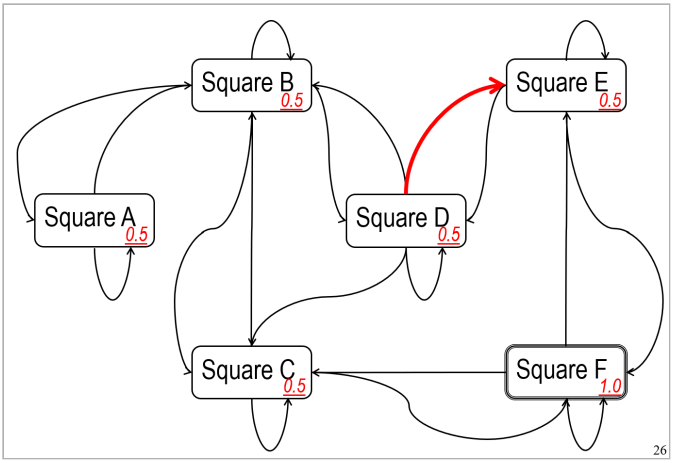
The following list shows three overall methods for carrying this out, including pure exploitation (greedy), \(\varepsilon\)-greedy (a name for the “occasionally greedy” approach we outlined before), to an approach that normalizes the options at a given state. The latter converts the estimated rewards to numbers that sum to 1, using values that include exploitation and exploration, but decreasing the exploration part over time.

- **Greedy:** Highest reward for this state at this time
- **\(\varepsilon\)-Greedy:** Greedy, but with small probability of random other action
- **Softmax:** rescaled probability of highest reward compared with rewards of the other actions
    - Dampen over time

Example Action; e.g., from D

Suppose, for example, that the random start place selected is D. No next-square is better than any other (except back to D) so we pick one at random. Suppose it’s the alley to E.

Figure: Example—Random Start Places from Square D

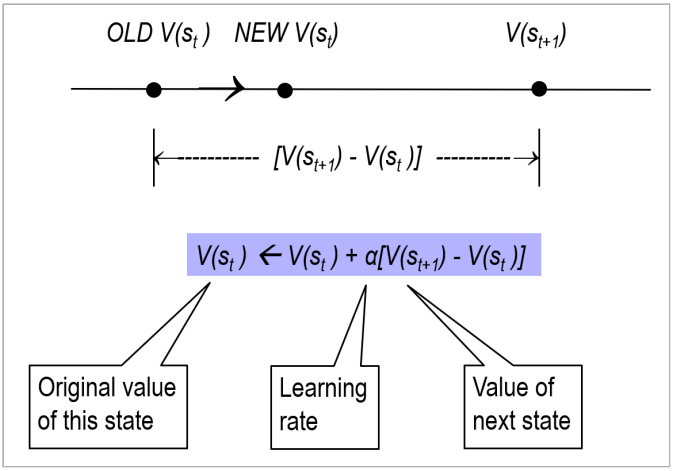


Update Value of a State: Get Closer to Value of Next State

A key question is how to reinforce after taking an action when the answer is not black-and-white. We have already seen the more obvious reinforcements. The figure shows a general way to reinforce a value for state \(i\) which is based on the difference in value between state \(i\) and state \(i+1\). The learning rate is to be determined, but let's take it as 0.2 for now, and suppose that we transition from a state i worth 10 to a state \(i+1\) worth 30. Since the second state is more valuable, state i should be upgraded in value because it is a way to get there. The formula upgrades the value of state \(i\) to \(10+0.2(30-10)=14\).

Figure: Update Value of a State

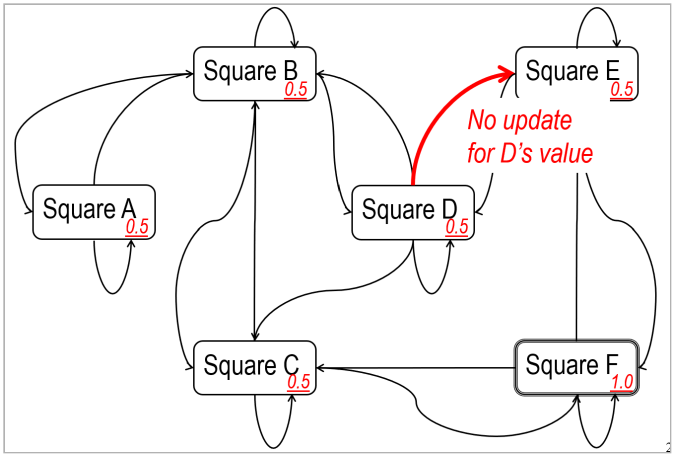
Get closer tovalue of next state.



Move Greedily Most of the Time

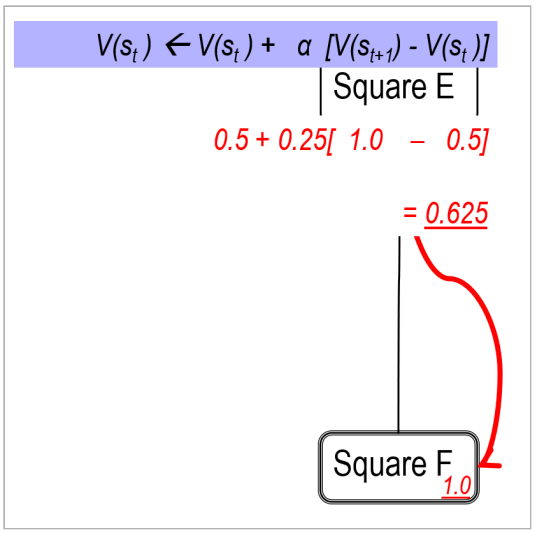
The following figure shows no reinforcement going from a state to one of equal value, e.g., from D.

Figure: Move (e.g., from D) Greedily Most of the Time



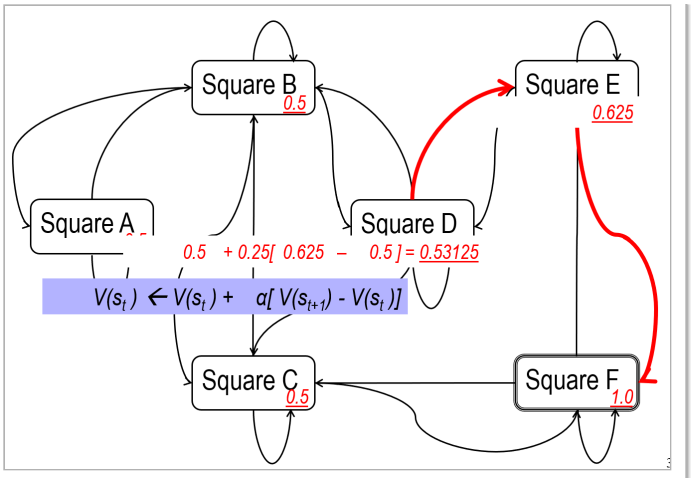
Propagate Backwards (e.g., \(\alpha=0.25\))

The figure below shows an increase in the value of state (square) E.



This change in value propagates backwards through the state diagram, as shown in the figure.

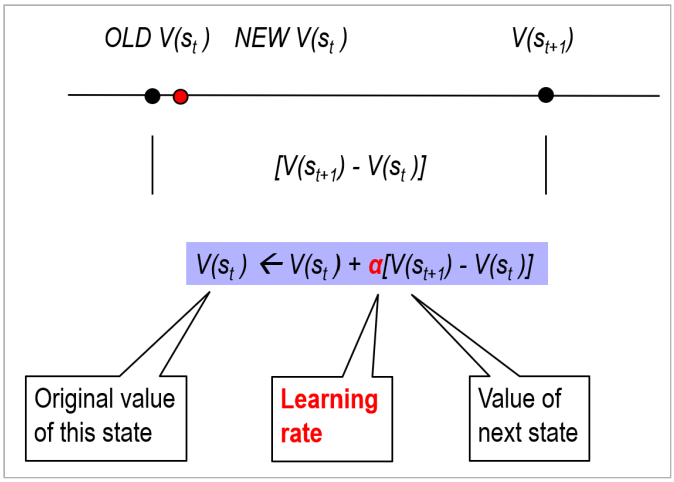
Figure: Move (e.g., from D) Greedily Most of the Time



Reduce \(\alpha\) Over Time

The wild card so far is the value of \(\alpha\). For learning in general, we tend to allow a lot of leeway at the beginning, but reduce it over time. We can do this by, for example, reducing the value of itself by 10% every time it is applied to an updated value.

Figure: Reduce \(\alpha\) Over Time



Finite Monte Carlo Decision Processes

Simplifying Assumption: Markovian Process

Markov processes may sound sophisticated but the opposite is true: they depend only on how matters stand (current state), not on how the state was arrived at.

- (Multiple possible next states are allowed.)
- Probability of next state \(S_{t+1}\) and reward \(R_{t+1}\)
- depend only on current state St and action \(A_t\)
- (i.e., not on how St was arrived at)

Characteristics of Monte Carlo

The advantages of simplifying in this way are that we do not depend on a model, and that the process becomes uncomplicated. As with most simplifications, we learn from experience when what's lost is not too great.

- No model required
- Conceptually simple

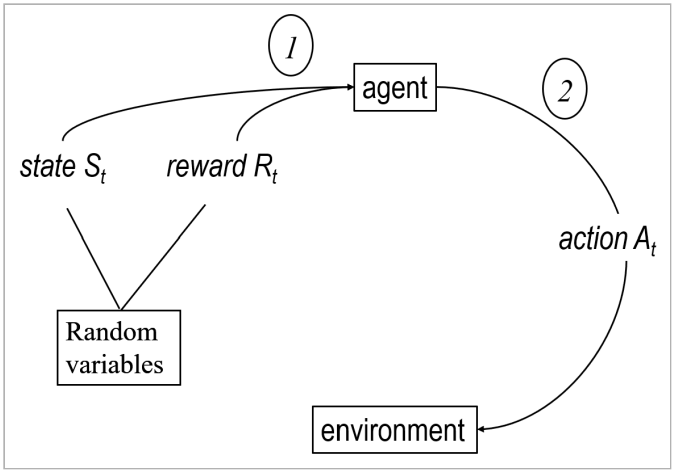
Example of Monte Carlo Process

The reference Markov Chains is a simple Markov process that can be used, for example, (quoting from the site) "to check how frequently a new dam will overflow, which depends on the number of rainy days in a row." It allows us to include real-world rules such as "if it's sunny one day, then the next day is also much more likely to be sunny."

Reinforcement Learning with MDP's

Recall that reinforcement learning takes place in some environment, where state \(S_i\) reflects the circumstances under which the action is taken. This is what's strengthened. The figure shows the state and reward to initially be random since we have no knowledge about them.

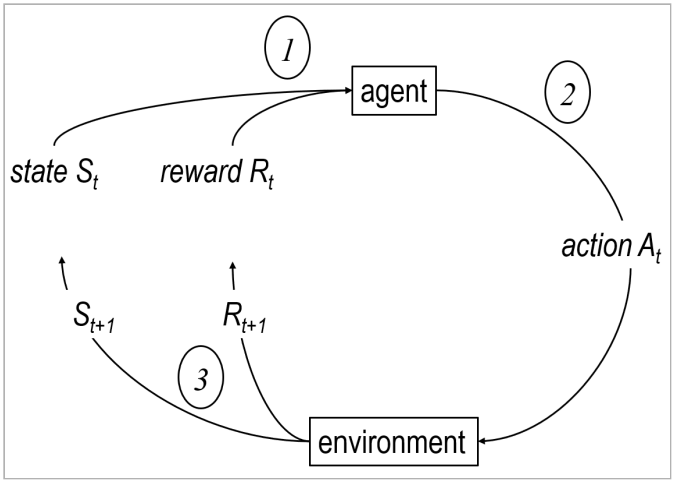
Figure: Reinforcement Learning with MDP's



Loading [Contrib]/a11y/accessibility-menu.js

Once an action is taken in a Markov decision process, the state and reward are updated.

Figure: Reinforcement Learning with MDP's



Source: Barto & Sutton, p. 48

Recycling Robot Implementation

Barto and Sutton describe a recycling robot implemented via reinforcement learning, with the rules follow.

- At each step, robot must decide:
    1. actively search for a can,
    2. wait for someone to bring it a can, or
    3. go to home base and recharge
- Searching is better* but runs down battery
    - runs out of power ⇒ needs rescue!
- Decisions made per current energy level
    - high, low
- Value = # cans collected

* all other things being equal.

The resulting states, actions, and rewards are shown in the following summary.

Set of states

- S={high,low}

actions available when high

- A(high) = {search, wait}
- A(low) = {search, wait, recharge}

- \(R^{search}\) = expected # cans searching
- \(R^{wait}\) = expected # cans while waiting
- \(R^{search}\gt{R^{wait}}\)

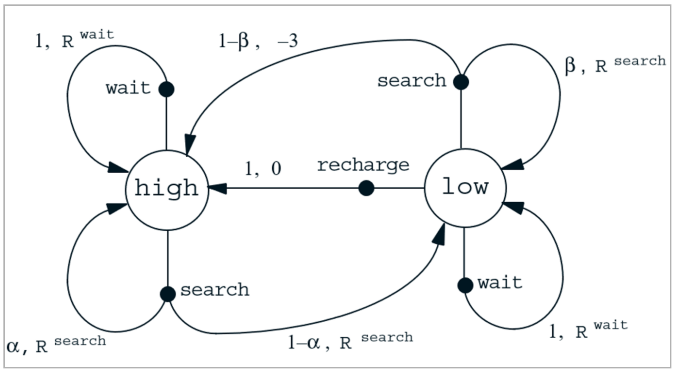
Probability Assumptions

We can parameterize the probability of energy levels after searching for a can. This depends on whether the robot begins searching when its energy is high (which leaves energy either high or low afterwards) or begins when its energy is low (a high-risk action, which results in low or depleted).

1. Begin searching, at high energy p(high energy afterwards) = \(\alpha\) (vs. low)
2. Begin searching, at low energy p(low energy afterwards) = \(\beta\) (vs. depleted)

Transition Probabilities

The figure concentrates on the robot's energy level (the key factor). "Depleted" is not shown because it's a dead end at which the robot is incapable of any action. Assuming that the robot's maintainer has set all of the parameter values, this MDP state diagram (or its tabular form) contains all of the information that the robot needs to operate. For example, when the robot starts, it is in high state, with a choice of searching for a can or else waiting for one to be deposited with it. The choice is made by comparing \(R^{wait}\) with \(R^{search}\). Given that \(R^{wait}\lt{R^{search}}\)}, the robot will search for a standard time. At the conclusion, its energy will be high again or else low. The probabilities are used for decision-making and for simulation. Simulation mode allows maintainers to set the reward quantities. Not shown is data collection for the number of cans found, which also influences the value of the parameters.



Source: Barto & Sutton

Discounting, Example, References

Learning with MDP's

In deciding on actions, there are two cases. If the concept of the "last step" makes sense, it is possible to compute the overall rewards for various actions, and to select the one with highest value. In many cases, it is not feasible to think all the way to a final step. This is closer to the real world. In the latter case, a dampening technique known as ***discounting*** is used.

IF "FINAL STEP" MAKES SENSE*:
    Add rewards over time episodes
IF NOT, APPLY DISCOUNTING.
    * (for the application in question)

Discounting

To calculate long-term reward, we can use a discount factor. For example, if I am trying to evaluate the eventual payoff of buying a restaurant, I can map out a sequence of what will follow but I discount each of the steps to reflect my uncertainty about whether they will come to pass. If my time frame is in months, I may calculate the payoff in terms of …

```
Buy restaurant……..$10K profit first month then …
Remodel……$5K additional profit second month then …
Modernize menu……$3K additional profit third month
```

But we don't simply add these ($18K) because the further out in time an even, the less certain we can be about it. So we apply a discount (let's say 20%), obtaining:

\($10K + 20%\times{$5K} + 20%\times{20%}\times{$3K} (=$11,120)\). This reflects the increasing uncertainty of the \($2K\) and \($3K\).

Discounted reward at time \(t =\underbrace{r_{t+1}}_{\text{Reward from next action}}+\dots\)

Discounting: Factoring Subsequent Rewards

Discount factor \(0\le{Y}\lt1\)

Discount reward at time \(t =r_{t+1}+\underbrace{Yr_{t+2}}_{\text{Reward from action following}}+Y^2r_{t+3}\dots\)

Guaranteed to converge

Applications

Reinforcement learning can be applied in many ways. The application referenced uses RL to train a robot to lift diverse objects. It begins with rudimentary gripping skills and computes rewards via crowdsourcing every time it takes and action.

Optional video: Robot Reinforcement Learning using Crowdsourced Rewards (Note: there is no audio in this clip.)

TensorFlow RL Libraries

Tensorflow has RL libraries, including these referenced.

- API – Reinforcement Learning
- Reinforcement Learning

Summary

RL is based on **rewards** and **actions**. Learning takes place when an action is taken. In realist application, we often use M**arkov Decision Processes**, an approach that simplifies by ignoring how the system arrives at a state. Another heuristic is to apply is the **discounting** of outcomes into the future as a way of dealing with the impracticability, in the real world, of seeing all chains of actions through to final conclusions.

- Reinforcement learning encourages rewards
- Define states and actions
- Monte Carlo RL applies probability
- Use discounting when there is no clear ending