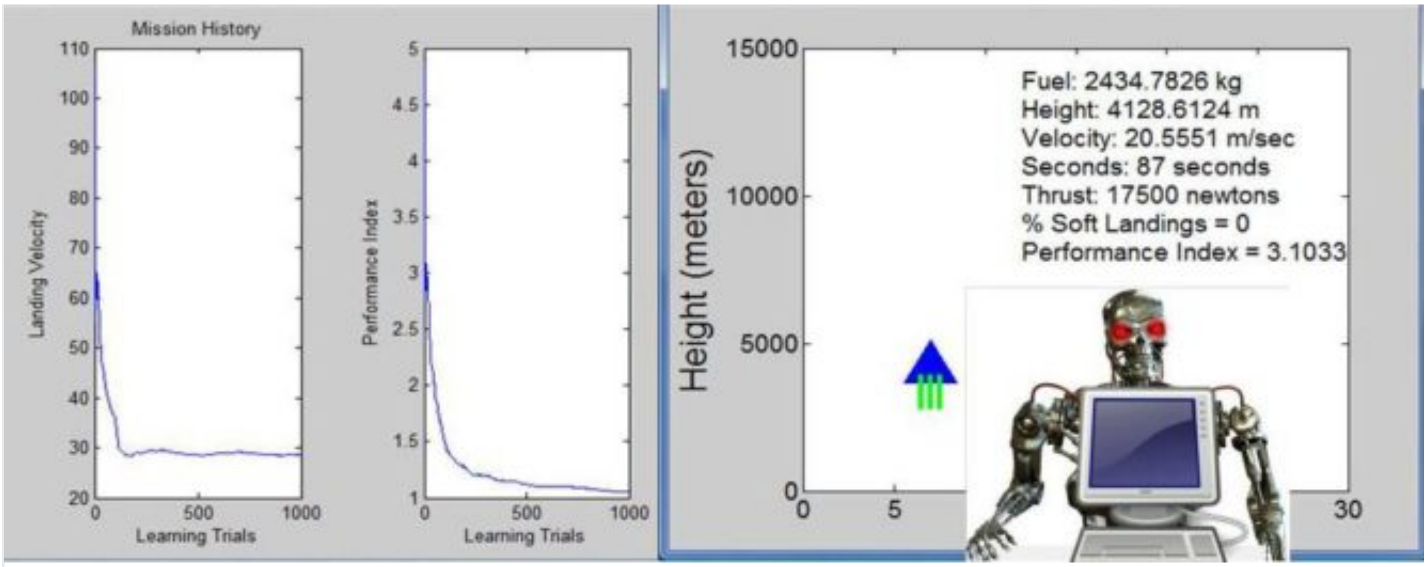
Learning Machines 101

A Gentle Introduction to Artificial Intelligence and Machine Learning

LM101-049: How to Experiment with Lunar Lander Software

By Richard Golden | April 21, 2016 0 Comment



00:00 00:00

Podcast: Play in new window | Download | Embed

LM101-049: How to Experiment with Lunar Lander Software

Episode Summary:

In this episode we continue the discussion of learning when the actions of the learning machine can alter the characteristics of the learning machine's statistical environment. We describe how to download free lunar lander software so you can experiment with an autopilot for a lunar lander module that learns from its experiences and describe the results of some simulation studies.

Show Notes:

Hello everyone! Welcome to the forty-ninth podcast in the podcast series *Learning Machines 101*. In this series of podcasts my goal is to discuss important concepts of artificial intelligence and machine learning in hopefully an entertaining and educational manner.

In this episode we consider the problem of learning when the actions of the learning machine can alter the characteristics of the learning machine's statistical environment. We illustrate the solution to this problem by describing a simple lunar lander software program which simulates a lunar lander landing on the moon. Next, we report some simple experiments with the lunar lander software which are mainly designed to illustrate how it works rather than characterize its performance in detail.

But first, let’s begin with a quick review of the lunar lander problem which was discussed in detail in Episode 25. In a nutshell, the lunar lander problem is a one-dimensional problem where the lunar lander simply can go up or down. The lunar lander begins its descent under the influence of the gravitational field of the moon which generates an accelerative force which attempts to pull the spacecraft downwards. In addition, the lunar lander has an initial downwards velocity which is randomly chosen to make the problem a little more interesting. The lunar lander also contains a certain amount of fuel. You can burn the fuel and this generates an upwards accelerative force which counteracts the downwards accelerative force due to the gravitation pull of the moon. Notice that the amount of fuel required for the mission is of the same magnitude as the mass of the spacecraft. This means that the amount of upwards force generated by the lunar lander throttle control will vary as the spacecraft mass changes due to fuel burning. So this generates some interesting nonlinear behavior of the spacecraft as the mass of the spacecraft changes. The amount of fuel burned is adjusted by increasing or decreasing the thrust at each instant in time. If no pressure is applied to the throttle then fuel is burned at the current rate. If pressure is increased, fuel is burned faster and the accelerative upwards force is slightly increased. If pressure is decreased, fuel is burned slower and the accelerative upwards force is slightly decreased.

So that’s the physics of the lunar lander situation. At the end of a lunar lander mission there are three general qualitative outcomes. First, if you are a good pilot, then you will land safely on the moon. A safe landing is defined as reaching the lunar surface with a sufficiently low velocity. A “crash landing” is defined as not a safe landing. The third possibility is that one applies to much thrust and ends up accelerating away from the lunar surface to be lost forever in outer space.

The lunar lander autopilot learns to land the spacecraft from its previous landing experiences. However, these previous landing experiences are not supervised learning experiences where an instructor teaches the lunar lander at what moments in time thrust should be applied or the instructor rewards or punishes the lunar lander. These previous landing experiences are also not unsupervised learning experiences in which the environment provides no feedback to the lunar lander regarding whether or not it did a good job. Instead, the lunar lander learns through a special temporal reinforcement learning algorithm as described in Episode 2, 44, and 48 of Learning Machines 101.

The lunar lander lands the spacecraft without any knowledge of whether or not it is doing the right thing. The feedback signal it receives at the lunar surface is the magnitude of the spacecraft’s velocity. During flight, the only feedback signals it receives indicate whether or not the space craft’s descent is bumpy or smooth or too fast. Somehow, the lunar lander must use this information to improve its performance in adjusting its controls so that it is more successful in future landings. In situations where the lunar lander is drifting away from the lunar surface, then velocity of the lander at the time the lander is declared to be traveling off course is used as the reinforcement signal.

The lunar lander problem is a nice example of a kind of learning experience which is quite common to us all. In many real world situations, we must make a series of decisions over time without always fully understanding the ultimate consequences of our actions. Eventually, we receive feedback in the form of some reinforcement signal from the environment. Somehow we must use that reinforcement signal to modify our behavior so that in future situations, when we are faced with a similar series of decisions we are likely to make a sequence of decisions which will lead to better outcomes.

In addition, there is another important feature about the lunar lander problem. *The actions of the lunar lander alter the types of experiences encountered by the lunar lander.* In other words, the lunar lander is not a passive observer of its environment. One way to understand this idea is that suppose we wanted to teach someone who had never piloted a lunar lander to land a lunar lander under adverse conditions. The learning problem might be insurmountable for that person if they were simply placed in the cockpit without instruction. They would probably make decisions which would not lead to good outcomes and learning probably would never take place. On the other hand, if the novice was watching an expert land the lunar lander under adverse conditions, they might be able to learn more effectively since the expert's actions result in a set of situations which eventually lead to good outcomes and these situations could then be experienced in person by the novice.

So how does the learning algorithm of the lunar lander actually work? The first idea is that the lunar lander generates an action from the current state of the environment and its current learning parameter values at time t. This causes the environment to change. Then the lunar lunder generates a second action from the new state of the environment and the current parameter values at time t. Thus, we now have the first state of the environment, the first action generated in response to the first state using the parameter values at time t, the second state of the environment generated by the environment in response to the first action, and the second action generated by the learning machine using the parameter values at time t. The environment then generates a punishment reinforcement signal which is basically going to be the current velocity magnitude of the lunar lander plus the magnitude of the lunar lander's change in height and change in velocity between the two environmental states. This punishment reinforcement signal is weighted by a parameter called LAMBDA. Furthermore, if the lunar lander has impacted the lunar surface or deviated from its flight trajectory, an additional penalty equal to the magnitude of the lunar lander's velocity is added to the punishment signal. And, finally, an additional penalty for crash landing onto the lunar surface is incurred as well. The lunar lander's goal is to minimize expected punishment signal by adjusting the parameters of its control law.

Let's go over some of these ideas a little more slowly. Let's first consider the simpler special situation where at the next instant in time the lunar lander reaches the lunar surface at a particular velocity. Thus, because this is the end of a particular lunar landing trajectory this particular moment in time has the characteristics of unsupervised learning. Specifically, we can think of the velocity at the next instant in time as a performance error measure and we can adjust the parameters or weights of the lunar lander control law using gradient descent to minimize this performance error. So this is one aspect of the learning algorithm. Another aspect of the learning algorithm is when the lunar lander is in the process of landing we use another type of performance error which corresponds to how the lunar lander's height and speed changes from one instant in time to the next. In this case we introduce a "smoothness constraint" or "regularization term" into the objective function which penalizes large changes in the lunar lander's height and speed from one time instant to another. The regularization term also penalizes the lunar lander for a fast descent. That is, the objective function penalizes the lunar lander for adjusting its controls for a "fast bumpy ride". Also it again should be emphasized that the reinforcement learning signal indicating the velocity upon impact on the lunar surface is only generated once at the end of the lunar landing trajectory. The error signal characterizing the speed and smoothness of the landing can be used throughout the trajectory to adjust the parameters of the lunar lander control law. Note that the the reinforcement learning signal indicating whether or not the lander crashes is only available at the end of the lunar lander trajectory.

An additional constraint upon the learning process is the weight decay parameter which introduces a penalty into the objective function which is equal to the sum of the squares of the parameter value magnitudes. This regularization term has the effect of stabilizing the learning process and preventing overfitting. It is called an L2 regularization term in the machine learning literature.

It is important to emphasize the presence of what appears at first glance as a circular reasoning argument. We are using the lunar lander's current parameter values to generate actions and then we are using the resulting actions and their environmental consequences to improve the lunar lander's parameter values. That is, we are trying to estimate how to modify the landing control parameter values based upon our current best guesses (which might be wrong) about the current parameter values. Fortunately, it turns out that this reasoning is mathematically legitimate in that sense that we can mathematically prove that this method under specific conditions will cause the parameter values to converge with probability one to values which optimize the desired performance criteria.

Specifically, this method of learning is called Stochastic Approximation Expectation Maximization and was mentioned briefly in the context of constraint satisfaction learning in Episode 43. A reference to one of my recent mathematical analyses of Stochastic Approximation Expectation Maximization adaptive temporal reinforcement learning can be found in the show notes of this episode of Learning Machines

101. A technical note describing the mathematical details of this temporal reinforcement learning algorithm can also be found in the show notes of this episode of Learning Machines 101.

Let's now experiment with the lunar lander software program and see how it actually works. You can download a copy of the software program off the website. . Once you download the software associated with this episode you can explore and evaluate the effectiveness of the lunar lander software program. The software is written in the computer programming language MATLAB which is sold by the MATHWORKS (www.mathworks.com). If you have MATLAB installed on your computer, then you can run the software by downloading the source code from this website by running the program "demolunarlander.m" which is located inside the folder "LunarLander". If you do not have MATLAB installed on your computer, then you can still run the software on a WINDOWS computer, but you will need to install the MATLAB function libraries on your computer using a computer program called the MCR Installer. If you have already successfully installed these libraries on your computer to run the software associated with Episode 13, then **you <u>do not</u> have to reinstall these libraries**. You can just download the executable software directly. Unfortunately, a MAC OS-X version of the Lunar Lander is not currently available.

Here is a brief overview of the software installation procedure which is described in great detail in Episode 13. First, obtain the software download password by visiting the website: www.learningmachines101.com. If you are not a member of the Learning Machines 101 community, sign up and make sure to fill in a few words about your interests. This will help me plan future episodes of Learning Machines 101. If you are already a member, then the secret password can be found in the emails which we send out twice a month.

Second, if you have not already done so, install the MATLAB Compiler Runtime Library. If you have already done this installation on your computer as described in Episode 13, then you can skip this step. Third, download the Windows version of the free software provided under the Apache 2.0 license. The software is downloaded as a ZIP file which you unzip using the software program WINZIP. Make sure you "unzip" this file so that it becomes a standard folder BEFORE you try running the software!

When you startup the program you will see a pop-up screen with the following options:

Initial Fuel: 3500 kg,
Initial Height: 15,000 meters,
Initial Velocity: 100 meters/second
Number of Landings: 1000
Lambda: 0.4

Extra Penalty for Non-Safe Landing: 1
Weight Decay: 0.0001
Momentum: 0.4

Let me briefly explain each of these options.

Initial Fuel is the amount of fuel initially in the lander. Each time you apply the thrusters you burn fuel. Also fuel weights quite a bit so the loss of fuel can actually have a highly nonlinear influence on the downward accelerative force due to the moon's gravitational pull.

Initial Height is the initial height above the lunar surface where you start in meters. The actual initial height is randomly perturbed each time you start a landing to make characteristics of the lunar lander's environment less predictable.

Initial Velocity is the initial downwards velocity in meters per second. It is also randomly perturbed each time you start a landing to again make things a less less predictable.

Number of Landings is the total number of lunar landings. The default is 1000 lunar landings.

Lambda specifies the magnitude of the regularization constant. When Lambda = 0 then the constraint that the lunar lander learning machine is searching for a "slow and smooth landing" is eliminated. As lambda increases, the constraint that the landing must be "slow and smooth" is enforced at the expense of compromising the lunar lander's performance.

Extra Penalty for Non Safe Landing specifies the extra penalty which is received at the end of each lunar landing attempt when the lunar lander crashes or moves along an escape velocity trajectory. This extra penalty is added on to the magnitude of the velocity at impact to obtain the total penalty at the end of a landing run. If the lunar lander starts accelerating away from the lunar surface at a designated escape velocity, then the magnitude of the escape velocity is used as a penalty and the simulation of the lunar landing is stopped.

Weight Decay specifies the magnitude of a regularization constraint which tries to make all of the parameters of the learning machine as small as possible. This constraint essentially forces the learning machine to only learn important statistical regularities. When "Weight Decay" is zero this turns off the regularization constraint. Another way to think about this constraint is that larger values of the weight decay parameter tend to make the learning machine ignore weaker statistical regularities. If the weight decay parameter is too large, the learning machine won't learn anything.

Momentum specifies a number which is used to compute how the parameters of the learning machine are changed at a particular instant in time. Specifically, the gradient descent direction is computed as described in Episode 31 of Learning Machines 101. Then the new parameter values are equal to the old parameters values multiplied by the learning rate multiplied by the desired parameter value change. The desired parameter value change is approximately the gradient descent search direction plus momentum multiplied by the previous parameter value change. Intuitively, when momentum is set equal to zero then the search direction is exactly a stochastic gradient descent search direction. When momentum is set to a larger number, the search direction is a weighted sum of the previous search direction and the current gradient. Hence, the terminology "momentum"…you tend to keep going in the direction you were traveling before when the gradient direction is not providing useful information.

For now, just hit the **"ok"** button and it will accept the default values for these parameters.

When you start up the program you will see four distinct options: **"manual"**, **"autopilot"**, **"autopilot-nofuel"**, and **"autopilot-zero-learningrate"**.

The option **"manual"** lets you try landing the lunar lander on your own….

When you choose this option you will see two choices which appear in the command window of MATLAB or in a windows operating system command window:

- Decrease Thrust by 33% and   9) Increase Thrust by 33%.

This means that if you hit the "1" key on the keyboard that will decrease the thrust by 33% and if you hit the "9" key on the keyboard that will increase the thrust by 33%. If you decrease the thrust too much, then this has the effect of turning off all thrusters. If you increase the thrust too much, this has the effect of applying maximum upwards thrust to the spacecraft. As you hit these keys, you will see a cartoon image of the lunar lander lander Landing. If more or less thrust is applied, then the fuel jets become longer or shorter respectively. If the lunar lander impacts the moon at a speed which is greater than some critical threshold, then this is defined as a crash; otherwise the landing is defined as safe.

If you do decide to download the software, I encourage you to try out the program and get a feeling for the difficulty of the problem of landing the lunar lander.

For the next option, "autopilot-nofuel" corresponds to the case where the lunar lander has no fuel. The purpose of this option is to illustrate that the lander will crash onto the surface if thrusters are not applied. For the parameter settings provided here, the lunar lander consistently crashes into the lunar surface with an impact velocity of approximately 230 or 250 meters per second. Hit the control-c key to halt the simulation once you get bored of seeing the lunar lander blow itself up multiple times.

The next option “autopilot-nolearn-randomcontrol” corresponds to the case where the lunar lander has fuel but learning is not enabled. This means that the lunar lander is essentially controlled by the initial random guess for its parameter values. The purpose of this option is to illustrate that the lander will crash onto the surface if the thrusters are simply randomly applied. The lunar lander for the most part will crash onto the lunar surface or float away from the lunar surface. It sometimes does land successfully. Spend at least several minutes watching the lunar lander’s behavior with a random control law to get a feeling of the performance of the random control law. The random control law typically causes the lunar lander to hit the lunar surface at about 50 meters/second. A soft landing is defined as a landing which hits the lunar surface at less than 30 meters/second.

Ok…do now we are ready to turn on the automated lander option which is called “autopilot”.

At first, it crashes into the lunar surface with a landing velocity of 123 meters/second. But almost immediately, it generates thrust and on the second simulation run lands at 80 meters/second. After 5 lunar landers, it successfully manages to land with out crashing on mission number 6 but then it crashes on missions 7, 8, 9, 10, 11, 12, and 13. A complete simulation run is defined as 1000 lunar landings. The simulation runs take a long time to complete because I have inserted pause statements into the code so that the graphics can be displayed and interrupted. So you can speed up the learning process by getting rid of the pause statements! Also the code has not been optimized for speed. With these comments, it takes at least several hours for a simulation run to complete. So I did the following experiment. Using exactly the same initial parameters for the simulation study, I repeated the simulation experiment six times.

At the end of the 1000 lunar landings a figure window pops up which displays the simulation results.

The left-most graph displays the landing velocity which typically starts at about 90 meters/second and drops to a safe landing velocity of approximately 30 meters per second within the first 100 learning trials for all six simulation runs. The second graph from the left is the performance index. Recall that the lunar lander is a operating in a statistical environment and is updating its parameters every two time steps. So it actually has obtained a number of learning experiences before it hits the ground on its first lunar landing attempt! The performance index was previously described and you can see that it decreasing consistently on the average. Sometime the performance index does not decrease and increases a little, this is not surprising and does not indicate poor optimization algorithm design. The mathematics describing the lunar lander only guarantees that the performance index will decrease on the average. The third graph from the left specifies the % of safe landings. Even though the lunar lander rapidly learns to obtain approximately a safe landing velocity within the first 100 learning trials, typically the percentage of safe landings after about 100 learning trials is only about 10% or 40%. I observed in all six simulation runs that the percentage of safe landings consistently increased to about 55% or 60%.

So let's discuss the strengths and limitations of the approach we have introduced. First, the algorithm is extremely simple yet powerful. The lunar lander control law learns to land the lunar lander using only the following types of knowledge: (1) the magnitude of velocity of the lunar lander should be minimized, (2) the change in the velocity and height of the lunar lander from one time instant to another should be minimized, (3) the magnitude of the parameter values should be minimized, and (4) the penalty incurred at the end of lunar landing attempt such as when the lunar lander hits the lunar surface or moves too rapidly away from the lunar surface is defined by the magnitude of the velocity of the lunar lander at that endpoint in time. The lunar lander is never provided with explicit instructions regarding how it should operate. Because the statistical environment is random, the lunar lander must generalize from its past lunar landing experiences and is constantly dealing with situations which are slightly or very different from its past experiences. As a result, the adaptive control law is able to successfully land the lunar lander approximately 60% of the time. Moreover, its behavior appears to be intentional. For example, some solutions have the lander decrease rapidly to a certain point and then the lunar lander's velocity is carefully controlled to be a small value to ensure a smooth and gentle landing.

On the other hand, there are a number of limitations with the proposed approach. In fact, this episode was originally planned to be scheduled approximately a year ago but I wanted to see if I could boost the performance of the lunar lander so I held off. Today, however, I am more comfortable with my understanding of the strengths and limitations of the lunar lander technology.

So why does the lunar lander only land safely about 60% of the time? Why couldn't we get performance boosted to 100% of the time? I'm still in the process of trying to understand this but let me share my current thoughts. First, the current lunar lander uses "hand coded feature representations". So, I do not represent the current state of the lunar lander as a triplet of numbers consisting of the lunar lander's height, velocity, and fuel level. Instead, this current state is represented as a nonlinear transformation of these numbers which I chose based upon my engineering knowledge and experience. I only experimented with 2 or 3 choices of nonlinear transformations in this initial investigation. My ultimate plan is to have the lunar lander use deep learning methods as described in Episodes 23 and 41 of Learning Machines 101 so that it can learn its own internal feature representations which will probably be much better than my initial choices.

Second, the problem may be an intrinsically hard problem. The criteria for landing safely is to hit the ground at 30 meters/second. In the real world, if the lunar lander hit the ground at 30 meters/second it is doubtful that there would be any survivors. Furthermore, I made the problem more difficult by having the lunar lander propelled towards the lunar surface at an initial velocity of 100 meters/second. Similarly, I can make the lunar lander problem easier or more difficult by varying the engine efficiency, the criteria for landing as just discussed, the random variation in the initial velocity and initial height, the magnitude of the maximum rocket thrust, the precision of the control of the rocket thrust, and many other factors. This is, in part, why this episode was delayed because every time I would make some progress in getting the lunar lander to land in a challenging situation I immediately made the problem harder. This, in turn, resulted in further attempts to reach the performance levels I had obtained in earlier versions of the system. This is a fundamental characteristic of research in artificial intelligence and machine learning. Each time you make progress in solving a hard problem, there is a strong tendency to make the problem harder and this, in turn, results in a reduction of performance.

Third, the objective function for the specific lunar lander temporal reinforcement learning algorithm could be improved. This analysis is based upon the assumption that the episodes are independent and identically distributed. Recall that an episode is defined as the quintuple (initial state, initial action, final state, final action, reinforcement signal). Because it is an adaptive algorithm, we can interpret its behavior as learning a sequence of statistical environments where it uses its current guess for the parameter estimates for the next statistical environment given the previous statistical environment. Also the episodes are based upon predicting the reinforcement signal received for a particular episode rather than the total expected reinforcement received into the distant future. The concepts of a "finite event horizon" and "infinite event horizon" for designing objective functions for optimal control and temporal reinforcement learning were discussed in Episode 44 of Learning Machines 101.

As previously noted, I have made the lunar lander software available as Windows Executable or as MATLAB software. You can download the software by going to the website: www.learningmachines101.com and choosing the SOFTWARE tab and then the LUNAR LANDER tab underneath the SOFTWARE tab. Post the results of your experiments on this website or in our LINKED IN group titled "Statistical Machine Learning Forum". You will need the password to download the software. The password can be obtained for free by simply joining the Learning Machines 101 community by going to the website: www.learningmachines101.com . If you are already a member, the password is in the email announcements which you should receive about twice a month.

Also please note that you may have had difficulty with getting other software packages to download from the website in the past, hopefully those problems have since been resolved.

Finally, if you are a member of the Learning Machines 101 community, please update your user profile.

You can update your user profile when you receive the email newsletter by simply clicking on the: *"Let us know what you want to hear"* link!

Or if you are not a member of the Learning Machines 101 community, when you join the community by visiting our website at: www.learningmachines101.com you will have the opportunity to update your user profile at that time.

From time to time, I will review the profiles of members of the Learning Machines 101 community and do my best to talk about topics of interest to the members of this group!

Keywords: Temporal Reinforcement Learning, Unsupervised Learning, Supervised Learning, Adaptive gradient descent, lunar lander.

Further Reading:

1. Early paper covering mathematics of the lunar lander (Golden, 2014). (http://arxiv.org/pdf/1412.5744.pdf)

2. Excerpt from my new (unpublished) book which describes the specific technical details of the lunar lander implemented in the attached software. Requires Password to download.
   Enter Password to Download:

   Submit

3. Good reference for Policy Gradient Methods http://www.scholarpedia.org/article/Policy_gradient_methods

4. An original description of Samuel's Checker playing program written by Samuels in 1959 may be found in:Arthur, Samuel (1959).“Some Studies in Machine Learning Using the Game of Checkers”(PDF). *IBM Journal* **3** (3): 210–229.

This was reprinted recently in the *IBM Journal of Research and Development* (Vol. 44, Issue 1.2) in the January, 2000.

5. The book *The Quest for Artificial Intelligence* by Nils Nilsson (Cambridge University Press) has a nice discussion of Samuel's *Checker playing program* on pages 90-93. Also on pages 415-420 there is a nice discussion of *temporal reinforcement learning*.
6. Niv,Y.(2009).Reinforcement learning in the brain.*Journal of Mathematical Psychology, 53*,139-154.
7. Sutton, R. S., and Barton, A. (1988).*Reinforcement learning: An introduction.* MIT Press.

Tweet

Gradient Descent Learning   Reinforcement Learning

Adaptive gradient descent   lunar lander   supervised learning   temporal reinforcement learning

Unsupervised Learning

← LM101-048: How to Build a Lunar Lander Autopilot Learning Machine (Rerun)

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

☐ Confirm you are NOT a spammer

Submit a Review to ITUNES!

Submit a Review to STITCHER!

Hear Us On STITCHER RADIO ON DEMAND Get It Now ▸

LM101-049: How to Experiment with Lunar Lander Software

LM101-048: How to Build a Lunar Lander Autopilot Learning Machine (Rerun)

LM101-047: How to Build a Support Vector Machine to Classify Patterns (Rerun)

LM101-046: How to Optimize Student Learning using Recurrent Neural Networks (Educational Technology)

LM101-045: How to Build a Deep Learning Machine for Answering Questions about Images

LM101-044: What happened at the Deep Reinforcement Learning Tutorial at the 2015 Neural Information Processing Systems Conference?

LM101-043: How to Learn a Monte Carlo Markov Chain to Solve Constraint Satisfaction Problems (Rerun)

LM101-042: What happened at the Monte Carlo Markov Chain Inference Methods Tutorial at the 2015 Neural Information Processing Systems Conference?

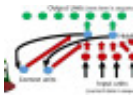
LM101-041: What happened at the 2015 Neural Information Processing Systems Deep Learning Tutorial?

LM101-040: How to Build a Search Engine, Automatically Grade Essays, and Identify Synonyms using Latent Semantic Analysis

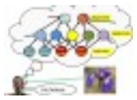
LM101-039: How to Solve Large Complex Constraint Satisfaction Problems (Monte Carlo Markov Chain)[Rerun]

LM101-038: How to Model Knowledge Skill Growth Over Time using Bayesian Nets (Educational Technology)

LM101-037: How to Build a Smart Computerized Adaptive Testing Machine using Item Response Theory


LM101-036: How to Predict the Future from the Distant Past using Recurrent Neural Networks

LM101-035: What is a Neural Network and What is a Hot Dog?


LM101-034: How to Use Nonlinear Machine Learning Software to Make Predictions (Feedforward Perceptrons with Radial Basis Functions)[Rerun]

LM101-033: How to Use Linear Machine Learning Software to Make Predictions (Linear Regression Software)[RERUN]

LM101-032: How To Build a Support Vector Machine to Classify Patterns

LM101-031: How to Analyze and Design Learning Rules using Gradient Descent Methods (RERUN)

LM101-030: How to Improve Deep Learning Performance with Artificial Brain Damage (Dropout and Model Averaging)

Search Blog by Keyword

Search