that inbox or you to the team is on time sorry about the problems that we've been having with equipment and

LECTURE OUTLINE

let me welcome you to this third lecture on this series of six lectures on approximate dynamic programming

欢迎你来听这个六天的动态规划课程的第三天

we spent the first two lectures to to discuss exact dynamic programming

我们用了两天时间讨论精确动态规划

with an emphasis on discounted cost problems which are the easiest ones

我们重点讨论了折扣回报问题，最简单的一个问题

we are going to review these problems to start this lecture

这个课程的开始，我们要回顾一下这个问题

and then we are going to go into we are going to go into approximate dynamic programming

然后开始讲动态规划

today's lecture is the purpose of today's lectures overview I'll give you an overview the bigger picture of the subject

今天的课程目的是给你们一个更全面的综述

and then next week we're going to pick up various parts of this picture and we're going to go into greater depth

接下来我们要讲这个图上的一部分内容，然后我们会讲一些更深入的内容

so today it's going to be sometimes not quite very detailed description

所以今天不会讲非常细节的内容

I just won't want to describe the whole field to you

我只是想向你介绍这个领域的状况

and then I'll point out what the major points are the major areas where one needs to pay attention

然后我会指出这个领域比较重要的，需要你重点关注的问题

what were to discuss our approximation architectures

我们会介绍近似结构

to to introduce approximations you need to have some parametric forms and that's what this approximation architectures are

介绍你需要的一些参数化近似形式和这个近似结构是什么

then I'm going to discuss the politics iteration method which we saw earlier in the previous lectures but with approximations and also using simulation

然后我会介绍我们之前讲过的策略迭代，但是今天会讲使用近似与仿真的策略迭代

and we are going to discuss why do we need simulation why simulation so important in this field

然后我们会讨论我们为什么需要进行仿真与为什么仿真在这个领域如此重要

we're going to focus on methods of approximate policy evaluation which is the first the first part of a policy duration

我们会关注策略迭代的第一部分-策略评估近似方法，

and then we're going to talk about some issues of policy improvement as well

然后我们会讨论一些关于策略改善的话题

and close with a few discussion of a few general issues on approximation methods

最后讲一些关于近似方法的一般性的问题

REVIEW

DISCOUNTED PROBLEMS/BOUNDED COST

so let's go with a few slides slides of review

下面的几个slide用来回顾我们之前讲的内容

we are considering dynamic systems who state is X K at time K we consider the systems over an infinite horizon and now the systems evolve according to an equation a discrete time equation where the new state is obtained from the current state the control that you apply this UK and some random quantity wk

我们在考虑一个动态系统，时间k的状态用x\_k表示，这个系统是一个无限期系统，现在这个系统使用一个离散时间方程表示，新状态通过当前状态，执行的控制u\_k和随即量w\_k一起作用产生

there are some assumptions associated with that we disturbed the VW cake which we call disturbance or noise the probability distribution of that depends only on XK and UK and not on previous stuff the Markov property that that we assume here

现在有一些假设，w\_k被叫做扰动或噪声，它的概率分布只依赖于当前的x\_k和u\_k，与之前的信息无关，这就是我们假设的马尔科夫性

and and we are interested in policies policies are functions of state to control there are rules that tell us at every state what control to apply so once we have this new we know how to do that and that's the objective to obtain such a sequence of Meuse such that the corresponding cost is minimized

我们还需要知道策略，策略就是一些输出控制的状态的函数，这些规则告诉我们在每一个状态时应该使用什么控制，所以如果我们有了一个策略mu，我们就知道怎么控制一个系统，这是目标函数，，需要知道一个策略mu的序列最小化总成本

so if we plug in one of these PI's the sequence of control functions there is a cost associated with each stage at stage K because G depends on XK the control that you apply and the random disturbance so this is a random number the discount factor alpha between 1 and 0 which keeps this thing bounded and let's say future costs here are discounted heavily so that the future doesn't count as much as the present mathematically this is good because it allows us to define this as a real number this causes a real number

如果我们使用一个策略序列对系统进行控制，那么这个系统的总成本就与每一个阶段k有关，g依赖于x\_k，u\_k与随机扰动w\_k，所以成本是一个随机数，这个折扣因子alpha是0到1之间的数我们用这个数限制成本的上界，为来成本被折扣因子限制的很严重，这样就可以保证未来的成本对当前决策的影响比较小。在数学上这是一个很好的东西，因为它让我们正本的定一是一个实数

now each one of these terms here is a random the sum is also random and we take the expected value of that over W and becomes a number and we take the limit of this series as over over an infinite number of stages and that gives us the cost of policy by starting at state x 0

现在每一个阶段成本都是一个随机数，这些成本的和也是随机数，我们求这个总成本在随机变量w下的期望值可以得到一个数，如果阶段数量区域无穷的时候，成本函数计算出的就是策略在系统从状态x0开始的成本

so for one state we have a cost another state another costs a whole function associate with this policy and we want to make this function of cost as low as possible by optimal choice of Pi

在一个阶段我们付出了一个成本，另一个阶段我们付出了另一个成本，那么整个函数就与这个策略相关，我们想要获得这个函数获得最小化的策略pi

ok now with the double some shorthand notation associate the first thing is this mapping that takes functions into functions this appears in the dynamic programming algorithm is very basic

第一个内容是从一个函数到另一个函数的映射，这是动态规划中一个非常基础的概念

so to a given J it provides it will produce another function T sub J

给定J，可以使用算子T让这个函数变成另一个函数TJ

and also there's a corresponding mapping for a certain policy stationary policy station reports are important of course because in infinite horizon problems optimal policies can usually be found within policies that do not change over time

对于平稳策略这个映射同样存在，因为无限期问题的最优策略通常是一个时间无关的策略

and this is the mapping T mu associated with a policy mu given by this expression

T\_mu表示这个表达式给定的策略mu相应的映射

okay now we have been talking the nice thing about this formulation is that it applies to very general systems arbitrary state space arbitrary control space arbitrary disturbance space it's very general

现在我们在讨论一个非常好的事情，就是它可以应用在很常见的系统中，任意状态空间，任意控制空间任意分布空间都可以使用

MDP - TRANSITION PROBABILITY NOTATION

however most of the work in approximate dynamic programming and also within this course is going to assume a simpler system

近似动态规划的工作，包括这个课程要讲的内容，都使用了一个更简单的系统

a system that has n states in other words a Markov chain with a finite number of states N

这个系统有n个状态，也就是这是一个状态空间有限的马尔科夫链

and we are going to it's convenient to switch to Markov chain notation

我们要使用一种简单的符号描述这个马尔科夫链

so the states can be either 1 2 up to n and I will use the symbol I to denote the state as opposed to X ok

状态可以是1，2直到n，我们可以使用符号I表达状态，与x不一样

so if you see I that means a discrete state finite number of states if you see X that means I have discrete or continuous or whatever

如果你看到I，它表示要给离散有限状态空间，如果你看到了x，它表示离散状态空间，连续状态空间或者随便什么样的状态空间

and instead of the system equation I'm going to use also transition probability notation which I mentioned in last week that in the earlier lectures that's an equivalent way of representing a system

我要用状态转移概率符号表示的系统方程代替我上周提到的的系统方程，这两种方式是等价的

so this P I ki k plus 1 is the probability when you are at state I K to go at the next state at state i k plus 1 under the influence of control you change the transition probabilities depend on UK and they are a substitute to the system equation if you're given a system equation like that then you can obtain transition probabilities through the problems tribution of W

P\_{i\_ki\_{k+1}}表示从状态i\_k转移到i\_{k+1}这个转移概率是收到控制影响的，你执行的控制会改变这个转移概率，这是系统方程的替代方程，就是说你得到了系统方程，就可以根据系统方程和随机变量w的分布得到转移概率

the cost is going to be denoted again with I notation as opposed to X notation and so this is cost associated with going from state I K to study Ik plus 1 with control UK

由于i与x不一样，成本函数也被重新定义了，新的成本函数是状态i\_k，i\_{k+1}和控制u\_k相关的函数

cost functions now are functions of a finite number of qualities there are only a finite number of points where this function is defined so a J is a vector an N dimensional vector a vector in RN with components J 1 up to J N

成本函数是一个有限集合，只有有限个成本函数构成的，在这里，J是一个n维的向量R^n，从J\_1到J\_n

and the cost of a policy again is transcribed in terms of the discrete notation

策略的成本也被离散符号重新定义

and the mappings can also be described like so

映射可以被这样定义

expected value over this one stage cost plus cost to go is obtained by taking the transition probabilities and adding over those transition probabilities similarly for team you so instead of expectation I have waited summation with the transition probabilities equivalent notation for discrete systems discrete state systems

这个阶段的成本与cost-to-go的和的期望，其中每一期成本都需要乘以转移概率，这是一个与离散状态系统等价的表达方式

“SHORTHAND” THEORY – A SUMMARY

okay so the theory and will go on to the throat one more time J star is the optimal cost vector for an end-state prop system it is an N dimensional vector is the optimal cost that's what we want to find

这个理论我要再讲一次，J\*是我们想要求的n维最优成本向量，

and it satisfies the system of equations it's the unique solution of this bellman equation the unknowns are J 1 up to Jn it's n equations with n unknowns nonlinear because this minimization over u

他满足这个系统方程，并且是bellman方程的唯一解，未知数是J\_1到J\_n，有n个方程和n个未知数，由于方程要最小化成本，所以是一个非线性方程

if you have a fixed policy then this J mu are a vector which is the unique solution of this system of equations

如果你有一个固定的策略，这个向量J\_mu就是这个系统方程组的唯一解

and this system of equations is linear so if you want to find J mu for a given policy you have to solve this linear system of equations

这个系统方程组是线性的，如果你想要计算一个给定策略的J\_mu，你必须解这个线性方程组

and the optimality condition is that the policy is optimal if and only if the policy attains the minimum in Bellman's equation with J star here ok that's an if and only if condition

策略的最优性条件是策略满足这个bellman方程的J\*

THE TWO MAIN ALGORITHMS: VI AND PI

and we can also we also have two fundamental algorithms value direction and policy direction

我们有两个基本的算法，值迭代和策略迭代

and value iteration starts from any vector in our end and applies the T mapping to it this dynamic programming operator again and again and again and in the limit it has the property that no matter where you start you converge to J star so that's one basic method for exact dynamic programming

值迭代从任意一个向量J开始，不断使用动态规划映射算子T，使用无数次之后就可以收敛到最优成本J\*，这就是一个求解精确动态规划的基本的方法

and the second method is policy direction given a stationary policy new k and you start to some arbitrary policy mu 0 but given the typical policy new k in the algorithm you evaluate it by solving this system of equations which is balanced equation for the cost vector of that policy again a linear system as I mentioned

第二个方法是策略迭代，给定一个平稳策略mu\_k，你可以从任意一个策略mu\_0开始，迭代过程中计算mu\_k，然后通过求解这个线性bellman方程组评估这个策略的值函数

in abstract notation is given like this in longhand limitations given like this you can solve it in any way that's convenient

抽象符号和完整符号的表达方式在这里，你可以使用任意方法方便地求解这个方程组

and then you improve the policy by finding a new policy that attains the minimum in this expression or in shorthand notation this it's this expression here

然后你通过最小化这个表达式，完整的或者抽象的都可以，他们标识的是同一个内容，就可以获得一个新的策略，然后用这个新的策略进行策略改进

so start with mu 0 compute new one evaluate mu 0 compute me 1 that policy improvement mu then evaluate that new – and so on

从一个策略mu0开始，评估mu0，改进mu0得到mu1，一直这么进行下去

and for a finite number of states we have shown last time that that it converges to an optimal policy in a finite number of directions

这是我最后一次提这个，策略迭代经过有限次迭代可以收敛到最优策略

and some of the points that are important here

有一些比较重要的内容

is that policy evaluation is equivalent to solving a linear system of equations

策略评估等价于求解一个线性方程组

however for large end and these are the problems that we are interested in large systems with a large number of states

我们关心的是这个系统有非常多的状态

you cannot really solve this system of equations exactly

所以你没法求这个方程组的精确解

because it has too large dimensions and the alternative perhaps the only alternative is to use use some value iteration for this system here

所以规模太大无法求解，唯一的替代方案就是使用值迭代解策略评估的成本

apply T nu repeatedly to what you have as a valuation of J\_mu and then the evaluation of J\_mu approximate

重复地使用算子T\_mu，你就可以得到J\_mu的近似解了

but still the method works this is the optimistic policy Direction method where the policy evaluation is done with a few value iterations okay

这种方法一直有效，叫做近似策略迭代(optimistic policy Direction)，策略迭代使用若干次值迭代进行策略评估

so now approximate dynamic programming amounts to introducing approximations and and obtaining approximate versions of this algorithms that I sort of make sense and that's what it is for the most part

这就是近似动态规划的介绍

APPROXIMATE DP

okay so now let me give you a broad description or approximate dynamic programming

现在我要粗略地介绍一下近似动态规划

GENERAL ORIENTATION TO ADP

it dates to the late 80s and it arose out of real need bellman coined the term curse of dimensionality for dynamic programming

追溯到80年代末期，bellman提出了动态规划维数灾的概念

many problems could be addressed in theory but in practice they were too big for the computers of the day or for any computers of the present or for any computers of the future

in fact

许多问题都可以在理论上被解决但是实践中由于规模过于庞大无法被计算机求解，无论是当年的，现在的还是将来的计算机

approximations became a necessity in order to solve problems

为了解决问题，近似变成了一个必须要做的事情

and then real problems and then people started looking more systematically proclamations

然后人们开始寻找更系统的理论

and there were several strands of research that originated in different parts of the scientific scientific community

在科学界有几种不同的研究方向

in artificial intelligence it the name of the field this is reinforcement learning reinforcement learning is a natural evolution of an old field people have been talking about reinforcement learning artificial intelligence in the early days in the s and now this came in as the idea of of adjusting policy as a result of observation and reinforcing the good actions relative to the bad actions this is something that's ingrained in the field of artificial intelligence and so there is this name there for smart learning and a lot of interesting ideas came from that field

在人工智能领域被叫做强化学习，强化学习是一种从已有的领域自然演变来的领域，早期人们在谈论强化学习的时候实际上是在研究如何根据观察到的信息调整策略，强化好的行动鱼坏的行动的影响，人工智能领域这是一个一直在做的事情，这个领域有很多有趣的想法和研究

neural dynamic program is the name of one of your textbooks it's a name that I've introduced with the joneses checklist in our book and and it means just about the same thing but we had a different orientation we're not computer scientists were not artificial intelligence people we are control and optimization operations research people we have a more mathematical viewpoint and and so it's a different direction to some of the same problems but substantially different attitude their methodology came out from these two sides of the field

神经动态规划是我介绍给你的一本书的名字，他和强化学习描述的是同一个事情，但是从不同的方向去进行研究，我们不是计算机科学家，不是人工智能研究者，我们是控制优化和运筹学研究者，我们从数学的角度来进行研究，所以这是一个不同的方向，一些相同的问题，从不同的方向进行研究，但方法论是来自于这两个领域的

finally there's a third third line of research called unfortunately ADP also but it doesn't mean approximate dynamic programming it means adaptive dynamic programming and came from came from the field of adaptive control adaptive control has to do with controlling systems for which you don't know an exact model however some unknown parameters like for example a robot may move something from here to here but the dynamics of the robot may be unpredictable may depend for example on the weight of the finger the carries okay

很不幸，第三个方向也被叫做ADP，但是他不是指近似动态规划，他指的是自适应动态规划，这个领域是从自适应控制发展来的，自适应控制必须控制一些你不知道参数的系统，比如一个机器人从一个地方移动到另一个地方，但是机器人的动态性是无法预测的，这个动态性可能依赖于机器人手指携带的货物的重量

and so adaptive control tries to find the control policies good control policies that are independent of the model but model free operation is also part and parcel of this approximate dynamic approximate dynamic programming field so that's where the common points where in adaptive operation and there have been contributions from that side of the field as well and there are substantial groups working on this in every part of the world including including China

所以自适应控制尝试找到一个好的控制策略能够不依赖于模型进行控制，但是模型无关的操作也被视为近似动态规划的一部分，所以这是一个共同点，同时自适应控制对adp做出了很大的贡献，全世界每个地区包括中国的研究者都在做相关的工作

okay we will focus on n State discounted problems which is the easiest case to address however you got to think of a huge number of states one possibility is to have a continued state space which you discretize with many points that's one way to get a few state space I will show you other examples of huge state space that arise in other ways while we are going to focus on finite state discounted problems a lot of the ideas apply to other kinds of dynamic programming cop models continued space continues time and discounted and extensions are possible but there are they're a little bit not quite as easy as for discounted problems and we may discuss selectively some of those later

我们会关注最简单的n个状态的折扣问题，如果状态空间非常大，比如连续状态空间，你可以把状态空间离散化为非常多的点，这也是一种减小状态空间的做法，我会给你们几个大规模状态空间的例子。我们在解决有限状态折扣问题的时候使用的方法可以应用在其他类型的动态规划模型中，连续状态空间，连续时间，离散时间和其他扩展类型，但是这些问题就没有折扣问题那么容易解决了，我们会有选择性地讨论他们

now for approximations there are several approaches

现在有几种常用的近似方法

one approach that is quite useful at times is the following you have a certain problem that's very difficult very complicated complicated dynamics large state space whatever and then you construct an approximation of that problem you neglect some of the difficult dynamics you simplify some things here some things there you obtain a simpler problem related to the original which you can solve by an exact method

一种非常有用的方法是你有一个非常困难和复杂的问题需要解决，他的状态空间什么的非常大，然后你忽略了原问题专用一些复杂的东西构造了一个这个问题的近似结构，你就有了一个与原问题相关的更简单的，可以使用精确算法求解的问题，

if you can solve the simpler problem by an exact method you can use the cost function of that problem as a substitute for the cost function the optimal cost function of the more difficult problem in bellman equation and obtained and a policy that sub optimal because it does not use the true optimal cost but rather the approximation obtained through this simpler problem so that's called problem approximation there are many many applications of that kind and but we're not going to go into that except in very small ways we are going to focus primarily on approaches that are based on simulation

如果你能够使用精确算法求解这个更简单的问题，就可以用获得的成本函数代替原问题的成本函数，然后可以得到一个次优策略，因为他没有使用最优成本函数而是使用近似成本函数代替，但是更容易求解，这就叫做问题近似，很多应用都在使用这种方法，但是我们不会深入讨论他，我们会关注基于仿真的求解原问题的方法

and there are three major types that use simulation one is a roll out a simple and very effective technique that I discussed in the previous two lectures I'm not going to say anything further about that the other two approaches that we are going to talk about our approximation in value space where we approximate the j function okay we try to find an approximate version of that through through a parameterization and the other is to approximate directly an optimal policy you introduce approximations into the policy and you try to find the parameters of the approximation that give you a good policy we're going to focus primarily in value space approximation less than policy space but I'm going to mention both of these in my lecture today

主要有三种仿真的使用方法，一种是roll out，这是一种很简单但是很有效的方法，我之前讲过这个方法，但是我会再讲点关于它的东西，另一种我们要讨论的方法是值空间的近似，我们用参数化的方法近似成本函数J，最后一种方法是直接近似最优策略，通过参数化方法近似最优策略，然后寻找这个近似策略的参数，这样可以得到一个不错的策略，今天的课程我们会更多地关注值空间而不是策略空间的近似

okay now simulation is I've mentioned so many times simulation why simulation important for us we may have a perfectly deterministic problem why do we want to simulate it with Monte Carlo random number generators and random trajectories that seems a little strange the idea is is simple and it's it's very important let me to understand it

我很多次都提到仿真，那么为什么仿真方法对我们很重要呢，如果我有一个完美的确定性问题，我们为什么需要使用蒙特卡罗模拟，数值生成和路径采样之类的方法类解决它，这看起来有一点奇怪，但是这样做的思路很简单也很重要，下面我要解释一下

WHY DO WE USE SIMULATION?

so why do we use simulation

那么我们为什么要用仿真呢

there are two reasons

有两个原因

and the main reason is that it helps the computation large problems involve huge calculations

主要原因是他有助于计算涉及大量计算的问题

for example inner products the simpler operation from linear algebra is to compute an inner product of two vectors to compute an inner product of two very large vectors requires a very large computation

比如内积，计算连个向量的乘机，如果向量非常大的话，需要的计算量也会非常大

so can we simplify that it turns out that simulation can simplify this calculation and you may have heard about methods for computing integrals complicated integrals by Monte Carlo integration it's the same idea

所以我们可以很简单的证明仿真可以很容易地获得这些计算的结果，使用蒙特卡洛计算复杂积分也是这个想法

so let me describe this we want to compute efficiently sums or expected values involving a very large number of terms one per state let's say well any sum of this form let's take a generic sum involving summation of n numbers a 1 up to a n

所以让我描述一下，我们想要在状态数量非常多的时候有效地计算总和或期望值，一般需要从1加到n

then I can write this as an expected value like this I introduce a probability distribution so I want side 2 and so on up to size the vector components I want up to say n it could be a uniform distribution like 1 over N probability for every I

我可以把它写成期望形式，假设状态分布是一个从1到n的均匀分布

what it could be it does not have to be uniform it can be any kind of probability distribution I can write the Sun like so ok multiply and divide with xi and this becomes the expected value of the ratio AI of XII with respect to the distribution of the Y vector

他可以不是均匀分布，可以是任意形式的分布，我也一把它写成xi乘以a除以xi的形式，这时候值就变成了y向量分布下a\_i/xi的期望值

so any summation can be written as computing an expected value in computing an expected value can be done either by adding the terms or by flipping coins in doing simulation

所以任何求和都可以写成计算期望的形式，这个期望可以通过提添项或者仿真来完成

now if the number of terms is gigantic it may be better to flip coins fewer number of times and get an approximation which however may be adequate for practical purposes

如果求和项数量非常多，那么就可以通过采样使用更少的计算来求期望，这个期望是一个近似值，但是一般可以满足实际需求

this is a method that's used very widely in all sorts of fields and it's an idea that's important in our context

这种方法被所有领域广泛应用，也是我们的课程的一个重要方法

and here it is again the expected value or the sum can be approximated by generating many samples change samples from 1 up to M and according to this distribution site and doing Monte Carlo averaging in other words approximate this sum with a Monte Carlo average corresponding to the samples that you have generated

求期望和累加值可以根据状态分布生成很多样本，然后进行蒙特卡洛平均来近似期望或者累加值

potentially it's a lot fewer computations if n again is very very large

如果n非常大，这种方法可以减少很多计算量

so that's one reason doing linear algebra operations in a product's matrix vector multiplies involving gigantic matrices and gigantic vectors

所以使用仿真的一个原因是使用线性代数运算涉及的矩阵和向量规模都非常巨大

there's a second reason why simulation is also convenient

第二个原因是仿真很方便

I mentioned model free operation if I don't have a mathematical model of the system and I have instead a computer model of the system I can use simulation to carry out values of the algorithms that that we are interested in like valuing policy iteration

我之前提到的模型无关的操作，如果我没有系统的数学模型，我就可以使用仿真模型代替数学模型来计算算法中需要的数值，比如值迭代和策略迭代

these are the two main reasons lack of an analytical model and instead having simple access to simulation and computing large sums a complexity a computational complexity issue

这就是使用仿真的两个原因，仿真很容易获得与计算累加需要的计算量过于庞大

APPROXIMATION IN VALUE AND POLICY SPACE

okay I think questions before we go into more specifics okay let's discuss in some generality approximation in value and policy space

下面我要讲一些通用的值空间与策略空间的近似方法

APPROXIMATION IN VALUE SPACE

first of all approximation and value space what

第一个是值空间的近似

this amounts to is replacing J star or J mu from some other using some other function from a parametric class in other words this J till the eyes of R is a vector one with one component for every state but depending on a parameter vector R okay so R is a parameter vector and this is some functional form that I haven't told you anything about yet how you how you pick it but that's the idea of using some parametric approximation to J star or J\_mu

也就是使用其他参数化函数代替J\*或者J\_mu，r是一个向量，状态空间中每一个状态都对应r中的一个元素，所以r是一个参数向量，我还没有告诉你如何选择它的信息，但是这种方法的思路就是使用参数化函数来近似J\*或者J\_mu

and then we can use them right in various theoretical expressions or in values algorithms and so on

然后我们就可以用理论表达式或者数值算法中使用它们

okay now what is the role of this parameter this J tilde is a function okay a function of state if you choose one value of parameter it may be something like this another value of parameter it's different something like that for each value of the parameter we get different functions now you have something that you want to approximate and you want to adjust the parameters so that you get a good fit between the two that's the idea

参数r在这里的作用就是，J tilde是一个状态的函数，如果你选择了一组参数，这个函数就是这样的，选择另一组参数函数就是这样的，对于每一组不同的参数，函数都可能是不一样的，所以你需要选择一组合适的参数来近似

that's the role of the parameter we can change the shape of J tilde so that it's close to J star or J mu

这就是参数的作用你可以选择参数来改变J tilde的形状来近似J\*或者J\_mu

so how are the issues here the first one is to select the parametric class which is also called the approximation architecture do you want this J tilde to be linear in R you want it to be quadratic do you want it to be polynomial do you want it to have a more complicated expression this choice is very important because that becomes your new search space within which you're going to do your approximation

第一个问题是如何选择参数的形式，这也被叫做近似结构，你希望J tilde是线性的，二次的，还是多项式的，或者你希望有一个更复杂的形式的近似形式，这个选择非常重要因为你选择的参数表达式形式会决定你近似的搜索空间

the second field is after you have selected the approximation architecture find a good method for for obtaining the parameters this is called training the architecture you may have heard of training neural networks training neural networks neural networks is an approximation architecture that depends on certain parameters training the neural network means adjusting the parameters based on data to achieve a certain objective so choice of approximation architecture and then training the architecture by using some kind of algorithm

第二个事情是选择了近似结构之后，需要用一个比较好的方法调整参数，也叫做结构训练，比如训练一个神经网络，神经网络就是一种近似结构，训练神经网络表示根据获得的数据和目标函数来调整神经网络的参数，所以这种方法，需要选择合适的近似结构，然后使用合适的算法训练这个近似结构

and these are the two key issues success depends very strongly on how these two issues are handled and of course it's important to have insight about the problem in order to make good choices here

所以这个问题解决的是否成功，取决于这两个事情处理的好不好，也取决于对这个问题的理解，理解比较好就可以选择一个好的近似结构

now the training may be done using a simulator if you have or may be done using the real system and but if there is no mathematical model

训练模型可以使用仿真，也可以使用真实的系统，如我们不知道数学模型是什么

and we are going to focus on the case of simulation but there are many other possibilities that I'm not going to get into

我们会关注仿真方法，当然还有其他可以使用的方法，我不打算深入讲

also we may use parametric approximation for J star or J nu but yet there is also an alternative of approximating two factors or cost function differences in the previous two lectures I mentioned that there is this possibility of using Q factors which are associated with statement repairs we may want to approximate those or cost function differences that may be another idea of using approximation to get something close to those

我们使用参数化函数近似J\*或者J\_mu，当然还有其他近似方法，比如我之前讲的Q函数的近似或者成本函数差分，Q函数是对状态动作对的值，我们对它进行近似或者使用成本函数差分来近似

APPROXIMATION ARCHITECTURES

okay how about approximation architectures well they are divided in linear and nonlinear

近似结构被分为线性的和非线性的

linear dependence of J tilde on R or nonlinear dependence

线性结构与非线性结构都是一个依赖于r的函数J tilde

linear architectures are easier to train easier to deal with they have more algorithms and with better performance guarantees on relative to nonlinear architectures

线性结构更好训练，有很多算法可以使用，能够比非线性结构表现得更好

but there are some nonlinear architectures which are very useful in practice neural networks is one possibility I don't know if you if you know about how much you know about neural networks I do not have time to explain but what I can tell you is that they involve a sequence of mappings that are nonlinear nonlinear and depend on parameter R on the parameter vector R so so they given I they give you a J tilde sub I of R where R is whatever parameter happens to be in the network

但是非线性结构实际中更好用，比如神经网络，我不知道你了解多少关于神经网络的内容，我没有时间去讲它，但是我要告诉你的是，它包括一系列依赖于参数r的映射，给定一个i可以根据参数r计算出J tilde的值

now neuronal would have their own culture and their own algorithms by which you train them and there are very rich class of of of architectures but we're not going to look at those we're gonna focus primarily on linear architectures

神经网络有很多自己的算法可以用来训练参数，可以提供种类非常多的近似结构但是我们这个课程关心的是原始的线性结构

now linear architecture quite rich as well and I'll explain why

线性结构很强大，我来解释一下为什么

okay here's a here's an example of an architecture from computer chess computer chess has made great strides now computer chess programs can be to do routinely with the world champion okay they start from very low but over a period of time then improved a great deal thanks also to great advances in computing hardware

举个例子，计算机象棋，计算机象棋现在已经可以与世界冠军较量了，他们是从很低的水平开始的，由于计算机硬件的发展，获得了很大的进步

what does chest' have to do with approximation architectures well at each point in again there's a board position the current board position think of this board position as the state of the game and think of all the moves that you can make at the position has controls or decisions so you have you have the state and you have also all these possible decisions and you want to calculate the score for each move that's like a Q factor okay

象棋是怎么设计近似结构的呢，在每一个点都获取棋盘信息，把棋盘信息当成一个状态，把所有能走的位置当成决策，所以你现在就有了系统状态和能执行的决策，接下来你需要计算每一个决策的得分，就像Q值

state control pair you want to evaluate a number that tells you how good this move is

你想要给状态控制对估值以便于知道这个决策有多好

now the way chess programs do is they involve a feature extraction mapping first of all they project several moves forward okay and then at the end of the search they look at that at a position and they extract some features from it now the features are things that chess play is recognized as being important like for example the material balance do I have more material than my opponent okay that's good do I have more mobility in the sense that I can move my pieces to more squares than my opponent that's also a good feature safety of my King strategic features open files and so on they're like 30 40 50 features that just players take into account

现在国际象棋的做法是给一个状态提取映射，首先他们预测几步，在搜索结束的时候观察位置并提取特征，目前特征是国际象棋中很重要的内容已经被认识到了，比如物料平衡，我的材料比对手更多，我可以把棋子放到比我的对手更多的位置去，那么这就是一个好的特征，比如能保证王安全的特征，玩家可能考虑30，40，50个特征

and these are weighted with various weighting factors and all of this is put into this complicated position evaluator and outcomes the score which is the Q factor associated with the starting position with your current position and the move that you are considering

这些都是用不同的权重加权计算获得的，通过对位置进行估值计算出得分，也就是起始位置和当前位置和决策的Q值

so you get a score for this move for that move for that move for that move and you pick the move that gives you the highest score and that's the way the computer chess programs work

所以你获得了各种移动的得分，然后选择一个得分最高的移动，这就是国际象棋的工作方式

generally speaking there are very few features in this mapping involves a few unknowns actually

一般来说这个映射中特征比较少，未知信息也非常少

the the the number of features is like I said 30 40 50 so there are only a few parameter 30-50 parameters to select in this box

特征的数量就像我说的，30，40，50个特征，所以只有很少的参数需要调整，差不多30-50个

the number of possible chess positions of course is astronomical get your search space involves against your position evaluator involves only a few numbers and it's quite remarkable that it worked so well

你的搜索空间涉及的棋盘位置数量可能是个天文数字，但是你需要评估的位置数量会比这少得多，而且国际象棋程序工作的很好

now in computer chess the weights of various features have been selected empirically over time over a period of 50 years okay

现在国际象棋的特征的权重设置是根据50年以来的经验完成的

people have found that this they look at the performance of the chess program and they decide to maybe increase the weight of some feature relative to another feature and and that and through trial and error good position evaluators have developed

人们观察国际象棋程序的表现然后决定增加特征的权重，通过试错与调整的方式寻找好的估值方式

in our field we want to do this more systematically

在我们的研究领域中，估值方式会用一种更系统的方式完成

LINEAR APPROXIMATION ARCHITECTURES

so let's look first at linear approximation architectures

让我们来看看线性近似结构

the function that we want to approximate may be very very complicated however if we have good features that encode a lot of the non-linearity a lot of the complexity of the cost function

then if we can use such features then we can do simple weighting between them like linear waiting

我们想要近似的函数可能非常复杂，如果我们有很好的非线性特征编码就可以用这些特征像线性结构一样使用简单的权重进行近似

so if you have if the non-linearity inheriting the cost function is encoded in the features then the approximation may be quite accurate without a complicated architecture XT

所以如果你使用成本函数的非线性进行特征编码，可能可以使用很精确而且不复杂的结构进行近似

after an extreme example the ideal feature is the true function you want to evaluate if you had that then that would be great you need anything else if you have something that's approximately that then that's a good feature and then you can use simple waiting for being for the architecture

so with one chosen features we can use a linear architecture okay a linear architecture is described by these three terms which are called the features of state I now for each state I these are all vector of features prime denotes transposition R is a vector weights and this gives you a number the inner product between the feature vector of the state and the weights that you are considering

对于一组给定的特征，我们可以使用线性结构，线性结构被描述为三个部分，状态i的特征，这是要给特征向量，权重向量r，r和特征的内积就是我们要近似的函数值

the cost associated with the overall cost function that's associated with R is a linear combination of the columns of a certain matrix

与r相关的成本函数是一个矩阵的列的线性组合

this is a Miss fee matrix is a matrix that has a row dimension equal to the state number of states and column dimension equal to the number of features

这个矩阵是一个费用矩阵，行的维度等于状态的维度，列的维度等于特征的维度

So fee is a gigantic matrix in the long direction and a short matrix in the horizontal direction

这是一个非常大的矩阵，竖着比较长，横着比较短，也就是说行非常多，列比较少

its columns are this Phi of Phi I capital they can be viewed as basis functions for a subspace approximation

列表示phi(i)，即基函数

in particular J tilde is some function within a lower dimensional space

J tilde是一个很小空间里的函数

spanned by the columns of the matrix field

and once you have a certain vector R a certain vector weights you have you can take any state extract the features associated with that state extract the feature vector Phi I and then wait linearly with r and you obtain the approximation associated with state I this J tilde eyes are

如果你有了一个确定的权重向量r，你可以提取任意状态的特征向量phi(i)然后进行线性组合计算近似值J tilde

of course what's important here is to find good features that characterize well being the state

这里比较重要的事情就是找到一个能比较好地描述状态的合适的状态特征

so for example this so for example if the optimal cost function you know that's approximately quadratic well you might use as features a polynomial a quadratic polynomial this phi here each row is involves a constant a linear for a constant I I squared and so on

比如你知道一个最优成本函数是二次型的，你就可以使用二次多项式phi(i)来提取特征，每一行都是一个线性常数，比如i的平方之类的

okay there are many examples of feature types polynomial approximation radial basis functions is a very very extensive literature in methodology a societal approximation and I'm going to give you some examples

有很多类型的特征，比如多项式近似，径向基函数等，这些都是应用非常广泛的近似方法，下面我要给你举几个例子

ILLUSTRATIONS: POLYNOMIAL TYPE

the first and simplest is polynomial approximation

第一个也是最简单的一个是多项式近似

for example a quadratic approximation function suppose that the state is I from and the same I has two dimensions

举个例子，二次项近似函数，假设状态i有两个维度

let's say you're considering two dimensional space approximation so there are two components I 1 and I 2 three dimensional and so on

这样你就在两个维度的状态空间内近似，有两个元素i1和i2，一共三个维度

these are not States the desired dimensions the components of the state vector

这不是状态向量的维度

so you may consider a quadratic approximation involving for each I which is of this form a linear weighting of these components plus a quadratic weighting of these components

所以你需要考虑一个二次项近似，没有给状态i都使用这种形式近似，线性权重加上二次项权重

so this is in two dimensions a quadratic kind of function

所以这种类型的二次函数有两个维度

and the coefficients are this is the constant coefficient a coefficient that multiplies the linear terms and a coefficient that multiplies the quadratic terms so the R vector has all of these components here

这些是常系数，一个常系数乘以线性项，另一个常系数乘以二次项，所以向量r包括这里所有的元素

so that's a quadratic fit it could you could have a quad linear a linear feed or cubic fit to be to be function that you want to approximate

所以这是一个二次项拟合，你可以使用线性拟合或者三次项拟合你想要近似的函数

another possibility is interpolation

另一种可行的方案是插值

you have this function that you want to approximate it you take some selected points and you calculate the this function and then you make a piecewise constant approximation

你有一个想要近似的函数，然后你选定一些点并且计算这个函数，之后使用分段近似

now the values are the selected points are exactly your features

那么这些分段函数的值就是你选择的点的特征值

also you do not need to have you you do not need to have a a piecewise constant approximation you may have a piecewise linear approximate or a piecewise quadratic approximation

或者你不需要用分段常数近似，可以用分段线性近似或者分段二次项近似

all of this fits into the interpolation category

这些近似都是插值方法的一种

so let me read here you select a subset I of special representative states

我们读一下这个地方，你选择了一个代表整个状态空间的状态的子集合

and the parameter vector has one component R I per state

每个状态的参数向量都只有一个元素，即r\_i

and then the approximating function is exactly RI at the selected points

在被选中的点的近似函数的值就是r\_i了

and some interpolation for the intermediate points using the values at I could be a linear interpolation quadratic interpolation spline or whatever many possibilities

被选中的点中间点的值通过插值完成，被插入的值就是被选中的点的值，当然也可以是线性插值，二次插值或者是所有形式的插值

okay so this is standard stuff people in numerical analysis do this sort of thing all the time

这是一个标准的方法，人们在做数值分析的时候都是这么做的

they approximate the solutions of partial differential equations with piecewise constant functions splines linear and wavelets many different ways

求解偏微分方程的时候他们可能使用分段常数，样条线性和小波等很多不同的方式近似

now here's another example which is sort of more problem specific

现在我要举另一个特殊问题的例子

you look at your problem and you try to figure out what are the features that are important for this problem and then you construct an architecture based on that

你观察你的问题，试图找到对这个问题重要的特征，然后根据这些分析设计近似结构

A DOMAIN SPECIFIC EXAMPLE

I don't know if you know this game I know that video games are popular in China and this is one of the first video games Tetris to visit this is it known in in China

我不知道你们是不是知道这个游戏，我知道这个游戏在中国很受欢迎

in Tetris you have you have you have a certain matrix of possessions and and then objects fall from the top and then you try to fit them onto the top of the wall and then you can move them horizontally or rotate them and there are several different types of objects and they come in

俄罗斯方块有一个方块矩阵，不同形状的物体从顶上掉下来，你需要水平移动或者旋转让他们安装在墙上

and each time you complete a row solid row you gain a point you eliminate that row and then the wall goes down by one and the objective is to play for as long as possible earn as many points as possible before the wall rises up all the way to the top at which time the game stops

每一次你组成一条线时，都可以得到一分，然后墙壁高度下降一，游戏的目标是尽可能长时间地玩下去并获得更多积分，当墙的高度上升到达方框顶部时，游戏就结束了

okay so this is a problem that can be viewed as a dynamic programming problem

这个游戏可以被看作一个动态规划问题

the starting state is the empty board the the the state at any point is the the board position you know the zeros and ones where in this matrix there is an object or there is it's filled or it's empty there's a gigantic number of states for a standard ten by twenty board the number of state is created by two to the 200 now I don't know if you have a you can think of how big two to the 200 is but it's more than the number of molecules in the entire universe okay it's it's a gigantic number

这个游戏的起始状态是一个空板，你知道这个板子上每一个格子的状态，0或者1，表示某一个位置被东西填满或者是空的，状态数量非常多，有2^200个，我不知道你们对这个数量有没有概念，宇宙中的原子也没有这么多

now why is this a control problem well at every state there is and given any object that falls I have a different set of different options I can move it horizontally and I can rotate it okay so this is my decision the number of controls here are in the order of of let's say 30 or something like that

为什么说这是一个控制问题呢，给定一个物体从顶部掉下来，有很多操作可以执行，平移或者旋转，我可以从大概30个决策中选一个执行

okay the number of states is gigantic the control space is relatively small there is randomness here while there is randomness because the object that comes out is random it can take one of several shapes

状态的数量特别多，决策空间相对比较小，同时这是一个随机问题，因为掉落的物体是随机出现的

so there is randomness and there is there are decisions this cost or reward every time you eliminate that row you get one point okay so you want to maximize number of oh let's minimize number of negative role negative number of rows that you have

这就是问题的随机性和能执行的决策，每一次你消掉一行就可以得到一点积分，你希望最大化得分或者最小化得分的相反数

and what is J star J star sub I is the optimal score starting from position I so from the starting position this J starts off of the empty board and for any possible board position there is a corresponding optimal score that you can get if you are an optimal player if you are the world champion we were the absolute optimal because nobody now has to have to play optimally and and nobody can calculate this j-stars of I there is such a vector but it has dimension greater than two to the 200 and we will never calculate it

J\*(i)是从状态i开始的最优得分，J\*就是从游戏开始即空方块的最优得分，假设你是最好的玩家，或者是世界冠军，反正是最好的那一个，因为没有人知道怎么玩才是最优的，同样没有人能够计算出J\*(i)到底是多少，这是一个超过2^200个元素的向量，没有人能算出它到底是多少

what we can do instead is culturally an approximation of that using some kind of algorithm likely involved in simulation and use the J tilde in place of J star in choosing a move

我们现在能做的就是使用有仿真的算法计算J tilde来近似J\*选择行动

now this problem has been started quite a bit has a history of more than twenty twenty years I think it was at MIT I was involved also when it got started

在MIT这个问题已经被研究超过20年了

and and and even though the problem is so gigantic with just twenty two features we have been able to get a reasonably good tested Tetris players using policy direction in other words you start from a certain bad that display a very primitive player and then that's your initial policy in a policy duration scheme then you train using this twenty two features the corresponding architecture to obtain well you use simulation to calculate the score that this the average score that this player can achieve then you obtain a better player by using policy improvement and so on and starting from players that would achieve like an average of thirty points per game we got up to like three thousand points four thousand points five thousand points for various variations and we were very happy than we thought that this is just great

这个问题规模这么大，但是我们发现只用22个特征就可以很好地描述这个问题，然后使用策略迭代来训练一个游戏策略。即你从一个很坏的游戏策略开始，然后开始进行策略迭代训练这22个特征对应的近似结构，训练时使用仿真计算的平均得分估计这个游戏策略的得分，然后不断地使用策略改进就可以获得一个更好的游戏策略，平均得分从30开始，直到三千，四千，五千，我们很高兴能看到这个很好的结果

and indeed for a for for a number of years for like maybe ten fifteen years this was the best that could be achieved

这是这十五年最好的结果了

eventually people started using other methods and it turned out that with other methods not policy duration methods we the schools that were achieved were in the order of seven hundred thousand nine hundred thousand so we thought that we're doing well but we have no way of telling of course because we did not know the optimal scores and yet was not possible to get with a small number of features either twenty-two features or some other additional features very very high scores so an interesting probably has been used as a testbed in competitions okay like they have competitions but finding a good soccer player about finding you know a good chess player and they have a competition for for tetris players

最后人们开始使用其他方法，事实证明他们使用非策略迭代方法得到了七十万与九十万的得分，尽管我们做的比较好，但是我们仍然不知道最优得分是多少，最优得分几乎不可能知道。至少不能用22个特征或者更多特征得到非常非常高的分数了。还有一个很有意思的测试平台，可以找到优秀的玩家，比如国际象棋玩家或者俄罗斯方块玩家

now let me tell you also quite a bit 22 features the approximation architecture used here is a linear weighting of 22 numbers 22 numbers that when you look at a certain position you extract these 22 numbers from the position and you weigh them with corresponding weights

现在我们使用22个特征的线性近似结构，你从所有信息中提取出22个特征然后训练相应的权重

what are these features well if you are a Tetris player you sort of recognize these features an important feature is how high the world is okay and in particular how high the different columns are so that's ten features right there for a if there are ten columns the height of each column gives you ten features

这些权重是这样的，如果你是一个俄罗斯方块玩家，你会认识到对游戏重要的特征，就是墙的高度，尤其是不同列的高度，如果有十列，就有十个特征表示这些列的高度

what's also important is the difference between the heights of successive columns if you have a at the top you have a lot of jaggedness a lot of differences between heights that's a bad thing

同样重要的是高度之间的差异，如果高度差异很大，顶部有很多锯齿状的方块，这就是一个很差的状态

so by taking differences of heights you get another nine features

所以这些列的高度差可以提供另外9个特征

the number of holes in the wall is an important feature we the there was a constant that was viewed as a feature and there was one more that I can't remember but there are features of this study the only 22 numbers that's the important thing

墙上的洞也是一个很重要的特征，这是一个常数，也是一个特征，我忘记是多少了，但是一共有22个特征对游戏影响很大

even though you have as a cost vector that is gigantic it is only at a low dimensional dimensional subspace over which you approximate it and yet you get a very very good very very good performance

即使你有了这个规模非常大的向量，他也只是在低维空间近似这个问题，但是仍然能获得很好的效果

so we have standard approximation architectures and we have also the main specific approximation architectures like in chess and Tetris

所以我们有了这样一个标准的近似结构，还有具体的，比如国际象棋和俄罗斯方块的特定近似结构

APPROX. PI - OPTION TO APPROX. Jμ OR Qμ

and now let's see how these architects are used with policy duration

现在我们看看这些结构是怎么使用策略迭代的

we can use simulation to approximate the cost of the current policy

我们可以使用仿真来近似当前策略的成本

and then generate an improved policy by minimizing in the approximate bellman equation

然后可以通过最小化近似bellman方程对策略进行改进

so here is approximate policy direction for costs you start with a certain policy you evaluate this policy approximately by using an approximation architecture in finding J tilde sub mu essentially finding are the corresponding vector associate with the sponsor then given that you go into balance equation and find an improved policy mu bar and then you go back evaluate that and so on

这就是近似策略迭代的流程图，你从一个策略开始迭代，使用近似结构计算J tilde\_mu对这个策略进行近似评估，可以获得一个相关的成本向量，然后通过最小化bellman方程对策略进行改进，这样评估-改进不断进行，就可以得到一个近似最优的策略

so it's the same policy direction that you have for exact points evaluation except that you use an approximate an approximation architecture in between so in the context of Tetris you start with a certain policy for playing Tetris you play a lot of games and you evaluate the score of that policy from be approximated by find corresponding vector of parameters then using that you generate a new Tetris player you can calculate the decisions online using this function here for nu bar and then evaluate and so on

这与精确策略迭代是相似的，区别就是计算策略成本时使用近似结构，假定要找俄罗斯方块的策略，从一个策略开始，使用这个策略仿真很多次评估这个策略的得分，可以获得这些参数向量的值，然后使用这个值计算一个新的俄罗斯方块策略，用这个策略可以计算online决策，然后评估策略，改进策略，一直这么进行下去

alternatively you could use you could approximate the Q factors of MU where you start with a certain initial policy evaluate approximate Q factors involving a parameter vector R and then generate an improved policy by just minimizing the new policy is obtained by minimizing and finding the optimal Q factor at any minimize no value and frankly a problem that the optimal Q factor at point I

或者你可以近似策略mu的Q值，从一个初始策略开始，使用参数向量r评估策略的Q值，然后通过最小化Q值生成一个改进策略，然后继续评估-改进循环下去

I'm sorry that this letters are a little bit small I don't know how well you can read them have any questions okay now we use approximation in the context of policy in aeration to approximate the cost or the Q factors of policies

抱歉这些字有点小，不知道你们能不能看到，有任何问题么。现在我们会近似成本或者Q值了

APPROXIMATING J∗ OR Q∗

an alternative is to try to approximate directly J star or Q star

另一种方法是直接近似J\*和Q\*

Q learning is an algorithm for finding the optimal Q factors the optimal Q factors as you recall are given by this expression they are the sum of the one stage costs associated with I in here plus the expected cost to go using an optimal policy

Q learning是一个计算最优Q值的算法，Q值的计算表达式在这里，当前阶段i的成本加后最优策略产生的后续成本

and if you know the Q factors Q star you can't calculate the optimal costs by by means of by minimization

如果你已经知道了Q\*的值，你可以计算最优成本同坐最小化这个表达式

so that's one possibility this Q learning algorithm that aims at finding directly Q star in j\*

所以一种解释是Q learning通过J\*直接计算Q\*

there's also another method called the bellman equation error approach remember bellman equations J equals TJ so if you so if you form a least squares objective where you minimize over all functions J this error in satisfying the equation then the optimal solutions is J star and and so however if you look over an approximation architecture what you try to do here is minimize the errors in satisfying Bellman's equation using J tilde and you minimize over all possible parameter vectors so this is a generic approach for solving approximately equations you within a certain within a certain class of approximations you try to find one that minimizes the error in satisfying this equation the expected value here is taken with respect to some distribution over the state so this is a number and this involves this is Lee squares but non linear least squares because T is a nonlinear mapping

另一种方法叫做bellman误差法，使用bellman方程的J和TJ构造一个最小二乘目标函数，对所有J最小化这个误差，其中最优方案是J\*，如果你用一种近似结构和J tilde最小化满足bellman方程的误差，就可以获得近似结构中的参数向量的值，这是一种通用的近似方法，使用这种方法近似时，你试图找到最小化bellman方程误差的参数向量，目标函数的期望是关于状态的分布的，同时由于T是一个非线性映射，所以这是一个非线性最小二乘问题

there's another method called approximately Iinear programming but let's not discuss it here you can find in the literature

还有一种方法被叫做近似线性规划，这个课程不做讨论，你可以自行去看相关文献

where is Q learning in its standard form finds exact values of Q star and J star it can be used also with approximations although not very rigorously we'll get into that later

Q-learning计算得到的是精确的Q\*和J\*的值，你也可以使用近似算法去计算他的值

also you can use both Q learning and the bellman error approach in the context of approximating of a policy evaluation as opposed to calculating we exact exactly optimal functions

你也可以用Q-learning和bellman误差而不是精确算法来进行策略评价

APPROXIMATION IN POLICY SPACE

so you can either approximate the cost of policies or the Q factors of policies or the optimal things and now all of this had to do with approximation in value space let me just spend one slide on approximation in policy space

所以你可以近似策略的成本，策略的Q值或者最优成本和最优Q值，这些近似都是在值空间中进行近似，我接下来要给你们讲策略空间的近似

we want to find an optimal policy we want to find a mu in the control action for every R

我们想要在所有的r中找到一个最优的控制策略mu

suppose that we parameterize the set of policies using some parameter vector

我们使用参数向量将策略参数化

we use some parametric form of policies

我们使用一些参数形式的策略

for example we may look at linear policies

比如这个线性策略

that are of this form without this quadratic term

没有二次项的策略

or you may look at quadratic policies with this r1 r2 and r3 being weights

或者引入二次项策略

and/or you may want to look at a linear feature based kind of approximation where the parameterization of policy is is done through feature functions of the state with corresponding weights are

你也可以看到基于线性特征的近似，这时候你使用基于状态的特征函数与相应的权重将策略参数化

so once you define approximation architecture your search space becomes a limited subset of the set of all policies and then you may minimize over all of this this limited search space

所以你设计了一种近似结构，搜索空间就被策略的子集限制住了(因为近似结构只能表示一部分策略)，你可以在这个有限的搜索空间中搜索最小化成本的策略

so each value of r defines a stationary policy with with cost at state I denoted by J tilde I in R

每一组r都定义了一个平稳策略，状态i的近似成本用J tilde(I;r)表示

then you try to find r that minimizes a certain objective that involve some weighting of the states so our parameter Isis policies you plug those policies in they give you some kind of cost for each state then you weigh this by probabilities and then you minimize that

然后你尝试寻找最小化权重目标函数的r，所以我可以说参数就是策略，你把策略放到这里，就可以计算出每个状态相应的成本，然后根据状态出现的概率乘以相应的成本，需要最小化的就是他们的和

and it's possible to to to have methods to minimize objectives like that involving gradient methods random search other kinds of methods these are its variety of possibilities

想要最小化目标函数有很多方法可以用，比如梯度方法，随即搜索等

this is very different than approximation in value space there is no policy duration policy improvement this is more direct search parameterize the objects of interest and then do some kind of a search to find an optimal object within this class

这是一种与值空间近似很不一样的方法，没有策略迭代也没有策略改进，而是直接搜索参数找到最优策略对应的参数

it's also possible to make a connection with parametric architectures for course for example if you introduce a cost approximation architecture J hat which depends on R then you can obtain a parametric class of policies that's obtained by minimization in bellman equation

这种方法也可以与参数结构联系起来，比如你定义了一种依赖于r的成本近似结构J hat，然后就可以通过最小化bellman方程获得一些参数化策略

so this is sort of indirect parameterization of policies through parameterization of the cost function

这就是一种通过参数化成本函数的间接参数化策略方法

okay I'm going to have a few things to say about proximation policy space in the in the in in the last lecture however let me just mention that the Tetris success that I mentioned earlier the fact that policy duration type of algorithms gave you scores like 5000 points and then there were some other methods that gave you many years later we tried many League years later and they gave you scores of the hundreds of thousands five hundred thousand seven hundred thousand and million points out on the con the average per game the methods were of this type okay there were a random search type of methods in the space of policies so these methods have to show some impressive successes as well even though they have a little bit less of a over solid theoretical foundation behind them

关于策略空间的近似最后一次课我还有一些事情要说，我之前提到的俄罗斯方块游戏，策略迭代方法获得了成功，能够得到五千分，但是很多年后其他方法能够获得更高的分，比如平均每局几十万或者几百万的分数，所以在策略空间中随即搜索之类的方法同样获得了很大的成功，虽然理论支撑比较弱

Q&A

okay I think it's a good time to break here have any questions we have been discussing these this these broad issues for approximation in particular approximation architectures and Proclamation value space and policy space all of this is done very quickly we're going to come back to it in next week in more details at least parts of them but are any questions now yeah okay so the question is okay given this approximate policy direction you have a policy and when you evaluate it how do you do you update are at the same time as we update the policy well you do because our defines the approximation so our depends on mu you find forgiven you you find R that gives you a good approximation to the cost of that new cost function of that new so this should be ours of mu okay it's cool so you're different so actually the policies you never you never see them they are too big to be eyeballed okay they are gigantic they have gigantic number of components what you have is in fact a vector of parameters R which is a much smaller quantity so you move from one hour to an hour R to another R but implicitly you evaluate these policies and you do increase the policy improvement I'm sorry how to how to calculate our okay so that's a major issue how do okay how do you calculate our here to do this Li squared is this fitting between J nu and J tilde the initial policy how do you choose the initial policy okay in principle it can be anything it can be any policy in practice you may have some good policy some reasonable policy that you may be able to implement and calculate the corresponding R [Music]

in optimal control ok nocturnal control you mean how do we get started not know control how do we find an initial policy not know control ok by optimal control I think you mean a continuous time system described by differential equations in how do you find a feedback function to start with that's what that's what I thought nope that's what the policy is right so ordinarily enough no control the most important thing is to have a stable policy a policy that stabilized the system and gives you a stable differential equation that converges to some equilibrium in some way good or bad slow or fast but at least it converges and does not explode it's not unstable so you got to have to apply policy direction the context of control of this type you need to have an initial stable policy yes now the nice thing is here is we start with a stable policy policy duration ordinarily we'll give you another stable policy we'll give you a sequence of stable policies that's a consequence of the policy improvement property that each policy is better than the previous one this is true of course in in exact dynamic programming if you introduce approximations it is possible that you may start with a stable policy okay

I think what your ass is how do we find good features and how do we know that we have good features what happens if we missed some good features well if you miss some good features then you're going to get something worse than what when what we might have that whether the better features you have the better performance you will get and the more features you have the better performance you'll get at least in principle the however how do you get good features how do you get good features is mostly an art to a great extent it is an art and also a question of insight into your problem like for example if you ask a person who has never played Tetris to select good features for the tetris game he will not be very successful it's through some experience with a particular type of problem that you get insight about what are good features and what are not there are also some automatic methods for choosing features based on the performance of certain features you add some more features which is okay it's it's something that is a methodology that's still in the experimental phase and has not been fully established as far as I know if you okay so what you're asking is if you have a certain set of features and you're not sure if this is an ideal set okay you're looking for some more features if you add some more then we'll things get better is it possible the bacon get worse it's not very likely it will get worse the reason is the following if you have more features that means that you are expanding the subspace over which you're doing an approximation with fewer features you get only a subset of that subspace so if your approximation method is good getting giving yourself more options to approximate from should help you but of course this is under ideal circumstances and this there's no guarantee so it's principle it's it's possible in principle that you take a method you add more features and you're not doing better now it's possible that you that that the new features that you add are not very good you think they might be good but they are not very good also but it's also possible that so there is so how do you find good features it's really not a question that doesn't have a clear answer and as an example let me mention to you this Tetris problem the initial features chosen for this tetris problem gave a score of about fifty an average of points but per game when the features were increased to we got into the thousands now people added some more features and were able to get even higher with some more and some more they want to get a lot higher and we don't know if there are some hidden features that nobody has ever discovered yet but if you add them in you might not go into the hundreds of millions of score okay we just don't know that unless you have J star the optimal cost factor which is unattainable usually there's no way of telling how good your approximation is and whether you're using good features you need fixed point so we get the optimal just our given function goes high and means we get our feature selected not equally not yes also so that'll narrow function because Metin is known in your neck yeah I think you're making good points what you're saying is that one way to tell whether we have a reasonably good approach in a reasonably some reasonably good features is to look at the error that we're making in satisfying Bellman's equation if we had the ideal features that error would be identically zero however if we get an error that's large okay how can you tell it's large well maybe you can tell maybe not if that looks large than that that gives you the thought that you should be searching for some better features we can dynamically add some filtering system with coffee yes some people also have looked at approaches like that dynamically sparsity yeah okay again you're making very good points the you're saying that in machine learning there's a lot of interest in learning good features and identifying good features and perhaps some of this could be applied here I suspect that it can but it but it hasn't been done yet and you mentioned the methodology of compressed sensing where you you want to you have you have a certain parametric model that has a lot of weights and you know that some of them are should be and you're looking for a metric parameterization that has only few nonzero weights and relatively few features and compressed sensing allows you to do that by adding to the bellmen error function little squares from objective if you are beyond norm yes you're right if you add the l known here then you will get compressed sensing type of effect you'd get sparse solutions to my knowledge nobody has tried that look at Belmont error methods with with l norm has not been done it it should be doable but has not been done to it as far as right now maybe someone can do a thesis on it certainly a worthwhile topic because it allows you to not to worry too much about the correct features but just put many features in there and then the the algorithm will figure out which ones are important and which are not any other questions I am I'm afraid that the the clarity is not very good though that of the type and I don't know what to do about maybe maybe in the next lecture we'll improve it but I'm having a hard time reading it and I'm standing in front of the first Rojo okay maybe we can fix it while we take a break let's take a break for ten minutes and we'll be back okay