IEOR 265 - Lecture 1
Introduction to Dynamic Programming

1 What is Dynamic Programming?

Inventory Control, Self-Driving Vehicles, Robotic Manipulation, are some examples of real-life applications that are, in essence, Dynamic Programming (DP) Problems. Regardless of their nature and particularities, those problems share a key common trait: The decision-making is done in stages and the outcome of each decision may or may not be fully predicted before next decision has to made. They all share the same "tension" (or "tradeoff") between decisions in the present and decisions in the future. Another key similarity is that DP problems can be formulated as a minimization of a suitable optimization problem. This formulation allow us to provide a mathematical framework to cost (or if we are maximizing, to reward) outcomes that align with our current interests in the particular application.

Let's begin with an example of a DP problem, a chess game strategy optimization (taken from [1]): A player is about to play a two-game chess match with an opponent and their goal is to maximize the winning probability. Each game can have one of two outcomes:

1. A win by one of the players (1 point for the winner and 0 for the loser)
2. A draw (0.5 point for each of the two players)

If the score is tied at 1-1 at the end of both games, the players will go to sudden-death, that is playing until one of them wins a game (thus ending the match). The player is considering two different strategies and they can choose which strategy to choose in each game (independent of the previous games). The strategies are as follows:

1. Timid play: By playing timid the player will draw with some probability $p_d > 0$, and they will lose with probability $(1 - p_d)$.
2. Bold play: By playing bold the player will win with some probability $p_w > 0$, and they will lose with some probability $(1 - p_w)$.

As can be observed by their respective strategies, timid play never wins, while bold play never draws. Again, the player goal is to select a sequence of strategies for each game in order to maximize the probability of winning the match. We note already an important feature of this problem: If the match goes to sudden-death, then the player must play bold, otherwise they have no chance of winning, while having the chance of losing.

Hence, we already solved part of the problem: If the game reaches sudden-death mode, then the player must play bold. Now, let's focus on the remaining part, the first two games of the match and it's two decisions. We can model this decision-making problem as a two-stage DP and we can schematically represent all the possible outcomes in the following figure:

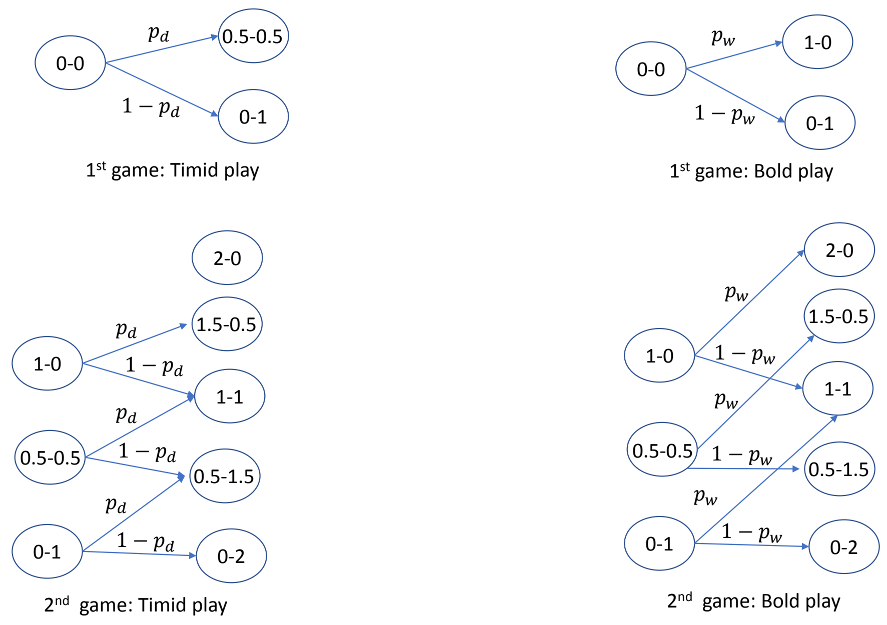

Figure 1: Schematic example of the chess match, where each node represent the score of the game, and the transitions are dependent on the action/decision of timid play or bold play.

Now, it seems reasonable to think that if $p_w < 0.5$, then the player have less than a 50-50 chance of winning the match, even playing optimally, since their probability of actually winning any single game is less than 50%, regardless of strategy taken. Surprisingly that is not the case! The reason for that is the third (and perhaps the most important) element of DP problems: Decisions can be taken in **closed-loop**, that is decisions can be adapted to the current state. In order words, instead of having a sequence of decisions, one can have a **policy** that specifies decisions given the states they found themselves in. The ideas of policy learning, in the Reinforcement Learning literature, closed-loop optimal control in Engineering, etc, all converge to the point of obtaining optimal (or close-to optimal) solutions to Dynamic Programming Problems. Back to the chess game, we will find out later that, with a closed-loop policy, the player is able to win more often than not (so better than 50-50) even if $p_w < 0.5$ (with the actual threshold depending on p_d).

2 DP Formulation

2.1 Key elements:

Now we will formalize the arguments made previously with appropriate mathematical elements. Throughout the course, we will usually assume the following (unless stated otherwise):

1. Each DP problem will have an underlying discrete dynamic system;
2. The objective function to be minimized is additive over different (discrete) time periods.

We define $x_k \in \mathcal{X}_k$ as being the state vector belonging to some space $\mathcal{X}_k$, where the subscript $k \in \{0, 1, ..., N\}$ indexes different time periods up to period N. We let $u_k \in \mathcal{U}_k$ be the control decision belonging to some space $\mathcal{U}_k$ and we will use a_k interchangeably with u_k, calling it the action decision; control/action decision represent the same object, but are called differently depending on the application. In addition, we will impose constraints on u_k, such that $u_k \in U_k(x_k) \subset \mathcal{U}_k$, where $U_k(x_k)$ is some subset of possible control decisions that depends on the current state x_k (think of as the allowable moves in chess given a particular board state).

Lastly we let $w_k \in \mathcal{W}_k$ be some random "disturbance" vector belonging to some space $\mathcal{W}_k$. Furthermore w_k follows some probability distribution $P_k(\cdot|x_k, u_k)$ that may depend on the state x_k and control u_k but it is independent of prior disturbances $w_{k-1}, ..., w_0$. The underlying DP dynamics is given by:

$$x_{k+1} = f_k(x_k, u_k, w_k), \forall k = 0, ..., N-1 \tag{1}$$

The objective (cost) function is defined by $g_k(x_k, u_k, w_k)$ and accumulates over time. Hence the total cost function is given by:

$$g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \tag{2}$$

Note that the above expression depends on the random vectors $w_0, ..., w_{N-1}$. So in order to define a proper objective function for optimization we will take the expectation of the above expression with respect to the joint distribution on the random vectors:

$$\mathbb{E}\big[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\big] \tag{3}$$

With the above function we can write the following optimization problem:

$$\min_{(u_0,...,u_{N-1})\in U} \left\{\mathbb{E}\big[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k)\big]\right\} \tag{4}$$

where U is the set of all feasible control **sequences**. The optimization problem above, once solved, provide us a sequence of decisions $u^* = (u_0^*, ..., u_{N-1}^*)$. As we discussed in the chess match example, the central aspect of DP problems is to find optimal **policies** that provide the best possible action given any conceivable state of the system. Formally, we can define a policy $\pi = \{\mu_0, ..., \mu_{N-1}\}$ as mapping from $\mathcal{X}_k$ to $U_k(x_k)$, such that each μ_k maps the state x_k into the control $u_k = \mu_k(x_k)$. We call such policies **admissible**.

Hence, given the initial system state x_0 and an admissible policy π, we can write the following expect cost function:

$$J_\pi(x_0) = \mathbb{E}\big[g_N(x_N) + \sum_{k=0}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\big] \tag{5}$$

Then we can define the optimal policy π^* as the solution of another optimization problem as follows:

$$J^*(x_0) = \min_{\pi \in \Pi} \big\{J_\pi(x_0)\big\} \tag{6}$$

where Π is the set of all admissible policies and we call $J^*(x_0)$ as the optimal **value function**. Now, observe that π^* is the optimal closed-loop policy, while u^* is the optimal control sequence, or as is more commonly called the optimal open-loop control sequence. The distinction between open-loop and closed-loop solutions is essential in DP, as it reflects the **value of information**, that is, by observing the current state x_k it may be possible to use such information in order to adapt/adjust our decision-making in order to further reduce the cost function. We will illustrate such differences when we return to the chess match example.

2.2 Alternative formulation: Markov-Decision Processes (MPD's):

One alternative for describing DP's is by utilizing Markov-Decision Processes (MDP). This formulation is particularly useful when the underlying DP states are discrete and finite (that is, the different system states take values from a discrete set, like the integers). In these situations, we can define the transition probability from a state i to state j given that we take action u at stage k as follows:

$$p_{i,j}(u,k) = \Pr(x_{k+1} = j | x_k = i, u_k = u) \tag{7}$$

where the **Markov Property** can be seen explicitly: The transition to the next state only depends on the current state and the chosen control/action.

Formally, an MDP is typically composed by a tuple $(\mathcal{X}, \mathcal{U}, P_u, R_u)$, where $\mathcal{X}$ is the set of discrete states, $\mathcal{U}$ is a set of discrete control decisions, P_u the transition probabilities that depend on the control u, and the "reward" R_u which is a function of the reward accumulated for each transition given the control u. To see how the MDP formulation connects to our DP formulation, observe we can write the following dynamic function:

$$x_{k+1} = w_k \tag{8}$$

and have:

$$\Pr(w_k = j | x_k = i, u_k = u) = p_{i,j}(u,k) \tag{9}$$

and we can let $R(i,j,u,k) = g_k(i,u,j)$, where we explicitly include the time dependence index k. The converse (from DP to MDP) can be easily achieved by similar arguments.

The connection between DP and MDP is important as well when we consider policies that are stochastic, that is, policies in which the selected control is randomized, and the policy is defined as a probability distribution over the possible controls, given the current state.

As an example, it is common in (Deep) Reinforcement Learning to train Neural Network (NN) architectures to output Gaussian policies such that:

$$\mu(x_k) \sim \mathcal{N}(m(x_k), \Sigma) \tag{10}$$

so the NN is trained to map the current state x_k to the mean of a Normal Random Variable, and the actual control decision is randomly sampled from that distribution. We will discuss the merits of randomized policies against deterministic policies in the future lectures.

3 DP Algorithm

The fundamental idea of solving DP's lies is the famous principle of optimality, or as is most known, the **Bellman Principle** (in honor of Richard Bellman) and it states the following:

Proposition 1 ***(Bellman Principle)*** *Let $\pi^* = \{\mu_0^*, \mu_1^*, ..., \mu_{N-1}^*\}$ be an optimal policy for the DP problem, and assume that when using π^*, a given state x_i occurs at time i with positive probability. Consider the subproblem whereby we are at x_i and wish to minimize the* ***cost to go*** *from time i to time N:*

$$\mathbb{E}\big[g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k)\big] \tag{11}$$

Then, the truncated policy $\{\mu_i^*, ..., \mu_{i+1}^*, ..., \mu_{N-1}^*\}$ is optimal for this subproblem.

The key observation that can be derived from the Bellman Principle is that one can solve the DP problem by constructing the optimal policy "piece-by-piece": We first start with the last stage (the "tail" problem), and then we extend the optimal policy backwards, until reaching the initial stage. This backward propagation of the optimal policy lies in the center of the DP algorithm:

Proposition 2 ***(DP Algorithm)*** *For every initial state x_0, the optimal cost $J^*(x_0)$ of the basic problem is equal to $J_0(x_0)$, given by the last step of the following algorithm, which proceeds backward in time from period $N-1$ to period 0:*

$$J_N(x_n) = g_N(x_N) \tag{12}$$

$$J_i(x_i) = \min_{u_i \in U_i(x_i)} \big\{\mathbb{E}_{w_i}\big[g_i(x_i, u_i, w_i) + J_{i+1}(f_i(x_i, u_i, w_i))\big]\big\}, \forall i \in \{0, ..., N-1\} \tag{13}$$

where the expectation is taken w.r.t. the probability distribution of w_i, which depends on x_i and u_i. Furthermore, if $u_i^* = \mu_i^*(x_i)$ minimizes the right side of Eq. (1.6) for each x_i and i, the policy $\pi^* = \{\mu_0^*, ..., \mu_{N-1}^*\}$ is optimal.

Proof:

For any admissible policy $\pi = \{\mu_0, ..., \mu_{N-1}\}$, let's define the truncated policy $\pi^i = \{\mu_i, ..., \mu_{N-1}\}$ at every stage $i \in \{0, ..., N-1\}$. Then for each stage i, $J_i^*(x_i)$ is the optimal cost for the $(N-i)$-stage problem that starts at state x_i and time i, and ends at time N:

$$J_i^*(x_i) = \min_{\pi^i} \left\{ \mathbb{E}_{w_i,...,w_{N-1}} \left[g_N(x_N) + \sum_{k=i}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right] \right\} \qquad (14)$$

For $i = N$, we define $J_N^* = g_N(x_N)$. We prove the result by induction that the functions J_i^* are equal to the functions J_i generated by the DP algorithm (Eq. 13), so that for $i = 0$ we have the result.

For the base case, the result holds, as $J_N^* = J_N = g_N$. Assume that for some i and all states x_{i+1}, we have $J_{i+1}^*(x_{i+1}) = J_{i+1}(x_{i+1})$. Then by definition of the truncated policy we have $\pi^k = (\mu_k, \pi^{k+1})$ and we have for all x_i:

$$J_i^*(x_i) = \min_{(\mu_i, \pi^{i+1})} \left\{ \mathbb{E}_{w_i,...,w_{N-1}} \left[g_N(x_N) + g_i(x_i, \mu_i(x_i), w_i) + \sum_{k=i+1}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right] \right\} =$$

$$\min_{\mu_i} \left\{ \mathbb{E}_{w_i} \left[g_i(x_i, \mu_i(x_i), w_i) + \min_{\pi^{i+1}} \left\{ \mathbb{E}_{w_{i+1},...,w_{N-1}} \left[g_N(x_N) + g_i(x_i, \mu_i(x_i), w_i) + \sum_{k=i+1}^{N-1} g_k(x_k, \mu_k(x_k), w_k) \right] \right\} \right] \right\}$$

$$= \min_{\mu_i} \left\{ \mathbb{E}_{w_i} \left[g_i(x_i, \mu_i(x_i), w_i) + J_{i+1}^*(f_i(x_i, \mu_i(x_i), w_i)) \right] \right\} =$$

$$\min_{\mu_i} \left\{ \mathbb{E}_{w_i} \left[g_i(x_i, \mu_i(x_i), w_i) + J_{i+1}(f_i(x_i, \mu_i(x_i), w_i)) \right] \right\} =$$

$$\min_{u_i \in U_i(x_i)} \left\{ \mathbb{E}_{w_i} \left[g_i(x_i, \mu_i(x_i), w_i) + J_{i+1}(f_i(x_i, \mu_i(x_i), w_i)) \right] \right\} = J_i(x_i)$$

Q.E.D

(Remark: This proof is very closely related to [1], thus making the simplifying assumption that all the functions are well-defined and finite and the expectations are well-defined and finite for every admissible policy. For the full technical details we refer to [2])

This proof is interesting because it highlights the notion that $J_i(x_i)$ is the optimal cost for a $(N-i)$-subproblem that starts at x_i at stage i and ends at stage N. Hence, $J_i(x_i)$ is precisely the **cost-to-go** function at time i. Ideally, we would like to apply the above algorithm to all DP problems in order to obtain closed-form solutions. However, in practice, it is often not possible to analytically solve the DP (that is provide a closed-form for all the cost-to-go functions and optimal policy). Hence, we need to rely on numerical methods that carry out the optimization of the right-hand side of Eq.13, and that lead us to the central limitation, or challenge, of solving DP's: the **Curse of Dimensionality**. In words: The optimization on Eq.13 need to be carried out for every state x_i; then the state space must be discretized somehow, if it is not a finite set; the same applies for the control/actions. Hence the complexity of the problem scales with the discretization of state and action spaces. For complex problems (for example, the game of chess), that leads to an excessive amount of possible combinations. But this limitation is what gives birth to a myriad of different algorithms that attempt to overcome the curse of dimensionality and provide practical solutions to real-life DP problems. Our goal on the first part of the course is to precisely analyze and study those algorithms, which come from the fields of Optimal Control, Reinforcement Learning, and Stochastic Optimization.

3.1 Solving the chess match example

Now let's return to the game of chess described in Section I, and see how the DP Algorithm (Eq.12-13) can be used in this simple example to obtain the optimal policy. Recall that the player wishes to maximize their probability of winning a two-game match and has to decide whether to play bold or timid at each game. For our analyzes we will assume that $p_d > p_w$ (we refer to Section I for a detailed description of each type of play).

The first step of any DP analysis is to appropriately define what the states are. It is often that a proper modeling of the state space can simplify the problem tremendously. In this case we will define the state x_i to be the *net score* of the match at the beginning of the i-th game, that is the difference between points of each player (where 0 corresponds to a even score, and 1 means the player is winning by one point). We note that the first game is indexed by $i = 0$. By using DP formulation, we can write the cost-to-go at the i-th game as follows:

$$J_i(x_i) = \max \left[p_d J_{i+1}(x_i) + (1-p_d) J_{i+1}(x_i - 1), p_w J_{i+1}(x_i+1) + (1-p_w) J_{i+1}(x_i - 1) \right] \tag{15}$$

where we note that there are only two possible decisions:

1. Timid play: This keeps the score at x_i with probability p_d and changes x_i to $x_i - 1$ with probability $(1 - p_d)$
2. Bold play: This changes the score x_i to $x_i + 1$ with probability p_w and changes x_i to $x_i - 1$ with probability $(1 - p_w)$

Hence, the optimization of Eq.15 can be carried out by a simple comparison, where we play *bold* if:

$$p_w J_{i+1}(x_i + 1) + (1 - p_w) J_{i+1}(x_i - 1) > p_d J_{i+1}(x_i) + (1 - p_d) J_{i+1}(x_i - 1) \tag{16}$$

This can be simplified to the following policy:

$$\mu_i(x_i) = \begin{cases} \text{“play bold” : if } \frac{p_w}{p_d} > \frac{J_{i+1}(x_i) - J_{i+1}(x_i-1)}{J_{i+1}(x_i+1) - J_{i+1}(x_i-1)} \\ \text{“play timid” : otherwise} \end{cases} \tag{17}$$

Now, applying the DP algorithm, starting from $N = 2$ (so after the *end* of the second game) we write:

$$J_2(x_2) = \begin{cases} 1, \text{if } x_2 > 0 \\ p_w, \text{if } x_2 = 0 \\ 0, \text{if } x_2 < 0 \end{cases} \tag{18}$$

Note that the above is consistent to our goal: If by the end of the 2nd game the score is negative, then we lost, so our probability of winning is zero; If the score is positive, then we won, so our probability of winning is one; lastly, if it is even, then we enter in sudden-death, and we have to play bold to win, so our probability of winning is p_w. Now let's execute the DP algorithm using the policy described in Eq.17 for every possible value of $x_1 \in \{-1, 0, 1\}$:

$$\begin{aligned} &J_1(1) = \max\left[p_d + (1-p_d)p_w, p_w + (1-p_w)p_w\right] \\ &= p_d + (1-p_d)p_w, \text{for } x_1 = 1; \text{ optimal decision: timid play} \\ &J_1(0) = p_w, \text{for } x_1 = 0; \text{ optimal decision: bold play} \\ &J_1(-1) = p_w^2, \text{for } x_1 = -1; \text{ optimal decision: bold play} \end{aligned} \tag{19}$$

We have the analytical forms for $J_1(x_1)$, now we proceed backwards to obtain $J_0(x_0)$ for $x_0 = 0$, since the match starts at even score. So we get:

$$J_0(0) = \max\left\{p_d p_w + (1-p_d)p_w^2, p_w(p_d + (1-p_d)p_w) + (1-p_w)p_w^2\right\} =$$

$$p_w(p_w + (p_w + p_d)(1-p_w)); \text{ optimal decision: bold play} \tag{20}$$

So at the start of the match, it is optimal to play bold. In addition we can summarize the optimal policy as follows: In the two-game match, the optimal policy is to play timid if and only if the player is ahead in the score.

Lastly, we can explicitly characterize the region in the probability space of (p_d, p_w) that the player has more than 50-50 chance of winning, that is the region such that $J_0(0) > 0.5$:

$$R = \left\{ (p_d, p_w) : \begin{cases} 0 \le p_d \le 1, 0 \le p_w \le 1 \\ p_w(p_w + (p_w + p_d)(1-p_w)) > 0.5 \end{cases} \right\} \tag{21}$$

The point $(p_d, p_w) = (0.833, 0.45)$, for example, lies in this space. This goes to show, that with a closed-loop policy the player can have more than 50-50 chance of winning, even if the probability of winning any individual game (p_w) is less than 50%.

References

[1] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas, *Dynamic programming and optimal control.* Athena scientific Belmont, MA, 1995, vol. 1, no. 2.

[2] D. P. Bertsekas and S. Shreve, *Stochastic optimal control: the discrete-time case*, 2004.