Tutorial Sections 7-8

1. In financial data analysis, data scientists are often tasked with analyzing historical stock market data to identify periods that would have maximized investment returns in specific stocks. This involves examining daily stock price changes (which could be positive, negative, or zero).

   Suppose you are analyzing a particular company to identify the time segment (i.e., a continuous sequence of days) where investing in this company's stock would have yielded the highest return. Consider a series of daily stock price changes, such as $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$.

   (a) Utilize the Bellman recursion defined by

   $$R_n^\star = \left(\max\left(T_n, R_{n-1}^{\star 1}\right), T_n\right),$$

   with base case $R_0^\star = (0, 0)$, where $T_n = \max\left(0, x_n + R_{n-1}^{\star 2}\right)$. Here, R^{*1} refers to the first element in a pair and $R^{\star 2}$ refers to the second element in a pair. Employ this recursion to find the time segment that offers the maximum return and calculate the maximum value for the stock within this timeframe.

   (b) What would be the computational complexity required to solve this problem using a fully exhaustive approach? Contrast this result with the recursive strategy outlined above.

Solution:

(a) This is a contextualized version of the Bellman recursion for solving the *maximum list segment sum* problem explored in the DP session. We can follow the principles of dynamic programming as applied to the maximum sum tail subsequence problem discussed in Session 5. In this case, the list represents the daily gains (or losses), i.e., each element x_n in the input list $[x_1, x_2, ..., x_9] = [-2, 1, ..., 4]$ represents the gain/loss for each day. The problem consists of finding the segment between the 9 days that yields the maximum sum (which, in the context of this problem, translates to finding the days in which holding the stock will produce the highest gain).

To solve the problem, we can use the Bellman recursion given above, in which $R_n^\star$ represents the optimal solution up to the day n. As it is written, $R_n^\star$ is a *tuple* that stores two numbers, which can be represented as $R_n^\star = \left(R_n^{\star 1}, R_n^{\star 2}\right)$. The second element, $R_n^{\star 2}$, is given by T_n, i.e., $R_n^{\star 2} = T_n = \max\left(0, x_n + R_{n-1}^{\star 2}\right)$, i.e., at step n, $R_n^{\star 2}$ builds on the previous value (from step $n-1$) by adding the value of the current element of the list, x_n, to it. Therefore, T_n represents the contribution of the current value of the segment to the current subsequence (note that if the addition is a loss, then we keep 0, meaning the start of a new subsequence at step n).

The first element, $R_n^{\star 1}$, compares the current maximum (computed in T_n) with the previous optimal solution (stored in $R_{n-1}^{\star 1}$). This comparison will return the maximum between these two values, so that the optimal solution at step n will be either the sum of the current subsequence or the optimal solution found in the previous step. This guarantees that the solution stored in the first element will be the optimal solution up to that point.

Once we understand the meaning of $R_n^\star$, we can find the segment by looking at how the maximum sum was built from the past iterations (since the recursion above does not explicitly store the segment). To find this solution, let's iterate:

$$n = 0 : R_0^\star = (0, 0) \text{ (initialization; max sum of an empty segment is 0)}$$

$$n = 1 : x_1 = -2, T_1 = \max\left(0, x_1 + R_0^{\star 2}\right) = \max(0, -2 + 0) = 0,$$

$$R_1^\star = \left(\max\left(T_1, R_0^{\star 1}\right), T_1\right) = (\max(0, 0), 0) = (0, 0)$$

Since the first element is a loss, we would better not consider it to the maximum sum and start a new segment at $n = 2$ (i.e., $T_1 = 0$ and $R_1^{\star 1} = (0,0)$). Continuing,

$$\begin{aligned}
n = 2 : x_2 = 1, T_2 &= \max\left(0, x_2 + R_1^{\star 2}\right) = \max(0, 1+0) = 1, \\
R_2^\star &= \left(\max\left(T_2, R_1^{\star 1}\right), T_2\right) = (\max(1,0), 1) = (1,1) \\
n = 3 : x_3 = -3, T_3 &= \max\left(0, x_3 + R_2^{\star 2}\right) = \max(0, -3+1) = 0, \\
R_3^\star &= \left(\max\left(T_3, R_2^{\star 1}\right), T_3\right) = (\max(0,1), 0) = (1,0) \\
n = 4 : x_4 = 4, T_4 &= \max\left(0, x_4 + R_3^{\star 2}\right) = \max(0, 4+0) = 4, \\
R_4^\star &= \left(\max\left(T_4, R_3^{\star 1}\right), T_4\right) = (\max(4,1), 4) = (4,4) \\
n = 5 : x_5 = -1, T_5 &= \max\left(0, x_5 + R_4^{\star 2}\right) = \max(0, -1+4) = 3, \\
R_5^\star &= \left(\max\left(T_5, R_4^{\star 1}\right), T_5\right) = (\max(3,4), 3) = (4,3) \\
n = 6 : x_6 = 2, T_6 &= \max\left(0, x_6 + R_5^{\star 2}\right) = \max(0, 2+3) = 5, \\
R_6^\star &= \left(\max\left(T_6, R_5^{\star 1}\right), T_6\right) = (\max(5,4), 5) = (5,5)
\end{aligned}$$

Continuing the iteration, we have:

$$\begin{aligned}
n &= 7 : x_7 = 1, T_7 = 6, R_7^\star = (6,6) \\
n &= 8 : x_8 = -5, T_8 = 1, R_8^\star = (6,1) \\
n &= 9 : x_9 = 4, T_9 = 5, R_9^\star = (6,5)
\end{aligned}$$

So, after analyzing the last element of the list, the current segment (considering the ninth element) yielded a gain of $R^{\star 2} = 5$. However, the maximum gain was $R^{\star 1} = 6$, obtained for an earlier subsequence that started at $n = 4$ and reached the maximum at $n = 7$. Therefore, the optimal segment is given by $[4, -1, 2, 1]$, meaning that we would have the maximum gain by buying the stock on day 4 and selling it by day 7.

(b) To solve the maximum sum subarray problem using exhaustive calculations, you would need to evaluate the sum of every possible subarray within the given array. For an array of length N, there are $\frac{N(N+1)}{2}$ non-empty subarrays, because for each starting position in the array (of which there are N), there are $N - i$ possible ending positions for the subarray starting at position i. To make it more clear, let's consider a sequence of stock price changes at different N days, $1, 2, ..., N$. Any subsequence can start at at any index i, so that $i \in [1, N]$. For each subsequence that starts at i, the end point j will be $j \in [i, N]$. Writing this possibility explicitly, we would have:

$$\begin{aligned}
i &\longrightarrow j = i \\
i &\longrightarrow j = i + 1 \\
&\vdots \\
i &\longrightarrow j = N.
\end{aligned}$$

So, there are a total of $N - i + 1$ subsequences that can be formed starting at i. Summing across all possible starting points i, we have:

$$\begin{aligned}
\sum_{i=1}^{N} (N - i + 1) =& (N - 1 + 1) + (N - 2 + 1) + ... + (N - N + 1) \\
=& N + (N - 1) + ... + 1
\end{aligned}$$

The sum of this sequence is given by $N(N+1)/2$. To see this result, let's call S_n as the right side of the sum. We can write S_n as two different ways:

$$\begin{aligned} S_n =& 1+2+3+...+(n-1)+n \\ S_n =& n+(n-1)+...+3+2+1. \end{aligned}$$

Summing the two equations, we would have that

$$\begin{aligned} 2S_n =& (n+1)+(2+n-1)+...+(n-1+2)+(n+1) \\ =& (n+1)+(n+1)+...+(n+1)+(n+1) \\ =& n\times(n+1), \end{aligned}$$

so that $S_n = n(n+1)/2$. Therefore, the number of subsequences for all possible starting points is $N(N+1)/2$.
However, calculating the sum for each subarray is not a single operation. To compute the sum of a subarray, you would have to sum each element within that subarray. In the worst case, for each subarray, you might end up summing N elements (for subarrays that span the entire array). Thus, the worst-case number of calculations for summing elements of all possible subarrays could be approximated to the sum of the first N integers times the sum of the first N integers, so that the total number of computations would be

$$N \times \frac{N(N+1)}{2},$$

which leads to a complexity of $O(N^3)$ for the fully exhaustive approach.
In the DP approach we used, each element in the array is processed exactly once in a single pass. For each element x_n, we perform the following calculations based on the Bellman recursion:

- Calculate $T_n = \max\left(0, x_n + R^{\star 2}_{n-1}\right)$.
- Update $R^{\star}_n = \left(\max\left(T_n, R^{\star 1}_{n-1}\right), T_n\right)$.

Each of these steps involves a constant number of operations (specifically, a couple of additions, comparisons, and assignments), regardless of the size of the input array. Therefore, the total number of calculations performed is proportional to the length of the input array, N. Given an array of length $N = 9$, and considering that each element involves roughly 2 primary calculations (for T_n and $R^{\star}_n$), the total number of calculations performed with DP can be estimated as $2N = 18$. This illustrates the efficiency of the DP approach, which operates with a linear complexity of $O(N)$, significantly reducing the computational effort compared to the exhaustive approach that has a cubic complexity of $O(N^3)$.

2. Working in a company, consider you were given a set of tasks, each with a start time, an end time, and its importance measured on a $0-10$ scale: As you can see, there is some overlap in the tasks, which

Task	Start Time	End Time	Importance
1	1	3	3
2	2	5	5
3	4	6	2
4	6	7	4
5	5	8	6

makes it not feasible to perform all tasks on a given day. We can assign a list segment, p, indicating the last non-conflicting task with the current one. For example, the segment $p = [0, 0, 1, 3, 2]$ implies that task 5 can follow task 2 (as they don't overlap), task 4 can follow task 3, and so on.

(a) Use the given recursion:

$$M_0^\star = 0$$
$$M_n^\star = \max\left(M_{n-1}^\star, M_{p_n}^\star + w_n\right)$$

to select a subset of non-overlapping tasks that maximizes the sum of their importance.

(b) What would be the computational complexity required to solve this problem using a fully exhaustive approach? Contrast this result with the recursive strategy outlined above.

Solution:

(a) This problem is a variant of a *subsequence* or *sublist* selection problem. The list p indicates whether items be selected with each other in a sequence. Each item is associated with a weight or value w_n. The problem is to select a subsequence of maximum weight sum, given the recursion above.
Considering the table given, the compatibility array $p = [0, 0, 1, 3, 2]$ indicates the last non-conflicting interval for each task. Applying the Bellman recursion given, we have:

$$n = 0 : M_0^\star = 0 \text{ (initialization, no intervals selected)}$$
$$n = 1 : M_1^\star = \max\left(M_0^\star, M_{p_1}^\star + w_1\right)$$

Since $p_1 = 0$ (first element of the list segment p), $M_1^\star = \max\left(M_0^\star, M_0^\star + w_1\right) = \max\left(0, 0+3\right) = 3$. Therefore, selecting task 1 alone provides a maximum weight (importance) of 3. If we continue the iteration,

$$n = 2 : M_2^\star = \max\left(M_1^\star, M_{p_2}^\star + w_2\right) = \max\left(3, 0+5\right) = 5$$
$$n = 3 : M_3^\star = \max\left(M_2^\star, M_{p_3}^\star + w_3\right) = \max\left(5, 3+2\right) = 5$$
$$n = 4 : M_4^\star = \max\left(M_3^\star, M_{p_4}^\star + w_4\right) = \max\left(5, 5+4\right) = 9$$
$$n = 5 : M_5^\star = \max\left(M_4^\star, M_{p_5}^\star + w_5\right) = \max\left(9, 5+6\right) = 11$$

Thus, the optimal solution is $M_5^\star = 11$ for the sequence $X_5^\star = [5, 2]$.

(b) To solve this problem exhaustively, we would need to consider all possible subsets of tasks to find the one that maximizes the sum of weights without any overlapping tasks. Given N tasks, there are 2^N possible subsets since each task can either be included or excluded from a subset. Therefore, the total computational complexity of the exhaustive approach is exponential, $O(2^N)$, making this problem intractable for large N. (In fact, it's even worse because, for each subset, we have to check if it consists of non-overlapping intervals, which can be done in $O(N \log N$ time by first sorting the intervals by their end times and then iterating through the subset to check for overlaps.)
The number of calculations to solve the problem using DP depends on the number of tasks, N, and the computation involved in updating the DP table based on the recursion formula. Considering the recursion given, for each task n from 1 to N, we perform one calculation to compute $M_{p_n}^\star + w_n$ and one comparison to determine the maximum between $M_{n-1}^\star$ and $M_{p_n}^\star + w_n$, which results in 2 calculations per task (+ 1 from the initialization step), resulting in $2N + 1$.

3. Suppose P and Q are the statements: P: "Jack passed math". Q: "Jill passed math".

(a) Translate "Jack and Jill both passed math" into symbols.

(b) Translate "If Jack passed math, then Jill did not" into symbols.

(c) Translate "$P \vee Q$" into English.

(d) Translate "$\neg(P \wedge Q) \implies Q$" into English.

(e) Suppose you know that if Jack passed math, then so did Jill. What can you conclude if you know that (i) Jill passed math? (ii) Jill did not pass math?

Solution: This is a very simple practice of logical operations.

(a) The statement can be rephrased as "Jack passed math" and "Jill passed math", which can be symbolised as $P \wedge Q$.

(b) The statement can be symbolised as $P \implies \neg Q$.

(c) The symbol $\vee$ means "or," so $P \vee Q$ translates to "Either Jack or Jill passed math".

(d) $P \wedge Q$ translates to "Both Jack and Jill passed math," so $\neg(P \wedge Q)$ means "it is not the case that both Jack and Jill passed math", and the whole statement means "If it is not the case that both Jack and Jill passed math, then Jill passed math."

(e) The information given in this item can be translated as $P \implies Q$.(i) Given this, and knowing Q is true (Jill passed math), we cannot directly conclude anything about P (whether Jack passed math) without additional information (since the conditional proposition will be true no matter if P is either true or false). (ii) If we know Q is false (Jill did not pass math), we can conclude that P must be false (Jack did not pass math) for the conditional proposition to hold true.

4. Consider the statement about a party, "If it's your birthday or there will be cake, then there will be cake."

   (a) Translate the above statement into symbols. Clearly state which statement is P and which is Q.

   (b) Make a truth table for the statement.

   (c) Assuming the statement is true, what (if anything) can you conclude if there will be cake?

   (d) Assuming the statement is true, what (if anything) can you conclude if there will not be cake?

   (e) Suppose you found out that the statement was a lie. What can you conclude?

Solution: This is a similar problem that works more with inference.

(a) Let P be the statement "It's your birthday" and Q be "There will be cake." The statement can be symbolised as $(P \vee Q) \implies Q$.

(b) The truth table for the statement is as follows:

P	Q	$P \vee Q$	$(P \vee Q) \implies Q$
T	T	T	T
T	F	T	F
F	T	T	T
F	F	F	T

(c) If there will be cake (Q is true), nothing can be concluded about P (whether it's your birthday). This is because the outcome of Q being true satisfies the implication regardless of the value of P.

(d) If there will not be cake (Q is false), then the only way the statement can be true is if $(P \vee Q)$ is false. Since Q is already false, P must also be false for $(P \vee Q)$ to be false. Therefore, it's not your birthday.

(e) If the statement is a lie, it means that $(P \vee Q) \implies Q$ is false. For the implication to be false, the antecedent $(P \vee Q)$ must be true, and the consequent Q must be false; this implies that P must be true (you can check this with the truth table above). Therefore, we can conclude that it's your birthday (P is true) and there won't be cake (Q is false).

5. Consider a community health program with the following facts:

    1. Individual is vaccinated
    2. Individual exercises regularly
    3. Individual is considered to have a high level of health

    Assume the program has the following rule: if an individual is vaccinated and exercises regularly, then they are considered to have a high level of health.

    (a) What is the proposition knowledge base for this case?

    (b) Does the knowledge base entail that if an individual does not exercise regularly, they are not considered to have a high level of health?

Solution: This is the same example of automated propositional reasoning from Section 8.3 of the notes, but in a different context.

(a) For this specific community health program, the **knowledge base (KB)** contains the following propositions:
V = 'individual is vaccinated;'
E = 'individual exercises regularly;"
H = 'individual has a high level of health,"
along with the rule R = 'if an individual is vaccinated and exercises regularly, they are considered to have a high level of health,' which can be represented as $R = (V \wedge E) \implies H$.

(b) The analysis objective is to determine if the knowledge base $KB = R$ entails whether $Y = \neg E \implies \neg H$. As in the example from the notes, there are 3 propositions, therefore we have $2^3 = 8$ possible models for these three binary propositions, which are given in the first 3 columns of the table below.

V	E	H	$R = (V \wedge E) \implies H$	$Y = \neg E \implies \neg H$
F	F	F	T	T
F	F	T	T	F
F	T	F	T	T
F	T	T	T	T
T	F	F	T	T
T	F	T	T	F
T	T	F	F	T
T	T	T	T	T

From the table above, we can see that the set of models where the KB holds is explicitly given by

$$M(KB) = \{[F,F,F],[F,F,T],[F,T,F],[F,T,T],[T,F,F],[T,F,T],[T,T,T]\}$$

In parallel, the set of models where Y holds (last column of the table) is given by

$$M(Y) = \{[F,F,F],[F,T,F],[F,T,T],[T,F,F],[T,T,F],[T,T,T]\}$$

By comparing the sets, we can easily see that $M(KB) \not\subseteq M(Y)$ because there are elements in $M(KB)$ that are not in $M(Y)$ (i.e., R is true in cases where Y is not). So, $KB \models Y$ is false, i.e., the KB does not entail that if an individual does not exercise regularly, they are not considered to have a high level of health.

6. Consider a neighbourhood safety program with the following facts and rule:
    1. Neighbourhood watch is active
    2. Street lighting is adequate
    3. Neighbourhood is safe
    4. If the neighbourhood watch is active and street lighting is adequate, then the neighbourhood is considered safe.

    Does the knowledge base entail that if the neighbourhood is not safe, then either the neighbourhood watch is not active or the street lighting is not adequate?

Solution: Again, we can use logical inference to solve this problem. The KB consists of the following propositions:
N = 'Neighbourhood watch is active'
L = 'Street lighting is adequate'
S = 'Neighbourhood is safe,'
along with the rule $R = (N \wedge L) \implies S$ (if the neighbourhood watch is active and street lighting is adequate, then the neighbourhood is considered safe).
The goal is to determine whether $KB \models Y$, where $Y = \neg S \implies (\neg N \vee \neg L)$.
Again, there are 2^3 possible combinations of truth values:

N	L	S	$R = (N \wedge L) \implies S$	Y $= \neg S \implies (\neg N \vee \neg L)$
F	F	F	T	T
F	F	T	T	T
F	T	F	T	T
F	T	T	T	T
T	F	F	T	T
T	F	T	T	T
T	T	F	F	F
T	T	T	T	T

(Note: if you are still being familiarised with propositional calculus, you may want to break R and Y into smaller steps to really see their outcome.)
Here, the set of models where the KB holds is given by

$$M(KB) = \{[F, F, F], [F, F, T], [F, T, F], [F, T, T], [T, F, F], [T, F, T], [T, T, T]\}.$$

We can see from the table that Y remains true in all cases where the KB is true,

$$M(Y) = M(KB) = \{[F, F, F], [F, F, T], [F, T, F], [F, T, T], [T, F, F], [T, F, T], [T, T, T]\},$$

so that $M(KB) \subseteq M(Y)$, therefore $KB \models Y$.
(In fact, the proposition Y is called the *contrapositive* proposition to R, which is a logically equivalent proposition to the original proposition. Therefore, it must be true whenever R is true.)