Markov assumption:

State the two Markov assumptions, and explain why they are important in the definition of Hidden Markov Models.

The two (first-order) Markov assumptions are:

- The next state is only dependent on the previous state.
- The observation is dependent only on the current hidden state

These two observations mean that the hidden state sequence displays the optimal substructure property – so we can employ dynamic programming to efficiently calculate the most likely sequence using the Viterbi algorithm – or the most likely state when using the Forward algorithm.

HMM Artificial data:

The data you were given with task 7 (parallel sequence of observations and states created by the "dice" HMM) was artificially created using an HMM (remember that we called HMMs and Naive Bayes **generative models**). In this exercise, you will explore how this was done.

1. What is the information you need in order to be able to design an algorithm for generating artifical data using an HMM?

   The transition probabilities and emission probabilities.

2. Describe an algorithm for creating artificial data.

   Begin in the start state.

   While you are not in the end state:

   Move to the next state and emit an emission with probabilities determined by the transition and emission probabilities.

   Either return the data in an appropriate format or save it to disk depending on the size of data we are creating.

```
def generate_data(n, start_state, end_state, start_emission,
end_emission, transition_probs, emission_probs):
    import random
    stateset = list({[state for transition in transition_probs
        for state in transition]})
    emissionset = list({map(lambda x: x[1], emission_probs)})
    transitions = {start: ([end for end in stateset], [
        transition_probs.get((start, end), 0) for end in stateset])
        for start in stateset}
    emissions = {start: ([end for end in emissionset], [
        emission_probs.get((start, end), 0) for end in emissionset])
        for start in stateset}
    sample_data = []
    for _ in range(n):
        hidden = [start_state]
        observed = [start_emission]
        state = random.choices(transitions[start_state][0],
                transitions[start_state][1])
        while state != end_state:
            state = random.choices(transitions[state][0],
                        transitions[state][1])[0]
            observation = random.choices(emissions[state][0],
                        emissions[state][1])[0]
            hidden.append(state)
            observed.append(observation)
        observed.append(end_emission)
        sample_data.append((hidden, observed))
    return sample_data
```

3. Transition probabilities into the final state are expressed as an extra parameter for an HMM. In some models these final transition probabilities are irrelevant. Under what circumstances would the prediction result be affected by transitions into the final state? Can you think of some examples of real world situations where this might happen?

   The transition into the final state is relevant when the probabilities of ending on different states have different probabilities. For example if we were speech tagging, the probability of ending on a conjunction is significantly less than ending on a noun.

   To illustrate this, I downloaded a text version of the "The Hobbit" and "The Lord of the Rings" by J.R.R.Tolkien and tokenised it using nltk. These are the results:

Type	Times Seen	Final Token	Final Probability
coordinating conjunction	32254	31	0.00096
cardinal digit	3372	161	0.048
determiner	58625	460	0.0078
existential there	2460	13	0.0053
foreign word	34	4	0.12
preposition/subordinating conjunction	71251	903	0.013
adjective (large)	31525	2206	0.070
adjective, comparative (larger)	1817	138	0.076
adjective, superlative (largest)	978	67	0.069
list market	8	3	0.38
modal (could, will)	10185	232	0.023
noun, singular (cat, tree)	70923	12312	0.17
proper noun, singular (sarah)	30689	5933	0.19
proper noun, plural (indians or americans)	245	43	0.18
noun plural (desks)	24991	3560	0.14
predeterminer (all, both, half)	957	61	0.064
possessive ending (parent's)	2791	3450	0.0022
personal pronoun (hers, herself, him, himself)	48886	3450	0.071
possessive pronoun (her, his, mine, my, our)	12341	16	0.0013
adverb (occasionally, swiftly)	40118	4211	0.10
adverb, comparative (greater)	1066	121	0.11
adverb, superlative (biggest)	146	5	0.034
particle (about)	3724	380	0.10
infinite marker (to)	12023	66	0.0055
interjection (goodbye)	459	78	0.17
verb (ask)	22263	1618	0.073
verb past tense (pleaded)	42149	1940	0.045
verb gerund (judging)	10149	684	0.067
verb past participle (reunified)	14527	1539	0.11
verb, present tense not 3rd person (wrap)	11408	536	0.046
verb, present tense with 3rd person singular	7858	243	0.031
determiner (that, what)	2281	5	0.0022
pronoun (who)	2347	25	0.010
adverb (how)	2679	17	0.018
Total	679628	41191	

In this circumstance we can see that final state transitions are very relevant. There is a factor of 200 difference between the probability of ending on a conjunction and ending on a proper noun!

```
import nltk
from tqdm import tqdm


endings = {}
totals = {}
with open('lotr.txt') as lotr:
        for line in tqdm(lotr.readlines(10000)):
                tokens = nltk.word_tokenize(line)
                tagged = nltk.pos_tag(tokens)
                for i in range(len(tagged) - 1):
                        totals[tagged[i][1]] = totals.get(
                        tagged[i][1], 0) + 1
                        if tagged[i + 1][0] in ('.', '!', '?'):
                                endings[tagged[i][1]] = endings.get(
                                tagged[i][1], 0) + 1
```

```
                    if tagged:
                        totals[tagged[-1][1]] = totals.get(
                        tagged[-1][1], 0) + 1
    print(totals)
    print(endings)
```

Smoothing in HMMs:

We did not smooth the Dice HMM in task 7 nor did you smooth the protein HMM in task 9.

1. In which situations can smoothing be counterproductive, and why?

   Smoothing makes models better when we have classes with zero or near-zero probabilities due to lack of data rather than a true zero probability. However with HMM's, there are some cases where we do not want this.

   - Sometimes for HMM's, it's easy to get large amounts of training data – for example if we have historical data, can generate data ourselves or can get data by web scraping or from a large online repository.

     If we have a large amount of training data and a relatively small amount of hidden states and classes of observations (such as in the dice databset), then we have a very low uncertainty in the transition and emission probabilities. Since we are so certain our probabilities are accurate, smoothing only skews the model marginally with no benefit. In this case we would not want to smooth.

   - Note that in HMM's we have "dummy" start and end states which should have emission and transition probabilities of 0 or 1. Smoothing would mean there was a nonzero probability of introducing a start or end state in the middle of the sequence – which is not possible.

   - Sometimes with things we model with HMM's, certain transitions or emissions are physically impossible. We want them to ahve a zero probability.

     For example if the loaded dice in task 7 did not have a 1, then we would never want our HMM to predict that the loaded dice rolled a 1. If we smoothed the probabilities then this impossibility observation becomes possible in our model.

2. In the case of the protein model, which of the two types of probability are better candidates for smoothing and why?

   We would want smoothing in a HMM if we have a large amount of states and emissions and so a high uncertainty on the transition and emission probabilities. Particularly if there are real sequences of observations where every hidden-state sequence has zero probability.

   If we have n hidden states and m emissions, then the number of transition probabilities we have is n^2 and the number of emission probabilities we have is nm. Since $n > m$ in the protein model, I would suggest smoothing the transition probabilities since they have the highest uncertainty.

Veterbi and Forward algorithm:

Study the Forward algorithm in the Jurafsky and Martin textbook. This is the algorithm for estimating the likelihood of an observation. It is another instance of the dynamic programming paradigm.

1. Give and explain the recursive formula for this dynamic programming algorithm in terms of a_{ij} and $b_i(o_t)$.

   Let $P(i_t)$ be the probability of being in state i at time t.

$$P(j_{t+1}) = b_j(o_{t+1}) \sum_{i=0}^{n-1} P(i_t) \cdot a_{ij}$$

   The probability of being in state j at time $t+1$ is equal to the sum of the probabilities of being in state i at time t and going from state i to j multiplied by the probability of state j emitting the observation seen at time $t+1$.

   Note that the base case is that the probability of being in the start state at $t = 0$ is 1.

2. Explain why there is a summation over the paths.

   We are trying to work out the state which is most likely at a given time. This can be from any path. So we have to sum over the endpoints of all possible paths.

Parts of Speech tagging with HMM:

Hidden Markov Models (HMM) can be used for **Part of Speech Tagging**. This is the task of assigning parts of speech, such as **verb**, **noun**, **pronoun**, determiner to words in a text sample.

A particular HMM is defined as follows: $S_e = \{s_1 = \text{verb}; s_2 = \text{noun}, s_3 = \text{personal pronoun}, s_4 = \text{auxiliary verb}\}$; s_0, s_F designated start state and end state.

$$A = \begin{bmatrix} a_{01} = 0.01 & a_{02} = 0.10 & a_{03} = 0.60 & a_{04} = 0.29 & \\ a_{11} = 0.02 & a_{12} = 0.63 & a_{13} = 0.07 & a_{14} = 0.13 & a_{1f} = 0.15 \\ a_{21} = 0.49 & a_{22} = 0.20 & a_{23} = 0.10 & a_{24} = 0.01 & a_{2f} = 0.20 \\ a_{31} = 0.40 & a_{32} = 0.05 & a_{33} = 0.05 & a_{34} = 0.40 & a_{3f} = 0.10 \\ a_{41} = 0.73 & a_{42} = 0.01 & a_{43} = 0.15 & a_{44} = 0.01 & a_{4f} = 0.10 \end{bmatrix}$$

$$B = \begin{bmatrix} b_1(fish) = 0.89 & b_2(fish) = 0.75 & b_3(fish) = 0 & b_4(fish) = 0 \\ b_1(can) = 0.10 & b_2(can) = 0.24 & b_3(can) = 0 & b_4(can) = 1 \\ b_1(we) = 0.01 & b_2(we) = 0.01 & b_3(we) = 1 & b_4(we) = 0 \end{bmatrix}$$

The observation sequence O is the following: $O =$ We can fish

1. Consider the two state sequences $X_a = s_0, s_3, s_4, s_1, s_f$ and $X_b = s_0, s_3, s_1, s_2, s_f$. Which interpretations of the above observation sequence do they represent?

   The first sequence interprets the sequence as meaning that the people speaking are able to go fishing.

   The second sequence interprets the sequence as meaning that the people speaking are "canning" fish – ie they are putting the fish into cans.

2. Give the probabilities $P(X_a)$, $P(X_b)$, $P(X_a, O)$ and $P(X_b, O)$, and state which of these probabilities are used in the HMM.

$$\begin{aligned} P(X_a) &= a_{03} \times a_{34} \times a_{41} \times a_{1f} \\ P(X_a) &= 0.60 \times 0.40 \times 0.73 \times 0.15 \\ p(X_a) &= 0.02628 \end{aligned} \quad (1)$$

$$\begin{aligned} P(X_a) &= a_{03} \times a_{31} \times a_{12} \times a_{2f} \\ P(X_a) &= 0.60 \times 0.40 \times 0.63 \times 0.20 \\ p(X_a) &= 0.03024 \end{aligned} \tag{2}$$

$$\begin{aligned} P(X_a) &= a_{03} \times b_3(we) \times a_{34} \times b_4(can) \times a_{41} \times b_1(fish) \times a_{1f} \\ P(X_a) &= 0.60 \times 1 \times 0.40 \times 1 \times 0.73 \times 0.89 \times 0.15 \\ p(X_a) &= 0.0233892 \end{aligned} \tag{3}$$

$$\begin{aligned} P(X_a) &= a_{03} \times b_3(we) \times a_{31} \times b_1(can) \times a_{12} \times b_2(fish) \times a_{2f} \\ P(X_a) &= 0.60 \times 1 \times 0.40 \times 0.10 \times 0.63 \times 0.75 \times 0.20 \\ p(X_a) &= 0.002268 \end{aligned} \tag{4}$$

The HMM will use $P(X_a, O)$ and $P(X_b, O)$.

3. Demonstrate the use of the Viterbi algorithm for deriving the most probable sequence of parts of speech given O above. Explain your notation and intermediate results.

	State					
Word	s_0	s_1	s_2	s_3	s_4	s_f
	1	0	0	0	0	0
"we"	0	0.0001, s_0	0.001, s_0	0.6, s_0	0.29, s_0	0
"can"	0	0.024, s_3	0.0072, s_3	0	0.24, s_3	0
"fish"	0	0.155928, s_4	0.01134, s_1	0	0	0
	0	0	0	0	0	0.0233892, s_1

We can backtrack from the most likely end state. This gives that the most likely sequence is:

$$s_0 \longrightarrow s_3 \longrightarrow s_4 \longrightarrow s_1 \longrightarrow s_f$$

4. Does the model arrive at the correct disambiguation? If so, how does it achieve this? If not, what could you change so that it does?

This model did arrive at the disambiguation which I believe is most likely. Note that without further context or a labelling from the author, it is impossible to know what the "correct" disambiguation is – for all we know the author could work in a cannery.

It achieved this by using the transition and emission probabilities to work out the probability of arriving at every state at every word by it's most likely sequence. We keep references to which state precedes this state in its most likely sequence.

Since HMM's exhibit optimal substructure, we know that the most likely sequencce leading to the previous state is a subsequence of the most likely sequence leading to the end state.

We then used this fact at the final iteration: we take the state at the end of the path with the highest probability and work backwards. This will give us the path with the highest probability.

5. If a labelled sample of text is available, then the emission probability matrix B can be estimated from a labelled sample of text. Describe one way how this can be done.

We can estimate the probablility of a hidden state S emitting an observation O by dividing the total amount of times which the state S appeared.

6. The statistical laws of language imply that there is a potential problem when training emission probabilities for words. This problem manifests itself in the probability of the word *can* in the state sequence X_b from (a) above. What is the problem, and how could it be fixed?

   I had two ideas about issues with viterbi which relate to the statistical laws of language – unseen words (no dataset is large enough to contain every word you could possibly see by Heaps' Law); and the probabilities of most words occuring being very small by Zipf's law. I could not see any strong links between either of these and X_b so I tried to invent a tenuous one.

   Zipf's Law states that the frequency of the i$^{\text{th}}$ most common word is inversely proportional to its rank. This means that many less frequent words will have *very* low probabilities of being emitted by particular states. This can be problematic.

   Since we multiply probabilities (and all probabilities are positive and ≤ 1), we will end up with $\lim_{P(x_t)\longrightarrow 0}$ – meaning that either the uncertainty on values due to float precision will make the algorithm behave seemingly randomly or we will have to use such a high float precision that computation is intractable.

   Note that in X_b, the probability reached 0.002 after only three (moderately common) words! If we were to analyse a nontrivial example – say one page of text discussing a niche topic using this method then we would have paths with probabilities in the order of 1.36×10^{-350} (extrapolating our toy example gives this value – far smaller values would be expected in practice). Even for this example, I had to use a special library to get high enough precision. We would either have to use special libraries or all probabilities would round to 0, leaving us with a completely useless model.

   The way to solve this is to use log-probabilities. So rather than multiplying the original probabilities, we should sum the logarithmic probabilities. This ensures that the numbers we deal with are large enough to be represented and can be manipulated easily. Note also that the cost of multiplication is far higher than that of addition.

   Using this method, we can comfortably represent probabilities down to $2^{-10^{300}}$ without using any special libraries. This covers the whole sample space of possible inputs.

Viterbi with higher order HMMs:

Viterbi is a clever algorithm that allows you to process the input in time that is linear to the observation sequence. With a first order HMM, we keep N (number of states) maximum probabilities per observation at each step.

1. How many states do we need to keep for an N order HMM?

   If there are M hidden states then to keep an N order HMM we need M^N states.

   IE for a first order HMM we need M states, for a second order HMM we need M^2 states etc.

2. What are the implications for the asymptotic complexity of Viterbi?

   The complexity of Viterbi is $\Theta(LM^{N+1})$ where L is the length of the input.

1 2021 Paper 3 Question 8

You are a 22nd century historian researching the "FEE" (First Epidemic Era) of 2019–2025, for which records are patchy. You research which government policy was in place in any given week during this historic phase. Policies, in order of severity, are: No restrictions, Tier 1, Tier 2, Tier 3, and Lockdown.

https://www.cl.cam.ac.uk/teaching/exams/pastpapers/y2021p3q8.pdf

(a) From other historic sources, you know the following about sequences of policy levels: if you are in a given policy level, there is a 40% chance you will stay there, a 20% chance that you will be upgraded to the next-highest (more severe) level next week, and a 10% chance that you will be downgraded to the next-lowest (less severe) policy level. The background lockdown probability (which applies if nothing more informative is known about lockdown) is 10%. For each observation sequence, there is also a 5% chance of the sequence ending at any point. Transitions to any other policy level beyond those already described are equally likely. Observation sequences begin with each policy level at equal likelihood.

Using the information given above, construct the full transition probability table.

		Next State						
		Start	No Restriction	Tier 1	Tier 2	Tier 3	Lockdown	End
Current State	Start	0	0.2	0.2	0.2	0.2	0.2	0
	No Restriction	0	0.4	0.2	0.125	0.125	0.1	0.05
	Tier 1	0	0.1	0.4	0.2	0.15	0.1	0.05
	Tier 2	0	0.15	0.1	0.4	0.2	0.1	0.05
	Tier 3	0	0.125	0.125	0.1	0.4	0.2	0.05
	Lockdown	0	0.15	0.15	0.15	0.1	0.4	0.05
	End	0	0	0	0	0	0	0

(b) You want to estimate which policy was in place for the first six weeks of 2025, but unfortunately, the only information you have about this is a sequence of Covid case numbers for these six weeks:

$$[0-99], [0-99], [>200], [>200], [>200], [100-199] \tag{5}$$

You know that case loads are associated with policy levels as in the Table below. Describe how you can calculate the sequence of most likely policy levels for these 6 weeks, giving numbers for at least three steps of the calculation. Assume that all policies are equally likely in the week preceding the first week.

	No Restriction	Tier 1	Tier 2	Tier 3	Lockdown
0 - 99 cases	5%	20%	20%	50%	90%
100 - 199 cases	15%	40%	30%	40%	9%
> 200 cases	80%	50%	20%	20%	1%

We can calculate the sequence of most likely policy levels by building a hidden markov model with the transition probabilities found in part (a) and emission probabilities given above. Then perform the viterbi algorithm on it to find the most likely sequence of restriction levels.

To make the maths more understandable, I will use normal probabilities rather than log probabilities. In a real hidden markov model, we would either use log probabilities or normalise probabilities so the values do not tend to zero.

Let R denote the set of possible policies. Let R_n denote the probabilities of policies on the n^{th} week.

The probability of a given policy S on the $(k+1)^{\text{th}}$ week is equal to $P(o|S) \times \operatorname{argmax}_{r \in R_i}(P(r) \times P(r \to S))$ where $P(o|S)$ is the probability of emitting the observation seen on the $(k+1)^{\text{th}}$ week given the hidden state is S and $P(r \longrightarrow S)$ is the probability of transitioning from state r to state S. We start in the start state and at the end multiply the probability by the probability of a transition from that state to the end state.

When we have run out of all observations we take $\operatorname{argmax}_{r \in R_n} P(r)$ and work backwards using the pointers we have to the previous state in that sequence.

Applying this gives the following table:

In the table the following abbreviations are used:

Start $\to$ ST	No Restrictions $\to$ NR
Tier 1 $\to$ T1	Tier 2 $\to$ T2
Tier 3 $\to$ T3	Lockdown $\to$ LD
End $\to$ ED	$\times 10^n \to e^n$

Note that all probabilities are rounded to 3 S.F. for presentation – actual calculation was done at full precision.

Week	ST	NR	T1	T2	T3	LD	ED
0	1,	0,	0,	0,	0,	0,	0,
1	0,	$1.00e^{-2}$, ST	$4.00e^{-2}$, ST	$4.00e^{-2}$, ST	$1.00e^{-1}$, ST	$1.80e^{-1}$, ST	0,
2	0,	$1.35e^{-3}$, LD	$5.40e^{-3}$, LD	$5.40e^{-2}$, LD	$2.00e^{-2}$, T3	$6.48e^{-2}$, LD	0,
3	0,	$7.78e^{-3}$, LD	$4.86e^{-3}$, LD	$1.94e^{-2}$, LD	$1.60e^{-3}$, T3	$2.59e^{-3}$, LD	0,
4	0,	$2.49e^{-3}$, NR	$9.72e^{-4}$, Tier 1	$1.94e^{-3}$, NR	$1.94e^{-4}$, NR	$7.78e^{-6}$, NR	0,
5	0,	$7.96e^{-4}$, NR	$2.49e^{-4}$, NR	$6.22e^{-5}$, NR	$6.22e^{-5}$, NR	$2.49e^{-6}$, NR	0,
6	0,	$4.78e^{-5}$, NR	$6.37e^{-5}$, NR	$2.99e^{-5}$, NR	$3.98e^{-5}$, NR	$7.17e^{-6}$, NR	0,
7	0,	0,	0,	0,	0,	0,	$3.19e^{-6}$, T1

By this table we can conclude that the last state in the most likely sequence was "Tier 1".

We can repeatedly backtrack to find the states in the most likely sequence.

Week	1	2	3	4	5	6
Policy	Lockdown	Lockdown	No Restrictions	No Restrictions	No Restrictions	Tier 1

(c) In which respects is the modelling described above not fully adequate to describe an actual epidemic?

I don't really know where to begin. Put simply this is an *awful* model and basically no part of it is suitable when scrutinized.

I saw six reasons why the modelling is not adequate to describe an actual epidemic:

- The first-order markov assumption is not suitable since it takes more than one week for restrictions to have any affect on case numbers.

  Coronavirus and other viruses have incubation periods – this means that there is a lag between infection and recorded cases. As a result it takes more than one week for any restrictions to be reflected in cases.

- The next state is not only dependent on the previous state but also on the previous *observations*.

  Restrictions are implemented in *response* to case numbers, which change as a result of restrictions. This is not a dependency relationship between case numbers and restrictions and so is unsuitable as a hidden markov model.

- The emission probabilities are unrealistic.

  I couldn't tell if this was intentional or if the question is just dated but it was so obvious to me that I had to mention it – historically lockdowns are implemented in response to high case numbers – so high case numbers would indicate that there was a lockdown. Not the other way round as this suggests.

- Transition and emission probabilities are not constant for different times and different countries.

  As new variantes emerge, testing programs get better and vaccination rates get better, the meaning behind case rates changes. IE in the first lockdown we had far fewer recorded cases than we do now and yet the impact was far higher.

Additionally, different countries had different reasons for case rates. IE as an extreme example some island nations had low or zero case rates throughout despite having no restrictions because of a closed-border policy. This model would predict they were in lockdown throughout.

- The hidden states are not well-defined.

  This model is meant to be generalised across different countries. Different countries had different approaches. IE New Zealand had very few domestic restrictions but a very tight travel policy throughout the pandemic – which category does this belong to?

  Some countries also had regional approaches. IE when the tiered approach was implemented and regions had restrictions ranging from tier 1 to tier 3 – what would the restrictions of the UK be?

- The observations are neither broad nor fine-grained enough and are not well-defined.

  The category 200+ is the overwhelming majority weeks in every country throughout the pandemic. As an example, Britain, France, Germany and Spain have all had 200+ cases in every single one of the 108 weeks since March 2020 – despite all four countries having varying levels of restrictions throughout.

  Additionally, there is a huge and important difference between 0 weekly cases and 1 - 99. This is not reflected in the model.

  The case rates are not specified to be “total cases” or “cases per 100k”. There is a very important distinction which is never explicitly stated. I have assumed these are total cases (as is implied) although the classes would match cases per 100k better.