Markov Chains: Teaching Probability to Remember

Musa Qamar

1. Law of Large Numbers

The Law of large numbers first discovered by Bernoulli in 1713, states that as the number of independent trials increases, the average result approaches the expected value. Let's illustrate this using a fair coin flip and note down the results.

Heads	Tails
7	3

Since the probability of heads is 0.5, we would expect a 50-50 split yet we observe a 70-30 split. If we however increase the number of trials to 1000 and repeat the experiment the results converge. This is far closer to the theoretical expectation. Therefore if you have truly independent events, their long-run averages will obey the Law of Large Numbers. This was the exact assumption probability theory had relied upon for 200 years until the arrival of Andrey Markov.

Heads	Tails
498	502

```python
import random
trials = 1000
head = 0
tail = 0

for x in range(trials):
    flip = random.randint(0,1)
    if flip == 0:
        head += 1
    else:
        tail += 1

print(f"Heads: {head}")
print(f"Tails: {tail}")
```

Markov was a socialist, atheist Russian mathematician whose intellectual rival was Nekrasov, a religious monarchist who supported the Tsarist regime. Nekrasov took Bernoulli's result and argued it in reverse: if a sequence of events obeys the Law of Large Numbers, then the underlying events must be independent. From this he claimed that mathematics could be used to prove free will since social statistics such as crime rates and marriage rates converge toward stable averages over large populations, the individual decisions producing them must therefore be independent. Markov however considered this reasoning deeply flawed and set out to disprove that the Law of Large Numbers could only

be observed if the underlying events were independent. In doing so Markov created an entirely new branch of mathematics.

2. Vowels and Consonants

To prove Nekrasov wrong Markov first had to find events which were dependent. For this he turned to language where the probability of a letter is clearly conditioned on what precedes it. For example, the probability of 'U' following 'Q' is near certain, while 'U' following 'Z' is rare. To prove this dependency Markov analysed 20,000 letters in a Russian poem “Eugene Onegin”. He simplified each letter into one of two states, vowel (V) or consonant (C), and observed the following frequencies

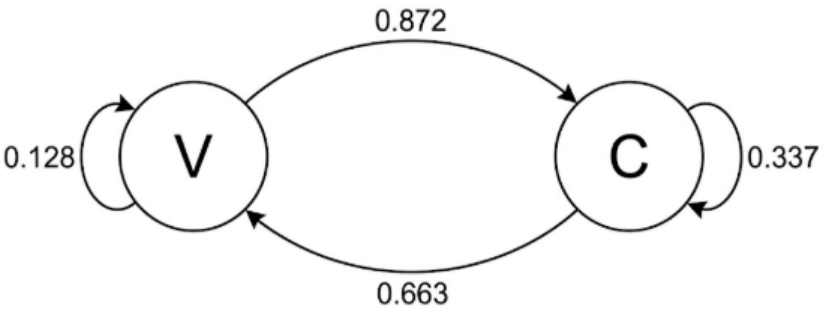
Vowel	8638	0.432
Consonant	11362	0.568

Markov broke the string of letters into pairs which gave him 4 different possibilities: Vowel-vowel, consonant-consonant, vowel-consonant, or consonant-vowel. If the letters were indeed Independent the probability of a Vowel-Vowel pair would be:

$$P(V) \times P(V) = 0.432 \times 0.432 = 0.186624 \approx 0.187$$

This gives us a 18.7% chance of a vowel vowel pair however when Markov actually counted the pair he found it showed up only 5.52% of the time. Doing this for other pairs he found that all actual values deviated from what independence would predict thus Markov had proved that the letters were indeed dependent.

All Markov had to do now was show that these letters followed the Law of Large Numbers.
To do this, he constructed a simple model with two states: V (vowel) and C (consonant). From either state, the system could transition to the other, or remain in the same one, represented by arrows connecting the states.To find out the probability of these transitions Markov turned to the frequencies he had painstakingly counted by hand. For instance, finding the transition probability of a vowel to another vowel, he divided the frequency of vowel-vowel pairs (5.52%) by the overall frequency of vowels (43.2%), yielding a transition probability of roughly 0.128. He repeated this calculation for every possible pair and completed his model.



The transition probabilities of this model can be represented using a stochastic matrix, also known as a markov matrix. In this matrix each row sums up to 1.

	To V	To C
From V	0.13	0.87
From C	0.67	0.33

With the model assembled, Markov picked a random starting state and let his machine run, generating a long string of C and V's . At first the split of C and V was random but as the machine ran the split converged to 43.2-56.8 the exact split Markov had counted by hand. The implication was devastating for Nekrasov. Markov had demonstrated that the Law of Large Numbers does not require events to be independent, it holds even when each outcome is influenced by the one before it. In dismantling Nekrasov's assumption, Markov had also built something new: the Markov chain.

3. Formalization

From the above experiment a key insight emerges: the probability of a vowel following a consonant is not affected by whatever came before that consonant. This is the Markov property, better known as the memorylessness of Markov chains. This is expressed through conditional probability.

$$P(X_{n+1} = x \mid X_0, X_1, \ldots, X_n) = P(X_{n+1} = x \mid X_n)$$

The L.H.S shows the probability of the future state X_{n+1} given all the previous states X_0, X_1,..X_n while the R.H.S shows the probability of the future state X_{n+1} given only the present state X_n.
As both sides are equal therefore the entire history collapses into a single condition, the present. Thus a Markov chain is a sequence of random variables X_1,X_2 ,X_3 where the future is independent of the past, given the present. To calculate the probability of a future state (X_{n+1}) only the current state (X_n) needs to be known. The entire history of how you reached the current state is irrelevant. For instance when you are on GO in monopoly, while calculating the probability of landing on the community chest you don't take into account the fact you were in Jail an hour ago. The only thing which determines your next position is your current position and the random roll of the dice. The history of how you got to that position is forgotten.

4. Applications of the Markov Chain

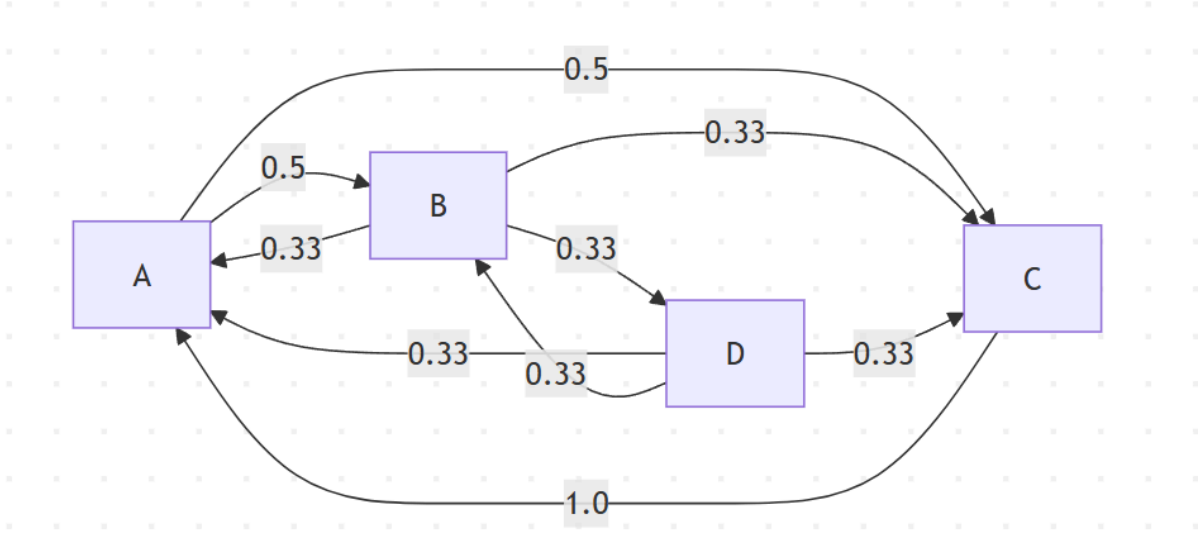
(i) PageRank

A single word search on Google returns billions of results in milliseconds. But how does Google decide which one deserves the top spot? The answer lies in an algorithm called PageRank. In the past browsers like yahoo or lycos used to display webpages in terms of relevance, meaning how many times a certain keyword was included in the webpage. However there was no way to check the quality of the webpage and developers could easily hide hundreds of keywords in white text.

To solve this problem PageRank was developed. PageRank sets up the webpages as markov chains. PageRank treats every webpage as a state and every hyperlink as a transition probability. If a page has three outgoing links, each link will carry a transition probability of ⅓. It then simulates a "random surfer" who starts on a random webpage and clicks links indefinitely. At each page they simply pick one of the outgoing links at random and follow it. As the surfer keeps clicking, the new system will eventually converge to a stationary distribution π.

This is solved using the equation $\pi P = \pi$ where P is the transition matrix of the web. The PageRank score of any webpage is simply π_i, the proportion of time the random surfer spends there in the long run. A page that many important pages link to will be visited frequently, and therefore receives a high π_i. A page with few or low-quality inbound links will be visited rarely, and ranks poorly.

For instance lets look at a Network with only 4 webpages and links between them as shown below:



	To A	To B	To C	To D
From A	0.0	0.5	0.5	0
From B	0.33	0.0	0.33	0.33
From C	1.0	0.0	0.0	0.0
From D	0.33	0.33	0.33	0.0

Using $\pi P = \pi$ we can compute the values of π_i for each state showing us which webpage would be ranked the highest. To calculate the probability for each state, we multiply the row vector by the corresponding column of the matrix:

$$\begin{bmatrix} \pi_A & \pi_B & \pi_C & \pi_D \end{bmatrix} \begin{bmatrix} 0 & 0.5 & 0.5 & 0 \\ 0.33 & 0 & 0.33 & 0.33 \\ 1 & 0 & 0 & 0 \\ 0.33 & 0.33 & 0.33 & 0 \end{bmatrix} = \begin{bmatrix} \pi_A & \pi_B & \pi_C & \pi_D \end{bmatrix}$$

For π_A:
$\pi_A = 0\pi_A + 0.33\pi_B + 1\pi_C + 0.33\pi_D$

For π_B:
$\pi_B = 0.5\pi_A + 0\pi_B + 0\pi_C + 0.33\pi_D$

For π_C:
$\pi_C = 0.5\pi_A + 0.33\pi_B + 0\pi_C + 0.33\pi_D$

For π_D:
$\pi_D = 0\pi_A + 0.33\pi_B + 0\pi_C + 0\pi_D$
By solving the following equations simultaneously we get the following values of π_i

Page	π_i	Rank
A	0.400	1st
B	0.302	2nd
C	0.224	3rd
D	0.074	4th

(ii) Text Predictions

Going back to the first markov chain we can use individual letters instead of only vowels and consonants as states. When this markov chain is run we get a series of gibberish texts but occasionally it forms short recognizable words. For example ‘of’ or ‘is’. However by expanding the number of states to include more than letters such as words and subwords collectively known as tokens the model is able to generate text in which strings of 3-4 words make sense. By increasing the number of states from 26 letters to thousands of unique tokens we can improve the probability of the markov chain and move from generating random letters to coherent and relevant text.

(iii) Hidden Markov Chains

Human speech, unlike text which has clear spaces, has messy and continuous vibrations. To combat this HMM is used. Because microphones pick up background noise like fans and echoes, a computer cannot get a "perfect match" for audio. Instead, HMM calculates probability. For example, in fuzzy noise, it asks if the person said "-it" or "-ic." By looking at the previous state "Ed-," it determines "-it" is statistically more likely.
Unlike text predictions which use tokens such as words and letters, HMM uses phonemes. HMM treats speech as a chain of states where each state only knows the one right before it. For e.g phoneme 1 -> phoneme 2-> phoneme 3.

5. Monte Carlo Method

Monte Carlo is a method which uses randomness to solve problems that might be deterministic in nature. For example if you want to find the area of a circle inside a 2 meter by 2 meter square and you have forgotten the value of π, you can throw a 1000 darts at it and count the number of darts that land inside the circle.(All darts land inside the square). Now you can find the area of the square by using the ratio of

For complex systems in the real world we often do not know the transition probabilities. However by using Monte Carlo method we can estimate these probabilities especially when the system is too complex. For example in Solitaire where the system is too complex due to there being 52! games it is difficult to estimate how many starting layouts will result in a win. However by simulating thousands of random games and keeping a count of how many layouts lead to a win versus a stuck state we can derive the transition probabilities for the overall Markov chain without needing to simulate all 52! Layouts.

6. Problem with Memorylessness and the Rise of Neural Networks.

The memorylessness nature of the Markov Chain gives rise to a major problem in language modelling: only the previous state is looked at and broader context is ignored. The probability of a person saying hospital is much higher if they spent the last minute talking about feeling ill even though their present state gives no indication of this. Additionally the probability of a person typing money after banks is much lower if they wrote about rivers and water in the previous paragraph. This problem is also present in PageRank. PageRank treats the user as a random surfer, however humans do not randomly surf the web. While searching humans have an intent, a student searching universities is far more likely to follow links related to admission or rankings than to click links about cooking recipes. The Markov chain however treats each user as a random surfer assigning transition probabilities on the basis of link structure rather than the underlying goal of the user. A markov chain by design is memory-less and discards everything except the present state and thus loses context.

Neural Networks initially tried to fix the memoryless nature of Markov chains by using Recurrent Neural Networks. RNN processes the inputs one step at a time but carries over a hidden state with all the previous inputs and not just the last input. This is like a person reading the book one word at a time but after each word he keeps a summary of all the

previous words he has read and in RNN case a hidden state containing information about all the previous states. While this gives RNN a sort of memory it suffered from the vanishing gradient problem. Taking the above example as the person would read more words in the book the summary in the person's head would become vague and blurry, especially the words in the beginning, causing him to forget how the book started. Hence even though it was designed to have memory it ends up acting as a Markov chain.

Transformers were the biggest breakthrough in neural networks as they eliminated the need for a step by step approach and instead used a parallel one. Instead of reading the words in a line, a transformer would look at every word in a paragraph and use an attention mechanism to decide how much attention should be given to each word. (How contextually relevant each word is). For example when talking about biology and bacteria in the beginning of the paragraph the neural network knows that the word cell used in the last sentence most likely refers to the basic unit of life rather than a prison cell. Thus allowing for words at the beginning of a page to connect to words at the very end without phasing through states and forgetting context.

7. Conclusion

What began as a political feud gave rise to one of the most consequential frameworks in modern mathematics. It gave a way for mathematicians to do probability on dependent events and laid the foundation of LLM’s we use today. Yet memorylessness, the property that makes Markov chains so elegant, is also what makes them contextually blind. It was this property that Transformers were built to fix, not by discarding the Markov chain but by finally giving it memory.