

ARTIFICIAL NEURONAL NETWORK

How AI can help us understand ourselves and the world around us

Mrwan Fahad A Alhandi
08/2023

Data Science Role: Chaos Theory, Randomness and AI

When we delve into any subject, patterns and regularities consistently appear. Every aspect of our world, from the workings of society and the nuances of our brain to the broader mechanisms of the universe, showcases this inherent order. One key to understanding different complex systems is focusing on how they are currently functioning. This is what we call systems patterns. The paradox is that when these patterns of relationship persist over time, then the system becomes relatively stable and thus to a certain extent simple. In other words, looking at patterns is a way to find out what is simple in a nest of complexity (Chanal, 2020). Does randomness exist, or we simply are limited to the number of attributes/knowledges that contributed to the initial conditions of a complex system? "Is Albert Einstein's famous statement, 'God does not play dice with the universe,' indeed accurate (Weisberger, 2019)?" Chaos theory describes the behaviour of certain dynamical systems that are highly sensitive to initial conditions (Frąckiewicz, 2023). By leveraging machine learning algorithms and advanced computational techniques, AI can help researchers analyse complex data sets and uncover hidden patterns that were previously undetectable. The position I aspire to occupy involves spearheading research at the intersection of chaos theory and artificial intelligence, striving to unravel the mystery of randomness.

As AI continues to evolve, it is increasingly being applied to complex systems. An example of how AI and Data Science contributed to this is related to Kuramoto-Sivashinsky equation. This is a fourth-order nonlinear partial differential that model the diffusive-thermal instabilities in laminar flame front. A machine learning algorithm called reservoir was computed to learn the dynamics of it. The algorithm knows nothing about the equation itself; it only sees data recorded about the evolving solution. This approach is powerful; in many cases, the equations describing a chaotic system aren't known. This is also called symbolic regression (Wolchover & Quanta, 2020).

My dream is to use AI to delve deeply into the human mind and the expansive mysteries of the universe. I seek insights from data to understand our cognition and celestial patterns. Beyond just making predictions, my goal is to unearth truths that remain obscured. As a dedicated student, I'm rigorously exploring AI methodologies and advanced data interpretation techniques. I aspire to use AI to fathom complex systems: from our brain's neural networks to the cosmos' celestial dance. This union of domains, powered by AI, captures my deepest aspirations.

Integrating neuroscience, astronomy, and AI in data science enriches analyses with varied perspectives, fostering comprehensive problem-solving. Techniques effective in one domain inspire solutions in another, as seen with AI's neural networks influenced by brain functions. Such interdisciplinary collaborations not only deepen insights but cultivate an inclusive environment, making projects more representative.

Data Set: Language Model AI Case Study

The first dataset can be accessed by clicking [here](#). The dataset contains the most common 32K names takes from [ssa.gov](#) for the year 2018. The second dataset is a concatenation of all Shakespeare work and it's roughly 1115394 characters. It can be accessed [here](#). Both datasets are less than 2 MB.

I am trying to understand how neural network which is the basis of AI works. The goal is to build an algorithm to generate new names derived from an existing list of names. This character-based language model operates by comprehending the probability distribution of all characters within the dataset. For instance, it will assess the likelihood of an 'e' following an 'a', thereby generating novel names. Subsequently, I aim to grasp the workings of a word-based language model and implement an algorithm for the same.

Data Analysis

To forecast the next character based solely on the preceding one, we must create a probability distribution indicating the likelihood of a subsequent character in a sequence. For instance, determining the likelihood that 'b' will come after 'a'. In achieving this, two primary methods can be employed. The first method involves leveraging bigram counts, while the second utilizes a neural network approach.

First approach: Bigrams Counts

In the initial methodology, it is imperative to construct a two-dimensional array of data type torch.int32 dedicated to capturing the bigram counts from the dataset, as illustrated in Figure 1. It is noteworthy that the first row signifies the frequency with which specific alphabets initiate names, while the first column represents the concluding character. Subsequently, a matrix of identical dimensions must be formulated to encapsulate the corresponding bigram probabilities. This can be accomplished by normalizing each bigram relative to the cumulative sum of its corresponding row.

0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	
0	1306	1542	1690	1531	417	669	874	591	2422	2983	1572	1146	394	915	92	1639	2055	1208	78	376	307	134	5	925	250	13	1	1	1	1	1	1	1	1	1	
1	556	541	470	1042	692	134	168	2332	1650	175	568	2528	1634	1575	63	82	60	124	1118	687	381	834	161	182	2650	43	1	1	1	1	1	1	1	1	1	
2	114	321	38	1	85	655	13	10	41	217	1	10	103	0	4	105	0	642	8	2	45	0	16	0	83	0	0	0	0	0	0	0	0	0	0	
3	97	815	0	42	1	551	0	0	0	0	0	116	0	0	380	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	516	1303	0	1	3	149	0	5	25	118	0	674	0	3	60	30	31	378	0	1	424	29	4	92	17	23	0	317	0	0	0	0	0	0	0	0
5	679	121	153	384	1271	82	125	152	618	55	178	25	769	2672	269	83	14	1556	861	580	69	463	50	132	1070	65	181	0	0	0	0	0	0	0	0	
6	80	242	0	0	0	123	44	7	1	160	0	0	2	20	0	4	60	0	0	114	0	18	10	0	0	0	0	0	0	0	0	0	0	0	0	0
7	108	350	0	0	19	334	0	1	25	360	150	0	0	32	6	27	83	0	0	201	30	31	85	1	26	0	11	0	0	0	0	0	0	0	0	0
8	2409	2244	8	2	24	674	0	1	1	729	0	29	105	117	138	187	1	1	204	31	71	166	39	10	213	20	18	0	0	0	0	0	0	0	0	0
9	2489	2445	110	0	509	440	1653	101	428	95	82	76	443	1345	427	2128	588	53	52	849	1316	541	109	269	8	89	779	27	0	0	0	0	0	0	0	0
a	71	1473	0	4	440	0	0	0	0	0	119	0	2	0	0	0	0	0	11	7	0	202	5	0	0	0	0	0	0	0	0	0	0	0	0	0
b	363	1731	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
c	1314	2623	52	25	138	2951	22	6	19	2480	0	24	1345	60	14	692	15	3	18	94	77	324	72	16	0	1588	0	0	0	0	0	0	0	0	0	
d	516	2550	0	112	51	24	818	0	5	1256	0	1	5	168	20	452	38	0	97	35	4	139	3	2	0	287	0	0	0	0	0	0	0	0	0	
e	363	1731	0	0	213	704	1520	11	273	26	1725	44	58	195	19	1906	496	5	44	278	443	96	25	11	6	465	145	0	0	0	0	0	0	0	0	
f	855	149	140	0	130	132	34	44	171	69	0	16	68	619	261	2411	115	95	3	1059	504	118	275	176	114	45	103	54	0	0	0	0	0	0	0	
g	33	209	0	0	0	197	0	0	204	61	0	1	16	1	1	59	39	0	151	16	17	4	0	0	0	0	0	0	0	0	0	0	0	0	0	
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
i	1377	2356	41	99	187	1697	0	76	121	5032	25	90	413	162	140	869	14	16	425	190	208	252	80	21	3	773	23	0	0	0	0	0	0	0	0	
j	1169	1201	21	60	9	884	0	2	2	1185	684	0	2	82	279	90	24	531	51	1	55	481	785	185	14	24	56	0	0	0	0	0	0	0	0	
k	483	1027	1	15	0	716	0	0	0	8	532	3	0	134	4	22	667	0	0	352	35	374	78	15	11	2	341	105	0	0	0	0	0	0	0	0
l	155	163	103	103	136	169	19	47	58	121	14	14	93	301	154	275	10	16	414	474	82	3	37	86	34	13	45	0	0	0	0	0	0	0	0	
m	88	642	0	1	0	1	0	0	0	911	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
n	51	280	0	0	0	149	0	0	0	148	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
o	164	103	1	4	5	36	0	0	1	102	0	0	0	39	1	1	41	0	0	31	70	5	0	3	38	30	19	27	0	0	0	0	0	0	0	
p	2007	2443	0	27	115	272	301	12	30	192	23	0	86	1104	14	1626	271	15	0	291	401	104	141	106	4	28	23	27	0	0	0	0	0	0	0	
q	160	860	0	0	0	373	0	0	0	43	364	0	0	2	123	35	0	0	0	2	4	2	0	1	147	45	0	0	0	0	0	0	0	0	0	

Figure 1. Bigrams counts in the dataset.

Now comes the modelling part. A model is said to learn from data if its performance on a given task improves after the data is considered (Deisenroth et al., 2021). The goal is to find

good models that generalize well to yet unseen data, which we may care about in the future. To further clarify, let's examine two distinct situations. In the first, our model doesn't use any specific "knowledge" from the data and assumes that all bigrams are equally likely. In the second, our model considers the bigram probabilities we previously calculated. Referring to Figure 2, it's evident that the names generated by the first model lack coherence, while those from the second model are notably more sensible.

```
A model that does not have knowledge about bigrams probabilities:
• juwjdvdpkcaz.
• p.
• cfqywocnzqfjiirtozcogsgwzvudlhnpuayjbilevhajkdbduinrwibtzsnjyievyvaftbzfvmumthyfodumjrpfytszwjhrjagq.
• coreaysezocfkyljabdywejmoifmwyfinwagaasnhsvfihofszxhddgosfmptpagicz.
• rjpiufmthdt.

A model that does have knowledge about bigrams probabilities:
• junide.
• janasah.
• p.
• cony.
• a.
```

Figure 2. Model Knowledge

But how good is the second model? We want a single number that represents how good the model is (training loss). If all bigrams are equally likely, $1/27 = 0.04$. This means that the chance of any next character is 0.04. Now, anything above 0.04 means that our model learned something new. A single instance from the dataset with its probabilities from the second model is given below:

Bigrams probabilities (logprob)		
.e:	0.0478	-3.0410
em:	0.0377	-3.2793
mm:	0.0253	-3.6753
ma:	0.3885	-0.9454
a.:	0.1958	-1.6305

Figure 3. Single instance probabilities and its log

To delve deeper into model evaluation, one must consider the product of probabilities across the dataset, termed as the 'likelihood'. This likelihood quantifies the probability assigned to the entirety of the dataset by our trained model, serving as an indicator of its quality. A heightened likelihood suggests that the model adeptly captures the underlying patterns of the data.

For computational simplicity and numerical stability, practitioners often resort to the log-likelihood. As illustrated in Figure 4, as the probability approaches unity, its corresponding log value tends toward zero. Given that a loss function conventionally conveys that a lower value is preferable, the negative log-likelihood is employed. Additionally, the mean negative log-likelihood can also be harnessed as a performance metric.

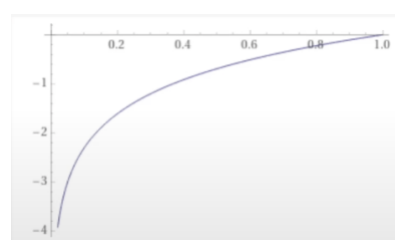


Figure 4. Log function

For the secondary model under consideration, the computed negative log-likelihood stands at 559,891.75, while its corresponding average negative log-likelihood is evaluated to be 2.45.

Second approach: Neural Network

The primary objective is to optimize the data model parameters by maximizing the likelihood in statistical modelling. This aim is congruent with minimizing either the negative log-likelihood or its average counterpart.

To design a neural network for this task, we first need to set up training and label datasets. These should be tensors containing the character indices. Using the name 'emma' as an example, sequences such as '.e', 'em', 'mm', 'ma', and 'a.' can be seen in the table below:

Training instance	Training indices	Label instance	Label indices
.	0	e	5
e	5	m	13
m	13	m	13
m	13	a	1
a	1	.	0

Table 1. Illustration of Training and Label Data Structures.

[illegible]

A neural network devoid of non-linear activation functions, often termed a linear neural network, can be characterized by the subsequent equation:

$$h_{W,b}(X) = XW + b$$

- X (training set) represents the matrix of input features.
- The weight matrix W contains all the connection weights except for the ones from the bias neuron.
- The bias vector b contains all the connection weights between the bias neuron and the artificial neurons.

In selecting W to be `torch.rand((27,27))`, where the `rand` function draws values from a standard normal distribution, we infer the following architectural layout for the neural network:

- The network comprises 27 input neurons.
- It further boasts 27 output neurons.
- There are no hidden/middle layers between input and output. Very simple neural network

- Each of these output neurons possesses 27 weights, each corresponding to an input neuron.
- Collectively, this configuration culminates in a comprehensive weight matrix containing $27 \times 27 = 729$ distinct weight values.

In our neural network model, our objective is to emulate the probabilities generated in the bigram model. Upon computing $h_{W,b}(X)$, which is often referred to as logits or log-counts, we obtain both positive and negative values. Ideally, these values should represent the probabilities of predicting the subsequent character. However, true probabilities are non-negative and sum to one, while counts are restricted to integer values. To convert these log-counts into meaningful positive decimals, we utilize an exponentiation process. As illustrated in Figure 5, when the values of $h_{W,b}(X)$ approach negative infinity, they converge towards zero. Conversely, as they approach positive infinity, their values increase indefinitely. Moreover, values close to zero result in an outcome nearing one.

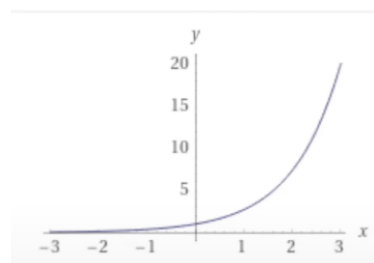


Figure 5. Exponential function

The exponentiated matrix is analogous to a matrix of synthetic counts. Each bigram is then standardized based on the cumulative sum of its associated row. This procedure is emblematic of the SoftMax function. As a result, we procure a probability distribution for the subsequent character given any input character. The sequence from encoding to generating these probability distributions is termed the 'forward pass'. The subsequent illustration depicts the processing of the first bigram of the word 'emma' within the neural network framework:

```
bigram example 1: .e (indexes 0,5)
input to the neural net: 0
output probabilities from the neural net: tensor([0.0607, 0.0100, 0.0123, 0.0042, 0.0168, 0.0123, 0.0027, 0.0232, 0.0137,
0.0313, 0.0079, 0.0278, 0.0091, 0.0082, 0.0500, 0.2378, 0.0603, 0.0025,
0.0249, 0.0055, 0.0339, 0.0109, 0.0029, 0.0198, 0.0118, 0.1537, 0.1459])
label (actual next character): 5
probability assigned by the net to the the correct character: 0.01228625513613224
log likelihood: -4.399273872375488
negative log likelihood: 4.399273872375488
```


Figure 6. A single bigram fed to the neural network.


Following the forward pass, we execute the backward pass, updating the parameters of W through gradient descent. With these parameter updates, the neural network achieves an average negative log-likelihood of 2.4, comparable to the first methodology. Notably, the neural network offers enhanced flexibility, allowing us to expand this approach to consider more than just a single character, potentially even entire words.


References

- Brown , W. (2021) *The universe organizes in a galactic neuromorphic network.*, *Unified Science Course*. Available at: <https://www.resonancescience.org/blog/The-Universe-Organizes-in-a-Galactic-Neuromorphic-Network> (Accessed: 16 August 2023).
- Chanal, V. (2020) *Systems patterns: A key to understanding complex systems*, *LinkedIn*. Available at: <https://www.linkedin.com/pulse/systems-patterns-key-understanding-complex-valerie-chanal/> (Accessed: 16 August 2023).
- Weisberger, M. (2019) 'god plays dice with the universe,' *Einstein writes in letter about his qualms with quantum theory*, *LiveScience*. Available at: <https://www.livescience.com/65697-einstein-letters-quantum-physics.html#:~:text=Einstein%20described%20his%20%22private%20opinion,prescribed.%22%20This%20variation%20clarified%20his> (Accessed: 16 August 2023).
- Frąckiewicz, M. (2023) *Ai and chaos theory, TS2 SPACE*. Available at: <https://ts2.space/en/ai-and-chaos-theory/> (Accessed: 16 August 2023).
- Wolchover, N. and Quanta Magazine moderates comments to facilitate an informed, substantive (2020) *Machine Learning's 'amazing' ability to predict chaos*, *Quanta Magazine*. Available at: <https://www.quantamagazine.org/machine-learnings-amazing-ability-to-predict-chaos-20180418/> (Accessed: 16 August 2023).
- Deisenroth, M.P., Ong, C.S. and Faisal, A.A. (2021) *Mathematics for Machine Learning*. Cambridge: Cambridge University Press.
- Social Security* (no date) *Popular Baby Names*. Available at: <https://www.ssa.gov/oact/babynames/> (Accessed: 16 August 2023).
- (No date) *GitHub*. Available at: <https://github.com/karpathy/makemore/blob/master/names.txt> (Accessed: 16 August 2023).
- (No date a) *GitHub*. Available at: <https://github.com/karpathy/ng-video-lecture/blob/master/input.txt> (Accessed: 16 August 2023).
- Andrej Karpathy. (2022, November). *The spelled-out intro to language modelling: building makemore* [Video]. YouTube. <https://www.youtube.com/watch?v=guRoWTYfxMs>

Generative AI




 What's chaos theory?





 Chaos theory is a branch of mathematics that deals with systems that appear to be disordered or random, but are actually governed by underlying patterns and deterministic laws. That is, in theory, their future behavior is fully determined by their initial conditions, with no random elements involved. However, in practice, these systems are so sensitive that small changes in initial conditions can lead to drastically different outcomes. This phenomenon is commonly referred to as the "butterfly effect."

Here are some key points about chaos theory:

1. **Deterministic yet Unpredictable:** While the systems described by chaos theory are deterministic (their future behavior is set by initial conditions), they can appear unpredictable because of their sensitivity to initial conditions.
2. **The Butterfly Effect:** This is a popular metaphor associated with chaos theory. It suggests that the flap of a butterfly's wings in Brazil could set off a tornado in Texas. While this is an exaggeration, it emphasizes the theory's focus on how small changes in a system's initial conditions can lead to dramatic differences in outcome.
3. **Fractals:** In chaos theory, many systems display fractal patterns, which are complex structures that look similar (but not identical) at any level of magnification.

 what was einstein thoughts about quantum mechanics?

 Albert Einstein had a complex relationship with quantum mechanics, the theory that describes the behavior of matter and energy on the smallest scales. Though Einstein made foundational contributions to quantum theory (most notably the paper on the photoelectric effect for which he received the Nobel Prize), he was deeply troubled by certain aspects of the evolving quantum framework, particularly as it moved towards what is now known as quantum mechanics or quantum theory.

Here are some main points regarding Einstein's views on quantum mechanics:

1. **Wave-Particle Duality:** Einstein's 1905 paper on the photoelectric effect provided evidence for the quantization of light, effectively showing light can be thought of as discrete packets (quanta) called photons. This work laid some of the foundation for quantum mechanics.
2. **God does not play dice:** As quantum mechanics was being developed, it became apparent that, at the quantum level, events are inherently probabilistic rather than deterministic. This troubled Einstein, who famously declared, "God does not play dice with the universe." Einstein's discomfort was not so much with the use of probabilities (after all, classical statistical mechanics uses probabilities) but with the idea that these probabilities might reflect a fundamental indeterminacy in nature.



why do we exponentiate log counts in neural network?



In neural network-based models, especially those used in tasks such as language modeling, machine translation, and speech recognition, it's common to see operations involving logarithms and exponentiation. The reason to exponentiate log counts (or generally, log probabilities) often comes from the mathematical properties and the computational advantages that these operations provide.

Here are some reasons why we use these operations:

1. **Stability and Precision:** Probabilities can be very small numbers. When multiplying many small numbers together (like when computing the joint probability of a sequence of words), there's a risk of numerical underflow. Working in the logarithmic domain turns these multiplications into additions, which are more numerically stable.
2. **Dynamic Range:** Probabilities vary over a limited range $[0, 1]$, but log probabilities can span a much larger dynamic range, from negative infinity to zero. This is particularly helpful when working with very small probabilities.
3. **Efficiency:** In the logarithmic domain, multiplication becomes addition, and division becomes subtraction. This can sometimes be computationally more efficient and less prone to errors.
4. **Softmax Operation:** In neural networks, especially in classification tasks, the softmax function is used to convert the final layer's logits (raw scores) into probabilities. The softmax function involves exponentiation. If you want to get the original logits back from the probabilities (or more accurately, their log probabilities), you'd need to take the logarithm.
5. **Connection to Information Theory:** Logarithms naturally appear in the context of information theory. For instance, entropy, cross-entropy, and the Kullback-Leibler divergence are all defined using logarithms. When neural networks are trained to minimize cross-entropy loss, they are effectively being trained to minimize the divergence between the predicted and the actual distribution, a concept deeply rooted in information theory.
6. **Gradient Amplification:** Exponentiating can amplify the gradients for values that are significantly off-target, potentially leading to faster learning. For instance, the softmax function combined with cross-entropy loss provides strong gradients for incorrect predictions, which can aid in faster convergence during training.

When you exponentiate log counts or log probabilities, you're essentially converting back from the log domain to the original domain. Depending on the context, this might be necessary, especially when the final result needs to be in the form of probabilities or counts rather than their logarithms.