



The next word

Description

“We are so in sync we finish each other’s sandwiches.” That’s a variation on a joke from several sources: *The Simpsons*, *Arrested Development*, *The Good Place*, etc.

In some social situations where I can’t think of anything appropriate to say, I just start a sentence anyway, not knowing what comes next or how it will end. I don’t think I’m alone in this unreflective, reckless, predictive text-like, practice. Supported by a suitably sociable context, language carries momentum, and we can rely on that to generate something to say.

Listening and reading also rely on such flow. It is obvious in the case of slow, formulaic speech-making, as delivered by Hakeem Jeffreys to the US Congress on 7 January 2023.

“House Democrats will always put:
American values over autocracy
Benevolence over bigotry
The Constitution over the cult
Democracy over demagogues
Economic opportunity over extremism
Freedom over fascism
Governing over ‘it’”

In this speech Jeffreys went through the entire alphabet, including “Maturity over Mar-a-Lago” and ending with “Zealous representation over a zero-sum confrontation.”

Once the attentive listener picks up on the pattern, it’s a matter of anticipating what the next bit of alliteration will deliver. Such anticipation heightens engagement, though anticipation operates in more prosaic language understanding as well. See posts tagged [interpretation](#).

Jeffreys’s speech is a good test for [OpenAI](#)’s (and ChatGPT’s) natural language processing (NLP) capability. The speech post-dates the creation of the platform’s learning corpus. So his speech was not part of the neural network training corpus. It is new.

I ran part of Jeffries's transcript through the OpenAI natural language platform to see if it could predict the next word. OpenAI came up with an incorrect but highly plausible word to follow "Governing over." It settled on "greed." Jeffries said "gaslighting," which is more pointed than "greed" as it latches onto a familiar talking point by the critics of the right wing.

OpenAI detected the alphabetical theme though and continued with "Human rights over hatred. Inclusion over intolerance," etc. Jeffries actually said "Hopefulness over hatred. Inclusion over isolation." (Presumably, part of the skill of speech making is to say something a little different to what the listener expects.)

To predict is to produce

In the world of automated NLP, **predicting** what comes next in a word sequence is the same as **producing** what comes next. (Predictive text in messaging apps and search engines deploys NLP.) Words flow from one another. Such prediction is explicit in the operations of NLP.

This post is more of a "thinking piece" than usual as I wrestle with NLP techniques and the implications for language, space, place, design, architecture, urbanism, and everything! Updates may follow.

A neural network (NN) consists of interconnected nodes arranged as an input layer, one or more hidden layers and an output layer. A **recurrent** neural network (RNN) follows a similar architecture, but during training, words (i.e. their index from a lookup table) in a sentence are presented as input to the NN one word at a time. During training, the first word in the sentence serves as input, with the second word as output. The weights and biases of the nodes in the network adjust to accommodate that binary sequence. The second word is then presented as input, with the third word as output. The process continues until the end of the sentence, or until the training algorithm encounters some end of sentence marker.

By the end of the sentence the network parameters hold a kind of "memory" of the word sequence. An RNN is able to thereby store information about the order of words (or tokens) in a block of text.

I'm trying in this post to bypass the mathematical explanation of the process. Chat GPT offers the best maths-free accounts I have found to date. Though ChatGPT is not 100% reliable when it comes to "facts" I assume that the OpenAI developers made sure it has been trained and tuned to deliver reliable information about NLP. I asked ChatGPT for further explanation.

In the context of training an RNN on thousands of sentences, the hidden state would be passed from one word in a sentence to the next word in the same sentence. Once the RNN has processed the last word in a sentence, the hidden state is reset and used to process the first word in the next sentence.

In other words, the hidden state is a kind of memory that the network uses to store information about the input it has previously seen.

I wasn't clear on how a *resetting* operation could preserve what has just been learned. I asked ChatGPT.

In the context of training an RNN on sequential data such as sentences, resetting the hidden state means clearing or initializing the hidden state with a specific value before processing the next sequence of data. This is important because the hidden state is used to capture information about the context of the input data, and if the hidden state is not reset between sequences of data, the network may use information from the previous sequence when processing the current sequence, which would be incorrect.

For example, if an RNN is trained on a dataset of sentences, after processing a sentence the hidden state is reset before processing the next sentence, so that the network doesn't use the context of the last sentence to process the current sentence.

That makes sense, suggesting that an RNN has different NN components. I asked it if a hidden state is the same as a hidden layer.

A hidden state and a hidden layer are related but they are not the same thing.

A hidden state refers to the internal state of an RNN that captures information about the context of the input data. It is a kind of memory that the network uses to store information about the input it has previously seen. The hidden state is passed from one step to the next, allowing the network to take into account the context of previous words when processing a new word.

A hidden layer, on the other hand, refers to a layer in a neural network that is not directly connected to the input or output layers. It is a layer between the input and output layers, which is used to extract features from the input data and pass them to the output layer. A hidden layer can be considered as a set of neurons that are responsible for some specific feature extraction from the input.

In RNNs, the hidden state is used as the recurrent connection between the time steps and the hidden layer is used as the standard feedforward connections between the input, hidden, and output layers.

I've placed further ChatGPT Q&A in the notes at the end of this post explaining that parameters are updated based on the error between the actual next word and the predicted next word.

Predicting text

I'll use a fragment of the speech above to rehearse the training process. Imagine a snapshot of the training for a large corpus of texts, perhaps several thousand political speeches entered one sentence at a time in any order. Eventually the RNN is presented with our training sentence that includes 'governing over gaslighting, human

The RNN doesn't process strings of text, but numbers in a lookup table contained in a lexicon, but here I'll illustrate with words (and ignore punctuation and capitalisation). By the time the network encounters this instance of "governing" it has already been trained on thousands of sentences, some of which include instances of "governing." So the parameters of the hidden layer in the network will already be populated with values for the nodes and the arcs between nodes within one or more hidden layers (depending on the network design).

The number of input and output nodes for the RNN will be the same as the number of words (or tokens) in the lexicon, which for a large corpus is likely to be over 50,000.

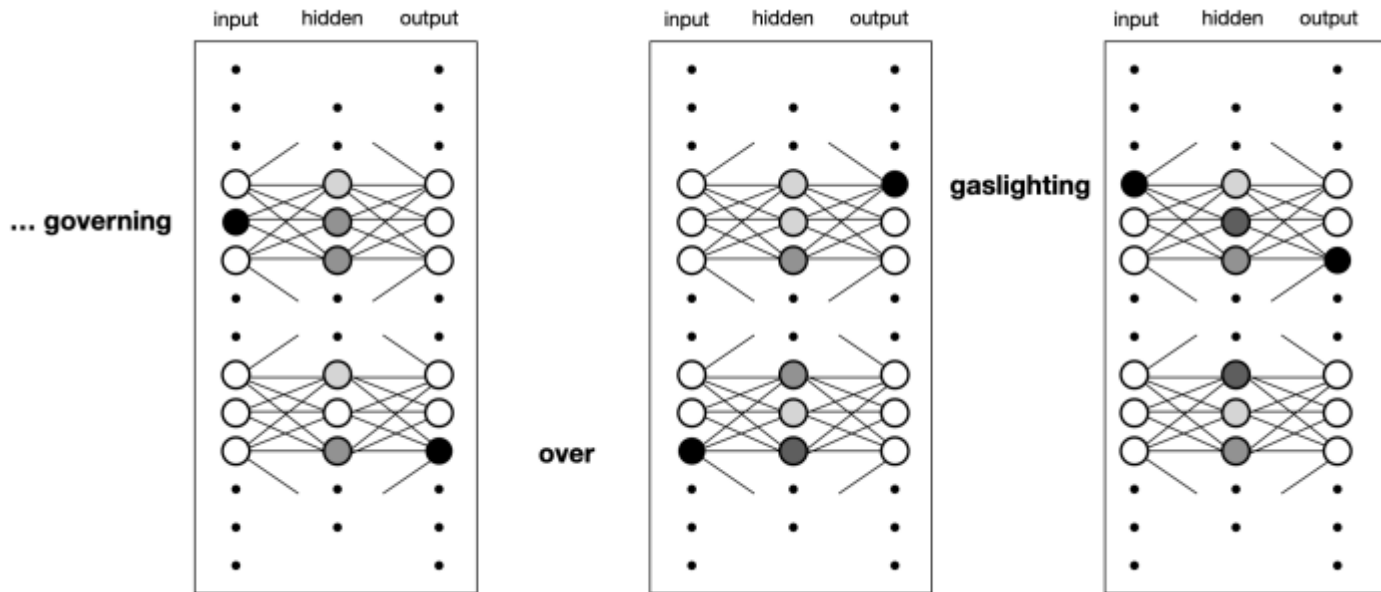
During training, the word "governing" activates the designated input node (value = 1). All the other input nodes will be 0. The desired output for that instance of the word is the next word in the sentence "over." The RNN algorithm adjusts its parameters to accommodate that training instance, while retaining its performance for all the other word sequences it was already trained on. (in a later post I'll explore "embedding," which involves more information-rich representations of words/tokens than index numbers in a lookup table.)

The next word in the sequence is "gaslighting" as it follows "over." The word "over" is allocated as input to the same network, with "gaslighting" as output. The network adjusts its parameters to accommodate that input-output pairing as well.

The word "gaslighting" is fed into the network following the same process. Its output is the next word in the sequence "human."

The training process continues. In most text book illustrations, the sequential training begins and ends with the start and end of each sentence. So, the the RNN retains information about the words in the current sentence. See the Notes for the question about what happens to network parameters for all the other sentences on which the network has been trained.

Here's my own attempt to draw the process. Each rectangle represents the state of the network each time it encounters a new word. At each state the output is the next word in the sequence, which is then the input for the next state.



The NN is thereby primed to complete sequences during run time. That does not mean it will automatically reconstruct sentences contained in its learning corpus, nor that every run time input has to start as a sentence. As I have shown, in the case where I fed ChatGPT the block of text ending with "governing over" it continued with the highly plausible word "greed" consistent, presumably, with its training corpus. The NN may result in more than one output, each with a certain probability. In my example, the word "greed" had a high probability. The RNN will not only produce single words that continue the sequence, but whole phrases and even paragraphs.

That's the basics of RNNs, though several challenges call for further sophistications to the model.

- The position of words and phrases in sentences matters, and from a grammatical point of view at least, words that come later in a sentence can influence what comes earlier. There needs to be some kind of training that also considers sentences from the back to the front as well as front to back.
- For sentences that are longer than a few words, the effect of earlier words can get diluted in the network. If OpenAI was not programmed to exhibit sufficient "short term memory" it could have neglected to consider the binary structure in the speech example above, and that the theme is *democracy*, and completed "governing over" with something like "all the people," which misses the point of the speech.
- I've assumed here that an RNN trains on single sentences, but it could do the same for paragraphs or any arbitrarily sized chunking of the training data.
- Scaling up the RNN process imposes costs in terms of processing time during training and run time, for which other efficiency strategies are required.

Here's another experiment I carried out with OpenAI, feeding it successive parts of the paragraph on a graffiti strategy for Edinburgh featured in a previous post.

Robust policies and procedures on Graffiti Management are key components of the future strategy, aiming to reduce instances of "tagging," while still providing space for the more creative elements and potential benefits of graffiti, street art, and murals for local

communities. This will also ensure that the city's residents and stakeholders are clear on the approach being taken by the local authority.

I started with "Robust" and watched OpenAI's predictions for what comes next, highlighted in yellow for each case. After each attempt by OpenAI to complete the text I substituted the actual words (in bold).

Robustness

Robust policies and

Robust policies and procedures are

Robust policies and procedures on data

Robust policies and procedures on Graffiti,

Robust policies and procedures on Graffiti Management,

Robust policies and procedures on Graffiti Management are designed

Robust policies and procedures on Graffiti Management are key components of the future strategy, aiming to reduce instances of graffiti

Robust policies and procedures on Graffiti Management are key components of the future strategy, aiming to reduce instances of tagging, while also promoting public art, education, and community involvement in graffiti

Robust policies and procedures on Graffiti Management are key components of the future strategy, aiming to reduce instances of tagging, while still providing space for the more creative elements and potential benefits of public art. Such policies could include targeted enforcement of existing laws, installing physical barriers to discourage vandalism, providing alternative outlets for creative expression, and implementing public education campaigns.

References

- Grinberg, Miguel. "The Ultimate Guide to OpenAI's GPT-3 Language Model." *Twilio blog*, 25 August, 2020. Accessed 25 January 2023. <https://www.twilio.com/blog/ultimate-guide-openai-gpt-3-language-model>
- Kostadinov, Simeon. *Recurrent Neural Networks with Python Quick Start Guide : Sequential Learning and Language Modeling with TensorFlow*. Birmingham, UK: Packt Publishing, 2018.
- Moroney, Laurence. "Training an AI to create poetry (NLP Zero to Hero Part 6)." *TensorFlow*, 2021. Accessed 21 January 2023. <https://www.youtube.com/watch?v=ZMudJXhsUpY>

Note

- Featured image generated by MidJourney prompted by "We are so in sync we finish each other's sandwiches; Simpsons"
- I asked ChatGPT: If during training the hidden state is reset after each sentence (or block of data), how then are its parameters retained in the machine learning process?

In the training process of an RNN, the hidden state is reset after each sentence (or block of data) to avoid the network from using information from the previous sentence to process the current sentence. However, the parameters of the RNN are updated during the training process and are retained after each iteration to capture the information about the input data and make predictions.

The parameters of an RNN include the weights and biases of the network, which are used to compute the hidden state and the output of the network. During the training process, the parameters are updated based on the error between the actual output and the predicted output, using an optimization algorithm such as stochastic gradient descent (SGD) or its variants.

For example, in a language modeling task, after processing a sentence, the hidden state is reset and the parameters are updated based on the error between the actual next word and the predicted next word. The updated parameters are then used to process the next sentence, and the process continues until all the sentences in the training data have been processed.

At the end of the training process, the parameters of the RNN are retained and can be used to make predictions on new data. These parameters represent the learned information about the patterns and relationships in the training data, and they can be used to generate text, translate text, or perform any other task that requires processing sequential data.

Category

1. Artificial Intelligence

Tags

1. natural language processing

Date Created

February 25, 2023

Author

rcoyne99