# HANGMAN GAME

## ALGORITHM

1. **Initial Guess Based on Letter Frequency:**
   - The algorithm starts by making its first guess based on the letter frequency within the word groups of the training set.
   - Word groups are created based on the lengths of words in the training set. For instance, if the training set includes words like ["abc", "de", "adf"], then we have two word groups: one for words with a length of 2 and another for words with a length of 3.
   - Within each group, the algorithm determines the letter with the highest frequency. For example, if "a" has the highest frequency in the length == 2 group, then "a" becomes the initial guess.
   - If the guessed letter is not present in the hidden word, the algorithm proceeds to guess the letter with the second-highest frequency, and so on, until a valid guess is made.

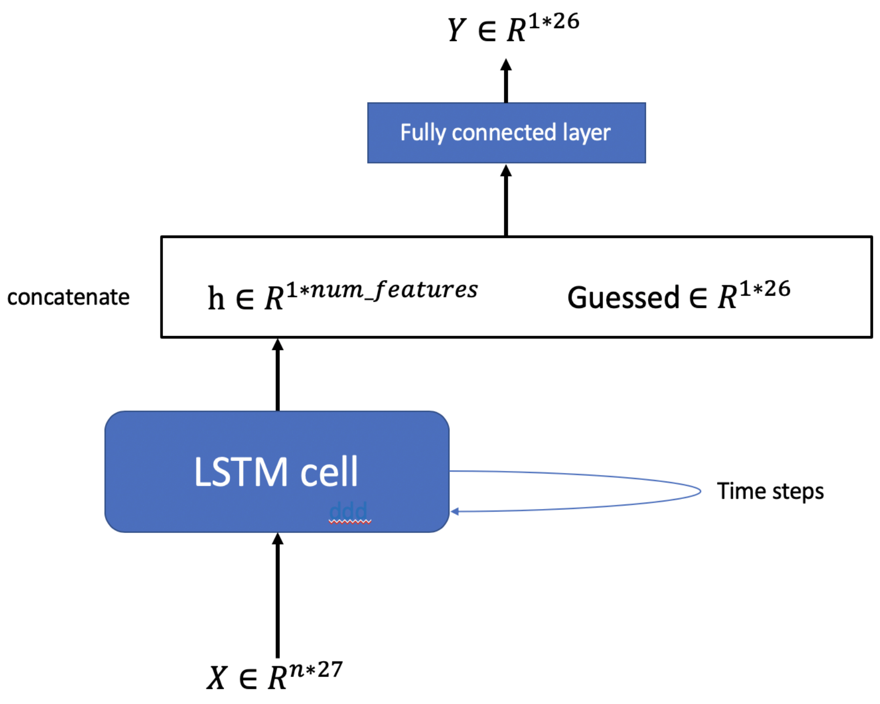
2. **Guessing Using 2-Gram Frequency:**
   - After the initial guess, the algorithm proceeds to guess other letters based on the frequency of 2-grams within the word groups.
   - Starting from the first letter guessed in the first step, the algorithm calculates the frequency of all 2-grams containing that letter within the respective word group.
   - It then selects the 2-gram with the highest frequency as the next guess.
   - This process continues until a valid guess is made.

3. **Utilizing LSTM Model for Predictions:**
   - In cases where no matched 2-gram is found, an LSTM (Long Short-Term Memory) model comes into play to fill in the remaining blanks.
   - The LSTM model is pre-trained to introduce uncertainty, allowing predictions beyond combinations seen in the training set.
   - The model's structure involves encoding the obscured word along with the already guessed letters as input.
   - The output is a 1 by 26 vector, with each element representing the probability of the next letter in the alphabet.

By following these steps, the Hangman game algorithm can intelligently make guesses based on letter and 2-gram frequencies, while leveraging an LSTM model to handle cases where predictable patterns are not found in the training data. This approach enhances the gameplay experience by introducing strategic and dynamic guessing mechanisms.

## ARCHITECTURE

n is the length of the word, time step is how many rounds we have guessed. The shape of X is 27 because other than 26 letters we need to preserve a space for "_".



## N-Grams

One way to estimate the probability of the next letter is from relative frequency counts, take the large corpus, count the number of times the letter already guessed followed by the next guessed letter basically means out of the times we saw the history h how many times was it followed by letter w by conditional probability. While this method of estimating probabilities directly from counts works fine in many cases, it turns out that even the dataset we provided isn't big enough to give us good estimates in most cases. This is because language is creative; new words are created all the time.

$$\begin{aligned} P(X_1 \ldots X_n) &= P(X_1)P(X_2|X_1)P(X_3|X_{1:2})\ldots P(X_n|X_{1:n-1}) \\ &= \prod_{k=1}^{n} P(X_k|X_{1:k-1}) \end{aligned}$$

Applying the chain rule to the letters, we get

$$\begin{aligned} P(w_{1:n}) &= P(w_1)P(w_2|w_1)P(w_3|w_{1:2})\ldots P(w_n|w_{1:n-1}) \\ &= \prod_{k=1}^{n} P(w_k|w_{1:k-1}) \end{aligned}$$

Equations suggest that we could estimate the joint probability of an entire sequence of letters by multiplying together a number of conditional probabilities. But using the chain rule doesn't really seem to help us! We don't know any way to compute the exact probability of a letter given a long sequence of preceding words, P(wn|w1:n−1).
The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.

## 2-Gram Model

The 2-gram model (Bigram model)  approximates the probability of a letter given all the previous words P(wn|w1:n−1) by using only the conditional probability of the preceding word P(wn|wn−1).
When we use a bigram model to predict the conditional probability of the next letter, we are thus making the following approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

The assumption that the probability of a letter depends only on the previous letter is Markov called a Markov assumption. Markov models are the class of probabilistic models that assume we can predict the probability of some future unit without looking too far into the past.

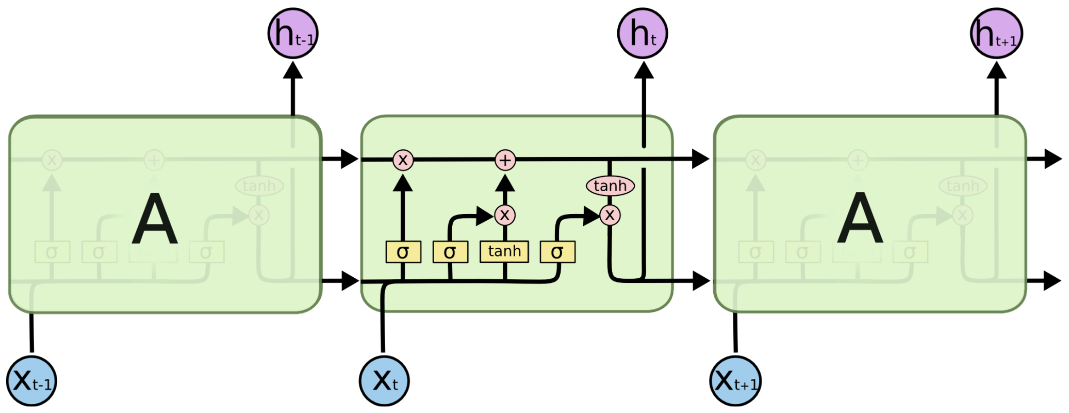
## LSTM

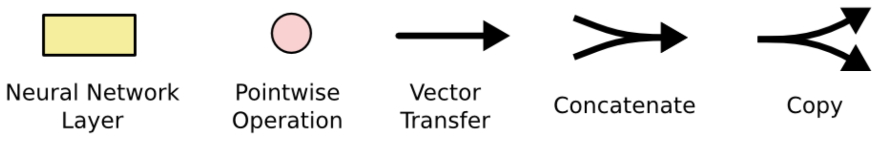
Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work.1 They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn!

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have this chain-like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.





## REFERENCES

https://web.stanford.edu/~jurafsky/slp3/3.pdf
https://colah.github.io/posts/2015-08-Understanding-LSTMs/