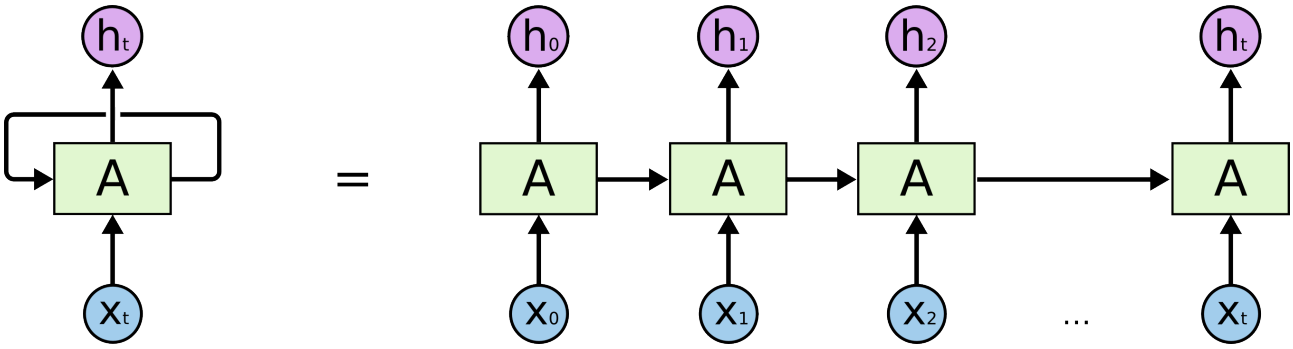
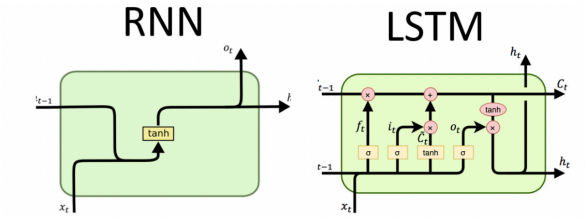
Keshav's Hangman

According to the structure of the problem, we required to develop an AI model that tries to play the classic game of hangman given that the words it will be tested on will be completely new to the model. Clearly initially it needs to guess according to the frequencies of different English letters in different words and as eventually the word gets unfolded, it needs to pay attention to the phonetics as a human player will. For example it's highly unlikely that the "'_" in s_n will be filled with an "x". This requires the model to not just process the letters individually rather as a sequence. Also the length of this sequence is variable so a natural choice was to use an RNN-Model.



The problem with the vanilla-RNN was exploding/diminishing gradients which gives an output filled with "nan"s. The obvious solution was to use an LSTM. LSTMs not only solve the problem of exploding/diminishing gradients but also provide a sense for remembering what's important and forgetting what's not. So the LSTM that we implemented takes the tensor of on-hot tensor representations of the characters of a word and another tensor for the characters that have already been guessed and returns a probability distribution that represents the probability of occurrences of each character in the word.



Internally the LSTM is fed only a single letter(represented as a one-hot vector) at a time and the output from the last cell is concatenated with the tensor that represents the letters that have already been guessed. Then the resultant tensor is fed into a single dense layer that should ideally give the tensor representing probabilities for each letter to be in the "_" of the obscure word. For training we used the cross-entropy loss between two tensors.

The trained model wins the game nearly 62% of the times in the testing data from the 227300 words given for practice. It makes nearly 5.1 wrong guesses on average. The model learnt to predict "e" when the word is totally obscured, which makes sense as "e" is the most frequent letter in the English language. Though it guesses "a" if the word is too short and "i" if the word is too long. Further it keeps guessing a vowel if there are 3 or more consecutive "_".

The results were fairly encouraging. It would make sense to try using transformers for further improvements.