**Introduction**

Include project objectives

**Data Description**

Include EDA findings

**Methodology**

*@ Include data preprocessing, model selection, and evaluation techniques*

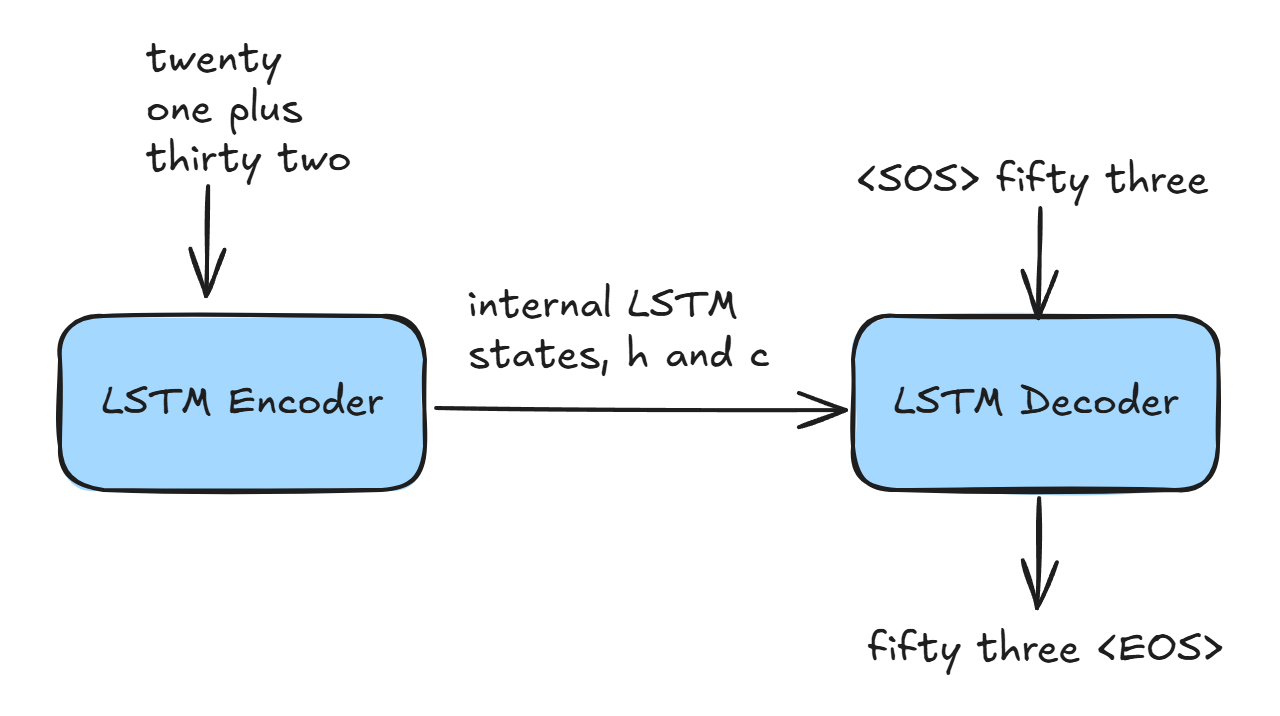
Show evidence as to why we made this choice of model, i.e. links to articles etc justifying why this would be a good choice…………..

The approach taken leverages a ***Neural machine translation*** (NMT) approach that employs deep neural networks to perform machine translation from the source question to the target answer. This requires the neural translation mechanism to take in the question as text (that is, the source language) as a sequence of inputs and encodes these to a hidden representation, which is then decoded back to produce the translated text sequence as the target answer. One of the key advantages of this NMT system is that the whole machine translation system can be trained from end-to-end together.

Generally speaking, ***Recurrent Neural Networks*** (**RNNs**) architectures such as ***Long Short-Term Memory*** (**LSTMs**) and/or ***Gated Recurrent Units*** (**GRUs**) are used in the neural translation machine architecture.

Describe LSTM…..

Our prescribed approach adopts the encoder-decoder model approach, with the training described using the following diagram which shows the high-level architecture of our proposed neural translation mechanism that uses an LSTM as the encoder to encode the input question sequence into final hidden states and final memory cell states .

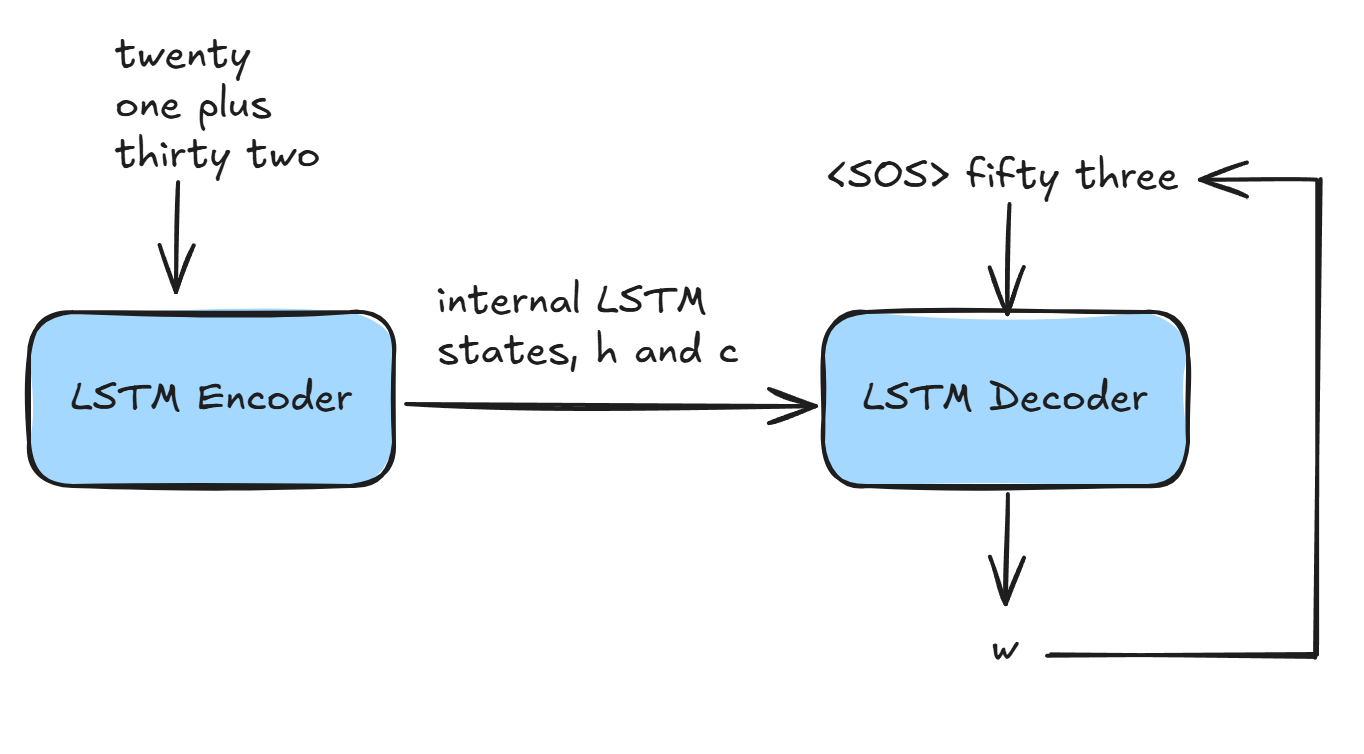


The final hidden states and cell states [,] capture the context of the whole input sequence, therefore [,] becomes a good candidate on which the decoder network can be conditioned. This hidden and cell state information, [,], is passed to the decoder LSTM model as the initial hidden and cell states. The decoder is then trained on the target sequence, with the input target sequence being one step behind the output target sequence. As per the decoder, the first word of the input sequence is the token word <SOS>, while the output label is the word *fifty*. The decoder network is just trained as a generative language model, where at any time step , the output label, is just the next word with respect to the input, that is, . The only new thing is that the final hidden and cell states of the encoder (that is, [,]) is fed to the initial hidden and cell states of the decoder to provide content for the translation.

This means the training process can be thought of as building a language model for the target language, in our case the math answers represented by the decoder, conditioned on the hidden states of the encoder that represent the source language, in our case the math questions. In the simplest terms, what we are trying to achieve when training the model is to make it as accurate as possible when predicting the next word in the target language translation, given the source text and the previous words it has already translated. This involves adjusting the model's internal parameters, , so that the probability it assigns to the correct next word is as high as possible at each step. Essentially, it's like teaching the model to step through the translation process one word at a time, constantly improving its understanding of the relationship between the source and target languages.

With training complete, how can we use the trained model during infer an answer to our math-based question?

The architectural flow for running inference on the model is a little different from what we’ve previously considered for training. During inference, the input sequence is fed to the encoder network and the final hidden and cell state produced, [,], is fed to the decoder’s hidden and cell states. The decoder is converted into a single time step, and the first input fed to the decoder is the dummy <SOS> token. So, based on [,] and the initial token <SOS>, the decoder will output a word, w, and also new hidden and cell states, [,]. This word is fed to the decoder again with the new hidden and cell states, [,], to generate the next word. This process is repeated until we encounter an end-of-sequence <EOS> token. The following diagram demonstrates this high-level architecture.



Our intention is to build a neural machine translation system that will learn to translate simple mathematical questions in sentence form, and produce a correct answer, again in word form.

We cannot directly feed our question and answer (Q&A) text data directly into our proposed model, since neural networks can only understand numerical values. As such, we will treat each word as an index value uniquely assigned to a given word, the length of this index will be equal to the number of words present in each corpus. If the Q&A combination contains 120 unique words, we will have a vocabulary index of size 120, plus any additional standardised tokens we care to add. We will read through the English and the French corpus and determine the number of unique words in each of them. For example, let's assume that in the text corpus, we have four words, “***thirty six plus twenty two***” then we can define the indices of each of the words as follows,

|  |  |
| --- | --- |
| **Word** | **Index** |
| *<SOS>* | 0 |
| <*EOS*> | 1 |
| *plus* | 2 |
| *Six* | 3 |
| *two* | 4 |
| *Twenty* | 5 |
| *thirty* | 6 |
| *eight* | 7 |
| *fifty* | 8 |

So, if we consider the input question, we will have a sequence of words represented as a vector of indexed values, the vector for question pose above will be [6,3,2,5,4].

The next obvious question is how to manage the sequence length, since this might vary. The most accepted approach is to have a fixed sequence length either equal to the maximum sequence length of the sentence in the corpus, or a predetermined reasonable length. We will be using the answer word sequences twice, once as the output sequence of translation from the decoder, and once as the input to the decoder, with the only difference being that the output sequence will be ahead of the input sequence by one time step. So, the first word in the input target sequence would be the token <**SOS**> representing the ‘Start-Of-Sequence’, while the last word in the output target sequence would be the token <**EOS**>, marking the ‘End-Of-Sequence’.

If the target question sequence is “***thirty six plus twenty two***”, the input question and the output answer sequence in the decoder would be as follows,

Question: [**<SOS>**], [***thirty***], [***six***], [***plus***], [***twenty***], [***two***]

Answer: [***fifty***], [***eight***], [**<EOS>**]

This in turn is translated into the required index vectors, which for our example would be as follows,

Question: [***6,3,2,5,4***]

Answer: [***8,7,1***]

**Results**

Discuss results and any conclusions and recommendations

**Conclusion**

Here is my conclusion

**References**

Learn to Add Numbers with an Encoder-Decoder LSTM Recurrent Neural Network

By Jason Brownlee on August 27, 2020 in Long Short-Term Memory Networks

https://machinelearningmastery.com/learn-add-numbers-seq2seq-recurrent-neural-networks/