**Introduction**

The Department for Education (DfE) is committed to supporting the [AI Opportunities Action Plan](https://www.gov.uk/government/publications/ai-opportunities-action-plan/ai-opportunities-action-plan). Most recently, the department has become strongly focussed on the delivery of Generative artificial intelligence models to facilitate teaching and learning, helping to alleviate the learning facilitation burden experienced by many teaching professionals.

The objective of this project is to develop a prototype which demonstrates how such technology could potentially be leveraged in a classroom setting. One of the most pervasive issues, and a major cause for concern for the DfE is the continuing struggle to support STEM subjects beyond the standard curriculum. However, the ability of AI to make a tangible difference in this area is fast becoming apparent, and it is the application of such tools, together with a comprehensive evaluation, to which this project will focus.

More specifically, we will concern ourselves with the development of a prototype **maths question and answer teaching support assistant**, utilising **Sequence-to-Sequence** (**encoder-decoder**) modelling alongside ***Long-Short-Term Memory*** (**LSTM**) models, trained on a variety of mathematics question and answer samples. The approach will include a defined data preprocessing pipeline, a robust model training regime employing ***k-fold cross-validation*** to avoid overfitting, a comprehensive **hyperparameter tuning** regime to maximise performance, and detailed evaluation utilising **BLEU scores**.

The intention here is not to create a first-class, production ready system, but rather to demonstrate and evaluate the potential benefits such an AI model and approach can bring to this learning need. Based on this study, conclusions will be drawn and recommendations made on those areas that offer potential for further development.

**Data Description**

Include EDA findings

Exploratory Data Analysis (EDA) for our encoder-decoder models focuses on understanding the characteristics of the question-input, and answer-output pairs before training. Several techniques were applied both before, and during the training and model iteration phases.

1. \*\*Sequence Length Analysis\*\* – Examine the distribution of input and output sequence lengths to optimize padding and truncation.

2. \*\*Token Frequency Analysis\*\* – Identify common and rare tokens in the dataset to improve vocabulary selection and handling of out-of-vocabulary tokens.

3. \*\*Attention Visualization\*\* – If using attention mechanisms, analyse how the model focuses on different parts of the input sequence.

6. \*\*Alignment Patterns\*\* – Examine how input sequences map to output sequences, especially for tasks like translation or summarization.

7. \*\*Noise and Missing Data Handling\*\* – Detect inconsistencies, incomplete sequences, or irregular patterns that may impact training.

**Methodology**

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

**Choice of Model**

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…………..

There are a number of techniques we could employ to create our prototype, not least the …..blah blah

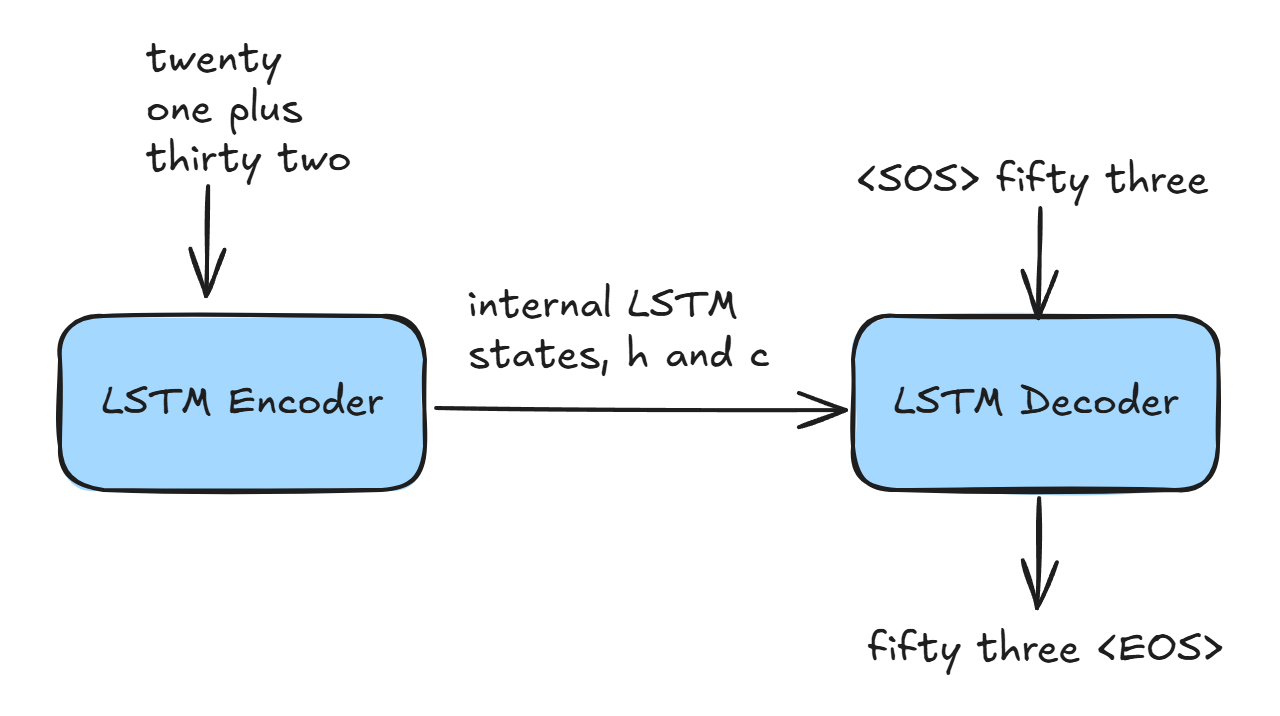
The approach taken leverages a ***Neural machine translation*** (**NMT**) approach that employs deep neural networks to perform machine translation from the source question to the target answer. This requires the neural translation mechanism to take in the question as text (that is, the source language) 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.

We will leverage the power of LSTM models as the basis for our models given their ability to handle sequential data by remembering important information while selectively forgetting irrelevant details. Unlike traditional models that struggle with long-term dependencies, LSTMs generally excel at processing time-series data, such as predicting words or patterns in text. By maintaining context over time, LSTMs enable smarter predictions and more accurate understanding of sequences, thus forming the basis for our prototype.

**Model Overview**

Our prescribed approach adopts an ***encoder-decoder*** model (or what is often referred to as ‘sequence to sequence’), with the training described using the following diagram which shows the high-level architecture of our proposed neural translation mechanism, using LSTMs, firstly 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.

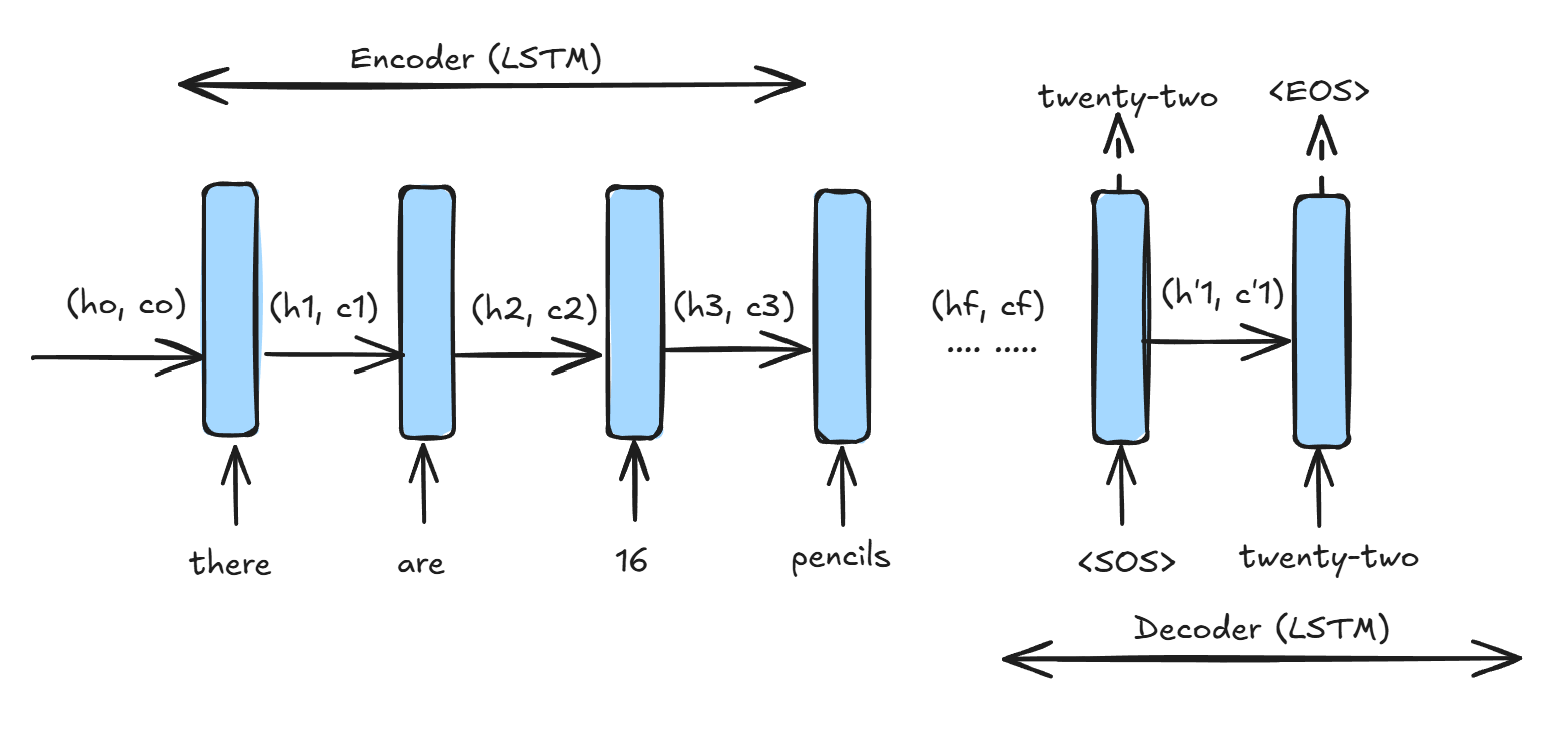
This means the training process can be thought of as building a language model for the target language, in our case the math answers represented by the decoder, conditioned on the hidden states of the encoder that represent the source language, in our case the math questions.

**Model Training**

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.

To summarise, the encoder will process the source input sequence through an LSTM and encode the source text into a meaningful summary. The meaningful summary is stored in the final sequence step hidden and cell states and . These vectors together ([,]) provide a meaningful context about the source input, and the decoder is trained to produce its own target sequence conditioned on the hidden and cell state vectors [,].

The following diagram gives a more detailed view of the training process associated with our math word problem assistant model. The input sequence *“****there are 16 pencils…..****”* is converted to a meaningful summary through the LSTM, which is then stored in the hidden and cell-state vectors [,]. The decoder is then made to generate its own target sequence (**Answer**), conditioned on the input source from the encoder, through the information embedded [,]. The decoder at time step is made to predict the next target answer, that is, the word at time step , given the source input (**Question**).

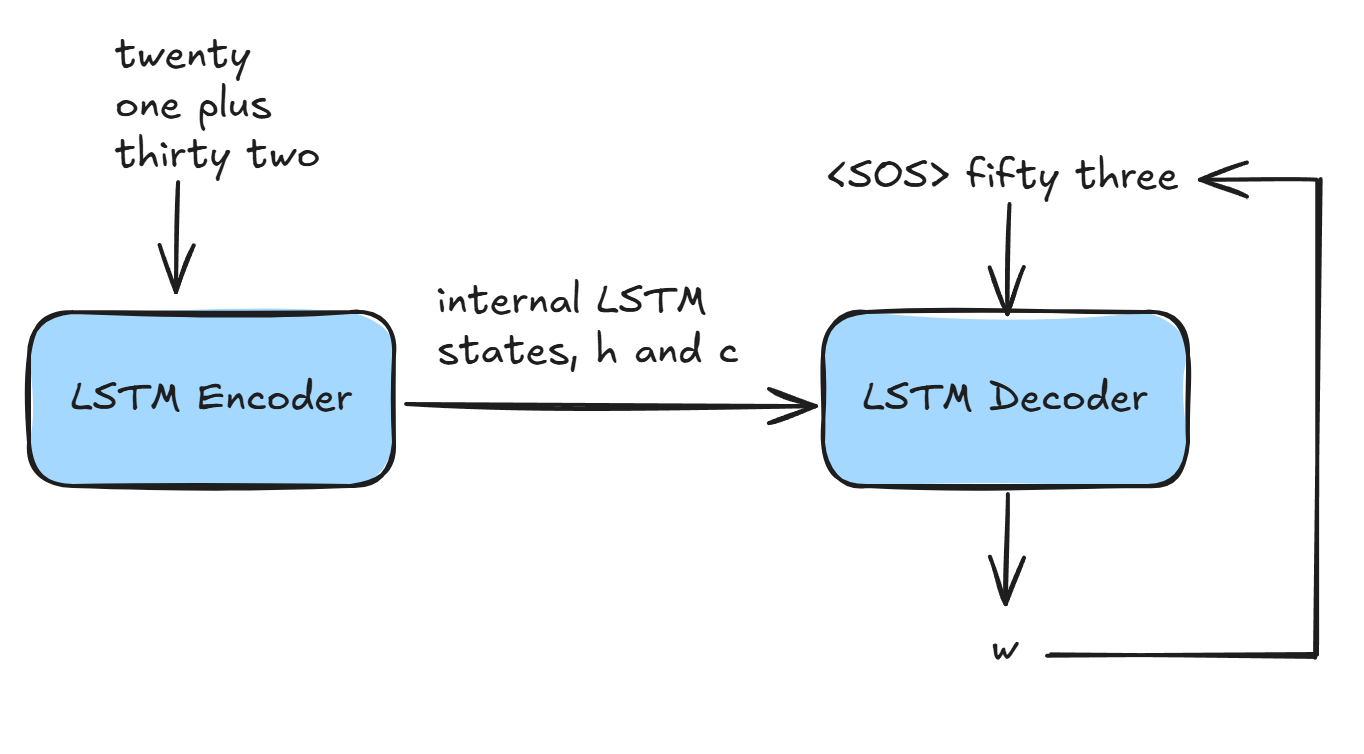


This is why there is a one-step lag between input sequence (**Question**) and target sequence (**Answer**). For the first time step, the decoder doesn't have any prior words in the target input sequence, and so the only information available to predict the target sequence (**Answer**) is the information encoded in [,]that is fed as the initial hidden and cell-state vectors. Like the encoder, the decoder also uses an LSTM and as discussed, the output target sequence is ahead of the input target sequence by one time-step.

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

**Model Inference**

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 the following question and answer, “***there are 16 pencils in one box and 6 pencils in another box - how many pencils are there altogether? twenty-two***” then we can define the indices for each of the words as follows,

|  |  |
| --- | --- |
| **Word** | **Index** |
| *<SOS>* | 0 |
| <*EOS*> | 1 |
| *Altogether?* | 2 |
| *many* | 3 |
| *pencils* | 4 |
| *box* | 5 |
| *there* | 6 |
| *are* | 7 |
| *and* | 8 |
| *another* | 9 |
| *twenty-two* | 10 |
| *there* | 11 |
| *-* | 12 |
| *how* | 13 |
| *one* | 14 |
| *in* | 15 |
| *16* | 16 |
| *6* | 17 |

So, if we consider the input question, we will have a sequence of words represented as a vector of indexed values. 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 as before, the input question and the output answer sequence in the decoder would be as follows,

Question: [**<SOS>**], [***there***],[***are***],[***16***], [***pencils***], [***in***], [***one***], [***box***], [***and***],[***6***], [***pencils***], [***in***], [***another***], [***box***], [***-***], [***how***], [***many***], [***pencils***],[***are***], [***there***], [***altogether?***]

Answer: [***twenty-two***], [**<EOS>**]

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

Question: [**6**,**7**,**16**,**4**,**15**,**14**,**5**,**8**,**17**,**4**,**15**,**9**,**5**,**12**,**13**,**3**,**4**,**7**,**11**,**2**]

Answer: [**10**]

From this, we have identified three stages to our data creation process,

1. Reading the input files for the source (**Question**) and target (**Answer**) texts;
2. Building the vocabulary from the source and target texts;
3. Processing the input **Question** and **Answer** corpuses to their numeric representation so that they can be used in the neural machine translation network.

*\* Mention data preprocessing in terms of padding and adding the hyphen between answer \**

**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/