**Word Embeddings:**

The one-hot vector representations do not draw dependencies and similarities between two very similar inputs. For example, the words apple and orange are not considered as similar. With the one-hot vector representation, these two words are as different as man and water; this is because the dot product of these two words is 0 (as in the form of one-hot vector representations). So, we must look for other word embeddings. One possible approach is to add multiple features that would describe each word. For example, we could add gender, fruit, alive, etc for each word. These features describe the word and what it is. The result would be a vector of n entries corresponding to n features of this word. The dot product of these representations will no longer be 0.

Another way to think of a word embedding using the word’s features is to think of each word being plotted in a 300D (300 = # of features) space. If we want word similarities, like man is to woman like king is to what? We can calculate the trajectory from man to woman and then find a similar trajectory that starts from king and ends at a word (like queen). For this you will subtract the vector representing man from the vector representing woman and then you compare that result with the result of subtracting king from other words in the dictionary. The resultant vector that is most similar to the resultant vector of subtracting man from woman is chosen as the answer. This helps building analogies. e\_man – e\_woman ~= e\_king – e\_w -> argmax (sim(e\_ w, e\_king – e\_man + e\_woman)). Note that if the word embeddings are transformed into lower dimensional vectors, you might not get the desired accuracy when using the method.

The function sim is the cosine similarity. If the two vectors have exactly the same angle, then the theta between them is 0 and sim = cos(0) = 1. If the angle between them is 90 degrees, then sim = cos(90) = 0. And If they are the exact opposite, theta = 180 and sim = cos(180) = -1.

Using this method, the algorithm can recognize CAD to Canada is USD to USA. Or Ottawa to Canada is Nairobi to Kenya. The more similar the are the closer the value of sim is to 1.

More resources at: Linguistic Similarities in Continuous Space Word Representations [Mikolov et. al., 2013]

**A machine translation**

A machine translation task could be quite different from an ordinary natural language programming task. Translation from one language to another uses an encoder-decoder structure, in which the input is mapped into a vector representation and is then passed to the RNN as the initial hidden layer :

… ….

**Figure 1**. Encoder – Decoder network in Machine Translation

**Beam search:**

For some machine translations between languages, the objective is to maximize the probability of translating each word correctly. Therefore, the gradient should be updated simultaneously. This means that we want to maximize the probability of the right translation:

Objective: arg max

Imagine having a dictionary of 10,000 words, we want to predict the most probable 10-word sentence as the outcome of the translation. We would need to calculate the probability of 10^10,000 possible sentences. This is tedious specially since dictionaries can contain so many more words. This brings us to the notion of constructing a search algorithm that finds the sentences with the highest probabilities. One such algorithm is beam search.

Beam search is a heuristic search algorithm used in various computational tasks, particularly in natural language processing and machine translation, to find the most probable output sequence given a sequence of input data [1]. It's widely used in tasks where the search space is too vast to explore exhaustively, such as in generating sequences of words or predicting the next word in a sentence.

Suppose the output of the encoder part of the model is represented as x; note that x will act as for the rest of the RNN. A beam search would then calculate the probability of getting each word in the dictionary as the first word in the translated sequence of words; the beam search then picks n number of those words which have the highest respective probabilities. Note that when using the beam search, we are taking the maximum values among the conditional probability:

Similarly, for the second word in the sequence, beam search finds the n most probable words that could be selected for that position; to do this, beam search takes the n highest probabilities from the conditional distribution:

Generally, to choose the kth word in the sequence, the beam search picks n most likely words from the probability distribution:

Therefore, in order to maximize the probability of choosing the correct translation, we would be maximizing a set of conditional probabilities:

arg max (Eq.1)

Note: if n = 3, this means that the seam search picks 3 words in every step of the word sequence. This means that for a 10-word sentence, we would have 3^10 probable sentences that are likely to be the correct output as compared to 10^10,000 possible combinations of words in our dictionary.

**Beam Search Optimization:**

There are several problems with the beam search algorithm as stated above. Notice that since the probability values are ranged between 0 and 1, if the length of the sequence is long, the values in Eq.1 tend to zero. Additionally, if only one of the probabilities is zero, it sets the outcome of the equation to zero; this could be tackled by taking the log of the probabilities:

arg max (Eq.2)

Another problem with this algorithm is that it is biased towards selecting shorter sentences. In Eq.1, the longer the sentence gets, the smaller its probability. In Eq.2 the values are all negative and the longer the sentence, the larger the negative number, and, therefore, the less probable it is to be chosen by the algorithm. To avoid this biasness, we normalize Eq.2 as the following:

Finally, among all the possible sentences constructed by the beam search and given the probabilities as above, we find the most probable output based on the highest conditional probability.

A note on how to choose the value of n (called beam width) : this number can be optimized to produce the best outcome using trial and error for each model; note that the larger n gets, the more possible sequences are considered and, therefore, the model is likely to produce more accurate results but the algorithm works slower as n gets larger.

**BLEU Scores**

What if there are multiple good reference translations?

Bilingual Evaluation Understudy\*, BLEU, Scores is a metric used to evaluate the quality of machine-generated translations by comparing them to one or more reference translations. It's widely used in natural language processing tasks, particularly in machine translation. BLEU scores range from 0 to 1, where higher scores indicate better translations. A perfect translation receives a BLEU score of 1, while a completely incorrect translation receives a score close to 0. This scoring system rates the machine-generated translation based on its overlap with one or several reference translations; more specifically, the overlaps can be of the form of one word, called unigrams, or multiple words, called bigrams for two words, trigrams for three words, etc. The sum of all overlaps is then divided by the total number of overlap types (unigram, bigram, trigram, etc) to generate a BLEU score. To learn more about BLEU scores refer to the article by (Papineni et al., 2002). [2]

\*Understudy in theater means the role of an actor who studies the act of the main actor and if necessary can undertake the role of the senior actor.

**Transformer networks:**

Transformer networks are a combination of CNN and attention which can parallelize the computations instead of a sequential processing of the data. There are several setbacks of RNNs that the transformer networks address:

1, Problem of vanishing or exploding gradients

2, Difficulty accessing data from long ago

3, Slow computations for long sequences (not able to parallelize the computations)

**Attention Models:**

The attention mechanism was developed first to improve the performance of machine translation task in an encoder-decoder network. At every segment of the prediction, this mechanism allows the algorithm to identify and base the prediction on the most relevant parts of the input vectors by assigning weights to each input vector.

The encoder-decoder network introduced in **Figure 1** has a profound weakness and that is that all of the information in the input of the encoder network is given in a fixed-length vector as the initial hidden layer to the decoder network. This means that no matter how complex or simple the input of the encoder network is, its output representation is still a fixed-length vector which might not suffice to carry all the required information to the decoder network to make accurate predictions. To tackle this problem, (Bahdanau et al., 2014) proposed an encoder-decoder network equipped with an attention mechanism [3].

Let us refer to the sequences in the encoder network with and in the decoder network with time . Note that and The encoder network outputs two parameters at each time ; one is the encoded vector, representing the input and the other is an attention parameter, which weighs the encoded vector based on its relevance to make a prediction at time t. A context vector, , is an input to the decoder network such that:

Where is the attention parameter corresponding to th word in the input to predict the tth word of the output sentence and is the encoded vector representing the input .

To determine the weights of the attention parameters, we run a Softmax function as the following:

Note that the sum of the attention weights inputted to make a prediction at time t should equal to 1 due to using the Softmax function. , called energies or alignment score, is a feedforward neural network which takes the hidden layer of the decoder network at the previous time step, , and the output of the encoder network at time step to output the energies or alignment scores for the encoded vector .

A diagram of a complex function

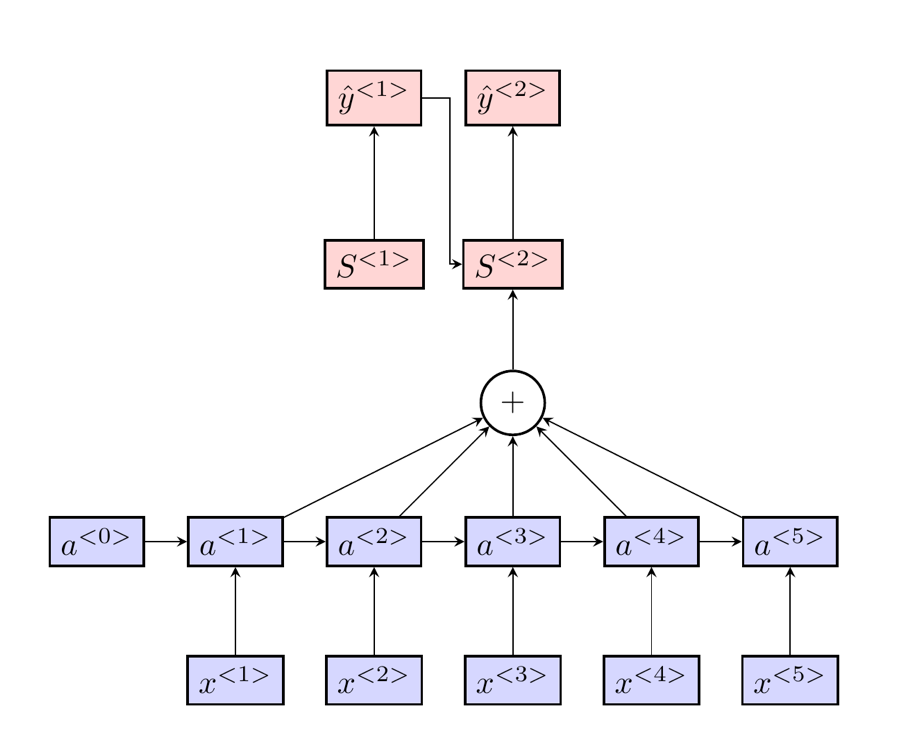
Description automatically generated with medium confidence

Figure 2. Recurrent Neural Network using Encoder – Decoder Structure and Attention model. Diagram is created using Latex. The encoder and decoder network are respectively shown with colors purple and red.

<https://www.coursera.org/learn/nlp-sequence-models/programming/L0BBe/neural-machine-translation/lab?path=%2Fnotebooks%2FW3A1%2FNeural_machine_translation_with_attention_v4a.ipynb>

**General attention models**

The general attention mechanism has three components, namely queries, Q, keys, K, and values, V [4]. Comparing this attention mechanism to the model proposed by Bahdanau et al, we can think of queries being related to the output the hidden layer of the decoder network at time t-1, . Keys and values are related to the hidden layer of the encoder network at time , . More specifically, we will design 3 arbitrary matrices and such that:

Where if is a vector and is a matrix of , then has dimensions .

Where if is a vector and is a matrix of , then has dimensions .

Where if is a vector and is a matrix of , then has dimensions . Note that the values do not need to be of the same size as the queries and keys.

To find a measurement of similarity between and , we take the dot product of the following:

Energy() =

Where is and is ; therefore, Energy() is .

Note that Energy() is the similarity of the tth output of the decoder network with the th hidden unit of the encoder network. We can further vectorize our calculations such that we calculate the similarity of the tth output of the decoder network with all the hidden layer outputs of the encoder network simultaneously:

Consider the K matrix such that its ith row is a vector containing the ith hidden layer output of the encoder network:

Therefore, K matrix has dimensions . The following yields a similarity measurement for all of the hidden layer outputs of the encoder network:

[ energy(), energy() ,..., energy() ] =

Where is a and is a matrix. The above dot product will yield a vector of energies which has the size ; each entry i will correspond to the similarity of (t-1)th output of the decoder network with the ith hidden layer output of the encoder network. These energies are then used to calculate the attention parameters as the following:

Alternatively, one can calculate the attention weights simultaneously as the following:

] = Softmax()

The input to the decoder network is then named a context vector and is calculated as the following:

Therefore, the context vector can be thought of as a weighted average of the hidden layer outputs of the encoder network.

**Pros and Cons of Each Attention Model:**

Benefit of using keys, queries and values is that previously, the energy corresponding to every needed to be calculated individually with a feedforward neural network while now the computations are vectorized and hence are faster and require less amount of storage. More interpretable as the similarity of two vectors is calculated via dot product. Flexibility in Architecture: Keys, queries, and values offer flexibility in designing different attention mechanism variants, such as self-attention and multi-head attention. These variants can capture different types of dependencies and improve performance in various tasks.

FNN-based attention mechanisms offer flexibility in modeling complex relationships and dependencies in the input data. The neural network can learn non-linear transformations and capture complex patterns, that might not be captured by simply taking the dot product. This can potentially lead to improved performance in certain tasks. Furthermore, there is a possibility for overfitting specially if we have a small dataset or if there are complex underlying patterns. The internal workings of FNN-based attention mechanisms may be less interpretable compared to mechanisms based on keys, queries, and values. Understanding how the model assigns weights to different parts of the input sequence can be challenging.

**Self-Attention vs Multi-head Attention:**

The self-attention is about the intra – sequence relativeness. For example, if we have the sentence my cat “is” of bread x, the words “my cat” determine the verb used “is”. This dependency is learned through self-attention. However, we also have dependencies between sentences. For example, if the model has learned the sentence “Sally Jonson is an orange farmer”, it must quickly recognize the pattern with “Albert Su is an apple farmer”. This is between sequence dependencies.

If there is only one attention structure in the model, it is called self-attention. Alternatively, we can define multiple self-attention structures (called multi-head attention) that jointly capture multiple dependencies and relationships within the dataset or the input. In multi-head attention, each head i will have its own set of matrices , and . Each head will create a context vector that is

= Attention(Q, K V) = Softmax() V

Finally, in multi-head attention, the output is calculated as the concatenation of the outputs of each head, context vector, followed by another linear transformation. Formally:

MultiHead(Q,K,V)=Concat(head1​,...,headn​)

**Attention Is All You Need:**

The model consists of two parts: encoder and decoder networks

In the encoder network:

**Step 1**: input the sentence and find the one-hot vector representation of the word. Size: (1 x size of dictionary)

**Step 2**: use word embeddings to represent each word of the input. (based on the dims of word embedding. E.x. if each word is represented by a vector of 512 values, then a sentence of 5 words after this word embedding step will be represented as a 6 x 512 matrix.

**Step 3:** Note that the word embeddings allow the model to understand how similar or related the words are to each other. However, the model has no information on the place of the words in the sequence. This brings us to positional encoding, where the position of each word is represented by a vector of 512 values (same dimension as vector embedding for each word). This positional vector is then added to the embedding for each word along the sequence. Note that positional embeddings do not change if we change the inputted words as their positions remain the same; consequently, these positional vectors will be calculated only once for our model and will remain unchanged while training the model.

Once the input is properly embedded into a matrix of values, it is ready for self-attention or multi-head attention algorithms where dependencies within the sequence of input and its relationship with the previous sequences known to the model will be examined.

Self - attention is each word in the sentence related to the rest of the words in the sentence. So this is about the intra-sentence dependencies. Q, K, and V matrices are the same for each word inputted in the sequence. Then take the dot product of the first word Q\_1 with all the keys K\_i in the sequence to see how similar they are. Where does V appear though ?

Before going through what V is, imagine you put all the embedding next to each other. Instead of working on the attention mechanisms each word at a time, we combine them all. So the matrix Q will have all the word embeddings. In our example its 5 x 512. Multiply this Q matrix with K\_transpose. Note that K and Q are identical and multiplying Q with K\_transpose will give us how similar each word of the sequence is to the rest of the sentences. In terms of the dimensions, Q x K\_transpose will be (5 x 512) x (512 x 5); so the resultant matrix will be a 6 x 6 matrix. Each row i will show the similarity (or dependency) of the word i with the rest of the words in the input sequence. Note that we also normalize this matrix by sqrt(512) = sqrt(the dim of the word embessings). Then use Softmax activation function so that the attention parameters (or the entries in each row) sum to 1 and values fall between 0 and 1. Then you multiply this matrix of 6 x 6 with the matrix V. So, previously, we added the word embeddings to represent the word, then positional embedding that represented the position of the word, and then by multiplying the attention parameters and the V matrix, we will capture the dependencies within the inputted sequence. This V then has lots of information about what the word is, what is its position in the input, and what is its relation to the rest of the words in the input.

A screenshot of a graph

Description automatically generated

Source: [https://github.com/hkproj/transformer-from-scratch notes/blob/main/Diagrams\_V2.pdf](https://github.com/hkproj/transformer-from-scratch%20notes/blob/main/Diagrams_V2.pdf)

**? Will we have parameter matrices in self\_attention? Could it be the word embedding that is being trained? Or maybe before computing Q . K^t, we multiply each with their corresponding parameter weights.**

**Interpretation of the multiplication.**

**Multi-head Attention**

In the multi-head attention, we have 4 copies of the input embedded which contains the word embedding and positional embedding.

Three copies are considered as the query, key, and value matrices. Each query, key, and value matrix will be multiplied by parameter matrix W^q, W^k, W^v. Here it comes to the multi-heads. The resultant matrixes after the multiplication will be split between multiple heads. And each head will receive part of the embedding for each word. In other words, each head will receive all the sentence input but only part of its embedding. Refer to Figure below for a more clear understating.

A screenshot of a computer

Description automatically generatedsource: <https://github.com/hkproj/transformer-from-scratch-notes/blob/main/Diagrams_V2.pdf>

So each head will learn a different aspect of the sequence depicted by the word embeddings it attends to.

A close-up of a graph

Description automatically generated

**Source: Attention is all you need.**

**Masked Multi-head Attention:**

In the decoder part of the model, a masked multi-head attention structure is used. This makes the decoder part of the model causal meaning that we want the model to only make inferences based on the previous words and the future words will be ‘masked’ from the model. To achieve this, we make 3 copies of the output embedding as Query, Key, and Value matrices and apply the parameter weights W^Q, W^K, and W^V to them. then the next step is to take the dot product of Q’ and K’\_transpose. Once this is done we will replace all the entries on the upper diagonal of this resultant matrix with negative infinity. This makes the corresponding matrices 0 once the softmax function os applied to the resultant matrix.

\*\*Note that using transformers we were able to transform sequential learning with n time steps to a model that captures all dependencies and is trained with the sentence all in only one step.

A diagram of a process flow

Description automatically generatedNext steps …

Source: Attention is all you need.

Run a simple attention model.

LSTM models: (incomplete)

In a simple RNN structure, due to the problem of vanishing gradients, the RNN model is not able to keep hold of memory for words far before or after the current state. Therefore, if the prediction depends on such words, the model accuracy might drop significantly. For example given the sentence below, we want to predict the word Italian in the model:

Shirin eats pasta every day. The quality of the food depends on ….. ingredients must be fresh etc… it is obvious that her favorite cuisine is …?

What if we have the example Shirin eats pasta every day. The quality of the food depends on ….. ingredients must be fresh etc… it is obvious that her favorite cuisine is Italian. Sara on the other hand, loves Kebabs and her favorite cuisine is? Persian.

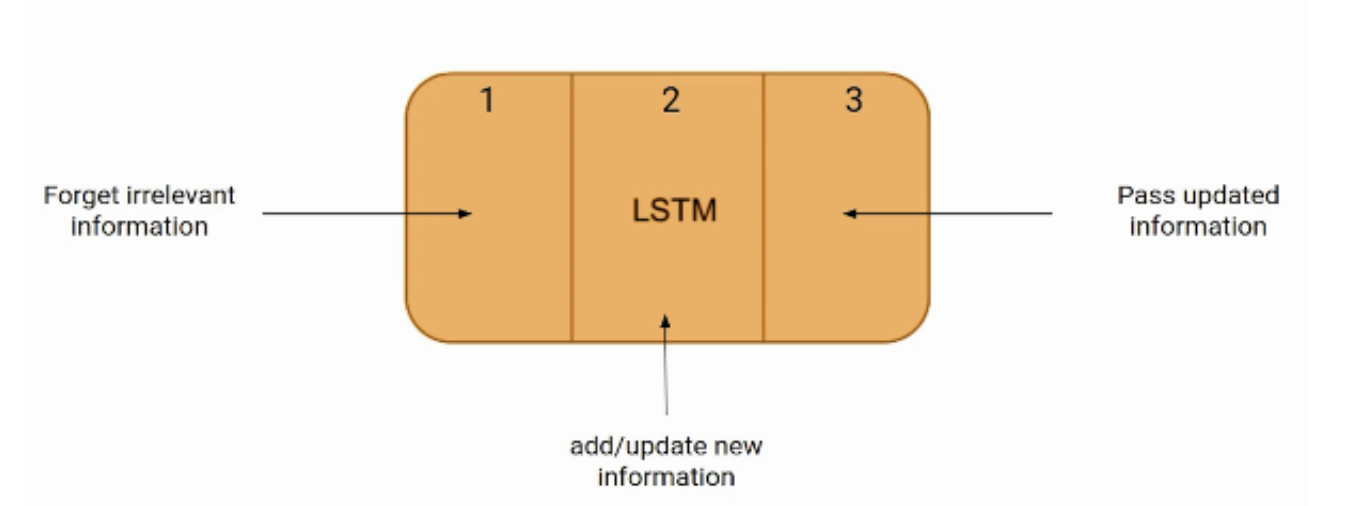
To predict the Italian, the model needs to go all the way back to the word pasta as this is the most significant indicator for the prediction.

Three gates:

1, forget gate:

2, input gate:

A diagram of a computer program

Description automatically generated with medium confidence3, output gate:

More specifically, According to Shipra Saxena's article on analyticsvidhya.com, the model using LSTM will have two hidden layers [5]. The previously defined hidden layer denoted as h will become the short term memory and the output of the LSTM structure, C\_t, will be accounted as the long-term memory.

Intuition: the way an encoder-decoder algorithm works is to first feed all the input to the algorithm, and then start making predictions on what words best translate the word. However, consider the example below:

*Shirin studies applied mathematics at York University. She is currently taking a MATH 4000 independent project with Professor Xin Gao. She is planning for joining a master’s program in the next year*.

If we were to translate this sentence, what do you think would be the first word of the translated sentence?

Do you need to read all the paragraph to predict the first word of the translated output? Or just reading the first sentence suffice? How about knowing only the first 3 words of the paragraph?

The idea lies in the fact that in order to start the translation and predict a word in the sequence, the algorithm does not need to know all the words used in the input. Logically, we would need an algorithm to tell the model where to look for to make the best prediction. One could conclude that this attention algorithm should assign probability values to different sections of the input text, so that the model can pick the most likely sections which include the key words for translation.

Note that without an attention structure in the model, the prediction accuracy relies on the model’s ability to memorize all the text; therefore, depending on the length of the sentences, the accuracy of the model is prone to reduction.

**Resources:**

1. Freitag, Markus, et al. "Beam search strategies for neural machine translation". Proceedings of the First Workshop on Neural Machine Translation, 2017. <https://doi.org/10.18653/v1/w17-3207>
2. Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. [Bleu: a Method for Automatic Evaluation of Machine Translation](https://aclanthology.org/P02-1040). In Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.
3. Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate." *arXiv*, 2014, eprint 1409.0473, arXiv, cs.CL.
4. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
5. "What is LSTM? Introduction to Long Short-Term Memory." analyticsvidhya.com, Shipra Saxena, 04 Jan. 2024, <https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/>. Accessed 04 April 2024.