Exploring Recurrent Neural Networks, LSTMs, and Transformers in Deep Learning

David Felipe Alvear Goyes

28/03/2023

1 Introduction

In the following report, I will discuss the main contributions of Long Short-Term Memory and Transformers. Also the evolution of LSTMs from RNN and the alternative solution of Gated Recurrent Unit networks, and how these network architectures handle sequential data.

2 RNN, LSTM, and GRU

Recurrent Neural, Long Short-Term Memory, and Gated Recurrent Unit Networks, are specialized neural network architectures to handle sequential data. LSTMs and GRUs networks are the evolution of RNN due to the vanishing gradient problem. These architectures are commonly used in the application of natural language processing, time series analysis, and speech recognition.

2.1 Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNN) are a class of Neural Networks designed to handle sequential data. This architecture is designed to have cyclic connections allowing to share of a sequential memory or internal state. This is commonly used in applications for time series analysis, specifically in natural language processing.

The architecture of the RNN consists of an input layer followed by hidden layers, the key element of this implementation is the use of recurrent connection, which allows the network to maintain a hidden state across time steps. The RNN process each element of a sequence one by one, and each cell share a hidden state connection to share the previous states with the next cells. For example, an input sequence of words is transformed into vectors to then processed by the RNN one by one. While processing the first word and generating the output, the cell shares a hidden state of the first processed word with the next cell in charge of processing the next word, then the input word is added to the hidden state and passed to a Tanh activation that helps to regulate the values through the network bounding the values between -1 and 1.

RNNs have some limitations, during training occur vanishing gradient problems when the gradients become too small as they propagate for the network, impeding that early layer continue learning for small gradient updates. Then these neural networks have difficulties learning relationships between distant elements in a sequence, named short-term memory. To overcome these problems Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks are developed. However, RNNs are more computationally efficient than the others proposed

alternatives.

2.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory Networks are the result of the solution to short-term memory due to the vanishing gradient problem. Short-term memory consists of the difficulty of carrying information from early time steps. We have seen previously that due to small gradient updates in RNN, early layers stop training resulting in forgetting what they have seen in longer sequences. Then LSTM networks overcome this problem evoluted from the vanilla RNN.

In LSTM we found a core concept defined as a cell state capable of transporting information in the sequence, having memory, and using gates that are composed of neural networks capable of adding or deleting information in the sequence. The LSTM contains three gates such as forget, input, and output. Also the cell state computation.

- Forget gate: This gate decides to receive the input and decide which information s not important to keep or eliminate. The gate is composed of a sigmoid function that compresses the values between 0 and 1. The gate receives the sum of the previous hidden state and the current input and passes through the gate deciding to keep or throw away.
- Input Gate: Similar to the forget gate, the input gate has two steps. First, the previous hidden state and the current state are passed into a sigmoid function to decide the information that should be kept or not. The other step is to pass the hidden state and the input through a tanh function that help to regulate the network. Finally, the gate multiplies the tanh output with the sigmoid output.
- Cell State: The cell state is calculated sequentially, first passes the previous cell state and multiply by the output of the forget gate, this allows to drop values. Next, the output of the multiplication is added to the output from the input gate aiming to update the cell state to the current sequence. Finally, the result is a new cell state that can be passed to the next cell and the output gate.
- Output Gate: This gate is in charge of deciding the next hidden state. In the beginning, the previous state is added with the current input and passed through a sigmoid function. The output of the output gate is multiplied by the result of passing the current cell state, to decide what information from the cell state the hidden state should carry.

2.3 Gated Recurrent Unit (GRU)

Similarly to LSTMs, Gated Recurrent Unit Networks (GRU) is an evolution of the vanilla Recurrent Neural Networks to overcome the problem of vanishing gradients. GRUs address the need to capture long-range dependencies in sequential data and are a simpler alternative method to LSTMs.

GRU networks use a gating mechanism to control the information in the cell. The proposed method eliminates the idea of the cell state of LSTMs and uses the hidden stat to transfer the information between cells. GRU networks have an update gate and a reset gate. The update gate is responsible to decide what info should be kept or discarded, similar functionality to the forget gate in LSTMs. The reset gate controls which past information influences the current hidden state.

GRUs networks have a simple structure compared to LSTMs with fewer parameters, reducing the computational complexity. This alternative offer a simple and computationally alternative to LSTMs, and capture long-range dependencies more effectively than the vanilla RNN, solving the gradient problem.

3 Transformer

The Transformer introducer in the paper "Attention is all you need" offers a solution to current sequence transduction models which are based on convolutions and recurrent neural networks in the same encoder-decoder structure. The transformer is based on a selt-attention mechanism outperforming translation models and lending to parallelization. Looking at a high level we find that the transformer is a black box that takes an input and generates an output for a certain natural language processing task. The black box has the structure of an encoder-decoder, where the transform is composed of 1 encoder block composed of 6 encoders, and 1 decoder block composed of 6 decoders. The last encoder is responsible to pass the information to the bottom decoder to then generate the output.

3.1 Encoder

The encoder is composed of a self-attention layer and a Feed Forward Network (FFN). The first is designed to help the encoder look at other positions or words in the input sequence while encoding a word. Then, the output of the self-attention layer is passed to the FFN to apply to each position of the sequence after the encoding. In more detail, bottom encoders embed each word in a vector with a defined size of 512, similarly, in the rest of the encoders the input sequence comes from the list of outputs of other encoders, and the size of the list is a hyperparameter calculated from the longest sequence in the training dataset.

One of the main contributions of this study is the main property of the transform, which consist that the words being processed in parallel with the same network. It means that each word has its path in the block and then the same FFN is applied to each word in parallel.

The pipeline can be described as a list of input vectors each of them encodes a word, then the list is passed to the self-attention layer to process each vector while looking at other positions. The list of vectors is passed to the Feed Forward Network layer to process in parallel each vector from the list. The output of the last step is passed as an input to another encoder block.

3.2 Self-Attention

The self-Attention layer is the base mechanism for the encoder and decoder. Going deeper into the attention mechanism, the process starts with the input words that are embedded in a vector. The first step occurs with the computation of three vectors named **Query, Key, and Value** that are abstractions for calculating attention. These vectors result from the multiplication of learned matrices W^Q, W^K, W^v. The second step in the pipeline consists in calculating the score for each embedding, in other words, we assign a score of how much focus from the encoding word to other parts in the sequence. The score is computed with the multiplication of the query vector with each one of the key vectors of the other words in the sequence. In the third and fourth steps, we divide the score by 8 (dimension of the Key vector divided by two) to pass the result through a softmax operation to know how much a word is expressed in the position. The fifth and sixth steps consist of multiplying each Value vector by the softmax score to know the values that should be focused and drown-out irrelevant words and finally sum up the weighted

values to obtain a single output result. The matrix calculation in the attention layer is done by packing the embeddings into a single matrix to then multiplying it by the trained weights.

3.3 Multi-head attention

One of the contributions of the study is the multi-head attention, which improves the attention layer expanding the model's ability to focus on other positions and give multiple possible representation subspaces. The transformer is composed of 8 attention heads, having W^Q, W^K, W^v matrices per set. Each of the heads projects the embeddings in a different representation subspace. In total we have 8 output vectors from the attention heads, given that we only pass 1 single matrix to the Feed Forward Network, the output vectors are concatenated in a single matrix followed by a multiplication for a learned matrix W^o, and the result is a single output matrix that captures the information of from all the heads.

3.4 Positional encoding

Following the encoder implementation. There exists the need to add positional encoding to the sequence to account for the order of the words. This is to provide meaningful distances between embedded vectors. The transform adds a positional vector to the embedded input. This positional vector is generated by concatenating the result of two functions, one based in Sin, and the other in Cos. This mechanism allows to scale of the unseen length of sequences.

3.5 Residual

Having defined the structure of the encoder, the assembly of the block starts with the self-Attention layer followed by a residual connection from the input to then apply a layer normalization. In the same way, the feed-forward layer is followed by a residual connection and a layer normalization. The resulting output vector is passed to the next encoder block.

3.6 Decoder

The decoder section contains similar components to the encoder with some differences. Summarizing, the encoder takes and processes the input sequence to obtain a set of attention vectors Key and Value. The decoder can be described as an iterative process. First, the decoder generates tokens(words) until a spatial symbol is produced, then the output generated at each timestep is fed to the bottom decoder to have access to previously generated tokens. Each decoder layer refines the output of the previous later to obtain a final output produced after the information passed all the decoders, to finally embed a positional encoding to the result.

3.7 Masked self-Attention

The model is only allowed to attend early positions in the output sequence, this can be done using a mask to ensure that the model doesn't have access to future positions. The mask is applied before the softmax. One of the differences in the attention layer in the decoder is that the Q, V, and K values are computed in a different approach. In the decoder, the Query matrix is computed from the previous decoder layer output. The Key and Value are taken from the encoder stack, allowing the decoder to attend to information generated by the decoder stack.

Finally, the decoder is composed of 6 decoder blocks, each block contains a self-attention layer, a residual connection with layer normalization, followed by an Encoder-Decoder attention

where the Key and value vectors are fed from the encoder, and followed by a residual connection with normalization. Finally, the output vectors are fed into a feed-forward network with a residual connection and a normalization layer. As we discussed previously, the decoder has a similar structure to the encoder, the key differences are the computation of the Q, V, and K matrices and the encoder-decoder attention layer. The result of the 6 blocks is stacked to be fed into a linear neural network layer, the result of this operation is the logits vector which has the size of the vocabulary of the words that the model learned from the training dataset. Finally, Softmax is applied to the logits vector to obtain a vector of scores in probability, the cell with the highest probability is chosen to output the word.

3.8 Training and Loss function

The transformation model follows the same approach as an untrained model, performing the forward pass as a trained model. The authors used labeled data to compare the model's output with the correct information using hot encoding. The loss function is designed to minimize the difference between the output probability distributions. The produced probability distributions can be compared using cross-entropy and Kullbak_leibler divergence. The weights of the model are adjusted using the backpropagation step.

4 Handling Extremely Long Sequences

LSTMs, RNNs, and Transformers have different capabilities to handle long data sequences. RNNs have the limitation of processing long sequences because of vanishing gradient problems. When the network process long data sequences the gradients become too small and the network can learn relationships between distant elements in the sequence. LSTM Network architecture is an evolution of RNN, it uses a gating mechanism to control the flow of information allowing it to capture long-range dependencies. However, LSTMs can struggle with the computational complexity of training large models. Finally, Transformers are designed to overcome LSTMs and RNN using a self-attention mechanism to look at other input elements in the sequence while processing a current element. Transformers can learn long-range dependencies without the sequential processing limitation ls LSTM.

5 Conclusion

In this report, I discussed the main contributions of Transformers, Recurrent Neural Networks, and Long-Short term memory networks. LSTMs can handle longer sequences solving the problems of RNN but they still face challenges with sequential processing. Transformers overcome the performance of recurrent-based neural networks using the self-attention mechanism.