University of Strathclyde Glasgow

Natural Language Processing For Program Language Processing

Automatic Code Comment Generator

By - Syed Junaid Iqbal

Supervisor - Dr Feng Dong

NLP Pipeline

Step 01

Data Pre-Processing

Speaking about NLP models, ***text preprocessing*** plays a very important role. By changing all characters to *lowercase, eliminating punctuation,* and *eliminating stop words and typos*, it is possible to reduce the amount of noise in the data. When you wish to perform text analysis on data pieces like comments or tweets, removing noise is useful.

Step 02

Tokenization

Tokenisation is breaking up a long block of text into tokens. Tokens in this context can be **words**, **characters**, or **subwords**. With transformers, we will be using pre-trained tokenisers. BERT tokeniser is based on the " word piece tokens". This helps us deal with oov words.

Step 03

Word-Embedding and Position Encoding

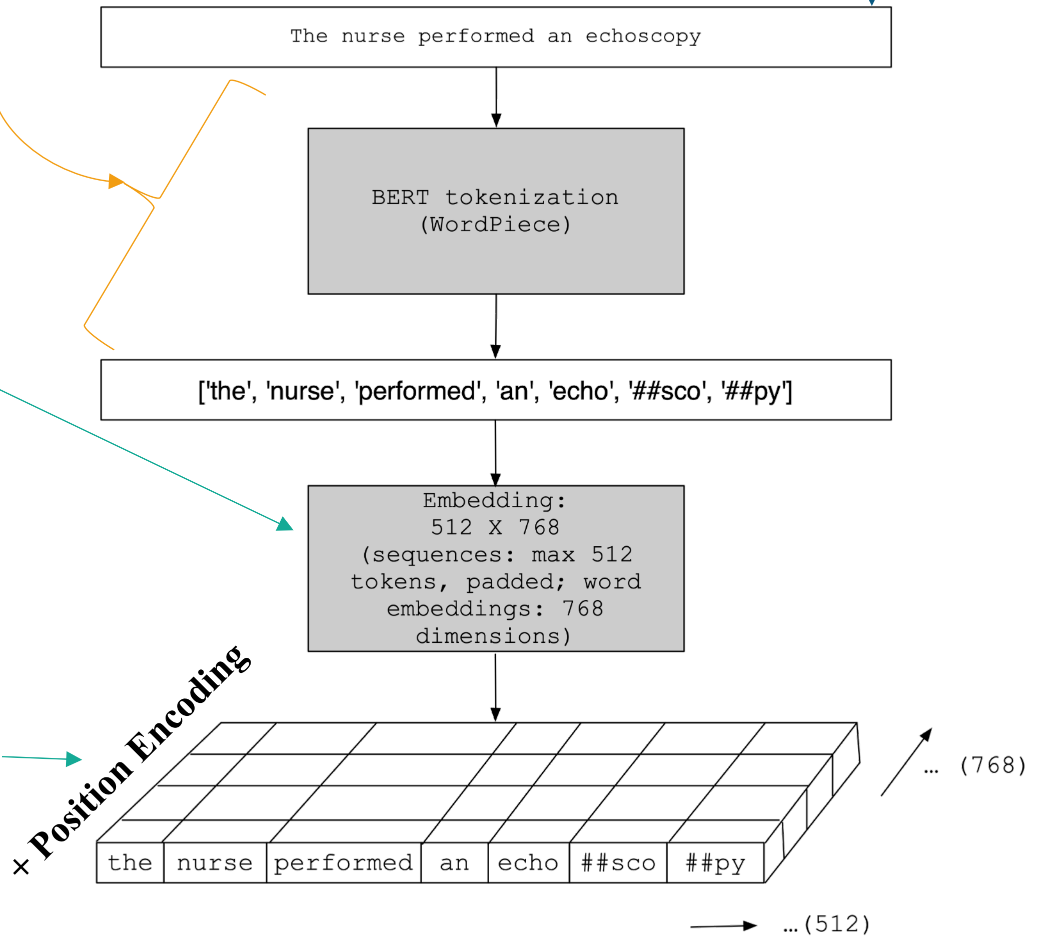
One way to visualise a ***word embedding layer*** is as a lookup table that can be used to retrieve a learnt vector representation of each word. Since neural networks learn through numbers, each word is represented by a vector with continuous values. Each word is represented by a "n" dimension.

To give each position a distinct representation, ***positional encoding*** represents the location or position of an entity in a sequence.

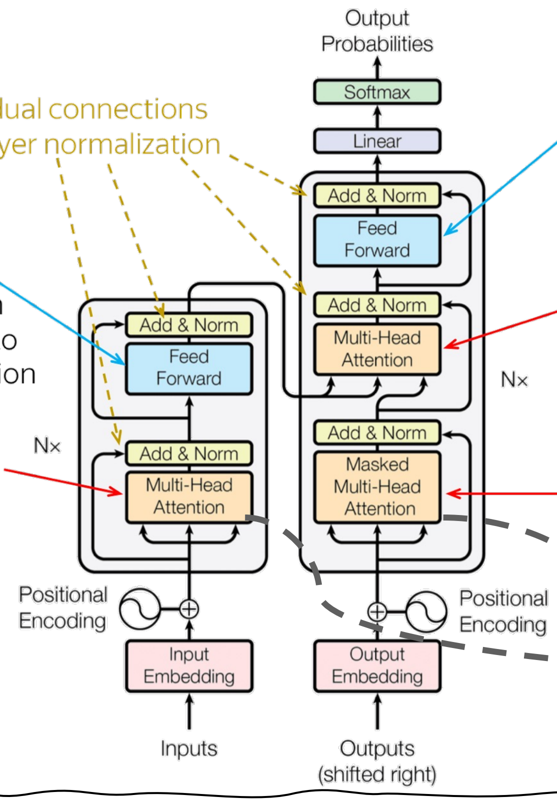
Step 04

Develop and Train Encoder-Decoder model

The major advantage of using ***transformers*** is that it allows us to save weights and use them on various other models. Here we have an encoder-decoder style model. All we do is load the weights and fine-tune model



The nurse performed an echoscopy
BERT tokenization (WordPiece)
['the', 'nurse', 'performed', 'an', 'echo', '##sco', '##py']
Embedding: 512 X 768 (sequences: max 512 tokens, padded; word embeddings: 768 dimensions)
Position Encoding
the
nurse
performed
an
echo
##sco
##py
... (768)
...(512)



Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

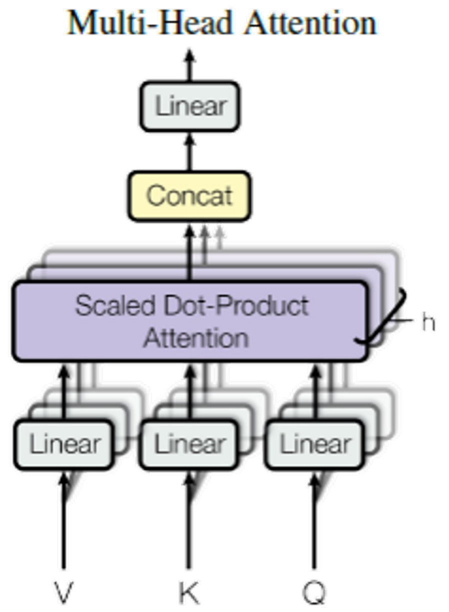
queries, keys, values are computed from encoder states

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

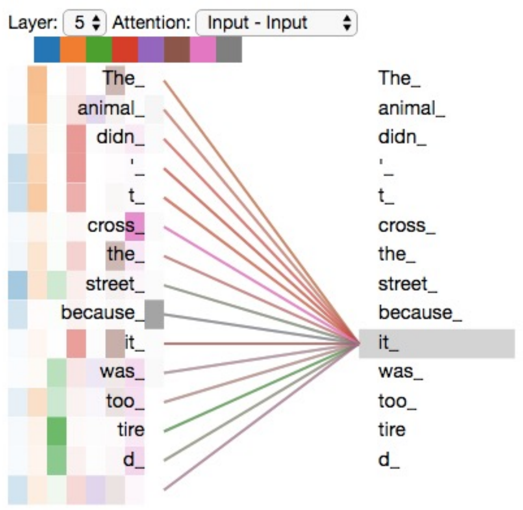
Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

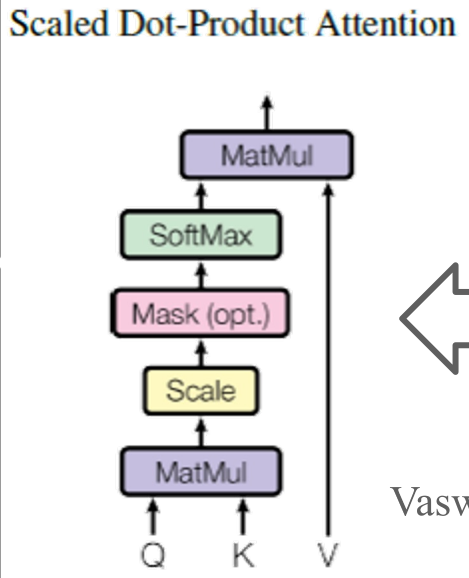


Transformers – Attention is All you need !

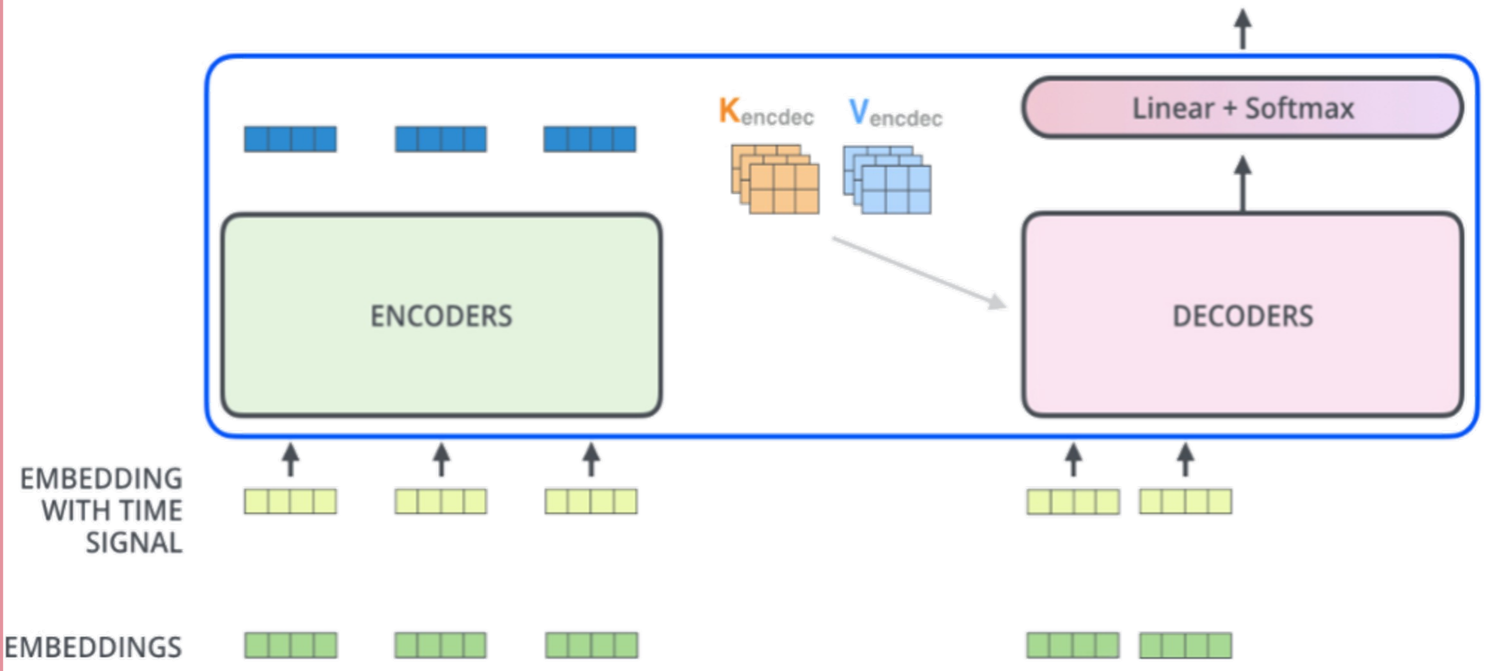
To determine how essential each word in the phrase concerns the other words, the Transformer model extracts features for each word using a self-attention process. Additionally, these features are obtained using only weighted sums and activations rather than recurrent units, making them highly parallelisable and effective.



Attention Scores

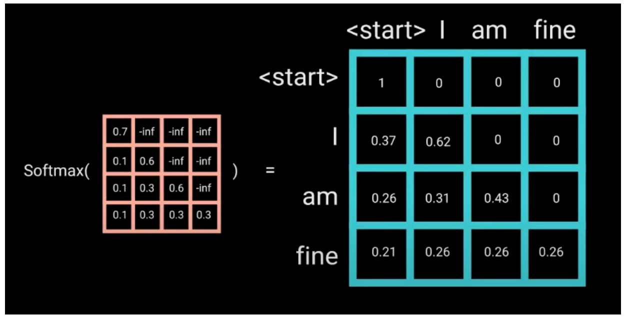


Vaswani(2017)



Each encoder layer's job is to create encodings that indicate which elements of the inputs are related to one another. A feed-forward neural network and a self-attention mechanism are the two main parts of each encoder. The self-attention mechanism takes the input encodings provided by the prior encoder and evaluates their interrelationships to produce output encodings. Each output encoding is then further processed independently by the feed-forward neural network. The following encoder receives these output encodings as input, along with the decoders.

In contrast to the encoder layer, the decoder layer creates an output sequence using all the encodings and the contextual information they have included. To create an output sequence, the decoder gets both the encoder's output and the decoder's output from the preceding time step. Each encoder and decoder layer employs an attention technique to do this. Attention considers the value of each input concerning each other input and draws on them to produce the result for each input. The model is auto-regressive and uses the output of the previous phase as extra input.



$$\text{Softmax}\left(\begin{array}{cccc} 0.7 & -\text{inf} & -\text{inf} & -\text{inf} \\ 0.1 & 0.6 & -\text{inf} & -\text{inf} \\ 0.1 & 0.3 & 0.6 & -\text{inf} \\ 0.1 & 0.3 & 0.3 & 0.3 \end{array}\right) = \begin{array}{c|cccc} & \text{<start>} & \text{I} & \text{am} & \text{fine} \\ \hline \text{<start>} & 1 & 0 & 0 & 0 \\ \text{I} & 0.37 & 0.62 & 0 & 0 \\ \text{am} & 0.26 & 0.31 & 0.43 & 0 \\ \text{fine} & 0.21 & 0.26 & 0.26 & 0.26 \end{array}$$

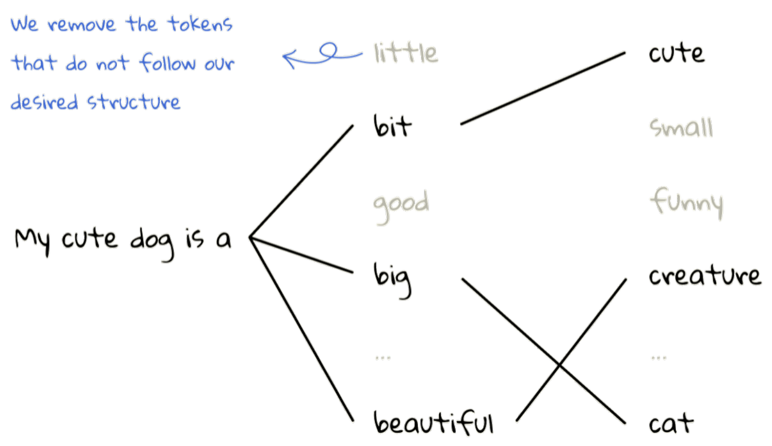
To make sure that our model is unable to gaze ahead into the sequence we apply a self-attention mask and compute a dot product with the raw weights. This allows the model to be autoregressive. With this, we can ensure that model cannot look forward in the sequence.

The input characteristics of linear layers are multiplied by a weight matrix to create the output features. To generate the sequence, we iterate the output sequence number of times and apply SoftMax and determine the probability of a word.

Efficient Search Algorithm

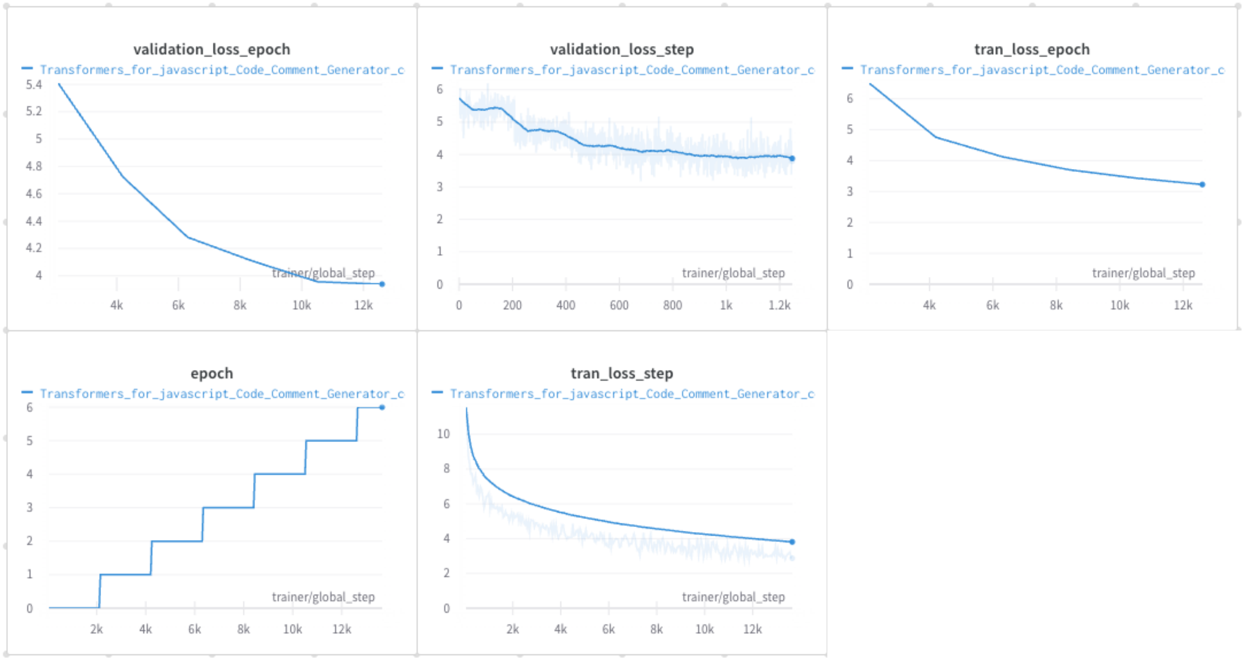
University of Strathclyde

Constrained Beam Search



Based on conditional probability, the beam search algorithm chooses various tokens at a point in each sequence. The algorithm can consider any number of N optimal options through a hyperparameter known as Beam width. The Beam search algorithm works in 3 stages; Grab the top three projected words at each place in each sequence with our beam width set to N, then calculate the conditional probability. Finally, decide the word that would best fit the series.

Model Training Performance



We trained the model based on the code-x-glue dataset. This dataset provides us with code and their respective doc string. To ensure we train our model most efficiently, we use a learning rate schedular to enhance our optimisation process. As the model had around 175 million trainable parameters, it took around 12 hours to complete each epoch. During the training process, we observed that the training loss and evaluation loss decreased as the model proceeded with its training process. The model stops its training process as the evaluation loss remained unchanged for the previous 3 epochs

Generated Comments

```
code :
def shutdown_request(self, client_id, msg):
        """"""""
        self.session.send(self.query, 'shutdown_reply', content={'status': 'ok'}, ident=client_id)
        # also notify other clients of shutdown
        self.session.send(self.notifier, 'shutdown_notice', content={'status': 'ok'})
        dc = ioloop.DelayedCallback(lambda : self._shutdown(), 1000, self.loop)
        dc.start()


True comment :
handle shutdown request.


Generated Comment  :
['send a shutdown request.']
```

```
code :
def patches_after(self, patch):
        """  """

        return [line.get_patch() for line in self._patchlines_after(patch) if
                line.get_patch()]


True comment :
Returns a list of patches after patch from the patches list


Generated Comment  :
[':type patch: str :rtype: list[str]']
```

```
code :
def parse_notifier_name(name):
    """
    """
    if isinstance(name, str):
        return [name]
    elif name is None:
        return ['anytrait']
    elif isinstance(name, (list, tuple)):
        for n in name:
            assert isinstance(n, str), "names must be strings"
        return name


True comment :
Convert the name argument to a list of names.


Generated Comment  :
['parses a name into a list of strings.']
```

As we see, in most cases we can accurately generate comments for our code snippets. But when the code cannot generate comments, it returns a list that shows the data type of the input argument and the output.

```
code :
def copy(self):
        """"""""

        return Striplog([i.copy() for i in self],
                        order=self.order,
                        source=self.source)


True comment :
Returns a shallow copy.


Generated Comment  :
['create a copy of this object.']
```

Future works

- Deploy a model as an API such that it can be used by developers in real-time
- The current algorithm uses text as a sequence without understanding its structure, we can overcome this using graph neural networks. Here AST tokens can understand the semantics of code structure