tl;dr Abstractive: Milestone Report

Problem:

We live in a world where advances in technology have increased the production of information. One concern is that massive amounts of information can be distracting and negatively impact productivity, decision-making and cognitive control (information overload), another concern might also be the ‘contamination’ of useful information with information that might not be totally accurate (information pollution).
In this pilot project, we will try a new approach to address the first concern, information overload, by leveraging the power of deep learning.

In a previous project, we already explored other approaches, such as extractive summarization. Extractive methods yield good accuracy overall but the inherent nature of the process of selecting a subset of the words or sentences in an existing document and assembling them together to make a summary makes the result less cohesive. The extracted sentences are basically concatenated to each other and there is no smooth transition between ideas or concepts in different sentences.

To overcome this problem, one good approach to explore is abstractive text summarization which basically consists in generating summaries from texts from the main ideas. New sentences are generated from the original text and the sentences generated through abstractive summarization might not be in the original text, this makes for a more cohesive result which might look more like a human generated summary.

Approach

Transformers and Transfer Learning are still active areas of research so new theories, approaches and concepts are being developed on a regular basis. Consequently, this project builds on recent papers, articles as well as a few tutorials in order to construct the transformer and reuse the bert pre trained model for transfer learning (See Reference Section).

Our approach is based on using the power of transfer learning along with the transformer architecture to create a model that will be able to ‘interpret’ meaning and generate summary sentences based on attention mechanisms.

Transfer Learning

Transfer learning is based on storing knowledge gained while training a model to solve a problem and applying this stored knowledge to a different problem that still has a similarities with the initial problem it was trained on. So instead of training a deep network from scratch, we take a pre trained model, and apply the knowledge it acquired on our problem at hand.

For our project, we will use the `bert_en_uncased_L-12_H-768_A-12` model from tensorflow hub as the encoder. This pretrained model uses:

- 12 Hidden layers (transformer blocks)
- Hidden size of 768
- 12 Attention Heads

This model has been pre trained for english on Wikipedia and a large book corpus. The text has been lowercased before WordPiece tokenization and any accent markers have been stripped away. Random input masking has been applied independently to tokens.

The Transformer architecture

From the 'Attention is all you need' paper, the transformer architecture has significantly improved various natural language understanding tasks, it has also the advantage of being much faster to train and easier to parallelize than other techniques.

Components

- Encoder: Takes as input a batch of sentences (as a sequence of word ids) and encodes each words in a 512 dimensional representation. The input is of shape [batch size, max sequence input length] and the output is of shape [batch size, max input sequence length, 512]. In our approach we will make use of transfer learning by using a pretrained bert model as the encoder.

- Decoder: While training, it takes target sentences as input (sequence of words ids). These sentences are shifted one time step to the right (inserting a start of sequence token at the beginning). It receives the output of the encoder as well and outputs a probability for each possible next word at each time step (output shape is [batch_size, max output sentence length, vocabulary length]). We bear in mind that during inference, we cannot feed targets to the decoder, so we feed it the previously output words (starting with a start of sequence token). This model needs to be called repeatedly, predicting one or more words at every round which is in turn fed to the decoder at the next round until it outputs an end-of-sequence token.

Dataset

For this project, we chose to use the cnn_dailymail dataset which we can directly load from tensorflow datasets. The first feature is the article and the second one are the highlights which is the target summary (for supervised training). This dataset contains 287,113 training examples 13,368 validation examples and 11,490 examples for testing.

Preprocessing

To preprocess this dataset, we must build word tokenizers so that our model understands the input and is able to output word sequences properly.

For the encoder, we need to encode the text in a way the pre trained bert model will understand. in order to do do this, we use the Full Tokenizer from the bert module to convert

the input text to tokens then we insert a start token [CLS] and a [SEP] token at the start and end of each sequence. Finally we use the `convert_tokens_to_ids` method to convert the concatenated tokens into word ids.

For the decoder, we create a custom subword tokenizer from the training set and store it in a vocabulary (vocab) file for faster access later. For tokenization, we use the SubwordTextEncoder which does tokenization on spaces and punctuation jointly with splitting to subwords.

For simplicity and faster training we set the max sequence length to 300. The data was batched with a batch_size of 64 and the dataset was cached to memory for faster reading.

Building the Transformer

Encoder

BERT

Bidirectional Encoder Representations from Transformers (BERT) is a deeply bidirectional, unsupervised language representation, pre-trained using only a plain text corpus. To pre train BERT, two unsupervised tasks are used:

- Masked LM: Some percentage of the input tokens are randomly masked, then those masked tokens are predicted. This procedure is referred to as MLM. To mitigate the mismatch between pre-training and fine-tuning, the training data generator chooses 15% of the token positions at random for prediction. If for instance the i-th token is chosen, it will be replaced by the [MASK] token 80% of the time, a random token 10% of the time or the unchanged token 10% of the time.
- Next Sentence Prediction (NSP): To train a model that understands sentence relationships, we pre-train for a binarized next sentence prediction. When choosing the sentence A and B for each pre training example, 50% of the time B is the actual next sentence that follows A and 50% of the time a random sentence from the corpus.

Attention in BERT

In BERT, attentions is a function that takes a sequence X where each element is a vector and returns a sequence Y of the same length. Each word can be associated with a word embedding (vector) that captures different attributes of that word. When we apply attention to a word embedding, we create what is called a composite word embedding that helps understand the relationships between words in a sentence.

- Multi Head Attention: BERT actually learns multiple attention mechanisms called heads which work in a parallelized way. Repeated composition of word embeddings allows BERT to form richer representations as it goes into the deeper layers of the

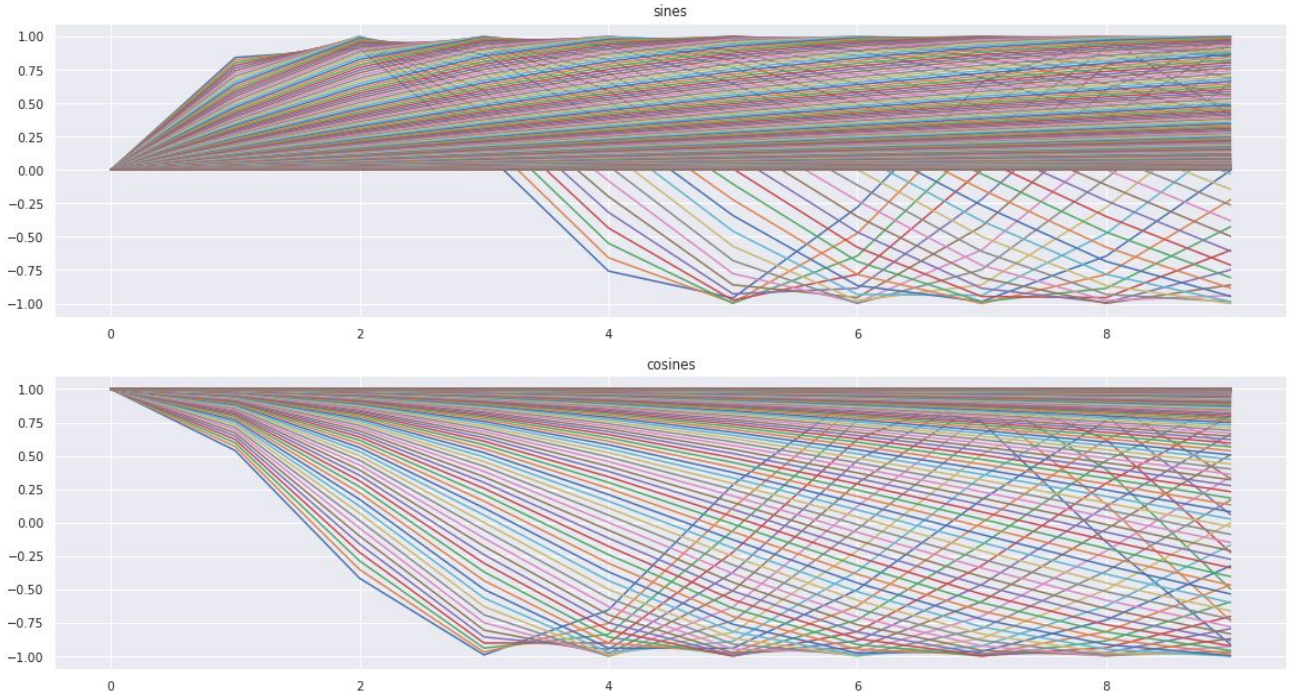
model. Attention heads do not share parameters. Each head learns a unique attention pattern, in our case we have 12 hidden layers x 12 heads = 144 attention mechanisms.

- Attention weights: They are computed using a compatibility function which assigns a score to each pair of words indicating how important the attention should be between them. To compute the compatibility, the model assigns to each word a query vector and a key vector, then the compatibility score is computed by taking the dot product of the query vector of one word and the key vector of the other word. The compatibility scores are then normalized to be positive and summing to 1 (since attention weights are used to compute weighted averages). For this, we use the softmax function over the scores of a given word.

Positional Encoding

A positional embedding is a dense vector that encodes the position of a word within a sentence. We add the i-th positional embedding to the word embedding of the i-th word in the sentence. Here, we need fixed positional embeddings, for that we use the sine and cosine functions of different frequencies. We chose this solution because we can extend it to arbitrarily long sentences.

After Adding the positional embedding to the word embedding, the model has access to the absolute position of each word in the sentence (unique positional embedding for each position). The sine and cosine functions are oscillating functions that allow the model to learn relative positions as well.



We use the sine and cosine functions of different frequencies. The sinusoidal version was chosen because it allows the model to extrapolate to sequence lengths longer than the ones encountered while training.

Masking

- Padding mask: We use masking to ensure that the model does not treat padding as the input. Where the pad value 0 is present, it outputs 1 at those locations and 0 otherwise.
- Look Ahead Mask: Masks the future tokens in a sequence (indicates which entries should not be used). For instance, to predict the third word, only the first and second word will be used. Similarly to predict the fourth word, only the the first, second and third word will be used and so on.

Scaled Dot Product Attention

The scaled dot product attention layer is the base of multi head attention. When the encoder analyzes an input sentence, for instance “He studied math”, it manages to understand that “He” is the subject and the word “studied” is the verb so it encodes this information in the representation of these words. If the decoder has already predicted the subject and thinks it has to predict the verb next, for this it needs to fetch the verb from the input sentence.

It works like a dictionary lookup, as if the encoder created a dictionary and the decoder wanted to look up the value corresponding to the key “verb” but since the model has vectorized representations of the tokens (learned during training), the key that will be used for lookup (the query) will not perfectly match any key in the dictionary.

So we compute a similarity score between the query and each key in the dictionary and then use the softmax function to convert these scores to weights that sum up to 1. The more similar the key to the query , the closer it will be to 1.

The model can then compute the weighted sum of the corresponding values. The similarity measure used by the Transformer is simply the dot product.

In the following function:

$$\text{Attention (Q, K, V)} = \text{softmax}(Q K^T / \sqrt{d_{keys}})V$$

- **Q** is a matrix that contains one row per query (shape = [n_queries, d_keys]), where n_queries is the number of queries and d_keys is the number of dimensions of each query and each key.
- **K** is a matrix containing one row per key (shape=[n_keys, d_keys]), where n_keys is the number of keys and values.
- **V** is a matrix containing one row per value (shape=[n_keys, d_values]), where d_values is the number of each value.
- **Q K ^T** is of shape [n_queries, n_keys] and contains one similarity score per query/key pair. The output of the softmax function has the same shape but all the rows sum up to 1.

We can mask out some key-value pairs by adding a very large negative number, (in this case 1e-09) to the corresponding similarity scores, just before computing the softmax. Useful in the Masked Multi Head Attention layer.

Multi Head Attention Layer

The Multi head Attention layer is a group of Scaled Dot Product Attention layers, each of them preceded by a linear transformation of the values, keys and queries (Time Distributed Dense layer with no activation function).

We concatenate all the outputs and they go through a final layer transformation (Time Distributed).

The word representation encodes many different characteristics of the word. The Multi Head Attention layer apples multiple different linear transformations of the values, keys and queries. It allows the model to apply many different projections of the word representation into different subspaces, each focuses on a subset of the word's characteristics.

So, for instance, maybe one of the linear layers will project the word representation into a subspace where all that remains is the information that the word is a verb, another linear layer layer will select the fact that it is past tense and so on. The Scaled Dot Product Attention layers apply the lookup phase and and we concatenate all the results and send them back to the original space.

Point Wise Feed Forward Network

It is composed of 2 dense layers, the first one uses ReLU activation function and the second one has no activation function.

We can describe it as two convolutions with kernel size 1, The dimensionality of input and output is d_model = 512, and the inner layer has dimensionality `dff`=1024

Decoder Layer

The decoder layer is composed of the following sublayers:

- Masked Multi Head Attention: with a look ahead mask and padding mask.
- Multi Head Attention:
    - V (value) and K (key) receive the encoder output as inputs.
    - Q (query) receives the output from the masked multi head attention sublayer.
- Point wise feed forward networks: Applied to each position separately and identically.
- Dropout layers: for regularization, we apply dropout to the output of each sublayer before adding it to the sublayer input and normalized.

Each sublayer has:

- Residual Connection: also called skip connections, they allow gradients to flow through a network directly, without passing through non-linear activation functions which, depending on the weights, are a cause of vanishing or exploding gradients.
- Normalization layer: it normalizes the activations of the previous layer for each given example.

Each sublayer output is of type LayerNormalization(x + Sublayer(x)), the normalization is done on the last axis of d_model (512-dimensional encoded representation of each word).

Decoder

The Decoder is composed of:

- Output embedding: x of shape [batch_size, target_seq_len, d_model]
- Positional encoding: encoding the absolute and relative position of each word.
- N decoder layers: the decoder layers are stacked N times. Note that the encoder stack's final outputs are fed to the decoder at each of these N levels.

The target goes through an embedding that we sum to the positional encoding and feed the sum's output to the decoder layers. Finally, the output of the decoder is the input of the last linear layer.
We also apply dropout to the sums of the embeddings and the positional encodings for regularizations. Here we use a dropout rate of 0.1.

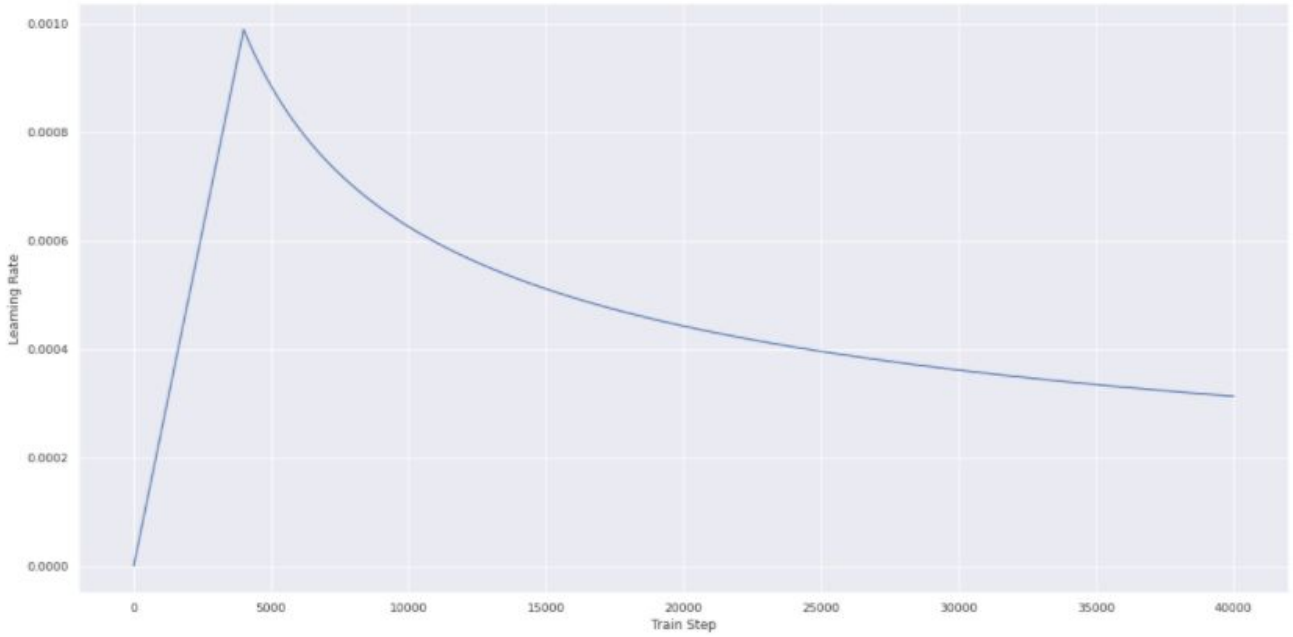
Transformer

To create the transformer, we build the encoder by loading the pre trained weights of the bert model. Then we build the decoder using the previously built `Decoder` class and we pass the decoder output through a Dense layer. The final output is of shape [batch_size, tar_seq-len, target_vocab_size]. To keep the training small and relatively fast, we use a dropout rate of 0.1, we create a `Config` class and we instantiate it with the following parameters:

- Number of layers (num_layers): 6
- Input and output dimensionality (d_model): 256
- Inner layer dimensionality (dff): 1024
- Number of attention heads (num_heads): 8

Optimizer and Learning Rate

We use the Adam optimizer with a custom learning rate scheduler. The custom optimizer consists in increasing the learning rate linearly for the first warm up training steps, then decreasing it proportionally to the inverse square root of the step number. In our example, there are 4000 warm up steps.



Loss and Metrics

Since the target sequences were padded, we apply the padding mask while calculating the loss. We use sparse categorical cross entropy because there are more than two label classes. It is an integer-based version of the categorical cross entropy loss function, which means we don't have to convert the targets into categorical format anymore.

Training

To save checkpoints every epoch, we create the checkpoint path and checkpoint manager. To give the model training a boost, we use teacher forcing by replacing each sequence prediction at each time step with the correct sequence at that time step. So rather than using a prediction for a sequence at time t, we would use the actual value from the training set.

Self-attention gives the transformer the opportunity to look at the input sequences so far (previous words) so it can predict the next word more accurately.
We mask the decoder with a padding mask and a look ahead mask. The look ahead mask forbids the model from looking at the expected output. To implement this inside of the scaled dot product attention mechanism we mask out (set to a big negative number) all values in the input of the softmax which corresponds to illegal connections.

For training, we define a function to train our data at a specific step during which we make predictions, compute the loss and accuracy and use the gradient tape context manager for automatic differentiation (calculating the gradient of a computation with respect to its input variables).

The data was trained over 20 epochs, checkpoints were set to resume training when training get interrupted for some reason (due to colab limitations sessions kept crashing or

disconnecting). We monitor accuracy and loss by printing them every few batches. At the end of training the model weights are saved for later use.

Evaluation

First, we create a function to encode input text using the bert tokenizer. To evaluate our model we create an evaluation function where:

- We normalize the input sentence (tokenize and encode it in bert) and we pad it to the appropriate size.
- We compute the combined masks (padding and look ahead) as well as the decoder padding mask for the input text.
- We use the trained transformer to output the predictions and the attention weights, the transformer decoder outputs the predictions after looking at the encoder's output and using the self attention mechanism.
- We compute the argmax of the selected last word.
- We pass the concatenated predicted word to the decoder.

The decoder predicts the next word based on the previously predicted words.

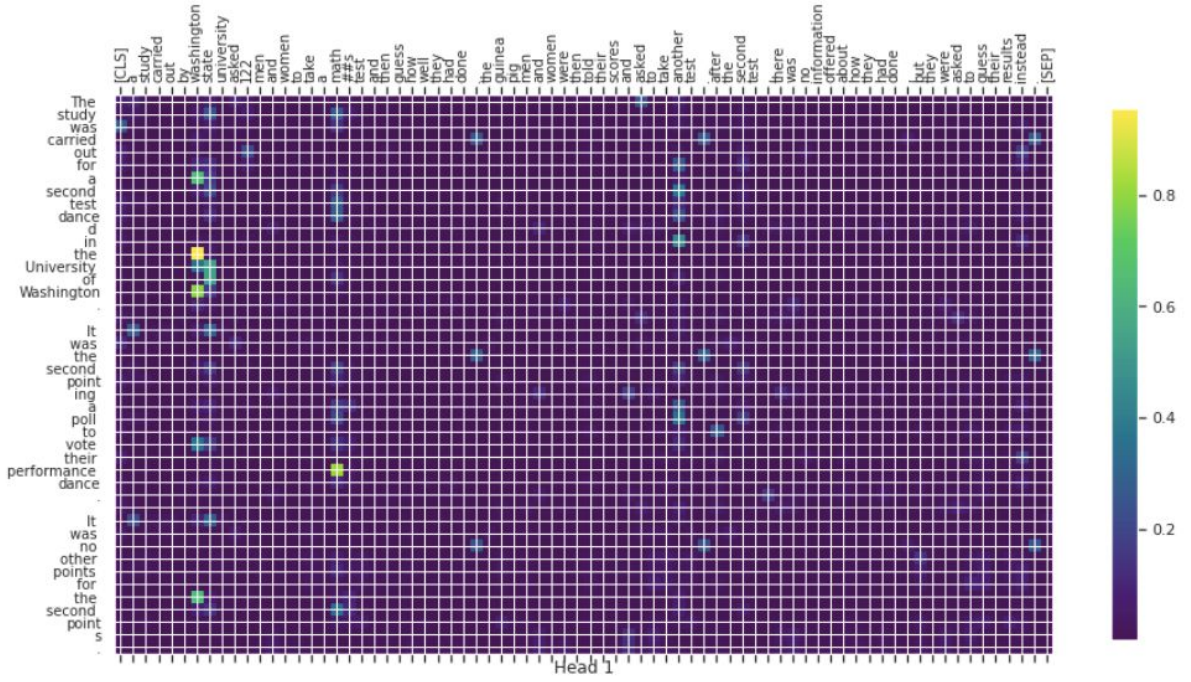
Results

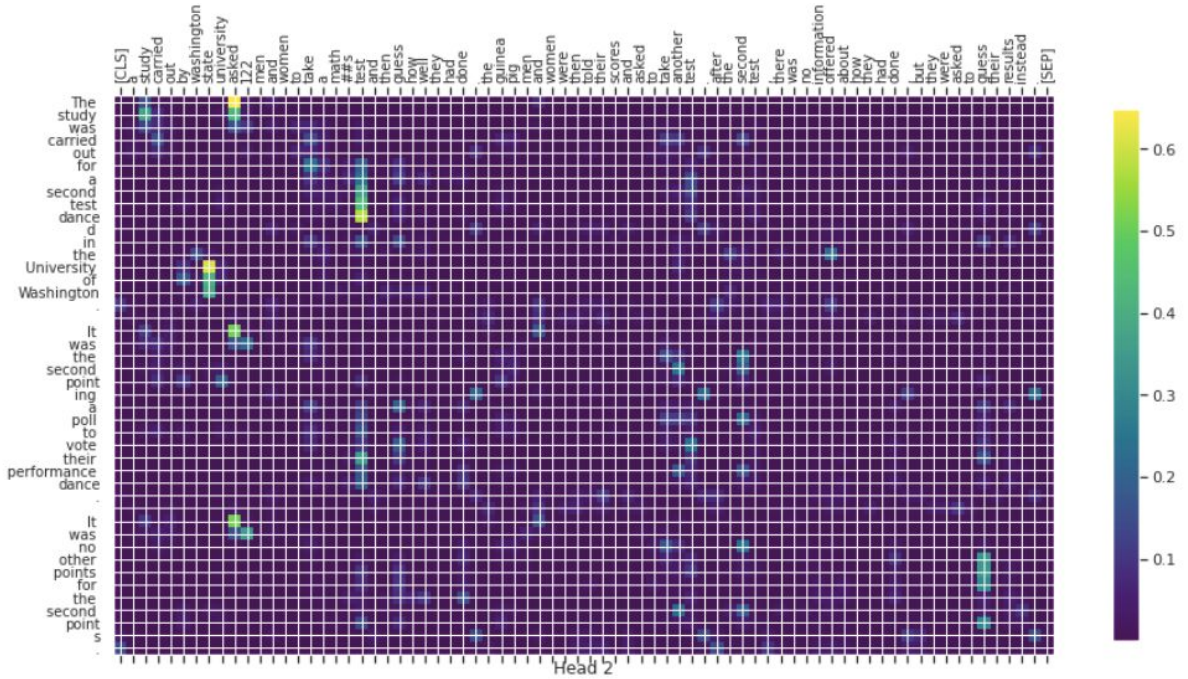
We use the previously defined evaluation function in a summarising function to get the predictions and the attention weights. To obtain the predicted summary, we use the decode of the summary tokenizer. A `plot` argument is added inside the summarization function to optionally plot the attention weights of a given block of a given layer (attention heatmap). The resulting summaries were the following:

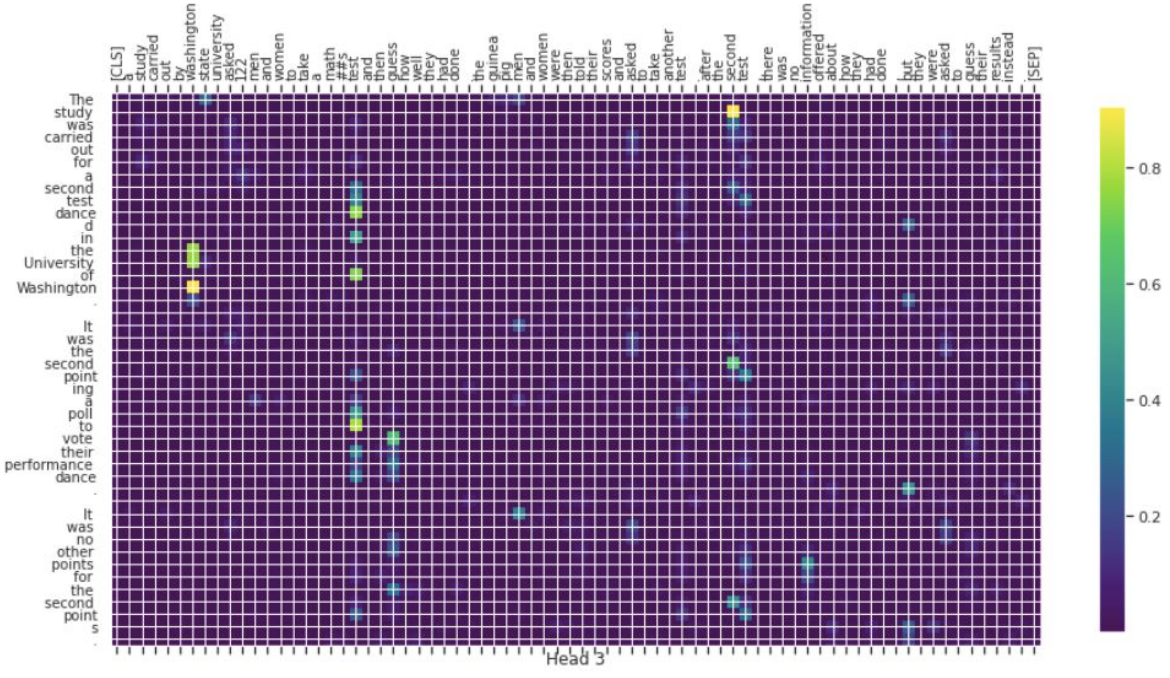
```
Input: A study carried out by Washington State University asked 122 men
and women to take a maths test and then guess how well they had done.
The guinea pig men and women were then told their scores and asked to
take another test. After the second test, there was no information
offered about how they had done, but they were asked to guess their
results instead.

Predicted summary: The study was carried out for a second test danced
in the University of Washington .
It was the second pointing a poll to vote their performance dance .
It was no other points for the second points .
```

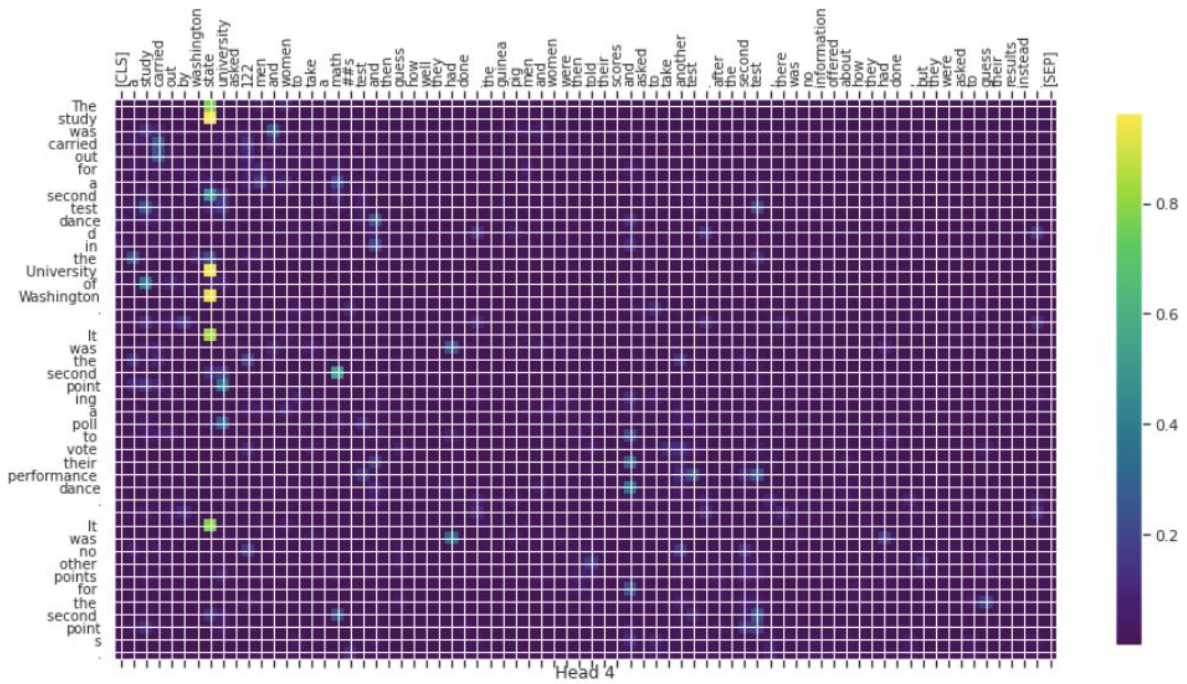
We can plot the attention weights and see which parts of the text the model is focusing on in a given time in a specific layer in a particular attention block. Here, for instance, we can see that, for various heads, the model will focus on a specific group of words. Many of the following attention heads here are linking the words ‘state’ and ‘university’ with `Washington`, they focus on the relationship between these groups of words to extract the information that the text is about a university in Washington.

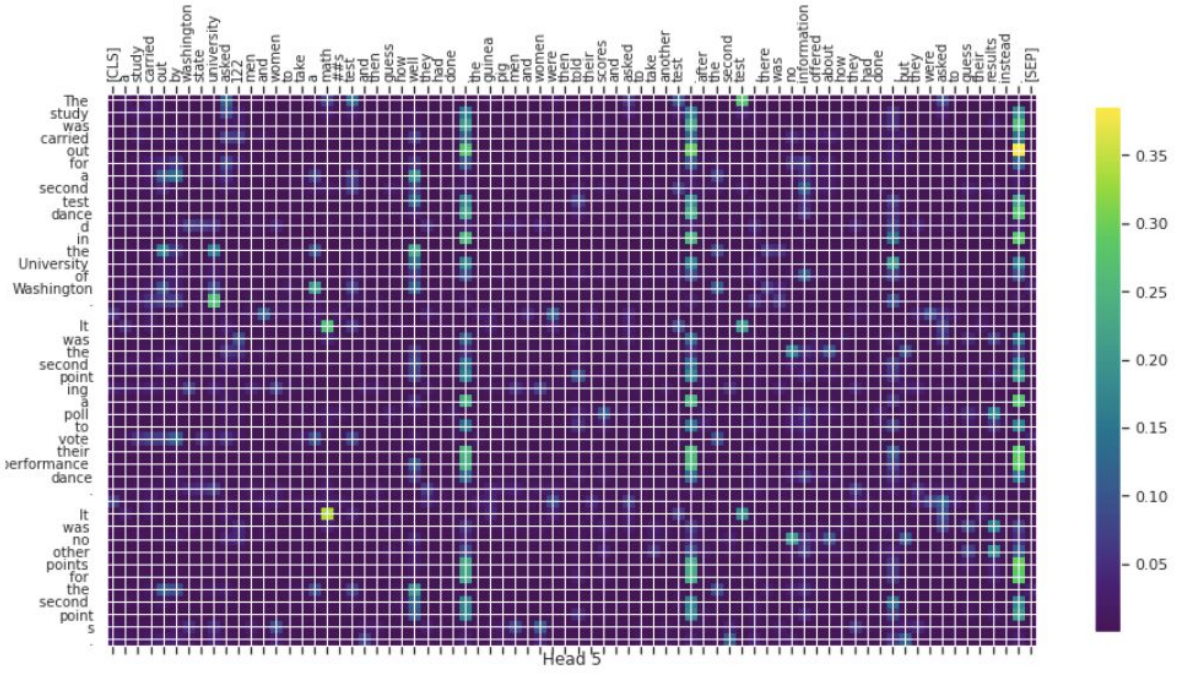
Other heads link the word `women` with the word ‘study’, and ‘asked’ was associated by the group of words ‘was carried out’. this shows the model’s potential to focus on specific areas of text, the various ways they can be linked to ‘understand’ the text and finally outputting its own ‘interpretation’ of the input.

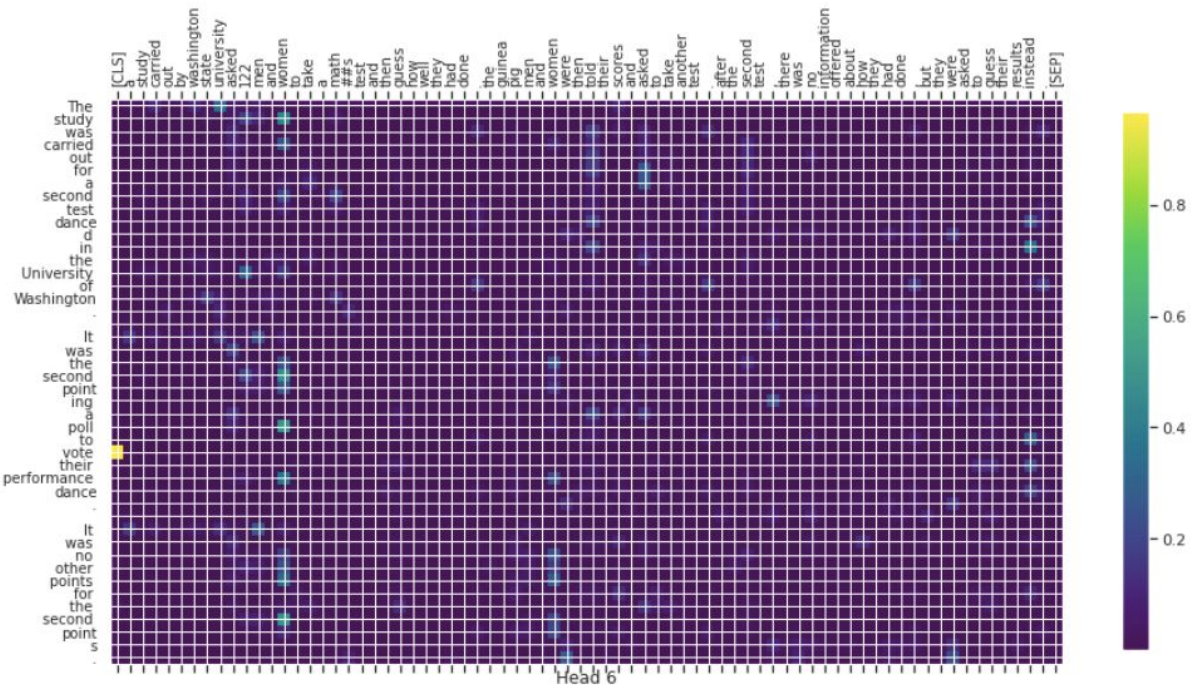


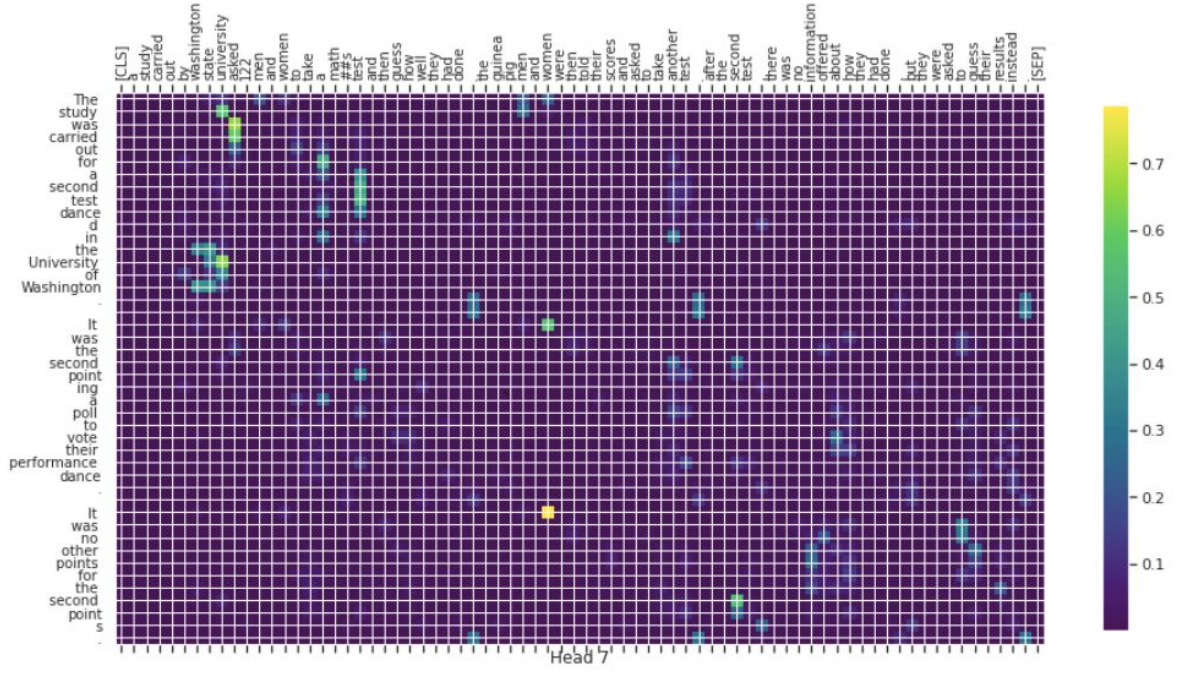

[CLS] a study carried out by washington state university asked 122 men and women to take a math ##s test and then guess how well they had done the guinea pig men and women were then told their scores and asked to take another test after the second test there was no information offered about how they had done but they were asked to guess their results instead [SEP]
The study was carried out for a second test dance d in the University of Washington .
It was the second point ing a poll to vote their performance dance .
It was no other points for the second point s .
Head 2
0.6
0.5
0.4
0.3
0.2
0.1

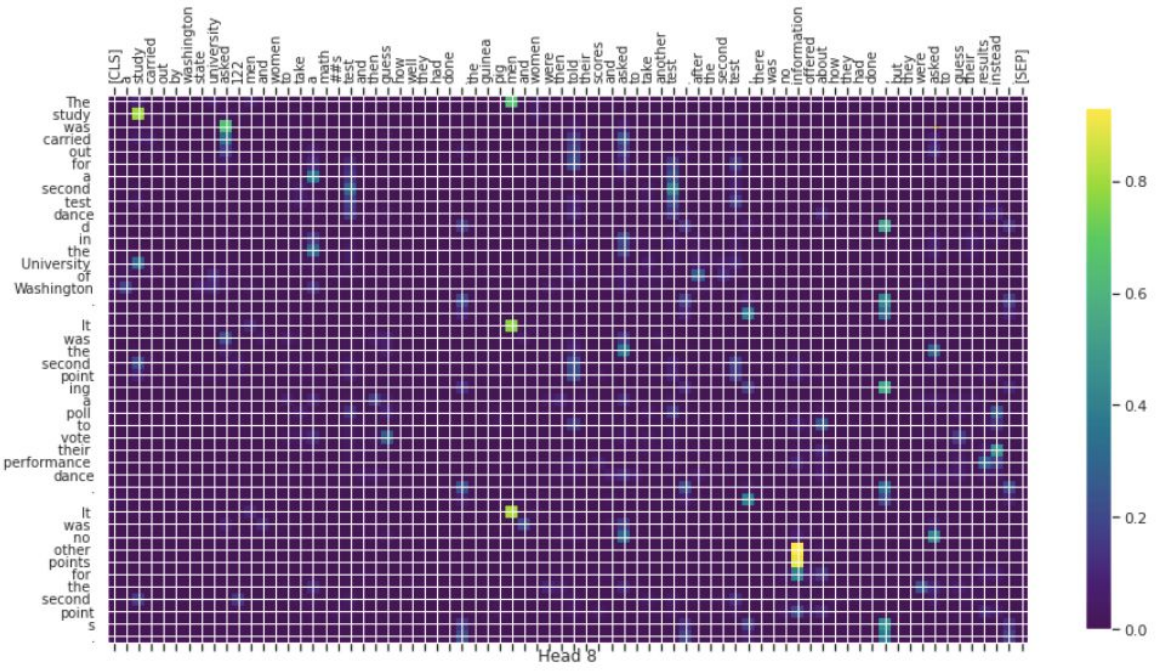

[CLS] a study carried out by washington state university asked 122 men and women to take a math ##s test and then guess how well they had done the guinea pig men and women were then told their scores and asked to take another test after the second test there was no information offered about how they had done but they were asked to guess their results instead [SEP]
The study was carried out for a second test dance d in the University of Washington .
It was the second point ing a poll to vote their performance dance .
It was no other points for the second point s .
Head 3
0.8
0.6
0.4
0.2


Head 4
0.8
0.6
0.4
0.2


Head 5
0.35
0.30
0.25
0.20
0.15
0.10
0.05


[CLS] a study carried out by washington state university asked 122 men and women to take a math ##s test and then guess how well they had done the guinea pig men and women were then told their scores and asked to take another test after the second test there was no information offered about how they had done but they were asked to guess their results instead [SEP]
The study was carried out for a second test dance d in the University of Washington . It was the second point ing a poll to vote their performance dance . It was no other points for the second point s .
0.8
0.6
0.4
0.2
Head 6


[CLS] a study carried out by washington state university asked 122 men and women to take a math ##s test and then guess how well they had done the guinea pig men and women were then told their scores and asked to take another test after the second test there was no information offered about how they had done but they were asked to guess their results instead [SEP]
The study was carried out for a second test dance d in the University of Washington . It was the second point ing a poll to vote their performance dance . It was no other points for the second point s .
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Head 7



Discussion

To improve accuracy, the model could be trained over more epochs, more data and the model likely needs more hyperparameter adjustments. Here the hyperparameters were kept simple and training data was kept minimal due to colab limitations. But using bert with transformers for summarization tasks has shown promising results. Further exploration is needed to fully be able to leverage the power of transfer learning with bert and the transformer architecture for abstractive summarization tasks.

References

- Text Summarization with Pre Trained Encoders
- On Extractive and Abstractive Neural Document Summarization with Transformer Language Models
- Attention Is All You Need
- Inner Workings of Attention
- Pre trained BERT Model
- BERT Classifier Example
- Transformer Model for Natural Language Understanding
- BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding
- Hands-On Machine Learning with Scikit-Learn Keras and Tensorflow
- Deep Contextualized Word Representations