Attention for Mental Health

George Pinto 06/2021

Executive Summary

This report explores the feasibility of creating a state-of-the-art question and answer transformer model to assist with questions that users might have regarding mental health during the Covid-19 pandemic. Rather than comparing models we focus on reproducing the research paper “Attention is all you need”, using a limited set of mental health data (a situation many data science practitioners could be faced with). The architecture follows the paper very closely, and even though the paper demonstrates a translation task, here it is used for a question-and-answer task. It would be expected that users sitting at home during the pandemic would just want to enter the question and get an answer. Many Transformer architectures ask for the question and context (showing language understanding of the provided context) but to create a useful application in this particular setting, users would be much more interested in getting the correct answer (from reliable sources) quickly and with proper grammatical structure rather than visualizing what the model can understand from a larger text. The Dataset consists of 98 question and answer pairs and was prepared by https://www.kaggle.com/narendrageek with the following sources:

https://www.thekimfoundation.org/faqs/

https://www.mhanational.org/frequently-asked-questions

https://www.wellnessinmind.org/frequently-asked-questions/

https://www.heretohelp.bc.ca/questions-and-answers

The model was designed using TensorFlow 2 following the “Attention is all you Need” paper parameters. The dataset structure was slightly changed to make sure each answer was within the context of each question. Each question-and-answer pair was rewritten to answer one idea or concept at a time. This way, in terms of language, the answer contextually matched the question and relationships could be calculated more effectively by the model. This significantly helped the readability of predictions. Structuring question and answer pairs this way expanded the number of question-and-answer pairs from 98 to 158. Answers had to also be reduced in size without sacrificing the main idea to provide approximate the number of words that fit within the scope of the maximum length internal parameter so that truncation would not cut off the predictions and make them non-sensical. A maximum answer length of 26 words (matching question and answer pair lengths) was used in the final iteration. Finally, the embedding size was increased to 1024 from 512 (base Transformer parameters) and the number of encoding layers was reduced to 4 for a final size of approximately 87 million parameters. These changes also significantly improved final prediction readability. The model was developed using Google Colab as using GPUs was necessary, in the beginning training was slow, but increasing the batch size helped tremendously with the speed of iterations (due to vectorization I presume) using a final batch size of 512. The model gained additional prediction speed when checkpoints were removed, since even on Colab, saving the check points exceeded the 2TB space allotted.

Because this experiment requires replicating a research paper for a specific task using a limited data set, a traditional train test split would not be helpful here, the questions that could be submitted to the

model would only be those used in training, so the recommendation is to reduce the scope to those questions that score highly on the chosen metric during training and test by rephrasing the questions. Training required the inclusion of every question because the questions differ from each other (this situation could change if enough resources are available to generate a large enough data set, or if transfer learning is employed using a pre-trained model from which the model inherits grammatical articulations of the language in a very complete manner, however, in that scenario, the insights obtained from this experiment would not necessarily be apparent and I believe these insights could be very useful in a situation in which data is limited or not readily available). F1 scores were used to conduct a rigorous review of predictions sentence by sentence:

F1= 2precisionrecall/(precision+recall) precision = tp/(tp+fp) recall=tp/(tp+fn)

precision = 1.0 * num_same / len(pred_toks)=tp/(tp+fp)

recall = 1.0 * num_same / len(gold_toks)=tp/(tp+fn)

tp=number of tokens that are shared between the correct answer and the prediction

fp=number of tokens that are in the prediction but not in the correct answer

fn=number of tokens that are in the correct answer but not in the prediction

https://kierszbaumsamuel.medium.com

Sentence by sentence F1 scores:

F1_Score
0.882302944
0.999950002
0.961488464
0.999950002
0.961488464
0.905610398
0.999950002
0.791616757
0.641459456
0.999950002
0.999950002
0.999950002
0.999950002
0.999950002
0.999950002
0.999950002
0.999950002
0.88883896
0.622172251
0.652123917

0.162112506
0.999950002
1
0.933283361
0.893567047
0.999950002
0.703653776
0.649950129
0.999950002
0.926779301
0.999950002
0.749953128
0.565167396
0.482709161
0.814764886
0.961488464
0
0.133283574
0.851801923
0.438977994
0.846103849
0.923026926
0.545404699
0.999950002
0.374950354
0.905610398
0.999950002
0.961488464
0.884565387
0.961488464
0.884565387
0.999950002
0.719950083
0.905610398
0.88883896
0.923026926
0.999950002
0.999950002
0.999950002
0.692257696
0.941126492
0.199950137
0.943346247

0.999950002
0.999950002
0.958283336
0.583283685
0.943346247
0.999950002
0.923026926
0.959950003
0.961488464
0.961488464
0.88883896
0.784263748
0.943346247
0.943346247
0.943346247
0.959950003
0.923026926
0.961488464
0.961488464
0.999950002
0.999950002
0.97430901
0.823479434
0.692257696
0.812450003
0.999950002
0.999950002
0
0
0.999950002
0.999950002
0.999950002
0.923026926
0.923027
0.943346247
0.599950084
0.959950003
0.897909207
0.260819669
0.961488464
0.823479434
0.784263748
0.784263748

0.784263748
0.879950083
0.461488467
0.769180772
0.943346247
0.943346247
0.999950002
0.943346247
0.943346247
0.170163346
0.599950084
0.627401004
0.823479434
0.86269512
0.961488464
0.730719234
0.730719234
0.41661702
0.941126492
0
0.943346247
0.943346247
0.299950008
0.618131971
0.618131971
0.943346247
0.961488464
0.961488464
0.961488464
0.714235973
0.272677695
0.943346247
0.857092881
0.943346247
0.745048062
0.66661669
0.905610398
0.961488464
0.923026926
0.905610398
0.848434897
0.148098782
0.941126492

0.941126492
0.959950003
0.943346247
0.559950004
0.999950002
0.872677424
0
0.923026926
0.66661669

We can see that most training predictions had an F1 score above 90%, the overall average F1 score for training was 81%. A few of the predictions really did not match the gold standard at all, therefore the overall average score was lower than the score obtained by most predictions. I think the scope of model predictions should be clearly defined to be within the question answer pairs that scored 90% or above, meaning only those or similar questions should be asked. This could always be improved by refining the quality of the question-and-answer pairs and/or increasing the size of the data set.

It is important to understand that the encoder portion of the transformer is creating the prediction word by word and that it uses contextual connections between words. Achieving a high F1 score in training creates very robust question understanding compared to, for example sequence models.

For testing, a couple of questions with an F1 score of 90% or above were chosen to create out of test questions (this was simply done by rephrasing the questions in a way that the model had not seen), the model achieved F1 scores of 0.92 and 0.81 on these questions, with the model giving very robust answers (from a human perspective when reading the answers). The F1 score method, sentence by sentence, is much more robust than accuracy, for example. This final model had a final training accuracy of 92.63% but previous iterations had a much higher accuracy and much lower F1 scores with many unintelligible sentences. More focus had to be placed on tuning and on the quality of the dataset for higher F1 scores. Focusing on higher F1 scores and on reading each response sentence from a human perspective increased the number of responses that demonstrate language understanding.

Data Exploration and Processing

The original data set contained 98 question and answer pairs on mental health data. The first issues encountered were encoding errors which were removed using pandas:

```
df['Answers'][0]
```

```
'Mental illnesses are health conditions that disrupt a personâ€™s thoughts, emotions, relationships, and daily functioning. Th
ey are associated with distress and diminished capacity to engage in the ordinary activities of daily life.\nMental illnesses
fall along a continuum of severity: some are fairly mild and only interfere with some aspects of life, such as certain phobia
s. On the other end of the spectrum lie serious mental illnesses, which result in major functional impairment and interference
with daily life. These include such disorders as major depression, schizophrenia, and bipolar disorder, and may require that t
he person receives care in a hospital.\nIt is important to know that mental illnesses are medical conditions that have nothing
to do with a personâ€™s character, intelligence, or willpower. Just as diabetes is a disorder of the pancreas, mental illness
is a medical condition due to the brainâ€™s biology.\nSimilarly to how one would treat diabetes with medication and in...'
```

```
# There are some encoding errors we are going to need to fix


df['Answers'] = df['Answers'].map(lambda x: x.encode('ascii', errors = 'replace').decode('utf-8'))
df['Answers'][0]
```

```
'Mental illnesses are health conditions that disrupt a person???s thoughts, emotions, relationships, and daily functioning. Th
ey are associated with distress and diminished capacity to engage in the ordinary activities of daily life.\nMental illnesses
fall along a continuum of severity: some are fairly mild and only interfere with some aspects of life, such as certain phobia
```

Issues not removed using encoding methods were removed using the string replace method:

```
df['Answers'] = df['Answers'].map(lambda x: x.replace('\n', ' '))
```

```
df['Answers'] = df['Answers'].map(lambda x: x.replace("???", "'"))
```

```
df['Answers'] = df['Answers'].map(lambda x: x.replace(" ? s", "'s"))
```

```
df['Answers'][0]
```

```
'Mental illnesses are health conditions that disrupt a person's thoughts, emotions, relationships, and daily functioning. The
are associated with distress and diminished capacity to engage in the ordinary activities of daily life. Mental illnesses fal
along a continuum of severity: some are fairly mild and only interfere with some aspects of life, such as certain phobias. On
the other end of the spectrum lie serious mental illnesses, which result in major functional impairment and interference with
daily life. These include such disorders as major depression, schizophrenia, and bipolar disorder, and may require that the p
rson receives care in a hospital. It is important to know that mental illnesses are medical conditions that have nothing to d
with a person's character, intelligence, or willpower. Just as diabetes is a disorder of the pancreas, mental illness is a me
ical condition due to the brain's biology. Similarly to how one would treat diabetes with medication and insulin, me...'
```

After processing, start and end characters were created for tokenization

```
# Add start and end tokens to sentences

df['Questions'] = ["<start> " + utils.preprocess_sentence(sentence) + " <end>" for sentence in df['Questions'].values.tolist
df['Answers'] = ["<start> " + utils.preprocess_sentence(sentence) + " <end>" for sentence in df['Answers'].values.tolist()]
```

```
df['Questions'][0]
```

```
'<start> what does it mean to have a mental illness ? <end>'
```

```
df['Answers'][0]
```

```
'<start> mental illnesses are health conditions that disrupt a person's thoughts , emotions , relationships , and daily func
oning . they are associated with distress and diminished capacity to engage in the ordinary activities of daily life . menta
illnesses fall along a continuum of severity some are fairly mild and only interfere with some aspects of life , such as cer
in phobias . on the other end of the spectrum lie serious mental illnesses , which result in major functional impairment and
nterference with daily life . these include such disorders as major depression , schizophrenia , and bipolar disorder , and
y require that the person receives care in a hospital . it is important to know that mental illnesses are medical conditions
hat have nothing to do with a person's character , intelligence , or willpower . just as diabetes is a disorder of the pancr
s , mental illness is a medical condition due to the brain's biology . similarly to how one would treat diabetes with m...'
```

Because of the large sentence structure and multi-concept nature of the answers, a customized data set was created taking into consideration that the answer would be truncated at 26 characters. In prior iterations of the model, higher word counts gave nonsensical predictions. If we think of each character in the sentence as a column, 30 was too much for our compute resources (the maximum length was set to 26 characters which gave the best results). These parameters are best set by running iterations of the model and noting where best results occur. The model is creating context relationships in the multi-head attention step. Our predictions, which are created using the learned context word by word in the encoder need to make sense within the 26-character maximum sequence size. For this purpose, it was important to reduce sentence size in the pre-processed data set further. To obtain good context out of the question-and-answer pairs an effort was made to make sure the answer was responding to just one question (one concept at a time) for each answer.

Here is an example showing the first question before editing:

<start> what does it mean to have a mental illness ? <end>

<start> mental illnesses are health conditions that disrupt a person's thoughts , emotions , relationships , and daily functioning . they are associated with distress and diminished capacity to engage in the ordi nary activities of daily life . mental illnesses fall along a continuum of severity some are fairly mild and o nly interfere with some aspects of life , such as certain phobias . on the other end of the spectrum lie se rious mental illnesses , which result in major functional impairment and interference with daily life . thes e include such disorders as major depression , schizophrenia , and bipolar disorder , and may require t hat the person receives care in a hospital . it is important to know that mental illnesses are medical con ditions that have nothing to do with a person's character , intelligence , or willpower . just as diabetes is a disorder of the pancreas , mental illness is a medical condition due to the brain's biology . similarly to how one would treat diabetes with medication and insulin , mental illness is treatable with a combinatio n of medication and social support . these treatments are highly effective , with percent of individuals re ceiving treatment experiencing a reduction in symptoms and an improved quality of life . with the proper treatment , it is very possible for a person with mental illness to be independent and successful . <end>

This question can be broken down into several question answer pairs which can be checked against other question answer pairs in the dataset to prevent repetition. Broken down this way, the model can more effectively learn contextual patterns from questions to answers:

<start> what is mental illness ? <end>

<start> mental illnesses are health conditions that disrupt a person s thoughts , emotions , relationships , and daily functioning . <end>

<start> what does mental illness cause ? <end>

<start> mental illnesses cause distress and diminished capacity to engage in the ordinary activities of d aily life . <end>

<start> who does mental illness affect ? <end>

<start> mental illness can affect anyone regardless of gender , age , income , social status , ethnicity , r eligion , sexual orientation , or background . <end>

<start> what causes mental illness ? <end>

<start> possible causes of mental illness include genetics, infections, brain defects or injury, prenatal d amage, substance abuse and other factors . <end>

<start> can people with mental illness recover ? <end>

<start> people can recover from mental illness but early identification and treatment are of vital importa nce . <end>

<start> how can people with mental illness recover ? <end>

<start> people can recover from mental illness through a wide range of effective treatments that are sp ecific to the particular type of mental illness . <end>

Formatting the questions this way is conducive to better contextual mapping and robust, intelligible predictions that are within the scope of our limitations (maximum length of 26 words). This could be further refined by ensuring the grammatical consistency of each sentence for example, and creating a question and answer set for every single concept in the data set if time and resources allow.

Tokenization

Sequences needed to be numeric before being fed into the model, once the dataset formatting and preprocessing had been completed, TensorFlow was used for tokenization of all question-and-answer pairs:

```
answers = df['Answers'].values.tolist()
```

```
# Tokenizer for answers

tokenizer_answers = tf.keras.preprocessing.text.Tokenizer(num_words=None, filters='', # list of characters
                                                          lower=True)                 # to filter is empty
tokenizer_answers.fit_on_texts(answers)                                               # string

answers_sequence = tokenizer_answers.texts_to_sequences(answers)
```

Padding

We noted that questions were generally much shorter in length than answers in the dataset. The model will expect equal length question and answer pairs (as explained earlier, it was found that a particularly good maximum length, so as not to overwhelm the memory and get good intelligible prediction sequences within this chosen length, was 26 characters). To input sequences shorter than 26 characters, empty spaces needed to be replaced with 0 padding, and the same maximum sequence length was set for all sequences, TensorFlow was used for this purpose as well:

```
# Create padding so that we keep the sequences at the same length and establish a max length
MAX_LENGTH = 26

questions = tf.keras.preprocessing.sequence.pad_sequences(questions_sequence,
                                                          value=0,
                                                          padding='post',
                                                          maxlen=MAX_LENGTH)
answers = tf.keras.preprocessing.sequence.pad_sequences(answers_sequence,
                                                        value=0,
                                                        padding='post',
                                                        truncating='post',
                                                        maxlen=MAX_LENGTH)
```

```
answers[0]
```

```
array([  7,  12, 195,  21,  15, 243,  22, 457,   4,  74,  47,  91,   2,
       126,   2, 127,   2,   5, 111, 458,   1,   8,   0,   0,   0,   0],
      dtype=int32)
```

TensorFlow Dataset

The model could run very slowly depending on the set up here, since the purpose of the project was to develop from a research paper, the most important aspect was being able to figure out how to iterate quickly and effectively while following the paper. Smaller batch sizes were taking hours with just this small dataset and through iteration it was found (probably due to vectorization) that large batch sizes were ideal, ending with a batch size of 512. Also, it was noted that saving checkpoints not only slowed down training quite a bit, but also prevented the completion of training altogether due to running out of memory, with saved checkpoints going well over 2 TB. Checkpoints were removed and iterations became quick and efficient. At this point, it was concluded that the focus of the project was to develop the dataset for performance and to use the research paper to obtain successful results on the training metrics, basically to figure out how to navigate a problem like this effectively. Developing the application beyond this point would be the focus of future studies with different tools, this could require expanding the dataset extensively and using a cloud framework like CPG or AWS with additional memory and compute resources.

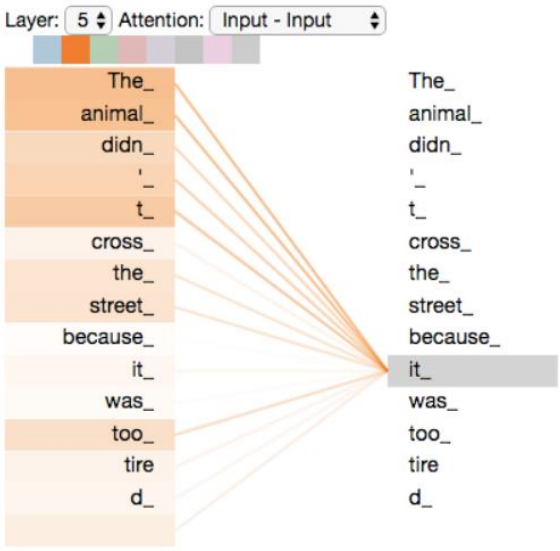
```
# Create the dataset, batch size and improve accessibility to data during training

BUFFER_SIZE = 1000
BATCH_SIZE = 512
dataset = tf.data.Dataset.from_tensor_slices((questions, answers))
dataset = dataset.cache()
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)
```

GPUs

Before running several iterations of our deep learning model to compare performance with changes to the dataset, parameters, architecture and hyperparameter changes, it is of utmost importance to mention that running on a GPU is an absolute necessity. Time as a resource will be prohibitive without the use of a GPU. The notebooks were run on Google Colab, which has the option of running on a GPU.

Positional Embeddings

This is not a sequential model, in fact the entire sequence gets fed into the model at once, what we use is attention, which measures contextual relationships between words, on this image we see attention weights represented as orange lines, the thicker lines with stronger color are stronger relationships between words https://jalammar.github.io/illustrated-transformer:



As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

The base model in the paper uses embeddings of size 512. It is important to note here that during model iterations larger embeddings were more effective and we used a 1024-dimension embeddings such as those used in the larger Transformer in the paper. TensorFlow has an off the shelf embedding layer and we can set the dimensions we need. However, because we only use attention as mentioned above and we do not have time information, input embeddings will need some way of representing the distance between words and to figure out word order. For this purpose, we add a positional encoding component to the embeddings. We follow the research paper and the formula the authors share to accomplish this https://papers.nips.cc/paper/2017/file:

In this work, we use sine and cosine functions of different frequencies:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

Here we create a positional encoding class inheriting from the layer class in TensorFlow, our maximum word length is 26, pos is all possible integers from 0 to 25 and i is all the possible dimensions in our chosen embedding dimensions, even numbers will use sin and odd numbers will use cosine. In essence positions that are close to each other will have similar numbers according to this formula. This information or time signal regarding proximity or order of the words is added to the input embeddings:

```
class PositionalEncoding(layers.Layer):
    def __init__(self):
        super(PositionalEncoding, self).__init__()

    def get_angles(self, pos, i, d_model): # pos: (seq_len, 1), i: (1, d_model)
        angles = 1/np.power(10000., (2*(i//2))/np.float32(d_model))
        return pos*angles # (seq_len, d_model)
```

```
    def __call__(self, inputs):
        seq_length = inputs.shape.as_list()[-2]
        d_model = inputs.shape.as_list()[-1]
        angles = self.get_angles(np.arange(seq_length)[:, np.newaxis],
                                 np.arange(d_model)[np.newaxis, :],
                                 d_model)
        angles[:, 0::2] = np.sin(angles[:, 0::2])
        angles[:, 1::2] = np.cos(angles[:, 1::2])
        pos_encoding = angles[np.newaxis, ...]

        return inputs + tf.cast(pos_encoding, tf.float32)
```

Attention

To code attention we follow the paper carefully, using dot product attention with a scaling factor, explained here https://papers.nips.cc/paper/2017/file:

In practice, we compute the attention function on a set of queries simultaneously, packed together into a matrix Q. The keys and values are also packed together into matrices K and V. We compute the matrix of outputs as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \quad (1)$$

The two most commonly used attention functions are additive attention [2], and dot-product (multiplicative) attention. Dot-product attention is identical to our algorithm, except for the scaling factor of $\frac{1}{\sqrt{d_k}}$. Additive attention computes the compatibility function using a feed-forward network with a single hidden layer. While the two are similar in theoretical complexity, dot-product attention is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code.

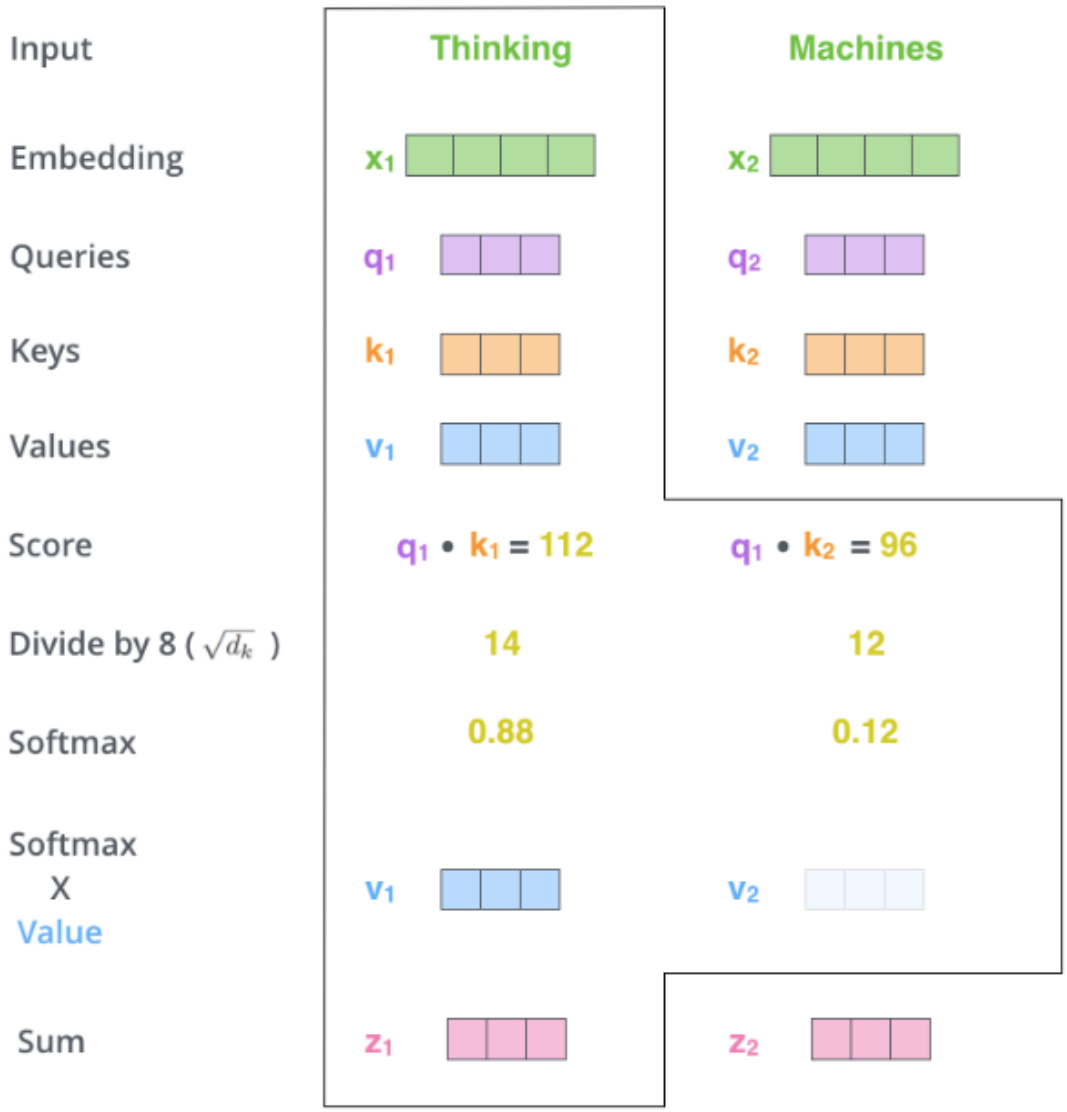
While for small values of d_k the two mechanisms perform similarly, additive attention outperforms dot product attention without scaling for larger values of d_k [3]. We suspect that for large values of d_k, the dot products grow large in magnitude, pushing the softmax function into regions where it has extremely small gradients [4]. To counteract this effect, we scale the dot products by $\frac{1}{\sqrt{d_k}}$.

We can write a function using python and TensorFlow to efficiently multiply the matrices involved, keep in mind that inputs queries, keys and values are dot products of our embedding vectors (plus the positional encodings) and the weights plus the biases (the weights are learned through back propagation). The mask component could be either a mask preventing the decoder from looking at words in the future or just a mask for the zeros at the end of each sentence to make sure they are not contributing to the results going into the SoftMax function (depending on when you use this function):

```
def scaled_dot_product_attention(queries, keys, values, mask):
    product = tf.matmul(queries, keys, transpose_b=True)
    keys_dim = tf.cast(tf.shape(keys)[-1], tf.float32)
    scaled_product = product / tf.math.sqrt(keys_dim)
    if mask is not None:
        scaled_product += mask * -1e9
    attention = tf.matmul(tf.nn.softmax(scaled_product, axis=-1), values)
```

```
    return attention
```

Attention is best summarized in the following diagram https://jalammar.github.io/illustrated-transformer in which the final dot product is a vector representation of the strength of each relationship between each word and the other words in the sentence:

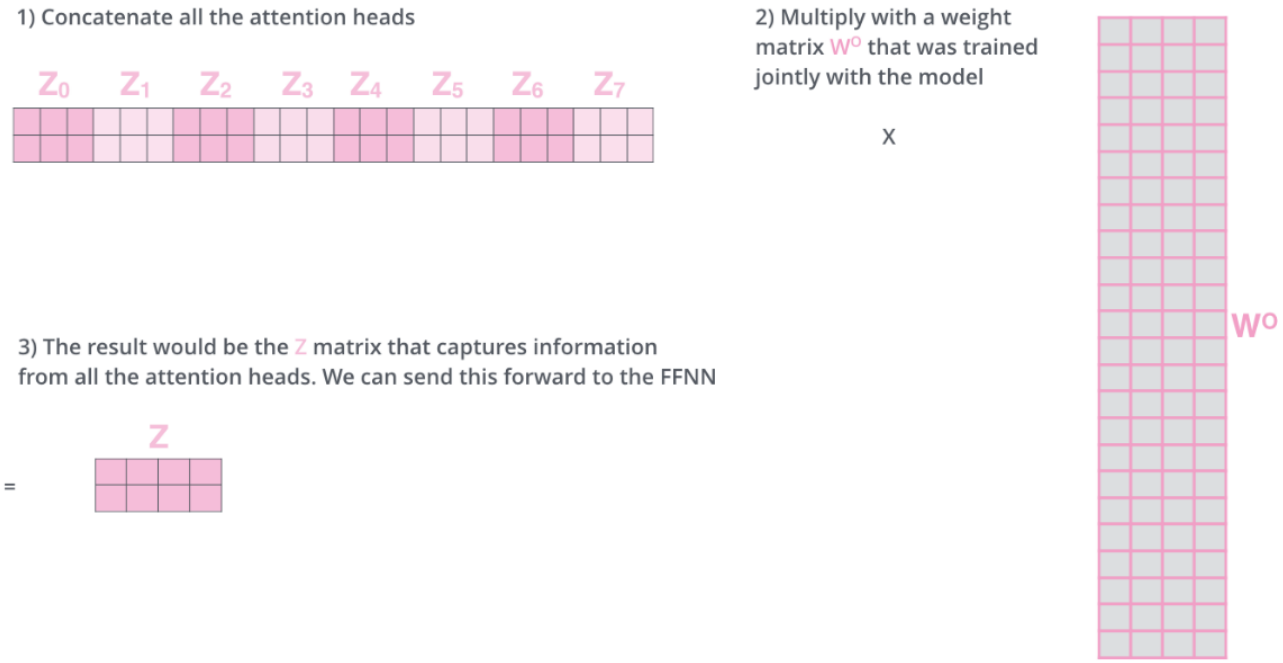


Multi-Head Attention

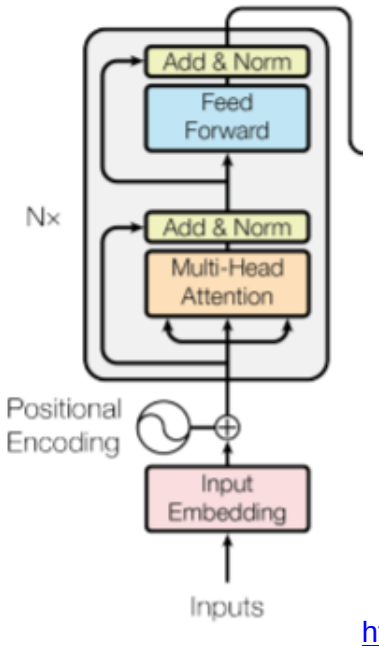
The vector representation we generated through attention represents a single scaled attention vector for each word. Certain word relationships might be stronger in this representation, and we might overlook other important relationships. To get a more complete sense of the context of each word we perform multi-head attention, in which attention is performed independently (we know the Q, K, V weights are initialized randomly) 8 times for each word. The 8 separate attention results are concatenated and multiplied by another learnable weight matrix and then passed through a linear layer, once again we can inherit from the layer class using TensorFlow to code this up:

```
class MultiHeadAttention(layers.Layer):

    def __init__(self, nb_proj):
        super(MultiHeadAttention, self).__init__()
        self.nb_proj = nb_proj

    def build(self, input_shape):
        self.d_model = input_shape[-1]
        assert self.d_model % self.nb_proj == 0

        self.d_proj = self.d_model // self.nb_proj

        self.query_lin = layers.Dense(units=self.d_model)
        self.key_lin = layers.Dense(units=self.d_model)
        self.value_lin = layers.Dense(units=self.d_model)

        self.final_lin = layers.Dense(units=self.d_model)

    def split_proj(self, inputs, batch_size): # inputs: (batch_size,
seq_length, d_model)
        shape = (batch_size,
                 -1,
                 self.nb_proj,
                 self.d_proj)
        splited_inputs = tf.reshape(inputs, shape=shape) # (batch_size,
seq_length, nb_proj, d_proj)
        return tf.transpose(splited_inputs, perm=[0, 2, 1, 3]) # (batch_size,
nb_proj, seq_length, d_proj)

    def call(self, queries, keys, values, mask):
        batch_size = tf.shape(queries)[0]

        queries = self.query_lin(queries)
        keys = self.key_lin(keys)
        values = self.value_lin(values)

        queries = self.split_proj(queries, batch_size)
        keys = self.split_proj(keys, batch_size)
        values = self.split_proj(values, batch_size)

        attention = scaled_dot_product_attention(queries, keys, values, mask)

        attention = tf.transpose(attention, perm=[0, 2, 1, 3])

        concat_attention = tf.reshape(attention,
                                      shape=(batch_size, -1, self.d_model))

        outputs = self.final_lin(concat_attention)

        return outputs
```

Also best summarized here https://jalammar.github.io/illustrated-transformer:



Encoder



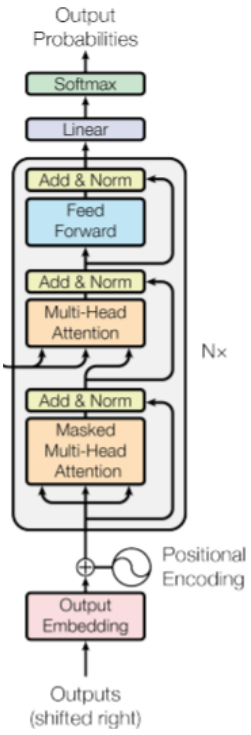
https://papers.nips.cc/paper/2017/file

To code up the encoder architecture from the paper we inherit from the layer class again, first creating a single encoding layer and then creating the class for the actual encoder with multiple encoding layers. The base model in the paper uses 6 encoding layers, this model worked best with 4, also note the residual layers with batch normalization which are conducted after applying dropout (to prevent overfitting during training only) using recommended settings in the research paper, also 2048 feed forward units were used as in the base model:

```
class EncoderLayer(layers.Layer):

    def __init__(self, FFN_units, nb_proj, dropout_rate):
        super(EncoderLayer, self).__init__()
        self.FFN_units = FFN_units
        self.nb_proj = nb_proj
        self.dropout_rate = dropout_rate

    def build(self, input_shape):
        self.d_model = input_shape[-1]

        self.multi_head_attention = MultiHeadAttention(self.nb_proj)
        self.dropout_1 = layers.Dropout(rate=self.dropout_rate)
        self.norm_1 = layers.LayerNormalization(epsilon=1e-6)

        self.dense_1 = layers.Dense(units=self.FFN_units, activation="relu")
        self.dense_2 = layers.Dense(units=self.d_model)
        self.dropout_2 = layers.Dropout(rate=self.dropout_rate)
        self.norm_2 = layers.LayerNormalization(epsilon=1e-6)

    def call(self, inputs, mask, training):
        attention = self.multi_head_attention(inputs,
                                              inputs,
                                              inputs,
                                              mask)
        attention = self.dropout_1(attention, training=training)
        attention = self.norm_1(attention + inputs)

        outputs = self.dense_1(attention)
        outputs = self.dense_2(outputs)
        outputs = self.dropout_2(outputs, training=training)
        outputs = self.norm_2(outputs + attention)

        return outputs



class Encoder(layers.Layer):

    def __init__(self,
                 nb_layers,
                 FFN_units,
                 nb_proj,
                 dropout_rate,
                 vocab_size,
```

```
                d_model,
                name="encoder"):
        super(Encoder, self).__init__(name=name)
        self.nb_layers = nb_layers
        self.d_model = d_model

        self.embedding = layers.Embedding(vocab_size, d_model)
        self.pos_encoding = PositionalEncoding()
        self.dropout = layers.Dropout(rate=dropout_rate)
        self.enc_layers = [EncoderLayer(FFN_units,
                                        nb_proj,
                                        dropout_rate)
                           for _ in range(nb_layers)]

    def call(self, inputs, mask, training):
        outputs = self.embedding(inputs)
        outputs *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        outputs = self.pos_encoding(outputs)
        outputs = self.dropout(outputs, training)

        for i in range(self.nb_layers):
            outputs = self.enc_layers[i](outputs, mask, training)

        return outputs
```

Decoder



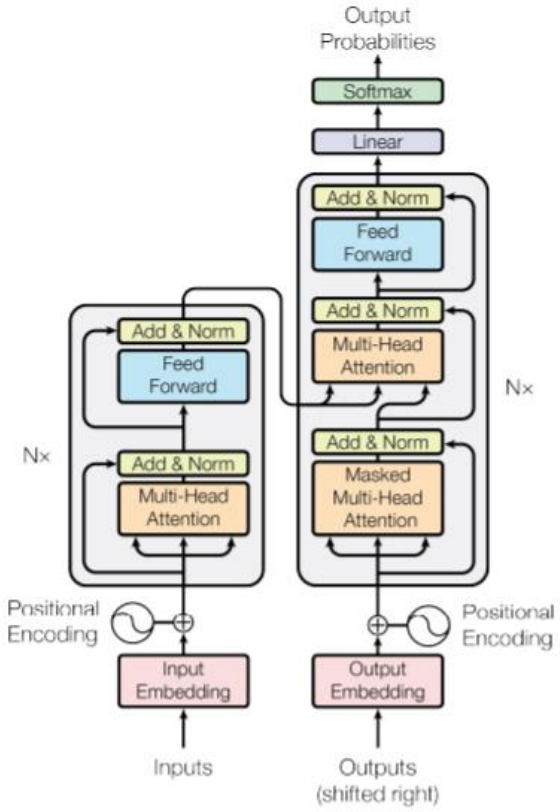
https://papers.nips.cc/paper/2017/file

To code up the decoder architecture from the paper we inherit from the layer class again, first creating a single decoding layer and then creating the class for the actual decoder with multiple decoder layers. The architecture is similar to the encoder architecture also applying dropout and residuals with normalization but here attention is used twice, first using the inputs of the decoder (self-attention) and the second time using both the previous inputs of the decoder and the output of the encoder. Two masks are used: the first mask is for the self-attention layer, while the second mask is for the output of the encoder.

```
class DecoderLayer(layers.Layer):

    def __init__(self, FFN_units, nb_proj, dropout_rate):
        super(DecoderLayer, self).__init__()
        self.FFN_units = FFN_units
        self.nb_proj = nb_proj
        self.dropout_rate = dropout_rate

    def build(self, input_shape):
        self.d_model = input_shape[-1]

        # Self multi head attention
        self.multi_head_attention_1 = MultiHeadAttention(self.nb_proj)
        self.dropout_1 = layers.Dropout(rate=self.dropout_rate)
        self.norm_1 = layers.LayerNormalization(epsilon=1e-6)

        # Multi head attention combined with encoder output
        self.multi_head_attention_2 = MultiHeadAttention(self.nb_proj)
        self.dropout_2 = layers.Dropout(rate=self.dropout_rate)
        self.norm_2 = layers.LayerNormalization(epsilon=1e-6)

        # Feed foward
        self.dense_1 = layers.Dense(units=self.FFN_units,
                                    activation="relu")
        self.dense_2 = layers.Dense(units=self.d_model)
        self.dropout_3 = layers.Dropout(rate=self.dropout_rate)
        self.norm_3 = layers.LayerNormalization(epsilon=1e-6)

    def call(self, inputs, enc_outputs, mask_1, mask_2, training):
        attention = self.multi_head_attention_1(inputs,
                                                inputs,
                                                inputs,
                                                mask_1)
        attention = self.dropout_1(attention, training)
        attention = self.norm_1(attention + inputs)

        attention_2 = self.multi_head_attention_2(attention,
                                                  enc_outputs,
                                                  enc_outputs,
                                                  mask_2)
        attention_2 = self.dropout_2(attention_2, training)
        attention_2 = self.norm_2(attention_2 + attention)

        outputs = self.dense_1(attention_2)
```

```
        outputs = self.dense_2(outputs)
        outputs = self.dropout_3(outputs, training)
        outputs = self.norm_3(outputs + attention_2)

        return outputs



class Decoder(layers.Layer):

    def __init__(self,
                 nb_layers,
                 FFN_units,
                 nb_proj,
                 dropout_rate,
                 vocab_size,
                 d_model,
                 name="decoder"):
        super(Decoder, self).__init__(name=name)
        self.d_model = d_model
        self.nb_layers = nb_layers

        self.embedding = layers.Embedding(vocab_size, d_model)
        self.pos_encoding = PositionalEncoding()
        self.dropout = layers.Dropout(rate=dropout_rate)

        self.dec_layers = [DecoderLayer(FFN_units,
                                        nb_proj,
                                        dropout_rate)
                           for i in range(nb_layers)]

    def call(self, inputs, enc_outputs, mask_1, mask_2, training):
        outputs = self.embedding(inputs)
        outputs *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        outputs = self.pos_encoding(outputs)
        outputs = self.dropout(outputs, training)

        for i in range(self.nb_layers):
            outputs = self.dec_layers[i](outputs,
                                         enc_outputs,
                                         mask_1,
                                         mask_2,
                                         training)

        return outputs
```

Transformer



For coding up the transformer, we inherit from the TensorFlow Model class, and we connect the encoder and decoder and all associated layers. We also define the masking functions that we mentioned previously (see create_padding_mask and create_look_ahead_mask):

```
class Transformer(tf.keras.Model):

    def __init__(self,
                 vocab_size_enc,
                 vocab_size_dec,
                 d_model,
                 nb_layers,
                 FFN_units,
                 nb_proj,
                 dropout_rate,
                 name="transformer"):
        super(Transformer, self).__init__(name=name)

        self.encoder = Encoder(nb_layers,
                               FFN_units,
                               nb_proj,
                               dropout_rate,
                               vocab_size_enc,
                               d_model)
```

```
        self.decoder = Decoder(nb_layers,
                               FFN_units,
                               nb_proj,
                               dropout_rate,
                               vocab_size_dec,
                               d_model)
        self.last_linear = layers.Dense(units=vocab_size_dec, name="lin_ouput")

    def create_padding_mask(self, seq):
        mask = tf.cast(tf.math.equal(seq, 0), tf.float32)
        return mask[:, tf.newaxis, tf.newaxis, :]

    def create_look_ahead_mask(self, seq):
        seq_len = tf.shape(seq)[1]
        look_ahead_mask = 1 - tf.linalg.band_part(tf.ones((seq_len, seq_len)),
-1, 0)
        return look_ahead_mask

    def call(self, enc_inputs, dec_inputs, training):
        enc_mask = self.create_padding_mask(enc_inputs)
        dec_mask_1 = tf.maximum(
            self.create_padding_mask(dec_inputs),
            self.create_look_ahead_mask(dec_inputs)
        )
        dec_mask_2 = self.create_padding_mask(enc_inputs)

        enc_outputs = self.encoder(enc_inputs, enc_mask, training)
        dec_outputs = self.decoder(dec_inputs,
                                   enc_outputs,
                                   dec_mask_1,
                                   dec_mask_2,
                                   training)

        outputs = self.last_linear(dec_outputs)

        return outputs
```

Loss

Since for model inputs we ended up producing sequences of integers, the best loss function to use is sparse categorical cross entropy. We also want to set “from logits” to true, since we will be passing the output through the SoftMax function so that we can get probabilities and we set reduction to none, this way we can use a mask here for the same reasons that we did for modeling (to mask the padding tokens), and then we can sum over all dimensions:

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
                                                             reduction="none")

def loss_function(target, pred):
```

```
    mask = tf.math.logical_not(tf.math.equal(target, 0))
    loss_ = loss_object(target, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)

train_loss = tf.keras.metrics.Mean(name="train_loss")
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name="train_accurac
y")
```

Optimizer settings and learning rate schedule

The research paper is followed for the optimizer settings and learning rate schedule:

> We used the Adam optimizer [17] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:
>
> $$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \quad (3)$$
>
> This corresponds to increasing the learning rate linearly for the first $warmup\_steps$ training steps, and decreasing it thereafter proportionally to the inverse square root of the step number. We used $warmup\_steps = 4000$.

```
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):

    def __init__(self, d_model, warmup_steps=4000):
        super(CustomSchedule, self).__init__()

        self.d_model = tf.cast(d_model, tf.float32)
        self.warmup_steps = warmup_steps

    def __call__(self, step):
        arg1 = tf.math.rsqrt(step)
        arg2 = step * (self.warmup_steps**-1.5)

        return tf.math.rsqrt(self.d_model) * tf.math.minimum(arg1, arg2)

learning_rate = CustomSchedule(D_MODEL)

optimizer = tf.keras.optimizers.Adam(learning_rate,
                                     beta_1=0.9,
                                     beta_2=0.98,
                                     epsilon=1e-9)
```

Training

The classes and functions that make up the model were all collected into a utils.py file for the transformer architecture. In the Jupyter notebook, in the first part of the modeling section, the transformer was called with the same parameters as the base model in the research paper except for an increase to 1024-dimension embeddings and a reduction of encoding layers from 6 to 4:

```
tf.keras.backend.clear_session()

# Hyper-parameters
D_MODEL = 1024
NB_LAYERS = 4
FFN_UNITS = 2048
NB_PROJ = 8
DROPOUT_RATE = 0.1

transformer = utils.Transformer(vocab_size_enc=VOCAB_SIZE_QUESTIONS,
                                vocab_size_dec=VOCAB_SIZE_ANSWERS,
                                d_model=D_MODEL,
                                nb_layers=NB_LAYERS,
                                FFN_units=FFN_UNITS,
                                nb_proj=NB_PROJ,
                                dropout_rate=DROPOUT_RATE)
```

```
EPOCHS = 315
for epoch in range(EPOCHS):
    print("Start of epoch {}".format(epoch+1))


    train_loss.reset_states()
    train_accuracy.reset_states()

    for (batch, (enc_inputs, targets)) in enumerate(dataset):
        dec_inputs = targets[:, :-1]
        dec_outputs_real = targets[:, 1:]
        with tf.GradientTape() as tape:
            predictions = transformer(enc_inputs, dec_inputs, True)
            loss = loss_function(dec_outputs_real, predictions)

        gradients = tape.gradient(loss, transformer.trainable_variables)
        optimizer.apply_gradients(zip(gradients, transformer.trainable_variables))

        train_loss(loss)
        train_accuracy(dec_outputs_real, predictions)

        if batch % 50 == 0:
            print("Epoch {} Batch {} Loss {:.4f} Accuracy {:.4f}".format(
                epoch+1, batch, train_loss.result(), train_accuracy.result()))
```

```
Loss 0.1208 Accuracy 0.9263
```

Testing Metrics

It was challenging to come up with a good metric to test this model. The research paper uses Bleu scores but, in this case, I repurposed the model for a question-and-answer task, so this is not a translation, and it is not really the right metric. I originally used Bleu score because using cumulative scores I could iterate for the highest possible 4-gram scores, and this was helpful in making sure I was going in the right direction correcting how intelligible the sentences were, but the scores are too high to

present them and a problem with transformers is that the answer sentence might be perfectly comprehensible but might not be answering the question at all. The F1 score proved to be an excellent metric to improve the model. F1 scores were used to conduct a rigorous review of predictions sentence by sentence, iterating to increase this metric did dramatically improve the quality of results from a human perspective:

F1= 2precisionrecall/(precision+recall) precision = tp/(tp+fp) recall=tp/(tp+fn)

precision = 1.0 * num_same / len(pred_toks)=tp/(tp+fp)

recall = 1.0 * num_same / len(gold_toks)=tp/(tp+fn)

tp=number of tokens that are shared between the correct answer and the prediction

fp=number of tokens that are in the prediction but not in the correct answer

fn=number of tokens that are in the correct answer but not in the prediction

https://kierszbaumsamuel.medium.com

```
def compute_f1(gold_toks, pred_toks):
    common = collections.Counter(gold_toks) & collections.Counter(pred_toks)
    num_same = sum(common.values())
    if len(gold_toks) == 0 or len(pred_toks) == 0:
        return 1 # If either is no-answer, then F1 is 1 if they agree, 0 otherwise return int(gold_toks == pred_toks)
        if num_same == 0:
            return 0
    precision = 1.0 * num_same / len (pred_toks)
    recall = 1.0 * num_same / len (gold_toks)
    f1 = (2 * precision * recall) / ((precision + recall) + 0.0001)
    return f1
```

Unseen Data Test Results

A couple of questions were changed so that the model would not have seen the re-phrasing but the questions retained the same meaning as the questions they were derived from, f1 scores were calculated against the gold standard answers from the original questions:

```
# For unseen samples select a few examples with high f1 scores, we can rephrase the questions to test
# the model

# Unseen example - a question used in training was "what is mental illness?"
# we will use the same reference answer as the gold standard but will phrase the question differently

new = reply(tokenizer_answers, 'define mental illness?')
```

```
Input: define mental illness?
Predicted Response: mental illness are health conditions that disrupt a person's thoughts , emotions , relationships , and daily functioning .
```

```
reference = preprocess_reference(df.iloc[0]['Answers'])
```

```
score = compute_f1(reference[:len(new)].split(), new.split())
score
```

```
0.918868958158396
```

```
# Unseen example - a question used in training was "are there local resources?"
# we can use the same reference answer as the gold standard but will phrase
# the question differently

new = reply(tokenizer_answers, 'where are there local resources?')
```

```
Input: where are there local resources?
Predicted Response: you can learn more about resources in your community by searching online .
```

```
reference = preprocess_reference(df.iloc[34]['Answers'])
reference
```

```
'yes , you can learn more about resources in your community by searching online . '
```

```
# This reference answer is the closest we have in the dataset
# calculating the f1 score doesn't really give the predicted response justice
# since as humans we can understand it was a very strong
# reply to that unseen question but we are limited by our methods

score = compute_f1(reference[:len(new)].split(), new.split())
score
```

```
0.814764886461503
```

Conclusion

I think that the issues that I ran into like the dataset producing unintelligible sentences in the beginning and having to customize it, running out of memory when trying to do check points filling up my storage to well over 2TB, having to slightly change the original base model parameters to fit my customized dataset, dealing with limited data without any immediate resources to expand it short of creating my own data set and having to research metrics in a situation where there isn't too much data available online on how to systematically rate a question and answer model ended up creating a very realistic situation in the world of trying to solve machine learning problems. I can emphatically say that I learned a lot about transformers, and the testing results here, considering the situation, really impressed me. I have worked with traditional sequence models before and if you change the structure of a question you trained the model with without changing the meaning and put it as an unseen sentence, even just a one-word change, and you might get a vastly different answer that does not necessarily make sense unless you trained on a very large amount of data. The quality of the context and ordering of the prediction sequences that had F1 scores of 90% or above is very impressive considering how small the final data set was. It can be appreciated that the scoring is not perfect. Even though we only tested a couple of questions, the context representation is strong enough to question the score, particularly on the second question, where the model perhaps gave a more accurate answer than the gold standard (due to the gold standard grammatically fitting the original question more and the prediction fitting the out of test question more). I would love to learn more in the future about how this type of project can be taken further maybe using AWS or another cloud platform and making a larger and cleaner dataset while finally saving those checkpoints on something like S3 to finalize the model and test deployment.

References

https://papers.nips.cc/paper/2017/file

https://jalammar.github.io/illustrated-transformer

https://kierszbaumsamuel.medium.com