Attention For Mental Health

Springboard Data Science Track
Final Capstone Project

Fellow: George Pinto
Mentor: Srdjan Santic

Problem Identification

- During the Covid-19 Pandemic many people may experience mental health issues
- When looking for help online, people will be most interested in asking simple questions and getting quick answers to those questions
- People will want to know the knowledge base comes from reliable sources
- If an issue is serious enough proper recommendations should follow

Problem Identification : Dataset

- Prepared by https://www.kaggle.com/narendrageek

- Consists of 98 mental health question-and-answer pairs gathered from 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

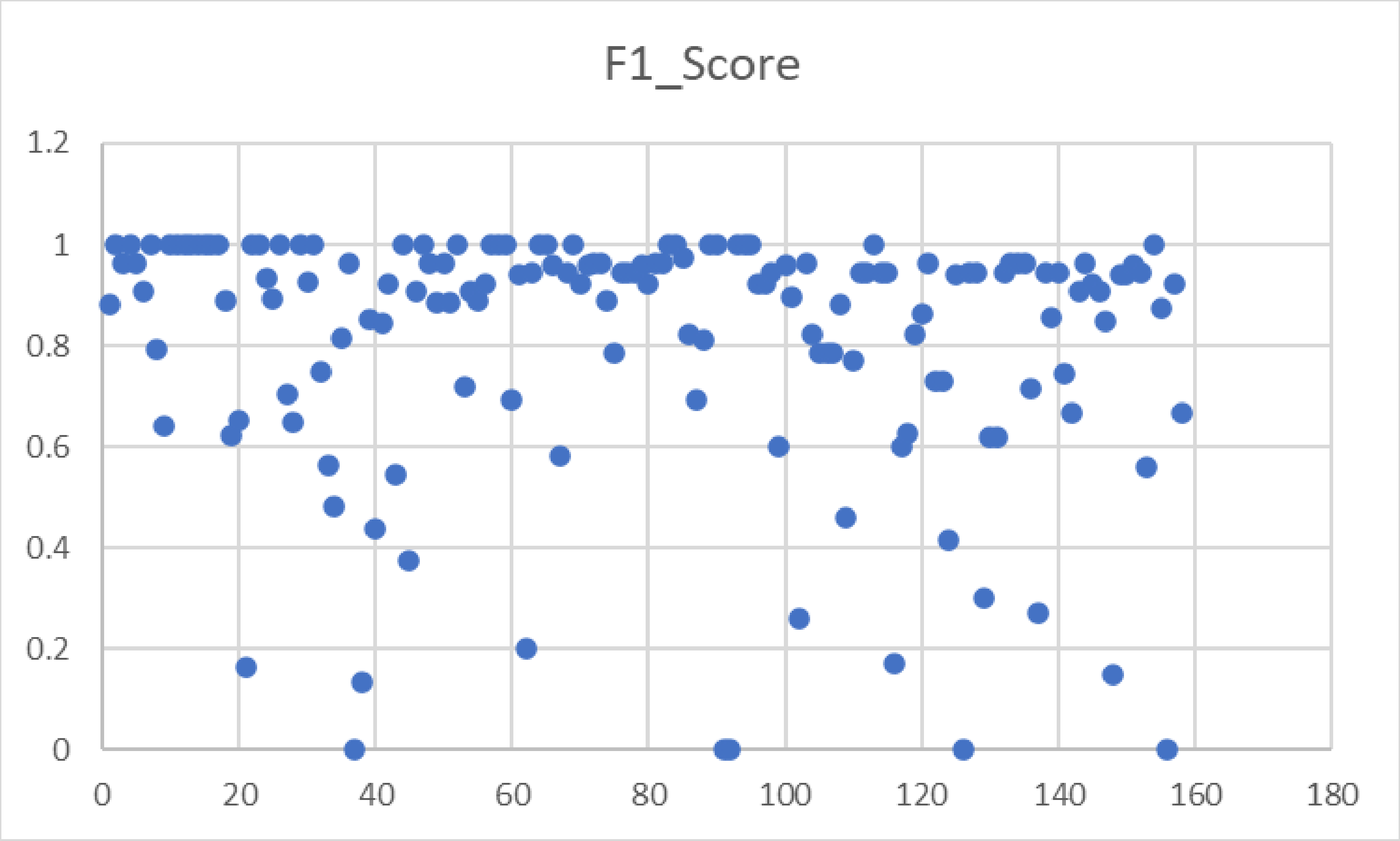
Recommendations and Key Findings

- Adapt a research paper, in this case "Attention is all you need" to help create a question-and-answer mental health solution
- A transformer model made from scratch trained on mental health data can achieve sentence by sentence out of sample F1 scores above 80%
- Responds with clear intelligible sentences
- The user only needs to input their question
- The information will be as reliable as the carefully selected sources

Recommendations and Key Findings

- The challenge is to increase the sentence-by-sentence F1 score of the answers against the gold standard in training
- Once most responses have an F1 score of 90 or above, sentences that scored in that range will be considered the scope of the question answer pairs that the model can answer.

Recommendations and Key Findings – Training Results



Sentence by sentence overall average F1 training score reaches 0.81

Recommendation and Key Findings - Testing

- Iterate, make changes and score training model sentence by sentence until overall average F1 score reaches 0.81
- Questions are chosen from the scope generated by the question-and-answer pairs that scored an F1 score 90% or above but the questions are rephrased to make them out of sample
- Questions are scored with the same gold standard as the original question
- The transformer model understands context, and high scoring question and answer pairs should be robust to changes in the basic question structure if the context is the same (the question is asking the same thing in a different way)

Recommendations and Key Findings - Testing

```
# 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
```

Recommendations and Key Findings - Testing

```
# 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
```

Modeling Results and Analysis – Text Processing

- The first issues encountered in the dataset were encoding issues and were resolved using Pandas

Modeling Results and Analysis – Text Processing

```
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
```

Modeling Results and Analysis – Text Processing

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

```python
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...'
```

Modeling Results and Analysis – Custom Dataset

- On first iterations of predictions many sentences where unintelligible
- Sentences needed to be limited to 26 words for best results
- For the 26-word predictions to make sense, the original answer format had to be changed
- Consideration had to be given to the fact that both questions and answers would be truncated
- Limitations of available resources and memory had to be considered
- This was done without altering source content just the format of the questions

Modeling Results and Analysis – Custom Dataset

- Example of original dataset question and answer pairs (98):

<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>

Modeling Results and Analysis – Custom Dataset

- Example of custom dataset question and answer pairs (158):

```
<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 daily life . <end>

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

<start> mental illness can affect anyone regardless of gender , age , income , social status , ethnicity , religion , 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 damage, 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 importance . <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 specific to the particular type
of mental illness . <end>
```

Modeling Results and Analysis – Tokenization

- For data to be used by the model it needs to be numeric
- Once the dataset formatting and preprocessing had been completed, TensorFlow was used for tokenization of all question-and-answer pairs:

Modeling Results and Analysis – Tokenization

```
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)
```

Modeling Results and Analysis – Padding

- To input sequences shorter than 26 characters, empty spaces needed to be replaced with 0 padding
- This maximum sequence length was set for both question-and-answer sequences
- TensorFlow was used for this purpose as well

Modeling Results and Analysis – Padding

```python
# 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)
```

```python
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)
```

Modeling Results and Analysis – TensorFlow Dataset

- it was found (probably due to vectorization) that large batch sizes were ideal, ending with a batch size of 512

```
# 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)
```

Modeling Results and Analysis – GPU

- Running GPUs is an absolute necessity
- Multiple iterations must be used to try things out
- 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

Modeling – Positional Encodings (Inputs)

- This is not a sequential model
- The entire sequence gets fed into the model at once
- We use “attention”, which measures contextual relationships between words
- For this reason, input embeddings will need some way of representing the distance between words so the model can figure out word order.
- Positional encodings can be added to the original embeddings

Modeling – Positional Encodings (Inputs)

- 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:

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

Modeling – Positional Encodings (Inputs)

```
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)
```

Modeling – Attention

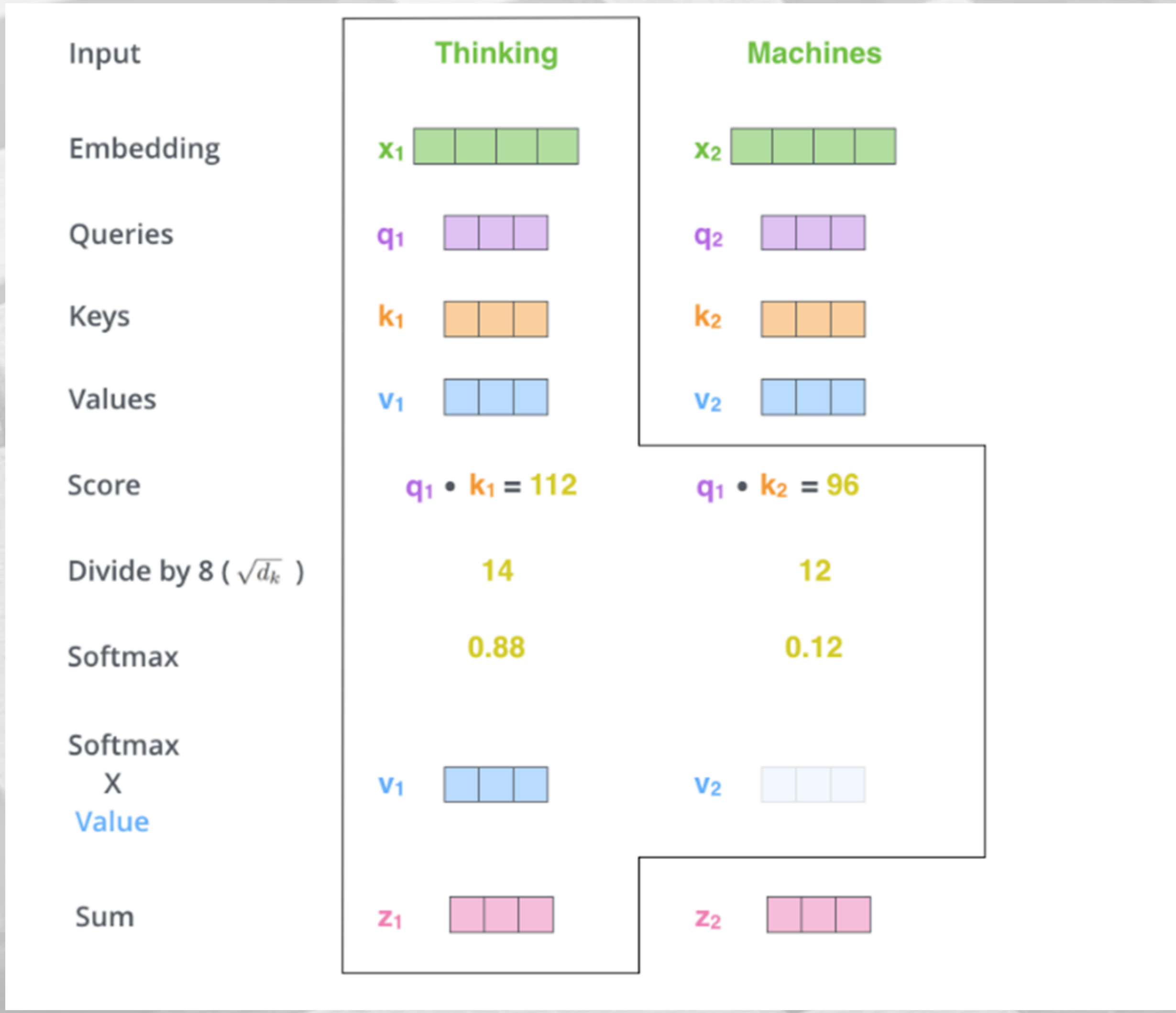
- To code “attention” we follow the paper carefully, using dot product attention with a scaling factor, explained here https://papers.nips.cc/paper/2017/file:

$$\mathrm{Attention}(Q,K,V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Modeling – Attention

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

Modeling – Attention



Modeling – Attention

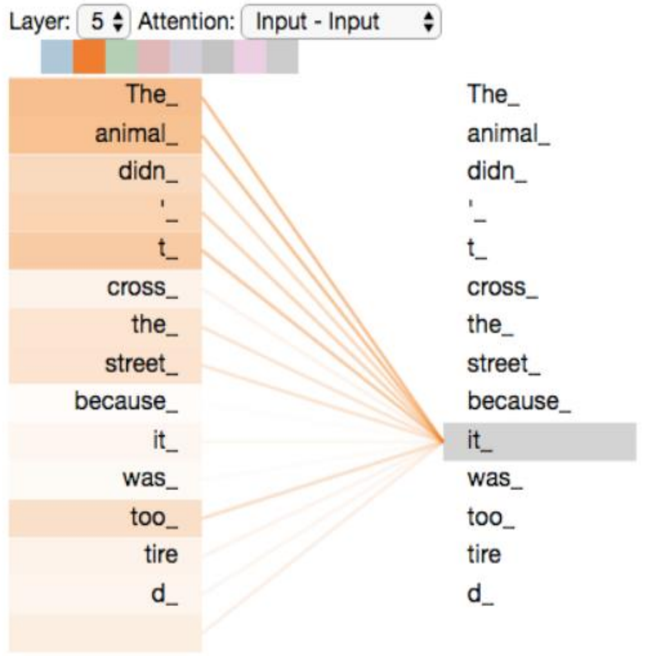
- We can write an attention function using python and TensorFlow
- We inherit from the Layer class (TensorFlow)
- keep in mind that inputs queries, keys and values are dot products of our embedding vectors (plus the positional encodings) and the weights plus the biases (the weights are learned through 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

Modeling – Attention

```
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
```

Modeling – Attention

- the thicker lines with stronger color are stronger relationships between words https://jalammar.github.io/illustrated-transformer:



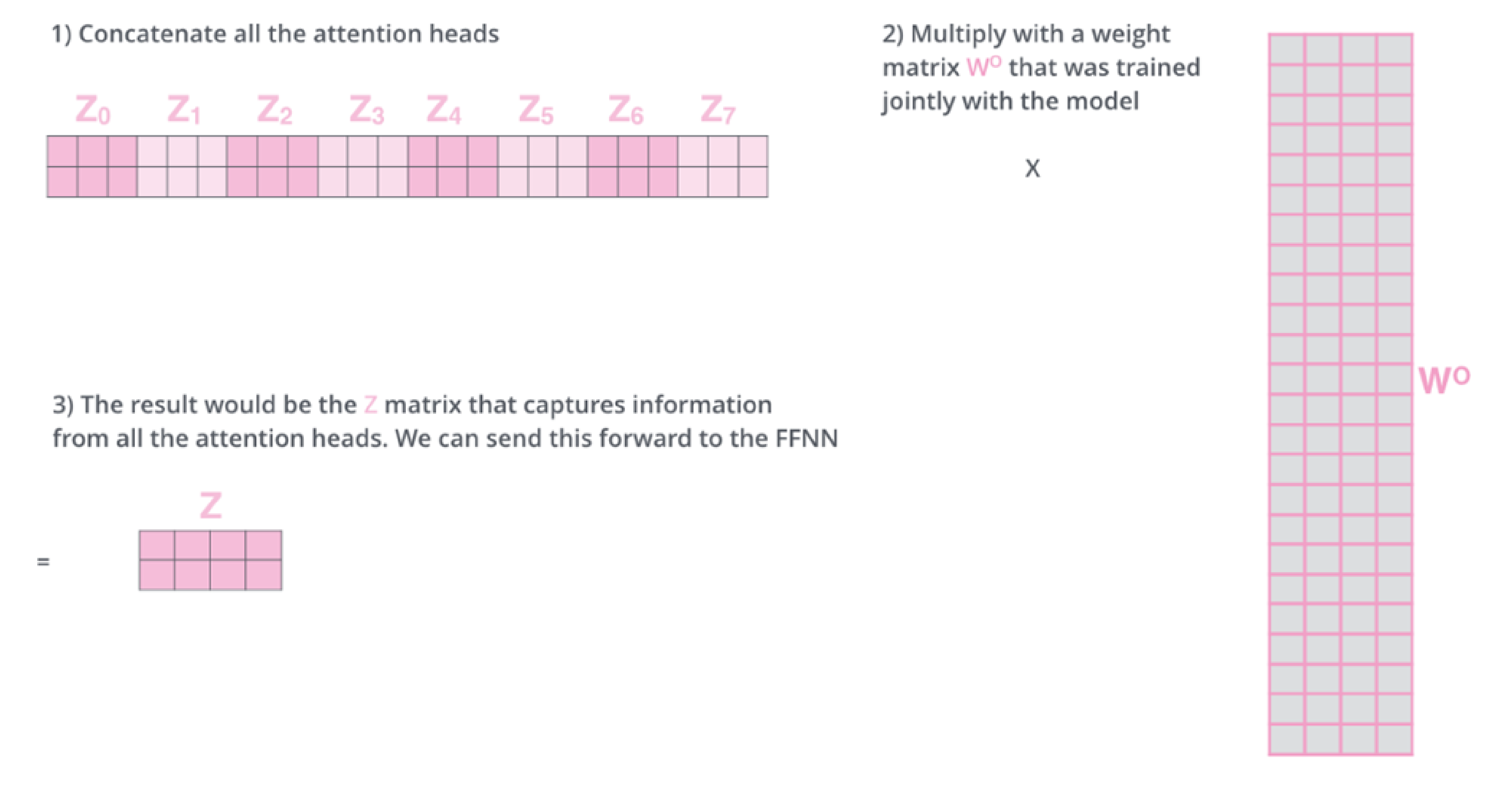
As we are encoding the word "it" in encoder #5 (the top encoder in the stack), part of the attention mechanism was focusing on "The Animal", and baked a part of its representation into the encoding of "it".

Modeling – Multi-head Attention

- In the previous diagram we see a strong relationship between “The animal” and “it” for example
- But the word “tired” is very weakly associated to “it”
- Certain word relationships might be stronger in this representation, and we might overlook other important relationships.
- To get a more robust context for each word we perform multi-head attention
- Attention is performed independently (we know the Q, K, V weights are initialized randomly) 8 times for each word

Modeling – Multi-head Attention

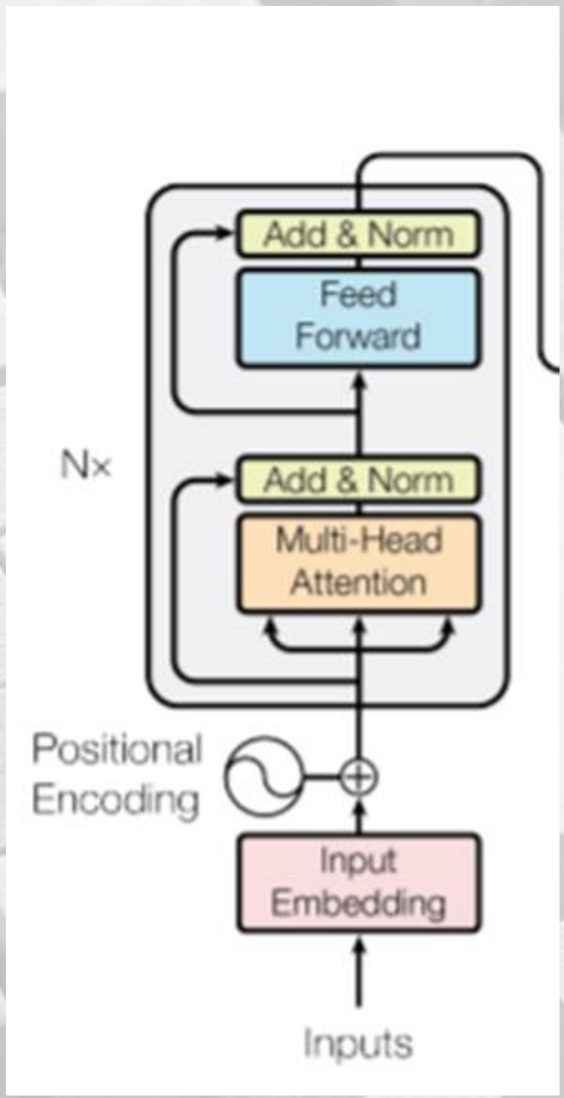
- Best summarized here https://jalammar.github.io/illustrated-transformer:



Modeling – Multi-head Attention

```
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
```

Modeling – Encoder



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

Modeling – Encoder

- Once again, we inherit from the layer class
- The base model in the paper uses 6 encoding layers, this model worked best with 4
- Note the residual layers with batch normalization which are conducted after applying dropout (dropout is applied to prevent overfitting during training only using recommended settings from the research paper).
- 2048 feed forward units were used as in the base model

Modeling – Encoder

- A single encoder layer is created first

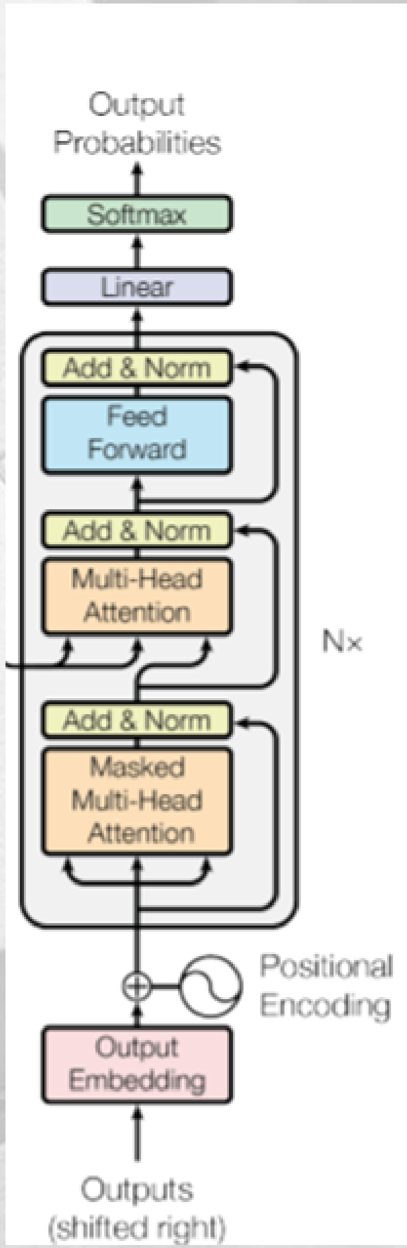
```
class EncoderLayer(layers.Layer):

    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
```

Modeling – Encoder

- The full encoder uses multiple encoder layers

```
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
```

Modeling – Decoder



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

Modeling – Decoder

- The architecture is similar to the encoder architecture also applying dropout and residuals with normalization
- On this architecture attention is used twice
- We first apply attention using the inputs of the decoder (self-attention)
- The second time we use 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
- The masks are used so that we can ignore padding values or since the decoder predicts word by word, to avoid looking ahead of the current prediction (only past values are considered)

Modeling – Decoder

- A single decoder layer is created first

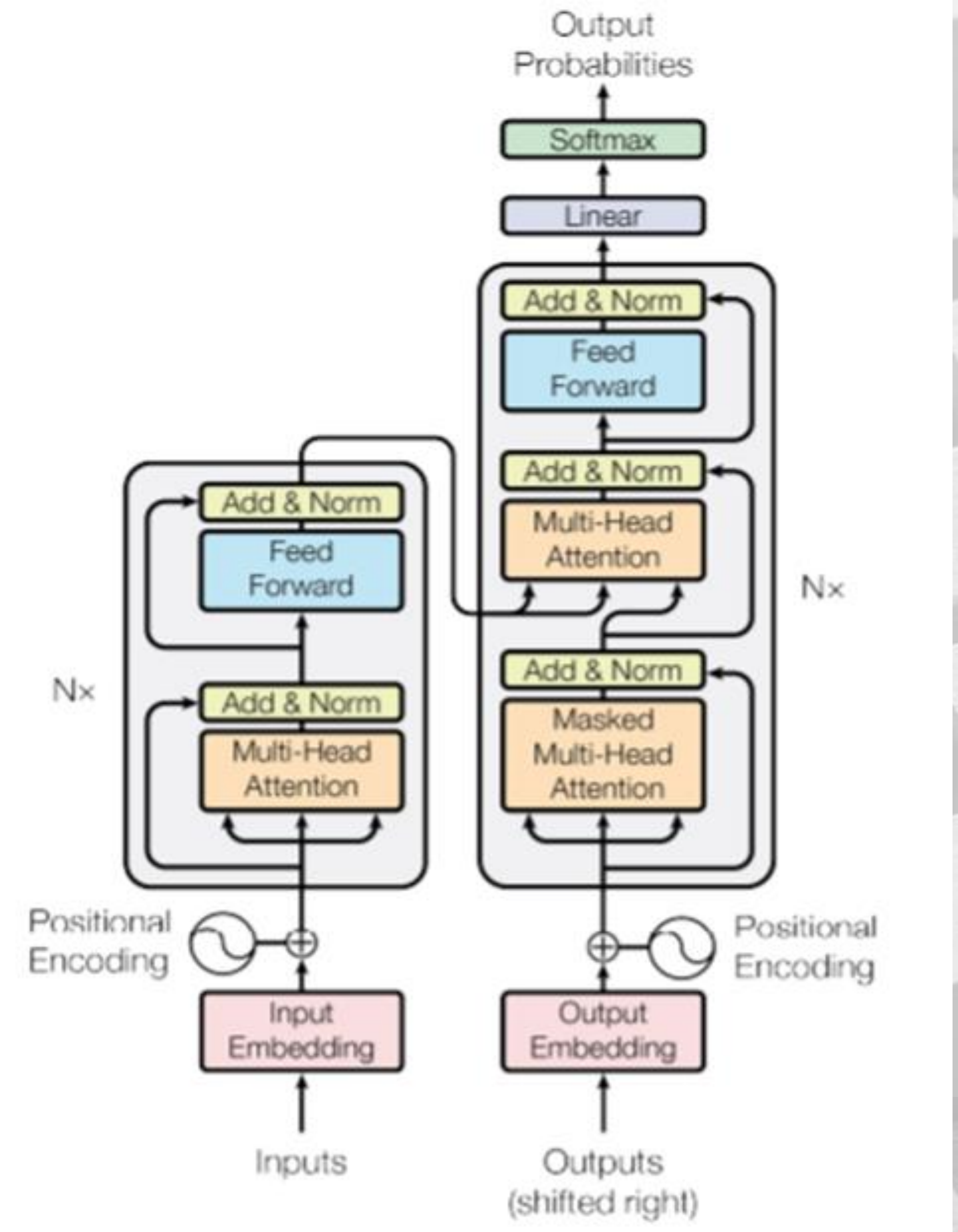
```
class DecoderLayer(layers.Layer):

    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
```

Modeling – Decoder

- The full decoder uses multiple decoder layers that match the number of encoder layers

```
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
```

Modeling – Transformer



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

Modeling – Transformer

- For coding up the transformer, we inherit from the TensorFlow Model class
- We connect the encoder and decoder and all associated layers
- We also define the masking functions that we use in the classes

Modeling – Transformer

```
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
```

Modeling – 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
- We 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

Modeling – Loss

```
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")
```

Modeling – Optimizer Settings and Learning Rate Schedule

- We follow optimizer and learning rate schedule settings as explained here https://papers.nips.cc/paper/2017/file:

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:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}) \tag{3}$$

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 $warmup\_steps = 4000$.

Modeling – Optimizer Settings and Learning Rate Schedule

```
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)
```

Modeling – 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 from 512 and a reduction of encoding layers from 6 to 4 for the best results

Modeling – Training

```
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)
```

Modeling – Training

```
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
```

Modeling – Testing Metrics and Results

- https://kierszbaumsamuel.medium.com

```
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
```

Sentence by sentence overall average F1 training score reaches 0.81

Modeling – Testing Metrics and Results

```
# 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
```

Modeling – Testing Metrics and Results

```
# 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

- The quality of the context and word ordering of the prediction sequences using question and answer pairs that had F1 scores of 90% or above in training 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 whether the F1 metric covers everything
- For example, on the second test question 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)

Conclusion

- I would love to learn more in the future about how this type of project can be taken further:
  - using AWS or another cloud platform
  - making a larger and cleaner dataset
  - saving checkpoints on something like S3 to finalize the model and test deployment
  - Using distributed training

References

- https://papers.nips.cc/paper/2017/file
- https://jalammar.github.io/illustrated-transformer
- https://kierszbaumsamuel.medium.com

THANK YOU

QUESTIONS?