18 - Transformers

Ludwig Krippahl

Transformers

Transformers

Summary

- Attention
- Transformers
- Implementing a tranformer model

- Translate Portuguese to English
  - (Complex details outside the scope of the course)

Attention

Attention

- Inspired on biological systems

- We do not pay equal attention to everything but constantly focus on more relevant details

- Motivated by difficulties with encoder-decoder recurrent networks

- Hard to learn long-term dependencies

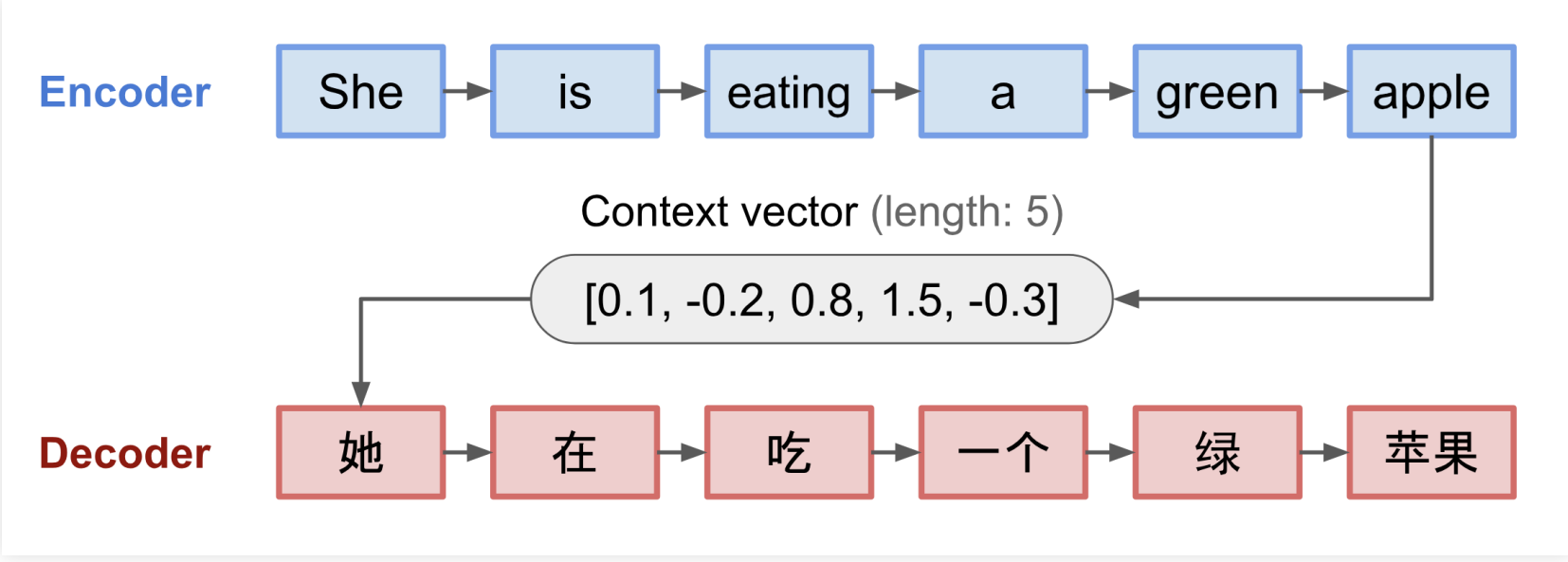


Image by Lilian Weng

Attention

- Attention mechanism proposed to create dynamic context from weighted sum of encoder states

Image by Lilian Weng, adapted from Bahdanau et al., 2014.

Attention

- It worked so well that we can do away with recurrent networks altogether
- Vaswani et al., 2017, proposed the Transformer architecture

- Feed-forward, so faster to train
- Generates sequence iteratively from the input sequence and the previously generated sequence
- Uses scaled dot product attention

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

- Distinction between keys, query and values is merely conceptual

- E.g. it can be used for self-attention

- (There are other attention mechanisms)

Attention

- The authors split attention into multiple "heads"
  - Facilitates learning relations at different scales

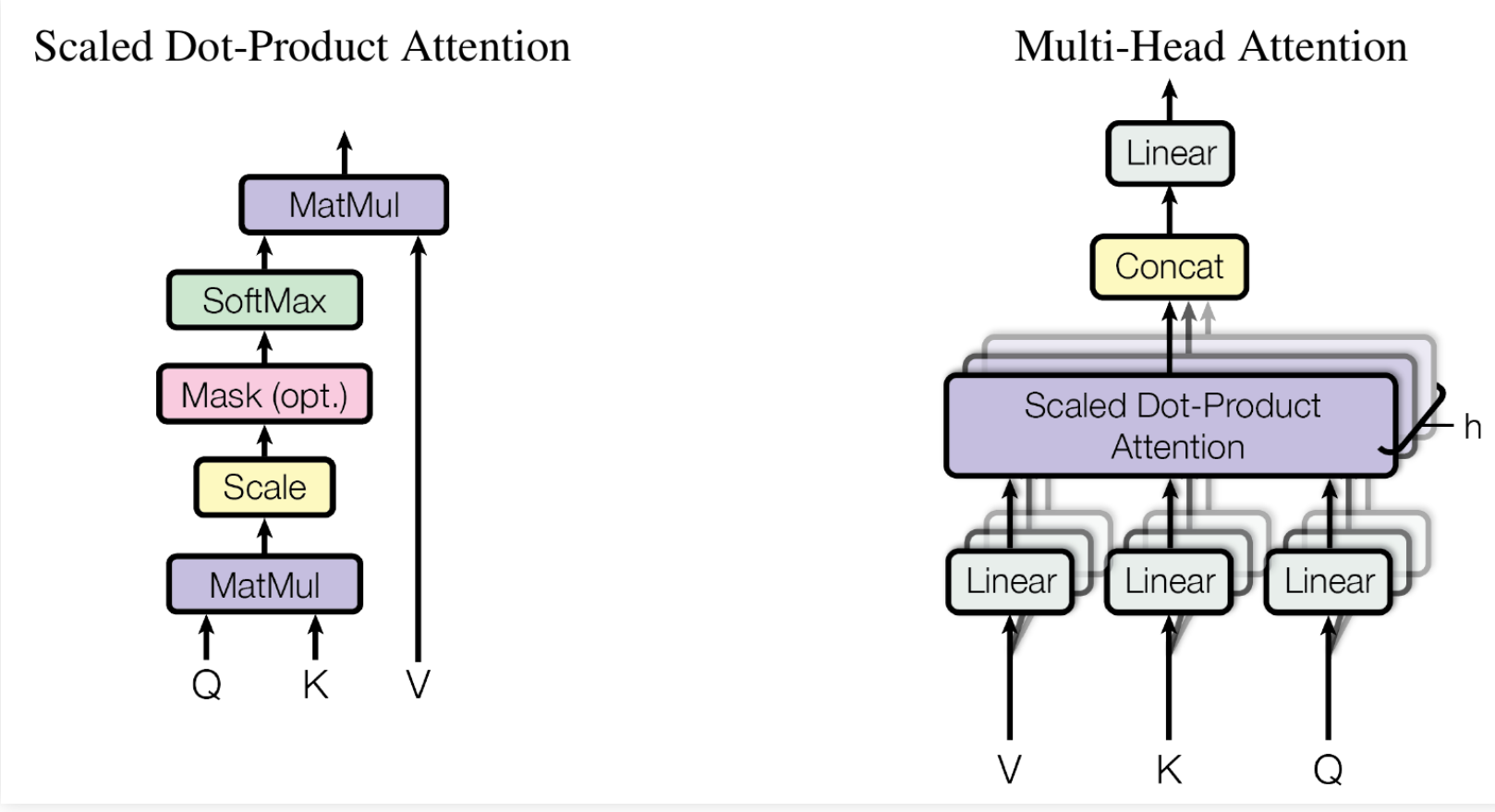
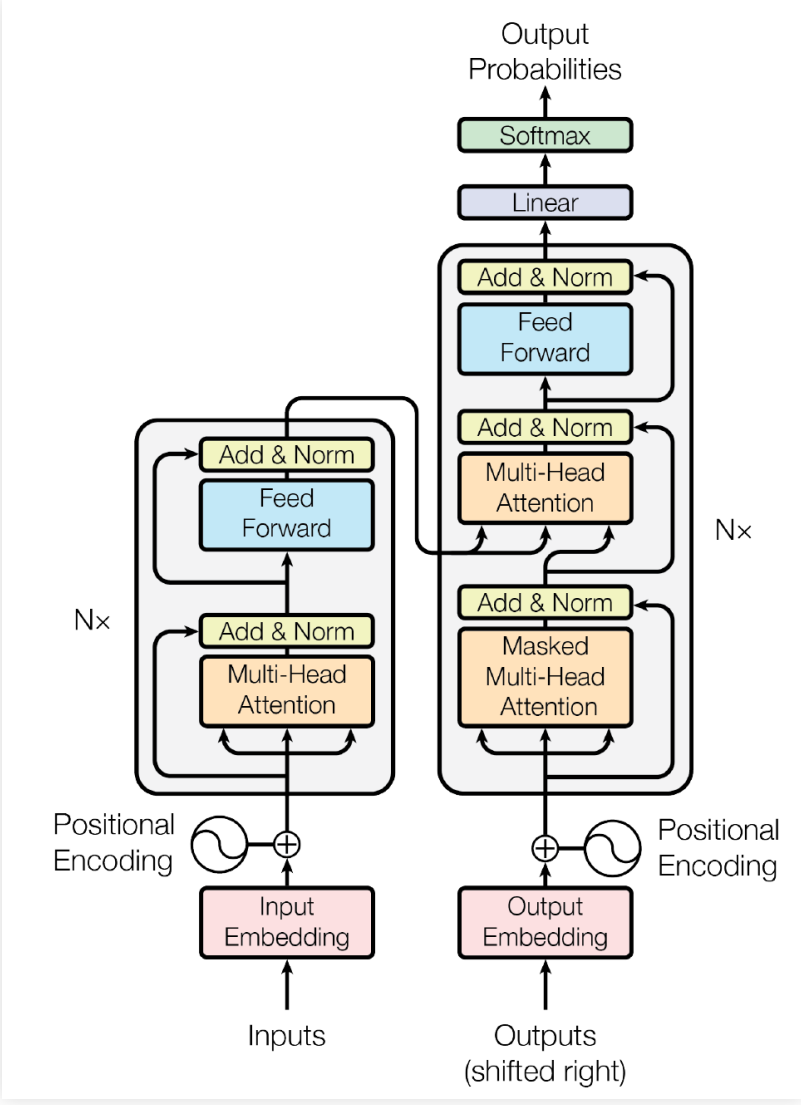


Image from Vaswani et al., 2017

Transformer



Transformer model

- Feed-forward (faster to train)
- Encoder receives input sequence
- Decoder receives sequence generated so far
- Encoder output feeds into attention layer
- Self attention layers
- Several blocks repeated

Generating sequences

- Iteratively feeding input and previously generated output
- Predicts next token

Implementing a Transformer

Example

Transformer to translate

- From Portuguese to English
- We will need:

- Dataset
- Tokenizer, to convert words to vector

- Istal necessary libraries into your Tensorflow environment

```
pip install tensorflow_datasets
pip install tensorflow_text
```

Example

```
import time
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow as tf
examples, metadata = tfds.load('ted_hrlr_translate/pt_to_en', with_info=True,
                               as_supervised=True)
train_examples, val_examples = examples['train'], examples['validation']

model_name = "ted_hrlr_translate_pt_en_converter"
tf.keras.utils.get_file(
    f"{model_name}.zip",
    f"https://storage.googleapis.com/download.tensorflow.org/models/{model_name
    cache_dir='.', cache_subdir='', extract=True)

tokenizers = tf.saved_model.load(model_name)
```

- Download data and tokenizers for PT and EN

Example

- Data pipeline using `tf.data.Dataset`

```
def tokenize_pairs(pt, en):
    pt = tokenizers.pt.tokenize(pt)
    pt = pt.to_tensor()
    en = tokenizers.en.tokenize(en)
    en = en.to_tensor()
    return pt, en

BUFFER_SIZE = 20000
BATCH_SIZE = 64

def make_batches(ds):
  return (
      ds
      .cache()
      .shuffle(BUFFER_SIZE)
      .batch(BATCH_SIZE)
      .map(tokenize_pairs, num_parallel_calls=tf.data.AUTOTUNE)
      .prefetch(tf.data.AUTOTUNE))

train_batches = make_batches(train_examples)
val_batches = make_batches(val_examples)
```

Example

Positional encoding

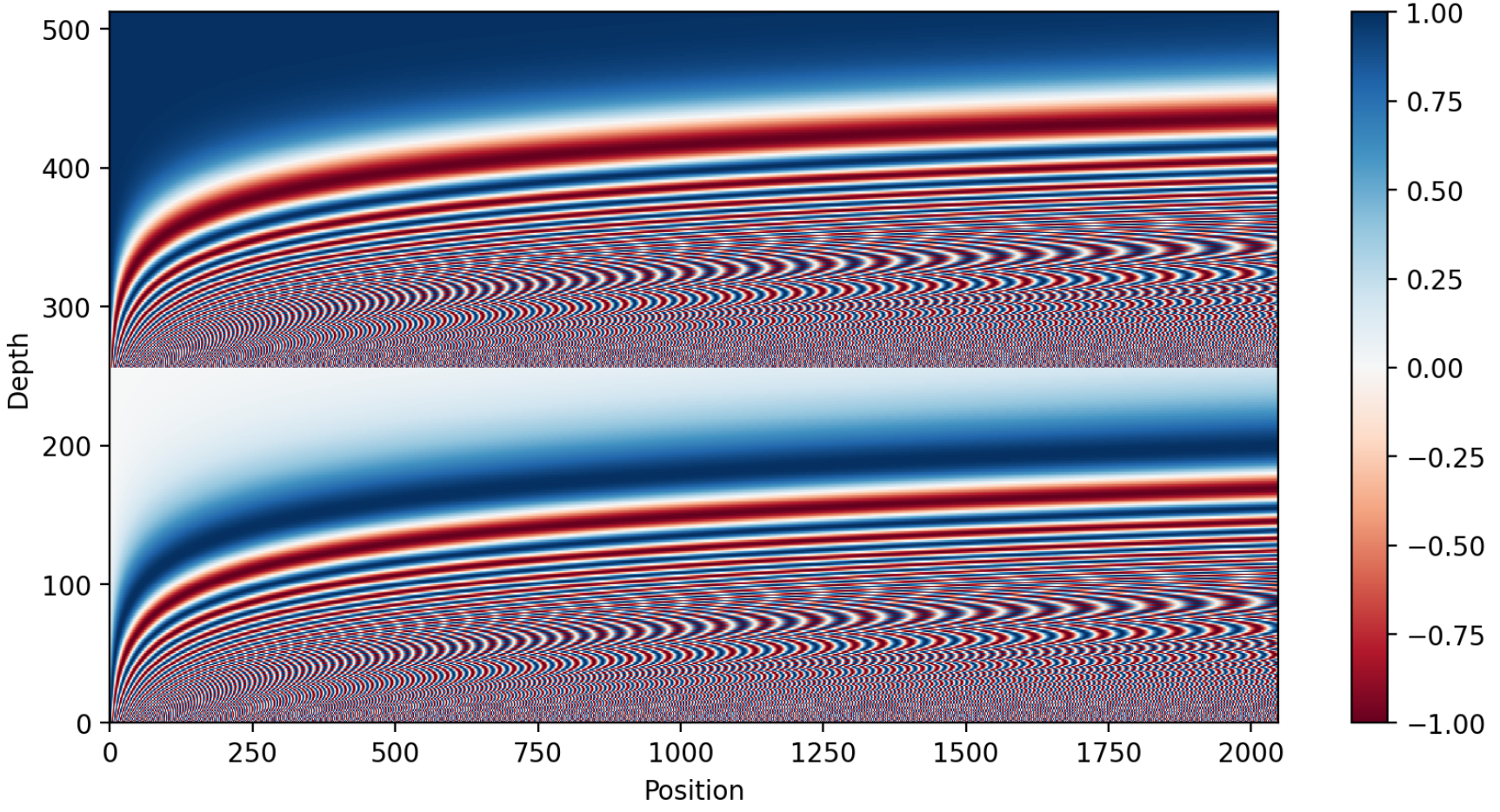
- We are always looking at the complete sentences, so we need positional information
- Add to word embedding vectors that are similar in similar positions but different when farther apart

$$PE_{(pos,2i)} = \sin(pos/100000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/100000^{2i/d_{model}})$$

Example

Positional encoding



Example

Positional encoding

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

def positional_encoding(position, d_model):
  angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                          np.arange(d_model)[np.newaxis, :],
                          d_model)
  angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])
  angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])
  pos_encoding = angle_rads[np.newaxis, ...]
  return tf.cast(pos_encoding, dtype=tf.float32)
```

Example

- We also need to mask our vectors

- Ignore 0 at the end when sentences smaller than maximum length
- Ignore future tokens when receiving the target sequences

```
def create_padding_mask(seq):
  seq = tf.cast(tf.math.equal(seq, 0), tf.float32)
  return seq[:, tf.newaxis, tf.newaxis, :]  # (batch_size, 1, 1, seq_len)

def create_look_ahead_mask(size):
  mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
  return mask
```

```
In : create_look_ahead_mask(4)
Out:
<tf.Tensor: shape=(4, 4), dtype=float32, numpy=
array([[0., 1., 1., 1.],
       [0., 0., 1., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]], dtype=float32)>       [0., 0., 0.]], dtype=float32)>
```

Example

- Scaled dot-product attention

```
def scaled_dot_product_attention(q, k, v, mask):
  matmul_qk = tf.matmul(q, k, transpose_b=True)
  dk = tf.cast(tf.shape(k)[-1], tf.float32)
  scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)
  if mask is not None:
    scaled_attention_logits += (mask * -1e9)
  attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)
  output = tf.matmul(attention_weights, v)
  return output, attention_weights
```

- Masking sets values to a large negative number

- This outputs zeros after softmax

Example

- Create a Keras layer for multi-head attention

- This is optional; you will not be graded on this

```
class MultiHeadAttention(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads):
    super(MultiHeadAttention, self).__init__()
    self.num_heads = num_heads
    self.d_model = d_model
    assert d_model % self.num_heads == 0
    self.depth = d_model // self.num_heads
    self.wq = tf.keras.layers.Dense(d_model)
    self.wk = tf.keras.layers.Dense(d_model)
    self.wv = tf.keras.layers.Dense(d_model)
    self.dense = tf.keras.layers.Dense(d_model)

  def split_heads(self, x, batch_size):
    x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
    return tf.transpose(x, perm=[0, 2, 1, 3])
```

Example

- The `call` method is executed when using the layer

```
class MultiHeadAttention(tf.keras.layers.Layer):
 ...
  def call(self, v, k, q, mask):
    batch_size = tf.shape(q)[0]
    q = self.wq(q)
    k = self.wk(k)
    v = self.wv(v)
    q = self.split_heads(q, batch_size)
    k = self.split_heads(k, batch_size)
    v = self.split_heads(v, batch_size)
    scaled_attention, attention_weights = scaled_dot_product_attention(q,
                                                                       k, v, mask)
    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])
    concat_attention = tf.reshape(scaled_attention,
                                  (batch_size, -1, self.d_model))
    output = self.dense(concat_attention)
    return output, attention_weights
```

Example

- The point-wise network is a stack of two dense layers with a relu between them

```
def point_wise_feed_forward_network(d_model, dff):
  return tf.keras.Sequential([
      tf.keras.layers.Dense(dff, activation='relu'),
      tf.keras.layers.Dense(d_model)
      ])
```

Example

- The encoder block

```
class EncoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(EncoderLayer, self).__init__()
    self.mha = MultiHeadAttention(d_model, num_heads)
    self.ffn = point_wise_feed_forward_network(d_model, dff)
    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask):
    attn_output, _ = self.mha(x, x, x, mask)
    attn_output = self.dropout1(attn_output, training=training)
    out1 = self.layernorm1(x + attn_output)
    ffn_output = self.ffn(out1)
    ffn_output = self.dropout2(ffn_output, training=training)
    out2 = self.layernorm2(out1 + ffn_output)
    return out2
```

Example

```
class DecoderLayer(tf.keras.layers.Layer):
  def __init__(self, d_model, num_heads, dff, rate=0.1):
    super(DecoderLayer, self).__init__()
    self.mha1 = MultiHeadAttention(d_model, num_heads)
    self.mha2 = MultiHeadAttention(d_model, num_heads)
    self.ffn = point_wise_feed_forward_network(d_model, dff)
    self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.layernorm3 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
    self.dropout1 = tf.keras.layers.Dropout(rate)
    self.dropout2 = tf.keras.layers.Dropout(rate)
    self.dropout3 = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,look_ahead_mask, padding_mask):
    attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)
    attn1 = self.dropout1(attn1, training=training)
    out1 = self.layernorm1(attn1 + x)
    attn2, attn_weights_block2 = self.mha2(
        enc_output, enc_output, out1, padding_mask)
    attn2 = self.dropout2(attn2, training=training)
    out2 = self.layernorm2(attn2 + out1)
    ffn_output = self.ffn(out2)
    ffn_output = self.dropout3(ffn_output, training=training)
```

Example

- Encoder: embedding, positional and encoder blocks

```
class Encoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Encoder, self).__init__()
    self.d_model = d_model
    self.num_layers = num_layers
    self.embedding = tf.keras.layers.Embedding(input_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding,
                                            self.d_model)
    self.enc_layers = [EncoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, training, mask):
    seq_len = tf.shape(x)[1]
    x = self.embedding(x)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]
    x = self.dropout(x, training=training)
    for i in range(self.num_layers):
      x = self.enc_layers[i](x, training, mask)
    return x
```

Example

- Decoder, similar

```
class Decoder(tf.keras.layers.Layer):
  def __init__(self, num_layers, d_model, num_heads, dff, target_vocab_size,
               maximum_position_encoding, rate=0.1):
    super(Decoder, self).__init__()
    self.d_model = d_model
    self.num_layers = num_layers
    self.embedding = tf.keras.layers.Embedding(target_vocab_size, d_model)
    self.pos_encoding = positional_encoding(maximum_position_encoding, d_model)
    self.dec_layers = [DecoderLayer(d_model, num_heads, dff, rate)
                       for _ in range(num_layers)]
    self.dropout = tf.keras.layers.Dropout(rate)

  def call(self, x, enc_output, training,look_ahead_mask, padding_mask):
    seq_len = tf.shape(x)[1]
    attention_weights = {}
    x = self.embedding(x)
    x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
    x += self.pos_encoding[:, :seq_len, :]
    x = self.dropout(x, training=training)
    for i in range(self.num_layers):
      x, block1, block2 = self.dec_layers[i](x, enc_output, training,
                                             look_ahead_mask, padding_mask)
      attention_weights[f'decoder_layer{i+1}_block1'] = block1
```

Example

- The transformer model

```
class Transformer(tf.keras.Model):
  def __init__(self, num_layers, d_model, num_heads, dff, input_vocab_size,
               target_vocab_size, pe_input, pe_target, rate=0.1):
    super(Transformer, self).__init__()
    self.tokenizer = Encoder(num_layers, d_model, num_heads, dff,
                             input_vocab_size, pe_input, rate)
    self.decoder = Decoder(num_layers, d_model, num_heads, dff,
                           target_vocab_size, pe_target, rate)
    self.final_layer = tf.keras.layers.Dense(target_vocab_size)

  def call(self, inp, tar, training, enc_padding_mask,
           look_ahead_mask, dec_padding_mask):
    enc_output = self.tokenizer(inp, training, enc_padding_mask)
    dec_output, attention_weights = self.decoder(
        tar, enc_output, training, look_ahead_mask, dec_padding_mask)
    final_output = self.final_layer(dec_output)
    return final_output, attention_weights
```

Example

- Scheduling the learning rate

```
class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):
  def __init__(self, d_model, warmup_steps=4000):
    super(CustomSchedule, self).__init__()
    self.d_model = d_model
    self.d_model = tf.cast(self.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)
```

Example

Masked loss functions

- Loss and running means as global objects for graph compilation

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

def loss_function(real, pred):
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  loss_ = loss_object(real, pred)
  mask = tf.cast(mask, dtype=loss_.dtype)
  loss_ *= mask
  return tf.reduce_sum(loss_)/tf.reduce_sum(mask)

def accuracy_function(real, pred):
  accuracies = tf.equal(real, tf.argmax(pred, axis=2))
  mask = tf.math.logical_not(tf.math.equal(real, 0))
  accuracies = tf.math.logical_and(mask, accuracies)
  accuracies = tf.cast(accuracies, dtype=tf.float32)
  mask = tf.cast(mask, dtype=tf.float32)
  return tf.reduce_sum(accuracies)/tf.reduce_sum(mask)

train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.Mean(name='train_accuracy')
```

Example

- Creating the model

```
num_layers = 4
d_model = 128
dff = 512
num_heads = 8
dropout_rate = 0.1

learning_rate = CustomSchedule(d_model)
optimizer = tf.keras.optimizers.Adam(learning_rate, beta_1=0.9, beta_2=0.98,
                                     epsilon=1e-9)
transformer = Transformer(
    num_layers=num_layers,
    d_model=d_model,
    num_heads=num_heads,
    dff=dff,
    input_vocab_size=tokenizers.pt.get_vocab_size(),
    target_vocab_size=tokenizers.en.get_vocab_size(),
    pe_input=1000,
    pe_target=1000,
    rate=dropout_rate)
```

Example

- Checkpoints, to save regularly and resume

```
checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(transformer=transformer,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path, max_to_keep=5)
if ckpt_manager.latest_checkpoint:
  ckpt.restore(ckpt_manager.latest_checkpoint)
  print('Latest checkpoint restored!!')
```

- And a function to create the masks

```
def create_masks(inp, tar):
  enc_padding_mask = create_padding_mask(inp)
  dec_padding_mask = create_padding_mask(inp)
  look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])
  dec_target_padding_mask = create_padding_mask(tar)
  combined_mask = tf.maximum(dec_target_padding_mask, look_ahead_mask)
  return enc_padding_mask, combined_mask, dec_padding_mask
```

Example

- Compile the graph (more code, but faster to train)

```
train_step_signature = [
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
    tf.TensorSpec(shape=(None, None), dtype=tf.int64),
]

@tf.function(input_signature=train_step_signature)
def train_step(inp, tar):
  tar_inp = tar[:, :-1]
  tar_real = tar[:, 1:]
  enc_padding_mask, combined_mask, dec_padding_mask = create_masks(inp, tar_inp
  with tf.GradientTape() as tape:
    predictions, _ = transformer(inp, tar_inp,
                                 True,
                                 enc_padding_mask,
                                 combined_mask,
                                 dec_padding_mask)
    loss = loss_function(tar_real, predictions)
  gradients = tape.gradient(loss, transformer.trainable_variables)
  optimizer.apply_gradients(zip(gradients, transformer.trainable_variables))
  train_loss(loss)
  train_accuracy(accuracy_function(tar_real, predictions))
```

Example

- The training loop

```
EPOCHS=20
for epoch in range(EPOCHS):
  start = time.time()
  train_loss.reset_states()
  train_accuracy.reset_states()
  for (batch, (inp, tar)) in enumerate(train_batches):
    train_step(inp, tar)
    if batch % 50 == 0:
      print(f'Epoch {epoch + 1} Batch {batch} Loss {train_loss.result():.4f} Ac
  if (epoch + 1) % 5 == 0:
    ckpt_save_path = ckpt_manager.save()
    print(f'Saving checkpoint for epoch {epoch+1} at {ckpt_save_path}')
  print(f'Epoch {epoch + 1} Loss {train_loss.result():.4f} Accuracy {train_accu
  print(f'Time taken for 1 epoch: {time.time() - start:.2f} secs\n')
```

- Note: mean loss and accuracy are running averages and need resetting

Example

- After training (Can take a bit, even with simplified model):

```
def evaluate(sentence, max_length=40):
   sentence = tf.convert_to_tensor([sentence])
   sentence = tokenizers.pt.tokenize(sentence).to_tensor()
   encoder_input = sentence
   start, end = tokenizers.en.tokenize([''])[0]
   output = tf.convert_to_tensor([start])
   output = tf.expand_dims(output, 0)
   for i in range(max_length):
     enc_padding_mask, combined_mask, dec_padding_mask = create_masks(encoder_i
     predictions, attention_weights = transformer(encoder_input,
                                                  output,
                                                  False,
                                                  enc_padding_mask,
                                                  combined_mask,
                                                  dec_padding_mask)
     predictions = predictions[:, -1:, :]  # (batch_size, 1, vocab_size)
     predicted_id = tf.argmax(predictions, axis=-1)
     output = tf.concat([output, predicted_id], axis=-1)
     if predicted_id == end:
       break
   text = tokenizers.en.detokenize(output)[0]
   tokens = tokenizers.en.lookup(output)[0]
   return text, tokens, attention_weights
```

Example

- Example of results

```
def translate(sentence):
    translated_text, _, _ = evaluate(sentence)
    print(translated_text.numpy().decode("utf-8"))

In : translate("este é um problema que temos que resolver")
Out: this is a problem we have to solve .

In : translate("este modelo parece funcionar bem")
Out: this model seems to work well .

In : translate("quem tudo quer tudo perde")
Out: who always wants everything to lose .

In : translate("estás aqui estás a levar")
Out: these are actually taking over here .

In : translate("estou feito ao bife")
Out: i ' m done by the different tree .
```

Summary

Transformers

Summary

- Attention mechanisms
- Transformer networks

- Use attention instead of recurrent layers

- Implementing a tranformer
- More advanced Keras and Tensorflow (optional)

- Datasets
- Schedulers and checkpoints
- Creating layers and models
- `tf.function`