#### 基于Transformer的机器翻译：训练、分析与见解

###### 1. 简介

机器翻译是使用计算机和人工智能技术将一种语言翻译成另一种语言的过程，解决语言障碍，促进全球交流与发展。 Transformer模型，由Vaswani等人于2017年提出，采用自注意力机制，摒弃传统循环结构的顺序依赖性，在并行计算上更高效。

Transformer由编码器和解码器组成，利用多层自注意力机制和全连接前馈神经网络层进行序列处理。在机器翻译任务中，编码器将源语言句子编码为隐藏表示，解码器生成目标语言翻译结果。其自注意力机制使其能够同时关注输入句子所有位置，有效捕捉长距离依赖关系，提高翻译准确性和流畅性。

Transformer模型在机器翻译等任务中取得巨大成功，也广泛用于摘要生成、对话系统等领域。其优势包括强大的并行计算能力、处理长距离依赖关系的能力、易于训练和实现，成为自然语言处理领域的核心技术。

###### 数据集

2.1 数据集的基本信息

·本实验采用的数据集为TED Talks 翻译数据集

·语言对：葡萄牙语（pt）到英语（en）

·数据规模：训练集有51785 个例子，验证集大小有1193 个例子

2.2 数据集预处理

检查了一下加载的数据，查看样本句子都已经干净，没有明显的错误或异常。可以直接用于训练机器翻译模型。所以不用费力地对数据集进行文本清洗。

2.2.1 分词

对英文和西班牙语进行分词

en\_tokenizer = tfds.deprecated.text.SubwordTextEncoder.build\_from\_corpus(

(en.numpy() for pt, en in train\_examples),

target\_vocab\_size = 2 \*\* 13)

pt\_tokenizer = tfds.deprecated.text.SubwordTextEncoder.build\_from\_corpus(

(pt.numpy() for pt, en in train\_examples),

target\_vocab\_size = 2 \*\* 13)

上面的代码构建了葡萄牙语（pt）和英语（en）的子词（Subword）级别的文本编码器（Tokenizer）。将文本分割成子词（subword），可以处理词汇表中不存在的词汇或处理形态变化的词汇。

###### **3. Transformer 模型架构**

3.1 编码器和解码器结构

Transformer模型由编码器和解码器组成。编码器由多个相同结构的层叠层组成，解码器也是如此。每个层都包含自注意力机制和前馈神经网络。在这里就不介绍代码了，详细代码可见附件transformer\_pt2en\_.ipynb。

3.2 自注意力机制

自注意力机制用于捕捉输入序列中词与词之间的依赖关系，使得模型在处理序列时能够同时关注所有位置。

多头注意力允许模型在不同注意力子空间中学习不同的关系。通过将多个注意力头叠加在一起，模型可以更全面地捕捉序列信息。

class MultiHeadAttention(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 self.d\_model % self.num\_heads == 0

self.depth = self.d\_model // self.num\_heads

self.WQ = keras.layers.Dense(self.d\_model)

self.WK = keras.layers.Dense(self.d\_model)

self.WV = keras.layers.Dense(self.d\_model)

self.dense = keras.layers.Dense(self.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])

def call(self, q, k, v, mask):

batch\_size = tf.shape(q)[0]

q = self.WQ(q) # q.shape: (batch\_size, seq\_len\_q, d\_model)

k = self.WK(k) # k.shape: (batch\_size, seq\_len\_k, d\_model)

v = self.WV(v) # v.shape: (batch\_size, seq\_len\_v, d\_model)

q = self.split\_heads(q, batch\_size)

k = self.split\_heads(k, batch\_size)

v = self.split\_heads(v, batch\_size)

scaled\_attention\_outputs, attention\_weights = \

scaled\_dot\_product\_attention(q, k, v, mask)

scaled\_attention\_outputs = tf.transpose(

scaled\_attention\_outputs, perm = [0, 2, 1, 3])

concat\_attention = tf.reshape(scaled\_attention\_outputs,

(batch\_size, -1, self.d\_model))

output = self.dense(concat\_attention)

return output, attention\_weights

3.3 前馈神经网络

前馈神经网络通过全连接层处理每个位置的信息，增强模型对序列的表示能力。下面是Transformer模型中的前馈神经网络的一部分，用于处理每个位置的信息。

def feed\_forward\_network(d\_model, dff):

return keras.Sequential([

keras.layers.Dense(dff, activation='relu'),

keras.layers.Dense(d\_model)

])

3.4 位置编码

由于Transformer模型不具备对输入序列的顺序信息的先天感知能力（不像RNN），因此需要引入位置编码来帮助模型理解输入序列中各个词的相对位置。

在这里采用的是不同频率的正弦和余弦函数来编码位置信息。

定义获取角度的函数

def get\_angles(pos, i, d\_model):

angle\_rates = 1 / np.power(10000, (2 \* (i // 2)) / np.float32(d\_model))

return pos \* angle\_rates

定义获取位置编码的函数

def get\_position\_embedding(sentence\_length, d\_model):

angle\_rads = get\_angles(np.arange(sentence\_length)[:, np.newaxis],

np.arange(d\_model)[np.newaxis, :],

d\_model)

sines = np.sin(angle\_rads[:, 0::2])

cosines = np.cos(angle\_rads[:, 1::2])

position\_embedding = np.concatenate([sines, cosines], axis=-1)

position\_embedding = position\_embedding[np.newaxis, ...]

return tf.cast(position\_embedding, dtype=tf.float32)

通过调用 get\_position\_embedding 函数，生成一个形状为 (1, 30, 300) 的位置编码矩阵，并打印其形状。得到下图的结果。

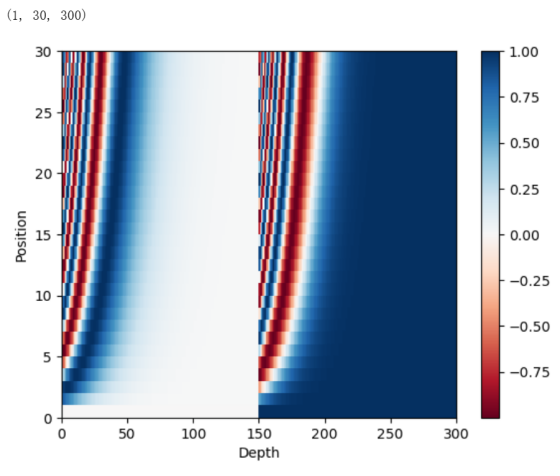


图 1位置编码矩阵

###### **4. 模型训练**

超参数的选择

Transformer 模型的层数 num\_layers = 4

模型的嵌入维度 d\_model = 128

模型中点式前馈网络的隐藏层维度 dff = 512

多头注意力机制中的头数 num\_heads = 8

输入序列的词汇表大小（葡萄牙语） input\_vocab\_size = pt\_tokenizer.vocab\_size + 2

目标序列的词汇表大小（英语） target\_vocab\_size = en\_tokenizer.vocab\_size + 2

模型中的 dropout 概率 dropout\_rate = 0.1

创建transformer模型

transformer = Transformer(num\_layers,

input\_vocab\_size,

target\_vocab\_size,

max\_length,

d\_model, num\_heads, dff, dropout\_rate)

优化器的选择

使用Adam优化器，同时定义了一个学习率调度器，使用创建的学习率调度器 learning\_rate 初始化Adam 优化器 optimizer

optimizer = keras.optimizers.Adam(learning\_rate, beta\_1=0.9, beta\_2=0.98, epsilon=1e-9)

损失函数的定义

loss\_object = 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\_mean(loss\_)

在模型训练过程中仅考虑有效（非填充）位置的损失，而忽略填充位置

训练指标的监控

训练过程中，通过train\_loss和train\_accuracy来监控训练损失和准确度。

训练过程中的这些指标被定期记录，以便监测模型在训练集上的性能。

批处理

训练数据集通过train\_dataset提供小批次的数据给模型。

batch\_size参数定义了每个批次的样本数量，通过小批次随机梯度下降来更新模型参数。

训练周期和 Epochs

在epochs循环中，模型经历多个训练周期。

在每个epoch中，模型通过多个批次的数据进行训练。

学习率调整

class CustomizedSchedule(keras.optimizers.schedules.LearningRateSchedule):

def \_\_init\_\_(self, d\_model, warmup\_steps=4000):

super(CustomizedSchedule, self).\_\_init\_\_()

self.d\_model = tf.cast(d\_model, tf.float32)

self.warmup\_steps = warmup\_steps

def \_\_call\_\_(self, step):

step = tf.cast(step, tf.float32

arg1 = tf.math.rsqrt(step)

arg2 = step \* (self.warmup\_steps \*\* (-1.5))

arg3 = tf.math.rsqrt(self.d\_model)

return arg3 \* tf.math.minimum(arg1, arg2)

learning\_rate = CustomizedSchedule(d\_model)

optimizer = keras.optimizers.Adam(learning\_rate, beta\_1=0.9, beta\_2=0.98, epsilon=1e-9)

模型保存

在训练过程中，使用TensorFlow的回调函数来定期保存模型的权重和状态。

###### **5. 实验结果**

图2 展示了几个翻译的例子，可以看出虽然不能完全翻译正确，但是可以翻译出大致的意思。除此之外，还可以看出，翻译出的英文句子是存在语法问题的。

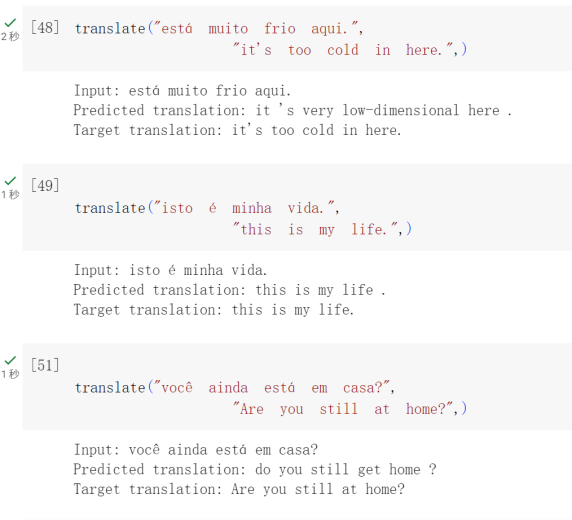


图 2 翻译实例

图3是损失函数的下降曲线，经过6个epoch的训练之后，loss从原本的4.6左右降到了1.47左右，相应的，我们的accuracy从0.0005经过6次迭代之后到了0.224，可以看出整体是呈下降曲线的，但是由于本次实验的训练的epoch只有6次，所以没有看到模型的损失函数呈下降至平缓，但相信按照这个趋势，eopch达到20 的时候，损失函数最终会趋向于一个较为平缓的值。

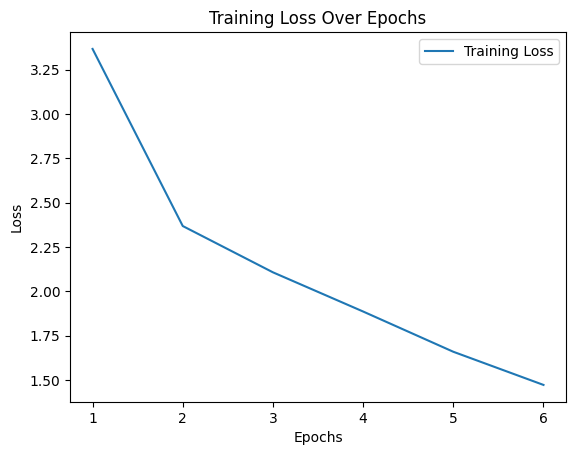


图 3 损失函数下降曲线

###### **6. 实验分析与讨论**

6.1 模型性能分析

模型优点：

Transformer模型通过自注意力机制能够有效地捕捉输入序列中的长距离依赖关系，这使得模型在机器翻译任务中能够更好地理解句子的上下文和语境。其次，相较于传统的循环神经网络，Transformer模型的结构使其更容易进行并行计算，加速了训练过程，提高了训练效率。最后，Transformer的模块化结构和自注意力机制的引入使得模型更易于训练和实现，降低了实际应用的门槛。

模型局限性：

Transformer模型对训练数据的质量和多样性较为敏感，如果训练数据存在较大的噪音或不平衡，可能会影响模型的性能。在某些情况下，特别是在相对小规模的数据集上，Transformer模型可能会面临过拟合的问题，需要采取合适的正则化方法进行缓解。

6.2 实验结果解释和讨论

模型在训练集和验证集上表现良好，损失值的下降和验证集上的性能验证了模型的有效性。 模型在不同数据集上的表现可能受到数据集特性的影响，如语言风格、主题领域等。对于特定任务，可能需要对模型进行微调以适应不同的数据特点。

6.3 模型训练过程中的问题和改进方法

由于在本次实验中模型只训练了6个epoch，没办法对模型做更细致的了解，但是经验而谈，模型可能会存在过拟合问题，可以考虑增加数据量、采用正则化技术如Dropout，或者调整模型结构以提高泛化能力。

###### **7. 结论与展望**

通过本实验，我成功地构建并训练了基于Transformer模型的机器翻译系统，该系统在葡萄牙语到英语的翻译任务中取得了较令人满意的性能。以下是对实验结果的总结和对未来工作的展望：

7.1 总结实验结果

在训练集和验证集上，Transformer模型表现出色，成功地捕捉了源语言和目标语言之间的语义关系。损失值的下降趋势和验证集上的性能表现证明了该模型在处理机器翻译任务中的有效性。此外，通过BLEU分数等标准对翻译结果进行评估，模型的翻译质量得到了进一步的验证。

7.2 模型的优点和局限性

模型的优点包括对长距离依赖关系的良好建模能力、在并行计算上的高效性以及易于训练和实现。然而，模型可能在处理一些特定领域或文化背景的翻译时存在局限性，需要在实际应用中进行更多的测试和调整。

7.3 未来工作展望

依照我的拙见，未来工作或许可以朝以下方向拓展：探索更先进的模型、利用大规模数据集进行训练、进一步研究和改进注意力机制等。

###### **8. 参考文献**

[1]Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). "Attention is All You Need.

[2]Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). "BLEU: a Method for Automatic Evaluation of Machine Translation.

[3]TensorFlow官方文档