实验2 机器翻译

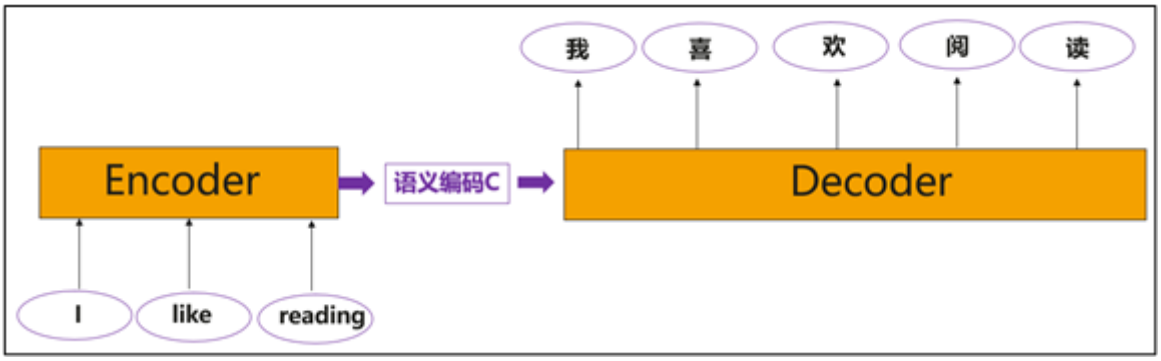
1.实验背景及简介

1.1实验背景

在实际生活中，机器翻译是人工智能技术比较广泛的一个应用。在机器翻译应用中，由于输入序列与输出序列的长度通常不一样。需要采用序列到序列（Sequence to Sequene, Seq2Seq）的映射框架来解决这一问题，它能将一个可变长序列映射到另一个可变长序列。本次实验将探索Seq2Seq基础模型在机器翻译中的应用，以及Attention注意力机制、Transformer模型对基础Seq2Seq模型的改进。

1.2实验简介

翻译任务在日常生活应用广泛，如手机中有各种翻译软件，可以满足人们交流、阅读的需求。本实验基于Seq2Seq编码器-解码器框架，结合GRU单元实现英文转中文的翻译任务，框架示意图如下：



> GRU（门递归单元）是一种递归神经网络算法，就像LSTM（长短期存储器）一样。它是由Kyunghyun Cho、Bart van Merrienboer等在2014年的文章“使用RNN编码器-解码器学习短语表示用于统计机器翻译”中提出的。本文提出了一种新的神经网络模型RNN Encoder-Decoder，该模型由两个递归神经网络（RNN）组成，为了提高翻译任务的效果，我们还参考了“神经网络的序列到序列学习”和“联合学习对齐和翻译的神经机器翻译”。

2.实验目的

1. 掌握神经网络在NLP机器翻译领域的应用
2. 理解seq2seq解码器-编码器框架
3. 理解attention注意力机制，transformer架构

3.实验步骤

3.1 数据上传与同步(OBS)

- Seq2Seq

```
In [1]: import moxing as mox
        mox.file.copy_parallel(src_url="s3://bucket-mzh/lab2/源代码-Seq2Seq/data/", dst_url='./data/')
        path_to_file = "data/cmn.txt"
```

```
INFO:root:Using MoXing-v1.17.3-d858ff4a
INFO:root:Using OBS-Python-SDK-3.20.9.1
```

- Transformer

```
In [1]: import moxing as mox
        mox.file.copy_parallel(src_url="s3://bucket-mzh/lab2/源代码-transformer/mt_transformer_mindspore_1.1/data/", dst_url='./data/')
        mox.file.copy_parallel(src_url="s3://bucket-mzh/lab2/源代码-transformer/mt_transformer_mindspore_1.1/src/", dst_url='./src/')
```

```
INFO:root:Using MoXing-v1.17.3-d858ff4a
INFO:root:Using OBS-Python-SDK-3.20.9.1
```

3.1 Seq2Seq

- 数据预处理

```python
def prepare_data(data_path, vocab_save_path, max_seq_len):
    with open(data_path, 'r', encoding='utf-8') as f:
        data = f.read()

    # 读取文本文件，按行分割，再将每行分割成语句对
    data = data.split('\n')

     # 截取前2000行数据进行训练
    data = data[:2000]
```

```
    # 分割每行中的中英文
    en_data = [normalizeString(line.split('\t')[0]) for line in data]

    ch_data = [line.split('\t')[1] for line in data]

    # 利用集合，获得中英文词汇表
    en_vocab = set(' '.join(en_data).split(' '))
    id2en = [EOS] + [SOS] + list(en_vocab)
    en2id = {c:i for i,c in enumerate(id2en)}
    en_vocab_size = len(id2en)
    # np.savetxt(os.path.join(vocab_save_path, 'en_vocab.txt'), np.array(id2en), fmt='%s')

    ch_vocab = set(''.join(ch_data))
    id2ch = [EOS] + [SOS] + list(ch_vocab)
    ch2id = {c:i for i,c in enumerate(id2ch)}
    ch_vocab_size = len(id2ch)
    # np.savetxt(os.path.join(vocab_save_path, 'ch_vocab.txt'), np.array(id2ch), fmt='%s')

    # 将句子用词汇表id表示
    en_num_data = np.array([[1] + [int(en2id[en]) for en in line.split(' ')] + [0] for line in en_data])
    ch_num_data = np.array([[1] + [int(ch2id[ch]) for ch in line] + [0] for line in ch_data])

    #将短句子扩充到统一的长度
    for i in range(len(en_num_data)):
        num = max_seq_len + 1 - len(en_num_data[i])
        if(num >= 0):
            en_num_data[i] += [0]*num
        else:
            en_num_data[i] = en_num_data[i][:max_seq_len] + [0]

    for i in range(len(ch_num_data)):
        num = max_seq_len + 1 - len(ch_num_data[i])
        if(num >= 0):
            ch_num_data[i] += [0]*num
        else:
            ch_num_data[i] = ch_num_data[i][:max_seq_len] + [0]


    np.savetxt(os.path.join(vocab_save_path, 'en_vocab.txt'), np.array(id2en), fmt='%s')

    np.savetxt(os.path.join(vocab_save_path, 'ch_vocab.txt'), np.array(id2ch), fmt='%s')

    return en_num_data, ch_num_data, en_vocab_size, ch_vocab_size
```

- 设置超参数

```
from easydict import EasyDict as edict

# CONFIG
cfg = edict({
```

```
    'en_vocab_size': 1154,
    'ch_vocab_size': 1116,
    'max_seq_length': 10,
    'hidden_size': 1024,
    'batch_size': 16,
    'eval_batch_size': 1,
    'learning_rate': 0.001,
    'momentum': 0.9,
    'num_epochs': 15,
    'save_checkpoint_steps': 125,
    'keep_checkpoint_max': 10,
    'dataset_path':'./preprocess',
    'ckpt_save_path':'./ckpt',
    'checkpoint_path':'./ckpt/gru-15_125.ckpt'
})
```

- GRU单元

```
class GRU(nn.Cell):
    def __init__(self, config, is_training=True):
        super(GRU, self).__init__()
        if is_training:
            self.batch_size = config.batch_size
        else:
            self.batch_size = config.eval_batch_size
        self.hidden_size = config.hidden_size
        self.weight_i, self.weight_h, self.bias_i, self.bias_h = \
            gru_default_state(self.batch_size, self.hidden_size,
self.hidden_size)
        self.rnn = P.DynamicGRUV2()
        self.cast = P.Cast()

    def construct(self, x, hidden):
        x = self.cast(x, mstype.float16)
        y1, h1, _, _, _, _ = self.rnn(x, self.weight_i, self.weight_h,
self.bias_i, self.bias_h, None, hidden)
        return y1, h1
```

- Encoder

```
class Encoder(nn.Cell):
    def __init__(self, config, is_training=True):
        super(Encoder, self).__init__()
        self.vocab_size = config.en_vocab_size
        self.hidden_size = config.hidden_size
        if is_training:
            self.batch_size = config.batch_size
        else:
            self.batch_size = config.eval_batch_size

        self.trans = P.Transpose()
        self.perm = (1, 0, 2)
        self.embedding = nn.Embedding(self.vocab_size, self.hidden_size)
        self.gru = GRU(config,
is_training=is_training).to_float(mstype.float16)
```

```
        self.h = Tensor(np.zeros((self.batch_size,
self.hidden_size)).astype(np.float16))

    def construct(self, encoder_input):
        embeddings = self.embedding(encoder_input)
        embeddings = self.trans(embeddings, self.perm)
        output, hidden = self.gru(embeddings, self.h)
        return output, hidden
```

- Decoder

```
class Decoder(nn.Cell):
    def __init__(self, config, is_training=True, dropout=0.1):
        super(Decoder, self).__init__()

        self.vocab_size = config.ch_vocab_size
        self.hidden_size = config.hidden_size
        self.max_len = config.max_seq_length

        self.trans = P.Transpose()
        self.perm = (1, 0, 2)
        self.embedding = nn.Embedding(self.vocab_size, self.hidden_size)
        self.dropout = nn.Dropout(1-dropout)
        self.attn = nn.Dense(self.hidden_size, self.max_len)
        self.softmax = nn.Softmax(axis=2)
        self.bmm = P.BatchMatMul()
        self.concat = P.Concat(axis=2)
        self.attn_combine = nn.Dense(self.hidden_size * 2,
self.hidden_size)

        self.gru = GRU(config,
is_training=is_training).to_float(mstype.float16)
        self.out = nn.Dense(self.hidden_size, self.vocab_size)
        self.logsoftmax = nn.LogSoftmax(axis=2)
        self.cast = P.Cast()
   def construct(self, decoder_input, hidden, encoder_output):
        embeddings = self.embedding(decoder_input)
        embeddings = self.dropout(embeddings)
        # calculate attn
        attn_weights = self.softmax(self.attn(embeddings))
        encoder_output = self.trans(encoder_output, self.perm)
        attn_applied = self.bmm(attn_weights,
self.cast(encoder_output,mstype.float32))
        output =  self.concat((embeddings, attn_applied))
        output = self.attn_combine(output)


        embeddings = self.trans(embeddings, self.perm)
        output, hidden = self.gru(embeddings, hidden)
        output = self.cast(output, mstype.float32)
        output = self.out(output)
        output = self.logsoftmax(output)

        return output, hidden, attn_weights
```

- Seq2Seq整体结构

```
class Seq2Seq(nn.Cell):
    def __init__(self, config, is_train=True):
        super(Seq2Seq, self).__init__()
        self.max_len = config.max_seq_length
        self.is_train = is_train

        self.encoder = Encoder(config, is_train)
        self.decoder = Decoder(config, is_train)
        self.expanddims = P.ExpandDims()
        self.squeeze = P.Squeeze(axis=0)
        self.argmax = P.ArgMaxWithValue(axis=int(2), keep_dims=True)
        self.concat = P.Concat(axis=1)
        self.concat2 = P.Concat(axis=0)
        self.select = P.Select()

    def construct(self, src, dst):
        encoder_output, hidden = self.encoder(src)
        decoder_hidden = self.squeeze(encoder_output[self.max_len-
2:self.max_len-1:1, ::, ::])
        if self.is_train:
            outputs, _ = self.decoder(dst, decoder_hidden, encoder_output)
        else:
            decoder_input = dst[::,0:1:1]
            decoder_outputs = ()
            for i in range(0, self.max_len):
                decoder_output, decoder_hidden, _ =
self.decoder(decoder_input,

decoder_hidden, encoder_output)
                decoder_hidden = self.squeeze(decoder_hidden)
                decoder_output, _ = self.argmax(decoder_output)
                decoder_output = self.squeeze(decoder_output)
                decoder_outputs += (decoder_output,)
                decoder_input = decoder_output
            outputs = self.concat(decoder_outputs)
        return outputs
```

- 损失函数

```
class NLLLoss(_Loss):
    '''
       NLLLoss function
    '''
    def __init__(self, reduction='mean'):
        super(NLLLoss, self).__init__(reduction)
        self.one_hot = P.OneHot()
        self.reduce_sum = P.ReduceSum()

    def construct(self, logits, label):
        label_one_hot = self.one_hot(label, F.shape(logits)[-1],
F.scalar_to_array(1.0),
                                     F.scalar_to_array(0.0))
        #print('NLLLoss label_one_hot:',label_one_hot, label_one_hot.shape)
        #print('NLLLoss logits:',logits, logits.shape)
        #print('xxx:', logits * label_one_hot)
        loss = self.reduce_sum(-1.0 * logits * label_one_hot, (1,))
        return self.get_loss(loss)
```

- 训练网络与优化器

```
network = Seq2Seq(cfg)
network = WithLossCell(network, cfg)
optimizer = nn.Adam(network.trainable_params(),
learning_rate=cfg.learning_rate, beta1=0.9, beta2=0.98)
model = Model(network, optimizer=optimizer)
```

- 构建模型

```
loss_cb = LossMonitor()
config_ck =
CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps,
keep_checkpoint_max=cfg.keep_checkpoint_max)
ckpoint_cb = ModelCheckpoint(prefix="gru", directory=cfg.ckpt_save_path,
config=config_ck)
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
callbacks = [time_cb, ckpoint_cb, loss_cb]

model.train(cfg.num_epochs, ds_train, callbacks=callbacks,
dataset_sink_mode=True)
```

3.2 Transformer

- 数据处理函数

```
def data_prepare(cfg, eval_idx):
    tokenizer = tokenization.WhiteSpaceTokenizer(vocab_file=cfg.vocab_file)

    writer_train = FileWriter(cfg.train_file_mindrecord, cfg.num_splits)
    writer_eval = FileWriter(cfg.eval_file_mindrecord, cfg.num_splits)
    data_schema = {"source_sos_ids": {"type": "int32", "shape": [-1]},
                   "source_sos_mask": {"type": "int32", "shape": [-1]},
                   "source_eos_ids": {"type": "int32", "shape": [-1]},
                   "source_eos_mask": {"type": "int32", "shape": [-1]},
                   "target_sos_ids": {"type": "int32", "shape": [-1]},
                   "target_sos_mask": {"type": "int32", "shape": [-1]},
                   "target_eos_ids": {"type": "int32", "shape": [-1]},
                   "target_eos_mask": {"type": "int32", "shape": [-1]}
                   }

    writer_train.add_schema(data_schema, "tranformer train")
    writer_eval.add_schema(data_schema, "tranformer eval")

    index = 0
    f_train = open(cfg.train_file_source, 'w', encoding='utf-8')
    f_test = open(cfg.eval_file_source,'w',encoding='utf-8')
    f = open(cfg.input_file, "r", encoding='utf-8')
    for s_line in f:
        print("finish {}/{}".format(index, 23607), end='\r')

        line = tokenization.convert_to_unicode(s_line)

        source_line, target_line = line.strip().split("\t")
```

```
        source_tokens = tokenizer.tokenize(source_line)
        target_tokens = tokenizer.tokenize(target_line)

        if len(source_tokens) >= (cfg.max_seq_length-1) or
len(target_tokens) >= (cfg.max_seq_length-1):
            if cfg.clip_to_max_len:
                source_tokens = source_tokens[:cfg.max_seq_length-1]
                target_tokens = target_tokens[:cfg.max_seq_length-1]
            else:
                continue

        index = index + 1
        # print(source_tokens)
        instance = create_training_instance(source_tokens, target_tokens,
cfg.max_seq_length)

        if index in eval_idx:
            f_test.write(s_line)
            features = write_instance_to_file(writer_eval, instance,
tokenizer, cfg.max_seq_length)
        else:
            f_train.write(s_line)
            features = write_instance_to_file(writer_train, instance,
tokenizer, cfg.max_seq_length)
    f.close()
    f_test.close()
    f_train.close()
    writer_train.commit()
    writer_eval.commit()
```

- 定义数据加载函数

```
def load_dataset(batch_size=1, data_file=None):
    """
    Load mindrecord dataset
    """
    ds = de.MindDataset(data_file,
                        columns_list=["source_eos_ids", "source_eos_mask",
                                      "target_sos_ids", "target_sos_mask",
                                      "target_eos_ids", "target_eos_mask"],
                        shuffle=False)
    type_cast_op = deC.TypeCast(mstype.int32)
    ds = ds.map(input_columns="source_eos_ids", operations=type_cast_op)
    ds = ds.map(input_columns="source_eos_mask", operations=type_cast_op)
    ds = ds.map(input_columns="target_sos_ids", operations=type_cast_op)
    ds = ds.map(input_columns="target_sos_mask", operations=type_cast_op)
    ds = ds.map(input_columns="target_eos_ids", operations=type_cast_op)
    ds = ds.map(input_columns="target_eos_mask", operations=type_cast_op)
    # apply batch operations
    ds = ds.batch(batch_size, drop_remainder=True)
    ds.channel_name = 'transformer'
    return ds
```

- 训练函数

```
def train(cfg):
```

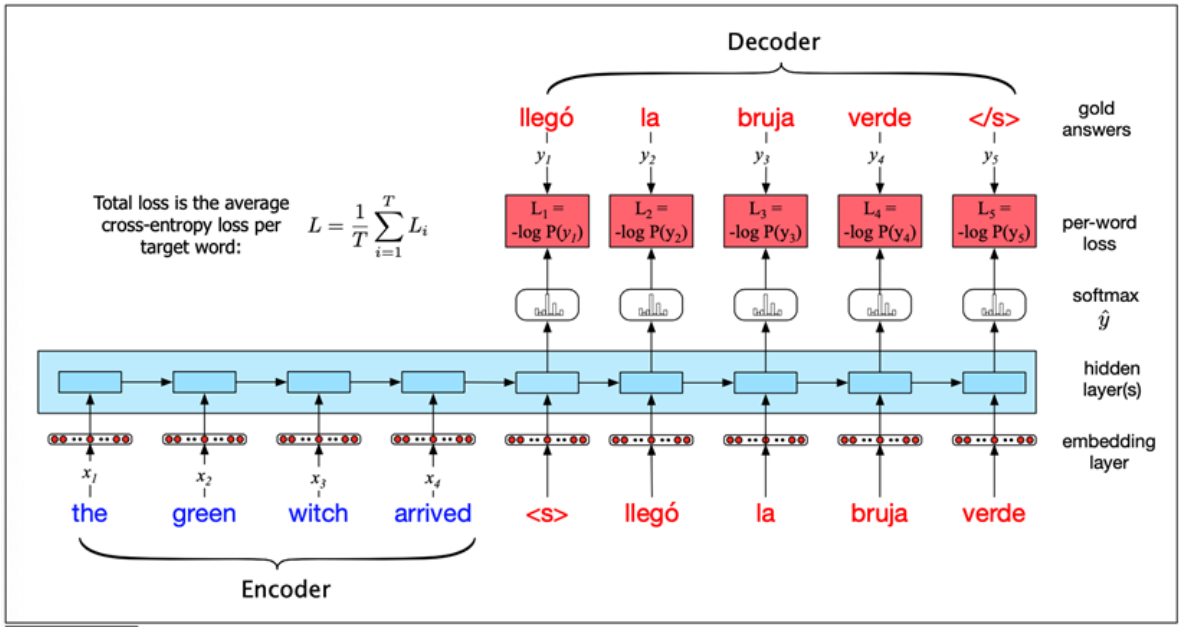
```
    """
    Transformer training.
    """

    train_dataset = load_dataset(cfg.batch_size, data_file=cfg.data_path)

    netwithloss = TransformerNetworkWithLoss(transformer_net_cfg, True)

    if cfg.checkpoint_path:
        parameter_dict = load_checkpoint(cfg.checkpoint_path)
        load_param_into_net(netwithloss, parameter_dict)

    lr =
Tensor(create_dynamic_lr(schedule="constant*rsqrt_hidden*linear_warmup*rsqr
t_decay",

 training_steps=train_dataset.get_dataset_size()*cfg.epoch_size,

 learning_rate=cfg.lr_schedule.learning_rate,

 warmup_steps=cfg.lr_schedule.warmup_steps,

 hidden_size=transformer_net_cfg.hidden_size,

 start_decay_step=cfg.lr_schedule.start_decay_step,
                                  min_lr=cfg.lr_schedule.min_lr),
mstype.float32)
    optimizer = Adam(netwithloss.trainable_params(), lr)

    callbacks = [TimeMonitor(train_dataset.get_dataset_size()),
LossCallBack()]
    if cfg.enable_save_ckpt:
        ckpt_config =
CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps,

keep_checkpoint_max=cfg.save_checkpoint_num)
        ckpoint_cb = ModelCheckpoint(prefix=cfg.save_checkpoint_name,
directory=cfg.save_checkpoint_path, config=ckpt_config)
        callbacks.append(ckpoint_cb)

    if cfg.enable_lossscale:
        scale_manager =
DynamicLossScaleManager(init_loss_scale=cfg.init_loss_scale_value,

 scale_factor=cfg.scale_factor,

 scale_window=cfg.scale_window)
        update_cell = scale_manager.get_update_cell()
        netwithgrads =
TransformerTrainOneStepWithLossScaleCell(netwithloss,
optimizer=optimizer,scale_update_cell=update_cell)
    else:
        netwithgrads = TransformerTrainOneStepCell(netwithloss,
optimizer=optimizer)

    netwithgrads.set_train(True)
    model = Model(netwithgrads)
```

```
41      model.train(cfg.epoch_size, train_dataset, callbacks=callbacks,
    dataset_sink_mode=cfg.enable_data_sink)
```

4.实验原理

4.1 Seq2Seq

> 在自然语言处理的很多应用中，输入和输出都可以是不定长序列。以机器翻译为例，输入可以是一段不定长的英语文本序列，输出可以是一段不定长的德语文本序列，如下面图中所示。



> 当输入和输出都是不定长序列时，我们可以使用编码器—解码器（encoder-decoder）或者seq2seq模型。序列到序列模型，简称seq2seq模型。这两个模型本质上都用到了两个循环神经网络，分别叫做**编码器和解码器**。编码器用来分析输入序列，解码器用来生成输出序列。两 个循环神经网络是共同训练的。

4.1.1 编码器

> 编码器的作用是把一个不定长的输入序列变换成一个定长的背景变量 c，并在该背景变量中编码输入序列信息。常用的编码器是循环神经网络。假设输入序列是$x_1, \ldots, x_T$，例如 x_i是输入句子中的第 i个词。在时间步 t，循环神经网络将输入 x_t 的特征向量x_t和上个时间步的隐藏状态h_{t-1}变换为当前时间步的隐藏状态h_t。用函数 f 表达循环神经网络隐藏层的变换：

$$h_t = f(x_t, h_{t-1})$$

> 编码器通过自定义函数 q 将各个时间步的隐藏状态变换为背景变量：

$$c = q(h_1, \ldots, h_T)$$

4.1.2 译码器

> 编码器输出的背景变量 c 编码了整个输入序列$x_1, \ldots, x_T$的信息。给定训练样本中的输出序列 $y_1, y_2, \ldots, y_{T'}$，对每个时间步 t'（符号与输入序列或编码器的时间步t有区别），解码器输出$y_{t'}$的条件概率将基于之前的输出序列 $y_1, \ldots, y_{t'-1}$ 和背景变量 c，即：

$$P(y_{t'} \mid y_1, \ldots, y_{t'} - 1, c)$$

> 可以使用另一个循环神经网络作为解码器。在输出序列的时间步 t'，解码器将上一时间步的输出 $y_{t'-1}$ 以及背景变量c作为输入，并将它们与上一时间步的隐藏状态 $s_{t'-1}$ 变换为当前时间步的隐藏状态$s_{t'}$。因此，我们可以用函数g表达解码器隐藏层的变换：

$$s_{t'} = g\left(y_{t'-1}, c, s_{t'-1}\right)$$

4.1.3 训练模型

> 根据最大似然估计，我们可以最大化输出序列基于输入序列的条件概率：

$$\begin{aligned} P\left(y_1, \ldots, y_{t'-1} \mid x_1, \ldots, x_T\right) &= \prod_{t'=1}^{T'} P\left(y_t' \mid y_1, \ldots, y_{t'-1}, x_1, \ldots, x_T\right) \\ &= \prod_{t'=1}^{T'} P\left(y_{t'} \mid y_1, \ldots, y_{t'-1}, c\right) \end{aligned}$$

> 并得到该输出序列的损失：

$$-\log P\left(y_1, \ldots, y_{t'-1} \mid x_1, \ldots, x_T\right) = -\sum_{t'=1}^{T'} \log P\left(y_{t'} \mid y_1, \ldots, y_{t'-1}, c\right)$$

> 假设解码器的输出是一段文本序列。设输出文本词典Y（包含特殊符号“”）的大小为|Y|，输出序列的最大长度为T'。所有可能的输出序列一共有 $O(|y|^{T'})$ 种。这些输出序列中所有特殊符号“”后面的子序列将被舍弃。

4.2 Transformer

Transformer网络如下图所示，其中左边为编码网络，右边为解码网络。

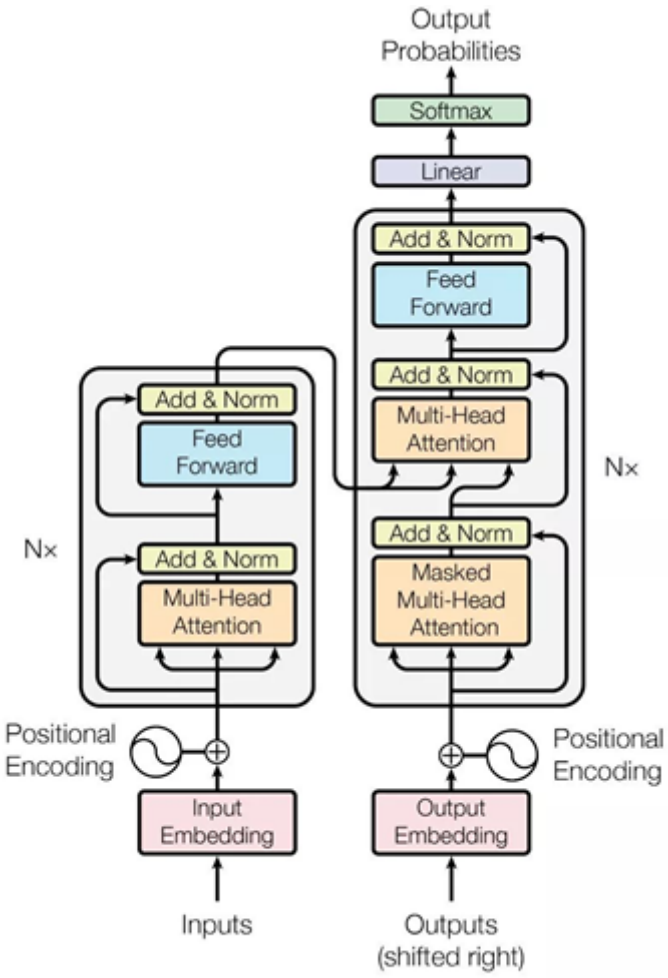


Figure 1: The Transformer - model architecture.

> 每一个编码器在结构上都是一样的，但它们的权重参数是不同的。每一个编码器里面，可以分为2层（Self-Attention 层、前馈神经网络）。输入编码器的文本数据，首先会经过一个 Self Attention 层，这个层处理一个词的时候，不仅会使用这个词本身的信息，也会使用句子中其他词的信息。接下来，Self Attention 层的输出会经过前馈神经网络。同理，解码器也具有这两层，但是这两层中间还插入了一个 Encoder-Decoder Attention 层。

> 在self-attention中，每个单词有3个不同的向量，它们分别是Query向量（Q），Key向量（K）和Value向量（V）。它们是通过3个不同的权值矩阵由嵌入向量乘以三个不同的权值矩阵，self-attention的公式如下：

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

5 实验结果

5.1 Seq2Seq

- 训练

```
In [15]: loss_cb = LossMonitor()
config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpoint_steps, keep_checkpoint_max=cfg.keep_checkpoint_max)
ckpoint_cb = ModelCheckpoint(prefix="gru", directory=cfg.ckpt_save_path, config=config_ck)
time_cb = TimeMonitor(data_size=ds_train.get_dataset_size())
callbacks = [time_cb, ckpoint_cb, loss_cb]

model.train(cfg.num_epochs, ds_train, callbacks=callbacks, dataset_sink_mode=True)
```

```
epoch: 1 step: 125, loss is 2.901339
epoch time: 77857.532 ms, per step time: 622.860 ms
epoch: 2 step: 125, loss is 2.4917648
epoch time: 11361.552 ms, per step time: 90.892 ms
epoch: 3 step: 125, loss is 2.2108736
epoch time: 11393.523 ms, per step time: 91.148 ms
epoch: 4 step: 125, loss is 1.6616195
epoch time: 11329.437 ms, per step time: 90.635 ms
epoch: 5 step: 125, loss is 1.5475513
epoch time: 11329.803 ms, per step time: 90.638 ms
epoch: 6 step: 125, loss is 1.2502322
epoch time: 11373.320 ms, per step time: 90.987 ms
epoch: 7 step: 125, loss is 0.97378296
epoch time: 11316.333 ms, per step time: 90.531 ms
epoch: 8 step: 125, loss is 0.3811617
epoch time: 11423.199 ms, per step time: 91.386 ms
epoch: 9 step: 125, loss is 0.2629779
epoch time: 11470.042 ms, per step time: 91.760 ms
epoch: 10 step: 125, loss is 0.27675834
epoch time: 11495.968 ms, per step time: 91.968 ms
epoch: 11 step: 125, loss is 0.2847989
epoch time: 11563.872 ms, per step time: 92.511 ms
epoch: 12 step: 125, loss is 0.10026628
epoch time: 11553.943 ms, per step time: 92.432 ms
epoch: 13 step: 125, loss is 0.20696855
epoch time: 11676.776 ms, per step time: 93.414 ms
epoch: 14 step: 125, loss is 0.10150813
epoch time: 11522.073 ms, per step time: 92.177 ms
epoch: 15 step: 125, loss is 0.15692167
epoch time: 11475.951 ms, per step time: 91.808 ms
```

- 测试

```
In [23]: translate('i need a bottle of juice')
translate('Tom is a good teacher')
```

```
English ['i', 'need', 'a', 'bottle', 'of', 'juice']
中文 我需要一张邮票。
English ['tom', 'is', 'a', 'good', 'teacher']
中文 汤姆是个真的。
```

5.2 Transformer

- 训练

10. 启动训练

```
In [11]: train(train_cfg)
time: 436852, epoch: 15, step: 8832, outputs are [2.9468496]
time: 436886, epoch: 15, step: 8833, outputs are [2.8108442]
time: 436920, epoch: 15, step: 8834, outputs are [2.9115794]
time: 436954, epoch: 15, step: 8835, outputs are [2.9215643]
time: 436988, epoch: 15, step: 8836, outputs are [2.813668]
time: 437022, epoch: 15, step: 8837, outputs are [2.9565923]
time: 437057, epoch: 15, step: 8838, outputs are [2.885069]
time: 437091, epoch: 15, step: 8839, outputs are [2.8448014]
time: 437125, epoch: 15, step: 8840, outputs are [3.0417776]
time: 437159, epoch: 15, step: 8841, outputs are [3.0115216]
time: 437193, epoch: 15, step: 8842, outputs are [3.072069]
time: 437227, epoch: 15, step: 8843, outputs are [3.1246238]
time: 437261, epoch: 15, step: 8844, outputs are [2.9308126]
time: 437295, epoch: 15, step: 8845, outputs are [3.0508318]
time: 437329, epoch: 15, step: 8846, outputs are [3.0099447]
time: 437363, epoch: 15, step: 8847, outputs are [3.260268]
time: 437397, epoch: 15, step: 8848, outputs are [3.2783072]
time: 437431, epoch: 15, step: 8849, outputs are [3.5525923]
time: 437466, epoch: 15, step: 8850, outputs are [3.7860718]
epoch time: 21820.916 ms, per step time: 36.985 ms
```

- 测试

```
[17]: evaluate(eval_cfg)
```

```
source: Hop in .          跳 进 来 。
result: 在 大 部 份 的 时 候 受 到 了 一 个 普 通 的 东 西 。
source: I ' m up .        我 已 经 起 来 了 。
result: 我 起 床 了 。
source: Why me ?          为 什 么 是 我 ？
result: 为 什 么 我 为 什 么 为 什 么 为 我 为 什 么 要 ？
source: Be fair .         公 平 点 。
result: 公 平 等 公 平 运 公 平 。
source: Be kind .         友 善 点 。
result: 好 运 点 点 点 点 穿 点 点 点 好 运 。
source: Be nice .         和 气 点 。
result: 友 善 点 。
```