中国科学院
自动化研究所
INSTITUTE OF AUTOMATION
CHINESE ACADEMY OF SCIENCES

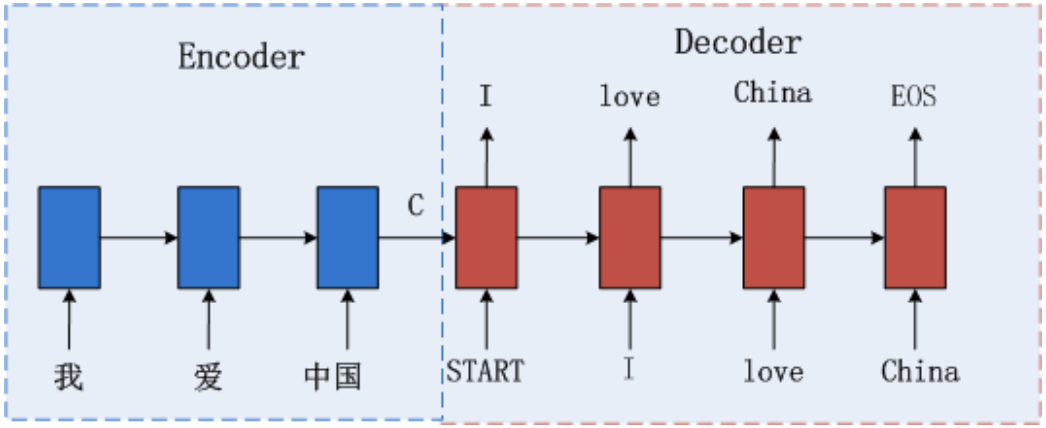
自然语言处理研讨课
（实践课）
第8章 神经机器翻译实践

赵 阳
中国科学院自动化研究所
yang.zhao@nlpr.ia.ac.cn

本章内容

1. 神经机器翻译介绍
2. 实践基础
3. 实践参考
4. 本章实践

1. 神经机器翻译



- 编码器（Encoder）：

  将源语言X编码成语义向量C

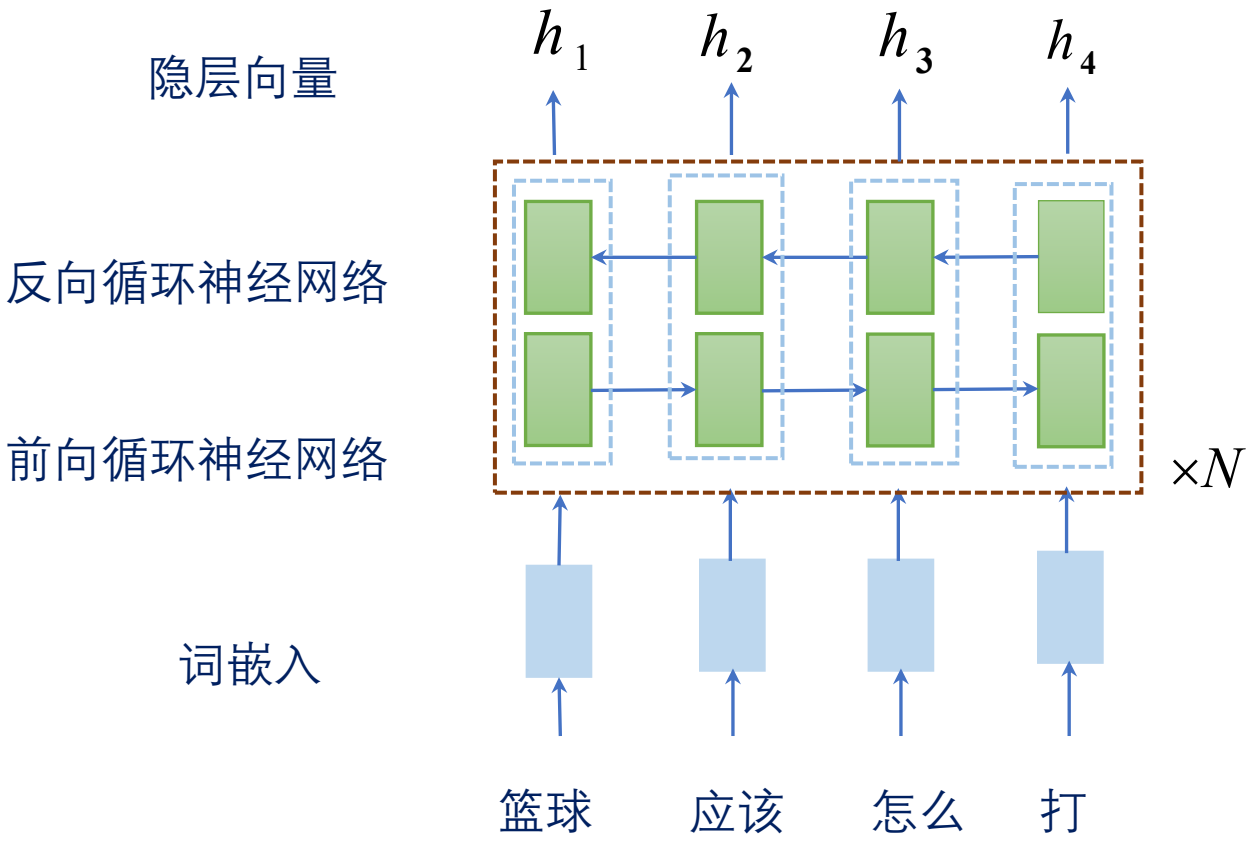
- 编码器（Decoder):

  根据语义向量C逐步生成目标语言Y

$$X \rightarrow C \rightarrow Y$$

1. 神经机器翻译

- **基于循环神经网络建模句子**

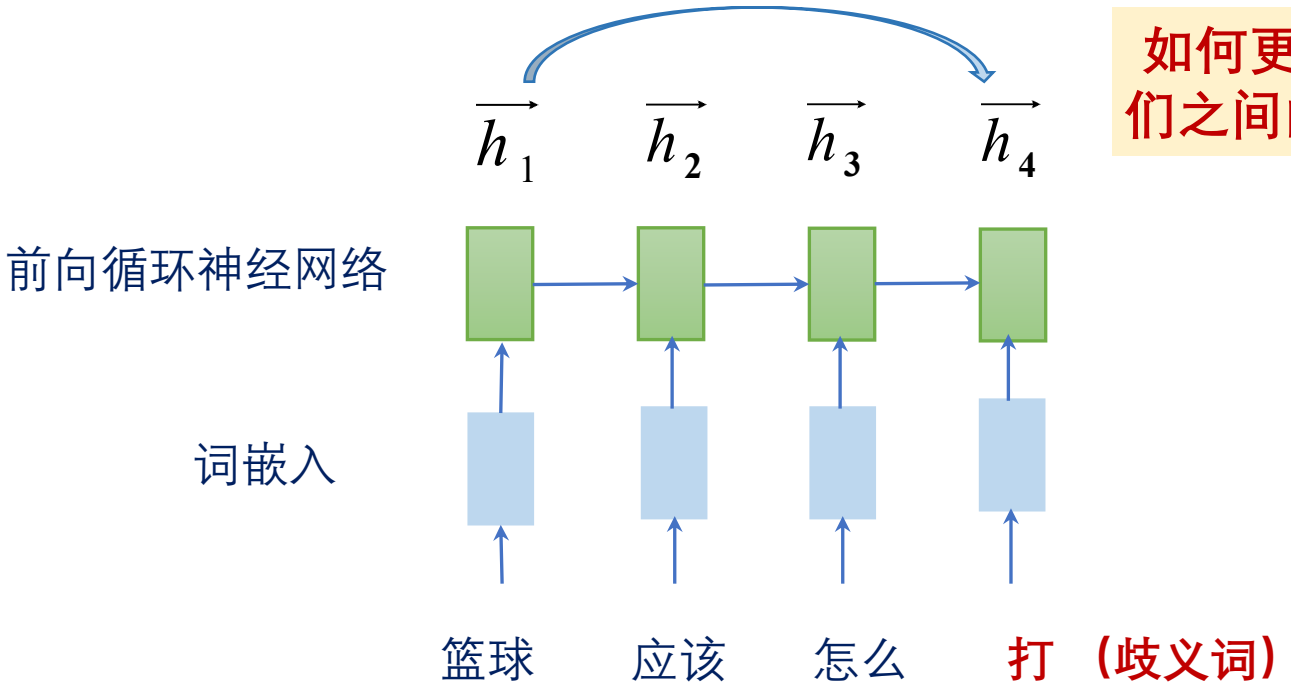


1. 神经机器翻译

- **基于循环神经网络实现句子建模的缺点**

（1）无法较好地处理长距离依赖问题

循环神经网络通过逐词建模的方式对句子进行建模，无法较好处理长距离单词之间的依赖关系。



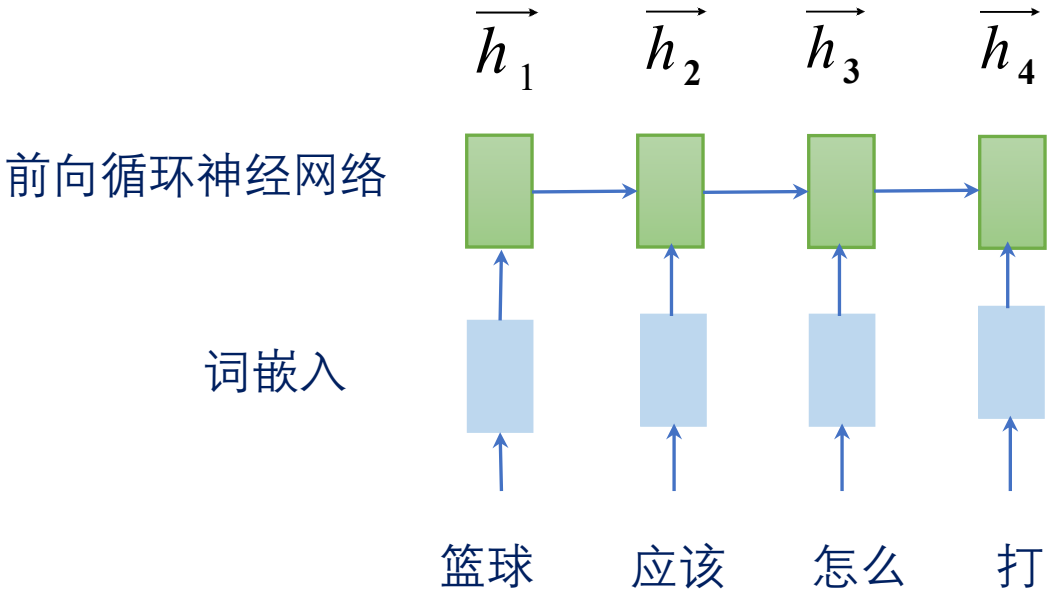
如何更好地建模它们之间的依赖关系？

1. 神经机器翻译

- **基于循环神经网络实现句子建模的缺点**

（2）无法并行计算、计算效率较低

循环神经网络中，位置i的隐层状态的计算$\overrightarrow{h}_i$，需要依赖前一位置i-1的隐层状态 $\overrightarrow{h}_{i\text{-}1}$。



1. 神经机器翻译

- **基于自注意力机制（Self-attention）的神经网络-Transformer**

  1. 更好地建模单词之间的长距离依赖关系
  2. 更好地实现并行计算，提升计算效率

◆ 代表论文：

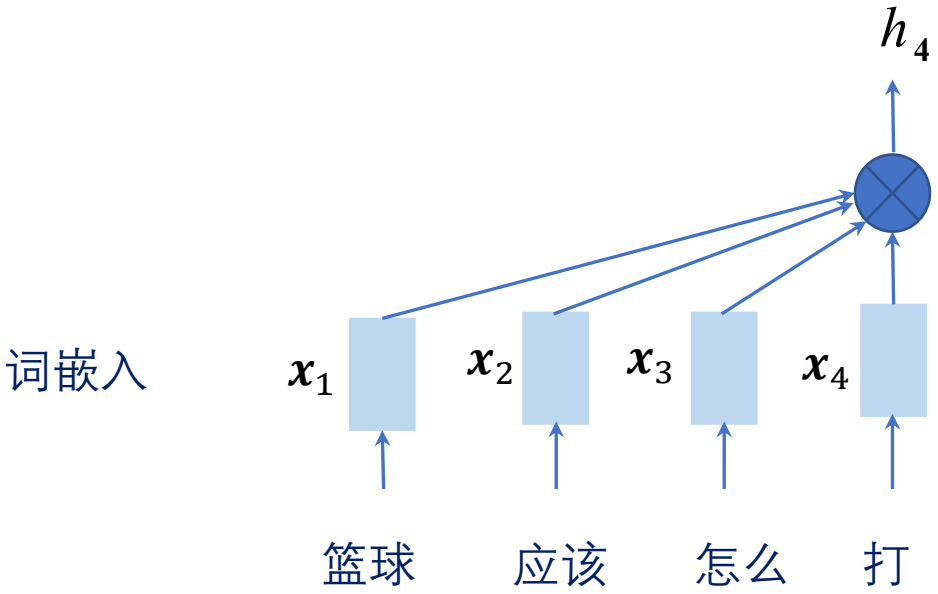
A. Vaswani et al. Attention Is All You Need. Advances in Neural Information Processing Systems 30 (NIPS 2017) December 4-7, 2017, Long Beach, CA, USA, Pages 6000–6010

1. 神经机器翻译

- **自注意力机制的优势：**

（1）能够较好地建模单词间的长距离依赖关系

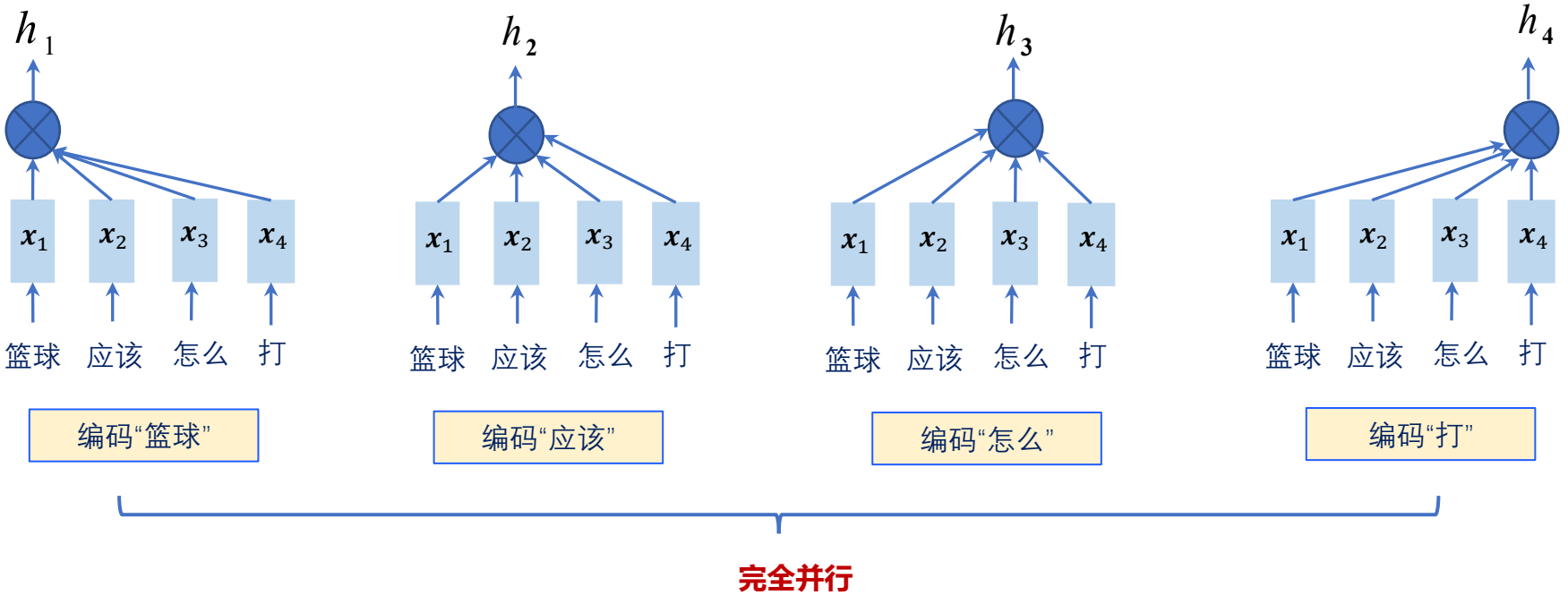
句子中任何两个单词之间通过自注意力机制，能够直接进行连接。



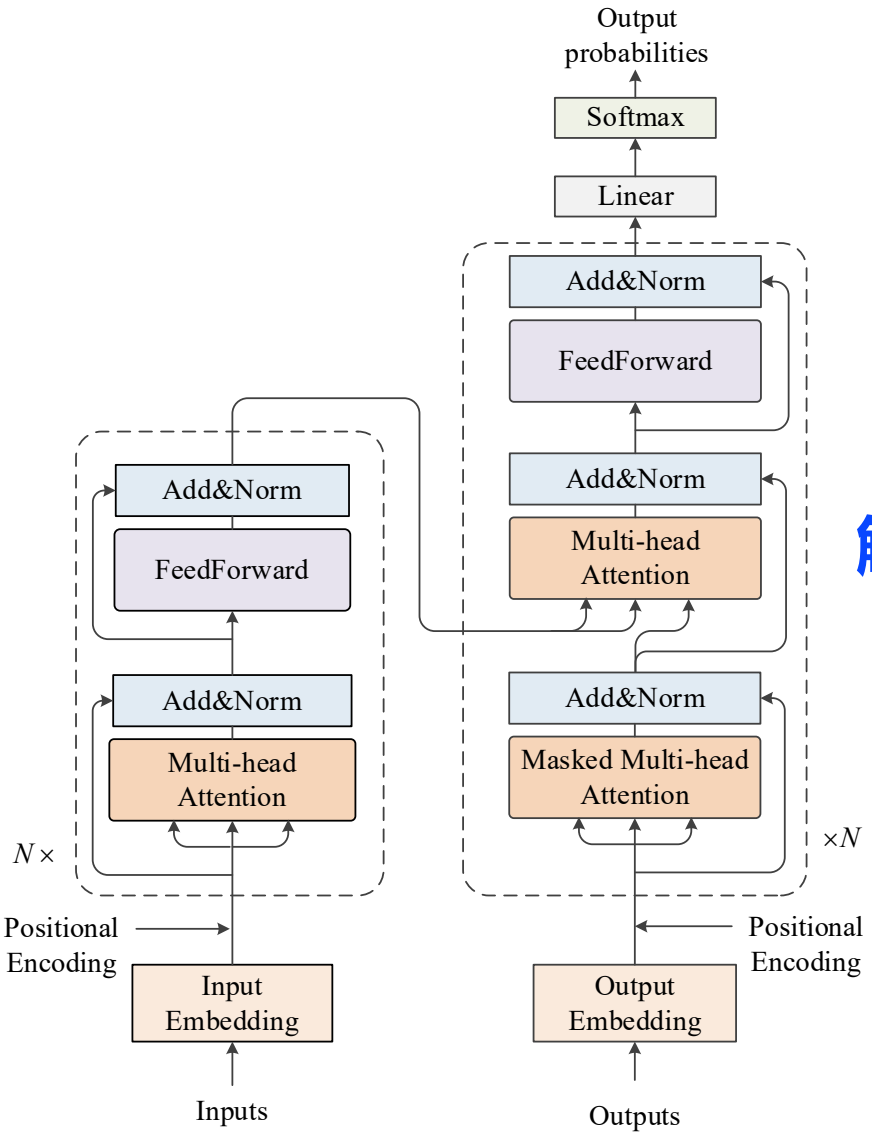
1. 神经机器翻译

- **自注意力机制的优势：**

  （2）能够实现并行计算，提升计算效率



1. 神经机器翻译

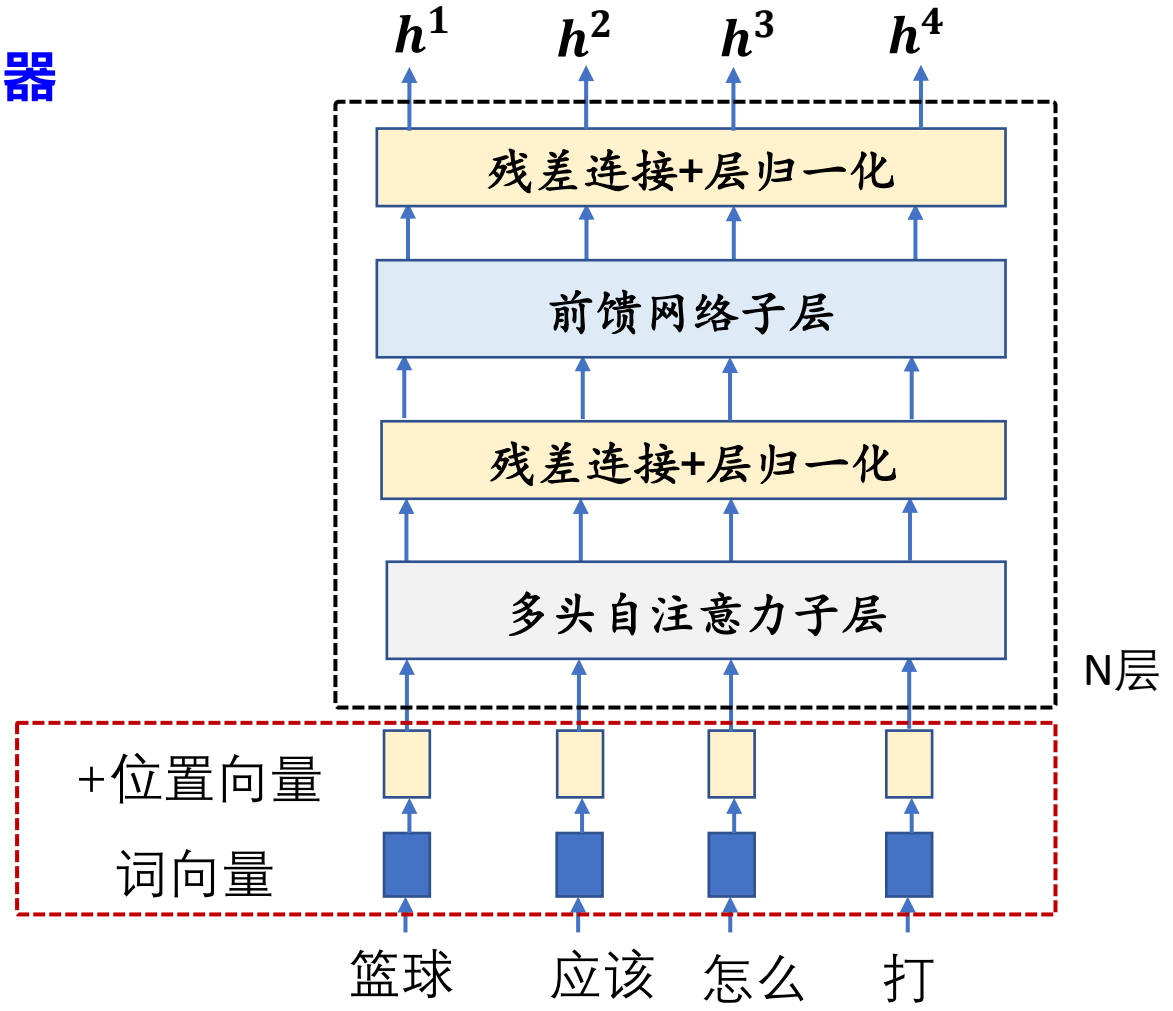


编码器

解码器

1. 神经机器翻译

- Transformer编码器



1. 神经机器翻译

- **为什么要增加位置向量?**
  - 句子中**单词的位置关系**对于句子语义有重要作用
  - 自注意力机制不采用**循环结构**或者**卷积结构**，不包含位置信息

语义不等价

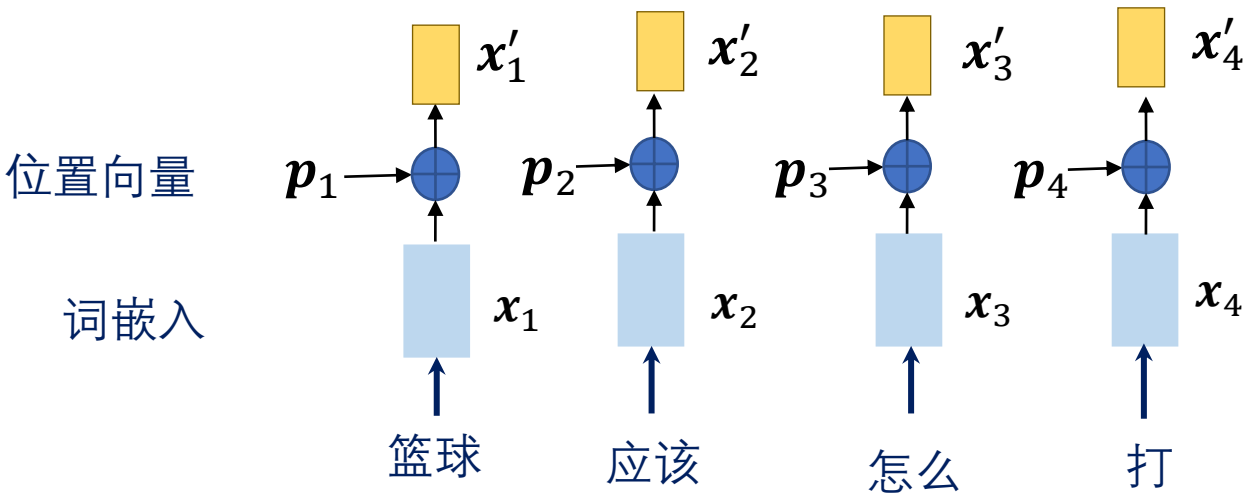
句子1： A队 打败 了 B队 ⟺ **句子2：** B队 打败 了 A队

1. 神经机器翻译

增加位置向量

查询n个单词的词向量$(x_1, x_2, \ldots, x_n)$，添对应的位置向量 $(p_1, p_2, \ldots, p_n)$

$$(\boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_n) = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) + (\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_n)$$



1. 神经机器翻译

- **向量位置如何确定?**

(1) 看作参数，随机初始化，根据训练数据进行优化;

(2) 按照规则确定，并固定不变。

其中，pos是位置，d是总的维度数。$2i$或$2i+1$分别表示第$2i$或者$2i+1$维元素，$0 \le i \le d/2-1$。

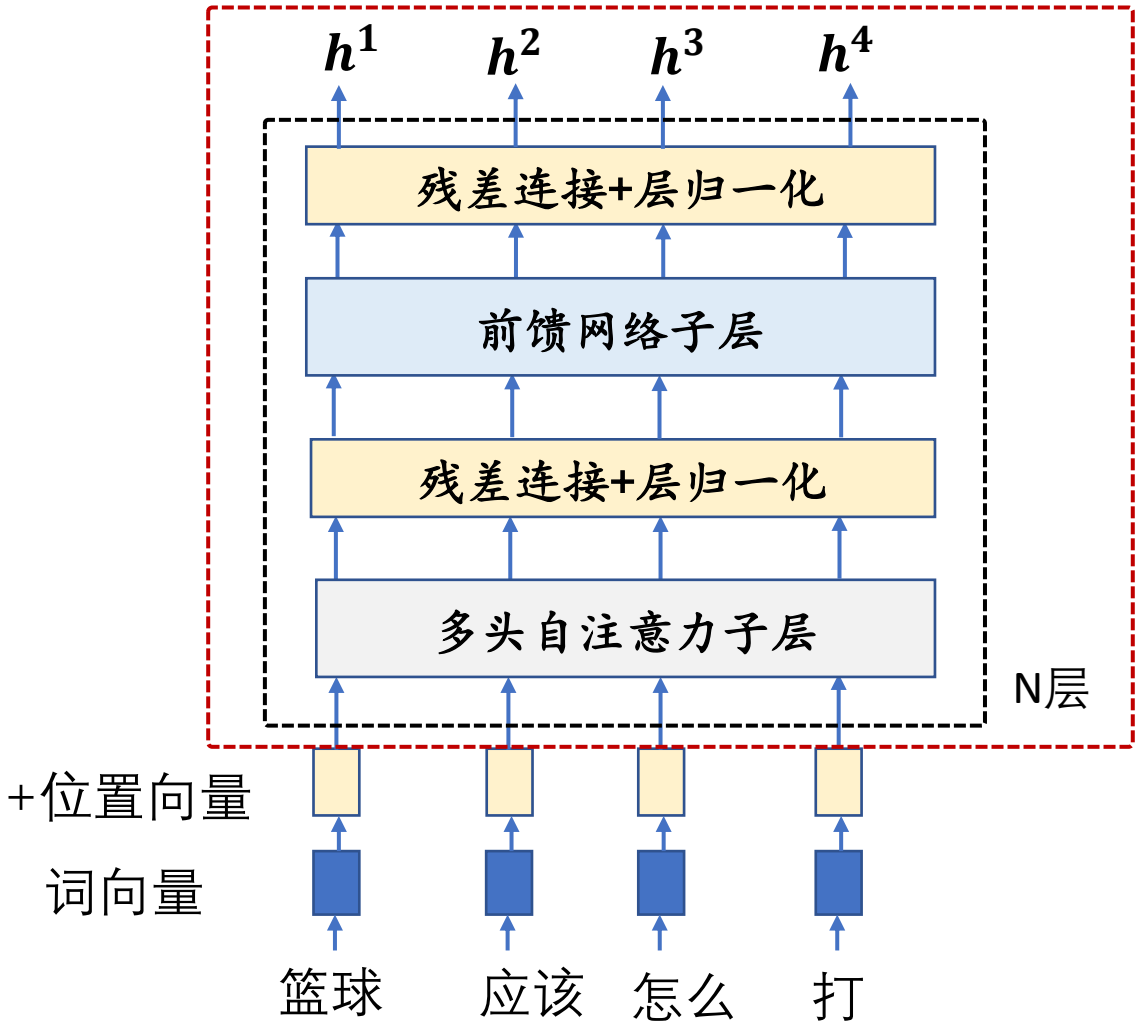
$$\boldsymbol{p}_{pos} = \begin{bmatrix} y_0 \\ \vdots \\ y_i \\ \vdots \\ y_{d-1} \end{bmatrix} \Bigg\} d \text{ (如: } d\text{=500)}$$

$$y_{2i} = \sin\left(\frac{pos}{10000^{(2i/d)}}\right)$$

$$y_{2i+1} = \cos\left(\frac{pos}{10000^{(2i/d)}}\right)$$

1. 神经机器翻译

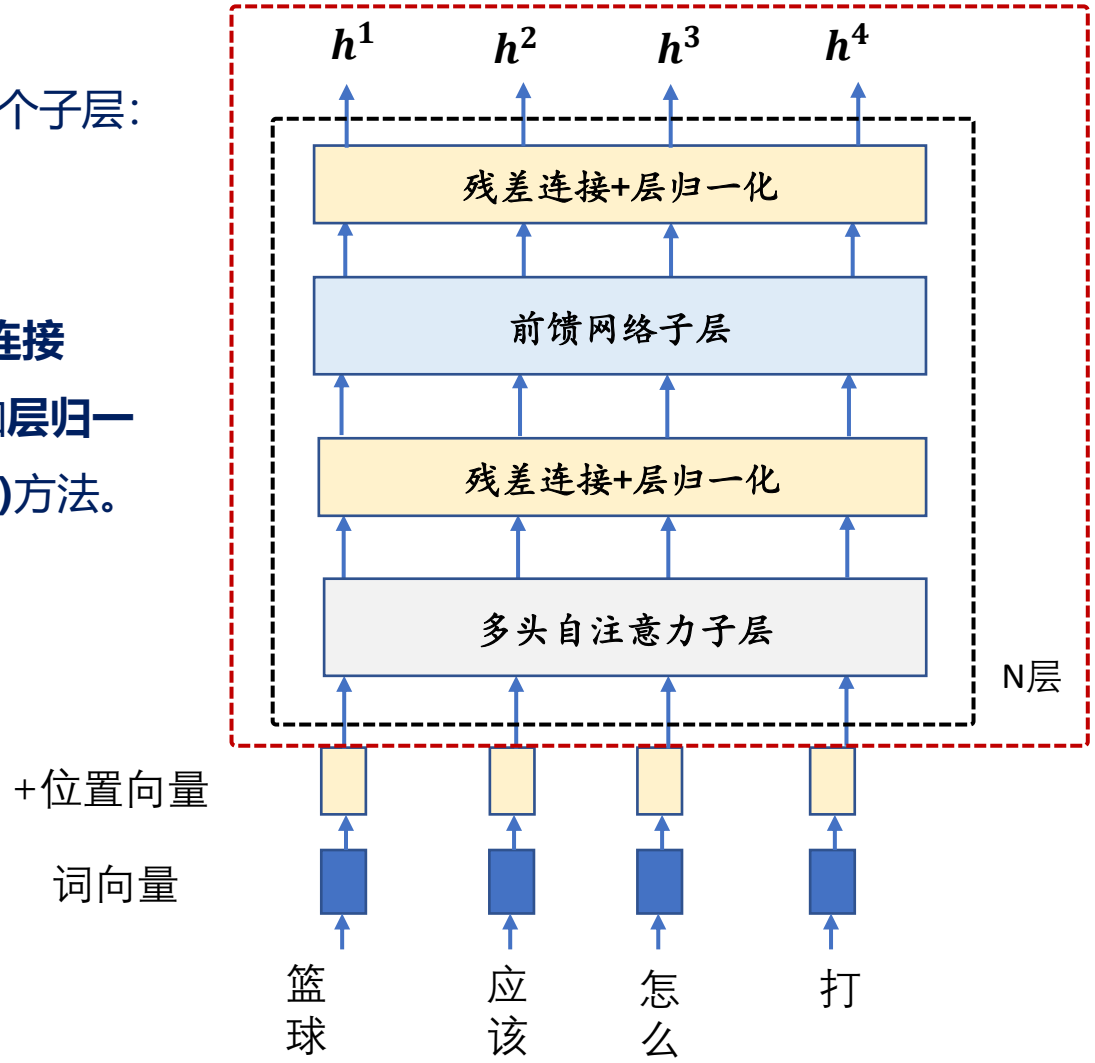
- Transformer编码器



1. 神经机器翻译

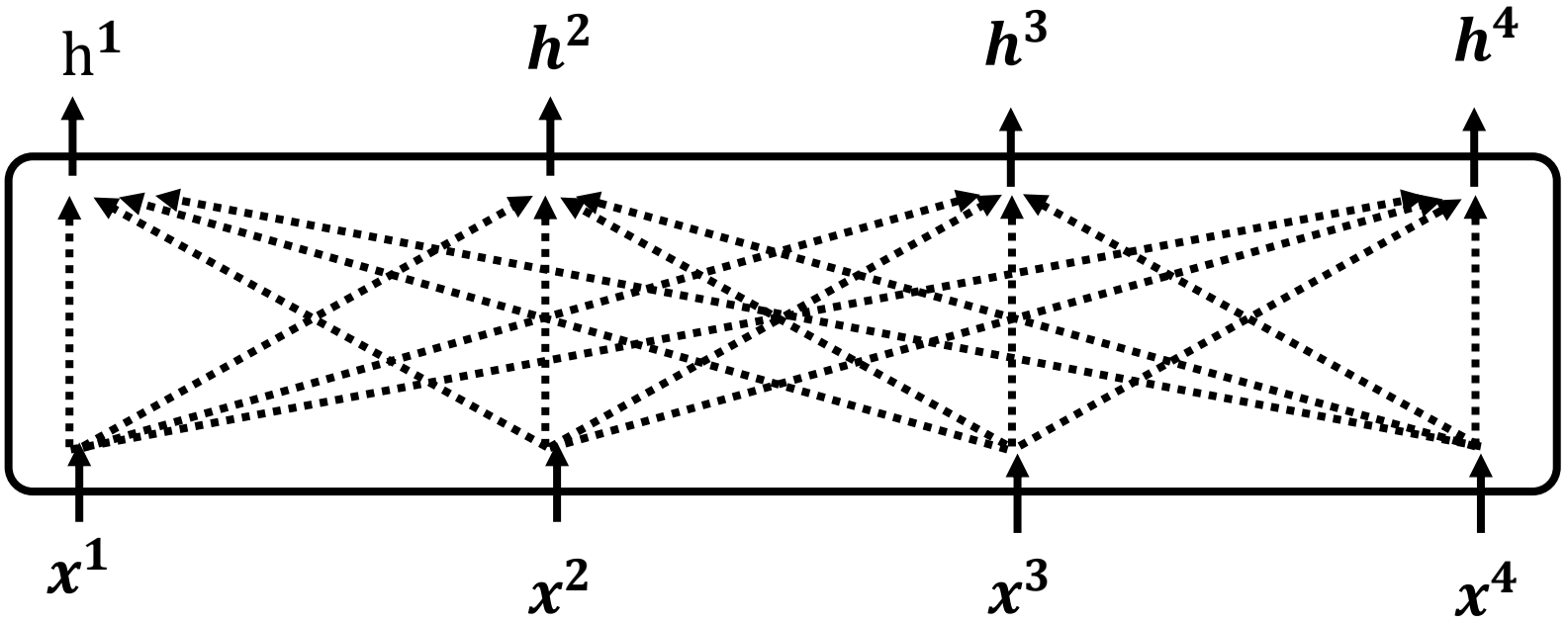
Transformer编码器

- 由N层组成，每一层包括2个子层：
  - **多头自注意力子层**
  - **前馈网络子层**
- 每个子层之后采用了**残差连接(residual connection)**和**层归一化 (layer normalization)**方法。



1. 神经机器翻译

- 多头自注意力子层

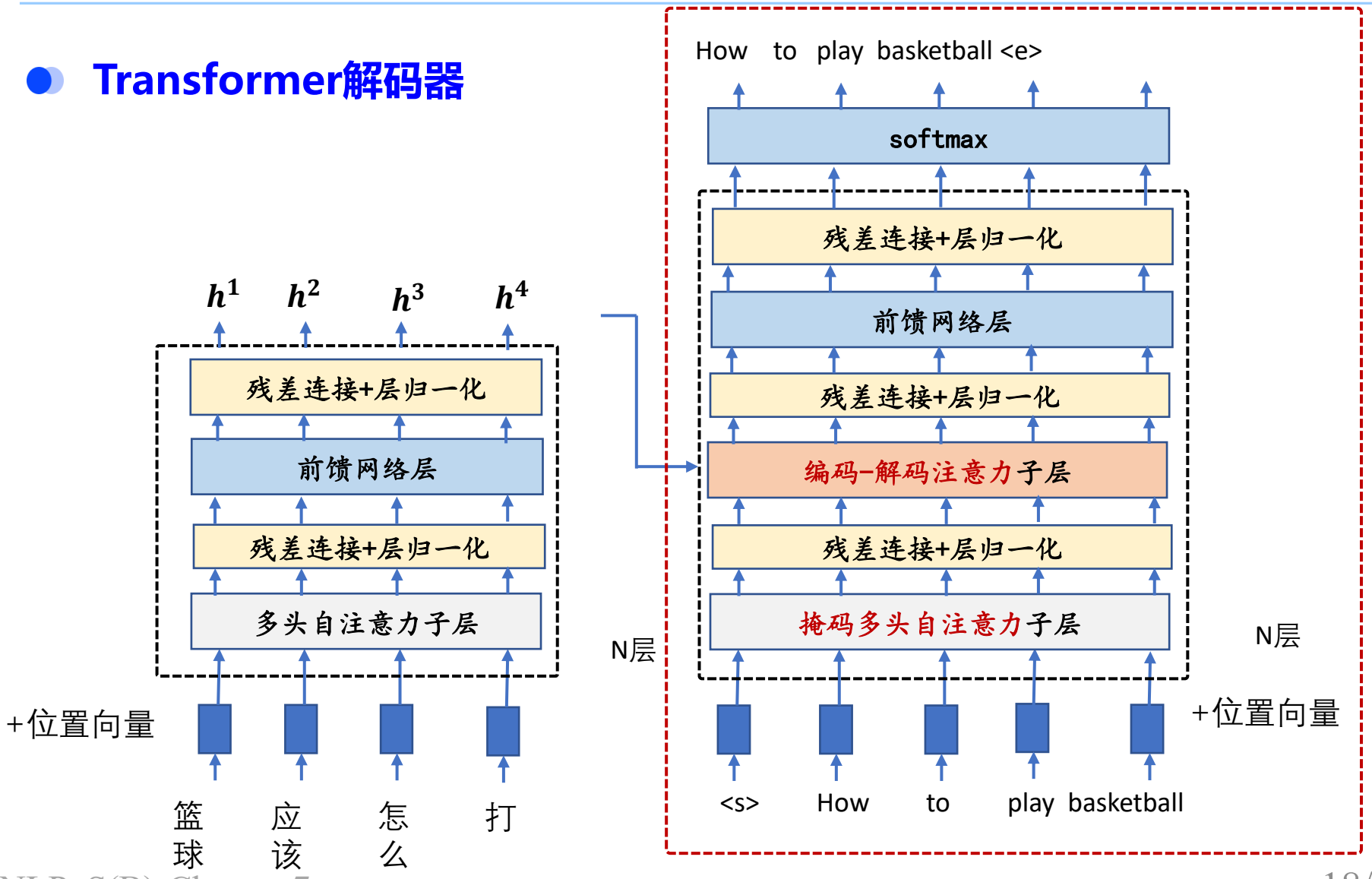


编码器的自注意力机制示意图

（同时注意左右两端上下文）

1. 神经机器翻译

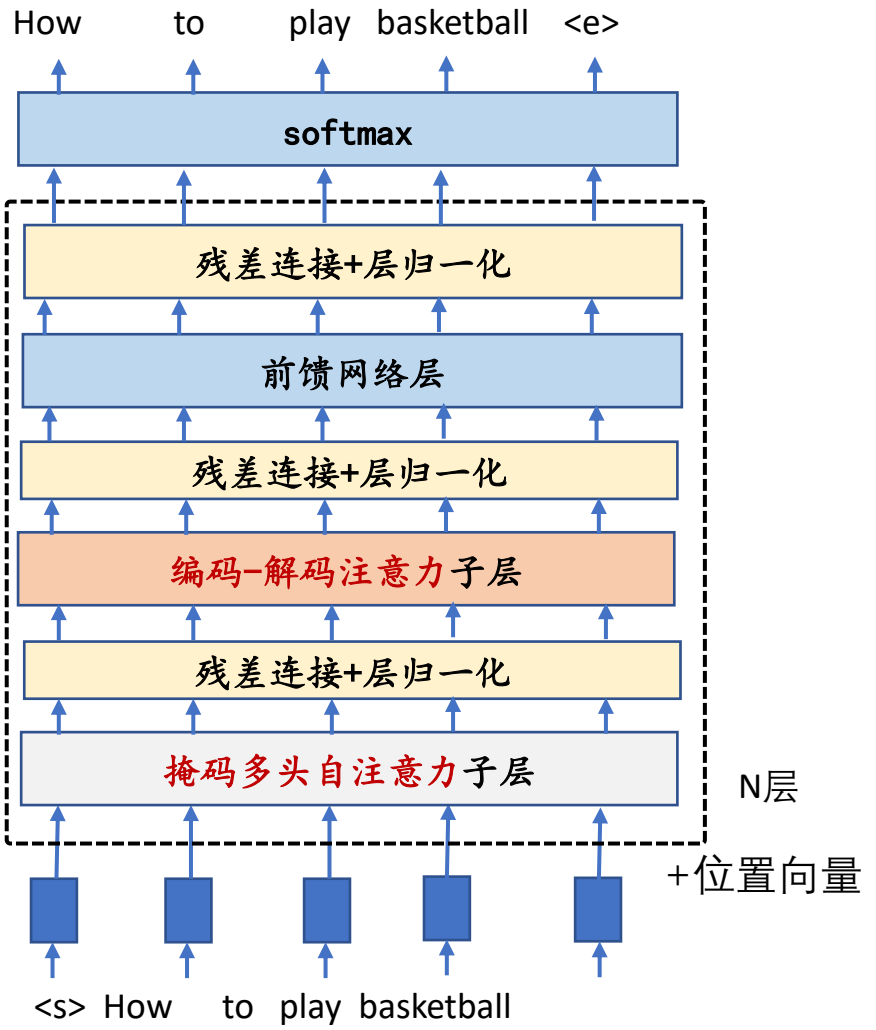
- Transformer解码器



1. 神经机器翻译

解码器

- 由N层组成，每一层包括3个子层：
  - **掩码多头自注意力子层；**
  - **编码-解码注意力子层；**
  - **前馈网络子层**
- 每个子层之间采用了**残差连接(residual connection)**和**层归一化(layer normalization)**方法。



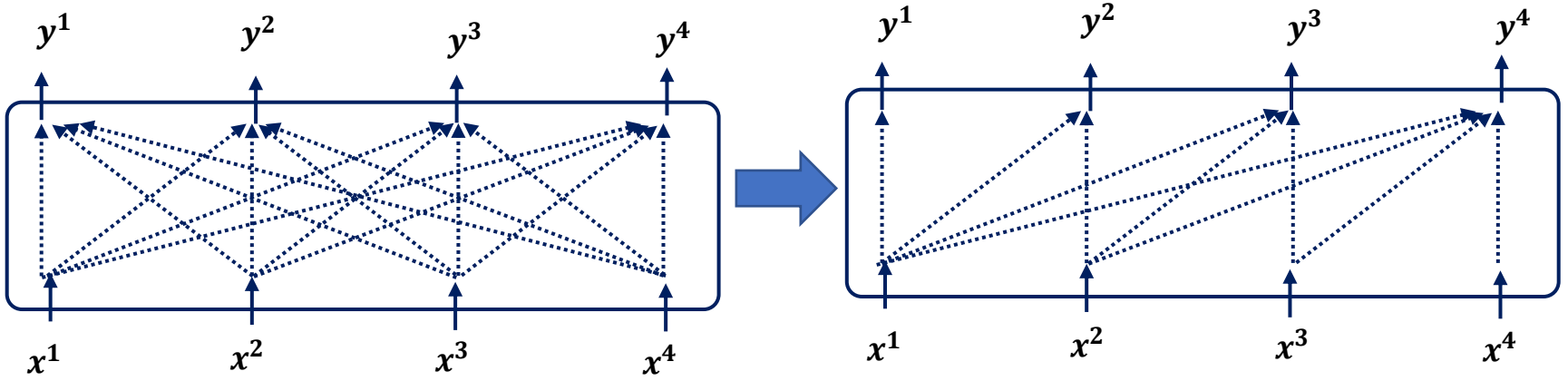
1. 神经机器翻译

掩码多头自注意力子层

- 在对于某一个目标端单词而言，只与之前单词做注意力计算，即把后面的单词掩盖。
- 为了保证模型在**训练阶段和测试阶段保持一致**，因此在测试阶段，需要将后续的单词掩盖。

训练阶段，标准译文是已知的；

测试阶段，模型仅已知之前的单词（模型自身预测得到）。



编码器自注意力　　掩码自注意力

本章内容

1. 神经机器翻译介绍
2. 实践基础
3. 实践参考
4. 本章实践

2. 实践基础

■ 实践三个主要途径

- 本次实践采用PyTorch自带的Transformer

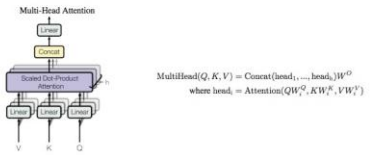
  https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html#torch.nn.Transformer

- Hugging Face的Transformers

https://github.com/huggingface/transformers/blob/main/i18n/README_zh-hans.md

- 不用自带的模型和库，可以阅读下面代码 **(强烈建议大家学习下)**:

https://github.com/hyunwoongko/transformer




```
class MultiHeadAttention(nn.Module):

    def __init__(self, d_model, n_head):
        super(MultiHeadAttention, self).__init__()
        self.n_head = n_head
        self.attention = ScaleDotProductAttention()
        self.w_q = nn.Linear(d_model, d_model)
        self.w_k = nn.Linear(d_model, d_model)
        self.w_v = nn.Linear(d_model, d_model)
        self.w_concat = nn.Linear(d_model, d_model)

    def forward(self, q, k, v, mask=None):
        # 1. dot product with weight matrices
        q, k, v = self.w_q(q), self.w_k(k), self.w_v(v)

        # 2. split tensor by number of heads
        q, k, v = self.split(q), self.split(k), self.split(v)

        # 3. do scale dot product to compute similarity
        out, attention = self.attention(q, k, v, mask=mask)

        # 4. concat and pass to linear layer
        out = self.concat(out)
        out = self.w_concat(out)
```

2. 实践基础

■ 机器翻译实践的数据

- 平行数据
(逐句对应)

英语句子

```
1 new Questions Over California Water Project
2 cri@@ tics and a state law@@ maker say they want more explan@@ ations on who@@
  back@@ ed by Go@@ v@@ . Jer@@ ry Brow@@ n, after a leading California water di
  government fun@@ ding to fin@@ ish the planning for the two gi@@ ant water tur
3 cri@@ tics said the government fun@@ ding described by the Los Angel@@ es@@ -k
  could run coun@@ ter to long-@@ standing state ass@@ ur@@ ances that various I
  pay for Brow@@ n@@ 's vision of di@@ gging tw@@ in 3@@ 5-@@ mil@@ e-@@ long tu
  River sou@@ th, mainly for Central and Southern Californi@@ a.
4 the $@@ 24@@ 8 million in pre@@ li@@ min@@ ary sp@@ ending for the tun@@ n@@ e
  is the topic of an on@@ going fed@@ eral aud@@ it.
5 on W@@ ed@@ nes@@ day, state law@@ makers ordered a state au@@ dit of the tun@
6 on Th@@ urs@@ day, state spo@@ k@@ es@@ woman N@@ ancy V@@ og@@ el said that c
  ropol@@ itan Water District, no money from the state@@ 's general fun@@ d woul
  tw@@ in tun@@ n@@ els.
```

汉语句子

```
1 加利福尼亚州@@ 水@@ 务@@ 工程@@ 的新@@ 问题
2 在@@ 加利福尼亚州@@ 一个@@ 主要@@ 水@@ 务@@ 管理@@ 区@@ 披露@@ 州@@
  政府@@ 资金@@ 以@@ 完成@@ 两@@ 条@@ 巨@@ 型@@ 输@@ 水@@ 隧@@ 道的@@
  表示@@ ，@@ 他们@@ 想@@ 进一步@@ 了解@@ 由@@ 谁@@ 来@@ 为@@ 州@@ 长@
  承担@@ 费用。
3 评论@@ 家@@ 表示@@ ，@@ 洛杉矶@@ 的@@ M@@ W@@ D@@ 周@@ 四@@ 所@@ 提及
  来的@@ 承诺@@ ，@@ 该@@ 承诺@@ 说@@ ，@@ 为了@@ 实现@@ 布@@ 朗@@ 所@
  以@@ 便于@@ 把@@ 水@@ 从@@ 萨@@ 克拉@@ 门@@ 托@@ 河@@ 向@@ 南@@ 输送
  地区的@@ 愿@@ 景@@ ，@@ 各@@ 地方@@ 水@@ 管@@ 区@@ （@@ 而非@@ 加利福
4 这两@@ 条@@ 隧@@ 道的@@ 2.@@ 48@@ 亿美元@@ 初@@ 期@@ 费用@@ 支@@ 出@
  进行的@@ 联邦@@ 审计@@ 的@@ 主题@@ 。
5 一些@@ 州@@ 议@@ 员@@ 也@@ 于@@ 周@@ 三@@ 责@@ 令@@ 对@@ 隧@@ 道@@ 费
6 该@@ 州的@@ 发言@@ 人@@ 南@@ 希@@ ·@@ 沃@@ 格尔@@ 周@@ 四@@ 表示@@ ，
  却@@ 不会@@ 动@@ 用@@ 该@@ 州@@ 的任何@@ 普通@@ 资金@@ 来@@ 完成@@ 这
```

2. 实践基础

- 位置编码（自定义层）

$$y_{2i} = \sin\left(\frac{pos}{10000^{(2i/d)}}\right) \qquad y_{2i+1} = \cos\left(\frac{pos}{10000^{(2i/d)}}\right)$$

继承nn.Module

初始化

[max_len, 1] 的张量

[max_len, 1, d_model] 的张量和赋值

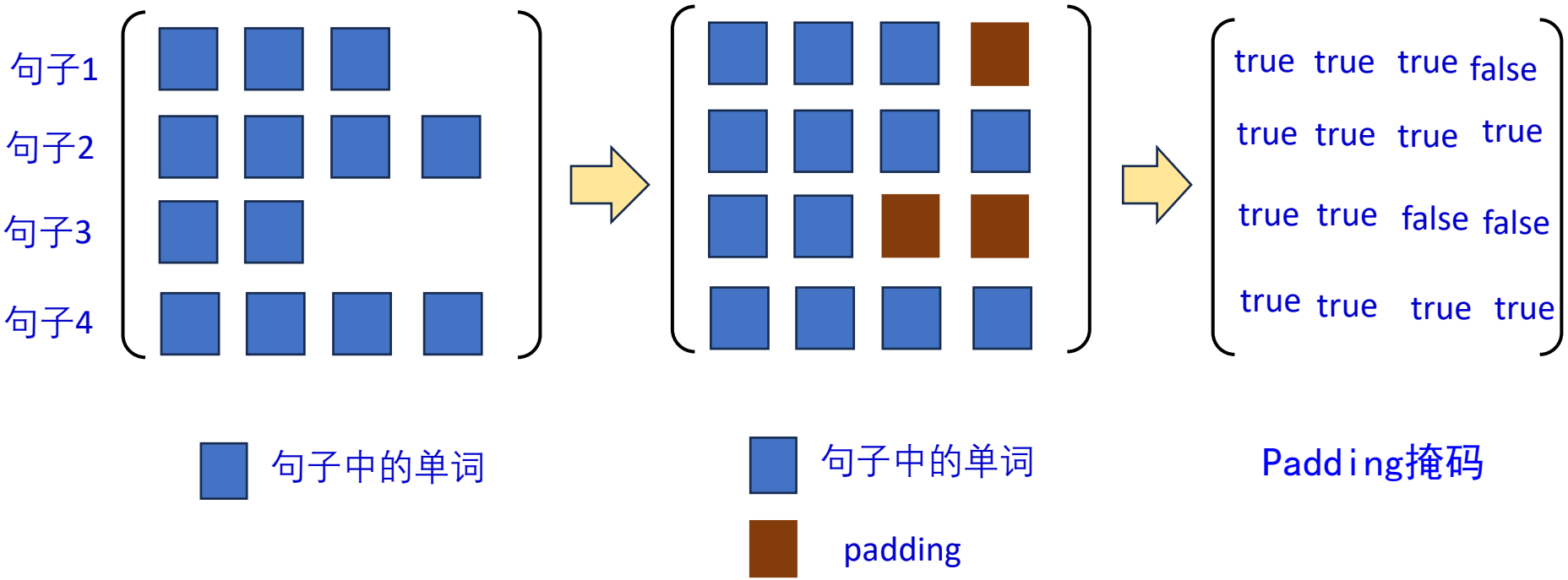
自定义前向计算

```
class PositionalEncoding(nn.Module):

    def __init__(self, d_model: int, dropout: float = 0.1, max_len: int = 5000):
        super().__init__()
        self.dropout = nn.Dropout(p=dropout)
        position = torch.arange(max_len).unsqueeze(1)
        div_term = torch.exp(torch.arange(0, d_model, 2) * (-math.log(10000.0) / d_model))
        pe = torch.zeros(max_len, 1, d_model)
        pe[:, 0, 0::2] = torch.sin(position * div_term)
        pe[:, 0, 1::2] = torch.cos(position * div_term)
        self.register_buffer('pe', pe)

    def forward(self, x):
        """
        Arguments:
            x: Tensor, shape ``[seq_len, batch_size, embedding_dim]``
        """
        x = x + self.pe[:x.size(0)]
        return self.dropout(x)
```

2. 实践基础

■ 批处理的Padding掩码

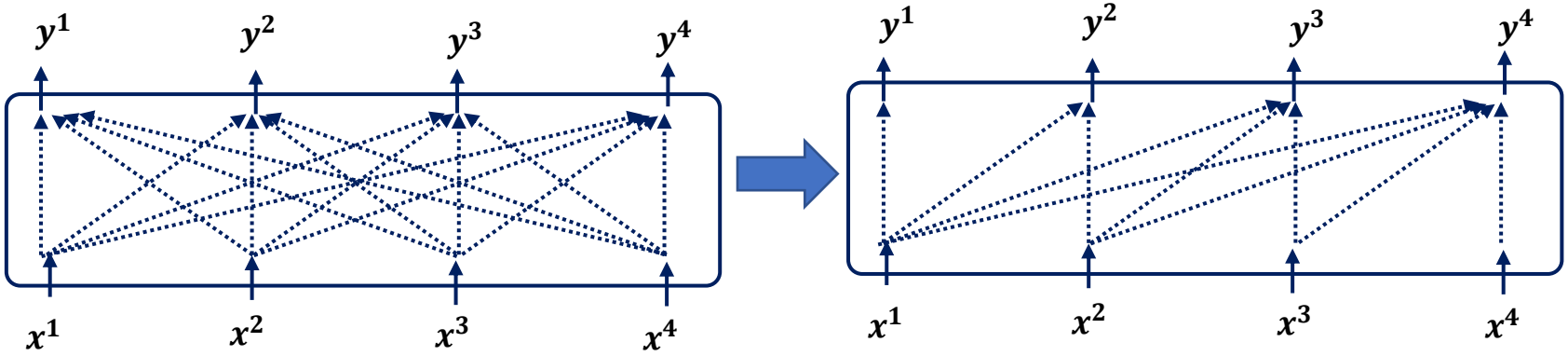
- 同一批次里，句子长度不同，需要额外增加padding；
- Padding掩码能够指示对应位置是否是padding。



2. Transformer语言模型实践

■ 因果掩码

- 因果掩码能够指示模型是否能够注意未来信息。



编码器自注意力

掩码自注意力

src_mask

```
tensor([[0,  0,  0,  0],
        [0,  0,  0,  0],
        [0,  0,  0,  0],
        [0,  0,  0,  0]])
```

tgt_mask

```
tensor([[0.0,  -inf,  -inf,  -inf],
        [0.0,  0.0,  -inf,  -inf],
        [0.0,  0.0,  0.0,  -inf],
        [0.0,  0.0,  0.0,  0.0]])
```

causal_mask

2. Transformer语言模型实践

- PyTorch集成的Transformer

forward(*src*, *tgt*, *src_mask=None*, *tgt_mask=None*, *memory_mask=None*, *src_key_padding_mask=None*, *tgt_key_padding_mask=None*, *memory_key_padding_mask=None*, *src_is_causal=None*, *tgt_is_causal=None*, *memory_is_causal=False*)

Parameters

- **src** (*Tensor*) – the sequence to the encoder (required).
- **tgt** (*Tensor*) – the sequence to the decoder (required).
- **src_mask** (*Optional*[*Tensor*]) – the additive mask for the src sequence (optional).
- **tgt_mask** (*Optional*[*Tensor*]) – the additive mask for the tgt sequence (optional).
- **memory_mask** (*Optional*[*Tensor*]) – the additive mask for the encoder output (optional).
- **src_key_padding_mask** (*Optional*[*Tensor*]) – the Tensor mask for src keys per batch (optional).
- **tgt_key_padding_mask** (*Optional*[*Tensor*]) – the Tensor mask for tgt keys per batch (optional).
- **memory_key_padding_mask** (*Optional*[*Tensor*]) – the Tensor mask for memory keys per batch (optional).

因果掩码

Padding掩码

2. Transformer语言模型实践

- PyTorch新的版本

无需额外构建因果矩阵

```
output = model(src, tgt, src_mask=None, tgt_mask= None, src_key_padding_mask=src_mask,
tgt_key_padding_mask=tgt_mask, src_is_causal=False, tgt_is_causal=True)
```

只需设置是否因果即可

本章内容

1. 神经机器翻译介绍
2. 实践基础
3. 实践参考
4. 本章实践

3. 实践参考

- 导入的库

```
import math
import torch
from torch import nn
from torch.nn import Transformer
from torch.utils.data import Dataset, DataLoader
import json
import argparse
from collections import namedtuple
```

3. 实践参考

- Main函数（1/3）

解析超参数

层数和维度不需要设置太多

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument("--src_file", type=str, default="./cleaned_translation/train.en")
    parser.add_argument("--tgt_file", type=str, default="./cleaned_translation/train.zh")
    parser.add_argument("--src_eval_file", type=str, default="./cleaned_translation/val.en")
    parser.add_argument("--tgt_eval_file", type=str, default="./cleaned_translation/val.zh")
    parser.add_argument("--src_vocab_file", type=str,
default="./cleaned_translation/train.en.json")
    parser.add_argument("--tgt_vocab_file", type=str,
default="./cleaned_translation/train.zh.json")
    parser.add_argument("--batch_size", type=int, default=16)
    parser.add_argument("--epochs", type=int, default=10)
    parser.add_argument("--lr", type=float, default=1e-4)
    parser.add_argument("--check_interval", type=int, default=100)
    parser.add_argument("--device", type=str, default="cpu"
    parser.add_argument("--max_len", type=int, default=300)
    parser.add_argument('--d_model', type=int, default=512)
    parser.add_argument('--nhead', type=int, default=8)
    parser.add_argument('--num_encoder_layers', type=int, default=6)
    parser.add_argument('--num_decoder_layers', type=int, default=6)
    parser.add_argument('--dim_feedforward', type=int, default=512)
    parser.add_argument('--dropout', type=float, default=0.1)
```

3. 实践参考

- Main函数（2/3）

首先构建Config 字典然后转为具名元组

```
    args = parser.parse_args()
    config = {
        "src_vocab_size": len(json.load(open(args.src_vocab_file, "r", encoding="utf-8"))),
        "tgt_vocab_size": len(json.load(open(args.tgt_vocab_file, "r", encoding="utf-8"))),
        "max_len": args.max_len,
        "d_model": args.d_model,
        "nhead": args.nhead,
        "num_encoder_layers": args.num_encoder_layers,
        "num_decoder_layers": args.num_decoder_layers,
        "dim_feedforward": args.dim_feedforward,
        "dropout": args.dropout,
    }
    config = namedtuple("Config", config.keys())(*config.values())
```

实例化模型

```
    model = TransformerModel(config)
    model.to(args.device)
```

Dataset和dataloader数据批处理

```
    train_dataset = TranslateDataset(args.src_file, args.tgt_file, args.src_vocab_file, args.tgt_vocab_file)
    train_dataloader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True, collate_fn=collate_fn)
    eval_dataset = TranslateDataset(args.src_eval_file, args.tgt_eval_file, args.src_vocab_file, args.tgt_vocab_file)
    eval_dataloader = DataLoader(eval_dataset, batch_size=args.batch_size, shuffle=False, collate_fn=collate_fn)
```

模型训练

```
    train(model, train_dataloader, args, eval_dataloader)
```

3. 实践参考

- Main函数（3/3）

在开发集上解码翻译结果

```
eval_dataloader = DataLoader(eval_dataset, batch_size=1,
shuffle=False, collate_fn=collate_fn)
model.load_state_dict(torch.load("model_best.pth"))
model.eval()
text2id = json.load(open(args.tgt_vocab_file, "r"))
id2text = {v: k for k, v in text2id.items()}

for step, batch in enumerate(eval_dataloader):
    src, src_mask, tgt, tgt_mask, causal_mask = batch
    src = src.to(args.device)
    src_mask = src_mask.to(args.device)
    tgt = tgt.to(args.device)
    tgt_mask = tgt_mask.to(args.device)
    causal_mask = causal_mask.to(args.device)
    output = greedy_generate(model, src, src_mask, args, 0)
    output = output.squeeze(0).tolist()
    output = [id2text[x] for x in output]
    output = " ".join(output)
    print(output)
```

自己完成

3. 实践参考

- 位置向量类

位置向量
参考之前的内容

```
class PositionalEncoding(nn.Module):
```

$$y_{2i} = \sin\left(\frac{pos}{10000^{(2i/d)}}\right)$$

$$y_{2i+1} = \cos\left(\frac{pos}{10000^{(2i/d)}}\right)$$

3. 实践参考

- TransformerModel类（1/2）

Transformer层的定义

```
class TransformerModel(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.srcEmbedding = nn.Embedding(config.src_vocab_size, config.d_model)
        self.tgtEmbedding = nn.Embedding(config.tgt_vocab_size, config.d_model)
        self.PositionalEncoding = PositionalEncoding(config.d_model, config.dropout,
config.max_len)
        self.model = Transformer(config.d_model, config.nhead,
config.num_encoder_layers, config.num_decoder_layers, config.dim_feedforward,
config.dropout, batch_first=True)
        self.fc = nn.Linear(config.d_model, config.tgt_vocab_size)
```

3. 实践参考

- TransformerModel类（2/2）

Transformer
前向传播

```
def forward(self, src, tgt, src_mask=None, tgt_mask=None,
src_key_padding_mask=None, tgt_key_padding_mask=None,
            src_is_causal=False, tgt_is_causal=False):
        src = self.srcEmbedding(src)
        src = self.PositionalEncoding(src)
        tgt = self.tgtEmbedding(tgt)
        tgt = self.PositionalEncoding(tgt)
        out = self.model(src=src, tgt=tgt, src_mask=src_mask, tgt_mask=tgt_mask,
src_key_padding_mask=src_key_padding_mask,tgt_key_padding_mask=tgt_ke
y_padding_mask,memory_key_padding_mask=src_key_padding_mask)
        out = self.fc(out)
        return out
```

3. 实践参考

- 字典类

加载字典

句子前后增加开始和结束字符

```
class Tokenizer:
    def __init__(self, vocab):
        self.vocab = json.load(open(vocab, "r", encoding="utf-8"))
        # eos 0, bos 1, unk 2, pad 3

    def encode(self, text):
        chrs = text.strip().split(' ')
        ids = [self.vocab.get(chr, 2) for chr in chrs]
        if ids[0] != 1:
            ids = [1] + ids
        if ids[-1] != 0:
            ids = ids + [0]
        return ids
```

3. 实践参考

- TranslateDataset

处理源语言和目标语言的数据，构建TranslateDataset

```
class TranslateDataset(Dataset):
    def __init__(self, src_file, tgt_file, src_tokenizer, tgt_tokenizer):
        self.src_file = src_file
        self.tgt_file = tgt_file
        self.src_tokenizer = Tokenizer(src_tokenizer)
        self.tgt_tokenizer = Tokenizer(tgt_tokenizer)
        self.src_texts = []
        self.tgt_texts = []
        self._load_data()

    def _load_data(self):
        with open(self.src_file, "r", encoding="utf-8") as f:
            self.src_texts = f.readlines()
        with open(self.tgt_file, "r", encoding="utf-8") as f:
            self.tgt_texts = f.readlines()
        assert len(self.src_texts) == len(self.tgt_texts)

    def __len__(self):
        return len(self.src_texts)

    def __getitem__(self, idx):
        src_text = self.src_texts[idx]
        tgt_text = self.tgt_texts[idx]
        src_ids = self.src_tokenizer.encode(src_text)
        tgt_ids = self.tgt_tokenizer.encode(tgt_text)
        return src_ids, tgt_ids
```

3. 实践参考

- collate_fn

```
def collate_fn(data):
    src_input = [item[0] for item in data]
    tgt_input = [item[1] for item in data]
    batch_size = len(src_input)
    src_seq_len = max([len(item) for item in src_input])
    tgt_seq_len = max([len(item) for item in tgt_input])
    src_mask = torch.ones((batch_size, src_seq_len)).long()
    tgt_mask = torch.ones((batch_size, tgt_seq_len)).long()
    src = torch.ones((batch_size, src_seq_len)).long().fill_(3)
    tgt = torch.ones((batch_size, tgt_seq_len)).long().fill_(3)
    for i in range(batch_size):
        src[i, :len(src_input[i])] = torch.tensor(src_input[i])
        tgt[i, :len(tgt_input[i])] = torch.tensor(tgt_input[i])
    src_mask = (src == 3)
    tgt_mask = (tgt == 3)
    causal_mask = make_causal_mask(tgt_mask)
    return src, src_mask, tgt, tgt_mask, causal_mask
```

与之前章节的内容相似，

- pytorch旧版本需要构建因果掩码和padding掩码
- Pytorch新版本只需padding掩码

```
def make_causal_mask(attention_mask):
    """
    0 for unmasked, -inf for masked
    """
    bsz, seq_len = attention_mask.shape
    mask = torch.ones((seq_len, seq_len)).tril()
    return mask
```

3. 实践参考

- 模型训练（1/2）

```
def train(model, dataloader, args, eval_dataloader):
    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr)
    criterion = nn.CrossEntropyLoss()
    lowest_loss = 1e12
    for epoch in range(args.epochs):
        for step, batch in enumerate(dataloader):
            src, src_mask, tgt, tgt_mask, causal_mask = batch
            src = src.to(args.device)
            src_mask = src_mask.to(args.device)
            tgt = tgt.to(args.device)
            tgt_mask = tgt_mask.to(args.device)
            causal_mask = causal_mask.to(args.device)
            output = model(src, tgt, src_mask=None, tgt_mask=causal_mask,
src_key_padding_mask=src_mask, tgt_key_padding_mask=tgt_mask,
src_is_causal=False, tgt_is_causal=True)
            output = output[:, :-1, :]
            output = output.reshape(-1, output.shape[-1])
            tgt = tgt[:, 1:]
            tgt = tgt.reshape(-1)
            loss = criterion(output, tgt)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

将因果掩码和padding掩码传给模型

3. 实践参考

- 模型训练（2/2）

验证模型性能

```
if step % args.check_interval == 0:
            print("Epoch: {}, Step: {}, Loss: {}".format(epoch, step, loss.item()))
            eval_loss = eval(model, eval_dataloader, args)
            if eval_loss < lowest_loss:
                lowest_loss = eval_loss
                torch.save(model.state_dict(), "model_best.pth")
    return
```

自己实现

3. 实践参考

- 模型验证

经过check_interval步，需要验证模型效果，计算loss，自己实现，难点从output->loss

```
def eval(model, dataloader, args):
    criterion = nn.CrossEntropyLoss(reduction='none')
    total_loss = 0
    total_num = 0
    model.eval()
    with torch.no_grad():
        for step, batch in enumerate(dataloader):
            ...
    print("Loss: {}".format(total_loss / total_num))
    model.train()
    return total_loss / total_num
```

3. 实践参考

- 贪婪解码
  - 解码阶段，采用贪婪解码，每次选择概率最大的字符，直到得到句子结束标签

选择概率最大的单词，自己实现

解码得到句子结束标签

```
def greedy_generate(model, src, src_mask, args, eos_id):
    model.eval()
    …


   if pred == eos_id:
              break
    return tgt
```

本章内容

1. 神经机器翻译介绍
2. 实践基础
3. 实践参考
4. 本章实践

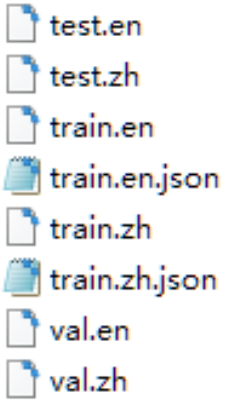
本章实践

■ 机器翻译实践

- 利用提供的翻译数据集，实现基于Transformer机器翻译方法。
- 在提供的数据集上训练实现的模型，并汇报最终的性能指标。
- 可以尝试新的方法和语言对。

不同语言对、不同机器翻译方法等

- 鼓励大家去阅读不依赖第三方库独立实现的Transformer方法，便于理解Transformer原理



本章实践

■ 基本要求：

- 序列标注模型（第6章）、语言模型（第7章）和编码解码生成模型（第8章） 至少完成一项，鼓励大模型完成更多任务，完成任务越多分数越高。
- 报告需要说明使用的模型结构和方法、超参数、实验结果等；
- 不需要把程序大段复制上去，如果有需要，只需要截取和复制关键程序即可。
- 满足了基本要求就可以得到B（80-85分），剩余的15分取决于拓展分析和实验等。
- 提交时间（12月9日）。
- 提交地址课程网站。

谢谢！

Thanks!