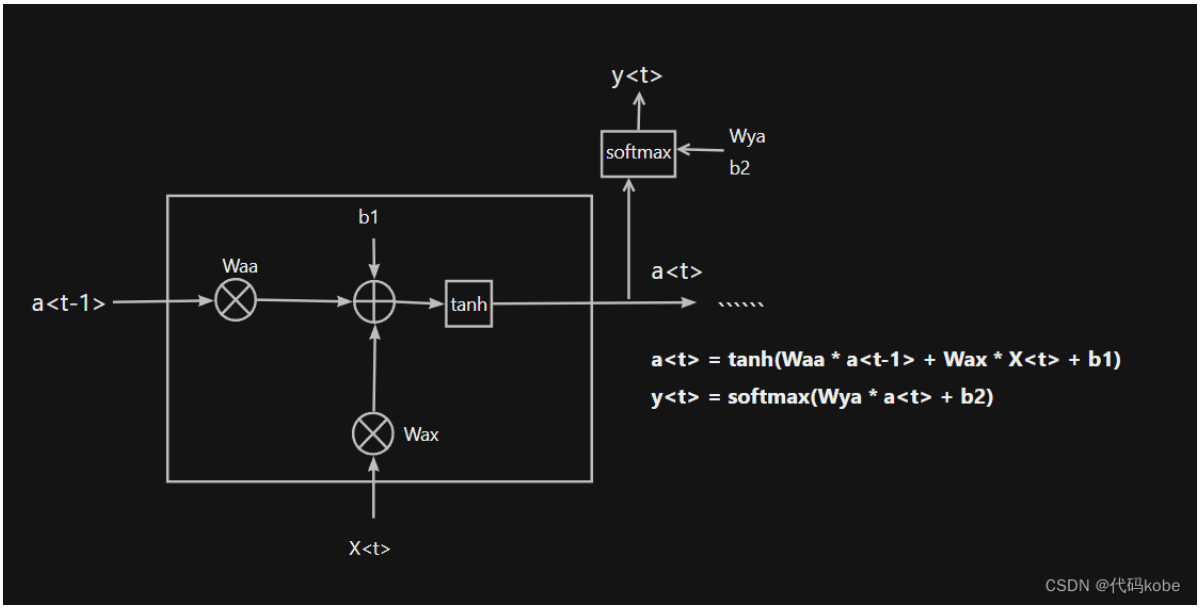
1、RNN，LSTM，GRU模型介绍

1.1 RNN（循环神经网络）

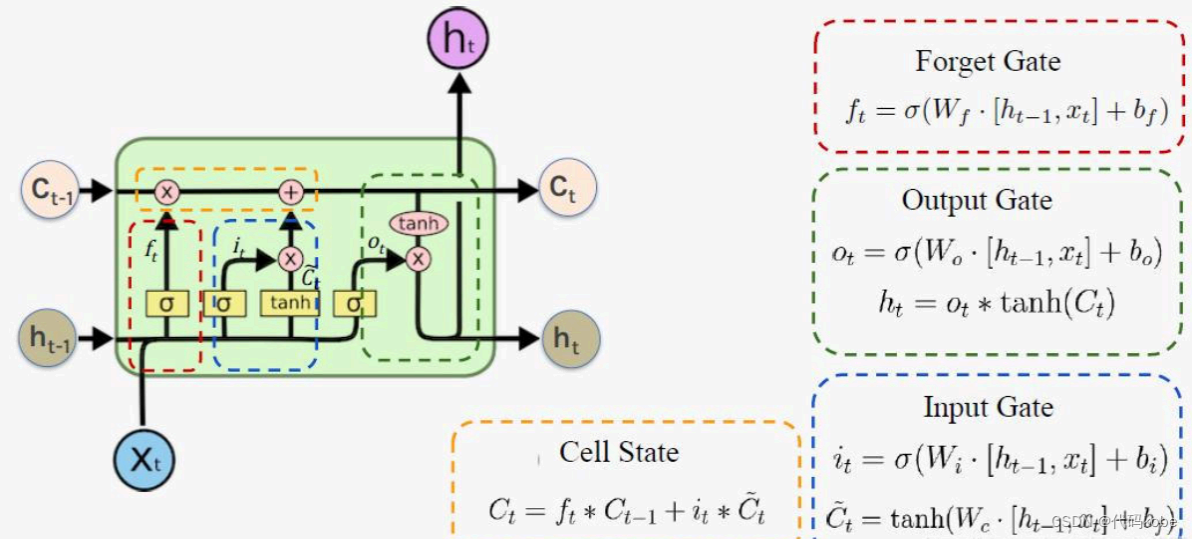
RNN是一种最基础的循环神经网络，用于处理序列数据。它通过将当前输入和前一时刻的隐藏状态（即记忆）结合起来，传递信息，从而捕捉到序列中的时序依赖性。



但是RNN在处理长序列时会遇到梯度消失和梯度爆炸的问题，导致它无法有效捕捉长期依赖关系。

1.2 LSTM（长短期记忆网络）

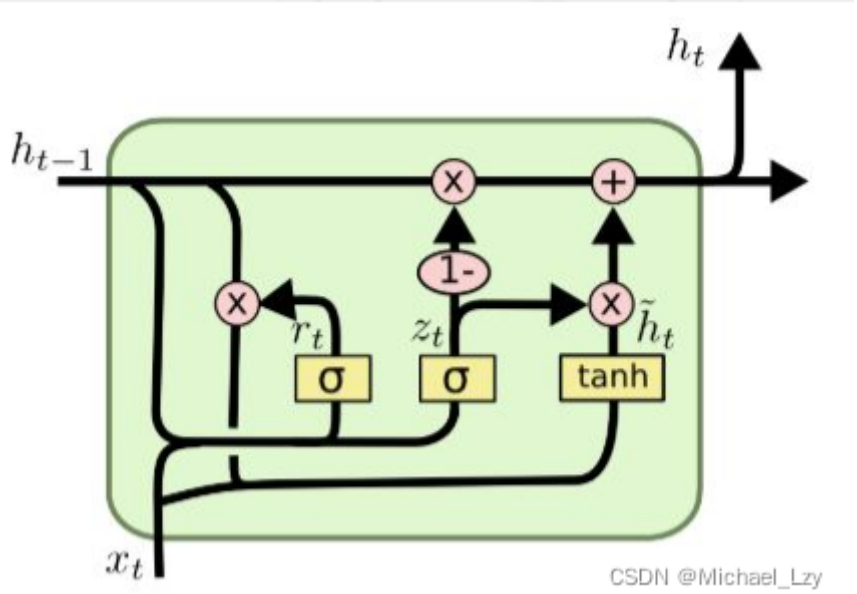
LSTM是一种特殊的RNN，它通过引入“门控”机制来解决标准RNN在长期依赖学习中的不足。LSTM网络使用了三个主要的门（输入门、遗忘门和输出门）来控制信息的流动和更新。LSTM的核心结构包括：①遗忘门决定丢弃前一时刻的多少记忆 ②输入门决定当前输入有多少信息应该被保存。 ③输出门决定下一时刻的隐藏状态（即记忆）应该传递多少。



LSTM通过这些门控机制避免了RNN的梯度消失问题，能更好地捕捉长期依赖。

1.3 GRU（门控循环单元）

GRU是LSTM的一种变体，目的是进一步简化LSTM的结构，并保持相似的性能。GRU的主要创新在于它将LSTM中的“输入门”和“遗忘门”合并为一个“更新门”，从而减少了需要学习的参数数量。GRU的结构包括：①更新门决定当前的记忆应该保持多少。 ②重置门决定丢弃多少前一时刻的记忆。



2、诗歌生成过程

tensorflow版本：

数据处理

1、读取文件

首先读取指定路径的文件 fileName。每一行的格式是以 :分割，后面的部分即为诗歌内容。

2、构建词汇表

统计每个词的出现频率，然后根据出现频率对词汇进行排序，并选取出现频率最高的词汇。

3、将文本转换为索引

诗歌中的每个字符都被映射为一个整数索引，word2id字典将每个词汇转换为对应的索引。得到的是一个由整数索引构成的诗歌列表。

4、生成 TensorFlow 数据集

通过 TensorFlow 的数据管道创建一个可用于训练的数据集，并进行必要的预处理，如批处理、填充和数据转换。

模型结构

1、嵌入层：将输入的词汇索引转换为词嵌入向量，嵌入维度为64。

2、RNN 单元：使用 SimpleRNNCell 来定义一个 RNN 层，输出的维度是 128。

3、RNN 层：利用 RNN 类封装了上面的 SimpleRNNCell，返回序列以便能传递到后续层。

4、全连接层：输出的维度是词汇表大小，用来生成每个时间步的预测。

训练过程

1、计算损失：在每个步骤中，通过模型的输出和真实标签计算损失。

2、计算梯度：使用 GradientTape记录梯度，计算损失对模型参数的梯度。

3、更新参数：使用优化器Adam更新模型参数。

4、循环训练：通过多个 epoch 和批次训练模型，不断调整参数以最小化损失。

生成语句

1、初始化当前 token：确定开始符号，将其映射到对应的词索引，并将其作为模型的第一个输入

2、循环生成 token：传入当前的 cur_token 和当前的状态 state，以获得下一个预测的 token 和更新后的状态。

3、生成词汇并返回：将 collect中的每个索引值（即生成的词汇索引）通过id2word 映射回词汇表中的单词，生成完整的句子。

pytorch版本：

数据处理

1、读取诗歌文件：从poems.txt文件中读取诗歌内容，并按行处理。

2、去除无效诗歌：过滤掉包含特殊字符的诗歌，或长度不符合要求的诗歌。

3、添加开始和结束标记：对每首诗歌，添加 start_token 和end_token，表示诗歌的开始和结束。

4、统计词频并构建词汇表：对诗歌中的所有字符进行统计，构建词到索引的映射和反向映射。

5、转换为索引形式：将每首诗中的字符转换为对应的索引，即每首诗对应的词汇索引序列。

模型结构

1、词嵌入层：将输入的词汇索引转换为词嵌入向量。

2、LSTM层：通过LSTM捕捉词汇之间的时序关系，具有两层LSTM堆叠。

3、全连接层：将LSTM输出的隐藏状态映射到词汇表的大小，用于生成词汇预测。

4、Softmax层：通过LogSoftmax计算每个词的预测概率。

训练过程

1、初始化模型：构建一个基于 LSTM 的 RNN 模型，模型的输入是词嵌入向量，输出是下一个词的预测。

2、定义优化器和损失函数：使用 RMSprop优化器和负对数似然损失函数NLLLoss进行模型训练。

3、训练模型

①在每个 epoch 中，生成批次数据并进行训练。

②对于每个批次，输入数据通过模型进行前向传播，计算损失，并反向传播更新模型参数。

③每 20 个批次保存一次模型。

生成诗歌

在训练完成后，使用训练好的模型生成诗歌：

1、加载已训练模型：加载保存的模型权重。

2、指定起始字符：根据输入的起始字符，从该字符开始生成诗歌。

3、逐步生成：

①将当前诗歌传入模型，得到下一个词的预测。

②通过模型输出的索引转换回词汇，逐步构建完整的诗歌。

③循环直到生成结束标记或生成的诗歌长度达到设定限制。

打印生成的诗歌

1、去掉开始和结束标记：只保留有效的诗歌内容。

2、按句子分割，并在每个句子后添加句号：使得打印出的诗歌更加工整。

3、诗歌生成结果

开头词汇是日 、 红 、 山 、 夜 、 湖、 海 、 月等词汇作为begin word

tensorflow版本：

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['日']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, state = model.get_next_token(cur_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

[12] ✓ 0.0s

```
暮雨，一枝不见春风。eos来不得无人事，不得人间不可知。eos有不知何处处，不知何处不知君。eos来不得无人事，
```

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['红']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, stat                                  r_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

(variable) cur_token: Any

[15] ✓ 0.0s

```
袖下红衣影。eos来不得人，不见无人事。eos有不可见，不知何所知。eos来不可见，不得不相思。eos有不可见，不知
```

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['山']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, state = model.get_next_token(cur_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

28] ✓ 0.1s

畔，春风雨，风吹雨。eos云月，一枝花。eos风吹，一片花。eos风吹，一片花。eos风吹，一枝花。eos风吹，一片花。

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['夜']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, state = model.get_next_token(cur_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

25] ✓ 0.1s

深。风吹红叶落，一片月中秋。eos水无人去，山风满水风。eos风吹落月，风雨满花声。eos落花声起，风吹落月深。

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['湖']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, state = model.get_next_token(cur_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

46] ✓ 0.0s

水，春风吹雨声。eos来无处去，何处不知君。eos马无人去，何人不可怜。eos来无限处，不见一人人。eos马无人去，

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['海']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, state = model.get_next_token(cur_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

✓ 0.0s

阳峰，一片花花满水中。eos有不知何处处，不知何处不知君。eos来不得无人事，不得人间不得人。eos有不知君不得

```
def gen_sentence():
    state = [tf.random.normal(shape=(1, 128), stddev=0.5), tf.random.normal(shape=(1, 128), stddev=0.5)]
    cur_token = tf.constant([word2id['月']], dtype=tf.int32)
    collect = []
    for _ in range(50):
        cur_token, state = model.get_next_token(cur_token, state)
        collect.append(cur_token.numpy()[0])
    return [id2word[t] for t in collect]
print(''.join(gen_sentence()))
```

✓ 0.1s

，不得无人不可知。eos有不知何处处，不知何处不知君。eos来不得无人事，不得人间不得人。eos有不知君不得，不

pytorch版本：

训练截图：

运行时间较长，因此我将这个模型放到了服务器中运行

```
********************************
epoch    29 batch number 342 loss is:  6.161645889282227
prediction [10, 167, 4, 74, 85, 0, 16, 186, 218, 25, 13, 1, 4, 19, 22, 6, 8, 0, 9, 37, 10, 48, 5, 1, 4, 119, 83, 17, 177, 0, 7, 53, 4, 15, 40, 1, 4, 74, 13, 429, 91
, 0, 7, 123, 77, 82, 132, 1, 11, 466, 10, 45, 272, 0, 4, 29, 4, 7, 178, 1, 99, 19, 84, 692, 16, 0, 9, 7, 4, 4, 5, 1, 3, 3]
b_y         [318, 318, 201, 74, 85, 0, 375, 186, 261, 60, 13, 1, 110, 1295, 115, 283, 28, 0, 622, 713, 1187, 770, 471, 1, 8, 196, 200, 54, 707, 0, 7, 344, 99, 123, 7
14, 1, 1307, 137, 277, 452, 107, 0, 371, 690, 987, 347, 76, 1, 111, 38, 105, 2450, 393, 0, 99, 609, 1182, 2989, 121, 1, 101, 324, 214, 692, 789, 0, 46, 187, 61, 252
8, 5, 1, 3, 3]
********************************
epoch    29 batch number 343 loss is:  6.219127655029297
prediction [10, 82, 179, 669, 290, 0, 22, 119, 77, 202, 340, 1, 4, 77, 12, 61, 8, 0, 6, 74, 77, 9, 13, 1, 110, 357, 7, 65, 91, 0, 46, 7, 77, 85, 13, 1, 11, 180, 138
, 54, 8, 0, 7, 85, 91, 83, 132, 1, 4, 104, 9, 250, 42, 0, 15, 11, 4, 528, 103, 1, 47, 21, 4, 15, 41, 0, 15, 179, 43, 9, 5, 1, 3, 3]
b_y         [1279, 1279, 1310, 154, 72, 0, 206, 206, 43, 293, 1497, 1, 2085, 3029, 452, 209, 8, 0, 60, 237, 92, 546, 13, 1, 1219, 983, 7, 14, 177, 0, 220, 75, 964, 1
45, 76, 1, 137, 78, 444, 309, 245, 0, 178, 1762, 392, 87, 535, 1, 678, 357, 28, 260, 114, 0, 857, 11, 99, 1110, 121, 1, 6, 1523, 2101, 4, 51, 0, 35, 1775, 644, 922,
 1979, 1, 3, 3]
********************************
epoch    29 batch number 344 loss is:  6.128315448760986
prediction [10, 7, 155, 97, 61, 0, 10, 37, 142, 46, 13, 1, 4, 17, 11, 62, 61, 0, 6, 119, 16, 17, 123, 1, 7, 32, 9, 4, 43, 0, 22, 16, 77, 144, 46, 1, 4, 48, 69, 27,
145, 0, 6, 296, 4, 43, 205, 1, 11, 52, 4, 4, 85, 0, 7, 16, 4, 21, 88, 1, 39, 17, 138, 9, 8, 0, 9, 24, 117, 39, 58, 1, 3, 3]
b_y         [196, 330, 69, 44, 70, 0, 581, 451, 18, 714, 457, 1, 8, 169, 5, 24, 73, 0, 6, 40, 100, 54, 626, 1, 185, 11, 32, 1148, 1148, 0, 249, 16, 63, 236, 236, 1,
118, 48, 168, 450, 378, 0, 6, 21, 101, 289, 615, 1, 22, 50, 259, 232, 252, 0, 33, 364, 117, 93, 594, 1, 233, 161, 88, 201, 943, 0, 86, 30, 43, 39, 632, 1, 3, 3]
********************************
epoch    29 batch number 345 loss is:  6.3069233894348145
prediction [10, 18, 74, 22, 0, 0, 13, 8, 19, 103, 13, 1, 4, 52, 19, 45, 61, 0, 22, 41, 10, 32, 20, 1, 4, 18, 6, 88, 16, 0, 5, 23, 19, 123, 12, 1, 10, 16, 4, 22, 14,
 0, 10, 56, 4, 17, 17, 1, 11, 27, 22, 17, 145, 0, 4, 43, 10, 107, 5, 1, 3, 39, 9, 155, 42, 0, 42, 0, 4, 65, 17, 96, 10, 157, 62, 1, 3, 3]
b_y         [457, 152, 475, 147, 70, 0, 223, 8, 37, 313, 1117, 1, 302, 457, 231, 652, 1824, 0, 222, 704, 70, 147, 5, 1, 457, 48, 84, 88, 16, 0, 201, 72, 1766, 372, 1
209, 1, 88, 16, 112, 594, 93, 0, 12, 57, 56, 23, 79, 1, 149, 278, 80, 309, 2542, 0, 28, 175, 254, 217, 1113, 1, 271, 488, 450, 135, 43, 15, 42, 0, 12, 65, 564, 475,
 540, 157, 29, 1, 3, 3]
```

生成结果：