

RNN 循环神经网络 1（基础模型）

Author: Zhouqi Hua, Tongji University

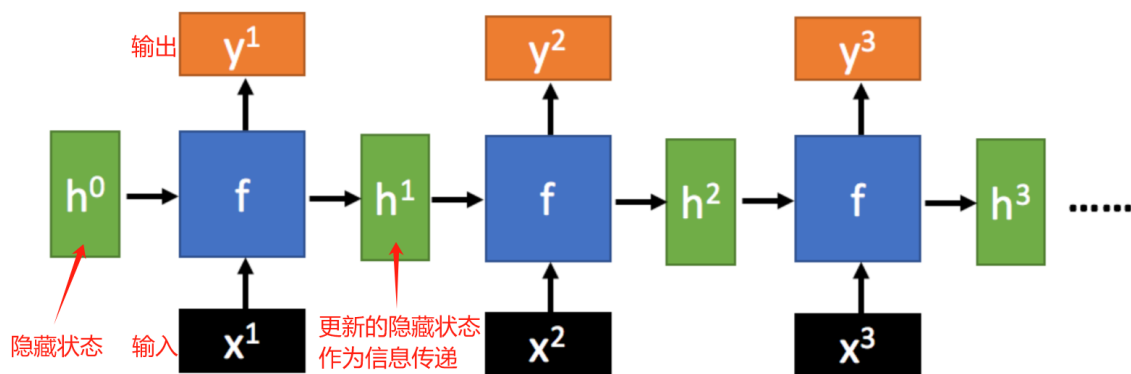
Email: henryhua0721@foxmail.com

Date: 2024/5/4

Code: `RNN_basic.py`

☀ 一句话模型总结

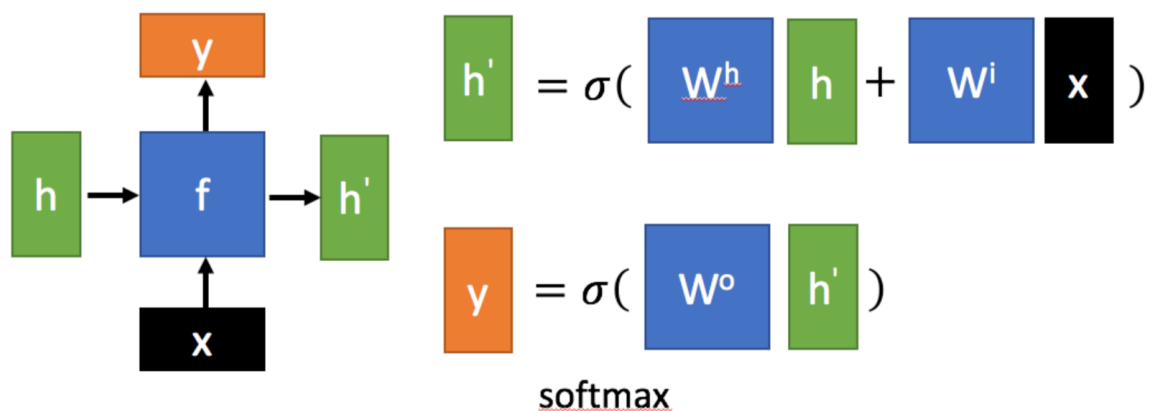
为了更好地利用**序列信息**的前后文关系，引入时间序列上实时更新的**隐藏状态**，在下一个时间步作为输入的一部分被传递，从而实现信息的传递。



☀ 模型架构解释

使用传统的 CNN 难以处理长序列的问题，而 RNN 通过其**隐藏状态**的传递可以实现前后文的**信息传递**，从而可以处理**序列变化的数据**（如某个单词的含义会根据上文内容而变化）。

具体实现如下：（此处忽略网络偏置）



- x : 当前时间步的输入
- h : 上一个时间步传递给当前的**历史状态**
- h' : 根据 x 和 h 获得的传递给下一个时间步的**当前状态**
- y : 根据 x 和 h 获得的**输出**

通常情况下：

- $h' = \sigma(W^h h + W^i x)$, 即当前状态是由历史状态和输入共同决定的
- $y = \sigma(W^o h')$, 即输出一般是当前状态的一个**维度映射**后通过softmax得到需要的数据

🌟 模型优缺点

- 优点：
 - **处理序列数据**：RNN 是一种适用于处理序列数据的神经网络，能够有效地捕捉序列中的时序信息
 - **共享权重**：RNN 在每个时间步都使用相同的参数，可以有效地共享权重，减少模型的复杂度和训练的参数数量
 - **上下文依赖建模**：RNN 能够记忆之前的信息，并在后续时间步中利用该信息进行预测或决策，对于处理依赖于上下文的任务非常有用
- 缺点：
 - **梯度消失/爆炸**：RNN 在反向传播时，由于参数共享和多次连乘的特性，容易出现梯度消失或梯度爆炸的问题，导致模型难以训练或无法收敛。
 - **长期依赖问题**：由于梯度消失的原因，RNN 在处理长序列时难以捕捉到长期依赖关系，只能有效利用较短的上下文信息。
 - **计算效率较低**：RNN 的计算过程是基于时间步展开的，每个时间步都需要依次计算，造成计算效率较低，尤其是处理较长序列时。

🌟 代码解析

(调包) 模型定义：

```
1 import torch
2 from torch import nn
3
4 class Rnn(nn.Module):
5     def __init__(self, INPUT_SIZE):
6         super(Rnn, self).__init__()
7
8         self.rnn = nn.RNN(
9             input_size=INPUT_SIZE,
10            hidden_size=32,
11            num_layers=1,
12            batch_first=True
13        )
14
15        self.out = nn.Linear(32, 1)
16
17        def forward(self, x, h_state):
18            r_out, h_state = self.rnn(x, h_state)    # 实现RNN的链式推理过程，将
19
20            outs = []
21            for time in range(r_out.size(1)):
22                outs.append(self.out(r_out[:, time, :]))
```

```
23         return torch.stack(outs, dim=1), h_state
24
```

代码 Line 18 展示了完整的 RNN 推理过程，从 torch 的源码可以看到 `torch.nn.RNN` 返回为完整的输出序列、以及重新排序的隐藏状态。

```
503
504         return output, self.permute_hidden(hidden, unsorted_indices)
505
```

其中使用 `permute_hidden` 方法：

`self.permute_hidden(hidden, unsorted_indices)` 的作用是根据 `unsorted_indices` 参数指定的索引顺序，对隐藏状态 `hidden` 进行重新排序。如果没有提供索引，那么隐藏状态将保持不变。

🎓 附录

附上 pytorch 官方文档的代码解释，有兴趣建议阅读：

Recurrent layers

`class torch.nn.RNN(args, * kwargs)[source]`

将一个多层的 Elman RNN，激活函数为 `tanh` 或者 `ReLU`，用于输入序列。

对输入序列中每个元素，RNN 每层的计算公式为

$$h_t = \tanh(w_{ih} * x_t + b_{ih} + w_{hh} * h_{t-1} + b_{hh})$$

h_t 是时刻 t 的隐状态。 x_t 是上一层时刻 t 的隐状态，或者是第一层在时刻 t 的输入。如果 `nonlinearity='relu'`，那么将使用 `relu` 代替 `tanh` 作为激活函数。

参数说明：

- `input_size` – 输入 `x` 的特征数量。
- `hidden_size` – 隐层的特征数量。
- `num_layers` – RNN 的层数。
- `nonlinearity` – 指定非线性函数使用 `tanh` 还是 `relu`。默认是 `tanh`。
- `bias` – 如果是 `False`，那么 RNN 层就不会使用偏置权重 $b_i h$ 和 $b_h h$ ，默认是 `True`。
- `batch_first` – 如果 `True` 的话，那么输入 `Tensor` 的 shape 应该是 `[batch_size, time_step, feature]`，输出也是这样。
- `dropout` – 如果值非零，那么除了最后一层外，其它层的输出都会套上一个 `dropout` 层。
- `bidirectional` – 如果 `True`，将会变成一个双向 RNN，默认为 `False`。

```
1 RNN` 的输入:  **(input, h_0)** - input (seq_len, batch, input_size): 保存输入序列特征的`tensor`。`input` 可以是被填充的变长的序列。细节请看`torch.nn.utils.rnn.pack_padded_sequence()
```

