

从头构建循环神经网络RNN的向前传播(rnn in pure python)

原创 思秀 [sigua心底的小声音](#) 2020-03-19

收录于话题

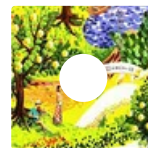
#深度学习

4个


预测点击上方蓝字，关注公众号 

明日の朝には

Humbert Humbert - 11のみじかい話



小声音

受我一个高中童鞋音乐博客的启发，我在这篇开始在里面加一些我喜欢/珍藏的音乐，看完一篇文章大概就是几分钟的时间，正好和听完音乐时间相重合，希望大家能拉到最后看完。(能留言反馈我会感激不尽的)(笔芯 )

前言

目录:

- 向量表示以及它的维度
- rnn cell
- rnn 向前传播

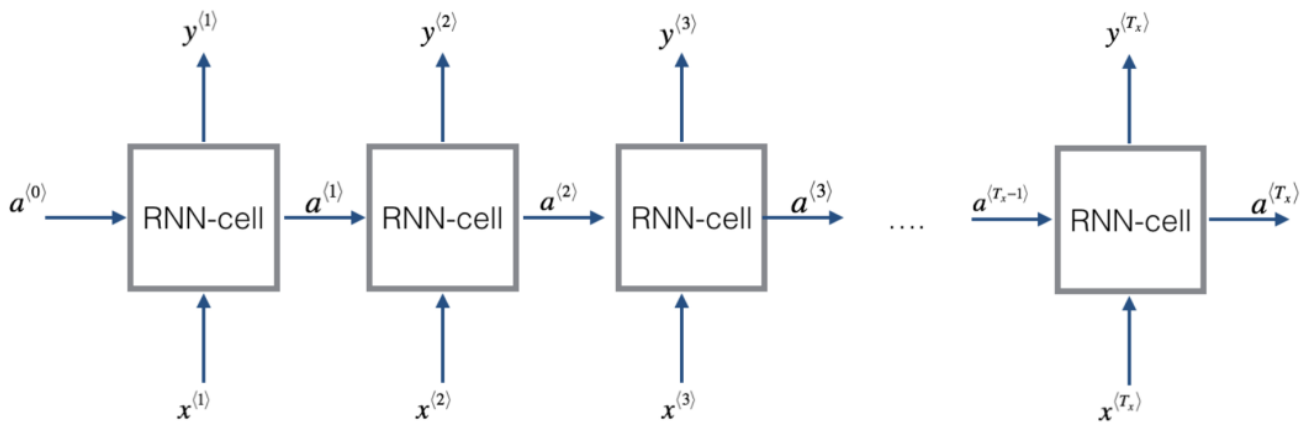
重点关注:

- 如何把数据向量化的，它们的维度是怎么来的
- 一共其实就是两步: 单个单元的rnn计算，拉通来的rnn计算

在看本文前，可以先看看这篇文章回忆一下:

[吴恩达deepLearning.ai循环神经网络RNN学习笔记\(理论篇\)](#)

我们将实现以下结构的RNN，在这个例子中 $T_x = T_y$ 。



向量表示以及它的维度

Input with n_x number of units

- 对单个输入样本, $x(i)$ 是一维输入向量。
- 用语言来举个例子, 将具有5k个单词词汇量的语言用one-hot编码成具有5k个单位的向量, 所以 $x(i)$ 的维度是 (5000,)。
- 我们将用符号 n_x 表示单个训练样本的单位数。

Batches of size m

- 如果我们取小批量(mini-batches), 每个批次有20个训练样本。
- 为了受益于向量化, 我们将20个样本 $x(i)$ 变成一个2维数组(矩阵)。
- 比如一个维度是(5000, 20)的向量。
- 我们用 m 来表示训练样本的数量。
- 所以小批量训练数据的维度是 (n_x, m) 。

Time steps of size T_x

- 循环神经网络有多个时间步骤, 我们用 t 来表示。
- 我们将看到训练样本 $x(i)$ 将经历多个时间步骤 T_x , 比如如果有10个时间步骤, 那么 $T_x = 10$ 。

3D Tensor of shape (n_x, m, T_x)

- 输入 x 就是用维度是 (n_x, m, T_x) 的三维张量来表示。

Taking a 2D slice for each time step:

- 每一个时间步骤, 我们用小批量训练样本(不是单个的训练样本)。
- 所以针对每个时间步骤 t , 我们用维度是 (n_x, m) 的2维切片。
- 我们把它表示成 x_t 。

隐藏状态 a 的维度

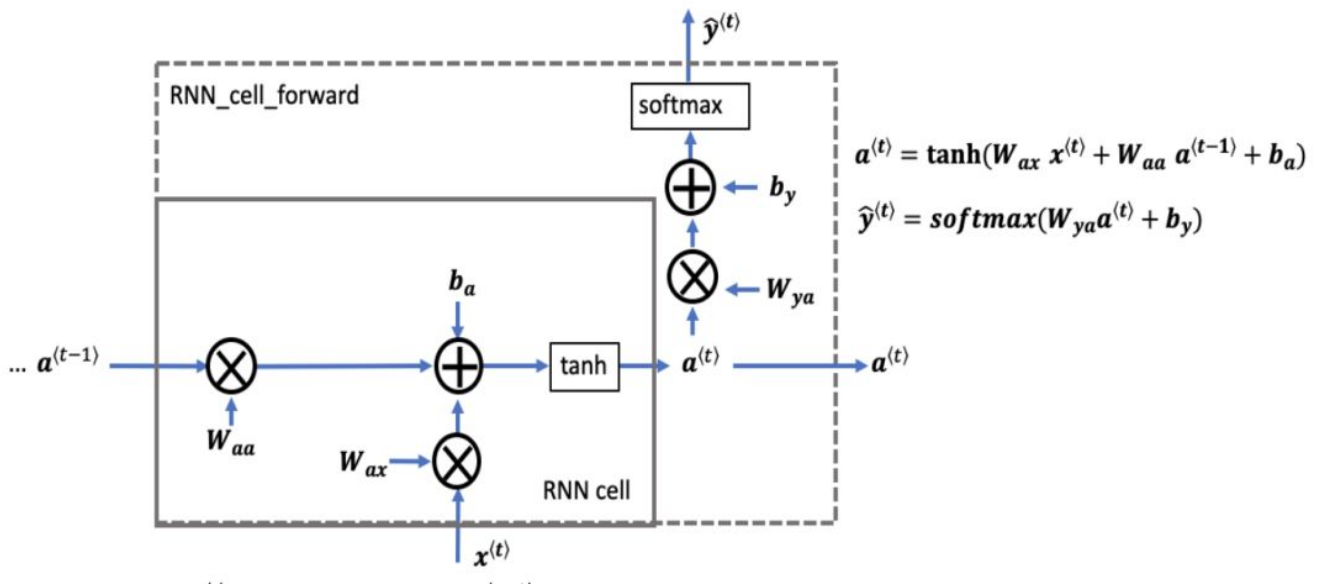
- a 的定义: 从一个时间步骤到另一个时间步骤的激活值 a_t , 我们把它叫做隐藏状态。
- 同输入张量 x 一样, 对于单个训练样本的隐藏状态, 它的向量长度是 n_a 。
- 如果我们是包含了 m 个训练样本的小批量数据, 那么小批量维度是 (n_a, m) 。
- 如果我们将时间步加进去, 那么隐藏状态的维度就是 (n_a, m, T_x) 。
- 我们将用索引 t 来遍历时间步, 每次操作是从3维张量切片成的2维向量。
- 我们用 a_t 来表示2维的切片, 它的维度是 (n_a, m) 。

预测值 y^{\wedge} 的维度

- 同输入x和隐藏状态一样, \hat{y} 是一个维度是 (n_y, m, T_y) 的3维张量。
 - n_y : 代表预测值的单位数。
 - m : 小批次训练的样本数量。
 - T_y : 预测的时间数。
- 比如单个时间步 t , 2维的切片 \hat{y}^t 的维度是 (n_y, m) 。

RNN cell

我们的第一个任务就是执行单个时间步骤的计算, 计算如下图。



输入是 $a^{(t-1)}$, $x^{(t)}$, 输出是 $a^{(t)}$, $\hat{y}^{(t)}$ 。以下的代码其实就是把上面的公式代码化, 总的步骤分成4步:

1. 取出参数。
2. 计算 $a^{(t)}$ 。
3. 计算 $\hat{y}^{(t)}$ 。
4. 返回输出的 $a^{(t)}$, $\hat{y}^{(t)}$, 还要存储一些值缓存起来。

```

1  import numpy as np
2
3  def rnn_cell_forward(xt, a_prev, parameters):
4      """
5      Implements a single forward step of the RNN-cell as described in Figure
6
7      Arguments:
8      xt -- your input data at timestep "t", numpy array of shape (n_x, m).
9      a_prev -- Hidden state at timestep "t-1", numpy array of shape (n_a, m)
10     parameters -- python dictionary containing:
11
12         Wax -- Weight matrix multiplying the input, numpy array of shape (n_a, n_x)
13         Wya -- Weight matrix relating the hidden-state to the output, numpy array of shape (n_y, n_a)
14         ba -- Bias, numpy array of shape (n_a, 1)
15         by -- Bias, numpy array of shape (n_y, 1)
16
17     Returns:
18     a -- next hidden state, numpy array of shape (n_a, m)
19     y -- next prediction, numpy array of shape (n_y, m)
20     """
21
22     # Compute the next hidden state
23     z = np.dot(Wax, xt) + np.dot(Wya, a_prev) + ba
24     a = np.tanh(z)
25
26     # Compute the next prediction
27     z = np.dot(Wya, a) + by
28     y = np.softmax(z)
29
30     return a, y

```

```

14         by -- Bias relating the hidden-state to the output,
15     Returns:
16     a_next -- next hidden state, of shape (n_a, m)
17     yt_pred -- prediction at timestep "t", numpy array of shape (n_y, m)
18     cache -- tuple of values needed for the backward pass, contains (a_next
19     """
20     # 取计算的参数
21     Wax = parameters["Wax"]
22     Waa = parameters["Waa"]
23     Wya = parameters["Wya"]
24     ba = parameters["ba"]
25     by = parameters["by"]
26
27     # 用公式计算下一个单元的激活值
28     a_next = np.tanh(np.dot(Waa, a_prev) + np.dot(Wax, xt) + ba)
29     # 计算当前cell的输出
30     yt_pred = softmax(np.dot(Wya, a_next) + by)
31
32     # 用于向后传播的缓存值
33     cache = (a_next, a_prev, xt, parameters)
34
35     return a_next, yt_pred, cache

```

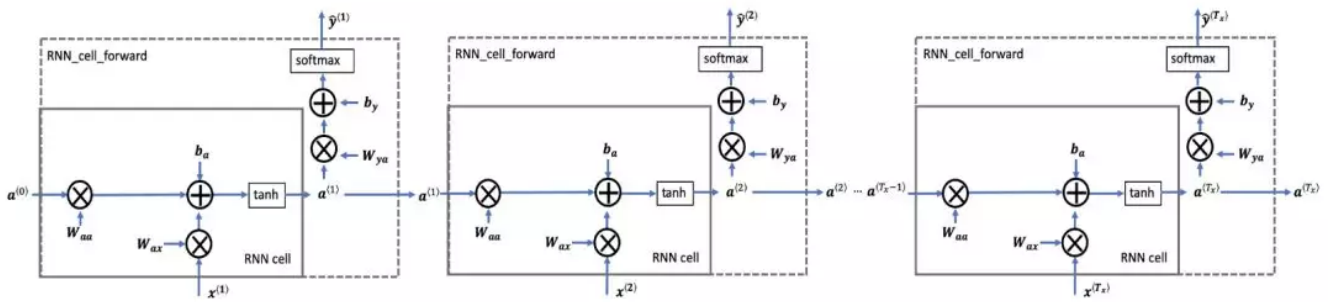
```

1 def softmax(x):
2     """Compute the softmax of vector x."""
3     exp_x = np.exp(x)
4     softmax_x = exp_x / np.sum(exp_x)
5     return softmax_x

```

RNN向前传播

- 一个循环神经网络就是不断的重复你上面创建的rnn 单元。
 - 如果你的输入数据序列是10个时间步，那么你就重复你的rnn cell 10次。
- 在每个时间步中，每个单元将用2个输入：
 - a_{t-1} : 前一个单元的隐藏状态。
 - x_t : 当前时间步的输入数据。
- 每个时间步有两个输出：
 - 一个隐藏状态 a_t
 - 一个测值 $y^{(t)}$
- 权重和偏差 (Waa, ba, Wax, bx) 将在每个时间步中循环使用，它们保存在"parameters"的变量中。



```

1 def rnn_forward(x, a0, parameters):
2     """
3     Implement the forward propagation of the recurrent neural network described above.
4
5     Arguments:
6     x -- Input data for every time-step, of shape (n_x, m, T_x).
7     a0 -- Initial hidden state, of shape (n_a, m)
8     parameters -- python dictionary containing:
9
10         Waa -- Weight matrix multiplying the hidden state, numpy array of shape (n_a, n_a)
11         Wax -- Weight matrix multiplying the input, numpy array of shape (n_a, n_x)
12         Wya -- Weight matrix relating the hidden-state to the output, numpy array of shape (n_y, n_a)
13         ba -- Bias numpy array of shape (n_a, 1)
14         by -- Bias relating the hidden-state to the output, numpy array of shape (n_y, 1)
15
16     Returns:
17     a -- Hidden states for every time-step, numpy array of shape (n_a, m, T_x)
18     y_pred -- Predictions for every time-step, numpy array of shape (n_y, m, T_x)
19     caches -- tuple of values needed for the backward pass, contains (list of gradients, list of caches)
20
21     # 用于存储所有cache的列表，初始化它
22     caches = []
23
24     # 取一些维度值，用于后面初始化变量
25     n_x, m, T_x = x.shape
26     n_y, n_a = parameters["Wya"].shape
27
28
29     # 初始化 a 和 y_pred
30     a = np.zeros((n_a, m, T_x))
31     y_pred = np.zeros((n_y, m, T_x))

```

```
32
33     # 初始化 a_next
34     a_next = a0
35
36     # loop over all time-steps of the input 'x'
37     for t in range(T_x):
38         # Update next hidden state, compute the prediction, get the cache
39         xt = x[:, :, t] # 通过切片的方式从输入变量x中取出当前t时间步的输入xt
40         a_next, yt_pred, cache = rnn_cell_forward(xt, a_next, parameters)
41         # 保存当前单元计算的a_next值
42
43         a[:, :, t] = a_next
44         # 保存当前单元的预测值y
45
46         y_pred[:, :, t] = yt_pred
47         # 添加每个单元的缓存值
48         caches.append(cache)
49
50
51     # store values needed for backward propagation in cache
52     caches = (caches, x)
53
54     return a, y_pred, caches
```

恭喜你(*^▽^*)🎉🎉，到这里你已经能够从0到1的构建循环神经网络的向前传播过程。

在现代深度学习框架中，您仅需实现前向传递，而框架将处理后向传递，因此大多数深度学习工程师无需理会后向传递的细节。我就不写向后传播了。

原创文章，欢迎转载，转载请在公众号菜单栏查看【联系我】。