深度学习第37讲：循环神经网络 RNN 入门

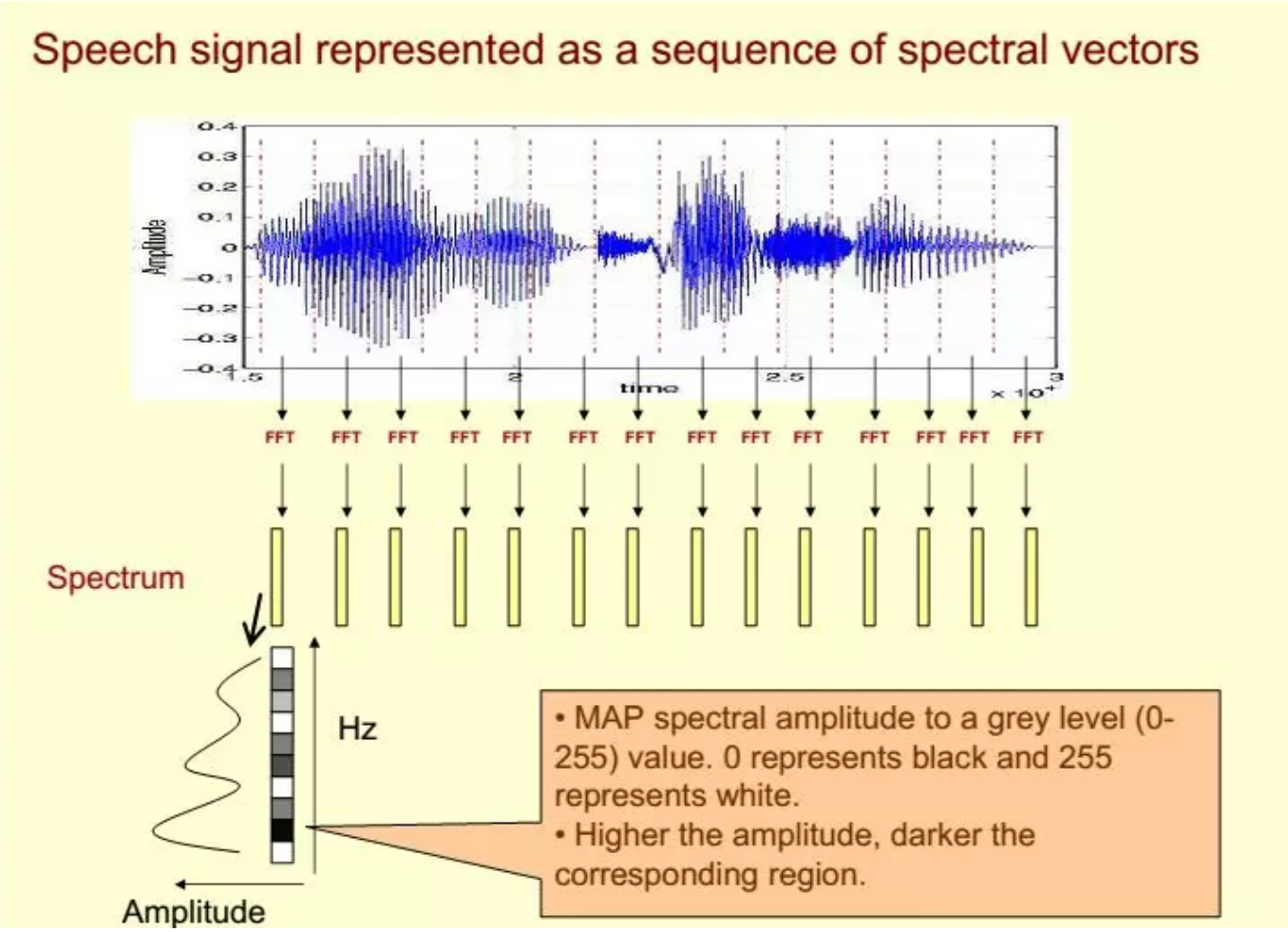
原创 louwill 机器学习实验室 2018-09-30

在前面内容中，笔者和大家一起学习了深度神经网络 DNN 和卷积神经网络 CNN，其中我们在 CNN 上花了大量时间和精力对其基本原理和应用以及在计算机视觉领域的应用进行了详细的介绍。从本节开始，笔者将和大家一起学习一种新的神经网络结构——循环神经网络（Recurrent Neural Networks）。

相较于 CNN 在图像识别和检测方面的广泛应用，基于序列模型的 RNN 的应用方面则是语音识别、文本翻译和自然语言处理等其他更为激动人心的领域。所以，正如 CNN 在计算机视觉中的应用一样，在 RNN 中笔者将重点关注其在自然语言处理的应用与研究。

在前面内容中，笔者和大家一起学习了深度神经网络 DNN 和卷积神经网络 CNN，其中我们在 CNN 上花了大量时间和精力对其基本原理和应用以及在计算机视觉领域的应用进行了详细的介绍。从本节开始，笔者将和大家一起学习一种新的神经网络结构——循环神经网络（Recurrent Neural Networks）。

相较于 CNN 在图像识别和检测方面的广泛应用，基于序列模型的 RNN 的应用方面则是语音识别、文本翻译和自然语言处理等其他更为激动人心的领域。所以，正如 CNN 在计算机视觉中的应用一样，在 RNN 中笔者将重点关注其在自然语言处理的应用与研究。



RNN 使用场景

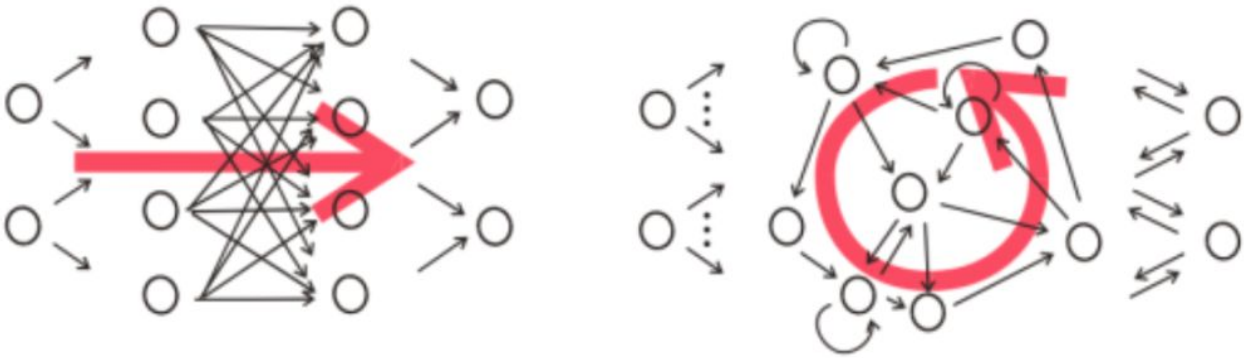
相较于 DNN 和 CNN，RNN 网络结构有什么特别之处？它与前两者又有哪些不一样的结构设计？在对 RNN 的结构进行深入了解之前，我们必须对使用 RNN 面临的问题场景进行梳理。假设我们在进行语音识别时，给定了一个输入音频片段 x，要求我们输出一个文本片段 y，其中输入 x 是一个按照时间播放的音频片段，y 是一个按照顺序排列的单词组成的一句话，所以在 RNN 中我们的输入输出都是序列性质的。针对这样的输入输出（x,y）的有监督学习，最适合的神经网络结构就是循环神经网络。为什么循环神经网络就最适用这种场景？在正式介绍 RNN 前，我们先来看下对于序列问题使用常规的神经网络看看会有什么问题。

假设我们现在需要对输入的一段话识别其中每个单词是否是人名，即输入是一段文本序列，输出是一个每个单词是否是人名的序列。假设这段话有9个单词，我们将其转化为 9 个 one-hot 向量输入到标准神经网络中去，经过一些隐藏层和激活函数得到最终 9 个值为 0/1 的输出。但这样做的问题有两个：

一是输入输出的长度是否相等以及输入大小不固定的问题。在语音识别问题中，输入音频序列和输出文本序列很少情况下是长度相等的，普通网络难以处理这种问题。

二是普通神经网络结构不能共享从文本不同位置上学到的特征，简单来说就是如果神经网络已经从位置 1 学到了 louwill 是一个人名，那么如果 louwill 出现在其他位置，神经网络就可以自动识别到它就是已经学习过的人名，这种共享可以减少训练参数和提高网络效率，普通网络不能达到这样的目的。

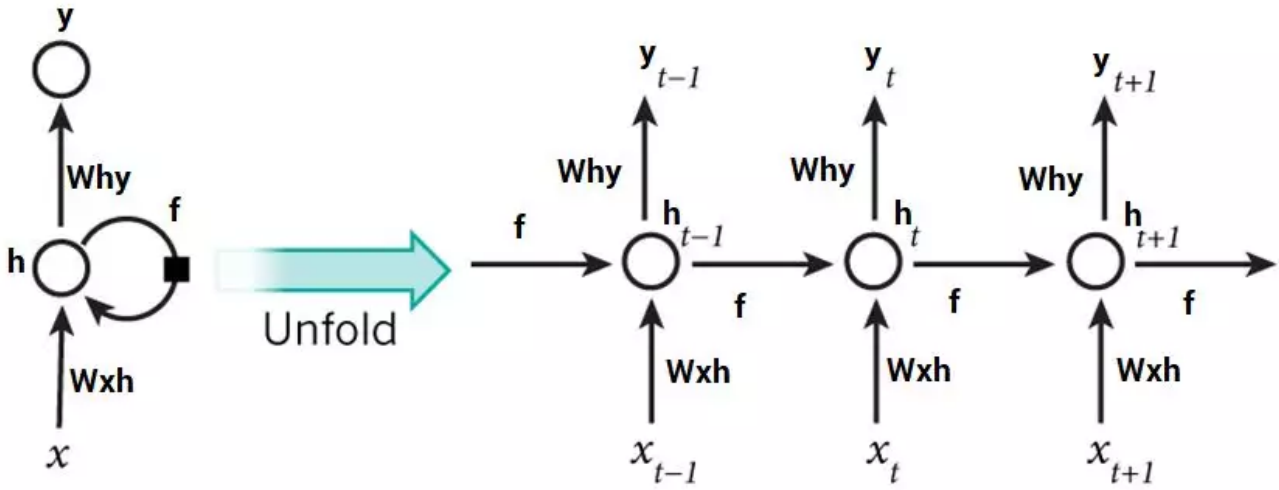
所以直观上看，普通神经网络和循环神经网络的区别如下图所示：



那么 RNN 到底长什么样子呢？

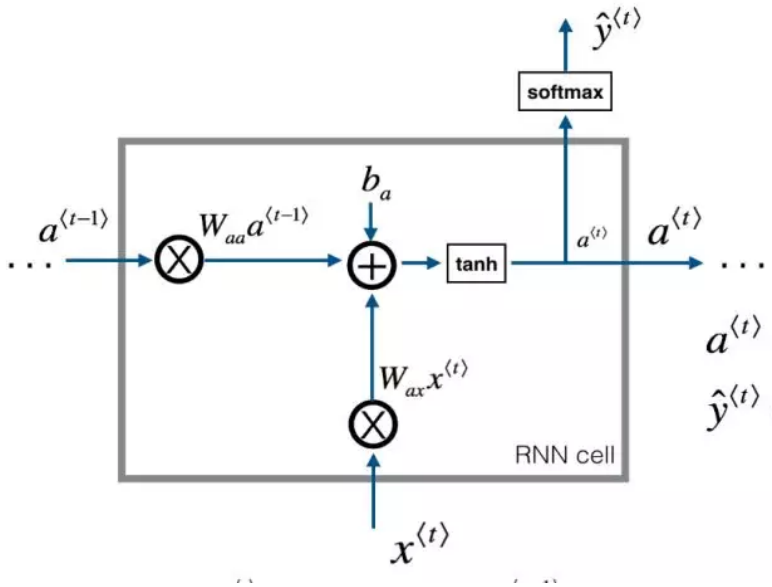
RNN 结构

假设我们将一个句子输入 RNN，第一个输入的单词就是 x1，我们将 x1 输入到神经网络，经过隐藏层得到输出判断其是否为人名，即输出为 y1。同时网络初始化隐藏层激活值，并在隐藏层中结合输入 x1 进行激活计算传入到下一个时间步。当输入第二个单词 x2 的时候，除了使用 x2 预测输出 y2 之外，当前时间步的激活函数会基于上一个时间步的进行激活计算，即第二个时间步利用了第一个时间步的信息。这便是循环（Recurrent）的含义。如此下去，一直到网络在最后一个时间步输出 yn 和 激活值 an。所以在每一个时间步中，RNN 传递一个激活值到下一个时间步中用于计算。



上图便是循环神经网络的基本结构。左边是一个统一的表现形式，右边则是左边的展开图解。在这样的循环神经网络中，当我们在预测 yt 时，不仅要使用 xt 的信息，还要使用 xt-1 的信息，因为在横轴路径上的隐藏层激活信息得以帮助我们预测 yt。

所以， RNN 单元结构通常需要两次计算，一次是隐藏层隐变量激活函数的计算，一个是结合隐变量和输入的计算。一个 RNN 单元和两次计算如下图所示：

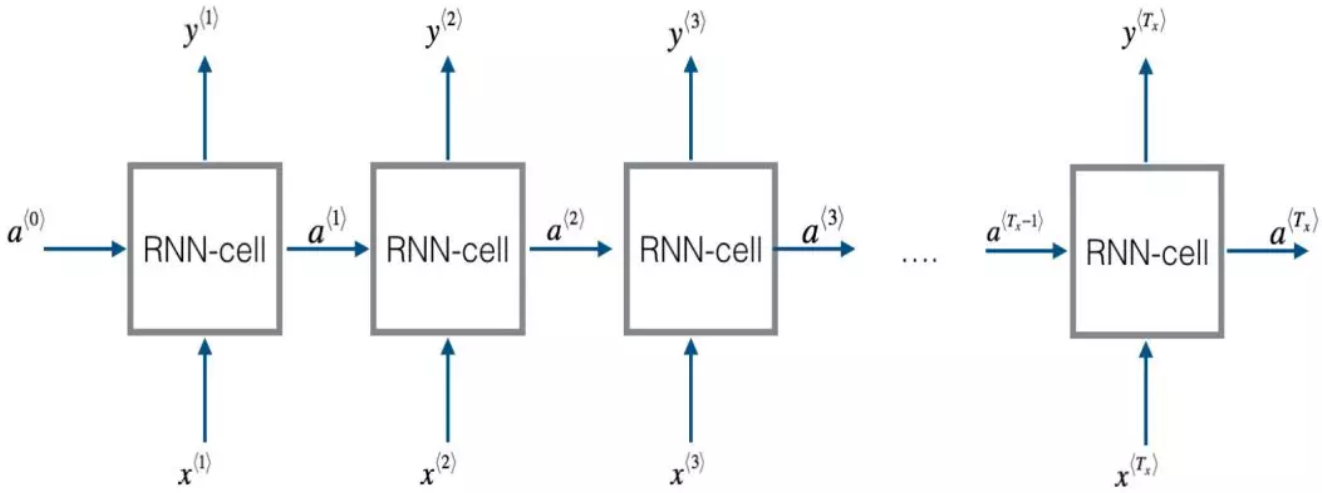


$$a^{\langle t \rangle} = \tanh(W_{ax}x^{\langle t \rangle} + W_{aa}a^{\langle t-1 \rangle} + b_a)$$

$$\hat{y}^{\langle t \rangle} = soft\max(W_{ya}a^{\langle t \rangle} + b_y)$$

其中隐藏层的激活函数一般采用 `tanh` ，而输入输出的激活函数一半使用 `sigmoid` 或者 `softmax` 函数。

当多个 RNN 单元组合到一起便是 RNN 结构：



RNN 结构的 numpy 实现

定义 `sigmoid` 和 `softmax` 函数：

```
import numpy as np

def softmax(x):
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum(axis=0)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

定义 RNN 单元结构：

```
def rnn_cell_forward(xt, a_prev, parameters):
    """
    Arguments:
    xt -- your input data at timestep "t", numpy array of shape (n_x, m).
    a_prev -- Hidden state at timestep "t-1", numpy array of shape (n_a, m)
    parameters -- python dictionary containing:
                        Wax -- Weight matrix multiplying the input, numpy array of
                        Waa -- Weight matrix multiplying the hidden state, numpy a
                        Wya -- Weight matrix relating the hidden-state to the outp
                        ba --  Bias, numpy array of shape (n_a, 1)
                        by -- Bias relating the hidden-state to the output, numpy
    Returns:
    a_next -- next hidden state, of shape (n_a, m)
    yt_pred -- prediction at timestep "t", numpy array of shape (n_y, m)
    cache -- tuple of values needed for the backward pass, contains (a_next, a_pre
    """
```

```
    # Retrieve parameters from "parameters"
    Wax = parameters["Wax"]
    Waa = parameters["Waa"]
    Wya = parameters["Wya"]
    ba = parameters["ba"]
    by = parameters["by"]
    # compute next activation state using the formula given above
    a_next = np.tanh(np.matmul(Wax, xt) + np.matmul(Waa, a_prev) + ba)
    # compute output of the current cell using the formula given above
    yt_pred = softmax(np.matmul(Wya, a_next) + by)

    # store values you need for backward propagation in cache
    cache = (a_next, a_prev, xt, parameters)
    return a_next, yt_pred, cache
```

计算示例：

```
np.random.seed(1)
xt = np.random.randn(3,10)
a_prev = np.random.randn(5,10)
Waa = np.random.randn(5,5)
Wax = np.random.randn(5,3)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
parameters = {"Waa": Waa, "Wax": Wax, "Wya": Wya, "ba": ba, "by": by}

a_next, yt_pred, cache = rnn_cell_forward(xt, a_prev, parameters)
print("a_next[4] = ", a_next[4])
print("a_next.shape = ", a_next.shape)
print("yt_pred[1] =", yt_pred[1])
print("yt_pred.shape = ", yt_pred.shape)
```

```
a_next[4] =  [ 0.59584544  0.18141802  0.61311866  0.99808218  0.85016201  0.99980978
 -0.18887155  0.99815551  0.6531151   0.82872037]
a_next.shape =  (5, 10)
yt_pred[1] = [0.9888161  0.01682021 0.21140899 0.36817467 0.98988387 0.88945212
 0.36920224 0.9966312  0.9982559  0.17746526]
yt_pred.shape =  (2, 10)
```

基于 RNN 单元构建 RNN 网络结构：

```
def rnn_forward(x, a0, parameters):
    """
    Arguments:
    x -- Input data for every time-step, of shape (n_x, m, T_x).
    a0 -- Initial hidden state, of shape (n_a, m)
    parameters -- python dictionary containing:
                        Waa -- Weight matrix multiplying the hidden state, numpy a
                        Wax -- Weight matrix multiplying the input, numpy array of
                        Wya -- Weight matrix relating the hidden-state to the outp
```

```
                    ba --  Bias numpy array of shape (n_a, 1)
                    by -- Bias relating the hidden-state to the output, numpy

    Returns:
    a -- Hidden states for every time-step, numpy array of shape (n_a, m, T_x)
    y_pred -- Predictions for every time-step, numpy array of shape (n_y, m, T_x)
    caches -- tuple of values needed for the backward pass, contains (list of cach
    """

    # Initialize "caches" which will contain the list of all caches
    caches = []
    # Retrieve dimensions from shapes of x and parameters["Wya"]
    n_x, m, T_x = x.shape
    n_y, n_a = parameters["Wya"].shape
    # initialize "a" and "y" with zeros (≈2 lines)
    a = np.zeros((n_a, m, T_x))
    y_pred = np.zeros((n_y, m, T_x))
    # Initialize a_next (≈1 line)
    a_next = a0
    # loop over all time-steps
    for t in range(T_x):
        # Update next hidden state, compute the prediction, get the cache
        a_next, yt_pred, cache = rnn_cell_forward(x[:,:,t], a_next, parameters)
        # Save the value of the new "next" hidden state in a
        a[:,:,t] = a_next
        # Save the value of the prediction in y
        y_pred[:,:,t] =
        # Append "cache" to "caches"
        caches.append(cache)

    # store values needed for backward propagation in cache
    caches = (caches, x)
    return a, y_pred, caches
```

计算示例：

```
np.random.seed(1)
x = np.random.randn(3,10,4)
a0 = np.random.randn(5,10)
Waa = np.random.randn(5,5)
Wax = np.random.randn(5,3)
Wya = np.random.randn(2,5)
ba = np.random.randn(5,1)
by = np.random.randn(2,1)
parameters = {"Waa": Waa, "Wax": Wax, "Wya": Wya, "ba": ba, "by": by}

a, y_pred, caches = rnn_forward(x, a0, parameters)
print("a[4][1] = ", a[4][1])
print("a.shape = ", a.shape)
print("y_pred[1][3] =", y_pred[1][3])
print("y_pred.shape = ", y_pred.shape)
print("caches[1][1][3] =", caches[1][1][3])
print("len(caches) = ", len(caches))
```

```
a[4][1] =  [-0.99999375  0.77911235 -0.99861469 -0.99833267]
a.shape =  (5, 10, 4)
y_pred[1][3] = [0.79560373 0.86224861 0.11118257 0.81515947]
y_pred.shape =  (2, 10, 4)
caches[1][1][3] = [-1.1425182  -0.34934272 -0.20889423  0.58662319]
len(caches) =  2
```

这样一个简单的 RNN 结构就搭建起来了。至于 RNN 的反向传播和更为复杂的结构模式我们将在下一讲继续探讨学习。

参考资料：

deeplearningai.com

https://zhuanlan.zhihu.com/p/22930328

往期精彩：

深度学习第36讲：图像实例分割经典论文研读之 Mask R-CNN

深度学习第35讲：图像语义分割经典论文研读之 u-net

深度学习第34讲：图像语义分割经典论文研读之 FCN 全卷积网络

深度学习第33讲：CNN图像语义分割和实例分割综述