Learning RNN

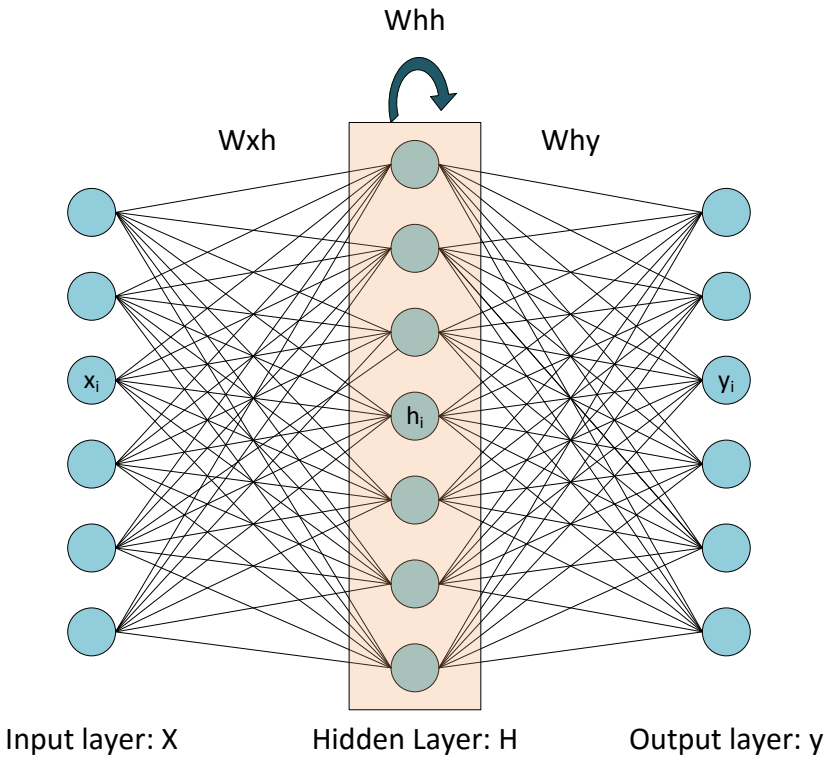
1.Introduction

In this paper I will introduce RNN technology and explain RNN and explain how to do back propagation to compute the weight of RNN. This tutorial will use character prediction as example to show how to use RNN to predict next character based on current input and hidden state.

Exmaple code: https://github.com/weixsong/min-char-rnn

2.Simple RNN Theory

RNN 的图片表示如下：



基本的 RNN 的 python 代码如下：

```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

其中，t 时刻的 hidden state 的计算公式为： $h^t=\tanh(Whh*h^{t-1} + Wxh*x^t)$, x^t 表示 t 时刻的 RNN 输入。

特别注意：这里的 RNN 的 hidden layer 采用了 tanh 作为激活函数（Activation function），Output layer 则没有采用任何激活函数，即为线性的 output.

3. Using RNN to predict next character

我们利用 RNN 来进行下一个 character 的预测，利用已知的字母序列来预测下一个可能出现的字母，首先我们有以下若干 RNN 使用的假设，用来初始化我们的 RNN 模型：

1. 假设我们一共有 V 个 character, 即字典的大小为 V，那么 RNN 的输入为 one hot vector, dim=V
2. 假设我们当前的 RNN 的 hidden layer 的 size 为 H,则我们的 RNN 的权重矩阵的维度分别为：
    a. Wxh: H*V, input to hidden layer weight
    b. Whh: H*H, hidden to hidden layer weight
    c. Why: V*H, hidden to output layer weight
3. Output layer 的维度为 V，输出为每个字母可能出现的概率，output layer 做了一个 softmax 操作，获得每个字母可能出现的概率分布，在预测下一个字母的时候就可以在这个得到的字母分布的基础上进行采样。
4. 设 Hidden layer 的输入为 u,输出为 h, output layer 的输入为 u', 这里，因为 hidden layer 采用 tanh 作为激活函数，所以 h = tanh(u), output layer 没有采用激活函数，所以 y = u' .

接下来我们讲的所有的操作，forward propagation & back propagation 都是以本节的设定为基础的。

4. Loss Function

在 NN 中常用的两种 Loss function 有：Sum of squared error (Quadratic error) & cross entropy error。

假设 t 为训练样本的真实值，y 为神经网络的输出，我们的训练样本为(x, t)，一个样本。我们下面的公式也是针对一个样本而言，对于所有的样本也就很简单了。

4.1 Sum of Squared error (Quadratic error)

$$E = \frac{1}{2}(t - y)^2 \tag{1}$$

当神经网络的 output layer 没有采用激活函数的时候，我们应该采用 Quadratic error，这样能够比较快速的进行梯度下降参数估计。

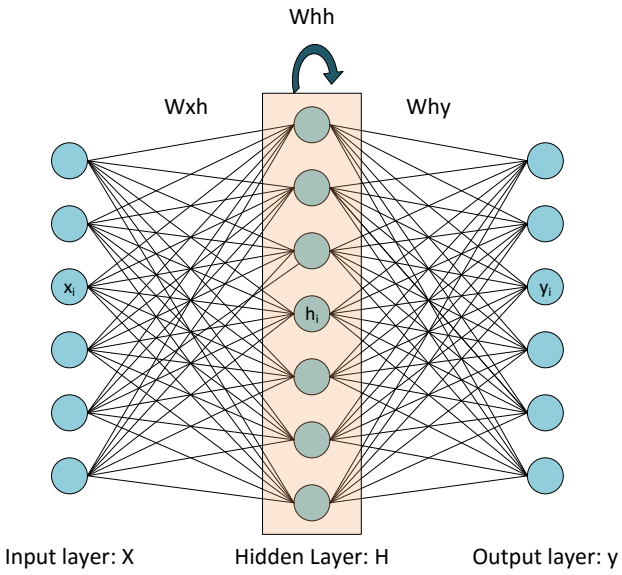
4.2 Cross Entropy Error

$$E(t, y) = -[t \ln(y)] + (1 - t) \ln(1 - y)] \tag{2}$$

如果当 output layer 采用 sigmoid 激活函数的时候，我们应该采用 cross entropy error 进行参数估计，这是因为如果采用 sigmoid 激活函数，采用 cross entropy error 能够在求导的时候消掉 sigmoid 函数，这样能够加快梯度下降的速度。接下来的参数估计推导部分会有详细的推导说明。如果 output layer 没有采用 sigmoid 函数，但是却使用 cross entropy error 来进行参数估计，那么就会得到不是很舒服的偏导数公式，可能对梯度下降的速度有所影响。

5. Forward Propagation

离得太远了，再把图拿过来看一下



Forward propagation 的计算过程如下：

(1) u = Wxh * x (u 为 hidden layer 的输入) (3)

(2) $h^t = \tanh(Whh*h^{t-1} + u)$ (h 为 hidden layer 的输出) (4)

$= \tanh(z^t + u)$ ($z^t = Whh*h^{t-1}$)

(3) u' = Why*h (u'为 output layer 的输入) (5)

(4) y = u' (y 为 output layer 的输出, output layer 为 linear layer, 所以 y = u') (6)

上面的计算公式中，具体的每个神经元的计算如下：

(1) $u_i = \sum_{j=1}^{V} Wxh(i,j) * x_j$ (7)

(2) $z_i^t = \sum_{j=1}^{H} Whh(i,j) * h_j^{t-1}$ (8)

(3) $h_i = \tanh(u_i + z_i^t)$ (9)

(4) $u_i' = \sum_{j=1}^{H} Why(i,j) * h_j$ (10)

(5) $y_i = u_i'$ (11)

6. Quadratic Error VS Cross Entropy Error

In this chapter we will introduce how to select cost function and why we select cross entropy error as cost function when using sigmoid activation function in output layer.

6.1 Derivative of error with regard to the output of output layer

All the computation is regard to only one training sample, not all training sample.

(A) use Quadratic error:

Quadratic error function: $E = \frac{1}{2}(t-y)^2$

$$\frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(t-y)^2 = y - t \tag{12}$$

(B) use cross entropy error

Cross entropy error: $E(t,y) = -[t\ln(y)] + (1-t)\ln(1-y)]$

$$\frac{\partial E}{\partial y} = \frac{\partial}{\partial y}(-[t\ln(y)] + (1-t)\ln(1-y)])$$
$$= \frac{-t}{y} + \frac{1-t}{1-y}$$
$$= \frac{-t(1-y) + y(1-t)}{y(1-y)}$$
$$= \frac{y-t}{y(1-y)} \tag{13}$$

因为 output layer 没有采用激活函数，所以 the derivative of error with regard to the input of output layer is the same as $\frac{\partial E}{\partial y}$:

$$\frac{\partial E}{\partial u'} = \frac{\partial E}{\partial y} = \frac{y-t}{y(1-y)} \tag{14}$$

通过这个公式，可以看出代码（https://github.com/weixsong/min-char-rnn）中作者虽然采用了 cross entropy error 来评估当前的模型的误差，但是并没有用 cross entropy error 进行梯度下降计算更新参数。关于代码的解释请看代码中的详细注释。同时，这也符合 NN 中当 output layer 没有激活函数的时候采用 Quadratic error 的 trick.

6.2 Derivative of error with regard to the input of output layer

(A) Linear output layer

如果 output layer 没有激活函数，那么就如上面得到的一样：$\frac{\partial E}{\partial u'} = \frac{\partial E}{\partial y}$，那针对不同的 cost function 就分别对应不同的公式，公式 13 或者公式 14.

(B) Sigmoid output layer

假如我们的 output layer 采用 sigmoid 激活函数，那么我们来证明一下为什么使用 cross entropy error 能够是参数估计收敛的更快。

Let compute the derivative of error with regard to the input of output layer, let assume that the input of output layer is z, here z = u', then y = sigmoid(z)

In NN, the layer error of output layer is the same as the derivative of error with regard to the input of output layer. And we could represent the layer error by δ^L

Firstly, remember that the derivative of sigmoid function is:

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x)) \tag{15}$$

Now, let's compute the derivative of error with regard to the input of output layer:

$$\begin{aligned}
\delta^L &= \frac{\partial E}{\partial z} = \frac{\partial E}{\partial y} * \frac{\partial y}{\partial z} \\
&= \frac{y - t}{y(1-y)} \frac{\partial y}{\partial z} \\
&= \frac{y - t}{y(1-y)} \sigma(z)\big(1 - \sigma(z)\big) \\
&= \frac{y - t}{y(1-y)} y(1-y) \\
&= y - t
\end{aligned} \tag{16}$$

哈哈，怎么样，看到公式最后把 sigmoid 函数消掉了，这里要特别注意 $y = \sigma(z)$, 所以如果 output layer 采用 sigmoid 激活函数的时候，应该采用 cross entropy error 进行梯度下降参数更新，能够避免 sigmoid 函数最后迭代速度减慢的问题。

7. Error Measure of RNN for Character Prediction

In the code (https://github.com/weixsong/min-char-rnn), the user use cross entropy error to measure the RNN model quality, only to measure the quality of RNN, remember that for gradient descent user use Quadratic error.

对应一个样本，建模误差为：$loss = -\sum_{i=1}^{V} t_i \ln y_i$

$t = [0,0,....1,0,....0]^T$, 为训练样本对应的真实值，维度是 V，是 ont hot vector

$y = [0.1,0.02,....0.05]^T$，为 NN 的输出值，维度为 V，为每个字母出现的 概率。

因为$t = [0,0,....1,0,....0]^T$只有一个索引值为 1，所以上面的 loss 公式也可以表示为（i 为不为 0 的元素的索引）：

$$loss = -t_i \ln y_i \tag{17}$$

在作者的代码中，就是按照上面的公式计算 RNN 的误差的。

8. Back Propagation

** 特别注意：在进行参数估计的时候我们采用 Quadratic Error $E = \frac{1}{2}(t-y)^2$ 来衡量当前的系统误差并进行梯度下降计算。

8.1 compute the derivative of error with regard to the output of output layer

$$\frac{\partial E}{\partial y_i} = \frac{\partial}{\partial y_i}\frac{1}{2}(t_i - y_i)^2 = y_i - t_i \tag{18}$$

矩阵化得到：

$$\frac{\partial E}{\partial y} = \frac{\partial}{\partial y}\frac{1}{2}(t-y)^2 = y - t \tag{19}$$

8.2 compute the derivative of error with regard to the input of output layer

因为 output layer 没有采用任何激活函数，为 linear output，所以$u_i' = y_i$, 那么偏导数也是一样的，如下：

$$\frac{\partial E}{\partial u_i'} = \frac{\partial E}{\partial y_i} = y_i - t_i$$

(20)

矩阵化得到：

$$\frac{\partial E}{\partial u'} = \frac{\partial E}{\partial y} = y - t$$

(21)

8.3 compute the derivative of error with regard to the weight between hidden layer and output layer

We already know: $u_i' = \sum_{j=1}^{H} Why(i,j) * h_j$

$$\begin{aligned}\frac{\partial E}{\partial Why(i,j)} &= \frac{\partial E}{\partial u_i'} * \frac{\partial u_i'}{\partial Why(i,j)} \\ &= (y_i - t_i) * \frac{\partial}{\partial Why(i,j)} \left(\sum_{j=1}^{H} Why(i,j) * h_j \right) \\ &= (y_i - t_i) * h_j\end{aligned}$$

(22)

这里，公式中的$\frac{\partial E}{\partial u_i'}$我们在公式 20 中就已经计算得到了。矩阵化上面的公式得到：

$$\frac{\partial E}{\partial Why} = \frac{\partial E}{\partial y} * H^T$$

(23)

8.4 compute the derivative of error with regard to the output of hidden layer

Be careful about the derivative of error with regard to the output of hidden layer, 因为 h 收到两个公式影响，所以计算关于 h 的偏导数的时候，需要将这两个公式都计算关于 h 的偏导数。

(1) $h_i^{t+1} = \tanh\left(u_i^t + z_i^t\right),\ z_i^t = \sum_{j=1}^{H} Whh(i,j) * h_j^{t-1}, u_i = \sum_{j=1}^{V} Wxh(i,j) * x_j$
(2) $u_i' = \sum_{j=1}^{H} Whh(i,j) * h_j$

上面的公式（1）如果是考虑 tanh 括号中的项，还可以写成：$u_i^{t+1} = u_i^t + z_i^t$，矩阵化表示就是：$u^{t+1} = Whh * h^t + Whx * x$

因为在 RNN 存在 hidden layer 到 hidden layer 的计算，在计算 h^{t+1} 的时候用到了 h^t, 所以在计算

的偏导数时要加上来自下一个时刻 t+1 时针对 u 的误差的偏导数，即将 t+1 时刻的 hidden layer 的输入的误差反向传播到 t 时刻的输出误差。也可以理解为依赖于 h^t 的变量有 u^{t+1}。

我们将下一时刻的 hidden layer 的 input 的关于误差的偏导数称为 dnext (代码中也是这样定义的)，然后计算下面的关于误差的针对 output of hidden layer 的偏导数。

(A)

$$\begin{aligned}\frac{\partial E}{\partial h_i} &= \sum_{k=1}^{V} \frac{\partial E}{\partial u'_k} * \frac{\partial u'_k}{\partial h_i} \\ &= \sum_{k=1}^{V} \frac{\partial E}{\partial y_k} * \frac{\partial u'_k}{\partial h_i} \\ &= \sum_{k=1}^{V} (y_k - t_k) * \frac{\partial u'_k}{\partial h_i} \\ &= \sum_{k=1}^{V} (y_k - t_k) * Why(k,i)\end{aligned} \tag{24}$$

这里$u'_i = \sum_{j=1}^{H} Whh(i,j) * h_j$，并且$\frac{\partial E}{\partial y_k}$已经计算得到（公式 18）

(B)

$$\begin{aligned}dhnext_i = \frac{\partial E}{\partial h_i^t} &= \sum_{j=1}^{H} \frac{\partial E}{\partial u_j^{t+1}} \frac{\partial u_j^{t+1}}{\partial h_i^t} \\ &= \sum_{j=1}^{H} \frac{\partial E}{\partial u_j^{t+1}} \frac{\partial}{\partial h_i^t} \left(\sum_{i=1}^{H} Whh(j,i) * h_i^t + \sum_{i=1}^{V} Wxh(j,i) * x_i \right) \\ &= \sum_{j=1}^{H} \frac{\partial E}{\partial u_j^{t+1}} \left(\frac{\partial}{\partial h_i^t} \sum_{i=1}^{H} Whh(j,i) * h_i^t + \frac{\partial}{\partial h_i^t} \sum_{i=1}^{V} Wxh(j,i) * x_i \right) \\ &= \sum_{j=1}^{H} \frac{\partial E}{\partial u_j^{t+1}} \left(\frac{\partial}{\partial h_i^t} \sum_{i=1}^{H} Whh(j,i) * h_i^t + 0 \right) \\ &= \sum_{j=1}^{H} \frac{\partial E}{\partial u_j^{t+1}} (Whh(j,i) + 0) \\ &= \sum_{j=1}^{H} \frac{\partial E}{\partial u_j^{t+1}} Whh(j,i)\end{aligned} \tag{25}$$

矩阵化得到：

$$dhnext = Whh^T \frac{\partial E}{\partial u}$$

(26)

注意上面公式中的$\frac{\partial E}{\partial u_j^{t+1}}$我们暂时还没有计算得到，在下一步计算中会计算$\frac{\partial E}{\partial u_j^{t+1}}$的值。

矩阵化(A)和(B)得到：

$$\frac{\partial E}{\partial h} = Why^T * \frac{\partial E}{\partial u'} + dhnext$$
$$\frac{\partial E}{\partial h} = Why^T * \frac{\partial E}{\partial y} + dhnext$$

(27)

8.5 Compute the derivative of error with regard to the input of hidden layer

在代码中，hidden layer 的 input 针对 error 的偏导数称为 hraw

$$hraw_i = \frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial h_i}\frac{\partial h_i}{\partial u_i}$$
$$= \frac{\partial E}{\partial h_i}\frac{\partial}{\partial u_i}(\tanh(u_i))$$
$$= \frac{\partial E}{\partial h_i}(1 - (\tanh(u_i)^2))$$

(28)

其中$\frac{\partial E}{\partial h_i}$已经在公式 27 中计算得到，并且$\frac{\partial h_i}{\partial u_i} = 1 - (\tanh(u_i)^2$, 因为 hidden layer 采用了 tanh 作为激活函数，$h_i = \tanh(u_i)$， tanh(x)的倒数为 $1-\tanh(x)^2$。

矩阵化得到：

$$hraw = \frac{\partial E}{\partial u} = (1 - h \odot h)\frac{\partial E}{\partial h}$$

here ⊙ means elements products.

8.6 compute the derivative of error with regard to the weight between input layer and hidden layer

$$\frac{\partial E}{\partial Wxh(i,j)} = \frac{\partial}{\partial u_i}\frac{\partial u_i}{\partial Wxh(i,j)}$$

$$
\begin{aligned}
&= \frac{\partial}{\partial u_i} \frac{\partial}{\partial Wxh(i,j)} \left(\sum_{j=1}^{V} Wxh(i,j) * x_j \right) \\
&= \frac{\partial}{\partial u_i} x_j
\end{aligned}
\tag{29}
$$

这里，已知$u_i = \sum_{j=1}^{V} Wxh(i,j) * x_j$, 矩阵化得到：

$$
\frac{\partial E}{\partial Wxh} = \frac{\partial E}{\partial u} x^T
\tag{30}
$$

8.7 compute the derivative of error with regard to the weight between hidden layer and hidden layer

$$
\begin{aligned}
&\frac{\partial E}{\partial Whh(i,j)} = \frac{\partial}{\partial u_i} \frac{\partial u_i}{\partial Whh(i,j)} \\
&= \frac{\partial}{\partial u_i^t} \frac{\partial}{\partial Whh(i,j)} \left(\sum_{i=1}^{H} Whh(j,i) * h_i^{t-1} + \sum_{i=1}^{V} Wxh(j,i) * x_i \right) \\
&= \frac{\partial}{\partial u_i^t} \left(\frac{\partial}{\partial Whh(i,j)} \sum_{i=1}^{H} Whh(j,i) * h_i^{t-1} + \frac{\partial}{\partial Whh(i,j)} \sum_{i=1}^{V} Wxh(j,i) * x_i \right) \\
&= \frac{\partial}{\partial u_i^t} \left(\frac{\partial}{\partial Whh(i,j)} \sum_{i=1}^{H} Whh(j,i) * h_i^{t-1} + 0 \right) \\
&= \frac{\partial}{\partial u_i^t} h_i^{t-1}
\end{aligned}
\tag{31}
$$

这里，$u_i^t = \sum_{i=1}^{H} Whh(j,i) * h_i^{t-1} + \sum_{i=1}^{V} Wxh(j,i) * x_i$，矩阵化得到：

$$
\frac{\partial E}{\partial Whh} = \frac{\partial E}{\partial u} * (h^{t-1})^T
\tag{32}
$$

9. 总结

到目前为止，我们已经计算得到了所有的参数矩阵关于误差的偏导数，然后就可以根据偏导数进行梯度下降进行参数更新了。

$$\frac{\partial E}{\partial Why} = \frac{\partial E}{\partial y} * H^T$$

$$\frac{\partial E}{\partial Wxh} = \frac{\partial E}{\partial u} x^T$$

$$\frac{\partial E}{\partial Whh} = \frac{\partial E}{\partial u} * (h^{t-1})^T$$

$$\frac{\partial E}{\partial u} = (1 - h \odot h) \frac{\partial E}{\partial h}$$

$$\frac{\partial E}{\partial h} = Why^T * \frac{\partial E}{\partial y} + dhnext$$

$$dhnext = Whh^T \frac{\partial E}{\partial u}$$

其中$dhnext$在计算的过程中，初始化为 0.

例如在(https://github.com/weixsong/min-char-rnn)代码中，每次进行根据当前模型生成 char 序列的时候，假设每次用 30 个字母序列进行参数估计，那么在计算 RNN 参数的时候，就需要从第三十个序列反向的向前面传播误差，所以第三十个字母的 dhnext 可以初始化为 0.