Lecture 6. Neural Tips and Tricks

蜡笔大龙猫@2017.03.07

邮箱：houlisha1987@126.com

Lecture 6主要介绍了深度学习应用的一些小技巧，例如多任务训练、梯度检测、正则化、多种激活函数、参数初始化、学习速率等。

多任务学习（也叫权重共享）

对比上节课我们学到的神经网络，多任务学习就是在输出层用softmax分类器取代标量得分。训练方法依然采用后向传播。

神经网络和传统机器学习方法的不同在于，深度学习需要同时学习词向量和权重。

主要思想：我们在训练多种不同NLP任务（例如NER和POS），可以共享两个任务的词向量和隐藏层的权重，只有输出层的softmax权重不同。损失函数是不同任务损失函数相加，例如：

$$\delta^{total} = \delta^{POS} + \delta^{NER}$$

参考论文《NLP(almost from scratch, Collobert et al.2011)》

成功的神经网络的通用步骤是这样的：

1. 选择合适的网络框架
   1. 框架：单个词、固定窗口、词袋、循环神经网络、递归神经网络、CNN等；
   2. 非线性神经单元

2. 用梯度检测器检查实现代码是否存在bug
3. 参数初始化
4. 优化技巧
5. 检查模型是否强大到过拟合
    1. 如果没有过拟合，改变模型框架或者模型调大
    2. 如果过拟合，请用正则化

下面我们根据这些步骤分别介绍使用技巧。

梯度检测

梯度检测我们在第一次作业中用过，从导数的本质上求得参数的梯度，和我们用后向传播计算得到的梯度对比。公式如下：

$$f\prime(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2\epsilon}$$

其中，$\theta^{(i+)} = \theta + \epsilon \times e_i$

简单的代码实现为：

```
old_value = x[ix]
x[ix] = old_value + h
fxh = f(x)
x[ix] = old_value
grad[ix] = (fxh - fx) / h
```

如果梯度检测失败了应该怎么做？修改代码确定没有bug！

正则化

正则化前面课程提到的次数也很多，和大多数分类器一样，神经网络也需要避免过拟合，使得验证集和测试集能够获得良好的表现。正则化后的损失函数为：

$$J_R = J + \lambda \sum_{i=1}^{L} ||W^{(i)}||_F$$

上式中，$||W^{(i)}||_F$是矩阵$W^{(i)}$的F范数，λ是正则化选项的相对权重。

非线性神经元

目前为止我们讨论的非线性神经元有sigmoid，然而在很多应用中有更好的激活函数。常用的有：

Sigmoid

公式：

$$\sigma(z) = \frac{1}{1 + exp(-z)} \in (0, 1)$$

梯度：

$$\sigma\prime(z) = \sigma(z)(1 - \sigma(z))$$

Tanh

与sigmoid相比，tanh函数收敛更快，输出范围为-1到1。

公式：

$$tanh(z) = \frac{exp(z) - exp(-z)}{exp(z) + exp(-z)} = 2\sigma(2z) - 1 \in (-1, 1)$$

梯度：

$$tanh\prime(z) = 1 - tanh^2(z)$$

Hard Tanh

hard tanh比tanh更容易计算。

公式：

$$hardtanh(z) = \begin{cases} -1 & : & z < -1 \\ z & : & -1 \leq z \leq 1 \\ 1 & : & z > 1 \end{cases}$$

梯度：

$$hardtanh\prime(z) = \begin{cases} 1 & : & -1 \leq z \leq 1 \\ 0 & : & otherwise \end{cases}$$

Soft sign

公式：

$$softsign(z) = \frac{z}{1 + |z|}$$

梯度：

$$softsign\prime(z) = \frac{sgn(z)}{(1 + z)^2}$$

其中 $sgn(z)$是符号函数，根据z的不同返回不同，$sgn(z)=\begin{cases}1 & z>0\\0 & z=0\\-1 & z<0\end{cases}$

ReLu

ReLu，Rectify Linear Unit，是一种比较流行的激活函数，因为它的上限不饱和，在计算视觉应用方面获得成功。

公式：

$$rect(z)=max(z,0)$$

梯度：

$$rect\prime(z)=\begin{cases}1 & : & z>0\\0 & : & otherwise\end{cases}$$

Leaky ReLu

传统ReLu激活单元对非正的z不具备传播误差的功能，leaky ReLu改进了这一缺陷，使得晓得误差能够被后向传播。

公式：

$$leaky(z)=max(z,k\cdot z) \qquad 其中\ 0<k<1$$

梯度：

$$leaky\prime(z)=\begin{cases}1 & : & z>0\\k & : & otherwise\end{cases}$$

MaxOut Network

最近出现的一个非线性网络，公式：

$$f_i(z)=max_{j\in[1,k]}z_{ij}$$

$$z_{ij}=x^TW_{..ij}+b_{ij}$$

这种方法在一些图片数据集上取得了不错的效果。

参数初始化

论文《Understanding the difficulty of training deep feedfor-ward neural networks (2010), Xavier et al》中研究了权重和偏置的初始值不同对训练的影响，结果表明，当权重矩阵 $W\in R^{n^{(l+1)}\times n^{(l)}}$ 采用以下范围的均匀分布来随机初始化时，对sigmoid和tanh激活函数会得到更低的误差率和更快的收敛速度：

$$U[-\sqrt{\frac{6}{n^{(l)}+n^{(l+1)}}},\sqrt{\frac{6}{n^{(l)}+n^{(l+1)}}}]$$

其中，$n^{(l)}$表示输入单元的个数，n^{l+1}表示输出单元的个数。

目的：维护层层之间激活方差和后向传播的梯度方差。

学习速率

模型中梯度更新的速度使用学习速率这个变量来控制，在下面公式中，α表示学习速率：

$$\theta^{new} = \theta^{old} - \alpha\Delta_{\theta}J_t(\theta)$$

梯度更新的速度并不是越快越好，$alpha$太大，可能会导致无法收敛到最优解。在非凸模型中(我们遇到的大部分模型都是非凸的)，很大的学习速率导致损失函数的发散几率更高。

关于学习速率的设置有很多变种，详细信息可以看讲义。

AdaGrad

AdaGrad可以说是标准的SGD，但是只有一点不同：它的每个参数的学习速率是不同的。每个参数的学习速率依赖于历史更新信息，换句话说，没有更新过的参数的学习速率可能更高，用公式表示：

$$\theta_{t,i} = \theta_{t-1,i} - \frac{\alpha}{\sqrt{\sum_{\tau=1}^{t} g_{\tau,i}^2}} g_{t,i}$$

其中，$g_{t,i} = \frac{\partial}{\partial\theta_i^t}J_t(\theta)$

简单的代码实现：

```
cache += dx**2
x += - learning_rate * dx / np.sqrt(cache + 1e-8)
```

其他方法