



Differentiable neural computers

可微分神经计算机：从手调feature到手调网络
《利用神经网络与外部动态存储器进行混合计算》

深度学习框架Caffe学习与应用 第4课

DATAGURU专业数据分析社区

本节课内容

- 1. Caffe可视化工具
 - 1.0 准备pycaffe环境
 - 1.1 网络模型可视化
 - 1.2 caffemodel可视化
 - 1.3 特征图可视化
 - 1.4 可视化loss和accuracy曲线
- 2. Caffe中五种类型的层的实现和参数配置
 - 2.1 卷积层 (Convolution)
 - 2.2 池化层 (Pooling)
 - 2.3 激活层 (ReLU、Sigmoid、TanH、AbsVal、Power)
 - 2.4 全连接层 (InnerProduct)
 - 2.5 softmax层 (SoftmaxWithLoss、Softmax)

1.0 准备pycaffe环境

已经执行过的caffe安装命令：

```
# cmake -D CPU_ONLY=on -D CMAKE_INSTALL_PREFIX=/usr/local ..  
# make all  
# make install
```

首先执行完下面命令，进入\${CAFFE_ROOT}/python 才可以执行其中的python脚本：

```
sudo apt-get update  
sudo apt-get install python-pip python-dev python-numpy  
sudo apt-get install gfortran graphviz  
sudo pip install -r ./requirements.txt  
sudo pip install pydot
```

1.1 网络结构可视化工具

■ 1. 代码：{caffe_root}/tool/draw_net.py

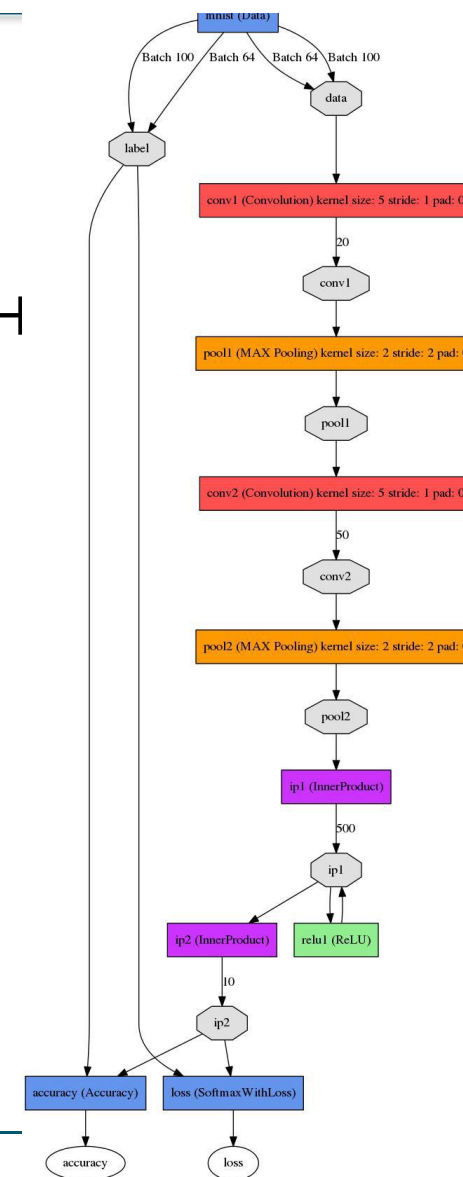
命令：./draw_net.py --rankdir TB ./lenet_train_test.prototxt mnist.png

caffe的环境变量：export PYTHONPATH=/home/yll/caffe/python:\$PYTHONPATH

Tips:Caffe' s Python interface works with Python 2.7. Python 3 or earlier

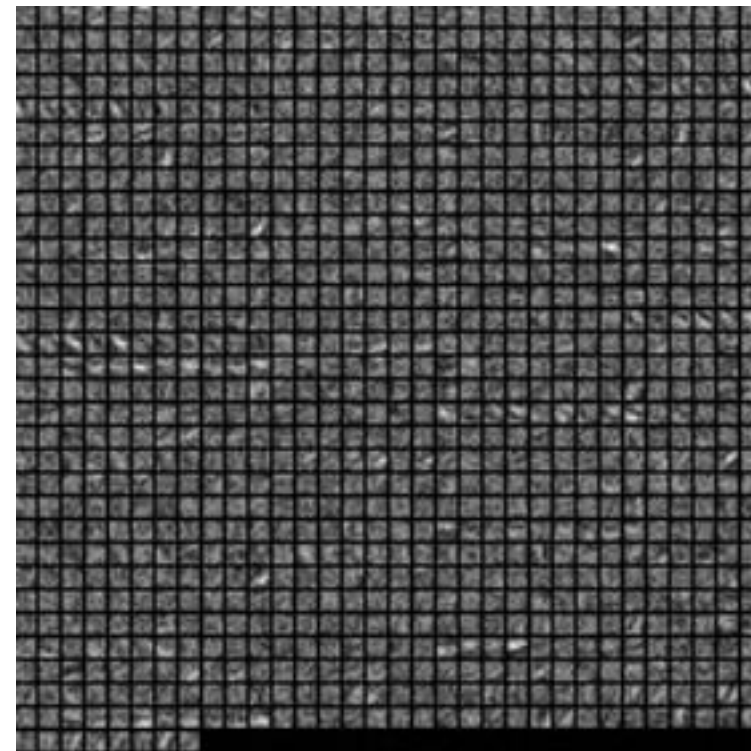
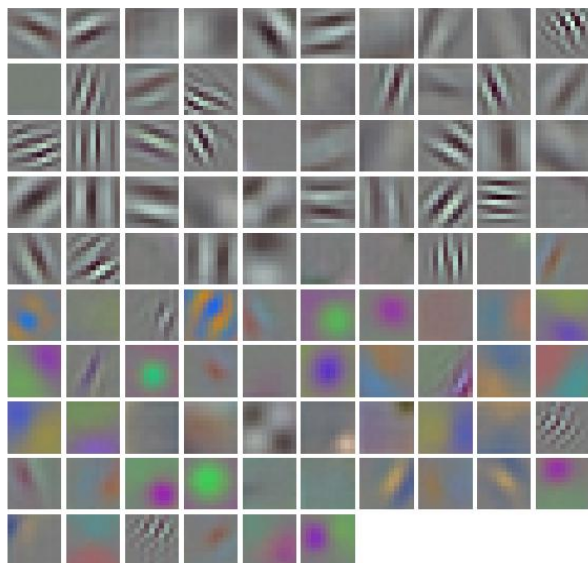
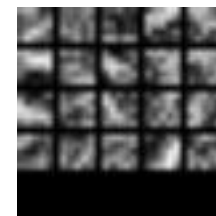
■ 2. 在线可视化工具 <http://ethereon.github.io/netscope/#/editor>

(推荐使用第二种方式)



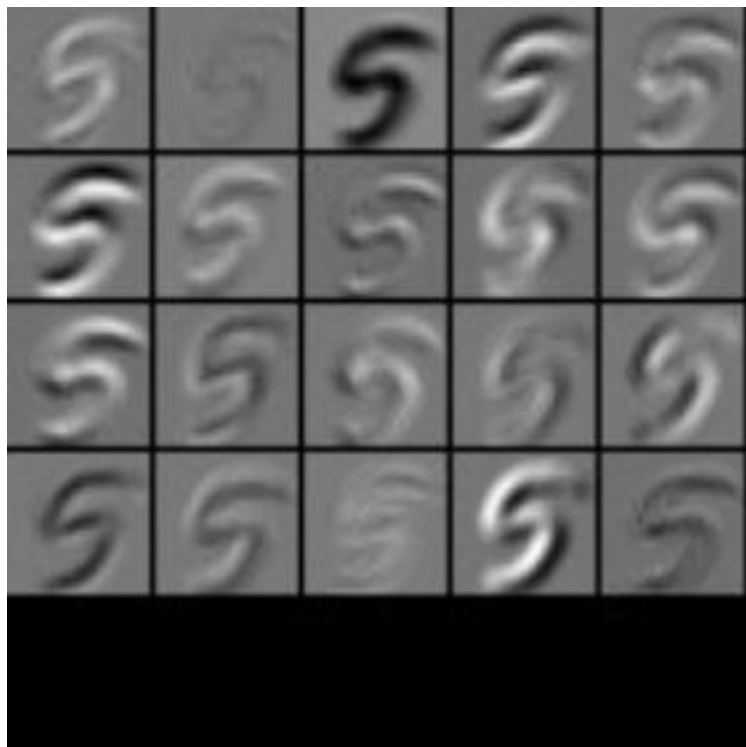
1.2 caffemodel的可视化

- 课程代码：test_extract_weights.py
- 右图为训练mnist生成的LeNet Conv1和Conv2的权重值的可视化：
 - 训练良好：美观、光滑的滤波器
 - 训练时间不够或者过拟合：出现噪声图样
- 下图为CaffeNet的Conv1图：



1.3 特征图可视化

- 课程代码：test_extract_data.py
- * 代码注释讲解



1.4 可视化loss和accuracy曲线

- 工具代码：{caffe_root}/tools/extra/plot_training_log.py.example

Usage:

```
./plot_training_log.py chart_type[0-7] /where/to/save.png /path/to/first.log
```

...

Notes:

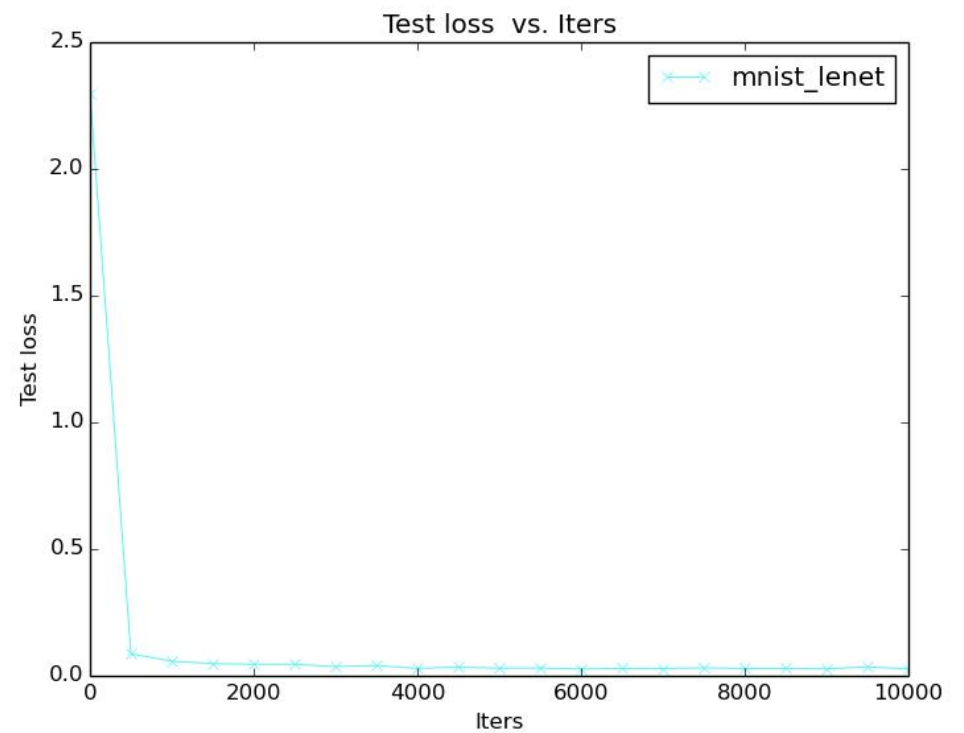
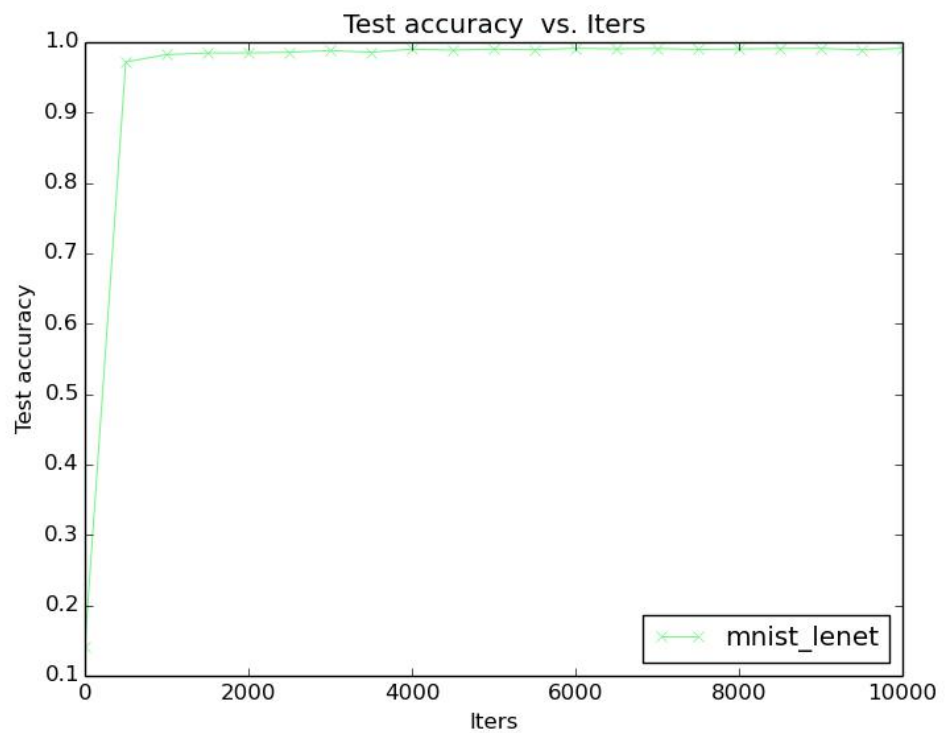
1. Supporting multiple logs.
2. Log file name must end with the lower-cased ".log".

Supported chart types:

- 0: Test accuracy vs. Iters
- 1: Test accuracy vs. Seconds
- 2: Test loss vs. Iters
- 3: Test loss vs. Seconds
- 4: Train learning rate vs. Iters
- 5: Train learning rate vs. Seconds
- 6: Train loss vs. Iters
- 7: Train loss vs. Seconds

例如：cd ~/caffe/tools/extra

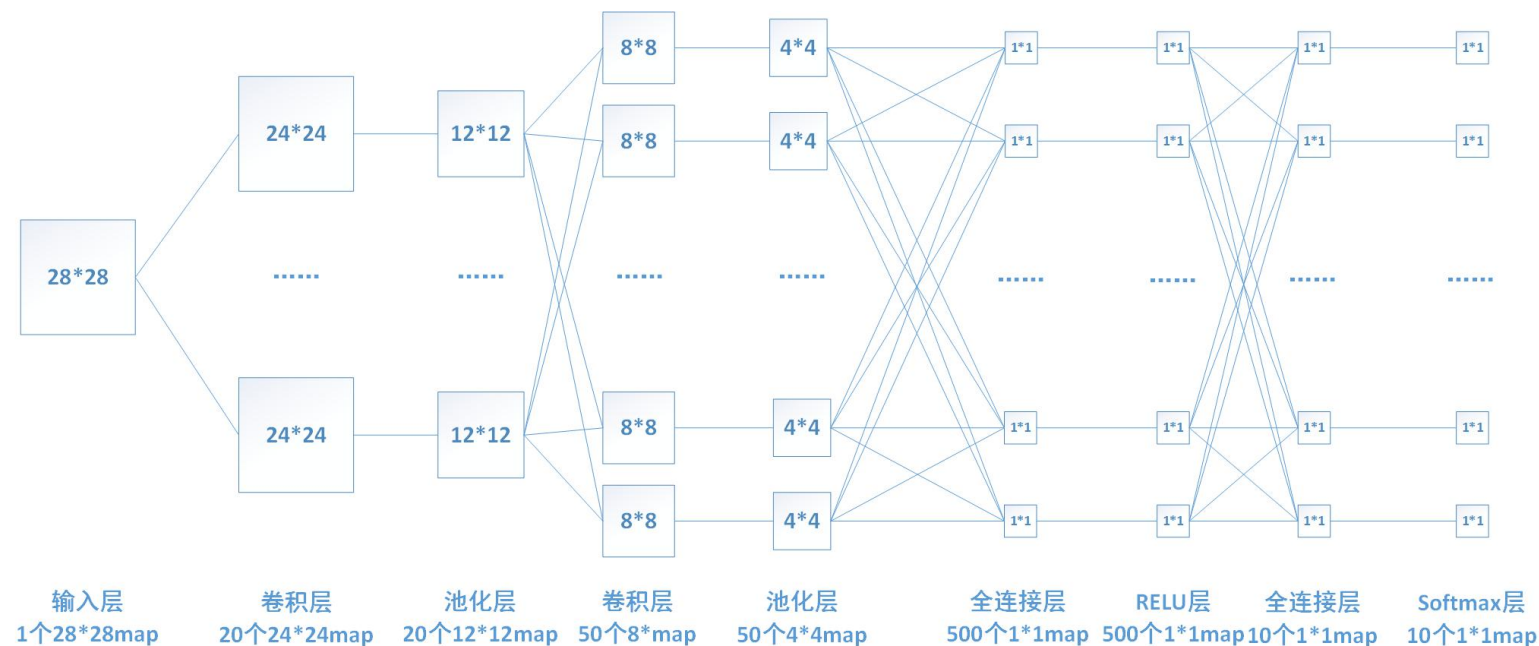
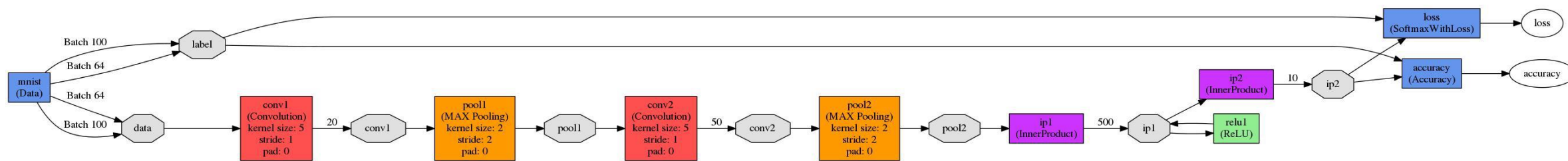
```
python plot_training_log.py 0 test.png ~/caffe/examples/mnist/mnist_lenet.log
```



2.Caffe中五种层的实现和参数配置

Data: mnist

Net : lenet_train_test.prototxt



2.1 卷积层

Diagram illustrating a 2D convolution operation:

Input matrix $A(4 \times 4)$:

1*1	1*0	0	1
1*1	0*1	0	1
0	1	1	0
1	1	1	1

Kernel matrix $B(2 \times 2)$:

1	0
1	1

Operation: \times 卷积

Output matrix $C(3 \times 3)$:

2	1	1
2	2	1
2	3	3

例子：

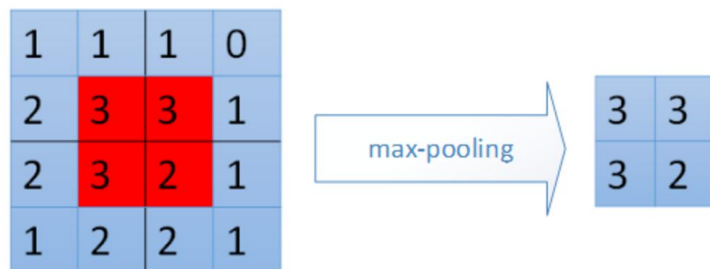
输入为 28×28 的图像，经过 5×5 的卷积之后，得到一个 $(28-5+1) \times (28-5+1) = 24 \times 24$ 的map。

* 每个map是不同卷积核在前一层每个map上进行卷积，并将每个对应位置上的值相加然后再加上一个偏置项。

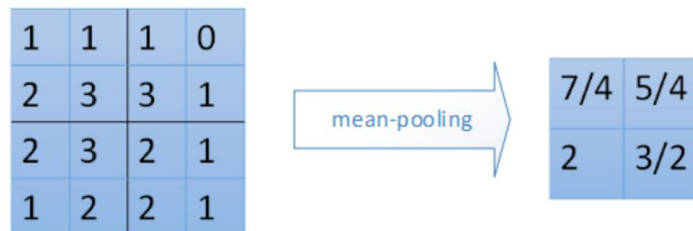
```
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  param {
    lr_mult: 1          #学习率1，和权值更新相关
  }
  param {
    lr_mult: 2          #学习率2，和权值更新相关
  }
  convolution_param {
    num_output: 50      # 50个输出的map
    kernel_size: 5       #卷积核大小为5*5
    stride: 1            #卷积步长为1
    weight_filler {      #权值初始化方式
      type: "xavier"     #默认为 "constant",值全为0，很多时候
    }                   #我们也可以用"xavier"或者" gaussian"来进行初始化
    bias_filler {        #偏置值的初始化方式
      type: "constant"   #该参数的值和weight_filler类似，一般
    }                   #设置为"constant"，值全为0
  }
}
```

2.2 池化层

max-pooling:



mean-pooling:

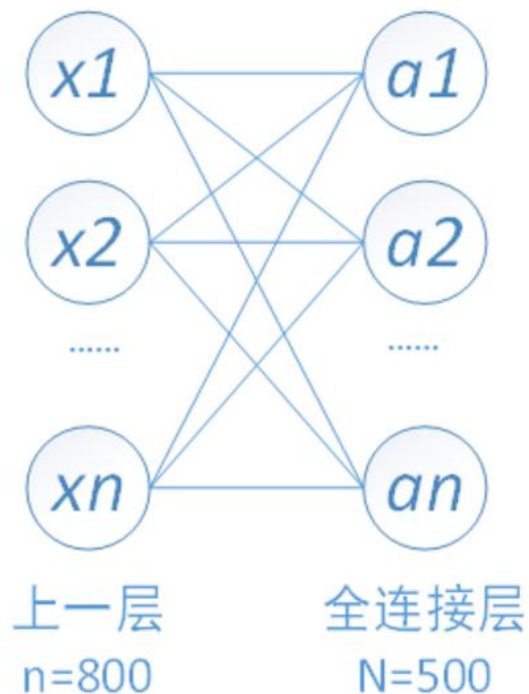


例子：

输入为卷积层1的输出，大小为 24×24 ，对每个不重叠的 2×2 的区域进行降采样。对于max-pooling，选出每个区域中的最大值作为输出。而对于mean-pooling，需计算每个区域的平均值作为输出。最终，该层输出一个 $(24/2) \times (24/2)$ 的map

```
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX #Pool为池化方式，默认值为MAX，可以选择的参数有MAX、AVE、STOCHASTIC
    kernel_size: 2 #池化区域的大小，也可以用kernel_h和kernel_w分别设置长和宽
    stride: 2 #步长，即每次池化区域左右或上下移动的距离，一般和kernel_size相同，即为不重叠池化。也可以也可以小于kernel_size，即为重叠池化，Alexnet中就用到了重叠池化的方法
  }
}
```

2.3 全连接层



50*4*4=800个输入结点和500个输出结点

#参数和卷积层表达一样

```
layer {  
  name: "ip1"  
  type: "InnerProduct"  
  bottom: "pool2"  
  top: "ip1"  
  param {  
    lr_mult: 1  
  }  
  param {  
    lr_mult: 2  
  }  
  inner_product_param {  
    num_output: 500  
    weight_filler {  
      type: "xavier"  
    }  
    bias_filler {  
      type: "constant"  
    }  
  }  
}
```

2.4 激活函数层

- 激活函数作用：激活函数是用来引入非线性因素的。
- 激活函数一般具有以下性质：
 - 非线性：线性模型的不足我们前边已经提到。
 - 处处可导：反向传播时需要计算激活函数的偏导数，所以要求激活函数除个别点外，处处可导。
 - 单调性：当激活函数是单调的时候，单层网络能够保证是凸函数。
 - 输出值的范围：当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征表示受有限权值的影响更显著；当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况下，一般需要更小的learning rate.

```
layer {  
  name: "relu1"  
  type: "ReLU"  
  bottom: "ip1"  
  top: "ip1"  
}
```

```
layer {  
  name: "layer"  
  bottom: "in"  
  top: "out"  
  type: "Power"  
  power_param {  
    power: 2  
    scale: 1  
    shift: 0  
  }  
}
```

Type为该层类型，可取值分别为：

(1)ReLU：表示我们使用relu激活函数，relu层支持in-place计算，这意味着该层的输入和输出共享一块内存，以避免内存的消耗。

(2)Sigmoid：代表使用sigmoid函数；

(3) TanH：代表使用tanh函数；

(4) AbsVal：计算每个输入的绝对值 $f(x)=Abs(x)$

(5)power对每个输入数据进行幂运算

$f(x) = (shift + scale * x) ^ power$

层类型：Power


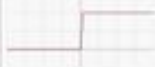












可选参数：

power: 默认为1

scale: 默认为1

shift: 默认为0

激活函数列表：

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$
Bent identity		$f(x) = \frac{\sqrt{x^2 + 1} - 1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2 + 1}} + 1$
SoftExponential		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1 - \alpha(x + \alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1 - \alpha(x + \alpha)} & \text{for } \alpha < 0 \\ 1 & \text{for } \alpha = 0 \\ e^{\alpha x} & \text{for } \alpha > 0 \end{cases}$
Sinnsoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$

- 一般比较常见的激活函数有sigmoid、tanh和Relu，其中Relu由于效果最好，现在使用的比较广泛。

relu函数的表达式为 $f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$ ，所以前向传播时，大于0的输入不变，小于0的置零即可。

2.5 softmax层

Softmax回归模型是logistic回归模型在多分类问题上的推广，在多分类问题中，待分类的类别数量大于2，且类别之间互斥。

Softmax公式：

$$f(z_j) = \frac{e^{z_j}}{\sum_{i=1}^n e^{z_i}}$$

通常情况下softmax会被用在网络中的最后一层，用来进行最后的分类和归一化。

#可以计算给出每个样本对应的损失函数值

```
layer {  
  name: "loss"  
  type: "SoftmaxWithLoss"  
  bottom: "ip2"  
  bottom: "label"  
  top: "loss"  
}
```

#输出为每个类别的概率值

```
layers {  
  name: "prob"  
  type: "Softmax"  
  bottom: "ip2"  
  top: "prob"  
}
```