深蓝学院
shenlanxueyuan.com

深度学习理论到实践（一）

——Keras和常见网络结构

主讲人 常博士在读

课程目的

深蓝学院
shenlanxueyuan.com

- ✓深度学习库Keras入门
- ✓训练深度网络的困难
  - ✓ 梯度问题
- ✓ 基本的处理方案
- ✓ 从网络结构上的改进
  - ◆ Highway Network：Highway 网络
  - ◆ Deep Residual Network：深度残差网络

课程目的

深蓝学院
shenlanxueyuan.com

- ✓**深度学习库Keras入门**
- ✓训练深度网络的困难
  - ✓ 梯度问题
- ✓ 基本的处理方案
- ✓ 从网络结构上的改进
  - ◆ Highway Network：Highway 网络
  - ◆ Deep Residual Network：深度残差网络

Keras入门-简介

深蓝学院
shenlanxueyuan.com

- ✓选择Keras原因：
  - ✓简单
  - ✓GPU和CPU之间的无缝结合
  - ✓可扩展性强
  - ✓依赖于Python，大量的库可以使用
  - ✓以thenao,CNTK和tensorflow为底层

Keras入门-简介

深蓝学院
shenlanxueyuan.com

- ✓Keras怎么做：
  - ✓搭积木
    - ✓神经网络是高度模块化的
    - ✓有了“层”，网络便可以搭建
  - ✓提供了丰富的API层接口
  - ✓给出自行实现“层”的接口

Keras入门-安装

深蓝学院 shenlanxueyuan.com

- ✓ Windows

这里面最后一条是我们第2步安装的MinGW的位置，这个按照自己的系统来设置，注意最后一

5：关机重启，因为添加了新的路径进去。

6：安装我们的库。给一个不能更简单的方法。打开cmd，保证有网，输入“pip install keras==1.0.5”。不需要单独下载安装theano的，这个会给你自动安装的，一般是最新版本的theano

```
5  [gcc]
6  cxxflags = -IC:\Users\lg\Anaconda2\MinGW
```

Keras入门-安装

深蓝学院
shenlanxueyuan.com

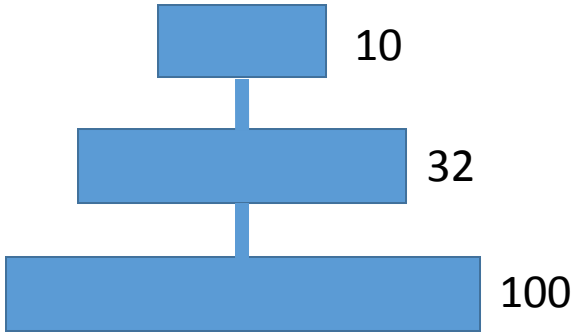
8：到目前为止，所有需要的算是安装完了。最后配置.theanorc 文件内容，如下：

```
[cuda]
root=/usr/local/cuda-8.0/bin
[global]
floatX=float32
device=gpu
[nvcc]
fastmath=True
[lib]
cnmem=1
```

Keras入门-简单例子

深蓝学院 shenlanxueyuan.com

```
 8 import numpy as np
 9 import h5py
10 import os
11 os.environ['KERAS_BACKEND'] = 'theano'
12 os.environ['THEANO_FLAGS']='device=gpu7,lib.cnmem=0.1'
13
14 from keras.models import Sequential
15 from keras.layers import Dense, Activation
16
17 model = Sequential()
18 model.add(Dense(32, activation='relu', input_dim=100))
19 model.add(Dense(10, activation='sigmoid'))
20 model.compile(optimizer='rmsprop',
21               loss='binary_crossentropy',
22               metrics=['accuracy'])
23
24 file = h5py.File('my_data.h5')
25 x = file['X_train'][:]
26 y = file['Y_train'][:]
27 file.close()
28
29 model.fit(x, y, epochs=10, batch_size=32)
```

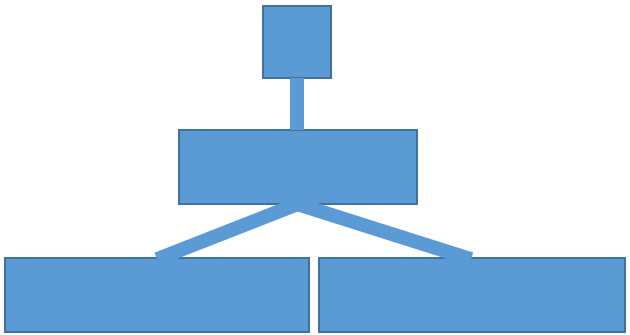


Keras入门-复杂例子

深蓝学院
shenlanxueyuan.com

✓复杂例子：

```
16 from keras.models import Sequential, Model
17 from keras.layers import Dense, Activation
18
19 x_left = Input(shape=(100,))
20 x_right = Input(shape=(100,))
21 x_left = Dense(32, activation='relu')(x_left)
22 x_right = Dense(32, activation='relu')(x_right)
23 x = keras.layers.concatenate([x_left, x_right], axis=-1)
24 y = Dense(10, activation='sigmoid')(x)
25 model = Model([x_left, x_right],[y])
26
27 file = h5py.File('my_data.h5')
28 x = file['X_train'][:]
29 y = file['Y_train'][:]
30 file.close()
31
32 model.fit([x,x], y, epochs=10, batch_size=32)
```



Ker

深蓝学院
shenlanxueyuan.com

✓用C]:

```
changjianlong@node01:~$ python test.py
Using Theano backend.
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 169s - loss: 0.3945 - acc: 0.8800
```

```
changjianlong@node01:~$ python test.py
Using Theano backend.
Using gpu device 0: GeForce GTX 1080 (CNMeM is disabled, cuDNN 5005)
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 7s - loss: 0.3820 - acc: 0.8824
```

```
changjianlong@node01:~$ python test.py
Using Theano backend.
Using gpu device 0: GeForce GTX 1080 (CNMeM is enabled with initial size: 95.0% of memory, cuDNN 5005)
X_train shape: (60000, 1, 28, 28)
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 2s - loss: 0.3827 - acc: 0.8822 - val_loss: 0.1029 - val_acc: 0.9676
Epoch 2/12
60000/60000 [==============================] - 2s - loss: 0.1507 - acc: 0.9559 - val_loss: 0.0673 - val_acc: 0.9788
Epoch 3/12
60000/60000 [==============================] - 2s - loss: 0.1153 - acc: 0.9657 - val_loss: 0.0556 - val_acc: 0.9821
Epoch 4/12
```

课程目的

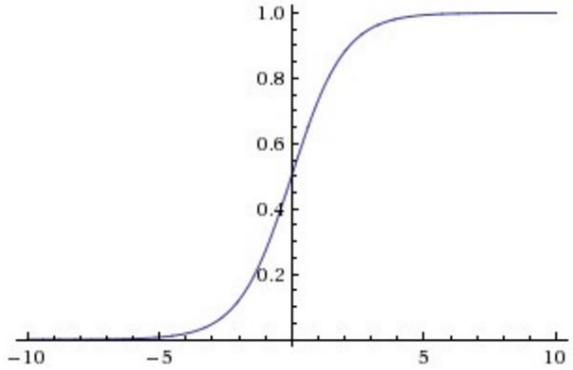
深蓝学院
shenlanxueyuan.com

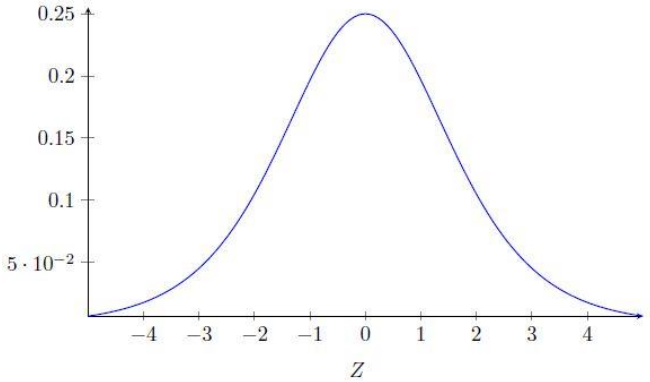
- ✓深度学习库Keras入门
- ✓训练深度网络的困难
  - ✓ 梯度问题
- ✓ 基本的处理方案
- ✓ 从网络结构上的改进
  - ◆ Highway Network：Highway 网络
  - ◆ Deep Residual Network：深度残差网络

训练深度网络 - 困难

深蓝学院 shenlanxueyuan.com

- ✓ 神经网络的非线性拟合能力
  - ✓ 激活函数（非线性）
    - ✓ sigmoid：非线性，易求导，压缩数据
    - ✓ tanh：前向传递有优势

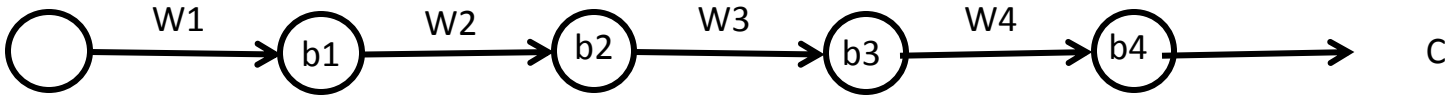




训练深度网络 – 困难

深蓝学院 shenlanxueyuan.com

- ✓ 梯度消失
  - ✓ 连乘使得底层网络参数无法训练
  - ✓ 没有更好地优化参数
  - ✓ 属于BP算法的缺陷（梯度更新的时候，用的同意学习率，这是不合理的。不同特征不属于一个尺度）



$$a_j = w_j * a_j + b_j \qquad b_j = \sigma'(a_j)$$

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial b_1} a_4$$

训练深度网络 - 解决方案

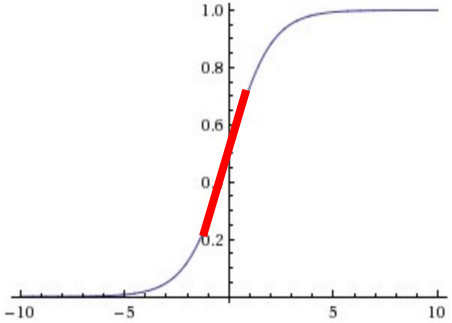
深蓝学院
shenlanxueyuan.com

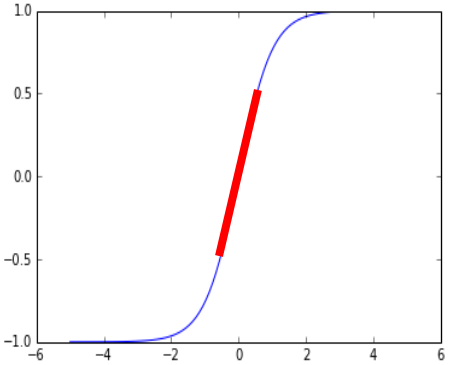
- ✓ 好的初始值
  - ✓ xavier 初始化方法
- ✓ 好的激活函数
  - ✓ ReLU, LeakyReLU
- ✓ 层归一化
  - ✓ Batch normalization
- ✓ 新的网络结构
  - ✓ Highway网络 -> 残差网络

解决方案

深蓝学院 shenlanxueyuan.com

- ✓ 好的初始值：xavier 初始化方法
  - ✓ 激活函数在靠近0的地方近似于一个线性函数
  - ✓ 假设输入X和W属于不同的 (1)0均值 (2)固定方差
  - ✓ Z = WX：有着0均值，但是方差会变大



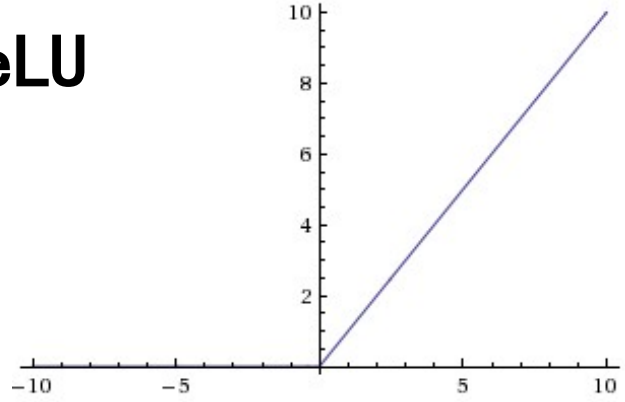


  - ✓ 前向中：每层输出是0均值且方差不变
  - ✓ 反向中：每层梯度是0均值且方差不变
  - ✓ W的分布为$[-\sqrt{\frac{6}{n^k+n^{k+1}}},\sqrt{\frac{6}{n^k+n^{k+1}}}]$即可

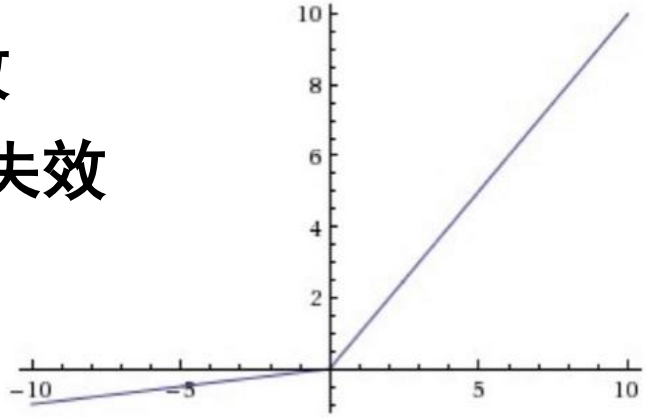
解决方案

深蓝学院
shenlanxueyuan.com

- ✓ 好的激活函数：ReLU，Leaky ReLU



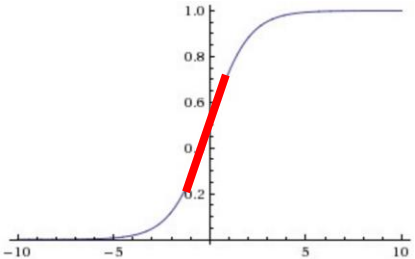
- ✓ 没有对数据压缩
  - ✓ sigmoid,tanh:前向优秀，反向失效
  - ✓ ReLU,Leaky ReLU:反向优秀，前向失效

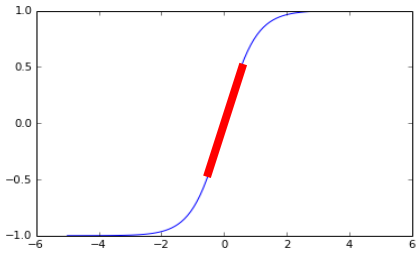


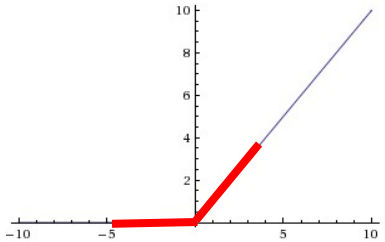
解决方案

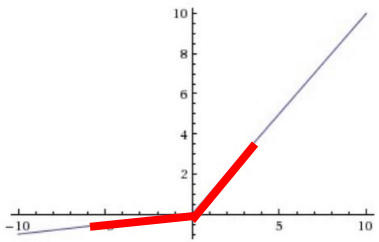
深蓝学院
shenlanxueyuan.com

- ✓ 层归一化：Batch Normalization
  - ✓ 对数据的归一化操作，使得数据位于0附近
  - ✓ Sigmoid，tanh：反传梯度变大
  - ✓ ReLU，Leaky ReLU：稀疏，压缩了数据









解决方案

深蓝学院
shenlanxueyuan.com

- ✓ 新的网络结构：Highway网络和残差网络
  - ✓ 受启发于LSTM：加入门的概念
  - ✓ 使得数据传输更流畅
  - ✓ 梯度消失：让信息更好地在网络中流通

网络结构 - Highway网络

深蓝学院 shenlanxueyuan.com

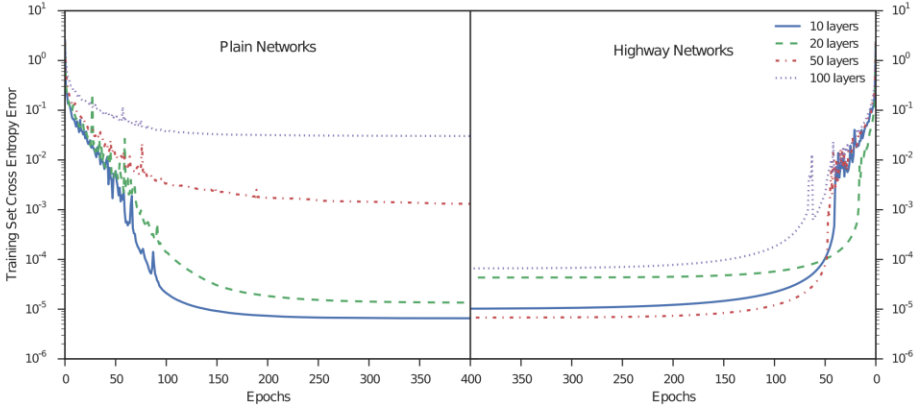
- ✓ 控制输入
- ✓ 控制输出

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H})$$

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C})$$

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T}))$$

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$



网络结构 – 残差网络

深蓝学院 shenlanxueyuan.com

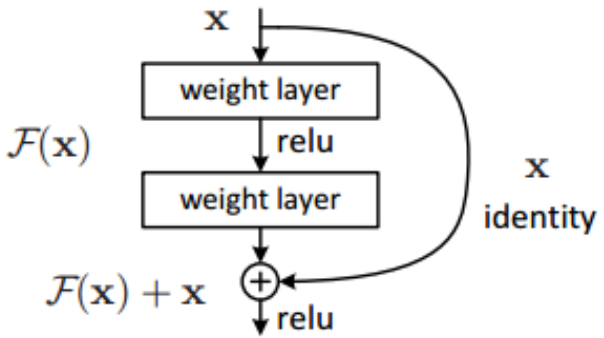
- ✓ 残差是Highway网络的一个特例
  - ✓ 更少的参数
  - ✓ 输入门永远不关

$$y = H(x, W_H)T(x, W_T) + x(1 - T(x, W_T))$$

$$y = (H(x, W_H) - x)T(x, W_T) + x$$

令：$F(x, W_F) = H(x, W_H) - x, T(x, W_T) = I,$

$$\Rightarrow y = F(x, W_F) + x$$



Highway网络 - Keras实现

深蓝学院 shenlanxueyuan.com

```
Class Highway(Layer):  # 继承Layer类
    def __init__(self, init='glorot_uniform', transform_bias=-2,activation='linear', weights=None,
                 W_regularizer=None, b_regularizer=None, activity_regularizer=None,W_constraint=None,
                 b_constraint=None,bias=True, input_dim=None, **kwargs):  # 初始化参数
        self.init = initializations.get(init)  # 初始化方法
        self.transform_bias = transform_bias  # 偏置初始化值
        self.activation = activations.get(activation)  # 激活函数
        self.W_regularizer = regularizers.get(W_regularizer)  # 矩阵参数正则
        self.b_regularizer = regularizers.get(b_regularizer)  # 偏置正则
        self.activity_regularizer = regularizers.get(activity_regularizer)  # 激活值正则
        self.W_constraint = constraints.get(W_constraint)  # 矩阵参数约束
        self.b_constraint = constraints.get(b_constraint)  # 偏置约束
        self.bias = bias  # 是否需要偏置
        self.initial_weights = weights  # 初始化权重
        self.input_dim = input_dim  # 输入维度
        if self.input_dim:
            kwargs['input_shape'] = (self.input_dim,)
        super(Highway, self).__init__(**kwargs)
```

Highway网络 - Keras实现

深蓝学院 shenlanxueyuan.com

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T}))$$

```
def build(self, input_shape):  # 建立可学习变量并初始化
# Highway中输入维度必须等于输出维度：因为上式中第二项需要与x进行求和
    input_dim = input_shape[1]
    #  self.W --- W_{H}     self.W_carry --- W_{T}
    self.W = self.init((input_dim, input_dim),name='{}_W'.format(self.name))
    self.W_carry = self.init((input_dim, input_dim),name='{}_W_carry'.format(self.name))
    #  self.W, self.W_carry 分别对应的偏置  self.b, self.b_carry
    self.b = K.zeros((input_dim,), name='{}_b'.format(self.name))
    self.b_carry = K.variable(np.ones((input_dim,)) * self.transform_bias,
                              name='{}_b_carry'.format(self.name))

    # 指定可学习参数，没有指定的保持初始值不变
    self.trainable_weights = [self.W, self.b, self.W_carry, self.b_carry]
```

Highway网络 - Keras实现

深蓝学院 shenlanxueyuan.com

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T}))$$

```
def call(self, x, mask=None):
        #     self.W --- W_{H}      self.W_carry --- W_{T}
        y = K.dot(x, self.W_carry)
        y += self.b_carry
        transform_weight = activations.sigmoid(y)  # T(x,W_{T})
        y = K.dot(x, self.W)
        y += self.b
        act = self.activation(y)  # H(x,W_{H})
        act *= transform_weight  # 第一个等式
        output = act + (1 - transform_weight) * x  # 两项相加，且均为点乘
        return output
```

Highway网络 - Keras实现

深蓝学院
shenlanxueyuan.com

- inp_ = Input(shape = (nb_features,))
- x = Highway(activation='relu')(inp_)
- for _ in range(nb_layer): # 多个Highway层堆叠
- x = Highway(activation='relu')(x)

- model = Model(inp_img,x)
- model.compile(loss = 'mse',optimizer = 'rmsprop')

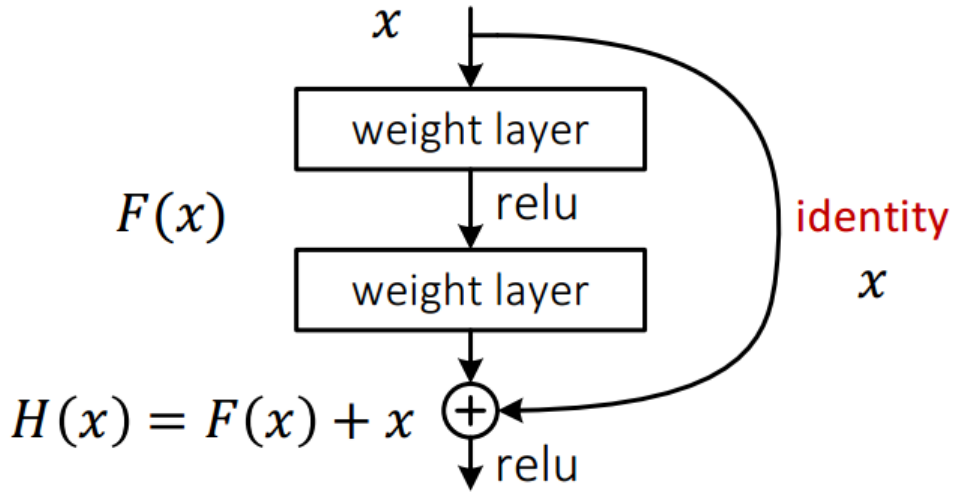
残差网络 – Keras实现

深蓝学院
shenlanxueyuan.com

```
def block(x, filters, bn_axis):
# 实现一个残差block
    # 用一个子神经网络实现残差函数F(x)
    f = Conv2D(filters1, (1, 1))(x)
    f = BatchNormalization(axis=bn_axis)(f)
    f = Activation('relu')(f)


    # x:输入 + f:残差F(x)
    y = layers.add([f, x])
    y = Activation('relu')(y)


    return y
```



残差网络 – Keras实现

深蓝学院
shenlanxueyuan.com

```
input_shape = _obtain_input_shape(input_shape)

input_ = Input(shape=input_shape)
for _ in range(nb_block):   # 多个block的堆叠来建立更深的网络
    x = block(x, [64],2)
 x = AveragePooling2D((7, 7), name='avg_pool')(x)

model = Model(input_, x, name='resnet')
model.compile(loss = 'mse',optimizer = 'rmsprop')
```

网络结构 - SNN网络

深蓝学院 shenlanxueyuan.com

✓ 梯度消失和扩散已经被几乎完全解决

Self-Normalizing Neural Networks

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, Sepp Hochreiter

(Submitted on 8 Jun 2017 (v1), last revised 22 Jun 2017 (this version, v3))

Deep Learning has revolutionized vision via convolutional neural networks (CNNs) and natural language processing via recurrent neural networks (RNNs). However, success stories of Deep Learning with standard feed-forward neural networks (FNNs) are rare. FNNs that perform well are typically shallow and, therefore cannot exploit many levels of abstract representations. We introduce self-normalizing neural networks (SNNs) to enable high-level abstract representations. While batch normalization requires explicit normalization, neuron activations of SNNs automatically converge towards zero mean and unit variance. The activation function of SNNs are "scaled exponential linear units" (SELUs), which induce self-normalizing properties. Using the Banach fixed-point theorem, we prove that activations close to zero mean and unit variance that are propagated through many network layers will converge towards zero mean and unit variance -- even under the presence of noise and perturbations. This convergence property of SNNs allows to (1) train deep networks with many layers, (2) employ strong regularization, and (3) to make learning highly robust. Furthermore, for activations not close to unit variance, we prove an upper and lower bound on the variance, thus, vanishing and exploding gradients are impossible. We compared SNNs on (a) 121 tasks from the UCI machine learning repository, on (b) drug discovery benchmarks, and on (c) astronomy tasks with standard FNNs and other machine learning methods such as random forests and support vector machines. SNNs significantly outperformed all competing FNN methods at 121 UCI tasks, outperformed all competing methods at the Tox21 dataset, and set a new record at an astronomy data set. The winning SNN architectures are often very deep. Implementations are available at: github.com/bioinf-jku/SNNs.

Comments: 9 pages (+ 93 pages appendix)

网络结构 - SNN网络

深蓝学院
shenlanxueyuan.com

- ✓ 梯度消失和扩散已经被几乎完全解决

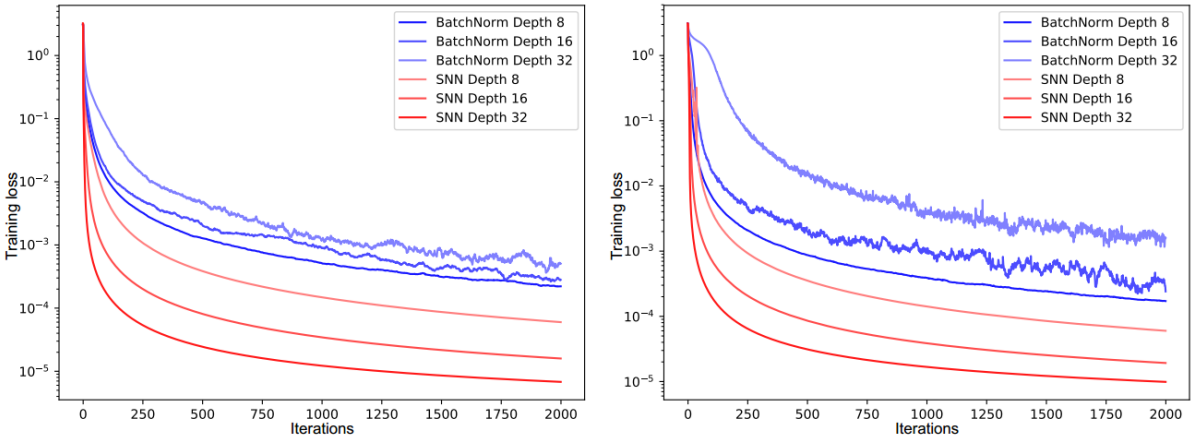


Figure 1: The left panel and the right panel show the training error (y-axis) for feed-forward neural networks (FNNs) with batch normalization (BatchNorm) and self-normalizing networks (SNN) across update steps (x-axis) on the MNIST dataset the CIFAR10 dataset, respectively. We tested networks with 8, 16, and 32 layers and learning rate 1e-5. FNNs with batch normalization exhibit high variance due to perturbations. In contrast, SNNs do not suffer from high variance as they are more robust to perturbations and learn faster.

在线问答

深蓝学院
shenlanxueyuan.com

Q&A

课程地址

《点击此处添加演讲主题演讲主题演讲主题演讲主题》

深蓝学院
shenlanxueyuan.com

《点击此处添加演讲主题演讲主题演讲主题演讲主题》

课程地址

感谢各位聆听！

Thanks for Listening