NobleProg

第二部分 深度学习

沈来信 副教授 博士生

目录

NobleProg

1. K-means特征学习

- 1）通过最小化数据点和最近邻中心的距离来寻找各个类中心
- 2）“矢量量化vector quantization”

把K-means当成是在构建一个字典$D\in R^{nxk}$，通过最小化重构误差，一个数据样本$x^{(i)}\in R^{n}$可以通过这个字典映射为一个k维的码矢量。所以K-means实际上就是寻找D的一个过程：

$s^{(i)}$就是一个与输入$x^{(i)}$对应的码矢量。$D^{(j)}$是字典D的第j列。K-means毕生的目标就是寻找满足上面这些条件的一个字典D和每个样本$x^{(i)}$对应的码矢量$s^{(i)}$

$$\underset{\mathcal{D},s}{\text{minimize}} \sum_i ||\mathcal{D}s^{(i)} - x^{(i)}||_2^2$$

$$\text{subject to } ||s^{(i)}||_0 \leq 1, \forall i$$

$$\text{and } ||\mathcal{D}^{(j)}||_2 = 1, \forall j$$

$$\underset{\mathcal{D},s}{\text{minimize}} \sum_i ||\mathcal{D}s^{(i)} - x^{(i)}||_2^2 + \lambda||s^{(i)}||$$

$$\text{subject to } ||D^{(j)}||_2 = 1, \forall j.$$

$$s_j^{(i)} = \begin{cases} \mathcal{D}^{(j)\top} x^{(i)} & \text{if } j == \arg\max_l |\mathcal{D}^{(l)\top} x^{(i)}| \\ 0 & \text{otherwise.} \end{cases}$$

1. Normalize inputs:

$$x^{(i)} := \frac{x^{(i)} - \text{mean}(x^{(i)})}{\sqrt{\text{var}(x^{(i)}) + \epsilon_{\text{norm}}}}, \forall i$$

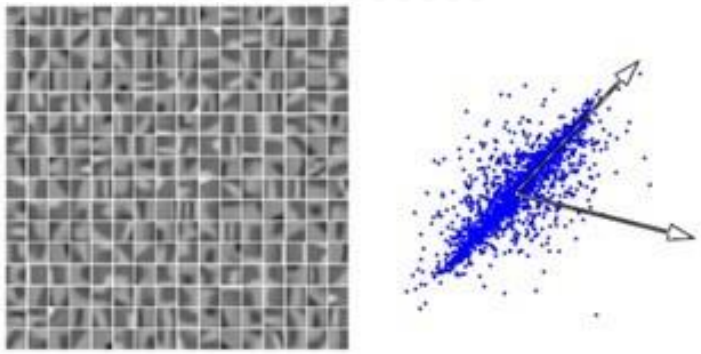
2. Whiten inputs:

$$[V, D] := \text{eig}(\text{cov}(x)); \ // \text{ So } VDV^\top = \text{cov}(x)$$
$$x^{(i)} := V(D + \epsilon_{\text{zca}} I)^{-1/2} V^\top x^{(i)}, \forall i$$

3. Loop until convergence (typically 10 iterations is enough):

$$s_j^{(i)} := \begin{cases} \mathcal{D}^{(j)\top} x^{(i)} & \text{if } j == \arg\max_l |\mathcal{D}^{(l)\top} x^{(i)}| \\ 0 & \text{otherwise.} \end{cases} \forall j, i$$

$$\mathcal{D} := XS^\top + \mathcal{D}$$
$$\mathcal{D}^{(j)} := \mathcal{D}^{(j)} / ||D^{(j)}||_2 \forall j$$



(a) (b) (c)

- 1）**白化：**先用白化来去除数据的相关性，以驱使K-means在正交方向上分配更多的聚类中心。实现whitening白化一个方法是ZCA白化。先对数据点x的协方差矩阵进行特征值分解cov(x)=VDV^T。然后白化后的点可以表示为：

$$V(D+\epsilon_{\text{zca}}I)^{-1/2}V^{\top}x$$

2）**聚类中心的抑制更新**damped updates

$$\begin{aligned}\mathcal{D}_{\text{new}} &:= \arg\min_{\mathcal{D}} ||\mathcal{D}S - X||_2^2 + ||\mathcal{D} - \mathcal{D}_{\text{old}}||_2^2 \\ &= (SS^{\top} + I)^{-1}(XS^{\top} + \mathcal{D}_{\text{old}}) \\ &\propto XS^{\top} + \mathcal{D}_{\text{old}} \\ \mathcal{D}_{\text{new}} &:= \text{normalize}(\mathcal{D}_{\text{new}}).\end{aligned}$$

3）**在大规模情况下，采用软阈值非线性函数:**

$$(f(x;\mathcal{D},\alpha) = \max\{0, \mathcal{D}^{\top}x - \alpha\}$$

4）构建深层网络：

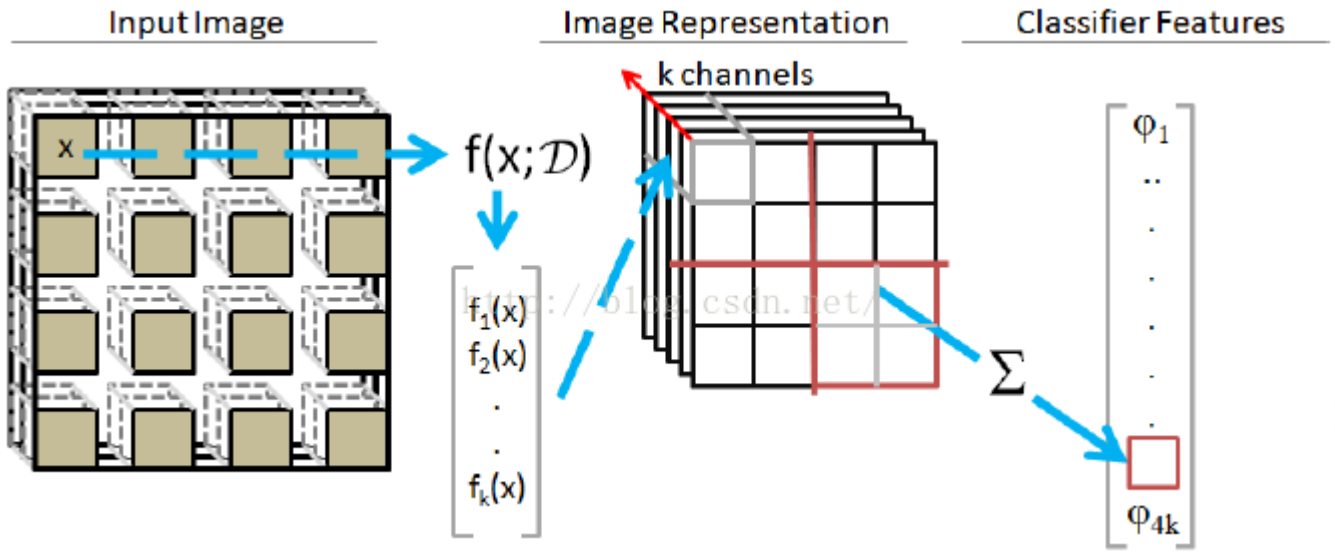
先从一些无标签数据中随机提取一些patches，然后用k-means来学习其中的特征，得到一个字典。然后对于一个大的图像，用这个字典（特征提取器）去卷积图像的每个局部感受野，从而提取整幅图像的特征。最后再通过一个pooling步骤去对提取到的特征进行绛维

对无标签数据集X中，通过上面的方法得到pooling后的特征后，然后得到新的数据集Z，再在这个数据库Z中用同样的特征学习过程去学习新的特征。从而得到第二层的特征

NobleProg

图片识别应用

给定一堆无标签图片，通过K-means算法，进行学习字典D。接着就要知道它具体是怎么用于图片识别分类的



1)通过无监督学习，学习到一系列的字典D
2)接着利用学习到的字典，把一张输入图片，进行函数映射：$f : \mathcal{R}^n \rightarrow \mathcal{R}^k$

可以把这个过程看成是卷积，比较容易理解。这个映射过程不一定是非重叠的
3)采用pooling进行降维
4)如果是构建多层网络，那么就重复上面的过程。如果单层网络，那么后面接一个分类器，比如svm

NobleProg

```
from __future__ import print_function
import numpy as np
import tensorflow as tf
from tensorflow.contrib.factorization import KMeans
# Ignore all GPUs, tf random forest does not benefit from it.
import os
os.environ["CUDA_VISIBLE_DEVICES"] = ""
from tensorflow.examples.tutorials.mnist import input_data   # Import MNIST data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
full_data_x = mnist.train.images
# Parameters
num_steps = 50          # Total steps to train
batch_size = 1024       # The number of samples per batch
k = 25                  # The number of clusters
num_classes = 10        # The 10 digits
num_features = 784      # Each image is 28x28 pixels
X = tf.placeholder(tf.float32, shape=[None, num_features]) # Input images
Y = tf.placeholder(tf.float32, shape=[None, num_classes]) # Labels (for assigning a label to a centroid and testing)
# K-Means Parameters
kmeans = KMeans(inputs=X, num_clusters=k, distance_metric='cosine',use_mini_batch=True)
# Build KMeans graph
(all_scores, cluster_idx, scores, cluster_centers_initialized, init_op,train_op) = kmeans.training_graph()
cluster_idx = cluster_idx[0]       # fix for cluster_idx being a tuple
avg_distance = tf.reduce_mean(scores)
```

```
init_vars = tf.global_variables_initializer() # Initialize the variables (i.e. assign their default value)
sess = tf.Session() # Start TensorFlow session
sess.run(init_vars, feed_dict={X: full_data_x}) # Run the initializer
sess.run(init_op, feed_dict={X: full_data_x})
for i in range(1, num_steps + 1): # Training
    _, d, idx = sess.run([train_op, avg_distance, cluster_idx], feed_dict={X: full_data_x})
    if i % 10 == 0 or i == 1:
        print("Step %i, Avg Distance: %f" % (i, d))
counts = np.zeros(shape=(k, num_classes))
for i in range(len(idx)):
    counts[idx[i]] += mnist.train.labels[i]
abels_map = [np.argmax(c) for c in counts] # Assign the most frequent label to the centroid
labels_map = tf.convert_to_tensor(labels_map)
cluster_label = tf.nn.embedding_lookup(labels_map, cluster_idx) # Evaluation ops, Lookup: centroid_id -> label
correct_prediction = tf.equal(cluster_label, tf.cast(tf.argmax(Y, 1), tf.int32)) # Compute accuracy
accuracy_op = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
test_x, test_y = mnist.test.images, mnist.test.labels  # Test Model
print("Test Accuracy:", sess.run(accuracy_op, feed_dict={X: test_x, Y: test_y}))
```

```
C:\Anaconda3\python.exe E:/pywork/TensorFlow-Ex
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
Step 1, Avg Distance: 0.341471
Step 10, Avg Distance: 0.221609
Step 20, Avg Distance: 0.220328
Step 30, Avg Distance: 0.219784
Step 40, Avg Distance: 0.219427
Step 50, Avg Distance: 0.219164
Test Accuracy: 0.7127

Process finished with exit code 0
```

K-means-2

NobleProg

- 1、需要先对数据进行均值和对比度的归一化
- 2、使用白化来消去数据的相关性。注意白化参数的选择
- 3、通过高斯噪声和归一化来初始化k-means的聚类中心
- 4、使用抑制更新来避免空聚类和增加稳定性
- 5、注意数据的维数和稀疏的影响。K-means通过寻找数据分布的”heavy-tailed”方向来找到输入数据的稀疏投影。但如果数据没有经过适当的白化，或者数据维数太高了，或者存在无效的数据，那么效果往往是不好的
- 6、对于高维数据，k-means需要增加大量的样本来能得到好的效果
- 7、一些外在的参数（例如pooling，编码方法的选择等等）对于系统性能的影响比学习算法本身要大。所以最好先化时间和成本在通过更多的交叉检验的方法来挑选好的参数比花更多的成本在学习算法上要明智的多。
- 8、对于图像识别来说，更多的聚类中心往往是有帮助的。当然，前提是我们要有足够的训练样本。
- 9、当有标签数据多的时候，采用一个简单的编码器。如果标签数据有限（例如每类只有百来个训练样本），那么采用一个复杂的编码器会更好。
- 10、尽可能地使用局部感受野。K-means是否成功，其瓶颈在于输入数据的维数。越低越好。如果无法手动的选择好的局部感受野，那么尝试去通过一个自动的依赖性测试dependency test来帮助从数据中挑选出一组低维的数据。这对于深度网络来说是很有必要的

2. 稀疏滤波Sparse Filtering

- sparse filtering的核心思想就是避免对数据分布的显式建模，而是优化特征分布的稀疏性从而得到好的特征表达
- 引入一个特征分布矩阵，矩阵的每一行是一个特征，每一列是一个样本。每个元素$f_j^{(i)}$表示第i个样本的第j个特征的激活值：
- 1）每个样本的特征应该是稀疏的（Population Sparsity）
- 2）样本间的特征应该是稀疏的（Lifetime Sparsity）
- 3）特征的分布应该是均匀的（High Dispersal）：如果提取到相同的特征，那么特征既冗余，又没有增加信息量，所以一般都要求提取到的特征是正交的

Examples

Feature Values

	x_1	x_2	x_3	x_4	...	x_m
f_1	0.5	2	0	1.5	...	4
f_2	0	0	2.5	0	...	0
...			...			
f_{99}	3.2	0	1.6	0.3	...	1
...			...			
f_n	4	0.5	0	1	...	3

Sparse Filtering Objective Function

1. Normalize across rows
2. Normalize across columns
3. Cost Function = Sum of the normalized entries

$$M_{ij} = |w_i^T x_j|$$

$$f_j^{(i)} = \log(1 + (\mathbf{w_j^T x^{(i)}})^2)$$

$$f_j^{(i)} = \mathbf{w}_j{}^T \mathbf{x}^{(i)}$$

$$\tilde{\mathbf{f}}_j = \mathbf{f}_j / \|\mathbf{f}_j\|_2$$

$$\hat{\mathbf{f}}^{(i)} = \tilde{\mathbf{f}}^{(i)} / \|\tilde{\mathbf{f}}^{(i)}\|_2$$

$$\text{minimize} \quad \sum_{i=1}^{M} \left\| \hat{\mathbf{f}}^{(i)} \right\|_1 = \sum_{i=1}^{M} \left\| \frac{\tilde{\mathbf{f}}^{(i)}}{\|\tilde{\mathbf{f}}^{(i)}\|_2} \right\|_1$$

$$s^{(i)} = \left[\sum_j \tilde{f}_j^{(i)} / F\right]^2 / \left[\sum_j (\tilde{f}_j^{(i)})^2 / F\right]$$

NobleProg

1）Optimizing for population sparsity: 最小化 $||f^{(i)}||1$ ，将会驱使归一化的特征趋于稀疏和大部分接近于0 。也就是一些特征会比较大，其他的特征值都很小（接近于0）。这个目标函数会优化特征的population sparsity

2）Optimizing for high dispersal:首先通过将每个特征除以它在所有样本上面的二范数来归一化每个特征，使他们具有相同的激活值 $f_j=f_j/||f_j||_2$

3）Deep sparse filtering: 因为sparse filtering的目标函数是不可知的，可以自由地选择一个前向网络来计算这些特征。一般来说，都使用比较复杂的非线性函数，或者多层网络来计算这些特征。这样，sparse filtering也算是训练深度网络的一种自然的框架了；有sparse filtering的深度网络可以使用权威的逐层贪婪算法来训练。可以先用sparse filtering来训练得到一个单层的归一化的特征，然后用它当成第二层的输入去训练第二层，其他层一样

4）与divisive normalization的联系：sparse filtering把divisive normalization结合到了特征学习过程中，通过让特征间竞争，学习到满足population sparse的特征表达

5）与ICA 和sparse coding的联系：sparse filtering（归一化的稀疏惩罚）的目标函数可以看出ICA（滤波器间相互正交的约束）目标函数的归一化版本

采用软绝对值函数作为我们的激活函数

$$f_j^{(i)} = \sqrt{\epsilon + (\mathbf{w}_j^T \mathbf{x}^{(i)})^2} \approx |\mathbf{w}_j^T \mathbf{x}^{(i)}|$$

其中ε=10^-8，然后用现有的L-BFGS算法来优化sparse filtering的目标函数直至收敛

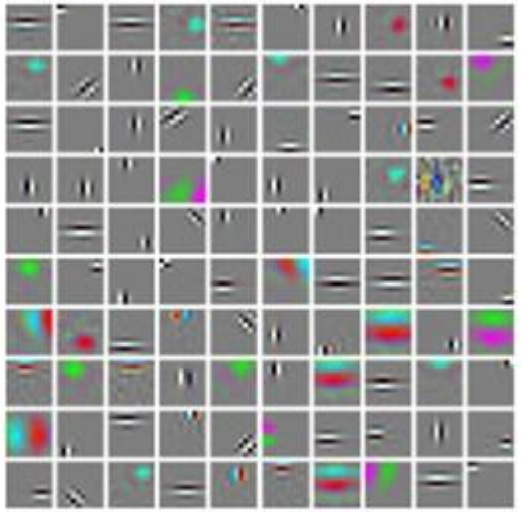


Figure 4: A subset of the learned filters from 10×10 patches extracted from the STL dataset.

plot_sparse_coding.py

NobleProg

3. 单层非监督学习网络

- 1、通过以下步骤去学习一个特征表达：

  1）从无标签的训练图像中随机提取一些小的patches；

  2）对这些patches做预处理（每个patch都减去均值，也就是减去直流分量，并且除以标准差，以归一化。对于图像来说，分别相当于局部亮度和对比度归一化。然后还需要经过白化）；

  3）用非监督学习算法来学习特征映射，也就是输入到特征的映射函数

- 2、学习到特征映射函数后，给定一个有标签的训练图像集，我们用学习到的特征映射函数，对其进行特征提取，然后用来训练分类器：

  1）对一个图像，用上面学习到的特征来卷积图像的每一个sub-patches，得到该输入图像的特征；

  2）将上面得到的卷积特征图进行pooling，减少特征数，同时得到平移等不变性；

  3）用上面得到特征，和对应标签来训练一个线性分类器，然后在给定新的输入时，预测器标签

- 3、特征学习：

  1）sparse auto-encoders：用BP去训练一个K个隐藏节点的自动编码机，代价函数是重构均方误差，并存在一个惩罚项。主要限制隐藏节点，使其保持一个低的激活值。算法输出一个权值矩阵W（KxN维）和一组基B（K维），特征映射函数为：f(x)=g(Wx+b)。这里g(z)=1/(1+exp(-z))是sigmoid函数，对向量z的每个元素求值。

NobleProg

- 2）sparse Restricted Boltzmann machine：RBM是一个无向图模型，包含K个二值隐层随机变量。稀疏RBMs可以通过contrastive divergence approximation（对比分歧近似）方法来训练。其中的稀疏惩罚项和自动编码机一样。训练模型还是输入权值矩阵W和基b，我们也可以使用和上面的自动编码机同样的特征映射函数。但它的训练方法和自动编码机是完全不一样的
- 3）K-means clustering：用K-means聚类算法来从输入数据中学习K个聚类中心$c^{(k)}$。当学习到这K个聚类中心后，我们可以有两种特征映射f的方法。第一种是标准的1-of-K，属于硬分配编码；
- 第二种是采用非线性映射，属于软编码 $f_k(x) = \max\{0, \mu(z) - z_k\}$ $f_k(x) = \begin{cases} 1 & \text{if } k = \arg\min_j \|c^{(j)} - x\|_2^2 \\ 0 & \text{otherwise.} \end{cases}$
-
- 4）Gaussian mixtures：高斯混合模型GMM用K个高斯分布的混合来描述了输入数据的密度分布。GMMs可以通过EM算法来训练。一般来说需要先运行一次K-means算法来初始化GMM的K个高斯分量。这样可以避免其陷入较差的局部最小值 $f_k(x) = \phi_k \frac{1}{(2\pi)^{d/2}|\Sigma_k|^{1/2}} \exp\left(-\frac{1}{2}(x - c^{(k)})^\top \Sigma_k^{-1}(x - c^{(k)})\right)$
- Σ_k是对角协方差矩阵，Φk是由EM学习得到的每个类的先验概率。其实这里和上面k-means的软分配有点像。对每个样本x都要计算其属于每一个类别j的概率。然后用这些后验概率来编码对应x的特征向量
- 4、**特征提取和分类：**通过上面的步骤，就可以得到了将一个输入patch x（N维）映射到一个新的描述y=f(x)(K维)的函数f。这时候就可以用这个特征提取器来提取有标签图像数据的特征来训练分类器了

NobleProg

4. Multi-Stage多级架构

- 在当前的很多目标识别系统中，特征提取阶段一般由一组滤波器，再进行非线性变换和一些类型的特征pooling层组成。大部分系统都使用一级特征提取，这时候的滤波器是hard-wired（人工选择的，硬连线的，参数不可学习调整）的，或者使用两级，这时候其中的一级或者两级的滤波器都可以通过监督或者非监督的方式学习得到
- 1)滤波器组后面接的non-linearities非线性算法是如何影响识别的准确性的？
- 2)通过监督或者非监督方式学习到的滤波器组是否比随机的滤波器组或者人工指定的滤波器要好？
- 3)与仅有一级的特征提取对比，两级的特征提取是否还有其他优点？
- 结论：

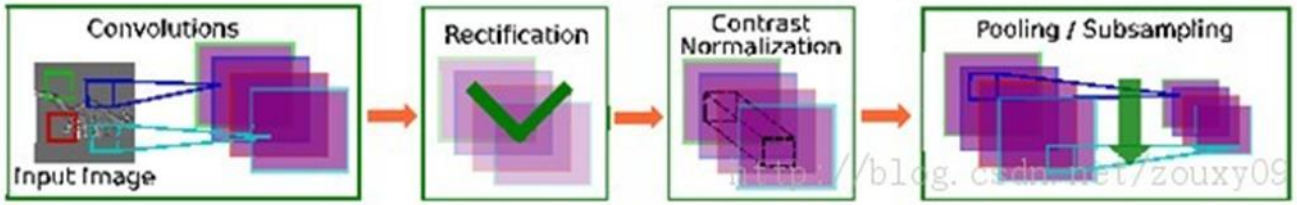
1)使用包含校正和局部对比度归一化的非线性算子对增加目标识别的准确性来说有很大帮助

2)两级的特征提取比一级的要好，准确率更高。

3)惊喜的是，用随机初始化的滤波器组的两级系统却可以在Caltech这个数据中达到63%的识别率。当然，这里面包含了合适的非线性算子和pooling层

4)经过监督微调，系统在NORB数据库上达到当前领先水平。而且非监督预训练后再加监督微调可以在Caltech这个数据库中达到更好的准确率(大于63%)。然后在没有处理过的MNIST数据库中，可以达到目前我们知道的最低的0.53%的错误率

- 输入经过一个滤波器组filter bank（一般是基于方向性的边缘检测器），再经过一个非线性算子non-linear operation（quantization, winner-take-all, sparsification, normalization, and/or point-wise saturation），然后用一个pooling操作（把实空间或者特征空间邻域的值通过一个max, average, or histogramming operator）来绛维和得到一定的不变性。例如熟知的SIFT特征，它先通过对每个小patch经过方向性边缘检测器，然后用winner-take-all算子来获取最显著的方向。最后，在更大块的patch上面统计局部方向的直方图，pooling成一个稀疏向量
- 有一个或者多个特征提取层、滤波器组后使用的非线性算子、滤波器组的得到(人工选择、非监督学习还是监督学习)和顶层的分类器的选择(线性分类器还是更复杂的分类器)
- 一般对滤波器组的选择是Gabor小波，还有选择一些简单的方向性检测滤波器组，也就是梯度算子，例如SIFT和HOG
- 分级通过堆叠一个或者多个特征提取阶段，每个阶段包括一个滤波器组合层、非线性变换层和一个pooling层，pooling层通过组合(取平均或者最大的)局部邻域的滤波器响应，因而达到对微小变形的不变性



NobleProg

- 1、滤波器组层Filter Bank Layer-F_{CSG}：包括三部分：一组卷积滤波器C、再接一个sigmoid/tanh非线性变换函数S，然后是一个可训练的增益系数G，运算：

$$y_j = g_j \tanh(\sum k_{ij} * x_i)$$

- 2、校正层Rectification Layer-R_{abs}：只是简单的一个取绝对值的操作，除了绝对值算子外，其他的非线性算子，产生的效果差不多
- 3、局部对比度归一化层Local Contrast Normalization Layer-N：主要进行的是局部做减和做除归一化，它会迫使在特征map中的相邻特征进行局部竞争，还会迫使在不同特征maps的同一空间位置的特征进行竞争。在一个给定的位置进行减法归一化操作，实际上就是该位置的值减去邻域各像素的加权后的值，权值是为了区分与该位置距离不同影响不同，权值可以由一个高斯加权窗来确定。除法归一化实际上先计算每一个特征maps在同一个空间位置的邻域的加权和的值，然后取所有特征maps这个值的均值，然后每个特征map该位置的值被重新计算为该点的值除以max(那个均值，该点在该map的邻域的加权和的值)。分母表示的是在所有特征maps的同一个空间邻域的加权标准差
- 4、平均池化和子采样层Average Pooling and Subsampling Layer -P_A：使得提取的特征对微小变形鲁棒，和视觉感知中的复杂细胞的角色差不多。采样窗口所有值取平均得到下采样层的值
- 5、最大值池化和子采样层Max-Pooling and Subsampling Layer -P_M：可以用任何一种对称的pooling操作实现对提取的特征的平移不变性。最大池与平均池相似，只是最大取代了平均。一般来说，池化窗口是不重叠的

NobleProg

MLCPS（Multi-Level Cache Processing System）

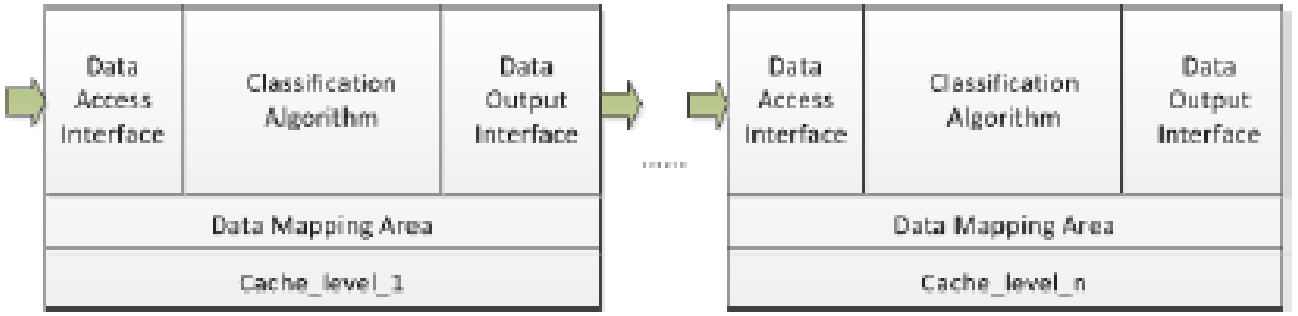
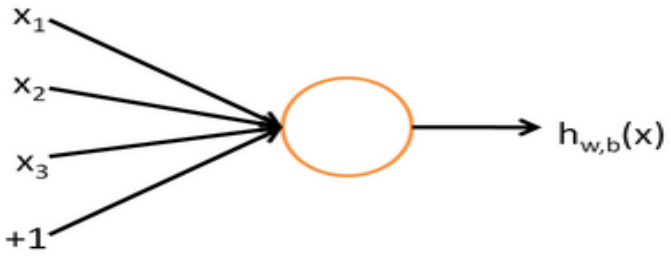


Fig. 1. The image *MLCPS* framework

与浅层学习相比，深度学习具有的优点：
1）深度结构的神经网络能够有效表征复杂的目标函数
2）在表达相同的函数时，浅结构神经网络所需的计算因子是深度学习神经网络的指数级，因此深度结构神经网络大大降低了计算复杂度。同时因为需要的计算因子减少，提高了网络的泛化能力
3）深度学习通过模拟大脑皮层，采用分层处理的方式处理数据，神经网络的每一层能提取输入数据不同水平的特征，逐层建立从底层信号到高层特征的映射

NobleProg

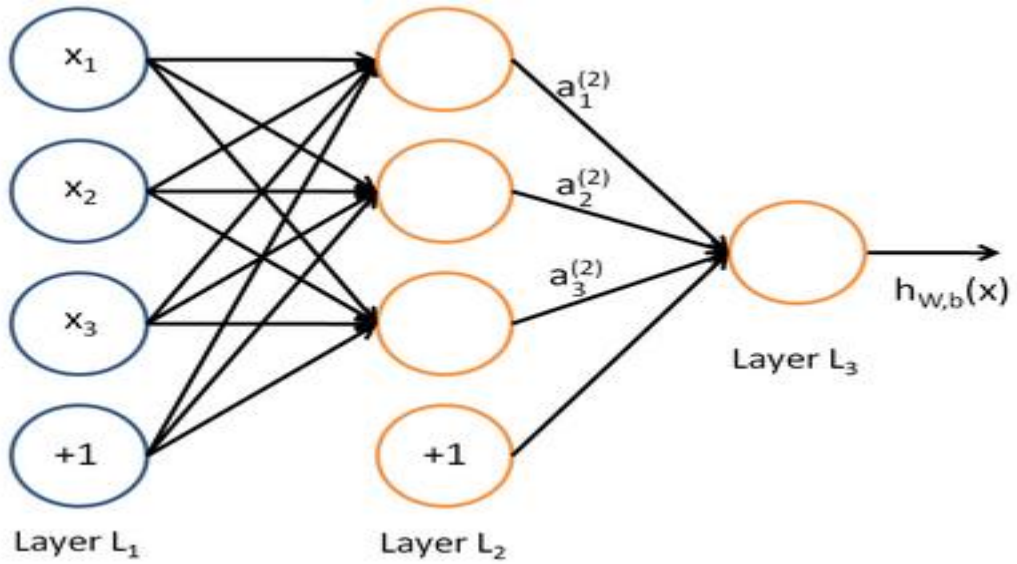
5. DNN深度神经网络



$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

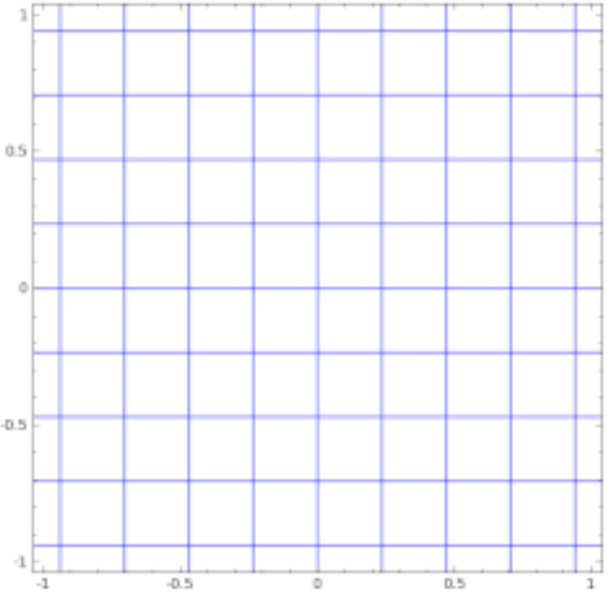
1）该单元称作Logistic回归模型

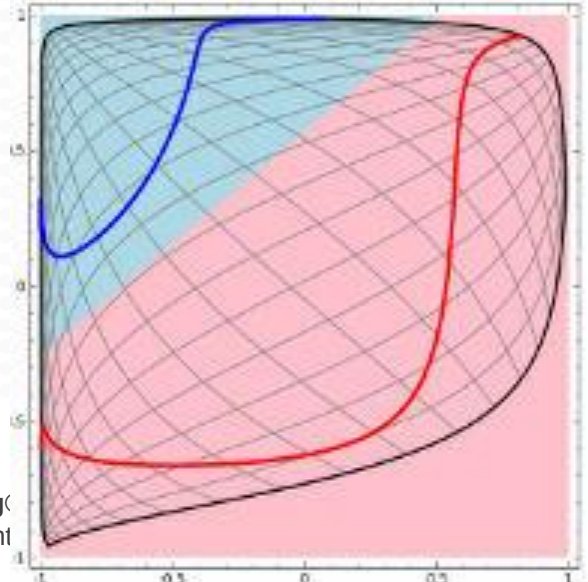
2）当将多个单元组合起来并具有分层结构时，就形成神经网络模型

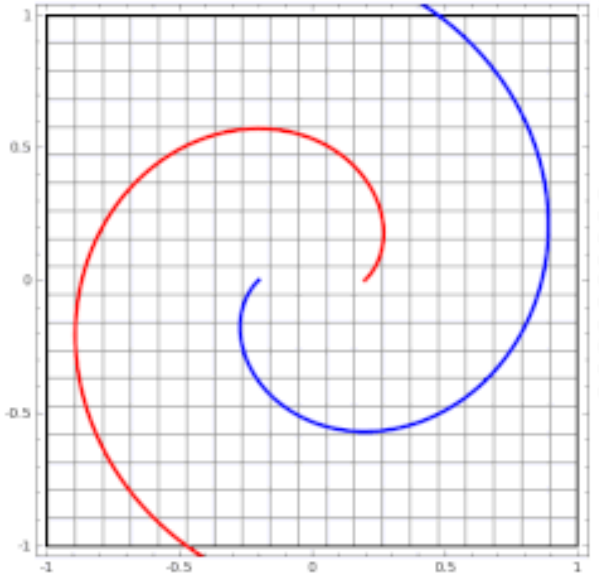


$$a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)})$$

3）通过如下5种对输入空间（输入向量的集合）的操作，完成 **输入空间 —> 输出空间** 的变换 (矩阵的行空间到列空间)
升维/降维、放大/缩小、旋转、平移、弯曲这，前三操作由w.x完成，4操作由+b完成，5操作由f来实现







神经网络的训练

- 通过**比较**当前网络的**预测值**和我们真正想要的**目标值**，再根据两者的差异情况来更新每一层的权重矩阵（比如网络的预测值高了，就调整权重让它预测低一些，不断调整，直到能够预测出目标值
- 需要先定义“如何比较预测值和目标值的差异”，这便是损失函数或目标函数（loss function or objective function），用于衡量预测值和目标值的差异的方程。loss function的输出值loss越高表示差异性越大。那神经网络训练就变成尽可能的缩小loss的过程
- 所用的方法是梯度下降Gradient descent：通过使loss值向当前点对应梯度的反方向不断移动，来降低loss。一次移动多少是由学习速率learning rate来控制的

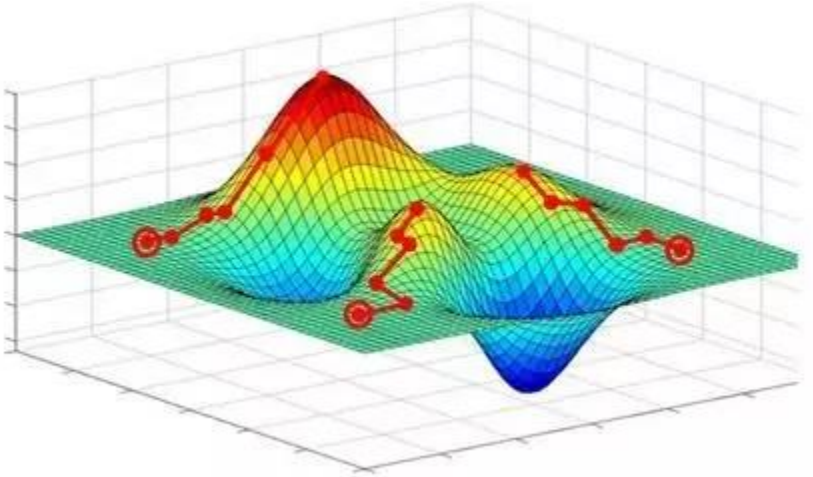
梯度下降的问题:

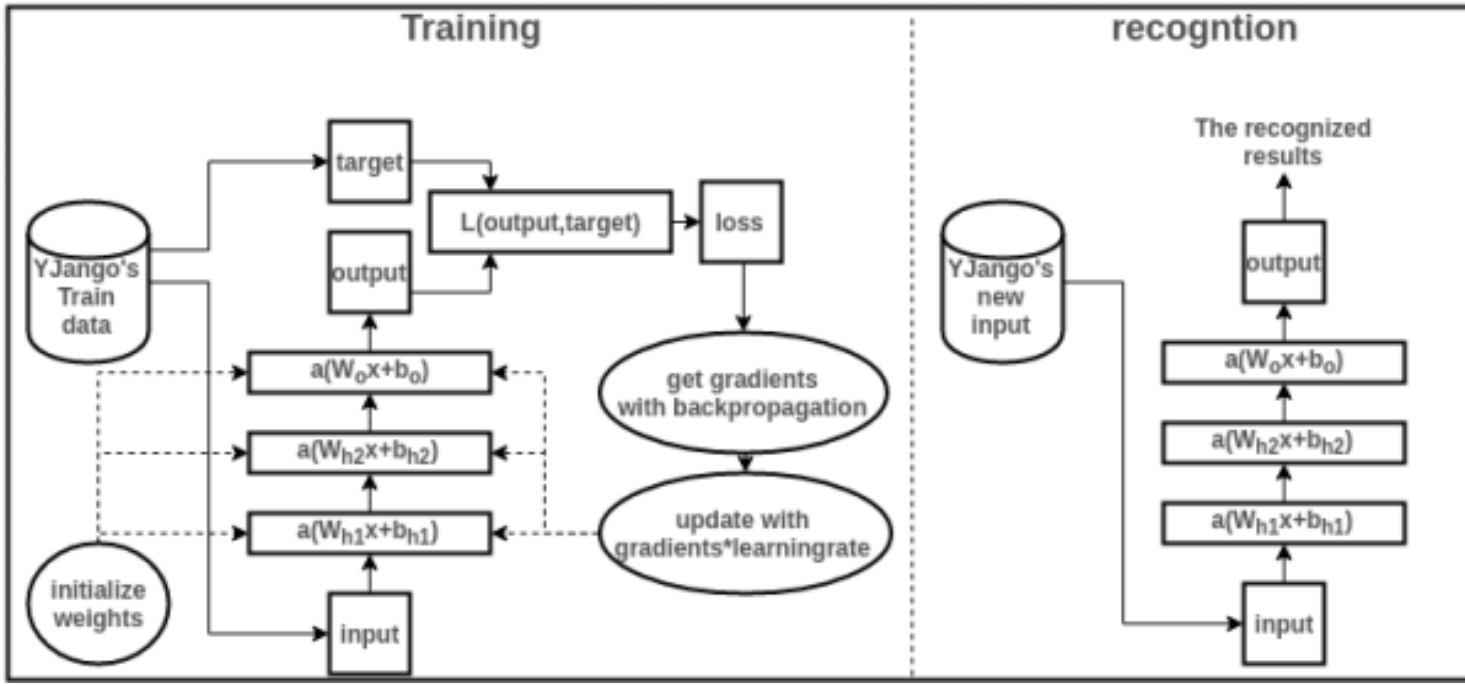
1) 局部极小值: 调节步伐

--随机梯度下降SGD\小批量梯度下降\动量\Adagrad、Adadelta、Adam\优化起点

2)梯度的计算:如何快速计算梯度、如何更新**隐藏层**的权重？

--计算图：**反向传播算法**





训练和识别过程

NobleProg

- 1）收集训练集train data：input+label--sample
- 2）设计网络结构architecture：确定层数、每一隐藏层的节点数和激活函数，输出层的激活函数和损失函数
  - 训练数据：$input \in R^{39}; label \in R^{48}$
  - 权重矩阵：$W_{h1} \in R^{1000x39}; W_{h2} \in R^{1000x1000}; W_o \in R^{48x1000}$
  - 偏移向量：$b_{h1} \in R^{1000}; b_{h2} \in R^{1000}; b_o \in R^{48}$
  - 网络输出：$output \in R^{48}$

3）数据预处理preprocessing：中心化mean subtraction、归一化normalization、主成分分析PCA、白化whitening

4）权重初始化weights initialization：wh2\wh1\w0初始化决定了loss在loss function中从哪个点开始作为起点训练网络。可用均匀分布初始权重Uniform distribution

5）训练网络training：训练过程就是用训练数据的input经过网络计算出output，再和label计算出loss，再计算出gradients来更新weights的过程

5.1）正向传递：y，算当前网络的预测值

$$output = linear(W_o \cdot Relu(W_{h2} \cdot Relu(W_{h1} \cdot input + b_{h1}) + b_{h2}) + b_o)$$

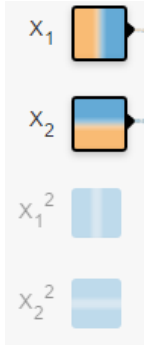
5.2）计算loss：

5.3）计算梯度：从loss开始反向传播计算每个参数对应的梯度。可用$loss = mean((output - target)^2)$新所计算的梯度都是从一个样本计算出来的。Gradient Descent是正向传递所有样本来计算梯度

5.4）更新权重：$W = W - learningrate * gradient$

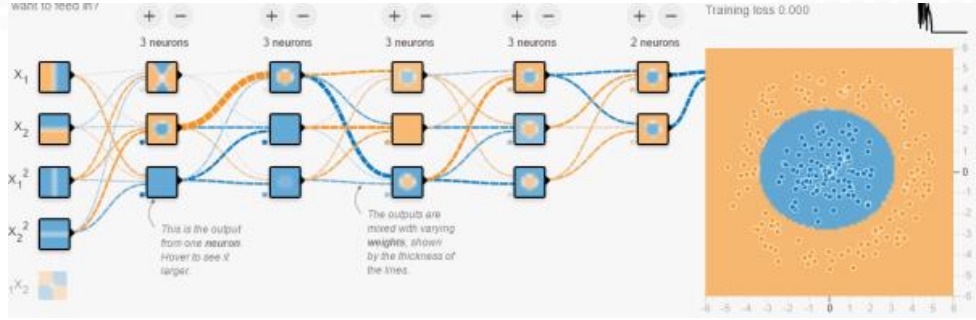
5.5）预测新值：训练过过所有样本后，打乱样本顺序再次训练若干次。训练完毕后，当再来新的数据input，就可以利用训练网络来预测

1）数据

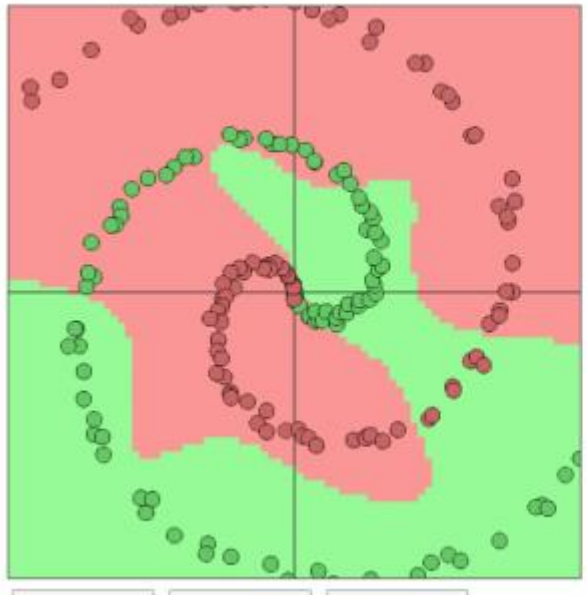


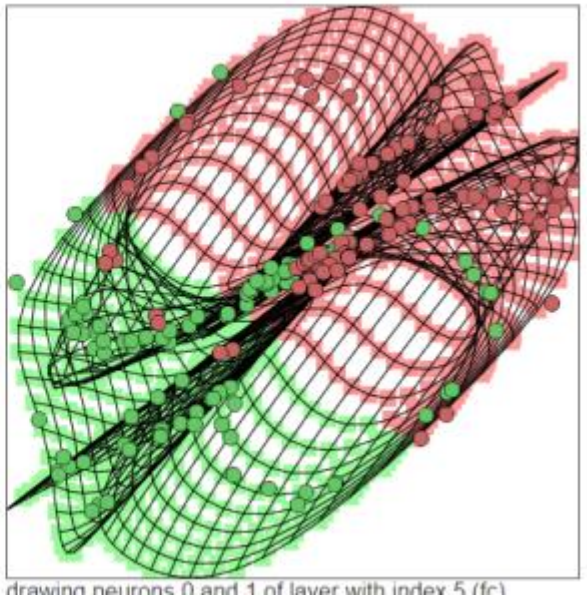
2）输入

3）连接线

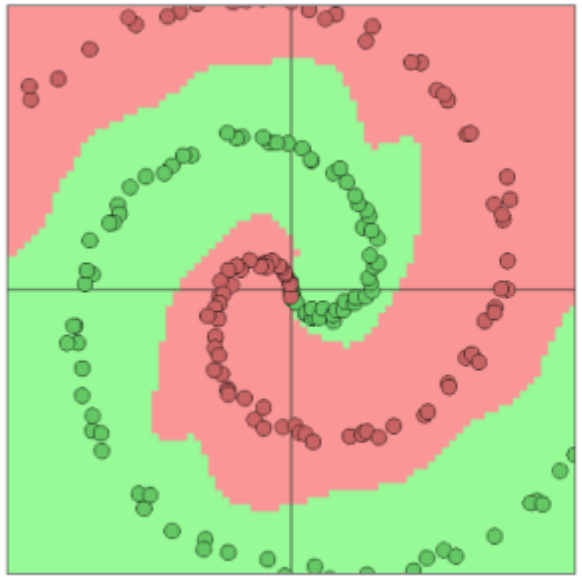


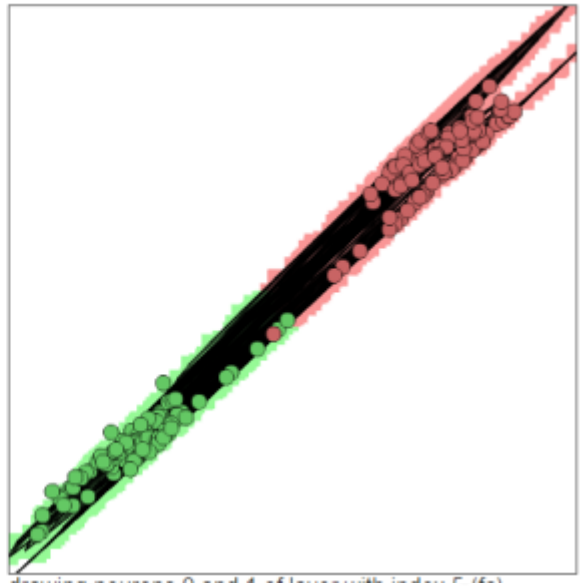
4）输出





左侧是原始输入空间下的分类图，右侧是转换后的高维空间下的扭曲图

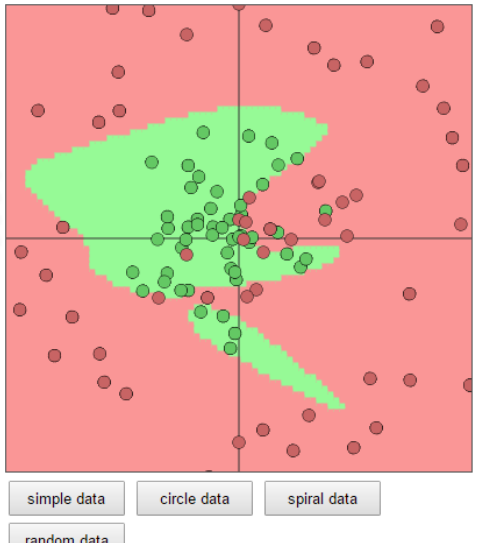


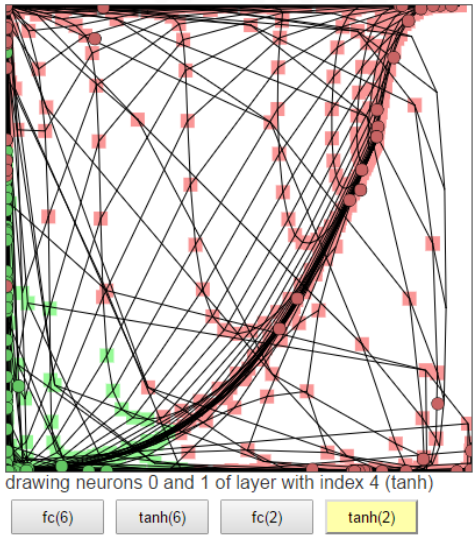


最终扭曲效果是所有绿点都被扭曲到一侧，而所有红点都被扭曲到另一侧。这样就可以线性分割（用超平面在中间分开两类）

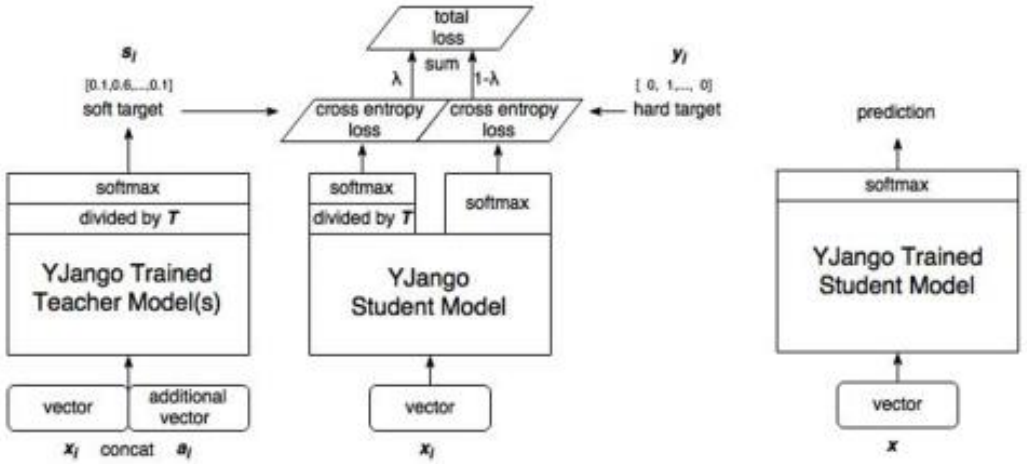
1）深层神经网络就是拥有更多层数的神经网络
2）为什么更深的网络会更加容易识别，增加容纳变异体（variation）（红苹果、绿苹果）的能力、鲁棒性（robust）
3）数学视角：变异体variation很多的分类的任务需要高度非线性的分割曲线。不断的利用5种空间变换操作将原始输入空间像“捏橡皮泥一样”在高维空间下捏成更为线性可分/稀疏的形状
4）物理视角：通过对“抽象概念”的判断来识别物体，而非细节。如对飞机的判断，即便人类自己也无法用语言或者若干条规则来解释自己如何判断一个飞机。因为人脑中真正判断的不是是否“有机翼”、“能飞行”等细节现象，而是一个抽象概念。层数越深，这种概念就越抽象，所能涵盖的变异体就越多，就可以容纳战斗机，客机等很多种不同种类的飞机

http://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html





蒸馏模型 Distillation



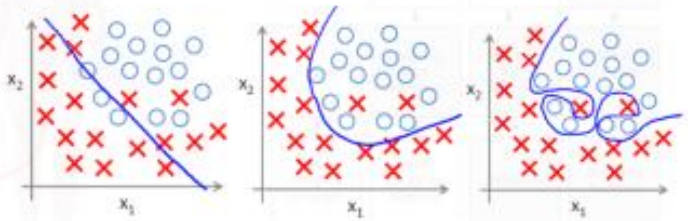
详细参考：点我

本质属于迁移学习，但使用的是不同的迁移方式

抑制过拟合 Overfitting

dropout（遗忘）：
细节有时也能形成规律，但不会每次都形成。
遗忘可以去掉那些偶然形成的细节规律，提高普遍性。

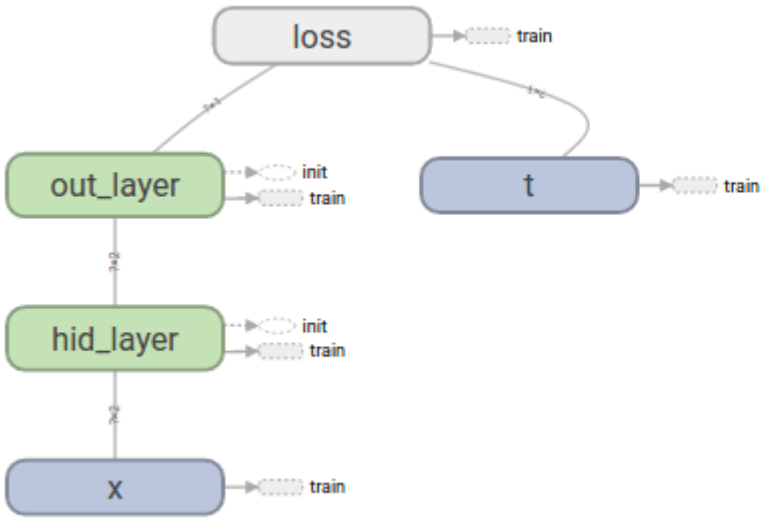
shuffle（乱序）：
训练的样本不要有固定顺序，而要随机打乱，
同样可以抑制偶然形成的细节规律。
比如不要一直从abandon开始背单词一样。

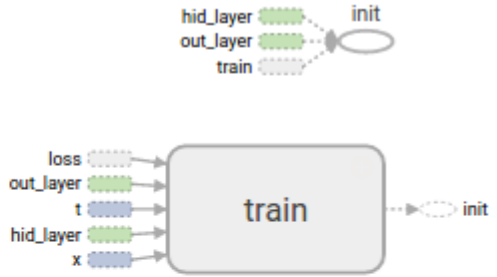


L2 regularization（保持最简化）：
解决的方案不要过于复杂。
不然只能顾及特例而失去普遍性。

mini-batch（多题一起做）：
相互比较后得出结论。
比如同时看两本描述不同的书可以得到更好地理解。

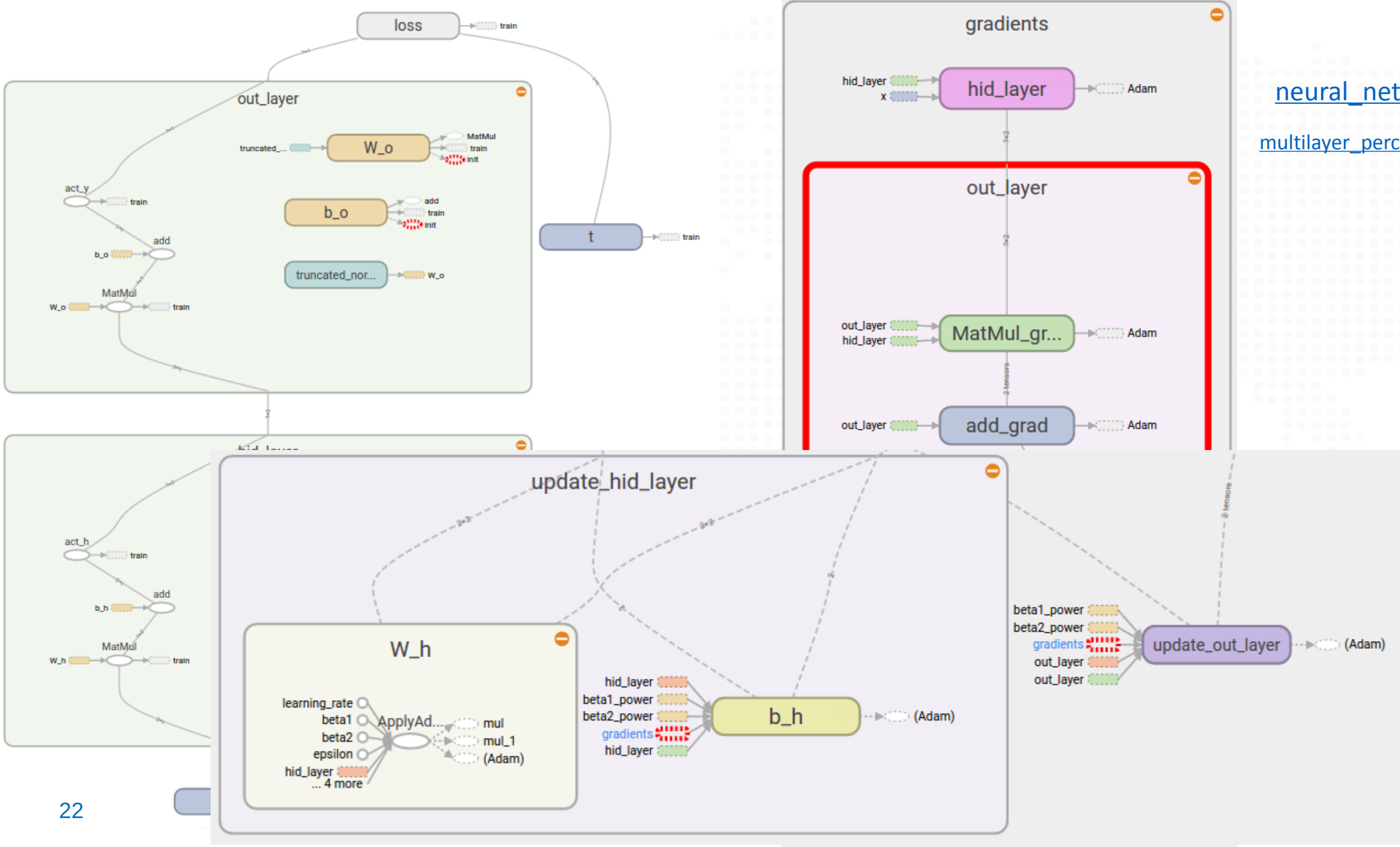
noisy layer（加噪音）：
题目加入一些干扰项、改变考前环境、教室、平时状态等。





TensorFlow网络结构

NobleProg



[neural_network.py]

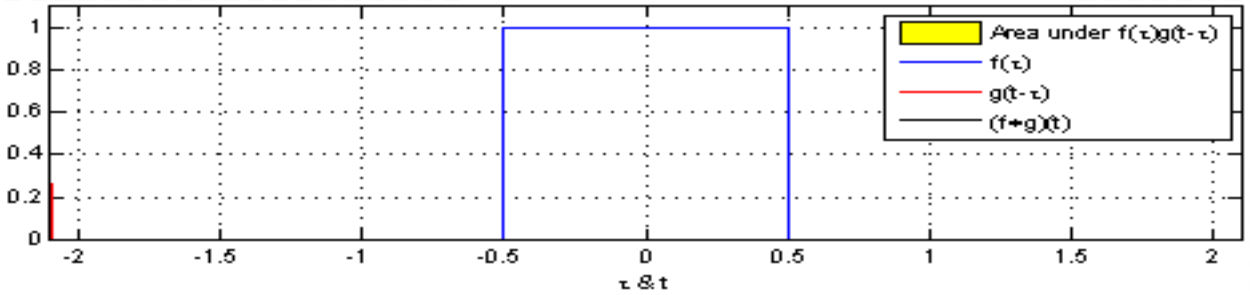
[multilayer_perceptron.py]

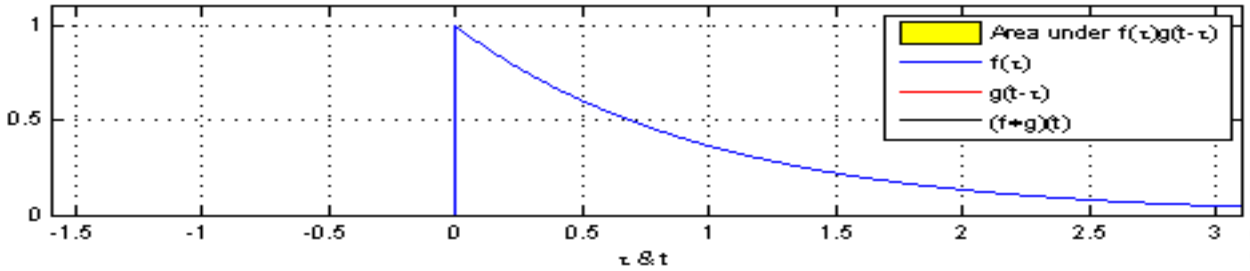
6. CNN卷积神经网络

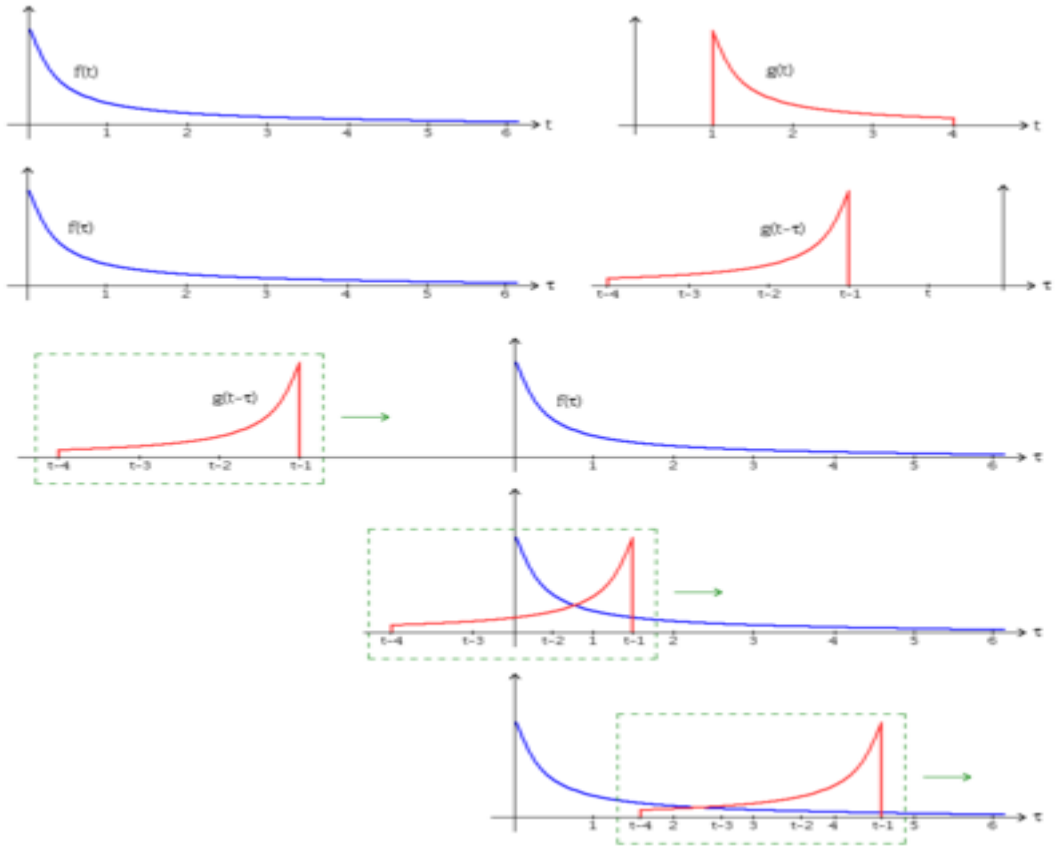
- 6.1 卷积Convolution
- 6.2 卷积神经网络CNN
- 6.3 实现与应用

NobleProg

6.1.1 卷积的数学过程







$$(f * g)(t) \ \stackrel{\text{def}}{=} \int_{\mathbb{R}^n} f(\tau) g(t - \tau)\, d\tau.$$

$$(f * g)[n] = \sum_{m=-M}^{M} f[n - m] g[m].$$

$$f(x, y) * g(x, y) = \int_{\tau_1=-\infty}^{\infty} \int_{\tau_2=-\infty}^{\infty} f(\tau_1, \tau_2) \cdot g(x - \tau_1, y - \tau_2)\, d\tau_1 d\tau_2$$

$$f[x, y] * g[x, y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$

..obleProg

6.1.2 卷积-滤波filter

Figure: Input image; Mean image with 3×3 kernel and Mean filter with 5×5 kernel.

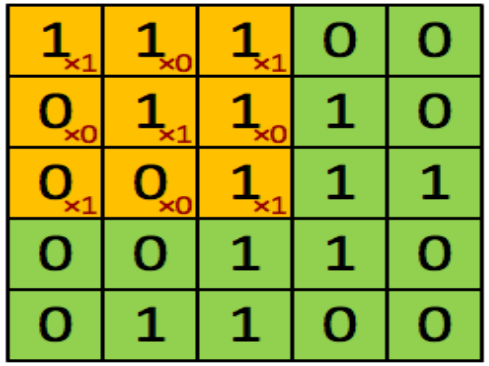
低通滤波，核称为滤波器，整个操作过程称为卷积

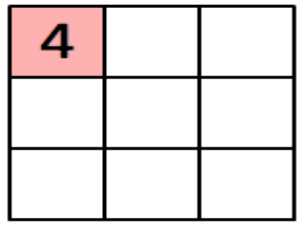
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

$\frac{1}{273}$

1	4	7	4	1
4	16	26	16	4
7	26	41	26	7
4	16	26	16	4
1	4	7	4	1

高斯滤波



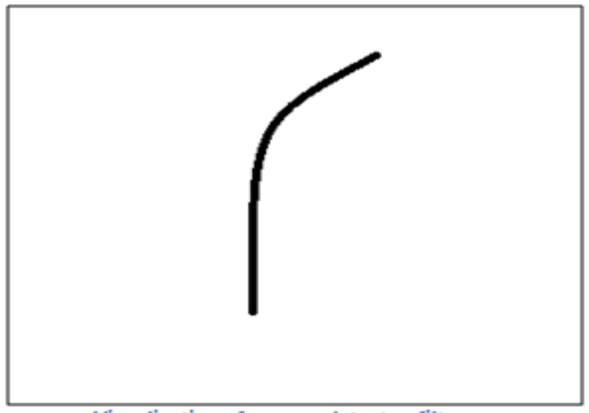


NobleProg

6.1.3卷积-激活响应

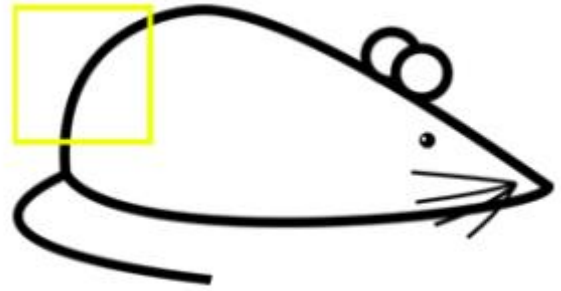
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

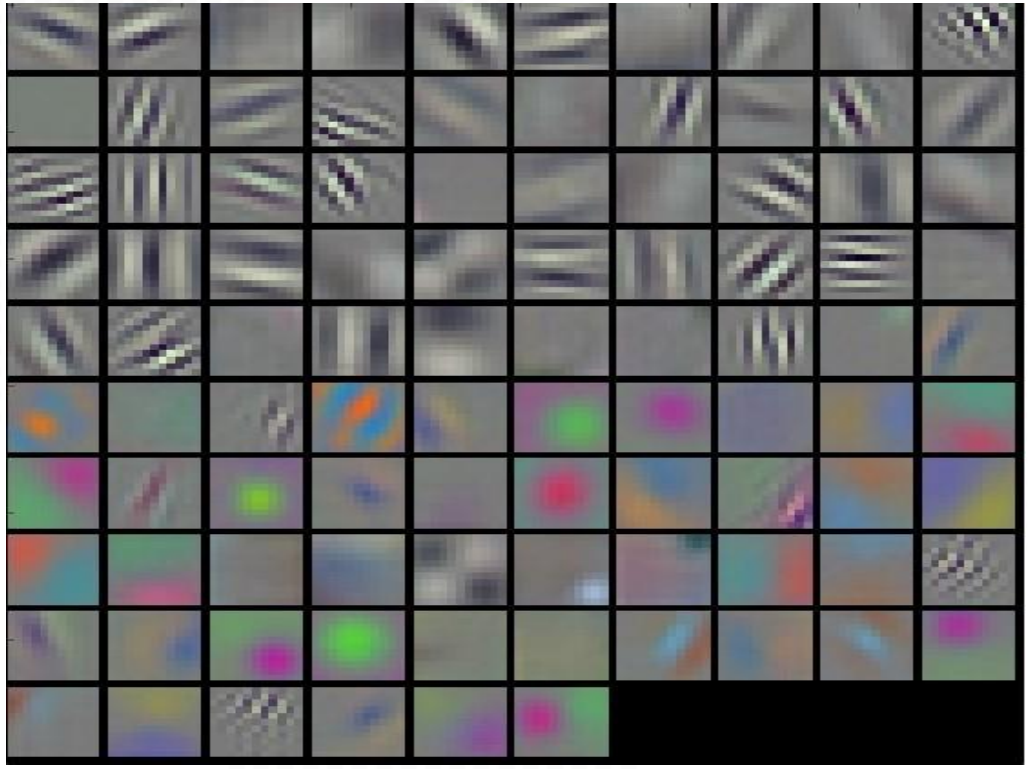
0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

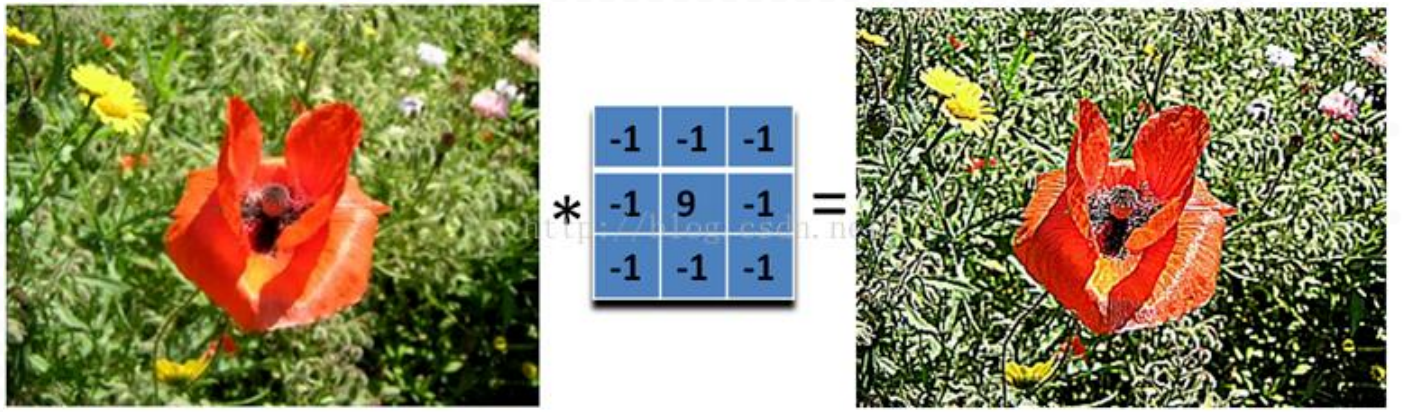
Pixel representation of filter

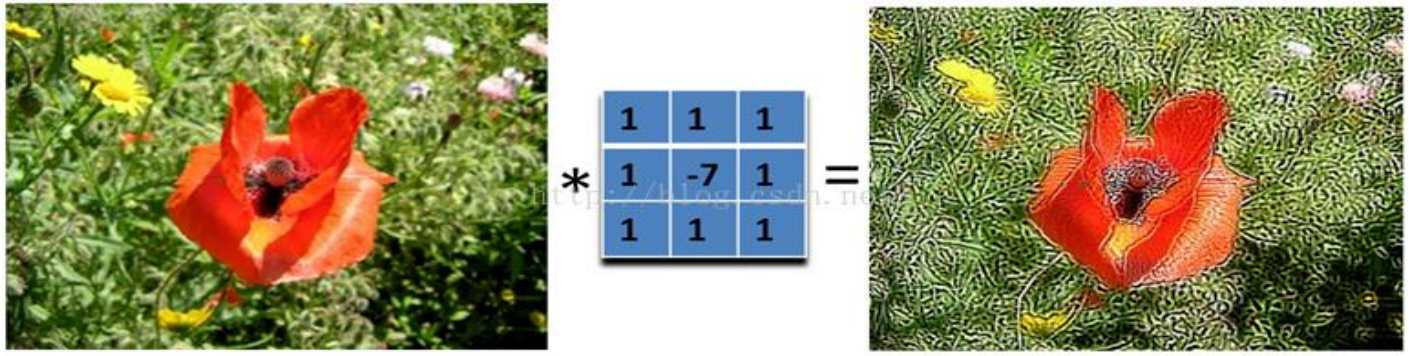
Multiplication and Summation = 0

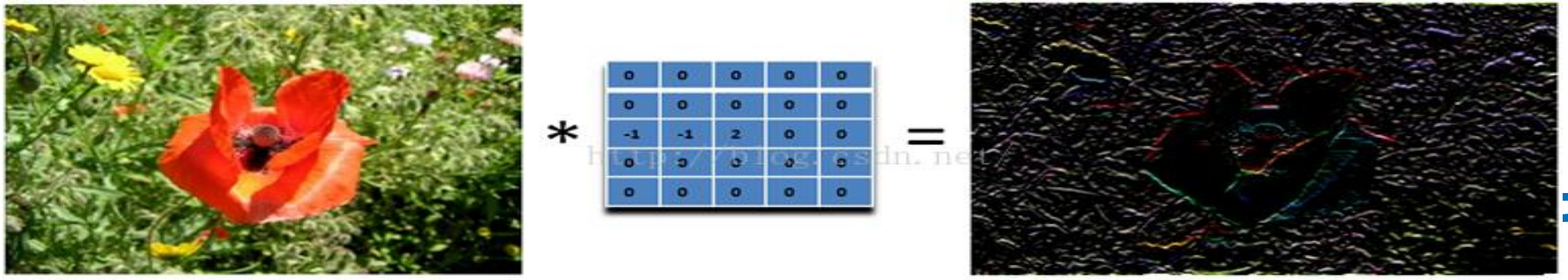
NobleProg

一个实际CNN(AlexNet)第一个卷积层的滤波器

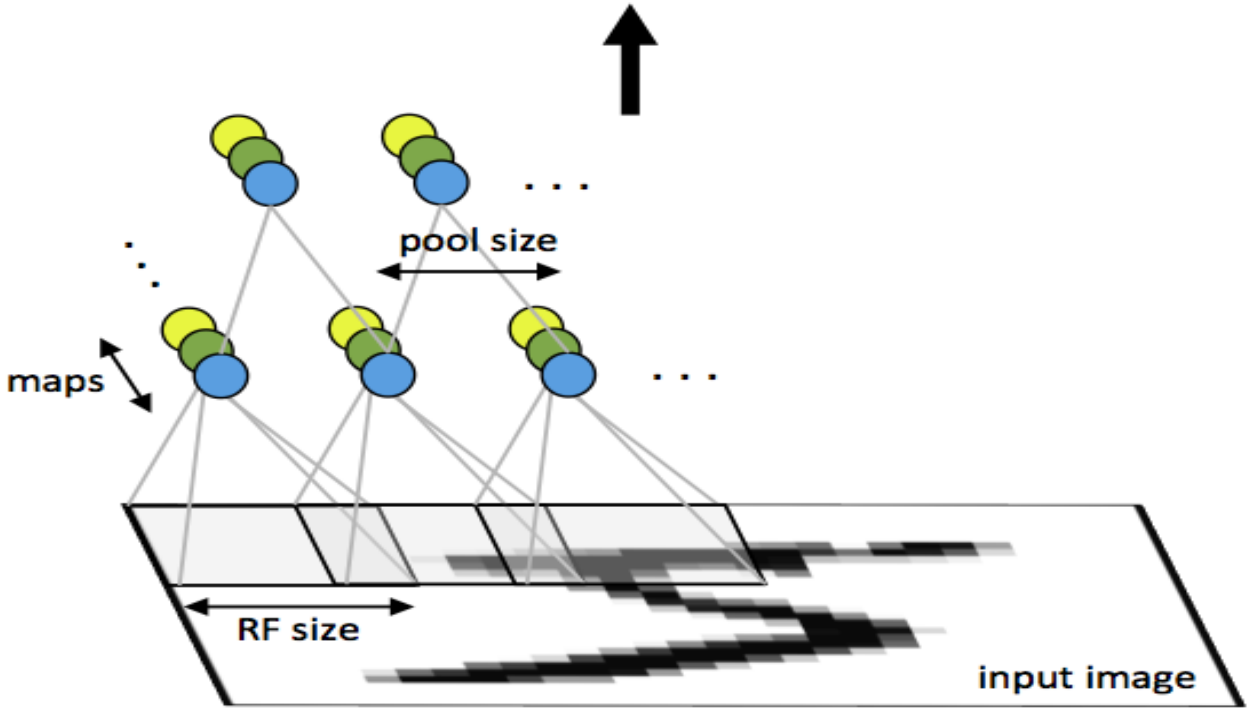


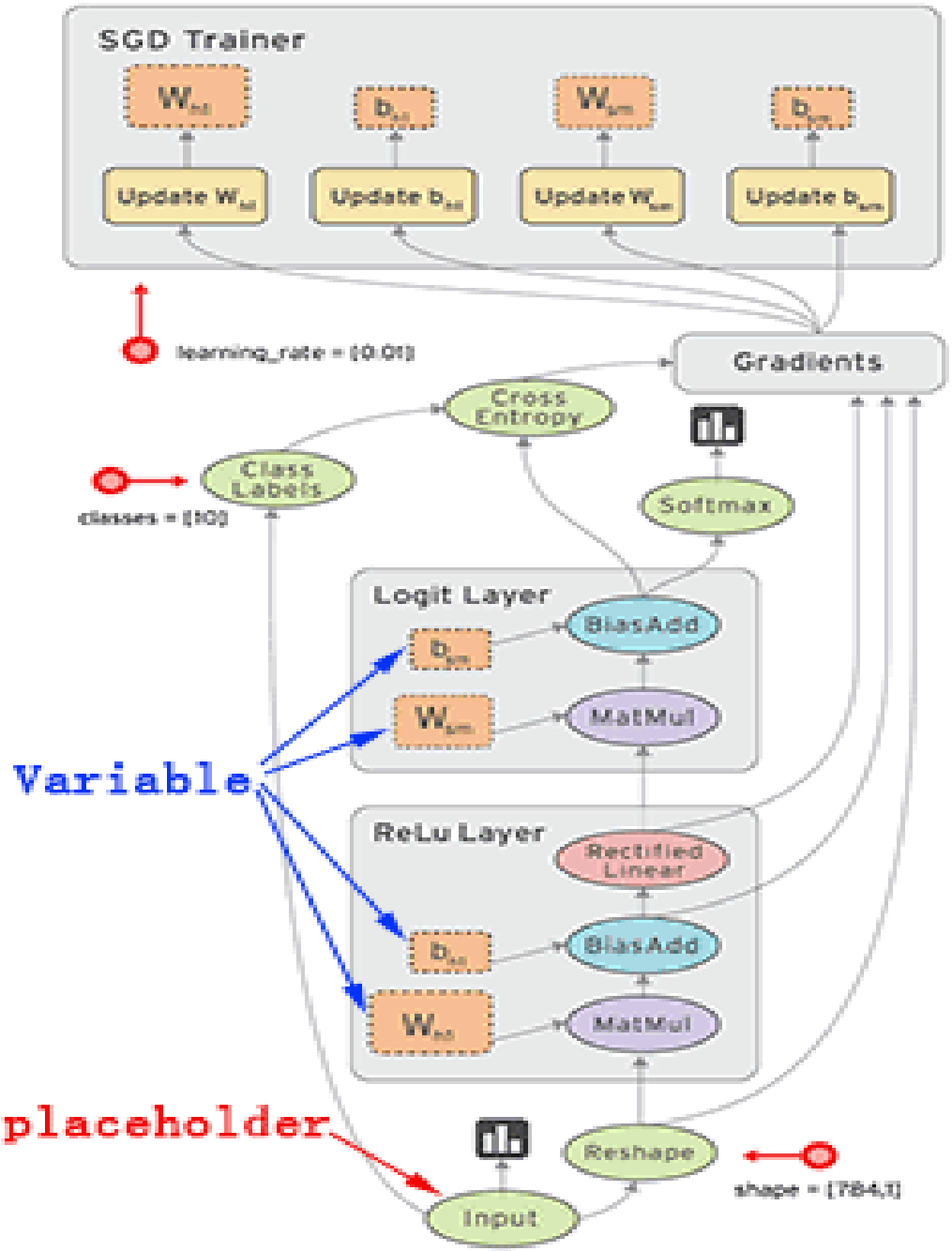






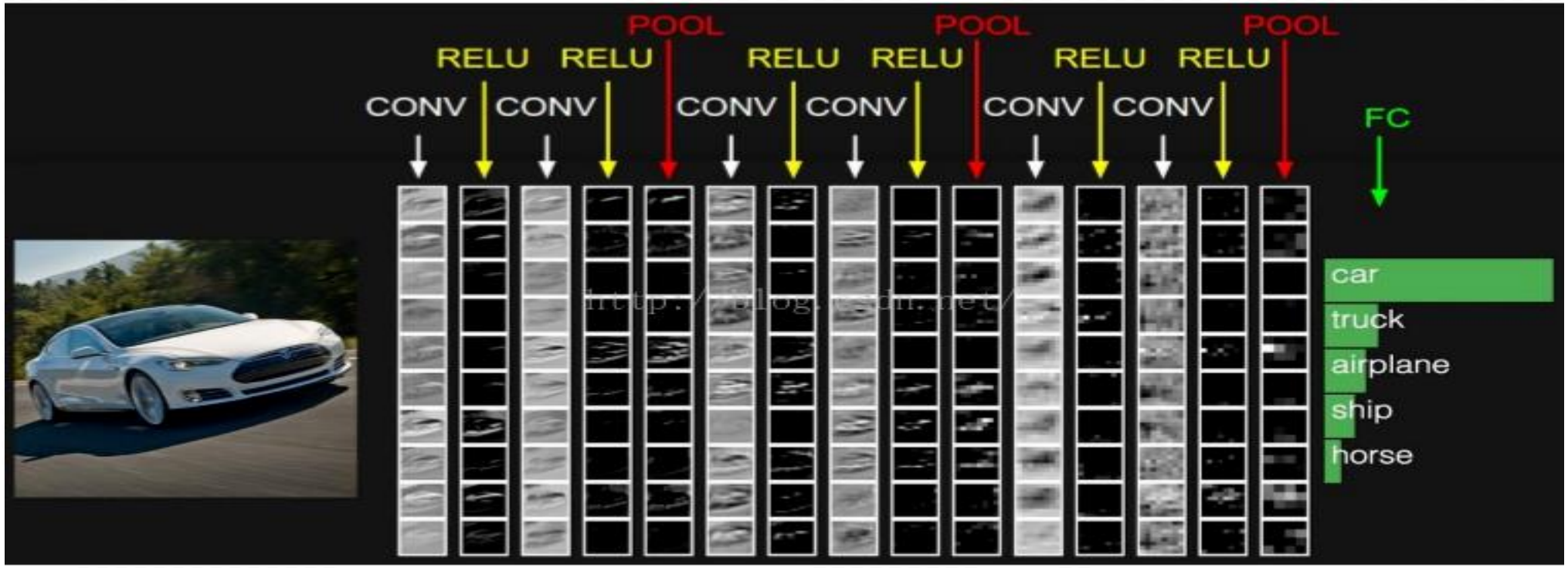
6.1.4 卷积-CNN模型





5.2.1 CNN-网络模型

1）卷积神经网络由三部分构成。第一部分是输入层。第二部分由n个卷积层和池化层的组合组成。第三部分由全连结的多层感知机分类器构成。



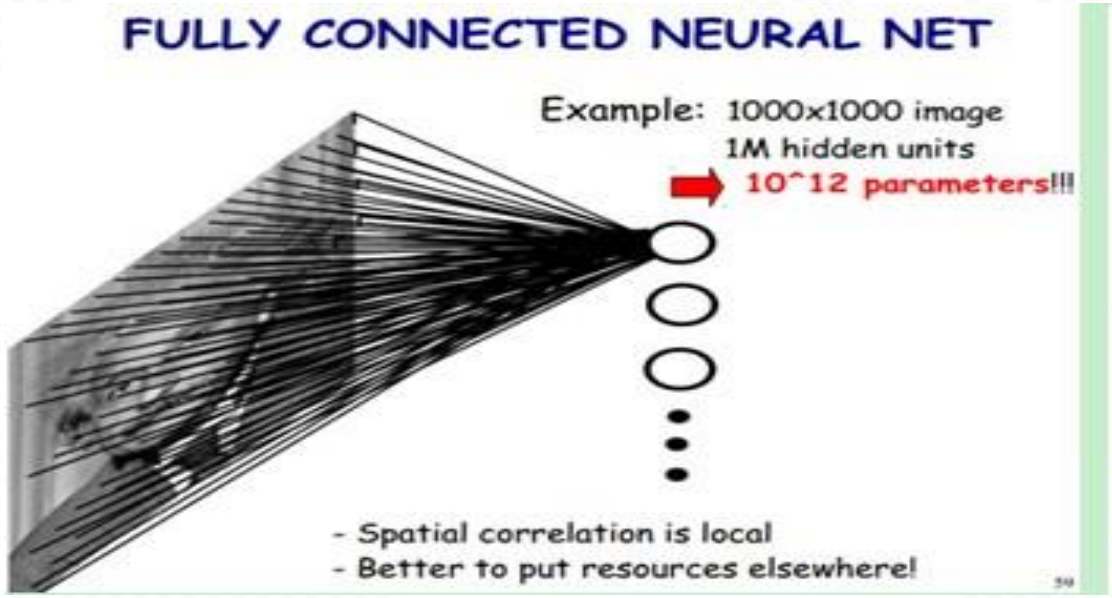
2）在CNN的卷积层中，包含若干个特征平面Feature Map，每个FM由一些矩形排列的的神经元组成，同一FM的神经元共享权值（卷积核）

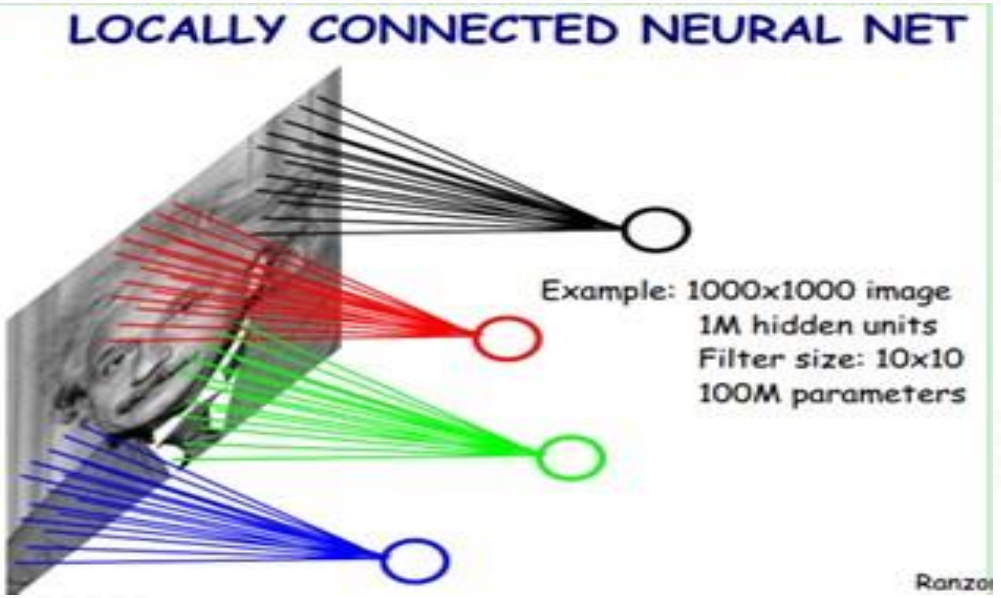
3）子采样又叫池化pooling，有mean pooling和max pooling

NobleProg

5.2.2 CNN-局部感受野

1）生物学的视觉系统结构：视觉皮层的神经元就是局部接受信息的（即这些神经元只响应某些特定区域的刺激）





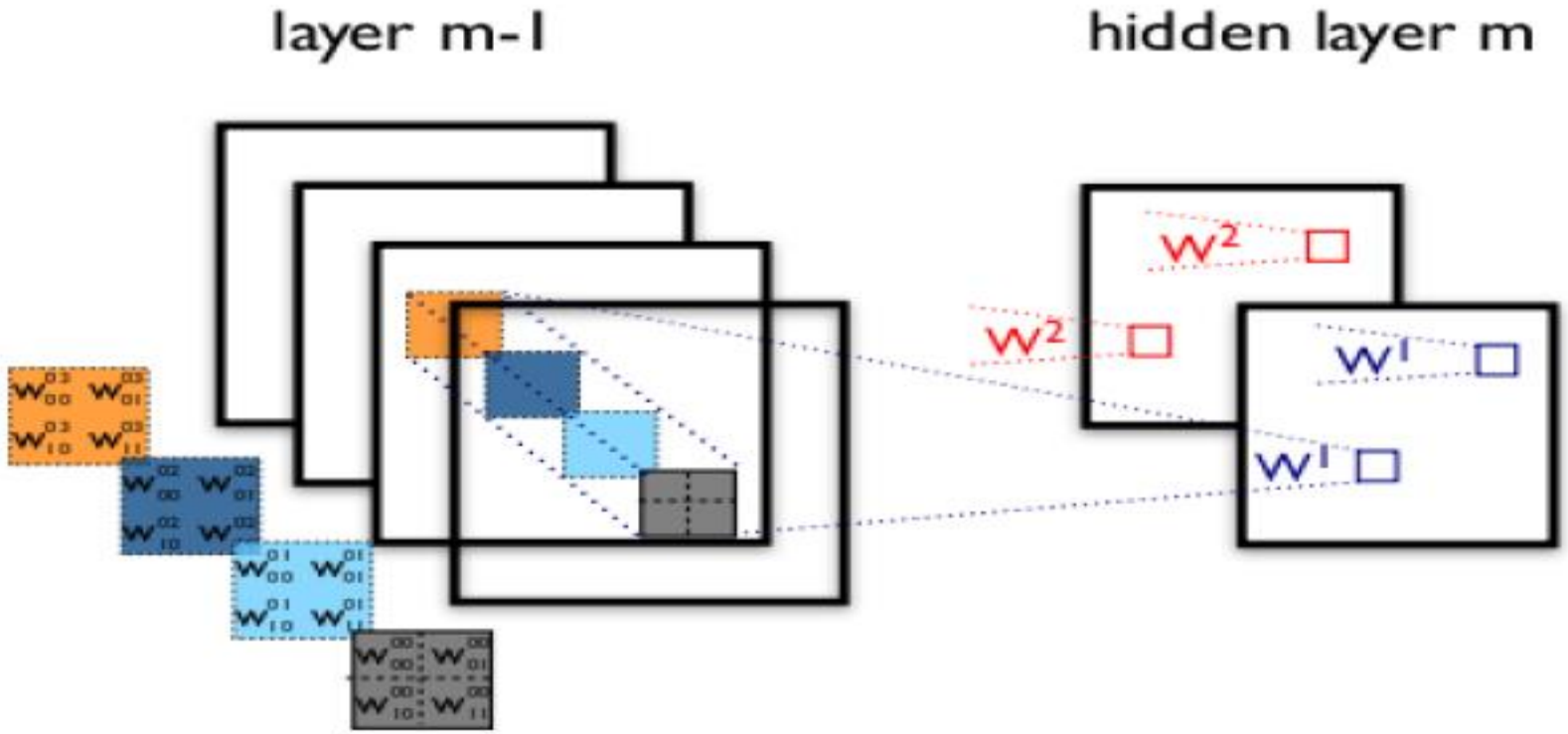
2）每个神经元不必对全局图像进行感知，只需对局部进行感知，然后在更高层将局部信息综合得到全局信息

3）上右图中，若每个神经元只和10×10个像素值相连，那么权值数据为1000000×100个参数，减少为原来的万分之一。而那10×10个像素值对应的10×10个参数，其实就相当于卷积操作

NobleProg

5.2.3 CNN-共享权值(卷积核)

1）卷积核一般以随机小数矩阵形式初始化，训练过程中将学习得到合理权值，共享权值好处是减少网络各层之间的连接，又降低了过拟合的风险



$$x_j^{\ell} = f\left(\sum_{i \in M_j} x_i^{\ell-1} * k_{ij}^{\ell} + b_j^{\ell}\right)$$

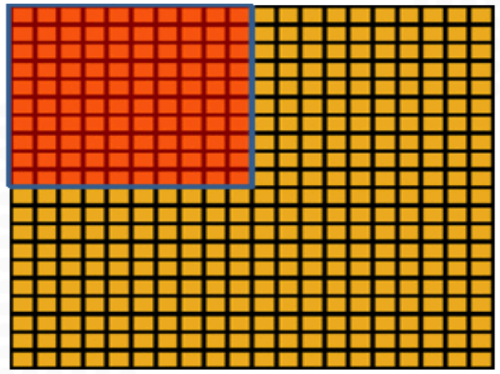
2）在四个通道上的卷积操作，有两个卷积核，生成两个通道。四个通道上每个通道对应一个卷积核，先将w2忽略，只看w1，那么在w1的某位置（i,j）处的值，是由四个通道上该处卷积结果相加然后再取激活函数值得到的

NobleProg

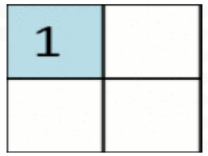
5.2.4 down-pooling

1）一个 96X96 像素图像，设已经学习得到400个定义在8X8输入上的特征，每一个特征和图像卷积都会得到一个 (96 − 8 + 1) × (96 − 8 + 1) = 7921 维的卷积特征，由于有 400个特征，所以每个样例example都会得到一个 7921×400=3,168,400维的卷积特征向量。学习一个拥有超过3百万特征输入的分类器十分不便，且容易出现过拟合

2）需要计算图像一个区域的某个特定特征的平均值 (或最大值)，这些概要统计特征具有低得多的维度，还防止过拟合，这种聚合操作叫池化pooling



Convolved feature



Pooled feature

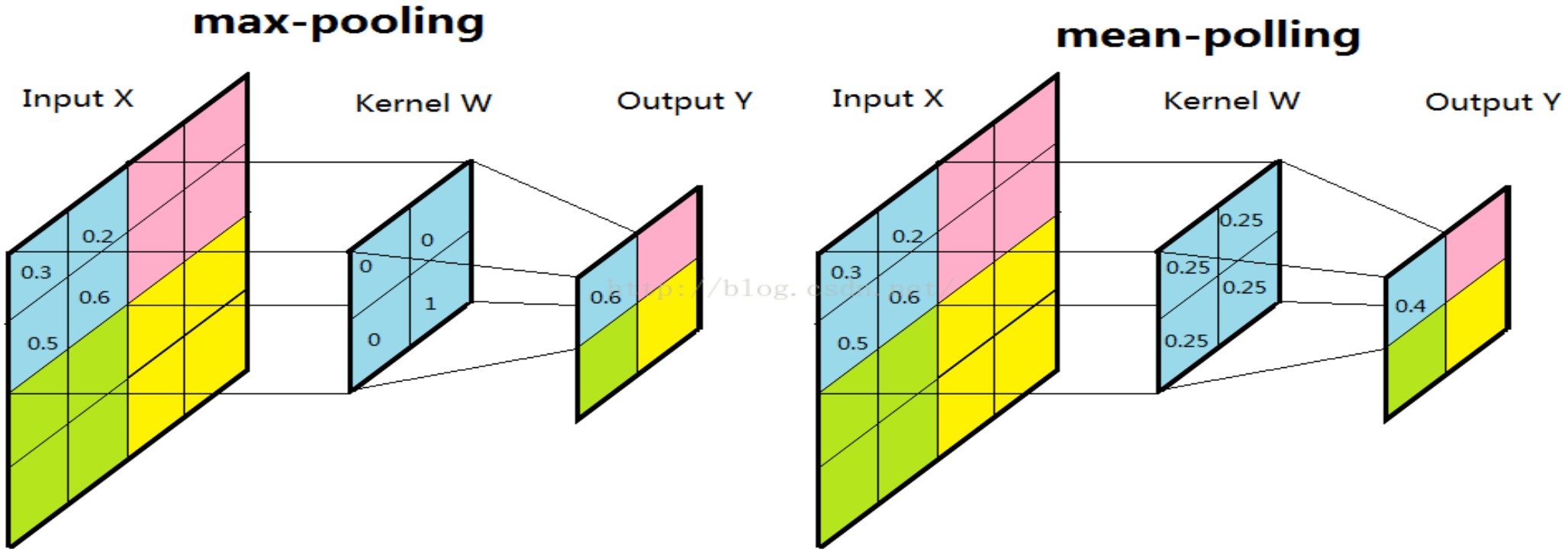
$$x_j^{\ell} = f\left(\beta_j^{\ell}\,\mathrm{down}(x_j^{\ell-1}) + b_j^{\ell}\right)$$

NobleProg

5.2.5 CNN-max/mean pooling

1)均值子采样：卷积核每个权重都是0.25，滑动步长为2。把原图模糊缩减至原来的1/4

2)最大值子采样：卷积核各权重只有一个为1，其余均为0，卷积核中为1的位置对应input X被卷积核覆盖部分值最大的位置。滑动步长为2。把原图缩减至原来的1/4，并保留每个2*2区域的最强输入

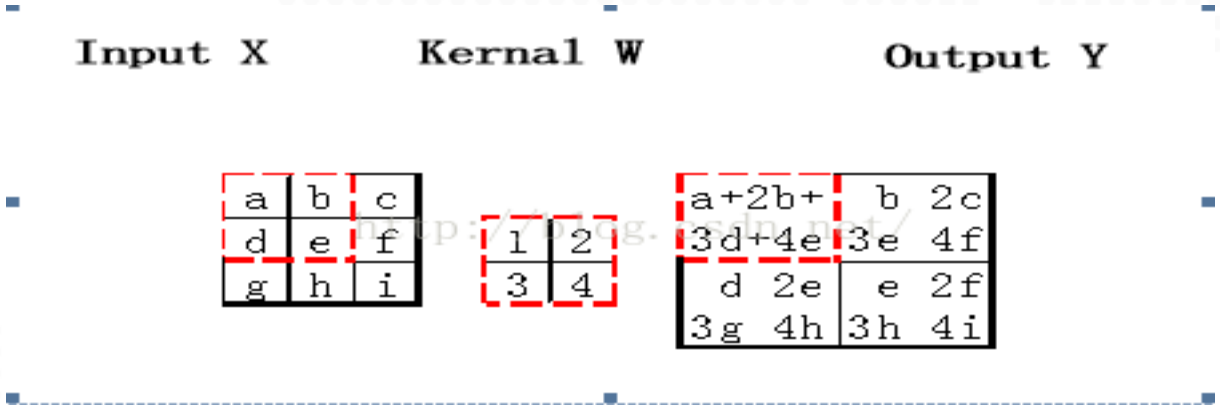


NobleProg

5.2.6 CNN训练-前向传播Forward

1）前向过程的卷积为典型valid卷积过程，即卷积核kernal W覆盖在输入图上，对应位置求积再求和得到一个值并赋给输出图对应的位置

2）每次卷积核在输入图上移动一个位置，从上到下从左到右交叠覆盖一遍之后得到输出矩阵

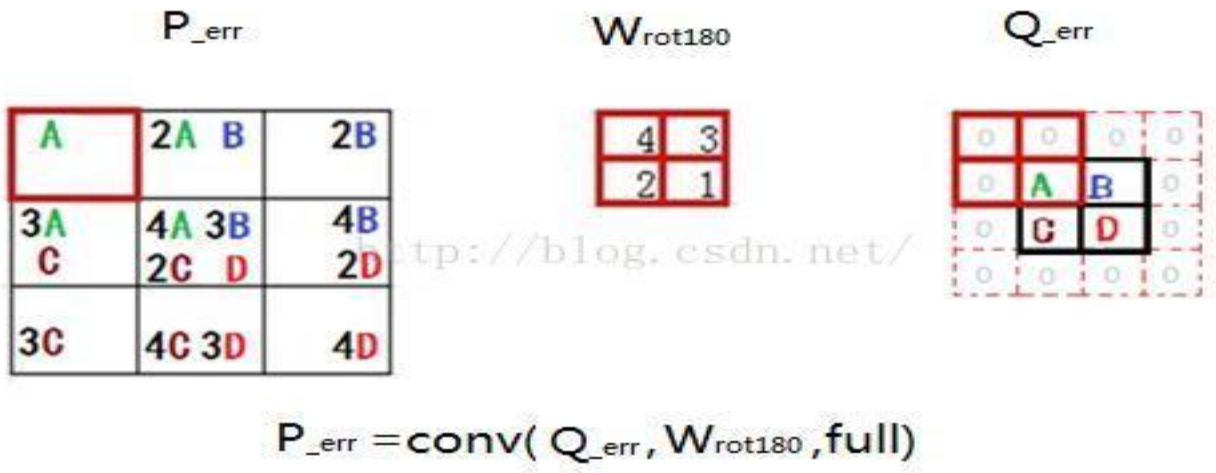


3）如果卷积核的输入图为Mx*Nx，卷积核为Mw*Nw，那么输出图大小为（Mx-Mw+1）*（Nx-Nw+1）

NobleProg

5.2.7 CNN训练-反向传播Back Forward

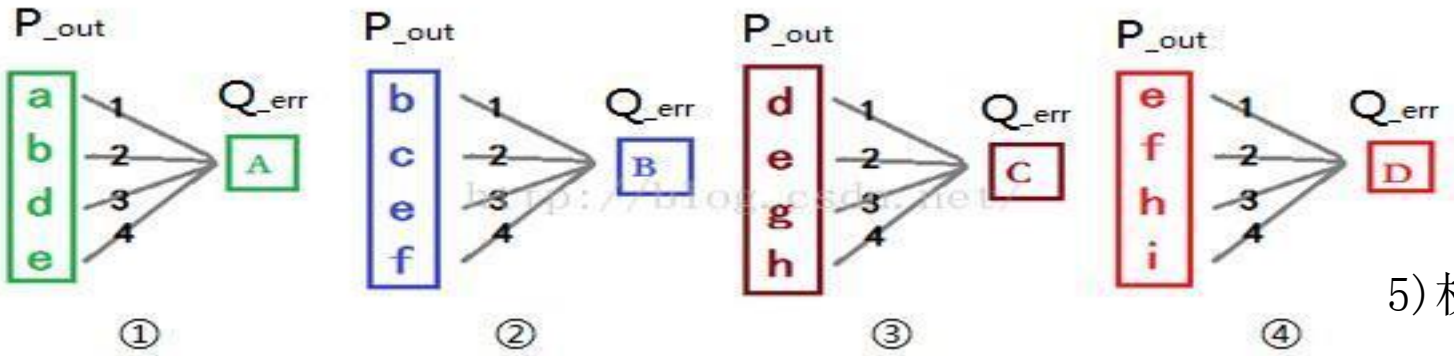
1）错误信号从子采样层的特征图subFeatureMap往前面卷积层的特征图FeatureMap传播要通过一次full卷积过程来完成
2）subFeatureMap的错误信号矩阵P_err等于featureMap的误差矩阵Q_err卷积旋转180度的卷积核W_rot180
3）错误信号矩阵Q_err中的A，它的产生是P中左上2*2小方块导致的，该2*2的小方块的对A的责任正好可以用卷积核W表示，错误信号A通过卷积核将错误信号加权传递到与错误信号量为A的神经元所相连的神经元a、b、d、e中，所以在下图中的P_err左上角的2*2位置错误值包含A、2A、3A、4A，错误信号反向传播过程可以用下图中的卷积过程表示



NobleProg

5.2.8 CNN训练-权值更新

1）卷积神经网络中卷积层的权重更新过程本质是卷积核的更新过程
2）由神经网络的权重修改策略，一条连接权重的更新量为该条连接的前层神经元的兴奋输出乘以后层神经元的输入错误信号，卷积核的更新也是按照这个规律来进行
3）把前向卷积过程当做切割小图进行多个神经网络训练过程，得到四个4*1的神经网络的前层兴奋输入和后层输入错误信号：



Kernal W

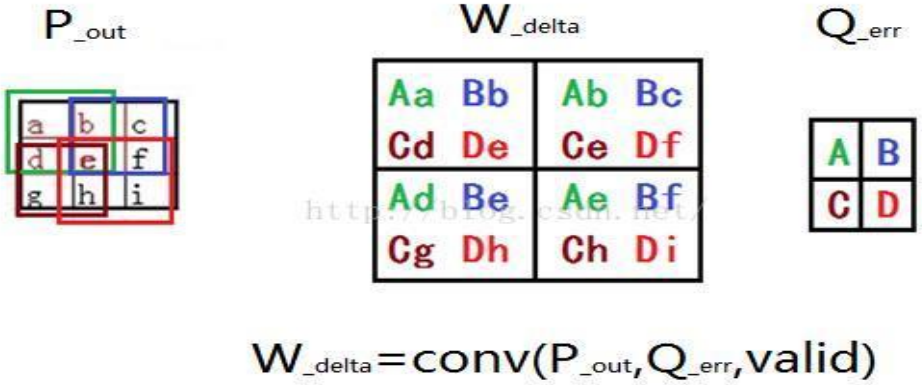
1	2
3	4

W_delta

Aa Bb	Ab Bc
Cd De	Ce Df
Ad Be	Ae Bf
Cg Dh	Ch Di

4)根据神经网络的权重修改策略，算出卷积核的更新量W_delta

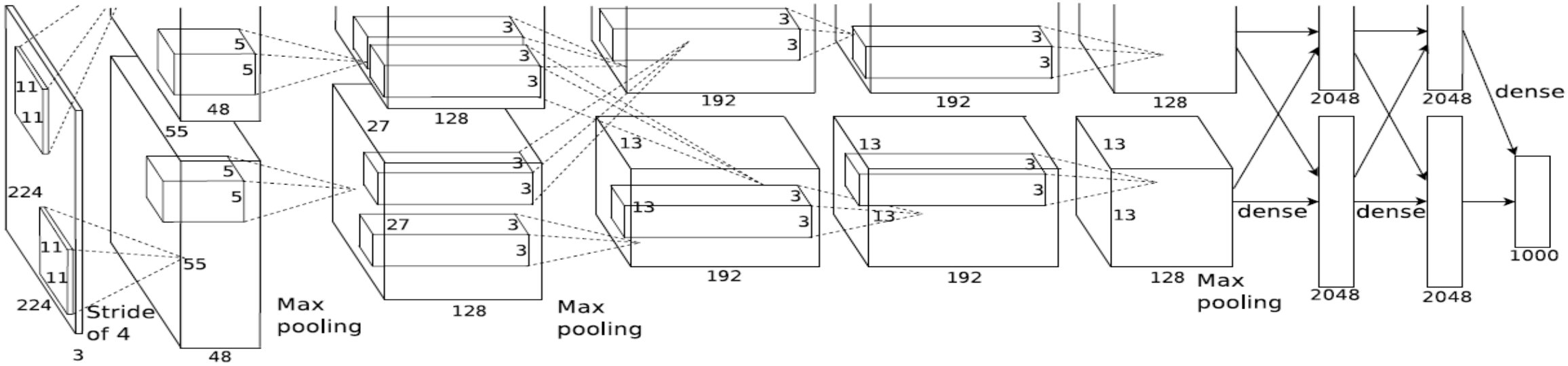
5)权重更新量W_delta可由P_out和Q_err卷积得到



$W_{delta} = conv(P_{out}, Q_{err}, valid)$

NobleProg

5.2.9 CNN- ImageNet-2010

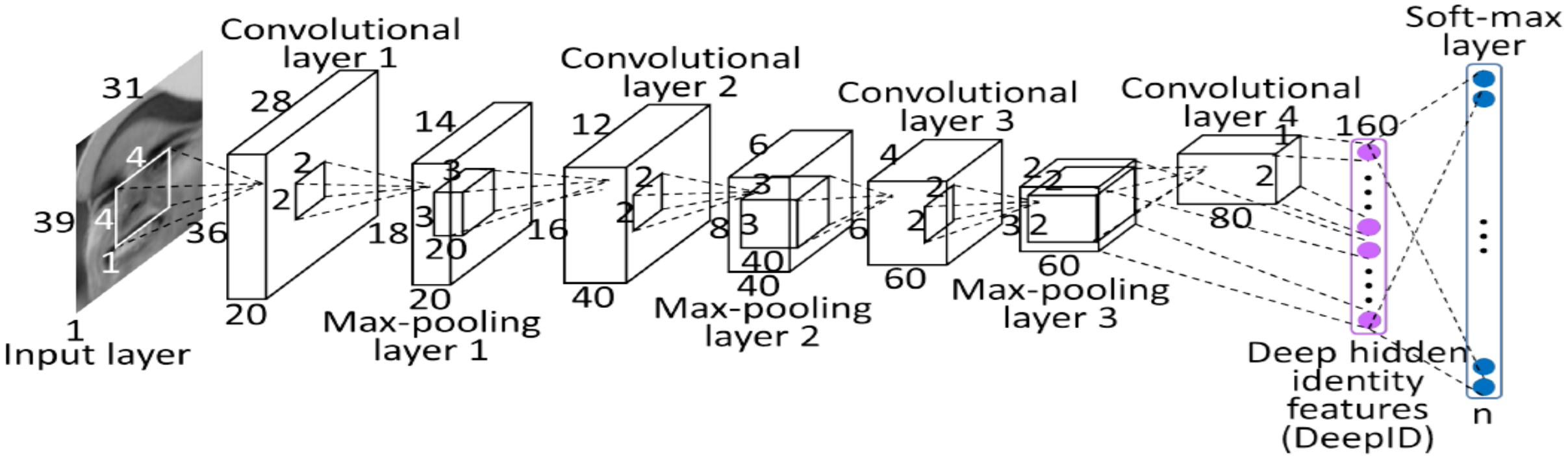


输入：224×224大小的图片，3通道
第一层卷积：11×11大小的卷积核96个，每个GPU上48个
第一层max-pooling：2×2的核
第二层卷积：5×5卷积核256个，每个GPU上128个
第二层max-pooling：2×2的核
第三层卷积：与上一层是全连接，3*3的卷积核384个。分到两个GPU上个192个
第四层卷积：3×3卷积核384个，两个GPU各192个。该层与上一层连接没有经过pooling层
第五层卷积：3×3的卷积核256个，两个GPU上个128个
第五层max-pooling：2×2的核
第一层全连接：4096维，将第五层max-pooling的输出连接成为一个一维向量，作为该层的输入。第二层全连接：4096维
Softmax层：输出为1000，输出的每一维都是图片属于该类别的概率。

NobleProg

5.2.10 CNN-DeepID网络结构

1)香港中文大学的Sun Yi开发出来用来学习人脸特征的卷积神经网络



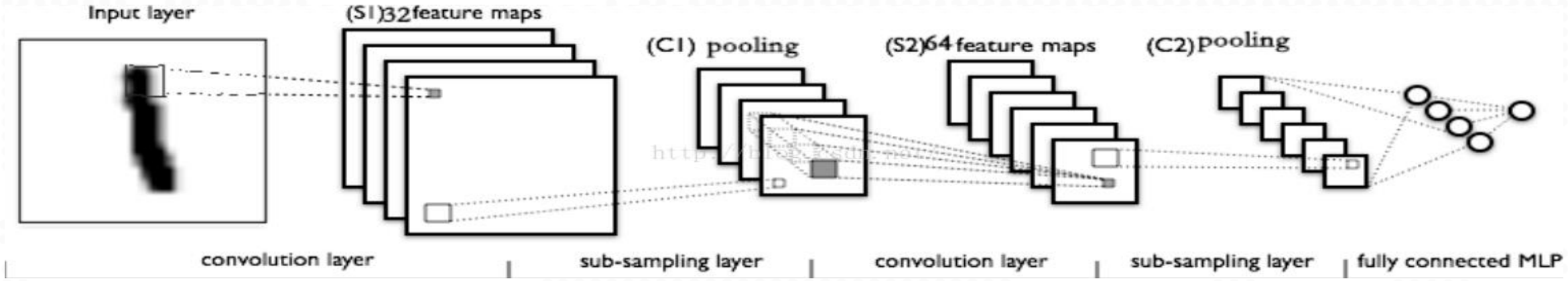
2）在最后只有一层全连接层，然后就是softmax层了。以该全连接层作为图像的表示

3）在全连接层，以第四层卷积和第三层max-pooling的输出作为全连接层的输入，这样可以学习到局部的和全局的特征

NobleProg

5.3.1 实现Tensorflow-MNIST

- LeNet5结构:输入-卷积-pooling-卷积-pooling-全连接层-Dropout-Softmax输出



第一层卷积：5*5的patch，32个卷积核，算出32个特征，然后进行2*2 maxpooling
第二层卷积：5*5的patch，64个卷积核，算出64个特征，然后进行2*2max pooling
输入图片为28*28，步长stride size为1，0填充模块zero padded
第一层卷积后，卷积特征为28*28，后通过ReLU函数激活，pooling后得到14*14特征
第二层卷积后，卷积特征为14*14，后通过ReLU函数激活，再经pooling后得到特征为7*7。后加入一个1024个神经元的全连接层，把池化层输出的张量展开成向量，乘上权重矩阵，加上偏置，然后对其使用ReLU
为避免过拟合，在全连接层输出接上dropout层，在训练时屏蔽一半的神经元

mnist.py mnist1.py mnist2.py mnist3.py mnist4.py mnist5.py

NobleProg

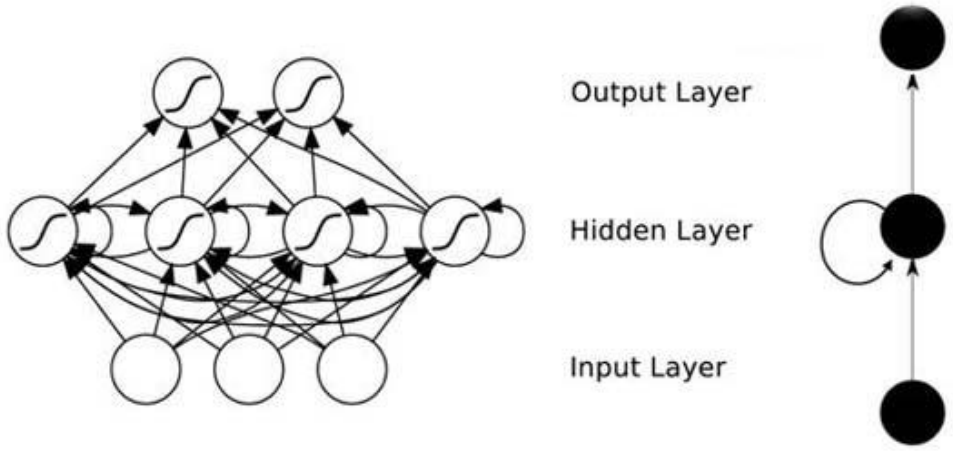
5.3.2 实现-TFLearn

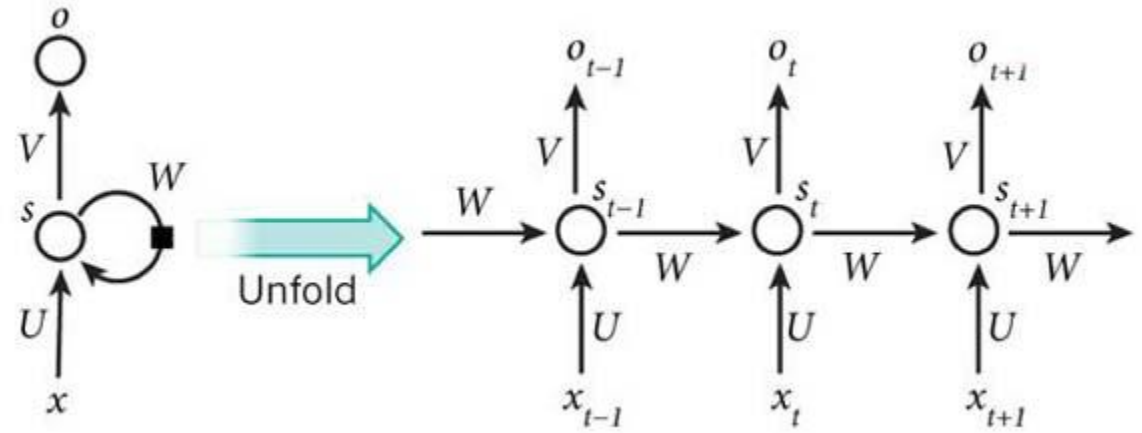
- Basic:linear、logic、weights_persistence、use_hdf5、use_dask
- CV:inception_resnet、convnet_mnist、convnet_cifar10、network_in_network、alexnet、vgg_network、rnn_pixels、highway_dnn、dnn、residual_network、autoencoder
- NLP:LSTM、bidirectional_lstm、dynamic_lstm、seq2seq、dynamic_generator_cityname、cnn_sentence_classification
- Reinforcement_learning:atari_1step_qlearning、Recommender-Wide&Deep Network、Notebooks
- extending:layers、trainer、builtin-ops、summaries、variables
- ..>Python setup.py install

NobleProg

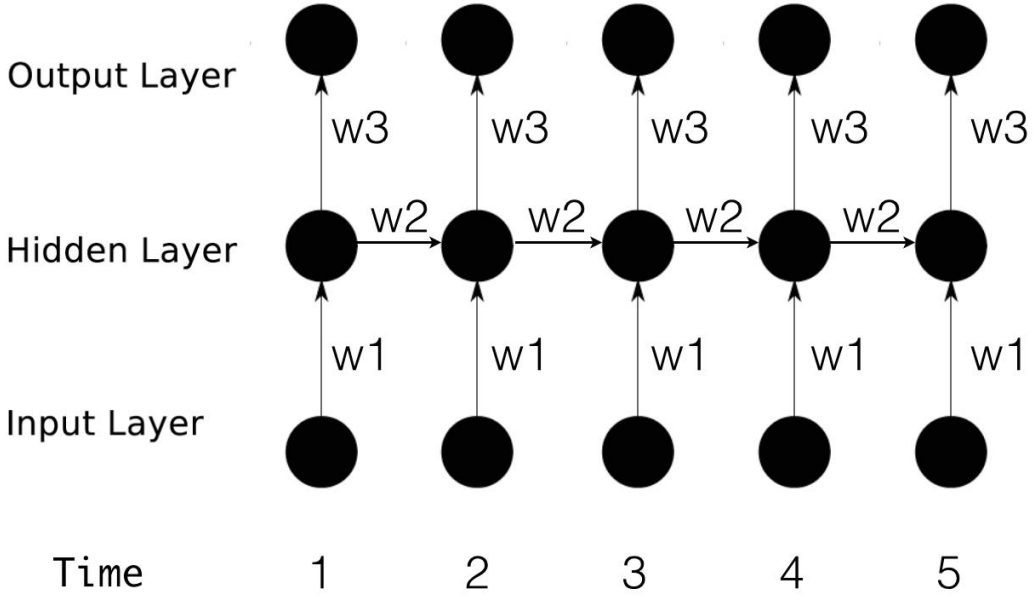
6. RNN递归神经网络

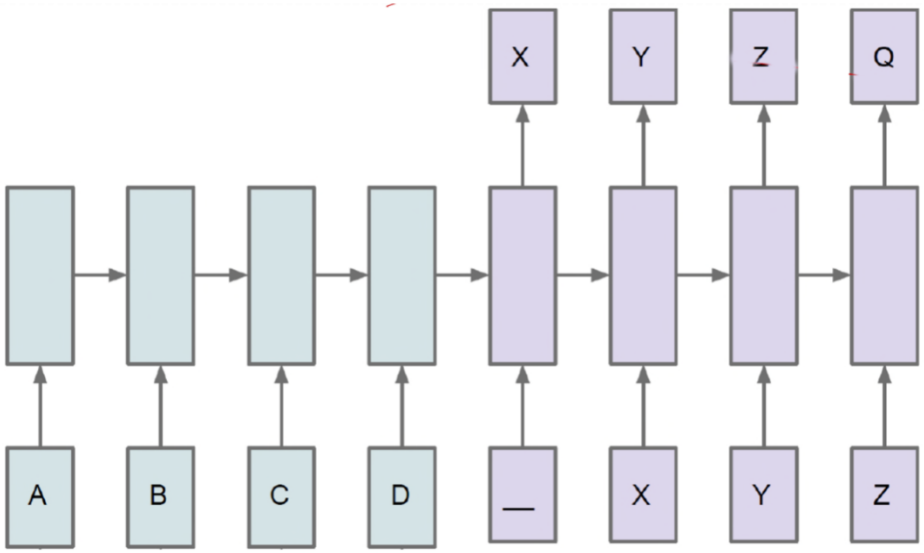
- 全连接的DNN还存在着另一个问题——无法对时间序列上的变化进行建模。然而，样本出现的时间顺序对于自然语言处理、语音识别、手写体识别等应用非常重要。对了适应这种需求，就出现了题主所说的另一种神经网络结构——循环神经网络RNN。
- 在普通的全连接网络或CNN中，每层神经元的信号只能向上一层传播，样本的处理在各个时刻独立，因此又被成为前向神经网络(Feed-forward Neural Networks)。而在RNN中，神经元的输出可以在下一个时间戳直接作用到自身，即第i层神经元在m时刻的输入，除了(i-1)层神经元在该时刻的输出外，还包括其自身在(m-1)时刻的输出!表示成图就是这样的；
- (t+1)时刻网络的最终结果O(t+1)是该时刻输入和所有历史共同作用的结果!这就达到了对时间序列建模的目的



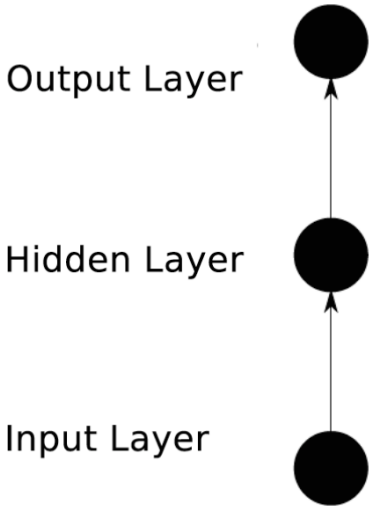


NobleProg





BPTT（Back Propagation Through Time）算法

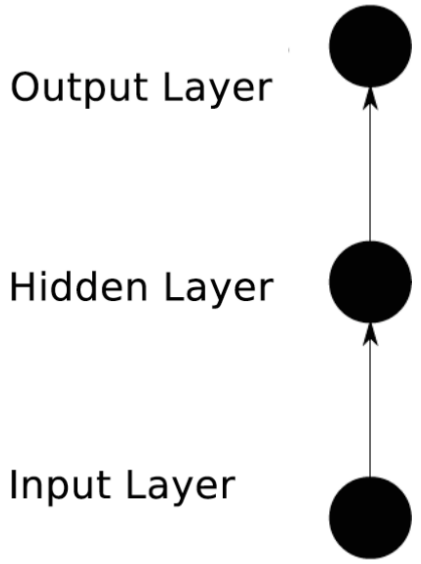


$$a_k^t = \sum_{h=1}^{H} w_{hk} b_h^t$$

$$a_h^t = \sum_{i=1}^{I} w_{ih} x_i^t + \sum_{h'=1}^{H} w_{h'h} b_{h'}^{t-1}$$

$$b_h^t = \theta_h(a_h^t)$$

前向传播



$$\delta_h^t = \theta'(a_h^t)\left(\sum_{k=1}^{K} \delta_k^t w_{hk} + \sum_{h'=1}^{H} \delta_{h'}^{t+1} w_{hh'}\right)$$

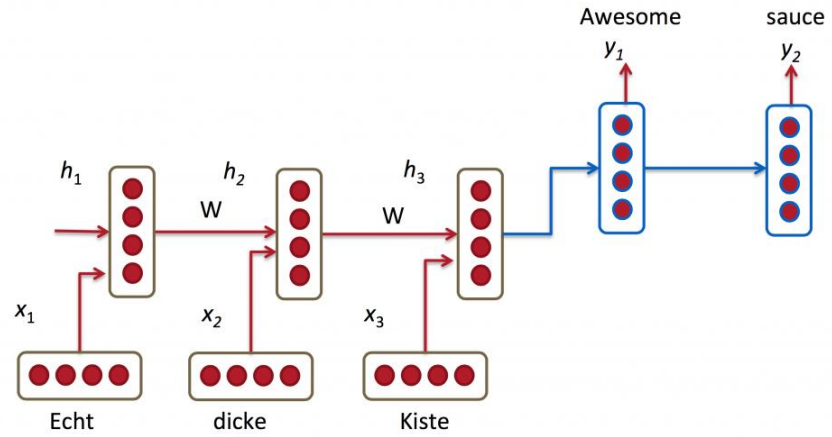
$$\delta_j^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial a_j^t}$$

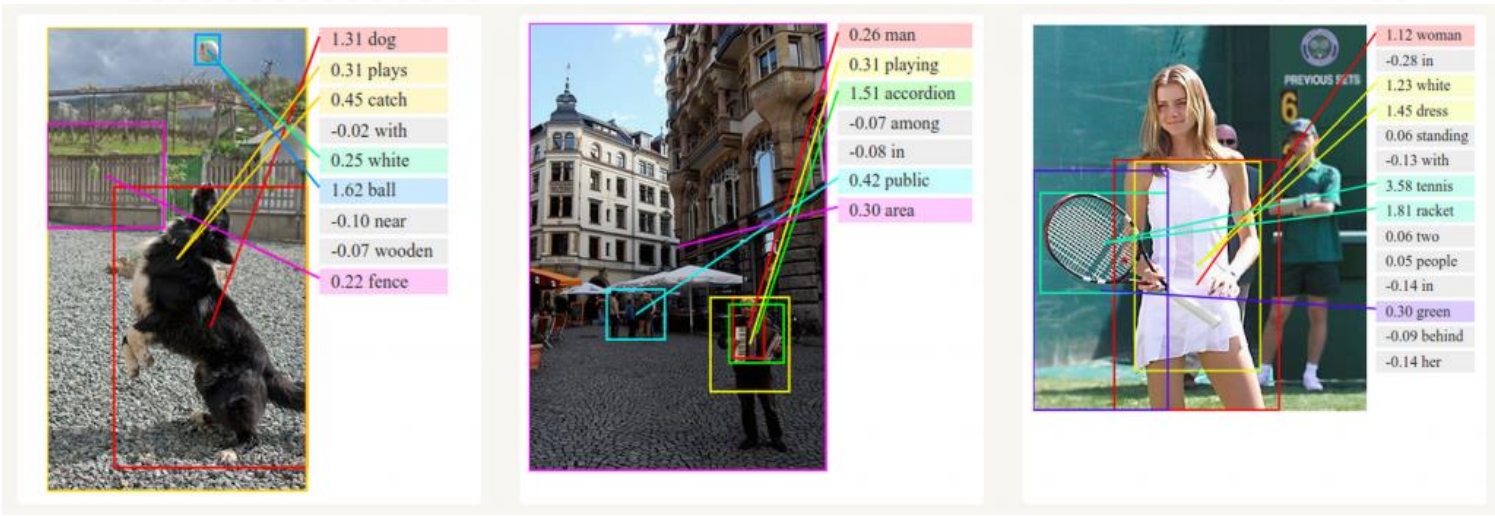
$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}}{\partial a_j^t} \frac{\partial a_j^t}{\partial w_{ij}} = \sum_{t=1}^{T} \delta_j^t b_i^t$$

后向传播

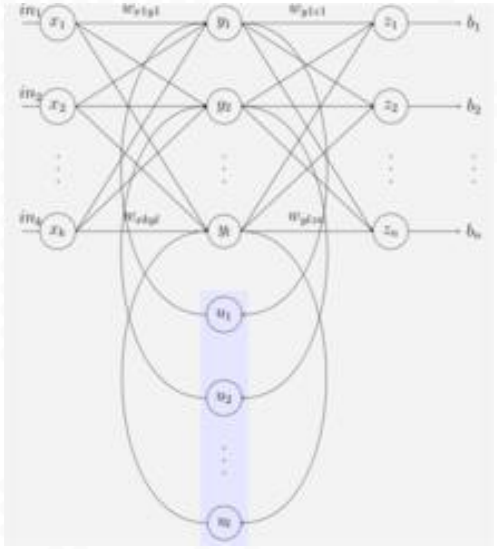
NobleProg

- x_t表示第t, t=1, 2, 3...步(step)的输入
- s_t为隐藏层的第t步的状态，它是网络的记忆单元。 s_t根据当前输入层的输出与上一步隐藏层的状态进行计算。$s_t=f(Ux_t+Ws_t-1)$，其中f一般是非线性的激活函数，如tanh或ReLU，在计算s_0时，即第一个单词的隐藏层状态，需要用到s-1，但是其并不存在，在实现中一般置为0向量；
- o_t是第t步的输出，如下个单词的向量表示，$o_t=softmax(Vs_t)$
- 在RNNs中，每输入一步，每一层各自都共享参数U,V,W。其反应者RNNs中的每一步都在做相同的事
- 对于RNN是的训练和对传统的ANN训练一样。同样使用BP误差反向传播算法，不过有一点区别，如果将RNNs进行网络展开，那么参数W,U,V是共享的。并且在使用梯度下降算法中，每一步的输出不仅依赖当前步的网络，并且还以来前面若干步网络的状态。比如在t=4时，我们还需要向后传递三步，已经后面的三步都需要加上各种的梯度。该学习算法称为Backpropagation Through Time (BPTT)
- 语言模型与文本生成(Language Modeling and Generating Text)、机器翻译(Machine Translation)
- 语音识别(Speech Recognition)、图像描述生成 (Generating Image Descriptions)



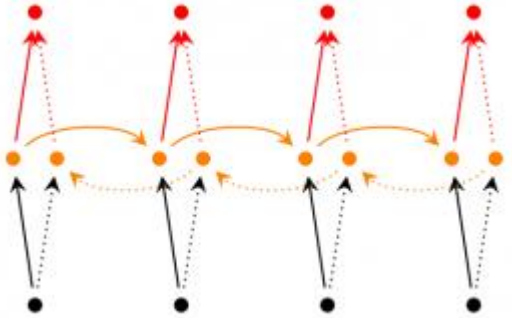


Simple RNNs(SRNs)



三层网络，在隐藏层增加了上下文单元

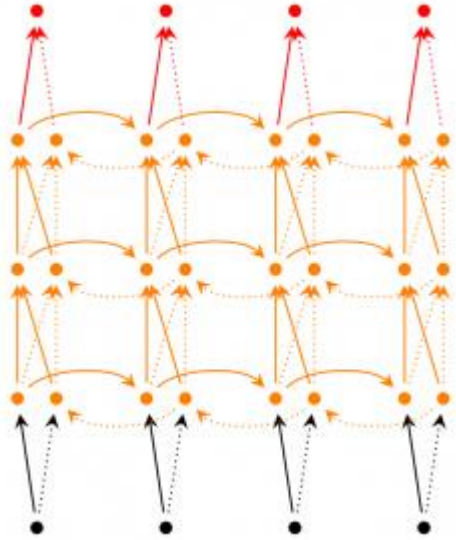
Bidirectional RNNs



假设当前的输出(第t步的输出)不仅仅与前面的序列有关，并且还与后面的序列有关

recurrent_network.py

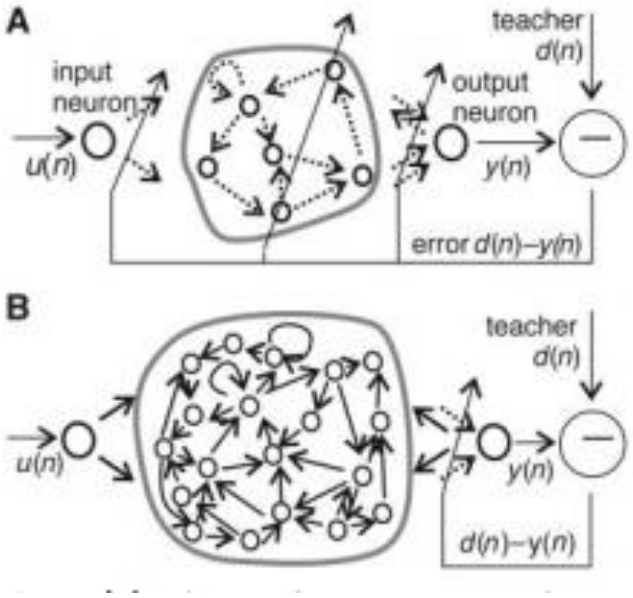
Deep(Bidirectional)RNNs



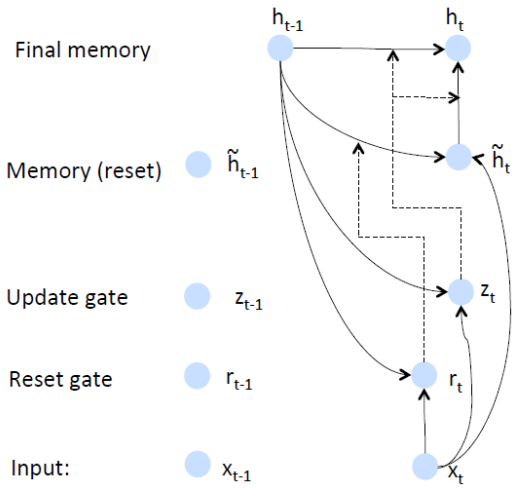
只是对于每一步的输入有多层网络。这样，该网络便有更强大的表达与学习能力

Echo State Networks

它的核心结构时一个随机生成、且保持不变的储备池Reservoir，储备池是大规模的、随机生成的、稀疏连接(SD通常保持1%～5%，SD表示储备池中互相连接的神经元占总的神经元个数N的比例)的循环结构；
其储备池到输出层的权值矩阵是唯一需要调整的部分；
简单的线性回归就可完成网络的训练



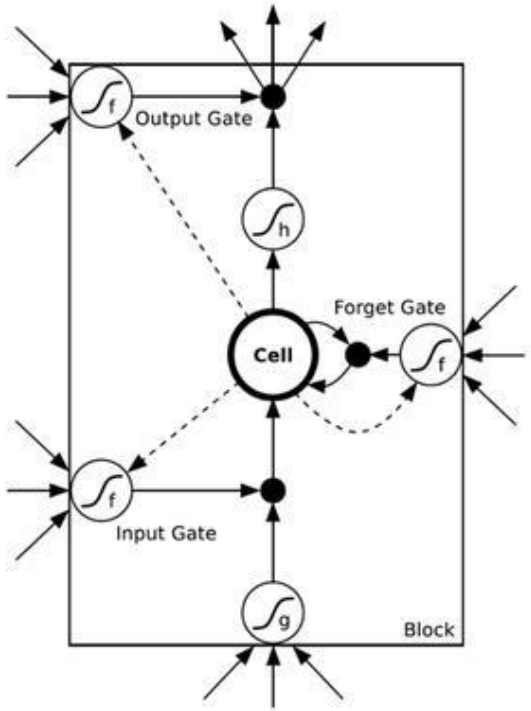
Gated Recurrent Unit Recurrent Neural Networks

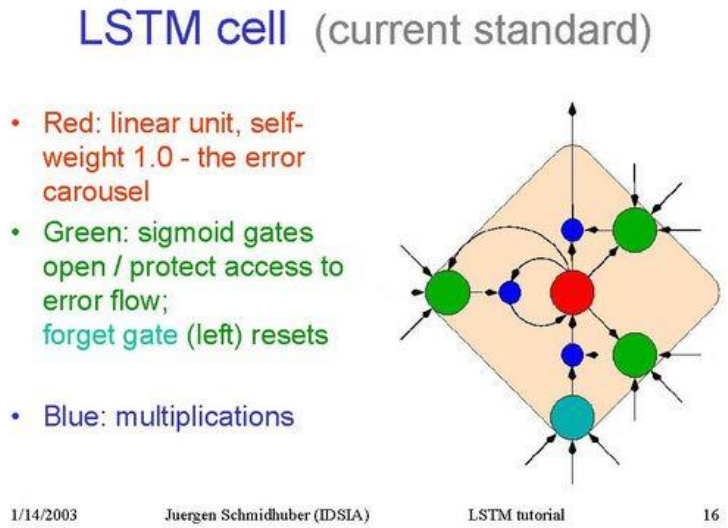


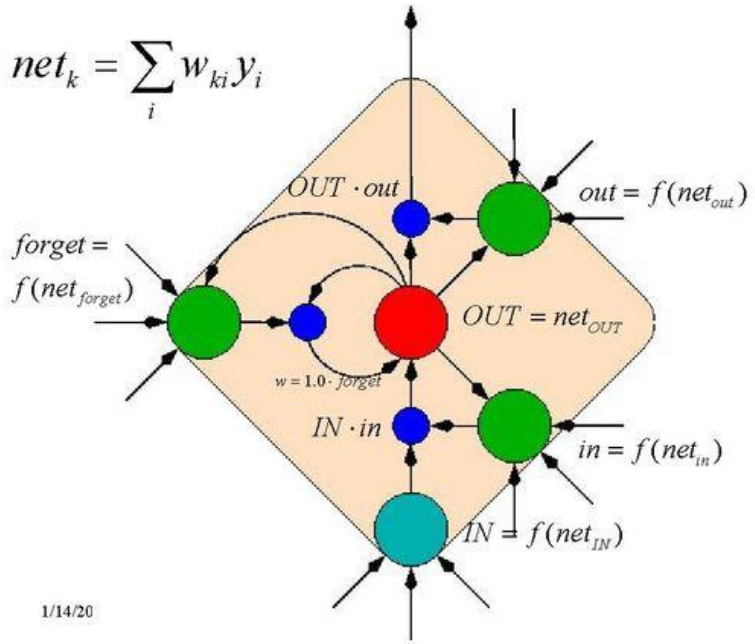
$$z_t = \sigma\left(W^{(z)}x_t + U^{(z)}h_{t-1}\right)$$
$$r_t = \sigma\left(W^{(r)}x_t + U^{(r)}h_{t-1}\right)$$
$$\tilde{h}_t = \tanh\left(Wx_t + r_t \circ Uh_{t-1}\right)$$
$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

7. LSTM长短词记忆网络

- RNN可以看成一个在时间上传递的神经网络，它的深度是时间的长度！“梯度消失”现象又要出现了，只不过这次发生在时间轴上
- 对于t时刻来说，它产生的梯度在时间轴上向历史传播几层之后就消失了，根本就无法影响太遥远的过去。因此，之前说“所有历史”共同作用只是理想的情况，在实际中，这种影响也就只能维持若干个时间戳。
- 为解决时间上的梯度消失，长短时记忆单元LSTM，通过门开关实现时间上记忆功能，并防止梯度消失







前向传播

Cell Outputs

$$b_c^t = b_\omega^t h(s_c^t) \tag{4.10}$$

Input Gates

$$a_\iota^t = \sum_{i=1}^{I} w_{i\iota} x_i^t + \sum_{h=1}^{H} w_{h\iota} b_h^{t-1} + \sum_{c=1}^{C} w_{c\iota} s_c^{t-1} \tag{4.2}$$

$$b_\iota^t = f(a_\iota^t) \tag{4.3}$$

Forget Gates

$$a_\phi^t = \sum_{i=1}^{I} w_{i\phi} x_i^t + \sum_{h=1}^{H} w_{h\phi} b_h^{t-1} + \sum_{c=1}^{C} w_{c\phi} s_c^{t-1}$$

$$b_\phi^t = f(a_\phi^t)$$

Cells

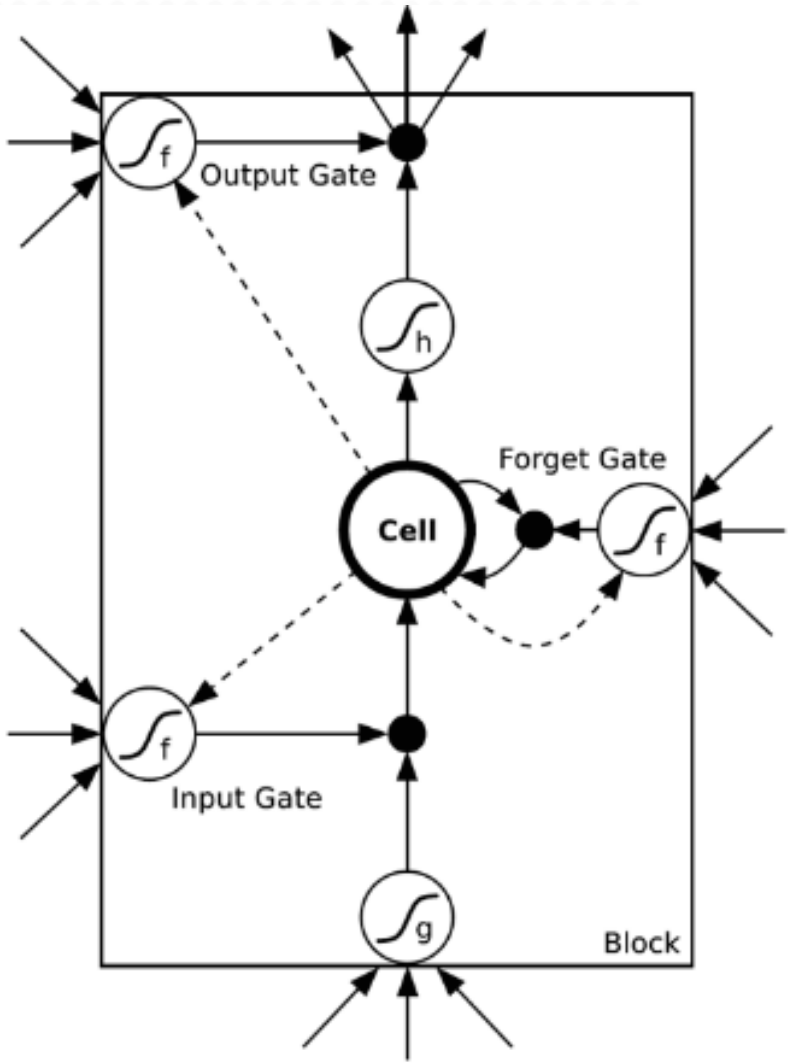
$$a_c^t = \sum_{i=1}^{I} w_{ic} x_i^t + \sum_{h=1}^{H} w_{hc} b_h^{t-1} \tag{4.6}$$

$$s_c^t = b_\phi^t s_c^{t-1} + b_\iota^t g(a_c^t) \tag{4.7}$$

Output Gates

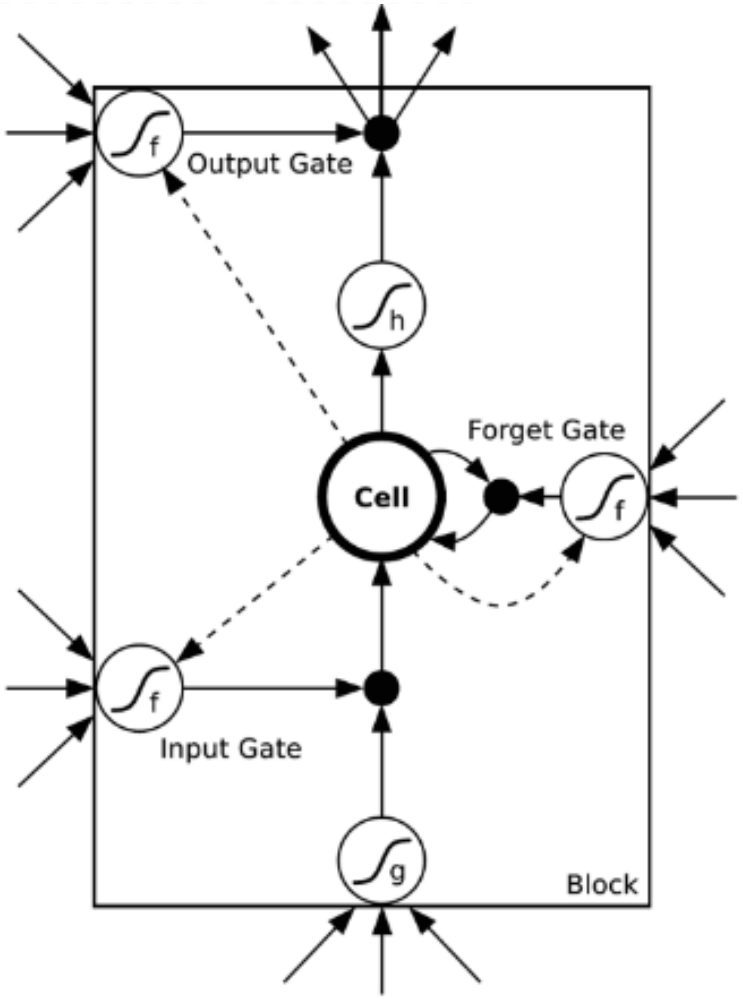
$$a_\omega^t = \sum_{i=1}^{I} w_{i\omega} x_i^t + \sum_{h=1}^{H} w_{h\omega} b_h^{t-1} + \sum_{c=1}^{C} w_{c\omega} s_c^t \tag{4.}$$

$$b_\omega^t = f(a_\omega^t) \tag{4.9}$$



NobleProg

后向传播



$$\epsilon_c^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial b_c^t} \qquad \epsilon_s^t \stackrel{\text{def}}{=} \frac{\partial \mathcal{L}}{\partial s_c^t}$$

Cell Outputs

$$\epsilon_c^t = \sum_{k=1}^{K} w_{ck}\delta_k^t + \sum_{g=1}^{G} w_{cg}\delta_g^{t+1} \tag{4.11}$$

Output Gates

$$\delta_\omega^t = f'(a_\omega^t)\sum_{c=1}^{C} h(s_c^t)\epsilon_c^t \tag{4.12}$$

States

$$\epsilon_s^t = b_\omega^t h'(s_c^t)\epsilon_c^t + b_\phi^{t+1}\epsilon_s^{t+1} + w_{c\iota}\delta_\iota^{t+1} + w_{c\phi}\delta_\phi^{t+1} + w_{c\omega}\delta_\omega^t \tag{4.13}$$

Cells

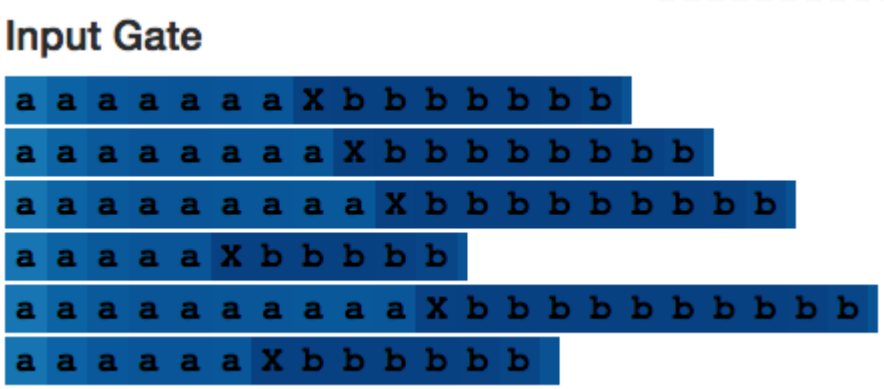
$$\delta_c^t = b_\iota^t g'(a_c^t)\epsilon_s^t \tag{4.14}$$

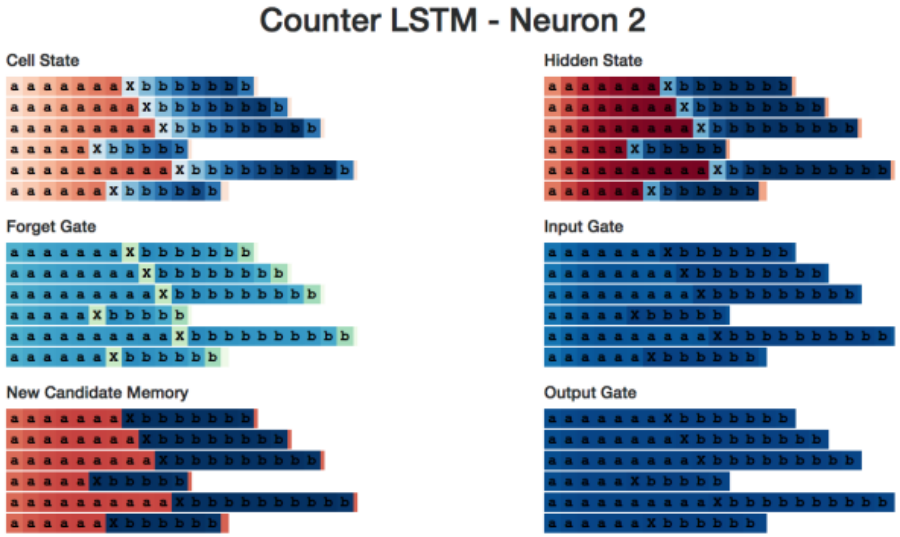
Forget Gates

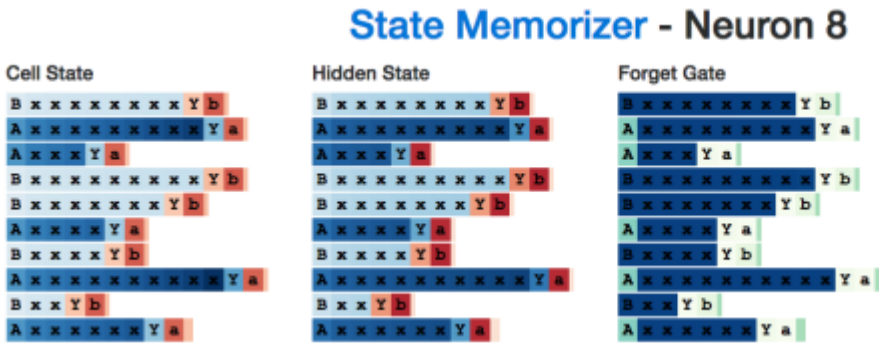
$$\delta_\phi^t = f'(a_\phi^t)\sum_{c=1}^{C} s_c^{t-1}\epsilon_s^t \tag{4.15}$$

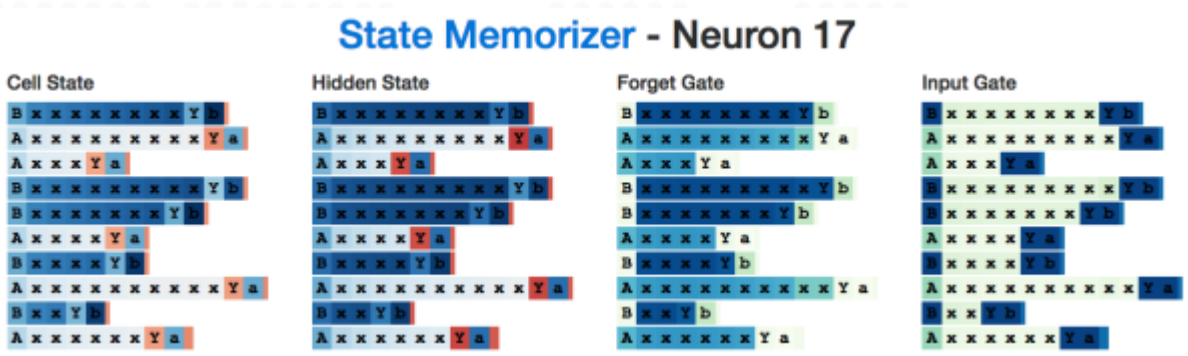
Input Gates

$$\delta_\iota^t = f'(a_\iota^t)\sum_{c=1}^{C} g(a_c^t)\epsilon_s^t \tag{4.16}$$









[LSTM Explorer]

NobleProg

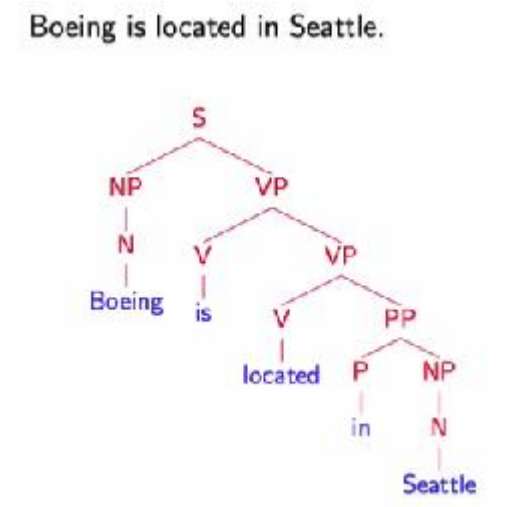
9. NLP-主题模型

- 9.1 语法分析
- 9.2 词向量与语言模型
- 9.3 主题模型
- 9.4 关键字树

NobleProg

9.1 语法分析

- 句子可以用主语、谓语、宾语来表示
- 树状结构图：S表示句子；NP、VP、PP是名词、动词、介词短语（短语级别）；N、V、P分别是名词、动词、介词



(S (NP (N Boeing)) (VP (V is) (VP (V located) (PP (P in) (NP (N Seattle))))))

上下文无关语法（Context-Free Grammer）

1）N表示一组非叶子节点的标注，例如{S、NP、VP、N...}
2）Σ表示一组叶子结点的标注，例如{boeing、is...}
3）R表示一组规则，每条规则可以表示为$X->Y_1Y_2...Y_n$，$X \in N$，$Y_i \in (N \cup \Sigma)$
4）S表示语法树开始的标注

句子the man sleeps就可以表示为(S (NP (DT the) (NN man)) (VP sleeps))

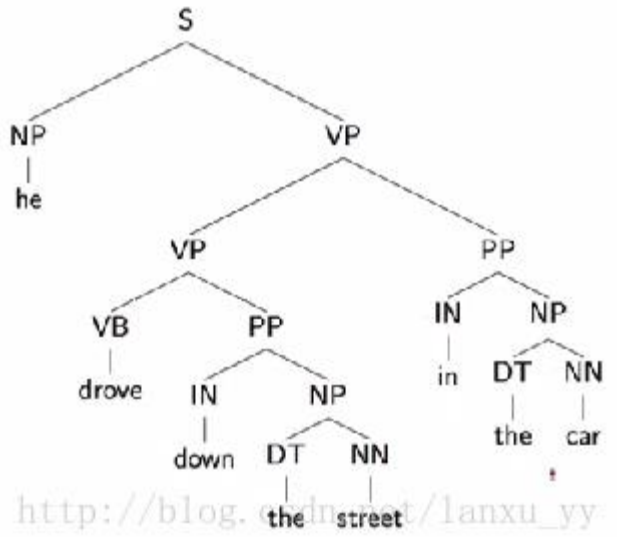
$N = \{S, NP, VP, PP, DT, Vi, Vt, NN, IN\}$
$S = S$
$\Sigma = \{sleeps, saw, man, woman, telescope, the, with, in\}$

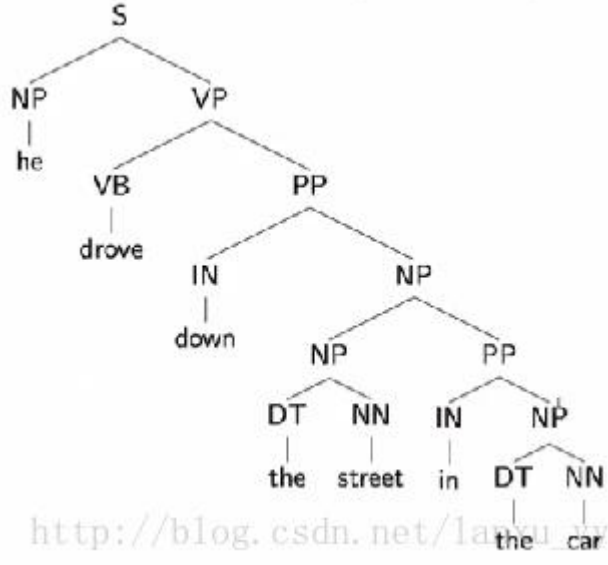
$R =$

S	→	NP	VP
VP	→	Vi	
VP	→	Vt	NP
VP	→	VP	PP
NP	→	DT	NN
NP	→	NP	PP
PP	→	IN	NP

Vi	→	sleeps
Vt	→	saw
NN	→	man
NN	→	woman
NN	→	telescope
DT	→	the
IN	→	with
IN	→	in

存在二义性





概率分布的上下文无关语法(Probabilistic Context-Free Grammar)

S	⇒	NP	VP	1.0
VP	⇒	Vi		0.4
VP	⇒	Vt	NP	0.4
VP	⇒	VP	PP	0.2
NP	⇒	DT	NN	0.3
NP	⇒	NP	PP	0.7
PP	⇒	P	NP	1.0

Vi	⇒	sleeps	1.0
Vt	⇒	saw	1.0
NN	⇒	man	0.7
NN	⇒	woman	0.2
NN	⇒	telescope	0.1
DT	⇒	the	1.0
IN	⇒	with	0.5
IN	⇒	in	0.5

训练算法：Penn Treebank通过手工的方法已经提供了一个非常大的语料数据集，就是从语料库中训练出PCFG所需要参数：

1）统计出语料库中所有的N与Σ

2）利用语料库中的所有规则作为R

3）针对每个规则A->B，从语料库中估算p(x)=p(A->B)/p(A)

Chomsky语法格式：每条规则只能是X -> Y1 Y2或者X -> Y

语法树预测算法：一个PCFG模型，包含N、Σ、R、S、p(x)等参数，且语法树Chomsky语法格式，当输入一个句子x1, x2, ... , xn时，如何计算句子对应的语法树？

1）暴力遍历：每个单词x有m = len(N)种取值，句子长度是n，每种情况存在n个规则，在$O(mn^2)$的情况下，可以判断出所有可能的语法树并计算出最佳的

2）动态规划：定义w[i, j, X]是第i个单词至第j个单词由标注X来表示的最大概率。如xi, xi+1, ... , xj，当X=PP时，子树可能是多种解释，如(P NP)或者(PP PP)，但是w[i, j, PP]代表是继续往上一层递归时，只选择当前概率最大的组合方式。特殊情况下，w[i, i, X] = p(X -> xi)，动态规划的方程表示为w[i, j, X] = max (p(X -> Y Z) * w(i, s, Y) * w(s+1, j, Z))

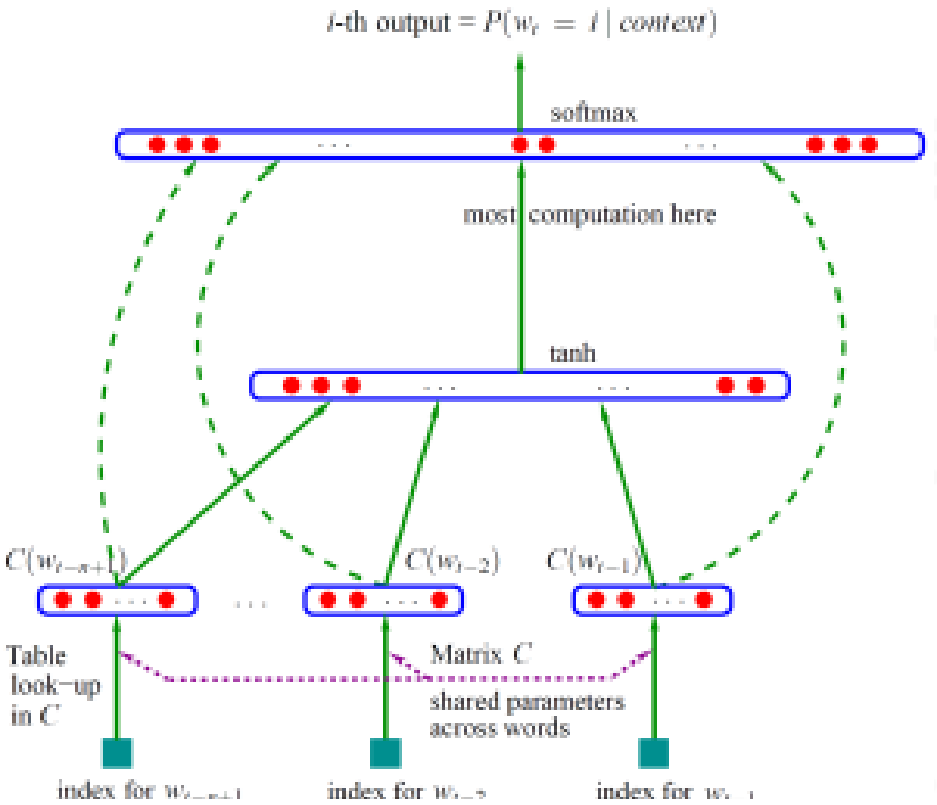
plot_topics_extraction_with_nmf_lda.py

NobleProg

9.2 词向量与语言模型

- One-hot Representation：把每个词表示为一个很长的向量。向量的维度是词表大小，其中大多数元素为 0，只有一个维度值为 1，这个维度就代表这个词
- Word Embedding：一种低维实数向量，让相关或者相似的词，在距离上更接近了，在训练语言模型，得到词向量
- 语言模型：给定一个字符串，看它是自然语言的概率 $P(w_1,w_2,\ldots,w_t)$
  $P(w_1,w_2,\ldots,w_t)=P(w_1)\times P(w2|w_1)\times P(w3|w_1,w_2)\times\ldots\times P(w_t|w_1,w_2,\ldots,w_{t-1})$
- 常用语言模型是在近似地求 $P(w_t|w_1,w_2,\ldots,w_{t-1})$,如 n-gram 模型就是用 $P(w_t|w_{t-n+1},\ldots,w_{t-1})$近似表示

Bengio 语言模型：
n-gram 模型



1）$w_{t-n+1},\cdots,w_{t-2},w_{t-1}$就是前n-1 个词，需要根据已知n-1个词预测下一个词 w_t。C(w) 表示词w 对应词向量，存在矩阵C（|V|×m ），|V|词表大小，m词向量维度。w到C(w)的转化就是从矩阵中取出一行

2）网络的输入层是将$C(w_{t-n+1}),\cdots,C(w_{t-2}),C(w_{t-1})$ 这n-1 个向量首尾相接拼起来，形成一个(n-1)m 维的向量，记为x

3）网络隐藏层就是普通的神经网络，直接使用d+Hx计算得到。d 是一个偏置项，使用tanh作为激活函数

4）网络输出层一共有|V| 个节点，每个节点yi表示下一个词为i的未归一化 log 概率，最后使用 softmax 激活函数将输出值y归一化成概率，y=b+Wx+Utanh(d+Hx)

NobleProg

C&W 的 SENNA

- 直接去近似 计算$P(w_1, w_2, \cdots, w_t)$：没有去求一个字符串的**概率**，而是求窗口连续n 个词的**打分** $f(w_{t-n+1}, \cdots, w_{t-1}, w_t)$，打分f 越高说明这句话越是正常的话；打分低说明这句话不是太合理；如果是随机把几个词堆积在一起，那肯定是负分
- 使用 pair-wise 的方法训练词向量，最小化目标函数：$\sum_{x \in X} \sum_{w \in D} \max\{0, 1-f(x)+f(x(w))\}$
- X为训练集所有连续的n元短语，D是整个字典，第一个求和枚举了训练语料中的所有的n元短语，作为正样本。第二个对字典的枚举是构建负样本。x(w) 是将短语x的最中间的那个词，替换成w。x 是正样本，x(w) 是负样本，f(x)是对正样本打分，f(x(w))是对负样本打分。最后希望正样本打分要比负样本打分至少高 1 分
- 把窗口中的n个词对应的词向量串成一个长的向量，经过一层网络（乘一个矩阵）得到隐藏层。输出层只有一个节点，表示得分。窗口大小n=11，字典大小 $|V|$=130000
- 应用：词性标注、命名实体识别、短语识别、语义角色标注

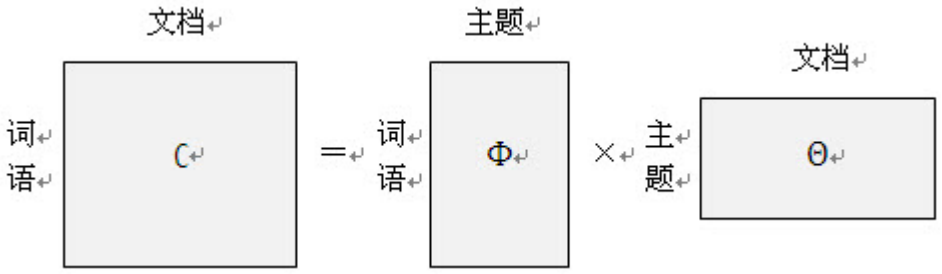
名称	训练语料及规模	词向量	特点	资源
C&W	English Wikipedia + Reuters RCV1 共 631M + 221M 词	130000 词 50 维	不区分大小写；经过有监督修正；训练了 7 周	测试代码、词向量 [链接]
C&W - Turian	Reuters RCV1 63M 词	268810 词 25、50、100、200 维	区分大小写；训练了若干周	训练代码、词向量 [链接]
M&H - Turian	Reuters RCV1	246122 词 50、100 维	区分大小写；用GPU训练了7天	词向量 [链接]
Mikolov	Broadcast news	82390 词 80、640、1600 维	不区分大小写；训练了若干天	训练、测试代码、词向量 [链接]
Huang 2012	English Wikipedia	100232 词 50 维	不区分大小写；最高频的6000词，每词有10种表示	训练、测试代码、语料及词向量 [链接]

NobleProg

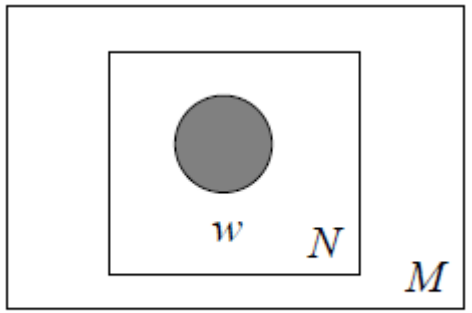
9.3 主题模型

- 生成模型：认为**一篇文章的每个词都是通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语”**这样一个过程得到的。如果我们要生成一篇文档，它里面的每个词语出现概率为：

$$p(\text{词语}|\text{文档}) = \sum_{\text{主题}} p(\text{词语}|\text{主题}) \times p(\text{主题}|\text{文档})$$

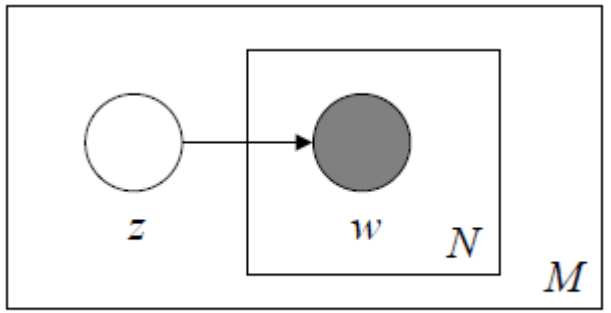


”文档-词语”矩阵表示每个文档中每个单词的词频，即出现的概率
”主题-词语”矩阵表示每个主题中每个单词的出现概率
”文档-主题”矩阵表示每个文档中每个主题出现的概率



(a) unigram

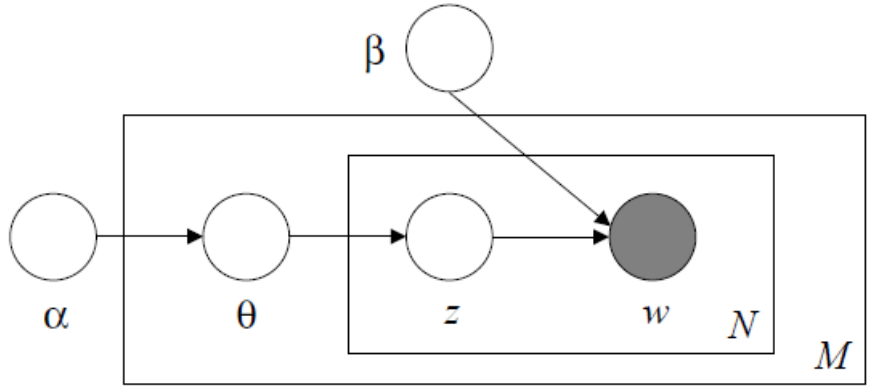
For each of the N words w_n:
Choose a word w_n ～ p(w);
1）N表示要生成文档的单词个数，w_n表示生成第n个单词w，p(w)表示单词w分布，可通过语料进行统计学习得到，如给一本书，统计各个单词在书中出现概率
2）通过训练语料获得一个单词的概率分布函数，然后根据概率分布函数每次生成一个单词，使用这个方法M次生成M个文档



(b) mixture of unigrams

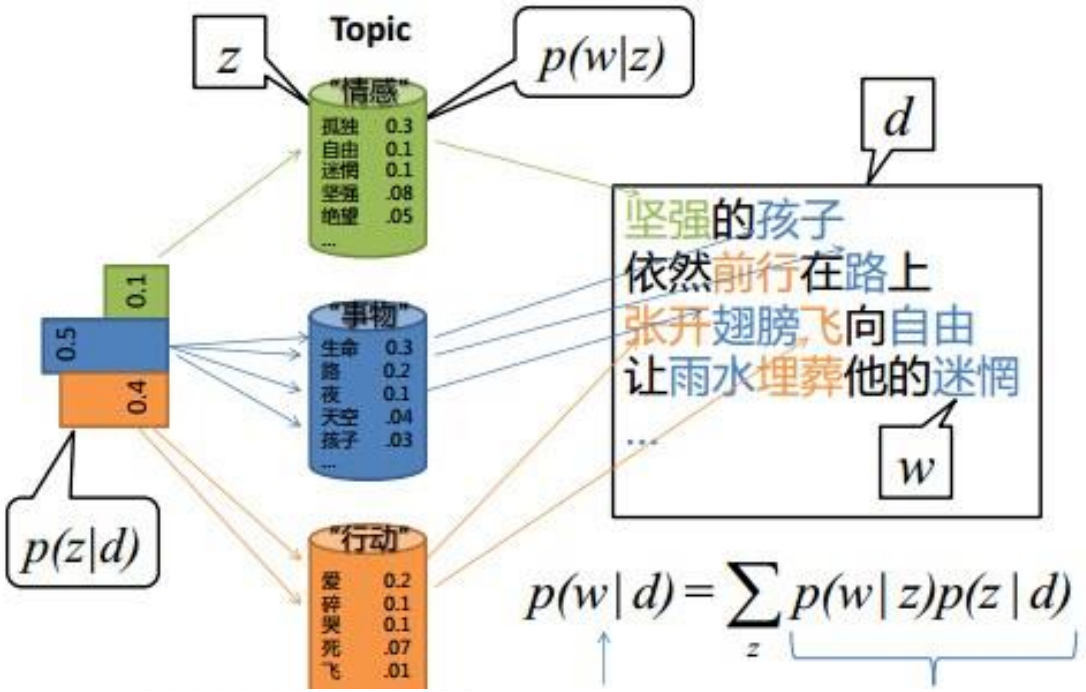
Choose a topic z ～ p(z);
For each of the N words w_n:
Choose a word w_n ～ p(w|z);
1）z表示一个主题，p(z)是主题概率分布，z通过p(z)按概率产生；p(w|z)是给定z时w分布，是一个k×V矩阵，k主题数，V单词数，每行表示这个主题对应单词的概率分布，即主题z所包含的各个单词概率，通过这概率分布按一定概率生成每个单词
2）首先选定一个主题z，主题z对应一个单词的概率分布p(w|z)，每次按这个分布生成一个单词，使用M次这个方法生成M份不同文档
3）只允许一个文档只有一个主题

LDA(Latent Dirichlet Allocation)
Choose parameter θ～p(θ);
For each of the N words w_n:
Choose a topic z_n ～ p(z|θ);
Choose a word w_n ～ p(w|z);



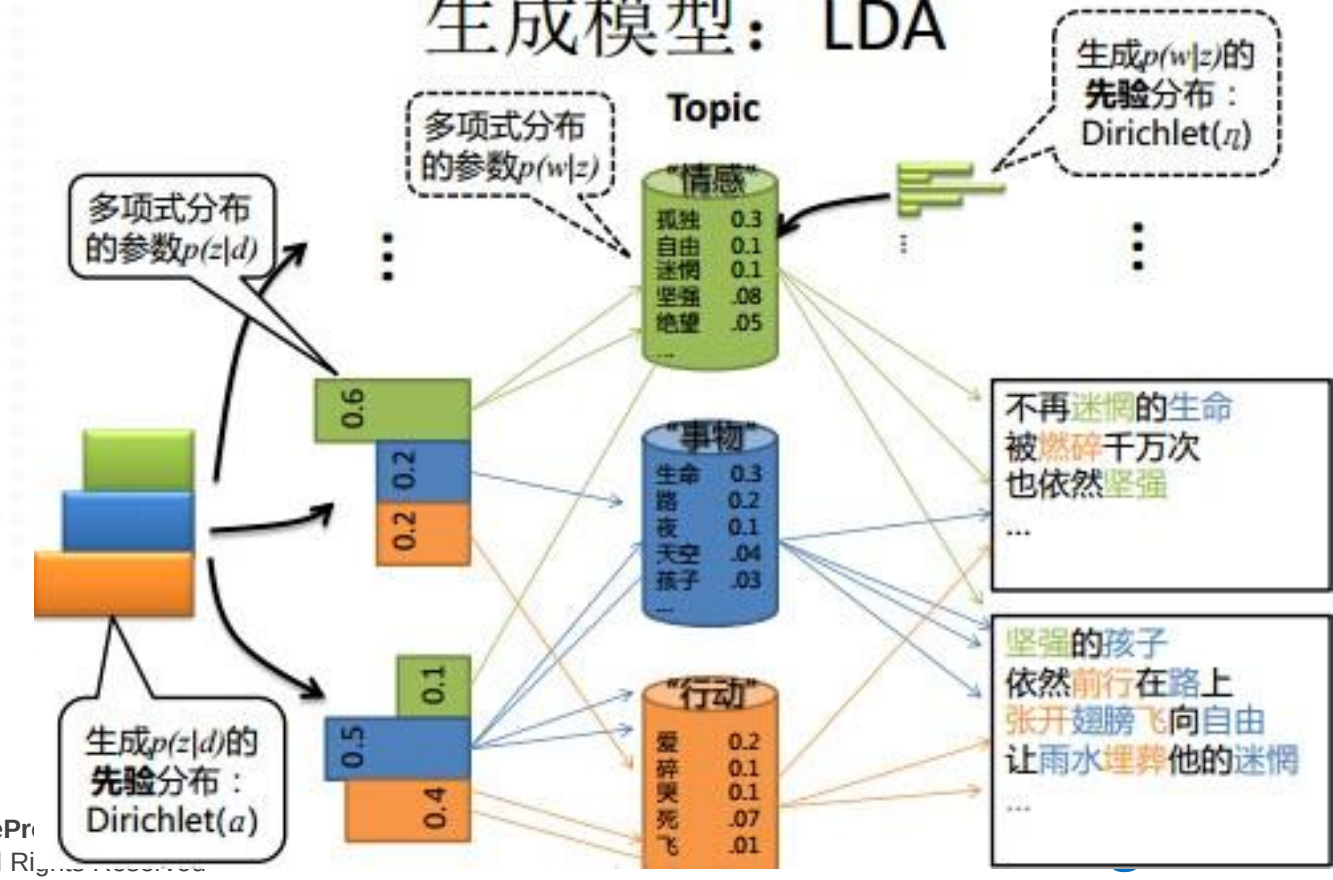
1）θ是一个主题向量，向量每一列是每个主题在文档出现的概率，为非负归一化向量；p(θ)是θ分布，为Dirichlet分布，即分布的分布；z_n表示选择的主题，p(z|θ)表示给定θ时主题z概率分布，为θ的值，即p(z=i|θ)= θ_i

生成模型： PLSA



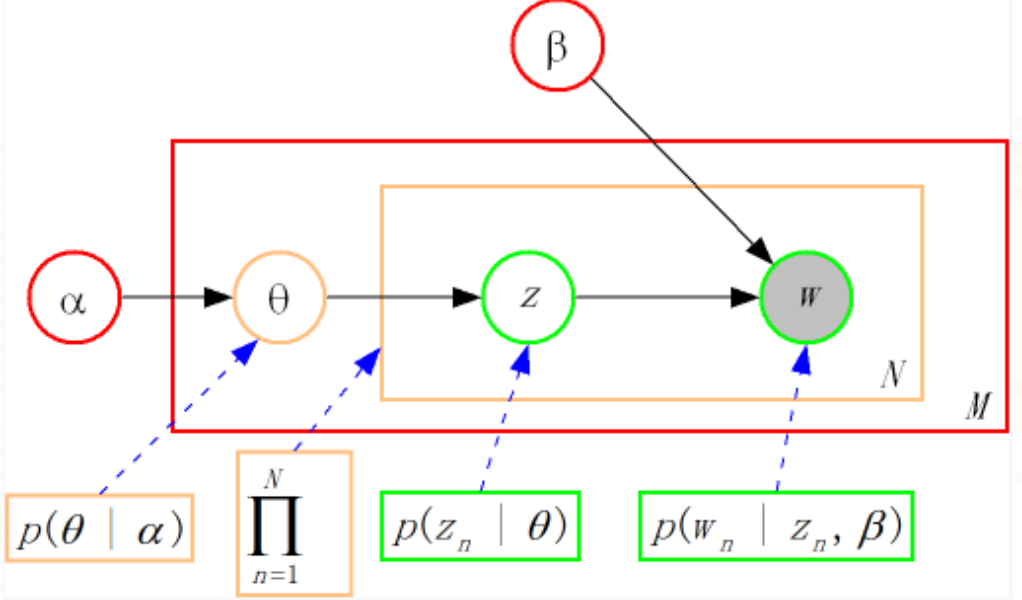
$$p(w \mid d) = \sum_z p(w \mid z) p(z \mid d)$$

生成模型： LDA



- 首先选定一个主题向量θ，确定每个主题被选择的概率。
- 然后在生成每个单词时候，从主题分布向量θ中选择一个主题z，按主题z的单词概率分布生成一个单词

1. corpus-level（红色）：α和β表示语料级别参数，每个文档都一样，生成过程只采样一次
2.document-level（橙色）：θ是文档级别变量，每个文档对应一个θ，每个文档产生主题z概率是不同，所有生成每个文档采样一次θ
3. word-level（绿色）：z和w都是单词级别变量，z由θ生成，w由z和β共同生成，一个单词w对应一个主题z
LDA模型主要是从给定的输入语料中学习训练两个控制参数α和β，可以用来生成文档
α：分布p(θ)需要一个向量参数，即Dirichlet分布的参数，用于生成一个主题θ向量
β：各个主题对应的单词概率分布矩阵p(w|z)
把w当做观察变量，θ和z当做隐藏变量，可通过EM算法学习出α和β，求解过程中遇到后验概率p(θ,z|w)无法直接求解，需要找一个似然函数下界来近似求解，使用基于分解factorization假设的变分法进行计算。每次E-step输入α和β，计算似然函数，M-step最大化这个似然函数，算出α和β，不断迭代直到收敛



$$p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta) = p(\theta \mid \alpha) \prod_{n=1}^{N} p(z_n \mid \theta) p(w_n \mid z_n, \beta)$$

二项分布 $$P(K = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

多项分布 $$P(x_1, x_2, ..., x_k; n, p_1, p_2, ..., p_k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

Beta分布 $$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

Dirichlet分布 $$f(x_1, x_2, ..., x_k; \alpha_1, \alpha_2, ..., \alpha_k) = \frac{1}{B(\alpha)} \prod_{i=1}^{k} x_i^{\alpha^i - 1}$$

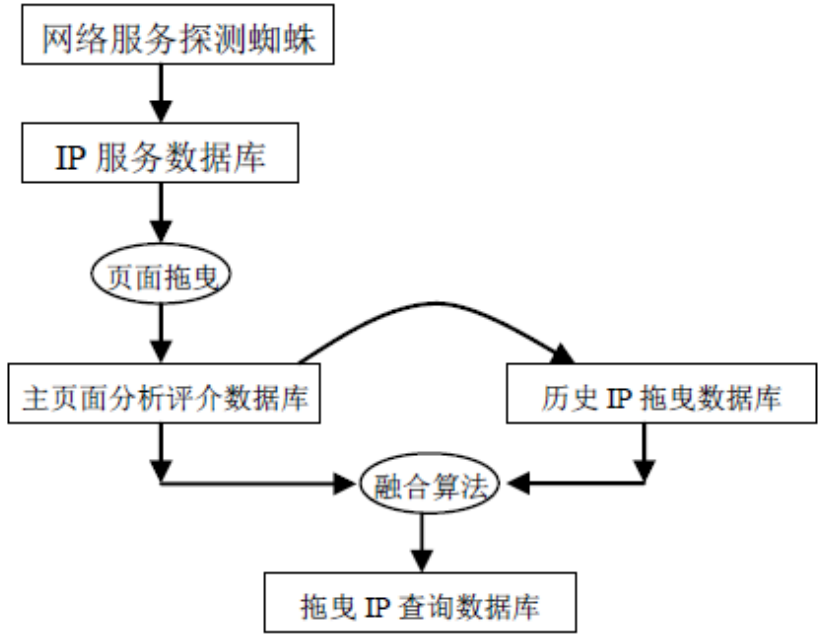
NobleProg

9.4 关键字树

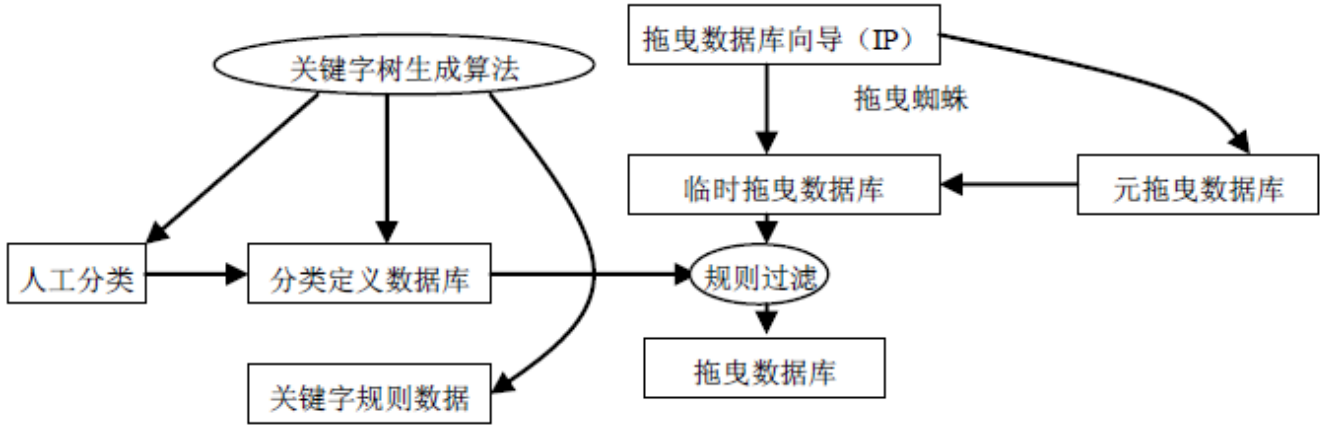
1．3 关键字生成树

核心关键字为“黄山”、“徽州”。以此产生一个供搜索使用的“关键字树”，关键字树的生成方法有三种：联想方法、字意组合、人工建立。树生成算法如下表所示：

	算法内容	算法排序说明
联想方法	对词组的常用“联连词”、组词、文章、口语、相关行政关系、学术关系、娱乐关系、旅游关系上的所有词组，生成一颗树	对频率、时间、重要性（权重）等进行排序；
字意组合	单纯从字的组合上进行，依据是字典解释进行排序与树生成。研究一下使用那几本词典或字典比较好；	对树深与含义进行排序，主要是对学术与古典文献进行排序；
人工建立	专业用的人工分类；用于对科学研究的分类树形成，主要是通过对学科分类进行树分解	对文献的关键字、摘要等进行人工整理；对索引进行年代、编者、写者、传者、研究者、记录者以及地点、位置、方向、文献特征词定义，进行描述



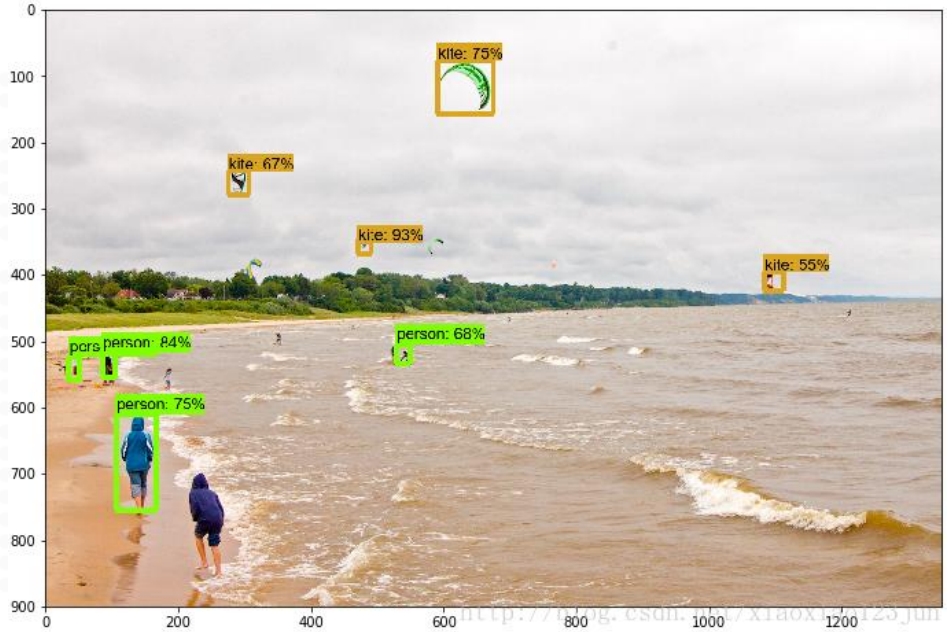
图二 索引数据库

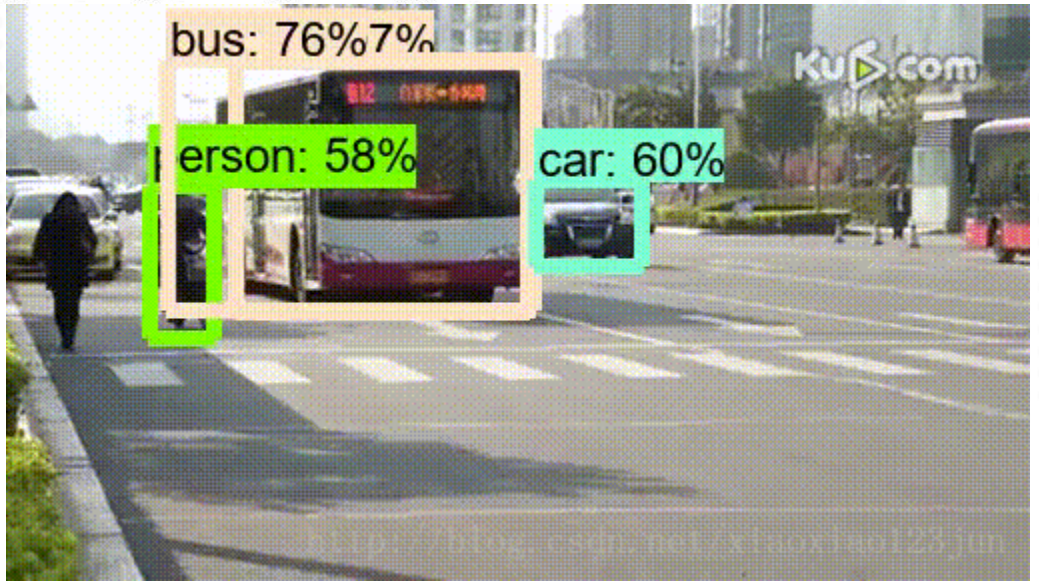


图三 拖曳数据库的生成

10. Tensorflow object detection

- 1.安装Protocol（下载后解压，将protoc.exe拷贝到c:\windows\system32）
- 2.安装git（下载后安装）
- 3.c:>pip install pillow lxml jupyter matplotlib
- 4.启动git，$>git clone https://github.com/tensorflow/models.git(c:\user\administrator)，把models文件夹拷贝到lib\site-packages\TensorFlow下
- 5.c:\..\models>protoc object_detection/protos/*.proto --python_out=.
- 6.c:\..models>jupyter-notebook
- 7.进入object_detection文件夹中的object_detection_tutorial.ipynb,点击Cell内的Run All，等待三分钟左右
- 8.下载opencv的cv2包，pip install opencv_python-3.2.0.8-cp35-cp35m-win_amd64.whl





http://blog.csdn.net/xiaoxiao123jun/article/details/76605928