DeepDriver解密之二

本文作者： 李明 蔡龙军

DeepDriver创建者：蔡龙军

Source codes: https://github.com/LongJunCai/DeepDriver

DeepDriver的DNN代码导读

-- DNN（在ArtifactNeuroNetworkV2基础上添加预训练过程）

总体分析

代码位置:

DNN (dnn/DNN.java)

预训练
learnSelf

继承

继承

ArtifactNeuroNetworkV2
(ann/ArtifactNeuroNetworkV2.java)

DNN网络

Sda二层网络

SparseAutoEncoder
(ann/SparseAutoEncoder.java)

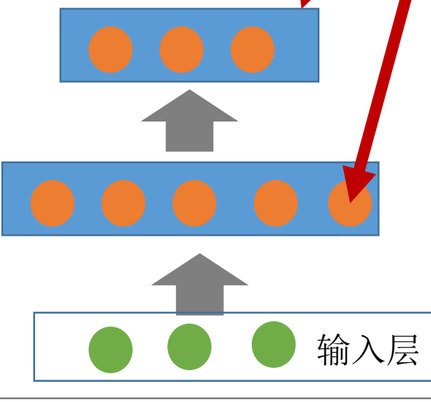
SparseAutoEncoderLayer

SparseAutoEncoderNeuro

SparseAutoEncoderCfgFromANNCfg
(ann/SparseAutoEncoderCfgFromANNCfg.java)

参数配置

SparseAutoEncoder



一层da网络

SparseAutoEncoderLayer
(ann/SparseAutoEncoderLayer.java)

继承

LayerImpV2
(ann/imp/LayerImpV2.java)

某个da神经元

SparseAutoEncoderNeuro
(ann/SparseAutoEncoderNeuro.java)

继承

NeuroUnitImpV3
(ann/imp/NeuroUnitImpV3.Java)

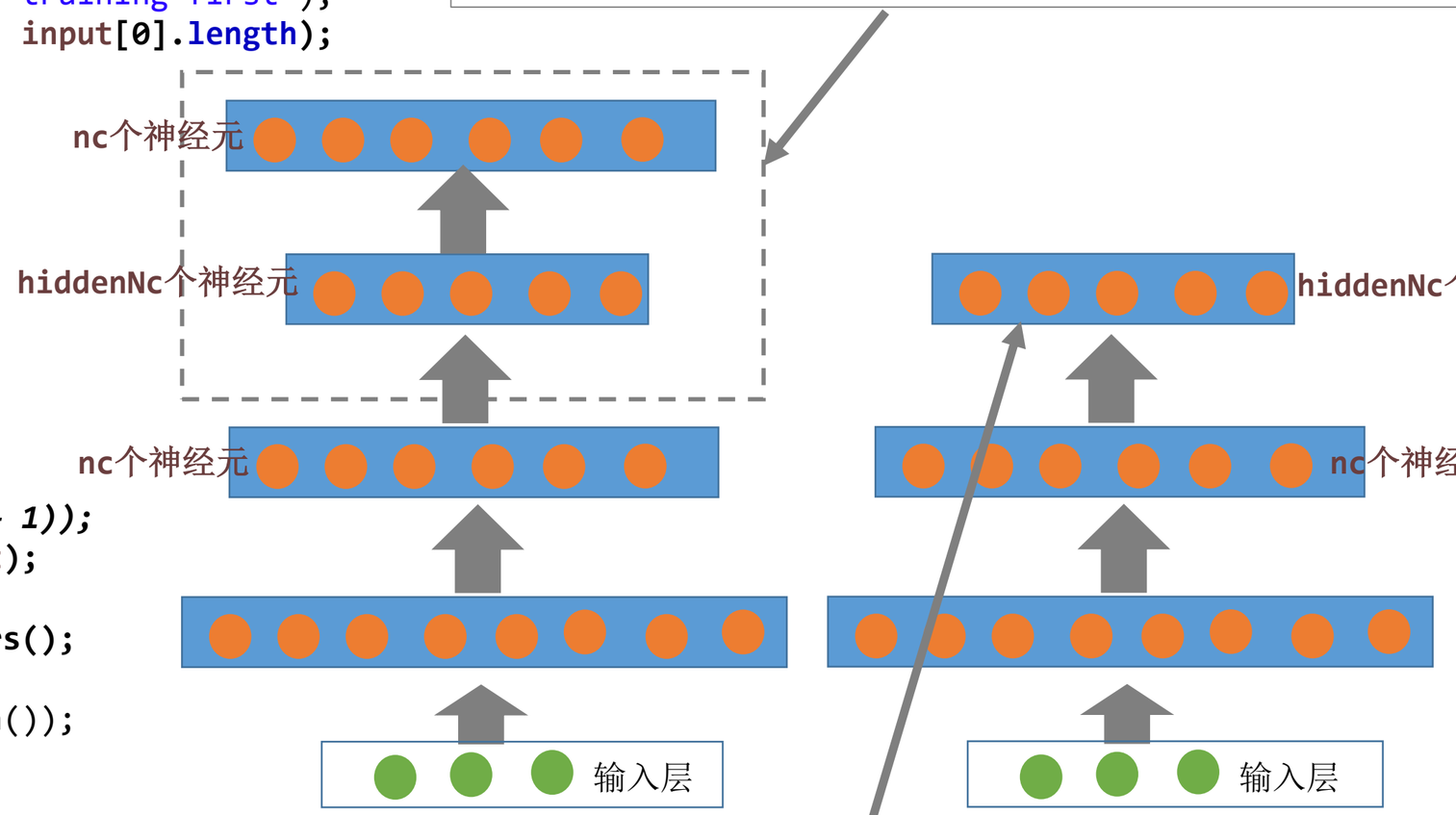
DNN的预训练过程 learnself

DNN (dnn/DNN.java)

```
public void learnSelf(InputParameters orignalPramameters) {
  double [][] input = normalizer.transformParameters(orignalPramameters.getInput());
  int ln = orignalPramameters.getLayerNum();
  firstLayer = createLayer();
  slLamda = orignalPramameters.getLamda();
  this.setFirstLayer(firstLayer);
  debugPrint("Begin to build up the DNN, start to pre-training first");
  firstLayer.buildup(null, input, createAcf(), false, input[0].length);
  //ILayer currentLayer = firstLayer;
  boolean sl4LastLayer = true;
  if (sl4LastLayer) {
  } else {
    ln = ln - 1;
  }
  for (int i = 0; i < ln; i++) {
    int hiddenNc = input[0].length;
    if (orignalPramameters.getNeuros() != null) {
      hiddenNc = orignalPramameters.getNeuros()[i];
    }
    System.out.println("Pre-training for layer "+(i + 1));
    double [][] newInput = caculateHiddenInputs(input);
    ILayer last= getLastLayer();
    InputParameters newPramaters = new InputParameters();
    newPramaters.setInput(newInput);
    newPramaters.setAlpha(orignalPramameters.getAlpha());
    newPramaters.setLamda(slLamda);
    newPramaters.setIterationNum(slLoopNum);
    int nc = newInput[0].length;
    //SparseAutoEncoderCfgFromANNCfg annCfg4DNN = new SparseAutoEncoderCfgFromANNCfg();
    //annCfg4DNN.setP(0.05);
    SparseAutoEncoder sae = new SparseAutoEncoder();
    sae.setUseNormalizer(false);
    if (i == ln - 1) {
      System.out.println("Pre-training for last layer. "+(i + 1));
```

建里一个2层的SparseAutoEncoder,第一层是hiddenNc,第二层是nc

```
SparseAutoEncoder sae = new SparseAutoEncoder();
newPramaters.setNeuros(new int[] {hiddenNc, nc});
sae.trainModel(newPramaters);
```

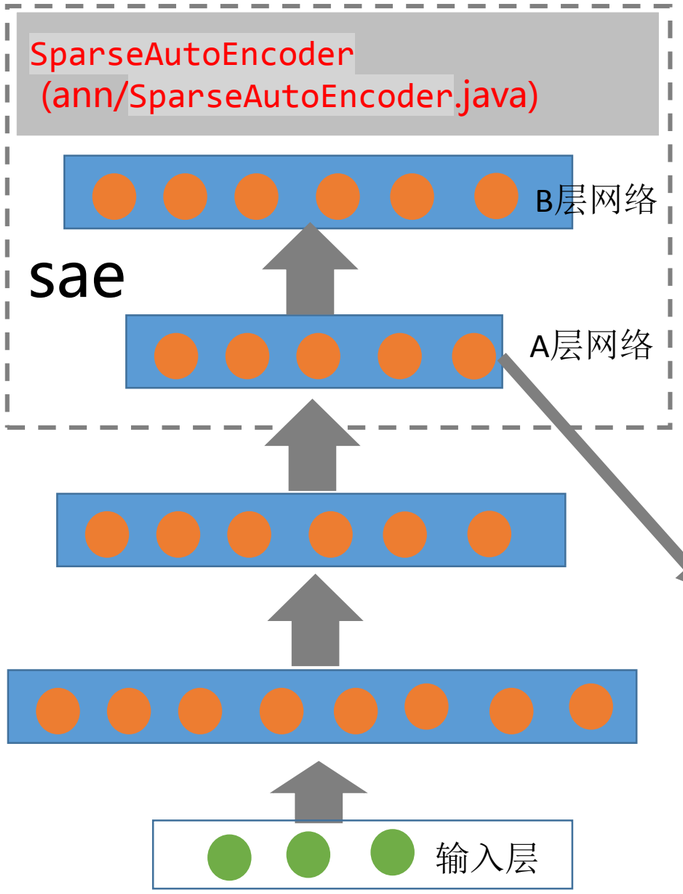


把sae的前半部分放入dnn中，并转化为V2网络

```
ILayer newHiddenLayer = sae.getFirstLayer().getNextLayer();
newHiddenLayer.setPos(i+1);
last.setNextLayer(newHiddenLayer);
```

SparseAutoEncoder

SparseAutoEncoderNeuro
(ann/SparseAutoEncoderNeuro.java)




```java
//参考 http://blog.csdn.net/evan123mg/article/details/40149601
//和一般bp的差异是 在计算局部梯度deltaZ时 多了一个 beta * (p/p1 + (1- p)/(1 - p1))
@Override
public void backPropagation(
  List<INeuroUnit> previousNeuros,
  List<INeuroUnit> nextNeuros,
  double [][] result,
  InputParameters parameters) {
  super.backPropagation(previousNeuros, nextNeuros, result, parameters);
  ISparseAutoEncoderCfg cfg = getSparseAutoEncoderCfg();
  if (cfg == null) {
    return ;
  }
  if (nextNeuros == null) {//B层网络
  } else {                  //A层网络
    if (layer.getPreviousLayer() == null) {
      return ;
    }
    for (int i = 0; i < deltaZ.length; i++) {
      double sumDelta = 0;
      double p = cfg.getP();
      double p1 = aas[i];
      sumDelta = - p/p1 + (1- p)/(1 - p1);
      deltaZ[i]=deltaZ[i]+cfg.getBeta()*(sumDelta)*activationFunction.deActivate(zzs[i]);
    }
  }
}
```

而加入了稀疏性后，神经元节点的误差表达式由公式：

$$\delta_i^{(2)} = \left(\sum_{j=1}^{s_2} W_{ji}^{(2)}\delta_j^{(3)}\right) f'(z_i^{(2)}),$$

变成公式：

$$\delta_i^{(2)} = \left(\left(\sum_{j=1}^{s_2} W_{ji}^{(2)}\delta_j^{(3)}\right) + \beta\left(-\frac{\rho}{\hat{\rho}_i} + \frac{1-\rho}{1-\hat{\rho}_i}\right)\right) f'(z_i^{(2)}).$$

B层网络的BP仅仅做原始V3的**backpropagation**

A层网络的BP先做原始V3的**backpropagation，然后再加上一个**

SparseAutoEncoder 稀疏编码的含义

稀疏编码是对网络的隐含层的输出有了约束，即隐含层节点输出的平均值应尽量为0，这样的话，大部分的隐含层节点都处于非activ

sparse autoencoder损失函数表达式为：

$$J_{\text{sparse}}(W,b) = J(W,b) + \beta \sum_{j=1}^{s_2} \text{KL}(\rho||\hat{\rho}_j),$$

编码层的神经元尽量的少被激活，

后面那项为KL距离，其表达式如下：

$$\text{KL}(\rho||\hat{\rho}_j) = \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

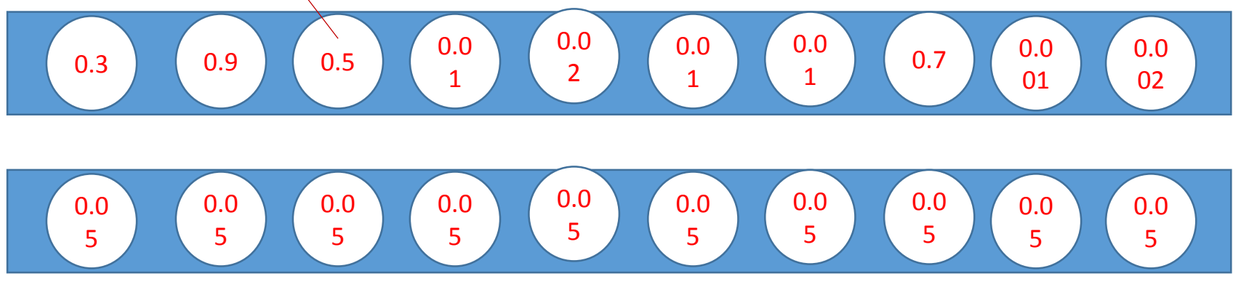
问:超参数- β 如何设置？

隐含层节点输出平均值求法如下：

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x^{(i)}) \right]$$

编码层中的第j个神经元在所有样本中的平均激活值

其中的参数一般取很小，比如说0.05，也就是小概率发生事件的概率。这说明要求隐含层的每一个节点的输出均值接近0.05



编码层的激活值

希望每个神经元激活值均等于p=0.05

KL距离做惩罚项，其中KL约小说明两个序列(分布)越接近

请开始你们的表演