决策模型提取和压缩框架

模型越复杂，进行预测所需要的计算代价就越多，特别是当我们使用深度学习模型的时候，计算的时间代价与卷积核的边长平方成正比，还与卷积核数量成正比。

解析一个单层卷积神经网络中某一层的计算复杂度为：



这要求我们需要降低参数规模，降低模型复杂度，尽量在预测阶段使用参数规模更小的模型。

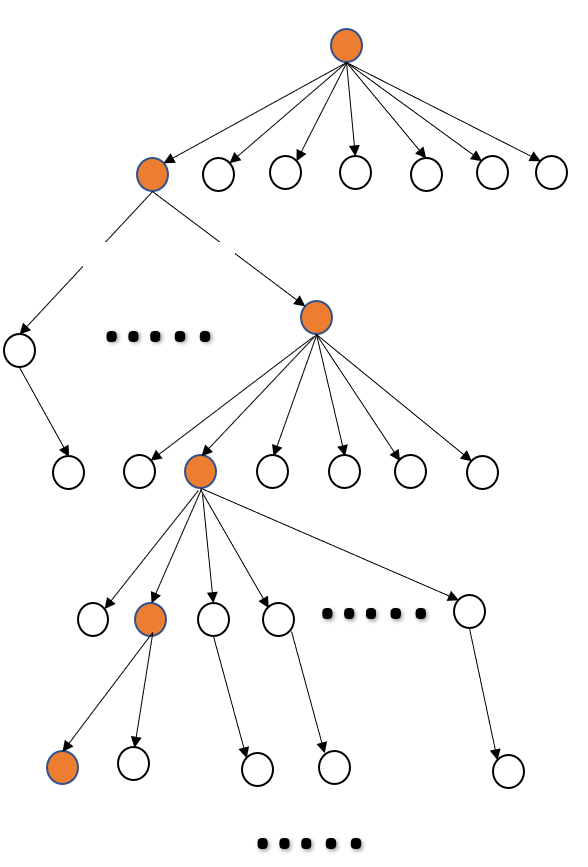
除此以外，模型的层级越深，则需要解析的次数就越多，因而需要减少层次。

为了解决这一问题，我们提出一种层次树算法的提取框架，用来压缩深度网络的参数规模，以达到快速解码的目的。这两种模型的主要区别在：

结合思路：

（1）如果我们能够获取神经网络习得的知识，并借助依赖于层级决策的模型表达出来，那么解释一个特定的决策将会容易很多。

（2）使用已训练的神经网络创建软决策树的方法，它比直接从训练数据中学习的决策树有着更优的泛化能力。

（3）相较于从训练数据中直接学习的决策树，软决策树通过层级决策模型把 DNN 所习得的知识表达出来，具体决策解释容易很多。平衡了泛化性与可解释性。

**硬决策树**

（1）对样本空间进行层次划分，每个样本被划分到相应路径的分支树中。

（2）每个子树只会拟合这些来自父节点划分出的样本。

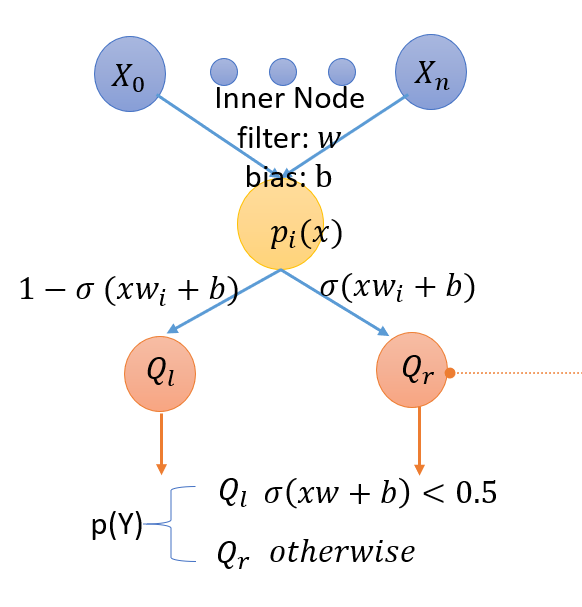
（3）越远离根节点的划分越容易过拟合。需要进行剪枝操作提高模型的泛化能力。

**软决策树**

（1）以某种接受概率确定样本被划分到相应到相应分支的概率。

（2）每个子树会拟合全体样本集合，样本的权重由通过该路径到达叶子节点的路径概率确定。

（3）分支树具有更好的泛化能力，树结构更为精简，路径权重基于梯度训练动态调整。



软决策树用分支接受概率确定样本的分支选择，以单层二元软决策树为例，定义第Gate函数为：

表示选择右子树的接受概率， 为选择左子树的接受概率。其中该内节点的参数， 软决策树的平衡度。

定义第叶节点的静态分布为：

是第叶节点的学习参数。

图片包含 屏幕截图

已生成极高可信度的说明

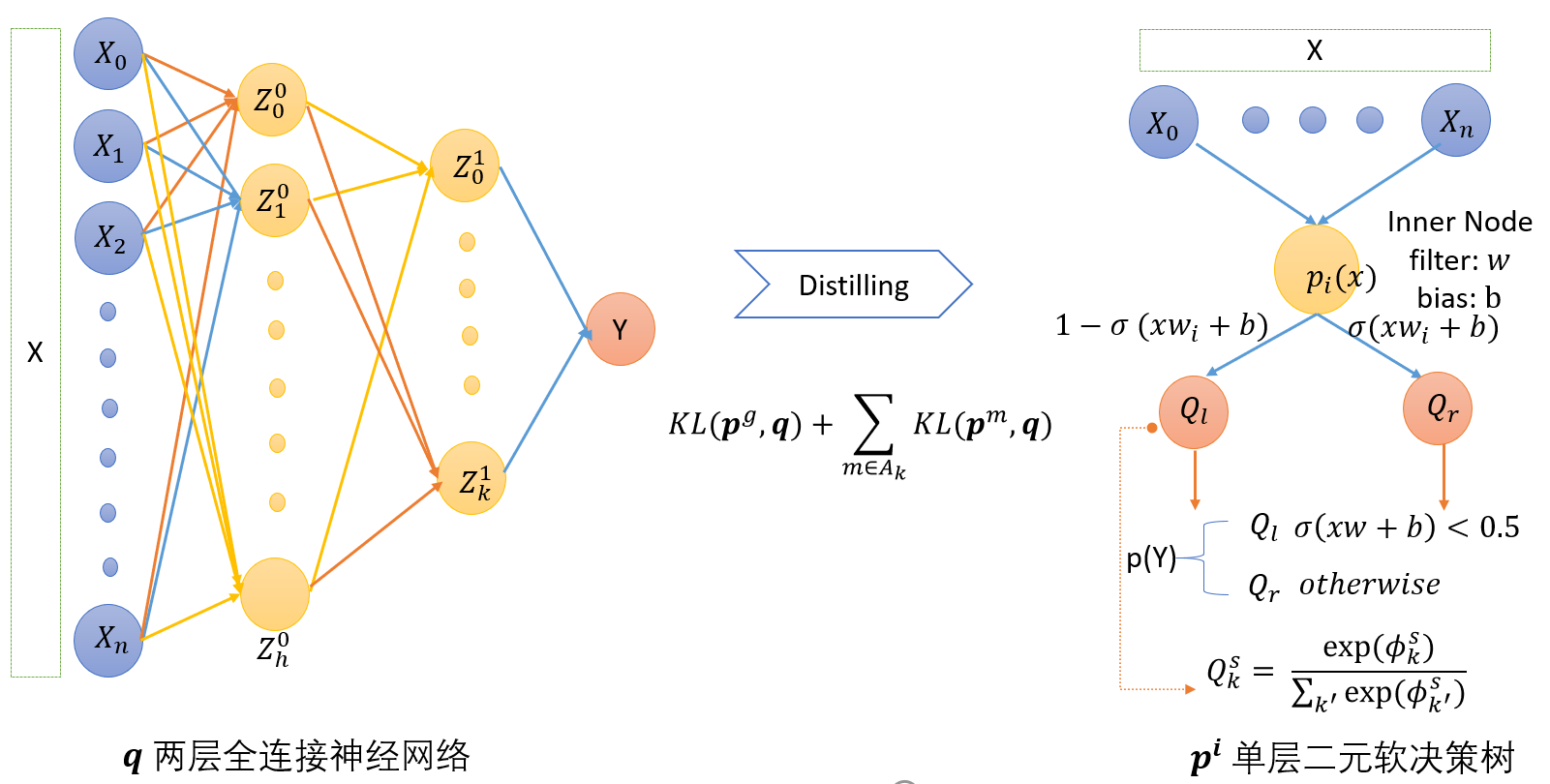
为了在模型泛化性和可解释性之间取得了一个平衡。我们使用深度神经网络训练一个决策树，让这个决策树去模仿神经网络发现的输入输出函数。

（1）专家模型概率分布

（2）广义泛化模型的概率分布

（3） 表示提取的目标概率分布

提取过程（Distillation）最小化如下的目标函数：



最小化损失函数如下:

对于多层二元软决策树等价于:

其中分别是到达叶节点的路径概率，目标分布，叶节点静态分布。为了平衡二元树的分支选择，惩罚项为：

优化算法：

符号：

类别数目：*n*；树深度：*d*；

批量大小：M；样本集；

样本的维度：*dim*；

过滤器的数目：(*inner nodes*) ；

假设各个过滤器的顺序为}；

过滤器的参数：是*dim*长度的向量，是一个标量。

静态分布的数目：(*leaf nodes*) ；

假设各个叶子的顺序为}；

叶子节点的参数：}

静态分布：代表叶子节点中第*k*类的可能性，与输入数据无关。

叶子节点的父节点*ord*满足：

训练好的深度神经网络：代表DNN输出第*k*类的可能性大小，表示一个*n*维度的向量，代表各个类别的可能性。

算法：

步骤1：初始化各个节点的参数

For  *i* in :

, 高斯随机发生器

For  *i* in :

For  *j*  in :

if : 设置为1而不损失通用性

if : 高斯随机发生器

步骤2：准备训练数据集

随机从训练集中选取一批量为*m*的样本，它通过从输入空间采样来完成。通过参数时间序列仿真可以降低采样的计算复杂度，或使用样本生成器 with GAN。

（有待核实）

minibatch

While m--:

产生一个样本；

是一个样本集

步骤3：训练

While直至收敛：

#增量梯度初始化

For *i* in

0

0

End

For *l* in

For *k* in

0

End

End

While 直至收敛：

For *m* in [0,M-1] (For each *x* in the batch )

#计算各个节点路径的可能性

1

For

if 2 == 0

=\*()

#是sigmoid函数

if 2 == 1

=\*(1 ())

End

#计算每个叶子节点的交叉熵 }

For *l* in

End

End

For *i* in #计算的梯度

For *m* in [0,M-1]

#计算导数= for all *j* 在子树*i*

For *j* in #初始化

= 0

End

=()(1-())

=

=

For *j* in subtree 2*i*+1

End

For *j* in subtree 2*i*+2

End

#添加每个叶子节点上的损失梯度

For *l* in

End

End

#在每个内部节点上添加惩罚梯度在

For *j* in

dpenalty \*(0.5\* - 0.5\*)

+ dpenalty

End

End

For *i* in #计算的梯度

For *m* in [0,M-1]

#计算导数= for all *j* 在子树*i*

For *j* in #初始化

= 0

End

=()(1-())

=

=

For *j* in subtree 2*i*+1

End

For *j* in subtree 2*i*+2

End

#添加每个叶子节点上的损失梯度

For *l* in

End

End

#在每个内部节点上添加惩罚梯度在

For *j* in

dpenalty \*(0.5\* - 0.5\*)

+ dpenalty

End

End

#计算的梯度

For in

For *l* in

For *m* in [0,M-1]

End

End

End

#更新

For *i* in

End

For *l* in

For in

End

End

计算loss，核实停止条件

End while