1. 首先，需要介绍main函数的流程。
2. 一个深度学习应用必须包含三个部分：深度学习模型、模型参数、数据。因此main函数中包括三个部分：

load\_param.load(params,data\_addr,global\_info);//从文件中读取模型参数

load\_data(data\_addr.train\_data\_info,train\_data);//从文件读取数据，因为有四份数据，而且某些可以不要，因此有一系列判断条件和读取函数。

Model model(input\_size,params[0],init\_mat\_from\_file,file\_path);//新建模型

1. Model的类型有两种，一种为无监督模型，另一种为有监督模型，这在参数文件中已经定义了。这两种模型的流程有差异，因此需要加if判断并调用model的不同函数。

无监督模型首先需要pretrain过程，pretrain过程只需要训练数据，不需要标签，当然如果标签数据也存在的话，需要对模型的分类器进行训练；之后还要查看是否有fine-tune开关打开，如果打开，则对整个模型进行一次训练（之前的训练是逐层训练的）；最后根据训练得到的权值矩阵对测试数据的标签进行预测。

有监督模型流程简单一些，只需要对整个模型进行训练，这跟无监督模型fine-tune时调用的函数一模一样，后面还会详细介绍。

如下所示：

if(global\_info.params[params\_name[MODELTYPE]] == "UnsuperviseModel"){

model.pretrain(train\_data,params[0]);

if(data\_addr.train\_labels\_info.name != ""){

model.train\_classifier(train\_data,train\_labels,params[0]);

}

if(finetune\_data\_switch && global\_info.params[params\_name[FINETUNESWITCH]] == "ON"){

LogOut << "Begin finetuning!" << endl;

cout << "Begin finetuning!" << endl;

model.train(finetune\_data,finetune\_labels,params[0]);

}

if(data\_addr.test\_data\_info.name != ""){

LogOut << "Begin predicting!" << endl;

cout << "Begin predicting!" << endl;

pred\_labels = model.predict(test\_data,test\_labels,params[0]);

if(data\_addr.test\_labels\_info.name != ""){

pred\_acc = model.predict\_acc(pred\_labels,test\_labels);

}

}

}

else{

model.train(train\_data,train\_labels,params[0]);

if(data\_addr.test\_data\_info.name != ""){

LogOut << "Begin predicting!" << endl;

cout << "Begin predicting!" << endl;

pred\_labels = model.predict(test\_data,test\_labels,params[0]);

if(data\_addr.test\_labels\_info.name != ""){

pred\_acc = model.predict\_acc(pred\_labels,test\_labels);

}

}

}

1. 最后需要保存训练和预测结果
2. 下面介绍Model.cpp的几个主要函数
3. Pretrain(),前面已经介绍，该函数是针对无监督模型进行的预训练过程。

首先判断最后一层是不是分类器，如果是分类器，预训练只训练前面N-1层（N为层数），最后一层需要用前面提到的函数进行单独训练。如果最后一层不是分类器，则所有层全部训练。

每一层训练调用的是各层的pretrain函数，后面会单独介绍，第一层训练的数据是从main函数中读取的，其他各层训练的数据都是上一层训练后调用forwardpropagate函数得到的。

1. Train\_classifer()函数

该函数就是训练分类器，会判断最后一层是不是分类器，如果是，则调用分类器模块的train函数。

1. Create\_module函数，根据参数文件构建模型中的各层。
2. Train函数

对整个模型进行训练的函数。

对整个模型进行训练时，首先要定义模型的代价函数，这是在function包中的ModelCost.cpp中完成的。该类继承自CostFunction类，所有的代价函数类必须定义value\_gradient函数给出代价函数值得计算和梯度向量的计算结果。

之后要定义用什么优化器对上面定义的代价函数进行优化，模型中的优化默认为sgd优化。优化函数的具体流程会在后面详细介绍。

1. 下面介绍各个模块，即模型中的各层。

一个模型可以包括多层，各层的算法也可以不同，目前主要包括的无监督算法模块主要有五种：RBM、Sparse Coding、 Auto-Encoder、ConvolutionalRBM、Pooling。所有这些模块都必须实现以下几个函数：pretrain用于各层的预训练、forwardpropagate用于前向传播（即通过输入和已训练好的权值矩阵得到输出），backwardpropagate用于后向传播误差，根据上一层传递的误差计算本层误差并传递到下一层，process\_delta为根据本层误差为下一层误差做一些处理，calculate\_grad\_using\_delta指根据本层计算得到的误差计算本层的梯度向量。

1. 下面逐一介绍各个模块的pretrain过程。

RBM模块

上图可以看出，主要调用CD\_k算法，CD\_k函数主要调用Gibbs\_hvh，该过程是v->h,h->v的过程。

Sparse Coding：  


Sparse Coding流程中最重要的一个函数式optimize，这个函数是通过Optimizer类定义的，需要在optimizer包中查找。如果method为cg和lbfgs，则流程基本相同，只不过search\_dir函数有所不同，这要在对应的文件中查找该函数。例如lbfgs的search\_dir函数要在LbfgsOptimizer.cpp中找到search\_direction函数，需要注意的是，整个优化过程都会调用一个函数，即value\_gradient函数，这个函数在SCFeatureCost.cpp中可以找到。Sgd优化方法流程稍有不同，它是一种在线更新的算法，所以将数据再次分为小块，每个小块都计算函数值和梯度值，将权值更新一次。

Auto-Encoder(对应的文件包括：module/AutoEncoder.cpp, function/SAECostFunction.cpp, optimizer包)

 其实Auto-Encoder的流程与SC的流程完全一样，只不过现在没有将数据拆成和SC一样的小块进行训练而已。

Pooling层不需要pretrain函数

CRBM的流程如RBM的流程，只不过运算类型为卷积。

1. 下面介绍train函数的流程，这对应于无监督模型的微调过程和有监督模型的训练过程。



可以看出其基本流程也是与前面的相同，但是不同的地方在于它的函数值和梯度向量的计算函数value\_gradient。下面是该函数的计算流程。



图中调用的函数forwardpropagate和backpropagate，calculate\_grad\_using\_delta以及process\_delta函数，在各个module中都有实现，因此都需要优化。