Java Deep Learning Library and Continues RBM

Tianyi Chen, PhD@Johns Hopkins University

1 Abstract

Java as a popular programming language, even lacks a Deep Learning Library somehow due to the lack of support from powerful mathematical calculation package , like numpy in python. To fix it up, a Deep learning library named as DLcty is created. The documentation is included in this writeup. Besides it, in order to fix up the shortage of binary case of RBM, DBN, a continues value RBM, DBN is also constructed.

2 DLcty Documentation

DLcty provides interfaces:

1. Binary case RBM
2. Binary case DBN
3. MLP

2.1 Preliminaray

1.To allocate the methods built in DLcty, DLcty.jar file should be attached as external library on the Java project.
2. Add external jar file: math3 of apache.

2.2 LoadData

Input data should convert to RealMatrix Class in math3 library.

`DataStream.load(String fileX, String Y)`

function can be used to load data.
For example:

```
String inputX="german_numer01_X.out";
String inputY="german_numer01_Y.out";

Map<String, RealMatrix> infoMap=DataStream.loadData(inputX, inputY);

RealMatrix X=infoMap.get("X");
RealMatrix Y=infoMap.get("Y");
```

where X is the Data matrix, the size of X is m by n, where m is number of samples, n is number of features, Y is a column matrix saveing label.

2.3 RBM

To run RBM on input data, we need to do the following steps:

Step 1: create an instance of RBM class.

```
int num_features=X.getColumnDimension();
int num_samples=X.getRowDimension();
//RBM(int num_visible, int num_hidden)
RBM rbm=new RBM(num_features, num_features/2);
```

Step 2: set parameters, including Contrasive-Divergence K, learning rate α, training epoch, and batch size. For examples:

```
int K=50;
double alpha=0.1;
rbm.setK(K);
rbm.setAlpha(alpha);
int training_epoches=100;
int batch_size=111;
```

Step 3: train RBM:

```
// train RBM
for(int t=0;t<training_epoches;t++){
        Random randomGenerator = new Random();

        // select a batch of data randomly
        RealMatrix inputData=otherTools.stochasticSubmatrix(X, batch_size, randomGenerator);

        // update parameters of rbm1 with SGD solver
        rbm.updateParams(inputData);

        // calculate reconstruction loss value
        double loss=rbm.CrossEntropy(inputData);
        System.out.println("loss value:"+loss);

}
```

The output on console should be like

```
training epoch:0 loss value:-0.7586267301050253
training epoch:1 loss value:-0.42361143804903734
training epoch:2 loss value:-0.5062772086049139
training epoch:3 loss value:-0.39472787021550987
training epoch:4 loss value:-0.3386319033418799
training epoch:5 loss value:-0.2885894840596366
training epoch:6 loss value:-0.1784915414427655
training epoch:7 loss value:-0.2563098524576439
training epoch:8 loss value:-0.20920941463328502
training epoch:9 loss value:-0.19087708883369667
training epoch:10 loss value:-0.19953132725629807
```

To obtain the weights of edges of RBM, and bias of hidden layer, visible layer. Run the following command:

```
// get weights of edges
RealMatrix weights=rbm.getW();

// get bias of hidden layer
RealMatrix hbias=rbm.getHbias();

// get bias of visible layer
RealMatrix vbias=rbm.getVbias();
```

2.4 DBN

To run DBN on input data, we need to do the following steps:

Step 1: Specify hidden layer size.

```
// three hidden layers, the first hidden layer contains 10 neurons, the second hidden layer
// contains 100 neurons, the third one has 50 units
int[] hiddensizes={10,100,50};
```

Step 2: Set various parameters, including batch size, CD-k, pretrain epoch, learning rate α.

```
int batch_size=111;
int K=10;
int pre_training_epoches=10;
double alpha=0.1;
```

Step 3: create an instance of DBN class

```
//DBN(int inputSize, int[] hiddenSizes, int outputSize, int K, int pretraining_epoch, double alpha)
DBN dbn=new DBN(num_features,hiddensizes,2,K,pre_training_epoches,alpha);
```

Step 4: train DBN

```
dbn.pretraining(X, batch_size);
```

The output on console should be like:

```
RBM layer 0:
pretraining_epoch:0 loss value:-0.881350017834405
pretraining_epoch:1 loss value:-0.8390507134244802
pretraining_epoch:2 loss value:-0.7762359475699605
pretraining_epoch:3 loss value:-0.7302838685660593
pretraining_epoch:4 loss value:-0.654893559361064
pretraining_epoch:5 loss value:-0.5922925681225922
pretraining_epoch:6 loss value:-0.7269764976888777
pretraining_epoch:7 loss value:-0.49953019658958897
pretraining_epoch:8 loss value:-0.44569765990116317
pretraining_epoch:9 loss value:-0.3966535711223775
RBM layer 1:
pretraining_epoch:0 loss value:-0.6499980083516476
pretraining_epoch:1 loss value:-0.35198477897381214
pretraining_epoch:2 loss value:-0.2386644401683641
pretraining_epoch:3 loss value:-0.16297411827225888
pretraining_epoch:4 loss value:-0.1795433610478966
pretraining_epoch:5 loss value:-0.1761257791313093
pretraining_epoch:6 loss value:-0.15042153063761374
pretraining_epoch:7 loss value:-0.14418155386811432
pretraining_epoch:8 loss value:-0.13350506524899258
pretraining_epoch:9 loss value:-0.13071603138811735
RBM layer 2:
pretraining_epoch:0 loss value:-0.7596952344886028
pretraining_epoch:1 loss value:-0.6522887226764831
pretraining_epoch:2 loss value:-0.6025875190341131
pretraining_epoch:3 loss value:-0.5659029584113164
pretraining_epoch:4 loss value:-0.5373838809163487
pretraining_epoch:5 loss value:-0.527928251170187
pretraining_epoch:6 loss value:-0.5386322026886582
pretraining_epoch:7 loss value:-0.5364908634840797
pretraining_epoch:8 loss value:-0.343428541703665
pretraining_epoch:9 loss value:-0.5728706249450987
RBM layer 3:
pretraining_epoch:0 loss value:-0.7681064220297842
pretraining_epoch:1 loss value:-0.6657686762639301
pretraining_epoch:2 loss value:-0.6537710337535471
pretraining_epoch:3 loss value:-0.6921091214259893
pretraining_epoch:4 loss value:-0.6890022501853523
pretraining_epoch:5 loss value:-0.6845761178581661
pretraining_epoch:6 loss value:-0.6831882728400412
pretraining_epoch:7 loss value:-0.6750384463330193
pretraining_epoch:8 loss value:-0.6749699540065863
pretraining_epoch:9 loss value:-0.6699778637401456
```

To obtain $i-th$ RBM layer, we can

```
RBM rbm=dbn.getRbmlayers().get(i);
```

Then we can obtain the weights, bias of RBM layer as the RBM instance in last part.

2.5 MLP

To run MLP, we should do the following steps
Step 1: create MLP instance.

```
MLP mlp=new MLP(X,num_features,hiddensizes,outputneurons,needPretrain,lr,batch_size,pretrainingepoch,alpha,K);
```

1. $needPretrain$ is a boolean value. $needPretrain = true$ will allocate DBN to pretrain, $needPretrain = false$ will initialize the weights and bias of MLP randomly from Gaussian distribution.
2. lr is the learning rate of training MLP
3. alpha is the learning rate of DBN pretrain
4. K is the CD-K in DBN pretrain.

For example,

```
int num_features=X.getColumnDimension();
int num_samples=X.getRowDimension();
int training_epoches=1000;
int pretrainingepoch=10;
int batch_size=23;

double lr=0.3; // learning rate of MLP
double alpha=0.1; // learning rate of DBN pretraining
int K=10;

int[] hiddensizes={14,14}; // hidden layer sizes.
MLP mlp=new MLP(X,num_features,hiddensizes,2,false,lr,batch_size,pretrainingepoch,alpha,K);
```

Step 2: train MLP

```
mlp.train(X, Y, batch_size, training_epoches);
```

Step 3: test MLP

```
RealMatrix test_Y=mlp.convertLabelToVector(testY,2);
mlp.predict(testX, test_Y);
```

References

[1] Frederick Jelinek, *Statistical Methods for SpeechRecognition* MIT press, pp 45-57, Sept 1996.

[2] S. Young, *A Review of Large Vocabulary Continuous Speech Recognition,* IEEE Signal Processing Magazine, pp 45-57, Sept 1996.

[3] Rabiner, L *A tutorial on hidden Markov models and selected applications in speech recognition,* Proceedings of the IEEE Volume:77 , Issue: 2 Feb 1989.

[4] Bahl, L.R. *Acoustic Markov models used in the Tangora speech recognition system,* Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on 11-14 Apr 1988.