# deep neural network programming exercises

Generated by Doxygen 1.8.13

# Contents

# Chapter 1

# Neural network exercises

Learning neural networks by exercises.

### Prerequisties

Libraries:

- icc,mkl

### Installing

sudo apt-get install g++ g++-multilib build-essential
install icc,mkl
make test_mlr test_ffnn

### Datasets

datasets/train.csv, datasets/test.csv

the training/validation data sets uses the MNIST datasets in csv format:
https://www.kaggle.com/c/digit-recognizer/data

### Running the test

#### multi-classes logistic regression

Using dropout with keep_prob=0.85 to perform regularization in the input layer
./test_mlr -n 100 -a 0.003

```
Cost of train/validation at epoch    0 :  0.58074164 0.35054046
Cost of train/validation at epoch   10 :  0.27973989 0.26971540
Cost of train/validation at epoch   20 :  0.27052698 0.27189568
Cost of train/validation at epoch   30 :  0.26301974 0.27001908
Cost of train/validation at epoch   40 :  0.25950569 0.26645681
Cost of train/validation at epoch   50 :  0.25770107 0.26931667
Cost of train/validation at epoch   60 :  0.26052868 0.27073887
Cost of train/validation at epoch   70 :  0.25515199 0.26677442
Cost of train/validation at epoch   80 :  0.25009876 0.26658201
Cost of train/validation at epoch   90 :  0.24885687 0.26687291
Cost of train/validation at epoch  100 :  0.24961139 0.26613927
validation set accuracy:0.928571
```

#### feed-forward neural network with two hidden-layers

Initialize the hidden layer dimensions with {768,512}
activation types with {"ReLU","ReLU"}
keep probabilities in dropout regularization with {0.85,0.4,0.5}:
./test_ffnn -n 50 -a 0.001

```
Cost of train/validation at epoch    0 :  0.75169951 0.24952073
Cost of train/validation at epoch   10 :  0.11169609 0.08198518
Cost of train/validation at epoch   20 :  0.06751236 0.07157799
Cost of train/validation at epoch   30 :  0.04579250 0.06736103
Cost of train/validation at epoch   40 :  0.03172652 0.06837747
Cost of train/validation at epoch   50 :  0.02747416 0.06784185
validation set accuracy:0.983095
```

The accuracy is much better than the logistic regression

#### convolutional neural network

The test initialize the convolutional network with three convolutional and pooling layers
Input layer: dropout keep_prob=0.9,
1st Conv2d layer: {filter_size=3, padding=1, stride=1, n_channel=16}
1st Pool layer: {filter_size=2,stride=2,n_channel=16}
2nd Conv2d layer: {filter_size=3, padding=1, stride=1, n_channel=32}
2nd Pool layer: {filter_size=2,stride=2,n_channel=32}
3rd Conv2d layer: {filter_size=3, padding=1, stride=1, n_channel=64}
3rd Pool layer: {filter_size=2,stride=2,n_channel=64}
1st Hidden layer: dim=512, dropout keep_prob=0.5
2nd Hidden layer: dim=256, dropout keep_prob=0.6

using batch_size=128 and Adam optimization with learning rate decay:

./test_conv -n 10 -a 0.0025

```
Cost of train/validation at epoch    0 :  0.51284289 0.12400043
Cost of train/validation at epoch    1 :  0.10023427 0.06492750
Cost of train/validation at epoch    2 :  0.06424299 0.04772324
Cost of train/validation at epoch    3 :  0.04936956 0.04153788
Cost of train/validation at epoch    4 :  0.03802959 0.04588846
Cost of train/validation at epoch    5 :  0.03091952 0.03321058
Cost of train/validation at epoch    6 :  0.02083942 0.03290170
Cost of train/validation at epoch    7 :  0.01673818 0.03719120
Cost of train/validation at epoch    8 :  0.01175991 0.03252161
Cost of train/validation at epoch    9 :  0.00862098 0.03288486
Cost of train/validation at epoch   10 :  0.00722232 0.03297896
validation set accuracy:0.992024
```

The accuracy improves, while we could see that we're still over-fitting, batch-normalization will be updated in the further version.

## License

MIT

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# Class Documentation

## 3.1 dnn Class Reference

`#include <dnn.h>`

### Public Member Functions

- dnn (int n_f, int n_c, layers *head, layers *tail)
- ~dnn ()

  *destructor, clean the memory space*
- void insert_after_input (layers *new_layer)
- void insert_before_output (layers *new_layer)

  *insert hidden layer before the output layer*
- void initialize_layers (const int &n_sample, const float &_lambda, const string &optimizer, const bool &batch_norm)

  *initialize all layer variables*
- void initialize_layers_caches (const int &n_sample, const bool &is_bp)

  *initialize layer caches*
- void clear_layers_caches (const bool &is_bp)

  *clar layer caches*
- int get_argmax (const float *x, const int &range)

  *return argmax of given vector in a range*
- void print_layers ()

  *print parameters of all layers*
- void train_and_dev (const vector< float > &X_train, const vector< int > &Y_train, const vector< float > &X_dev, const vector< int > &Y_dev, const int &n_train, const int &n_dev, const int num_epochs, float learning_rate, float lambda, int batch_size, string optimizer, bool batch_norm, bool print_cost, int print_↩ period)
- void predict (const vector< float > &X, vector< int > &Y_prediction, const int &n_sample)
- float predict_accuracy (const vector< float > &_X, const vector< int > &Y, vector< int > &Y_prediction, const int &n_sample)
- void shuffle (float *X, float *Y, int n_sample)
- void batch (const float *X, const float *Y, float *X_batch, float *Y_batch, int batch_size, int batch_id)
- void batch (const float *X, float *X_batch, int batch_size, int batch_id)
- void feed_transposed (const float *X, float *XT, int batch_size, int N)
- void multi_layers_forward (const bool &eval)

*multi layers forward propagate and activation*

- void multi_layers_backward (const float *Y, const int &n_sample)

  *multi layers backward propagate to get gradients*
- void **gradient_descent_optimize** (const float &initial_learning_rate, const int &num_epochs, const int &step)
- void **Adam_optimize** (const float &initial_learning_rate, const float &beta_1, const float &beta_2, const int &num_epochs, const int &epoch_step, const int &train_step)
- float cost_function (const float *Y, const int &n_sample)

### 3.1.1 Detailed Description

deep neural networks exercises with C++ logistic regression, feed forward neural network, convolutional neural network, recurrent neural network (to be updated)

**Author**

qiong zhu

**Version**

0.2 18/04/2019

### 3.1.2 Constructor & Destructor Documentation

#### 3.1.2.1 dnn()

```
dnn::dnn (
            int n_f,
            int n_c,
            layers * head,
            layers * tail )
```

constructor with input and output layers

**Parameters**

| | |
|---|---|
| *n_f* | No. of features in X |
| *n_c* | No. of classes in Y |
| *head* | input layer |
| *tail* | output layer |

### 3.1.3 Member Function Documentation

#### 3.1.3.1 batch() [1/2]

```
void dnn::batch (
            const float * X,
            const float * Y,
            float * X_batch,
            float * Y_batch,
            int batch_size,
            int batch_id )
```

Obtain a batch datasets from the full datasets (used for training/developing)

**Parameters**

| | |
|---|---|
| *X* | pointer to data X |
| *Y* | pointer to data Y |
| *X_batch* | pointer to datasets batched from X |
| *Y_batch* | pointer to datasets batched from Y |
| *batch_size* | batch size |
| *batch_id* | No. of batches extracted, used as an offset |

**Returns**

batched dataset stored in X_batch,Y_batch

#### 3.1.3.2 batch() [2/2]

```
void dnn::batch (
            const float * X,
            float * X_batch,
            int batch_size,
            int batch_id )
```

Obtain a batch datasets from the full datasets (used for predicting)

**Parameters**

| | |
|---|---|
| *X* | pointer to data X |
| *X_batch* | pointer to datasets batched from X |
| *batch_size* | batch size |
| *batch_id* | No. of batches extracted, used as an offset |

**Returns**

batched dataset stored in X_batch

#### 3.1.3.3 cost_function()

```
float dnn::cost_function (
            const float * Y,
            const int & n_sample )
```

Calculate the mean cost using the cross-entropy loss
input: output->A, Y

update:
J=-Y.*log(output->A)
cost=sum(J)/n_sample

output:

**Parameters**

| | |
|---|---|
| *Y* | pointer to the datasets Y |
| *n_sample* | No. of samples in the datasets the mean cost |

#### 3.1.3.4 feed_transposed()

```
void dnn::feed_transposed (
            const float * X,
            float * XT,
            int batch_size,
            int N )
```

feed the batch training data transposed

**Parameters**

| | |
|---|---|
| *X* | batched data |
| *XT* | transposed feeded data |
| *batch_size* | No. of samples |
| *feature/classes* | dimension |

#### 3.1.3.5 insert_after_input()

```
void dnn::insert_after_input (
            layers * new_layer )
```

insert hidden layer after the input layer

**Parameters**

| | |
|---|---|
| *new_layer* | the new layer to be inserted |

#### 3.1.3.6 predict()

```
void dnn::predict (
            const vector< float > & X,
            vector< int > & Y_prediction,
            const int & n_sample )
```

Perform prediction for the given unlabeled datasets

**Parameters**

| | |
|---|---|
| *X* | datasets X |
| *Y_prediction* | output integer vector containing the predicted labels |
| *n_sample* | No. of samples in the datasets |

**Returns**

the predicted labels stored in Y_prediction

#### 3.1.3.7 predict_accuracy()

```
float dnn::predict_accuracy (
            const vector< float > & _X,
            const vector< int > & Y,
            vector< int > & Y_prediction,
            const int & n_sample )
```

Predict and calculate the prediction accuracy for the given labeled datasets

**Parameters**

| | |
|---|---|
| *X* | datasets X |
| *Y* | datasets Y (labels) |
| *Y_prediction* | output integer vector containing the predicted labels |
| *n_sample* | No. of samples in the datasets |

**Returns**

accuracy , and the predicted labels stored in Y_prediction

#### 3.1.3.8 shuffle()

```
void dnn::shuffle (
            float * X,
            float * Y,
            int n_sample )
```

Shuffle the datasets

**Parameters**

| | |
|---|---|
| *X* | data X |
| *Y* | data Y |
| *n_sample* | range (or No. of samples) in X,Y to be shuffled |

**Returns**

X,Y being shuffled

#### 3.1.3.9 train_and_dev()

```
void dnn::train_and_dev (
            const vector< float > & X_train,
            const vector< int > & Y_train,
            const vector< float > & X_dev,
            const vector< int > & Y_dev,
            const int & n_train,
            const int & n_dev,
            const int num_epochs = 100,
            float learning_rate = 0.001,
            float lambda = 0,
            int batch_size = 128,
            string optimizer = "gradient_descent",
            bool batch_norm = false,
            bool print_cost = false,
            int print_period = 1 )
```

Perform stochastic batch gradient training and evaluation using the validation(developing) data sets

**Parameters**

| | |
|---|---|
| *X_train* | training datasets X |
| *Y_train* | training datasets Y |
| *X_dev* | validation datasets X |
| *Y_dev* | validation datasets Y |
| *n_train* | No. of training samples |
| *n_dev* | No. of validataion samples |
| *num_epochs* | No. of epochs to train |
| *learning_rate* | learning rate of of gradients updating |
| *lambda* | L2-regularization factor |

**Parameters**

| | |
|---|---|
| *batch_size* | batch size in the stochastic batch gradient training |
| *optimizer* | if "gradient_descent", uses mini-batch gradient descent, if "Adam", use Adam optimizer |
| *batch_norm* | if true, batch normalization is used |
| *print_cost* | print the training/validation cost every print_period epochs if print_cost==true |
| *print_period* | print results every period epochs |

**Returns**

weights and bias W,b updated in the object

The documentation for this class was generated from the following files:

- dnn.h
- dnn.cpp

## 3.2 layers Class Reference

Collaboration diagram for layers:

# Index