

deep neural network programming exercises

Generated by Doxygen 1.8.13



# Contents

<b>1</b>	<b>Neural network exercises</b>	<b>1</b>
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	dnn Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.1.2	Constructor & Destructor Documentation . . . . .	6
3.1.2.1	dnn() . . . . .	6
3.1.3	Member Function Documentation . . . . .	6
3.1.3.1	batch() [1/2] . . . . .	7
3.1.3.2	batch() [2/2] . . . . .	7
3.1.3.3	cost_function() . . . . .	8
3.1.3.4	insert_after_input() . . . . .	8
3.1.3.5	predict() . . . . .	8
3.1.3.6	predict_accuracy() . . . . .	9
3.1.3.7	shuffle() . . . . .	9
3.1.3.8	train_and_dev() . . . . .	10
3.2	layers Class Reference . . . . .	10
	<b>Index</b>	<b>11</b>



# Chapter 1

## Neural network exercises

Learning neural networks by exercises.

### Prerequisites

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****License**

MIT

## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">dnn</a>	.....	<a href="#">5</a>
<a href="#">layers</a>	.....	<a href="#">10</a>





## 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)  
*clear 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 **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

<i>n_f</i>	No. of features in X
<i>n_c</i>	No. of classes in Y
<i>head</i>	input layer
<i>tail</i>	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

<i>X</i>	pointer to data X
<i>Y</i>	pointer to data Y
<i>X_batch</i>	pointer to datasets batched from X
<i>Y_batch</i>	pointer to datasets batched from Y
<i>batch_size</i>	batch size
<i>batch_id</i>	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

<i>X</i>	pointer to data X
<i>X_batch</i>	pointer to datasets batched from X
<i>batch_size</i>	batch size
<i>batch_id</i>	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:

$J = -Y \cdot \log(\text{output} \rightarrow A)$

cost = sum(J)/n\_sample

output:

#### Parameters

<i>Y</i>	pointer to the datasets Y
<i>n_sample</i>	No. of samples in the datasets the mean cost

### 3.1.3.4 insert\_after\_input()

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

insert hidden layer after the input layer

#### Parameters

<i>new_layer</i>	the new layer to be inserted
------------------	------------------------------

### 3.1.3.5 predict()

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

Perform prediction for the given unlabeled datasets

#### Parameters

<i>X</i>	datasets X
<i>Y_prediction</i>	output integer vector containing the predicted labels
<i>n_sample</i>	No. of samples in the datasets

**Returns**

the predicted labels stored in Y\_prediction

**3.1.3.6 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**

<i>X</i>	datasets X
<i>Y</i>	datasets Y (labels)
<i>Y_prediction</i>	output integer vector containing the predicted labels
<i>n_sample</i>	No. of samples in the datasets

**Returns**

accuracy , and the predicted labels stored in Y\_prediction

**3.1.3.7 shuffle()**

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

Shuffle the datasets

**Parameters**

<i>X</i>	data X
<i>Y</i>	data Y
<i>n_sample</i>	range (or No. of samples) in X,Y to be shuffled

**Returns**

X,Y being shuffled

### 3.1.3.8 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

<i>X_train</i>	training datasets X
<i>Y_train</i>	training datasets Y
<i>X_dev</i>	validation datasets X
<i>Y_dev</i>	validation datasets Y
<i>n_train</i>	No. of training samples
<i>n_dev</i>	No. of validation samples
<i>num_epochs</i>	No. of epochs to train
<i>learning_rate</i>	learning rate of gradients updating
<i>lambda</i>	L2-regularization factor
<i>batch_size</i>	batch size in the stochastic batch gradient training
<i>optimizer</i>	if "gradient_descent", uses mini-batch gradient descent, if "Adam", use Adam optimizer
<i>batch_norm</i>	if true, batch normalization is used
<i>print_cost</i>	print the training/validation cost every print_period epochs if print_cost==true
<i>print_period</i>	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

batch  
dnn, [6](#), [7](#)

cost\_function  
dnn, [7](#)

dnn, [5](#)  
batch, [6](#), [7](#)  
cost\_function, [7](#)  
dnn, [6](#)  
insert\_after\_input, [8](#)  
predict, [8](#)  
predict\_accuracy, [9](#)  
shuffle, [9](#)  
train\_and\_dev, [9](#)

insert\_after\_input  
dnn, [8](#)

layers, [10](#)

predict  
dnn, [8](#)  
predict\_accuracy  
dnn, [9](#)

shuffle  
dnn, [9](#)

train\_and\_dev  
dnn, [9](#)