# Educational Deep Learning Network

Generated by Doxygen 1.9.2

**1 Todo List** 1

**2 Class Index** 3

2.1 Class List . . . . . 3

**3 File Index** 5

3.1 File List . . . . . 5

**4 Class Documentation** 7

4.1 Categorization Class Reference . . . . . 7

4.1.1 Detailed Description . . . . . 7

4.1.2 Constructor & Destructor Documentation . . . . . 7

4.1.2.1 Categorization() [1/4] . . . . . 7

4.1.2.2 Categorization() [2/4] . . . . . 8

4.1.2.3 Categorization() [3/4] . . . . . 8

4.1.2.4 Categorization() [4/4] . . . . . 8

4.1.3 Member Function Documentation . . . . . 8

4.1.3.1 close_enough() [1/2] . . . . . 8

4.1.3.2 close_enough() [2/2] . . . . . 9

4.1.3.3 normalize() . . . . . 9

4.1.3.4 probabilities() . . . . . 9

4.1.3.5 size() . . . . . 9

4.2 dataItem Class Reference . . . . . 10

4.2.1 Detailed Description . . . . . 10

4.2.2 Constructor & Destructor Documentation . . . . . 10

4.2.2.1 dataItem() [1/3] . . . . . 10

4.2.2.2 dataItem() [2/3] . . . . . 11

4.2.2.3 dataItem() [3/3] . . . . . 11

4.2.3 Member Function Documentation . . . . . 11

4.2.3.1 data_size() . . . . . 11

4.2.3.2 data_values() . . . . . 11

4.2.3.3 label_size() . . . . . 11

4.2.3.4 label_values() . . . . . 12

4.2.4 Member Data Documentation . . . . . 12

4.2.4.1 data . . . . . 12

4.2.4.2 label . . . . . 12

4.3 Dataset Class Reference . . . . . 12

4.3.1 Detailed Description . . . . . 13

4.3.2 Constructor & Destructor Documentation . . . . . 13

4.3.2.1 Dataset() [1/3] . . . . . 13

4.3.2.2 Dataset() [2/3] . . . . . 14

4.3.2.3 Dataset() [3/3] . . . . . 14

4.3.3 Member Function Documentation . . . . . 14

4.3.3.1 batch() . . . 14
4.3.3.2 data() . . . 14
4.3.3.3 data_size() . . . 15
4.3.3.4 data_vals() . . . 15
4.3.3.5 inputs() . . . 15
4.3.3.6 item() . . . 16
4.3.3.7 label_size() . . . 16
4.3.3.8 label_vals() . . . 16
4.3.3.9 labels() . . . 17
4.3.3.10 push_back() . . . 17
4.3.3.11 readTest() . . . 17
4.3.3.12 set_lowerbound() . . . 18
4.3.3.13 set_number() . . . 18
4.3.3.14 shuffle() . . . 18
4.3.3.15 size() . . . 19
4.3.3.16 split() . . . 19
4.3.3.17 stack() . . . 19
4.3.3.18 stacked_data_vals() . . . 20
4.3.3.19 stacked_label_vals() . . . 20
4.3.4 Member Data Documentation . . . 20
4.3.4.1 path . . . 20
4.4 Layer Class Reference . . . 21
4.4.1 Detailed Description . . . 22
4.4.2 Constructor & Destructor Documentation . . . 22
4.4.2.1 Layer() [1/2] . . . 22
4.4.2.2 Layer() [2/2] . . . 22
4.4.3 Member Function Documentation . . . 22
4.4.3.1 allocate_batch_specific_temporaries() . . . 22
4.4.3.2 backward() . . . 23
4.4.3.3 forward() . . . 23
4.4.3.4 input() [1/2] . . . 24
4.4.3.5 input() [2/2] . . . 24
4.4.3.6 input_size() . . . 24
4.4.3.7 intermediate() . . . 24
4.4.3.8 output_size() . . . 24
4.4.3.9 set_activation() [1/3] . . . 24
4.4.3.10 set_activation() [2/3] . . . 25
4.4.3.11 set_activation() [3/3] . . . 25
4.4.3.12 set_number() . . . 25
4.4.3.13 set_topdelta() . . . 25
4.4.3.14 set_uniform_biases() . . . 26
4.4.3.15 set_uniform_weights() . . . 26

4.4.3.16 update_dw() . . . . . 26
4.4.4 Friends And Related Function Documentation . . . . . 26
4.4.4.1 Net . . . . . 26
4.4.5 Member Data Documentation . . . . . 27
4.4.5.1 activation_name . . . . . 27
4.5 Matrix Class Reference . . . . . 27
4.5.1 Detailed Description . . . . . 28
4.5.2 Constructor & Destructor Documentation . . . . . 28
4.5.2.1 Matrix() [1/2] . . . . . 28
4.5.2.2 Matrix() [2/2] . . . . . 28
4.5.3 Member Function Documentation . . . . . 28
4.5.3.1 addvh() . . . . . 29
4.5.3.2 axpy() . . . . . 29
4.5.3.3 colsize() . . . . . 29
4.5.3.4 data() [1/2] . . . . . 29
4.5.3.5 data() [2/2] . . . . . 30
4.5.3.6 meanv() . . . . . 30
4.5.3.7 mmp() . . . . . 30
4.5.3.8 mvp() . . . . . 31
4.5.3.9 mvpt() . . . . . 31
4.5.3.10 nelements() . . . . . 31
4.5.3.11 normf() . . . . . 32
4.5.3.12 notinf() . . . . . 32
4.5.3.13 notnan() . . . . . 32
4.5.3.14 operator*() . . . . . 32
4.5.3.15 operator+() . . . . . 33
4.5.3.16 operator-() [1/2] . . . . . 33
4.5.3.17 operator-() [2/2] . . . . . 33
4.5.3.18 operator/() . . . . . 33
4.5.3.19 operator=() . . . . . 34
4.5.3.20 outerProduct() . . . . . 34
4.5.3.21 rowsize() . . . . . 34
4.5.3.22 show() . . . . . 34
4.5.3.23 square() . . . . . 35
4.5.3.24 transpose() . . . . . 35
4.5.3.25 values() [1/2] . . . . . 35
4.5.3.26 values() [2/2] . . . . . 35
4.5.3.27 zeros() . . . . . 35
4.5.4 Friends And Related Function Documentation . . . . . 36
4.5.4.1 operator* . . . . . 36
4.5.4.2 operator/ . . . . . 36
4.6 Net Class Reference . . . . . 36

4.6.1 Detailed Description . . . 37
4.6.2 Constructor & Destructor Documentation . . . 37
4.6.2.1 Net() [1/2] . . . 38
4.6.2.2 Net() [2/2] . . . 38
4.6.3 Member Function Documentation . . . 38
4.6.3.1 accuracy() . . . 38
4.6.3.2 addLayer() . . . 39
4.6.3.3 allocate_batch_specific_temporaries() . . . 39
4.6.3.4 at() [1/2] . . . 40
4.6.3.5 at() [2/2] . . . 40
4.6.3.6 backlayer() [1/2] . . . 40
4.6.3.7 backlayer() [2/2] . . . 40
4.6.3.8 backPropagate() . . . 41
4.6.3.9 calculate_initial_delta() . . . 41
4.6.3.10 calculateLoss() . . . 42
4.6.3.11 create_output_batch() . . . 43
4.6.3.12 decay() . . . 43
4.6.3.13 feedForward() [1/2] . . . 43
4.6.3.14 feedForward() [2/2] . . . 44
4.6.3.15 info() . . . 45
4.6.3.16 inputsize() . . . 45
4.6.3.17 layer() . . . 45
4.6.3.18 learning_rate() . . . 45
4.6.3.19 loadModel() . . . 46
4.6.3.20 momentum() . . . 46
4.6.3.21 optimizer() . . . 46
4.6.3.22 outputsize() . . . 46
4.6.3.23 push_layer() . . . 47
4.6.3.24 RMSprop() . . . 47
4.6.3.25 saveModel() . . . 48
4.6.3.26 set_decay() . . . 48
4.6.3.27 set_learning_rate() . . . 48
4.6.3.28 set_lossfunction() . . . 49
4.6.3.29 set_momentum() . . . 49
4.6.3.30 set_optimizer() . . . 49
4.6.3.31 set_uniform_biases() . . . 49
4.6.3.32 set_uniform_weights() . . . 50
4.6.3.33 SGD() . . . 50
4.6.3.34 show() . . . 50
4.6.3.35 train() . . . 51
4.6.4 Member Data Documentation . . . 51
4.6.4.1 optimize . . . 51

4.7 Vector Class Reference . . . . . 52
4.7.1 Detailed Description . . . . . 52
4.7.2 Constructor & Destructor Documentation . . . . . 52
4.7.2.1 Vector() [1/4] . . . . . 53
4.7.2.2 Vector() [2/4] . . . . . 53
4.7.2.3 Vector() [3/4] . . . . . 53
4.7.2.4 Vector() [4/4] . . . . . 53
4.7.3 Member Function Documentation . . . . . 53
4.7.3.1 add() . . . . . 54
4.7.3.2 copy_from() . . . . . 54
4.7.3.3 data() [1/2] . . . . . 54
4.7.3.4 data() [2/2] . . . . . 54
4.7.3.5 operator*() . . . . . 54
4.7.3.6 operator+() . . . . . 55
4.7.3.7 operator-() [1/2] . . . . . 55
4.7.3.8 operator-() [2/2] . . . . . 55
4.7.3.9 operator/() . . . . . 55
4.7.3.10 operator/=() . . . . . 56
4.7.3.11 operator=() . . . . . 56
4.7.3.12 operator[]() [1/2] . . . . . 56
4.7.3.13 operator[]() [2/2] . . . . . 56
4.7.3.14 positive() . . . . . 56
4.7.3.15 set_ax() . . . . . 57
4.7.3.16 show() . . . . . 57
4.7.3.17 size() . . . . . 57
4.7.3.18 square() . . . . . 57
4.7.3.19 values() [1/2] . . . . . 57
4.7.3.20 values() [2/2] . . . . . 58
4.7.3.21 zeros() . . . . . 58
4.7.4 Friends And Related Function Documentation . . . . . 58
4.7.4.1 Matrix . . . . . 58
4.7.4.2 operator* . . . . . 58
4.7.4.3 operator- . . . . . 59
4.7.4.4 VectorBatch . . . . . 59
4.7.5 Member Data Documentation . . . . . 59
4.7.5.1 c . . . . . 59
4.7.5.2 r . . . . . 59
4.8 VectorBatch Class Reference . . . . . 60
4.8.1 Detailed Description . . . . . 61
4.8.2 Constructor & Destructor Documentation . . . . . 61
4.8.2.1 VectorBatch() [1/4] . . . . . 61
4.8.2.2 VectorBatch() [2/4] . . . . . 62

4.8.2.3 VectorBatch() [3/4] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
4.8.2.4 VectorBatch() [4/4] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.8.3 Member Function Documentation . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.8.3.1 add_vector() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
4.8.3.2 addh() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
4.8.3.3 allocate() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 64
4.8.3.4 at() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.8.3.5 batch_size() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 65
4.8.3.6 copy_from() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
4.8.3.7 data() [1/3] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
4.8.3.8 data() [2/3] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
4.8.3.9 data() [3/3] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 66
4.8.3.10 display() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
4.8.3.11 extract_vector() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 67
4.8.3.12 get_col() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.8.3.13 get_row() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.8.3.14 get_vector() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.8.3.15 get_vectorObj() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
4.8.3.16 hadamard() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
4.8.3.17 item_size() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
4.8.3.18 meanh() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 69
4.8.3.19 nelements() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.8.3.20 normf() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.8.3.21 notinf() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 70
4.8.3.22 notnan() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
4.8.3.23 operator*() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
4.8.3.24 operator-() [1/2] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
4.8.3.25 operator-() [2/2] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
4.8.3.26 operator/() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.8.3.27 operator=() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.8.3.28 outer2() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 72
4.8.3.29 positive() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
4.8.3.30 resize() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
4.8.3.31 scaleby() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 73
4.8.3.32 set_batch_size() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
4.8.3.33 set_col() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
4.8.3.34 set_item_size() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
4.8.3.35 set_row() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
4.8.3.36 set_vector() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75
4.8.3.37 show() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
4.8.3.38 size() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76
4.8.3.39 v2mp() . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 76

4.8.3.40 v2mtp() . . . . . . . . . . 77
4.8.3.41 v2tmp() . . . . . . . . . . 77
4.8.3.42 vals_vector() [1/2] . . . . . . . . . . 78
4.8.3.43 vals_vector() [2/2] . . . . . . . . . . 78
4.8.4 Friends And Related Function Documentation . . . . . . . . . . 78
4.8.4.1 Matrix . . . . . . . . . . 78
4.8.4.2 operator* . . . . . . . . . . 78
4.8.4.3 operator/ . . . . . . . . . . 79
4.8.4.4 Vector . . . . . . . . . . 79

**5 File Documentation** **81**
5.1 configure.py . . . . . . . . . . 81
5.2 dataset.cpp . . . . . . . . . . 81
5.3 dataset.h . . . . . . . . . . 84
5.4 funcs.cpp . . . . . . . . . . 85
5.5 funcs.h . . . . . . . . . . 89
5.6 layer.cpp . . . . . . . . . . 90
5.7 layer.h . . . . . . . . . . 92
5.8 Make.inc . . . . . . . . . . 94
5.9 matrix.cpp . . . . . . . . . . 94
5.10 matrix.h . . . . . . . . . . 95
5.11 matrix_impl_blis.cpp . . . . . . . . . . 96
5.12 matrix_impl_reference.cpp . . . . . . . . . . 98
5.13 net.cpp . . . . . . . . . . 100
5.14 net.h . . . . . . . . . . 106
5.15 net_mpi.cpp . . . . . . . . . . 107
5.16 test_linear.cpp . . . . . . . . . . 108
5.17 test_mnist.cpp . . . . . . . . . . 110
5.18 test_mpi.cpp . . . . . . . . . . 112
5.19 test_posneg.cpp . . . . . . . . . . 114
5.20 trace.cpp . . . . . . . . . . 115
5.21 trace.h . . . . . . . . . . 116
5.22 unittest.cpp . . . . . . . . . . 116
5.23 vector.cpp . . . . . . . . . . 118
5.24 vector.h . . . . . . . . . . 120
5.25 vector2.cpp . . . . . . . . . . 121
5.26 vector2.h . . . . . . . . . . 124
5.27 vector_impl_blis.cpp . . . . . . . . . . 126
5.28 vector_impl_reference.cpp . . . . . . . . . . 127
5.29 vectorbatch_impl_blis.cpp . . . . . . . . . . 128
5.30 vectorbatch_impl_reference.cpp . . . . . . . . . . 130

**Index** **135**

# Chapter 1

# Todo List

**Member Net::addLayer (int l, acFunc activation)**
this one probably has to go

**Member Net::calculate_initial_delta (VectorBatch &result, VectorBatch &gTruth)**
stuff to be done?

**Member VectorBatch::add_vector (const std::vector$<$ float $>$ &v)**
use the allocate method

**Member VectorBatch::set_batch_size (int n)**
should be private, maybe eliminated?

**Member VectorBatch::set_col (int j, const std::vector$<$ float $>$ &v)**
rename to set_vector?

**Member VectorBatch::set_item_size (int n)**
should be private, maybe eliminated?

# Chapter 2

# Class Index

## 2.1 Class List

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


Categorization . . . . . . . . . . 7
dataItem . . . . . . . . . . 10
Dataset . . . . . . . . . . 12
Layer . . . . . . . . . . 21
Matrix . . . . . . . . . . 27
Net . . . . . . . . . . 36
Vector . . . . . . . . . . 52
VectorBatch . . . . . . . . . . 60

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all documented files with brief descriptions:


configure.py . . . ??
dataset.cpp . . . ??
dataset.h . . . ??
funcs.cpp . . . ??
funcs.h . . . ??
layer.cpp . . . ??
layer.h . . . ??
Make.inc . . . ??
matrix.cpp . . . ??
matrix.h . . . ??
matrix_impl_blis.cpp . . . ??
matrix_impl_reference.cpp . . . ??
net.cpp . . . ??
net.h . . . ??
net_mpi.cpp . . . ??
test_linear.cpp . . . ??
test_mnist.cpp . . . ??
test_mpi.cpp . . . ??
test_posneg.cpp . . . ??
trace.cpp . . . ??
trace.h . . . ??
unittest.cpp . . . ??
vector.cpp . . . ??
vector.h . . . ??
vector2.cpp . . . ??
vector2.h . . . ??
vector_impl_blis.cpp . . . ??
vector_impl_reference.cpp . . . ??
vectorbatch_impl_blis.cpp . . . ??
vectorbatch_impl_reference.cpp . . . ??

# Chapter 4

# Class Documentation

## 4.1 Categorization Class Reference

### Public Member Functions

- Categorization (Vector v)
- Categorization (std::vector< float > p)
- Categorization (int n)
- Categorization (int n, int i)
- const std::vector< float > & probabilities () const
- int size () const
- void normalize ()
- bool close_enough (const Categorization &approx) const
- bool close_enough (const std::vector< float > &approx) const

### 4.1.1 Detailed Description

Definition at line 68 of file vector.h.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Categorization() [1/4]

```
Categorization::Categorization (
            Vector v )  [inline]
```

Definition at line 72 of file vector.h.

```
00073     : _probabilities(v.values()) {};
```

#### 4.1.2.2 Categorization() [2/4]

```
Categorization::Categorization (
        std::vector< float > p ) [inline]
```

Definition at line 74 of file vector.h.

```
00075     : _probabilities(p) {};
```

#### 4.1.2.3 Categorization() [3/4]

```
Categorization::Categorization (
        int n ) [inline]
```

Definition at line 76 of file vector.h.

```
00077     : _probabilities( std::vector<float>(n) ) {};
```

#### 4.1.2.4 Categorization() [4/4]

```
Categorization::Categorization (
        int n,
        int i ) [inline]
```

Definition at line 78 of file vector.h.

```
00079     : _probabilities( std::vector<float>(n) ) {
00080     _probabilities.at(i) = 1.;
00081   };
```

### 4.1.3 Member Function Documentation

#### 4.1.3.1 close_enough() [1/2]

```
bool Categorization::close_enough (
        const Categorization & approx ) const [inline]
```

Definition at line 89 of file vector.h.

```
00089                                                        {
00090     return close_enough( approx.probabilities() );
00091   };
```

#### 4.1.3.2 close_enough() [2/2]

```
bool Categorization::close_enough (
            const std::vector< float > & approx ) const [inline]
```

Definition at line 92 of file vector.h.

```
00092                                                      {
00093     assert( size()==approx.size() );
00094     //return _probabilities==approx;
00095     bool close{true};
00096     for ( int i=0; i<size(); i++) {
00097       close = close and
00098     ( ( _probabilities.at(i)==approx.at(i) )
00099       or ( approx.at(i)==0. and ( std::abs(_probabilities.at(i))<1.e-5 ) )
00100       or ( std::abs( (_probabilities.at(i)-approx.at(i))/approx.at(i) )<1.e-5 )
00101       );
00102     }
00103     return close;
00104   };
```

#### 4.1.3.3 normalize()

```
void Categorization::normalize ( ) [inline]
```

Definition at line 84 of file vector.h.

```
00084                      {
00085     auto it = std::max_element(_probabilities.begin(), _probabilities.end());
00086     std::fill(_probabilities.begin(), _probabilities.end(), 0);
00087     *it = 1;
00088   };
```

#### 4.1.3.4 probabilities()

```
const std::vector< float > & Categorization::probabilities ( ) const [inline]
```

Definition at line 82 of file vector.h.

```
00082 { return _probabilities; };
```

#### 4.1.3.5 size()

```
int Categorization::size ( ) const [inline]
```

Definition at line 83 of file vector.h.

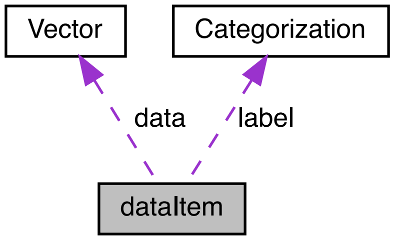
```
00083 { return _probabilities.size(); };
```

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

- vector.h

## 4.2 dataItem Class Reference

Collaboration diagram for dataItem:




### Public Member Functions

- dataItem (Vector data, Categorization label)
- dataItem (float data, Categorization label)
- dataItem (std::vector< float > indata, std::vector< float > outdata)
- int data_size () const
- const std::vector< float > & data_values () const
- int label_size () const
- const std::vector< float > & label_values () const

### Public Attributes

- Vector data
- Categorization label

### 4.2.1 Detailed Description

Definition at line 22 of file dataset.h.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 dataItem() [1/3]

```
dataItem::dataItem (
            Vector data,
            Categorization label )   [inline]
```

Definition at line 28 of file dataset.h.

```
00029     : data(data),label(label) {};
```

#### 4.2.2.2 dataItem() [2/3]

```
dataItem::dataItem (
            float data,
            Categorization label ) [inline]
```

Definition at line 30 of file dataset.h.

```
00031     : data( std::vector<float>(data) ),label(label) {};
```

#### 4.2.2.3 dataItem() [3/3]

```
dataItem::dataItem (
            std::vector< float > indata,
            std::vector< float > outdata ) [inline]
```

Definition at line 32 of file dataset.h.

```
00033     : data( Vector(indata) ),label( Categorization(outdata) ) {};
```

### 4.2.3 Member Function Documentation

#### 4.2.3.1 data_size()

```
int dataItem::data_size ( ) const [inline]
```

Definition at line 34 of file dataset.h.

```
00034 { return data.size(); };
```

#### 4.2.3.2 data_values()

```
const std::vector< float > & dataItem::data_values ( ) const [inline]
```

Definition at line 35 of file dataset.h.

```
00035 { return data.values(); };
```

#### 4.2.3.3 label_size()

```
int dataItem::label_size ( ) const [inline]
```

Definition at line 36 of file dataset.h.

```
00036 { return label.size(); };
```

#### 4.2.3.4 label_values()

```
const std::vector< float > & dataItem::label_values ( ) const [inline]
```

Definition at line 37 of file dataset.h.

```
00037 { return label.probabilities(); };
```

### 4.2.4 Member Data Documentation

#### 4.2.4.1 data

```
Vector dataItem::data
```

Definition at line 24 of file dataset.h.

#### 4.2.4.2 label

```
Categorization dataItem::label
```

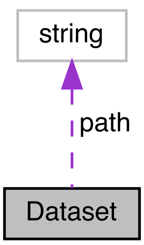
Definition at line 26 of file dataset.h.

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

- dataset.h

## 4.3 Dataset Class Reference

Collaboration diagram for Dataset:



## Public Member Functions

- Dataset (int n)
- Dataset (std::vector< dataItem >)
- void set_lowerbound (int b)
- void set_number (int b)
- void push_back (dataItem it)
- int size () const
- int data_size () const
- int label_size () const
- dataItem item (int i) const
- const Vector & data (int i) const
- const std::vector< float > data_vals (int i) const
- const std::vector< float > label_vals (int i) const
- std::vector< float > stacked_data_vals (int i) const

    *Same, of the stacked object.*
- std::vector< float > stacked_label_vals (int i) const

    *Same, of the stacked object.*
- const auto & inputs () const
- const auto & labels () const
- int readTest (std::string dataPath)
- void shuffle ()
- std::vector< Dataset > batch (int n) const
- void stack ()
- std::pair< Dataset, Dataset > split (float trainFraction) const

## Public Attributes

- std::string path

### 4.3.1 Detailed Description

Definition at line 40 of file dataset.h.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 Dataset() [1/3]

```
Dataset::Dataset ( )  [inline]
```

Definition at line 49 of file dataset.h.

```
00049 {};
```

#### 4.3.2.2 Dataset() [2/3]

```
Dataset::Dataset (
                int n )
```

Definition at line 33 of file dataset.cpp.

```
00033                         : nclasses(n) {
00034   assert(n>=0);
00035 };
```

#### 4.3.2.3 Dataset() [3/3]

```
Dataset::Dataset (
                std::vector< dataItem > dv )
```

Definition at line 37 of file dataset.cpp.

```
00037                                      {
00038   for ( const auto& v : dv )
00039     push_back(v);
00040 };
```

### 4.3.3 Member Function Documentation

#### 4.3.3.1 batch()

```
std::vector< Dataset > Dataset::batch (
                int n ) const
```

Definition at line 178 of file dataset.cpp.

```
00178                                                   {
00179
00180   std::vector<Dataset> batches;
00181   int nitems = size(), itemsize = data_size();
00182   int nbatches = nitems/batch_size + ( nitems%batch_size>0 ? 1 : 0 );
00183   for (int b=0; b<nbatches; b++) {
00184     int first = b*batch_size, last= std::min( (b+1)*batch_size,nitems );
00185     Dataset batch; //(itemsize,last-first);
00186     batch.set_lowerbound(first); batch.set_number(b);
00187     for (int i=first; i<last; i++) {
00188       batch.push_back( item(i) );
00189     }
00190     batches.push_back(batch);
00191   }
00192
00193   return batches;
00194 }
```

#### 4.3.3.2 data()

```
const Vector & Dataset::data (
                int i ) const
```

Get the i-th data object

Definition at line 92 of file dataset.cpp.

```
00092                                     {
00093   throw( string("Do not use Dataset::data") );
00094   //  return _items.at(i).data;
00095 };
```

#### 4.3.3.3 data_size()

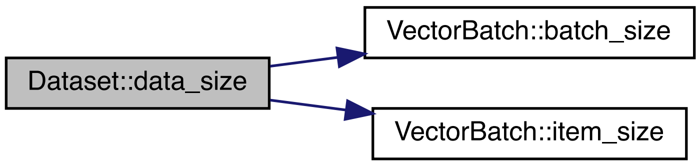
```
int Dataset::data_size ( ) const
```

What is the size of the feature vector in this dataset?

Definition at line 74 of file dataset.cpp.

```
00074                                      {
00075   if (dataBatch.batch_size()==0)
00076     throw( string("Can not get data size for empty dataset") );
00077   return dataBatch.item_size();
00078 };
```

Here is the call graph for this function:




#### 4.3.3.4 data_vals()

```
const vector< float > Dataset::data_vals (
                int i ) const
```

Get the features of i-th data object

Definition at line 99 of file dataset.cpp.

```
00099                                                        {
00100   return dataBatch.extract_vector(i);
00101 };
```

#### 4.3.3.5 inputs()

```
const auto & Dataset::inputs ( ) const  [inline]
```

Definition at line 75 of file dataset.h.

```
00075 { return dataBatch; };
```

#### 4.3.3.6 item()

```
dataItem Dataset::item (
            int i ) const
```

Definition at line 61 of file dataset.cpp.

```
00061                             {
00062   return dataItem( data_vals(i),label_vals(i) );
00063 };
```

#### 4.3.3.7 label_size()

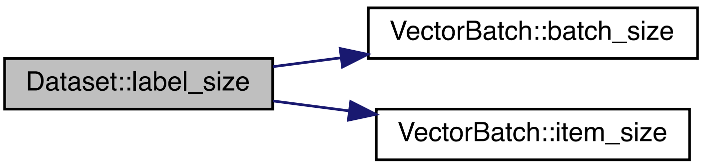
```
int Dataset::label_size ( ) const
```

What is the number of categories in the labels of this dataset?

Definition at line 83 of file dataset.cpp.

```
00083                           {
00084   if (labelBatch.batch_size()==0)
00085     throw( string("Can not get label size for empty dataset") );
00086   return labelBatch.item_size();
00087 }
```

Here is the call graph for this function:




#### 4.3.3.8 label_vals()

```
const vector< float > Dataset::label_vals (
            int i ) const
```

Get the categorization of i-th data object

Definition at line 105 of file dataset.cpp.

```
00105                                           {
00106   return labelBatch.extract_vector(i);
00107 };
```

#### 4.3.3.9 labels()

```
const auto & Dataset::labels ( ) const [inline]
```

Definition at line 76 of file dataset.h.

```
00076 { return labelBatch; };
```

#### 4.3.3.10 push_back()

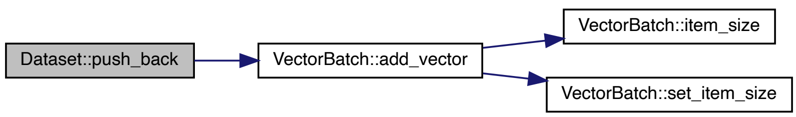
```
void Dataset::push_back (
            dataItem it )
```

Add a new data item, and check its consistency with previous items

Definition at line 48 of file dataset.cpp.

```
00048                                     {
00049   if (nclasses>0 and it.label_size()!=nclasses) {
00050     cout « "Set dimensionality " « nclasses
00051          « " does not match item size " « it.label_size() « endl;
00052     throw( string("Fail to add item to dataset") );
00053   }
00054   if (nclasses==0)
00055     nclasses = it.label_size();
00056   dataBatch.add_vector( it.data_values() );
00057   labelBatch.add_vector( it.label_values() );
00058   //_items.push_back(it);
00059 };
```

Here is the call graph for this function:




#### 4.3.3.11 readTest()

```
int Dataset::readTest (
            std::string dataPath )
```

Definition at line 120 of file dataset.cpp.

```
00120                                         {
00121     /*
00122      * This reader is specifically for a modified MNIST dataset which
00123      * does not include the file header, metadata, etc.
00124      * Link to the dataset: http://cis.jhu.edu/~sachin/digit/digit.html
00125      * I chose this dataset for now to make it easy to read the data;
00126      * in later iterations I will generalize the read function, maybe OpenCV support
00127      */
00128
00129     FILE *file;
00130     std::string fileName;
00131     uint8_t temp[IMSIZE * IMSIZE]; // Image buffer to read data into
```

```
00132     for (int dataid = 0; dataid < 10; dataid++) {
00133         fileName = dataPath + "/data" + std::to_string(dataid); // Put together the path
00134         file = fopen(fileName.c_str(), "r");
00135 
00136 
00137         if (!file) { // File checking
00138             cout « "Error opening file" « endl;
00139             return -2; // Arbitrary error code
00140         }
00141         for (int k = 0; k < 1000; k++) {
00142             Vector imageVec(IMSIZE * IMSIZE, 0); // initialize matrix to be read into
00143             fread(temp, 1, IMSIZE * IMSIZE, file); // Read 28*28 into buffer
00144 
00145             for (int i = 0; i < IMSIZE; i++) {
00146                 for (int j = 0; j < IMSIZE; j++) {
00147                     // Transfer from buffer into matrix
00148           float *i_data = imageVec.data();
00149           *( i_data + i * IMSIZE + j ) // imageVec.vals[i * IMSIZE + j]
00150             = static_cast<float>( temp[i * IMSIZE + j] );
00151                 }
00152             }
00153             fseek(file, k * IMSIZE * IMSIZE, SEEK_SET); // Seek to the kth image bytes
00154 
00155         Categorization label(10,dataid);
00156 
00157             dataItem x = {imageVec, label}; // Initialize an item with the data and the label in it
00158             push_back(x); // Store in the vector
00159 
00160         }
00161         fclose(file);
00162     }
00163     return 0;
00164 }
```

#### 4.3.3.12 set_lowerbound()

```
void Dataset::set_lowerbound (
               int b )
```

Definition at line 42 of file dataset.cpp.

```
00042 { lowerbound = b; };
```

#### 4.3.3.13 set_number()

```
void Dataset::set_number (
               int b )
```

Definition at line 43 of file dataset.cpp.

```
00043 { number = b; };
```

#### 4.3.3.14 shuffle()

```
void Dataset::shuffle ( )
```

Definition at line 167 of file dataset.cpp.

```
00167                         {
00168     std::random_device r;
00169     std::seed_seq seed{r(), r(), r(), r(), r(), r(), r(), r()}; // Seed
00170     std::mt19937 eng1(seed); // Randomizer engine
00171 
00172     throw( string("shuffling doesn't work") );
00173     //    std::shuffle(begin(_items), end(_items), eng1); // Shuffle the dataset
00174     return; // todo add return codes instead of printing
00175 }
```

#### 4.3.3.15 size()

```
int Dataset::size ( ) const
```

Definition at line 65 of file dataset.cpp.

```
00065                                {
00066     int ds = dataBatch.batch_size(), ls = labelBatch.batch_size();
00067     assert( ds==ls );
00068     return ds;
00069 }
```

#### 4.3.3.16 split()

```
std::pair< Dataset, Dataset > Dataset::split (
             float trainFraction ) const
```

Definition at line 220 of file dataset.cpp.

```
00220                                                                             {
00221       int dataset_size = size(); // _items.size();
00222       int testSize{0},trainSize;
00223       while ( true ) {
00224         trainSize= ceil( static_cast<float>( dataset_size ) * trainFraction);
00225         testSize = dataset_size - trainSize;
00226         if ( testSize>0 ) break;
00227         trainFraction *= .9;
00228       }
00229 
00230       auto random_index = permutation(dataset_size);
00231       Dataset trainSplit;
00232       if (trace_progress())
00233         cout « "split into " « trainSize « "+" « testSize « endl;
00234       for (int i=0; i<trainSize; i++) {
00235         const auto& di = item( random_index[i] );
00236         trainSplit.push_back(di);
00237       }
00238       Dataset testSplit;
00239       for (int i=trainSize; i<trainSize+testSize; i++) {
00240         const auto& di = item( random_index[i] );
00241         testSplit.push_back(di);
00242       }
00243 
00244       return std::make_pair(trainSplit,testSplit);
00245 }
```

#### 4.3.3.17 stack()

```
void Dataset::stack ( )
```

Definition at line 196 of file dataset.cpp.

```
00196                       { // Stacks vectors horizontally (column-wise) in a Matrix object
00197     throw( string("Stacking no longer needed") );
00198       //dataBatch  = VectorBatch( data_size(),  size(), 0);
00199       //labelBatch = VectorBatch( label_size(), size(), 0);
00200 
00201       dataBatch  = VectorBatch( size(),  data_size(), 0);
00202       labelBatch = VectorBatch( size(), label_size(), 0);
00203 
00204       for (int j = 0; j < size(); j++) {
00205           //dataBatch.set_col( j,data_vals(j) );
00206           dataBatch.set_row( j, data_vals(j) );
00207       }
00208 
00209       for (int j = 0; j < size(); j++) {
00210           //labelBatch.set_col( j,label_vals(j) );
00211           labelBatch.set_row( j, label_vals(j) );
00212       }
00213 }
```

#### 4.3.3.18 stacked_data_vals()

```
vector< float > Dataset::stacked_data_vals (
              int i ) const
```

Same, of the stacked object.

Definition at line 109 of file dataset.cpp.

```
00109                                                       {
00110   throw( string("Do not use Dataset::stacked_data_vals") );
00111   return dataBatch.get_row(i);
00112 };
```

#### 4.3.3.19 stacked_label_vals()

```
vector< float > Dataset::stacked_label_vals (
              int i ) const
```

Same, of the stacked object.

Definition at line 115 of file dataset.cpp.

```
00115                                                        {
00116   throw( string("Do not use Dataset::stacked_label_vals") );
00117   return labelBatch.get_row(i);
00118 };
```

### 4.3.4 Member Data Documentation

#### 4.3.4.1 path

```
std::string Dataset::path
```

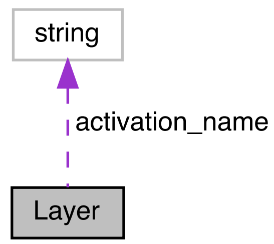
Definition at line 73 of file dataset.h.

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

- dataset.h
- dataset.cpp

## 4.4 Layer Class Reference

Collaboration diagram for Layer:




### Public Member Functions

- Layer (int insize, int outsize)
- Layer & set_uniform_weights (float)
- Layer & set_uniform_biases (float)
- auto & input ()
- const auto & input () const
- Layer & set_number (int n)
- int input_size () const
- int output_size () const
- void **set_recursive_deltas** (Vector &, const Layer &, const Layer &)
- void set_topdelta (const VectorBatch &, const VectorBatch &)
- void allocate_batch_specific_temporaries (int batchsize)
- void forward (const VectorBatch &, VectorBatch &)
- const VectorBatch & intermediate () const
- void backward (const VectorBatch &delta, const Matrix &W, const VectorBatch &prev)
- void **backward_update** (const VectorBatch &, const VectorBatch &, bool=false)
- void update_dw (const VectorBatch &delta, const VectorBatch &prevValues)
- void set_activation (acFunc f)
- Layer & set_activation (std::function< void(const VectorBatch &, VectorBatch &) > apply, std::function< void(const VectorBatch &, VectorBatch &) > activate, std::string name=std::string("custom"))
- Layer & set_activation (std::function< float(const float &) > activate_pt, std::function< float(const float &) > gradient_pt, std::string name=std::string("custom"))

### Public Attributes

- std::string activation_name {"custom"}

### Friends

- class Net

### 4.4.1 Detailed Description

Definition at line 25 of file layer.h.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 Layer() [1/2]

```
Layer::Layer ( )
```

Definition at line 21 of file layer.cpp.

```
00021 {};
```

#### 4.4.2.2 Layer() [2/2]

```
Layer::Layer (
            int insize,
            int outsize )
```

Definition at line 22 of file layer.cpp.

```
00023   : weights( Matrix(outsize,insize,1) ),
00024     dw     ( Matrix(outsize,insize, 0) ),
00025     //dW( Matrix(outsize,insize, 0) ),
00026     dw_velocity( Matrix(outsize,insize, 0) ),
00027     biases( Vector(outsize, 1 ) ),
00028     // biased_product( Vector(outsize, 0) ),
00029     activated( Vector(outsize, 0) ),
00030     d_activated( Vector(outsize, 0) ),
00031     delta( VectorBatch(outsize,1) ),
00032     // biased_productm( VectorBatch(outsize,insize,0) ),
00033     //    activated_batch( VectorBatch(outsize,1, 0) ),
00034     //    d_activated_batch ( VectorBatch(outsize,insize, 0) ),
00035     db( Vector(insize, 0) ),
00036     // delta_mean( Vector(insize, 0) ),
00037     dl( VectorBatch(insize, 1) ),
00038     db_velocity( Vector(insize, 0) ) {};
```

### 4.4.3 Member Function Documentation

#### 4.4.3.1 allocate_batch_specific_temporaries()

```
void Layer::allocate_batch_specific_temporaries (
            int batchsize )
```

Definition at line 43 of file layer.cpp.

```
00043                                                             {
00044   const int insize = weights.colsize(), outsize = weights.rowsize();
00045
00046   biased_batch.allocate( batchsize,outsize );
00047   input_batch.allocate( batchsize,insize );
00048   //  activated_batch.allocate( batchsize,outsize );
00049   d_activated_batch.allocate( batchsize,outsize );
00050   dl.allocate( batchsize, outsize );
00051   delta.allocate( batchsize,outsize );
00052 };
```

#### 4.4.3.2 backward()

```
void Layer::backward (
            const VectorBatch & delta,
            const Matrix & W,
            const VectorBatch & prev )
```

Definition at line 133 of file layer.cpp.

```
00134                                                                                          {
00135
00136   // compute delta ell
00137   activate_gradient_batch(prev_output, d_activated_batch);
00138   prev_delta.v2mtp( W, dl );
00139
00140    // delta  = Dl . sigma
00141   if (trace_progress())
00142     cout « "L-" « layer_number « " delta\n";
00143   delta.hadamard( d_activated_batch,dl ); // Derivative of the current layer
00144
00145   // prev_output.outer2( delta, dw );
00146   // if (trace_scalars())
00147   //   cout « "L-" « layer_number « " dw: "
00148   //     « delta.normf() « "x" « prev_output.normf() « " => " « dw.normf() « "\n";
00149
00150   update_dw(delta, prev_output);
00151   // weights.axpy( 1.,dw );
00152   // db = delta.meanh();
00153   // biases.add( db );
00154 }
```

#### 4.4.3.3 forward()

```
void Layer::forward (
            const VectorBatch & input,
            VectorBatch & output )
```

Definition at line 99 of file layer.cpp.

```
00099                                                                     {
00100   assert( input.batch_size()==output.batch_size() );
00101   if (trace_progress()) {
00102     cout « "Forward layer " « layer_number
00103      « ": " « input_size() « "->" « output_size() « endl;
00104   }
00105
00106   allocate_batch_specific_temporaries(input.batch_size());
00107   input_batch.copy_from(input);
00108   if (trace_progress()) {
00109     assert( input_batch.notnan() ); assert( input_batch.notinf() );
00110     assert( weights.notnan() ); assert( weights.notinf() );
00111   }
00112   input_batch.v2mp( weights, biased_batch );
00113   if (trace_progress()) {
00114     assert( biased_batch.notnan() ); assert( biased_batch.notinf() );
00115   }
00116
00117   biased_batch.addh(biases); // Add the bias
00118   if (trace_progress()) {
00119     assert( biased_batch.notnan() ); assert( biased_batch.notinf() );
00120   }
00121
00122   apply_activation_batch(biased_batch, output);
00123   //cout « "layer output: " « output.data()[0] « "\n";
00124   if (trace_progress()) {
00125     assert( output.notnan() ); assert( output.notinf() );
00126   }
00127   //return activated_batch;
00128 }
```

#### 4.4.3.4 input() [1/2]

```
auto & Layer::input ( ) [inline]
```

Definition at line 47 of file layer.h.

```
00047 { return input_batch; };
```

#### 4.4.3.5 input() [2/2]

```
const auto & Layer::input ( ) const [inline]
```

Definition at line 48 of file layer.h.

```
00048 { return input_batch; };
```

#### 4.4.3.6 input_size()

```
int Layer::input_size ( ) const [inline]
```

Definition at line 75 of file layer.h.

```
00075 { return weights.colsize(); };
```

#### 4.4.3.7 intermediate()

```
const VectorBatch & Layer::intermediate ( ) const
```

Definition at line 131 of file layer.cpp.

```
00131 { return biased_batch; };
```

#### 4.4.3.8 output_size()

```
int Layer::output_size ( ) const [inline]
```

Definition at line 76 of file layer.h.

```
00076 { return weights.rowsize(); };
```

#### 4.4.3.9 set_activation() [1/3]

```
void Layer::set_activation (
            acFunc f )
```

Definition at line 54 of file layer.cpp.

```
00054                                   {
00055   activation = f;
00056   apply_activation_batch  = apply_activation<VectorBatch>.at(f);
00057   activate_gradient_batch = activate_gradient<VectorBatch>.at(f);
00058 };
```

#### 4.4.3.10 set_activation() [2/3]

```
Layer & Layer::set_activation (
            std::function< float(const float &) > activate_pt,
            std::function< float(const float &) > gradient_pt,
            std::string name = std::string("custom") )
```

Definition at line 60 of file layer.cpp.

```
00065       {
00066     set_activation
00067     (
00068      [activate_pt] ( const VectorBatch& i,VectorBatch& o ) -> void {
00069         batch_activation( activate_pt,i,o ); },
00070      [gradient_pt] ( const VectorBatch& i,VectorBatch& o ) -> void {
00071         batch_activation( gradient_pt,i,o ); },
00072      name );
00073   return *this;
00074 };
```

#### 4.4.3.11 set_activation() [3/3]

```
Layer & Layer::set_activation (
            std::function< void(const VectorBatch &, VectorBatch &) > apply,
            std::function< void(const VectorBatch &, VectorBatch &) > activate,
            std::string name = std::string("custom") )
```

Definition at line 76 of file layer.cpp.

```
00079                         {
00080   activation = acFunc::RELU;
00081   apply_activation_batch  = apply;
00082   activate_gradient_batch = activate;
00083   return *this;
00084 };
```

#### 4.4.3.12 set_number()

```
Layer & Layer::set_number (
            int n )  [inline]
```

Definition at line 72 of file layer.h.

```
00072 { layer_number = n; return *this; };
```

#### 4.4.3.13 set_topdelta()

```
void Layer::set_topdelta (
            const VectorBatch & gTruth,
            const VectorBatch & output )
```

Definition at line 168 of file layer.cpp.

```
00168                                                                               {
00169
00170    // top delta ell is different
00171   activate_gradient_batch(output, d_activated_batch);
00172   dl = output - gTruth;
00173   dl.scaleby( 1.f / gTruth.batch_size() );
00174   // delta  = Dl . sigma
00175   delta.hadamard( d_activated_batch,dl );
00176   if (trace_scalars())
00177     cout « "L-" « layer_number « " top delta: "
00178      « d_activated_batch.normf() « "x" « dl.normf() « " => " « delta.normf() « "\n";
00179
00180   //  update_dw(delta, prev_output);
00181 };
```

#### 4.4.3.14 set_uniform_biases()

```
Layer & Layer::set_uniform_biases (
            float v )
```

Definition at line 92 of file layer.cpp.

```
00092                                        {
00093   for ( auto& e : biases.values() )
00094     e = v;
00095   return *this;
00096 };
```

#### 4.4.3.15 set_uniform_weights()

```
Layer & Layer::set_uniform_weights (
            float v )
```

Definition at line 86 of file layer.cpp.

```
00086                                         {
00087   for ( auto& e : weights.values() )
00088     e = v;
00089   return *this;
00090 };
```

#### 4.4.3.16 update_dw()

```
void Layer::update_dw (
            const VectorBatch & delta,
            const VectorBatch & prevValues )
```

Definition at line 156 of file layer.cpp.

```
00156                                                                             {
00157    prev_output.outer2( delta, dw );
00158    if (trace_scalars())
00159      cout « "L-" « layer_number « " dw: "
00160       « delta.normf() « "x" « prev_output.normf() « " => " « dw.normf() « "\n";
00161
00162   // Delta W = delta here X activated prevous
00163   weights.axpy( 1.,dw );
00164   db = delta.meanh();
00165   biases.add( db );
00166 }
```

### 4.4.4 Friends And Related Function Documentation

#### 4.4.4.1 Net

```
friend class Net  [friend]
```

Definition at line 26 of file layer.h.

### 4.4.5 Member Data Documentation

#### 4.4.5.1 activation_name

```
std::string Layer::activation_name {"custom"}
```

Definition at line 101 of file layer.h.

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

- layer.h
- layer.cpp

## 4.5 Matrix Class Reference

### Public Member Functions

- Matrix (int nRows, int nCols, int rand)
- std::vector< float > & values ()
- const std::vector< float > & values () const
- float * data ()
- const float * data () const
- int nelements () const
- int rowsize () const
- int colsize () const
- Matrix transpose () const
- void show () const
- void mvpt (const Vector &x, Vector &y) const
- void mvp (const Vector &x, Vector &y) const
- void addvh (const Vector &y)
- void mmp (const Matrix &x, Matrix &y) const
- void outerProduct (const Vector &x, const Vector &y)
- Vector meanv ()
- void zeros ()
- void square ()
- float normf () const
- bool notnan () const
- bool notinf () const
- Matrix operator- ()
- Matrix & operator= (const Matrix &m2)
- Matrix operator+ (const Matrix &m2) const
- Matrix operator* (const Matrix &m2)
- Matrix operator/ (const Matrix &m2)
- Matrix operator- (const Matrix &m2)
- void axpy (float a, const Matrix &x)

## Friends

- Matrix operator* (const float &c, const Matrix &m)
- Matrix operator/ (const Matrix &m, const float &c)

## 4.5.1 Detailed Description

Definition at line 22 of file matrix.h.

## 4.5.2 Constructor & Destructor Documentation

### 4.5.2.1 Matrix() [1/2]

```
Matrix::Matrix ( )
```

Definition at line 22 of file matrix.cpp.

```
00022                 { // Default constructor
00023     r = 0;
00024     c = 0;
00025     mat.clear();
00026 }
```

### 4.5.2.2 Matrix() [2/2]

```
Matrix::Matrix (
               int nRows,
               int nCols,
               int rand = 0 )
```

Definition at line 26 of file matrix_impl_blis.cpp.

```
00027         : r(nRows), c(nCols) {
00028
00029     mat = vector<float>(nRows * nCols);
00030     float scal_fac = 0.05; // randomize between (-1;1)
00031     if (random==0){
00032         float zero = 0.0;
00033         bli_ssetm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
00034                 r, c, &zero, &mat[0], c, 1);
00035     } else if (random==1){
00036         bli_srandm(0, BLIS_DENSE, r, c, &mat[0], c, 1);
00037         bli_sscalm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
00038                 r, c, &scal_fac, &mat[0], c, 1);
00039     }
00040 }
```

## 4.5.3 Member Function Documentation

#### 4.5.3.1 addvh()

```
void Matrix::addvh (
            const Vector & y )
```

Definition at line 108 of file matrix.cpp.

```
00108                                    {
00109     for (int j = 0; j < c; j++) {
00110         for (int i = 0; i < y.size(); i++) {
00111             mat[j + c * i] += y.vals[i];
00112         }
00113     }
00114 }
```

#### 4.5.3.2 axpy()

```
void Matrix::axpy (
            float a,
            const Matrix & x )
```

Definition at line 121 of file matrix_impl_blis.cpp.

```
00121                                             {
00122   assert( r==x.r );
00123   assert( c==x.c );
00124   const int n = nelements();
00125   assert( n==x.nelements() );
00126
00127   bli_saxpyv( BLIS_NO_CONJUGATE,
00128           n, &a, const_cast<float*>( x.data() ),1,
00129           data(),1 );
00130 };
```

#### 4.5.3.3 colsize()

```
int Matrix::colsize ( ) const  [inline]
```

Definition at line 39 of file matrix.h.

```
00039 { return c; };
```

#### 4.5.3.4 data() [1/2]

```
float * Matrix::data ( )
```

Definition at line 34 of file matrix.cpp.

```
00034 { return mat.data(); };
```

#### 4.5.3.5 data() [2/2]

```
const float * Matrix::data ( ) const
```

Definition at line 35 of file matrix.cpp.

```
00035 { return mat.data(); };
```

#### 4.5.3.6 meanv()

```
Vector Matrix::meanv ( )
```

Definition at line 116 of file matrix.cpp.

```
00116                           { // Returns a vector of column-wise means
00117     Vector mean(r, 0);
00118     float avg;
00119     for (int i = 0; i < r; i++) {
00120         avg = 0.0;
00121         for (int j = 0; j < c; j++) {
00122             avg += mat[i * c + j];
00123         }
00124         mean.vals[i] = avg;
00125     }
00126     return mean;
00127 }
```

#### 4.5.3.7 mmp()

```
void Matrix::mmp (
        const Matrix & x,
        Matrix & y ) const
```

Definition at line 105 of file matrix_impl_blis.cpp.

```
00105                                                    { // In place matrix matrix multiplication
00106     assert( c==x.r );
00107     assert( r==y.r );
00108     assert( x.c==y.c );
00109 
00110     float alpha = 1.0;
00111     float beta = 0.0;
00112     // m = r, n = x.c, k = c
00113     //printf("BLIS gemm %dx%dx%d\n",r,x.c,c);
00114     bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_NO_TRANSPOSE,
00115            r, x.c, c, &alpha, const_cast<float*>(&mat[0]),
00116            //c, 1, &t.mat[0],
00117            c, 1, const_cast<float*>( x.data() ),
00118            x.c, 1, &beta, &y.mat[0], x.c, 1);
00119 }
```

#### 4.5.3.8 mvp()

```
void Matrix::mvp (
            const Vector & x,
            Vector & y ) const
```

Definition at line 61 of file matrix_impl_blis.cpp.

```
00061                                                {
00062     assert( c==x.size() );
00063     assert( r==y.size() );
00064 
00065     float alpha = 1.0;
00066     float beta = 0.0;
00067     //printf("BLIS gemv %dx%d\n",c,r);
00068     bli_sgemv( BLIS_NO_TRANSPOSE, BLIS_NO_CONJUGATE,
00069            r, c, &alpha, const_cast<float*>(&mat[0]),
00070            c, 1, const_cast<float*>( x.data() ) /* &t.vals[0] */,
00071            // c, 1, &t.vals[0] ,
00072            1, &beta, &y.vals[0], 1 );
00073 
00074 }
```

#### 4.5.3.9 mvpt()

```
void Matrix::mvpt (
            const Vector & x,
            Vector & y ) const
```

Definition at line 76 of file matrix_impl_blis.cpp.

```
00076                                                 {
00077     assert( r==x.size() );
00078     assert( c==y.size() );
00079 
00080     float alpha = 1.0;
00081     float beta = 0.0;
00082     //printf("BLIS gemv %dx%d\n",c,r);
00083     bli_sgemv( BLIS_TRANSPOSE, BLIS_NO_CONJUGATE,
00084            r, c, &alpha, const_cast<float*>(&mat[0]), // test if r and c need to be flipped
00085            //c, 1, &t.vals[0],
00086            c, 1, const_cast<float*>( x.data() ),
00087            1, &beta, &y.vals[0], 1 );
00088 }
```

#### 4.5.3.10 nelements()

```
int Matrix::nelements ( ) const  [inline]
```

Definition at line 35 of file matrix.h.

```
00035                                  {
00036       return mat.size();
00037     };
```

#### 4.5.3.11 normf()

```
float Matrix::normf ( ) const
```

Definition at line 132 of file matrix_impl_blis.cpp.

```
00132                             {
00133   float norm;
00134   auto r = rowsize(), c = colsize();
00135   const auto mval = values().data();
00136   bli_snormfv( r*c, const_cast<float*>(mval), 1, &norm );
00137   return norm;
00138 };
```

#### 4.5.3.12 notinf()

```
bool Matrix::notinf ( ) const  [inline]
```

Definition at line 60 of file matrix.h.

```
00060                             {
00061       return all_of
00062     ( mat.begin(),mat.end(),
00063       [] (float e) { return not isinf(e); }
00064     );
00065     }
```

#### 4.5.3.13 notnan()

```
bool Matrix::notnan ( ) const  [inline]
```

Definition at line 54 of file matrix.h.

```
00054                             {
00055       return all_of
00056     ( mat.begin(),mat.end(),
00057       [] (float e) { return not isnan(e); }
00058     );
00059     }
```

#### 4.5.3.14 operator*()

```
Matrix Matrix::operator* (
             const Matrix & m2 )
```

Definition at line 82 of file matrix.cpp.

```
00082                                           { // Hadamard product
00083     Matrix out(m2.r, m2.c, 0);
00084     assert(out.mat.size()==m2.r * m2.c);
00085     for (int i = 0; i < m2.r * m2.c; i++) {
00086         out.mat[i] = this->mat[i] * m2.mat[i];
00087     }
00088     return out;
00089 }
```

#### 4.5.3.15 operator+()

```
Matrix Matrix::operator+ (
            const Matrix & m2 ) const
```

Definition at line 46 of file matrix.cpp.

```
00046                                               {
00047     Matrix out(m2.r, m2.c, 0);
00048     for (int i = 0; i < m2.r * m2.c; i++) {
00049         out.mat[i] = this->mat[i] + m2.mat[i];
00050     }
00051     return out;
00052 }
```

#### 4.5.3.16 operator-() [1/2]

```
Matrix Matrix::operator- ( )
```

Definition at line 99 of file matrix.cpp.

```
00099                           {
00100     Matrix result = *this;
00101     for (int i = 0; i < r * c; i++) {
00102         result.mat[i] = -mat[i];
00103     }
00104 
00105     return result;
00106 };
```

#### 4.5.3.17 operator-() [2/2]

```
Matrix Matrix::operator- (
            const Matrix & m2 )
```

Definition at line 54 of file matrix.cpp.

```
00054                                         {
00055     Matrix out(m2.r, m2.c, 0);
00056     for (int i = 0; i < m2.r * m2.c; i++) {
00057         out.mat[i] = this->mat[i] - m2.mat[i];
00058     }
00059     return out;
00060 }
```

#### 4.5.3.18 operator/()

```
Matrix Matrix::operator/ (
            const Matrix & m2 )
```

Definition at line 91 of file matrix.cpp.

```
00091                                         { // Hadamard product
00092     Matrix out(m2.r, m2.c, 0);
00093     for (int i = 0; i < m2.r * m2.c; i++) {
00094         out.mat[i] = this->mat[i] / m2.mat[i];
00095     }
00096     return out;
00097 }
```

#### 4.5.3.19 operator=()

```
Matrix & Matrix::operator= (
            const Matrix & m2 )
```

Definition at line 37 of file matrix.cpp.

```
00037                                         { // Overloading the = operator
00038     r = m2.r;
00039     c = m2.c;
00040
00041     this->mat = m2.mat; // IM Since we're using vectors we can just use the assignment from that
00042
00043     return *this;
00044 }
```

#### 4.5.3.20 outerProduct()

```
void Matrix::outerProduct (
            const Vector & x,
            const Vector & y )
```

Definition at line 91 of file matrix_impl_blis.cpp.

```
00091                                                               {
00092     assert( x.size() == r );
00093     assert( y.size() == c );
00094     float val = 1.0;
00095
00096     //printf("BLIS gemm (outer) %dx%d\n",r,c);
00097     bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_TRANSPOSE,
00098             r, c, 1, &val,
00099             const_cast<float*>(x.data()), 1, 1,
00100             const_cast<float*>(y.data()), c, 1, &val, &mat[0], c, 1);
00101
00102 }
```

#### 4.5.3.21 rowsize()

```
int Matrix::rowsize ( ) const  [inline]
```

Definition at line 38 of file matrix.h.

```
00038 { return r; };
```

#### 4.5.3.22 show()

```
void Matrix::show ( ) const
```

Definition at line 53 of file matrix_impl_blis.cpp.

```
00053                            {
00054
00055     char e[5] = "";
00056     char format[8] = "%4.4f";
00057     bli_sprintm( e, r, c, const_cast<float*>(&mat[0]), c, 1, format, e );
00058 }
```

#### 4.5.3.23 square()

void Matrix::square ( )

Definition at line 134 of file matrix.cpp.

```
00134                         {
00135     std::for_each(mat.begin(), mat.end(), [](auto &n) { n *= n;});
00136 }
```

#### 4.5.3.24 transpose()

Matrix Matrix::transpose ( ) const

Definition at line 42 of file matrix_impl_blis.cpp.

```
00042                                {
00043     Matrix result(c, r, 0); // Initialize a new matrix with inverted dimension values
00044 
00045     // m = r, n = c
00046     // rs = 1, cs = m, rsf = 1, csf = n
00047     //printf("BLIS copy %dx%d\n",c,r);
00048     bli_scopym( 0, BLIS_NONUNIT_DIAG, BLIS_DENSE, BLIS_TRANSPOSE,
00049         c, r, const_cast<float*>(&mat[0]), c, 1, &result.mat[0], r, 1);
00050     return result;
00051 }
```

#### 4.5.3.25 values() [1/2]

std::vector< float > & Matrix::values ( ) [inline]

Definition at line 31 of file matrix.h.

```
00031 { return mat; };
```

#### 4.5.3.26 values() [2/2]

const std::vector< float > & Matrix::values ( ) const [inline]

Definition at line 32 of file matrix.h.

```
00032 { return mat; };
```

#### 4.5.3.27 zeros()

void Matrix::zeros ( )

Definition at line 130 of file matrix.cpp.

```
00130                        {
00131     std::fill(mat.begin(), mat.end(), 0);
00132 }
```

### 4.5.4 Friends And Related Function Documentation

#### 4.5.4.1 operator*

```
Matrix operator* (
        const float & c,
        const Matrix & m ) [friend]
```

Definition at line 64 of file matrix.cpp.

```
00064                                                    {
00065     Matrix o = m;
00066     assert(o.mat.size()==o.r*o.c);
00067     for (int i = 0; i < o.r * o.c; i++) {
00068         o.mat[i] = c * o.mat[i];
00069     }
00070     return o;
00071 }
```

#### 4.5.4.2 operator/

```
Matrix operator/ (
        const Matrix & m,
        const float & c ) [friend]
```

Definition at line 73 of file matrix.cpp.

```
00073                                                    {
00074     Matrix o = m;
00075     for (int i = 0; i < o.r * o.c; i++) {
00076         o.mat[i] = o.mat[i] / c;
00077     }
00078     return o;
00079 }
```

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

- matrix.h
- matrix.cpp
- matrix_impl_blis.cpp
- matrix_impl_reference.cpp

## 4.6 Net Class Reference

### Public Member Functions

- Net (int s)
- Net (const Dataset &d)
- void addLayer (int l, acFunc activation)
- void **addLayer** (int l, std::function< float(const float &) > activate_pt, std::function< float(const float &) > gradient_pt, std::string name=std::string("custom"))
- void **addLayer** (int l, std::function< void(const VectorBatch &, VectorBatch &) > apply_activation_batch, std::function< void(const VectorBatch &, VectorBatch &) > activate_gradient_batch, std::string name=std::↩ ::string("custom"))

- void push_layer (const Layer &layer)
- int outputsize () const
- int inputsize (int layer) const
- VectorBatch create_output_batch (int batchsize)
- Layer & backlayer ()
- const Layer & backlayer () const
- const Layer & layer (int i) const
- const Layer & at (int i) const
- Layer & at (int i)
- void show ()
- Categorization **output_vector** () const
- const VectorBatch & **outputs** () const
- void set_lossfunction (lossfn lossFuncName)
- void set_uniform_weights (float)
- void set_uniform_biases (float)
- void feedForward (const Vector &, Vector &)
- void feedForward (const VectorBatch &, VectorBatch &)
- void allocate_batch_specific_temporaries (int batchsize)
- void **calcGrad** (Dataset data)
- void **calcGrad** (VectorBatch data, VectorBatch labels)
- void **backPropagate** (const Vector &input, const Vector &gTruth, const Vector &output)
- void backPropagate (const VectorBatch &input, const VectorBatch &gTruth, const VectorBatch &output)
- void calculate_initial_delta (VectorBatch &result, VectorBatch &gTruth)
- void SGD (float lr, float momentum)
- void RMSprop (float lr, float momentum)
- void set_learning_rate (float lr)
- float learning_rate () const
- void set_decay (float d)
- float decay () const
- void set_momentum (float m)
- float momentum () const
- void set_optimizer (int m)
- int optimizer () const
- void train (const Dataset &train, const Dataset &test, int epochs, int batchSize)
- float calculateLoss (const Dataset &testSplit)
- float accuracy (const Dataset &valSet)
- void saveModel (std::string path)
- void loadModel (std::string path)
- void info ()

## Public Attributes

- std::vector< std::function< void(float lr, float momentum) > > optimize

### 4.6.1 Detailed Description

Definition at line 26 of file net.h.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 Net() [1/2]

```
Net::Net (
            int s )
```

Definition at line 29 of file net.cpp.

```
00029                {  // Input vector size
00030     this->inR = s;
00031     this->inC = 1;
00032     samples = 0;
00033 }
```

#### 4.6.2.2 Net() [2/2]

```
Net::Net (
            const Dataset & d )
```

Definition at line 35 of file net.cpp.

```
00035                              {
00036   this->inR = data.data_size(); //data.items.at(0).data.size();
00037   this->inC = 1;
00038   samples = 0;
00039 }
```

### 4.6.3 Member Function Documentation

#### 4.6.3.1 accuracy()

```
float Net::accuracy (
            const Dataset & valSet )
```

Definition at line 407 of file net.cpp.

```
00407                                                   {
00408   if (trace_progress())
00409     cout « "Accuracy calculation\n";
00410
00411   int correct = 0;
00412   int incorrect = 0;
00413
00414   assert( test_set.size()>0 );
00415   const auto& test_inputs = test_set.inputs();
00416   const auto& test_labels = test_set.labels();
00417   assert( test_inputs.batch_size()==test_labels.batch_size() );
00418   assert( test_inputs.batch_size()>0 );
00419
00420   //      allocate_batch_specific_temporaries(test_inputs.batch_size());
00421   if (trace_arrays()) {
00422       cout « "inputs:\n"; test_inputs.show();
00423   }
00424   assert( test_inputs.normf()!=0.f );
00425   auto output_batch = this->create_output_batch(test_labels.batch_size());
00426   feedForward(test_inputs,output_batch);
00427   if (trace_arrays()) {
00428       cout « "outputs:\n"; output_batch.show();
00429   }
00430   assert( output_batch.notnan() );
00431
00432   for(int idx=0; idx < output_batch.batch_size(); idx++ ) {
00433       Vector oneItem = output_batch.get_vectorObj(idx);
00434       Categorization result( oneItem );
00435       result.normalize();
00436       if ( result.close_enough( test_labels.extract_vector(idx) ) ) {
```

```
      correct++;
      } else {
      incorrect++;
      }
  }
  assert( correct+incorrect==test_set.size() );

  float acc = static_cast<float>( correct ) / static_cast<float>( test_set.size() );
  return acc;
}
```

#### 4.6.3.2 addLayer()

```
void Net::addLayer (
            int l,
            acFunc f )
```

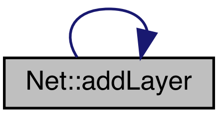
Add a layer from indexed activation function

**Todo** this one probably has to go

Definition at line 89 of file net.cpp.

```
00089                                  {
00090   addLayer( l,
00091         apply_activation<VectorBatch>.at(f),
00092         activate_gradient<VectorBatch>.at(f),
00093         activation_names.at(f)
00094         );
00095 }
```

Here is the call graph for this function:




#### 4.6.3.3 allocate_batch_specific_temporaries()

```
void Net::allocate_batch_specific_temporaries (
            int batchsize )
```

Resize temporaries to reflect current batch size

Definition at line 346 of file net.cpp.

```
00346                                                          {
00347 #ifdef DEBUG
00348   cout « "allocating temporaries for batch size " « batchsize « endl;
00349 #endif
00350   for ( auto& layer : layers )
00351     layer.allocate_batch_specific_temporaries(batchsize);
00352 }
```

#### 4.6.3.4 at() [1/2]

```
Layer & Net::at (
            int i )  [inline]
```

Definition at line 79 of file net.h.

```
00079 { return layers.at(i); };
```

#### 4.6.3.5 at() [2/2]

```
const Layer & Net::at (
            int i ) const  [inline]
```

Definition at line 78 of file net.h.

```
00078 { return layers.at(i); };
```

#### 4.6.3.6 backlayer() [1/2]

```
Layer & Net::backlayer ( )  [inline]
```

Definition at line 73 of file net.h.

```
00073                         {
00074     assert(layers.size()>0); return layers.back(); };
```

#### 4.6.3.7 backlayer() [2/2]

```
const Layer & Net::backlayer ( ) const  [inline]
```

Definition at line 75 of file net.h.

```
00075                                   {
00076     assert(layers.size()>0); return layers.back(); };
```

#### 4.6.3.8 backPropagate()

```
void Net::backPropagate (
            const VectorBatch & input,
            const VectorBatch & output,
            const VectorBatch & gTruth )
```

Full back propagation sweep

Definition at line 199 of file net.cpp.

```
{

  if (layers.size()==1) {
    throw(string("single layer case does not work"));
      // const VectorBatch& prev = input;
      // layers.back().update_dw(delta, prev);
      // return;
  } else {

    if (trace_progress()) cout « "Layer-" « layers.back().layer_number « "\n";
    layers.back().set_topdelta( gTruth,output );
    const VectorBatch& prev = layers.back().input();
    layers.back().update_dw(layers.back().delta, prev);


    for (unsigned i = layers.size() - 2; i > 0; i--) {
      if (trace_progress()) cout « "Layer-" « layers.at(i).layer_number « "\n";
      layers.at(i).backward
      ( layers.at(i+1).delta, layers.at(i+1).weights, layers.at(i).input());
    }

    if (trace_progress()) cout « "Layer-" « layers.at(0).layer_number « "\n";
    layers.at(0).backward(layers.at(1).delta, layers.at(1).weights, input);

  }
}
```

#### 4.6.3.9 calculate_initial_delta()

```
void Net::calculate_initial_delta (
            VectorBatch & input,
            VectorBatch & gTruth )
```

Set up delta values for back propagation

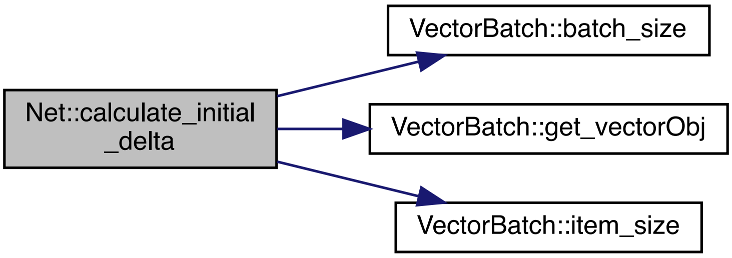
**Todo** stuff to be done?

Definition at line 178 of file net.cpp.

```
{
    VectorBatch d_loss = d_lossFunction( gTruth, input);

    if (layers.back().activation_name == "SoftMax" ) { // Softmax derivative function
        Matrix jacobian( input.item_size(), input.item_size(), 0 );
        for(int i = 0; i < input.batch_size(); i++ ) {
          auto one_column = input.get_vector(i);
          jacobian = smaxGrad_vec( one_column );
          Vector one_vector( jacobian.rowsize(), 0 );
          Vector one_grad = d_loss.get_vectorObj(i);
          jacobian.mvp( one_grad, one_vector );
          layers.back().d_activated_batch.set_vector(one_vector,i);
        }
    }
    /* Will add the rest of the code here, not done yet
    */
}
```

Here is the call graph for this function:




#### 4.6.3.10 calculateLoss()

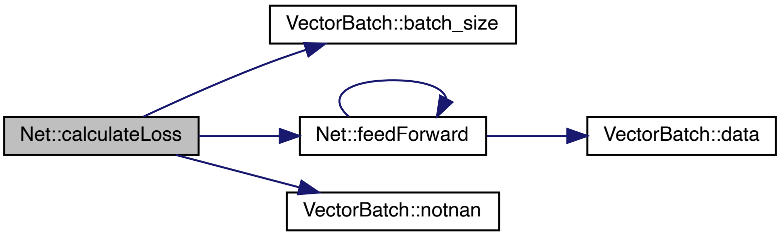
```
float Net::calculateLoss (
              const Dataset & testSplit )
```

Calculate the los function as sum of losses of the individual data point.

Definition at line 359 of file net.cpp.

```
00359                                                     {
00360
00361 #ifdef DEBUG
00362   cout « "Loss calculation\n";
00363 #endif
00364   //  allocate_batch_specific_temporaries(testSplit.inputs().batch_size());
00365   VectorBatch result = this->create_output_batch( testSplit.inputs().batch_size() );
00366   feedForward( testSplit.inputs(),result );
00367   assert( result.notnan() );
00368
00369     float loss = 0.0;
00370     auto tmp_labels =  testSplit.labels();
00371     if (trace_arrays()) {
00372       cout « "Compare results\n"; result.show();
00373       cout « " to label\n"; tmp_labels.show();
00374     }
00375     for (int vec=0; vec<result.batch_size(); vec++) { // iterate over all items
00376       const auto& one_result = result.extract_vector(vec); // VLE figure out const span !!!
00377       auto one_label  = tmp_labels.get_vector(vec);
00378       assert( one_result.size()==one_label.size() );
00379       for (int i=0; i<one_result.size(); i++) { // Calculate loss of result
00380     auto this_label = one_label[i], this_result = one_result[i];
00381     assert( not std::isnan(this_label) );
00382     assert( not std::isnan(this_result) );
00383     auto oneloss = lossFunction( this_label, this_result );
00384     assert( not std::isnan(oneloss) );
00385     loss += oneloss;
00386       }
00387     }
00388     const int bs = result.batch_size();
00389     assert( bs>0 );
00390     auto scale = 1.f / static_cast<float>(bs);
00391     loss = loss * scale;
00392
00393     return loss;
00394 }
```

Here is the call graph for this function:




#### 4.6.3.11 create_output_batch()

```
VectorBatch Net::create_output_batch (
              int batchsize )  [inline]
```

Definition at line 67 of file net.h.

```
00067                                                     {
00068     return VectorBatch( outputsize(),batchsize );
00069     };
```

#### 4.6.3.12 decay()

```
float Net::decay ( ) const  [inline]
```

Definition at line 117 of file net.h.

```
00117 { return _decay; };
```

#### 4.6.3.13 feedForward() [1/2]

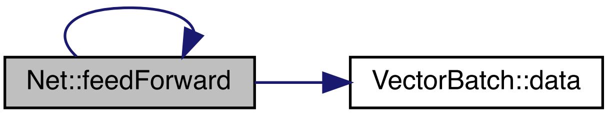
```
void Net::feedForward (
              const Vector & input,
              Vector & output )
```

Feed a single vector input forward through the network This first converts the vector to a batch, so it is not efficient.

Definition at line 155 of file net.cpp.

```
00155                                                           {
00156     VectorBatch input_batch(input), output_batch(input_batch);
00157     feedForward(input_batch,output_batch);
00158     const auto& in_data = input_batch.data();
00159     const auto& out_data = output_batch.data();
00160     //    cout « "net forward: " « in_data[0] « " -> " « out_data[0] « "\n";
00161     output.copy_from( output_batch );
00162 };
```

Here is the call graph for this function:




#### 4.6.3.14 feedForward() [2/2]

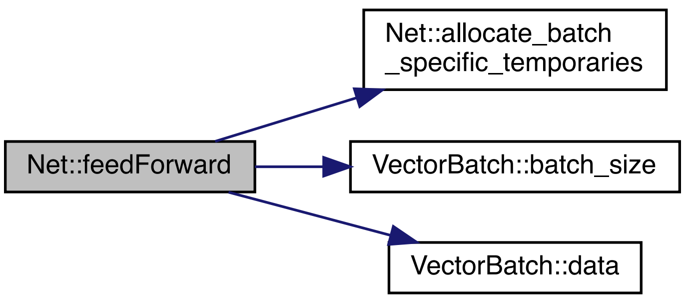
```
void Net::feedForward (
            const VectorBatch & input,
            VectorBatch & output )
```

Feed an input forward through the network The input here is a batch of vectors

Definition at line 126 of file net.cpp.

```
00126                                                                             {
00127   if (trace_progress())
00128     cout « "Feed forward batch of size " « input.batch_size() « endl;
00129   allocate_batch_specific_temporaries(input.batch_size());
00130
00131   if (layers.size()==1) {
00132     layers.front().forward(input,output);
00133     //cout « "single layer output: " « output.data()[0] « "\n";
00134   } else {
00135     layers.front().forward( input, layers.at(1).input() );
00136     cout « " first layer : "
00137      « input.data()[0] « " -> " « layers.at(1).input().data()[0] « "\n";
00138     for (unsigned i = 1; i<layers.size()-1; i++) {
00139       layers.at(i).forward
00140     (layers.at(i).input(),
00141      layers.at(i+1).input()
00142      );
00143     }
00144     layers.back().forward(layers.back().input(),output);
00145     cout « "last layer : "
00146      « layers.back().input().data()[0] « " -> " « output.data()[0] « "\n";
00147   }
00148 };
```

Here is the call graph for this function:



#### 4.6.3.15 info()

```
void Net::info ( )
```

Definition at line 515 of file net.cpp.

```
00515                     {
00516     cout « "Model info\n---------------\n";
00517 
00518     for ( auto l : layers ) {
00519         cout « "Weights: " « l.output_size() « " x " « l.input_size() « "\n";
00520         cout « "Biases: " « l.biases.size() « "\n";
00521         cout « "Activation: " « l.activation_name « "\n";
00522 
00523         // switch (l.activation) {
00524         //  case RELU: cout « "RELU\n"; break;
00525         //  case SIG: cout « "Sigmoid\n"; break;
00526         //  case SMAX: cout « "Softmax\n"; break;
00527         //  case NONE: break;
00528         // }
00529         cout « "---------------\n";
00530     }
00531 
00532 }
```

#### 4.6.3.16 inputsize()

```
int Net::inputsize (
              int layer ) const  [inline]
```

Definition at line 60 of file net.h.

```
00060                                     {
00061     assert(layer>=0); assert(layer<=layers.size());
00062     if (layers.empty())
00063         return inR; // Input's row size for the first layer
00064     else
00065         return layers.at(layer-1).output_size(); // Previous layer's row size
00066     };
```

#### 4.6.3.17 layer()

```
const Layer & Net::layer (
              int i ) const  [inline]
```

Definition at line 77 of file net.h.

```
00077 { return layers.at(i); };
```

#### 4.6.3.18 learning_rate()

```
float Net::learning_rate ( ) const  [inline]
```

Definition at line 112 of file net.h.

```
00112 { return _lr; };
```

#### 4.6.3.19 loadModel()

```
void Net::loadModel (
        std::string path )
```

Definition at line 486 of file net.cpp.

```
00486                                  {
00487     std::ifstream file(path);
00488     std::string buffer;
00489 
00490     int no_layers;
00491     file.read( reinterpret_cast<char *>(&no_layers), sizeof(no_layers) );
00492 
00493     float temp;
00494     layers.resize(no_layers);
00495     for ( int i=0; i < layers.size(); i++ ) {
00496         int insize,outsize;
00497         file.read( reinterpret_cast<char *>(&outsize), sizeof(int) );
00498         file.read( reinterpret_cast<char *>(&insize), sizeof(int) );
00499 
00500         file.read( reinterpret_cast<char *>(&layers[i].activation), sizeof(int) );
00501 
00502         layers[i].weights = Matrix( outsize, insize, 0 );
00503         float *w_data = layers[i].weights.data();
00504         file.read(reinterpret_cast<char *>( w_data ), //(&layers[i].weights.mat[0]),
00505               sizeof(temp) * insize*outsize);
00506 
00507         layers[i].biases = Vector( outsize, 0 );
00508         float *b_data = layers[i].biases.data();
00509         file.read(reinterpret_cast<char *>( b_data ), //(&layers[i].biases.vals[0]),
00510               sizeof(temp) * layers[i].biases.size());
00511     }
00512 }
```

#### 4.6.3.20 momentum()

```
float Net::momentum ( ) const  [inline]
```

Definition at line 122 of file net.h.

```
00122 { return _momentum; };
```

#### 4.6.3.21 optimizer()

```
int Net::optimizer ( ) const  [inline]
```

Definition at line 127 of file net.h.

```
00127 { return _optimizer; };
```

#### 4.6.3.22 outputsize()

```
int Net::outputsize ( ) const  [inline]
```

Definition at line 56 of file net.h.

```
00056                        {
00057     assert( layers.size()>0 );
00058     return layers.back().output_size();
00059     };
```

#### 4.6.3.23 push_layer()

```
void Net::push_layer (
            const Layer & layer )
```

Push a layer as last layer of a network

Definition at line 76 of file net.cpp.

```
00076                                         {
00077     if (trace_progress())
00078     cout « "Creating layer " « layers.size()
00079         « " of size " « layer.output_size() « "x" « layer.input_size()
00080        //       « " with " « name « " activation"
00081         « endl;
00082     layers.push_back(layer);
00083 };
```

#### 4.6.3.24 RMSprop()

```
void Net::RMSprop (
            float lr,
            float momentum )
```

Definition at line 256 of file net.cpp.

```
00256                                          {
00257     for (int i = 0; i < layers.size(); i++) {
00258         // Get average over all the gradients
00259         Matrix deltaWsq = layers.at(i).dw;
00260         Vector deltaBsq = layers.at(i).db;
00261 
00262     // Gradient step
00263         deltaWsq.square(); // dW^2
00264         deltaBsq.square(); // db^2
00265         // Sdw := m*Sdw + (1-m) * dW^2
00266 
00267     layers.at(i).dw_velocity = momentum * layers.at(i).dw_velocity + (1 - momentum) * deltaWsq;
00268         layers.at(i).db_velocity = momentum * layers.at(i).db_velocity + (1 - momentum) * deltaBsq;
00269 
00270     Matrix sqrtSdw = layers.at(i).dw_velocity;
00271         std::for_each(sqrtSdw.values().begin(), sqrtSdw.values().end(),
00272             [](auto &n) {
00273             n = sqrt(n);
00274             if(n==0) n= 1-1e-7;
00275             });
00276     Vector sqrtSdb = layers.at(i).db_velocity;
00277         std::for_each(sqrtSdb.values().begin(), sqrtSdb.values().end(),
00278             [](auto &n) {
00279             n = sqrt(n);
00280             if(n==0) n= 1-1e-7;
00281             });
00282 
00283         // W := W - lr * dW / sqrt(Sdw)
00284         layers.at(i).weights = layers.at(i).weights - lr * layers.at(i).dw / sqrtSdw;
00285         layers.at(i).biases = layers.at(i).biases - lr * layers.at(i).db / sqrtSdb;
00286 
00287         // Reset the values of delta sums
00288         layers.at(i).dw.zeros();
00289         layers.at(i).db.zeros();
00290     }
00291 }
```

### 4.6.3.25 saveModel()

```
void Net::saveModel (
            std::string path )
```

Write the model to file

Definition at line 453 of file net.cpp.

```
00453                                 {
00454     /*
00455     1. Size of matrix m x n, activation function
00456     2. Values of weight matrix
00457     3. Values of bias vector
00458     4. Repeat for all layers
00459     */
00460     std::ofstream file;
00461     file.open( path, std::ios::binary );
00462 
00463     float temp;
00464     int no_layers = layers.size();
00465     file.write( reinterpret_cast<char *>(&no_layers), sizeof(no_layers) );
00466     for ( auto l : layers ) {
00467         int insize = l.input_size(), outsize = l.output_size();
00468         file.write( reinterpret_cast<char *>(&outsize), sizeof(int) );
00469         file.write( reinterpret_cast<char *>(&insize), sizeof(int) );
00470         file.write( reinterpret_cast<char *>(&l.activation), sizeof(int) );
00471 
00472         const auto& weights = l.weights;
00473         file.write(reinterpret_cast<const char *>(weights.data()), sizeof(float)*weights.nelements());
00474         // const float* weights_data = l.weights.data();
00475         // file.write(reinterpret_cast<char *>(&weights_data), sizeof(temp)*insize*outsize);
00476 
00477         const auto& biases = l.biases;
00478         file.write(reinterpret_cast<const char*>(biases.data()), sizeof(float) * biases.size());
00479         //file.write(reinterpret_cast<char *>(&l.biases.vals[0]), sizeof(temp) * l.biases.size());
00480     }
00481     cout « endl;
00482 
00483     file.close();
00484 }
```

### 4.6.3.26 set_decay()

```
void Net::set_decay (
            float d )  [inline]
```

Definition at line 116 of file net.h.

```
00116 { _decay = d; };
```

### 4.6.3.27 set_learning_rate()

```
void Net::set_learning_rate (
            float lr )  [inline]
```

Definition at line 111 of file net.h.

```
00111 { _lr=lr; };
```

#### 4.6.3.28 set_lossfunction()

```
void Net::set_lossfunction (
            lossfn lossFuncName )
```

Set an indexed loss function

Definition at line 100 of file net.cpp.

```
00100                                                     {
00101   lossFunction = lossFunctions.at(lossFuncName);
00102   d_lossFunction = d_lossFunctions.at(lossFuncName);
00103 };
```

#### 4.6.3.29 set_momentum()

```
void Net::set_momentum (
            float m )  [inline]
```

Definition at line 121 of file net.h.

```
00121 { _momentum = m; };
```

#### 4.6.3.30 set_optimizer()

```
void Net::set_optimizer (
            int m )  [inline]
```

Definition at line 126 of file net.h.

```
00126 { _optimizer = m; };
```

#### 4.6.3.31 set_uniform_biases()

```
void Net::set_uniform_biases (
            float v )
```

Initialize biases of all layers to a uniform value

Definition at line 116 of file net.cpp.

```
00116                                         {
00117   for ( auto& l : layers )
00118     l.set_uniform_biases(v);
00119 };
```

#### 4.6.3.32 set_uniform_weights()

```
void Net::set_uniform_weights (
                float v )
```

Initialize weights of all layers to a uniform value

Definition at line 108 of file net.cpp.

```
00108                                       {
00109   for ( auto& l : layers )
00110     l.set_uniform_weights(v);
00111 };
```

#### 4.6.3.33 SGD()

```
void Net::SGD (
                float lr,
                float momentum )
```

Stochastic Gradient Descent algorithm

Definition at line 230 of file net.cpp.

```
00230                                        {
00231     assert( layers.size()>0 );
00232     int samplesize = layers.front().input().batch_size();
00233     for (int i = 0; i < layers.size(); i++) {
00234         // Normalize gradients to avoid exploding gradients
00235     Matrix deltaW = layers.at(i).dw / static_cast<float>(samplesize);
00236         Vector deltaB = layers.at(i).db / static_cast<float>(samplesize);
00237
00238         // Gradient descent
00239         if (momentum > 0.0) {
00240             layers.at(i).dw_velocity = momentum * layers.at(i).dw_velocity - lr * deltaW;
00241             //layers.at(i).weights = layers.at(i).weights + layers.at(i).dw_velocity;
00242         layers.at(i).weights.axpy( 1.f,layers.at(i).dw_velocity );
00243         } else {
00244       //layers.at(i).weights = layers.at(i).weights - lr * deltaW;
00245       layers.at(i).weights.axpy( -lr,deltaW );
00246         }
00247
00248         layers.at(i).biases = layers.at(i).biases - lr * deltaB;
00249
00250         // Reset the values of delta sums
00251         layers.at(i).dw.zeros();
00252         layers.at(i).db.zeros();
00253     }
00254 }
```

#### 4.6.3.34 show()

```
void Net::show ( )
```

Show the weights of all layers

Definition at line 167 of file net.cpp.

```
00167                     {
00168     for (unsigned i = 0; i < layers.size(); i++) {
00169         cout << "Layer " << i << " weights" << endl;
00170         layers.at(i).weights.show();
00171     }
00172 }
```

#### 4.6.3.35 train()

```
void Net::train (
        const Dataset & train,
        const Dataset & test,
        int epochs,
        int batchSize )
```

Definition at line 300 of file net.cpp.

```
                                    {

    const int Optimizer = optimizer();
    cout « "Optimizing with ";
    switch (Optimizer) {
    case sgd:  cout « "Stochastic Gradient Descent\n";  break;
    case rms:  cout « "RMSprop\n"; break;
    }

    std::vector<Dataset> batches = train_data.batch(batchSize);
    float lrInit = learning_rate();
    const float momentum_value = momentum();

    for (int i_epoch = 0; i_epoch < epochs; i_epoch++) {
      // Iterate through the entire dataset for each epoch
      cout « endl « "Epoch " « i_epoch+1 « "/" « epochs « endl;
      float current_learning_rate = lrInit; // Reset the learning rate to undo decay

      for (int j = 0; j < batches.size(); j++) {
    // Iterate through all batches within dataset
    auto& batch = batches.at(j);
#ifdef DEBUG
    cout « ".. batch " « j « "/" « batches.size() « " of size " « batch.size() « "\n";
#endif
    //  allocate_batch_specific_temporaries(batch.size());
    VectorBatch batch_output( batch.inputs().item_size(),batch.size() );
    feedForward(batch.inputs(),batch_output);
    backPropagate(batch.inputs(),batch.labels(),batch_output);

        // User chosen optimizer
        current_learning_rate = current_learning_rate / (1 + decay() * j);
        optimize.at(Optimizer)(current_learning_rate, momentum_value);

      }
      auto loss = calculateLoss(test_data);
      cout « " Loss: " « loss « endl;
      auto acc = accuracy(test_data);
      cout « " Accuracy on trest set: " « acc « endl;
    }

}
```

### 4.6.4 Member Data Documentation

#### 4.6.4.1 optimize

```
std::vector< std::function< void(float lr, float momentum) > > Net::optimize
```

**Initial value:**

```
{
    [this] ( float lr, float momentum ) { SGD(lr, momentum); },
    [this] ( float lr, float momentum ) { RMSprop(lr, momentum); }
  }
```

Definition at line 128 of file net.h.

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

- net.h
- net.cpp

## 4.7 Vector Class Reference

### Public Member Functions

- Vector (int n)
- Vector (std::vector< float > vals)
- Vector (int size, int init)
- int size () const
- void show ()
- void add (const Vector &v1)
- void set_ax (float a, Vector &x)
- std::vector< float > & values ()
- const std::vector< float > & values () const
- void copy_from (const VectorBatch &)
- float * data ()
- const float * data () const
- void zeros ()
- void square ()
- float & operator[ ] (int i)
- float operator[ ] (int i) const
- Vector operator- ()
- Vector & operator= (const Vector &m2)
- Vector operator+ (const Vector &m2)
- Vector operator* (const Vector &m2)
- Vector operator/ (const Vector &m2)
- Vector operator/= (float x)
- Vector operator- (const Vector &m2)
- bool positive () const

  *Test that all elements are positive.*

### Public Attributes

- int r
- int c =1

### Friends

- class VectorBatch
- class Matrix
- Vector operator- (const float &c, const Vector &m)
- Vector operator* (const float &c, const Vector &m)

### 4.7.1 Detailed Description

Definition at line 23 of file vector.h.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 Vector() [1/4]

Vector::Vector ( )

Definition at line 27 of file vector.cpp.

```
00027               {
00028 }
```

#### 4.7.2.2 Vector() [2/4]

Vector::Vector (
int *n* ) [inline]

Definition at line 30 of file vector.h.

```
00031     : vals(std::vector<float>(n)) {};
```

#### 4.7.2.3 Vector() [3/4]

Vector::Vector (
std::vector< float > *vals* )

Definition at line 32 of file vector.cpp.

```
00033   : vals(vals) {
00034   r = vals.size();
00035   c = 1;
00036 };
```

#### 4.7.2.4 Vector() [4/4]

Vector::Vector (
int *size*,
int *init* )

Definition at line 24 of file vector_impl_blis.cpp.

```
00024                             {
00025     r = s;
00026   vals = std::vector<float>(s);
00027   float scal_fac = 0.05;
00028   if (init==0){
00029     float zero = 0.0;
00030     bli_ssetv( BLIS_NO_CONJUGATE, s, &zero, &vals[0], 1);
00031   }else if (init==1)
00032     bli_srandv(s, &vals[0], 1);
00033     bli_sscalv( BLIS_NO_CONJUGATE, s, &scal_fac, &vals[0], 1 );
00034 }
```

### 4.7.3 Member Function Documentation

### 4.7.3.1 add()

```
void Vector::add (
        const Vector & v1 )
```

Definition at line 36 of file vector_impl_blis.cpp.

```
00036                                    {
00037   assert(v1.size()==this->size());
00038   bli_saddv( BLIS_NO_CONJUGATE, size(), const_cast<float*>(&v1.vals[0]), 1, &vals[0], 1 );
00039 }
```

### 4.7.3.2 copy_from()

```
void Vector::copy_from (
        const VectorBatch & batch )
```

Definition at line 38 of file vector.cpp.

```
00038                                                  {
00039     const auto& batch_vals = batch.data();
00040     assert( batch.nelements()>=vals.size() );
00041     for (int i=0; i<vals.size(); i++) {
00042     vals[i] = batch_vals[i];
00043     }
00044 };
```

### 4.7.3.3 data() [1/2]

```
float * Vector::data ( )  [inline]
```

Definition at line 42 of file vector.h.

```
00042 { return vals.data(); };
```

### 4.7.3.4 data() [2/2]

```
const float * Vector::data ( ) const  [inline]
```

Definition at line 43 of file vector.h.

```
00043 { return vals.data(); };
```

### 4.7.3.5 operator*()

```
Vector Vector::operator* (
        const Vector & m2 )
```

Definition at line 99 of file vector.cpp.

```
00099                                          { // Hadamard product
00100     Vector out(m2.size(), 0);
00101     for (int i = 0; i < m2.size(); i++) {
00102         out.vals[i] = this->vals[i] * m2.vals[i];
00103     }
00104     return out;
00105 }
```

#### 4.7.3.6 operator+()

```
Vector Vector::operator+ (
        const Vector & m2 )
```

Definition at line 57 of file vector.cpp.

```
00057                                              {
00058     assert(m2.size()==this->size());
00059     Vector out(m2.size(),0);
00060     for (int i=0;i<m2.size();i++) {
00061         out.vals[i] = this->vals[i] + m2.vals[i];
00062     }
00063     return out;
00064 }
```

#### 4.7.3.7 operator-() [1/2]

```
Vector Vector::operator- ( )
```

Definition at line 115 of file vector.cpp.

```
00115                              {
00116     Vector result = *this;
00117     for (int i = 0; i < size(); i++){
00118         result.vals[i] = -vals[i];
00119     }
00120 
00121     return result;
00122 };
```

#### 4.7.3.8 operator-() [2/2]

```
Vector Vector::operator- (
        const Vector & m2 )
```

Definition at line 66 of file vector.cpp.

```
00066                                              {
00067     assert(m2.size()==this->size());
00068     Vector out(m2.size(),0);
00069     for (int i=0;i<m2.size();i++) {
00070         out.vals[i] = this->vals[i] - m2.vals[i];
00071     }
00072     return out;
00073 }
```

#### 4.7.3.9 operator/()

```
Vector Vector::operator/ (
        const Vector & m2 )
```

Definition at line 107 of file vector.cpp.

```
00107                                              { // Element wise division
00108     Vector out(m2.size(), 0);
00109     for (int i = 0; i < m2.size(); i++) {
00110         out.vals[i] = this->vals[i] / m2.vals[i];
00111     }
00112     return out;
00113 }
```

### 4.7.3.10 operator/=()

```
Vector Vector::operator/= (
            float x )
```

Definition at line 92 of file vector.cpp.

```
00092                                 {
00093     for (int i=0;i<vals.size();i++) {
00094         vals[i] = vals[i] / c;
00095     }
00096     return *this;
00097 }
```

### 4.7.3.11 operator=()

```
Vector & Vector::operator= (
            const Vector & m2 )
```

Definition at line 50 of file vector.cpp.

```
00050                                         { // Overloading the = operator
00051     vals = m2.vals;
00052     return *this;
00053 }
```

### 4.7.3.12 operator[]() [1/2]

```
float & Vector::operator[] (
            int i ) [inline]
```

Definition at line 46 of file vector.h.

```
00046 { return vals[i]; };
```

### 4.7.3.13 operator[]() [2/2]

```
float Vector::operator[] (
            int i ) const [inline]
```

Definition at line 47 of file vector.h.

```
00047 { return vals[i]; };
```

### 4.7.3.14 positive()

```
bool Vector::positive ( ) const [inline]
```

Test that all elements are positive.

Definition at line 60 of file vector.h.

```
00060                         {
00061     return all_of
00062       ( vals.begin(),vals.end(),
00063     [] (float e) { return e>0; }
00064     );
00065   }
```

#### 4.7.3.15 set_ax()

```
void Vector::set_ax (
            float a,
            Vector & x )
```

Definition at line 41 of file vector_impl_blis.cpp.

```
00041                              {
00042   assert(x.size()==this->size());
00043
00044     float b = static_cast<float>(a);
00045     bli_sscal2v(BLIS_NO_CONJUGATE, this->size(),&b,&x.vals[0],1,&(this->vals)[0], 1);
00046 }
```

#### 4.7.3.16 show()

```
void Vector::show ( )
```

Definition at line 55 of file vector_impl_blis.cpp.

```
00055                 {
00056
00057     char sp[8] = " ";
00058     char format[8] = "%4.4f";
00059     bli_sprintm( sp, size(), 1, &vals[0], 1, size(), format, sp );
00060 }
```

#### 4.7.3.17 size()

```
int Vector::size ( ) const
```

Definition at line 30 of file vector.cpp.

```
00030 { return vals.size(); };
```

#### 4.7.3.18 square()

```
void Vector::square ( )
```

Definition at line 46 of file vector.cpp.

```
00046                  {
00047     std::for_each(vals.begin(), vals.end(), [](auto &n) {n*=n;});
00048 }
```

#### 4.7.3.19 values() [1/2]

```
std::vector< float > & Vector::values ( )  [inline]
```

Definition at line 39 of file vector.h.

```
00039 { return vals; };
```

#### 4.7.3.20 values() [2/2]

```
const std::vector< float > & Vector::values ( ) const [inline]
```

Definition at line 40 of file vector.h.

```
00040 { return vals; };
```

#### 4.7.3.21 zeros()

```
void Vector::zeros ( )
```

Definition at line 49 of file vector_impl_blis.cpp.

```
00049                     {
00050 
00051     float zero = 0.0;
00052     bli_ssetv( BLIS_NO_CONJUGATE, size(), &zero, &vals[0], 1 ); // Set all values to 0
00053 }
```

### 4.7.4 Friends And Related Function Documentation

#### 4.7.4.1 Matrix

```
friend class Matrix [friend]
```

Definition at line 25 of file vector.h.

#### 4.7.4.2 operator*

```
Vector operator* (
            const float & c,
            const Vector & m ) [friend]
```

Definition at line 84 of file vector.cpp.

```
00084                                                  {
00085     Vector o=m;
00086     for (int i=0;i<m.size();i++) {
00087         o.vals[i] = c * o.vals[i];
00088     }
00089     return o;
00090 }
```

#### 4.7.4.3 operator-

```
Vector operator- (
            const float & c,
            const Vector & m )  [friend]
```

Definition at line 76 of file vector.cpp.

```
                                                        {
    Vector o=m;
    for (int i=0;i<m.size();i++) {
        o.vals[i] = c - o.vals[i];
    }
    return o;
}
```

#### 4.7.4.4 VectorBatch

```
friend class VectorBatch  [friend]
```

Definition at line 24 of file vector.h.

### 4.7.5 Member Data Documentation

#### 4.7.5.1 c

```
int Vector::c =1
```

Definition at line 35 of file vector.h.

#### 4.7.5.2 r

```
int Vector::r
```

Definition at line 35 of file vector.h.

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

- vector.h
- vector.cpp
- vector_impl_blis.cpp
- vector_impl_reference.cpp

## 4.8 VectorBatch Class Reference

### Public Member Functions

- VectorBatch (int itemsize)

  *Construct and set the vector size.*
- VectorBatch (int nRows, int nCols, bool rand=false)
- VectorBatch (const Vector &)

  *Construct from a single vector.*
- void allocate (int, int)

  *resize the vals array*
- int size () const
- void resize (int m, int n)

  *resize the values vector*
- float normf () const

  *Frobenius norm.*
- bool positive () const

  *Test that all elements are positive.*
- bool notnan () const

  *Test that there are no NaN elements.*
- bool notinf () const

  *Test that there are no Inf elements.*
- int item_size () const

  *The size of any vector in the batch.*
- void set_item_size (int n)

  *Set the vector size.*
- int batch_size () const

  *How many vectors are there in this batch.*
- void set_batch_size (int n)

  *Set the batch size.*
- int nelements () const

  *Total number of eleements in the whole batch.*
- std::vector< float > & vals_vector ()

  *Return the values vector, for write.*
- const std::vector< float > & vals_vector () const

  *Return the values vector, for read only.*
- float * data ()

  *Return the float data. This is for the BLAS interface.*
- const float * data () const

  *Return the float data. This is for the BLAS interface.*
- const float * data (int disp) const

  *Return the float data with an offset. This is for the BLAS interface.*
- void v2mp (const Matrix &x, VectorBatch &y) const
- void v2tmp (const Matrix &x, VectorBatch &y) const
- void v2mtp (const Matrix &x, VectorBatch &y) const
- void outer2 (const VectorBatch &x, Matrix &y) const
- void add_vector (const std::vector< float > &v)

  *Extend the batch with a new vector.*
- float at (int i, int j) const

  *Return element i of the j'th vector in the batch, with bound checking.*
- void set_col (int j, const std::vector< float > &v)

*Copy a vector into batch j.*

- std::vector< float > get_col (int j) const
- void set_row (int j, const std::vector< float > &v)
- std::vector< float > get_row (int j) const
- std::vector< float > extract_vector (int v) const
- std::vector< float > get_vector (int v) const
- void set_vector (const Vector &v, int j)

  *Copy vector elements into vector j of this batch.*
- void **set_vector** (const std::vector< float > &v, int j)
- Vector get_vectorObj (int j) const

  *Extract one vector from the batch.*
- void show () const
- void display (std::string) const

  *Crude output.*
- void copy_from (const VectorBatch &in)

  *Copy a whole batch into this one.*
- void addh (const Vector &y)
- void **addh** (const VectorBatch &y)
- Vector meanh () const
- VectorBatch operator- ()
- VectorBatch & operator= (const VectorBatch &m2)
- void hadamard (const VectorBatch &m1, const VectorBatch &m2)
- VectorBatch operator* (const VectorBatch &m2)
- VectorBatch operator/ (const VectorBatch &m2)
- void scaleby (float)
- VectorBatch operator- (const VectorBatch &m2) const

## Friends

- class Matrix
- class Vector
- VectorBatch operator/ (const VectorBatch &m, const float &c)
- VectorBatch operator* (const float &c, const VectorBatch &m)

### 4.8.1 Detailed Description

Definition at line 32 of file vector2.h.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 VectorBatch() [1/4]

```
VectorBatch::VectorBatch ( )
```

Definition at line 34 of file vector2.cpp.

```
00034                           { // Default constructor
00035 }
```

### 4.8.2.2 VectorBatch() [2/4]

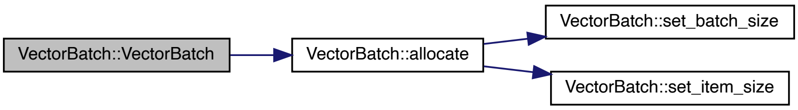
```
VectorBatch::VectorBatch (
            int itemsize )
```

Construct and set the vector size.

Definition at line 38 of file vector2.cpp.

```
00038                                   {
00039   allocate(0,vs);
00040 }
```

Here is the call graph for this function:




### 4.8.2.3 VectorBatch() [3/4]

```
VectorBatch::VectorBatch (
            int nRows,
            int nCols,
            bool rand = false )
```

Definition at line 29 of file vectorbatch_impl_blis.cpp.

```
00029                                                               {
00030   allocate(nRows,nCols);
00031   const int r=nRows, c=nCols;
00032 
00033   float scal_fac = 0.05; // randomize between (-scal;scal)
00034   if (not random){
00035     float zero = 0.0;
00036     bli_ssetm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
00037            r, c, &zero, &vals[0], c, 1);
00038   } else if (random){
00039     bli_srandm(0, BLIS_DENSE, r, c, &vals[0], c, 1);
00040     bli_sscalm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
00041         r, c, &scal_fac, &vals[0], c, 1);
00042   }
00043 }
```

#### 4.8.2.4 VectorBatch() [4/4]

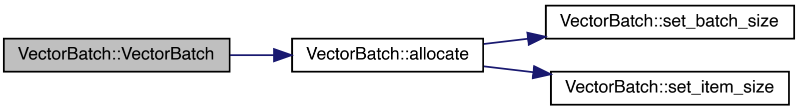
```
VectorBatch::VectorBatch (
            const Vector & v )
```

Construct from a single vector.

Definition at line 43 of file vector2.cpp.

```
00043                                    {
00044     int s = v.size();
00045     allocate(1,s);
00046     copy( v.vals.begin(),v.vals.end(),vals.begin() );
00047     // for ( int i=0; i<s; i++ )
00048     //  vals[i] = v[i];
00049 };
```

Here is the call graph for this function:




### 4.8.3 Member Function Documentation

#### 4.8.3.1 add_vector()

```
void VectorBatch::add_vector (
            const std::vector< float > & v )
```

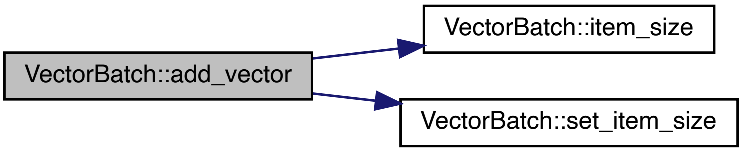
Extend the batch with a new vector.

**Todo** use the allocate method

Definition at line 83 of file vector2.cpp.

```
00083                                                  {
00084   const int Nelements = vals.size();
00085   const int vector_length = v.size();
00086   if (Nelements==0)
00087     set_item_size(vector_length);
00088   else {
00089     assert( vector_length==item_size() );
00090     assert( Nelements%vector_length==0 );
00091   }
00092   const int m = Nelements/vector_length;
00093   vals.resize(Nelements+vector_length); nvectors++;
00094   for (int i = 0; i < vector_length; i++) {
00095     vals.at( m * vector_length + i ) = v.at(i);
00096   };
00097 };
```

Here is the call graph for this function:




#### 4.8.3.2 addh()

```
void VectorBatch::addh (
             const Vector & y )
```

Definition at line 162 of file vector2.cpp.

```
00162                                      { // Add y to every row
00163   const int r = item_size(), c = batch_size();
00164   assert( r==y.size() );
00165   for (int j=0; j<c; j++ ) {
00166     for (int i=0; i<r; i++) {
00167       vals.at( INDEXc(i,j,r,c) ) += y.vals[i];
00168     }
00169   }
00170 }
```

#### 4.8.3.3 allocate()

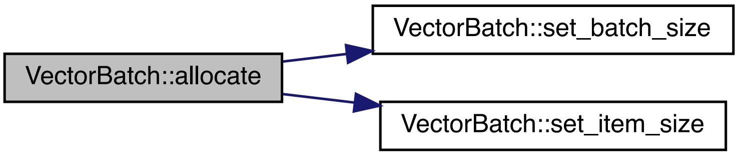
```
void VectorBatch::allocate (
             int batchsize,
             int itemsize )
```

resize the vals array

Definition at line 52 of file vector2.cpp.

```
00052                                                   {
00053   // we allow a batchsize of zero
00054   assert(batchsize>=0);
00055   assert(itemsize>0);
00056   vals.resize(batchsize*itemsize);
00057   set_batch_size(batchsize);
00058   set_item_size(itemsize);
00059 };
```

Here is the call graph for this function:




#### 4.8.3.4 at()

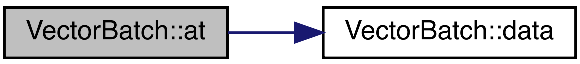
```
float VectorBatch::at (
            int i,
            int j ) const  [inline]
```

Return element i of the j'th vector in the batch, with bound checking.

Definition at line 120 of file vector2.h.

```
00120                                 {
00121       assert( i>=0 ); assert( i<vector_size );
00122       assert( j>= 0 ); assert( j<nvectors );
00123       return *data( i + j*vector_size );
00124       };
```

Here is the call graph for this function:




#### 4.8.3.5 batch_size()

```
int VectorBatch::batch_size ( ) const  [inline]
```

How many vectors are there in this batch.

Definition at line 87 of file vector2.h.

```
00087 { return nvectors; };
```

#### 4.8.3.6 copy_from()

```
void VectorBatch::copy_from (
            const VectorBatch & in ) [inline]
```

Copy a whole batch into this one.

Definition at line 155 of file vector2.h.

```
00155                                        {
00156     assert( vals.size()==in.vals.size() );
00157     std::copy( in.vals.begin(),in.vals.end(),vals.begin() );
00158     // for ( int i=0; i<vals.size(); i++)
00159     //   vals[i] = in.vals[i];
00160   };
```

#### 4.8.3.7 data() [1/3]

```
float * VectorBatch::data ( ) [inline]
```

Return the float data. This is for the BLAS interface.

Definition at line 102 of file vector2.h.

```
00102 { return vals.data(); };
```

#### 4.8.3.8 data() [2/3]

```
const float * VectorBatch::data ( ) const [inline]
```

Return the float data. This is for the BLAS interface.

Definition at line 104 of file vector2.h.

```
00104 { return vals.data(); };
```

#### 4.8.3.9 data() [3/3]

```
const float * VectorBatch::data (
            int disp ) const [inline]
```

Return the float data with an offset. This is for the BLAS interface.

Definition at line 106 of file vector2.h.

```
00106 { return vals.data()+disp; };
```

#### 4.8.3.10 display()

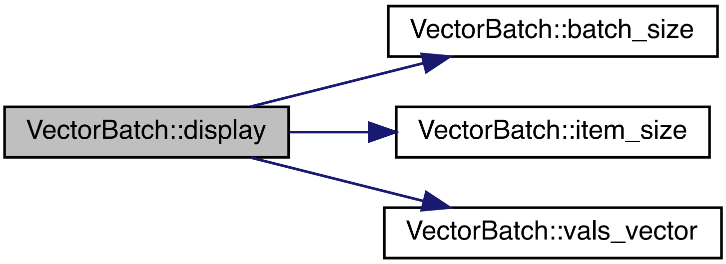
```
void VectorBatch::display (
            std::string  ) const
```

Crude output.

Definition at line 62 of file vector2.cpp.

```
00062                                                    {
00063   cout « header « "\n";
00064   for (int j=0; j<batch_size(); j++) {
00065     for (int i=0; i<item_size(); i++)
00066       cout « setprecision(5)
00067        « vals_vector().at( INDEXc(i,j,item_size(),batch_size()) )
00068        « " ";
00069     cout « "\n";
00070   }
00071 };
```

Here is the call graph for this function:




#### 4.8.3.11 extract_vector()

```
std::vector< float > VectorBatch::extract_vector (
            int v ) const
```

Definition at line 125 of file vector2.cpp.

```
00125                                                              {
00126   assert( j<batch_size() );
00127   const int m = item_size();
00128   std::vector<float> v(m);
00129   for (int i = 0; i < m; i++)
00130     v.at(i) = vals.at( INDEXc(i,j,m,batch_size()) );  // (j * n + i );
00131   return v;
00132   //  return get_row(v);
00133 };
```

#### 4.8.3.12 get_col()

```
std::vector< float > VectorBatch::get_col (
            int j ) const
```

Definition at line 99 of file vector2.cpp.

```
00099                                                   {
00100   assert( j<batch_size() );
00101   const int c = item_size();
00102   std::vector<float> col(c);
00103   for (int i=0; i<c; i++)
00104     col.at(i) = vals.at( j + c*i );
00105   return col;
00106 };
```

#### 4.8.3.13 get_row()

```
std::vector< float > VectorBatch::get_row (
            int j ) const
```

Definition at line 116 of file vector2.cpp.

```
00116                                                   {
00117   assert( j<batch_size() );
00118   const int n = item_size();
00119   std::vector<float> row(n);
00120   for (int i = 0; i < n; i++)
00121     row.at(i) = vals.at( j * n + i );
00122   return row;
00123 };
```

#### 4.8.3.14 get_vector()

```
std::vector< float > VectorBatch::get_vector (
            int v ) const
```

Definition at line 144 of file vector2.cpp.

```
00144                                                      {
00145   return extract_vector(v);
00146 };
```

#### 4.8.3.15 get_vectorObj()

```
Vector VectorBatch::get_vectorObj (
            int j ) const  [inline]
```

Extract one vector from the batch.

Definition at line 141 of file vector2.h.

```
00141                                        {
00142     Vector vec(vector_size);
00143     std::copy( vals.begin()+j*vector_size,vals.begin()+(j+1)*vector_size,
00144            vec.vals.begin() );
00145     // for (int i = 0; i < vector_size; i++ )
00146     //   vec.vals.at(i) = vals.at( j * vector_size + i );
00147     return vec;
00148   }
```

### 4.8.3.16 hadamard()

```
void VectorBatch::hadamard (
            const VectorBatch & m1,
            const VectorBatch & m2 )
```

Definition at line 232 of file vector2.cpp.

```
00232                                                           {
00233   const int r = item_size(), c = batch_size();
00234   assert( r==m1.item_size() ); assert( c==m1.batch_size() );
00235   assert( r==m2.item_size() ); assert( c==m2.batch_size() );
00236
00237   const auto& m1vals = m1.vals_vector();
00238   const auto& m2vals = m2.vals_vector();
00239   for (int i=0; i<r*c; i++) {
00240     vals.at(i) = m1vals.at(i) * m2vals.at(i);
00241     if ( isinf(vals.at(i)) )
00242       cout « "inf from " « m1vals.at(i) « " * " « m2vals.at(i) « "\n";
00243   }
00244   if (trace_progress()) {
00245     assert( m1.notinf() ); assert( m1.notnan() ); assert( m1.normf()!=0.f );
00246     assert( m2.notinf() ); assert( m2.notnan() ); assert( m2.normf()!=0.f );
00247     assert( this->notinf() ); assert( this->notnan() ); assert( this->normf()!=0.f );
00248   }
00249   if (trace_scalars()) {
00250     cout « "delta: "
00251      « m1.normf() « "x" « m2.normf() « " => " « this->normf() « "\n";
00252   }
00253 }
```

### 4.8.3.17 item_size()

```
int VectorBatch::item_size ( ) const  [inline]
```

The size of any vector in the batch.

Definition at line 83 of file vector2.h.

```
00083 { return vector_size; };
```

### 4.8.3.18 meanh()

```
Vector VectorBatch::meanh ( ) const
```

Definition at line 183 of file vector2.cpp.

```
00183                             { // Returns a vector of row-wise means
00184   const int r = item_size(), c = batch_size();
00185   Vector mean(r, 0);
00186   for (int i=0; i<r; i++) {
00187     float avg = 0.f;
00188     for ( int j=0; j<c; j++ ) {
00189       avg += vals.at( INDEXc(i,j,r,v) );
00190     }
00191     mean.vals[i] = avg/static_cast<float>( item_size() );
00192   }
00193   return mean;
00194 }
```

#### 4.8.3.19 nelements()

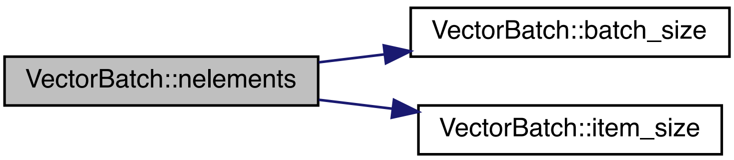
```
int VectorBatch::nelements ( ) const  [inline]
```

Total number of eleements in the whole batch.

Definition at line 91 of file vector2.h.

```
00091                            {
00092       int n = vals.size();
00093       assert( n==item_size()*batch_size() );
00094       return n;
00095     };
```

Here is the call graph for this function:




#### 4.8.3.20 normf()

```
float VectorBatch::normf ( ) const  [inline]
```

Frobenius norm.

Definition at line 53 of file vector2.h.

```
00053                          {
00054       float norm{0.f}; int count{0};
00055       for ( auto e : vals ) {
00056     norm += e*e; count++;
00057       }
00058       //std::cout « "norm squared over " « count « " elements: " « norm « "\n";
00059       return sqrt(norm);
00060     };
```

#### 4.8.3.21 notinf()

```
bool VectorBatch::notinf ( ) const  [inline]
```

Test that there are no Inf elements.

Definition at line 76 of file vector2.h.

```
00076                          {
00077       return all_of
00078     ( vals.begin(),vals.end(),
00079       [] (float e) { return not isinf(e); }
00080     );
00081     }
```

### 4.8.3.22 notnan()

```
bool VectorBatch::notnan ( ) const  [inline]
```

Test that there are no NaN elements.

Definition at line 69 of file vector2.h.

```
00069                        {
00070       return all_of
00071     ( vals.begin(),vals.end(),
00072       [] (float e) { return not isnan(e); }
00073     );
00074     }
```

### 4.8.3.23 operator*()

```
VectorBatch VectorBatch::operator* (
              const VectorBatch & m2 )
```

Definition at line 221 of file vector2.cpp.

```
00221                                                   { // Hadamard product
00222   assert( item_size()==m2.item_size() );
00223   assert( batch_size()==m2.batch_size() );
00224   const int c = m2.item_size(), r = m2.batch_size();
00225   VectorBatch out(r, c, 0);
00226   for (int i = 0; i < nelements(); i++) {
00227       out.vals[i] = this->vals[i] * m2.vals[i];
00228     }
00229     return out;
00230 }
```

### 4.8.3.24 operator-() [1/2]

```
VectorBatch VectorBatch::operator- ( )
```

Definition at line 272 of file vector2.cpp.

```
00272                                     {
00273     VectorBatch result = *this;
00274     for (int i = 0; i < nelements(); i++) {
00275         result.vals[i] = -vals[i];
00276     }
00277 
00278     return result;
00279 };
```

### 4.8.3.25 operator-() [2/2]

```
VectorBatch VectorBatch::operator- (
              const VectorBatch & m2 ) const
```

Definition at line 210 of file vector2.cpp.

```
00210                                                         {
00211   assert( item_size()==m2.item_size() );
00212   assert( batch_size()==m2.batch_size() );
00213   const int c = m2.item_size(), r = m2.batch_size();
00214   VectorBatch out(r, c, 0);
00215     for (int i = 0; i < r * c; i++) {
00216       out.vals[i] = this->vals[i] - m2.vals[i];
00217     }
00218     return out;
00219 }
```

#### 4.8.3.26 operator/()

```
VectorBatch VectorBatch::operator/ (
            const VectorBatch & m2 )
```

Definition at line 255 of file vector2.cpp.

```
00255                                                     { // Hadamard product
00256   const int c = m2.item_size(), r = m2.batch_size();
00257   assert( item_size()==c );
00258   assert( batch_size()==r );
00259   VectorBatch out(r, c, 0);
00260   for (int i = 0; i < nelements(); i++) {
00261     out.vals[i] = this->vals[i] / m2.vals[i];
00262   }
00263   return out;
00264 }
```

#### 4.8.3.27 operator=()

```
VectorBatch & VectorBatch::operator= (
            const VectorBatch & m2 )
```

Definition at line 200 of file vector2.cpp.

```
00200                                                      { // Overloading the = operator
00201   set_item_size( m2.item_size() );
00202   set_batch_size( m2.batch_size() );
00203
00204   this->vals = m2.vals; // IM Since we're using vectors we can just use the assignment from that
00205
00206   return *this;
00207 }
```

#### 4.8.3.28 outer2()

```
void VectorBatch::outer2 (
            const VectorBatch & x,
            Matrix & y ) const
```

Definition at line 163 of file vectorbatch_impl_blis.cpp.

```
00163                                                           {
00164   const int
00165     yr = item_size(),   yc = batch_size(),   // column storage
00166     mr = m.rowsize(),   mc = m.colsize(),    // row storage
00167     xr = x.item_size(), xc = x.batch_size(); // column
00168
00169   if (trace_scalars())
00170     cout « "outer product "
00171      « xr « "x" « xc
00172      « " & "
00173      « yr « "x" « yc
00174      « " => " « mr « "x" « mc
00175      « endl;
00176
00177   assert( yc==xc );
00178   assert( xr==mr );
00179   assert( yr==mc );
00180   const auto& xvals = x.vals_vector().data();
00181   const auto& yvals =   vals_vector().data();
00182   auto        mmat  = m.values().data();
00183
00184   float alpha = 1.0;
00185   float beta = 0.0;
00186   bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_TRANSPOSE,
00187          mr,mc,yc,
00188          &alpha,
00189          const_cast<float*>(xvals), /* rsa,csa */ 1,xr,
00190          const_cast<float*>(yvals), /* rsb,csb */ 1,yr,
00191          &beta,
00192          mmat,                      /* rsc,csc */ mc,1
00193          );
00194
00195 }
```

#### 4.8.3.29 positive()

```
bool VectorBatch::positive ( ) const  [inline]
```

Test that all elements are positive.

Definition at line 62 of file vector2.h.

```
00062                             {
00063       return all_of
00064     ( vals.begin(),vals.end(),
00065       [] (float e) { return e>0; }
00066     );
00067     }
```

#### 4.8.3.30 resize()

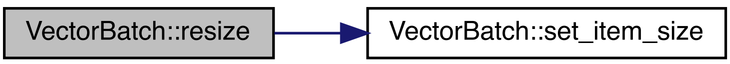
```
void VectorBatch::resize (
             int m,
             int n )  [inline]
```

resize the values vector

Definition at line 49 of file vector2.h.

```
00049                                {
00050       nvectors = m; set_item_size(n); //r = m; c = n;
00051       vals.resize(m*n); };
```

Here is the call graph for this function:




#### 4.8.3.31 scaleby()

```
void VectorBatch::scaleby (
             float f )
```

Definition at line 266 of file vector2.cpp.

```
00266                                     {
00267   for (int i = 0; i < nelements(); i++) {
00268     vals[i] *= f;
00269   }
00270 }
```

#### 4.8.3.32 set_batch_size()

```
void VectorBatch::set_batch_size (
             int n )  [inline]
```

Set the batch size.

**Todo** should be private, maybe eliminated?

Definition at line 89 of file vector2.h.

```
00089 { nvectors = n; };
```

#### 4.8.3.33 set_col()

```
void VectorBatch::set_col (
             int j,
             const std::vector< float > & v )
```

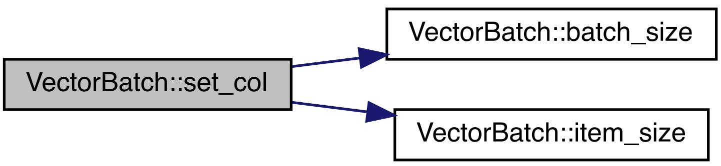
Copy a vector into batch j.

**Todo** rename to set_vector?

Definition at line 74 of file vector2.cpp.

```
00074                                                        {
00075   assert( j<batch_size() );
00076   assert( v.size()==item_size() );
00077   for (int i = 0; i<nvectors; i++) {
00078     vals.at( j + vector_size * i ) = v.at(i);
00079   };
00080 };
```

Here is the call graph for this function:



#### 4.8.3.34 set_item_size()

```
void VectorBatch::set_item_size (
            int n )  [inline]
```

Set the vector size.

**Todo** should be private, maybe eliminated?

Definition at line 85 of file vector2.h.

```
00085 { vector_size = n; };
```

#### 4.8.3.35 set_row()

```
void VectorBatch::set_row (
            int j,
            const std::vector< float > & v )
```

Definition at line 108 of file vector2.cpp.

```
00108                                                             {
00109   const int c = item_size();
00110   assert( v.size()==c );
00111   for (int i = 0; i < c; i++) {
00112     vals.at( j * c + i ) = v.at(i);
00113   };
00114 }
```

#### 4.8.3.36 set_vector()

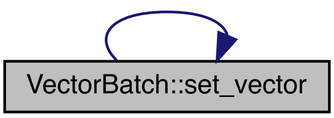
```
void VectorBatch::set_vector (
            const Vector & v,
            int j )
```

Copy vector elements into vector j of this batch.

Definition at line 150 of file vector2.cpp.

```
00150                                                      {
00151   set_vector( v.vals,j );
00152 }
```

Here is the call graph for this function:



#### 4.8.3.37 show()

```
void VectorBatch::show ( ) const
```

Definition at line 54 of file vectorbatch_impl_blis.cpp.

```
00054                                         {
00055 
00056     const int c = batch_size(), r = item_size();
00057     char e[5] = "";
00058     char forvals[8] = "%4.4f";
00059     bli_sprintm( e, r, c, const_cast<float*>(&vals[0]), c, 1, forvals, e );
00060 }
```

#### 4.8.3.38 size()

```
int VectorBatch::size ( ) const  [inline]
```

Definition at line 47 of file vector2.h.

```
00047 { return vals.size(); };
```

#### 4.8.3.39 v2mp()

```
void VectorBatch::v2mp (
              const Matrix & x,
              VectorBatch & y ) const
```

Definition at line 69 of file vectorbatch_impl_blis.cpp.

```
00069                                                                       {
00070   const int
00071     xr = item_size(),    xc = batch_size(),    // column storage
00072     mr = m.rowsize(),    mc = m.colsize(),     // row storage
00073     yr = y.item_size(), yc = y.batch_size(); // column
00074 
00075   if (trace_scalars())
00076     cout « "matrix vector product "
00077      « mr « "x" « mc
00078      « " & "
00079      « xr « "x" « xc
00080      « " => " « yr « "x" « yc
00081      « endl;
00082 
00083   assert( xc==yc );
00084   assert( yr==mr );
00085   assert( mc==xr );
00086 
00087   const auto& mmat  = m.values().data();
00088   const auto& xvals = vals_vector().data();
00089   auto  yvals       = y.vals_vector().data();
00090 
00091   float alpha = 1.0;
00092   float beta = 0.0;
00093   bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_NO_TRANSPOSE,
00094          yr,yc,mc,
00095          &alpha,
00096          const_cast<float*>(mmat),  /* rsa,csa */ mr,1,
00097          const_cast<float*>(xvals), /* rsb,csb */ 1,xr,
00098          &beta,
00099          yvals,                     /* rsc,csc */ 1,yr
00100          );
00101 }
```

#### 4.8.3.40 v2mtp()

```
void VectorBatch::v2mtp (
            const Matrix & x,
            VectorBatch & y ) const
```

Definition at line 120 of file vectorbatch_impl_blis.cpp.

```
00120                                                              {
00121   const int
00122     xr = item_size(),    xc = batch_size(),    // column storage
00123     mr = m.rowsize(),    mc = m.colsize(),     // row storage
00124     yr = y.item_size(), yc = y.batch_size(); // column
00125
00126   if (trace_scalars())
00127     cout « "matrix transpose vector product "
00128      « mr « "x" « mc
00129      « " & "
00130      « xr « "x" « xc
00131      « " => " « yr « "x" « yc
00132      « endl;
00133
00134   assert( xc==yc );
00135   assert( yr==mc );
00136   assert( mr==xr );
00137
00138   const auto& mmat  = m.values().data();
00139   const auto& xvals = vals_vector().data();
00140   auto         yvals = y.vals_vector().data();
00141
00142   float alpha = 1.0;
00143   float beta = 0.0;
00144   bli_sgemm( BLIS_TRANSPOSE, BLIS_NO_TRANSPOSE,
00145          yr,yc,mr,
00146          &alpha,
00147          const_cast<float*>(mmat),   /* rsa,csa */ mc,1,
00148          const_cast<float*>(xvals), /* rsb,csb */ 1,xr,
00149          &beta,
00150          yvals,                      /* rsc,csc */ 1,yr
00151          );
00152   if (trace_progress()) {
00153     assert( this->notinf() ); assert( this->notnan() ); assert( this->normf()!=0.f );
00154     assert( m.notinf() ); assert( m.notnan() ); assert( m.normf()!=0.f );
00155     assert( y.notinf() ); assert( y.notnan() ); assert( y.normf()!=0.f );
00156   }
00157
00158 }
```

#### 4.8.3.41 v2tmp()

```
void VectorBatch::v2tmp (
            const Matrix & x,
            VectorBatch & y ) const
```

Definition at line 103 of file vectorbatch_impl_blis.cpp.

```
00103                                                              {
00104
00105     const int c = batch_size(), r = item_size();
00106     assert( r==x.rowsize() );
00107     assert( c==y.batch_size() );
00108     assert( x.colsize()==y.item_size() );
00109
00110     float alpha = 1.0;
00111     float beta = 0.0;
00112     //printf("BLIS gemm %dx%dx%d\n",r,x.colsize(),c);
00113     bli_sgemm( BLIS_TRANSPOSE, BLIS_NO_TRANSPOSE,
00114            c, x.colsize(), r, &alpha, const_cast<float*>(&vals[0]),
00115            c, 1, const_cast<float*>( x.data() ),
00116            x.colsize(), 1, &beta, &y.vals[0], x.colsize(), 1);
00117 }
```

#### 4.8.3.42 vals_vector() [1/2]

```
std::vector< float > & VectorBatch::vals_vector ( ) [inline]
```

Return the values vector, for write.

Definition at line 98 of file vector2.h.

```
00098 { return vals; };
```

#### 4.8.3.43 vals_vector() [2/2]

```
const std::vector< float > & VectorBatch::vals_vector ( ) const [inline]
```

Return the values vector, for read only.

Definition at line 100 of file vector2.h.

```
00100 { return vals; };
```

### 4.8.4 Friends And Related Function Documentation

#### 4.8.4.1 Matrix

```
friend class Matrix [friend]
```

Definition at line 33 of file vector2.h.

#### 4.8.4.2 operator*

```
VectorBatch operator* (
        const float & c,
        const VectorBatch & m ) [friend]
```

Definition at line 289 of file vector2.cpp.

```
00289                                                                    {
00290     VectorBatch o = m;
00291     for (int i = 0; i < m.nelements(); i++) {
00292         o.vals[i] = o.vals[i] * c;
00293     }
00294     return o;
00295 }
```

#### 4.8.4.3 operator/

```
VectorBatch operator/ (
            const VectorBatch & m,
            const float & c )  [friend]
```

Definition at line 281 of file vector2.cpp.

```
                                                              {
    VectorBatch o = m;
    for (int i = 0; i < m.nelements(); i++) {
        o.vals[i] = o.vals[i] / c;
    }
    return o;
}
```

#### 4.8.4.4 Vector

```
friend class Vector  [friend]
```

Definition at line 34 of file vector2.h.

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

- vector2.h
- vector2.cpp
- vectorbatch_impl_blis.cpp
- vectorbatch_impl_reference.cpp

# Chapter 5

# File Documentation

## 5.1 configure.py

```
#!/usr/bin/env python3

import re
import sys

def usage():
    print("Usage: configure.py [ --blis BLIS_DIR ]")
    sys.exit(0)

args = sys.argv[1:]
if len(args)==0:
    print("No Make.inc generated")
    usage()



makeinc = open("Make.inc","w")
while len(args)>0:
    a,args = args[0],args[1:]
    if not re.match(r’--’,a):
        usage()
    elif re.match(r’--blis’,a):
        if len(args)==0:
            print("Missing blis argument")
            sys.exit(1)
        else:
            b,args = args[0],args[1:]
            if re.match(r’--’,b):
                print(f"Probably missing blis argument: «{b}»")
                sys.exit(1)
            makeinc.write(f"""CXX=clang++ -std=c++17 -fopenmp
            USE_BLIS=1
BLIS_INC_DIR={b}/include
BLIS_LIB_DIR={b}/lib
""")
    else:
        print(f"Unknown option {a}")
        sys.exit(1)

```

## 5.2 dataset.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** ‘Introduction to High-Performance Scientific Computing’
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/
```

```
#include <iostream>
using std::cout;
using std::endl;
#include <string>
using std::string;
#include <vector>
using std::vector;

#include <cstdio>
#include <cassert>
#include <algorithm>
using std::for_each;
#include <random>

#include "dataset.h"
#include "trace.h"

#define IMSIZE 28

Dataset::Dataset( int n ) : nclasses(n) {
  assert(n>=0);
};

Dataset::Dataset( std::vector<dataItem> dv ) {
  for ( const auto& v : dv )
    push_back(v);
};

void Dataset::set_lowerbound( int b ) { lowerbound = b; };
void Dataset::set_number( int b ) { number = b; };

void Dataset::push_back(dataItem it) {
  if (nclasses>0 and it.label_size()!=nclasses) {
    cout « "Set dimensionality " « nclasses
         « " does not match item size " « it.label_size() « endl;
    throw( string("Fail to add item to dataset") );
  }
  if (nclasses==0)
    nclasses = it.label_size();
  dataBatch.add_vector( it.data_values() );
  labelBatch.add_vector( it.label_values() );
  //_items.push_back(it);
};

dataItem Dataset::item(int i) const {
  return dataItem( data_vals(i),label_vals(i) );
};

int Dataset::size() const {
  int ds = dataBatch.batch_size(), ls = labelBatch.batch_size();
  assert( ds==ls );
  return ds;
}

int Dataset::data_size() const {
  if (dataBatch.batch_size()==0)
    throw( string("Can not get data size for empty dataset") );
  return dataBatch.item_size();
};

int Dataset::label_size() const {
  if (labelBatch.batch_size()==0)
    throw( string("Can not get label size for empty dataset") );
  return labelBatch.item_size();
}

const Vector& Dataset::data(int i) const {
  throw( string("Do not use Dataset::data") );
  //  return _items.at(i).data;
};
const vector<float> Dataset::data_vals(int i) const {
  return dataBatch.extract_vector(i);
};
const vector<float> Dataset::label_vals(int i) const {
  return labelBatch.extract_vector(i);
};
vector<float> Dataset::stacked_data_vals(int i) const {
  throw( string("Do not use Dataset::stacked_data_vals") );
  return dataBatch.get_row(i);
};

vector<float> Dataset::stacked_label_vals(int i) const {
  throw( string("Do not use Dataset::stacked_label_vals") );
  return labelBatch.get_row(i);
};
```

```
int Dataset::readTest(std::string dataPath) {
    /*
     * This reader is specifically for a modified MNIST dataset which
     * does not include the file header, metadata, etc.
     * Link to the dataset: http://cis.jhu.edu/~sachin/digit/digit.html
     * I chose this dataset for now to make it easy to read the data;
     * in later iterations I will generalize the read function, maybe OpenCV support
     */

    FILE *file;
    std::string fileName;
    uint8_t temp[IMSIZE * IMSIZE]; // Image buffer to read data into
    for (int dataid = 0; dataid < 10; dataid++) {
        fileName = dataPath + "/data" + std::to_string(dataid); // Put together the path
        file = fopen(fileName.c_str(), "r");


        if (!file) { // File checking
            cout « "Error opening file" « endl;
            return -2; // Arbitrary error code
        }
        for (int k = 0; k < 1000; k++) {
            Vector imageVec(IMSIZE * IMSIZE, 0); // initialize matrix to be read into
            fread(temp, 1, IMSIZE * IMSIZE, file); // Read 28*28 into buffer

            for (int i = 0; i < IMSIZE; i++) {
                for (int j = 0; j < IMSIZE; j++) {
                    // Transfer from buffer into matrix
          float *i_data = imageVec.data();
          *( i_data + i * IMSIZE + j ) // imageVec.vals[i * IMSIZE + j]
            = static_cast<float>( temp[i * IMSIZE + j] );
                }
            }
            fseek(file, k * IMSIZE * IMSIZE, SEEK_SET); // Seek to the kth image bytes

        Categorization label(10,dataid);

            dataItem x = {imageVec, label}; // Initialize an item with the data and the label in it
            push_back(x); // Store in the vector

        }
        fclose(file);
    }
    return 0;
}


void Dataset::shuffle() {
    std::random_device r;
    std::seed_seq seed{r(), r(), r(), r(), r(), r(), r(), r()}; // Seed
    std::mt19937 eng1(seed); // Randomizer engine

    throw( string("shuffling doesn't work") );
    //    std::shuffle(begin(_items), end(_items), eng1); // Shuffle the dataset
    return; // todo add return codes instead of printing
}


std::vector<Dataset> Dataset::batch(int batch_size) const {

  std::vector<Dataset> batches;
  int nitems = size(), itemsize = data_size();
  int nbatches = nitems/batch_size + ( nitems%batch_size>0 ? 1 : 0 );
  for (int b=0; b<nbatches; b++) {
    int first = b*batch_size, last= std::min( (b+1)*batch_size,nitems );
    Dataset batch; //(itemsize,last-first);
    batch.set_lowerbound(first); batch.set_number(b);
    for (int i=first; i<last; i++) {
      batch.push_back( item(i) );
    }
    batches.push_back(batch);
  }

  return batches;
}

void Dataset::stack() { // Stacks vectors horizontally (column-wise) in a Matrix object
  throw( string("Stacking no longer needed") );
    //dataBatch  = VectorBatch( data_size(),  size(), 0);
    //labelBatch = VectorBatch( label_size(), size(), 0);

    dataBatch  = VectorBatch( size(),  data_size(), 0);
    labelBatch = VectorBatch( size(), label_size(), 0);

    for (int j = 0; j < size(); j++) {
        //dataBatch.set_col( j,data_vals(j) );
        dataBatch.set_row( j, data_vals(j) );
```

```
    }

    for (int j = 0; j < size(); j++) {
        //labelBatch.set_col( j,label_vals(j) );
        labelBatch.set_row( j, label_vals(j) );
    }
}


/* forward definition, see below */ vector<int> permutation(int N);
std::pair< Dataset,Dataset > Dataset::split(float trainFraction) const {
    int dataset_size = size(); // _items.size();
    int testSize{0},trainSize;
    while ( true ) {
      trainSize= ceil( static_cast<float>( dataset_size ) * trainFraction);
      testSize = dataset_size - trainSize;
      if ( testSize>0 ) break;
      trainFraction *= .9;
    }

    auto random_index = permutation(dataset_size);
    Dataset trainSplit;
    if (trace_progress())
      cout « "split into " « trainSize « "+" « testSize « endl;
    for (int i=0; i<trainSize; i++) {
      const auto& di = item( random_index[i] );
      trainSplit.push_back(di);
    }
    Dataset testSplit;
    for (int i=trainSize; i<trainSize+testSize; i++) {
      const auto& di = item( random_index[i] );
      testSplit.push_back(di);
    }

    return std::make_pair(trainSplit,testSplit);
}

vector<int> permutation(int N) {

  std::random_device r;
  std::default_random_engine generator; // {r()};
  std::uniform_int_distribution<> distribution(0,N-1);

  vector<int> numbers(N);
  for (int i=0; i<N; i++)
    numbers[i] = i;

  for (int pass=0; pass<N; pass++) {
    int
      i = distribution(generator),
      j = distribution(generator);
    int t = numbers[i]; numbers[i] = numbers[j]; numbers[j] = t;
  }

  return numbers;
}
```

## 5.3 dataset.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#ifndef CODE_DATASET_H
#define CODE_DATASET_H
//#include "matrix.h"
#include "vector2.h"
#include "vector.h"
#include <vector>
#include <iostream>

class dataItem{
public: // should really be done through friends private:
    Vector data; // Data matrix
```

```
  //Vector label; // Label
    Categorization label; // Label
public:
  dataItem( Vector data,/* Vector */ Categorization label)
    : data(data),label(label) {};
  dataItem( float data,Categorization label)
    : data( std::vector<float>(data) ),label(label) {};
  dataItem( std::vector<float> indata,std::vector<float> outdata )
    : data( Vector(indata) ),label( Categorization(outdata) ) {};
  int data_size() const { return data.size(); };
  const std::vector<float>& data_values() const { return data.values(); };
  int label_size() const { return label.size(); };
  const std::vector<float>& label_values() const { return label.probabilities(); };
};

class Dataset {
private:
  int nclasses{0};
  //  std::vector<dataItem> _items; // All the data: {data, label}
private:
    VectorBatch dataBatch;
    VectorBatch labelBatch;
  int lowerbound{0},number{0};
public:
  Dataset() {};
  Dataset( int n );
  Dataset( std::vector<dataItem> );

  void set_lowerbound( int b );
  void set_number( int b );
  void push_back(dataItem it);
  int size() const;
  int data_size() const;
  int label_size() const;

  dataItem item(int i) const;
  //  const dataItem& at(int i) const { return _items.at(i); };
  //  dataItem& at(int i) { return _items.at(i); };
  const Vector& data(int i) const;
  const std::vector<float> data_vals(int i) const;
  const std::vector<float> label_vals(int i) const;
  std::vector<float> stacked_data_vals(int i) const;
  std::vector<float> stacked_label_vals(int i) const;
  //  const Categorization& label(int i) const { return _items.at(i).label; };

  //  const std::vector<dataItem>& items() const { return _items; };
  //const Vector& label(int i) const { return items.at(i).label; };

    std::string path; // Path of the dataset
public:
  const auto& inputs() const { return dataBatch; };
  const auto& labels() const { return labelBatch; };

    int readTest(std::string dataPath); // Read modified MNIST Dataset
    void shuffle(); // Mix the dataset
    std::vector<Dataset> batch(int n) const; // Divides the dataset into n batches
    void stack();
    std::pair<Dataset,Dataset> split(float trainFraction) const; // Train-test split
};


#endif //CODE_DATASET_H
```

## 5.4 funcs.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include "funcs.h"
#include "trace.h"

#include <iostream>
using std::cout;
```

```
using std:: endl;
#include <iomanip>
using std::boolalpha;

#include <algorithm>
#include <numeric>
using std::accumulate;
#include <functional>
using std::function;
#include <vector>
using std::vector;

#include <cmath>
#include <cassert>

void relu_io(const VectorBatch &mm, VectorBatch &a) {

  VectorBatch m(mm);
  assert( a.item_size()==m.item_size() );
  assert( a.batch_size()==m.batch_size() );
        auto& avals = a.vals_vector();
  const auto& mvals = m.vals_vector();
  avals.assign(mvals.begin(),mvals.end());
  const float alpha = 0.01; // used for leaky relu, for regular relu, set alpha to 0.0
  for (int i = 0; i < m.batch_size() * m.item_size(); i++) {
    // values will be 0 if negative, and equal to themselves if positive
    if (avals.at(i) < 0)
      avals.at(i) *= alpha;
    //cout « i « ":" « avals.at(i) « endl;
  }
#ifdef DEBUG
  m.display("Apply RELU to");
  a.display("giving");
#endif
}

//codesnippet netsigmoid
//template <typename VectorBatch>
void sigmoid_io(const VectorBatch &m, VectorBatch &a) {

    const auto& mvals = m.vals_vector();
    auto& avals = a.vals_vector();
    avals.assign(mvals.begin(),mvals.end());
    // for (int i = 0; i < m.batch_size() * m.item_size(); i++) {
    //   avals[i] = 1 / (1 + exp(-avals[i]));
    // }
    for ( auto& e: avals ) {
      e = 1.f / ( 1.f + exp( -e ) );
      if (e<1.e-5) e = 1.e-5;
      if (e>1-1.e-5) e = 1-1.e-5;
    }
    if (trace_scalars()) {
      bool limit{true};
      float min{2.f},max{-1.f};
      for_each( avals.begin(),avals.end(),
        [&min,&max,&limit] (auto e) {
          assert( not isinf(e) ); assert( not isnan(e) );
          if (e<min) min = e; if (e>max) max = e;
          limit = limit && e>0 && e<1;
        }
        );
      cout « "sigmoid limited: " « boolalpha « limit
       « ": " « min « "--" « max « "\n";
      assert( limit );
    }
}
//codesnippet end

void softmax_io(const VectorBatch &m, VectorBatch &a) {

  const int vector_size = a.item_size(), batch_size = a.batch_size();
  assert( vector_size==m.item_size() );
  assert( batch_size==m.batch_size() );
  vector<float> nB(batch_size,0);
  vector<float> mVectorBatch(batch_size,-99);

  // compute software independently for each vector j
  for (int j = 0; j < batch_size; j++) {
    // copy a vector from m into temporary aj
    auto aj = a.extract_vector(j);
    const auto& mj = m.extract_vector(j);
    std::copy( mj.begin(),mj.end(),aj.begin() );
    // find the max
    mVectorBatch.at(j) = *max_element( aj.begin(),aj.end() );
    for_each( aj.begin(),aj.end(),
              [=] ( auto& x ) { x -= mVectorBatch.at(j); } );
    for_each( aj.begin(),aj.end(),
```

```
               [] ( auto& x ) { x = exp(x); } );
    nB.at(j) = accumulate( aj.begin(),aj.end(),0 );
    if ( nB.at(j)==0 ) throw("softmax aj is zero");
    for_each( aj.begin(),aj.end(),
               [=] ( auto& x ) { x /= nB.at(j); } );
    for_each( aj.begin(),aj.end(),
               [] ( auto& x ) {
            if (x <= 1e-7)
              x = 1e-7;
            if (x >= 1 - 1e-7)
              x = 1 - 1e-7;
               } );
    a.set_vector( aj,j );
  }

#ifdef DEBUG
  assert( a.positive() );
  m.display("Apply SoftMAX to");
  a.display("giving");
#endif
}

void softmax_io( const Vector& input, Vector& output ) {
    VectorBatch input_batch(input), output_batch(input_batch);
    softmax_io(input_batch,output_batch);
    output.copy_from( output_batch );
};

//template <typename VectorBatch>
void linear_io(const VectorBatch &m, VectorBatch &a) {
    a.vals_vector().assign(m.vals_vector().begin(),m.vals_vector().end());
}

//template <typename VectorBatch>
void reluGrad_io(const VectorBatch &m, VectorBatch &a) {
  assert( a.item_size()==m.item_size() );
        auto& avals = a.vals_vector();
  const auto& mvals = m.vals_vector();
  avals.assign(mvals.begin(),mvals.end());
  float alpha = 0.01;
  for ( auto &e : avals) {
    if (e<=0)
      e = alpha;
    else
      e = 1.0;
  }
  if (trace_progress()) {
    assert( a.normf()!=0.f );
    assert( a.notinf() );
    assert( a.notnan() );
  }
}

//template <typename VectorBatch>
void sigGrad_io(const VectorBatch &m, VectorBatch &a) {
    assert( m.size()==a.size() );

    const auto& mvals = m.vals_vector();
    auto& avals = a.vals_vector();

    avals.assign(mvals.begin(),mvals.end());
    for ( auto &e : avals )
      e = e * (1.0 - e);
    if (trace_scalars())
      cout « "sigmoid grad " « m.normf() « " => " « a.normf() « "\n";
}

//template <typename VectorBatch>
void smaxGrad_io(const VectorBatch &m, VectorBatch &a) {
  assert( m.size()==a.size() );
  throw("Unimplemented smaxGrad_io");
}

#include <limits>
void nan_io(const VectorBatch &mm, VectorBatch &a) {

  VectorBatch m(mm);
  assert( a.item_size()==m.item_size() );
  assert( a.batch_size()==m.batch_size() );
  auto& avals = a.vals_vector();
  for (int i = 0; i < a.batch_size() * a.item_size(); i++) {
     avals.at(i) = std::numeric_limits<float>::signaling_NaN();
  }
}

#ifdef USE_GSL
Matrix smaxGrad_vec( const gsl::span<float> &v)
```

```
#else
Matrix smaxGrad_vec( const std::vector<float> &v)
#endif
{
    Matrix im(v.size(),1,0); // Input but converted to a matrix

    for (int i=0; i<v.size(); i++){
      float *i_data = im.data();
      *( i_data +i ) // im.mat[i]
    = v[i];
    }

    Matrix dM = im;

    Matrix diag(dM.rowsize(),dM.rowsize(),0);

    for (int i=0,j=0; i<diag.rowsize()*diag.colsize(); i+=diag.rowsize()+1,j++) {
        // identity * dM
      float *d_data = diag.data(), *m_data = dM.data();
      *( d_data+i ) // diag.mat[i]
    = *( m_data+j ); //dM.mat[j];
    }

    // S_(i,j) dot S_(i,k)
    Matrix dMT = dM.transpose();

    Matrix S(dM.rowsize(),dMT.colsize(),0);
    dM.mmp(dMT, S);
    im = diag - S; // Jacobian
    return im;

}


//template <typename VectorBatch>
void linGrad_io(const VectorBatch &m, VectorBatch &a) {
    assert( m.size()==a.size() );
    std::fill(a.vals_vector().begin(), a.vals_vector().end(), 1.0); // gradient of a linear function
}

float id_pt( const float& x ) {
    return x;
}

float relu_pt( const float& x ) {
  float y;
  if (x < 0)
      y = 0.;
  else
      y = x;
  return y;
}

float relu_slope_pt( const float& x ) {
  const float alpha = 0.01;
  if (x < 0)
      return x * alpha;
  else
      return x;
}

float sigmoid_pt( const float& x ) {
    float y = 1.f / ( 1.f + exp( -x ) );
    if (y<1.e-5)
    return 1.e-5;
    else if (y>1-1.e-5)
    return 1-1.e-5;
    else
    return y;
}

void batch_activation
    ( function< float(const float&) > pointwise,const VectorBatch &i,VectorBatch &o,bool by_vector ) {

  assert( i.item_size()==o.item_size() );
  assert( i.batch_size()==o.batch_size() );
  const auto& ivals = i.vals_vector();
  auto& ovals = o.vals_vector();
  ovals.assign(ivals.begin(),ivals.end());
  if (by_vector) {
    throw("need to think about by-vector activation");
    for ( int v=0; v<o.batch_size(); v++ ) {
      auto vec = o.get_vector(v);
      for_each( vec.begin(),vec.end(), pointwise );
      o.set_vector(vec,v);
    }
  } else {
```

```
    for ( size_t i=0; i<o.item_size()*o.batch_size(); i++ ) {
      ovals[i] = pointwise( ivals[i] );
    }
    //    for_each( ovals.begin(),ovals.end(), pointwise );
  }
#ifdef DEBUG
  o.display("Apply pointwise function to");
  i.display("giving");
#endif

}

```

## 5.5 funcs.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#ifndef SRC_FUNCS_H
#define SRC_FUNCS_H

#include <functional>

#include "matrix.h"
#include "vector.h"
#include "vector2.h"

#ifdef USE_GSL
#include "gsl/gsl-lite.hpp"
#endif

//template <typename VectorBatch>
void relu_io    (const VectorBatch &i, VectorBatch &v);
//template <typename VectorBatch>
void sigmoid_io (const VectorBatch &i, VectorBatch &v);
//template <typename VectorBatch>
void softmax_io (const VectorBatch &i, VectorBatch &v);
void softmax_io( const Vector& input, Vector& output );
//template <typename VectorBatch>
void linear_io    (const VectorBatch &i, VectorBatch &v);
void nan_io     (const VectorBatch &i, VectorBatch &v);

//template <typename VectorBatch>
void reluGrad_io(const VectorBatch &m, VectorBatch &a);
//template <typename VectorBatch>
void sigGrad_io (const VectorBatch &m, VectorBatch &a);
//template <typename VectorBatch>
void smaxGrad_io(const VectorBatch &m, VectorBatch &a);
//template <typename VectorBatch>
void linGrad_io (const VectorBatch &m, VectorBatch &a);

#ifdef USE_GSL
Matrix smaxGrad_vec( const gsl::span<float> &v);
#else
Matrix smaxGrad_vec( const std::vector<float> &v);
#endif

enum acFunc{RELU,SIG,SMAX,NONE};

static inline std::vector< std::string > activation_names{
  "ReLU", "Sigmoid", "SoftMax", "Linear" };

template <typename V>
static inline std::vector< std::function< void(const V&, V&) > > apply_activation{
  [] ( const V &v, V &a ) { relu_io(v,a); },
    [] ( const V &v, V &a ) { sigmoid_io(v,a); },
    [] ( const V &v, V &a ) { softmax_io(v,a); },
    [] ( const V &v, V &a ) { linear_io(v,a); }
};

template <typename V>
static inline std::vector< std::function< void(const V&, V&) > > activate_gradient{
  [] (  const V &m, V &v ) { reluGrad_io(m,v); },
```

```
    [] (  const V &m,  V &v ) { sigGrad_io(m,v); },
    [] (  const V &m,  V &v ) { smaxGrad_io(m,v); },
    [] (  const V &m,  V &v ) { linGrad_io(m,v); }
};

enum lossfn{cce, mse}; // categorical cross entropy, mean squared error
inline static std::vector< std::function< float( const float& groundTruth, const float& result) > >
 lossFunctions{
  [] ( const float &gT, const float &result ) {
    assert(result>0.f);
    return gT * log(result); }, // Categorical Cross Entropy
  [] ( const float &gT, const float &result ) {
    return pow(gT - result, 2); }, // Mean Squared Error
};

inline static std::vector< std::function< VectorBatch( VectorBatch&, VectorBatch&) > > d_lossFunctions
  {
  [] ( VectorBatch &gT, VectorBatch &result ) -> VectorBatch {
    std::cout « "Ambiguous operator\n" ; throw(27);
    //return -gT / ( result/ static_cast<float>( result.batch_size() ) );
  },
  [] ( VectorBatch &gT, VectorBatch &result ) -> VectorBatch {
    std::cout « "Ambiguous operator\n" ; throw(27);
    //return -2 * ( gT-result)/ static_cast<float>( result.batch_size() );
  },
};

float id_pt( const float &x );
float relu_pt( const float &x );
float relu_slope_pt( const float &x );
float sigmoid_pt( const float &x );
void batch_activation
    ( std::function< float(const float&) > pointwise,
      const VectorBatch &i,VectorBatch &o,bool=false );

#endif //SRC_FUNCS_H
```

## 5.6 layer.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include "layer.h"
#include "trace.h"

#include <iostream>
using std::cout;
using std::endl;

Layer::Layer() {};
Layer::Layer(int insize,int outsize)
  : weights( Matrix(outsize,insize,1) ),
    dw      ( Matrix(outsize,insize, 0) ),
    //dW( Matrix(outsize,insize, 0) ),
    dw_velocity( Matrix(outsize,insize, 0) ),
    biases( Vector(outsize, 1 ) ),
    // biased_product( Vector(outsize, 0) ),
    activated( Vector(outsize, 0) ),
    d_activated( Vector(outsize, 0) ),
    delta( VectorBatch(outsize,1) ),
    // biased_productm( VectorBatch(outsize,insize,0) ),
    //    activated_batch( VectorBatch(outsize,1, 0) ),
    //    d_activated_batch ( VectorBatch(outsize,insize, 0) ),
    db( Vector(insize, 0) ),
    // delta_mean( Vector(insize, 0) ),
    dl( VectorBatch(insize, 1) ),
    db_velocity( Vector(insize, 0) ) {};

/*
 * Resize temporaries to reflect current batch size
 */
void Layer::allocate_batch_specific_temporaries(int batchsize) {
  const int insize = weights.colsize(), outsize = weights.rowsize();
```

```
  biased_batch.allocate( batchsize,outsize );
  input_batch.allocate( batchsize,insize );
  //  activated_batch.allocate( batchsize,outsize );
  d_activated_batch.allocate( batchsize,outsize );
  dl.allocate( batchsize, outsize );
  delta.allocate( batchsize,outsize );
};

void Layer::set_activation(acFunc f) {
  activation = f;
  apply_activation_batch  = apply_activation<VectorBatch>.at(f);
  activate_gradient_batch = activate_gradient<VectorBatch>.at(f);
};

Layer& Layer::set_activation
    (
     std::function< float(const float&) > activate_pt,
     std::function< float(const float&) > gradient_pt,
     std::string name
     ) {
    set_activation
    (
     [activate_pt] ( const VectorBatch& i,VectorBatch& o ) -> void {
         batch_activation( activate_pt,i,o ); },
     [gradient_pt] ( const VectorBatch& i,VectorBatch& o ) -> void {
         batch_activation( gradient_pt,i,o ); },
     name );
  return *this;
};

Layer& Layer::set_activation
    ( std::function< void(const VectorBatch&,VectorBatch&) > apply,
      std::function< void(const VectorBatch&,VectorBatch&) > activate,
      std::string name ) {
  activation = acFunc::RELU;
  apply_activation_batch  = apply;
  activate_gradient_batch = activate;
  return *this;
};

Layer& Layer::set_uniform_weights(float v) {
  for ( auto& e : weights.values() )
    e = v;
  return *this;
};

Layer& Layer::set_uniform_biases(float v) {
  for ( auto& e : biases.values() )
    e = v;
  return *this;
};

//codesnippet layerforward
void Layer::forward( const VectorBatch& input, VectorBatch& output) {
  assert( input.batch_size()==output.batch_size() );
  if (trace_progress()) {
    cout « "Forward layer " « layer_number
     « ": " « input_size() « "->" « output_size() « endl;
  }

  allocate_batch_specific_temporaries(input.batch_size());
  input_batch.copy_from(input);
  if (trace_progress()) {
    assert( input_batch.notnan() ); assert( input_batch.notinf() );
    assert( weights.notnan() ); assert( weights.notinf() );
  }
  input_batch.v2mp( weights, biased_batch );
  if (trace_progress()) {
    assert( biased_batch.notnan() ); assert( biased_batch.notinf() );
  }

  biased_batch.addh(biases); // Add the bias
  if (trace_progress()) {
    assert( biased_batch.notnan() ); assert( biased_batch.notinf() );
  }

  apply_activation_batch(biased_batch, output);
  //cout « "layer output: " « output.data()[0] « "\n";
  if (trace_progress()) {
    assert( output.notnan() ); assert( output.notinf() );
  }
  //return activated_batch;
}
//codesnippet end

const VectorBatch& Layer::intermediate() const { return biased_batch; };
```

```
void Layer::backward
    (const VectorBatch &prev_delta, const Matrix &W, const VectorBatch &prev_output) {

  // compute delta ell
  activate_gradient_batch(prev_output, d_activated_batch);
  prev_delta.v2mtp( W, dl );

   // delta  = Dl . sigma
  if (trace_progress())
    cout « "L-" « layer_number « " delta\n";
  delta.hadamard( d_activated_batch,dl ); // Derivative of the current layer

  // prev_output.outer2( delta, dw );
  // if (trace_scalars())
  //   cout « "L-" « layer_number « " dw: "
  //     « delta.normf() « "x" « prev_output.normf() « " => " « dw.normf() « "\n";

  update_dw(delta, prev_output);
  // weights.axpy( 1.,dw );
  // db = delta.meanh();
  // biases.add( db );
}

void Layer::update_dw( const VectorBatch &delta, const VectorBatch& prev_output) {
   prev_output.outer2( delta, dw );
   if (trace_scalars())
     cout « "L-" « layer_number « " dw: "
      « delta.normf() « "x" « prev_output.normf() « " => " « dw.normf() « "\n";

  // Delta W = delta here X activated prevous
  weights.axpy( 1.,dw );
  db = delta.meanh();
  biases.add( db );
}

void Layer::set_topdelta( const VectorBatch& gTruth,const VectorBatch& output ) {

    // top delta ell is different
   activate_gradient_batch(output, d_activated_batch);
   dl = output - gTruth;
   dl.scaleby( 1.f / gTruth.batch_size() );
   // delta  = Dl . sigma
   delta.hadamard( d_activated_batch,dl );
   if (trace_scalars())
     cout « "L-" « layer_number « " top delta: "
      « d_activated_batch.normf() « "x" « dl.normf() « " => " « delta.normf() « "\n";

   //  update_dw(delta, prev_output);
};
```

## 5.7 layer.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#ifndef SRC_LAYER_H
#define SRC_LAYER_H

#include <functional>

#include "vector.h"
//#include "matrix.h"
#include "funcs.h"
#include "vector2.h"

class Net; // forward definition for friending
class Layer {
  friend class Net;
public:
    /*
     * Construction
     */
```

```
    Layer();
    Layer(int insize, int outsize);
    Layer& set_uniform_weights(float);
    Layer& set_uniform_biases(float);

    /*
     * Forward stuff
     */
private: // but note that Net is a 'friend' class!
    Vector biases; // Biases which come before the layer
    acFunc activation; // Activation functions of the layer
    //Vector biased_product; // Values in the layer n after multiplying vals from n-1 and weights
    Matrix weights; // Weights which come before the layer
    Vector activated;
    VectorBatch input_batch, biased_batch; //,activated_batch;
public:
    auto& input() { return input_batch; };
    const auto& input() const { return input_batch; };

    /*
     * Backward stuff
     */
private:
    VectorBatch delta,wdelta,dl,Dscale;
    Vector d_activated; // for backpropagation
    //VectorBatch biased_productm;
    VectorBatch d_activated_batch;
    Matrix dw;         // cumulative dw
    Matrix dw_velocity; // For SGD with Momentum, RMSprop
    Vector db_velocity;
    Vector db;         // cumulative deltas

    //Vector delta_mean; // mean of the deltas used in batch training
public:

    /*
     * Stats
     */
private:
    int layer_number{-1};
public:
    Layer& set_number(int n) { layer_number = n; return *this; };

public:
    int input_size() const { return weights.colsize(); };
    int output_size() const { return weights.rowsize(); };
  //    void set_initial_deltas( const Matrix&, const Vector& );
    void set_recursive_deltas( Vector &, const Layer&,const Layer& );
    void set_topdelta( const VectorBatch&,const VectorBatch& );
    void allocate_batch_specific_temporaries(int batchsize);

    /*
     * Action
     */
    void forward( const VectorBatch&,VectorBatch& );
    const VectorBatch& intermediate() const;
    void backward(const VectorBatch &delta, const Matrix &W, const VectorBatch &prev);
    void backward_update( const VectorBatch&, const VectorBatch& ,bool=false );
    void update_dw(const VectorBatch &delta, const VectorBatch& prevValues);


    /*
     * Activation function
     */
private:
  std::function< void(const VectorBatch&,VectorBatch&) > apply_activation_batch{
    [] ( const VectorBatch &v, VectorBatch &a ) { nan_io(v,a); } };
  std::function< void(const VectorBatch&,VectorBatch&) > activate_gradient_batch{
    [] ( const VectorBatch &v, VectorBatch &a ) { nan_io(v,a); } };
public:
  std::string activation_name{"custom"};
  void set_activation(acFunc f);
  Layer& set_activation
      ( std::function< void(const VectorBatch&,VectorBatch&) > apply,
    std::function< void(const VectorBatch&,VectorBatch&) > activate,
    std::string name=std::string("custom")
    );
    Layer& set_activation
      (
       std::function< float(const float&) > activate_pt,
       std::function< float(const float&) > gradient_pt,
       std::string name=std::string("custom")
       );
};


#endif //SRC_LAYER_H
```

## 5.8 Make.inc

```
USE_BLIS=0
BLIS_INC_DIR=/Users/eijkhout/Installation/blis/installation-git/include
BLIS_LIB_DIR=/Users/eijkhout/Installation/blis/installation-git/lib

USE_GSL=0
GSL_INC_DIR=../gsl-lite/include

CXX = clang++ -g -std=c++17 -fopenmp
CXXOPTS = ${HOME}/Installation/cxxopts/installation

DEBUG = 0
```

## 5.9 matrix.cpp

```
/*****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include "matrix.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

using std::vector;

Matrix::Matrix() { // Default constructor
    r = 0;
    c = 0;
    mat.clear();
}

// void Matrix::flatten() { // Matrices are initialized in a coalesced flat 1D representation, so
 changing the dimension values are enough
//      r = r * c;
//      c = 1;
// }


float* Matrix::data() { return mat.data(); };
const float* Matrix::data() const { return mat.data(); };

Matrix &Matrix::operator=(const Matrix &m2) { // Overloading the = operator
    r = m2.r;
    c = m2.c;

    this->mat = m2.mat; // IM Since we're using vectors we can just use the assignment from that

    return *this;
}

Matrix Matrix::operator+(const Matrix &m2) const {
    Matrix out(m2.r, m2.c, 0);
    for (int i = 0; i < m2.r * m2.c; i++) {
        out.mat[i] = this->mat[i] + m2.mat[i];
    }
    return out;
}

Matrix Matrix::operator-(const Matrix &m2) {
    Matrix out(m2.r, m2.c, 0);
    for (int i = 0; i < m2.r * m2.c; i++) {
        out.mat[i] = this->mat[i] - m2.mat[i];
    }
    return out;
}



Matrix operator*(const float &c, const Matrix &m) {
    Matrix o = m;
    assert(o.mat.size()==o.r*o.c);
```

```
    for (int i = 0; i < o.r * o.c; i++) {
        o.mat[i] = c * o.mat[i];
    }
    return o;
}

Matrix operator/(const Matrix &m, const float &c) {
    Matrix o = m;
    for (int i = 0; i < o.r * o.c; i++) {
        o.mat[i] = o.mat[i] / c;
    }
    return o;
}


Matrix Matrix::operator*(const Matrix &m2) { // Hadamard product
    Matrix out(m2.r, m2.c, 0);
    assert(out.mat.size()==m2.r * m2.c);
    for (int i = 0; i < m2.r * m2.c; i++) {
        out.mat[i] = this->mat[i] * m2.mat[i];
    }
    return out;
}

Matrix Matrix::operator/(const Matrix &m2) { // Hadamard product
    Matrix out(m2.r, m2.c, 0);
    for (int i = 0; i < m2.r * m2.c; i++) {
        out.mat[i] = this->mat[i] / m2.mat[i];
    }
    return out;
}

Matrix Matrix::operator-() {
    Matrix result = *this;
    for (int i = 0; i < r * c; i++) {
        result.mat[i] = -mat[i];
    }

    return result;
};

void Matrix::addvh(const Vector &y) {
    for (int j = 0; j < c; j++) {
        for (int i = 0; i < y.size(); i++) {
            mat[j + c * i] += y.vals[i];
        }
    }
}

Vector Matrix::meanv() { // Returns a vector of column-wise means
    Vector mean(r, 0);
    float avg;
    for (int i = 0; i < r; i++) {
        avg = 0.0;
        for (int j = 0; j < c; j++) {
            avg += mat[i * c + j];
        }
        mean.vals[i] = avg;
    }
    return mean;
}


void Matrix::zeros() {
    std::fill(mat.begin(), mat.end(), 0);
}

void Matrix::square() {
    std::for_each(mat.begin(), mat.end(), [](auto &n) { n *= n;});
}
```

## 5.10 matrix.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
```

```
 *****************************************************************
 *****************************************************************/

#ifndef CODE_MAT_H
#define CODE_MAT_H

#include <vector>
#include "vector.h"
//#include "vector2.h"
#include <initializer_list>

class Matrix{
private: // should really become private
    std::vector<float> mat;
    int r;
    int c;
public:
    Matrix();
    Matrix(int nRows, int nCols, int rand);
    // for mpl
    std::vector<float> &values() { return mat; };
    const std::vector<float> &values() const { return mat; };
    float* data() ;
    const float* data() const;
    int nelements() const {
      return mat.size();
    };
    int rowsize() const { return r; };
    int colsize() const { return c; };
    Matrix transpose() const;
    void show() const;
    //void flatten();
    void mvpt( const Vector &x, Vector &y ) const;
    void mvp( const Vector &x, Vector &y ) const;
    void addvh( const Vector &y); // Add a vector to each column

    void mmp( const Matrix &x, Matrix &y) const;
    void outerProduct( const Vector &x, const Vector &y );
    Vector meanv();
    void zeros();
    void square();
    float normf() const;

    bool notnan() const {
      return all_of
    ( mat.begin(),mat.end(),
      [] (float e) { return not isnan(e); }
    );
    }
    bool notinf() const {
      return all_of
    ( mat.begin(),mat.end(),
      [] (float e) { return not isinf(e); }
    );
    }
    //vector2 methods
    // Moving these to VectorBatch object
    //void mv2p( const VectorBatch &x, VectorBatch &y) const;
    //void mv2pt( const VectorBatch &x, VectorBatch &y ) const;
    //void outer2( const VectorBatch &x, const VectorBatch &y );


    Matrix operator-(); // Unary negate operator
    Matrix& operator=(const Matrix& m2); // Copy constructor
    Matrix operator+(const Matrix &m2) const; // Element-wise addition
    Matrix operator*(const Matrix &m2); // Hadamard Product Element-wise multiplication
    Matrix operator/(const Matrix &m2); // Element-wise division
    Matrix operator-(const Matrix &m2); // Element-wise subtraction
    friend Matrix operator*(const float &c, const Matrix &m); // for constant-matrix multiplication
    friend Matrix operator/(const Matrix &m, const float &c); // for matrix-constant division
  void axpy( float a,const Matrix &x );

};



#endif
```

## 5.11 matrix_impl_blis.cpp

```
/*****************************************************************
 *****************************************************************
 ****
```

```
**** This text file is part of the source of
**** `Introduction to High-Performance Scientific Computing'
**** by Victor Eijkhout, copyright 2012-2021
****
**** Deep Learning Network code
**** copyright 2021 Ilknur Mustafazade
****
****************************************************************
****************************************************************/

#include "matrix.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

using std::vector;

#ifdef BLISNN
#include "blis/blis.h"
#endif

Matrix::Matrix(int nRows, int nCols, int random = 0)
        : r(nRows), c(nCols) {

    mat = vector<float>(nRows * nCols);
    float scal_fac = 0.05; // randomize between (-1;1)
    if (random==0){
        float zero = 0.0;
        bli_ssetm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
                r, c, &zero, &mat[0], c, 1);
    } else if (random==1){
        bli_srandm(0, BLIS_DENSE, r, c, &mat[0], c, 1);
        bli_sscalm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
                r, c, &scal_fac, &mat[0], c, 1);
    }
}

Matrix Matrix::transpose() const {
    Matrix result(c, r, 0); // Initialize a new matrix with inverted dimension values

    // m = r, n = c
    // rs = 1, cs = m, rsf = 1, csf = n
    //printf("BLIS copy %dx%d\n",c,r);
    bli_scopym( 0, BLIS_NONUNIT_DIAG, BLIS_DENSE, BLIS_TRANSPOSE,
        c, r, const_cast<float*>(&mat[0]), c, 1, &result.mat[0], r, 1);
    return result;
}

void Matrix::show() const {

    char e[5] = "";
    char format[8] = "%4.4f";
    bli_sprintm( e, r, c, const_cast<float*>(&mat[0]), c, 1, format, e );
}


void Matrix::mvp(const Vector &x, Vector &y) const {
    assert( c==x.size() );
    assert( r==y.size() );

    float alpha = 1.0;
    float beta = 0.0;
    //printf("BLIS gemv %dx%d\n",c,r);
    bli_sgemv( BLIS_NO_TRANSPOSE, BLIS_NO_CONJUGATE,
           r, c, &alpha, const_cast<float*>(&mat[0]),
           c, 1, const_cast<float*>( x.data() ) /* &t.vals[0] */,
           // c, 1, &t.vals[0] ,
           1, &beta, &y.vals[0], 1 );

}

void Matrix::mvpt(const Vector &x, Vector &y) const {
    assert( r==x.size() );
    assert( c==y.size() );

    float alpha = 1.0;
    float beta = 0.0;
    //printf("BLIS gemv %dx%d\n",c,r);
    bli_sgemv( BLIS_TRANSPOSE, BLIS_NO_CONJUGATE,
           r, c, &alpha, const_cast<float*>(&mat[0]), // test if r and c need to be flipped
           //c, 1, &t.vals[0],
           c, 1, const_cast<float*>( x.data() ),
           1, &beta, &y.vals[0], 1 );
}


```

```
void Matrix::outerProduct(const Vector &x, const Vector &y) {
    assert( x.size() == r );
    assert( y.size() == c );
    float val = 1.0;

    //printf("BLIS gemm (outer) %dx%d\n",r,c);
    bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_TRANSPOSE,
            r, c, 1, &val,
            const_cast<float*>(x.data()), 1, 1,
            const_cast<float*>(y.data()), c, 1, &val, &mat[0], c, 1);

}


void Matrix::mmp(const Matrix &x, Matrix &y) const { // In place matrix matrix multiplication
    assert( c==x.r );
    assert( r==y.r );
    assert( x.c==y.c );

    float alpha = 1.0;
    float beta = 0.0;
    // m = r, n = x.c, k = c
    //printf("BLIS gemm %dx%dx%d\n",r,x.c,c);
    bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_NO_TRANSPOSE,
           r, x.c, c, &alpha, const_cast<float*>(&mat[0]),
           //c, 1, &t.mat[0],
           c, 1, const_cast<float*>( x.data() ),
           x.c, 1, &beta, &y.mat[0], x.c, 1);
}

void Matrix::axpy( float a,const Matrix &x ) {
  assert( r==x.r );
  assert( c==x.c );
  const int n = nelements();
  assert( n==x.nelements() );

  bli_saxpyv( BLIS_NO_CONJUGATE,
          n, &a, const_cast<float*>( x.data() ),1,
          data(),1 );
};

float Matrix::normf() const {
  float norm;
  auto r = rowsize(), c = colsize();
  const auto mval = values().data();
  bli_snormfv( r*c, const_cast<float*>(mval), 1, &norm );
  return norm;
};
```

## 5.12 matrix_impl_reference.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include "matrix.h"
#include <iostream>
#include <vector>
#include <algorithm>
#include <cassert>

using std::vector;

Matrix::Matrix(int nRows, int nCols, int random = 0)
  : r(nRows), c(nCols) {

  mat = vector<float>(nRows * nCols);
  int i, j;
  if (random==0) {
    std::fill(mat.begin(), mat.end(), 0);
  } else if (random==1) {
    //std::fill(mat.begin(), mat.end(), .5);
    for (i=0; i<nRows * nCols;i++){
      //mat[i] = -0.1 + static_cast <float> (rand()) /( static_cast <float>(RAND_MAX/(0.1-(-0.1))));
```

```
            mat[i] = -0.1 + static_cast <float> (rand()) /( static_cast <float>(RAND_MAX) );
        }
    }

}

Matrix Matrix::transpose() const {
    Matrix result(c, r, 0); // Initialize a new matrix with inverted dimension values
    int i1, i2; // Old and new index
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            i1 = i * c + j; // Old indexing
            i2 = j * r + i; // New indexing

            result.mat[i2] = mat[i1]; // Move transposed values to new array
        }
    }

    return result;
}

void Matrix::show() const {

    int i, j;
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            std::cout « mat[i * c + j] « ' ';
        }
        std::cout « std::endl;
    }
    std::cout « std::endl;

}


void Matrix::mvp(const Vector &x, Vector &y) const {
    assert( c==x.size() );
    assert( r==y.size() );
    for (int i = 0; i < r; i++) {
        float sum = 0.0;
        for (int j = 0; j < c; j++) {
            sum += x.vals[j] * mat[i * c + j];
        }
        y.vals[i] = sum;
    }

}

void Matrix::mvpt(const Vector &x, Vector &y) const {
    assert( r==x.size() );
    assert( c==y.size() );
    for (int i = 0; i < r; i++) {
        float sum = 0.0;
        for (int j = 0; j < c; j++) {
            sum += x.vals[j] * mat[i * c + j];
        }
        y.vals[i] = sum;
    }

}


void Matrix::outerProduct(const Vector &x, const Vector &y) {
    assert( x.size() == r );
    assert( y.size() == c );
    float val = 1.0;
    for (int i = 0; i < x.size(); i++) {
        for (int j = 0; j < y.size(); j++) {
            mat[i * c + j] = x.vals[i] * y.vals[j];
        }
    }

}

void Matrix::mmp(const Matrix &x, Matrix &y) const {
    assert( c==x.r );
    assert( r==y.r );
    assert( x.c==y.c );
    float sum;
    for (int i = 0; i < r; i++) { // Matrix multiplication subroutine
        for (int j = 0; j < x.c; j++) {
            sum = 0.0;
            for (int k = 0; k < x.r; k++) {
                sum += mat[i * c + k] * x.mat[k * x.c + j];
            }
            y.mat[i * y.c + j] = sum;
        }
```

```
    }
}

void Matrix::axpy( float a,const Matrix &x ) {
  assert( r==x.r );
  assert( c==x.c );
  const int n = nelements();
  assert( n==x.nelements() );

  float *ydata = this->data();
  const auto& xdata = x.data();
  for (int i=0; i<n; i++) {
    ydata[i] += a * xdata[i];
  }

};

float Matrix::normf() const {
  float norm{0.f};
  // auto r = rowsize(), c = colsize();
  // const auto mval = values().data();
  for ( auto e : values() )
    norm += e*e;
  return sqrt(norm);
};
```

## 5.13 net.cpp

```
/*******************************************************************
 *******************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 *******************************************************************
 *******************************************************************/

#include <iostream>
using std::cout;
using std::endl;
#include <fstream>

#include <algorithm>
#include <string>
using std::string;

#include <cmath>

#include "vector.h"
#include "net.h"
#include "trace.h"

Net::Net(int s) { // Input vector size
    this->inR = s;
    this->inC = 1;
    samples = 0;
}

Net::Net( const Dataset &data ) {
  this->inR = data.data_size(); //data.items.at(0).data.size();
  this->inC = 1;
  samples = 0;
}

void Net::addLayer( int outputsize,
            std::function< float(const float&) > activate_pt,
            std::function< float(const float&) > gradient_pt,
            string name ) {
    addLayer
    ( outputsize,
      [activate_pt] ( const VectorBatch& i,VectorBatch& o ) -> void {
          batch_activation( activate_pt,i,o ); },
      [gradient_pt] ( const VectorBatch& i,VectorBatch& o ) -> void {
          batch_activation( gradient_pt,i,o ); },
      name );
};

void Net::addLayer( int outputsize,
            std::function< void(const VectorBatch&,VectorBatch&) > apply_activation_batch,
```

```
          std::function< void(const VectorBatch&,VectorBatch&) > activate_gradient_batch,
          string name
          ) {
    // Initialize layer object and add the necessary parameters
    Layer layer(inputsize(layers.size()), outputsize);
    layer.set_activation(apply_activation_batch,activate_gradient_batch);
    layer.set_number( layers.size() );
    layer.activation_name = name;
    push_layer(layer);
};

void Net::push_layer( const Layer& layer ) {
    if (trace_progress())
    cout « "Creating layer " « layers.size()
         « " of size " « layer.output_size() « "x" « layer.input_size()
        //       « " with " « name « " activation"
         « endl;
    layers.push_back(layer);
};

void Net::addLayer(int l, acFunc f) {
  addLayer( l,
        apply_activation<VectorBatch>.at(f),
        activate_gradient<VectorBatch>.at(f),
        activation_names.at(f)
        );
}

void Net::set_lossfunction( lossfn lossFuncName ) {
  lossFunction = lossFunctions.at(lossFuncName);
  d_lossFunction = d_lossFunctions.at(lossFuncName);
};

void Net::set_uniform_weights(float v) {
  for ( auto& l : layers )
    l.set_uniform_weights(v);
};

void Net::set_uniform_biases(float v) {
  for ( auto& l : layers )
    l.set_uniform_biases(v);
};

//codesnippet netforward
void Net::feedForward( const VectorBatch& input,VectorBatch& output ) {
  if (trace_progress())
    cout « "Feed forward batch of size " « input.batch_size() « endl;
  allocate_batch_specific_temporaries(input.batch_size());

  if (layers.size()==1) {
    layers.front().forward(input,output);
    //cout « "single layer output: " « output.data()[0] « "\n";
  } else {
    layers.front().forward( input, layers.at(1).input() );
    cout « " first layer : "
     « input.data()[0] « " -> " « layers.at(1).input().data()[0] « "\n";
    for (unsigned i = 1; i<layers.size()-1; i++) {
      layers.at(i).forward
    (layers.at(i).input(),
     layers.at(i+1).input()
     );
    }
    layers.back().forward(layers.back().input(),output);
    cout « "last layer : "
     « layers.back().input().data()[0] « " -> " « output.data()[0] « "\n";
  }
};
//codesnippet end

void Net::feedForward( const Vector& input, Vector& output ) {
    VectorBatch input_batch(input), output_batch(input_batch);
    feedForward(input_batch,output_batch);
    const auto& in_data = input_batch.data();
    const auto& out_data = output_batch.data();
    //    cout « "net forward: " « in_data[0] « " -> " « out_data[0] « "\n";
    output.copy_from( output_batch );
};

void Net::show() {
    for (unsigned i = 0; i < layers.size(); i++) {
        cout « "Layer " « i « " weights" « endl;
        layers.at(i).weights.show();
    }
}

void Net::calculate_initial_delta(VectorBatch &input, VectorBatch &gTruth) {
    VectorBatch d_loss = d_lossFunction( gTruth, input);
```

```

    if (layers.back().activation_name == "SoftMax" ) { // Softmax derivative function
        Matrix jacobian( input.item_size(), input.item_size(), 0 );
        for(int i = 0; i < input.batch_size(); i++ ) {
          auto one_column = input.get_vector(i);
          jacobian = smaxGrad_vec( one_column );
          Vector one_vector( jacobian.rowsize(), 0 );
          Vector one_grad = d_loss.get_vectorObj(i);
          jacobian.mvp( one_grad, one_vector );
          layers.back().d_activated_batch.set_vector(one_vector,i);
        }
    }
    /* Will add the rest of the code here, not done yet
    */
}

void Net::backPropagate
    (const VectorBatch &input, const VectorBatch& output, const VectorBatch &gTruth) {

  if (layers.size()==1) {
    throw(string("single layer case does not work"));
      // const VectorBatch& prev = input;
      // layers.back().update_dw(delta, prev);
      // return;
  } else {

    if (trace_progress()) cout « "Layer-" « layers.back().layer_number « "\n";
    layers.back().set_topdelta( gTruth,output );
    const VectorBatch& prev = layers.back().input();
    layers.back().update_dw(layers.back().delta, prev);


    for (unsigned i = layers.size() - 2; i > 0; i--) {
      if (trace_progress()) cout « "Layer-" « layers.at(i).layer_number « "\n";
      layers.at(i).backward
      ( layers.at(i+1).delta, layers.at(i+1).weights, layers.at(i).input());
    }

    if (trace_progress()) cout « "Layer-" « layers.at(0).layer_number « "\n";
    layers.at(0).backward(layers.at(1).delta, layers.at(1).weights, input);

  }
}

void Net::SGD(float lr, float momentum) {
    assert( layers.size()>0 );
    int samplesize = layers.front().input().batch_size();
    for (int i = 0; i < layers.size(); i++) {
        // Normalize gradients to avoid exploding gradients
    Matrix deltaW = layers.at(i).dw / static_cast<float>(samplesize);
        Vector deltaB = layers.at(i).db / static_cast<float>(samplesize);

        // Gradient descent
        if (momentum > 0.0) {
            layers.at(i).dw_velocity = momentum * layers.at(i).dw_velocity - lr * deltaW;
            //layers.at(i).weights = layers.at(i).weights + layers.at(i).dw_velocity;
        layers.at(i).weights.axpy( 1.f,layers.at(i).dw_velocity );
        } else {
      //layers.at(i).weights = layers.at(i).weights - lr * deltaW;
      layers.at(i).weights.axpy( -lr,deltaW );
        }

        layers.at(i).biases = layers.at(i).biases - lr * deltaB;

        // Reset the values of delta sums
        layers.at(i).dw.zeros();
        layers.at(i).db.zeros();
    }
}

void Net::RMSprop(float lr, float momentum) {
    for (int i = 0; i < layers.size(); i++) {
        // Get average over all the gradients
        Matrix deltaWsq = layers.at(i).dw;
        Vector deltaBsq = layers.at(i).db;

    // Gradient step
        deltaWsq.square(); // dW^2
        deltaBsq.square(); // db^2
        // Sdw := m*Sdw + (1-m) * dW^2

    layers.at(i).dw_velocity = momentum * layers.at(i).dw_velocity + (1 - momentum) * deltaWsq;
        layers.at(i).db_velocity = momentum * layers.at(i).db_velocity + (1 - momentum) * deltaBsq;

    Matrix sqrtSdw = layers.at(i).dw_velocity;
        std::for_each(sqrtSdw.values().begin(), sqrtSdw.values().end(),
              [](auto &n) {
```

```
            n = sqrt(n);
            if(n==0) n= 1-1e-7;
              });
    Vector sqrtSdb = layers.at(i).db_velocity;
        std::for_each(sqrtSdb.values().begin(), sqrtSdb.values().end(),
              [](auto &n) {
            n = sqrt(n);
            if(n==0) n= 1-1e-7;
              });

        // W := W - lr * dW / sqrt(Sdw)
        layers.at(i).weights = layers.at(i).weights - lr * layers.at(i).dw / sqrtSdw;
        layers.at(i).biases = layers.at(i).biases - lr * layers.at(i).db / sqrtSdb;

        // Reset the values of delta sums
        layers.at(i).dw.zeros();
        layers.at(i).db.zeros();
    }
}

// this function no longer used
// void Net::calcGrad(VectorBatch data, VectorBatch labels) {
//     feedForward(data);
//     backPropagate(data, labels);
// }


void Net::train( const Dataset &train_data,const Dataset &test_data,
         int epochs, int batchSize ) {

    const int Optimizer = optimizer();
    cout « "Optimizing with ";
    switch (Optimizer) {
    case sgd:  cout « "Stochastic Gradient Descent\n";  break;
    case rms:  cout « "RMSprop\n"; break;
    }

    std::vector<Dataset> batches = train_data.batch(batchSize);
    float lrInit = learning_rate();
    const float momentum_value = momentum();

    for (int i_epoch = 0; i_epoch < epochs; i_epoch++) {
      // Iterate through the entire dataset for each epoch
      cout « endl « "Epoch " « i_epoch+1 « "/" « epochs « endl;
      float current_learning_rate = lrInit; // Reset the learning rate to undo decay

      for (int j = 0; j < batches.size(); j++) {
    // Iterate through all batches within dataset
    auto& batch = batches.at(j);
#ifdef DEBUG
    cout « ".. batch " « j « "/" « batches.size() « " of size " « batch.size() « "\n";
#endif
    //  allocate_batch_specific_temporaries(batch.size());
    VectorBatch batch_output( batch.inputs().item_size(),batch.size() );
    feedForward(batch.inputs(),batch_output);
    backPropagate(batch.inputs(),batch.labels(),batch_output);

        // User chosen optimizer
        current_learning_rate = current_learning_rate / (1 + decay() * j);
        optimize.at(Optimizer)(current_learning_rate, momentum_value);

      }
      auto loss = calculateLoss(test_data);
      cout « " Loss: " « loss « endl;
      auto acc = accuracy(test_data);
      cout « " Accuracy on trest set: " « acc « endl;
    }

}

void Net::allocate_batch_specific_temporaries(int batchsize) {
#ifdef DEBUG
  cout « "allocating temporaries for batch size " « batchsize « endl;
#endif
  for ( auto& layer : layers )
    layer.allocate_batch_specific_temporaries(batchsize);
}

//codesnippet netloss
float Net::calculateLoss(const Dataset &testSplit) {

#ifdef DEBUG
  cout « "Loss calculation\n";
#endif
  //  allocate_batch_specific_temporaries(testSplit.inputs().batch_size());
  VectorBatch result = this->create_output_batch( testSplit.inputs().batch_size() );
  feedForward( testSplit.inputs(),result );
```

```
  assert( result.notnan() );

    float loss = 0.0;
    auto tmp_labels =  testSplit.labels();
    if (trace_arrays()) {
      cout « "Compare results\n"; result.show();
      cout « " to label\n"; tmp_labels.show();
    }
    for (int vec=0; vec<result.batch_size(); vec++) { // iterate over all items
      const auto& one_result = result.extract_vector(vec); // VLE figure out const span !!!
      auto one_label  = tmp_labels.get_vector(vec);
      assert( one_result.size()==one_label.size() );
      for (int i=0; i<one_result.size(); i++) { // Calculate loss of result
    auto this_label = one_label[i], this_result = one_result[i];
    assert( not std::isnan(this_label) );
    assert( not std::isnan(this_result) );
    auto oneloss = lossFunction( this_label, this_result );
    assert( not std::isnan(oneloss) );
    loss += oneloss;
      }
    }
    const int bs = result.batch_size();
    assert( bs>0 );
    auto scale = 1.f / static_cast<float>(bs);
    loss = loss * scale;

    return loss;
}
//codesnippet end


#if 0
    const auto& result_vals = result.vals_vector();
    const auto& label_vals  = testSplit.labelBatch.vals_vector();
    for (int j = 0; j < result.r * result.c; j++) {
      loss += lossFunction( label_vals[j], result_vals[j] );
    }
#else
#endif

float Net::accuracy( const Dataset &test_set ) {
  if (trace_progress())
    cout « "Accuracy calculation\n";

  int correct = 0;
  int incorrect = 0;

  assert( test_set.size()>0 );
  const auto& test_inputs = test_set.inputs();
  const auto& test_labels = test_set.labels();
  assert( test_inputs.batch_size()==test_labels.batch_size() );
  assert( test_inputs.batch_size()>0 );

  //      allocate_batch_specific_temporaries(test_inputs.batch_size());
  if (trace_arrays()) {
      cout « "inputs:\n"; test_inputs.show();
  }
  assert( test_inputs.normf()!=0.f );
  auto output_batch = this->create_output_batch(test_labels.batch_size());
  feedForward(test_inputs,output_batch);
  if (trace_arrays()) {
      cout « "outputs:\n"; output_batch.show();
  }
  assert( output_batch.notnan() );

  for(int idx=0; idx < output_batch.batch_size(); idx++ ) {
      Vector oneItem = output_batch.get_vectorObj(idx);
      Categorization result( oneItem );
      result.normalize();
      if ( result.close_enough( test_labels.extract_vector(idx) ) ) {
      correct++;
      } else {
      incorrect++;
      }
  }
  assert( correct+incorrect==test_set.size() );

  float acc = static_cast<float>( correct ) / static_cast<float>( test_set.size() );
  return acc;
}



void Net::saveModel(std::string path){
    /*
    1. Size of matrix m x n, activation function
    2. Values of weight matrix
```

```
    3. Values of bias vector
    4. Repeat for all layers
    */
    std::ofstream file;
    file.open( path, std::ios::binary );

    float temp;
    int no_layers = layers.size();
    file.write( reinterpret_cast<char *>(&no_layers), sizeof(no_layers) );
    for ( auto l : layers ) {
        int insize = l.input_size(), outsize = l.output_size();
        file.write( reinterpret_cast<char *>(&outsize), sizeof(int) );
        file.write( reinterpret_cast<char *>(&insize), sizeof(int) );
        file.write( reinterpret_cast<char *>(&l.activation), sizeof(int) );

        const auto& weights = l.weights;
        file.write(reinterpret_cast<const char *>(weights.data()), sizeof(float)*weights.nelements());
        // const float* weights_data = l.weights.data();
        // file.write(reinterpret_cast<char *>(&weights_data), sizeof(temp)*insize*outsize);

        const auto& biases = l.biases;
        file.write(reinterpret_cast<const char*>(biases.data()), sizeof(float) * biases.size());
        //file.write(reinterpret_cast<char *>(&l.biases.vals[0]), sizeof(temp) * l.biases.size());
    }
    cout « endl;

    file.close();
}

void Net::loadModel(std::string path){
    std::ifstream file(path);
    std::string buffer;

    int no_layers;
    file.read( reinterpret_cast<char *>(&no_layers), sizeof(no_layers) );

    float temp;
    layers.resize(no_layers);
    for ( int i=0; i < layers.size(); i++ ) {
        int insize,outsize;
        file.read( reinterpret_cast<char *>(&outsize), sizeof(int) );
        file.read( reinterpret_cast<char *>(&insize), sizeof(int) );

        file.read( reinterpret_cast<char *>(&layers[i].activation), sizeof(int) );

        layers[i].weights = Matrix( outsize, insize, 0 );
        float *w_data = layers[i].weights.data();
        file.read(reinterpret_cast<char *>( w_data ), //(&layers[i].weights.mat[0]),
              sizeof(temp) * insize*outsize);

        layers[i].biases = Vector( outsize, 0 );
        float *b_data = layers[i].biases.data();
        file.read(reinterpret_cast<char *>( b_data ), //(&layers[i].biases.vals[0]),
              sizeof(temp) * layers[i].biases.size());
    }
}


void Net::info() {
    cout « "Model info\n---------------\n";

    for ( auto l : layers ) {
        cout « "Weights: " « l.output_size() « " x " « l.input_size() « "\n";
        cout « "Biases: " « l.biases.size() « "\n";
        cout « "Activation: " « l.activation_name « "\n";

        // switch (l.activation) {
        //  case RELU: cout « "RELU\n"; break;
        //  case SIG: cout « "Sigmoid\n"; break;
        //  case SMAX: cout « "Softmax\n"; break;
        //  case NONE: break;
        // }
        cout « "---------------\n";
    }

}

void loadingBar(int currBatch, int batchNo, float acc, float loss) {
    cout  « "[";
    int pos = 50 * currBatch/(batchNo-1);
    for (int k=0; k < 50; ++k) {
    if (k < pos) cout « "=";
    else if (k == pos) cout « ">";
    else cout « " ";
    }
    cout « "] " « int(float(currBatch)/float(batchNo-1)*100) « "% " « "loss: " « loss « " acc: " « acc
 « " \r";
```

```
    cout « std::flush;
}
```

## 5.14 net.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#ifndef CODE_NET_H
#define CODE_NET_H

#include <vector>
#include "vector.h"
#include "matrix.h"
#include "dataset.h"
#include "layer.h"
#include <cmath>

enum opt{sgd, rms}; // Gradient descent, RMSprop

class Net {
private:
    int inR; // input dimensions
    int inC;
    int samples;
    std::vector<Layer> layers;
    std::function<float( const float& groundTruth, const float& result)> lossFunction{
      [] ( const float& groundTruth, const float& result) -> float {
    throw(std::string("no loss function defined")); } };
    std::function<VectorBatch( VectorBatch& groundTruth, VectorBatch& result )> d_lossFunction{
      [] ( VectorBatch& groundTruth, VectorBatch& result ) -> VectorBatch {
    throw(std::string("no d_lossfunction defined")); } };
public:
    Net(int s); // input shape
    Net( const Dataset &d );
    void addLayer(int l, acFunc activation); // length of the dense layer
    void addLayer( int l,
           std::function< float(const float&) > activate_pt,
           std::function< float(const float&) > gradient_pt,
           std::string name=std::string("custom")
           );
    void addLayer( int l,
           std::function< void(const VectorBatch&,VectorBatch&) > apply_activation_batch,
           std::function< void(const VectorBatch&,VectorBatch&) > activate_gradient_batch,
           std::string name=std::string("custom")
           );
    void push_layer( const Layer& layer );
    /*
     * Stats
     */
    int outputsize() const {
    assert( layers.size()>0 );
    return layers.back().output_size();
    };
    int inputsize( int layer ) const {
    assert(layer>=0); assert(layer<=layers.size());
    if (layers.empty())
        return inR; // Input's row size for the first layer
    else
        return layers.at(layer-1).output_size(); // Previous layer's row size
    };
    VectorBatch create_output_batch( int batchsize ) {
    return VectorBatch( outputsize(),batchsize );
    };
    /*
     * Direct layer access
     */
    Layer& backlayer() {
    assert(layers.size()>0); return layers.back(); };
    const Layer& backlayer() const {
    assert(layers.size()>0); return layers.back(); };
    const Layer& layer(int i) const { return layers.at(i); };
    const Layer& at(int i) const { return layers.at(i); };
```

```
    Layer& at(int i) { return layers.at(i); };

    void show(); // Show all weights
    Categorization output_vector() const;
    const VectorBatch &outputs() const;
    void set_lossfunction( lossfn lossFuncName );
    void set_uniform_weights(float);
    void set_uniform_biases(float);

    //void feedForward( const Vector& );
    void feedForward( const Vector&,Vector& );
    //void feedForward( const VectorBatch& );
    void feedForward( const VectorBatch&,VectorBatch& );

    void allocate_batch_specific_temporaries(int batchsize);
    void calcGrad(Dataset data);
    void calcGrad(VectorBatch data, VectorBatch labels);

    void backPropagate(const Vector &input, const Vector &gTruth, const Vector &output);
    void backPropagate(const VectorBatch &input, const VectorBatch &gTruth, const VectorBatch &output);

    void calculate_initial_delta( VectorBatch& result, VectorBatch& gTruth);

    void SGD(float lr, float momentum);
    void RMSprop(float lr, float momentum);

  /*
   * Various settings
   */
private:
  float _lr{0.05};
public:
  void set_learning_rate(float lr) { _lr=lr; };
  float learning_rate() const { return _lr; };
private:
  float _decay{0.05}; // high decay causes very small training
public:
  void set_decay(float d) { _decay = d; };
  float decay() const { return _decay; };
private:
  float _momentum{0.0}; // 0.9 works well
public:
  void set_momentum(float m) { _momentum = m; };
  float momentum() const { return _momentum; };
private:
  int _optimizer;
public:
  void set_optimizer(int m) { _optimizer = m; };
  int optimizer() const { return _optimizer; };
  std::vector< std::function< void(float lr, float momentum) > > optimize{
    [this] ( float lr, float momentum ) { SGD(lr, momentum); },
    [this] ( float lr, float momentum ) { RMSprop(lr, momentum); }
  };

  void train( const Dataset& train,const Dataset& test, int epochs, int batchSize);
#if MPINN
    void trainmpi(Dataset &trainData, Dataset &testData, float lr, int epochs, opt Optimizer, lossfn lossFunc, int batchSize, float momentum = 0.0, float decay = 0.0);
#endif
    float calculateLoss(const Dataset &testSplit);
    float accuracy( const Dataset& valSet );


    void saveModel(std::string path);
    void loadModel(std::string path);

    void info();
};

void loadingBar(int currBatch, int batchNo, float acc, float loss);

#endif //CODE_NET_H
```

## 5.15 net_mpi.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
```

```
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 *****************************************************************
 *****************************************************************/

#include <mpl/mpl.hpp>

void Net::trainmpi(Dataset &data, Dataset &testData, float lr, int epochs, opt Optimizer, lossfn lossFuncName, int batchSize, float momentum, float decay) {
    const mpl::communicator &comm_world(mpl::environment::comm_world());
    lossFunction = lossFunctions.at(lossFuncName);
    int rank = comm_world.rank();
    int size = comm_world.size();
    if(rank==0) {
    std::cout « "Optimizing with ";
      switch (Optimizer) {
      case sgd:  std::cout « "Stochastic Gradient Descent\n";  break;
      case rms:  std::cout « "RMSprop\n"; break;
      }
    }
    int ssize = batchSize;//data.items.size();
    std::vector<Dataset> batches = data.batch(ssize);
    for (int i = 0; i < batches.size(); i++) {
        batches.at(i).stack(); // Put batch items into one matrix
    }

    float loss, acc;
    float lrInit = lr;

    for (int i_epoch = 0; i_epoch < epochs; i_epoch++) {
      // Iterate through the entire dataset for each epoch
      if(rank==0)
        std::cout « std::endl « "Epoch " « i_epoch+1 « "/" « epochs;
      lr = lrInit; // Reset the learning rate to undo decay
      int batchStart = batches.size() * rank / comm_world.size();
      int batchEnd = batches.size() * (rank + 1) / comm_world.size();

      for(int idx=batchStart; idx<batchEnd; idx++) {
        calcGrad(batches.at(idx).dataBatch, batches.at(idx).labelBatch);
        for(auto &layer : this->layers) {
          mpl::contiguous_layout<float> dw_layout(layer.dw.r * layer.dw.c);
          mpl::contiguous_layout<float> db_layout(layer.db.size());
          if(rank==0) {
            Matrix tempdw(layer.dw.r, layer.dw.c, 0);
            Vector tempdb(layer.db.size(), 0);
            Matrix currdw = layer.dw;
            Vector currdb = layer.db;
            comm_world.reduce(mpl::plus<float>(), 0, tempdw.data(), layer.dw.data(), dw_layout);
            comm_world.reduce(mpl::plus<float>(), 0, tempdb.data(), layer.db.data(), db_layout);
            layer.dw = layer.dw + currdw; // IM the on-rank values get discarded when reducing, or may be a misinterpretation on my side
            layer.db = layer.db + currdb;
            layer.dw = layer.dw / size;
            layer.db = layer.db / size;
          } else {
            comm_world.reduce(mpl::plus<float>(), 0, layer.dw.data(), dw_layout);
            comm_world.reduce(mpl::plus<float>(), 0, layer.db.data(), db_layout);
          }
        }
        comm_world.barrier(); //sync before processing the next batch
        if(rank==0) {
            lr = lr / (1 + decay * idx);
            optimize.at(Optimizer)(lr, momentum);
        }
      }
    }
}
```

## 5.16 test_linear.cpp

```
/*****************************************************************
 *****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 *****************************************************************
 *****************************************************************/
```

```

#include <iostream>
#include <chrono>
#include <vector>
#include <time.h>

#include "cxxopts.hpp"

#include "net.h"
#include "dataset.h"
#include "vector.h"
#include "funcs.h"
#include "trace.h"

using namespace std;

int main(int argc,char **argv){
  //    srand(time(NULL));

  try {
    using myclock = std::chrono::high_resolution_clock;

    // IM attempt to use cxxopts to specify location
    cxxopts::Options options("EduDL", "FFNNs w BLIS");

    options.add_options()
      ("h,help","usage information")
      ("l,levels", "Number of levels in the network",cxxopts::value<int>()->default_value("2"))
      ("s,sizes","Sizes of the levels",cxxopts::value<std::vector<int»())
      ("o,optimizer", "Optimizer to be used, 0: SGD, 1: RMSprop",cxxopts::value<int>()->default_value("0"))
      ("e,epochs", "Number of epochs to train the network", cxxopts::value<int>()->default_value("1"))
      ("r,learningrate", "Learning rate for the optimizer", cxxopts::value<float>()->default_value("0.001"))
      ("b,batchsize", "Batch size for the training data", cxxopts::value<int>()->default_value("5"))
      ("t,tracing","Level of tracing: 0=default 1=scalars 2=arrays",cxxopts::value<int>()->default_value("0"))
      ;

    auto result = options.parse(argc,argv);
    if (result.count("help")) {
      std::cout « options.help() « std::endl;
      return 1;
    }

    std::vector<int> level_sizes{12,12};

    if (result.count("sizes")) {
      level_sizes = result["s"].as< std::vector<int> >();
    } else if (result.count("levels")) {
      int number_of_levels = result["l"].as<int>();
      level_sizes = std::vector<int>(number_of_levels,12);
    }

    set_trace_level( result["t"].as<int>() );
    int network_optimizer = result["o"].as<int>();
    int epochs = epochs = result["e"].as<int>();
    float lr = result["r"].as<float>();
    int batchSize = result["b"].as<int>();

    /*
     * Input data set:
     * y = 2 x + 1
     */
    cout « "Creating data set\n";
    Dataset data(1);
    for (int input=0; input<5; input++) {
      float x = 1.f + 1.f*input, y = 2*x+1;
      std::vector<float> i(1,x), o(1,y);
      dataItem thisitem{i,o};
      data.push_back(thisitem);
    }
    cout  « "Number of data points: " « data.size() « endl; // Show size
    //    data.stack();

    {
      /*
       * First test a perfect net
       */
      Net test_net( data );
      test_net.addLayer(1,NONE);
      test_net.set_uniform_weights(1.f);
      test_net.set_uniform_biases(0.f);
      test_net.set_lossfunction(mse);

      auto loss = test_net.calculateLoss(data);
      cout « " Loss: " « loss « endl;
```

```
      auto acc = test_net.accuracy(data);
      std::cout « "Accuracy: " « acc « "\n";
      test_net.info();

      auto [train_data,test_data] = data.split(0.9);
      assert( train_data.data_size()!=0 );
      assert( test_data.data_size()!=0 );
      cout « "Initial accuracy: " « test_net.accuracy(test_data) « "\n";
      test_net.train(train_data,test_data, epochs, batchSize);
      cout « "Final Accuracy over test data: " « acc « "\n";

    }

  } catch ( string e ) {
    cout « "Error «" « e « "» \n";
  } catch ( std::out_of_range ) {
    cout « "Out of range error\n";
  } catch ( ... ) {
    cout « "Uncaught exception\n";
  }

    return 0;
}
```

## 5.17 test_mnist.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

/*
 * A simple test neural network
 *
 * Test data set: http://cis.jhu.edu/~sachin/digit/digit.html
 */

#include <iostream>
using std::cout;
using std::endl;
#include <chrono>
#include <string>
using std::string;
#include <vector>
using std::vector;
#include <time.h>

#include "cxxopts.hpp"

#include "net.h"
#include "dataset.h"
#include "vector.h"
#include "trace.h"

using namespace std;
static int trace_level;

#if 1
int main(int argc,char **argv){
  try {
    using myclock = std::chrono::high_resolution_clock;

    // IM attempt to use cxxopts to specify location
    cxxopts::Options options("EduDL", "FFNNs w BLIS");

    options.add_options()
      ("h,help","usage information")
      ("d,dir", "Dataset directory",cxxopts::value<std::string>())
      ("l,levels", "Number of levels in the network",cxxopts::value<int>()->default_value("2"))
      ("s,sizes","Sizes of the levels",cxxopts::value<std::vector<int»())
      ("o,optimizer", "Optimizer to be used, 0: SGD, 1:
 RMSprop",cxxopts::value<int>()->default_value("0"))
      ("e,epochs", "Number of epochs to train the network", cxxopts::value<int>()->default_value("1"))
      ("r,learningrate", "Learning rate for the optimizer",
 cxxopts::value<float>()->default_value("0.001"))
```

```
      ("b,batchsize", "Batch size for the training data", cxxopts::value<int>()->default_value("256"))
      ("t,tracing","Level of tracing: 0=default 1=scalars
2=arrays",cxxopts::value<int>()->default_value("0"))
      ;

    auto result = options.parse(argc,argv);
    if (result.count("help")) {
      cout « options.help() « endl;
      return 1;
    }

    std::vector<int> level_sizes{12,12};

    if (result.count("sizes")) {
      level_sizes = result["s"].as< std::vector<int> >();
      if (result.count("levels")) {
    cout « "Option for number of levels ignored when level sizes are given" « endl;
      }
    } else if (result.count("levels")) {
      int number_of_levels = result["l"].as<int>();
      level_sizes = std::vector<int>(number_of_levels,12);
    }

    set_trace_level( result["t"].as<int>() );
    int network_optimizer = result["o"].as<int>();
    int epochs = epochs = result["e"].as<int>();
    float lr = result["r"].as<float>();
    int batchSize = result["b"].as<int>();

    /*
     * Input data set handling
     */
    if (!result.count("dir")) {
      cout « "Must specify directory with -d/--dir option" « endl;
      return 1;
    }
    string mnist_loc = result["dir"].as<string>();
    Dataset data;
    data.readTest(mnist_loc.data()); // Placed MNIST in a neighbor directory

    // Parent
    //   |__mnist
    //   |      |_data0
    //   |      |_...
    //   |      |_data9
    //   |__src
    //          |_test.cpp  [You are here]


    cout « "Dataset size: " « data.size() « endl; // Show size

    Net test_net(data);
    if (level_sizes.size()==2) {
      test_net.addLayer(16, RELU);
      test_net.addLayer(10, SIG ); //SMAX);
    } else {
      for ( auto level_size : level_sizes ) {
    cout « "Adding level of size " « level_size « endl;
    test_net.addLayer(level_size,RELU);
      }
      test_net.addLayer(10, SIG ); //SMAX);
    }

    test_net.set_learning_rate(lr);
    test_net.set_decay(0.0);
    test_net.set_momentum(0.9);
    test_net.set_optimizer(network_optimizer);
      test_net.set_uniform_weights(.5f);
      test_net.set_uniform_biases(.1f);
      test_net.set_lossfunction(mse);

    /*
     * Train / Test
     */
    auto start_time = myclock::now();
    auto [train_data,test_data] = data.split(0.8);
    cout « "Initial accuracy: " « test_net.accuracy(test_data) « "\n";

    test_net.train(train_data,test_data, epochs, batchSize);
    auto duration = myclock::now()-start_time;

    int
      microsec_duration = std::chrono::duration_cast<std::chrono::microseconds>(duration).count(),
      seconds = microsec_duration/1000000,
      micros  = microsec_duration - 1000000*seconds;
    while (micros>=1000) micros /= 10;
    auto acc = test_net.accuracy(test_data);
```

```
    cout « "Final Accuracy over test data: " « acc « "\n"
     « "    attained in " « seconds « "." « micros « " sec" « "\n";

    test_net.saveModel("weights.bin");
    test_net.info();

  } catch ( string e ) {
    cout « "ERROR «" « e « "»\n";
  } catch ( std::out_of_range ) {
    cout « "Uncaught out of range error\n";
  } catch ( ... ) {
    cout « "Uncaught exception\n";
  }

    return 0;
}
#else
int main(){

    VectorBatch a(3,4,1); // Two feature vectors, each vector sized 4
    VectorBatch b(3,5,1);
    a.show();

    Net model(4);
    model.addLayer(2,RELU);
    model.addLayer(5,SMAX);

    model.show();
    model.feedForward(a);
    model.backPropagate(a, b);

    return 0;
}
#endif
```

## 5.18 test_mpi.cpp

```
/******************************************************************
 ******************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ******************************************************************
 ******************************************************************/

#include <iostream>
#include <chrono>
#include <vector>
#include <time.h>

#include "cxxopts.hpp"

#include "net.h"
#include "dataset.h"

#include <mpl/mpl.hpp>
using namespace std;

#if 0
int main(){
    srand(time(NULL));

    Net bar(0);
    bar.loadModel("weights.bin");
    bar.info();

    return 0;
}
#else
int main(int argc,char **argv){
    using myclock = std::chrono::high_resolution_clock;
    srand(time(NULL));

    const mpl::communicator &comm_world(mpl::environment::comm_world());
    // IM attempt to use cxxopts to specify location
    cxxopts::Options options("EduDL", "FFNNs w BLIS");

    options.add_options()
```

```
      ("h,help","usage information")
      ("d,dir", "Dataset directory",cxxopts::value<std::string>())
      ("l,levels", "Number of levels in the network",cxxopts::value<int>()->default_value("2"))
      ("s,sizes","Sizes of the levels",cxxopts::value<std::vector<int»())
      ("o,optimizer", "Optimizer to be used, 0: SGD, 1:
   RMSprop",cxxopts::value<int>()->default_value("0"))
      ("e,epochs", "Number of epochs to train the network", cxxopts::value<int>()->default_value("1"))
      ("r,learningrate", "Learning rate for the optimizer",
   cxxopts::value<float>()->default_value("0.001"))
      ("b,batchsize", "Batch size for the training data", cxxopts::value<int>()->default_value("256"))
      ;

    auto result = options.parse(argc,argv);
    if (result.count("help")) {
      std::cout « options.help() « std::endl;
      return 1;
    }

    std::vector<int> level_sizes{12,12};

    if (result.count("sizes")) {
      level_sizes = result["s"].as< std::vector<int> >();
    } else if (result.count("levels")) {
      int number_of_levels = result["l"].as<int>();
      level_sizes = std::vector<int>(number_of_levels,12);
    }

    int network_optimizer = result["o"].as<int>();

    int epochs = epochs = result["e"].as<int>();

    float lr = result["r"].as<float>();

    int batchSize = result["b"].as<int>();

    /*
     * Input data set handling
     */
    if (!result.count("dir")) {
      std::cout « "Must specify directory with -d/--dir option" « std::endl;
      return 1;
    }
    string mnist_loc = result["dir"].as<string>();
    //std::cout « mnist_loc « std::endl;
    Dataset data;
    data.readTest(mnist_loc.data()); // Placed MNIST in a neighbor directory

    // Parent
    //    |__mnist
    //    |       |_data0
    //    |       |_...
    //    |       |_data9
    //    |__src
    //            |_test.cpp  [You are here]

    if(comm_world.rank()==0)
      cout « data.items.size() « endl; // Show size

    data.shuffle();

    //    Vector v1 = data.items.at(0).data;
    //    Vector gT = data.items.at(0).label;

    Net test_net(data); //v1.size());
    //test_net.addLayer(256, RELU);
    //test_net.addLayer(64, RELU);
    //test_net.addLayer(32, RELU);
    if (level_sizes.size()==2) {
      test_net.addLayer(16, RELU);
      test_net.addLayer(10, SMAX);
    } else {
      for ( auto level_size : level_sizes ) {
    std::cout « "Adding level of size " « level_size « std::endl;
    test_net.addLayer(level_size,RELU);
      }
      test_net.addLayer(10, SMAX);
    }

    auto [trainSplit,testSplit] = data.split(0.95);
    if(comm_world.rank()==0)
      std::cout « "Split " « trainSplit.items.size() « " " « testSplit.items.size() « "\n";
    auto start_time = myclock::now();
//    std::cout « test_net.accuracy(data) « "\n";

    test_net.trainmpi(trainSplit,testSplit,
    lr, epochs, (opt)network_optimizer, cce,  batchSize, 0.9);

```

```
    comm_world.barrier();
    auto duration = myclock::now()-start_time;
    auto microsec_duration = std::chrono::duration_cast<std::chrono::microseconds>(duration);

    if(comm_world.rank()==0){
      std::cout « "\nFinal Accuracy over all data: " « test_net.accuracy(data) « "\n"
          « "    attained in " « microsec_duration.count() « "usec" « "\n";

      test_net.saveModel("weights.bin");
      test_net.info();
    }
    return 0;
}
#endif
```

## 5.19 test_posneg.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include <iostream>
#include <chrono>
#include <vector>
#include <time.h>

#include "cxxopts.hpp"

#include "net.h"
#include "dataset.h"
#include "vector.h"
#include "funcs.h"
#include "trace.h"

using namespace std;

int main(int argc,char **argv){

  try {
    using myclock = std::chrono::high_resolution_clock;

    // IM attempt to use cxxopts to specify location
    cxxopts::Options options("EduDL", "FFNNs w BLIS");

    options.add_options()
      ("h,help","usage information")
      ("l,levels", "Number of levels in the network",cxxopts::value<int>()->default_value("2"))
      ("s,sizes","Sizes of the levels",cxxopts::value<std::vector<int»())
      ("o,optimizer", "Optimizer to be used, 0: SGD, 1: RMSprop",cxxopts::value<int>()->default_value("0"))
      ("e,epochs", "Number of epochs to train the network", cxxopts::value<int>()->default_value("1"))
      ("r,learningrate", "Learning rate for the optimizer", cxxopts::value<float>()->default_value("0.001"))
      ("b,batchsize", "Batch size for the training data", cxxopts::value<int>()->default_value("5"))
      ("t,tracing","Level of tracing: 0=default 1=scalars 2=arrays",cxxopts::value<int>()->default_value("0"))
      ;

    auto result = options.parse(argc,argv);
    if (result.count("help")) {
      std::cout « options.help() « std::endl;
      return 1;
    }

    std::vector<int> level_sizes{12,12};

    if (result.count("sizes")) {
      level_sizes = result["s"].as< std::vector<int> >();
    } else if (result.count("levels")) {
      int number_of_levels = result["l"].as<int>();
      level_sizes = std::vector<int>(number_of_levels,12);
    }

    set_trace_level( result["t"].as<int>() );
```

```
    int network_optimizer = result["o"].as<int>();
    int epochs = epochs = result["e"].as<int>();
    float lr = result["r"].as<float>();
    int batchSize = result["b"].as<int>();

    /*
     * Input data set handling
     */
    Dataset data(2);
    for (float item=-99.5; item<100; item+=1) {
      if (item<0) {
    // negative numbers are yes/no
    dataItem thisitem{std::vector<float>{item},std::vector<float>{1,0}};
    data.push_back(thisitem);
      } else {
    // positive numbers are no/yes
    dataItem thisitem{std::vector<float>{item},std::vector<float>{0,1}};
    data.push_back(thisitem);
      }
    }
    cout « "Dataset size: " « data.size() « endl; // Show size

    Net test_net( data );
    if (level_sizes.size()==2) {
      test_net.addLayer(16, RELU);
      test_net.addLayer(2, SIG); // SMAX );
    } else {
      for ( auto level_size : level_sizes ) {
    std::cout « "Adding level of size " « level_size « std::endl;
    test_net.addLayer(level_size,RELU);
      }
      test_net.addLayer(2, SIG); // SMAX );
    }

    test_net.set_learning_rate(lr);
    test_net.set_decay(0.9);
    test_net.set_momentum(0.9);
    test_net.set_optimizer(network_optimizer);
    test_net.set_lossfunction(mse);

    auto start_time = myclock::now();
    auto [train_data,test_data] = data.split(0.9);
    cout « "Initial accuracy: " « test_net.accuracy(test_data) « "\n";

    test_net.train(train_data,test_data, epochs, batchSize);
    auto duration = myclock::now()-start_time;
    auto microsec_duration = std::chrono::duration_cast<std::chrono::microseconds>(duration);
    std::cout « "Final Accuracy over test data: " « test_net.accuracy(data) « "\n"
          « "    attained in " « microsec_duration.count() « "usec" « "\n";

    //test_net.saveModel("weights.bin");
    test_net.info();

  } catch ( string e ) {
    cout « "Error «" « e « "» \n";
  } catch ( std::out_of_range ) {
    cout « "Out of range error\n";
  } catch ( ... ) {
    cout « "Uncaught exception\n";
  }

    return 0;
}
```

## 5.20 trace.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include "trace.h"

int trace_level;

```

```
void set_trace_level(int ell) { trace_level = ell; };
bool trace_progress() { return trace_level>=1; };
bool trace_scalars() { return trace_level>=2; };
bool trace_arrays()  { return trace_level>=3; };
```

## 5.21 trace.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

bool trace_progress();
bool trace_scalars();
bool trace_arrays();
void set_trace_level(int);
```

## 5.22 unittest.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include <iostream>
using std::cout;
#include <vector>
using std::vector;

#define CATCH_CONFIG_MAIN
#include "catch2/catch_all.hpp"

#include "funcs.h"
#include "net.h"

TEST_CASE( "functions","[1]" ) {
  /*
   * ReLU is linear >0, zero <0
   */
  {
    auto highpass = [] ( const float &x ) -> float {
      return relu_pt(x); };
    auto x = GENERATE( -5., -.5, .5, 2.5 );
    float y;
    REQUIRE_NOTHROW( y = highpass(x) );
    if (x<0)
      REQUIRE( y==Catch::Approx(0.) );
    else
      REQUIRE( y==Catch::Approx(x) );
  }

  /*
   * relu_slope avoids zero'ing the negative inputs
   */
  {
    auto leak = [] ( const float &x ) -> float {
      return relu_slope_pt(x); };
    auto x = GENERATE( -5., -.5, .5, 2.5 );
    float y;
    REQUIRE_NOTHROW( y = leak(x) );
    if (x<0)
      REQUIRE( y<0 );
    else
```

```
      REQUIRE( y==Catch::Approx(x) );
  }
}

TEST_CASE( "batch activation","[2]" ) {
  Vector values( vector<float>{1,2,.5} ), maxes(3);
  REQUIRE_NOTHROW( softmax_io( values,maxes ) );
  REQUIRE( maxes.positive() );
  const auto& maxvalues = maxes.values();
  INFO( "maxes: " « maxvalues[0] « "," « maxvalues[1] « "," « maxvalues[2] );
  auto maxit = maxvalues.begin();
  REQUIRE_NOTHROW( maxit = find( maxvalues.begin(),maxvalues.end(),
                   *max_element( maxvalues.begin(),maxvalues.end() ) ) );
  REQUIRE( maxit!=maxvalues.end() );
  int maxloc{-1};
  REQUIRE_NOTHROW( maxloc = distance( maxvalues.begin(),maxit ) );
  REQUIRE( maxloc==1 );
}

TEST_CASE( "scalar layer","[layer][11]" ) {
    Layer multiply(1,1);
    REQUIRE_NOTHROW
    (
     multiply
     .set_uniform_weights(1.)
     .set_uniform_biases(0.)
     .set_activation(
             [] (const float &x ) -> float { return id_pt(x); },
             [] (const float &x ) -> float { return 1; },
             "id"
             )
      );
    float input_number = 2.5f;
    VectorBatch
    scalar_in( Vector( vector<float>( {input_number} ) ) ),
    scalar_out( Vector( vector<float>( {1.2f} ) ) );
    REQUIRE( scalar_in.batch_size()==1 );
    REQUIRE( scalar_in.item_size()==1 );
    REQUIRE( scalar_out.batch_size()==1 );
    REQUIRE( scalar_out.item_size()==1 );
    REQUIRE( multiply.input_size()==1 );
    REQUIRE( multiply.output_size()==1 );
    float correct_result;
    SECTION( "identity" ) {
    correct_result = input_number;
    }
    SECTION( "scale by 2" ) {
    float multiplier = 2.f;
    REQUIRE_NOTHROW( multiply.set_uniform_weights(multiplier) );
    correct_result = multiplier * input_number;
    }
    SECTION( "shift by 2" ) {
    float shift = 2.f;
    REQUIRE_NOTHROW( multiply.set_uniform_biases(shift) );
    correct_result = shift + input_number;
    }
    REQUIRE_NOTHROW( multiply.forward(scalar_in,scalar_out) );
    REQUIRE( scalar_out.at(0,0) ==Catch::Approx( correct_result ) );
}

TEST_CASE( "highpass net: batch version","[net][21]" ) {
  Net highpass(1);
  REQUIRE_NOTHROW( highpass.addLayer
           (1,
            [] (const float &x ) -> float { return relu_pt(x); },
            [] (const float &x ) -> float { return x; },
            "highpass" ) );
  REQUIRE_NOTHROW( highpass.backlayer()
           .set_uniform_weights(1.)
           .set_uniform_biases(0.)
           );
  VectorBatch input(1,1),output(1,1);
  auto x = GENERATE( -.5, -5, .5, 1.5 );
  input.data()[0] = x;
  REQUIRE_NOTHROW( highpass.feedForward(input,output) );
  auto y = output.data()[0];
  INFO( "Input = " « x « "-> output = " « y );
  if ( x>0 )
      REQUIRE( y>0 );
  else
      REQUIRE( y==Catch::Approx(0.0) );
}

TEST_CASE( "highpass net: vector version","[net][22]" ) {
  Net highpass(1);
  REQUIRE_NOTHROW( highpass.addLayer
           (1,
```

```
            [] (const float &x ) -> float { return relu_pt(x); },
            [] (const float &x ) -> float { return x; },
            "highpass" ) );
  REQUIRE_NOTHROW( highpass.backlayer()
           .set_uniform_weights(1.)
           .set_uniform_biases(0.)
           );
  Vector input(1),output(1);
  auto x = GENERATE( -.5, -5, .5, 1.5 );
  input[0] = x;
  REQUIRE_NOTHROW( highpass.feedForward(input,output) );
  INFO( "Input = " « input[0] « "-> output = " « output[0] );
  if ( x>0 )
      REQUIRE( output[0]>0 );
  else
      REQUIRE( output[0]==Catch::Approx(0.0) );
}

TEST_CASE( "low pass net: vector version","[net][23]" ) {
  Net lowpass(1);
  REQUIRE_NOTHROW( lowpass.addLayer
           (1,
            [] (const float &x ) -> float { return relu_pt(1-x); },
            [] (const float &x ) -> float { return x; },
            "lowpass" ) );
  REQUIRE_NOTHROW( lowpass.backlayer()
           .set_uniform_weights(1.)
           .set_uniform_biases(0.)
           );
  Vector input(1),output(1);
  auto x = GENERATE( -.5, -5, .5, 1.5 );
  input[0] = x;
  REQUIRE_NOTHROW( lowpass.feedForward(input,output) );
  INFO( "Input = " « input[0] « "-> output = " « output[0] );
  if ( x<1 )
      REQUIRE( output[0]>0 );
  else
      REQUIRE( output[0]==Catch::Approx(0.0) );
}

```

## 5.23 vector.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include <algorithm>
#include <iostream>
#include <cassert>

#include "vector.h"
#include "vector2.h"

/*
 * These are the vector routines that do not have a optimized implementation,
 * such as using BLIS.
 */


Vector::Vector(){
}

int Vector::size() const { return vals.size(); };

Vector::Vector( std::vector<float> vals )
  : vals(vals) {
  r = vals.size();
  c = 1;
};

void Vector::copy_from( const VectorBatch& batch ) {
    const auto& batch_vals = batch.data();
    assert( batch.nelements()>=vals.size() );
```

```
    for (int i=0; i<vals.size(); i++) {
    vals[i] = batch_vals[i];
    }
};

void Vector::square() {
    std::for_each(vals.begin(), vals.end(), [](auto &n) {n*=n;});
}

Vector& Vector::operator=(const Vector &m2) { // Overloading the = operator
    vals = m2.vals;
    return *this;
}

// Note: no element-wise, non destructive operations in BLIS, so no implementations for those yet
// There are element wise operations in MKL I believe
Vector Vector::operator+(const Vector &m2) {
    assert(m2.size()==this->size());
    Vector out(m2.size(),0);
    for (int i=0;i<m2.size();i++) {
        out.vals[i] = this->vals[i] + m2.vals[i];
    }
    return out;
}

Vector Vector::operator-(const Vector &m2) {
    assert(m2.size()==this->size());
    Vector out(m2.size(),0);
    for (int i=0;i<m2.size();i++) {
        out.vals[i] = this->vals[i] - m2.vals[i];
    }
    return out;
}


Vector operator-(const float &c, const Vector &m) {
    Vector o=m;
    for (int i=0;i<m.size();i++) {
        o.vals[i] = c - o.vals[i];
    }
    return o;
}

Vector operator*(const float &c, const Vector &m) {
    Vector o=m;
    for (int i=0;i<m.size();i++) {
        o.vals[i] = c * o.vals[i];
    }
    return o;
}

Vector Vector::operator/=( float c ) {
    for (int i=0;i<vals.size();i++) {
        vals[i] = vals[i] / c;
    }
    return *this;
}

Vector Vector::operator*(const Vector &m2) { // Hadamard product
    Vector out(m2.size(), 0);
    for (int i = 0; i < m2.size(); i++) {
        out.vals[i] = this->vals[i] * m2.vals[i];
    }
    return out;
}

Vector Vector::operator/(const Vector &m2) { // Element wise division
    Vector out(m2.size(), 0);
    for (int i = 0; i < m2.size(); i++) {
        out.vals[i] = this->vals[i] / m2.vals[i];
    }
    return out;
}

Vector Vector::operator-() {
    Vector result = *this;
    for (int i = 0; i < size(); i++){
        result.vals[i] = -vals[i];
    }

    return result;
};
```

## 5.24 vector.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#ifndef SRC_VECTOR_H
#define SRC_VECTOR_H
#include <algorithm>
#include <vector>
#include <cassert>
#include <cmath>

class VectorBatch; // forward for friending
class Matrix; // forward for friending
class Vector {
  friend class VectorBatch;
  friend class Matrix;
private:
    std::vector<float> vals;
public:
    Vector();
    Vector( int n )
    : vals(std::vector<float>(n)) {};
    Vector( std::vector<float> vals );
    Vector(int size, int init);
    int size() const;
    int r; int c=1; /* VLE these need to go! */
    void show();
    void add( const Vector &v1);
    void set_ax( float a, Vector &x );
    std::vector<float>& values() { return vals; };
    const std::vector<float>& values() const { return vals; };
    void copy_from( const VectorBatch& );
    float *data() { return vals.data(); };
    const float *data() const { return vals.data(); };
    void zeros();
    void square();
    float& operator[](int i) { return vals[i]; };
    float operator[](int i) const { return vals[i]; };
    Vector operator-(); // Unary negate operator
    Vector& operator=(const Vector& m2); // Copy constructor
    Vector operator+(const Vector &m2); // Element-wise addition
    Vector operator*(const Vector &m2); // Hadamard Product Element-wise multiplication
    Vector operator/(const Vector &m2); // Element-wise division
    Vector operator/=(float x);
    Vector operator-(const Vector &m2); // Element-wise subtraction
    friend Vector operator-(const float &c, const Vector &m); // for constant
    friend Vector operator*(const float &c, const Vector &m); // for constant-matrix multiplication
    //friend Vector operator/(const Vector &m, const float &c); // for matrix-constant division

  bool positive() const {
    return all_of
      ( vals.begin(),vals.end(),
    [] (float e) { return e>0; }
    );
  }
};

class Categorization {
private:
  std::vector<float> _probabilities;
public:
  Categorization( Vector v )
    : _probabilities(v.values()) {};
  Categorization( std::vector<float> p)
    : _probabilities(p) {};
  Categorization(int n)
    : _probabilities( std::vector<float>(n) ) {};
  Categorization(int n,int i)
    : _probabilities( std::vector<float>(n) ) {
    _probabilities.at(i) = 1.;
  };
  const std::vector<float>& probabilities() const { return _probabilities; };
  int size() const { return _probabilities.size(); };
  void normalize() {
    auto it = std::max_element(_probabilities.begin(), _probabilities.end());
    std::fill(_probabilities.begin(), _probabilities.end(), 0);
```

```
    *it = 1;
  };
  bool close_enough( const Categorization& approx ) const {
    return close_enough( approx.probabilities() );
  };
  bool close_enough( const std::vector<float>& approx ) const {
    assert( size()==approx.size() );
    //return _probabilities==approx;
    bool close{true};
    for ( int i=0; i<size(); i++) {
      close = close and
    ( ( _probabilities.at(i)==approx.at(i) )
      or ( approx.at(i)==0. and ( std::abs(_probabilities.at(i))<1.e-5 ) )
      or ( std::abs( (_probabilities.at(i)-approx.at(i))/approx.at(i) )<1.e-5 )
      );
    }
    return close;
  };
};

#endif //SRC_VECTOR_H
```

## 5.25 vector2.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include <iostream>
using std::cout;
using std::endl;
#include <iomanip>
using std::setprecision;
#include <string>
using std::string;
#include <vector>
#include <algorithm>
#include <cassert>
using std::vector;

#include "trace.h"
#include "vector2.h"

/*
 * These are the vector batch routines that do not have a optimized implementation,
 * such as using BLIS.
 */

VectorBatch::VectorBatch() { // Default constructor
}

VectorBatch::VectorBatch( int vs ) {
  allocate(0,vs);
}

VectorBatch::VectorBatch( const Vector& v ) {
    int s = v.size();
    allocate(1,s);
    copy( v.vals.begin(),v.vals.end(),vals.begin() );
    // for ( int i=0; i<s; i++ )
    //  vals[i] = v[i];
};

void VectorBatch::allocate(int batchsize,int itemsize) {
  // we allow a batchsize of zero
  assert(batchsize>=0);
  assert(itemsize>0);
  vals.resize(batchsize*itemsize);
  set_batch_size(batchsize);
  set_item_size(itemsize);
};

void VectorBatch::display( string header) const {
  cout « header « "\n";
```

```
  for (int j=0; j<batch_size(); j++) {
    for (int i=0; i<item_size(); i++)
      cout « setprecision(5)
       « vals_vector().at( INDEXc(i,j,item_size(),batch_size()) )
       « " ";
    cout « "\n";
  }
};

void VectorBatch::set_col(int j,const std::vector<float> &v ) {
  assert( j<batch_size() );
  assert( v.size()==item_size() );
  for (int i = 0; i<nvectors; i++) {
    vals.at( j + vector_size * i ) = v.at(i);
  };
};

void VectorBatch::add_vector( const std::vector<float> &v ) {
  const int Nelements = vals.size();
  const int vector_length = v.size();
  if (Nelements==0)
    set_item_size(vector_length);
  else {
    assert( vector_length==item_size() );
    assert( Nelements%vector_length==0 );
  }
  const int m = Nelements/vector_length;
  vals.resize(Nelements+vector_length); nvectors++;
  for (int i = 0; i < vector_length; i++) {
    vals.at( m * vector_length + i ) = v.at(i);
  };
};

std::vector<float> VectorBatch::get_col(int j) const {
  assert( j<batch_size() );
  const int c = item_size();
  std::vector<float> col(c);
  for (int i=0; i<c; i++)
    col.at(i) = vals.at( j + c*i );
  return col;
};

void VectorBatch::set_row( int j, const std::vector<float> &v ) {
  const int c = item_size();
  assert( v.size()==c );
  for (int i = 0; i < c; i++) {
    vals.at( j * c + i ) = v.at(i);
  };
}

std::vector<float> VectorBatch::get_row(int j) const {
  assert( j<batch_size() );
  const int n = item_size();
  std::vector<float> row(n);
  for (int i = 0; i < n; i++)
    row.at(i) = vals.at( j * n + i );
  return row;
};

std::vector<float> VectorBatch::extract_vector(int j) const {
  assert( j<batch_size() );
  const int m = item_size();
  std::vector<float> v(m);
  for (int i = 0; i < m; i++)
    v.at(i) = vals.at( INDEXc(i,j,m,batch_size()) );  // (j * n + i );
  return v;
  //  return get_row(v);
};
#ifdef USE_GSL
gsl::span<float> VectorBatch::get_vector(int v) {
  const int c = item_size();
  return gsl::span<float>( &vals[v*c], c );
};
// const gsl::span<float> VectorBatch::get_vector(int v) const {
//   const int c = item_size();
//   return gsl::span<float>( data(v*c) /* &vals[v*c] */, c );
// };
#else
std::vector<float> VectorBatch::get_vector(int v) const {
  return extract_vector(v);
};
#endif

void VectorBatch::set_vector( const Vector &v, int j) {
  set_vector( v.vals,j );
}

```

```
void VectorBatch::set_vector( const vector<float> &v, int j) {
  const int c = item_size();
  assert( v.size()==c );
  for (int i=0; i<c; i++)
    vals.at( j * c + i ) = v.at(i);
}

void VectorBatch::addh(const Vector &y) { // Add y to every row
  const int r = item_size(), c = batch_size();
  assert( r==y.size() );
  for (int j=0; j<c; j++ ) {
    for (int i=0; i<r; i++) {
      vals.at( INDEXc(i,j,r,c) ) += y.vals[i];
    }
  }
}

// void VectorBatch::add(const VectorBatch &y) { // Add y to every row
//   const int r = item_size(), c = batch_size();
//   assert( r==y.size() );
//   asserr( c==y.batch_size() );
//   for (int j=0; j<c; j++ ) {
//     for (int i=0; i<r; i++) {
//       vals.at( INDEXc(i,j,r,c) ) += y.vals.at( INDEXc(i,j,r,c) );
//     }
//   }
// }

Vector VectorBatch::meanh() const { // Returns a vector of row-wise means
  const int r = item_size(), c = batch_size();
  Vector mean(r, 0);
  for (int i=0; i<r; i++) {
    float avg = 0.f;
    for ( int j=0; j<c; j++ ) {
      avg += vals.at( INDEXc(i,j,r,v) );
    }
    mean.vals[i] = avg/static_cast<float>( item_size() );
  }
  return mean;
}


/*
 * VLE dangerous. et rid of this one
 */
VectorBatch &VectorBatch::operator=(const VectorBatch &m2) { // Overloading the = operator
  set_item_size( m2.item_size() );
  set_batch_size( m2.batch_size() );

  this->vals = m2.vals; // IM Since we're using vectors we can just use the assignment from that

  return *this;
}


VectorBatch VectorBatch::operator-(const VectorBatch &m2) const {
  assert( item_size()==m2.item_size() );
  assert( batch_size()==m2.batch_size() );
  const int c = m2.item_size(), r = m2.batch_size();
  VectorBatch out(r, c, 0);
    for (int i = 0; i < r * c; i++) {
        out.vals[i] = this->vals[i] - m2.vals[i];
    }
    return out;
}

VectorBatch VectorBatch::operator*(const VectorBatch &m2) { // Hadamard product
  assert( item_size()==m2.item_size() );
  assert( batch_size()==m2.batch_size() );
  const int c = m2.item_size(), r = m2.batch_size();
  VectorBatch out(r, c, 0);
  for (int i = 0; i < nelements(); i++) {
        out.vals[i] = this->vals[i] * m2.vals[i];
    }
    return out;
}

void VectorBatch::hadamard(const VectorBatch& m1,const VectorBatch& m2) {
  const int r = item_size(), c = batch_size();
  assert( r==m1.item_size() ); assert( c==m1.batch_size() );
  assert( r==m2.item_size() ); assert( c==m2.batch_size() );

  const auto& m1vals = m1.vals_vector();
  const auto& m2vals = m2.vals_vector();
  for (int i=0; i<r*c; i++) {
    vals.at(i) = m1vals.at(i) * m2vals.at(i);
    if ( isinf(vals.at(i)) )
```

```
      cout « "inf from " « m1vals.at(i) « " * " « m2vals.at(i) « "\n";
  }
  if (trace_progress()) {
    assert( m1.notinf() ); assert( m1.notnan() ); assert( m1.normf()!=0.f );
    assert( m2.notinf() ); assert( m2.notnan() ); assert( m2.normf()!=0.f );
    assert( this->notinf() ); assert( this->notnan() ); assert( this->normf()!=0.f );
  }
  if (trace_scalars()) {
    cout « "delta: "
     « m1.normf() « "x" « m2.normf() « " => " « this->normf() « "\n";
  }
}

VectorBatch VectorBatch::operator/(const VectorBatch &m2) { // Hadamard product
  const int c = m2.item_size(), r = m2.batch_size();
  assert( item_size()==c );
  assert( batch_size()==r );
  VectorBatch out(r, c, 0);
  for (int i = 0; i < nelements(); i++) {
    out.vals[i] = this->vals[i] / m2.vals[i];
  }
  return out;
}

void VectorBatch::scaleby( float f) {
  for (int i = 0; i < nelements(); i++) {
    vals[i] *= f;
  }
}

VectorBatch VectorBatch::operator-() {
    VectorBatch result = *this;
    for (int i = 0; i < nelements(); i++) {
        result.vals[i] = -vals[i];
    }

    return result;
};

VectorBatch operator/(const VectorBatch &m, const float &c) {
    VectorBatch o = m;
    for (int i = 0; i < m.nelements(); i++) {
        o.vals[i] = o.vals[i] / c;
    }
    return o;
}

VectorBatch operator*(const float &c, const VectorBatch &m) {
    VectorBatch o = m;
    for (int i = 0; i < m.nelements(); i++) {
        o.vals[i] = o.vals[i] * c;
    }
    return o;
}
```

## 5.26 vector2.h

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#ifndef CODE_VEC2_H
#define CODE_VEC2_H

#include <vector>
#include "vector.h"
#include "matrix.h"
#include <iostream>
#ifdef BLISNN
#include "blis/blis.h"
#endif

#ifdef USE_GSL
#include "gsl/gsl-lite.hpp"
```

```
#endif

#define INDEXr(i,j,m,n) (i)*(n)+(j)
#define INDEXc(i,j,m,n) (i)+(j)*(m)

class VectorBatch{
  friend class Matrix;
  friend class Vector;

private: //private:
  std::vector<float> vals;
  int nvectors{0},vector_size{0};
public:
    VectorBatch();
    VectorBatch( int itemsize );
    // this one is in the blis/reference file
    VectorBatch(int nRows, int nCols, bool rand=false);
    VectorBatch( const Vector& );
    void allocate(int,int);

    int size() const { return vals.size(); };
    void resize(int m,int n) {
      nvectors = m; set_item_size(n); //r = m; c = n;
      vals.resize(m*n); };
    float normf() const {
      float norm{0.f}; int count{0};
      for ( auto e : vals ) {
    norm += e*e; count++;
      }
      //std::cout « "norm squared over " « count « " elements: " « norm « "\n";
      return sqrt(norm);
    };
    bool positive() const {
      return all_of
    ( vals.begin(),vals.end(),
      [] (float e) { return e>0; }
    );
    }
    bool notnan() const {
      return all_of
    ( vals.begin(),vals.end(),
      [] (float e) { return not isnan(e); }
    );
    }
    bool notinf() const {
      return all_of
    ( vals.begin(),vals.end(),
      [] (float e) { return not isinf(e); }
    );
    }
    int item_size() const { return vector_size; };
    void set_item_size(int n) { vector_size = n; };
    int batch_size() const { return nvectors; };
    void set_batch_size(int n) { nvectors = n; };
    int nelements() const {
      int n = vals.size();
      assert( n==item_size()*batch_size() );
      return n;
    };

    std::vector<float>& vals_vector() { return vals; };
    const std::vector<float>& vals_vector() const { return vals; };
    float *data() { return vals.data(); };
    const float *data() const { return vals.data(); };
    const float *data( int disp ) const { return vals.data()+disp; };


    void v2mp( const Matrix &x, VectorBatch &y) const;
    void v2tmp( const Matrix &x, VectorBatch &y ) const;
    void v2mtp( const Matrix &x, VectorBatch &y ) const;
    void outer2( const VectorBatch &x, Matrix &y ) const;

  void add_vector( const std::vector<float> &v );

    /*
     * Indexing
     */
    float at(int i,int j) const {
    assert( i>=0 ); assert( i<vector_size );
    assert( j>= 0 ); assert( j<nvectors );
    return *data( i + j*vector_size );
    };
    void set_col(int j,const std::vector<float> &v );
    std::vector<float> get_col(int j) const;
    void set_row( int j, const std::vector<float> &v );
    std::vector<float> get_row(int j) const;
    std::vector<float> extract_vector(int v) const;
```

```
#ifdef USE_GSL
  gsl::span<float> get_vector(int v);
  //  const gsl::span<float> get_vector(int v) const;
  void set_vector( const gsl::span<float> &v, int j);
#else
  std::vector<float> get_vector(int v) const;
#endif
  void set_vector( const Vector &v, int j);
  void set_vector( const std::vector<float>&v, int j);

  Vector get_vectorObj(int j) const {
    Vector vec(vector_size);
    std::copy( vals.begin()+j*vector_size,vals.begin()+(j+1)*vector_size,
           vec.vals.begin() );
    // for (int i = 0; i < vector_size; i++ )
    //   vec.vals.at(i) = vals.at( j * vector_size + i );
    return vec;
  }


  void show() const;
  void display(std::string) const;

  void copy_from( const VectorBatch& in ) {
    assert( vals.size()==in.vals.size() );
    std::copy( in.vals.begin(),in.vals.end(),vals.begin() );
    // for ( int i=0; i<vals.size(); i++)
    //   vals[i] = in.vals[i];
  };

  void addh(const Vector &y);
  void addh(const VectorBatch &y);
  Vector meanh() const;

  VectorBatch operator-(); // Unary negate operator
  VectorBatch& operator=(const VectorBatch& m2); // Copy constructor
  void hadamard(const VectorBatch& m1,const VectorBatch& m2);
  VectorBatch operator*(const VectorBatch &m2); // Hadamard Product Element-wise multiplication
  VectorBatch operator/(const VectorBatch &m2); // Element-wise division
  void scaleby( float );
  VectorBatch operator-(const VectorBatch &m2) const; // Element-wise subtraction
  friend VectorBatch operator/(const VectorBatch &m, const float &c); // for matrix-constant division
  friend VectorBatch operator*(const float &c, const VectorBatch &m); // for matrix-constant division
};


#endif

#if 0
  // hm. this doesn't work
  friend void relu_io    (const Vector &i, Vector &v);
  friend void sigmoid_io (const Vector &i, Vector &v);
  friend void softmax_io (const Vector &i, Vector &v);
  friend void none_io    (const Vector &i, Vector &v);
  friend void reluGrad_io(const Vector &m, Vector &a);
  friend void sigGrad_io (const Vector &m, Vector &a);

  friend void relu_io    (const VectorBatch &i, VectorBatch &v);
  friend void sigmoid_io (const VectorBatch &i, VectorBatch &v);
  friend void softmax_io (const VectorBatch &i, VectorBatch &v);
  friend void none_io    (const VectorBatch &i, VectorBatch &v);
  friend void reluGrad_io(const VectorBatch &m, VectorBatch &a);
  friend void sigGrad_io (const VectorBatch &m, VectorBatch &a);
#endif

```

## 5.27 vector_impl_blis.cpp

```
/*****************************************************************
 *****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 *****************************************************************
 *****************************************************************/

#include <algorithm>
#include "vector.h"
```

```
#include <iostream>

#include <cassert>

#ifdef BLISNN
#include "blis/blis.h"
#endif

Vector::Vector(int s, int init) {
    r = s;
  vals = std::vector<float>(s);
  float scal_fac = 0.05;
  if (init==0){
    float zero = 0.0;
    bli_ssetv( BLIS_NO_CONJUGATE, s, &zero, &vals[0], 1);
  }else if (init==1)
    bli_srandv(s, &vals[0], 1);
    bli_sscalv( BLIS_NO_CONJUGATE, s, &scal_fac, &vals[0], 1 );
}

void Vector::add( const Vector &v1 ) {
  assert(v1.size()==this->size());
  bli_saddv( BLIS_NO_CONJUGATE, size(), const_cast<float*>(&v1.vals[0]), 1, &vals[0], 1 );
}

void Vector::set_ax( float a, Vector &x) {
  assert(x.size()==this->size());

    float b = static_cast<float>(a);
    bli_sscal2v(BLIS_NO_CONJUGATE, this->size(),&b,&x.vals[0],1,&(this->vals)[0], 1);
}


void Vector::zeros() {

    float zero = 0.0;
    bli_ssetv( BLIS_NO_CONJUGATE, size(), &zero, &vals[0], 1 ); // Set all values to 0
}

void Vector::show() {

    char sp[8] = " ";
    char format[8] = "%4.4f";
    bli_sprintm( sp, size(), 1, &vals[0], 1, size(), format, sp );
}


```

## 5.28 vector_impl_reference.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include <algorithm>
#include "vector.h"
#include <iostream>

#include <cassert>

Vector::Vector(int s, int init) {
    r = s;
  vals = std::vector<float>(s);
  if (init==0){
    std::fill(vals.begin(),vals.end(), 0);
  }else if (init==1){
    for (int i=0; i<size(); i++){
      vals[i] = -0.1 + static_cast <float> (rand()) /( static_cast <float> (RAND_MAX/(0.1-(-0.1))));
    }
  }

}

void Vector::add( const Vector &v1 ) {
```

```
  assert(v1.size()==this->size());
  for (int i=0; i<size(); i++){
    vals[i] += v1.vals[i];
  }
}

void Vector::set_ax( float a, Vector &x) {
  assert(x.size()==this->size());
    for (int i=0; i<size(); i++){
        vals[i] = a * x.vals[i];
    }

}


void Vector::zeros() {
    std::fill(vals.begin(),vals.end(),0);

}

void Vector::show() {
    int i;
    for (i=0;i<size();i++) {
        std::cout « vals[i] « '\n';
    }
    std::cout « '\n';

}


```

## 5.29 vectorbatch_impl_blis.cpp

```
/*****************************************************************
 *****************************************************************
 ****
 **** This text file is part of the source of
 **** `Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 *****************************************************************
 *****************************************************************/

#include <iostream>
using std::cout;
using std::endl;
#include <vector>
using std::vector;
#include <algorithm>
#include "trace.h"

#include "trace.h"
#include "vector2.h"

#ifdef BLISNN
#include "blis/blis.h"
#endif

VectorBatch::VectorBatch(int nRows, int nCols, bool random) {
  allocate(nRows,nCols);
  const int r=nRows, c=nCols;

  float scal_fac = 0.05; // randomize between (-scal;scal)
  if (not random){
    float zero = 0.0;
    bli_ssetm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
           r, c, &zero, &vals[0], c, 1);
  } else if (random){
    bli_srandm(0, BLIS_DENSE, r, c, &vals[0], c, 1);
    bli_sscalm( BLIS_NO_CONJUGATE, 0, BLIS_NONUNIT_DIAG, BLIS_DENSE,
        r, c, &scal_fac, &vals[0], c, 1);
  }
}

// VectorBatch VectorBatch::transpose() const {
//     const int c = batch_size(), r = item_size();
//     VectorBatch result(c, r, 0); // Initialize a new matrix with inverted dimension values

//  bli_scopym( 0, BLIS_NONUNIT_DIAG, BLIS_DENSE, BLIS_TRANSPOSE,
//          c, r, const_cast<float*>(&vals[0]), c, 1, &result.vals[0], r, 1);
```

```
//     return result;
// }

void VectorBatch::show() const {

    const int c = batch_size(), r = item_size();
    char e[5] = "";
    char forvals[8] = "%4.4f";
    bli_sprintm( e, r, c, const_cast<float*>(&vals[0]), c, 1, forvals, e );
}


/*
 * For explanation of the BLIS routines, see
 * https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md#gemm
 * and
 * https://github.com/flame/blis/blob/master/docs/BLISTypedAPI.md#computational-function-reference
 */
void VectorBatch::v2mp(const Matrix &m, VectorBatch &y) const {
  const int
    xr = item_size(),   xc = batch_size(),   // column storage
    mr = m.rowsize(),   mc = m.colsize(),    // row storage
    yr = y.item_size(), yc = y.batch_size(); // column

  if (trace_scalars())
    cout « "matrix vector product "
     « mr « "x" « mc
     « " & "
     « xr « "x" « xc
     « " => " « yr « "x" « yc
     « endl;

  assert( xc==yc );
  assert( yr==mr );
  assert( mc==xr );

  const auto& mmat  = m.values().data();
  const auto& xvals = vals_vector().data();
  auto  yvals       = y.vals_vector().data();

  float alpha = 1.0;
  float beta = 0.0;
  bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_NO_TRANSPOSE,
         yr,yc,mc,
         &alpha,
         const_cast<float*>(mmat),  /* rsa,csa */ mr,1,
         const_cast<float*>(xvals), /* rsb,csb */ 1,xr,
         &beta,
         yvals,                     /* rsc,csc */ 1,yr
         );
}

void VectorBatch::v2tmp(const Matrix &x, VectorBatch &y) const {

    const int c = batch_size(), r = item_size();
    assert( r==x.rowsize() );
    assert( c==y.batch_size() );
    assert( x.colsize()==y.item_size() );

    float alpha = 1.0;
    float beta = 0.0;
    //printf("BLIS gemm %dx%dx%d\n",r,x.colsize(),c);
    bli_sgemm( BLIS_TRANSPOSE, BLIS_NO_TRANSPOSE,
           c, x.colsize(), r, &alpha, const_cast<float*>(&vals[0]),
           c, 1, const_cast<float*>( x.data() ),
           x.colsize(), 1, &beta, &y.vals[0], x.colsize(), 1);
}

// matrix transpose x self => y
void VectorBatch::v2mtp(const Matrix &m, VectorBatch &y) const {
  const int
    xr = item_size(),   xc = batch_size(),   // column storage
    mr = m.rowsize(),   mc = m.colsize(),    // row storage
    yr = y.item_size(), yc = y.batch_size(); // column

  if (trace_scalars())
    cout « "matrix transpose vector product "
     « mr « "x" « mc
     « " & "
     « xr « "x" « xc
     « " => " « yr « "x" « yc
     « endl;

  assert( xc==yc );
  assert( yr==mc );
  assert( mr==xr );

```

```
  const auto& mmat  = m.values().data();
  const auto& xvals = vals_vector().data();
  auto        yvals = y.vals_vector().data();

  float alpha = 1.0;
  float beta = 0.0;
  bli_sgemm( BLIS_TRANSPOSE, BLIS_NO_TRANSPOSE,
         yr,yc,mr,
         &alpha,
         const_cast<float*>(mmat),  /* rsa,csa */ mc,1,
         const_cast<float*>(xvals), /* rsb,csb */ 1,xr,
         &beta,
         yvals,                     /* rsc,csc */ 1,yr
         );
  if (trace_progress()) {
    assert( this->notinf() ); assert( this->notnan() ); assert( this->normf()!=0.f );
    assert( m.notinf() ); assert( m.notnan() ); assert( m.normf()!=0.f );
    assert( y.notinf() ); assert( y.notnan() ); assert( y.normf()!=0.f );
  }

}

/*
 * x times self => m
 */
void VectorBatch::outer2(const VectorBatch &x, Matrix &m) const {
  const int
    yr = item_size(),   yc = batch_size(),   // column storage
    mr = m.rowsize(),   mc = m.colsize(),    // row storage
    xr = x.item_size(), xc = x.batch_size(); // column

  if (trace_scalars())
    cout « "outer product "
     « xr « "x" « xc
     « " & "
     « yr « "x" « yc
     « " => " « mr « "x" « mc
     « endl;

  assert( yc==xc );
  assert( xr==mr );
  assert( yr==mc );
  const auto& xvals = x.vals_vector().data();
  const auto& yvals =   vals_vector().data();
  auto        mmat  = m.values().data();

  float alpha = 1.0;
  float beta = 0.0;
  bli_sgemm( BLIS_NO_TRANSPOSE, BLIS_TRANSPOSE,
         mr,mc,yc,
         &alpha,
         const_cast<float*>(xvals), /* rsa,csa */ 1,xr,
         const_cast<float*>(yvals), /* rsb,csb */ 1,yr,
         &beta,
         mmat,                      /* rsc,csc */ mc,1
         );

}
```

## 5.30 vectorbatch_impl_reference.cpp

```
/****************************************************************
 ****************************************************************
 ****
 **** This text file is part of the source of
 **** 'Introduction to High-Performance Scientific Computing'
 **** by Victor Eijkhout, copyright 2012-2021
 ****
 **** Deep Learning Network code
 **** copyright 2021 Ilknur Mustafazade
 ****
 ****************************************************************
 ****************************************************************/

#include "trace.h"
#include "vector2.h"
#include <iostream>
using std::cout;
using std::endl;
#include <vector>
using std::vector;
#include <algorithm>

```

```
#ifdef DEBUG
#define ELEMENTc(X,I,J,M,N) X.at( INDEXc(I,J,M,N) )
#define ELEMENTr(X,I,J,M,N) X.at( INDEXr(I,J,M,N) )
#else
#define ELEMENTc(X,I,J,M,N) X[ INDEXc(I,J,M,N) ]
#define ELEMENTr(X,I,J,M,N) X[ INDEXr(I,J,M,N) ]
#endif

VectorBatch::VectorBatch(int batchsize, int itemsize, bool random) {
  allocate(batchsize,itemsize);

  int i, j;
  if (not random){
    std::fill(vals.begin(), vals.end(), 0);
  }else if (random){
    for (i=0; i<batchsize * itemsize;i++){
      vals[i] = -0.1 + static_cast <float> (rand()) /( static_cast <float>(RAND_MAX/(0.1-(-0.1))));
    }
  }
}

// VectorBatch VectorBatch::transpose() const {
//     const int c = batch_size(), r = item_size();
//      // Initialize a new matrix with inverted dimension values
//     VectorBatch result(item_size(),batch_size(), 0);
//     int i1, i2; // Old and new index
//     for (int i = 0; i < r; i++) {
//         for (int j = 0; j < c; j++) {
//             i1 = i * c + j; // Old indexing
//             i2 = j * r + i; // New indexing

//             result.vals[i2] = vals[i1]; // Move transposed values to new array
//         }
//     }
//     return result;
// }

void VectorBatch::show() const {
    const int c = batch_size(), r = item_size();
    int i, j;
    for (i = 0; i < r; i++) {
        for (j = 0; j < c; j++) {
            std::cout « vals[i * c + j] « ' ';
        }
        std::cout « std::endl;
    }
    std::cout « std::endl;
}


// y = x x
void VectorBatch::v2mp(const Matrix &m, VectorBatch &y) const {
  const int
    xr = item_size(),   xc = batch_size(),   // column storage
    mr = m.rowsize(),   mc = m.colsize(),    // row storage
    yr = y.item_size(), yc = y.batch_size(); // column

  if (trace_scalars())
    cout « "matrix vector product "
     « mr « "x" « mc
     « " & "
     « xr « "x" « xc
     « " => " « yr « "x" « yc
     « endl;

  assert( xc==yc );
  assert( yr==mr );
  assert( mc==xr );

  const auto& mmat = m.values();
  const auto& xvals = vals_vector();
  auto& yvals = y.vals_vector();
  int xi{0},yi{0},mi{0};
  for (int i = 0; i < yr; i++) { // Matrix multiplication subroutine
    for (int j = 0; j < yc; j++) {
      float sum = 0.0;
      for (int k = 0; k < mc; k++) {
    sum += ELEMENTr( mmat,i,k,mr,mc ) * ELEMENTc( xvals,k,j,xr,xc ) ;
    mi = INDEXr( i,k,mr,mc ); xi = INDEXc( k,j,xr,xc ) ;
    //sum += mmat.at( INDEXr(i,k,mr,mc) ) * xvals.at( INDEXc(k,j,xr,xc) ) ;
      }
      //yvals.at( INDEXc(i,j,yr,yc) ) = sum;
      ELEMENTc( yvals,i,j,yr,yc ) = sum;
      yi = INDEXc( i,j,yr,yc );
    }
  }
  assert( xi==xvals.size()-1 );
```

```
  assert( yi==yvals.size()-1 );
  assert( mi==mmat.size()-1 );
}


// matrix transpose x self => y
void VectorBatch::v2mtp(const Matrix &m, VectorBatch &y) const {
  const int
    xr = item_size(),   xc = batch_size(),   // column storage
    mr = m.rowsize(),   mc = m.colsize(),    // row storage
    yr = y.item_size(), yc = y.batch_size(); // column

  if (trace_scalars())
    cout « "matrix transpose vector product "
     « mr « "x" « mc
     « " & "
     « xr « "x" « xc
     « " => " « yr « "x" « yc
     « endl;

  assert( xc==yc ); assert( yr==mc ); assert( mr==xr );

  const auto& mmat  = m.values();
  const auto& xvals = vals_vector();
  auto&       yvals = y.vals_vector();
  for (int i = 0; i < yr; i++) { // Matrix multiplication subroutine
    for (int j = 0; j < yc; j++) {
      float sum = 0.0;
      for (int k = 0; k < mr; k++) {
    //sum += mmat[ INDEXr(k,i,mc,mr) ] * xvals[ INDEXc(k,j,xr,xc) ];
    // matrix is by rows, so transpose by columns!
    sum += ELEMENTc( mmat,i,k,mc,mr ) * ELEMENTc( xvals,k,j,xr,xc );
      }
      yvals.at( INDEXc(i,j,yr,yc) )  = sum;
    }
  }
  if (trace_progress()) {
    assert( this->notinf() ); assert( this->notnan() ); assert( this->normf()!=0.f );
    assert( m.notinf() ); assert( m.notnan() ); assert( m.normf()!=0.f );
    assert( y.notinf() ); assert( y.notnan() ); assert( y.normf()!=0.f );
  }
  if (trace_scalars()) {
    cout « "v2mtp (computes dl): "
        « this->normf() « "x" « m.normf() « " => " « y.normf() « "\n";
  }
}

/*
 * x times self => m
 */
void VectorBatch::outer2(const VectorBatch &x, Matrix &m ) const {
  const int
    yr = item_size(),   yc = batch_size(),   // column storage
    mr = m.rowsize(),   mc = m.colsize(),    // row storage
    xr = x.item_size(), xc = x.batch_size(); // column

  if (trace_scalars())
    cout « "outer product "
     « xr « "x" « xc
     « " & "
     « yr « "x" « yc
     « " => " « mr « "x" « mc
     « endl;

  assert( yc==xc );
  assert( xr==mr );
  assert( yr==mc );
  const auto& xvals = x.vals_vector();
  const auto& yvals =   vals_vector();
  auto& mmat = m.values();
    for (int i = 0; i < mr; i++) { // Matrix multiplication subroutine
      for (int j = 0; j < mc; j++) {
        float sum = 0.0;
            for (int k = 0; k < yc; k++) {
          /*
           * index (k,j) in Y transpose => (j,k) in Y
           */
          //sum += yvals[ INDEXc(i,k,yr,yc) ] * xvals[ INDEXc(j,k,xr,xc) ];
          sum += xvals.at( INDEXc(i,k,xr,xc) ) * yvals.at( INDEXc(j,k,yr,yc) );
            }
            //mmat[ INDEX(i,j,mr,mc) ] = sum;
            mmat.at( INDEXr(i,j,mr,mc) ) = sum;
        }
    }
  if (trace_progress()) {
    assert( this->notinf() ); assert( this->notnan() ); assert( this->normf()!=0.f );
    assert( x.notinf() ); assert( x.notnan() ); assert( x.normf()!=0.f );
```

```
    assert( m.notinf() ); assert( m.notnan() ); assert( m.normf()!=0.f );
  }
  if (trace_arrays()) {
    cout « "Outer2 left input:\n"; this->show();
    cout « "Outer2 right input:\n"; x.show();
    cout « "Outer2 output:\n"; m.show();
  }
}
```

# Index


accuracy
    Net, 38
activation_name
    Layer, 27
add
    Vector, 53
add_vector
    VectorBatch, 63
addh
    VectorBatch, 64
addLayer
    Net, 39
addvh
    Matrix, 28
allocate
    VectorBatch, 64
allocate_batch_specific_temporaries
    Layer, 22
    Net, 39
at
    Net, 39, 40
    VectorBatch, 65
axpy
    Matrix, 29

backlayer
    Net, 40
backPropagate
    Net, 40
backward
    Layer, 22
batch
    Dataset, 14
batch_size
    VectorBatch, 65

c
    Vector, 59
calculate_initial_delta
    Net, 41
calculateLoss
    Net, 42
Categorization, 7
    Categorization, 7, 8
    close_enough, 8
    normalize, 9
    probabilities, 9
    size, 9
close_enough
    Categorization, 8
colsize
    Matrix, 29
copy_from
    Vector, 54
    VectorBatch, 65
create_output_batch
    Net, 43

data
    dataItem, 12
    Dataset, 14
    Matrix, 29
    Vector, 54
    VectorBatch, 66
data_size
    dataItem, 11
    Dataset, 14
data_vals
    Dataset, 15
data_values
    dataItem, 11
dataItem, 10
    data, 12
    data_size, 11
    data_values, 11
    dataItem, 10, 11
    label, 12
    label_size, 11
    label_values, 11
Dataset, 12
    batch, 14
    data, 14
    data_size, 14
    data_vals, 15
    Dataset, 13, 14
    inputs, 15
    item, 15
    label_size, 16
    label_vals, 16
    labels, 16
    path, 20
    push_back, 17
    readTest, 17
    set_lowerbound, 18
    set_number, 18
    shuffle, 18
    size, 18
    split, 19
    stack, 19
    stacked_data_vals, 19

stacked_label_vals, 20
decay
Net, 43
display
VectorBatch, 66

extract_vector
VectorBatch, 67

feedForward
Net, 43, 44
forward
Layer, 23

get_col
VectorBatch, 67
get_row
VectorBatch, 68
get_vector
VectorBatch, 68
get_vectorObj
VectorBatch, 68

hadamard
VectorBatch, 68

info
Net, 45
input
Layer, 23, 24
input_size
Layer, 24
inputs
Dataset, 15
inputsize
Net, 45
intermediate
Layer, 24
item
Dataset, 15
item_size
VectorBatch, 69

label
dataItem, 12
label_size
dataItem, 11
Dataset, 16
label_vals
Dataset, 16
label_values
dataItem, 11
labels
Dataset, 16
Layer, 21
activation_name, 27
allocate_batch_specific_temporaries, 22
backward, 22
forward, 23
input, 23, 24
input_size, 24
intermediate, 24
Layer, 22
Net, 26
output_size, 24
set_activation, 24, 25
set_number, 25
set_topdelta, 25
set_uniform_biases, 25
set_uniform_weights, 26
update_dw, 26
layer
Net, 45
learning_rate
Net, 45
loadModel
Net, 45

Matrix, 27
addvh, 28
axpy, 29
colsize, 29
data, 29
Matrix, 28
meanv, 30
mmp, 30
mvp, 30
mvpt, 31
nelements, 31
normf, 31
notinf, 32
notnan, 32
operator*, 32, 36
operator+, 32
operator-, 33
operator/, 33, 36
operator=, 33
outerProduct, 34
rowsize, 34
show, 34
square, 34
transpose, 35
values, 35
Vector, 58
VectorBatch, 78
zeros, 35
meanh
VectorBatch, 69
meanv
Matrix, 30
mmp
Matrix, 30
momentum
Net, 46
mvp
Matrix, 30
mvpt
Matrix, 31

nelements
  Matrix, 31
  VectorBatch, 69
Net, 36
  accuracy, 38
  addLayer, 39
  allocate_batch_specific_temporaries, 39
  at, 39, 40
  backlayer, 40
  backPropagate, 40
  calculate_initial_delta, 41
  calculateLoss, 42
  create_output_batch, 43
  decay, 43
  feedForward, 43, 44
  info, 45
  inputsize, 45
  Layer, 26
  layer, 45
  learning_rate, 45
  loadModel, 45
  momentum, 46
  Net, 37, 38
  optimize, 51
  optimizer, 46
  outputsize, 46
  push_layer, 46
  RMSprop, 47
  saveModel, 47
  set_decay, 48
  set_learning_rate, 48
  set_lossfunction, 48
  set_momentum, 49
  set_optimizer, 49
  set_uniform_biases, 49
  set_uniform_weights, 49
  SGD, 50
  show, 50
  train, 50
normalize
  Categorization, 9
normf
  Matrix, 31
  VectorBatch, 70
notinf
  Matrix, 32
  VectorBatch, 70
notnan
  Matrix, 32
  VectorBatch, 70

operator∗
  Matrix, 32, 36
  Vector, 54, 58
  VectorBatch, 71, 78
operator+
  Matrix, 32
  Vector, 54
operator-
  Matrix, 33
  Vector, 55, 58
  VectorBatch, 71
operator/
  Matrix, 33, 36
  Vector, 55
  VectorBatch, 71, 78
operator/=
  Vector, 55
operator=
  Matrix, 33
  Vector, 56
  VectorBatch, 72
operator[]
  Vector, 56
optimize
  Net, 51
optimizer
  Net, 46
outer2
  VectorBatch, 72
outerProduct
  Matrix, 34
output_size
  Layer, 24
outputsize
  Net, 46

path
  Dataset, 20
positive
  Vector, 56
  VectorBatch, 72
probabilities
  Categorization, 9
push_back
  Dataset, 17
push_layer
  Net, 46

r
  Vector, 59
readTest
  Dataset, 17
resize
  VectorBatch, 73
RMSprop
  Net, 47
rowsize
  Matrix, 34

saveModel
  Net, 47
scaleby
  VectorBatch, 73
set_activation
  Layer, 24, 25
set_ax
  Vector, 56

set_batch_size
VectorBatch, 73
set_col
VectorBatch, 74
set_decay
Net, 48
set_item_size
VectorBatch, 74
set_learning_rate
Net, 48
set_lossfunction
Net, 48
set_lowerbound
Dataset, 18
set_momentum
Net, 49
set_number
Dataset, 18
Layer, 25
set_optimizer
Net, 49
set_row
VectorBatch, 75
set_topdelta
Layer, 25
set_uniform_biases
Layer, 25
Net, 49
set_uniform_weights
Layer, 26
Net, 49
set_vector
VectorBatch, 75
SGD
Net, 50
show
Matrix, 34
Net, 50
Vector, 57
VectorBatch, 75
shuffle
Dataset, 18
size
Categorization, 9
Dataset, 18
Vector, 57
VectorBatch, 76
split
Dataset, 19
square
Matrix, 34
Vector, 57
stack
Dataset, 19
stacked_data_vals
Dataset, 19
stacked_label_vals
Dataset, 20
train
Net, 50
transpose
Matrix, 35

update_dw
Layer, 26

v2mp
VectorBatch, 76
v2mtp
VectorBatch, 76
v2tmp
VectorBatch, 77
vals_vector
VectorBatch, 77, 78
values
Matrix, 35
Vector, 57
Vector, 52
add, 53
c, 59
copy_from, 54
data, 54
Matrix, 58
operator*, 54, 58
operator+, 54
operator-, 55, 58
operator/, 55
operator/=, 55
operator=, 56
operator[], 56
positive, 56
r, 59
set_ax, 56
show, 57
size, 57
square, 57
values, 57
Vector, 52, 53
VectorBatch, 59, 79
zeros, 58
VectorBatch, 60
add_vector, 63
addh, 64
allocate, 64
at, 65
batch_size, 65
copy_from, 65
data, 66
display, 66
extract_vector, 67
get_col, 67
get_row, 68
get_vector, 68
get_vectorObj, 68
hadamard, 68
item_size, 69
Matrix, 78

meanh, 69
nelements, 69
normf, 70
notinf, 70
notnan, 70
operator*, 71, 78
operator-, 71
operator/, 71, 78
operator=, 72
outer2, 72
positive, 72
resize, 73
scaleby, 73
set_batch_size, 73
set_col, 74
set_item_size, 74
set_row, 75
set_vector, 75
show, 75
size, 76
v2mp, 76
v2mtp, 76
v2tmp, 77
vals_vector, 77, 78
Vector, 59, 79
VectorBatch, 61, 62

zeros
Matrix, 35
Vector, 58