

## My Project

Generated by Doxygen 1.9.3



<b>1 Testdted on Ubuntu 20 and python&gt;=3.8</b>	<b>1</b>
1.0.1 Overview	1
1.0.1.1 Model specifications	1
1.0.2 Run these commands to set up the environment. We need conda to be installed. If not than you have to manually install the dependencies.	1
1.0.3 Run these commands to extract the MNIST dataset and to train the example network on it.	1
1.0.4 Run these commands to save the prediction to a sql database.	2
<b>2 Namespace Index</b>	<b>3</b>
2.1 Namespace List	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List	7
<b>5 Namespace Documentation</b>	<b>9</b>
5.1 ann.activation Namespace Reference	9
5.1.1 Detailed Description	9
5.2 ann.data Namespace Reference	9
5.2.1 Detailed Description	9
5.3 ann.layer Namespace Reference	9
5.3.1 Detailed Description	10
5.4 ann.loss Namespace Reference	10
5.4.1 Detailed Description	10
<b>6 Class Documentation</b>	<b>11</b>
6.1 ann.loss.Cross_Ent_Loss Class Reference	11
6.1.1 Detailed Description	11
6.1.2 Member Function Documentation	12
6.1.2.1 backward_loss() [1/2]	12
6.1.2.2 backward_loss() [2/2]	12
6.1.2.3 forward_loss() [1/2]	12
6.1.2.4 forward_loss() [2/2]	12
6.1.2.5 softmax() [1/2]	13
6.1.2.6 softmax() [2/2]	13
6.2 ann.layer.Linear Class Reference	13
6.2.1 Detailed Description	14
6.2.2 Member Function Documentation	14
6.2.2.1 backward_l() [1/2]	14
6.2.2.2 backward_l() [2/2]	14
6.2.2.3 forward_l() [1/2]	15
6.2.2.4 forward_l() [2/2]	15

6.2.2.5 reset_l() [1/2]	15
6.2.2.6 reset_l() [2/2]	15
6.2.2.7 update_l() [1/2]	16
6.2.2.8 update_l() [2/2]	16
6.3 ann.data.Mnist Class Reference	16
6.3.1 Detailed Description	17
6.3.2 Member Function Documentation	17
6.3.2.1 data() [1/2]	17
6.3.2.2 data() [2/2]	17
6.3.2.3 fetch() [1/2]	17
6.3.2.4 fetch() [2/2]	18
6.3.2.5 from_sql() [1/2]	18
6.3.2.6 from_sql() [2/2]	18
6.3.2.7 sql_database() [1/2]	18
6.3.2.8 sql_database() [2/2]	19
6.4 ann.example2l.Net Class Reference	19
6.4.1 Detailed Description	20
6.4.2 Constructor & Destructor Documentation	20
6.4.2.1 __init__() [1/2]	21
6.4.2.2 __init__() [2/2]	21
6.4.3 Member Function Documentation	21
6.4.3.1 backward() [1/2]	21
6.4.3.2 backward() [2/2]	22
6.4.3.3 evaluate() [1/2]	22
6.4.3.4 evaluate() [2/2]	22
6.4.3.5 forward() [1/2]	22
6.4.3.6 forward() [2/2]	23
6.4.3.7 loss() [1/2]	23
6.4.3.8 loss() [2/2]	23
6.4.3.9 reset() [1/2]	23
6.4.3.10 reset() [2/2]	23
6.4.3.11 train() [1/2]	24
6.4.3.12 train() [2/2]	24
6.4.3.13 update() [1/2]	24
6.4.3.14 update() [2/2]	24
6.5 ann.activation.Relu Class Reference	25
6.5.1 Detailed Description	25
6.5.2 Member Function Documentation	25
6.5.2.1 backward_a() [1/2]	25
6.5.2.2 backward_a() [2/2]	26
6.5.2.3 forward_a() [1/2]	26
6.5.2.4 forward_a() [2/2]	26

---

6.6 <a href="#">ann.activation.Sigmoid Class Reference</a> . . . . .	26
6.6.1 Detailed Description . . . . .	27
6.6.2 Member Function Documentation . . . . .	27
6.6.2.1 <a href="#">backward_a()</a> [1/2] . . . . .	27
6.6.2.2 <a href="#">backward_a()</a> [2/2] . . . . .	27
6.6.2.3 <a href="#">forward_a()</a> [1/2] . . . . .	28
6.6.2.4 <a href="#">forward_a()</a> [2/2] . . . . .	28



# Chapter 1

## Tesdted on Ubuntu 20 and python $\geq 3.8$

**1.0.0.0.1 This project is done for the CSE600A (Object Oriented Programming and Design) course requirement.**

### 1.0.1 Overview

1. In this work we implement a simple 2 layer feed forwad arrtificial neural netowork (ANN) from scratch using numpy only. Gradients are calculated locally and passed for downstream calculations similarly to this work: [http://cs231n.stanford.edu/slides/2021/discussion\\_2\\_backprop.pdf](http://cs231n.stanford.edu/slides/2021/discussion_2_backprop.pdf).
2. For the hierchy of different classes refer to the latex/refman.pdf file.
3. For the profiler file refer to the test/test.txt

#### 1.0.1.1 Model specifications

```
i_d = 784 # input dimension
m_d = 100 # no of cells in the mid layer
o_d = 10 # output dimension
epoch = 1 # we trainn for 1 epoch only
act = 'relu' # activation type
lr = 0.00001 # learning rate
X,Y,X_test,Y_test = train_features, train_label, test_features, test_label
```

1. example2l (2l > 2 layer) class has implements the training and testting of the 2 layer network.
2. We achieve an accuracy of 95 percent with these hyperparameters.

**1.0.2 Run these commands to set up the environment. We need conda to be installed. If not than you have to manually install the dependencies.**

```
conda env create -f environment.yml <br/>
conda activate oopd <br/>
python setup.py sdist bdist_wheel <br/>
pip install dist/ann-0.0.1-py3-none-any.whl
```

**1.0.3 Run these commands to extract the MNIST dataset and to train the example network on it.**

```
tar -xvf data.tar.gz <br/>
python run.py
```

#### 1.0.4 Run these commands to save the prediction to a sql database.

```
.mode csv <br/>
.import data/test_pred_true.csv oopd <br/>
.save data/oopd_pred_true.db
```



## Chapter 2

# Namespace Index

### 2.1 Namespace List

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

<a href="#">ann.activation</a>	9
<a href="#">ann.data</a>	9
<a href="#">ann.layer</a>	9
<a href="#">ann.loss</a>	10



## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

ann.loss.Cross_Ent_Loss . . . . .	11
ann.example2l.Net . . . . .	19
ann.example2l.Net . . . . .	19
ann.layer.Linear . . . . .	13
ann.example2l.Net . . . . .	19
ann.example2l.Net . . . . .	19
ann.data.Mnist . . . . .	16
ann.activation.Relu . . . . .	25
ann.example2l.Net . . . . .	19
ann.example2l.Net . . . . .	19
ann.activation.Sigmoid . . . . .	26
ann.example2l.Net . . . . .	19
ann.example2l.Net . . . . .	19



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">ann.loss.Cross_Ent_Loss</a>	11
<a href="#">ann.layer.Linear</a>	13
<a href="#">ann.data.Mnist</a>	16
<a href="#">ann.example2l.Net</a>	19
<a href="#">ann.activation.Relu</a>	25
<a href="#">ann.activation.Sigmoid</a>	26



## Chapter 5

# Namespace Documentation

### 5.1 ann.activation Namespace Reference

#### Classes

- class [Relu](#)
- class [Sigmoid](#)

#### 5.1.1 Detailed Description

```
@package docstring
This module contains the different activation functions used in AI.
```

### 5.2 ann.data Namespace Reference

#### Classes

- class [Mnist](#)

#### 5.2.1 Detailed Description

```
@package docstring
This module contains the different dataset download functions.
```

### 5.3 ann.layer Namespace Reference

#### Classes

- class [Linear](#)

### 5.3.1 Detailed Description

```
@package docstring  
This module contains all the different layers used in neural networks
```

## 5.4 ann.loss Namespace Reference

### Classes

- class [Cross\\_Ent\\_Loss](#)

### 5.4.1 Detailed Description

```
@package docstring  
This module contains all the los functions
```

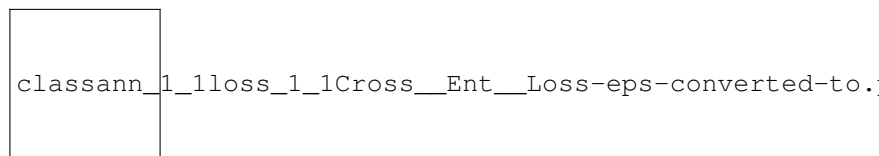


## Chapter 6

# Class Documentation

### 6.1 ann.loss.Cross\_Ent\_Loss Class Reference

Inheritance diagram for ann.loss.Cross\_Ent\_Loss:



#### Public Member Functions

- def [forward\\_loss](#) (self, logits, true)
- def [backward\\_loss](#) (self)
- def [softmax](#) (self, x)
- def [forward\\_loss](#) (self, logits, true)
- def [backward\\_loss](#) (self)
- def [softmax](#) (self, x)

#### Public Attributes

- **pred**
- **true**
- **z**

#### 6.1.1 Detailed Description

Applies the combined cross entropy and softmax loss function element-wise.  
For easier backprop calculation.

Args:  
None

## 6.1.2 Member Function Documentation

### 6.1.2.1 backward\_loss() [1/2]

```
def ann.loss.Cross_Ent_Loss.backward_loss (  
    self )
```

Implements the backprop calculation.

### 6.1.2.2 backward\_loss() [2/2]

```
def ann.loss.Cross_Ent_Loss.backward_loss (  
    self )
```

Implements the backprop calculation.

### 6.1.2.3 forward\_loss() [1/2]

```
def ann.loss.Cross_Ent_Loss.forward_loss (  
    self,  
    logits,  
    true )
```

Args:

logits: Predicted vector of shape [num\_classes,1].  
true: truth vector of shape [num\_classes,1].

### 6.1.2.4 forward\_loss() [2/2]

```
def ann.loss.Cross_Ent_Loss.forward_loss (  
    self,  
    logits,  
    true )
```

Args:

logits: Predicted vector of shape [num\_classes,1].  
true: truth vector of shape [num\_classes,1].

**6.1.2.5 softmax() [1/2]**

```
def ann.loss.Cross_Ent_Loss.softmax (
    self,
    x )
```

Implements the softmax dunction element wise.

**6.1.2.6 softmax() [2/2]**

```
def ann.loss.Cross_Ent_Loss.softmax (
    self,
    x )
```

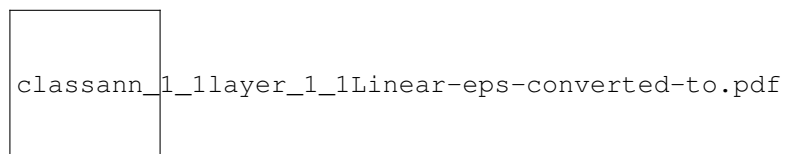
Implements the softmax dunction element wise.

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

- ann/loss.py
- build/lib/ann/loss.py

**6.2 ann.layer.Linear Class Reference**

Inheritance diagram for ann.layer.Linear:

**Public Member Functions**

- def `__init__` (self, i\_d, o\_d)
- def `forward_l` (self, x)
- def `backward_l` (self, x)
- def `update_l` (self, alpha)
- def `reset_l` (self)
- def `__init__` (self, i\_d, o\_d)
- def `forward_l` (self, x)
- def `backward_l` (self, x)
- def `update_l` (self, alpha)
- def `reset_l` (self)

## Public Attributes

- **weight**
- **bias**
- **weight\_grad**
- **bias\_grad**
- **x**

### 6.2.1 Detailed Description

This class implements the linear layer.

Args:

i\_d > input dimension  
o\_d > output dimension

Attributes:

weight: the leranable weight of the module.  
bias: the leranable bias of the module.  
  
weight\_grad: gradient for the weight matrix.  
bias\_grad: gradient for the bias matrix.

Examples:

```
>>> m = ann.Linear(10,20)
>>> inp = np.random.rand(i_d,1)
>>> out = m.forward(inp)
>>> out.shape
[o_d,1]
```

### 6.2.2 Member Function Documentation

#### 6.2.2.1 backward\_l() [1/2]

```
def ann.layer.Linear.backward_l (
    self,
    x )
```

Implements the Backprop calculations given the gradeints.

#### 6.2.2.2 backward\_l() [2/2]

```
def ann.layer.Linear.backward_l (
    self,
    x )
```

Implements the Backprop calculations given the gradeints.

### 6.2.2.3 forward\_l() [1/2]

```
def ann.layer.Linear.forward_l (
    self,
    x )
```

Implements the forward calculation given an input.

Input: matrix with shape [dim,1]  
ouput: matrix with shape [o\_d,1]

### 6.2.2.4 forward\_l() [2/2]

```
def ann.layer.Linear.forward_l (
    self,
    x )
```

Implements the forward calculation given an input.

Input: matrix with shape [dim,1]  
ouput: matrix with shape [o\_d,1]

### 6.2.2.5 reset\_l() [1/2]

```
def ann.layer.Linear.reset_l (
    self )
```

To reset th gradients to zero.

### 6.2.2.6 reset\_l() [2/2]

```
def ann.layer.Linear.reset_l (
    self )
```

To reset th gradients to zero.

### 6.2.2.7 `update_l()` [1/2]

```
def ann.layer.Linear.update_l (
    self,
    alpha )
```

To update the Gradients of each layer.

### 6.2.2.8 `update_l()` [2/2]

```
def ann.layer.Linear.update_l (
    self,
    alpha )
```

To update the Gradients of each layer.

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

- `ann/layer.py`
- `build/lib/ann/layer.py`

## 6.3 `ann.data.Mnist` Class Reference

### Public Member Functions

- `def __init__ (self, path)`
- `def from_sql (self, train_db='oopd_train.db', test_db='oopd_test.db')`
- `def data (self)`
- `def fetch (self, url)`
- `def sql_database (self, path_to_db)`
- `def __init__ (self, path)`
- `def from_sql (self, train_db='oopd_train.db', test_db='oopd_test.db')`
- `def data (self)`
- `def fetch (self, url)`
- `def sql_database (self, path_to_db)`

### Public Attributes

- `path`

### 6.3.1 Detailed Description

Download the MNIST dataset and save them in a given path.

Args:

path: where to save the dataset.

Return:

X: training input [60000,784]

Y: training output [60000,1]

X\_test = test input [10000,784]

Y\_test = test output [10000,1]

### 6.3.2 Member Function Documentation

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

```
def ann.data.Mnist.data (  
    self )
```

Download the data from source

Save it to the disk in self.path/ directory

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

```
def ann.data.Mnist.data (  
    self )
```

Download the data from source

Save it to the disk in self.path/ directory

#### 6.3.2.3 fetch() [1/2]

```
def ann.data.Mnist.fetch (  
    self,  
    url )
```

Downloads the MNIST data given the url.

#### 6.3.2.4 `fetch()` [2/2]

```
def ann.data.Mnist.fetch (
    self,
    url )
```

Downloads the MNIST data given the url.

#### 6.3.2.5 `from_sql()` [1/2]

```
def ann.data.Mnist.from_sql (
    self,
    train_db = 'oopd_train.db',
    test_db = 'oopd_test.db' )
```

Upload the load from a sql database

#### 6.3.2.6 `from_sql()` [2/2]

```
def ann.data.Mnist.from_sql (
    self,
    train_db = 'oopd_train.db',
    test_db = 'oopd_test.db' )
```

Upload the load from a sql database

#### 6.3.2.7 `sql_database()` [1/2]

```
def ann.data.Mnist.sql_database (
    self,
    path_to_db )
```

Read the data from a sql database.



### 6.3.2.8 sql\_database() [2/2]

```
def ann.data.Mnist.sql_database (
    self,
    path_to_db )
```

Read the data from a sql database.

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

- ann/data.py
- build/lib/ann/data.py

## 6.4 ann.example2l.Net Class Reference

Inheritance diagram for ann.example2l.Net:



### Public Member Functions

- def `__init__` (self, i\_d, m\_d, o\_d, act, lr, epoch, X, Y, X\_test, Y\_test)
- def `forward` (self, x)
- def `backward` (self)
- def `update` (self)
- def `reset` (self)
- def `loss` (self, pred, index)
- def `evaluate` (self, X=None, Y=None, save=True)
- def `train` (self, step\_u=1, step=100000, test=False)
- def `__init__` (self, i\_d, m\_d, o\_d, act, lr, epoch, X, Y, X\_test, Y\_test)

- def `forward` (self, x)
- def `backward` (self)
- def `update` (self)
- def `reset` (self)
- def `loss` (self, pred, index)
- def `evaluate` (self, X=None, Y=None, save=True)
- def `train` (self, step\_u=1, step=100000, test=False)

## Public Attributes

- `alpha`
- `epoch`
- `layer1`
- `act1`
- `layer2`
- `cross_ent_loss`
- `Y`
- `Y_test`
- `X_test`

### 6.4.1 Detailed Description

Create a sample 2 layer feed forward artificial neural network (ANN).

Args:

```
i_d: input dimension.
m_d: number of cells in 1st layer.
o_d: output dimension.
act: activation function to use.
lr: Learning rate
epoch: number of epochs to train.
X: training input [60000,784].
Y: training output [60000,1].
X_test = test input [10000,784].
Y_test = test output [10000,1].
```

Returns:

```
None
```

Example:

```
i_d, m_d, o_d = 784, 100, 10
act = 'relu'
lr, epoch = 0.00001, 10
X, Y, X_test, Y_test = data.Mnist(path='data').data()
net = example21.Net(i_d, m_d, o_d, act, lr, epoch, X, Y, X_test, Y_test)
net.train()
```

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 `__init__()` [1/2]

```
def ann.example21.Net.__init__ (
    self,
    i_d,
    m_d,
    o_d,
    act,
    lr,
    epoch,
    X,
    Y,
    X_test,
    Y_test )
```

Reimplemented from [ann.layer.Linear](#).

#### 6.4.2.2 `__init__()` [2/2]

```
def ann.example21.Net.__init__ (
    self,
    i_d,
    m_d,
    o_d,
    act,
    lr,
    epoch,
    X,
    Y,
    X_test,
    Y_test )
```

Reimplemented from [ann.layer.Linear](#).

### 6.4.3 Member Function Documentation

#### 6.4.3.1 `backward()` [1/2]

```
def ann.example21.Net.backward (
    self )
```

Calls `backward` function for each layer to calculate the gradients.

#### 6.4.3.2 backward() [2/2]

```
def ann.example21.Net.backward (
    self )
```

Calls backprop fucntion for each layer to calculate the gradients.

#### 6.4.3.3 evaluate() [1/2]

```
def ann.example21.Net.evaluate (
    self,
    X = None,
    Y = None,
    save = True )
```

Evaulate the created model given the test data.

#### 6.4.3.4 evaluate() [2/2]

```
def ann.example21.Net.evaluate (
    self,
    X = None,
    Y = None,
    save = True )
```

Evaulate the created model given the test data.

#### 6.4.3.5 forward() [1/2]

```
def ann.example21.Net.forward (
    self,
    x )
```

Calls the forward fucntion for each layer.

#### 6.4.3.6 forward() [2/2]

```
def ann.example2l.Net.forward (
    self,
    x )
```

Calls the forward fucntion for each layer.

#### 6.4.3.7 loss() [1/2]

```
def ann.example2l.Net.loss (
    self,
    pred,
    index )
```

Calculate the cross entropy loss.

#### 6.4.3.8 loss() [2/2]

```
def ann.example2l.Net.loss (
    self,
    pred,
    index )
```

Calculate the cross entropy loss.

#### 6.4.3.9 reset() [1/2]

```
def ann.example2l.Net.reset (
    self )
```

Calls the reset fucntion of each layer to reset the gradients back to 0.

#### 6.4.3.10 reset() [2/2]

```
def ann.example2l.Net.reset (
    self )
```

Calls the reset fucntion of each layer to reset the gradients back to 0.

**6.4.3.11 train() [1/2]**

```
def ann.example2l.Net.train (
    self,
    step_u = 1,
    step = 100000,
    test = False )
```

Train the created model given the input.

**6.4.3.12 train() [2/2]**

```
def ann.example2l.Net.train (
    self,
    step_u = 1,
    step = 100000,
    test = False )
```

Train the created model given the input.

**6.4.3.13 update() [1/2]**

```
def ann.example2l.Net.update (
    self )
```

Calls the update fucntion of each layer to update the gradients.

**6.4.3.14 update() [2/2]**

```
def ann.example2l.Net.update (
    self )
```

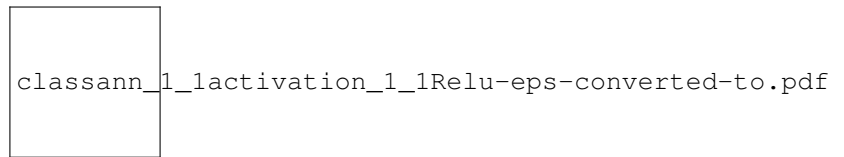
Calls the update fucntion of each layer to update the gradients.

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

- ann/example2l.py
- build/lib/ann/example2l.py

## 6.5 ann.activation.Relu Class Reference

Inheritance diagram for ann.activation.Relu:



### Public Member Functions

- def [forward\\_a](#) (self, x)
- def [backward\\_a](#) (self, x)
- def [forward\\_a](#) (self, x)
- def [backward\\_a](#) (self, x)

### Public Attributes

- **z**

### 6.5.1 Detailed Description

Applies the rectified linear unit function element-wise.

Args:  
None

### 6.5.2 Member Function Documentation

#### 6.5.2.1 [backward\\_a\(\)](#) [1/2]

```
def ann.activation.Relu.backward_a (  
    self,  
    x )
```

Implements the backprop calculation.

**6.5.2.2 backward\_a()** [2/2]

```
def ann.activation.Relu.backward_a (
    self,
    x )
```

Implements the backprop calculation.

**6.5.2.3 forward\_a()** [1/2]

```
def ann.activation.Relu.forward_a (
    self,
    x )
```

Implements the Forward calculation given an input.

**6.5.2.4 forward\_a()** [2/2]

```
def ann.activation.Relu.forward_a (
    self,
    x )
```

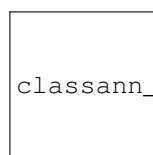
Implements the Forward calculation given an input.

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

- ann/activation.py
- build/lib/ann/activation.py

**6.6 ann.activation.Sigmoid Class Reference**

Inheritance diagram for ann.activation.Sigmoid:



classann\_1\_1activation\_1\_1Sigmoid-eps-converted-to.pdf



## Public Member Functions

- def `forward_a` (self, x)
- def `backward_a` (self, x)
- def `forward_a` (self, x)
- def `backward_a` (self, x)

## Public Attributes

- `z`

### 6.6.1 Detailed Description

Applies the sigmoid function element-wise.

Args:  
None

### 6.6.2 Member Function Documentation

#### 6.6.2.1 `backward_a()` [1/2]

```
def ann.activation.Sigmoid.backward_a (  
    self,  
    x )
```

Implements the backprop calculation.

#### 6.6.2.2 `backward_a()` [2/2]

```
def ann.activation.Sigmoid.backward_a (  
    self,  
    x )
```

Implements the backprop calculation.

### 6.6.2.3 forward\_a() [1/2]

```
def ann.activation.Sigmoid.forward_a (  
    self,  
    x )
```

Implements the Forward calculation given an input.

### 6.6.2.4 forward\_a() [2/2]

```
def ann.activation.Sigmoid.forward_a (  
    self,  
    x )
```

Implements the Forward calculation given an input.

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

- ann/activation.py
- build/lib/ann/activation.py