My Project

Generated by Doxygen 1.9.3

Chapter 1

README

This is a simple example of 2 layer feed forwad ANN.

1.0.0.1 Creating the wheel files.

1. Run python setup.py sdist bdist_wheel
2. pip install dist/ann-0.0.1-py3-none-any.whl"

1.0.0.2 Creating the environment to be used.

1. conda env create -f environment.yml
2. conda activate oopd

1.0.0.3 Training the example 2 layerd net.

1. create a folder named data/
2. python run.py

Chapter 2

Namespace Index

2.1 Namespace List

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

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

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

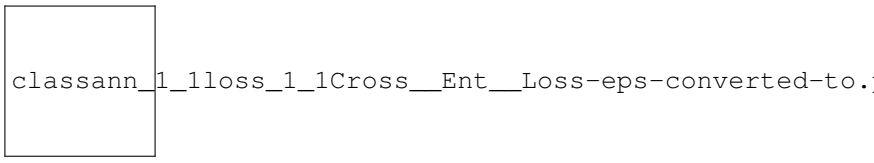
```
@package docstring
This module contains all the los functions
```

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)

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()

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

Implements the backprop calculation.

6.1.2.2 forward_loss()

```
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].

6.1.2.3 softmax()

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

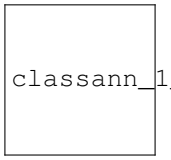
Implements the softmax dunction element wise.

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

- 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)

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()

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

```
Implements the  Backprop calculations given the gradeints.
```

6.2.2.2 forward_l()

```
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]
```

6.2.2.3 reset_l()

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

```
To reset th gradients to zero.
```

6.2.2.4 update_l()

```
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 file:

- ann/layer.py

6.3 ann.data.Mnist Class Reference

Public Member Functions

- def **__init__** (self, path)
- def **data** (self)
- def fetch (self, url)

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 fetch()

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

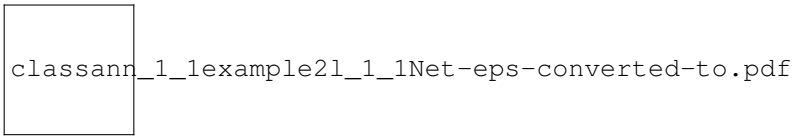
```
Downloads the MNIST data given the url.
```

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

- 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)
- def train (self, step_u=1, step=100000)

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 artitifical 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 = example2l.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__()

```
def ann.example2l.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()

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

Calls backprop fucntion for each layer to calculate the gradients.

6.4.3.2 evaluate()

```
def ann.example2l.Net.evaluate (
              self,
              X = None,
              Y = None )
```

Evaulate the created model given the test data.

6.4.3.3 forward()

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

Calls the forward fucntion for each layer.

6.4.3.4 loss()

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

Calculate the cross entropy loss.

6.4.3.5 reset()

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

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

6.4.3.6 train()

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

Train the created model given the input.

6.4.3.7 update()

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

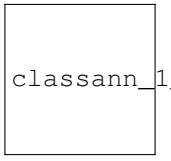
Calls the update fucntion of each layer to update the gradients.

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

- 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)

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()

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

Implements the backprop calculation.
```

6.5.2.2 forward_a()

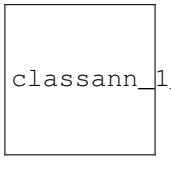
```
def ann.activation.Relu.forward_a (
              self,
              x )

Implements the Forward calculation given an input.
```

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

- ann/activation.py

6.6 ann.activation.Sigmoid Class Reference

Inheritance diagram for ann.activation.Sigmoid:



Public Member Functions

- 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()

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

```
Implements the backprop calculation.
```

6.6.2.2 forward_a()

```
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 file:

- ann/activation.py