NeuralNetwork_lib Documentation

Release alpha

Erik Hammon

May 17, 2018

Neural Networks:

CHAPTER 1

Perceptron

1.1 Initializing a Perceptron

```
int num_inputs = 2;
int num_outputs = 1;
Perceptron pnn = new Perceptron(num_inputs, num_outputs);
```

1.2 Feeding Data through a Perceptron and receiving the Output

```
float[] inputs = new float[] {1, 0};
float[] outputs = pnn.feedforward(inputs);
System.out.println(outputs);
// gives : [0.5345]
```

1.3 Training a Perceptron

```
float[] answers = new float[] {1};
pnn.train_gradient_descent(inputs, answers);

System.out.println(pnn.feedforward(inputs));
// gives : [0.98799]
```

1.3.1 use momentum for training

considered as a faster way of training

```
float[] answers = new float[] {1};
nnn.train_momentum_gradient_descent(inputs, answers);

System.out.println(nnn.feedforward(inputs));
// gives : [0.98799]
```

1.4 Setting extra Variables

```
nnn.set_learning_rate(float);
nnn.set_momentum_rate(float);
nnn.set_print_interval(int);
```

CHAPTER 2

Neural Network

2.1 Initializing a Neural Network

```
int num_inputs = 2;
int[] num_hidden = new int[] {4, 3};
int num_outputs = 1;
NeuralNetwork nn = new NeuralNetwork(num_inputs, num_hidden, um_outputs);
```

2.2 Feeding Data through a Neural Network and receiving the Output

```
float[] inputs = new float[] {1, 0};
float[] outputs = nn.feedforward(inputs);
System.out.println(outputs);
// gives : [0.5345]
```

2.3 Training a Neural Network

```
float[] answers = new float[] {1};
nn.train_gradient_descent(inputs, answers);

System.out.println(nn.feedforward(inputs));
// gives : [0.98799]
```

2.3.1 use momentum for training

considered as a faster way of training

```
float[] answers = new float[] {1};
nn.train_momentum_gradient_descent(inputs, answers);

System.out.println(nn.feedforward(inputs));
// gives : [0.98799]
```

2.4 Setting extra Variables

```
nn.set_learning_rate(float);
nn.set_momentum_rate(float);
nn.set_print_interval(int);
```

CHAPTER 3

Genetic Perceptron

3.1 Initializing a Genetic Perceptron

```
int num_inputs = 2;
int num_outputs = 1;
int population_size = 100;
int random_per_generation = 10;
Genetic_Perceptron gpnn = new Genetic_Perceptron(population_size, random_per_
↪generation, num_inputs, num_outputs);
```

3.2 Feeding Data through a Genetic Perceptron and receiving the Output

```
float[] inputs = new float[] {1, 0};
float[] outputs = gpnn.overall_best.feedforward(inputs);
System.out.println(outputs);
// gives : [0.5345]
```

3.3 Training a Genetic Perceptron

```
float[] answers = new float[] {1};
int total_generations = 1000;
for (int generation = 0; generation < total_generations; generation++) {
  Perceptron[] current_generation = gpnn.get_current_generation();
  float[] current_fitness = new float[current_generation.length];
  for (int aspect = 0; aspect < current_generation.length; aspect++) {
    current_fitness[aspect] = answers[0] - current_generation[aspect].
↪feedforward(inputs));
```

(continues on next page)

(continued from previous page)

```
  }
  gpnn.evolve_best(current_fitness);
}

System.out.println(gpnn.overall_best.feedforward(inputs));
// gives : [0.98799]
```

3.4 Setting extra Variables

```
gpnn.set_mutation_rate(float);
gpnn.set_evolution_rate(float);
gpnn.set_offspring_mutation_rate(float);
```

CHAPTER 4

Genetic Neural Network

4.1 Initializing a Genetic Neural Network

```
int num_inputs = 2;
int[] num_hidden = new int[] {4, 3};
int num_outputs = 1;
int population_size = 100;
int random_per_generation = 10;
Genetic_NeuralNetwork gnn = new Genetic_NeuralNetwork(population_size, random_per_
↪generation, num_inputs, num_hidden, num_outputs);
```

4.2 Feeding Data through a Genetic Neural Network and receiving the Output

```
float[] inputs = new float[] {1, 0};
float[] outputs = gnn.overall_best.feedforward(inputs);
System.out.println(outputs);
// gives : [0.5345]
```

4.3 Training a Genetic Neural Network

```
float[] answers = new float[] {1};
int total_generations = 1000;
for (int generation = 0; generation < total_generations; generation++) {
  NeuralNetwork[] current_generation = gnn.get_current_generation();
  float[] current_fitness = new float[current_generation.length];
  for (int aspect = 0; aspect < current_generation.length; aspect++) {
```

(continues on next page)

(continued from previous page)

```
    current_fitness[aspect] = answers[0] - current_generation[aspect].
↪feedforward(inputs));
  }
  gnn.evolve_best(current_fitness);
}

System.out.println(gnn.overall_best.feedforward(inputs));
// gives : [0.98799]
```

4.4 Setting extra Variables

```
gnn.set_mutation_rate(float);
gnn.set_evolution_rate(float);
gnn.set_offspring_mutation_rate(float);
```

CHAPTER 5

Convolutional Neural Network

Danger: DO NOT USE!!! NOT WORKING!!!

5.1 Initializing a Convolutional Neural Network

Danger: DO NOT USE!!! NOT WORKING!!!

```
int input_img_channels = 3;
int input_img_height = 16;
int input_img_width = 16;
int[] input_structure = int[] {input_img_channels, input_img_height, input_img_width};

int num_conv_layers = 2;
int num_conv_nodes = 100;
int filter_size = 3;
int[] conv_block = int[] {num_conv_layers, num_conv_nodes, filter_size};
int[][] conv_blocks = int[][] {conv_block, conv_block};

int num_hidden_1 = 12;
int num_hidden_2 = 9;
int num_outputs = 3;
int[] fully_connected_layer = int[] {num_hidden_1, num_hidden_2, num_outputs};

ConvolutionalNeuralNetwork cnn = new ConvolutionalNeuralNetwork(input_structure, conv_
↪blocks, fully_connected_layer);
```

5.2 Feeding Data through a Convolutional Neural Network and receiving the Output

Danger: DO NOT USE!!! NOT WORKING!!!

```
float[][] red_channel_img = // load image r
float[][] green_channel_img = // load image g
float[][] blue_channel_img = // load image b
float[][][] inputs = new float[][][] {red_channel_img, green_channel_img, blue_
↪channel_img};
float[] outputs = cnn.feedforward(inputs);
System.out.println(outputs);
// gives : [0.5345, 0.4325, 0.6578]
```

5.3 Training a Convolutional Neural Network

Danger: DO NOT USE!!! NOT WORKING!!!

```
float[] answers = new float[] {1, 0, 0};
cnn.train_gradient_descent(inputs, answers);

System.out.println(cnn.feedforward(inputs));
// gives : [0.98799, 0.1203, 0.0265]
```

5.4 Setting extra Variables

Danger: DO NOT USE!!! NOT WORKING!!!

```
cnn.set_learning_rate(float);
```

CHAPTER 6

Saving and Loading

Warning: save() may throw an IOException

6.1 Perceptron

```
String path = "path\\to\\file.nn";
Perceptron pnn = Load.Load_Perceptron(path);
pnn.save(path);
```

6.2 Neural Network

```
String path = "path\\to\\file.nn";
NeuralNetwork nn = Load.Load_NeuralNetwork(path);
nn.save(path);
```

6.3 Genetic Perceptron

Important: It is only possible to save the best Perceptron and the Genetic_Perceptron can not be reloaded

```
String path = "path\\to\\file.nn";
pnn.overall_best.save(path);
```

6.4 Genetic Neural Network

Important: It is only possible to save the best Neural Network and the Genetic_NeuralNetwork can not be reloaded

```
String path = "path\\to\\file.nn";
nn.overall_best.save(path);
```

6.5 Convolutional Neural Network

Danger: Convolutional Neural Networks are not yet supported in this

CHAPTER 7

Download

Important: There is no official Download link yet, because the Library is still in its alpha phase. If you want to test it, email me at neuralnetworklib@gmail.com to get the .jar. Thanks

CHAPTER 8

About

NeuralNetwork_lib is a Java Library made to easily construct Neural Networks and later on even Convolutional Neural networks and LSTM's.

It is a project by Erik Hammon who tries to gain more knowledge about the inner workings of Neural networks by building this Library.