
OCR

Trains a multi-layer perceptron (MLP) neural network to perform optical character recognition (OCR).

The training set is automatically generated using a heavily modified version of the captcha-generator node-captcha. Support for the MNIST handwritten digit database has been added recently (see performance section).

The network takes a one-dimensional binary array (default $20 \times 20 = 400$ -bit) as input and outputs an 10-bit array of probabilities, which can be converted into a character code. Initial performance measurements show promising success rates.

After training, the network is saved as a standalone module to `./ocr.js`, which can then be used in your project like this (from `test.js`):

```
1 var predict = require('./ocr.js');
2
3 // a binary array that we want to predict
4 var one = [
5   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
6   0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
7   0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
8   0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
9   0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
10  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
11  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
12  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
13  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
14  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
15  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
16  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
17  0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
18  0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
19  0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
20  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
21 ];
22
23 // the prediction is an array of probabilities
24 var prediction = predict(one);
25
26 // the index with the maximum probability is the best guess
27 console.log('prediction:', prediction.indexOf(Math.max.apply(null,
28   prediction)));
28 // will hopefully output 1 if trained with 0-9 :)
```

Usage

Clone this repository. The script is using canvas, so you'll need to install the **Cairo** rendering engine. On OS X, assuming you have Homebrew installed, this can be done with the following (copied from canvas README):

```
1 $ brew install pkg-config cairo jpeg giflib
```

Then install npm dependencies and test it:

```
1 $ npm install
2 $ node main.js
3 $ node test.js
```

Performance

All runs below were performed with a MacBook Pro Retina 13" Early 2015 with 8GB RAM.

MNIST [0-9]

To test with the MNIST dataset: click on the title above, download the 4 data files and put them in a folder called `mnist` in the root directory of this repository.

```
1 // config.json
2 {
3   "mnist": true,
4   "network": {
5     "hidden": 160,
6     "learning_rate": 0.03
7   }
8 }
```

Then run

```
1 $ node mnist.js
```

- **Neurons**
 - 400 input
 - 160 hidden
 - 10 output
- **Learning rate:** 0.03
- **Training set:** 60000 digits

-
- **Testing set:** 10000 digits
 - **Training time:** 21 min 53 s 753 ms
 - **Success rate:** 95.16%

[A-Za-z0-9]

```
1 // config.json
2 {
3   "mnist": false,
4   "text": "
5     abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012356789",
6   "fonts": [
7     "sans-serif",
8     "serif"
9   ],
10  "training_set": 2000,
11  "testing_set": 1000,
12  "image_size": 16,
13  "threshold": 400,
14  "network": {
15    "hidden": 60,
16    "learning_rate": 0.1,
17    "output": 62
18  }
19 }
```

- **Neurons**
 - 256 input
 - 60 hidden
 - 62 output
- **Learning rate:** 0.03
- **Training set**
 - **Size:** 124000 characters
 - **Sample:** a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9
- **Testing set:** 62000 characters
- **Training time:** 8 min 18 s 560 ms
- **Success rate:** 93.58225806451614%

[a-z]

```
1 // config.json
2 {
3   "mnist": false,
4   "text": "abcdefghijklmnopqrstuvwxyz",
5   "fonts": [
6     "sans-serif",
7     "serif"
8   ],
9   "training_set": 2000,
10  "testing_set": 1000,
11  "image_size": 16,
12  "threshold": 400,
13  "network": {
14    "hidden": 40,
15    "learning_rate": 0.1,
16    "output": 26
17  }
18 }
```

- **Neurons**

- 256 input
- 40 hidden
- 26 output

- **Learning rate:** 0.1

- **Training set**

- **Size:** 52000 characters
- **Sample:** a b c d e f g h i j k l m n o p q r s t u v w x y z

- **Testing set:** 26000 characters

- **Training time:** 1 min 55 s 414 ms

- **Success rate:** 93.83846153846153%

[0-9]

```
1 // config.json
2 {
3   "mnist": false,
4   "text": "0123456789",
5   "fonts": [
6     "sans-serif",
7     "serif"
8   ],
9   "training_set": 2000,
```

```
10  "testing_set": 1000,
11  "image_size": 16,
12  "threshold": 400,
13  "network": {
14    "hidden": 40,
15    "learning_rate": 0.1
16  }
17 }
```

- **Neurons**

- 256 input
- 40 hidden
- 10 output

- **Learning rate:** 0.1

- **Training set**

- **Size:** 20000 digits
- **Sample:** 0 1 2 3 4 5 6 7 8 9

- **Testing set:** 10000 digits

- **Training time:** 0 min 44 s 363 ms

- **Success rate:** 99.59%

Configuration

Tweak the network for your needs by editing the `config.json` file located in the main folder. Pasted below is the default config file.

```
1  // config.json
2  {
3    "mnist": false,
4    "text": "0123456789",
5    "fonts": [
6      "sans-serif",
7      "serif"
8    ],
9    "training_set": 2000,
10   "testing_set": 1000,
11   "image_size": 16,
12   "threshold": 400,
13   "network": {
14     "hidden": 40,
15     "learning_rate": 0.1
16   }
17 }
```

- **mnist**

- If set to true, the MNIST handwritten digit dataset will be used for training and testing the network. This setting will overwrite configured set sizes and will ignore the `image_size`, `threshold`, `fonts` and `text` settings.

- **text**

- A string containing the glyphs with which to train/test the network.

- **fonts**

- An array of fonts to be used when generating images.

- **training_set**

- Number of images to be generated and used as the network training set.

- **testing_set**

- Same as above, but these images are used for testing the network.

- **image_size**

- The size of the square chunk (in pixels) containing a glyph. The resulting network input size is `image_size^2`.

- **threshold**

- When analyzing the pixels of a glyph, the algorithm reduces each pixel (`r`, `g`, `b`) to (`r + g + b`) and everything below `threshold` is marked as 1 in the resulting binary array used as network input.

- **network**

- **hidden**

- ★ The size (number of neurons) of the hidden layer of the network.

- **learning_rate**

- ★ The learning rate of the network.