CS 5600/6600: F20: Intelligent Systems

Visual Pattern Recognition

Vladimir Kulyukin
Department of Computer Science
Utah State University

Motivation

The difficulty of visual pattern recognition becomes apparent if we attempt to write a computer program to recognize handwritten digits.

Simple intuitions about how we recognize shapes are hard to express algorithmically.

Many attempts to formalize shape recognitions have failed and fail due to multitudes of exceptions and special cases.

ANN Approach

Neural network engineers approach a problem as follows: take a large number of data samples, e.g., pictures of handwritten digits and develop a system that can be trained on these data samples.

Instead of encoding rules as algorithms, neural network engineers design systems that can learn rules automatically from training examples.

Increasing the number of training examples typically enable neural networks to improve its accuracy.

Recognizing Handwritten Digits

Handwritten digit recognition is an excellent prototype problem for learning neural networks.

The problem (although it is considered to be a toy problem by many in AI) is nonetheless challenging and provides an excellent opportunity to learn how neural networks work, because the problem is not as difficult as other problems tackled by neural networks.

This problem allows us to start from very basic neural networks and gradually improve them with more advanced techniques and transfer these ideas to other computer vision domains.

MNIST Database of Handwritten Digits

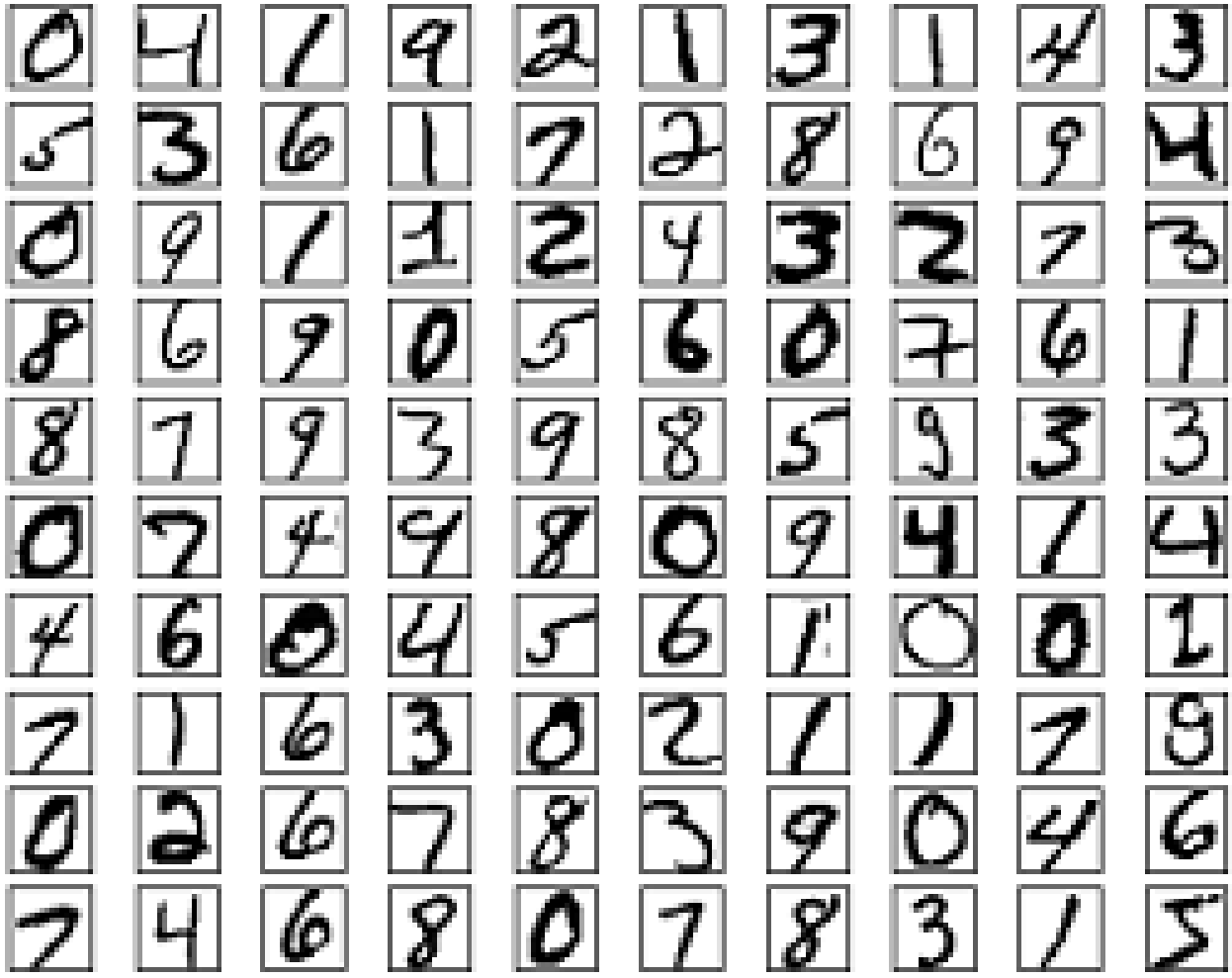
MNIST (Modified National Institute of Standards and Technology) database is a database of handwritten digits commonly used for training ANN. It is a commonly accepted benchmark.

MNIST database contains 60K training images and 10K testing images. In 2017, MNIST was extended to EMNIST which contains 240K training images and 40K testing images of handwritten digits.

The best error rate on MNIST has varied from 0.27 to 0.21.

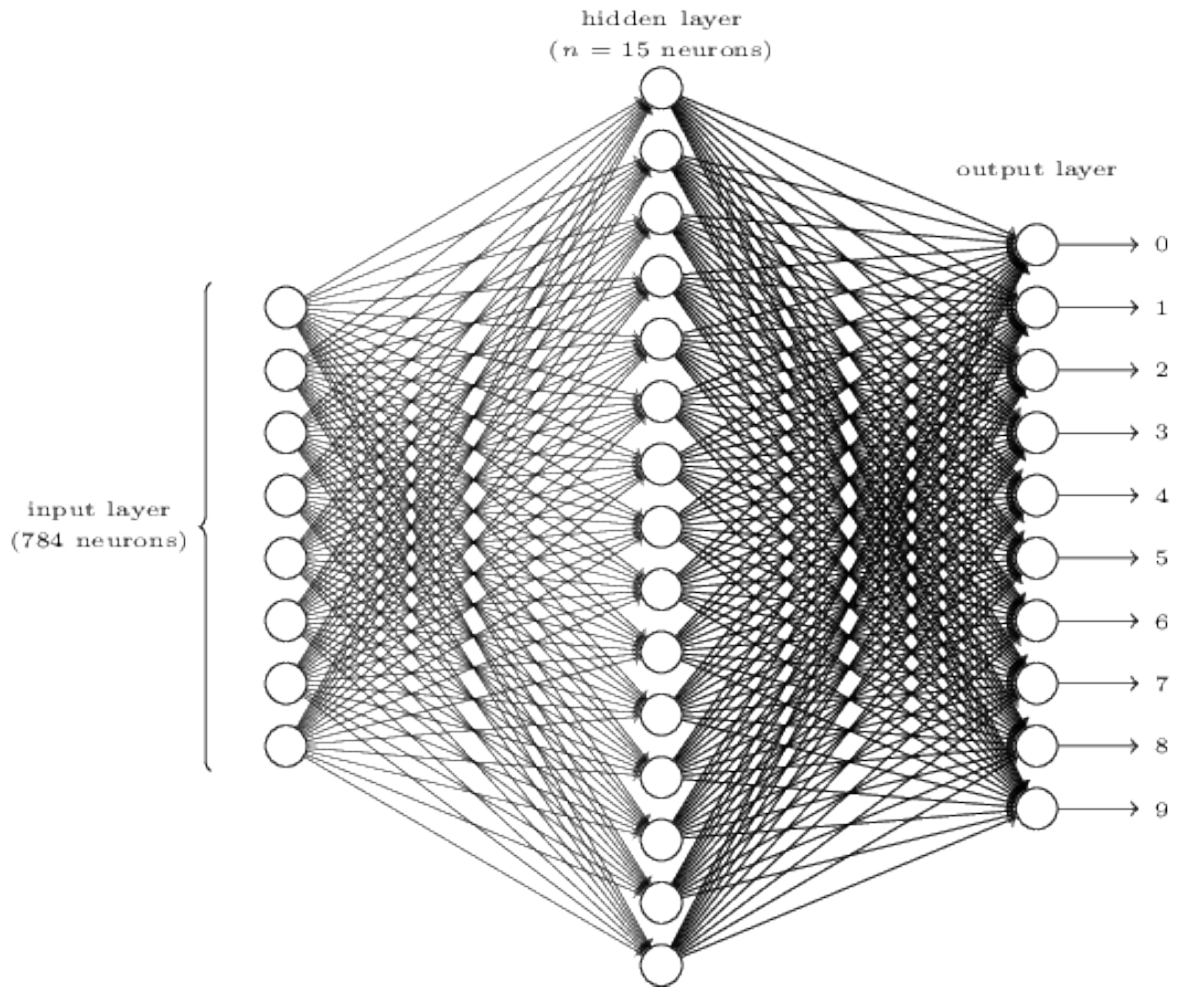
The models have been used on MNIST so far: logistic regression, KNN, SVMs, ANNs (shallow and deep), and ConvNets.

MNIST Database of Handwritten Digits



Example

Suppose that we want to design and train an ANN to recognize handwritten digits in 28×28 grayscale images. Here is one possible ANN.



Loading MNIST Data

The file `mnist_loader.py` (you'll get the source code) contains two functions `load_data()` and `load_data_wrapper()`.

The function `load_data` loads raw MNIST data.

The function `load_data_wrapper` takes raw MNIST data and converts it to the format appropriate for NN training.

Exploring Raw MNIST Data

```
>>> raw_train_d, raw_valid_d, raw_test_d = load_data()
## raw_train_d, raw_valid_d, raw_test_d are
## 2-tuples of numpy arrays.
>>> type(raw_train_d)
<type 'tuple'>
>>> raw_train_d[0].shape
(50000, 784)
>>> raw_train_d[1].shape
(50000,)
>>> raw_valid_d[0].shape
(10000, 784)
>>> raw_valid_d[1].shape
(10000,)
>>> raw_test_d[0].shape
(10000, 784)
>>> raw_test_d[1].shape
(10000,)
```

Accessing Raw MNIST Images and Their Targets

```
## This is raw image 0
>>> raw_train_d[0][0]
array([ 0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,
        ...
        0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.01171875,
        0.0703125 ,  0.4921875 ,  0.53125   ,
        ...
        0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,  0.],
        dtype=float32)
## this is the target of image 0
>>> raw_train_d[1][0]
5
```

Accessing Raw MNIST Images and Their Targets

```
## This is raw image 3
>>> raw_train_d[0][3]
array([ 0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,
        ...
        0.99609375,  0.24609375,  0.        ,
        0.        ,  0.        ,  0.        ,
        0.375     ,  0.953125  ,  0.98046875,
        ...
        0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,
        0.        ,  0.        ,  0.        ,],
        dtype=float32)
## this is the target of image 3
>>> raw_train_d[1][3]
3
```

Exploring Processed MNIST Data

```
## p_ - stands for processed.
>>> p_train_d, p_valid_d, p_test_d = load_data_wrapper()
>>> type(p_train_d)
<type 'list'>
>>> type(p_valid_d)
<type 'list'>
>>> type(p_test_d)
<type 'list'>
>>> type(p_train_d[0])
<type 'tuple'>
>>> p_train_d[0][0].shape
(784, 1)
>>> p_train_d[0][1].shape
(10, 1)
```

Accessing Processed MNIST Images and Their Target

```
>>> p_train_d[0][0]
array([[ 0.        ],
       [ 0.        ],
       [ 0.        ],
       [ 0.        ],
       ...

       [ 0.98828125],
       [ 0.9765625 ],
       [ 0.7109375 ],
       [ 0.        ]],
       dtype=float32)
>>> p_train_d[0][1]
array([[ 0.], [ 0.], [ 0.], [ 0.], [ 0.],
       [ 1.], [ 0.], [ 0.], [ 0.], [ 0.]])
```

Accessing Processed MNIST Images and Their Target

```
>>> raw_train_d[1][3]
1
>>> p_train_d[3][1]
array([[ 0.], [ 1.], [ 0.], [ 0.], [ 0.],
       [ 0.], [ 0.], [ 0.], [ 0.], [ 0.]])
>>> raw_train_d[1][10]
3
>>> p_train_d[10][1]
array([[ 0.], [ 0.], [ 0.], [ 1.], [ 0.],
       [ 0.], [ 0.], [ 0.], [ 0.], [ 0.]])
>>> raw_train_d[1][343]
8
>>> p_train_d[343][1]
array([[ 0.], [ 0.], [ 0.], [ 0.], [ 0.],
       [ 0.], [ 0.], [ 0.], [ 1.], [ 0.]])
```

Building ANNs of Various Architectures

The file `network.py` contains the class `Network.py`. Here is how you can use to construct ANNs of various architectures.

```
>>> net = Network([2, 3, 1])
>>> net.weights
[array([[ 0.61077501, -0.16264208],
        [-1.41462715,  0.68443288],
        [-1.20818535, -1.29271604]]),
 array([[-0.07313039,  0.57966367,  0.30828781]])]
>>> len(net.weights)
2
>>> net.weights[0].shape
(3, 2)
>>> net.weights[1].shape
(1, 3)
```

Training with Stochastic Gradient Descent

```
## construct 784 x 30 x 10 ANN
>>> net = network.Network([784, 30, 10])
## load the data
>>> train_d, valid_d, test_d = mnist_loader.load_data_wrapper()
## train the net with SGD for 30 iterations with
## the mini-batch size of 10 and the learning rate of 3.0.
>>> net.SGD(train_d, 30, 10, 3.0, test_data=test_d)
Epoch 29: 9459 / 10000
0.9459

## construct 784 x 30 x 10 ANN
>>> net = network.Network([784, 30, 10])
## train the net with SGD for 100 iterations with
## the mini-batch size of 10 and the learning rate of 3.0.
>>> net.SGD(train_d, 100, 10, 3.0, test_data=test_d)
Epoch 99: 9453 / 10000
0.9453
```

Training with Stochastic Gradient Descent

```
## construct 784 x 30 x 10 ANN
>>> net = network.Network([784, 30, 10])
## train it with SGD for 100 iterations, the mini-batch
## size of 10, and the learning rate of 1.0
>>> net.SGD(train_d, 100, 10, 1.0, test_data=test_d)
Epoch 99: 9461 / 10000
0.9461

## construct 784 x 30 x 10 ANN
>>> net = network.Network([784, 30, 10])
## train it with SGD for 200 iterations, the mini-batch
## size of 10, and the learning rate of 1.0
>>> net.SGD(train_d, 200, 10, 1.0, test_data=test_d)
Epoch 99: 9455 / 10000
0.9455
```

References

1. T.M. Mitchell. *Machine Learning*.
2. M. Neilsen. *Neural Networks and Deep Learning*.