Comparing MLP and CNN for Handwriting Digit Recognition

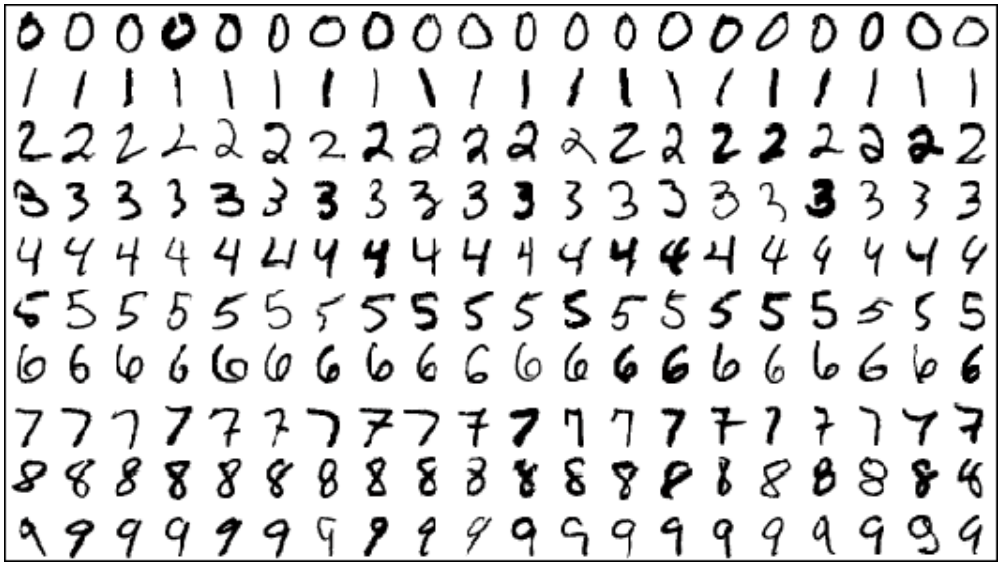
Liangliang Cao
Feb 11, 2014

http://llcao.net/cu-deeplearning15/

Problem and dataset

Recognize hand-written digits

- Input: 28x28 images
- Output: 0-9 (10 classes)
- Multi-class classification



Why is it useful? *Eichner's demo*

Questions we aim to answer in this class

- Is MLP better than logistic regression? When & why?
- Is CNN better than MLP/logistic regression?
- Are there more techniques which further improve the performance of deep CNN?

Review of last class

- Overview of Theano programming
  - Theano vs numpy
  - Symbolic language, functional programming
  - tensor.grad()
  - Implementation of SGD

- Example: logistic regression

Three components of a typical theano program

1. Dataset

2. Optimizer

3. Classification function and cost

We will next use the code example from
http://deeplearning.net/tutorial/logreg.html
to illustrate three components

1st components: Dataset

```
f = gzip.open('mnist.pkl.gz', 'rb')
train_set, valid_set, test_set = cPickle.load(f)
test_set_x, test_set_y = shared_dataset(test_set)
valid_set_x, valid_set_y = shared_dataset(valid_set)
train_set_x, train_set_y = shared_dataset(train_set)
```

2nd components: optimizer

```
while (epoch < n_epochs) and (patience <= iter):
    epoch = epoch + 1
    rand_inds = np.arange(n_train_batches)
    np.random.shuffle(rand_inds)
    for minibatch_index in rand_inds:
        minibatch_avg_cost = train_model(minibatch_index)
        iter = (epoch - 1) * n_train_batches + minibatch_index

        if (iter + 1) % validation_frequency == 0:
            validation_losses = [validate_model(i)  for i in xrange(n_valid_batches)]
            this_validation_loss = numpy.mean(validation_losses)
            if this_validation_loss < best_validation_loss:
                if this_validation_loss < best_validation_loss * improvement_threshold:
                    patience = max(patience, iter * patience_increase)
                best_validation_loss = this_validation_loss
                # test it on the test set
                test_losses = [test_model(i) for i in xrange(n_test_batches)]
                test_score = numpy.mean(test_losses)
```

3rd components: classification function and cost

self.p_y_given_x = T.nnet.softmax(T.dot(input, self.W) + self.b)

cost = -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]), y])

```
self.W = theano.shared(
    value=numpy.zeros(
        (n_in, n_out),
        dtype=theano.config.floatX
    ),
    name='W',
    borrow=True
)
# initialize the baises b as a vector of n_out 0s
self.b = theano.shared(
    value=numpy.zeros(
        (n_out,),
        dtype=theano.config.floatX
    ),
    name='b',
    borrow=True
)
```

```
# compute the gradient of cost with respect to theta = (W,b)
g_W = T.grad(cost=cost, wrt=classifier.W)
g_b = T.grad(cost=cost, wrt=classifier.b)

# start-snippet-3
# specify how to update the parameters of the model as a list of
# (variable, update expression) pairs.
updates = [(classifier.W, classifier.W - learning_rate * g_W),
           (classifier.b, classifier.b - learning_rate * g_b)]

# compiling a Theano function `train_model` that returns the cost, but in
# the same time updates the parameter of the model based on the rules
# defined in `updates`
train_model = theano.function(
    inputs=[index],
    outputs=cost,
    updates=updates,
    givens={
        x: train_set_x[index * batch_size: (index + 1) * batch_size],
        y: train_set_y[index * batch_size: (index + 1) * batch_size]
    }
)
```

Different classification functions

- Logistic regression

```
self.p_y_given_x = T.nnet.softmax(T.dot(input, self.W) + self.b)
cost =  -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]), y])
```

- MLP

```
lin_output = T.dot(input, self.W) + self.b
output0 = ( lin_output if activation is None else activation(lin_output))
self.p_y_given_x = T.nnet.softmax(T.dot(output0, self.W) + self.b)
cost = -T.mean(T.log(self.p_y_given_x)[T.arange(y.shape[0]), y])
```

Different classification functions

- CNN

```
layer0 = LeNetConvPoolLayer(rng, input=layer0_input, image_shape=(batch_size, 1, 28, 28), filter_shape=(nkerns[0], 1, 5, 5), poolsize=(2, 2))

layer1 = LeNetConvPoolLayer(rng, input=layer0.output, image_shape=(batch_size, nkerns[0], 12, 12), filter_shape=(nkerns[1], nkerns[0], 5, 5), poolsize=(2, 2))

layer2 = HiddenLayer(rng, input=layer2_input, n_in=nkerns[1] * 4 * 4, n_out=500, activation=T.tanh)

layer3 = LogisticRegression(input=layer2.output, n_in=500, n_out=10)

cost = layer3.negative_log_likelihood(y)
```

Question:

Can we reuse the same code of datasets and optimizers for different models?

Three components of a typical theano program

1. Dataset

We can reuse these two when comparing different models

2. Optimizer

3. Classification function and cost

Questions:

- Could CNN and hidden layer use the same input?

- Where should we change if we want to use a new cost?

An abstract definition: layer

- Input
  - e.g., T.matrix(), or T.tensor
- Key variable
  - Parameters
  - Parameter dimension
- Key output
  - Output()
  - Output_shape()

Codes in the following examples are just to help you understand, you are encouraged but not forced to follow.

Example 0: Abstract layer

```
class AbstractLayer(object):
    def __init__(self):
        self.input_layer = []
        self.params = []
        self._desc = ' '

    def set_params_values(self, param_values):
        for (p,v) in zip(self.params, param_values):
            p.set_value(v)

    def get_params_values(self):
        param_values = []
        for p in self.params:
            param_values.append(p.get_value())
        return param_values

    def output(self, *args, **kwargs)
        return None
```

Example 1: Hidden Layer

```
W_values = np.asarray(rng.uniform(
        low=-np.sqrt(6. / (n_in + n_out)),
        high=np.sqrt(6. / (n_in + n_out)),
        size=(n_in, n_out)), dtype=theano.config.floatX)
if activation == theano.tensor.nnet.sigmoid:
    W_values *= 4
self.W = theano.shared(value=W_values, name='W', borrow=True)

b_values = np.zeros((n_out,), dtype=theano.config.floatX)
self.b = theano.shared(value=b_values, name='b', borrow=True)

self.params = [self.W, self.b]
```

```
def output(self, *args, **kwargs):
    input = self.input_layer.output( *args, **kwargs)
    lin_output = T.dot(input, self.W) + self.b
    if self.activation is None:
        return lin_output
    else:
        return self.activation(lin_output)
```

Example 2: Convolution Layer

```
fan_in = np.prod(filter_shape[1:])
fan_out = (filter_shape[0] * np.prod(filter_shape[2:]) /4.0)
W_bound = np.sqrt(6. / (fan_in + fan_out))
self.W = theano.shared(np.asarray(
    rng.uniform(low=-W_bound, high=W_bound, size=filter_shape),
    dtype=theano.config.floatX), borrow=True)
b_values = np.zeros((filter_shape[0],), dtype=theano.config.floatX)
self.b = theano.shared(value=b_values, borrow=True)
self.params = [self.W, self.b]
```

```
def output(self, *args, **kwargs):
    conv_out = conv.conv2d(input=self.input_layer.output(), filters=self.W)
    if self.activation is None:
        return conv_out + self.b.dimshuffle('x', 0, 'x', 'x')
    else:
        return self.activation(conv_out + self.b.dimshuffle('x', 0, 'x', 'x'))
```

Example 3: Pooling Layer

```
self.poolsize = poolsize
self.params = []
```

```
def output(self, *args, **kwargs):
    return downsample.max_pool_2d(input=self.input_layer.output(), ds=self.poolsize)
```

This is a new layer definition. Guess what is its use?

A New Layer

```
self.dropout_rate = dropout_rate
self.params = []
```

```
def output(self, dropout_training = False,  *args, **kwargs):
    input = self.input_layer.output(dropout_training=dropout_training, *args, **kwargs)
    if dropout_training and (self.dropout_rate > 0):
        retain_prob = 1 - self.dropout_rate
        mask = srng.binomial(input.shape, p=retain_prob, dtype='int32').astype('float32')
        input = input / retain_prob * mask

    return input
```

Reference: Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, JMLR 2014

Questions?

Plans?

Projects?

In Class Programming Challenge

Performance

	Logistic regression	MLP	DeepCNN
28x28	7.5%	1.7%	0.9%
14x14	11%	3~4%	2.6%

What I would try next:

- Different activation function (e.g., rectify instead of sigmoid)
- 2 convolution layer followed by one pooling layer
- Dropout