SpencerHann_finalPresentation

March 19, 2020

1 Convolutional Neural Network

Spencer Hann EE 584 - Final Project

[1]:
```
import numpy as np

from matplotlib import pyplot as plt
from cnn.data import preprocess_data
```

[2]:
```
training_examples, training_targets = \
    preprocess_data("data/mnist_train.csv", max_rows=4000)
testing_examples, testing_targets = \
    preprocess_data("data/mnist_test.csv", max_rows=1000)

data = (training_examples, training_targets, testing_examples, testing_targets)
train_set = (training_examples, training_targets)
test_set = (testing_examples, testing_targets)
```

```
loading from file… done.
loading from file… done.
```

1.1 Proof of Concept: Simple Neural Net Layer

[3]:
```
from cnn.cnn_basic import CNN, Layer, DenseSoftmaxLayer
```

[4]:
```
class DenseLayer:
    def __init__(self, insize, outsize=10):
        self.insize = insize
        self.outsize = outsize

        self.w = np.random.randn(insize, outsize) / insize
        self.b = np.random.randn(outsize) / outsize

    def forward(self, image):
        image = image.flatten()
```

```python
        # fully-connected/matmul phase
        result = np.dot(image, self.w) + self.b
        return result
```

[5]:
```python
layer1 = DenseLayer(28*28, 64)
layer2 = DenseLayer(64, 10)
```

[6]:
```python
def forward(image, label):
    middle = layer1.forward(image)
    out = layer2.forward(middle)

    is_correct = np.argmax(out) == label
    return None, is_correct
```

[7]:
```python
def test(images, labels):
    n_correct = 0.0
    for image, label in zip(images, labels):
        _, c = forward(image, label)
        n_correct += c
    return n_correct / len(images)
```

[8]:
```python
accuracy = 100 * test(*test_set)  # should be about 1 / n_classes
print(f"Accuracy: {round(accuracy)}%")
```

```
Accuracy: 11.0%
```

1.2 Adding Back Propagation

[9]:
```python
class DenseLayer:
    def __init__(self, insize, outsize=10):
        self.insize = insize
        self.outsize = outsize

        self.w = np.random.randn(insize, outsize) / insize
        self.b = np.random.randn(outsize) / outsize

    def forward(self, image):
        self.last_image = image        # <<----
        image = image.flatten()

        # fully-connected/matmul phase
        fc = np.dot(image, self.w) + self.b
        self.last_fc = fc              # <<----

        return fc
```

```python
    def backprop(self, loss_grad, lr=0.002):
        # output gradients wrt input, biases, weights
        ograd_input = self.w
        ograd_biases = 1
        ograd_weights = self.last_image.flatten()

        # loss gradients wrt input, biases, weights
        lgrad_input = ograd_input @ loss_grad
        lgrad_biases = ograd_biases * loss_grad
        lgrad_weights = ograd_weights[:,np.newaxis] @ loss_grad[np.newaxis]

        # update layer
        self.w += lr * lgrad_weights
        self.b += lr * lgrad_biases
        return lgrad_input.reshape(self.last_image.shape)
```

[10]:
```python
layer1 = DenseLayer(28*28, 64)
layer2 = DenseLayer(64, 10)
layers = (layer1, layer2,)
```

[11]:
```python
def forward(image, label):
    out = image
    for layer in layers:
        out = layer.forward(out)

    is_correct = np.argmax(out) == label
    loss = -np.log(out[label])           # <<------

    return out, loss, is_correct
```

[12]:
```python
def learn(image, label):
    out, loss, correct = forward(image, label)

    if correct:
        return loss, correct

    # cross entropy gradient
    grad = np.zeros(10)
    grad[label] = - 1 / out[label]

    for layer in layers[::-1]:
        grad = layer.backprop(grad, lr=0.02)

    return loss, correct
```

[13]:
```python
def train(n_epochs, images, labels):
    n = len(images)
```

```
    for epoch in range(n_epochs):
        ncorrect = 0
        for image, label in zip(images, labels):
            _, c = learn(image, label)
            ncorrect += c

        accuracy = round(100 * ncorrect / n)
        print(f"Epoch:{epoch}, Accuracy: {100 * ncorrect / n}")
```

1.3 Demonstration of Forward propogation

This is a randomly initialized network that we will test the feed forward functionality on. With 10 evenly represented output classes in our testing data, we expect to see roughly 10% accuracy.

[14]:
```
from cnn.cnn_basic import (
    CNN,
    ConvolutionalLayer,
    MaxPoolingLayer,
    DenseSoftmaxLayer,
)
```

[15]:
```
cnn = CNN((
    ConvolutionalLayer(1,3,3, first_layer=True),
    MaxPoolingLayer(2),
    ConvolutionalLayer(3,1,3,),
    DenseSoftmaxLayer(14*14, 10),
))
```

[16]:
```
# cnn.test(*test_set);
```

Approximately random perfomance, this is to be expected. It shows that feed forward is working properly.

1.4 Demonstration of Back Propogation

[17]:
```
cnn = CNN(
    (
        ConvolutionalLayer(1,3,3, first_layer=True),
        DenseSoftmaxLayer(28*28*3, 10),
    ),
    lr = 0.01
)
```

[18]:
```
%time cnn.train_epochs(3, *train_set);
```

```
100%|      | 4000/4000 [02:46<00:00, 23.96it/s]
  0%|          | 2/4000 [00:00<03:21, 19.87it/s]

Epoch 0/3: 1.91 loss, 39.32% accurate

100%|      | 4000/4000 [02:46<00:00, 23.96it/s]
  0%|          | 2/4000 [00:00<03:38, 18.26it/s]

Epoch 1/3: 2.01 loss, 47.85% accurate

100%|      | 4000/4000 [03:20<00:00, 19.92it/s]

Epoch 2/3: 2.21 loss, 49.30% accurate
CPU times: user 18min 55s, sys: 12min 19s, total: 31min 15s
Wall time: 8min 54s
```

[19]:
```
cnn.test(*test_set);
```

```
100%|      | 1000/1000 [00:14<00:00, 71.36it/s]

Test: 2.74 loss, 41.40% accurate
```

Though this is significantly better than random, it is still not as high as I'd like it to be. I believe the reason for this is that the network in it current state is relatively low-capacity, and does not contain any non-linearities.