Convolutional Neural Nets

Using MXNet

Cyrus M. Vahid, Principal Solutions Architect,
Principal Solutions Architect @ AWS DeepLearning

cyrusmv@amazon.com

June 2017
Sparse Matrix and Spatial Correlation

<table>
<thead>
<tr>
<th>2</th>
<th>a</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>5</td>
<td>s</td>
</tr>
</tbody>
</table>
Feature Detection

- Male: 97.4%
- Eyes are open: 100%
- Happy: 96.0%
- Smiling: 96.3%
- Mustache: 100%
- Beard: 65.3%
Non-Convolutional Network

```
In [13]: from IPython.display import HTML

import cv2
import numpy as np

def classify(img):
    img = img[len('data:image/png;base64,'):].decode('base64')
    img = cv2.imdecode(np.fromstring(img, np.uint8), -1)
    img = cv2.resize(img[:, :, 3], (28, 28))
    img = img.astype(np.float32).reshape((1, 1, 28, 28))/255.0
    return model.predict(img)[0].argmax()

To see the model in action, run the demo notebook at

HTML(filename="mnist_demo.html")
```

Out[13]:

```
Classify  Clear  Result: 5
```
Convolution

- Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.
- We use a reduction mechanism that is weighted differently based on relevance.
  - Example: Spaceship measurement along a path creates a discrete set of measurement. Each one could be fuzzy, but averaging them helps remove the noise, and have better prediction on the current location with more weight given to the local position.

\[
S(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a)
\]

- \(x\) is often called *input* (often multi-dimensional array of data) and \(w\) is called *kernel* (often multi-dimensional array of parameters).
Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- Max Pooling operation reports the maximum output within a rectangular neighborhood.
- Pooling helps detect existence of features as opposed to detecting where a feature is through making a representation invariant to small translation in the input.

After stride of one pixel, the pooling stage has fewer changes compared to detector stage
Convolutional Neural Networks - Architecture

• Input Layer takes the raw array of data.
• Feature Extraction layers extract features through:
  • The first layer performs several convolutions in parallel to produce a set of linear activations.
  • In the second stage (detector), each linear activation is run through a nonlinear activation function, such as ReLU
  • The third layer performs pooling on the output
• In the end fully-connected layers, reassemble the features into final output and apply Softmax to predict create a probabilistic distribution.
Convolution in Example

**Input Volume (1x5x5 + pad 1)**

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>31</td>
<td>12</td>
<td>167</td>
</tr>
<tr>
<td>0</td>
<td>45</td>
<td>54</td>
<td>67</td>
<td>254</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>25</td>
<td>238</td>
<td>254</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>196</td>
<td>254</td>
<td>254</td>
</tr>
<tr>
<td>0</td>
<td>127</td>
<td>254</td>
<td>254</td>
<td>238</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Receptive Field \( I_{21} \)**

```
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>45</td>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>196</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Kernel \( K \) (1x3x3)**

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

**Output**

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>243</td>
</tr>
<tr>
<td>142</td>
<td></td>
</tr>
</tbody>
</table>
```

**Inner Product \( \langle I_{21}, K \rangle \)**

\[
= 0 \cdot 1 + 45 \cdot 0 + 54 \cdot (-1) + 0 \cdot 1 + 0 \cdot 1 + 25 \cdot 0 + 0 \cdot (-1) + 10 \cdot 0 + 196 \cdot 1 = 142
\]
Max Pooling in Example

2x2 Max Pooling

2x2 filter with 2x2 stride...halves height and width dimensions while keeping all channels intact.
CNN Feature Detection

Deep neural networks learn hierarchical feature representations.
Apache MXNet
Why Apache MXNet?

Most Open
Accepted into the Apache Incubator

Best On AWS
Optimized for deep learning on AWS
(Integration with AWS)
Amazon AI: Scaling With MXNet

91% Efficiency

Ideal
Inception v3
Resnet
Alexnet
Amazon AI: Scaling With MXNet

88% Efficiency

Ideal
Inception v3
Resnet
Alexnet

1 2 4 8 16 32 64 128 256
CNN in MXNet

- **Activation tanh**
  - Input: 50x8x8
  - Output: 20x12x12

- **Convolution 5x5/1, 50**
  - Input: 20x12x12
  - Output: 20x24x24

- **Pooling max, 2x2/2**
  - Input: 20x24x24
  - Output: 50x8x8

- **Activation tanh**
  - Input: 50x8x8
  - Output: 20x24x24

- **Convolution 5x5/1, 20**
  - Input: 1x28x28
  - Output: 20x24x24

- **Pooling max, 2x2/2**
  - Input: 20x24x24
  - Output: 50x4x4

- **FullyConnected 500**
  - Input: 50x4x4
  - Output: 800

- **Activation tanh**
  - Input: 500
  - Output: 500

- **FullyConnected 10**
  - Input: 500
  - Output: softmax_label

- **SoftmaxOutput**
  - Input: softmax_label
  - Output: 10
data = mx.symbol.Variable('data')
# first conv layer
conv1 = mx.sym.Convolution(data=data, kernel=(5,5), num_filter=20)
tanh1 = mx.sym.Activation(data=conv1, act_type="tanh")
pool1 = mx.sym.Pooling(data=tanh1, pool_type="max", kernel=(2,2), stride=(2,2))
# second conv layer
conv2 = mx.sym.Convolution(data=pool1, kernel=(5,5), num_filter=50)
tanh2 = mx.sym.Activation(data=conv2, act_type="tanh")
pool2 = mx.sym.Pooling(data=tanh2, pool_type="max", kernel=(2,2), stride=(2,2))
# first fullc layer
flatten = mx.sym.Flatten(data=pool2)
fc1 = mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 = mx.sym.Activation(data=fc1, act_type="tanh")
# second fullc
fc2 = mx.sym.FullyConnected(data=tanh3, num_hidden=10)
# softmax loss
lenet = mx.sym.SoftmaxOutput(data=fc2, name='softmax')
mx.viz.plot_network(symbol=lenet, shape=shape)
Training The Network

```python
# @@ AUTOTEST_OUTPUT_ignored_CELL
model = mx.model.FeedForward(
    ctx = mx.gpu(0),  # use GPU 0 for training, others are same as before
    symbol = lenet,
    num_epoch = 10,
    learning_rate = 0.1
)
model.fit(
    X=train_iter,
    eval_data=val_iter,
    batch_end_callback = mx.callback.Speedometer(batch_size, 200)
)
assert model.score(val_iter) > 0.98, "Low validation accuracy."
```
Define Network - gluon

```python
num_fc = 512
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Conv2D(channels=20, kernel_size=5, activation='relu'))
    net.add(gluon.nn.MaxPool2D(pool_size=2, strides=2))
    net.add(gluon.nn.Conv2D(channels=50, kernel_size=5, activation='relu'))
    net.add(gluon.nn.MaxPool2D(pool_size=2, strides=2))
    # The Flatten layer collapses all axis, except the first one, into one axis.
    net.add(gluon.nn.Flatten())
    net.add(gluon.nn.Dense(num_fc, activation="relu"))
    net.add(gluon.nn.Dense(num_outputs))
```
Initialize and Train

```python
net.collect_params().initialize(mx.init.Xavier(magnitude=2.24), ctx=ctx)
softmax_cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': .1})
```

```python
epochs = 10
smoothing_constant = .01

for e in range(epochs):
    for i, (data, label) in enumerate(train_data):
        data = data.as_in_context(ctx)
        label = label.as_in_context(ctx)
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
            loss.backward()
            trainer.step(data.shape[0])

            # Keep a moving average of the losses
            curr_loss = nd.mean(loss).asscalar()
            moving_loss = (curr_loss if ((i == 0) and (e == 0))
                            else (1 - smoothing_constant) * moving_loss + (smoothing_constant) * curr_loss)

    test_accuracy = evaluate_accuracy(test_data, net)
    train_accuracy = evaluate_accuracy(train_data, net)
    print("Epoch %d. Loss: %f, Train_acc %f, Test_acc %f" % (e, moving_loss, train_accuracy, test_accuracy))
```
Demo Time

- Hand-written Digits
- Predicting with a pre-trained Network (resnet)
Thank you!

Cyrus M. Vahid
cyrusmv@amazon.com