CS 4803 / 7643: Deep Learning

Topics:
- Finish forward and backward through conv
- Convolutional neural network (CNN) architectures

Zsolt Kira
Georgia Tech
Administrative

• PS1/HW1 Due Feb 11th!
Example: Reverse mode AD

\[ f(x_1, x_2) = x_1 x_2 + \sin(x_1) \]

\[ \bar{w}_3 = 1 \]

\[ \bar{w}_1 = \bar{w}_3 \quad \bar{w}_2 = \bar{w}_3 \]

\[ \bar{x}_1 = \bar{w}_1 \cos(x_1) \quad \bar{x}_1 = \bar{w}_2 x_2 \quad \bar{x}_2 = \bar{w}_2 x_1 \]
Duality in Fprop and Bprop
Convolutions for programmers

\[ y(r, c) = (x * \omega)(r, c) \]

\[
= \sum_{a=0}^{H-1} \sum_{b=0}^{W-1} x(a, b) \omega(r - a, c - b)
\]

- Iterate over the kernel instead of the image

\[
= \sum_{a=0}^{K-1} \sum_{b=0}^{K-1} x(r - a, c - b) \omega(a, b)
\]

- Implement cross-correlation instead of convolution

\[
= \sum_{a=0}^{K-1} \sum_{b=0}^{K-1} x(r + a, c + b) \omega(a, b)
\]

- Later - implementation as matrix multiplication

(C) Peter Anderson
In practice: Common to zero pad the border

```
0 0 0 0 0 0
0
0
0
0
```

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?

(recall:)
(N - F) / stride + 1
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Let us assume filter is an “eye” detector.

**Q.** how can we make the detection robust to the exact location of the eye?
Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Pooling Layer: Examples

Max-pooling:

\[ h_i^n(r, c) = \max_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c}) \]

Average-pooling:

\[ h_i^n(r, c) = \frac{\text{mean}_{\bar{r} \in N(r), \bar{c} \in N(c)}}{h_i^{n-1}(\bar{r}, \bar{c})} \]

L2-pooling:

\[ h_i^n(r, c) = \sqrt{\sum_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c})^2} \]
Receptive Field

\[ h^{(l-1)} \rightarrow \text{Conv. layer} \rightarrow h^{(l)} \rightarrow \text{Pool. layer} \rightarrow h^{(l+1)} \]
Pooling Layer: Receptive Field Size

\[ h^{(l-1)} \rightarrow \text{Conv. layer} \rightarrow h^{(l)} \rightarrow \text{Pool. layer} \rightarrow h^{(l+1)} \]
If convolutional filters are FxF and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch in \( h^{(l-1)} \) of size: \((P+F-1) \times (P+F-1)\)
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent $F$,
  - the stride $S$,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
Common settings:

- \( F = 2, \ S = 2 \)
- \( F = 3, \ S = 2 \)

- Accepts a volume of size \( W_1 \times H_1 \times D_1 \)
- Requires three hyperparameters:
  - their spatial extent \( F \),
  - the stride \( S \),
- Produces a volume of size \( W_2 \times H_2 \times D_2 \) where:
  - \( W_2 = (W_1 - F)/S + 1 \)
  - \( H_2 = (H_1 - F)/S + 1 \)
  - \( D_2 = D_1 \)
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
Convolutional Neural Networks

Image Credit: Yann LeCun, Kevin Murphy
The architecture of LeNet5
Handwriting Recognition Example

filters → tanh → average-tanh → filters → tanh → average-tanh → filters → tanh

Curved manifold

Flatter manifold
Translation Invariance
Some Rotation Invariance
Some Scale Invariance
Case Studies

• There are several generations of ConvNets
  – 2012 – 2014: AlexNet, ZNet, VGGNet
    • Conv-Relu, Pooling, Fully connected, Softmax
    • Deeper ones (VGGNet) tend to do better
  – 2014
    • Fully-convolutional networks for semantic segmentation
    • Matrix outputs rather than just one probability distribution
  – 2014-2016
    • Fully-convolutional networks for classification
    • Less parameters, faster than comparable Gen1 networks
    • GoogleNet, ResNet
  – 2014-2016
    • Detection layers (proposals)
    • Caption generation (combine with RNNs for language)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

**Q:** what is the output volume size? **Hint:** \((227-11)/4+1 = 55\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4

\[ \text{Output volume} \ [55x55x96] \]

Q: What is the total number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume [55x55x96]

Parameters: \((11*11*3)*96 = 35K\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: \((55-3)/2+1 = 27\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images  
After CONV1: 55x55x96

**Second layer** (POOL1): 3x3 filters applied at stride 2  
Output volume: 27x27x96

Q: What is the number of parameters in this layer?
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96

...
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)
Case Study: AlexNet
[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%
Case Study: ZFNet  [Zeiler and Fergus, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL stride 2

best model

11.2% top 5 error in ILSVRC 2013

->

7.3% top 5 error
INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64=800K params: 0

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128=400K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256=200K params: 0

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296

POOL2: [14x14x512] memory: 14*14*512=100K params: 0

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296

POOL2: [7x7x512] memory: 7*7*512=25K params: 0

FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448

FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216

FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! *2 for bwd)
TOTAL params: 138M parameters
INPUT: [224x224x3]  memory: 224*224*3=150K  params: 0  (not counting biases)

CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*64)*64 = 36,864
POOL2:  [112x112x64]  memory: 112*112*64=800K  params: 0
CONV3-128: [112x112x128]  memory: 112*112*128=1.6M  params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]  memory: 112*112*128=1.6M  params: (3*3*128)*128 = 147,456
POOL2:  [56x56x128]  memory: 56*56*128=400K  params: 0
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*256)*256 = 589,824
POOL2:  [28x28x256]  memory: 28*28*256=200K  params: 0
CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]  memory: 28*28*512=400K  params: (3*3*512)*512 = 2,359,296
POOL2:  [14x14x512]  memory: 14*14*512=100K  params: 0
CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]  memory: 14*14*512=100K  params: (3*3*512)*512 = 2,359,296
POOL2:  [7x7x512]  memory: 7*7*512=25K  params: 0

FC:  [1x1x4096]  memory: 4096  params: 7*7*512*4096 = 102,760,448
FC:  [1x1x4096]  memory: 4096  params: 4096*4096 = 16,777,216
FC:  [1x1x1000]  memory: 1000  params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image  (only forward! ~*2 for bwd)
TOTAL params: 138M parameters
example 5x5 filters (32 total)

one filter => one activation map

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Visualizing Learned Filters

Layer 1

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Layer 1

Layer 2

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Layer 3

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Layer 4

Layer 5

Figure Credit: [Zeiler & Fergus ECCV14]
We can learn image features now!

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun
Case Study: GoogLeNet  

[ Szegedy et al., 2014 ]

Inception module

ILSVRC 2014 winner (6.7% top 5 error)
image
Natural images are locally heavily correlated.
Filter activations reflect image correlations
Image correlations reflected in filter bank correlations

# of filters

image

filter bank
Correlations in natural images are multi-scale
Correlations in natural images are multi-scale

1x1 filters
3x3 filters
5x5 filters
Replace convolution with multi-scale convolution

Going Deeper with Convolutions
C Szegedy et al (2014)
Employ multi-scale and dimensional reduction.

Going Deeper with Convolutions
C Szegedy et al (2014)
Summary of Inception architecture.

- Multi-scale architecture to mirror correlation structure in images.
- Dimensional reduction to constrain representation along each spatial scale.

Going Deeper with Convolutions
C Szegedy et al (2014)
An Aside

- Auxiliary classifier connected to intermediate layers
Case Study: GoogLeNet

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/stride</th>
<th>output size</th>
<th>depth</th>
<th>#1x1</th>
<th>#3x3 reduce</th>
<th>#5x5 reduce</th>
<th>#5x5 pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7x7/2</td>
<td>112x112x64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>56x56x64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td>3x3/1</td>
<td>56x56x192</td>
<td>2</td>
<td></td>
<td>64</td>
<td>192</td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>28x28x192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (A)</td>
<td>28x28x256</td>
<td>2</td>
<td>64</td>
<td></td>
<td>66</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>inception (B)</td>
<td>28x28x480</td>
<td>2</td>
<td>128</td>
<td></td>
<td>128</td>
<td>192</td>
<td>22</td>
<td>96</td>
<td>64</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>14x14x480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (A)</td>
<td>14x14x512</td>
<td>2</td>
<td>192</td>
<td></td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>64</td>
</tr>
<tr>
<td>inception (B)</td>
<td>14x14x512</td>
<td>2</td>
<td>160</td>
<td></td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>inception (C)</td>
<td>14x14x512</td>
<td>2</td>
<td>128</td>
<td></td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>inception (D)</td>
<td>14x14x512</td>
<td>2</td>
<td>112</td>
<td></td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>inception (E)</td>
<td>14x14x512</td>
<td>2</td>
<td>256</td>
<td></td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>avg pool</td>
<td>7x7/1</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout</td>
<td>40%</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1x1x1000</td>
<td>1</td>
<td>100K</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>softmax</td>
<td>1x1x1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fun features:
- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

AlexNet: 60M params
ZNet: 75M
VGG: 138M
GoogleNet: 5M
Case Study: ResNet [He et al., 2015]
ILSVRC 2015 winner (3.6% top 5 error)

MSRA @ ILSVRC & COCO 2015 Competitions

• 1st places in all five main tracks
  • ImageNet Classification: “Ultra-deep” (quote Yann) 152-layer nets
  • ImageNet Detection: 16% better than 2nd
  • ImageNet Localization: 27% better than 2nd
  • COCO Detection: 11% better than 2nd
  • COCO Segmentation: 12% better than 2nd

*Improvements are relative numbers

Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w
Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet: 3.57
ILSVRC'14 GoogleNet: 6.7
ILSVRC'14 VGG: 7.3
ILSVRC'13: 11.7
ILSVRC'12 AlexNet: 16.4
ILSVRC'11: 25.8
ILSVRC'10: 28.2

152 layers

(slots from Kaiming He's recent presentation)
Importance of Depth

- After a while, adding depth decreases performance
- At first, vanishing/exploding gradients
  - normalized initialization
  - Batch normalization
  - 2nd order methods
- Then, optimization limitation
  - Deeper network should be able to mimic shallow ones
Case Study: ResNet [He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)

Revolution of Depth

- AlexNet, 8 layers (ILSVRC 2012)
- VGG, 19 layers (ILSVRC 2014)
- ResNet, 152 layers (ILSVRC 2015)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

(slide from Kaiming He’s recent presentation)
Case Study: ResNet

[He et al., 2015]

- 34-layer plain
- 34-layer residual

Spatial dimension only 56x56!
Case Study: ResNet  

[He et al., 2015]
Case Study: ResNet  

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used
Case Study: ResNet

[He et al., 2015]

(this trick is also used in GoogLeNet)
### Case Study: ResNet [He et al., 2015]

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv1</td>
<td>112×112</td>
<td>7×7, 64, stride 2</td>
<td>3×3 max pool, stride 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv2_x</td>
<td>56×56</td>
<td>[3×3, 64]×2</td>
<td>[3×3, 64]×3</td>
<td>[1×1, 64]×3</td>
<td>[1×1, 64]×3</td>
<td>[1×1, 64]×3</td>
</tr>
<tr>
<td>conv3_x</td>
<td>28×28</td>
<td>[3×3, 128]×2</td>
<td>[3×3, 128]×4</td>
<td>[1×1, 128]×4</td>
<td>[1×1, 128]×8</td>
<td></td>
</tr>
<tr>
<td>conv5_x</td>
<td>7×7</td>
<td>[3×3, 512]×2</td>
<td>[3×3, 512]×3</td>
<td>[1×1, 512]×3</td>
<td>[1×1, 512]×3</td>
<td></td>
</tr>
<tr>
<td>1×1</td>
<td>average pool, 1000-d fc, softmax</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOPs</td>
<td>1.8×10⁹</td>
<td>3.6×10⁹</td>
<td>3.8×10⁹</td>
<td>7.6×10⁹</td>
<td>11.3×10⁹</td>
<td></td>
</tr>
</tbody>
</table>