CS 4803 / 7643: Deep Learning

Topics:
- Forward and backward through conv
- (Beginning) of 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

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

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

- 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
Discrete convolution

- Discrete Convolution!
  - Very similar to correlation but associative

\[ F \otimes (G \otimes I) = (F \otimes G) \otimes I \]

\[
y_k = \sum_{n=0}^{N-1} h_n \cdot x_{k-n}
\]

1D Convolution

\[
y_0 = h_0 \cdot x_0
y_1 = h_1 \cdot x_0 + h_0 \cdot x_1
y_2 = h_2 \cdot x_0 + h_1 \cdot x_1 + h_0 \cdot x_2
y_3 = h_2 \cdot x_1 + h_1 \cdot x_2 + h_0 \cdot x_3
\vdots
\]

2D Convolution

\[ H[m,n] = f \otimes I = \sum_{k,l} f[k,l] I[m-k,n-l] \]
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Filters always extend the full depth of the input volume

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)  
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied \textbf{with stride 3}?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
\[(N - F) / \text{stride} + 1\]

e.g. N = 7, F = 3:
- \(\text{stride } 1 \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride } 2 \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride } 3 \Rightarrow (7 - 3)/3 + 1 = 2.33\)
In practice: Common to zero pad the border

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

(recall:)
(N - F) / stride + 1
In practice: Common to zero pad the border

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

7x7 output!
In practice: Common to zero pad the border

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

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
    F = 5 => zero pad with 2
    F = 7 => zero pad with 3
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
General Matrix Multiply (GEMM)

Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: \(32 \times 32 \times 3\)  
10 5x5 filters with stride 1, pad 2

Output volume size:  
\[
(32 + 2 \times 2 - 5) / 1 + 1 = 32 \text{ spatially, so } 32 \times 32 \times 10
\]
Examples time:

Input volume: $32 \times 32 \times 3$
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

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

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has \(5 \times 5 \times 3 + 1 = 76\) params  
(+1 for bias)
=> \(76 \times 10 = 760\)
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P) / S + 1$
  - $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
Common settings:

- $K = (\text{powers of 2, e.g. } 32, 64, 128, 512)$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ? \ (\text{whatever fits})$
- $F = 1, S = 1, P = 0$

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P) / S + 1$
  - $H_2 = (H_1 - F + 2P) / S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
Example: CONV layer in Torch

**SpatialConvolution**

```python
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane x height x width`).

The parameters are the following:

- `nInputPlane`: The number of expected input planes in the image given into `forward()`.
- `nOutputPlane`: The number of output planes the convolution layer will produce.
- `kW`: The kernel width of the convolution
- `kH`: The kernel height of the convolution
- `dW`: The step of the convolution in the width dimension. Default is 1.
- `dH`: The step of the convolution in the height dimension. Default is 1.
- `padW`: The additional zeros added per width to the input planes. Default is 0, a good number is `(kW-1)/2`.
- `padH`: The additional zeros added per height to the input planes. Default is `padW`, a good number is `(kH-1)/2`.

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width`, the output image size will be `nOutputPlane x oheight x owidth` where

```python
owidth = floor((width + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

**Summary.** To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.

*Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
**Preview:** ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
Backprop through Conv
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.

- **Example:**
  - Initial Convolutional Layer: 3x3x3 filters, followed by ReLU activation.
  - Subsequent Convolutional Layer: 5x5x6 filters, followed by ReLU activation.
  - Further Convolutional Layer: 5x5x3 filters, followed by ReLU activation.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolutional Neural Networks

![Diagram of Convolutional Neural Network](image-url)
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)