CNNs + PAC Learning
Reminders

• Homework 5: Neural Networks
  – Out: Sun, Feb 27
  – Due: Fri, Mar 18 at 11:59pm
Peer Tutoring

Tutor

- Improved course for everyone
- personal attention
- mastery
- deeper understanding
- better grades

Tutee

- better grades
- personal attention
- deeper understanding
- mastery
- improved course for everyone
Dynamic Programming

**Question:**
Have you studied dynamic programming in a previous course?
A. Yes
B. No

**Answer:**
*

**Question:**
What is the difference between memoization and dynamic programming, when applied to a recursive function f(x)?
A. **memoization** computes a function recursively without storing intermediate results, whereas **dynamic programming** stores intermediate results
B. **memoization** stores function values as they are encountered top-down, whereas **dynamic programming** stores function values as they are encountered bottom-up
C. **memoization** stores only the output of a tertiary function g(x), whereas **dynamic programming** stores the outputs of f(x) directly
D. **memoization** typically increases computational complexity of an algorithm while decreasing space complexity, whereas **dynamic programming** typically decreases computational complexity and increases space complexity
E. **memoization** memorizes a function, whereas **dynamic programming** has a programmer generate code for the function on-the-fly (i.e. I answered “Yes” to previous question)
BACKGROUND: COMPUTER VISION
Example: Image Classification

• ImageNet LSVRC-2011 contest:
  – **Dataset**: 1.2 million labeled images, 1000 classes
  – **Task**: Given a new image, label it with the correct class
  – **Multiclass** classification problem
• Examples from http://image-net.org/
Bird
Warm-blooded egg-laying vertebrates characterized by feathers and forelimbs modified as wings
German iris, Iris kochii

Iris of northern Italy having deep blue-purple flowers; similar to but smaller than Iris germanica

- halophyte (0)
- succulent (39)
- cultivar (0)
- cultivated plant (0)
- weed (54)
- evergreen, evergreen plant (0)
- deciduous plant (0)
- vine (272)
- creeper (0)
- woody plant, ligneous plant (1868)
- geophyte (0)
- desert plant, xerophyte, xerophytic plant, xerophile (0)
- mesophyte, mesophytic plant (0)
- aquatic plant, water plant, hydrophyte, hydrophytic plant (11)
- tuberous plant (0)
- bulbous plant (179)
- indaceous plant (27)
  - iris, flag, fleur-de-lis, sword lily (19)
    - bearded iris (4)
      - Florentine iris, orris, Iris germanica florentina, Iris Germanica florentina (0)
      - German iris, Iris germanica (0)
      - German iris, Iris kochii (0)
      - Dalmatian iris, Iris pallida (0)
    - beardless iris (4)
    - bulbous iris (0)
    - dwarf iris, Iris cristata (0)
    - stinking iris, gladdon, gladdon iris, stinking gladwyn, Persian iris, Iris persica (0)
    - yellow iris, yellow flag, yellow water flag, Iris pseudacorus (0)
    - dwarf iris, vernal iris, Iris verna (0)
    - blue flag, iris versicolor (0)
Court, courtyard

An area wholly or partly surrounded by walls or buildings; "the house was built around an inner court"
Feature Engineering for CV

Edge detection (Canny)

Corner Detection (Harris)

Scale Invariant Feature Transform (SIFT)

Figures from http://opencv.org

Figure from Lowe (1999) and Lowe (2004)
4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter-dependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.
CNNs for Image Recognition

Revolution of Depth

ILSVRC'15
ResNet
3.57

ILSVRC'14
GoogleNet
6.7

ILSVRC'14
VGG
7.3

ILSVRC'13
8 layers
11.7

ILSVRC'12
AlexNet
16.4

ILSVRC'11
shallow
25.8

ILSVRC'10
28.2


Slide from Kaiming He
Backpropagation and Deep Learning

Convolutional neural networks (CNNs) and recurrent neural networks (RNNs) are simply fancy computation graphs (aka. hypotheses or decision functions).

Our recipe also applies to these models and (again) relies on the backpropagation algorithm to compute the necessary gradients.
CONVOLUTION
What’s a convolution?

• Basic idea:
  – Pick a 3x3 matrix F of weights
  – Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation

• Key point:
  – Different convolutions extract different types of low-level “features” from an image
  – All that we need to vary to generate these different features is the weights of F

Ex: 1 input channel, 1 output channel

\[
\begin{align*}
  y_{11} &= \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_0 \\
  y_{12} &= \alpha_{11} x_{12} + \alpha_{13} x_{13} + \alpha_{21} x_{22} + \alpha_{22} x_{23} + \alpha_0 \\
  y_{21} &= \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{31} + \alpha_{22} x_{32} + \alpha_0 \\
  y_{22} &= \alpha_{11} x_{22} + \alpha_{13} x_{23} + \alpha_{21} x_{32} + \alpha_{22} x_{33} + \alpha_0
\end{align*}
\]

Slide adapted from William Cohen
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

<table>
<thead>
<tr>
<th>Convolution</th>
<th>Input Image</th>
<th>Convolved Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0 0 0 0 0 0</td>
<td>1 1 1 1 1 1 1 1</td>
</tr>
<tr>
<td>0 1 1 1 1 1 0</td>
<td>0 1 0 0 1 0 0 0</td>
<td>1 0 0 1 0 0 0 0</td>
</tr>
<tr>
<td>0 1 0 1 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1</td>
<td>1 0 1 0 0 0 0 0</td>
</tr>
<tr>
<td>0 1 1 0 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1</td>
<td>1 1 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 1 0 0 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0 0</td>
<td>0 1 1 1 1 1 1 1</td>
<td>1 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

![Convolution Matrix Example](image-url)
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

```
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0
0 1 0 0 1 0 0
0 1 0 1 0 0 0
0 1 1 0 0 0 0
0 1 1 0 0 0 0
0 1 0 0 0 0 0
0 0 0 0 0 0 0
```

**Convolution**

```
0 0 0
0 1 1
0 1 0
```

**Convolved Image**

```
3 2 2 3 1
2 0 2 1 0
2 2 1 0 0
3 1 0 0 0
1 0 0 0 0
```
Background: Image Processing

A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

Convolution

Convolved Image
A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

```
0 0 0 0 0 0 0 0
0 1 1 1 1 1 0 0
0 1 0 0 1 1 0 0
0 1 0 1 1 0 0 0
0 1 1 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0
```

**Convolution**

```
3 2 2 3 1
2 0 2 1 0
2 2 1 0 0
3 1 0 0 0
1 0 0 0 0
```
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Convolution Matrix Example](image)

<table>
<thead>
<tr>
<th>Input Image</th>
<th>Convolved Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Input Image Matrix" /></td>
<td><img src="image" alt="Convolved Image Matrix" /></td>
</tr>
</tbody>
</table>

The convolution process involves applying the convolution matrix to the input image to produce the convolved image.
A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

**Convolution**

**Convolved Image**
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Convolution**

**Convoluted Image**

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

23
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image1)

![Convolution](image2)

![Convolved Image](image3)
A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Convolution**

**Convolved Image**

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image)

![Convolution Matrix](image)

![Convolved Image](image)
A convolution matrix is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

Convolved Image

Convolution
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

**Input Image**

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Convolved Image**

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Convolution**

- The convolution process involves sliding the matrix over the input image and performing element-wise multiplications followed by summations.
- The output image is generated by applying this process across the entire input image.
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Convolution Matrix Diagram](image)

- **Input Image**
- **Convolved Image**

![Convolution Example](image)
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

![Input Image](image1)

![Identity Convolution](image2)

![Convolved Image](image3)
Background: Image Processing

A **convolution matrix** is used in image processing for tasks such as edge detection, blurring, sharpening, etc.

Input Image

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Convolved Image

| 0.4 | 0.5 | 0.5 | 0.5 | 0.4 |
| 0.4 | 0.2 | 0.3 | 0.6 | 0.3 |
| 0.5 | 0.4 | 0.4 | 0.2 | 0.1 |
| 0.5 | 0.6 | 0.2 | 0.1 | 0 |
| 0.4 | 0.3 | 0.1 | 0 | 0 |

Blurring Convolution

| 0.1 | 0.1 | 0.1 |
| 0.1 | 0.2 | 0.1 |
| 0.1 | 0.1 | 0.1 |
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo

Slide from William Cohen
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

http://matlabtricks.com/post-5/3x3-convolution-kernels-with-online-demo
What’s a convolution?

• Basic idea:
  – Pick a 3x3 matrix F of weights
  – Slide this over an image and compute the “inner product” (similarity) of F and the corresponding field of the image, and replace the pixel in the center of the field with the output of the inner product operation

• Key point:
  – Different convolutions extract different types of low-level “features” from an image
  – All that we need to vary to generate these different features is the weights of F

Ex: 1 input channel, 1 output channel

\[
\begin{align*}
  y_{11} &= \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_0 \\
  y_{12} &= \alpha_{11} x_{12} + \alpha_{12} x_{13} + \alpha_{21} x_{22} + \alpha_{22} x_{23} + \alpha_0 \\
  y_{21} &= \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{31} + \alpha_{22} x_{32} + \alpha_0 \\
  y_{22} &= \alpha_{11} x_{22} + \alpha_{12} x_{23} + \alpha_{21} x_{32} + \alpha_{22} x_{33} + \alpha_0
\end{align*}
\]

Slide adapted from William Cohen
DOWNNSAMPLING
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output
Downsampling

• Suppose we use a convolution with stride 2
• Only 9 patches visited in input, so only 9 pixels in output
Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

![Convolution Diagram](image-url)
Downsampling

• Suppose we use a convolution with stride 2
• Only 9 patches visited in input, so only 9 pixels in output
Downsampling

• Suppose we use a convolution with stride 2
• Only 9 patches visited in input, so only 9 pixels in output
## Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

### Input Image

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### Convolution

```
1 1
1 1
```

### Convolved Image

<table>
<thead>
<tr>
<th>3</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Downsampling

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

Input Image

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Convolution

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Convolved Image

<table>
<thead>
<tr>
<th>3</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
**Downsampling**

- Suppose we use a convolution with stride 2
- Only 9 patches visited in input, so only 9 pixels in output

<table>
<thead>
<tr>
<th>Input Image</th>
<th>Convolved Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 1 1 1 1 0</td>
<td>3 3 1</td>
</tr>
<tr>
<td>1 0 0 1 0 0 0</td>
<td>3 1 0</td>
</tr>
<tr>
<td>1 0 1 0 0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 1 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td></td>
</tr>
</tbody>
</table>

**Convolution**

\[
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
\end{bmatrix}
\]
Downsampling by Averaging

- Downsampling by averaging is a special case of convolution where the weights are fixed to a uniform distribution.
- The example below uses a stride of 2.

**Input Image**

**Convolved Image**

**Convolution**

```
1 1 1 1 1 0
1 0 0 1 0 0
1 0 1 0 0 0
1 0 1 0 0 0
1 1 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 0
```

```
3/4 3/4 1/4
3/4 1/4 0
1/4 0 0
```

```
1/4 1/4
1/4 1/4
```
Max-Pooling

- Max-pooling is another form of downsampling
- Instead of averaging, we take the max value within the same range as the equivalently-sized convolution
- The example below uses a stride of 2

\[
y_{ij} = \max(x_{ij}, x_{ij+1}, x_{i+1,j}, x_{i+1,j+1})
\]
CONVOLUTIONAL NEURAL NETS
A Recipe for Machine Learning

1. Given training data:
   \[ \{x_i, y_i\}_{i=1}^N \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_\theta(x_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:
   \[ \theta^* = \arg\min_\theta \sum_{i=1}^N \ell(f_\theta(x_i), y_i) \]

4. Train with SGD:
   (take small steps opposite the gradient)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i) \]
1. Given training data:

2. Choose each of these:
   - Decision function
   - Loss function

3. Define goal:

4. Train with SGD:
   - Convolutional Neural Networks (CNNs) provide another form of decision function
   - Let’s see what they look like...

\[
\hat{y} = f_\theta(x_i)
\]

\[
\ell(\hat{y}, y_i) \in \mathbb{R}
\]

\[
\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_\theta(x_i), y_i)
\]
Convolutional Layer

**CNN key idea:**
Treat convolution matrix as parameters and learn them!

---

Input Image

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

Convolved Image

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>.4</td>
<td>.5</td>
<td>.5</td>
<td>.5</td>
<td>.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4</td>
<td>.2</td>
<td>.3</td>
<td>.6</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>.4</td>
<td>.4</td>
<td>.2</td>
<td>.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>.6</td>
<td>.2</td>
<td>.1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.4</td>
<td>.3</td>
<td>.1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax
- These can be arranged into arbitrarily deep topologies

**Architecture #1: LeNet-5**

---

**Fig. 2.** Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e., a set of units whose weights are constrained to be identical.
TRAINING CNNS
1. Given training data:
   \[ \{ x_i, y_i \}_{i=1}^{N} \]

2. Choose each of these:
   - Decision function
     \[ \hat{y} = f_{\theta}(x_i) \]
   - Loss function
     \[ \ell(\hat{y}, y_i) \in \mathbb{R} \]

3. Define goal:
   \[ \theta^* = \arg\min_{\theta} \sum_{i=1}^{N} \ell(f_{\theta}(x_i), y_i) \]

4. Train with SGD:
   (take small steps opposite the gradient)
   \[ \theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \ell(f_{\theta}(x_i), y_i) \]
1. Given training data:

$$\{x_i, y_i\}_{i=1}^{N}$$

2. Choose each of these:

- Decision function
- Loss function

3. Define goal:

$$f_\theta(x_i)$$

Train with SGD:

$$(\text{take small steps \ opposite the gradient})$$

• Q: Now that we have the CNN as a decision function, how do we compute the gradient?

• A: Backpropagation of course!
SGD for CNNs

Example: Architecture:

Given $\tilde{x}, y^*$

$$J = l(y, y^*)$$
$$y = \text{softmax}(z^{(5)})$$
$$z^{(5)} = \text{linear}(z^{(4)}, W)$$
$$z^{(4)} = \text{relu}(z^{(3)})$$
$$z^{(3)} = \text{conv}(z^{(2)}, \beta)$$
$$z^{(2)} = \text{max-pool}(z^{(1)})$$
$$z^{(1)} = \text{conv}(\tilde{x}, \alpha)$$

Parameters $\hat{\theta} = [\alpha, \beta, W]$

SGD:

1. Initialize $\hat{\theta}$
2. While not converged:
   - Simple $i \in \{1, ..., N\}$
   - Forward: $y = h_{\theta}(\tilde{x}^{(i)})$, $J_i(\theta) = l(y, y^*)$
   - Backward: $\nabla_{\hat{\theta}} J_i(\theta) = ...$
   - Update: $\hat{\theta} \leftarrow \hat{\theta} - \lambda \nabla_{\hat{\theta}} J_i(\theta)$
LAYERS OF A CNN
ReLU Layer

Input: \( \tilde{x} \in \mathbb{R}^k \)  
Output: \( \tilde{y} \in \mathbb{R}^k \)

Forward:
\[
\tilde{y} = \sigma(\tilde{x}) \leftarrow \text{element-wise}
\]
\[
\sigma(a) = \max(0, a)
\]

\[
\max(0, a)
\]

\[
\frac{dJ}{dx_i} = \frac{dJ}{dy_i} \frac{dy_i}{dx_i}
\]

where
\[
\frac{dy_i}{dx_i} = \begin{cases} 
1 & \text{if } x_i > 0 \\
0 & \text{otherwise}
\end{cases}
\]
Softmax Layer

Input: \( \hat{x} \in \mathbb{R}^K \)  
Output: \( \hat{y} \in \mathbb{R}^K \)

Forward:
\[
y_i = \frac{\exp(x_i)}{\sum_{k=1}^{K} \exp(x_k)}
\]

Backward:
\[
\frac{dJ}{dx_j} = \sum_{i=1}^{K} \frac{dJ}{dy_i} \frac{dy_i}{dx_j}
\]

where
\[
\frac{dy_i}{dx_j} = \begin{cases} 
  y_i(1-y_i) & \text{if } i = j \\
  -y_i y_j & \text{otherwise}
\end{cases}
\]
Fully-Connected Layer

- Suppose input is a 3D Tensor: $X = [x_1, \ldots, x_i, \ldots, x_{(C \times H \times W)}]$.
- Stretch out into a long vector: $x = \mathbf{x}$.
- Then standard linear layer:

$$y = \alpha^T \mathbf{x} + \alpha_0 \text{ where } \alpha \in \mathbb{R}^{A \times B}, \quad |\mathbf{x}| = A, \quad |y| = B$$
Convolutional Layer

Ex: 1 input channel, 1 output channel

Input
\[
\begin{array}{ccc}
X_{11} & X_{12} & X_{13} \\
X_{21} & X_{22} & X_{23} \\
X_{31} & X_{32} & X_{33}
\end{array}
\]

Conv
\[
\begin{array}{c}
\alpha_1 \\
\alpha_2 \\
\alpha_3 \\
\alpha_4
\end{array}
\]

Output
\[
\begin{array}{c}
y_{11} \\
y_{12} \\
y_{21} \\
y_{22}
\end{array}
\]

\[
\begin{align*}
y_{11} &= \alpha_{11} x_{11} + \alpha_{12} x_{12} + \alpha_{21} x_{21} + \alpha_{22} x_{22} + \alpha_0 \\
y_{12} &= \alpha_{11} x_{12} + \alpha_{13} x_{13} + \alpha_{21} x_{22} + \alpha_{22} x_{23} + \alpha_0 \\
y_{21} &= \alpha_{11} x_{21} + \alpha_{12} x_{22} + \alpha_{21} x_{31} + \alpha_{22} x_{32} + \alpha_0 \\
y_{22} &= \alpha_{11} x_{22} + \alpha_{13} x_{23} + \alpha_{21} x_{32} + \alpha_{22} x_{33} + \alpha_0
\end{align*}
\]

Ex: 1 input channel, 2 output channels

Input
\[
\begin{array}{ccc}
X_{11} & X_{12} & X_{13} \\
X_{21} & X_{22} & X_{23} \\
X_{31} & X_{32} & X_{33}
\end{array}
\]

Conv #1
\[
\begin{array}{c}
\alpha_{11} \\
\alpha_{12} \\
\alpha_{21} \\
\alpha_{22}
\end{array}
\]

Output #1
\[
\begin{array}{c}
y_{11} \\
y_{12} \\
y_{21} \\
y_{22}
\end{array}
\]

\[
\begin{align*}
y_{11}^{(1)} &= \alpha_{11}^{(1)} x_{11} + \alpha_{12}^{(1)} x_{12} + \alpha_{21}^{(1)} x_{21} + \alpha_{22}^{(1)} x_{22} + \alpha_0^{(1)} \\
y_{12}^{(1)} &= \alpha_{11}^{(1)} x_{12} + \alpha_{13}^{(1)} x_{13} + \alpha_{21}^{(1)} x_{22} + \alpha_{22}^{(1)} x_{23} + \alpha_0^{(1)} \\
y_{21}^{(1)} &= \alpha_{11}^{(1)} x_{21} + \alpha_{12}^{(1)} x_{22} + \alpha_{21}^{(1)} x_{31} + \alpha_{22}^{(1)} x_{32} + \alpha_0^{(1)} \\
y_{22}^{(1)} &= \alpha_{11}^{(1)} x_{22} + \alpha_{13}^{(1)} x_{23} + \alpha_{21}^{(1)} x_{32} + \alpha_{22}^{(1)} x_{33} + \alpha_0^{(1)}
\end{align*}
\]

Conv #2
\[
\begin{array}{c}
\alpha_{11} \\
\alpha_{12} \\
\alpha_{21} \\
\alpha_{22}
\end{array}
\]

Output #2
\[
\begin{array}{c}
y_{11} \\
y_{12} \\
y_{21} \\
y_{22}
\end{array}
\]

\[
\begin{align*}
y_{11}^{(2)} &= \alpha_{11}^{(2)} x_{11} + \alpha_{12}^{(2)} x_{12} + \alpha_{21}^{(2)} x_{21} + \alpha_{22}^{(2)} x_{22} + \alpha_0^{(2)} \\
y_{12}^{(2)} &= \alpha_{11}^{(2)} x_{12} + \alpha_{13}^{(2)} x_{13} + \alpha_{21}^{(2)} x_{22} + \alpha_{22}^{(2)} x_{23} + \alpha_0^{(2)} \\
y_{21}^{(2)} &= \alpha_{11}^{(2)} x_{21} + \alpha_{12}^{(2)} x_{22} + \alpha_{21}^{(2)} x_{31} + \alpha_{22}^{(2)} x_{32} + \alpha_0^{(2)} \\
y_{22}^{(2)} &= \alpha_{11}^{(2)} x_{22} + \alpha_{13}^{(2)} x_{23} + \alpha_{21}^{(2)} x_{32} + \alpha_{22}^{(2)} x_{33} + \alpha_0^{(2)}
\end{align*}
\]
Convolutional Layer

\[ \text{Ex: } C^I \text{ input channels, } C^O \text{ output channels} \]

\[ \text{Input} \rightarrow \text{Conv}^1 \rightarrow \text{Output} \]

\[ \text{Conv}^n \rightarrow \text{Conv}^{n+1} \]

\[ H^O = \left( H^I + 2p - K \right)/s + 1 \]

\[ W^O = \left( W^I + 2p - K \right)/s + 1 \]

where \( p \) = # pixels of padding on input

\( K \) = size of conv. matrix

\( s \) = stride length

Forward:

\[ y^{(k)}_{ij} = x_0^{(k)} + \sum_{c=1}^{C^I} \sum_{q=1}^{K} \sum_{r=1}^{K} w^{(k)}_{r,c} x^{(c)}_{mn} \quad \text{where} \quad m = s(i-1) + q, \]

\[ n = s(j-1) + r \]

Backward:

\[ \frac{dJ}{dx_0^{(k)}} = \sum_i \sum_j \frac{dJ}{y^{(k)}_{ij}} \frac{dx_0^{(k)}}{dJ_{ij}} \]

\[ \frac{dJ}{dx_{q,c}^{(k)}} = \sum_i \sum_j \frac{dJ}{y^{(k)}_{ij}} \frac{dx_{q,c}^{(k)}}{dJ_{ij}} \]

\[ \frac{dJ}{dx^{(c)}_{mn}} = \sum_i \sum_j \frac{dJ}{y^{(k)}_{ij}} \frac{dx^{(c)}_{mn}}{dJ_{ij}} \]

Just some calculus.
Max-Pooling Layer

Ex: 1 input channel, 1 output channel, stride of 1

Input
\[
\begin{array}{ccc}
X_{11} & X_{12} & X_{13} \\
X_{21} & X_{22} & X_{23} \\
X_{31} & X_{32} & X_{33}
\end{array}
\]

Pool Size

Output
\[
\begin{array}{cc}
Y_{11} & Y_{12} \\
Y_{21} & Y_{22}
\end{array}
\]

\[
\begin{align*}
Y_{11} &= \max \left( X_{11}, \ldots, X_{12}, X_{21}, \ldots, X_{23} \right) \\
Y_{11} &= \max \left( X_{11}, \ldots, X_{12}, X_{21}, \ldots, X_{23} \right) \\
Y_{21} &= \max \left( X_{21}, \ldots, X_{22}, X_{31}, \ldots, X_{33} \right) \\
Y_{22} &= \max \left( X_{22}, \ldots, X_{23}, X_{32}, \ldots, X_{33} \right)
\end{align*}
\]
Max-Pooling Layer

Forward:
\[ Y_{ij}^{(k)} = \max_{q \in \mathbb{K}, \ell \in \mathbb{K}} X_{m n}^{(k)} \] where \( m = s(i-1)+q \), \( n = s(j-1)+r \)

Backward:
\[ \frac{dJ}{dx_{mn}^{(k)}} = \sum_i \sum_j \frac{dJ}{dy_{ij}^{(k)}} \frac{dy_{ij}^{(k)}}{dx_{mn}^{(k)}} \]

Subderivatives
+ Max() is not differentiable, but subdifferentiable.
+ There are a set of derivatives and we can just choose one for SGD.

\[ y = \max(a, b) \]

\[ \Rightarrow \frac{dJ}{da} = \frac{dJ}{dy} \frac{dy}{da} \] where \( \frac{dy}{da} = \begin{cases} 1 & \text{if } a > b \\ 0 & \text{otherwise} \end{cases} \)
Convolutional Neural Network (CNN)

- Typical layers include:
  - Convolutional layer
  - Max-pooling layer
  - Fully-connected (Linear) layer
  - ReLU layer (or some other nonlinear activation function)
  - Softmax

- These can be arranged into arbitrarily deep topologies

**Architecture #1: LeNet-5**

![Architecture of LeNet-5](image)

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.
Architecture #2: AlexNet

**CNN for Image Classification**
(Krizhevsky, Sutskever & Hinton, 2012)
15.3% error on ImageNet LSVRC-2012 contest

- Five convolutional layers (w/max-pooling)
- Three fully connected layers

![Diagram of the AlexNet architecture](image)

- Input image (pixels)
- 1000-way softmax

**Input image (pixels)**
- 224 \times 224 \times 3-dimensional.

**Five convolutional layers (w/max-pooling)**
- Max pooling

**Three fully connected layers**
- 1000-way softmax

**1000-way softmax**
CNNs for Image Recognition

Revolution of Depth

ImageNet Classification top-5 error (%)

ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

CNN VISUALIZATIONS
3D Visualization of CNN

http://scs.ryerson.ca/~aharley/vis/conv/
Convolution of a Color Image

- Color images consist of 3 floats per pixel for RGB (red, green blue) color values.
- Convolution must also be 3-dimensional.
Animation of 3D Convolution

http://cs231n.github.io/convolutional-networks/

Figure from Fei-Fei Li & Andrej Karpathy & Justin Johnson (CS231N)
### MNIST Digit Recognition with CNNs (in your browser)

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

#### Network Visualization

<table>
<thead>
<tr>
<th>Layer</th>
<th>Configuration</th>
<th>Max Activation</th>
<th>Min Activation</th>
<th>Max Gradient</th>
<th>Min Gradient</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input (2x2x1)</td>
<td>max activation: 1, min: 0</td>
<td></td>
<td></td>
<td>max gradient: 0.00015, min: -0.00014</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conv (2x2x8)</td>
<td>filter size 5x5x1, stride 1</td>
<td>4.78388, min: -3.44063</td>
<td></td>
<td>max gradient: 0.00005, min: -0.00006</td>
<td>parameters: 8x5x5x1+8 = 208</td>
<td></td>
</tr>
<tr>
<td>Softmax (1x1x10)</td>
<td>max activation: 0.99768, min: 0</td>
<td></td>
<td></td>
<td>max gradient: 0, min: 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Example predictions on Test set

<table>
<thead>
<tr>
<th>Prediction</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="example_predictions.png" alt="Image of predictions" /></td>
<td><img src="actual_digits.png" alt="Image of actual digits" /></td>
</tr>
</tbody>
</table>
CNN Summary

CNNs

– Are used for all aspects of computer vision, and have won numerous pattern recognition competitions
– Able learn interpretable features at different levels of abstraction
– Typically, consist of convolution layers, pooling layers, nonlinearities, and fully connected layers

Other Resources:

– Readings on course website
– Andrej Karpathy, CS231n Notes
  http://cs231n.github.io/convolutional-networks/
Deep Learning Objectives

You should be able to...

- Implement the common layers found in Convolutional Neural Networks (CNNs) such as linear layers, convolution layers, max-pooling layers, and rectified linear units (ReLU)
- Explain how the shared parameters of a convolutional layer could learn to detect spatial patterns in an image
- Describe the backpropagation algorithm for a CNN
- Identify the parameter sharing used in a basic recurrent neural network, e.g. an Elman network
- Apply a recurrent neural network to model sequence data
- Differentiate between an RNN and an RNN-LM
ML Big Picture

Learning Paradigms:
What data is available and when? What form of prediction?
- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

Problem Formulation:
What is the structure of our output prediction?
- boolean
  - Binary Classification
- categorical
  - Multiclass Classification
- ordinal
  - Ordinal Classification
- real
  - Regression
- ordering
  - Ranking
- multiple discrete
  - Structured Prediction
- multiple continuous
  - (e.g. dynamical systems)
- both discrete & cont.
  - (e.g. mixed graphical models)

Facets of Building ML Systems:
How to build systems that are robust, efficient, adaptive, effective?
1. Data prep
2. Model selection
3. Training (optimization / search)
4. Hyperparameter tuning on validation data
5. (Blind) Assessment on test data

Big Ideas in ML:
Which are the ideas driving development of the field?
- inductive bias
- generalization / overfitting
- bias-variance decomposition
- generative vs. discriminative
- deep nets, graphical models
- PAC learning
- distant rewards

Theoretical Foundations:
What principles guide learning?
- probabilistic
- information theoretic
- evolutionary search
- ML as optimization

Application Areas
Key challenges?
NLP, Speech, Computer Vision, Robotics, Medicine, Search

Facets of Building ML Systems:
How to build systems that are robust, efficient, adaptive, effective?
1. Data prep
2. Model selection
3. Training (optimization / search)
4. Hyperparameter tuning on validation data
5. (Blind) Assessment on test data
LEARNING THEORY
PAC(-MAN) Learning

For some hypothesis $h \in \mathcal{H}$:

1. True Error
   
   \[ R(h) \]

2. Training Error
   
   \[ \hat{R}(h) \]

Question 2:
What is the expected number of PAC-MAN levels Matt will complete before a Game-Over?

A. 1-10
B. 11-20
C. 21-30
Questions for today (and next lecture)

1. Given a classifier with zero training error, what can we say about true error (aka. generalization error)? (Sample Complexity, Realizable Case)

2. Given a classifier with low training error, what can we say about true error (aka. generalization error)? (Sample Complexity, Agnostic Case)

3. Is there a theoretical justification for regularization to avoid overfitting? (Structural Risk Minimization)
PAC/SLT Model for Supervised ML

\( x^{(i)} \sim p^*(\cdot) \)

\( D_{\text{train}} \)

\( x^{(1)} \)

\( x^{(2)} \)

\( x^{(3)} \)

\( y^{(1)} \)

\( y^{(2)} \)

\( y^{(3)} \)

Learning Algorithm

\( h(x) \)
PAC/SLT Model for Supervised ML

- **Problem Setting**
  - Set of possible inputs, \( x \in \mathcal{X} \) (all possible patients)
  - Set of possible outputs, \( y \in \mathcal{Y} \) (all possible diagnoses)
  - Distribution over instances, \( p^*(\cdot) \)
  - Exists an unknown target function, \( c^*: \mathcal{X} \to \mathcal{Y} \) (the doctor’s brain)
  - Set, \( \mathcal{H} \), of candidate hypothesis functions, \( h: \mathcal{X} \to \mathcal{Y} \) (all possible decision trees)

- **Learner is given** \( N \) training examples
  \( D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\} \)
  where \( x^{(i)} \sim p^*(\cdot) \) and \( y^{(i)} = c^*(x^{(i)}) \)
  (history of patients and their diagnoses)

- **Learner produces** a hypothesis function, \( \hat{y} = h(x) \), that best approximates unknown target function \( y = c^*(x) \) on the training data
PAC/SLT Model for Supervised ML

• **Problem Setting**
  - Set of possible inputs, \( x \in \mathcal{X} \) (all possible patients)
  - Set of possible outputs, \( y \in \mathcal{Y} \) (all possible diagnoses)
  - Distribution over instances, \( p^*(\cdot) \)
  - Exists an unknown target function, \( c^* : \mathcal{X} \rightarrow \mathcal{Y} \)
    (the doctor’s brain)
  - Set, \( \mathcal{H} \), of candidate hypothesis functions, \( h : \mathcal{X} \rightarrow \mathcal{Y} \)
    (all possible decision trees)

• **Learner is given** \( N \) training examples \( D = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\} \) where \( x^{(i)} \sim p^*(\cdot) \) and \( y^{(i)} = c^*(x^{(i)}) \) (history of patients and their diagnoses)

• **Learner produces** a hypothesis function, \( \hat{y} = h(x) \), that best approximates unknown target function \( y = c^*(x) \) on the training data

Two important settings we’ll consider:

1. **Classification**: the possible outputs are **discrete**
2. **Regression**: the possible outputs are **real-valued**
PAC/SLT Model for Supervised ML

$x^{(i)} \sim p^*(\cdot)$

$D_{\text{train}}$

$c^*(x)$

$D_{\text{test}}$

$\hat{y}^{(4)}$

$\hat{y}^{(5)}$

Learning Algorithm

$h(x)$

Predictions

Test Error Rate

$\sim p^*(\cdot)$
Two Types of Error

1. True Error (aka. expected risk)

\[ R(h) = P_{x \sim p^*}(c^*(x) \neq h(x)) \]

2. Train Error (aka. empirical risk)

\[ \hat{R}(h) = P_{x \sim S}(c^*(x) \neq h(x)) \]

\[ = \frac{1}{N} \sum_{i=1}^{N} 1(c^*(x^{(i)}) \neq h(x^{(i)})) \]

\[ = \frac{1}{N} \sum_{i=1}^{N} 1(y^{(i)} \neq h(x^{(i)})) \]

where \( S = \{x^{(1)}, \ldots, x^{(N)}\} \) is the training data set, and \( x \sim S \) denotes that \( x \) is sampled from the empirical distribution.
PAC / SLT Model

1. Generate instances from unknown distribution $p^*$

$$x^{(i)} \sim p^*(x), \forall i \quad (1)$$

2. Oracle labels each instance with unknown function $c^*$

$$y^{(i)} = c^*(x^{(i)}), \forall i \quad (2)$$

3. Learning algorithm chooses hypothesis $h \in \mathcal{H}$ with low(est) training error, $\hat{R}(h)$

$$\hat{h} = \arg\min_h \hat{R}(h) \quad (3)$$

4. Goal: Choose an $h$ with low generalization error $R(h)$
Three Hypotheses of Interest

The **true function** \( c^* \) is the one we are trying to learn and that labeled the training data:

\[
y^{(i)} = c^*(x^{(i)}), \quad \forall i
\]  

(1)

The **expected risk minimizer** has lowest true error:

\[
h^* = \arg\min_{h \in \mathcal{H}} R(h)
\]

Question: True or False: \( h^* \) and \( c^* \) are always equal.

The **empirical risk minimizer** has lowest training error:

\[
\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{R}(h)
\]  

(3)
PAC LEARNING
Probably Approximately Correct (PAC) Learning

Whiteboard:

– PAC Criterion
– Meaning of “Probably Approximately Correct”
– Def: PAC Learner
– Sample Complexity
– Consistent Learner
– Realizable vs. Agnostic Cases
– Finite vs. Infinite Hypothesis Spaces
SAMPLE COMPLEXITY RESULTS
Sample Complexity Results

**Definition 0.1.** The **sample complexity** of a learning algorithm is the number of examples required to achieve arbitrarily small error (with respect to the optimal hypothesis) with high probability (i.e., close to 1).

Four Cases we care about...

We’ll start with the finite case...

| Finite $|\mathcal{H}|$ | Realizable |
|----------------|------------|
| Infinite $|\mathcal{H}|$ | Agnostic   |
Probably Approximately Correct (PAC) Learning

Whiteboard:

– Theorem 1: Realizable Case, Finite $|H|$
– Proof of Theorem 1
Sample Complexity Results

**Definition 0.1.** The *sample complexity* of a learning algorithm is the number of examples required to achieve arbitrarily small error (with respect to the optimal hypothesis) with high probability (i.e. close to 1).

**Four Cases we care about...**

| Finite $|\mathcal{H}|$ | Realizable | Agnostic |
|----------------------|------------|----------|
| $\text{Thm. 1} \quad N \geq \frac{1}{\epsilon} \left[ \log(|\mathcal{H}|) + \log\left(\frac{1}{\delta}\right) \right]$ labeled examples are sufficient so that with probability $(1 - \delta)$ all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \epsilon$. | | |

| Infinite $|\mathcal{H}|$ | | |
| | | |
Example: Conjunctions

Question:
Suppose $H = \text{class of conjunctions over } x \text{ in } \{0, 1\}^M$

Example hypotheses:
- $h(x) = x_1 (1-x_3) x_5$
- $h(x) = x_1 (1-x_2) x_4 (1-x_5)$

If $M = 10$, $\varepsilon = 0.1$, $\delta = 0.01$, how many examples suffice according to Theorem 1?

Answer:
A. $10*(2*\ln(10)+\ln(100)) \approx 92$
B. $10*(3*\ln(10)+\ln(100)) \approx 116$
C. $10*(10*\ln(2)+\ln(100)) \approx 116$
D. $10*(10*\ln(3)+\ln(100)) \approx 156$
E. $100*(2*\ln(10)+\ln(10)) \approx 691$
F. $100*(3*\ln(10)+\ln(10)) \approx 922$
G. $100*(10*\ln(2)+\ln(10)) \approx 924$
H. $100*(10*\ln(3)+\ln(10)) \approx 1329$

Thm. 1 $N \geq \frac{1}{\varepsilon} \left[ \log(|\mathcal{H}|) + \log\left(\frac{1}{\delta}\right) \right]$ labeled examples are sufficient so that with probability $(1-\delta)$ all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \varepsilon$. 