Lecture 18: Deep Learning

Many slides adapted from Stanford's CS231N by Fei-Fei Li, Justin Johnson, Serena Yeung, as well as Slides by Marc'Aurelio Ranzato (NYU), Dhruv Batra & Devi Parikh (Georgia Tech)
Topics

1. Supervised Learning
2. Convolutional Neural Networks
3. Learning CNN Parameters
Motivation

- Robotics:
  - Perception, thinking, acting
- Deep learning has revolutionized perception
- Getting increasingly important in thinking/acting
- This lecture:
  - High-level intro to CNNs and learning for perception
- Next lecture:
  - Applications in robotics
1. Supervised Learning

- Example: classification

(assume given set of discrete labels) 
{dog, cat, truck, plane, ...}
The Problem: Semantic Gap

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)
An image classifier

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for recognizing a cat, or other classes.
ML: A Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
2. Convolutional Neural Networks
Example: 200x200 image
40K hidden units
~2.4B parameters!!!

- Spatial correlation is local
- Waste of resources + we do not have enough training samples anyway..

Slide Credit: Marc'Aurelio Ranzato
Locally Connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

Slide Credit: Marc'Aurelio Ranzato
Locally Connected Layer

**STATIONARITY?** Statistics is similar at different locations

Slide Credit: Marc'Aurelio Ranzato
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary): Convolutions with learned kernels

Slide Credit: Marc'Aurelio Ranzato
Convolutional Layer

\[
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix}
\ast
\begin{bmatrix}
-1 & 0 & 1 \\
-1 & 0 & 1 \\
-1 & 0 & 1 \\
\end{bmatrix}
= 
\]

Slide Credit: Marc'Aurelio Ranzato
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolution Layer

32x32x3 image -> preserve spatial structure
Convolution Layer

- 32x32x3 image
- 5x5x3 filter
- convolve (slide) over all spatial locations

activation map

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Multiple filters: if we have 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!

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

- **3** channels, **32** filters
- **CONV**, **ReLU**
- Example: **6** 5x5x3 filters

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

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

Layer 1

Layer 2

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

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

Figure Credit: [Zeiler & Fergus ECCV14]
two more layers to go: POOL/FC
Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
Example: 200x200 image
40K hidden units

~2B parameters!!!

- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Slide Credit: Marc'Aurelio Ranzato
3. Learning CNN Parameters

Suppose: 3 training examples, 3 classes. With some \( W \) the scores \( f(x, W) = Wx \) are:

<table>
<thead>
<tr>
<th></th>
<th>cat</th>
<th>car</th>
<th>frog</th>
</tr>
</thead>
<tbody>
<tr>
<td>score</td>
<td>3.2</td>
<td>5.1</td>
<td>-1.7</td>
</tr>
<tr>
<td></td>
<td>1.3</td>
<td>4.9</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>2.2</td>
<td>2.5</td>
<td>-3.1</td>
</tr>
</tbody>
</table>

A **loss function** tells how good our current classifier is.

Given a dataset of examples

\[
\{(x_i, y_i)\}_{i=1}^{N}
\]

Where \( x_i \) is image and \( y_i \) is (integer) label.

Loss over the dataset is a sum of loss over examples:

\[
L = \frac{1}{N} \sum_i L_i(f(x_i, W), y_i)
\]
How to minimize the loss by changing the weights?

Strategy: Follow the slope of the loss function
Strategy: **Follow the slope**

In 1-dimensional, the derivative of a function:

\[
\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x + h) - f(x)}{h}
\]

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension.

The slope in any direction is the **dot product** of the direction with the gradient.

The direction of steepest descent is the **negative gradient**.
Gradient Descent

```python
# Vanilla Gradient Descent

while True:
    weights_grad = evaluate_gradient(loss_fun, data, weights)
    weights += - step_size * weights_grad  # perform parameter update
```
Stochastic Gradient Descent (SGD)

$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Full sum expensive when $N$ is large!

Approximate sum using a minibatch of examples
32 / 64 / 128 common

# Vanilla Minibatch Gradient Descent

```
while True:
    data_batch = sample_training_data(data, 256)  # sample 256 examples
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
    weights += -step_size * weights_grad  # perform parameter update
```
How do we compute gradients?

• Analytic or “Manual” Differentiation
• Symbolic Differentiation
• Numerical Differentiation
• Automatic Differentiation!
  • Forward mode AD
  • Reverse mode AD
    • aka “backpropagation”
• Implemented in specialized frameworks:
  • pytorch (Facebook)
  • TensorFlow (Google) frameworks
• Main computation, mainly done on GPU (or TPU)