CS 558: Computer Vision
12th Set of Notes

Instructor: Philippos Mordohai
Webpage: www.cs.stevens.edu/~mordohai
E-mail: Philipppos.Mordohai@stevens.edu
Office: Lieb 215
Overview

• Deep Learning for Computer Vision

  – Based on slides by M. Ranzato (mainly), S. Lazebnik, R. Fergus and Q. Zhang
Natural Neurons

• Human recognition of digits
  – visual cortices
  – neuron interaction
Recognizing Handwritten Digits

• How to describe a digit to a computer
  – "a 9 has a loop at the top, and a vertical stroke in the bottom right“
  – Algorithmically difficult to describe various 9s
Perceptrons

• Perceptrons
  • 1950s ~ 1960s, Frank Rosenblatt, inspired by earlier work by Warren McCulloch and Walter Pitts

• Standard model of artificial neurons
Binary Perceptrons

• Inputs
  • Multiple binary inputs

• Parameters
  • Thresholds & weights

• Outputs
  • Thresholded weighted linear combination

\[
\text{output} = \begin{cases} 
0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\
1 & \text{if } \sum_j w_j x_j > \text{threshold} 
\end{cases}
\]
Layered Perceptrons

• Layered, complex model
  • 1st layer, 2nd layer of perceptrons

• Perceptron rule
  • Weights, thresholds

• Similarity to logical functions (NAND)
Sigmoid Neurons

• Sigmoid neurons
  • Stability
    • Small perturbation, small output change
  • Continuous inputs
  • Continuous outputs
  • Soft thresholds
Output Functions

• Sigmoid neurons

• Output $\sigma(w \cdot x + b), \quad \sigma(z) \equiv \frac{1}{1 + e^{-z}}$

$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$.

• Sigmoid vs conventional thresholds
Smoothness & Differentiability

• Perturbations and Derivatives
  • Continuous function
  • Differentiable
• Layers
  • Input layers, output layers, hidden layers
Layer Structure Design

• Design of hidden layer
  • Heuristic rules
  • Number of hidden layers vs. computational resources
  • Feedforward network
    • No loops involved
Cost Function & Optimization

- Learning with gradient descent
  - Cost function
  - Euclidean loss
  - Non-negative, smooth, differentiable

\[
C'(w, b) \equiv \frac{1}{2n} \sum_x \|y(x) - a\|^2
\]
Cost Function & Optimization

• Gradient Descent

\[ \Delta C' \approx \frac{\partial C'}{\partial v_1} \Delta v_1 + \frac{\partial C'}{\partial v_2} \Delta v_2. \]

• Gradient vector

\[ \nabla C \equiv \left( \frac{\partial C'}{\partial v_1}, \frac{\partial C'}{\partial v_2} \right)^T. \]

\[ \Delta C' \approx \nabla C \cdot \Delta v. \]

\[ v \rightarrow v' = v - \eta \nabla C. \]
Cost Function & Optimization

- Extension to multiple dimension
  - $m$ variables $v_1, \ldots, v_m$
  - Small change in variable $\Delta v = (\Delta v_1, \ldots, \Delta v_m)^T$
  - Small change in cost $\Delta C \approx \nabla C \cdot \Delta v$,

$$\nabla C \equiv \left( \frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m} \right)^T$$

$$\Delta v = -\eta \nabla C \quad v \rightarrow v' = v - \eta \nabla C.$$
Neural Nets for Vision

Based on Tutorials at CVPR 2012 and 2014 by Marc’Aurelio Ranzato
Building an Object Recognition System

IDEA: Use data to optimize features for the given task
Building an Object Recognition System

What we want: Use parameterized function such that
a) features are computed efficiently
b) features can be trained efficiently
Building an Object Recognition System

- Everything becomes adaptive
- No distinction between feature extractor and classifier
- Big non-linear system trained from raw pixels to labels
Q: How can we build such a highly non-linear system?
A: By combining simple building blocks we can make more and more complex systems.
Building a Complicated Function

- Function composition is at the core of deep learning methods
- Each “simple function” will have parameters subject to training

Simple Functions

- \( \sin(x) \)
- \( \cos(x) \)
- \( \log(x) \)
- \( x^3 \)
- \( \exp(x) \)

One Example of Complicated Function

\[ \log(\cos(\exp(\sin^3(x)))) \]
Building a Complicated Function

• Function composition is at the core of deep learning methods
• Each “simple function” will have parameters subject to training
Implementing a Complicated Function

Complicated Function

$\log(\cos(\exp(\sin^3(x))))$

$\sin(x)$ $\rightarrow$ $x^3$ $\rightarrow$ $\exp(x)$ $\rightarrow$ $\cos(x)$ $\rightarrow$ $\log(x)$
Intuition Behind Deep Neural Nets

“CAR”
Intuition Behind Deep Neural Nets

Each black box can have trainable parameters. Their composition makes a highly non-linear system.
Intuition Behind Deep Neural Nets

System produces hierarchy of features

"CAR"
Intuition Behind Deep Neural Nets
Intuition Behind Deep Neural Nets

"CAR"

[Diagram showing the intuition behind deep neural nets with a car image and a representation of neural network layers.]
Intuition Behind Deep Neural Nets
Key Ideas of Neural Nets

IDEA # 1
Learn features from data

IDEA # 2
Use differentiable functions that produce features efficiently

IDEA # 3
End-to-end learning:
no distinction between feature extractor and classifier

IDEA # 4
“Deep” architectures:
cascade of simpler non-linear modules
Key Questions

• What is the input-output mapping?
• How are parameters trained?
• How computational expensive is it?
• How well does it work?
Supervised Deep Learning

Marc’Aurelio Ranzato
Supervised Learning

\[ \{(x_i, y_i), i=1...P \} \text{ training set} \]

- \( x_i \) i-th input training example
- \( y_i \) i-th target label
- \( P \) number of training examples

- Goal: predict the target label of unseen inputs
Supervised Learning Examples

Classification

Denoising

OCR
Supervised Deep Learning

Classification

Denoising

OCR
Neural Networks

Assumptions (for the next few slides):
• The input image is vectorized (disregard the spatial layout of pixels)
• The target label is discrete (classification)

Question: what class of functions shall we consider to map the input into the output?
Answer: composition of simpler functions.

Follow-up questions: Why not a linear combination? What are the “simpler” functions? What is the interpretation?
Answer: later...
Neural Networks: example

Example of a 2 hidden layer neural network (or 4 layer network, counting also input and output)

- $x$: input
- $h^1$: 1-st layer hidden units
- $h^2$: 2-nd layer hidden units
- $o$: output
Forward Propagation

Forward propagation is the process of computing the output of the network given its input.
Forward Propagation

\[ x \mapsto h^1 = \max(0, W^1 x + b^1) \]

- The non-linearity \( u = \max(0, v) \) is called **ReLU** in the DL literature.
- Each output hidden unit takes as input all the units at the previous layer: each such layer is called “**fully connected**”
Rectified Linear Unit (ReLU)
Forward Propagation

\[ x \xrightarrow{\text{max} \left(0, W^1 x\right)} h^1 \xrightarrow{\text{max} \left(0, W^2 h^1\right)} h^2 \xrightarrow{W^3 h^2} o \]

\[ h^1 \in \mathbb{R}^{N_1}, \quad W^2 \in \mathbb{R}^{N_2 \times N_1}, \quad b^2 \in \mathbb{R}^{N_2}, \quad h^2 \in \mathbb{R}^{N_2} \]

\[ h^2 = \text{max} \left(0, W^2 h^1 + b^2\right) \]

- \( W^2 \) 2\textsuperscript{nd} layer weight matrix or weights
- \( b^2 \) 2\textsuperscript{nd} layer biases
Forward Propagation

\[ x \xrightarrow{\text{max} \ (0, W^1 x)} h^1 \xrightarrow{\text{max} \ (0, W^2 h^1)} h^2 \xrightarrow{W^3 h^2} o \]

\[ h^2 \in \mathbb{R}^{N_2}, \ W^3 \in \mathbb{R}^{N_3 \times N_2}, \ b^3 \in \mathbb{R}^{N_3}, \ o \in \mathbb{R}^{N_3} \]

\[ o = \text{max} \ (0, W^3 h^2 + b^3) \]

\( W^3 \) 3\(^{rd}\) layer weight matrix or weights

\( b^3 \) 3\(^{rd}\) layer biases
Alternative Graphical Representations
Interpretation

• **Question:** Why can't the mapping between layers be linear?
• **Answer:** Because composition of linear functions is a linear function. Neural network would reduce to (1 layer) logistic regression.

• **Question:** What do ReLU layers accomplish?
• **Answer:** Piece-wise linear tiling: mapping is locally linear.
Interpretation

- **Question**: Why do we need many layers?
- **Answer**: When input has hierarchical structure, the use of a hierarchical architecture is potentially more efficient because intermediate computations can be re-used. DL architectures are efficient also because they use **distributed representations** which are shared across classes.

![Image of truck feature with binary representation]
Interpretation

\[ [1 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ ... \ ] \quad \text{motorbike} \]

\[ [0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1 \ 0 \ ... \ ] \quad \text{truck} \]
Interpretation

- Distributed representations
- Feature sharing
- Compositionality
Interpretation

**Question:** What does a hidden unit do?
**Answer:** It can be thought of as a classifier or feature detector.

**Question:** How many layers? How many hidden units?
**Answer:** Cross-validation or hyper-parameter search methods are the answer. In general, the wider and the deeper the network the more complicated the mapping.

**Question:** How do I set the weight matrices?
**Answer:** Weight matrices and biases are learned. First, we need to define a measure of quality of the current mapping. Then, we need to define a procedure to adjust the parameters.
How Good is a Network

- Probability of class $k$ given input (softmax):
  \[
p(c_k = 1|x) = \frac{e^{o_k}}{\sum_{j=1}^{C} e^{o_j}}
\]

- (Per-sample) **Loss**; e.g., negative log-likelihood (good for classification of small number of classes):
  \[
  L(x, y; \theta) = - \sum_j y_j \log p(c_j|x)
  \]
Training

- Learning consists of minimizing the loss (plus some regularization term) w.r.t. parameters over the whole training set.

\[ \theta^* = \arg \min_\theta \sum_{n=1}^{P} L(x^n, y^n; \theta) \]

**Question:** How to minimize a complicated function of the parameters?

**Answer:** Chain rule, a.k.a. **Backpropagation**! That is the procedure to compute gradients of the loss w.r.t. parameters in a multi-layer neural network.
Key Idea: Wiggle to Decrease Loss

Let's say we want to decrease the loss by adjusting $W^1_{i,j}$.

We could consider a very small $\epsilon=1e-6$ and compute:

\[
L(x, y; \theta)
\]

\[
L(x, y; \theta \setminus W^1_{i,j}, W^1_{i,j} + \epsilon)
\]

Then update:

\[
W^1_{i,j} \leftarrow W^1_{i,j} + \epsilon \text{sgn}(L(x, y; \theta) - L(x, y; \theta \setminus W^1_{i,j}, W^1_{i,j} + \epsilon))
\]
Backward Propagation

\[
\frac{\partial L}{\partial W^3} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial W^3}
\]

\[
\frac{\partial L}{\partial h^2} = \frac{\partial L}{\partial o} \frac{\partial o}{\partial h^2}
\]
Backward Propagation

Given \( \frac{\partial L}{\partial h^2} \) we can compute now:

\[
\frac{\partial L}{\partial W^2} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial W^2} \quad \frac{\partial L}{\partial h^1} = \frac{\partial L}{\partial h^2} \frac{\partial h^2}{\partial h^1}
\]
Backward Propagation

Given $\frac{\partial L}{\partial h^1}$ we can compute now:

$$\frac{\partial L}{\partial W^1} = \frac{\partial L}{\partial h^1} \frac{\partial h^1}{\partial W^1}$$
Optimization

Stochastic Gradient Descent

$$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta}, \eta \in (0, 1)$$

Or one of its many variants
Convolutional Neural Networks
Marc’Aurelio Ranzato
Fully Connected Layer

Example: 200x200 image
40K hidden units

~2B parameters!!!

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

Share the same parameters across different locations (assuming input is stationary):

Convolutions with learned kernels
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer

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

Learn multiple filters.

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

\[ h_j^n = \max(0, \sum_{k=1}^{K} h_{k}^{n-1} \ast w_{kj}^n) \]

- output feature map
- input feature map
- kernel

Conv. layer

\[ h_1^n \]
\[ h_2^n \]
Convolutional Layer

\[ h_j^n = \max(0, \sum_{k=1}^{K} h_k^{n-1} \ast w_k^n) \]
Convolutional Layer

\[ h^n_j = \text{max} \left( 0, \sum_{k=1}^{K} h^{n-1}_k * w^n_{kj} \right) \]

output feature map

input feature map

kernel

\[ h^n_1 \]
\[ h^n_2 \]
\[ h^n_3 \]
Convolutional Layer

**Question:** What is the size of the output? What's the computational cost?

**Answer:** It is proportional to the number of filters and depends on the stride. If kernels have size $K \times K$, input has size $D \times D$, stride is 1, and there are $M$ input feature maps and $N$ output feature maps then:
- the input has size $M \times D \times D$
- the output has size $N \times (D-K+1) \times (D-K+1)$
- the kernels have $M \times N \times K \times K$ coefficients (which have to be learned)
- cost: $M \times K \times K \times N \times (D-K+1) \times (D-K+1)$

**Question:** How many feature maps? What's the size of the filters?

**Answer:** Usually, there are more output feature maps than input feature maps. Convolutional layers can increase the number of hidden units by big factors (and are expensive to compute). The size of the filters has to match the size/scale of the patterns we want to detect (task dependent).
Key Ideas

• A standard neural net applied to images:
  – scales quadratically with the size of the input
  – does not leverage stationarity

• Solution:
  – connect each hidden unit to a small patch of the input
  – share the weight across space

• This is called: convolutional layer

• A network with convolutional layers is called convolutional network
Pooling Layer

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**

**Question:** What is the size of the output? What's the computational cost?

**Answer:** The size of the output depends on the stride between the pools. For instance, if pools do not overlap and have size $K \times K$, and the input has size $D \times D$ with $M$ input feature maps, then:
- output is $M \times (D/K) \times (D/K)$
- the computational cost is proportional to the size of the input (negligible compared to a convolutional layer)

**Question:** How should I set the size of the pools?

**Answer:** It depends on how much “invariant” or robust to distortions we want the representation to be. It is best to pool slowly (via a few stacks of conv-pooling layers).
Local Contrast Normalization

\[ h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))} \]
Local Contrast Normalization

$$h^{i+1}(x, y) = \frac{h^i(x, y) - m^i(N(x, y))}{\sigma^i(N(x, y))}$$

We want the same response.
Local Contrast Normalization

\[ h_{i+1, x, y} = \frac{h_{i, x, y} - m_{i, N(x, y)}}{\sigma_{i, N(x, y)}} \]
Local Contrast Normalization

\[ h_{i+1,x,y} = \frac{h_{i,x,y} - m_{i,N(x,y)}}{\sigma_{i,N(x,y)}} \]
ConvNets: Typical Stage

One stage (zoom)

- Convol.
- LCN
- Pooling

Filter Bank
- Rectification + Contrast Normalization
- Pooling

courtesy of K. Kavukcuoglu
ConvNets: Typical Architecture

One stage (zoom)

Whole system

Input Image

1^{\text{st}} \text{ stage} \rightarrow 2^{\text{nd}} \text{ stage} \rightarrow 3^{\text{rd}} \text{ stage} \rightarrow \text{Fully Conn. Layers} \\

\text{Class Labels}
ConvNets: Typical Architecture

Conceptually similar to:
SIFT $\rightarrow$ k-means $\rightarrow$ Pyramid Pooling $\rightarrow$ SVM
Engineered vs. learned features

Convolutional filters are trained in a supervised manner by back-propagating classification error

slide credit: S. Lazebnik
SIFT Descriptor

Image Pixels → Apply gradient filters

Spatial pool (Sum)

Normalize to unit length

Feature Vector

slide credit: R. Fergus
AlexNet

- Similar framework to LeCun’98 but:
  - Bigger model (7 hidden layers, 650,000 units, 60,000,000 params)
  - More data ($10^6$ vs. $10^3$ images)
  - GPU implementation (50x speedup over CPU)
    - Trained on two GPUs for a week

A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012
Conv Nets: Examples

• Pedestrian detection
Conv Nets: Examples

- Scene Parsing
Conv Nets: Examples

- Denoising

![Original Image](image1)
![Noised Image](image2)
![Denoised Image](image3)
Conv Nets: Examples

- Object Detection
Conv Nets: Examples

- Face Verification and Identification (DeepFace)
Conv Nets: Examples

- Regression (DeepPose)