Deep Learning & Computer Vision: Review & Overview

- Neural networks & nodes as features
- Nonlinearity: choices, implications for learning
- Benefits of deep over shallow
- How to train/fit/learn
- New ideas for tackling vision applications
- What if we don’t have much data?
Deep Learning & Computer Vision: Review & Overview

• Neural networks & nodes as features
• Nonlinearity: choices, implications for learning
• Benefits of deep over shallow
• How to train/fit/learn
• New ideas for tackling vision applications
• What if we don’t have much data?
Recall: Supervised ML

Input features $x^{(i)} \in \mathbb{R}^n$

Outputs $y^{(i)} \in \mathcal{Y}$ (e.g. $\mathbb{R}$, $\{-1, +1\}$, $\{1, \ldots, p\}$)

Model parameters $\theta \in \mathbb{R}^k$

Hypothesis function $h_\theta : \mathbb{R}^n \rightarrow \mathbb{R}$

Loss function $\ell : \mathbb{R} \times \mathcal{Y} \rightarrow \mathbb{R}_+$

Machine learning optimization problem

$$\text{minimize} \sum_{i=1}^{m} \ell(h_\theta(x^{(i)}), y^{(i)})$$
Neural Networks for Classification

Can interpret “z”s as features

Output class is computed based on these features instead of directly on input $x$

$$h_{\theta}(x) = f_2(W_2f_1(W_1x + b_1) + b_2)$$
Why Do We Want Hidden “Features”? 

- May help us generalize to new input (easier and more robust to identify people by wearing hats than that pixel346 is red)
  - Can be useful for interpretation
- Can equivalently just be viewed as allowing more complex function class to relate input $x$ and output $y$
Object detection

Where are the objects of interest?
Improvements in Object Detection

Mean Average Precision (VOC 2007)


Statistical Template Matching

HOG: Dalal-Triggs 2005
Improvements in Object Detection

Better Models of Complex Categories

Key Advance: Learn effective features from massive amounts of labeled data and adapt to new tasks with less data.
CONV NETS: EXAMPLES

- Scene Parsing

Farabet et al. “Learning hierarchical features for scene labeling” PAMI 2013
Deep Learning & Computer Vision: Review & Overview

• Neural networks & nodes as features
• Nonlinearity: choices, implications for learning
• Benefits of deep over shallow
• How to train/fit/learn
• New ideas for tackling vision applications
• What if we don’t have much data?
Ideal Features Are Non-Linear

$I_1$ → Ideal Feature Extractor → - club, angle = 90
- man, frontal pose
...

$I_2$ → Ideal Feature Extractor → - club, angle = 270
- man, frontal pose
...

- club, angle = 360
- man, side pose
...
Ideal Features Are Non-Linear

INPUT IS NOT THE AVERAGE!

$I_1$  
- club, angle = 90  
- man, frontal pose  
...

$I_2$  
- club, angle = 270  
- man, frontal pose  
...

- club, angle = 360  
- man, side pose  
...
Ideal Features Are Non-Linear

$I_1$ → Ideal Feature Extractor →
- club, $\text{angle} = 90$
- man, frontal pose
...

$I_2$ → Ideal Feature Extractor →
- club, $\text{angle} = 270$
- man, frontal pose
...

Ideal Feature Extractor
- club, $\text{angle} = 360$
- man, side pose
...
NonLinear Transformation

• Increases expressive power
  – Universal function approximator with 1 hidden layer

• When chaining nonlinear transformations, makes optimization harder
  – Not convex
  – Potentially lots of local optima
Which NonLinear Functions to Use?

Activation Functions

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

2 BIG problems:

Sigmoid
Which NonLinear Functions to Use?

**Activation Functions**

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

**2 BIG problems:**

1. Saturated neurons “kill” the gradients
Which NonLinear Functions to Use?

Activation Functions

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

- Squashes numbers to range \([0, 1]\)
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

2 BIG problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
Consider what happens when the input to a neuron is always positive...

\[ f \left( \sum_{i} w_i x_i + b \right) \]

What can we say about the gradients on \( \mathbf{w} \)?
Consider what happens when the input to a neuron is always positive...

\[ f \left( \sum_i w_i x_i + b \right) \]

What can we say about the gradients on \( \mathbf{w} \)?
Always all positive or all negative :( (this is also why you want zero-mean data!)
Activation Functions

- Squashes numbers to range [-1, 1]
- zero centered (nice)
- still kills gradients when saturated :(

$tanh(x)$
Activation Functions

ReLU

\[ f(x) = \max\{0, x\} \]
Deep Learning & Computer Vision: Review & Overview

- Neural networks & nodes as features
- Nonlinearity: choices, implications for learning
- Benefits of deep over shallow
- How to train/fit/learn
- New ideas for tackling vision applications
- What if we don’t have much data?
Learning Non-Linear Features

Q.: which class of non-linear functions shall we consider?
Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination

$$f(x) \approx \sum_j g_j$$

Proposal #2: composition

$$f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$$
Learning Non-Linear Features

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

Proposal #1: linear combination

- Kernel learning
- Boosting
- ...

Proposal #2: composition

- Deep learning
- Scattering networks (wavelet cascade)
- S.C. Zhou & D. Mumford “grammar”
Theoretician's dilemma: “We can approximate any function as close as we want with shallow architecture. Why would we need deep ones?”

\[ y = \sum_{i=1}^{P} \alpha_i K(X, X^i) \quad y = F(W^1 . F(W^0 . X)) \]

Kernel machines (and 2-layer neural nets) are “universal”.

Deep learning machines

\[ y = F(W^K . F(W^{K-1} . F(\ldots F(W^0 . X)\ldots))) \]

Deep machines are more efficient for representing certain classes of functions, particularly those involved in visual recognition.

They can represent more complex functions with less “hardware”

We need an efficient parameterization of the class of functions that are useful for “AI” tasks (vision, audition, NLP...).
Computer Vision: Earlier Approaches

Solution #1: Freeze first N-1 layers (engineer the features) Essentially turns it into shallow learning
Computer Vision: Now

Optimization is difficult: non-convex, non-linear system

Solution #2: live with it!
It will converge to a local minimum.
It is much more powerful!!
Deep Learning in Practice

Optimization is easy, need to know a few tricks of the trade.

Q: What's the feature extractor? And what's the classifier?
A: No distinction, end-to-end learning!
Deep Learning & Computer Vision: Review & Overview

• Neural networks & nodes as features
• Nonlinearity: choices, implications for learning
• Benefits of deep over shallow
• How to train/fit/learn
• New ideas for tackling vision applications
• What if we don’t have much data?
Stochastic gradient descent for neural networks

Recall that stochastic gradient descent computes gradients with respect to loss on each example, updating parameters as it goes.

\[
\text{function } \text{SGD}(\{(x^{(i)}, y^{(i)})\}, h_\theta, \ell, \alpha) \\
\text{Initialize: } W_j, b_j \leftarrow \text{Random}, j = 1, \ldots, k \\
\text{Repeat until convergence:} \\
\text{For } i = 1, \ldots, m: \\
\text{Compute } \nabla_{W_j, b_j} \ell(h_\theta(x^{(i)}), y^{(i)}), j = 1, \ldots, k - 1 \\
\text{Take gradient steps in all directions:} \\
W_j \leftarrow W_j - \alpha \nabla_{W_j} \ell(h_\theta(x^{(i)}), y^{(i)}), j = 1, \ldots, k \\
b_j \leftarrow b_j - \alpha \nabla_{b_j} \ell(h_\theta(x^{(i)}), y^{(i)}), j = 1, \ldots, k \\
\text{return } \{W_j, b_j\}
\]

So how do we compute the gradients \( \nabla_{W_j, b_j} \ell(h_\theta(x^{(i)}), y^{(i)}) \), this is a complex function of the parameters.
Backpropagation

Backpropagation is a method for computing all the necessary gradients using one “forward pass” (just computing all the values at layers), and one “backward pass” (computing gradients backwards in the network)

The equations sometimes look complex, but it’s just an application of the chain rule of calculus
Deep Learning & Computer Vision: Review & Overview

• Neural networks & nodes as features
• Nonlinearity: choices, implications for learning
• Benefits of deep over shallow
• How to train/fit/learn
• New ideas for tackling vision applications
• What if we don’t have much data?
Convolutional Neural Network

http://cs231n.github.io/
• Each internal node is connected to all nodes in prior layer
• Each node in the same layer independent (separate set of weights)
CNN
- uses volumes

E.g. height and width are the size of the image
Depth = 3 color values per pixel

Output is scores for each of the possible classes
Depth \sim \#\ of\ classes
Height=width=1
FULLY CONNECTED NEURAL NET

Example: 1000x1000 image

How many weight parameters for a single node which is a linear combination of input?
FULLY CONNECTED NEURAL NET

Example: 1000x1000 image
1M hidden units

$10^{12}$ parameters!!!

- Spatial correlation is local
- Better to put resources elsewhere!
LOCALLY CONNECTED NEURAL NET

Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

Filter/Kernel/Receptive field:
input patch which the hidden unit is connected to.
Volumes and Depths

Each node here has the same input but different weight vectors (e.g. computing different features of same input)
Nodes: same structure as before, but input is only over a restricted width and height (but the complete depth) of a prior volume.

Think of node as a filter over input.
What is the size of the volume in the next layer? Depth, Stride, Zero Padding

Each node here has the same input but different weight vectors (e.g. computing different features of same input)
Stride and Zero Padding

Stride:
how far (spatially) move over filter

Zero padding:
how many 0s to add to either side of input layer
Stride and Zero Padding

Stride: how far (spatially) move over filter
Zero padding: how many 0s to add to either side of input layer
What is the Stride and the Values in the Second Example?
LOCALLY CONNECTED NEURAL NET

STATIONARITY? Statistics are similar at different locations (translation invariance)
CONVOLUTIONAL NET

Share the same parameters across different locations:
Convolutions with learned kernels
CONVOLUTIONAL NET

Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters
CONVOLUTIONAL LAYER

Input feature maps

3D kernel (filter)

output feature map

Ranzato
CONVOLUTIONAL LAYER

NOTE: The number of output feature maps is usually larger than the number of input feature maps.
Special Layers

- Pooling
- Contrast Normalization (No More Used It Seems)
How many model parameters does this introduce into learning model?
Last Layer, Fully Connected
Deep Learning & Computer Vision: Review & Overview

- Neural networks & nodes as features
- Nonlinearity: choices, implications for learning
- Benefits of deep over shallow
- How to train/fit/learn
- New ideas for tackling vision applications
- What if we don’t have much data?
• Want to build a “is parking spot 3A outside of Wean Hall free” detector
• Have 100 pictures (1 an hr, taken over last few days) with labels as free or not free

• Can deep learning help?
Yes! Transfer Learning
Yes! Transfer Learning

• Use features from a really large dataset (e.g. N-1 layers of CNN) and just retrain final fully connected layer

• Start from an existing trained CNN and then train a bit further
Predicting Poverty Using Deep Transfer Learning (Ermon & colleagues)
What You Should Know

• Neural networks & nodes as features
  – Internal nodes can be viewed as features
  – Make more complicate function mapping input to output

• Benefits of deep over shallow
  – Number of parameters need to express complicated function may be way smaller
  – Important in terms of amount of data to train / fit classifier

• Nonlinearity: choices, implications for learning
  – Sigmoid (bad), ReLu (good)
  – Increases expressive power (1 hidden layer, universal approximator)
  – Optimization harder (not convex, many local optima)

• How to train/fit/learn
  – Gradient descent, backpropagation
  – Be able to derive gradient for simple case and use to update w

• New ideas for tackling vision applications
  – Convolutional networks
  – Reduce # parameters, exploit nodes as filters
  – How many parameters are involved?
  – Define common node types: conv, pooling, fully connected

• What if we don’t have much data?
  – Transfer learning!
  – Learn features using big data, then use for other applications