CS 335: Neural Networks

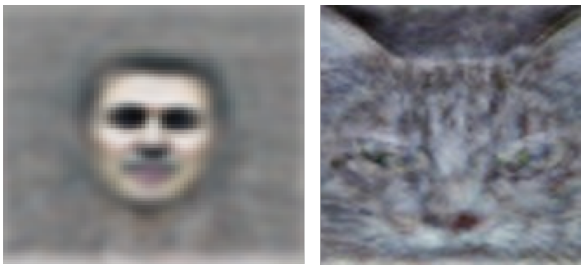
Dan Sheldon

Neural Networks

- Still seeking flexible, non-linear models for classfication and regression
- Enter Neural Networks!
  - Originally brain inspired
  - Can (and will) avoid brain analogies: non-linear functions defined by multiple levels of "feed-forward" computation
  - Very popular and effective right now
    - Attaining human-level performance on variety of tasks
    - "Deep learning revolution"

Deep Learning Revolution

- Resurgence of interest in neural nets ("deep learning") starting in 2006 [Hinton and Salakhutdinov 2006]
- Notable studies starting in early 2010s



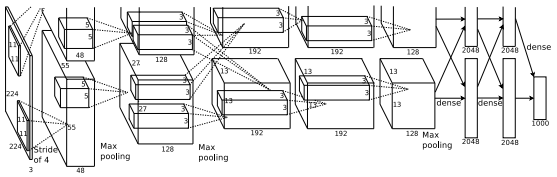
Building High-level Features Using Large Scale Unsupervised Learning [Le et al. 2011]

Deep Learning Revolution

- Neural nets begin dominating the field of image classification

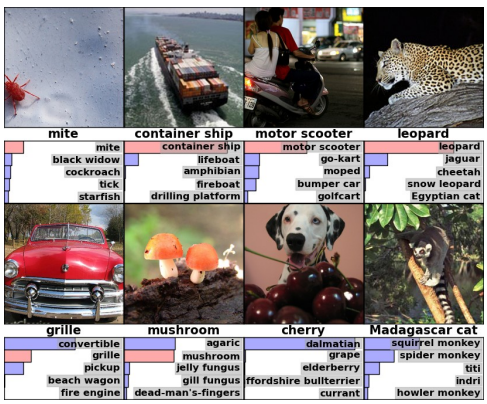
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky University of Toronto kriz@cs.utoronto.ca
Ilya Sutskever University of Toronto ilya@cs.utoronto.ca
Geoffrey E. Hinton University of Toronto hinton@cs.utoronto.ca



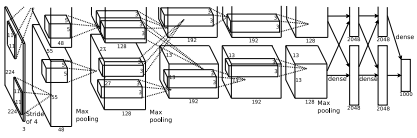
Deep Learning Revolution

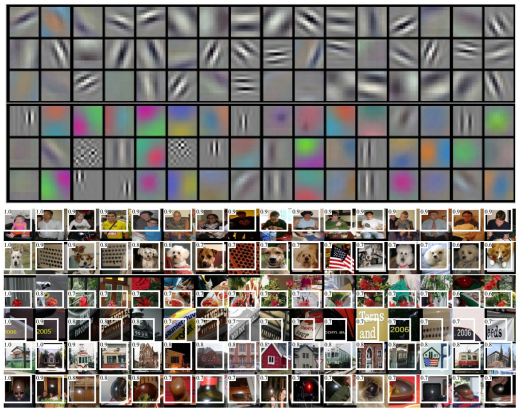
- Recognize hundreds of different objects in images



Deep Learning Revolution

- Learn "feature hierarchies" from raw pixels. Eliminate feature engineering for image classification



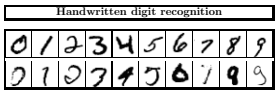


Deep Learning Revolution

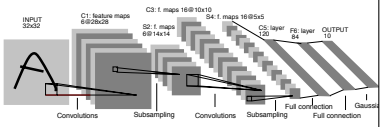
- Deep learning has revolutionized the field of computer vision (image classfication, etc.) in last 7 years
- It is having a similar impact in other domains:
  - Speech recognition
  - Natural language processing
  - Etc.

Some History

- "Shallow" networks in hardware: late 1950s
  - Perceptron: Rosenblatt ~1957
  - Adaline/Madaline: Widrow and Hoff ~1960
- Backprop (key algorithmic principle) popularized in 1986 by Rumelhart et al.
- "Convolutional" neural networks: "LeNet" [LeCun et al. 1998]



3-nearest-neighbor = 2.4% error
400–300–10 unit MLP = 1.6% error
LeNet: 768–192–30–10 unit MLP = 0.9% error



Why Now?

- Ideas have been around for many years. Why did "revolution" happen so recently?
- Massive training sets (e.g. 1 million images)
- Computation: GPUs
- Tricks to avoid overfitting, improve training

What is a Neural Network?

- Biological view: a model of neurons in the brain
- Mathematical view
  - Flexible class of non-linear functions with many parameters
  - Compositional: sequence of many layers
  - Easy to compute
    - $h(\mathbf{x})$: "feed-forward"
    - $\nabla_{\boldsymbol{\theta}} h(\mathbf{x})$: "backward propagation"

Neural Networks: Key Conceptual Ideas

1. Feed-forward computation
2. Backprop (compute gradients)
3. Stochastic gradient descent

Today: feed-forward computation and backprop

Feed-Forward Computation

Multi-class logistic regression:

$$h_{W,\mathbf{b}}(\mathbf{x}) = W\mathbf{x} + \mathbf{b}$$

Parameters:

- $W \in \mathbb{R}^{c \times n}$ weights
- $\mathbf{b} \in \mathbb{R}^c$ biases / intercepts

Output:

- $h(\mathbf{x}) \in \mathbb{R}^c$ = vector of class scores (before logistic transform)

draw picture

Feed-Forward Computation

Multiple layers

$$h(\mathbf{x}) = W_2 \cdot g(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2, \qquad \begin{array}{l} W_1 \in \mathbb{R}^{n_2 \times n_1}, \mathbf{b}_1 \in \mathbb{R}^{n_2} \\ W_2 \in \mathbb{R}^{c \times n_2}, \mathbf{b}_2 \in \mathbb{R}^c \end{array}$$

- $g(\cdot)$ = nonlinear tranformation (e.g., logistic). "nonlinearity"
- $\mathbf{h} = g(W_1\mathbf{x} + \mathbf{b}_1) \in \mathbb{R}^d$ = "hidden" layer

draw picture

- Q: why do we need $g(\cdot)$?

Deep Learning

$$f(\mathbf{x}) = W_3 \cdot g\Big(W_2 \cdot g(W_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2\Big) + \mathbf{b}_3$$

Feed-Forward Computation

General idea

- Write down complex models composed of many layers
  - Linear tranformations (e.g., $W_1\mathbf{x} + \mathbf{b}$)
  - Non-linearity $g(\cdot)$
- Write down loss function for outputs
- Optimize by (some flavor of) gradient descent

How to compute gradient? Backprop!

Backprop

- Boardwork: computation graphs, forward propagation, backprop
- Code demos

Backprop: Details

Input variables $v_1, \ldots, v_k$
Assigned variables $v_{k+1}, \ldots, v_n$ (includes ouput)

Forward propagation:

- For $j = k+1$ to n
  - $v_j = \phi_j(\cdots)$ (local function of predecessors $v_i : i \to j$)

Backward propagation: compute $\bar{v}_i = \frac{dv_n}{dv_i}$ for all i

- Initialize $\bar{v}_n = 1$
- Initialize $\bar{v}_i = 0$ for all $i < n$
- For $j = n$ down to $k+1$
  - For all i such that $i \to j$
  - $\bar{v}_i \mathrel{+}= \frac{d}{dv_i}\phi_j(\cdots) \cdot \bar{v}_j$

Stochastic Gradient Descent (SGD)

Setup

- Complex model $h_{\boldsymbol{\theta}}(\mathbf{x})$ with parameters $\boldsymbol{\theta}$
- Cost function

$$J(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$$
$$\approx \frac{1}{|B|}\sum_{i\in B} \text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$$

  - B = random "batch" of training examples (e.g., $|B| \in \{50, 100, \ldots\}$), or even a single example ($|B| = 1$)

Stochastic Gradient Descent (SGD) Algorithm

- Initialize $\boldsymbol{\theta}$ arbitrarily
- Repeat
  - Pick random batch B
  - Update

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \cdot \frac{1}{|B|}\sum_{i\in B} \nabla_{\boldsymbol{\theta}} \text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$$

- In practice, randomize order of training examples and process sequentially
- Discuss. Advantages of SGD?

Stochastic Gradient Descent Discussion: Summary

- This is the same as gradient descent, except we approximate the gradient using only training examples in batch
- It lets us take many steps for each pass through the data set instead of one.
- In practice, *much* faster for large training sets (e.g., $m = 1,000,000$)

How do we use Backprop to train a Neural Net?

- Idea: think of neural network as a feed-forward model to compute $\text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$ for a single training example
- Append node for cost of prediction on $\mathbf{x}^{(i)}$ to final outputs of network
- Illustration
- Use backprop to compute $\nabla_{\boldsymbol{\theta}} \text{cost}(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}), y^{(i)})$
- This is all there is conceptually, but there are a many implementation details and tricks to do this effectively and efficiently. These are accessible to you, but outside the scope of this class.

The Future of ML: Design Models, Not Algorithms

- Backprop can be automated!
  - You specify the model and loss function
  - Optimizer (e.g., SGD) computes gradient and updates model parameters
  - Suggestions: autograd, PyTorch
  - Demo: train a neural net with autograd
- Next steps
  - Learn more about neural network architectures
    examples: other slides
  - Code a fully connected neural network with one or two hidden layers yourself
  - Experiment with more complex architectures using autograd or PyTorch