STORY SO FAR

- Machine Learning
  - Learn input-output relationships from data
  - Algorithm design by trial and error
  - Preferred approach for very ill-posed problems

- Learning by Optimization
  - Select a function from a hypothesis space
  - Typically translates to learning parameters \( \theta \) for a parametric form \( y = f(x; \theta) \)
  - Find \( \theta \) that minimizes loss / error on training set (but be careful of overfitting)
  - In simple cases, closed form solution for \( \theta \)
  - In the more general case, iterative optimization

- Gradient Descent
  - Compute gradients / partial derivatives of error wrt individual parameters
  - Update parameters by moving in opposite direction
  - Guarantees if loss is a convex function of parameters
  - But can be used generally for arbitrary functional forms and losses
  - Stochastic versions of computational efficiency

CORE SEMANTIC TASKS IN VISION

- Image "Classification"
- Object Detection
- Semantic Segmentation

STORY SO FAR

- AutoGrad
  - Represent a complex function as a composition of simpler functions
  - Build routines that can backpropagate gradients through each simple function, given
    - Gradient at output
    - Values of inputs
  - Automated framework for computing gradients, and therefore gradient descent

- Choose hypothesis space / parametric form / network architecture
  - So that it can represent all the computation required to solve the problem
  - Has as few parameters as possible so that the optimization problem is easier
  - Has "healthy gradient flow"—gradients do not vanish (or blow up)

Now, let's look at some of the semantic vision tasks we apply this to.
**Image Classification:** Given an image, classify it as being one of N categories

**Scene Type**

Aquarium

**Scene Type**

Art Gallery

**Objects in Scenes**

grille  mushroom  cherry  Madagascar cat

But there might be more than one 'obvious' answer for an image
**CORE SEMANTIC TASKS IN VISION**

**Detection:** Check if an object exists in an image, and if so, draw a box around it.

Output: List of \( [x_1, y_1, x_2, y_2] \) co-ordinates +
class labels.

But a box is too coarse ...

**Semantic Segmentation:** Label the class of each pixel.

**Instance Segmentation:** Separate pixels of the same class into instances.
CORE SEMANTIC TASKS IN VISION

Human Pose Estimation

Generally, detect parts or keypoints of objects.

Face Recognition

Determine identity by matching input image to individual in database.

N-way classification

N = no of people in the database

Core Semantic Tasks in Vision

Face / Person Re-identification

Determine that two images are of the same person

Recognize Expressions from Faces
CORE SEMANTIC TASKS IN VISION

Action / Activity Recognition from Videos

CORE SEMANTIC TASKS IN VISION

New Semantic Tasks: Many at the Intersection of Vision and Language

Image Captioning (from COCO challenge)

CORE SEMANTIC TASKS IN VISION

New Semantic Tasks: Many at the Intersection of Vision and Language

Visual Question Answering (visualqa.org)

CORE SEMANTIC TASKS IN VISION

Semantic Tasks: Need to be evaluated (and trained) on data

Progress: Move from anecdata to data

The PASCAL Visual Object Classes Challenge 2007
Semantic Tasks: Need to be evaluated (and trained) on data

- Huge effort to label a large number of images (using Amazon MTurk)
- Labels were from "Word-net" words
- It was thought that having a taxonomy of words would be helpful for visual semantics (turns out not).

"Imagenet" Classification Challenge

1000 classes x 1300 images per class = 1.3 Million Images

Changed the nature of ML methods.
CORE SEMANTIC TASKS IN VISION

THE EFFECT OF DATA

- Older ML Methods designed for small training sets
  - Used more complex optimization methods (than gradient descent): second order methods, etc.
  - Methods had better guarantees if you chose “simpler” classifiers
  - And in practice, gave you better results than neural networks
  - But were quadratic in training set size
- With training set of millions, quadratic-time optimization was not feasible.
- So people first moved to gradient descent, but with the same simple classifiers.
- Found that with additional computation power, if you train with small step size for many iterations (still better than quadratic), gradient descent gives you a reasonable answer.
- But then, since gradient descent was working, the question was why not try more complex classifiers?
- And Krizhevsky and others demonstrated: in this large training set / high training computation budget, deep neural networks are much better!

ARCHITECTURES

Broad Design Principles

- Think of a network that can "express" the operations that you think are needed to solve the problem
  - What kind of a "receptive" field should it have.
  - How non-linear does it need to be.
  - What should be the nature of the flow of information across the image.
- Make sure its a function you can actually learn.
  - Think of the flow of gradients.
  - Try to make other architectures that you know can be successfully trained as a starting point.
- Dealing with Overfitting: One approach:
  - First find the biggest deepest network that will overfit the data
    (Given enough capacity, CNNs will often be able to just memorize the dataset)
  - Then scale it down so that it generalizes.

Let’s consider image classification.

- We will fix our input image to be a specific size.
  - Typically choose square images of size $S \times S$
  - Given image, resize proportionally so that smaller side (height or width) is $S$
  - Then take an $S$ crop along the other direction
  - (Sometime take multiple crops and average)
- The final output will be a $C$ dimensional vector for $C$ classes.
  - Train using soft-max cross entropy.
  - Classify using arg-max
- Often, you’ll hear about Top-K error.
  - How often is the true class in the top K highest values of predicted $C$ vector.
Let’s talk about VGG-16. Winner of Imagenet 2014.

Reference
Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, “Return of the Devil in the Details: Delving Deep into Convolutional Nets”

Karen Simonyan & Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.”

Four kinds of layers:
- Convolutional
- Max-pooling
- Fully Connected
- Soft-max

**Convolutional Layers**
Take a spatial input, produce a spatial output.

\[ B \times H \times W \times C \Rightarrow B \times H' \times W' \times C' \]

Can also combine with down-sampling.

\[ g[b, y, x, c_2] = \sum_{k_y} \sum_{k_x} f[b, y s + k_y, x s + k_x, c_1] k[k_y, k_x, c_1, c_2] \]

Here, \( s \) is stride.

- PSET 5 asks you to implement 'valid convolution'
- But often combined with padding (just like in regular convolution)

**Max-Pooling Layer**
For each channel, choose the maximum value in a spatial neighborhood.

\[ B \times H \times W \times C \Rightarrow B \times H' \times W' \times C \]

\[ g[b, y, x, c] = \max_{k_y, k_x} f[b, y s + k_y, x s + k_x, c_1] \]

For each channel, choose the maximum value in a spatial neighborhood.

- What will the gradients of this look like ?
- Motivated by intuition from traditional object recognition (deformable part models). Allows for some 'slack' in exact spatial location.

**Question**: Input activation is \( B \times H \times W \times C \), and I convolve it with a kernel of size \( K \times K \times C_1 \times C_2 \); what is the size of my output? Assume ‘valid’ convolution.

\[ B \times (H - K + 1) \times (W - K + 1) \times C_2 \]

**Question**: What if I do this with a stride of 2 ?

Downsample above by 2. Think of what happens when sizes are even or odd.

\[ B \times [(H - K/2) + 1] \times [(W - K/2) + 1] \times C_2 \]

In general, you want to pad such that \( H - K \) and \( W - K \) are even, so that you keep the right and bottom edge of your images.
ARCHITECTURES

VGG-16

Input is a 224x224x3 Image

- Block 1
  - 3x3 Conv (Pad 1): 3->64 + RELU ("pad 1 means on all sides, all conv layers have a "bias")

VGG-16

Input is a 224x224x3 Image

- Block 1
  - 3x3 Conv (Pad 1): 3->64 + RELU
  - 3x3 Conv (Pad 1): 64->64 + RELU
  - 2x2 Max-Pool (Pad 0, Stride 2): 64->64
  
  Input to Block 2 is 112x112x64 (called pool1)

- Block 2
  - 3x3 Conv (Pad 1): 64->128 + RELU
  - 3x3 Conv (Pad 1): 128->128 + RELU

  Input to Block 3 is 56x56x128 (called pool2)

- Block 3
  - 3x3 Conv (Pad 1): 128->256 + RELU
  - 3x3 Conv (Pad 1): 256->256 + RELU

  Input to Block 4 is 28x28x256 (called pool3)

- Block 4
  - 3x3 Conv (Pad 1): 256->512 + RELU
  - 3x3 Conv (Pad 1): 512->512 + RELU

  Input to Block 5 is 14x14x512 (called pool4)

- Block 5
  - 3x3 Conv (Pad 1): 512->512 + RELU
  - 3x3 Conv (Pad 1): 512->512 + RELU

  Output of Block 5 is 7x7x512 (called pool5)

  - Reshape to a (49*512=25088) dimensional vector (or $B\times 25088$
  - Fully connected (matmul + bias) 25088 -> 4096 + RELU
  - Fully connected (matmul + bias) 4096 -> 4096 + RELU
  - Fully connected (matmul + bias) 4096 -> 1000

This is the final output that is trained with a softmax + cross entropy.

- Lots of layers: 138 Million Parameters
- Compared to previous architectures, used really small conv filters.
  - This has now become standard.
  - Two 3x3 layers is "better" than a single 5x5 layer.
    - More non-linear
    - Fewer independent weights
- Train this with backprop!
- Back in the day, would take a week or more.