This Lecture: Advanced Machine Learning

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Gaussian Processes
- Learning with Latent Variables
  - Prob. Distributions & Approx. Inference
  - Mixture Models
  - EM and Generalizations
- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & Optimization
  - CNNs, RNNs, RBMs, etc.

Topics of This Lecture

- Tricks of the Trade
  - Recap
  - Initialization
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

Recap: Data Augmentation

- Effect
  - Much larger training set
  - Robustness against expected variations
- During testing
  - When cropping was used during training, need to again apply crops to get same image size.
  - Beneficial to also apply flipping during test.
  - Applying several ColorPCA variations can bring another ~1% improvement, but at a significantly increased runtime.

Recap: Normalizing the Inputs

- Convergence is fastest if
  - The mean of each input variable over the training set is zero.
  - The inputs are scaled such that all have the same covariance.
  - Input variables are uncorrelated if possible.

- Advisable normalization steps (for MLPs)
  - Normalize all inputs that an input unit sees to zero-mean, unit covariance.
  - If possible, try to decorrelate them using PCA (also known as Karhunen-Loeve expansion).

Recap: Choosing the Right Learning Rate

- Convergence of Gradient Descent
  - Simple 1D example
    \[ W^{(t+1)} = W^{(t)} - \eta \frac{dE(W)}{dW} \]
  - What is the optimal learning rate \( \eta_{opt} \)?
  - If \( E \) is quadratic, the optimal learning rate is given by the inverse of the Hessian
    \[ \eta_{opt} = \left( \frac{d^2E(W^{(t)})}{dW^2} \right)^{-1} \]
  - Advanced optimization techniques try to approximate the Hessian by a simplified form.
  - If we exceed the optimal learning rate, bad things happen!
Recap: Advanced Optimization Techniques

- **Momentum**
  - Instead of using the gradient to change the position of the weight “particle”, use it to change the velocity.
  - Effect: dampen oscillations in directions of high curvature
  - Nesterov-Momentum: Small variation in the implementation
- **RMS-Prop**
  - Separate learning rate for each weight: Divide the gradient by a running average of its recent magnitude.
- **AdaGrad**
- **AdaDelta**
- **Adam**

Some more recent techniques, work better for some problems. Try them.

Trick: Patience

- Saddle points dominate in high-dimensional spaces!
- Learning often doesn’t get stuck, you just may have to wait...

Reducing the Learning Rate

- Final improvement step after convergence is reached
  - Reduce learning rate by a factor of 10.
  - Continue training for a few epochs.
  - Do this 1-3 times, then stop training.
- Effect
  - Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches.
- **Be careful: Do not turn down the learning rate too soon!**
  - Further progress will be much slower after that.

Topics of This Lecture

- **Tricks of the Trade**
  - Recap
  - Initialization
  - Batch Normalization
  - Dropout
- **Convolutional Neural Networks**
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- **CNN Architectures**
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

Batch Normalization [Ioffe & Szegedy ’14]

- **Motivation**
  - Optimization works best if all inputs of a layer are normalized.
- **Idea**
  - Introduce intermediate layer that centers the activations of the previous layer per minibatch.
  - i.e., perform transformations on all activations and undo those transformations when backpropagating gradients
- **Effect**
  - Much improved convergence

Dropout [Srivastava, Hinton ’12]

- **Idea**
  - Randomly switch off units during training.
  - Change network architecture for each data point, effectively training many different variants of the network.
  - When applying the trained network, multiply activations with the probability that the unit was set to zero.
  - Greatly improved performance
Topics of This Lecture

- Tricks of the Trade
  - Recap
  - Initialization
  - Batch Normalization
  - Dropout
- Convolutional Neural Networks
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers
- CNN Architectures
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

Neural Networks for Computer Vision

- How should we approach vision problems?
  
  Face Y/N?

  - Architectural considerations
    - Input is 2D ⇒ 2D layers of units
    - No pre-segmentation ⇒ Need robustness to misalignments
    - Vision is hierarchical ⇒ Hierarchical multi-layered structure
    - Vision is difficult ⇒ Network should be deep

Why Hierarchical Multi-Layered Models?

- Motivation 1: Visual scenes are hierarchically organized

  ![Object](image1)
  ![Object parts](image2)
  ![Primitive features](image3)
  ![Input image](image4)
  ![Face](image5)
  ![Eyes, nose, ...](image6)
  ![Oriented edges](image7)

- Motivation 2: Biological vision is hierarchical, too

  ![Object](image8)
  ![Object parts](image9)
  ![Primitive features](image10)
  ![Input image](image11)
  ![Face](image12)
  ![Eyes, nose, ...](image13)
  ![Oriented edges](image14)

Inspiration: Neuron Cells

  - Visual cortex consists of a hierarchy of simple, complex, and hyper-complex cells

Hubel/Wiesel Architecture

- Hubel & Wiesel topographical mapping
  - hyper-complex cells
  - complex cells
  - simple cells
- featural hierarchy
  - high level
  - mid level
  - low level

Slide credits: Svetlana Lazebnik, Rob Fergus
**Why Hierarchical Multi-Layered Models?**

- **Motivation 3:** Shallow architectures are inefficient at representing complex functions

![Diagram of an MLP with 1 hidden layer that can implement any function (universal approximator)]

However, if the function is deep, a very large hidden layer may be required.

---

**What’s Wrong With Standard Neural Networks?**

- **Complexity analysis**
  - How many parameters does this network have?
    \[ |\theta| = 3D^2 + D \]
  - For a small 32 x 32 image
    \[ |\theta| = 3 \cdot 32^4 + 32^2 \approx 3 \cdot 10^6 \]

- **Consequences**
  - Hard to train
  - Need to initialize carefully

  *Convolutional nets reduce the number of parameters!*

---

**Convolutional Neural Networks (CNN, ConvNet)**

- Neural network with specialized connectivity structure
  - Stack multiple stages of feature extractors
  - Higher stages compute more global, more invariant features
  - Classification layer at the end

---

**Convolutional Networks: Intuition**

- **Locally connected net**
  - E.g. 1000 x 1000 image
  - 1M hidden units
  - 10 x 10 receptive fields
  \[ \Rightarrow 100M \text{ parameters!} \]

- **Ideas to improve this**
  - Spatial correlation is local
  - Want translation invariance

---

**Convolutional Networks: Intuition**

- **Convolutional net**
  - Share the same parameters across different locations
  - Convolutions with learned kernels
Convolutional Networks: Intuition

- **Convolutional net**
  - Share the same parameters across different locations
  - Convolutions with learned kernels

- **Learn multiple filters**
  - E.g. 1000x1000 image, 100 filters
  - 10x10 filter size
  - \( \Rightarrow 10k \) parameters

- **Result: Response map**
  - size: 1000x1000x100
  - Only memory, not params!

---

Convolution Layers

- **Example image:** 32x32x3 volume

**Before:** Full connectivity
- 32x32x3 weights

**Now:** Local connectivity
- One neuron connects to, e.g., 5x5x5 region.
- \( \Rightarrow \) Only 5x5x3 shared weights.

---

Convolution Layers

- **Naming convention:**
  - \( 7 \times 7 \) input assume \( 3 \times 3 \) connectivity
  - **stride 1**

- **Replicate this column of hidden neurons across space, with some stride.**

---

Important Conceptual Shift

- **Before**
  - input layer
  - hidden layer

- **Now:**
  - \( \rightarrow \) output layer
Convolution Layers

- Replicate this column of hidden neurons across space, with some **stride**.

Example:
7 x 7 input
assume 3 x 3 connectivity
stride 1

**Example:**
7 x 7 input
assume 3 x 3 connectivity
stride 1

**Example:**
7 x 7 input
assume 3 x 3 connectivity
stride 1
⇒ 5 x 5 output

What about stride 2?

**What about stride 2?**

**Slide credit:** FeiFei Li, Andrej Karpathy B. Leibe
Convolution Layers

- Replicate this column of hidden neurons across space, with some stride.

**Example:**
7x7 input, assume 3x3 connectivity, stride 1
\Rightarrow 5x5 output

What about stride 2?
\Rightarrow 3x3 output

- In practice, common to zero-pad the border.
  - Preserves the size of the input spatially.

**Activation Maps of Convolutional Filters**

- Each activation map is a depth slice through the output volume.

**Effect of Multiple Convolution Layers**

- Let’s assume the filter is an eye detector
  - How can we make the detection robust to the exact location of the eye?

**Solution:**
- By pooling (e.g., max or avg) filter responses at different spatial locations, we gain robustness to the exact spatial location of features.
**Max Pooling**

- **Effect:**
  - Make the representation smaller without losing too much information
  - Achieve robustness to translations

- **Note**
  - Pooling happens independently across each slice, preserving the number of slices.

---

**CNNs: Implication for Back-Propagation**

- **Convolutional layers**
  - Filter weights are shared between locations
  - Gradients are added for each filter location.

---

**Topics of This Lecture**

- **Tricks of the Trade**
  - Recap
  - Initialization
  - Batch Normalization
  - Dropout

- **Convolutional Neural Networks**
  - Neural Networks for Computer Vision
  - Convolutional Layers
  - Pooling Layers

- **CNN Architectures**
  - LeNet
  - AlexNet
  - VGGNet
  - GoogLeNet

---

**CNN Architectures: LeNet (1998)**

- Early convolutional architecture
  - 2 Convolutional layers, 2 pooling layers
  - Fully-connected NN layers for classification
  - Successfully used for handwritten digit recognition (MNIST)


---

**ImageNet Challenge 2012**

- **ImageNet**
  - ~14M labeled internet images
  - 20k classes
  - Human labels via Amazon Mechanical Turk

- **Challenge (ILSVRC)**
  - 1.2 million training images
  - 1000 classes
  - Goal: Predict ground-truth class within top-5 responses
  - Currently one of the top benchmarks in Computer Vision

[Deng et al., CVPR'09]
CNN Architectures: AlexNet (2012)

- Similar framework as LeNet, but
  - Bigger model (7 hidden layers, 650k units, 60M parameters)
  - More data ($10^6$ images instead of $10^3$)
  - GPU implementation
  - Better regularization and up-to-date tricks for training (Dropout)

AlexNet Results

ILSVRC 2012 Results

- AlexNet almost halved the error rate
  - 16.4% error (top-5) vs. 26.2% for the next best approach
  - A revolution in Computer Vision
  - Acquired by Google in Jan ‘13, deployed in Google+ in May ‘13

CNN Architectures: VGGNet (2015)

- Main ideas
  - Deeper network
  - Stacked convolutional layers with smaller filters (+ nonlinearity)
  - Detailed evaluation of all components

VGGNet Results

Comparison to AlexNet

K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015
CNN Architectures: GoogLeNet (2014)

- Main ideas
  - "Inception" module as modular component
  - Learns filters at several scales within each module

References and Further Reading

- LeNet

- AlexNet

- VGGNet
  - K. Simonyan, A. Zisserman, Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

- GoogLeNet