Deep Convolutional Neural Nets

COMPSCI 527 — Computer Vision
Outline

1. Why Neural Networks?
2. Circuits
3. Neurons, Layers, and Networks
4. AlexNet
Why Neural Networks?

- Neural networks are very *expressive* (large $\mathcal{H}$)
- Can approximate any well-behaved function from a hypercube $X$ in $\mathbb{R}^d$ to an interval $Y$ in $\mathbb{R}$ within any $\epsilon > 0$
- *Universal approximators*
- However
  - Complexity grows exponentially with $d = \text{dim}(X)$
  - $L_T$ is not convex (not even close)
  - Large $\mathcal{H}$ $\Rightarrow$ overfitting $\Rightarrow$ *lots* of data!
- Amazon’s Mechanical Turk made neural networks possible
- Even so, we cannot keep up with the curse of dimensionality!
Why Do Neural Networks Work?

- Neural networks are data hungry
- Availability of lots of data is not a sufficient explanation
- There must be deeper reasons
- Special structure of image space (or audio space)?
- Specialized network architectures?
- Regularization tricks and techniques?
- We don’t really know. Stay tuned...
- Be prepared for some hand-waving and empirical statements
Circuits

- Describe implementation of $h : X \rightarrow Y$ on a computer
- Algorithm: A sequence of finite steps
- *Circuit*: Many *gates* of few types, wired together

These are NAND gates. We’ll use *neurons*
- Algorithms and circuits are equivalent
- Algorithm can simulate a circuit
- Computer is a circuit that runs algorithms!
- Computer really only computes Boolean functions...
Deep Neural Networks as Circuits

- Neural networks are typically described as circuits
- Nearly always implemented as algorithms
- One gate, the neuron
- Many neurons that receive the same input form a layer
- A cascade of layers is a network
- A deep network has many layers
- Layers with a special constraint are called convolutional
The Neuron

- $y = \rho(a(x))$ where $a = v^T x + b$
  - $x \in \mathbb{R}^d$, $y \in \mathbb{R}$
- $v$ are the gains, $b$ is the bias
- Together, $w = [v, b]^T$ are the weights
- $\rho(a) = \max(0, a)$ (ReLU, Rectified Linear Unit)
The Neuron as a Pattern Matcher (Almost)

- Left pattern is a drumbeat $\mathbf{v}$ (a pattern template):

- Which of the other two patterns $\mathbf{x}$ is a drumbeat?
- Normalize both $\mathbf{v}$ and $\mathbf{x}$ so that $\|\mathbf{v}\| = \|\mathbf{x}\| = 1$
- Then $\mathbf{v}^T \mathbf{x}$ is the cosine of the angle between the patterns
- If $\mathbf{v}^T \mathbf{x} \geq -b$ for some threshold $-b$, output $a = \mathbf{v}^T \mathbf{x} + b$
  (amount by which the cosine exceeds the threshold)
  otherwise, output 0
- $y = \rho(\mathbf{v}^T \mathbf{x} + b)$
The Neuron as a Pattern Matcher (Almost)

- \( y = \rho(v^T x + b) \)
- A neuron is a pattern matcher, except for normalization...
- ...and if followed by a (trivial) classifier
- In neural networks, normalization may happen in later or earlier layers, and classification happens at the end
- This interpretation is not necessary to understand neural networks
- Nice to have a mental model, though
- Many neurons wired together can approximate any function we want
- A neural network is a function approximator
Layers and Networks

- A *layer* is a set of neurons that share the same input

\[
y_1 = \rho(x_1)
y_{d(1)} = \rho(x_{d(1)})
\]

- A *neural network* is a cascade of layers: \( y = \rho(Vx + b) \)
- A neural network is *deep* if it has many layers
- *Two* layers can make a universal approximator
- If neurons did not have nonlinearities, any cascade of layers would collapse to a single layer
Convolutional Layers

- A layer with input $\mathbf{x} \in \mathbb{R}^d$ and output $\mathbf{y} \in \mathbb{R}^e$ has $e$ neurons, each with $d$ gains and one bias
- Total of $(d + 1)e$ weights to be trained in a single layer
- For images, $d, e$ are in the order of hundreds of thousands or even millions
- Too many parameters
- *Convolutional layers* are layers restricted in a special way
- Many fewer parameters to train
- Also good justification in terms of heuristic principles
Hierarchy, Locality, Reuse

• To find a person, look for a face, a torso, limbs,…
• To find a face, look for eyes, nose, ears, mouth, hair,…
• To find an eye look for a circle, some corners, some curved edges,…
• A hierarchical image model is less sensitive to viewpoint, body configuration, …
• Hierarchy leads to a cascade of layers
• Low-level features are local: A neuron doesn’t need to see the entire image
• Circles are circles, regardless of where they show up: A single neuron can be reused to look for circles anywhere in the image
Correlation, Locality, and Reuse

- Does the drumbeat on the left show up in the clip on the right?

- Drumbeat $v$ has 25 samples, clip $x$ has 100
- Make $100 - 25 + 1 = 76$ neurons that look for $v$ in every possible position
- $y_i = \rho(v_i^T x + b_i)$ where $v_i^T = [0, \ldots, 0, v_0, \ldots, v_{24}, 0, \ldots 0]$

- Gain matrix $V =$

\[
\begin{bmatrix}
 v_0 & \cdots & v_{24} & 0 & 0 & \cdots & 0 \\
 0 & v_0 & \cdots & v_{24} & 0 & \cdots & 0 \\
 \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
 \vdots & \cdots & \ddots & \ddots & \ddots & \ddots & \vdots \\
 0 & \cdots & \cdots & 0 & v_0 & \cdots & v_{24} \\
\end{bmatrix}
\]
Compact Computation

\[ a = Vx + b \quad \text{where} \quad V = \begin{bmatrix}
    v_0 & \cdots & v_{24} & 0 & 0 & \cdots & 0 \\
    0 & v_0 & \cdots & v_{24} & 0 & \cdots & 0 \\
    \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
    \vdots & \vdots & \vdots & 0 & v_0 & \cdots & v_{24} \\
    0 & \cdots & \cdots & 0 & v_0 & \cdots & v_{24} \\
\end{bmatrix} \]

- \( V \) is \( 76 \times 100 \), the \( i \)-th row is one neuron: \( a_i = v_i^T x + b_i \)
- Same as

\[ a_i = b_i + \sum_{\ell=0}^{24} v_\ell x_{i+\ell} \quad \text{for} \quad i = 0, \ldots, 75 \]

- (One-dimensional) correlation (or convolution with \( v[:,-1] \))
- View \( a \) as either a single convolution with kernel \( v \) or the output from a layer with 76 neurons, each with 75 zero weights and the same 25 nonzeros in different positions
A Small Example

\[ a_i = \sum_{\ell=0}^{2} v_\ell x_{i+\ell} \quad \text{for} \quad i = 0, \ldots, 5 \]

\[
a = Vx = \begin{bmatrix} v_0 & v_1 & v_2 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & v_0 & v_1 & v_2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & v_0 & v_1 & v_2 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & v_0 & v_1 & v_2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & v_0 & v_1 & v_2 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & v_0 & v_1 & v_2 & 0 \end{bmatrix} x
\]
2D Correlation or Convolution

\[ a_{ij} = \sum_{\ell=0}^{k_1-1} \sum_{m=0}^{k_2-1} v_{\ell m} x_{i+\ell,j+m} + b_{ij} \]

- Output is \(2 \times 5\)
- Can be made to be \(4 \times 6\) using padding ("same" correlation)
**Stride**

- Activation $a_{ij}$ is often similar to $a_{i,j+1}$ and $a_{i+1,j}$
- Images often vary slowly over space
- Reduce the redundancy in the output by computing convolutions with a *stride* $s_m$ greater than one
- Only compute every $s_m$ output values in dimension $m$
- Output size shrinks from $d_1 \times d_2$ to about $d_1/s_1 \times d_2/s_2$
- Typically $s_m = s$ (same stride in all dimensions)
- Layers get smaller and smaller because of stride
Max Pooling

- Another way to reduce output resolution is *max pooling*
- This is a layer of its own, separate from convolution
- Consider $k \times k$ windows with stride $s$
- Often $s = k$ (adjacent, non-overlapping windows)
- For each window, output the maximum value
- Output is about $d_1/s \times d_2/s$
- Returns highest response in window, rather than the response in a fixed position
- Loosely analogous to using cells and histograms in HOG
The Input Layer of AlexNet

- AlexNet circa 2012, classifies color images into one of 1000 categories
- Trained on ImageNet, a large database with millions of labeled images

\[ y = \pi(\rho(a)) \]
A more Compact Drawing

\[
y = \pi(\rho(a))
\]
AlexNet

Deep Convolutional Neural Nets
AlexNet Numbers

- Input is $224 \times 224 \times 3$ (color image)
- First layer has 96 feature maps of size $55 \times 55$
- A fully-connected first layer would have about $224 \times 224 \times 3 \times 55 \times 55 \times 96 \approx 4.4 \times 10^{10}$ weights
- With convolutional kernels of size $11 \times 11$, there are only $96 \times 11^2 = 11,616$ weights
- That’s a big deal! Locality and reuse
- Most of the complexity is in the last few, fully-connected layers, which still have millions of parameters
- More recent neural networks have much lighter final layers, but many more layers
- There are also fully convolutional neural networks
Output

- For *regression*, the output of the network is the desired quantity.
- The last layer of a neural net used for *classification* is a *soft-max* layer.
  $$p = \sigma(y) = \frac{e^y}{1^Ty}$$
- As many entries in $y$ and $p$ as there are classes.
- One output score $p$ per class for classification.
- Classify by $\arg\max p$.
The Soft-Max Function

\[ p_k(y) = \frac{e^{y_k}}{\sum_{j=1}^{K} e^{y_j}} \quad \text{or} \quad p(y) = \frac{e^y}{1^T e^y} \]

- \( y \in \mathbb{R}^K \rightarrow p \in \mathbb{R}^K \)
- \( p_k(y) > 0 \) and \( \sum_{k=1}^{K} p_k(y) = 1 \) for all \( y \)
- If \( y_i \gg y_j \) for \( j \neq i \) then \( \sum_{j=1}^{K} e^{y_j} \approx e^{y_i} \)
- Therefore, \( p_i \approx 1 \) and \( p_j \approx 0 \) for \( j \neq i \)
- “Brings out the biggest:” soft-max

\[ \lim_{\alpha \to \infty} y^T p(\alpha y) = \max(y) \]