Contains slides from…

- LeCun & Ranzato
- Russ Salakhutdinov
- Honglak Lee
- Andrew Ng
- Google images
  - https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
Single Node

“bias unit”

\[ x_0 = 1 \]

\[ x_1 \]
\[ \theta_1 \]
\[ \theta_0 \]
\[ x_2 \]
\[ \theta_2 \]
\[ x_3 \]
\[ \theta_3 \]

\[ h_\theta(x) = g(\theta^T x) \]
\[ = \frac{1}{1 + e^{-\theta^T x}} \]

Sigmoid (logistic) activation function:

\[ g(z) = \frac{1}{1 + e^{-z}} \]
Neural Network

Layer 1
(Input Layer)

Layer 2
(Hidden Layer)

Layer 3
(Output Layer)
Multi-layer Neural Network

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]
Multiple Output Units: One-vs-Rest

We want:

\[ h_\Theta(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \]
when pedestrian

\[ h_\Theta(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \]
when car

\[ h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \]
when motorcycle

\[ h_\Theta(x) \approx \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \]
when truck

\[ h_\Theta(x) \in \mathbb{R}^K \]
Multiple Output Units: One-vs-Rest

Given \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}

Must convert labels to 1-of-\(K\) representation

\[ y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \text{ when motorcycle, } y_i = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ when car, etc.}\]
The neural network architecture is defined by the number of layers, and the number of nodes in each layer, but also by allowable edges.

We say a layer is **Fully Connected (FC)** if all linear mappings from the current layer to the next layer are permissible.

\[
a^{(k+1)} = g(\Theta a^{(k)}) \quad \text{for any } \Theta \in \mathbb{R}^{n_{k+1} \times n_k}
\]

A lot of parameters!!  
\[n_1 n_2 + n_2 n_3 + \cdots + n_L n_{L+1}\]
Neural Network Architecture

Objects are often localized in space so to find the faces in an image, not every pixel is important for classification—makes sense to drag a window across an image.

Similarly, to identify edges or other local structure, it makes sense to only look at local information.
Neural Network Architecture

\[ a^{(k+1)}_i = g \left( \sum_{j=1}^{n} \Theta_{i,j} a^{(k)}_j \right) \]

Parameters:
\[
\begin{bmatrix}
\Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} & \Theta_{1,5} \\
\Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} & \Theta_{2,5} \\
\Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} & \Theta_{3,5} \\
\Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} & \Theta_{4,5} \\
\Theta_{5,1} & \Theta_{5,2} & \Theta_{5,3} & \Theta_{5,4} & \Theta_{5,5}
\end{bmatrix}
\]

\[ n^2 \] vs.

\[
\begin{bmatrix}
\Theta_{1,1} & \Theta_{1,2} & 0 & 0 & 0 \\
\Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & 0 & 0 \\
0 & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} & 0 \\
0 & 0 & \Theta_{4,3} & \Theta_{4,4} & \Theta_{4,5} \\
0 & 0 & 0 & \Theta_{5,4} & \Theta_{5,5}
\end{bmatrix}
\]

\[ 3n - 2 \]
Neural Network Architecture

\[ \Theta_{1,1} \quad \Theta_{1,2} \quad \Theta_{1,3} \quad \Theta_{1,4} \quad \Theta_{1,5} \]
\[ \Theta_{2,1} \quad \Theta_{2,2} \quad \Theta_{2,3} \quad \Theta_{2,4} \quad \Theta_{2,5} \]
\[ \Theta_{3,1} \quad \Theta_{3,2} \quad \Theta_{3,3} \quad \Theta_{3,4} \quad \Theta_{3,5} \]
\[ \Theta_{4,1} \quad \Theta_{4,2} \quad \Theta_{4,3} \quad \Theta_{4,4} \quad \Theta_{4,5} \]
\[ \Theta_{5,1} \quad \Theta_{5,2} \quad \Theta_{5,3} \quad \Theta_{5,4} \quad \Theta_{5,5} \]

Parameters:

\[ n^2 \quad 3n - 2 \quad 3 \]

\[ a_i^{(k+1)} = g \left( \sum_{j=1}^{n} \Theta_{i,j} a_j^{(k)} \right) \]

Mirror/share local weights everywhere (e.g., structure equally likely to be anywhere in image)

\[ a_i^{(k+1)} = g \left( \sum_{j=1}^{k} \theta_{j} a_{i+j}^{(k)} \right) \]
Neural Network Architecture

Fully Connected (FC) Layer

\[
\begin{bmatrix}
\Theta_{1,1} & \Theta_{1,2} & \Theta_{1,3} & \Theta_{1,4} & \Theta_{1,5} \\
\Theta_{2,1} & \Theta_{2,2} & \Theta_{2,3} & \Theta_{2,4} & \Theta_{2,5} \\
\Theta_{3,1} & \Theta_{3,2} & \Theta_{3,3} & \Theta_{3,4} & \Theta_{3,5} \\
\Theta_{4,1} & \Theta_{4,2} & \Theta_{4,3} & \Theta_{4,4} & \Theta_{4,5} \\
\Theta_{5,1} & \Theta_{5,2} & \Theta_{5,3} & \Theta_{5,4} & \Theta_{5,5}
\end{bmatrix}
\]

\[
a^{(k+1)}_i = g \left( \sum_{j=1}^n \Theta_{i,j} a^{(k)}_j \right)
\]

Convolutional (CONV) Layer (1 filter)

\[
\begin{bmatrix}
\theta_2 & \theta_3 & 0 & 0 & 0 \\
\theta_1 & \theta_2 & \theta_3 & 0 & 0 \\
0 & \theta_1 & \theta_2 & \theta_3 & 0 \\
0 & 0 & \theta_1 & \theta_2 & \theta_3 \\
0 & 0 & 0 & \theta_1 & \theta_2
\end{bmatrix}
\]

\[m=3\]

\[
a^{(k+1)}_i = g \left( \sum_{j=1}^m \theta_j a^{(k)}_{i+j} \right) = g([\theta \ast a^{(k)}]_i)
\]

\[\theta = (\theta_1, \ldots, \theta_m) \in \mathbb{R}^m \text{ is referred to as a “filter”}\]

* Actually defined as the closely related quantity of “cross-correlation” but the deep learning literature just calls this “convolution”
Example (1d convolution)

\[(\theta * x)_i = \sum_{j=1}^{m} \theta_j x_{i+j-1}\]

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

Filter \(\theta \in \mathbb{R}^m\)

Output \(\theta * x\)
Example (1d convolution)

\[(\theta \ast x)_i = \sum_{j=1}^{m} \theta_j x_{i+j-1}\]

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

Filter \(\theta \in \mathbb{R}^m\)

Output \(\theta \ast x\)
Example (1d convolution)

\[(\theta \ast x)_i = \sum_{j=1}^{m} \theta_j x_{i+j-1}\]

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

Filter \(\theta \in \mathbb{R}^m\)

Output \(\theta \ast x\)
Example (1d convolution)

\[(\theta \ast x)_i = \sum_{j=1}^{m} \theta_j x_{i+j-1}\]
2d Convolution Layer

Example: 200x200 image
- Fully-connected, 400,000 hidden units = 16 billion parameters
- Locally-connected, 400,000 hidden units 10x10 fields = 40 million params
- Local connections capture local dependencies
Convolution of images (2d convolution)

$$(I * K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$

**Image**

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Convolved Feature** $I * K$

<table>
<thead>
<tr>
<th>4</th>
</tr>
</thead>
</table>

**Image**

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Filter** $K$

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Convolution of images

\[(I * K)(i, j) = \sum_{m} \sum_{n} I(i + m, j + n)K(m, n).\]

**Image I**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Filter</th>
<th>Convolved Image</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Edge detection</strong></td>
<td></td>
<td><img src="image1" alt="Edge detection" /></td>
</tr>
</tbody>
</table>
|                 | \[
|                 | \[
|                 | \[
|                 | \[
| **Sharpen**     | \[
|                 | \[
| **Box blur**    | \[
|                 | \[
| **Gaussian blur** |          | ![Gaussian blur](image2) |
|                 | \[
|                 | \[
|                 | \[
|                 | \[
|
Convolution of images

Input image $X$

filters $H_k$

convolved image $H_k \ast X$

flatten into vector

$\vec(H_1 \ast X)$
$\vec(H_2 \ast X)$
$\vdots$
Stacking convolved images

64 filters
Stacking convolved images

Apply Non-linearity to the output of each layer, Here: ReLu (rectified linear unit)

Other choices: sigmoid, arctan
Pooling

Pooling reduces the dimension and can be interpreted as “This filter had a high response in this general region”.

Single depth slice

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

6 8
3 4

27x27x64

pool

14x14x64
Pooling Convolution layer

Convolve with 64 6x6x3 filters

MaxPool with 2x2 filters and stride 2
Simplest feature pipeline

Convolve with 64 6x6x3 filters

MaxPool with 2x2 filters and stride 2

Flatten into a single vector of size $14 \times 14 \times 64 = 12544$

How do we choose all the hyperparameters?

How do we choose the filters?
- Hand crafted (digital signal processing, c.f. wavelets)
- Learn them (deep learning)
Some hand-created image features

- SIFT
- Spin Image
- HoG
- RIFT
- Texton
- GLOH

Slide from Honglak Lee
Learning Features with Convolutional Networks

Recall: Convolutional neural networks (CNN) are just regular fully connected (FC) neural networks with some connections removed. 
Train with back-propagation!
Training Convolutional Networks

Real example network: LeNet
Real example network: LeNet
Remarks

• Convolution is a fundamental operation in signal processing. Instead of hand-engineering the filters (e.g., Fourier, Wavelets, etc.) Deep Learning learns the filters and CONV layers with back-propagation, replacing fully connected (FC) layers with convolutional (CONV) layers.

• Pooling is a dimensionality reduction operation that summarizes the output of convolving the input with a filter.

• Typically the last few layers are Fully Connected (FC), with the interpretation that the CONV layers are feature extractors, preparing input for the final FC layers. Can replace last layers and retrain on different dataset+task.

• Just as hard to train as regular neural networks.

• More exotic network architectures for specific tasks.
Variable length sequences

Images are usually standardized to be the same size (e.g., 256x256x3)

Neural Network

- input layer
- hidden layer 1
- hidden layer 2
- output layer
Variable length sequences

Images are usually standardized to be the same size (e.g., 256x256x3)

But what if we wanted to do classification on country-of-origin for names?

Recurrent Neural Network

Scottish
English
Irish
Variable length sequences

Recurrent Neural Network

Standard RNN

LSTM

©Kevin Jamieson

Slide: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Backprop
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ : \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ : \]
\[ \hat{y} = a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]

\[ g(z) = \frac{1}{1 + e^{-z}} \]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[
\begin{align*}
\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \\
L(y, \hat{y}) &= y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \\
g(z) &= \frac{1}{1 + e^{-z}} \\
\delta_i^{(l+1)} &= \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}}
\end{align*}
\]
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g(z^{(2)}) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g(z^{(l+1)}) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l)}} = \sum_k \frac{\partial L(y, \hat{y})}{\partial z_k^{(l+1)}} \cdot \frac{\partial z_k^{(l+1)}}{\partial z_i^{(l)}} \]
\[ = \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i} g'(z_i^{(l)}) \]
\[ = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]
\[ g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)})$$
$$\vdots$$
$$z^{(l+1)} = \Theta^{(l)} a^{(l)}$$
$$a^{(l+1)} = g(z^{(l+1)})$$
$$\vdots$$
$$\hat{y} = a^{(L+1)}$$

$$L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})$$
$$g(z) = \frac{1}{1 + e^{-z}}$$
$$\delta_{i}^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_{i}^{(l+1)}}$$

$$\frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_{i}^{(l+1)}} \cdot \frac{\partial z_{i}^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_{i}^{(l+1)} \cdot a_{j}^{(l)}$$

$$\delta_{i}^{(l)} = a_{i}^{(l)} (1 - a_{i}^{(l)}) \sum_{k} \delta_{k}^{(l+1)} \cdot \Theta_{k,i}$$
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(L+1)} = \Theta^{(L)} a^{(L)} \]
\[ a^{(L+1)} = g \left( z^{(L+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta^{(l)}_{i,j}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta^{(l)}_{i,j}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = a_i^{(l)}(1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i} \]

\[ \delta_i^{(L+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(L+1)}} = \frac{\partial}{\partial z_i^{(L+1)}} \left[ y \log(g(z^{(L+1)})) + (1 - y) \log(1 - g(z^{(L+1)})) \right] \]
\[ = \frac{y}{g(z^{(L+1)})} g'(z^{(L+1)}) - \frac{1 - y}{1 - g(z^{(L+1)})} g'(z^{(L+1)}) \]
\[ = y - g(z^{(L+1)}) = y - a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]
\[ g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]

©Kevin Jamieson
Backprop

\[ a^{(1)} = x \]
\[ z^{(2)} = \Theta^{(1)} a^{(1)} \]
\[ a^{(2)} = g \left( z^{(2)} \right) \]
\[ \vdots \]
\[ z^{(l+1)} = \Theta^{(l)} a^{(l)} \]
\[ a^{(l+1)} = g \left( z^{(l+1)} \right) \]
\[ \vdots \]
\[ \hat{y} = a^{(L+1)} \]

Recursive Algorithm!

\[ \frac{\partial L(y, \hat{y})}{\partial \Theta_{i,j}^{(l)}} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \cdot \frac{\partial z_i^{(l+1)}}{\partial \Theta_{i,j}^{(l)}} =: \delta_i^{(l+1)} \cdot a_j^{(l)} \]

\[ \delta_i^{(l)} = a_i^{(l)} (1 - a_i^{(l)}) \sum_k \delta_k^{(l+1)} \cdot \Theta_{k,i} \]

\[ \delta^{(L+1)} = y - a^{(L+1)} \]

\[ L(y, \hat{y}) = y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}) \]
\[ g(z) = \frac{1}{1 + e^{-z}} \quad \delta_i^{(l+1)} = \frac{\partial L(y, \hat{y})}{\partial z_i^{(l+1)}} \]
Backpropagation

Set \( \Delta_{ij}^{(l)} = 0 \) \( \forall l, i, j \)

For each training instance \((x_i, y_i)\):

Set \( a^{(1)} = x_i \)

Compute \( \{a^{(2)}, \ldots, a^{(L)}\} \) via forward propagation

Compute \( \delta^{(L)} = a^{(L)} - y_i \)

Compute errors \( \{\delta^{(L-1)}, \ldots, \delta^{(2)}\} \)

Compute gradients \( \Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)} \)

Compute avg regularized gradient \( D_{ij}^{(l)} = \begin{cases} \frac{1}{n} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{n} \Delta_{ij}^{(l)} & \text{otherwise} \end{cases} \)

\( D^{(l)} \) is the matrix of partial derivatives of \( J(\Theta) \)