Multilayer perceptrons

- Key idea: build complex functions by composing simple functions

\[ f(x) = Wx \]
\[ g(x) = \max(x,0) \]
\[ f(x) = Wx \]
\[ g(x) = \max(x,0) \]
\[ f(x) = Wx \]
Multilayer perceptrons

• Key idea: build complex functions by composing simple functions
• Caveat: simple functions must include non-linearities
• $W(U(Vx)) = (WUV)x$
Reducing capacity
Reducing capacity

65K W 65K
Idea 1: local connectivity

• Inputs and outputs are *feature maps*
• Pixels only related to nearby pixels
Idea 2: Translation invariance

• Pixels only related to nearby pixels
Local connectivity + translation invariance = convolution

<table>
<thead>
<tr>
<th>5.4</th>
<th>0.1</th>
<th>3.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Local connectivity + translation invariance = \textit{convolution}

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5.4</td>
<td>0.1</td>
<td>3.6</td>
</tr>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Local connectivity + translation invariance = convolution

<table>
<thead>
<tr>
<th>5.4</th>
<th>0.1</th>
<th>3.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8</td>
<td>2.3</td>
<td>4.5</td>
</tr>
<tr>
<td>1.1</td>
<td>3.4</td>
<td>7.2</td>
</tr>
</tbody>
</table>
Convolution as a primitive
Invariance to distortions
Invariance to distortions
Invariance to distortions: Pooling
Invariance to distortions: Subsampling
Convolution subsampling convolution
Convolution subsampling convolution

• Convolution in earlier steps detects *more local* patterns *less resilient* to distortion
• Convolution in later steps detects *more global* patterns *more resilient* to distortion
• Subsampling allows capture of *larger, more invariant* patterns
Convolutional networks
Convolutional networks

Convolutional networks

Vagaries of optimization

• Non-convex
  • Local optima
  • Sensitivity to initialization

• Vanishing / exploding gradients
  \[
  \frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \cdots \frac{\partial z_{i+1}}{\partial z_i}
  \]
  • If each term is (much) greater than 1 \(\rightarrow\) explosion of gradients
  • If each term is (much) less than 1 \(\rightarrow\) vanishing gradients
Vanishing and exploding gradients

\[
\frac{\partial z}{\partial z_i} = \frac{\partial z}{\partial z_{n-1}} \frac{\partial z_{n-1}}{\partial z_{n-2}} \ldots \frac{\partial z_{i+1}}{\partial z_i}
\]

\[
\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial z_i}
\]

\[
\lambda_{\min}(\frac{\partial z}{\partial z_i}) \frac{\partial L}{\partial z} \leq \frac{\partial L}{\partial z} \frac{\partial z}{\partial z_i} \leq \lambda_{\max}(\frac{\partial z}{\partial z_i}) \frac{\partial L}{\partial z}
\]

\[
\lambda_{\max}(UV) \leq \lambda_{\max}(U) \lambda_{\max}(V)
\]

\[
\lambda_{\min}(UV) \geq \lambda_{\min}(U) \lambda_{\min}(V)
\]

\[
\lambda_{\max}(A^n) = \lambda_{\max}(A)^n
\]

\[
\lambda_{\min}(A^n) = \lambda_{\min}(A)^n
\]
Sigmoids cause vanishing gradients

Gradient close to 0
Convolution subsampling convolution
Rectified Linear Unit (ReLU)

• $\max(x, 0)$
• Also called half-wave rectification (signal processing)
Image Classification
ImageNet

- 1000 categories
- ~1000 instances per category

Challenge winner's accuracy

Convolutional Networks
Transfer learning
Transfer learning with convolutional networks

Trained feature extractor $\phi$
### Transfer learning with convolutional networks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Non-Convnet Method</th>
<th>Non-Convnet perf</th>
<th>Pretrained convnet + classifier</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech 101</td>
<td>MKL</td>
<td>84.3</td>
<td>87.7</td>
<td>+3.4</td>
</tr>
<tr>
<td>VOC 2007</td>
<td>SIFT+FK</td>
<td>61.7</td>
<td>79.7</td>
<td>+18</td>
</tr>
<tr>
<td>CUB 200</td>
<td>SIFT+FK</td>
<td>18.8</td>
<td>61.0</td>
<td>+42.2</td>
</tr>
<tr>
<td>Aircraft</td>
<td>SIFT+FK</td>
<td>61.0</td>
<td>45.0</td>
<td>-16</td>
</tr>
<tr>
<td>Cars</td>
<td>SIFT+FK</td>
<td>59.2</td>
<td>36.5</td>
<td>-22.7</td>
</tr>
</tbody>
</table>
Why transfer learning?

• Availability of training data

• Computational cost

• Ability to pre-compute feature vectors and use for multiple tasks

• Con: NO end-to-end learning
Finetuning
Finetuning

Initialize with pre-trained, then train with low learning rate
## Finetuning

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Non-Convnet Method</th>
<th>Non-Convnet perf</th>
<th>Pretrained convnet + classifier</th>
<th>Finetuned convnet</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caltech 101</td>
<td>MKL</td>
<td>84.3</td>
<td>87.7</td>
<td>88.4</td>
<td>+4.1</td>
</tr>
<tr>
<td>VOC 2007</td>
<td>SIFT+FK</td>
<td>61.7</td>
<td>79.7</td>
<td>82.4</td>
<td>+20.7</td>
</tr>
<tr>
<td>CUB 200</td>
<td>SIFT+FK</td>
<td>18.8</td>
<td>61.0</td>
<td>70.4</td>
<td>+51.6</td>
</tr>
<tr>
<td>Aircraft</td>
<td>SIFT+FK</td>
<td>61.0</td>
<td>45.0</td>
<td>74.1</td>
<td>+13.1</td>
</tr>
<tr>
<td>Cars</td>
<td>SIFT+FK</td>
<td>59.2</td>
<td>36.5</td>
<td>79.8</td>
<td>+20.6</td>
</tr>
</tbody>
</table>