Today’s class

• Part 1: What is deep learning?

• Part 2: Supervised Deep Learning
  • Neural networks
  • Convolutional Neural Networks (CNNs)

• Part 3: Unsupervised Deep Learning
  • Overview of some approaches
Slide credit

• Many slides are taken/adapted from Fei-Fei Li and Andrew Ng
Part 1:
What is deep learning?
Typical goal of machine learning

**input**
- images/video
- audio
- text

**output**
- Label: “Motorcycle”
- Suggest tags
- Image search
- Speech recognition
- Music classification
- Speaker identification
- Web search
- Anti-spam
- Machine translation
Typical goal of machine learning

Feature engineering: most time consuming!

input

images/video

audio

text

output

Label: “Motorcycle”
Suggest tags
Image search
...

Speech recognition
Music classification
Speaker identification
...

Web search
Anti-spam
Machine translation
...
Our goal in object classification

ML → “motorcycle”
Why is this hard?

You see this:

But the camera sees this:

<table>
<thead>
<tr>
<th>194</th>
<th>210</th>
<th>201</th>
<th>212</th>
<th>199</th>
<th>213</th>
<th>215</th>
<th>195</th>
<th>178</th>
<th>158</th>
<th>182</th>
<th>209</th>
</tr>
</thead>
<tbody>
<tr>
<td>180</td>
<td>189</td>
<td>190</td>
<td>221</td>
<td>209</td>
<td>205</td>
<td>191</td>
<td>167</td>
<td>147</td>
<td>115</td>
<td>129</td>
<td>163</td>
</tr>
<tr>
<td>114</td>
<td>126</td>
<td>140</td>
<td>188</td>
<td>176</td>
<td>165</td>
<td>152</td>
<td>140</td>
<td>170</td>
<td>106</td>
<td>78</td>
<td>88</td>
</tr>
<tr>
<td>87</td>
<td>103</td>
<td>115</td>
<td>154</td>
<td>143</td>
<td>142</td>
<td>149</td>
<td>153</td>
<td>173</td>
<td>101</td>
<td>57</td>
<td>57</td>
</tr>
<tr>
<td>102</td>
<td>112</td>
<td>106</td>
<td>131</td>
<td>122</td>
<td>138</td>
<td>152</td>
<td>147</td>
<td>128</td>
<td>84</td>
<td>58</td>
<td>66</td>
</tr>
<tr>
<td>94</td>
<td>95</td>
<td>79</td>
<td>104</td>
<td>105</td>
<td>124</td>
<td>129</td>
<td>113</td>
<td>107</td>
<td>87</td>
<td>69</td>
<td>67</td>
</tr>
<tr>
<td>68</td>
<td>71</td>
<td>69</td>
<td>98</td>
<td>89</td>
<td>92</td>
<td>98</td>
<td>95</td>
<td>89</td>
<td>88</td>
<td>76</td>
<td>67</td>
</tr>
<tr>
<td>41</td>
<td>56</td>
<td>68</td>
<td>99</td>
<td>63</td>
<td>45</td>
<td>60</td>
<td>82</td>
<td>58</td>
<td>76</td>
<td>75</td>
<td>65</td>
</tr>
<tr>
<td>20</td>
<td>43</td>
<td>69</td>
<td>75</td>
<td>56</td>
<td>41</td>
<td>51</td>
<td>73</td>
<td>55</td>
<td>70</td>
<td>63</td>
<td>44</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>57</td>
<td>69</td>
<td>75</td>
<td>75</td>
<td>73</td>
<td>74</td>
<td>53</td>
<td>68</td>
<td>59</td>
<td>37</td>
</tr>
<tr>
<td>72</td>
<td>59</td>
<td>53</td>
<td>66</td>
<td>84</td>
<td>92</td>
<td>84</td>
<td>74</td>
<td>57</td>
<td>72</td>
<td>63</td>
<td>42</td>
</tr>
<tr>
<td>67</td>
<td>61</td>
<td>58</td>
<td>65</td>
<td>75</td>
<td>78</td>
<td>76</td>
<td>73</td>
<td>59</td>
<td>75</td>
<td>69</td>
<td>50</td>
</tr>
</tbody>
</table>
Pixel-based representation

Input

Raw image

Learning algorithm

Motorbikes
"Non"-Motorbikes
Pixel-based representation

Input

Raw image

+ Motorbikes
- “Non”-Motorbikes

Learning algorithm
Pixel-based representation

Input

Raw image

Learning algorithm

Motorbikes
“Non”-Motorbikes
What we want

Input

Raw image

Feature representation

E.g., Does it have Handlebars? Wheels?

Motorbikes
“Non”-Motorbikes

Learning algorithm

Features

pixel 2

pixel 1

Wheels

Handlebars
Some feature representations

- **SIFT**
  - Image gradients
  - Keypoint descriptor

- **Spin image**
  - Normalized patch
  - Spin image

- **HoG**
  - Input Image
  - Gradient Image
  - Orientation Voting
  - Overlapping Blocks
  - Local Normalization

- **RIFT**
  - Normalized patch

- **Textons**

- **GLOH**
  - (a)
  - (b)
  - (c)
  - (d)
  - (e)
Some feature representations

Coming up with features is often difficult, time-consuming, and requires expert knowledge.
The brain: a potential motivation for deep learning

Auditory cortex learns to see!

Auditory cortex

[Roe et al., 1992]
The brain adapts!

- Seeing with your tongue
- Human echolocation (sonar)
- Haptic belt: Direction sense
- Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]
Basic idea of deep learning

• Also referred to as representation learning

• Is there some way to extract meaningful features from data in a supervised or unsupervised manner?

• Then, throw in some hierarchical ‘stuff’ to make it ‘deep’
Part 2:
Supervised deep learning
Speech recognition and Deep Learning

Posted by Vincent Vanhoucke, Research Scientist, Speech Team

The New York Times recently published an article about Google’s large scale deep learning project, which learns to discover patterns in large datasets, including... cats on YouTube!

What’s the point of building a gigantic cat detector you might ask? When you combine large amounts of data, large-scale distributed computing and powerful machine learning algorithms, you can apply the technology to address a large variety of practical problems.

With the launch of the latest Android platform release, Jelly Bean, we’ve taken a significant step towards making that technology useful: when you speak to your Android phone, chances are, you are talking to a neural network trained to recognize your speech.

Using neural networks for speech recognition is nothing new: the first proofs of concept were developed in the late
Impact on speech recognition

Word error rate on Switchboard

1990  2000  2010

Using DL
Application to Google Streetview
Object recognition

1000-way image classification

Classification error

<table>
<thead>
<tr>
<th>Year</th>
<th>Classification error</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>0.3</td>
</tr>
<tr>
<td>2011</td>
<td>0.25</td>
</tr>
<tr>
<td>2012</td>
<td>0.2</td>
</tr>
<tr>
<td>2013</td>
<td>0.15</td>
</tr>
<tr>
<td>2014</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Object recognition

BC (before ConvNets)
AD (after deep learning)

6.8%
Object recognition

BC (before ConvNets) vs AD (after deep learning)

Classification error

ImageNet Challenge Year
Neural networks
Neural networks
Neural networks
Neural networks

Diagram showing neuron structure with dendrites, cell body, axon, and synaptic connections.
A single neuron can be used as a binary linear classifier e.g., logistic regression
Activation functions

sigmoid activation function
\[
\frac{1}{1 + e^{-x}}
\]

tanh(x)

ReLU
\[
f(x) = \max(0, x)
\]
From neurons to neural network
Neural networks: architectures
Neural networks: architectures

“2-layer neural net,” or “1-hidden-layer neural net”

“3-layer neural net,” or “2-hidden-layer neural net”
Neural network: architectures

Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?
Neural network: architectures

Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?
Neural network: architectures

Number of Neurons: $4 + 2 = 6$
Number of Weights: $[4 \times 3 + 2 \times 4] = 20$
Number of Parameters: $20 + 6 = 26$ (biases!)

Number of Neurons: $4 + 4 + 1 = 9$
Number of Weights: $[4 \times 3 + 4 \times 4 + 1 \times 4] = 32$
Number of Parameters: $32 + 9 = 41$
Neural network: architectures

Modern CNNs: ~10 million neurons
Human visual cortex: ~5 billion neurons
Training a neural network

Given training set \((x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots\)

Adjust parameters \(\theta\) (for every node) to make:

\[
    h_\theta(x_i) \approx y_i
\]

(Use gradient descent - “Backpropagation” algorithm)
Convolutional neural networks

[LeNet-5, LeCun 1980]
Convolutional neural networks aka ConvNets aka CNNs aka Computer vision Savior

[LeNet-5, LeCun 1980]
ConvNets: architecture

before:

input layer

hidden layer

output layer

now:
ConvNets: architecture

All Neural Net activations arranged in 3 dimensions:

For example, a CIFAR-10 image is a 32x32x3 volume: 32 width, 32 height, 3 depth (RGB channels)
ConvNets: architecture

Convolutional Neural Networks are just Neural Networks BUT:
1. **Local connectivity**

![Diagram of local connectivity](image)

- a hidden neuron in next layer

![Neuron diagram](image)
Convolutions Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

   - image: 32x32x3 volume
   - before: full connectivity: 32x32x3 weights
   - now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

   - note that connectivity is:
     - **local in space** (5x5 inside 32x32)
     - **but full in depth** (all 3 depth channels)
ConvNets: architecture

Convolutional Neural Networks are just Neural Networks BUT:
1. Local connectivity

Multiple neurons all looking at the same region of the input volume, stacked along depth.
ConvNets: architecture

Convolutional Neural Networks are just Neural Networks BUT:
1. Local connectivity

These form a single [1 x 1 x depth] “depth column” in the output volume
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output
Replicate this column of hidden neurons across space, with some **stride**.

7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 2?
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 2?
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 2?
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1

---> 5x5 output

7x7 input
assume 3x3 connectivity, stride 2

---> 3x3 output
ConvNets: architecture

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

7x7 input
assume 3x3 connectivity, stride 1

--> 5x5 output

What about stride 3?
ConvNets: architecture

Replicate this column of hidden neurons across space, with some \textit{stride}.

7x7 input
assume 3x3 connectivity, stride 1

\textit{--> 5x5 output}

What about stride 3? \textbf{CANNOT}
ConvNets: architecture

Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7\), \(F = 3\):
- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = \ldots\)
ConvNets: architecture

Examples time:

Input volume: **32x32x3**
Receptive fields: **5x5, stride 1**
Number of neurons: **5**

Output volume: ?
ConvNets: architecture

Examples time:

Input volume: **32x32x3**
Receptive fields: **5x5, stride 1**
Number of neurons: **5**

Output volume: \((32 - 5) / 1 + 1 = 28\), so: **28x28x5**
How many weights for each of the **28x28x5** neurons?
ConvNets: architecture

Examples time:

Input volume: $32 \times 32 \times 3$
Receptive fields: $5 \times 5$, stride 1
Number of neurons: 5

Output volume: \((32 - 5) / 1 + 1 = 28\), so: $28 \times 28 \times 5$
How many weights for each of the $28 \times 28 \times 5$ neurons? $5 \times 5 \times 3 = 75$
ConvNets: architecture

Examples time:

Input volume: \textbf{32x32x3}
Receptive fields: \textbf{5x5, stride 2}
Number of neurons: \textbf{5}

Output volume: ?
ConvNets: architecture

Examples time:

Input volume: \textbf{32x32x3}
Receptive fields: \textbf{5x5, stride 2}
Number of neurons: \textbf{5}

Output volume: ? \textbf{Cannot}: (32-5)/2 + 1 = 14.5
ConvNets: architecture

Input volume of size \([W_1 \times H_1 \times D_1]\)
using \(K\) neurons with receptive fields \(F \times F\) and applying them at strides of \(S\) gives

Output volume: \([W_2, H_2, D_2]\)

\[
\begin{align*}
W_2 &= (W_1 - F)/S + 1 \\
H_2 &= (H_1 - F)/S + 1 \\
D_2 &= K
\end{align*}
\]
There’s one more problem...
Assume input $[32 \times 32 \times 3]$
30 neurons with receptive fields $5\times5$, applied at stride 1/pad1:
=> Output volume: $[32 \times 32 \times 30]$  \( (32*32*30 = 30720 \text{ neurons}) \)
Each neuron has $5*5*3$ (=75) weights
=> Number of weights in such layer: $30720 \times 75 \approx 3 \text{ million}$ \( :\)
There's one more problem...
Assume input [32 x 32 x3]
30 neurons with receptive fields 5x5, applied at stride 1/pad 1:
=> Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)
Each neuron has 5*5*3 (=75) weights
=> Number of weights in such layer: 30720 * 75 =~ 3 million :\
There's one more problem...
Assume input $[32 \times 32 \times 3]$
30 neurons with receptive fields $5\times 5$, applied at stride 1/pad1:
=> Output volume: $[32 \times 32 \times 30]$ ($32\times32\times30 = 30720$ neurons)
Each neuron has $5\times5\times3$ (=75) weights
=> Number of weights in such layer: $30720 \times 75 \approx 3$ million

IDEA: let's not learn the same thing across all spatial locations
ConvNets: architecture

Our first ConvNet layer had size \([32 \times 32 \times 3]\)
If we had 30 neurons with receptive fields 5x5, stride 1, pad 1
Output volume: \([32 \times 32 \times 30]\) \((32\times32\times30 = 30720 \text{ neurons})\)
Each neuron has 5x5x3 \((=75)\) weights

Before:
#weights in such layer: \((32\times32\times30) \times 75 = 3 \text{ million} \)\]

Now: (parameter sharing)
#weights in the layer: 30 \times 75 = 2250.
ConvNets: architecture

These layers are called **Convolutional Layers**

1. Connect neurons only to local receptive fields
2. Use the same neuron weight parameters for neurons in each "depth slice" (i.e. across spatial positions)
ConvNets: architecture

one filter = one depth slice (or activation map)

5x5 filters

Can call the neurons “filters”

We call the layer convolutional because it is related to convolution of two signals (kind of):

\[ f[x,y] \ast g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2] \]

elementwise multiplication and sum a filter and the signal (image)
Fast-forward to today [From recent Yann LeCun slides]

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]
ConvNets: Max pooling

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)
ConvNets: Max pooling

Single depth slice

```
  1 1 2 4
  5 6 7 8
  3 2 1 0
  1 2 3 4
```

max pool with 2x2 filters and stride 2

```
  6 8
  3 4
```
ConvNets: Max pooling

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers.

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)

*Input volume of size* \([W_1 \times H_1 \times D_1]\)

Pooling unit receptive fields \(F \times F\) and applying them at strides of \(S\) gives

*Output volume:* \([W_2, H_2, D_1]\)

\[
W_2 = \frac{(W_1-F)}{S}+1, \quad H_2 = \frac{(H_1-F)}{S}+1
\]

Note: pooling happens independently across each slice, preserving number of slices.

E.g. a pooling “neuron” of size 2x2 will perform MAX operation over 4 numbers.
ConvNets: summary
ConvNets: tips & tricks

In practice: Common to zero pad the border
(in each channel)

e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?
**ConvNets: tips & tricks**

**In practice: Common to zero pad the border**

(in each channel)

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

- e.g. input 7x7 neuron with receptive field 3x3, stride 1 pad with 1 pixel border => what is the output?

- 7x7 => preserved size!

- in general, common to see stride 1, size F, and zero-padding with \((F-1)/2\). (Will preserve input size spatially)
ConvNets: tips & tricks

- start with image that has power-of-2 size
- for **conv layers**, use stride 1 filter size 3x3 pad input with a border of zeros (1 spatially)

This makes it so that: $[W_1,H_1,D_1] \rightarrow [W_1,H_1,D_2]$ (i.e. spatial size exactly preserved)

- for **pool layers**, use pool size 2x2 (more = worse)
Part 3:
Unsupervised deep learning
Methods of Unsupervised Deep Learning

• Autoencoders

• Deep belief networks (DBNs)
  • Restricted Boltzmann Machines (RBMs)
  • Deep Boltzmann Machines (DBMs)

• Sparse coding
Feature learning problem

• Given a 14x14 image patch $x$, we can represent it using 196 real numbers.

\[
\begin{pmatrix}
255 \\
98 \\
93 \\
87 \\
89 \\
91 \\
48 \\
\ldots
\end{pmatrix}
\]

• Problem: Can we find a better feature vector to represent this?
First stage of visual processing: V1

V1 is the first stage of visual processing in the brain. Neurons in V1 typically modeled as edge detectors:

Neuron #1 of visual cortex (model)  Neuron #2 of visual cortex (model)
Learning sensor representations

Sparse coding (Olshausen & Field, 1996)

Input: Images $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (each in $\mathbb{R}^{n \times n}$)

Learn: Dictionary of bases $\phi_1, \phi_2, \ldots, \phi_k$ (also $\mathbb{R}^{n \times n}$), so that each input $x$ can be approximately decomposed as:

$$x \approx \sum_{j=1}^{k} a_j \phi_j$$

s.t. $a_j$’s are mostly zero ("sparse")
Sparse coding illustration

Natural Images

Learned bases \((\phi_1, \ldots, \phi_{64})\): “Edges”

Test example

\[
x \approx 0.8 \ast \phi_{36} + 0.3 \ast \phi_{42} + 0.5 \ast \phi_{63}
\]

\([a_1, \ldots, a_{64}] = [0, 0, \ldots, 0, 0.8, 0, \ldots, 0, 0.3, 0, \ldots, 0, 0.5, 0]\)

(feature representation)
Sparse coding illustration

Represent as: \[ a_{15} = 0.6, \ a_{28} = 0.8, \ a_{37} = 0.4 \]

\[ 0.6 \ast \phi_{15} + 0.8 \ast \phi_{28} + 0.4 \ast \phi_{37} \]

Represent as: \[ a_{5} = 1.3, \ a_{18} = 0.9, \ a_{29} = 0.3 \]

\[ 1.3 \ast \phi_{5} + 0.9 \ast \phi_{18} + 0.3 \ast \phi_{29} \]

• **Method “invents” edge detection**

• Automatically learns to represent an image in terms of the edges that appear in it. Gives a more succinct, higher-level representation than the raw pixels.

• Quantitatively similar to primary visual cortex (area V1) in brain.
Going deep

Training set: Aligned images of faces.

object models

object parts (combination of edges)

edges

pixels

[Honglak Lee]
Network is trained to output the input (learn identify function).

\[ h_\theta(x) \approx x \]

Trivial solution unless:
- Constrain number of units in Layer 2 (learn compressed representation), or
- Constrain Layer 2 to be **sparse**.
Autoencoder

Layer 1: \(x_1, x_2, x_3, x_4, x_5, x_6, +1\)

Layer 2: \(a_1, a_2, a_3, +1\)

Layer 3: \(\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4, \hat{x}_5, \hat{x}_6\)

\(h_\theta(x)\)
Autoencoder

Layer 1

\[ x_1 \]
\[ x_2 \]
\[ x_3 \]
\[ x_4 \]
\[ x_5 \]
\[ x_6 \]
\[ +1 \]

Layer 2

\[ a_1 \]
\[ a_2 \]
\[ a_3 \]
\[ +1 \]

New representation for input.

\[
\begin{bmatrix}
a_1 \\ a_2 \\ a_3
\end{bmatrix}
\]
Autoencoder

Layer 1

Layer 2

+1

+1

x_1

x_2

x_3

x_4

x_5

x_6

Layer 1

Layer 2

a_1

a_2

a_3

a_1

a_2

a_3
Train parameters so that $h_\theta(x) \approx a$, subject to $b_i$'s being sparse.
Autoencoder

Train parameters so that \( h_\theta(x) \approx a \), subject to \( b_i \)'s being sparse.
Deep Belief Net

- Deep Belief Net (DBN) is another algorithm for learning a feature hierarchy.

- Building block: 2-layer graphical model (Restricted Boltzmann Machine).

- Can then learn additional layers one at a time.
Restricted Boltzmann Machine (RBM)

Layer 2. \([a_1, a_2, a_3]\) (binary-valued)

Input \([x_1, x_2, x_3, x_4]\)

MRF with joint distribution:

\[
P(x, a) \propto \exp \left( - \sum_{i,j} x_i a_j W_{i,j} \right)
\]

Use Gibbs sampling for inference.

Given observed inputs \(x\), want maximum likelihood estimation:

\[
\max_W P(x) = \max_W \sum_a P(x, a)
\]
Deep Belief Network

Similar to a sparse autoencoder in many ways. Stack RBMs on top of each other to get DBN.

Input $[x_1, x_2, x_3, x_4]$

Layer 2. $[a_1, a_2, a_3]$

Layer 3. $[b_1, b_2, b_3]$
Deep Belief Network

Layer 4. \([c_1, c_2, c_3]\)

Layer 3. \([b_1, b_2, b_3]\)

Layer 2. \([a_1, a_2, a_3]\)

Input \([x_1, x_2, x_3, x_4]\)
Convolutional DBN for audio

Max pooling unit

Detection units
Convolutional DBN for audio
Convolutional deep belief networks illustration

Input image

Layer 1 activation (coefficients)

Layer 2 activation (coefficients)

Layer 3 activation (coefficients)

Filter visualization
Benefits of unsupervised feature learning

Task: video activity recognition

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hessian + ESURF [Williems et al 2008]</td>
<td>38%</td>
</tr>
<tr>
<td>Harris3D + HOG/HOF [Laptev et al 2003, 2004]</td>
<td>45%</td>
</tr>
<tr>
<td>Cuboids + HOG/HOF [Dollar et al 2005, Laptev 2004]</td>
<td>46%</td>
</tr>
<tr>
<td>Dense + HOG / HOF [Laptev 2004]</td>
<td>47%</td>
</tr>
<tr>
<td>Cuboids + HOG3D [Klaser 2008, Dollar et al 2005]</td>
<td>46%</td>
</tr>
<tr>
<td>Unsupervised feature learning (our method)</td>
<td>52%</td>
</tr>
</tbody>
</table>
# Table of Results

## Audio

<table>
<thead>
<tr>
<th>Task</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIMIT Phone classification</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Clarkson et al., 1999)</td>
<td>79.6%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>80.3%</strong></td>
</tr>
<tr>
<td><strong>TIMIT Speaker identification</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Reynolds, 1995)</td>
<td>99.7%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>100.0%</strong></td>
</tr>
</tbody>
</table>

## Images

<table>
<thead>
<tr>
<th>Task</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CIFAR Object classification</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Ciresan et al., 2011)</td>
<td>80.5%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>82.0%</strong></td>
</tr>
<tr>
<td><strong>NORB Object classification</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Scherer et al., 2010)</td>
<td>94.4%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>95.0%</strong></td>
</tr>
</tbody>
</table>

## Video

<table>
<thead>
<tr>
<th>Task</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hollywood2 Classification</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Laptev et al., 2004)</td>
<td>48%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>53%</strong></td>
</tr>
<tr>
<td><strong>KTH</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Wang et al., 2010)</td>
<td>92.1%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>93.9%</strong></td>
</tr>
<tr>
<td><strong>UCF</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Wang et al., 2010)</td>
<td>85.6%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>86.5%</strong></td>
</tr>
</tbody>
</table>

## Text/NLP

<table>
<thead>
<tr>
<th>Task</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Paraphrase detection</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Das &amp; Smith, 2009)</td>
<td>76.1%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>76.4%</strong></td>
</tr>
<tr>
<td><strong>Sentiment (MR/MPQA data)</strong></td>
<td></td>
</tr>
<tr>
<td>Prior art (Nakagawa et al., 2010)</td>
<td>77.3%</td>
</tr>
<tr>
<td>Feature learning</td>
<td><strong>77.7%</strong></td>
</tr>
</tbody>
</table>
ImageNet classification: 22,000 classes

... smoothhound, smoothhound shark, Mustelus mustelus
American smooth dogfish, Mustelus canis
Florida smoothhound, Mustelus norrisi
whitetip shark, reef whitetip shark, Triaenodon obesus
Atlantic spiny dogfish, Squalus acanthias
Pacific spiny dogfish, Squalus suckleyi
hammerhead, hammerhead shark
smooth hammerhead, Sphyrna zygaena
smalleye hammerhead, Sphyrna tudes
shovelhead, bonnethead, bonnet shark, Sphyrna tiburo
angel shark, angelfish, Squatina squatina, monkfish
electric ray, crampfish, numbfish, torpedo
guitartfish
smalltooth sawfish, Pristis pectinatus
guitarfish
roughtail stingray, Dasyatis centroura
butterfly ray
eagle ray
spotted eagle ray, spotted ray, Aetobatus narinari
cownose ray, cow-nosed ray, Rhinoptera bonasus
manta, manta ray, devilfish
Atlantic manta, Manta birostris
devil ray, Mobula hypostoma
grey skate, gray skate, Raja batis
little skate, Raja erinacea
...

Stingray

Mantaray
ImageNet Classification: 14M images, 22k categories

0.005% 9.5% ？
Random guess State-of-the-art (Weston, Bengio ‘11) Feature learning From raw pixels

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012
ImageNet Classification: 14M images, 22k categories

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random guess</td>
<td>0.005%</td>
</tr>
<tr>
<td>State-of-the-art (Weston, Bengio ‘11)</td>
<td>9.5%</td>
</tr>
<tr>
<td>Feature learning From raw pixels</td>
<td>21.3%</td>
</tr>
</tbody>
</table>

Le, et al., *Building high-level features using large-scale unsupervised learning*. ICML 2012