Artificial Neural Networks: Intro

“Making Connections” by Filomena Booth (2013)
Sample task

- Training set: 6 actors, with 100 $64 \times 64$ photos of faces for each
- Test set: photos of faces of the same 6 actors
- Want to classify each face as one of ['Fran Drescher', 'America Ferrera', 'Kristin Chenoweth', 'Alec Baldwin', 'Bill Hader', 'Steve Carell']
Images  ↔  Vectors

Images:

Vectors:

60  60  255  255
60  60  255  255
60  60  255  255
128 128 128 128

60
60
255
255
60
60
255
255
255
255
60
60
255
255
128
128
128
128
128
128
The Face Recognition Task

- **Training set:**
  - \(\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)})\}\)
    - \(x^{(i)}\) is a k-dimensional vector consisting of the intensities of all the pixels in in the i-th photo (20 \(\times\) 20 photo \(\rightarrow\) \(x^{(i)}\) is 400-dimensional)
    - \(y^{(i)}\) is the label (i.e., name)

- **Test phase:**
  - We have an input vector \(x\), and want to assign a label \(y\) to it
    - Whose photo is it?
Reminder: Face Recognition using 1-Nearest Neighbors (1NN)

- Training set: \{((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(N)}, y^{(N)}))\}
- Input: x
- 1-Nearest Neighbor algorithm:
  - Find the training photo/vector $x^{(i)}$ that’s as “close” as possible to $x$, and output the label $y^{(i)}$
The Simplest Possible Neural Network for Face Recognition

\[
z_k = \sigma \left( \sum_{j=1}^{4096} W^{(1,j,k)} x_j + b^{(1,k)} \right)
\]

\[
= \sigma \left( W^{(1,*,k)} \cdot x + b^{(1,k)} \right)
\]

\[
h_{\theta} = h_{W,b}
\]
Training a neural network

• Adjust the $W$'s ($4096 \times 6$ coefs) and $b$'s (6 coefs)

• Try to make it so that if
  - $x$ is an image of actor 1, $z$ is as close as possible to (1, 0, 0, 0, 0, 0)
  - $x$ is an image of actor 2, $z$ is as close as possible to (0, 1, 0, 0, 0, 0)
  ......
Training a neural network

• Adjust the W’s (4096 × 6 coefs) and b’s (6 coefs)
  • Try to make it so that if
    \( x \) is an image of actor 1, \( z \) is as close as possible to (1, 0, 0, 0, 0, 0)
    \( x \) is an image of actor 2, \( z \) is as close as possible to (0, 1, 0, 0, 0, 0)
    ......
Face recognition

- Compute the \( z \) for a new image \( x \)
- If \( z_k \) is the largest output, output name \( k \)
An interpretation

$z_1$ is large if $W^{(1,*,1)} \cdot x$ is large
$z_2$ is large if $W^{(1,*,2)} \cdot x$ is large
$z_3$ is large if $W^{(1,*,3)} \cdot x$ is large

....

$W^{(1,*,1)}, W^{(1,*,2)}, ..., W^{(1,*,6)}$ are *templates* for the faces of actor 1, actor 2, ..., actor 6

**Actor 3 neuron activated:**

$$\sigma(W^{(1,*,3)} \cdot x + b^{(1,3)})$$ is large

---

**Diagram:**

- $x_1$ ...
- $x_{20}$ ...
- $x_{4096}$
- $W^{(1,1,3)}$ ...
- $W^{(1,20,3)}$ ...
- $W^{(1,4096,3)}$
- $b^{(1,3)}$

**Input vector:** (flattened 64x64 Image)
Visualizing the parameters \( W \)

- Baldwin: \( W^{(1,*1)} \)
- Carrel: \( W^{(1,*2)} \)
- Hader: \( W^{(1,*3)} \)
- Ferrera: \( W^{(1,*4)} \)
- Drescher: \( W^{(1,*5)} \)
- Chenoweth: \( W^{(1,*6)} \)

![Diagram of neural network with input vector (flattened 64x64 image) and outputs (one per actor).]
Deep Neural Networks: Introducing Hidden Layers

\[
h_k = \sigma(W^{(1,*,k)} \cdot x + b^{(1,k)}) \\
z_m = \sigma(W^{(2,*,m)} \cdot h + b^{(2,m)})
\]
Why a hidden layer?

• Instead of checking whether \( x \) looks like one of 6 templates, we’ll be checking whether \( x \) looks like one of \( K \) templates, for a large \( K \)

\[
\begin{align*}
    h_k &= \sigma(W^{(1,*,k)} \cdot x + b^{(1,k)}) \\
    z_m &= \sigma(W^{(2,*,m)} \cdot h + b^{(2,m)})
\end{align*}
\]
Recap: Face Recognition with ML

- 1-Nearest-Neighbor: match $x$ to all the images in the training set
- 0-hidden-layer neural network*: match $x$ to several templates, with one template per actor
  - The templates work better than any individual photo
- 1-hidden-layer neural network: match $x$ to $K$ templates
  - The templates work better than any individual photo
  - More templates means better accuracy on the training set

*A.K.A. multinomial logistic regression to its friends
Visualizing a One-Hidden-Layer NN
 Demo

http://playground.tensorflow.org/
Deep Neural Networks as a Model of Computation

• Most people’s first instinct a face classifier is to write a complicated computer program
• A deep neural network is a computer program:
  \[
  h_1 = f_1(x) \\
  h_2 = f_2(h_1) \\
  h_3 = f_3(h_2) \\
  \ldots \\
  h_9 = f_9(h_8)
  \]

• Can think of every layer of a neural network as one step of a parallel computation
• Features/templates are the functions that are applied to the previous layers
• Learning features ⇔ Learning what function to apply at step t of the algorithm
What are the hidden units doing?

• Find the images in the dataset that activate the units the most

• *Let’s see some visualizations of neurons of a large deep network trained to recognize objects in images*
  • Then network classifies images as one of 1000 objects (sample objects: toy poodle, flute, forklift, goldfish...)
  • The network has 8 layers
  • Note: more tricks were used in designing the networks than we have time to mention! In particular, a convolutional architecture is crucial
Units in Layer 3

Matthew Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks” (ECCV 2014)
Units in Layer 4

Matthew Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks” (ECCV 2014)
Units in Layer 5

Matthew Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks” (ECCV 2014)
Which pixels are responsible for the output?

• For each pixel in a particular image ask:
  • If I changed this pixel $j$ by a little bit, how would that influence the output $i$?
  • Equivalent to asking: what’s the gradient $\frac{\partial \text{output}_i}{\partial \text{input}_j}$?
  • We can visualize why a particular output was chosen by the network by computing $\frac{\partial \text{output}_i}{\partial \text{input}_j}$ for every $j$, and displaying that as an image.
Gradient and Guided Backpropagation

<table>
<thead>
<tr>
<th>Image I</th>
<th><img src="image1.png" alt="Image" /></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Delta$Cat-Neuron $\frac{\partial}{\partial I}$</td>
<td><img src="image2.png" alt="Guided Backpropagation visualization" /></td>
</tr>
<tr>
<td>Guided Backpropagation visualization</td>
<td><img src="image3.png" alt="Guided Backpropagation visualization" /></td>
</tr>
</tbody>
</table>
Guided backpropagation

- Instead of computing $\frac{\partial p_m}{\partial x}$, only consider paths from $x$ to $p_m$ where the weights are positive and all the units are positive (and greater than 0). Compute this modified version of $\frac{\partial p_m}{\partial x}$.

- Only consider evidence for neurons being active, discard evidence for neurons having to be not active.
Guided Backpropagation Intuition

Pixel provides both positive (via a cat eye detection) and negative (via absence of cat eye detection) evidence for a cat in the image.
Application: Photo Orientation

- Detect the correct orientation of a consumer photograph
- Input photo is rotated by 0°, 90°, 180° or 270°
- Help speed up the digitization of analog photos
- Need correctly oriented photos as inputs for other systems
A Neural Network for Photo Orientation

Layer legend:
- Convolution - ReLU
- Max pooling
- Fully Connected
- Softmax

224x224x3  224x224x64
112x112x128
56x56x256
28x28x512
14x14x512
7x7x512
4096
4096
4
Correctly Oriented Photos

• Display pixels that provide direct positive evidence for $0^\circ$
Incorrectly-oriented photos