Understanding How Neural Networks See
Recent successes of neural networks

› Can recognize what object is in the photo
› Can tell bad Go positions/shapes from good Go positions
› Can tell a self-driving car where to go
› Can decide on what key to press to win at a video game by looking at the screen
About this lecture

› A very brief introduction to artificial neural networks (ANNs)
  – Why and how ANNs work
› A very brief introduction to Explainable AI
  – Understanding how “black box” models work
“Review:” Supervised Machine Learning

Training set:
- Training example 1: \( x^{(1)} = (x_{1}^{(1)}, x_{2}^{(1)}, ..., x_{m}^{(1)}) \) output: \( y^{(1)} \)
- Training example 2: \( x^{(2)} = (x_{1}^{(2)}, x_{2}^{(2)}, ..., x_{m}^{(2)}) \) output: \( y^{(2)} \)
- ...
- Training example N: \( x^{(N)} = (x_{1}^{(N)}, x_{2}^{(N)}, ..., x_{m}^{(N)}) \) output: \( y^{(N)} \)

Test set:
- Test Example 1: \( x^{(N+1)} = (x_{1}^{(N+1)}, x_{2}^{(N+1)}, ..., x_{m}^{(N+1)}) \) output: \( y^{(N+1)} \)
- Test Example 2: \( x^{(N+2)} = (x_{1}^{(N+2)}, x_{2}^{(N+2)}, ..., x_{m}^{(N+2)}) \) output: \( y^{(N+2)} \)
- ...
- Test Example K: \( x^{(N+K)} = (x_{1}^{(N+K)}, x_{2}^{(N+K)}, ..., x_{m}^{(N+K)}) \) output: \( y^{(N+K)} \)

- Goal: Find a \( \theta \) such that \( h_{\theta}(x^{(i)}) \approx y^{(i)} \) for \( i \in 1, ..., N \)
- Hope: \( h_{\theta}(x^{(i)}) \approx y^{(i)} \) for any \( i \)
- For new input \( x \), predict \( h_{\theta}(x) \)
Machine Learning vs. Intro to Programming

› Programming **done badly**

```r
CountryMaxIncome <- function(gap):
  return(min(gap$gdpPercap))

> CountryMaxIncome(gapminder)
10
```

› Machine Learning **done right**

```r
>>> h_0,1,2,0,1([0, 0])
[0, 0]
>>> h_0,1,2,0,1([1, 2])
[1.3, 2.8]
```

\[ h_{(\theta_1, \theta_2, \theta_3)}(x) = \theta_1 + \theta_2 x + \theta_3 x^2 \]
Sample ML task: Recognizing Justin Bieber
What Justin Bieber looks like to a computer
The Face Recognition Task

› Training set:
  – \{((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (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 × 20 photo → \(x^{(i)}\) 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?
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 \(y^{(i)}\)

Input \(x\)

Closest training image to the input \(x\)

Output: Paul
Supervised Machine Learning

Training set:
- Training example 1: \( x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \ldots, x_m^{(1)}) \) output: \( y^{(1)} \)
- Training example 2: \( x^{(2)} = (x_1^{(2)}, x_2^{(2)}, \ldots, x_m^{(2)}) \) output: \( y^{(2)} \)
- …
- Training example N: \( x^{(N)} = (x_1^{(N)}, x_2^{(N)}, \ldots, x_m^{(N)}) \) output: \( y^{(N)} \)

Test set:
- Test Example 1: \( x^{(N+1)} = (x_1^{(N+1)}, x_2^{(N+1)}, \ldots, x_m^{(N+1)}) \) output: \( y^{(N+1)} \)
- Test Example 2: \( x^{(N+2)} = (x_1^{(N+2)}, x_2^{(N+2)}, \ldots, x_m^{(N+2)}) \) output: \( y^{(N+2)} \)
- …
- Test Example K: \( x^{(N+K)} = (x_1^{(N+K)}, x_2^{(N+K)}, \ldots, x_m^{(N+K)}) \) output: \( y^{(N+K)} \)

- Goal: Find a \( \theta \) such that \( h_\theta(x^{(i)}) \approx y^{(i)} \) for \( i \in 1, \ldots, N \)
- Hope: \( h_\theta(x^{(i)}) \approx y^{(i)} \) for any \( i \)
- For new input \( x \), predict \( h_\theta(x) \)
Are the two images $a$ and $b$ close?

› Key idea: think of the images as **vectors**
  - Reminder: to turn an image into a vector, simply “flatten” all the pixels into a 1D vector

› Is the distance between the endpoints of vectors $a$ and $b$ small?

$$|a - b| = \sqrt{\sum_i (a_i - b_i)^2} \text{ small}$$

› Is the cosine of the angle between the vectors $a$ and $b$ large?

$$\cos \theta_{ab} = \frac{a \cdot b}{|a||b|} = \frac{\sum_i a_i b_i}{\sqrt{\sum_i a_i^2} \sqrt{\sum_i b_i^2}} \text{ large}$$

› Is $a \cdot b = \sum_i a_i b_i$ large?
  - Assume $|a| \approx |b| \approx \text{const}$
SML310 Project 3 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']
The Simplest Possible Neural Network for Face Recognition

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

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

\[ h_\theta = h_{W,b} \]

The transformation with \( \sigma \) is not necessary here, but will be useful later.
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 \((0, 1, 0, 0, 0, 0)\)
    ……

\( W(1, 1, 1) \)
\( b(1, 3) \)
\( W(1, 4096, 6) \)
\( b(1, 3) \)
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

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 $W^{(1,*,6)}$
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 \( K \)
  – If template \( k \) (i.e., \( W^{(1,k)} \)) looks like actor 6, \( W^{(2,k,6)} \) will be large
 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

** With minor modifications made to make this lecture clearer
Visualizing a One-Hidden-Layer NN
Deep Learning: More hidden layers!

$W^{(10)}$  

$W^{(1)}$  

$W^{(0)}$  

Templates for template matchers (combinations of very simple shapes)  

Templates for the input (very simple shapes)
Deep Neural Networks as a Model of Computation

› Most people’s first instinct when building a face classifier is to write a complicated computer program
› A deep neural network *is* a computer program:
  
  \[
  \begin{align*}
  h_1 &= f_1(x) \\
  h_2 &= f_2(h_1) \\
  h_3 &= f_3(h_2) \\
  & \vdots \\
  h_9 &= f_9(h_8)
  \end{align*}
  \]

› 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
Deep Neural Networks

› Can perform a wide range of computation
› Can be learned automatically
  – (using gradient descent)

- Powerful but not (computer) learnable: Python
  - Can’t make a learning algorithm that takes lots of inputs and outputs and produces Python code that generates the outputs on new inputs
  - (But can do it with simpler languages!)

- Learnable but not powerful:
  - Logistic regression
  - Deep Neural Networks that aren’t deep enough

Graphic and idea by Ilya Sutskever
The *Deep Learning Hypothesis*

› Human perception is fast
   – (Human) neurons fire at most 100 times a second
   – Humans can solve simple perceptual tasks in 0.1 seconds
     › So out neurons fire in a sequence of 10 times at most

Anyhing a human can do in 0.1 seconds, a big 10-layer neural network can do, too!

› Success stories:
   – Classifying images of objects
   – Classifying Go positions as good or bad
What are the hidden units doing?
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*
  
  – The 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 the 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 ("saliency map")
Gradient and Guided Backpropagation

<table>
<thead>
<tr>
<th>Image I</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Image" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \frac{\partial \text{Cat-Neuron}}{\partial I} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="gradient.png" alt="Gradient" /></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Guided Backpropagation visualization</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="visualization.png" alt="Visualization" /></td>
</tr>
</tbody>
</table>

Graphic and idea by Andrej Karpathy
Why the gradient with respect to the input is noisy

Pixel provides both positive (via a cat eye detection) and negative (via absence of cat eye detection) evidence for a cat in the image.
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
Questions?
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

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

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