Logistics

• More info on projects today/tomorrow but tl;dr:
  • We have pre-made projects and will help you match
  • If you’d like to design your own project, we’ll ask you to write things up a bit more

• HW3
  • Definitely go to discussion this week
  • No guarantees about piazza during spring break
...and I took that personally
Previously – Backpropagation

\[ f(x) = (-x + 3)^2 \]

Forward pass: compute function
Backward pass: compute derivative of all parts of the function
Setting Up A Neural Net

Input       Hidden       Output
x₁           h₁           y₁
x₂           h₂           y₂
                h₃           y₃
                h₄
Setting Up A Neural Net

Input  Hidden 1  Hidden 2  Output

x_1  a_1  h_1  y_1
x_2  a_2  h_2  y_2
          a_3  h_3  y_3
                      a_4  h_4
Fully Connected Network

Each neuron connects to each neuron in the previous layer.
Fully Connected Network

\[ h = f (Wa + b) \]

- All layer \( a \) values
- Neuron \( i \) weights, bias
- Activation function

\[ a = [a_1, a_2, a_3, a_4] \]
\[ w_i, b_i \]
\[ f \]
Fully Connected Network

Define New Block: “Linear Layer”
(It’s Technically Affine)

\[ L(n) = Wn + b \]

Can get gradient with respect to all the inputs
(do on your own; useful trick: have to be able
to do matrix multiply)
Fully Connected Network
Backpropagation lets us calculate derivative of the output/error with respect to all the Ws at a given point $x$. 
Putting It All Together – 1

Function: \( \text{NN}(x; W_i,b_i) \)
Parameterized by \( W = \{W_i,b_i\} \)
Putting It All Together

\[ W = \text{initializeWeights()} \]

for i in range(numIterations):
    #sample a batch
    batch = random.subset(0, #datapoints, K)
    batchX, batchY = dataX[batch], dataY[batch]
    #compute gradient with batch
    gradW = backprop(Loss(NN(batchX, W), batchY))
    #update W with gradient step
    W += -stepsize * gradW

return W
What Can We Represent?

\[ L(n) = Wn + b \]
What Can We Represent

- Recall: $ax+by+z$ is
  - proportional to **signed** distance to line
  - equal to signed distance if you do it right
- Generalization to N-D: hyperplane $w^T x + b$
Can We Train a Network To Do It?

$$y_1 = x_1 + x_2$$
Can We Train a Network To Do It?
Can We Train a Network To Do It?

\[
\max(w_1^T x + b, 0) + \max(-w_1^T x + b, 0) = \text{Distance to line defined by } w_1
\]

\[
\max(w_2^T x + b, 0) + \max(-w_2^T x + b, 0) = \text{Distance to line defined by } w_2
\]
Can We Train a Network To Do It?

Next layer computes: \( w_1 \) Distance - \( w_2 \) Distance > 0

Distance to \( w_1 \)

Distance to \( w_2 \)
Can We Train a Network To Do It?

Result: feedforward neural networks with a finite number of neurons in a hidden layer can approximate any reasonable* function


In practice, doesn’t give a practical guarantee. Why?

*Continuous, with bounded domain.
Developing Intuitions

There is no royal road to geometry. – Euclid

- Best way: play with data, be skeptical of everything you do, be skeptical of everything you are told
- Remember: this is linear algebra, not magic
- Common technique: How would you set the weights by hand if you were forced to be a deep net
Parameters

How many parameters does this network have?

Weights: 1x2
Parameters: 3 (bias!)
Parameters

How many parameters does this network have?

Weights: $1 \times 4 + 4 \times 2 = 12$
Parameters: $12 + 5 = 17$
Parameters

How many parameters does this network have?

Weights: $3 \times 4 + 4 \times 4 + 4 \times 2 = 36$

Parameters: $36 + 11 = 47$
Parameters

Make Px1 vector

P: 285x350 picture *(terrible!)*, H: 1000, O: 3
102 million parameters *(400MB)*
Parameters

- First layer converts all visual information into a single N dimensional vector.
- Suppose you want a neuron to represent $dx/dy$ at each pixel. How many neurons do you need?
- $2^P$!
Parameters

Make Px1 vector

P: 285x350, H: 2P, O: 3
100 billion parameters (400GB)
Convnets

Keep Spatial Resolution Around

Neural net:
Data: vector Fx1
Transform: matrix-multiply

Convnet:
Data: image HxWxF
Transform: convolution
**Convnet**

**Fully connected:**
Connects to everything

**Convnet:**
Connects locally

Slide credit: Karpathy and Fei-Fei
Convnet

Neuron is the same: weighted linear average

$$\sum_{i=1}^{F_h} \sum_{j=1}^{F_w} \sum_{k=1}^{c} F_{i,j,k} \ast I_{y+i,x+j,k}$$

Slide credit: Karpathy and Fei-Fei
Convnet

Neuron is the same: weighted linear average

Filter is local in space: sum only over $F_h \times F_w$ pixels

Filter is global over channels/depth: sum over all channels

$$F_{i,j,k} \ast I_{y+i,x+j,k}$$

Slide credit: Karpathy and Fei-Fei
Convnet

Get spatial output by sliding filter over image

\[
F_{i,j,k} * I_{y+i,x+j,k}
\]

Slide credit: Karpathy and Fei-Fei
Differences From Earlier Filtering

(a) #input channels can be greater than one
(b) forget you learned the difference between convolution and cross-correlation

\[
\text{Output}[1,2] = I[1,2]*F[1,1] + I[1,3]*F[1,2] + \ldots + I[3,4]*F[3,3]
\]
Convnet

How big is the output?

Height? $32 - 5 + 1 = 28$
Width? $32 - 5 + 1 = 28$
Channels? 1

One filter not very useful by itself
Multiple Filters
You’ve already seen this before

Input: 400x600x1

Output: 400x600x2
Convnet
Multiple out channels via multiple filters.
How big is the output?

Height? $32 - 5 + 1 = 28$
Width? $32 - 5 + 1 = 28$
Channels? 200

Slide credit: Karpathy and Fei-Fei
Convnet

Multiple out channels via multiple filters.

How big is the output?

Height? $32-5+1=28$

Width? $32-5+1=28$

Channels? 200

Slide credit: Karpathy and Fei-Fei
Convnet, Summarized

Neural net: series of matrix-multiples parameterized by $W, b$ + nonlinearity/activation
Fit by gradient descent

Convnet: series of convolutions parameterized by $F, b$ + nonlinearity/activation
Fit by gradient descent
One Additional Subtlety – Stride

Warmup: how big is the output spatially?

<table>
<thead>
<tr>
<th>F11</th>
<th>F12</th>
<th>F13</th>
<th>I14</th>
<th>I15</th>
<th>I16</th>
<th>I17</th>
</tr>
</thead>
<tbody>
<tr>
<td>F21</td>
<td>F22</td>
<td>F23</td>
<td>I24</td>
<td>I25</td>
<td>I26</td>
<td>I27</td>
</tr>
<tr>
<td>F31</td>
<td>F32</td>
<td>F33</td>
<td>I34</td>
<td>I35</td>
<td>I36</td>
<td>I37</td>
</tr>
<tr>
<td>I41</td>
<td>I42</td>
<td>I43</td>
<td>I44</td>
<td>I45</td>
<td>I46</td>
<td>I47</td>
</tr>
<tr>
<td>I51</td>
<td>I52</td>
<td>I53</td>
<td>I54</td>
<td>I55</td>
<td>I56</td>
<td>I57</td>
</tr>
<tr>
<td>I61</td>
<td>I62</td>
<td>I63</td>
<td>I64</td>
<td>I65</td>
<td>I66</td>
<td>I67</td>
</tr>
<tr>
<td>I71</td>
<td>I72</td>
<td>I73</td>
<td>I74</td>
<td>I75</td>
<td>I76</td>
<td>I77</td>
</tr>
</tbody>
</table>

Normal (Stride 1): 5x5 output

Example credit: Karpathy and Fei-Fei
One Additional Subtlety – Stride

Stride: skip a few (here 2)

Normal (Stride 1): 5x5 output

Example credit: Karpathy and Fei-Fei
One Additional Subtlety – Stride

Stride: skip a few (here 2)

Normal (Stride 1):
5x5 output

Example credit: Karpathy and Fei-Fei
One Additional Subtlety – Stride

Stride: skip a few (here 2)

<table>
<thead>
<tr>
<th>I11</th>
<th>I12</th>
<th>I13</th>
<th>I14</th>
<th>F11</th>
<th>F12</th>
<th>F13</th>
</tr>
</thead>
<tbody>
<tr>
<td>I21</td>
<td>I22</td>
<td>I23</td>
<td>I24</td>
<td>F21</td>
<td>F22</td>
<td>F23</td>
</tr>
<tr>
<td>I31</td>
<td>I32</td>
<td>I33</td>
<td>I34</td>
<td>F31</td>
<td>F32</td>
<td>F33</td>
</tr>
<tr>
<td>I41</td>
<td>I42</td>
<td>I43</td>
<td>I44</td>
<td>I45</td>
<td>I46</td>
<td>I47</td>
</tr>
<tr>
<td>I51</td>
<td>I52</td>
<td>I53</td>
<td>I54</td>
<td>I55</td>
<td>I56</td>
<td>I57</td>
</tr>
<tr>
<td>I61</td>
<td>I62</td>
<td>I63</td>
<td>I64</td>
<td>I65</td>
<td>I66</td>
<td>I67</td>
</tr>
<tr>
<td>I71</td>
<td>I72</td>
<td>I73</td>
<td>I74</td>
<td>I75</td>
<td>I76</td>
<td>I77</td>
</tr>
</tbody>
</table>

Normal (Stride 1): 5x5 output

Stride 2 convolution: 3x3 output

Example credit: Karpathy and Fei-Fei
One Additional Subtlety – Stride

What about stride 3?

Normal (Stride 1): 5x5 output

Stride 2 convolution: 3x3 output

Example credit: Karpathy and Fei-Fei
One Additional Subtlety – Stride

What about stride 3?

Normal (Stride 1): 5x5 output

Stride 2 convolution: 3x3 output

Stride 3 convolution: Doesn’t work!

Example credit: Karpathy and Fei-Fei
One Additional Subtlety

Zero padding is extremely common, although other forms of padding do happen.

- **Symm**: fold sides over
- **Circular/Wrap**: wrap around
- **pad/fill**: add value, often 0
In General

\[
\text{Output Size} = \frac{(N - F)}{S} + 1
\]
More Examples

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

\[
\frac{(N - F)}{s} + 1
\]

Output volume size?

Slide credit: Karpathy and Fei-Fei
More Examples

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

\[
\frac{(N - F)}{s} + 1
\]

Output volume: \((32 - 5) / 1 + 1 = 28\), so: 28x28x5

Number of Parameters?

Slide credit: Karpathy and Fei-Fei
More Examples

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

\[
\frac{(N - F)}{s} + 1
\]

Output volume: \((32 - 5) / 1 + 1 = 28\), so: **28x28x5**

How many parameters? \(5x5x3x5 + 5 = 380\)

Slide credit: Karpathy and Fei-Fei
More Examples

Input volume: 32x32x3
Receptive fields: 5x5, stride 3
Number of neurons: 5

\[
\frac{(N - F)}{S} + 1
\]

Output volume size?

Slide credit: Karpathy and Fei-Fei
More Examples

Input volume: \textbf{32x32x3}
Receptive fields: \textbf{5x5, stride 3}
Number of neurons: \textbf{5}

Output volume: \((32 - 5) / 3 + 1 = 10\), so: \textbf{10x10x5}

\textbf{Number of Parameters?}

Slide credit: Karpathy and Fei-Fei
More Examples

Input volume: \(32 \times 32 \times 3\)
Receptive fields: \(5 \times 5\), stride 3
Number of neurons: 5

Output volume: \((32 - 5) / 3 + 1 = 10\), so: \(10 \times 10 \times 5\)
How many parameters? \(5 \times 5 \times 3 \times 5 + 5 = 380\). Same!

Slide credit: Karpathy and Fei-Fei
Thought Problem

- How do you write a normal neural network as a convnet?
Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel

Slide credit: Karpathy and Fei-Fei
Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel

Slide credit: Karpathy and Fei-Fei
Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel

<table>
<thead>
<tr>
<th></th>
<th>I11</th>
<th>I12</th>
<th>I13</th>
<th>I14</th>
<th>I15</th>
<th>I16</th>
<th>I17</th>
</tr>
</thead>
<tbody>
<tr>
<td>I21</td>
<td>I22</td>
<td>I23</td>
<td>I24</td>
<td>I25</td>
<td>I26</td>
<td>I27</td>
<td></td>
</tr>
<tr>
<td>I31</td>
<td>I32</td>
<td>I33</td>
<td>I34</td>
<td>I35</td>
<td>I36</td>
<td>I37</td>
<td></td>
</tr>
<tr>
<td>I41</td>
<td>I42</td>
<td>I43</td>
<td>I44</td>
<td>I45</td>
<td>I46</td>
<td>I47</td>
<td></td>
</tr>
<tr>
<td>I51</td>
<td>I52</td>
<td>I53</td>
<td>I54</td>
<td>I55</td>
<td>I56</td>
<td>I57</td>
<td></td>
</tr>
<tr>
<td>I61</td>
<td>I62</td>
<td>I63</td>
<td>I64</td>
<td>I65</td>
<td>I66</td>
<td>I67</td>
<td></td>
</tr>
<tr>
<td>I71</td>
<td>I72</td>
<td>I73</td>
<td>I74</td>
<td>I75</td>
<td>I76</td>
<td>I77</td>
<td></td>
</tr>
</tbody>
</table>

Max-pool
3x3 Filter
Stride 2

O11 = maximum value in blue box
Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel

Max-pool 3x3 Filter Stride 2

O12 = maximum value in blue box
Other Layers – Pooling

Idea: just want spatial resolution of activations/images smaller; applied per-channel

Max-pool
3x3 Filter
Stride 2

O13 = maximum value in blue box
Other Layers – Pooling

Idea: just want spatial resolution of activations / images smaller; applied per-channel
Squeezing a Loaf of Bread

Max-pool
2x2 Filter
Stride 2
Example Network

Suppose we want to convert a 32x32x3 image into a 10x1 vector of classification results.

Figure Credit: Karpathy and Fei-Fei; see http://cs231n.stanford.edu/
Example Network

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: \((3*3*3)*10 + 10 = 280\)
RELU

CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: \((3*3*10)*10 + 10 = 910\)
RELU

POOL with 2x2 filters, stride 2:
gives: [16x16x10]
parameters: 0

Slide credit: Karpathy and Fei-Fei
Example Network

Previous output: [16x16x10]
CONV with 10 3x3 filters, stride 1:
gives: [16x16x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
CONV with 10 3x3 filters, stride 1:
gives: [16x16x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
POOL with 2x2 filters, stride 2:
gives: [8x8x10]
parameters: 0

Slide credit: Karpathy and Fei-Fei
Example Network

Conv, Relu, Conv, Relu, Pool continues until it’s [4x4x10]

Fully-Connected FC layer to 10 neurons
(which are our class scores)
Number of parameters:
$10 \times 4 \times 4 \times 10 + 10 = 1610$

done!
An Alternate Conclusion

Conv, Relu, Conv, Relu, Pool continues until it’s [4x4x10]

Average POOL 4x4x10 to 10 neurons
Fully-Connected FC layer to 10 neurons
(which are our class scores)
Number of parameters:
10 * 10 + 10 = 110

done!

Slide credit: Karpathy and Fei-Fei
Example Network

Figure Credit: Zeiler and Fergus, Visualizing and Understanding Convolutional Networks. ECCV 2014
Example Network

(1) filter image with 96 7x7 filters
(2) ReLU
(3) 3x3 max pool with stride 2 (and contrast normalization – now ignored)

Figure Credit: Zeiler and Fergus, Visualizing and Understanding Convolutional Networks. ECCV 2014
What Do The Filters Represent?

Recall: filters are images and we can look at them
What Do The Filters Represent?

First layer filters of a network trained to distinguish 1000 categories of objects.

Remember these filters go over color.

For the interested: Gabor filter

Figure Credit: Karpathy and Fei-Fei
What Do The Filters Do?

Figure Credit: Karpathy and Fei-Fei; see http://cs231n.stanford.edu/