Intro to Deep Learning

Computer Vision  236873, Winter 2019
(slides from CS131, Stanford)
Today’s agenda

- Review of convolutions and classification
- Creating a convolution-based classifier
- Overview of machine learning
  - Neural networks
  - Gradient descent
  - Backprop
- Our classifier’s performance
Recall convolutions...

\[
f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l]
\]
Recall convolutions…

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l] \]
Recall convolutions...

\[
\begin{array}{ccc}
12 & 3 & 19 \\
25 & 10 & 1 \\
9 & 7 & 17 \\
\end{array}
\]

\[
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}
\]

\[
\begin{array}{cc}
133 & 75 \\
? & ? \\
\end{array}
\]

\[
f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l]
\]
Recall convolutions…

\[
f[n, m] * h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l]
\]

\[
\begin{array}{ccc}
12 & 3 & 19 \\
25 & 10 & 1 \\
9 & 7 & 17 \\
\end{array} * \begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array} = \begin{array}{cc}
133 & 75 \\
100 & ? \\
\end{array}
\]
Recall convolutions...

\[
f[n, m] \ast h[n, m] = \sum_{k=\infty}^{\infty} \sum_{l=\infty}^{\infty} f[k, l] h[n - k, m - l]
\]
Recall convolutions...

\[
f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] \ h[n - k, m - l]
\]
Recall convolutions...

\[
f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l]
\]
Recall convolutions...

\[ f[n, m] \ast h[n, m] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l] h[n - k, m - l] \]
Why they are useful

Allow us to find **interesting insights/features** from images!
Recall Image Classification…

Allow us to use features to put **images in categories**!
Wait a Minute…

Convolution = Image -> Features

Classification Algorithm = Features -> Category
Wait a Minute…

Convolution = Image -> Features

Classification Algorithm = Features -> Category

Let’s put ‘em together!
Let’s build a **convolution-based** classification algorithm for the CIFAR-10 dataset (10 classes, 32x32 images):
Feature Extractor

32x32 “Airplane Filter” =
Feature Extractor

32x32 “Airplane Filter” = “probability” of the image being an airplane
Feature Extractor

32x32 “Airplane Filter”

“probability” of the image being an airplane
Feature Extractor

This is not really a probability but a score, because it can be less than 0 and greater than 1

32x32 “Airplane Filter”
Feature Extractor

32x32 “Automobile Filter”

“probability” of the image being an automobile
Feature Extractor

32x32 “Bird Filter” = “probability” of the image being a bird
Feature Extractor

32x32 “Truck Filter”

“probability” of the image being a truck
$c_{\text{pred}} = \arg \max(f(...))$
We predict the class that has the highest probability!

\[ c_{pred} = \text{arg max}(\quad) \]
The Whole Shebang

Image → Feature Extractor → Prediction $\hat{y}$ → Classifier → $\text{argmax}$ → $c_{pred}$
The Whole Shebang

Image → Feature Extractor → Prediction $\hat{y}$ → Classifier → $\text{argmax}$ → $c_{\text{pred}}$
The Whole Shebang

Image → Feature Extractor → Prediction → Classifier → $c_{pred}$

Classification Output
Reframing convolution

\[
\begin{array}{cc}
12 & 21 \\
18 & 31 \\
\end{array} \quad \ast \quad \begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}
\]
Reframing convolution

\[
\begin{bmatrix}
12 & 21 \\
18 & 31
\end{bmatrix}
\ast
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
= 
\begin{bmatrix}
12 \\
21 \\
18 \\
31
\end{bmatrix}
\cdot
\begin{bmatrix}
1 \\
2 \\
3 \\
4
\end{bmatrix}
\]
Reframed Feature Extractor

* 32x32 “Airplane Filter”
Reframed Feature Extractor

32x32 “Airplane Filter”

= Image Vector

= Airplane Weight Vector
New Feature Extractor

Image Vector  Airplane Weight Vector

\[ \cdot \]

\[ = \]

“probability” of the image being an airplane
New Feature Extractor

Image Vector  Automobile Weight Vector

\[ \text{"probability" of the image being an automobile} \]

\[ = \]
New Feature Extractor

Image Vector  Bird Weight Vector

“probability” of the image being a bird
New Feature Extractor

Image Vector

Truck Weight Vector

“probability” of the image being a truck

=
New Feature Extractor

Weight Matrix

Image Vector

\[ \text{Weight Matrix} \times \text{Image Vector} = \]
New Feature Extractor

\[ W x = \hat{y} \]

\( W \): the \((10 \times 1024)\) matrix of weight vectors

\( x \): the \((1024 \times 1)\) image vector

\( \hat{y} \): the \((10 \times 1)\) vector of class “probabilities”
New Feature Extractor

This simple computation is called a fully-connected layer!

\[ Wx = \hat{y} \]

\( W \): the (10x1024) matrix of weight vectors

\( x \): the (1024x1) image vector

\( \hat{y} \): the (10x1) vector of class “probabilities”
Aside: Fully-Connected Neural Networks

The diagram illustrates a neural network with inputs for Height, Width, and Num_Legs, which are fed into an algorithm. The output of the algorithm is represented by $\hat{y}$, indicating whether the input is a Dog or Not Dog.
Aside: Fully-Connected Neural Networks

\[ x \rightarrow \hat{y} \]

- **Height**
- **Width**
- **Num_Legs**

\( \gamma \)
Aside: Fully-Connected Neural Networks

Height
Width
Num_Legs

\[ x \]

\[ \hat{y} \]

\[ w_1 \]
\[ w_2 \]
\[ w_3 \]
\[ w_4 \]
\[ w_5 \]
\[ w_6 \]
Aside: Fully-Connected Neural Networks

\[ \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \end{bmatrix} \cdot \begin{array}{c} W \\ x \end{array} = \hat{y} \]

\[ Wx = \hat{y} \]
Aside: Fully-Connected Neural Networks

”Fully-Connected”

Every node is connected to every other node

”Neural Network”

Kinda looks like a neuron!
New Feature Extractor

\[ Wx = \hat{y} \]

\[ W: \text{ the (10x1024) matrix of weight vectors} \]

\[ x: \text{ the (1024x1) image vector} \]

\[ \hat{y}: \text{ the (10x1) vector of class “probabilities”} \]
New Feature Extractor

\[ Wx = \hat{y} \]

\( W \): the \((10 \times 1024)\) matrix of weight vectors

\( x \): the \((1024 \times 1)\) image vector

\( \hat{y} \): the \((10 \times 1)\) vector of class “probabilities”?
Class Probability Vector

- Must have values between 0 and 1
- Must sum to 1
- There’s no guarantee either requirement is satisfied!

\[ \hat{y} = Wx \]
Softmax Function

\[ a_{(x)}_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \]
Softmax Function

\[
\text{Softmax: } a(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}}
\]

Diagram:
- Input vector: \( a = [1, -3] \)
- Softmax output: \( SM(a) = [0.98, 0.02] \)
Class Probability Vector

- Must have values between 0 and 1
- Must sum to 1

\[ \hat{y} = Wx \]
Class Probability Vector

- Must have values between 0 and 1
- Must sum to 1

\[ \hat{y} = SM(Wx) \]
System so far...

- Feature extractor:

\[
\hat{y} = SM(Wx)
\]

- Classifier:

\[
c_{pred} = \text{arg max}(\hat{y})
\]
System so far...

- Feature extractor:

\[ \hat{y} = SM(Wx) \]

- Classifier:

\[ c_{pred} = \arg \max (\hat{y}) \]
System so far...

- Feature extractor:

- Classifier:

\[ c_{pred} = \text{arg max} (y) \]

\[ y = SM(Wx) \]
Using the label

Let’s compare our prediction with the real answer! For each image, we have the label $\gamma$ which tells us the true class:

$\gamma$

$\begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
1 \\
0 \\
0 \\
0 \\
0 \\
0
\end{bmatrix}$

Dog class index
Key Insight:

We want:

$$\arg\max(\hat{y}) = \arg\max(y)$$
Key Insight:

We want:

$$\arg\max(\hat{y}) = \arg\max(y)$$

Which we can accomplish by:

$$W^* = \arg\min_W \left( -\sum_{x,y} \log(p_c) \right)$$
Key Insight:

We want:

\[ \text{arg max}(\hat{y}) = \text{arg max}(y) \]

Which we can accomplish by:

\[ W^* = \text{arg min}_W \left( - \sum_{x,y} \log(p_c) \right) \]

Where \( p_c \) is the probability of the true class in \( \hat{y} \).
Cross-Entropy Loss

Our loss function represents *how bad we are currently doing*:

\[ L = -\log(p_c) \]
Cross-Entropy Loss

Our loss function represents *how bad we are currently doing*:

\[ L = -\log(p_c) \]

Examples:

\[ p_c = 0 \rightarrow L = -\log(0) = \infty \]
\[ p_c = 0.1 \rightarrow L = -\log(0.1) = 2.3 \]
\[ p_c = 0.9 \rightarrow L = -\log(0.9) = 0.1 \]
\[ p_c = 1 \rightarrow L = -\log(1) = 0 \]
Cross-Entropy Loss

Our loss function represents *how bad we are currently doing*:

\[ L = -\log(p_c) \]

Examples:

- \( p_c = 0 \rightarrow L = -\log(0) = \infty \)
- \( p_c = 0.1 \rightarrow L = -\log(0.1) = 2.3 \)
- \( p_c = 0.9 \rightarrow L = -\log(0.9) = 0.1 \)
- \( p_c = 1 \rightarrow L = -\log(1) = 0 \)

The larger the loss, the worse our prediction. We want to minimize L!
Minimizing Loss
Minimizing Loss
Minimizing Loss
Minimizing Loss
Minimizing Loss
Minimizing Loss
Minimizing Loss
Minimizing Loss
Minimizing Loss

\[ L \]

\[ w \]

GRADIENT DESCENT!
Gradient Descent Pseudocode

for i in {0,...,num_epochs}:
    for x, y in data:
        \( \hat{y} = SM(Wx) \)
        \( L = CE(\hat{y}, y) \)
        \( \frac{dL}{dW} = ??? \)
        \( W := W - \alpha \frac{dL}{dW} \)
Getting the Gradient

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = \frac{dL}{dz} \frac{dz}{dW} \]
Getting the Gradient

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = \frac{dL}{dz}(x) \]
Getting the Gradient

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = (SM(z) - y)(x) \]
Getting the Gradient

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = (SM(z) - y)(x^T) \]
Getting the Gradient

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = (SM(z) - y)^T \]
What is Backprop?

\[ z = Wx \]
\[ L = \text{SCE}(z, y) \]
\[ \frac{dL}{dW} = \frac{dL}{dz} \frac{dz}{dW} \]
What is Backprop?

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = \frac{dL}{dz} \frac{dz}{dW} \]
What is Backprop?

\[ z = W x \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = \frac{dL}{dz} \frac{dz}{dW} \]
What is Backprop?

When computations are treated as nodes, all derivatives depend only on inputs to that node.

\[ z = Wx \]
\[ L = SCE(z, y) \]

\[ \frac{dL}{dW} = \frac{dL}{dz} \frac{dz}{dW} \]
What is Backprop?

When computations are treated as nodes, all derivatives depend only on inputs to that node.

So, we can cache the initial computation and reuse!
class FullyConnected:
    def __init__(self):
        self.cache = {}
    
    def forward(self, W, x):
        self.cache['x'] = x
        return np.dot(W, x)
    
    def backward(self, dout):
        x = self.cache['x']
        return np.matmul(dout, x.T)

class SCELoss:
    def __init__(self):
        self.cache = {}
    
    def forward(self, z, y):
        self.cache['z'] = z
        self.cache['y'] = y
        return sce(z, y)
    
    def backward(self):
        z = self.cache['z']
        y = self.cache['y']
        return sce(z, y) - y
Gradient Descent Pseudocode (Updated)

for i in {0,...,num_epochs}:
    for x, y in data:
        \( \hat{y} = SM(Wx) \)
        \( L = CE(\hat{y}, y) \)
        \( \frac{dL}{dW} = \ldots \)
        \( W := W - \alpha \frac{dL}{dW} \)
Gradient Descent Pseudocode (Updated)

for i in {0,...,num_epochs}:
    for x, y in data:
        \( \hat{y} = SM(Wx) \)
        \( L = CE(\hat{y}, y) \)
        \( \frac{dL}{dW} = \text{backprop}(L) \)
        \( W := W - \alpha \frac{dL}{dW} \)
Gradient Descent Pseudocode (Updated)

for i in \{0, \ldots, \text{num\_epochs}\}:
    for x, y in data:
        \( \hat{y} = SM(Wx) \)
        \( L = CE(\hat{y}, y) \)
        \( \frac{dL}{dw} = \text{backprop}(L) \)
        \( W := W - \alpha \frac{dL}{dw} \)
Our Classification System

Image 32x32x10 Conv Block → Prediction \( \hat{\gamma} \) → argmax → \( c_{\text{pred}} \)

<table>
<thead>
<tr>
<th>Image</th>
<th>Feature Extractor</th>
<th>Prediction ( \hat{\gamma} )</th>
<th>Classifier</th>
<th>Classification Output</th>
</tr>
</thead>
</table>
Our Classification System (modified)

Input Image

Feature Extractor

Prediction $\hat{y}$

Classifier

Classification Output

Input Label

$\gamma$

$CE$

$L$

$argmax$

$c_{pred}$

Loss Function

Loss Value
Our Classification System (modified)

Input Image ➔ Feature Extractor ➔ Prediction $\hat{y}$ ➔ $\text{argmax}$ ➔ $c_{pred}$

Input Label ➔ $\gamma$ ➔ $CE$ ➔ $L$ ➔ Loss Value

1) Minimize this...
Our Classification System (modified)

Input Image

Feature Extractor

Prediction $\hat{y}$

Classifier

Classification Output

2) By modifying this...

$\gamma$

1) Minimize this...

$CE$

Loss Function

$L$

Loss Value
Our Classification System (modified)

1) Minimize this...

2) By modifying this...

3) Using gradient descent!
Our Classification System (modified)

Input Image  →  Feature Extractor  →  Classifier  →  Classification Output

Loss Function: \( CE \) → Loss Value: \( L \)

\( \gamma \) By modifying this...

1) Minimize this...

3) Using gradient descent!
Our System’s Performance

- ~40% accuracy on CIFAR-10 test
  - Best class: Truck (~60%)
  - Worst class: Horse (~16%)

- Check out the model at: https://tinyurl.com/cifar10

- What about the filters? What do they look like?
Visualizing the Filters

Airplane  Automobile  Bird  Cat  Deer

Dog  Frog  Horse  Ship  Truck