CS5670: Computer Vision
Noah Snavely

Image Classification

Slides from Fei-Fei Li, Justin Johnson, Serena Yeung
http://vision.stanford.edu/teaching/cs231n/
Announcements

• Project 4 due today
• Project 5 to be released soon
• Quiz 4 this Wednesday (4/24) (1\textsuperscript{st} 10 minutes)
  – Will cover material since last quiz (photometric stereo, multi-view stereo, structure from motion, image classification (today’s lecture))
• Guest lecture next Monday, 4/29, Jin Sun
  – Generative Adversarial Networks (GANs)
• Wednesday’s lecture: Convolutional Neural Networks
Today

• Image classification pipeline
• Training, validation, testing
• Nearest neighbor classification
• Linear classification

• Building up to CNNs for learning
  – Next four lectures on deep learning
References

• Stanford CS231N
  – http://cs231n.stanford.edu/
What matters in recognition?

• Machine learning methods (e.g., linear classification, deep learning)
• Representation (e.g., SIFT, HoG, deep learned features)
• **Data** (e.g., PASCAL, ImageNet, COCO)
Image Classification:
A core task in Computer Vision

• Assume given set of discrete labels, e.g. 
  \{cat, dog, cow, apple, tomato, truck, ... \}

\[
f(\text{apple}) = \text{"apple"}
\]
\[
f(\text{tomato}) = \text{"tomato"}
\]
\[
f(\text{cow}) = \text{"cow"}
\]
Image classification demo

https://cloud.google.com/vision/

See also:
https://aws.amazon.com/rekognition/
https://www.clarifai.com/
https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/
...
Image Classification

(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

→ cat
Image Classification: Problem

What the computer sees

82% cat
15% dog
2% hat
1% mug
Recall from last time: Challenges of recognition

- Viewpoint
- Illumination
- Deformation
- Occlusion

- Clutter
- Intraclass Variation
An image classifier

```python
def classify_image(image):
    # Some magic here?
    return class_label
```

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for recognizing a cat, or other classes.
Data-driven approach

• Collect a database of images with labels
• Use ML to train an image classifier
• Evaluate the classifier on test images

Example training set
Data-driven approach

- Collect a database of images with labels
- Use ML to train an image classifier
- Evaluate the classifier on test images

```python
def train(train_images, train_labels):
    # build a model of images -> labels

def predict(image):
    # evaluate the model on the image
    return class_label
```
Training

Training Images

Testing
Test Image

Image Features

Training

Training Labels

Learned Classifier

Prediction

Dataset: ETH-80, by B. Leibe
Slide credit: D. Hoiem, L. Lazebnik
Classifiers

- Nearest Neighbor
- kNN (“k-Nearest Neighbors”)
- Linear Classifier
- Neural Network
- Deep Neural Network
- ...

First: Nearest Neighbor (NN) Classifier

• **Train**
  – Remember all training images and their labels

• **Predict**
  – Find the closest (most similar) training image
  – Predict its label as the true label
CIFAR-10 and NN results

Example dataset: CIFAR-10
10 labels
50,000 training images, each image is tiny: 32x32
10,000 test images.
CIFAR-10 and NN results

Example dataset: CIFAR-10
10 labels
50,000 training images
10,000 test images.

For every test image (first column), examples of nearest neighbors in rows.

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/
k-nearest neighbor

- Find the k closest points from training data
- Take **majority vote** from K closest points
What does this look like?
What does this look like?
How to find the most similar training image? What is the distance metric?

**L1 distance:**

$$d_1(I_1, I_2) = \sum_{p} |I_1^p - I_2^p|$$

Where $I_1$ denotes image 1, and $p$ denotes each pixel.

<table>
<thead>
<tr>
<th>test image</th>
<th>training image</th>
<th>pixel-wise absolute value differences</th>
</tr>
</thead>
<tbody>
<tr>
<td>56 32 10 18</td>
<td>10 20 24 17</td>
<td>46 12 14 1</td>
</tr>
<tr>
<td>90 23 128 133</td>
<td>8 10 89 100</td>
<td>82 13 39 33</td>
</tr>
<tr>
<td>24 26 178 200</td>
<td>12 16 178 170</td>
<td>12 10 0 30</td>
</tr>
<tr>
<td>2 0 255 220</td>
<td>4 32 233 112</td>
<td>2 32 22 108</td>
</tr>
</tbody>
</table>

$\rightarrow$ 456
Choice of distance metric

• Hyperparameter

L1 (Manhattan) distance

\[ d_1(I_1, I_2) = \sum_p |I^p_1 - I^p_2| \]

L2 (Euclidean) distance

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I^p_1 - I^p_2)^2} \]

Slides from Andrej Karpathy and Fei-Fei Li
http://vision.stanford.edu/teaching/cs231n/
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

\[ d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \]

L2 (Euclidean) distance

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \]

K = 1

Demo: http://vision.stanford.edu/teaching/cs231n-demos/knn/
Hyperparameters

• What is the **best distance** to use?
• What is the **best value of k** to use?

• These are **hyperparameters**: choices about the algorithm that we set rather than learn

• How do we set them?
  – One option: try them all and see what works best
Setting Hyperparameters

Idea #1: Choose hyperparameters that work best on the data

Your Dataset
Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

**BAD:** $K = 1$ always works perfectly on training data

Your Dataset
Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

**BAD:** $K = 1$ always works perfectly on training data

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

| train | test |
Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

BAD: $K = 1$ always works perfectly on training data

---

Your Dataset

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

---

train | test
Setting Hyperparameters

**Idea #1:** Choose hyperparameters that work best on the data

BAD: $K = 1$ always works perfectly on training data

---

**Idea #2:** Split data into **train** and **test**, choose hyperparameters that work best on test data

BAD: No idea how algorithm will perform on new data

---

**Idea #3:** Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

Better!
Setting Hyperparameters

Idea #4: Cross-Validation: Split data into folds, try each fold as validation and average the results

Useful for small datasets, but not used too frequently in deep learning
Example of 5-fold cross-validation for the value of $k$.

Each point: single outcome.

The line goes through the mean, bars indicated standard deviation

(Seems that $k \sim 7$ works best for this data)
Recap: How to pick hyperparameters?

- Methodology
  - Train and test
  - Train, validate, test

- Train for original model
- Validate to find hyperparameters
- Test to understand generalizability
kNN -- Complexity and Storage

• N training images, M test images

• Training: $O(1)$
• Testing: $O(MN)$

• Hmm...
  – Normally need the opposite
  – Slow training (ok), fast testing (necessary)
k-Nearest Neighbor on images *never used*.

- terrible performance at test time
- distance metrics on level of whole images can be very unintuitive

(all 3 images have same L2 distance to the one on the left)
k-Nearest Neighbors: Summary

• In **image classification** we start with a **training set** of images and labels, and must predict labels on the **test set**

• The **K-Nearest Neighbors** classifier predicts labels based on nearest training examples

• Distance metric and K are **hyperparameters**

• Choose hyperparameters using the **validation set**; only run on the test set once at the very end!
Linear classifiers
Score function

class scores
Score function: $f$

Parametric approach

$\text{image} \rightarrow f(x, W) \rightarrow \text{parameters} \rightarrow 10 \text{ numbers, indicating class scores}$

$[32 \times 32 \times 3]$ array of numbers $0...1$
(3072 numbers total)
Parametric approach: Linear classifier

\[ f(x, W) = Wx \]

[32x32x3] array of numbers 0...1

parameters, or “weights”

10 numbers, indicating class scores

3072x1
Parametric approach: Linear classifier

\[ f(x, W) = Wx \]

[32x32x3] array of numbers 0...1

parameters, or “weights”

10 numbers, indicating class scores
Linear Classifier

Define a score function

\[ f(x_i, W, b) = Wx_i + b \]

data (image)

class scores

“weights”

“bias vector”

“parameters”
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)
Interpretation: Template matching

\[ f(x_i, W, b) = Wx_i + b \]
Geometric Interpretation

\[ f(x_i, W, b) = Wx_i + b \]
Linear classifiers

- Find linear function (hyperplane) to separate positive and negative examples

\[ x_i \text{ positive: } x_i \cdot w + b \geq 0 \]
\[ x_i \text{ negative: } x_i \cdot w + b < 0 \]

Which hyperplane is best? We will come back to this later
Hard cases for a linear classifier

**Class 1:**
First and third quadrants

**Class 2:**
Second and fourth quadrants

**Class 1:**
$1 \leq \text{L2 norm} \leq 2$

**Class 2:**
Everything else

**Class 1:**
Three modes

**Class 2:**
Everything else
Linear Classifier: Three Viewpoints

**Algebraic Viewpoint**

\[ f(x, W) = Wx \]

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space
So far: Defined a (linear) score function $f(x, W) = Wx + b$

Example class scores for 3 images for some $W$:

How can we tell whether this $W$ is good or bad?

<table>
<thead>
<tr>
<th>Class</th>
<th>Score 1</th>
<th>Score 2</th>
<th>Score 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
<td>3.42</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
<td>6.04</td>
<td>4.64</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
<td>2.65</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
<td>-4.22</td>
<td>5.1</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
<td>2.64</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
<td>3.58</td>
<td>5.55</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
<td>-4.34</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
<td>-4.37</td>
<td>-1.5</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
<td>-4.79</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
<td>-2.93</td>
<td>6.14</td>
</tr>
</tbody>
</table>
Recap

• Learning methods
  – k-Nearest Neighbors
  – Linear classification

• Classifier outputs a score function giving a score to each class

• Today: how do we define how good a classifier is based on the training data? (Spoiler: define a loss function)
Linear classification

Output scores

**TODO:**

1. Define a loss function that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. (optimization)
Suppose: 3 training examples, 3 classes.
With some $W$ the scores $f(x, W) = Wx$ are:

A **loss function** tells how good our current classifier is.

Given a dataset of examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

Where $x_i$ is image and $y_i$ is (integer) label

Loss over the dataset is a sum of loss over examples:

$$L = \frac{1}{N} \sum_{i} L_i(f(x_i, W), y_i)$$
Loss function, cost/objective function

• Given ground truth labels \((y_i)\), scores \(f(x_i, W)\)
  — how unhappy are we with the scores?

• Loss function or objective/cost function measures unhappiness

• During training, \textbf{want to find the parameters }W\textbf{ that minimizes the loss function}
Simpler example: binary classification

- Two classes (e.g., “cat” and “not cat”)
  - AKA “positive” and “negative” classes
Linear classifiers

• Find linear function (hyperplane) to separate positive and negative examples

\[ x_i \text{ positive}: \quad x_i \cdot w + b \geq 0 \]
\[ x_i \text{ negative}: \quad x_i \cdot w + b < 0 \]

Which hyperplane is best?
We need a **loss function** to decide
What is a good loss function?

- One possibility
  - Number of misclassified examples
    - Problems: discrete, can’t break ties
    - We want the loss to lead to *good generalization*
    - We want the loss to work for more than 2 classes
Softmax classifier

\[ f(x_i, W) = W x_i \]  \hspace{1cm} \text{(score function)}

softmax function

Example with three classes:

\[ [1, -2, 0] \rightarrow [e^1, e^{-2}, e^0] = [2.71, 0.14, 1] \rightarrow [0.7, 0.04, 0.26] \]

Interpretation: squashes values into \textit{probabilities} ranging from 0 to 1
Cross-entropy loss

\[ f(x_i, W) = W x_i \]  (score function)
Losses

• Cross-entropy loss is just one possible loss function
  – One nice property is that it reinterprets scores as probabilities, which have a natural meaning

• SVM (max-margin) loss functions also used to be popular
  – But currently, cross-entropy is the most common classification loss
Summary

• Have score function and loss function
  – Currently, score function is based on linear classifier
  – Next, will generalize to convolutional neural networks

• Find $W$ and $b$ to minimize loss

\[ L = \frac{1}{N} \sum_i -\log \left( \frac{e^{f_{yi}}}{\sum_j e^{f_j}} \right) + \lambda \sum_k \sum_l W_{k,l}^2 \]