Lecture 2:
Image Classification pipeline
Administrative: Piazza

For questions about midterm, poster session, projects, etc, use Piazza!

SCPD students: Use your @stanford.edu address to register for Piazza; contact scpd-customerservice@stanford.edu for help.
Administrative: Assignment 1

Out yesterday, due 4/17 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax
- Two-layer neural network
- Image features
Administrative: Friday Discussion Sections

(Some) Fridays 12:30pm - 1:20pm in Gates B03

Hands-on tutorials, with more practical detail than main lecture

We may not have discussion sections every Friday, check syllabus: http://cs231n.stanford.edu/syllabus.html

This Friday: Python / numpy / Google Cloud setup
Administrative: Course Project

Project proposal due 4/24
Python Numpy Tutorial

This tutorial was contributed by Justin Johnson.

We will use the Python programming language for all assignments in this course. Python is a great general-purpose programming language on its own, but with the help of a few popular libraries (numpy, scipy, matplotlib) it becomes a powerful environment for scientific computing.

We expect that many of you will have some experience with Python and numpy; for the rest of you, this section will serve as a quick crash course both on the Python programming language and on the use of Python for scientific computing.

http://cs231n.github.io/python-numpy-tutorial/
Administrative: Google Cloud

We will be using Google Cloud in this class.

We will be distributing coupons to all enrolled students.

See our tutorial here for walking through Google Cloud setup: 
https://github.com/cs231n/gcloud
Image Classification: A core task in Computer Vision

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

---

This image by Nikita is licensed under CC-BY 2.0
The Problem: Semantic Gap

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3

(3 channels RGB)
Challenges: Viewpoint variation

All pixels change when the camera moves!
Challenges: Background Clutter
Challenges: Illumination
Challenges: Deformation
Challenges: Occlusion
Challenges: 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.
Attempts have been made

Find edges

Find corners

John Canny, “A Computational Approach to Edge Detection”, IEEE TPAMI 1986
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning to train a classifier
3. Evaluate the classifier on new images

Example training set

```python
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```
First classifier: **Nearest Neighbor**

```python
def train(images, labels):
    # Machine learning!
    return model

def predict(model, test_images):
    # Use model to predict labels
    return test_labels
```

- Memorize all data and labels
- Predict the label of the most similar training image
Example Dataset: **CIFAR10**

- 10 classes
- 50,000 training images
- 10,000 testing images

Example Dataset: **CIFAR10**

- **10 classes**
- **50,000** training images
- **10,000** testing images

Test images and nearest neighbors

Distance Metric to compare images

L1 distance: \[ d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p| \]

<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>

add 456
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """ X is N x D where each row is an example. Y is 1-dimensional of size N ""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """ X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example
        return Ypred
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """X is N x D where each row is an example. Y is 1-dimension of size N ""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """X is N x D where each row is an example we wish to predict label for ""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in range(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred

Nearest Neighbor classifier

Memorize training data
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        # X is N x D where each row is an example. Y is 1-dimension of size N
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        # X is N x D where each row is an example we wish to predict label for
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i'th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
            min_index = np.argmin(distances) # get the index with smallest distance
            Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

        return Ypred
Nearest Neighbor classifier

Q: With N examples, how fast are training and prediction?

```python
import numpy as np

class NearestNeighbor:
    def __init__(self):
        pass

    def train(self, X, y):
        """X is N x D where each row is an example. Y is 1-dimension of size N \"""
        # the nearest neighbor classifier simply remembers all the training data
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        """X is N x D where each row is an example we wish to predict label for \"""
        num_test = X.shape[0]
        # lets make sure that the output type matches the input type
        Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

        # loop over all test rows
        for i in xrange(num_test):
            # find the nearest training image to the i' th test image
            # using the L1 distance (sum of absolute value differences)
            distances = np.sum(np.abs(self.Xtr - X[i, :]), axis = 1)
            min_index = np.argmin(distances)  # get the index with smallest distance
            Ypred[i] = self.ytr[min_index]  # predict the label of the nearest example

        return Ypred
```
Nearest Neighbor classifier

**Q:** With N examples, how fast are training and prediction?

**A:** Train $O(1)$, predict $O(N)$
Nearest Neighbor classifier

Q: With N examples, how fast are training and prediction?

A: Train O(1), predict O(N)

This is bad: we want classifiers that are fast at prediction; slow for training is ok.
Nearest Neighbor classifier

Many methods exist for fast / approximate nearest neighbor (beyond the scope of 231N!)

A good implementation:
https://github.com/facebookresearch/faiss

Johnson et al, “Billion-scale similarity search with GPUs”, arXiv 2017
What does this look like?
K-Nearest Neighbors

Instead of copying label from nearest neighbor, take **majority vote** from K closest points

K = 1

K = 3

K = 5
What does this look like?
What does this look like?
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-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-Nearest Neighbors: Demo Time

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

What is the best value of $k$ to use?
What is the best distance to use?

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

What is the best value of $k$ to use?
What is the best distance to use?

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

Very problem-dependent.
Must try them all out and see what works best.
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

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

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

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

**Your Dataset**

- train
- test

**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
## Setting Hyperparameters

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

BAD: K = 1 always works perfectly on training data

<table>
<thead>
<tr>
<th>Your Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>train</td>
</tr>
<tr>
<td>test</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>train</th>
<th>validation</th>
<th>test</th>
</tr>
</thead>
</table>

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

Better!

| train | validation | test |
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
Setting Hyperparameters

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 $k \approx 7$ works best for this data)
k-Nearest Neighbor on images never used.

- Very slow at test time
- Distance metrics on pixels are not informative

(all 3 images have same L2 distance to the one on the left)
k-Nearest Neighbor on images never used.

- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = $4^2$

Dimensions = 3
Points = $4^3$
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 Classification
Man in black shirt is playing guitar.

Construction worker in orange safety vest is working on road.

Two young girls are playing with lego toy.

Boy is doing backflip on wakeboard.
Recall CIFAR10

- 50,000 training images
- 10,000 test images
- Each image is 32x32x3
Parametric Approach

Image

Array of $32 \times 32 \times 3$ numbers
(3072 numbers total)

$\mathbf{W}$

parameters or weights

$f(\mathbf{x}, \mathbf{W})$ → 10 numbers giving class scores

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 2 - 53

April 4, 2019
Parametric Approach: Linear Classifier

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

Image

Array of 32x32x3 numbers (3072 numbers total)

10 numbers giving class scores

\[ W \]

parameters or weights
Parametric Approach: Linear Classifier

Image parameters or weights $W$

$\mathbf{f}(\mathbf{x}, W) = \mathbf{W} \mathbf{x}$

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

10x1 $\mathbf{x}$

10x3072 $\mathbf{W}$

3072x1

$\mathbf{f}(\mathbf{x}, \mathbf{W})$ → 10 numbers giving class scores
Parametric Approach: Linear Classifier

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

Array of $32 \times 32 \times 3$ numbers (3072 numbers total)

Image

W

parameters or weights

10 numbers giving class scores
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

Stretch pixels into column

<table>
<thead>
<tr>
<th>56</th>
<th>231</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>231</td>
</tr>
<tr>
<td>24</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Stretch pixels into column

Input image

$$W = \begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}$$

$$b = \begin{bmatrix} 56 \\ 231 \\ 24 \end{bmatrix}$$

$$\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}$$

$$\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$$

Cat score
Dog score
Ship score
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

**Algebraic Viewpoint**

\[ f(x, W) = Wx \]
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Algebraic Viewpoint

\[ f(x, W) = Wx \]
Interpreting a Linear Classifier

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Input image

W
0.2
1.5
0.1
0.1
2.1
2.0
0.2
1.3
1.0

b
1.1
3.2
1.1

Score
-96.8
437.9
61.95
0.0
-0.3
Interpreting a Linear Classifier: Visual Viewpoint

airplane  automobile
bird  cat  deer  dog  frog  horse  ship  truck

Input image

W
0.2 -0.5
0.1  2.0
1.5  1.3
2.1  0.0
0  0.2
-0.3

b
1.1  3.2

Score
-96.8
437.9
61.95
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of **32x32x3** numbers (3072 numbers total)
Hard cases for a linear classifier

Class 1:
First and third quadrants

Class 2:
Second and fourth quadrants

Class 1:
1 <= L2 norm <= 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</th>
</tr>
</thead>
<tbody>
<tr>
<td>airplane</td>
<td>-3.45</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
</tr>
</tbody>
</table>
Coming up:
- Loss function (quantifying what it means to have a “good” W)
- Optimization (start with random W and find a W that minimizes the loss)
- ConvNets! (tweak the functional form of f)

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