Topics:
- Image Classification
- Supervised Learning view
- K-NN

Dhruv Batra
Georgia Tech
Administrativia

• PS0
  – Due: Aug 20 11:59pm

• More seats
  – We were able to recruit 1 more TA
  – 25 more seats added to 7643

• Piazza
  – 117/~200 people signed up. Please use that for questions.

• Office hours start next week

• Gradescope/Canvas
  – Anybody not have access?
  – Please post on Piazza
What is the collaboration policy?

• Collaboration
  – Only on HWs and project (not allowed in HW0).
  – You may discuss the questions
  – Each student writes their own answers
  – Write on your homework anyone with whom you collaborate
  – Each student must write their own code for the programming part

• Zero tolerance on plagiarism
  – Neither ethical nor in your best interest
  – Always credit your sources
  – Don’t cheat. We will find out.
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/
Plan for Today

• Image Classification
• Supervised Learning view
• K-NN

• Next time: Linear Classifiers
Image Classification
Image Classification: A core task in Computer Vision

(assume given set of discrete labels)

\{dog, cat, truck, plane, \ldots\}

cat
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: Illumination
Challenges: Deformation
Challenges: Occlusion
Challenges: Background Clutter
An image classifier

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
ML: A 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
```

airplane
automobile
bird
cat
deer
Notation

Scalars: \( x, y, z \in \mathbb{R}^l \)

Vectors: \( \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \)

Matrices: \( \mathbf{X}, \mathbf{Y} \)

\( \mathbb{R}^k, \mathbb{S} \)

Input Dimension: \( d \in \mathbb{R}^d \)

Output/\# classes: \( k \in \mathbb{R}^k \)

\# samples: \( n, N \)

Parameters: \( \mathbf{w}, \Theta \)
Supervised Learning

- **Input:** $x$ (images, text, emails…)
- **Output:** $y$ (cats vs dogs, spam vs not…)
- **(Unknown) Target Function**
  - $f : X \rightarrow Y$ (the “true” mapping / reality)
- **Data**
  - $\{ (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \}$

Goal: find $f$

Predict $f(\bar{x})$ at new $\bar{x}$
Supervised Learning

1. Model Class / Hypothesis Set
   \[ H = \{ h : X \rightarrow \mathbb{R} \} \quad \hat{y} = h(x) \]

2. Loss Function
   \[ \text{Loss}(h, D) = \frac{1}{N} \sum_{i=1}^{N} \text{Loss}(h(x_i), y_i) \]

3. Learning = Search / Opt in MC.
   \[ h^* = \text{arg min}_{h \in H} \{ \text{Loss}(h, D) \} \]
Supervised Learning

• Input: \( x \) (images, text, emails ...)

• Output: \( y \) (spam or non-spam ...)

• (Unknown) Target Function
  \(- f: X \rightarrow Y \) (the “true” mapping / reality)

• Data
  \(- \{ (x_1, y_1), (x_2, y_2), ..., (x_N, y_N) \} \)

• Model / Hypothesis Class
  \(- H = \{ h: X \rightarrow Y \} \)
  \(- e.g. \ y = h(x) = \text{sign}(w^T x) \)

• Learning = Search in hypothesis space
  \(- \text{Find best } h \text{ in model class.} \)
Learning is hard!

\[ \frac{2^d - n}{2} \leq \#fns \text{ compatible with dataset} \]

A Learning Problem

\[ H = \{ h: x \to y \} \]

\[ \#fns \]

\# boolean \( x \)s

1 boolean output

\[ 2^d = |H| \]

Example

<table>
<thead>
<tr>
<th>Example</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y = f(x_1, x_2, x_3, x_4) \]

\[ f = x_3 \land (\overline{x_2}) \]

\[ x_1 \ x_2 \ x_3 \ x_4 \ | \ y \]

\[ 0 \ 1 \ 1 \ 1 \ | \ \gamma \]

(C) Dhruv Batra
Learning is hard!

- No assumptions = No learning

A Learning Problem

<table>
<thead>
<tr>
<th>Example</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( x_4 )</th>
<th>( y )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Procedural View

• Training Stage:
  – Training Data \( \{ (x_i, y_i) \} \xrightarrow{\text{h}} h \) (Learning)

• Testing Stage
  – Test Data \( x \xrightarrow{} h(x) \) (Apply function, Evaluate error)
Statistical Estimation View

• Probabilities to rescue:
  – X and Y are random variables
  – \( D = (x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N) \sim P(X, Y) \)

• IID: Independent Identically Distributed
  – Both training & testing data sampled IID from \( P(X, Y) \)
  – Learn on training set
  – Have some hope of generalizing to test set
Classical Learning Theory: Error Decomposition
Classical Learning Theory: Error Decomposition

- **Model Class**
- **Estimation Error**
- **Optimization Error**
- **Modeling Error**

- **AlexNet**
  - Softmax
  - FC 1000
  - FC 4096
  - FC 4096
  - Pool
  - 3x3 conv, 256
  - 3x3 conv, 384
  - Pool
  - 3x3 conv, 384
  - Pool
  - 5x5 conv, 256
  - 11x11 conv, 96
  - Input

- **Reality**

(C) Dhruv Batra
Classical Learning Theory: Error Decomposition
Classical Learning Theory: Error Decomposition

VGG19

model class

Optimization Error

Estimation Error

Modeling Error

Reality

(C) Dhruv Batra
Error Decomposition

- Approximation/Modeling Error
  - You approximated reality with model

- Estimation Error
  - You tried to learn model with finite data

- Optimization Error
  - You were lazy and couldn’t/didn’t optimize to completion

- Bayes Error
  - Reality just sucks
Caveats

• A number of recent empirical results question our intuitions built from this clean separation.
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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example Dataset: **CIFAR10**

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


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example Dataset: CIFAR10

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

Test images and nearest neighbors


Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Nearest Neighbours
Instance/Memory-based Learning

Four things make a memory based learner:

• A distance metric

• How many nearby neighbors to look at?

• A weighting function (optional)

• How to fit with the local points?
1-Nearest Neighbour

Four things make a memory based learner:

• **A distance metric**
  – Euclidean (and others)

• **How many nearby neighbors to look at?**
  – 1

• **A weighting function (optional)**
  – unused

• **How to fit with the local points?**
  – Just predict the same output as the nearest neighbour.
k-Nearest Neighbour

Four things make a memory based learner:

• **A distance metric**
  - Euclidean (and others)

• *How many nearby neighbors to look at?*
  - k

• A weighting function (optional)
  - unused

• *How to fit with the local points?*
  - Just predict the average output among the nearest neighbours.
1-NN for Regression

Here, this is the closest datapoint.

Here, this is the closest datapoint.

Here, this is the closest datapoint.

Figure Credit: Carlos Guestrin
Distance Metric to compare images

**L1 distance:**

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

<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>

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2} \]
```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
```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-dimensional of size N """
        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

Memorize training data
Nearest Neighbor classifier

```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
```
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

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

• Demo
  – 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
Hyperparameters

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

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

**Slide Credit:** Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
## 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>
</table>

| 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

| train | test |
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!

$$k^*, d^* = \max_{k, d} \text{Acc (val)}$$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Parametric vs Non-Parametric Models

• Does the capacity (size of hypothesis class) grow with size of training data?
  - Yes = Non-Parametric Models
  - No = Parametric Models
Scene Completion

Original

Input

Scene Matches

Output

[Hayes & Efros, SIGGRAPH07]
Context Matching

Hays and Efros, SIGGRAPH 2007
Graph cut + Poisson blending
Problems with Instance-Based Learning

• Expensive
  – No Learning: most real work done during testing
  – For every test sample, must search through all dataset – very slow!
  – Must use tricks like approximate nearest neighbour search

• Doesn’t work well when large number of irrelevant features
  – Distances overwhelmed by noisy features

• Curse of Dimensionality
  – Distances become meaningless in high dimensions
  – (See proof next)
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)

Original image is [CC0 public domain](https://creativecommons.org/publicdomain/zero/1.0/)
k-Nearest Neighbor on images never used.

- Curse of dimensionality

Dimensions = 1
Points = 4

Dimensions = 2
Points = $4^2$

Dimensions = 3
Points = $4^3$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Curse of Dimensionality

• Consider: Sphere of radius 1 in d-dims

• Consider: an outer $\varepsilon$-shell in this sphere

• What is $\frac{\text{shell volume}}{\text{sphere volume}}$?
Curse of Dimensionality
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!