CS 4803 / 7643: Deep Learning

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

Dhruv Batra
Georgia Tech
Administrativia

• Piazza
  – 165/222 people signed up. Please use that for questions.

• Gradescope/Canvas
  – Anybody not have access?
  – See note on Piazza
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/

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
The Problem: Semantic Gap

What the computer sees

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

- e.g. $800 \times 600 \times 3$
- (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:
- airplane
- automobile
- bird
- cat
- deer

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

Scalars: $x, y, z, i, \in \mathbb{R}^i$

$\tilde{x}, \tilde{y} \in \mathbb{R}^d$

Matrices: $X, Y$

$R, V, S_d$

input dim: $d$

output dim: $K$

parameters: $\tilde{w}, \Theta \in \mathbb{R}^d$

# samples: $n, N$
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
  - $\{(x_1,y_1), (x_2,y_2), \ldots, (x_N,y_N)\}$

Goal: find $f$ / Predict $f(\tilde{x})$ at new $\tilde{x}$
Supervised Learning

Model Class / Hypothesis Set

\[ H = \{ h : X \to Y \} \]

\[ h(x) = y \]

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

Learning = Search/Opt in MC

\[ \text{Learning} = \text{argmin}_{h \in H} \text{Loss}(h, D) \]
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**
  - \{ \((x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\) \}

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

- **Learning = Search in hypothesis space**
  - Find best \( g \) in model class.
Learning is hard!

A Learning Problem

Example

<table>
<thead>
<tr>
<th></th>
<th>x₁</th>
<th>x₂</th>
<th>x₃</th>
<th>x₄</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(x₁, x₂, x₃, x₄)

(C) Dhruv Batra
Learning is hard!

• No assumptions = No learning

A Learning Problem

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

<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>
Procedural View

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

• Testing Stage
  – Test Data x \rightarrow h(x) \quad \text{(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
Error Decomposition
Error Decomposition

\[ y = h(x, D) \]

\[ \text{Model class} \]

\[ \text{Estimation Error} \]

\[ \text{Optimization Error} \]

\[ \text{Modeling Error} \]

\[ \text{Reality} \]

\[ \text{horse} \quad \text{person} \]

AlexNet

- Softmax
- FC 1000
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- FC 4096
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- FC 4096
- Pool
- 5x5 conv, 256
- 3x3 conv, 384
- Input
Error Decomposition

Modeling Error

Multi-class Logistic Regression

Softmax
FC HxWx3
input

Estimation Error

Optimization Error

model class

Reality

horse person

(C) Dhruv Batra
Error Decomposition

model class

Reality

VGG19

Softmax

FC 1000

FC 4096

FC 4096

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

Pool

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

Pool

3x3 conv, 128

3x3 conv, 128

3x3 conv, 128

3x3 conv, 128

3x3 conv, 128

Pool

3x3 conv, 64

3x3 conv, 64

3x3 conv, 64

3x3 conv, 64

Input

(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
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

Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, Truck


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
Nearest Neighbours
Instance/Memory-based Learning

Four things make a memory based learner:

- A **distance metric** \( d(\bar{x}_i, \bar{x}_j) \)

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

- A **weighting function** (optional)

- **How to fit with the local points?**
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

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></td>
</tr>
<tr>
<td>90 23 128 133</td>
<td>8 10 89 100</td>
<td></td>
</tr>
<tr>
<td>24 26 178 200</td>
<td>12 16 178 170</td>
<td></td>
</tr>
<tr>
<td>2 0 255 220</td>
<td>4 32 233 112</td>
<td></td>
</tr>
<tr>
<td></td>
<td>46 12 14 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>82 13 39 33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>12 10 0 30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 32 22 108</td>
<td></td>
</tr>
</tbody>
</table>

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

For each test image:
Find closest train image
Predict label of nearest image
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?
**Nearest Neighbor classifier**

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

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

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
What does this look like?
Nearest Neighbour

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

\[ H = \{ h : X \rightarrow Y \} \]
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

\[ d_1(I_1, I_2) = \sum_P |I_{1P}^p - I_{2P}^p| \]

L2 (Euclidean) distance

\[ d_2(I_1, I_2) = \sqrt{\sum_P (I_{1P}^p - I_{2P}^p)^2} \]

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

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

Your Dataset

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

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

Your Dataset

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

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

Your Dataset

| train | test |
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

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

Better!

train validation test

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

Your Dataset

**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.

---

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

Original

Input

Scene Matches

Output

[Hayes & Efros, SIGGRAPH07]
Context Matching

Hays and Efros, SIGGRAPH 2007
Graph cut + Poisson blending

Hays and Efros, SIGGRAPH 2007
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 in the public domain.

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

Figure Credit: Kevin Murphy
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!