CS 4803 / 7643: Deep Learning

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

Zsolt Kira
Georgia Tech
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
• Linear Classifiers
Image Classification
Image Classification: A core task in Computer Vision

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

→ cat

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

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!

---

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

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

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

John Canny, "A Computational Approach to Edge Detection", IEEE TPAMI 1986
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

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

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

• Input: x  (images, text, emails…)

• Output: y  (spam or non-spam…)

• (Unknown) Target Function
  – f: X → Y  (the “true” mapping / reality)

• Data
  – (x₁,y₁), (x₂,y₂), …, (xₐ,yₐ)

• Model / Hypothesis Class
  – h: X → Y
  – y = h(x) = sign(wᵀx)

• Learning = Search in hypothesis space
  – Find best h in model class.
Procedural View

- **Training Stage:**
  - Training Data \( \{(x,y)\} \rightarrow f \) (Learning)

- **Testing Stage**
  - Test Data \( x \rightarrow f(x) \) (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
Error Decomposition

Reality

(C) Dhruv Batra and Zsolt Kira
Error Decomposition

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

horse
person

model class

Optimization Error

Estimation Error

Modeling Error
Error Decomposition

Model class

Optimization Error

Estimation Error

Modeling Error

Multi-class Logistic Regression

Softmax

FC HxWx3

Input

Reality

horse  person

(C) Dhruv Batra and Zsolt Kira
Error Decomposition

VGG19
- Softmax
- FC 1000
- FC 4096
- Pool
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- Pool
- 3x3 conv, 256
- 3x3 conv, 256
- Pool
- 3x3 conv, 128
- 3x3 conv, 128
- Pool
- 3x3 conv, 64
- 3x3 conv, 64
- Input

model class

Optimization Error

Estimation Error

Modeling Error

Reality

horse
person

(C) Dhruv Batra and Zsolt Kira
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
Guarantees

• 20 years of research in Learning Theory
  oversimplified:

• If you have:
  – Enough training data D
  – and H is not too complex
  – then *probably* we can generalize to unseen test data
First classifier: **Nearest Neighbor**

- **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
Nearest Neighbours
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

Figure Credit: Carlos Guestrin
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>

\[ \text{add} \rightarrow 456 \]
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-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

Nearest Neighbor classifier
Memorize training data
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

Nearest Neighbor classifier

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 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?

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

This is bad: we want classifiers that are fast at prediction; slow for training is ok
What does this look like?
Nearest Neighbour

• Demo 1
  - http://vision.stanford.edu/teaching/cs231n-demos/knn/

• Demo 2
  - http://www.cs.technion.ac.il/~rani/LocBoost/
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
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

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
K-Nearest Neighbors: Distance Metric

Scaled Euclidian ($L_2$)

$L_1$ norm (absolute)

Mahalanobis

$L_{\text{infinity}}$ (max) norm

Slide by Andrew W. Moore
More Distance Metrics

\[ D(x, x') = \sqrt{\sum_{i} S_i^{-1}(x_i - x'_i)^2} \]

Or equivalently,

\[ D(x, x') = \sqrt{(x - x')^T S^{-1}(x - x')} \]

where

\[ S = \begin{bmatrix}
\sigma_1^2 & 0 & \ldots & 0 \\
0 & \sigma_2^2 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & \sigma_N^2 
\end{bmatrix} \]

Other Metrics…
- Mahalanobis, Rank-based, Correlation-based (Stanfill+Waltz, Maes’ Ringo system…)

Slide by Andrew W. Moore
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
Idea #1: Choose hyperparameters that work best on the data

BAD: K = 1 always works perfectly on training data

Your Dataset
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>

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

| train | test |

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>
<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

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

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!
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

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 [Hayes & Efros, SIGGRAPH07]
... 200 total

Hays and Efros, SIGGRAPH 2007
Context Matching
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 in next lecture)
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 = $4^3$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Figure 1.16  Illustration of the curse of dimensionality. (a) We embed a small cube of side $s$ inside a larger unit cube. (b) We plot the edge length of a cube needed to cover a given volume of the unit cube as a function of the number of dimensions. Based on Figure 2.6 from (Hastie et al. 2009). Figure generated by curseDimensionality.
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!
Wrap-Up

• Next time:
  – *The* building block of deep learning: Linear classifier!
  – Softmax/SVMs

• Given by Peter Anderson

• Peter Anderson
• Georgia Tech
• [www.panderson.me](http://www.panderson.me)