CS 559 Deep Learning

A simple image classification pipeline

Gokberk Cinbis
Incoming slides and lectures

● Some fundamentals of machine learning
  ○ A traditional image classification pipeline (Use raw pixels as features and nearest neighbor classifier)
  ○ Linear classification
  ○ Loss functions
  ○ Training by numerical optimization

● Then, deep learning will be a natural extension..
A simple image classification pipeline
Image Classification: a core task in Computer Vision

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

cat
The problem: semantic gap

Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)
Challenges: Viewpoint Variation

What the computer sees

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Challenges: Illumination
Challenges: Deformation
Challenges: Occlusion

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Challenges: Background clutter

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
Challenges: Intraclass variation
An image classifier

```python
def predict(image):
    # ????
    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
Data-driven approach:
1. Collect a dataset of images and labels
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
    # build a model for images -> labels...
    return model

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```
First classifier: **Nearest Neighbor Classifier**

Remember all training images and their labels

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

def predict(model, test_images):
    # predict test_labels using the model...
    return test_labels
```

Predict the label of the most similar training image
Example dataset: **CIFAR-10**

10 labels

**50,000** training images, each image is tiny: 32x32

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

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
How do we compare the images? What is the distance metric?

**L1 distance:**

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

<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>

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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 ""
        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]
        Ypred = np.zeros((num_test, self.ytr.dtype))

        for i in range(num_test):
            # find the nearest training image to the i'th test image
            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

for every test image:
- find nearest train image with L1 distance
- predict the label of nearest training image
Nearest Neighbor classifier

Q: how does the classification speed depend on the size of the training data?

```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]
        # let's 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
```

Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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: how does the classification speed depend on the size of the training data? linearly :(

This is backwards:
- test time performance is usually much more important in practice.
- CNNs flip this: expensive training, cheap test evaluation
Aside: Approximate Nearest Neighbor
find approximate nearest neighbors quickly

What is ANN?

ANN is a library written in C++, which supports data structures and algorithms for both exact and approximate nearest neighbor searching in arbitrary high dimensions.

In the nearest neighbor problem a set of data points in d-dimensional space is given. These points are preprocessed into a data structure, so that given any query point q, the nearest or generally k nearest points of P to q can be reported efficiently. The distance between two points can be defined in many ways. ANN assumes that distances are measured using any class of distance functions called Minkowski metrics. These include the well known Euclidean distance, Manhattan distance, and max distance.

Based on our own experience, ANN performs quite efficiently for point sets ranging in size from thousands to hundreds of thousands, and in dimensions as high as 20. (For applications in significantly higher dimensions, the results are rather spotty, but you might try it anyway.)

The library implements a number of different data structures, based on kd-trees and box-decomposition trees, and employs a couple of different search strategies.

The library also comes with test programs for measuring the quality of performance of ANN on any particular data sets, as well as programs for visualizing the structure of the geometric data structures.
The choice of distance is a **hyperparameter**
common choices:

**L1 (Manhattan) distance**

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

**L2 (Euclidean) distance**

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I_{1p}^p - I_{2p}^p)^2} \]
k-Nearest Neighbor
find the k nearest images, have them vote on the label


Slide by Fei-Fei Li, Andrej Karpathy & Justin Johnson
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.
Q: what is the accuracy of the nearest neighbor classifier on the training data, when using the Euclidean distance?
Q2: what is the accuracy of the $k$-nearest neighbor classifier on the training data?
What is the best distance to use?
What is the best value of $k$ to use?

i.e. how do we set the hyperparameters?
What is the best distance to use?
What is the best value of $k$ to use?

i.e. how do we set the hyperparameters?

Very problem-dependent.
Must try them all out and see what works best.
Try out what hyperparameters work best on test set.
Trying out what hyperparameters work best on test set:
Very bad idea. The test set is a proxy for the generalization performance!
Use only **VERY SPARINGLY**, at the end.
Validation data
use to tune hyperparameters
Cross-validation

cycle through the choice of which fold is the validation fold, average results.
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 is **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)
Summary

- **Image Classification**: We are given a **Training Set** of labeled images, asked to predict labels on **Test Set**. Common to report the **Accuracy** of predictions (fraction of correctly predicted images)

- We introduced the **k-Nearest Neighbor Classifier**, which predicts the labels based on nearest images in the training set

- We saw that the choice of distance and the value of k are **hyperparameters** that are tuned using a **validation set**, or through **cross-validation** if the size of the data is small.

- Once the best set of hyperparameters is chosen, the classifier is evaluated once on the test set, and reported as the performance of kNN on that data.