Lecture 2: Image Classification
A Core Task in Computer Vision
Administrative: Assignment 1

Due 4/22 11:59pm

- K-Nearest Neighbor
- Linear classifiers: SVM, Softmax
- Two-layer neural network
- Image features
Administrative: Course Project

Project proposal due 4/27

Find your teammates on Piazza and Slack

Collaboration: Slack / Zoom
Administrative: Sections

This Friday 12:30pm

Python / Numpy, Google Cloud Platform, Google Colab

Presenter: Karen Yang, Kevin Zakka
Lecture 2:  
Image Classification  
A Core Task in Computer Vision  

Today:  
- The Image Classification Task  
- Nearest Neighbor Classifier  
- Linear Classifier
Image Classification: A core task in Computer Vision

(assume given a set of labels)
{dog, cat, truck, plane, ...}
The Problem: Semantic Gap

What the computer sees

An image is just a tensor of integers 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: Occlusion
Challenges: Deformation
**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
Machine Learning: Data-Driven Approach

1. Collect a dataset of images and labels
2. Use Machine Learning algorithms 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
```
Nearest Neighbor Classifier
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
First classifier: **Nearest Neighbor**

Training data with labels:
- deer
- bird
- plane
- cat
- car

Distance Metric: $d(\text{query data}, \text{training data}) \rightarrow \mathbb{R}$
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

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 l-dimension of size N
        self.Xtr = X
        self.ytr = y

    def predict(self, X):
        num_test = X.shape[0]
        Ypred = np.zeros(num_test, dtype=self.ytr.dtype)
        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-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

Nearest Neighbor classifier

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

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

A. O(1) for training and O(1) for evaluation
B. O(1) for training and O(N) for evaluation
C. O(N) for training and O(1) for evaluation
D. O(N) for training and O(N) for evaluation
Nearest Neighbor classifier

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

A. $O(1)$ for training and $O(1)$ for evaluation
B. $O(1)$ for training and $O(N)$ for evaluation
C. $O(N)$ for training and $O(1)$ for evaluation
D. $O(N)$ for training and $O(N)$ for evaluation
Nearest Neighbor classifier

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

**Ans:** 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

---

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

---

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](image1.png)

![K = 3](image2.png)

![K = 5](image3.png)
What does this look like?
What does this look like?
K-Nearest Neighbors: Distance Metric

L1 (Manhattan) distance

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

L2 (Euclidean) distance

\[ d_2(I_1, I_2) = \sqrt{\sum_p (I_{1p}^p - I_{2p}^p)^2} \]
K-Nearest 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 algorithms themselves.
Hyperparameters

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

These are hyperparameters: choices about the algorithms themselves.

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

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

**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

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!

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 with pixel distance **never used.**

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

(all 3 images have same L2 distance to the one on the left)
k-Nearest Neighbor with pixel distance **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 the K 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!

Pixel distance is not very informative.
Linear Classifier
Parametric Approach

Image

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

$\mathbf{W}$
parameters or weights

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

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

- **Image**
- Array of \(32 \times 32 \times 3\) numbers (3072 numbers total)
- **W**: parameters or weights
- **10** numbers giving class scores
Parametric Approach: Linear Classifier

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

Image

Array of \textbf{32x32x3} numbers (3072 numbers total)

parameters or weights

\[ W \]

10 numbers giving class scores

\[ f(x, W) \]

10x3072

3072x1

10x1

10 numbers giving class scores
Parametric Approach: Linear Classifier

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

Array of 32x32x3 numbers (3072 numbers total)

Parameters or weights
Neural Network

Linear classifiers
Two young girls are playing with lego toy.

Boy is doing backflip on wakeboard.

Man in black shirt is playing guitar.

Construction worker in orange safety vest is working on road.

[Krizhevsky et al. 2012]

[He et al. 2015]
Recall CIFAR10

50,000 training images
each image is 32x32x3

10,000 test images.
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

Input image
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Flatten tensors into a vector

\[ \begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix} \begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix} + \begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix} = \begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix} \]

Cat score  
Dog score  
Ship score
Interpreting a Linear Classifier

airplane  automobile  bird  cat  deer  dog  frog  horse  ship  truck

Input image

$W = \begin{bmatrix} 0.2 & -0.5 \\ 0.1 & 2.0 \end{bmatrix}$

$b = \begin{bmatrix} 1.1 \\ 3.2 \end{bmatrix}$

Score

$\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}$
Interpreting a Linear Classifier: Visual Viewpoint

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Input image

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

\[
b = \begin{bmatrix}
1.1 \\
3.2 \\
437.9 \\
61.95 \\
\end{bmatrix}
\]

Score

plane  car  bird  cat  deer  dog  frog  horse  ship  truck
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**

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

Input image

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

\[
\begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0   & 0.25 & 0.2 & -0.3
\end{bmatrix}
\]

Input image stretch pixels into column:

\[
\begin{bmatrix}
56 \\
231 \\
24 \\
2
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.1 \\
3.2 \\
-1.2
\end{bmatrix}
\]

**Visual Viewpoint**

Input image

\[
\begin{bmatrix}
56 \\
231 \\
24 \\
2
\end{bmatrix}
\]

\[
\begin{bmatrix}
0.2 & -0.5 & 0.1 & 2.0 \\
1.5 & 1.3 & 2.1 & 0.0 \\
0   & 0.25 & 0.2 & -0.3
\end{bmatrix}
\]

\[
\begin{bmatrix}
1.1 \\
3.2 \\
-1.2
\end{bmatrix}
\]

\[
\begin{bmatrix}
0 \\
.25
\end{bmatrix}
\]
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of \(32 \times 32 \times 3\) numbers
(3072 numbers total)
Hard cases for a linear classifier

**Class 1:**
First and third quadrants

**Class 2:**
Second and fourth quadrants

**Class 1:**
$1 \leq \text{L2 norm} \leq 2$

**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
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 $f$)

$$f(x, W) = Wx + b$$