CS 4803 / 7643: Deep Learning

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

Zsolt Kira
Georgia Tech
Last Time

- High-level intro to what deep learning is

- Fast brief of logistics
  - Requirements: ML, math (linear algebra, calculus), programming (python)
  - Grades: 80% PS/HW, 20% Project, Piazza Bonus
  - Project: Topic of your choosing (related to DL), groups of 3-4 with separated undergrad/grad
  - 7 free late days
  - 1 week re-grading period
  - No Cheating

- PS0 out, due **Tuesday 01/14 11:55pm**
  - Graded pass/fail
  - Intended to do on your own
  - Don’t worry if rusty! It’s OK to need a refresher on various subjects to do it. Some of it (e.g. last question) is more suitable for graduate students.
  - If not registered, email staff for gradescope account

- Look through slides on website for all details
Current TAs

- **Sameer Dharur**
  MS-CS student
  [https://www.linkedin.com/in/sameerdharur/](https://www.linkedin.com/in/sameerdharur/)

- **Rahul Duggal**
  2nd year CS PhD student

- **Patrick Grady**
  2nd year Robotics PhD student
  [https://www.linkedin.com/in/patrick-grady](https://www.linkedin.com/in/patrick-grady)

- **Anishi Mehta**
  MSCS student
  [https://www.linkedin.com/in/anishimehta](https://www.linkedin.com/in/anishimehta)

- **Yinquan Lu**
  2nd year MSCSE student
  [https://www.cc.gatech.edu/~jyang462/](https://www.cc.gatech.edu/~jyang462/)

- **Jiachen Yang**
  2nd year ML PhD
  [https://www.cc.gatech.edu/~jyang462/](https://www.cc.gatech.edu/~jyang462/)

- **New TAs**: Zhuoran Yu. Manas Sahni, (in process) Harish Kamath
  - Official office hours coming soon (TA and instructor)
  - For this & next week:
    - 11:30am-12:30pm **Friday 01/09** (Zhuoan Yu)
    - 11:30-12:30am on **Monday** (Patrick)
    - 11:30 AM to 12:30 PM on **Tuesdays**. (Sameer)
    - 4-5pm **Tuesday** (Jiachen)
    - 1:30-2:30 pm on **Wed.** (Anishi)
    - 11:30 AM to 12:30 PM on **Thursdays**. (Rahul)
Registration/Access

• Waitlist
  – Still a large waitlist for grad, still adding some capacity
• Canvas
  – Anybody not have access?
• Piazza
  – 110+ people signed up. Please use that for questions.

Website: http://www.cc.gatech.edu/classes/AY2020/cs7643_spring/
Piazza: https://piazza.com/gatech/spring2020/cs4803dl7643a/
Staff mailing list (personal questions): cs4803-7643-staff@lists.gatech.edu
Gradescope: https://www.gradescope.com/courses/78537
Canvas: https://gatech.instructure.com/courses/94450/

Course Access Code (Piazza): MWXKY8
Prep for HW1:
Python+Numpy Tutorial

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

• Reminder:
  – What changed to enable DL
• Some Problems with DL
• Image Classification
• Supervised Learning view
• K-NN
• (Beginning of) Linear Classifiers
Reminder: What Deep Learning Is

• We will learn a complex non-linear **hierarchical** (compositional) function in an **end-to-end manner**

• (Hierarchical) Compositionality
  – Cascade of non-linear transformations
  – Multiple layers of representations

• End-to-End Learning
  – Learning (goal-driven) representations
  – Learning to feature extraction

• Distributed Representations
  – No single neuron “encodes” everything
  – Groups of neurons work together

(C) Dhruv Batra & Zsolt Kira
What Changed?

• Few people saw this combination coming: gigantic growth in data and processing to enable depth and feature learning
  – Combined with specialized hardware (gpus) and open-source/distribution (arXiv, github)

• If the input features are poor, so will your result be
  – If your model is poor, so will your result be
    • If your optimizer is poor, so will your result be

• Now we have methods for feature learning that works (after some finesse)
  – Still have to guard against overfitting (very complex functions!)
  – Still tune hyper-parameters
  – Still design neural network architectures
  – Lots of research to automate this too, e.g. via reinforcement learning!
Problems with Deep Learning

• **Problem#1: Non-Convex! Non-Convex! Non-Convex!**
  – Depth≥3: most losses non-convex in parameters
  – Theoretically, all bets are off
  – Leads to stochasticity
    • different initializations → different local minima

• **Standard response #1**
  – “Yes, but all interesting learning problems are non-convex”
  – For example, human learning
    • Order matters → wave hands → non-convexity

• **Standard response #2**
  – “Yes, but it often works!”
Problems with Deep Learning

• Problem#2: Lack of interpretability
  – Hard to track down what’s failing
  – Pipeline systems have “oracle” performances at each step
  – In end-to-end systems, it’s hard to know why things are not working
Problems with Deep Learning

- Problem#2: Lack of interpretability

[1] [Fang et al. CVPR15]

[2] [Vinyals et al. CVPR15]
Problems with Deep Learning

• Problem#2: Lack of interpretability
  – Hard to track down what’s failing
  – Pipeline systems have “oracle” performances at each step
  – In end-to-end systems, it’s hard to know why things are not working

• Standard response #1
  – Tricks of the trade: visualize features, add losses at different layers, pre-train to avoid degenerate initializations…
  – “We’re working on it”

• Standard response #2
  – “Yes, but it often works!”
Problems with Deep Learning

• Problem#3: Lack of easy reproducibility
  – Direct consequence of stochasticity & non-convexity

• Standard response #1
  – It’s getting much better
  – Standard toolkits/libraries/frameworks now available
  – Caffe, Theano, (Py)Torch

• Standard response #2
  – “Yes, but it often works!”
Yes it works, but how?

Good work -- but I think we might need a little more detail right here.

(C) Dhruv Batra & Zsolt Kira
Image Classification
Image Classification: A core task in Computer Vision

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

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

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

```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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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**
  - \( h: X \rightarrow Y \)
  - \( y = h(x) = \text{sign}(w^T x) \)

- **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
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

model class

Reality

Optimization Error

Estimation Error

Modeling Error
Error Decomposition

- Multi-class Logistic Regression
- Softmax
- FC HxWx3
- Input
- Optimization Error
- Estimation Error
- Modeling Error
- Reality

(C) Dhruv Batra & 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, 256
- 3x3 conv, 256
- Pool
- 3x3 conv, 128
- 3x3 conv, 128
- Pool
- 3x3 conv, 64
- 3x3 conv, 64
- Input

model class

Estimation Error

Optimization Error

Modeling Error

Reality

horse
person

(C) Dhruv Batra & 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

• Note: Several ways to measure complexity
  – Vapnik–Chervonenkis dimension
  – Rademacher complexity
Learning is hard!

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>
Learning is hard!

- No assumptions = No learning

A Learning Problem

```
<table>
<thead>
<tr>
<th>Example</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>
```

\( y = f(x_1, x_2, x_3, x_4) \)
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
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>

Add: 456
```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
```

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-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

```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
```

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-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 $O(N)$
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
What does this look like?

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

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 = 1

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
- $L_{\infty}$ norm (max)
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

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

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

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

---

*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

---

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

```
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
```
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
Scene Completion [Hayes & Efros, SIGGRAPH07]
… 200 total

Hays and Efros, SIGGRAPH 2007
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
k-Nearest Neighbor on images *never used.*

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

(Original image is CC0 public domain)

(all 3 images have same L2 distance to the one on the left)
k-Nearest Neighbor on images never used.

- Curse of dimensionality

\[
\begin{align*}
\text{Dimensions} &= 1 \\
\text{Points} &= 4
\end{align*}
\]

\[
\begin{align*}
\text{Dimensions} &= 2 \\
\text{Points} &= 4^2
\end{align*}
\]

\[
\begin{align*}
\text{Dimensions} &= 3 \\
\text{Points} &= 4^3
\end{align*}
\]
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.
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!
Visual Question Answering

Image Embedding (VGGNet)

Question Embedding (LSTM)

“How many horses are in this image?”

Neural Network Softmax over top K answers
Recall CIFAR10

50,000 training images
each image is 32x32x3

10,000 test images.

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

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

Image

$f(x, W)$

$W$

10 numbers giving class scores

parameters or weights

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Parametric Approach: Linear Classifier

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

Image

Array of 32x32x3 numbers (3072 numbers total)

10 numbers giving class scores

W
parameters or weights

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Parametric Approach: Linear Classifier

Image

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

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

$10 \times 1$ $10 \times 3072$

$3072 \times 1$

$10$ numbers giving class scores

$W$

parameters or weights

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Parametric Approach: Linear Classifier

Image of a cat.

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

Mathematical expression:

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

- $x$: 10x1 vector
- $W$: 10x3072 matrix (parameters or weights)
- $b$: 10x1 vector (bias)

Result: 10 numbers giving class scores.
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

Model class

Modeling Error

Estimation Error

Optimization Error

Reality: horse, person
Error Decomposition

Multi-class Logistic Regression

- Softmax
- FC HxWx3
- Input

Modeling Error

Estimation Error

Optimization Error = 0

Reality

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Example with an image with 4 pixels, and 3 classes (cat/dog/ship)

Input image

Stretch pixels into column

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

\[
\begin{array}{c}
56 \\
231 \\
24 \\
2 \\
\end{array}
\]

\[
\begin{array}{c}
1.1 \\
3.2 \\
-1.2 \\
\end{array}
\]

\[
\begin{array}{c}
-96.8 \\
437.9 \\
61.95 \\
\end{array}
\]

Cat score
Dog score
Ship score

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

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Interpreting a Linear Classifier

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Interpreting a Linear Classifier: Visual Viewpoint

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Interpreting a Linear Classifier: Geometric Viewpoint

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

Array of 32x32x3 numbers (3072 numbers total)

Plot created using Wolfram Cloud

Cat image by Nikita is licensed under CC-BY 2.0

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Hard cases for a linear classifier

**Class 1:**
- First and third quadrants
- 1 \( <= \) L2 norm \( <= \) 2

**Class 2:**
- Second and fourth quadrants
- Everything else

**Class 1:**
- Three modes

**Class 2:**
- Everything else

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Linear Classifier: Three Viewpoints

**Algebraic Viewpoint**

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

**Visual Viewpoint**

One template per class

**Geometric Viewpoint**

Hyperplanes cutting up space
So far: Defined a (linear) score function

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

Example class scores for 3 images for some \( W \):

How can we tell whether this \( W \) is good or bad?
So far: Defined a (linear) **score function**

![Image](cat.png) by Nikita is licensed under CC-BY 2.0; ![Image](car.png) is CC0 1.0 public domain; ![Image](frog.png) is in the public domain

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Score</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>airplane</td>
<td>-3.45</td>
<td>-0.51</td>
<td>3.42</td>
</tr>
<tr>
<td>automobile</td>
<td>-8.87</td>
<td>6.04</td>
<td>4.64</td>
</tr>
<tr>
<td>bird</td>
<td>0.09</td>
<td>5.31</td>
<td>2.65</td>
</tr>
<tr>
<td>cat</td>
<td>2.9</td>
<td>-4.22</td>
<td>5.1</td>
</tr>
<tr>
<td>deer</td>
<td>4.48</td>
<td>-4.19</td>
<td>2.64</td>
</tr>
<tr>
<td>dog</td>
<td>8.02</td>
<td>3.58</td>
<td>5.55</td>
</tr>
<tr>
<td>frog</td>
<td>3.78</td>
<td>4.49</td>
<td>-4.34</td>
</tr>
<tr>
<td>horse</td>
<td>1.06</td>
<td>-4.37</td>
<td>-1.5</td>
</tr>
<tr>
<td>ship</td>
<td>-0.36</td>
<td>-2.09</td>
<td>-4.79</td>
</tr>
<tr>
<td>truck</td>
<td>-0.72</td>
<td>-2.93</td>
<td>6.14</td>
</tr>
</tbody>
</table>

TODO:

1. Define a **loss function** that quantifies our unhappiness with the scores across the training data.

2. Come up with a way of efficiently finding the parameters that minimize the loss function. *(optimization)*

---

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