COMPSCI 682

Neural Networks: A Modern Introduction

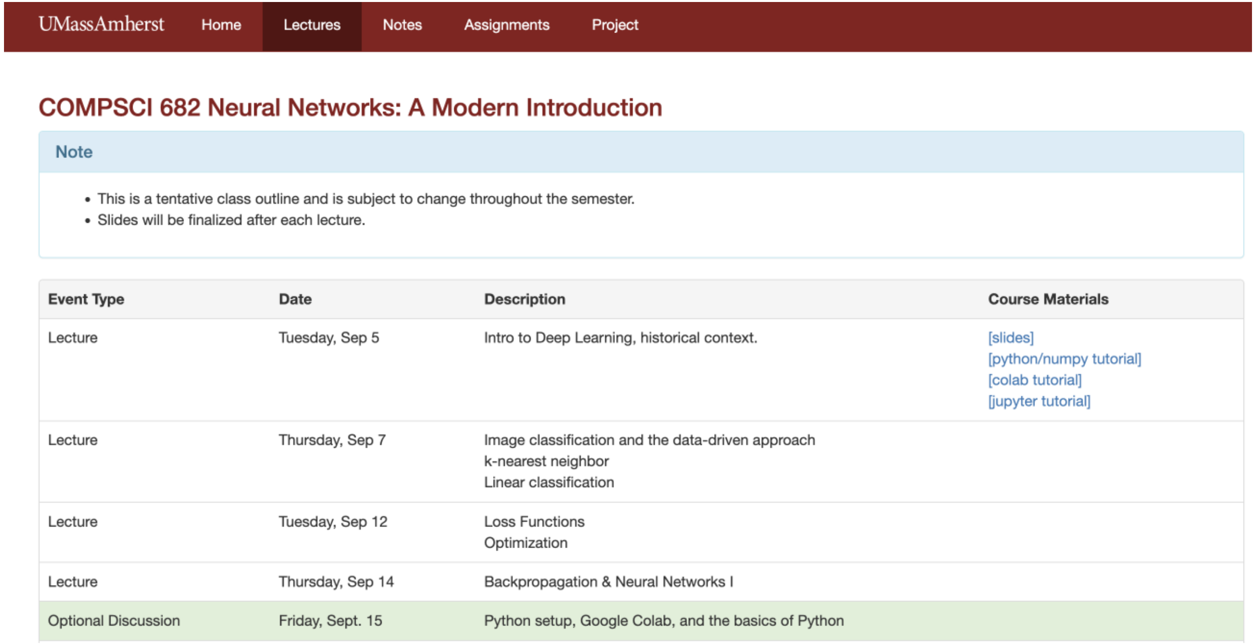
Lecture 1: Course Organization & Introduction of Deep Learning

Who we are

- Instructors
    - Prof. Subhransu Maji
    - Prof. Chuang Gan
- TAs:

Course web page

- https://cvl-umass.github.io/compsci682-fall-2023



UMassAmherst | Home | Lectures | Notes | Assignments | Project

COMPSCI 682 Neural Networks: A Modern Introduction

Note

- This is a tentative class outline and is subject to change throughout the semester.
- Slides will be finalized after each lecture.

Event Type	Date	Description	Course Materials
Lecture	Tuesday, Sep 5	Intro to Deep Learning, historical context.	[slides] [python/numpy tutorial] [colab tutorial] [jupyter tutorial]
Lecture	Thursday, Sep 7	Image classification and the data-driven approach k-nearest neighbor Linear classification	
Lecture	Tuesday, Sep 12	Loss Functions Optimization	
Lecture	Thursday, Sep 14	Backpropagation & Neural Networks I	
Optional Discussion	Friday, Sept. 15	Python setup, Google Colab, and the basics of Python	

Optional Discussion Sections

- Friday: 9:00-10:00 am, CS142
  - No discussion section this Friday
- Will cover background topics such as:
  - Python techniques
    - slicing and broadcasting
    - Other parallelization techniques
  - Math techniques
    - Derivatives of vectors, matrices, etc.
    - Complex chain rule examples

682 Neural Networks: A Modern Introduction

- Topics: first half of the class.
  - Intro to supervised learning with k-nearest neighbors
  - Support vector machines
  - Logistic regression for classification
  - Feed forward neural nets
    - Backpropagation
    - Batch normalization
    - Drop-out
    - Speed optimizations
  - Convolutional neural nets

682 Neural Networks: A Modern Introduction

- Balance of theory vs. practice
  - Heavily tilted toward practice.
  - Examples:
    - Regularization will be used, but not much theory of it.
    - No proofs of convergence
  - Instead:
    - Develop applications “from scratch”
    - Build “layered” architectures from scratch so new models can be easily assembled
    - Implement popular add-ons such as batch normalization
    - Learn techniques for training and setting hyperparameters.

682 Neural Networks: A Modern Introduction

- Applications
  - Mostly **Computer Vision**: Object recognition in particular.
  - However, can easily be applied to other domains.
    - You will learn what you need to know to apply neural nets broadly.
  - Will add more about **Natural Language Processing** (or **Large Language Models**) this semester.

682 Neural Networks: A Modern Introduction

- What this course is *not*:
    - General course on machine learning
    - General course on graphical models
    - Not even a general class on deep learning!!!
        - No Bayes Nets
        - No restricted Boltzmann machines or deep Boltzmann machines
    - Not a computer vision survey class
        - No tracking, stereo, depth estimation, etc., etc.

Course grades

- 3 Long Programming Assignments
  - Get started as soon as assignments are posted.
  - Most assignments require require in-depth understanding of python, such as numpy arrays and complicated indexing schemes..
  - If you don't know Python, please consider either withdrawing this class or working through tutorial now

Grading Policy (approximate)

- 3 Problem sets: 15%*3 = 45%
- Midterm exam: 15% - Mid November
- Final Course project: 40%
    - Proposal: 5% (out of 40%)
    - Milestone: 5% (out of 40%)
    - Final write-up: 20% (out of 40%)
    - Review of others: 10% (out of 40%)
- Late Policy:
    - 7 free late days in total: use them as you see fit
    - Afterwards: 25% off per day late
    - Not accepted after 3 late days
    - Does not apply to final course project (must be on time)

Getting Started

- Example: Mac
  - Language: Python
    1. Instructions for installing given under first assignment instructions.
    2. Development environment: Jupyter Notebook. Live code environment.
       - Poll
  - Running a shell on the side: Jupyter QtConsole
    1. Good for testing syntax, return values of functions.

Assignment #1

- Soon posted on course website
- Due in 3 weeks (Thursday, Sept. 23, 11:55pm) (in GradeScope).
- It includes:
- Write/train/evaluate a kNN classifier
- Write/train/evaluate a Linear Classifier (SVM and Softmax)
- Write/train/evaluate a 2-layer Neural Network (backpropagation!)
- Requires writing numpy/Python code

Compute: Use your own laptops. Talk to TA if you don’t have your own computer.

Plagiarism and Cheating

Who, me?

- Right now, cheating seems very far away.
- Now imagine:
    - You just started homework due in 2 days.
      You realize it will take you a week.
    - You just had an internship interview where they asked you if you are getting an A in Neural Nets.
    - You have a midterm tomorrow and a project due in another class in one week.
    - You were just surfing the web for information on Python slicing and you bumped into a full solution to the current problem set.
      Perhaps I should just take a quick peek...

Don’t do it!!!!!!!!!!

Cheating in the past

- 10% students were caught cheating during a recent semester.
- They were given penalties including
  - 0 for the given assignment
  - An additional grade reduction for the class.
  - A filing with the Academic Dishonesty board.
- Many people failed the class as a result.

UMass Culture

- If you cheat, you put me and a lot of other people in an awful position:
  - If I let you off the hook, I am being completely unfair to people who actually did the work, and I'm promoting the idea that it's ok.
  - If I punish you, I feel like a jerk, and you think I'm a jerk.
- The bottom line is, there is no good way to come out feeling good about a cheating incident. It creates massive stress between faculty and students. Please don't do it!

Advice

- Everyone knows you're not supposed to cheat.
- What people don't know is what you're supposed to do when you're desperate. Here's some advice:

1) If you're overloaded in the middle of the semester, consider dropping a class. Hopefully you can drop it without a "W", but even a "W" is a lot better than an "F" and a record of cheating. A "W" will not influence your grade point average.
   (I dropped the same class 4 times in grad school!)
2) Take a "0" on part of the problem set. Many people who did not do part of one problem set got an A-. Some people missed a whole problem set and still got a B for the course.

5 Rules: What is cheating?

1. Let's start with an easy one. Don't copy any piece of the solution of any problem.
2. Never **look at** solutions to any of the homework problems. Most people who were caught cheating last semester claimed that they only "looked at" on-line solutions. This is NOT ALLOWED.
3. Do not look at discussions of the homework problems. These are likely to include methods for solving parts of the problem, which is cheating.
4. Don't look up pieces of the problem on Google. For example:
    a. "Computing the derivative of softmax"
    b. "Gradient updates for the multi-class SVM loss".

    Once you've done the search, you cheated. You are likely to see something you cannot forget. You can't "unsee" the answer once you've seen it.
5. Common sense. If you look at something on the web and it made the problem easier, then you're probably cheating. To be safe, stick to class materials, TAs, and Professors.

Questions about what is allowed

1. Question: Can I work with other students on the homeworks?
   Answer: No. Do the homeworks yourself.
2. Question: Where can I get help?
   a. Look at the course notes
   b. Go to optional Friday sections
   c. Talk to the TAs
   d. Talk to the professor
3. Can I look at on-line materials that are not part of the course?
   a. Basically no. If you look at something and it's part of the solution, then you have cheated. So it's dangerous to go surfing around. Stick to the materials on the course web site. If there is something you want to look up, ask the TAs a question and we'll try to put materials on the course web site if it's appropirate.

Plagiarism

- When you write your final report, there are two ways you can use material from other papers:
  - Use the general ideas from another paper with your own writing. You *cannot* copy text from another paper unless you use quotation marks. Example:
    - In his famous 1915 paper, Einstein introduced the theory of general relativity [Einstein, 1915].
  - Quote a specific passage, usually because of the exact way it is worded:
    - Einstein said, “God does not place dice with the universe.” [Einstein, 1958]

Plagiarism

- You cannot copy sentences into your writing and justify by citing the paper. This is plagiarism, whether you cite it or not.

  If I Google a sentence in your paper that is not quoted, and I find it, that means you were plagiarising!

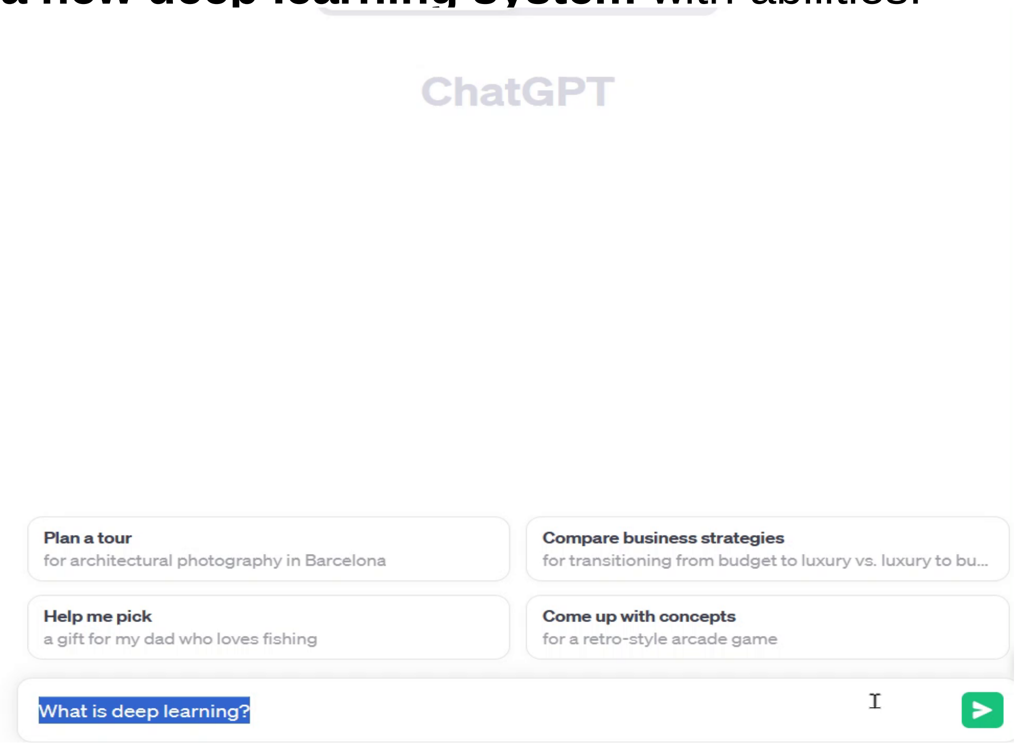
Final comment: If you don’t know whether something constitutes plagiarism or cheating, ASK! If you don’t ask, it will be too late.

OK…. now on to the fun stuff!

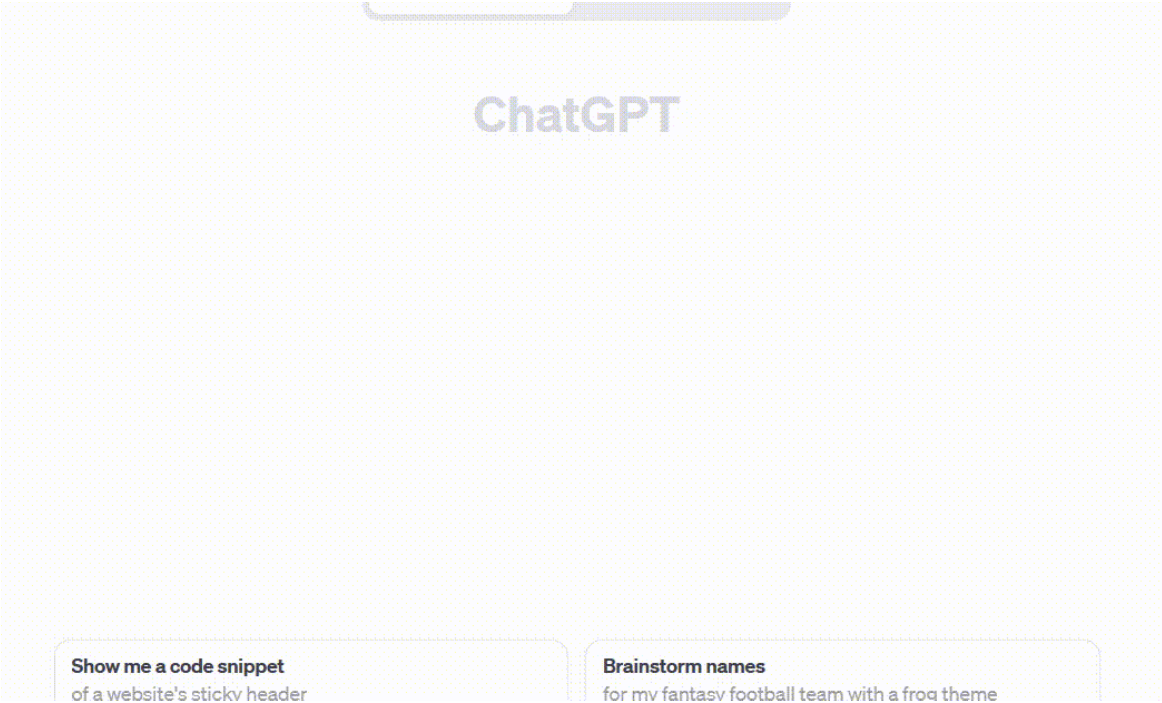
What is Deep Learning?

Recently, OpenAI developed **a new deep learning system** with abilities:

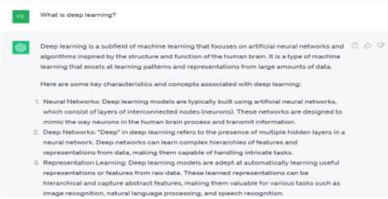
- Conversation
- Language Translation
- Text Summarization
- Data Analysis
- Code Debug
- Travel Guide
- Music Composition
- Math Problem Solving
- ...



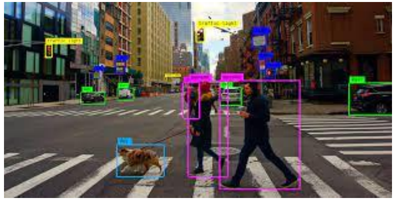
Why Deep Learning?



- Natural Language Processing



- Computer Vision

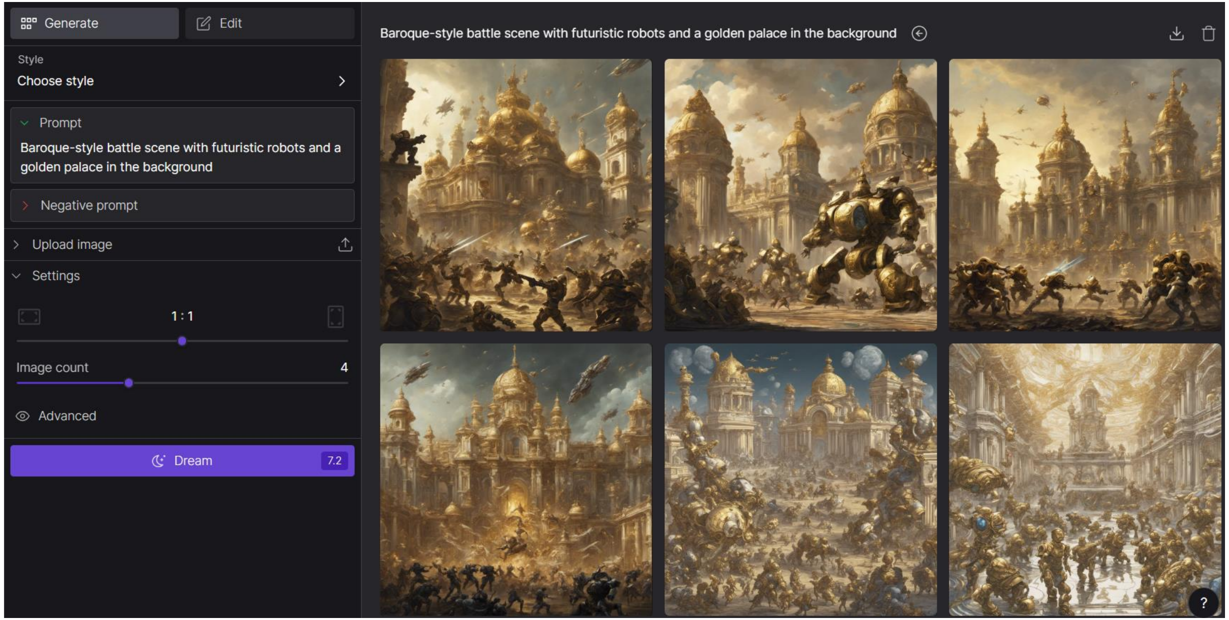


- Reinforcement Learning

Why Deep Learning?

Stable Diffusion

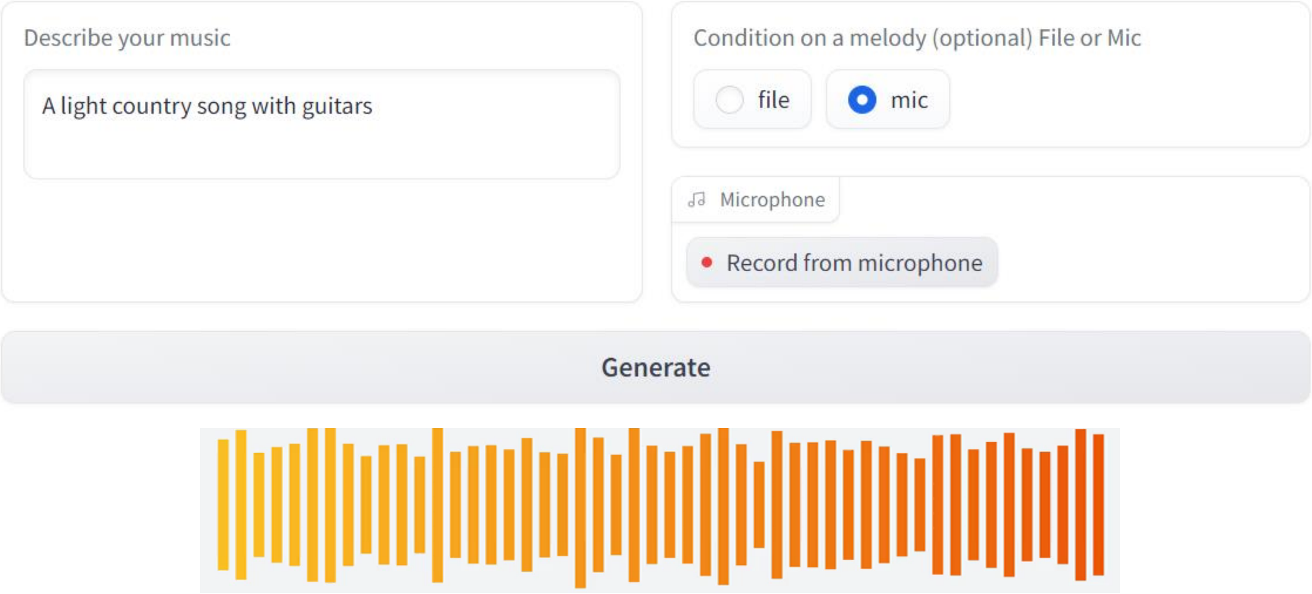
- Generate images / Edit existing images based on the text prompt



Why Deep Learning?

MusicGen

- Generate music based on the text prompt

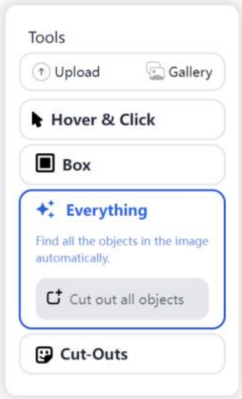


Why Deep Learning?

Segment Anything

Meta

- Segment image based on segmentation prompt



Why Now?

Neural networks have a history of over 70 years, but deep learning surged in the last decade.

1952	1958	1986	1995	2012
Stochastic Gradient Descent	Perceptron	Backpropagation	CNN for digit recognition	AlexNet (Igniting the wave of DL)

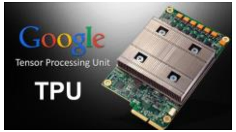
Big data

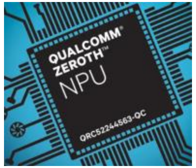
- Large Datasets
- Advances in data collection & storage

YouTube
hadoop
Google Cloud

Hardware

- GPU acceleration
- AI-specific chips
- Distributed computing





Software

- Open-Source Frameworks
- Active Community

TensorFlow
PyTorch
GitHub
Hugging Face

Image Classification: a core task in Computer Vision

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

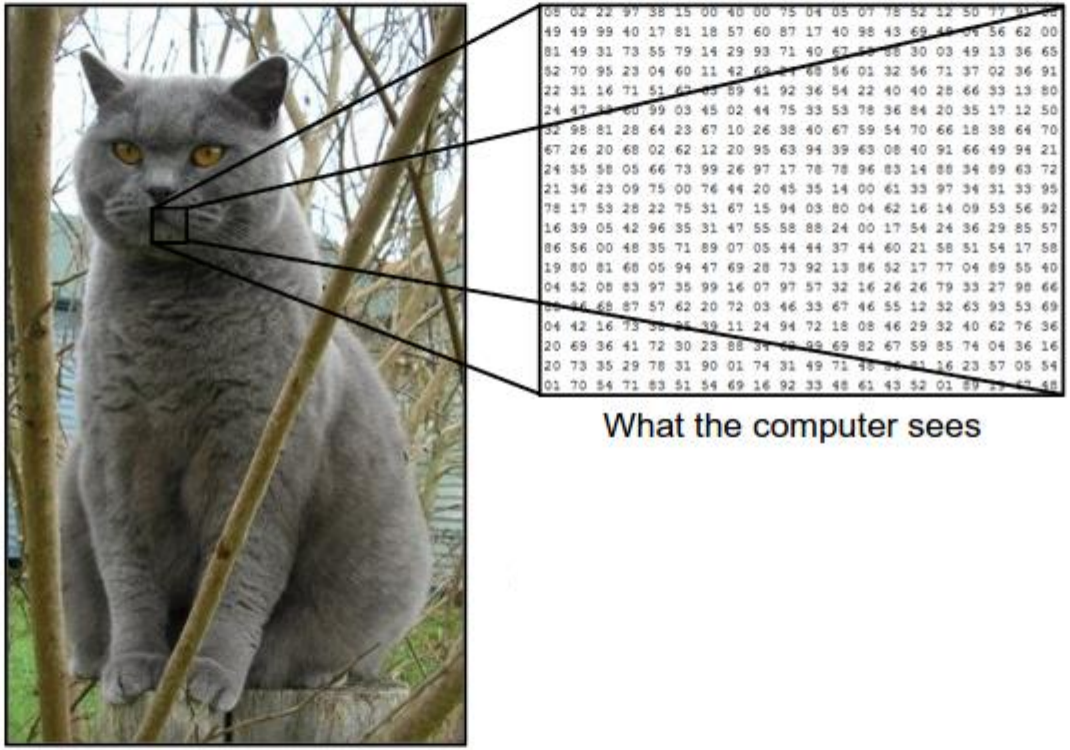
→ cat

The problem:
semantic gap

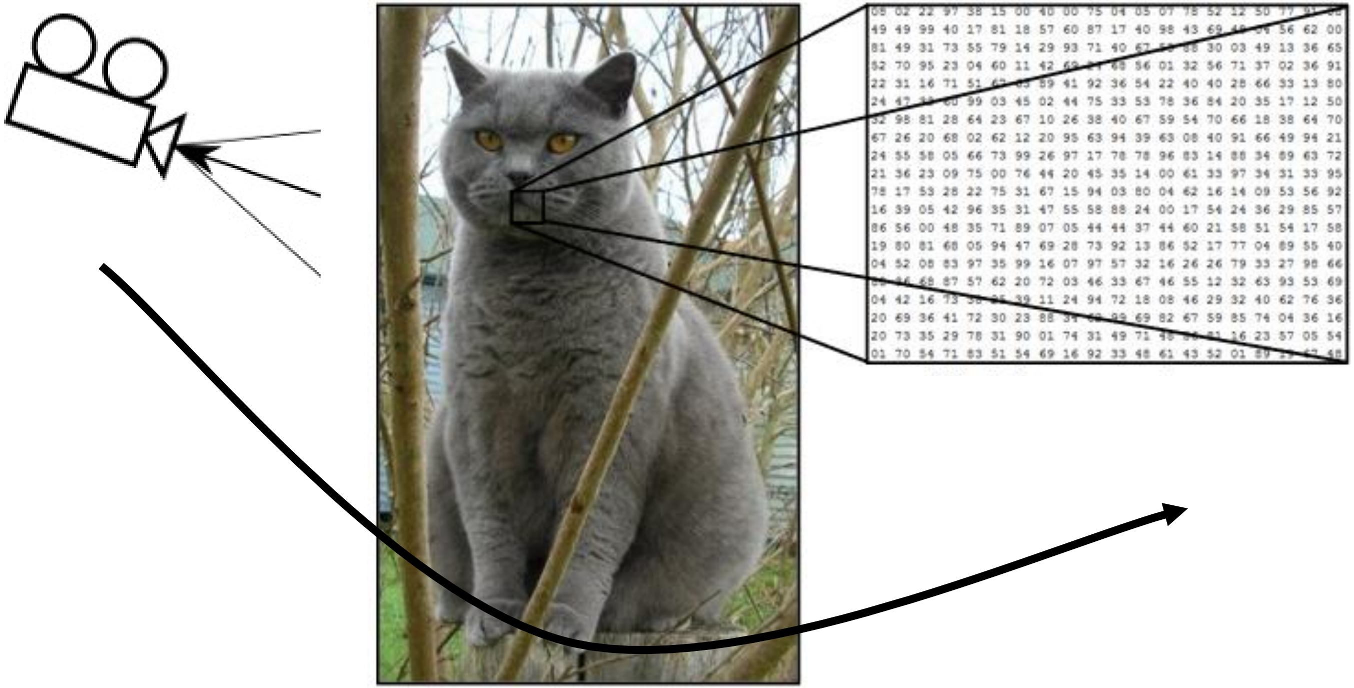
Images are represented as 3D arrays of numbers, with integers between [0, 255].

E.g.
300 x 100 x 3

(3 for 3 color channels RGB)



Challenges: Viewpoint Variation



Challenges: Illumination

Challenges: Deformation

Challenges: Occlusion

Challenges: Background clutter

Challenges: Intraclass variation

An image classifier

```
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hand-code the algorithm for recognizing a cat, or other classes.

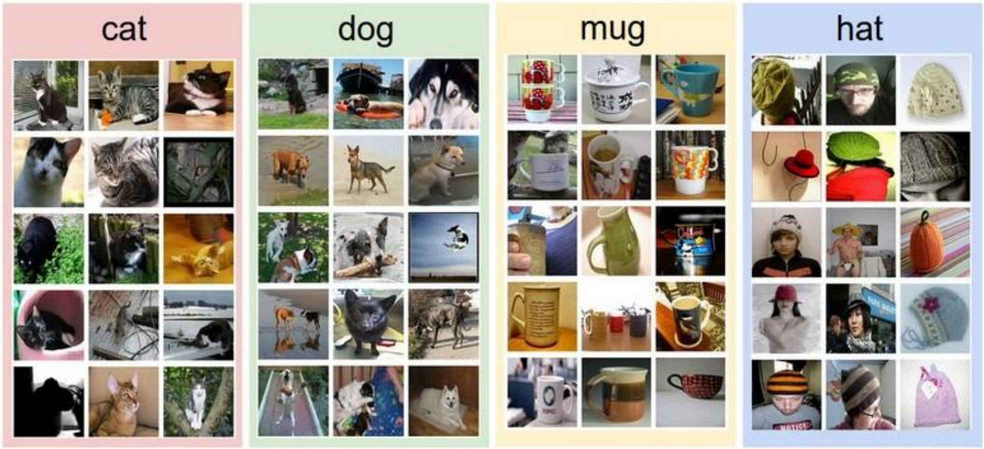
Data-driven approach:

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
```

Example training set



First classifier: **Nearest Neighbor Classifier**

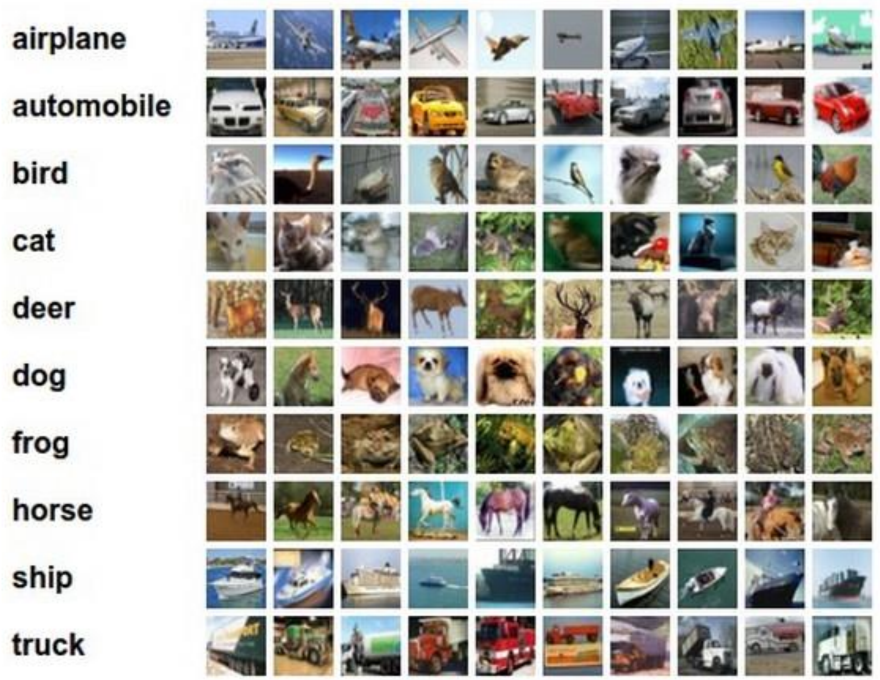
```
def train(train_images, train_labels):
  # build a model for images -> labels...
  return model

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

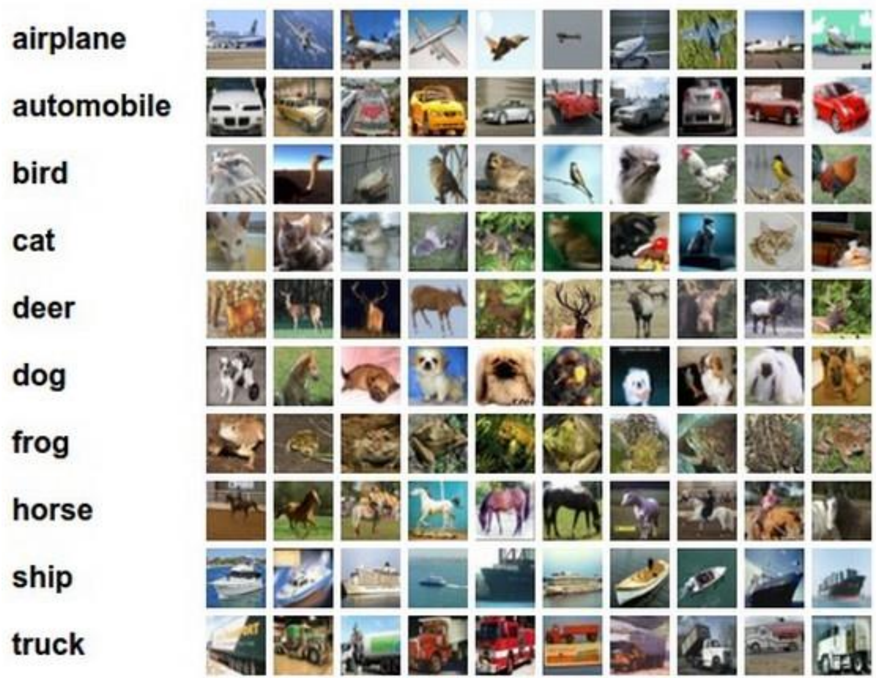
Remember all training images and their labels

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.



For every test image (first column), examples of nearest neighbors in rows

How do we compare the images? What is the **distance metric**?

L1 distance: $$d_1(I_1, I_2) = \sum_p \left| I_1^p - I_2^p \right|$$

test image, training image, pixel-wise absolute value differences

$$\left| \begin{bmatrix} 56 & 32 & 10 & 18 \\ 90 & 23 & 128 & 133 \\ 24 & 26 & 178 & 200 \\ 2 & 0 & 255 & 220 \end{bmatrix} - \begin{bmatrix} 10 & 20 & 24 & 17 \\ 8 & 10 & 89 & 100 \\ 12 & 16 & 178 & 170 \\ 4 & 32 & 233 & 112 \end{bmatrix} \right| = \begin{bmatrix} 46 & 12 & 14 & 1 \\ 82 & 13 & 39 & 33 \\ 12 & 10 & 0 & 30 \\ 2 & 32 & 22 & 108 \end{bmatrix} \xrightarrow{\text{add}} 456$$

Nearest Neighbor classifier

```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
```

Nearest Neighbor classifier

```
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
```

remember the training data

Nearest Neighbor classifier

```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
```

for every test image:

- find nearest train image with L1 distance
- predict the label of nearest training 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
```

Nearest Neighbor classifier

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

Do a poll...

```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
```

Nearest Neighbor classifier

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

The choice of distance is a **hyperparameter**
common choices:

L1 (Manhattan) distance

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euclidean) distance

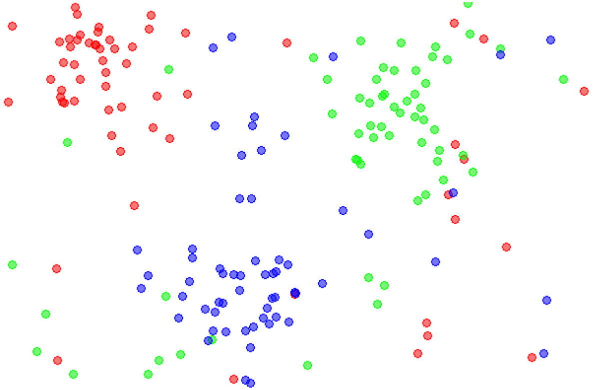
$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

k-Nearest Neighbor

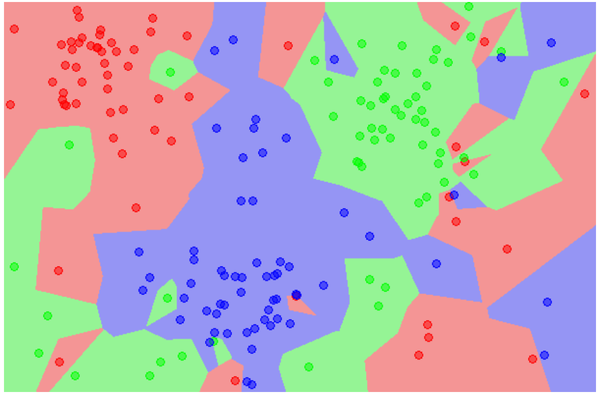
find the k nearest images, have them vote on the label

(Do a poll)

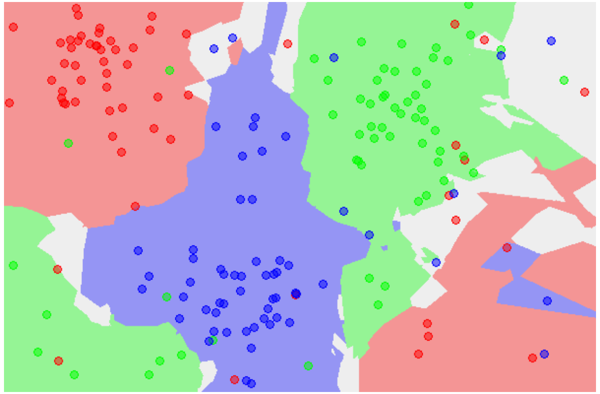
the data



NN classifier

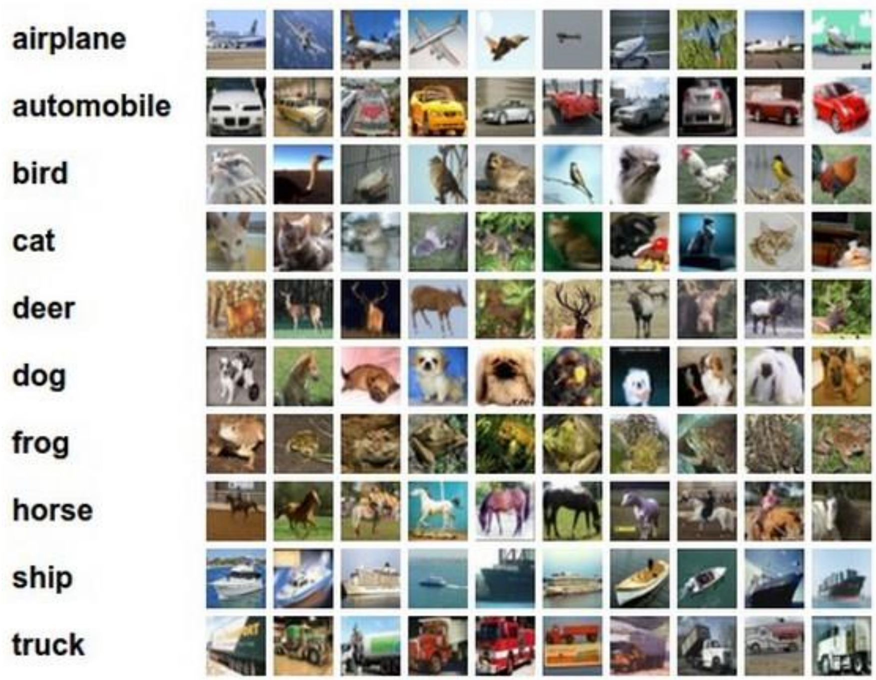


5-NN classifier



http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

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