Outline

Introduction to machine learning

Supervised learning

Unsupervised learning

Reinforcement learning
Problems that cannot be solved by typical programs

- Typically, programs are written by human in exact sequential order (what we have seen so far)

- However, it is hard to write programs that solve tasks like recognizing faces or speech:
  - We don’t know how to program it because we don’t know how our brains do it
  - Even if we had a good idea how our brain do it, the program will be horrendously complicated
Problems that cannot be solved by typical programs

Detecting disease-causing mutations

- We don’t know how to program it because we don’t fully understand the functions of our genome
- We have very limited understanding of the physiology underlying most of the complex phenotypes (e.g. Alzheimer’s disease, cancers) and how they interact with the environments (e.g., nutrition, exposed to radiation, neighbourhoods)
- There are unknown factors that we may not even observe or not yet have measurement to measure them in the real data that dictate the phenotypes
Introduction to machine learning: a data-driven approach

What we do in machine learning:

▶ We collect lots of data (e.g., genotype of over 1 million genetic variants and phenotypes for large population cohort)
▶ We develop a machine learning algorithm that take these individual data as “examples” or “training data” and automatically produces a program that does the job
▶ The program produced by the learning algorithm may look very different from a typical hand-written program. It may contain millions of numbers (e.g., one decimal number per mutation indicating their impact)
▶ If we succeed, the program will work on new data (“testing data”) that are not seen before (e.g., predicting risk of Alzheimer’s disease using genotype of a new individual)
▶ In practice, we need **two essential components**:
  1. An objective function that describe the problem (e.g., sum of errors in predicting diseases)
  2. An algorithm that optimizes the objective function (e.g., minimizing the errors disease prediction)
A classic example in machine learning:
recognizing hand-written digits

What makes a “2”?
First, how do we represent the data such as images: matrix

We represent each 28x28 image as a matrix or a two-dimensional list (see Lecture28.ipynb). Each entry in the 2D list is either 0 (‘.’) or number that is greater than 0 (‘@’)

```
from mnist import MNIST
import random
mndata = MNIST('mnist')
images, labels = mndata.load_training()
print(type(images))  # <class 'list'>
# display the first `2' in the data
for index, digit in enumerate(labels):
    if digit == 2:
        break
print(mndata.display(images[index]))
```
Outline

Introduction to machine learning

Supervised learning

Unsupervised learning

Reinforcement learning
Classification

- Outputs or labels are categorical (1-of-N) (e.g., 10 digits)
- Inputs: feature (e.g., flattened $28 \times 28$ image = 784 input features)
- Goal: select the correct class for the new inputs (e.g., correctly classify the new image into one of the 10 digits)

How to represent labels? Use “one-hot encoding”: create a vector as long as the number of categories we have, and set exactly one of the positions in the vector to 1 and the rest to 0.

```python
# This is the one-hot version of: [5, 0, 4, 1, 9]

[[0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

Classifying MNIST hand-written image with neural network

Classification of input data into one of the $K$ categories:

Simple math behind neural network (not required to understand in this class):

$$ h_j^{(1)} = \sum_{i=1}^{784} w_i^{(1)} x_i; $$

$$ h_j^{(m)} = \sum_{i=1}^{H_m} w_i^{(m)} h_i^{(m-1)} $$

$$ \hat{y}_k = \frac{\exp(-\sum_{i=1}^{M=128} w_k a_i)}{\sum_{k'=1}^{10} \exp(-\sum_{i=1}^{M=128} w_{k'} a_i)}; $$

error = $y_k \log \hat{y}_k + (1 - y_k) \log(1 - \hat{y}_k)$
Regression

- Outputs are continuous
- Inputs: feature (continuous or discrete)
- Goal: predict outputs accurately for new inputs

A toy example (see Lecture28.ipynb):
- Output (y): a continuously measured variable
- Input (x): another continuously measured variable
- Task: fitting a line as $y = wx + b$ by estimating $w$ and $b$
Example: predict transcription factor binding affinity

Transcription factors (TF) are proteins that bind to specific regions of the genome to regulate nearby gene expression.

Goal: predict the TF binding affinity based on the DNA sequence.

Input: representing DNA sequence as 2D matrix:

```
  NNNATGCAGCANNNN
```

Matrix representation of DNA sequence (darker = stronger)
Convolution – extracting feature maps

Applying 4 bp sequence filter along the DNA matrix:

ATGCAGCA

on 1st position

Yellow = high activity; blue = low activity
Predicting transcription factor binding affinity

ChIP-seq, PBM, SELEX Experiments

DNA sequence

\[ \text{ATGCAGCA A T G C A G C A N N N} \]

<table>
<thead>
<tr>
<th>A</th>
<th>T</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>G</th>
<th>C</th>
<th>A</th>
<th>N</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Convolution module

<table>
<thead>
<tr>
<th>GCRC</th>
<th>match</th>
<th>filter</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>TGRT</td>
<td>match</td>
<td>filter</td>
<td>max</td>
</tr>
<tr>
<td>ATRc</td>
<td>match</td>
<td>filter</td>
<td>max</td>
</tr>
</tbody>
</table>

Prediction module

<table>
<thead>
<tr>
<th>GCRC</th>
<th>GCRC</th>
<th>ATRc</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>T</td>
<td>R</td>
<td>C</td>
</tr>
</tbody>
</table>

DNA sequence

<table>
<thead>
<tr>
<th>ATGCAGCA</th>
<th>N</th>
<th>N</th>
</tr>
</thead>
</table>

ChIP-seq, PBM, SELEX Experiments

Individual motifs

- GCRC
- TGRT
- ATRc

Affinity

Higher-level combinations

...
Predicting disease mutations [Alipanahi 2015]
Example 3: predict phenotype by genotype

Promises of genomic medicine:
1. Disease mechanisms
2. Novel target genes
3. Novel therapeutics
4. Personalized medicine
Outline

Introduction to machine learning

Supervised learning

Unsupervised learning

Reinforcement learning
Unsupervised learning
Clustering
Modeling sequential data
Latent topic models
Autoencoder: neural net image compression [Hinton 2006]

Pretraining

Unrolling

Fine-tuning

Random samples of curves from data sets.

Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used to initialize the next layer in the stack.

1. **Top RBM**
   - 30
   - 500
   - 1000
   - 2000

2. **RBM**
   - 500
   - 1000
   - 2000

3. **Lower RBM**
   - 1000
   - 2000

Error derivatives. The whole system is trained by backpropagation of error derivatives. Figure 2 shows random samples from the test data set; reconstructions by 30-dimensional PCA. The average squared errors are 126 and 135.

By logistic PCA and standard PCA. The average squared error per image is 1.44, 7.64, 2.45, 5.90. The average squared errors for the last three rows are 3.00, 8.01, and 13.87.

Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used to initialize the next layer in the stack.
GAN
Outline

Introduction to machine learning

Supervised learning

Unsupervised learning

Reinforcement learning
Reinforcement learning