Lecture 11: Perceptron and Gradient Descent (Mitchell Chapter 4)

CS 167: Machine Learning

Extra Credit Opportunity

Prof. Tianbao Yang, University of Iowa

Deep Learning with Big and Small Data

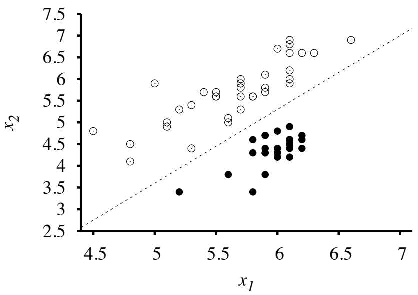
Deep learning has brought tremendous success in many areas with the help of big data and super computing. In this talk, I will present the state of art results of deep learning for image classification. I will also talk about our recent research on how to learn a deep convolutional neural network for fine-grained image classification where big labeled data is difficult to be obtained.

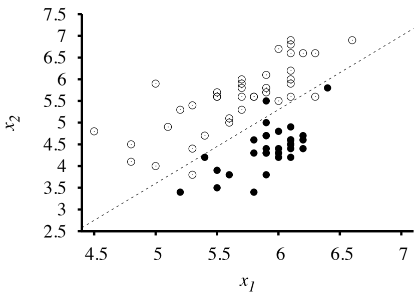
Friday, October 21, in Meredith 106 at 2:00pm.

Earn 3 extra credit (homework) points for

- attend the talk
- write up a paragraph on something you learned

Linear Separation



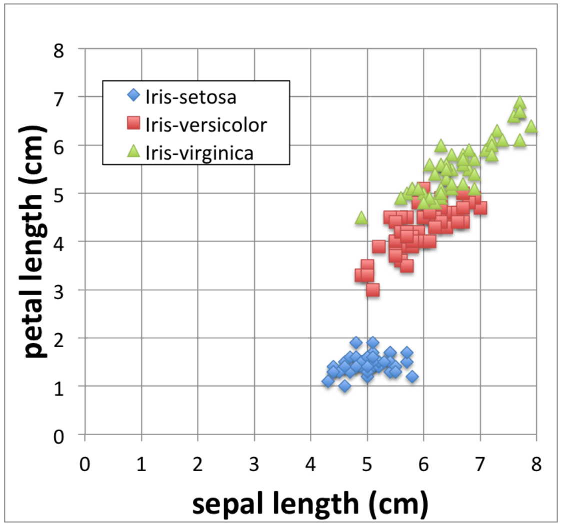


x_1: body wave magnitude, x_2: surface wave magnitude
white: earthquakes, black: underground explosions

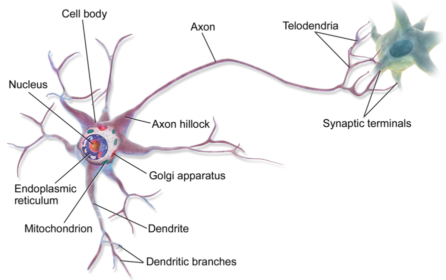
2D data: separate with a line **What do equations of lines look like?**
3D data: separate with a plane **What do equations of planes look like?**
***N*D data**: separate with a hyperplane **What do equations look like?**

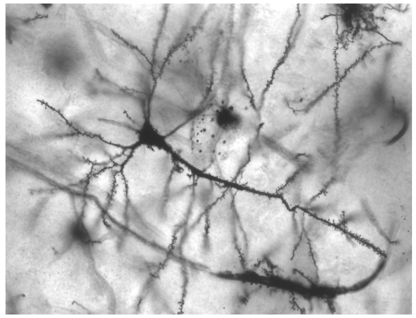
Discussion: Iris Data

Can we use a linear separator as our hypothesis on this data?



Biological Motivation

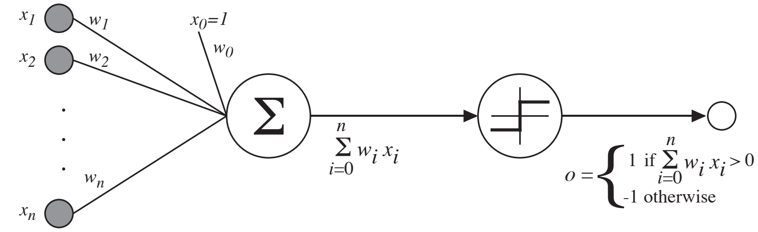




https://commons.wikimedia.org/wiki/Neuron#/media/File:Pyramidal_hippocampal_neuron_40x.jpg
https://en.wikipedia.org/wiki/Neuron#/media/File:Blausen_0657_MultipolarNeuron.png

- dense network, 10^{11} neurons
- each on average connected to 10^4 others
- neuron switching times $<$ .001 seconds - slow
- fast recognition $\Rightarrow$ highly parallel brain
- computer switching times $\approx$ .0000000001 seconds, can't handle complexity of brain

Perceptron



$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Let's train a perceptron

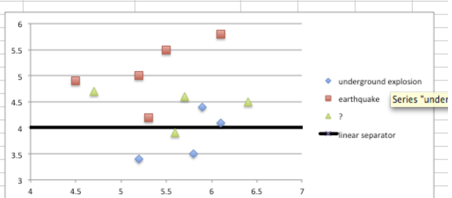
Exercise: *Use the earthquake spreadsheet to come up with the equation of some lines that separate the data.*

Seismometer Data

Body Wave Magnitude	Surface Wave Magnitude	Classification	Result of Weighting Function	After activation function	Prediction (-1 = earthquake, +1 = explosion)
5.2	3.4	underground explosion	0.6	1	underground explosion
5.8	3.5	underground explosion	0.5	1	underground explosion
5.9	4.4	underground explosion	-0.4	-1	earthquake
6.1	4.1	underground explosion	-0.1	-1	earthquake
5.2	5	earthquake	-1	-1	earthquake
4.5	4.9	earthquake	-0.9	-1	earthquake
5.3	4.2	earthquake	-0.2	-1	earthquake
5.5	5.5	earthquake	-1.5	-1	earthquake
6.1	5.8	earthquake	-1.8	-1	earthquake
5.6	3.9	?	0.1	1	underground explosion
5.7	4.6	?	-0.6	-1	earthquake
4.7	4.7	?	-0.7	-1	earthquake
6.4	4.5	?	-0.5	-1	earthquake

w_offset	4	<-- adjust to change offset
w_x	0	<-- adjust to change x weight
w_y	-1	<-- adjust to change y weight

x	y	
0	4	<-- don't change these, updated automatically
10	4	<-- don't change these, updated automatically



Perceptron training rule

Feed a training example into the perceptron, then for each weight w_i

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., 0.1) called **learning rate**

Exercise: What happens to the weight in each of these cases:

- t and o are both 1
- t and o are both -1
- t is 1, o is -1
- t is -1, o is 1

How to train your perceptron

Do the following over and over and over

1. feed in a training example
2. update weights with weight update rule

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small

Another similar approach

If the data isn't linearly separable, use the **Gradient-Descent** training rule:

Use *unthresholded* output (don't push it to 1 or -1)

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Sum up Δw_i for all training *examples* before updating w_i

Gradient-Descent Algorithm

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i$$

Summary

Perceptron training rule guaranteed to succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training rule uses gradient descent

- Guaranteed to converge to hypothesis with *minimum squared error* (see proof in textbook - uses partial derivatives)
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not separable by H

And just to make it more confusing, if you update weights one at a time for each training example but use an *unthresholded* unit, that's one way to do it too called **stochastic gradient descent**.

Let's do this with scikit-learn

Add this to the code you were using with the Iris data set last time

```
from sklearn.linear_model import Perceptron

perc = Perceptron()

perc.fit(iris_train[predictors],iris_train['species'])
iris_perc_predictions = perc.predict(iris_test[predictors])
print(metrics.accuracy_score(iris_test['species'],\
                             iris_perc_predictions))
```

Is this the accuracy you expected? Why does it perform this way?

Pandas get_dummies

Let's do a transformation to our original Iris data:

```
iris_dummies = pandas.get_dummies(iris_data)
print(iris_dummies)
```

Discuss: What did that just do?

Exercise: Create Perceptron classifiers for each of viginica, setosa, and versicolor. Evaluate each classifier's performance and explain why they performed as they did.

Review: Regression vs. Classification

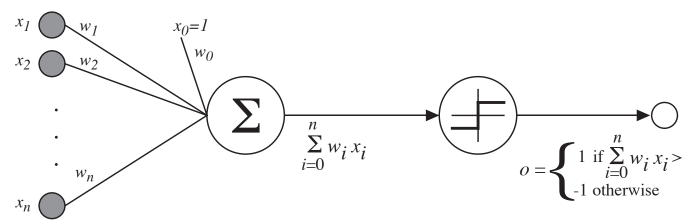
Classification problems: predict which category an examples goes in, examples:

- predict the iris species
- predict whether or not the passenger survived
- predict the acceptability level of a car
- predict creditworthiness
- predict a person's sex from their photo

Regression problems: predict a numerical value, examples:

- predict a car's fuel efficiency in MPG
- predict a person's age from their photo

Group Discussion



Weight update rule:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

How can we adapt perceptron, gradient descent, and/or stochastic-gradient-descent so that it does regression instead of classification?

Trying out regression in scikit-learn

Exercises:

- read in the Auto MPG data set (note: we will predict the `mpg` column)
- fill in missing values for `horsepower` column
- split into test/train
- try to fit a perceptron learner to the data, **What happens?**, **Is this expected?**
- visualize some of the attributes compared to the `mpg` column:

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.scatter(mpg_data['weight'],mpg_data['mpg'])
plt.show()
```

  Does it seem like we should be able to fit a line to this data?

Regression on MPG Data

Find a regression algorithm based on perceptron, gradient descent, or stochastic gradient descent

http://scikit-learn.org/stable/modules/classes.html

Exercises:

- Use it to make predictions on your test set. **How do these look?**
- Determine what happens when you try to measure the accuracy, explain why it does this

Common Regression Metrics

error(x) on an example x: (absolute value) difference between *predicted* and *actual* value for that example

Mean Absolute Error: find the mean of *error*(x) over all examples x from the test set

Mean Squared Error: find the mean of $(error(x))^2$ over all examples x from the test set

Exercise: *Find the `scikit-learn` functions for measuring these, use them on your earlier predictions*

Improving Performance

Weight update rule:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = \eta(t - o)x_i$$

We know there are at least two parameters of this algorithm that can affect performance

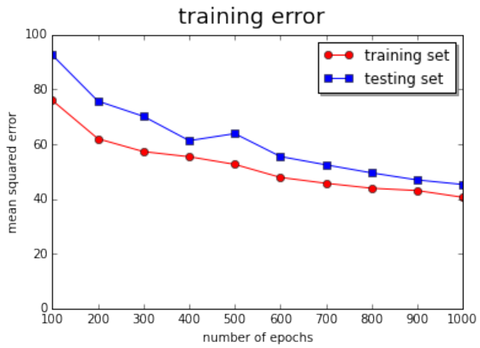
- η, the learning rate
- the number of *epochs*, i.e., the number of times you feed the set of training examples (and update the weights) through the perceptron unit during training

What are these values set to by default?

Exercise: *Find an η and number of epochs that seem to give you decent results.*

Exercise: Make a Plot

Make a plot like this that shows how the performance on the test and training set is affected by the number of epochs



At what point does the MSE stop decreasing?

Exercise: Regression with k-Nearest-Neighbor

Exercise:

- find a way to do regression with Random Forests or k-Nearest-Neighbor in scikit-learn
- comparetheir performance against the Perceptron/GD/SGD method