Machine Learning With Python

Bin Chen
Nov. 7, 2017
Research Computing Center
Outline

- Introduction to Machine Learning (ML)
- Introduction to Neural Network (NN)
- Introduction to Deep Learning NN
- Introduction to TensorFlow
- A little about GPUs
Motivation

- Statistical Inference
- Statistical Learning
- Machine Learning
- Deep Learning
- Artificial Intelligence
- Big Data
  - Super Computer
  - fuel
Supervised VS Unsupervised learning
Regression VS Classification
Linear VS Nonlinear Regression
Binary VS Multivariate Classification.
Clustering (e.g., K-Means)
Support Vector Machine (SVM)
Neural Network, Deep Neural Network
Machine Learning (p2)

- Regression:
  
  Predict the price of a house.

- Binary classification $y = [0,1]$:
  
  Online advertisement. (will this customer hit this AD?)

- Multivariate classification
  
  - Digit recognition $y = [0,1,2,3,4,5,6,7,8,9]$
  
  - Image recognition (is this a cat?)
Machine Learning (p3)

- **Structured data:**
  - Data like tables with records,
  - say, predicting house price, loan approvals.

- **Unstructured data:**
  - Images, Audios.
  - human’s natural perceptions often do a great job with accuracy close to Bayes error.

- ML has beaten human beings on many structured data
  - Amazon’s recommended list of books

- Deep learning is doing the same thing for unstructured data.
  - Autonomous driving
  - Natural language processing (NLP)
Deep learning is a subset of machine learning.

The statistics is essentially the same, e.g.,

- loss/cost function (minimize the cost)
- training/dev/test set
- bias-variance tradeoff
- model tuning/regularizing (hyper-parameters)

Details differ, and there are new concepts, e.g.,

- activation function (sigmoid, ReLU)
- gradient descent (momentum, RMSprop, AdamOptimizer)
- forward/backward propagation (vanishing/exploding gradient)
- dropout, batch normalization.
Am I under/over-fitting my data (Bias-Variance tradeoff)?

(source: Hastie, Tibshirani, & Friedman, text book E.S.L)
Machine Learning (p6)

- Training/Dev/Test splitting of data

(Traditional Machine Learning)

Train ~60%  Dev ~20%  Test ~20%

(Deep Learning)

Train ~98%  Dev ~1%  Test ~1%

(Deep Learning with Mis-Matched Data)

Train ~78%  Train-Dev 20%  Dev 1%  Test 1%
What Drives Deep Learning? (p1)

- Scale-Performance Relationship

![Graph showing the relationship between amount of data and performance for different types of neural networks and SVM, regressions, etc.]

- Performance vs. Amount of Data
  - Large NN
  - Medium NN
  - Small NN
  - SVM, regressions, etc.
What Drives Deep Learning? (p2)

- The amount of data available
- The amount of computation
  - The width and depth of the network
- Progress in algorithm design
  - Activation function (from sigmoid to ReLU)
    - from SNN, to CNN, RNN, etc.
- The computing power of modern hardware
  - E.g., Graphics Processing Units (GPUs)
Outline

- Introduction to Machine Learning (ML)
- Introduction to Neural Network (NN)
- Introduction to Deep Learning NN
- Introduction to TensorFlow
- A little about GPUs
From Regression to Neural Network (p1)

\[ y = wx + b \]

Standard linear regression

Size of house vs. Price

- Data points plotted on a scatter plot
- Linear regression line drawn through the points
Q1. So can I consider my simple linear regression as a neural network?

Answer: Yes, sort of.

It is a single-layer network, with activation function \( g(x) = x \)

Such simplistic activation function is almost never used.
From Regression to Neural Network (p3)

\[ y = f(x_1, x_2, x_3, x_4) \]

Still regression!

Neural network with one hidden layer
What is a neural network? (p1)

- Q1. How many layers are there?
- Q2. How many hidden units?
- Q2. Is it a deep neural network?
- Q3. What does the arrow mean?

(Picture from Wikipedia)
What is a neural network? (p2)

- Q1. How many layers are there?

- Q2. How many hidden units?
  - A2: 4.

- Q3. Is it a deep neural network?
  - A4: no! (>=2 hidden layers)

- Q4. What does the arrow mean?
  - A4: flow of data (tensorflow)

(Picture from Wikipedia)
What is a neuron? (p1)

(Picture from Wikipedia)
What is a neuron? (p2)

A neuron does simple and specific task: an affine transformation composed with an activation function.

(Pay attention to the naming of each variables: z, w, a, b, etc.)
Activation function

- Activation function adds **non-linearity** to your network.
- Popular activation functions include, sigmoid, tanh, ReLU
- Different layers of can use different activation function.

\[
a = \frac{1}{1 + e^{-z}}
\]

**Sigmoid**

\[
a = \tanh(z)
\]

**ReLU**

**Leaky ReLU**
Logistic Regression VS Neural Network

- The sigmoid activation function was also used in logistic regression in traditional statistical learning.
- Logistic regression is simple Neural Network with sigmoid activation function.

\[ a = \hat{y} = \frac{1}{1 + e^{-(w^T x + b)}} \]

\[ z = w^T x + b \]

\[ a = \sigma(z) \]
Loss Function and Cost Function

- **The Loss function** $L(\hat{y}_i, y_i)$ tells how well your model fits a data point (here $i$ labels the data point).

- **Cost Function $J$** is the average of the loss function over the sample.

- **Binary Classification** as an example

  \[ L(\hat{y}_i, y_i) = -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \]

  \[ J = \frac{1}{m} \sum_{i=1}^{m} L(\hat{y}_i, y_i) \]

- **Chi-square** for regression analysis as another…

  \[ J = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \]
Why we need the **Loss function, or the cost function**?

- Answer: we need them to determine the model parameters
- To train the NN we optimize the cost via **gradient descent**.
Gradient Descent

- Given labeled data \((x_i, y_i)\), find the parameters \((W_{jk}, b_j)\) by minimizing the cost function \(J\).
- Method: gradient descent

\[
\theta_j := \theta_j - \alpha \frac{\partial J}{\partial \theta_j}
\]

(\(\alpha\) is the learning rate)

(From Andrew Ng’s Lecture Notes)
Outline

▪ Introduction to Machine Learning (ML)
▪ Introduction to Neural Network (NN)
▪ Introduction to Deep Neural Network
▪ Introduction to TensorFlow
▪ A little about GPUs
Deep Neural Network

- A neural network with at least 2 hidden layers
- The hidden layers can be very wide (millions of hidden units)
- The width (# of units) varies from layer to layer.

A 4-layer deep neural network
Forward and Backward Propagation

- **Forward propagation**: given labeled data \((x_i, y_i)\), and parameters \((W, b)\) compute the cost function \(J\).

- **Backward propagation**: compute the derivatives of cost function w.r.t the model parameters. Update the model parameters \((W, b)\).
Compute the Derivatives

- Using **binary classification** an example

  \[ L(\hat{y}_i, y_i) = -[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \]

  \[ \Rightarrow \frac{\partial L}{\partial \hat{y}} = -\frac{y_i}{\hat{y}_i} + \frac{1 - y_i}{1 - \hat{y}_i} \]

- Assuming **sigmoid activation function**

  \[ \hat{y} = a = g(z) = \frac{1}{1 + e^{-z}} \Rightarrow \frac{\partial a}{\partial z} = a(1 - a) \]

- Derivatives for the affine/linear transformation is easy

  \[ \hat{z} = W\bar{x} + \bar{b} \Rightarrow \frac{\partial z_i}{\partial W_{ij}} = x_j, \frac{\partial z_i}{\partial b_j} = \delta_{ij} \]

- Now using **chain rule** to concatenate the above together.
The computation graph for \( J = 3(a + bc) \)

This really helps when you think about forward/backward propagation.

Understand/Stick with a good notation is also critical.
Parameters VS Hyper-parameters

- **Parameters**: \((W, b)\) for each layer of the NN.
  
  \((W, b)\) can be learned by training the NN using the training data set.

- **Hyper-parameters** include:
  
  1. # layers for the NN;
  2. # units for each layer;
  3. # learning rate \(\alpha\);
  4. the choice of activation function.
  5. batch data size.
  6. # iteration for convergence.

- Deep learning tends to have many more hyper-parameters than normal ML methods.

- Hyper-parameters are determined via the dev data set.
Parameters VS Hyperparameters (p2)

- Choosing between other machine learning methods and deep leaning can be empirical.
- Large number of hyper-parameters make deep learning very empirical.
Outline

- Introduction to Machine Learning (ML)
- Introduction to Neural Network (NN)
- Introduction to Deep Learning NN
- Introduction to TensorFlow
- A little about GPUs
Introduction to TensorFlow (p1)

- A framework (library/package) for deep learning.
- Open source (originally by Google Brain Team).
- Python/C++ frontend, and C++ backend.
- Support hardware accelerators GPU.
- Current stable release v1.3
How does TensorFlow work?

- User defines the **architecture** of the NN (*inference graph*).
- User defines the **loss/cost function** (*train graph*).
- User provides the **data** (train/dev/test).
- User chooses the **optimizer** to try.
- User picks hyper-parameters (mini-batch size, learning rate).
- **Tensorflow** does the rest automatically for you.

  - forward propagation to compute the loss function;
  - backward propagation to compute the derivatives;
  - many optimization algorithms are included
  
    (e.g., `tf.train.GradientDescentOptimizer()`,
    `tf.train.AdamOptimizer(...)`)
A Toy Example (ex01)

- Goal: train a toy Neural network with loss function

\[ L(w) = w^2 - 12w + 36 \]

- Here \( w \) is the only parameter to learn.
- The training output should be very close to 6.
- Sorry (no input at all, but will add later on).
A Toy Example (ex01)

In [1]:
```python
import tensorflow as tf
import numpy as np
```

In [2]:
```python
# cost function J = w**2 - 12*w + 36
# optimized w should be 6.

w = tf.Variable(0, dtype=tf.float32)
J = w**2 - 12*w + 36  # operator overloading
train = tf.train.GradientDescentOptimizer(0.01).minimize(J)
```

In [3]:
```python
# you must always create a Session, and initialize your variables
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
```

In [4]:
```python
# before training, w = 0.0
print(session.run(w))

# train with 1000 iteration
for i in range(1000):
    session.run(train)

# now the w should be very close to 5 now
print(session.run(w))
```
```
0.0
5.99999
```
Loss function $L = x_0 w^2 - x_1 w + x_2$

In [2]:
# data x is defined as placeholder
# variables is trainable, placeholders are not!
x = tf.placeholder(tf.float32, [3,1])

w = tf.Variable(0, dtype=tf.float32)
J = x[0] * w**2 + x[1] * w + x[2]  # operator overloading
train = tf.train.GradientDescentOptimizer(0.01).minimize(J)

In [3]:
# you must always create a Session, and initialize your variables
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)

In [4]:
# this will be my data "x"
coeffs = np.array([[1], [-12], [36]])

# train with 1000 iteration
for i in range(1000):
    session.run(train, feed_dict={x:coeffs})

# now the w should be very close to 5 now
print(session.run(w))

5.99999
Example-02: Linear Regression

- Mysterious equation: \( y = 0.2x + 0.5 + \varepsilon \)
- Model: \( y = wx + b \)
- Goal: given enough \((x_i, y_i)\) pairs, find out \((w, b)\).
Example-02: Linear Regression (p2)

- Generate the data: \( y = 0.2x + 0.5 + \varepsilon \)

```python
In [1]:
import tensorflow as tf
import numpy as np
import matplotlib as mpl
%matplotlib inline

In [2]:
# y = 0.2*x + 0.5 + epsilon
x_data = np.random.rand(100,1)
epsilon = 0.01*np.random.randn(100,1)
y_data = 0.2*x_data + 0.5 + epsilon
plt.plot(x_data, y_data,'.')
```
Define the model and the loss function, train it:

```
In [4]:
# syntax: tf.Variable(<initial-value>, name=<optional-name>)
w = tf.Variable(1, name='weight', dtype=tf.float32)
b = tf.Variable(0, name='bias', dtype=tf.float32)
y = w*x_data + b          # note the overloading and broadcasting
# loss function J
J = tf.reduce_mean((y - y_data)**2)
train = tf.train.GradientDescentOptimizer(0.25).minimize(J)
```

```
In [5]:
# train the model
session = tf.Session()
init = tf.global_variables_initializer()
session.run(init)
y_init = session.run(y)  # y prediction with untrained w, b
for i in range(5000):
    session.run(train)
print(session.run([w,b]))
```

```
[0.2023287, 0.49739757]
```
Example-02: Linear Regression (p4)

- Visualize the training out:

```python
In [6]:
    pl.plot(x_data, y_data, '.', color='r')
    pl.plot(x_data, y_init, '.', color='g')
    pl.plot(x_data, session.run(y), '.', color='b')

Out[6]: [<matplotlib.lines.Line2D at 0x1149e7908>]
```
Example-03: digit recognition (p1)

- Goal: given enough images and labels, find the weights, biases to identify digits.
- Dataset: MNIST dataset: http://yann.lecun.com/exdb/mnist/
- Ref: https://www.tensorflow.org/get_started/mnist/beginners
- Image size: 28*28=784, so $x[784, m]$, $y[10, m]$
Example-03: digit recognition (p2)

- Model: simple 1-layer neural network.
- Activation function:
  \[
  \text{softmax}(x)_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}
  \]
Example-03: digit recognition (p3)

- Cross entropy loss function

\[ L(y^{(i)}, \hat{y}^{(i)}) = -\sum_{j=1}^{10} y_j^{(i)} \log \hat{y}_j^{(i)} \]

- Cost function

\[ J = \frac{1}{m} \sum_{i=1}^{m} L(y^{(i)}, \hat{y}^{(i)}) \]

- One-hot vector
Example-03: digit recognition (p4)

- Import the data, and define the model

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

def myfunc():

    data_dir = '/Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/mnist'
    mnist = input_data.read_data_sets(data_dir, one_hot=True)

    # Create the model
    x = tf.placeholder(tf.float32, [None, 784])
    W = tf.Variable(tf.zeros([784, 10]))
    b = tf.Variable(tf.zeros([10]))
    y = tf.matmul(x, W) + b
    y_ = tf.placeholder(tf.float32, [None, 10])
```
Example-03: digit recognition (p5)

- Define the loss function (`cross_entropy`), and train the model

```python
import tensorflow as tf

# Define the loss function

cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y))

train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

init = tf.global_variables_initializer()

tf.Session()

tf.Session.run(init)

# Train the model

for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    tf.Session.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

# Test trained model

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))

accuracy = tf.reduce_mean(
    tf.cast(correct_prediction, tf.float32))

print("The accuracy on test data is ",
      tf.Session.run(accuracy, feed_dict={x: mnist.test.images,
                                          y_: mnist.test.labels})))
```
Example-03: digit recognition (p5)

- Accuracy on test data: ~91%

```python
myfunc()
Extracting /Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/workshop/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting /Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/workshop/mnist/input_data/t10k-images-idx3-ubyte.gz
Extracting /Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/workshop/mnist/input_data/t10k-labels-idx1-ubyte.gz

The accuracy on test data is 0.9171
```
Example-03 Improved (p1)

- Goal: MNIST, but with deep network, want higher accuracy
- 3 hidden layers with ReLU, output layer softmax

A 3 hidden layer deep neural network for MNIST
Example-03 Improved (p2)

- Goal: MNIST, but with deep network, want higher accuracy

```python
# Create the model
x = tf.placeholder(tf.float32, [None, 784])

W1 = tf.Variable(tf.truncated_normal([784, 100], stddev=0.1))
b1 = tf.Variable(tf.zeros([100]))

W2 = tf.Variable(tf.truncated_normal([100, 60], stddev=0.1))
b2 = tf.Variable(tf.zeros([60]))

W3 = tf.Variable(tf.truncated_normal([60, 30], stddev=0.1))
b3 = tf.Variable(tf.zeros([30]))

W4 = tf.Variable(tf.truncated_normal([30, 10], stddev=0.1))
b4 = tf.Variable(tf.zeros([10]))

y1 = tf.nn.relu(tf.matmul(x, W1) + b1)
y2 = tf.nn.relu(tf.matmul(y1, W2) + b2)
y3 = tf.nn.relu(tf.matmul(y2, W3) + b3)
y = tf.matmul(y3, W4) + b4
y_ = tf.placeholder(tf.float32, [None, 10])
```
The accuracy increases from ~91% to ~97%

Note tensorflow automatically used all 4 cores of my laptop

```python
tic_wall = timeit.default_timer()
tic_cpu = time.clock()
myfunc()
toc_wall = timeit.default_timer()
toc_cpu = time.clock()
print("the cpu time is %9.5f seconds" % float(toc_cpu - tic_cpu))
print("the wall time is %9.5f seconds" % float(toc_wall - tic_wall))
```

Extracting /Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/workshop/mnist/input_data/train-labels-idx1-ubyte.gz
Extracting /Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/workshop/mnist/input_data/t10k-images-idx3-ubyte.gz
Extracting /Users/binchen/Desktop/RCC/MachineLearn/tensorflow/examples/workshop/mnist/input_data/t10k-labels-idx1-ubyte.gz

The accuracy on test data is 0.9764
the cpu time is 79.76172 seconds
the wall time is 22.18648 seconds
One Page about Python on HPC

- Python 2.7 and Python 3.5 are available on HPC nodes.
- Popular packages such as numpy, scipy, matplotlib are preinstalled.
- Anaconda python with ~200 packages including tensorflow is available at
  
  /panfs/storage.local/opt/python/anaconda/bin/python

- Users are encouraged to install packages to their own disk space via the python virtual environment:
  
  https://rcc.fsu.edu/software/python
One Page about GPUs on HPC

- Hardware upgrade from Tesla M2050 to GeForce1080 Ti.
- Compute capability from 2.0 to 6.1 (*Fermi to Pascal*)
- CUDA driver upgraded from 6.5 to 9.0
- Each compute node with GPUs have 4 GPU cards

https://rcc.fsu.edu/software/cuda

```
1. #!/bin/bash
2.
3. #SBATCH -N 1
4. #SBATCH -n 1
5. #SBATCH -J "cuda-job"
6. #SBATCH -t 4:00:00
7. #SBATCH -p backfill
8. #SBATCH --gres=gpu:1
9. #SBATCH --mail-type=ALL
10.
11. # load the cuda module to set up the environment
12. module load cuda
13.
14. # the following line should provide the full path to the cuda compiler
15. which nvcc
16.
17. # execute your cuda executable a.out
18. srun -n 1 ./a.out <input.dat >output.txt
```
A Little about Convolution

- From fully connected to partially connected.
- Convolution adds locality back.
- Convolution reduce the parameter size significantly.

Picture from: Martin Gorner
A Little about Convolution (p2)

- Structure of the ILSVRC-2012 competition winner

(Alex Krizhevsky, Ilya Sutskever Geoffrey E. Hinton 2012 paper)