Lab: CudaVision – Learning Vision Systems on Graphics Cards (MA-INF 4308)

Assignment 2

6.6.2016

Prof. Sven Behnke
Dr. Seongyong Koo

universitätbonn ais

Logistic Regression Classifier with MNIST dataset

- **Goal: How to train a network model using TensorFlow?**
  - We will learn with a simple example, Linear regression.

- **Let's use a real dataset, MNIST**
  - We will learn how to download and visualize the data

- **The first simplist linear classifier,**
  - What is the Logistic Regression Classifier?
  - We will learn equations for the classifier

- **Your task is,**
  - Training a Logistic Regression Classifier with MNIST dataset using TensorFlow

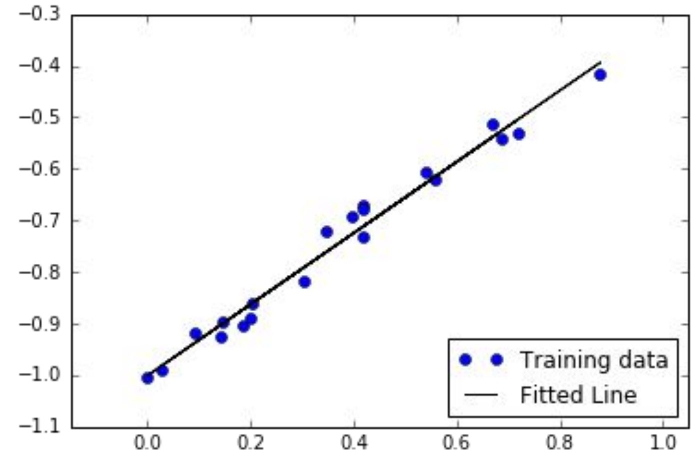
[http://deeplearning.net/software_links/]

Linear Regression

- Input: $\mathbf{X} \in \mathbb{R}^{N \times D}$ output: $\mathbf{y} \in \mathbb{R}^{N}$

- Estimate the linear relation between input and output such that minimizing the loss function

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{n=1}^{N} \left(\sum_{d=1}^{D} x_{n,d} w_d - y_n \right)^2$$



- There is an analytic way to determine the unique solution for w, however, we will concentrate on the case where X is unknown (i.e. becomes available during runtime) or too large to fit into memory.

universitätbonn ais

Linear Regression

- The way to solve this numerically is using the gradient decent algorithm. Because the loss function of linear regression is a convex quadratic function, it only has one global optima.

- First select initial parameters $\mathbf{w}_0$, then iteratively calculate gradient

$$\Delta w_i = \frac{\partial l(\mathbf{X}, \mathbf{y}, \mathbf{w})}{\partial w_i}$$

, and update $\mathbf{w}$ using gradient descent: $\mathbf{w} := \mathbf{w} - \eta \Delta \mathbf{w}$

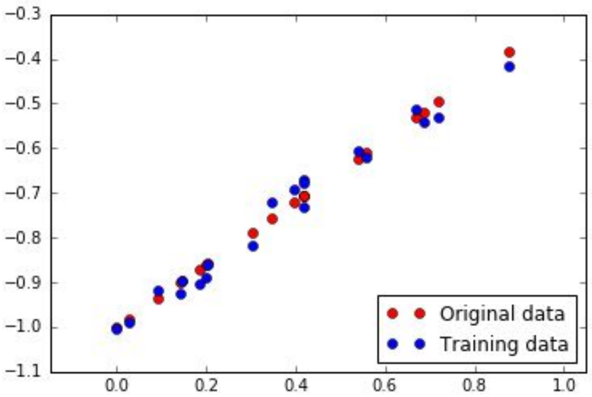
, where $\eta < 1$ is a (typically small) learning rate

, until a certain convergence rule $\|\Delta \mathbf{w}\| < \epsilon$

Linear Regression by TensorFlow

- in LinReg.ipynb
- Generate training data

```python
# Generate training data
np.random.seed(1)
def f(x, a, b):
    n    = train_X.size
    vals = np.zeros((1, n))
    for i in range(0, n):
        ax  = np.multiply(a, x.item(i))
        val = np.add(ax, b)
        vals[0, i] = val
    return vals

Wref = 0.7
bref = -1.
n    = 20
noise_var = 0.001

train_X   = np.random.random((1, n))
ref_Y     = f(train_X, Wref, bref)
train_Y   = ref_Y + np.sqrt(noise_var)*np.random.randn(1, n)

# Plot
plt.figure(1)
plt.plot(train_X[0, :], ref_Y[0, :], 'ro', label='Original data')
plt.plot(train_X[0, :], train_Y[0, :], 'bo', label='Training data')
plt.axis('equal')
plt.legend(loc='lower right')
```



Linear Regression by TensorFlow

- in LinReg.ipynb
- Linear Regression Model

```
# Prepare for Linear Regression

# Parameters
training_epochs = 2000
display_step    = 50

# Set TensorFlow Graph
X = tf.placeholder(tf.float32, name="input")
Y = tf.placeholder(tf.float32, name="output")
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")

# Construct a Model
activation = tf.add(tf.mul(X, W), b)

# Define Error Measure and Optimizer
learning_rate   = 0.01
cost = tf.reduce_mean(tf.pow(activation-Y, 2))
# learning_rate   = 0.001
# cost = tf.sqrt(tf.reduce_sum(tf.pow(activation-Y, 2)))
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) #Gradient descent

# Initializer
init = tf.initialize_all_variables()
```

Linear Regression by TensorFlow

- in LinReg.ipynb
- Run

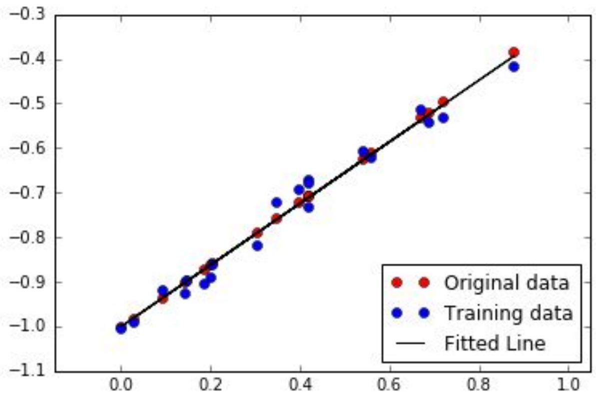
```
# Run!
sess = tf.Session()
# Initialize
sess.run(init)
for epoch in range(training_epochs):
    for (x, y) in zip(train_X[0, :], train_Y[0, :]):
        # print "x: ", x, " y: ", y
        sess.run(optimizer, feed_dict={X:x, Y:y})

    # Check cost
    if epoch % display_step == 0:
        costval = sess.run(cost, feed_dict={X: train_X, Y:train_Y})
        print ("Epoch:", "%04d"%(epoch+1), "cost=", "{:.5f}".format(costval))
        Wtemp = sess.run(W)
        btemp = sess.run(b)
        print (" Wtemp is", "{:.4f}".format(Wtemp), "btemp is", "{:.4f}".format(btemp)
        print (" Wref is", "{:.4f}".format(Wref), "bref is", "{:.4f}".format(bref))

# Final W and b
Wopt = sess.run(W)
bopt = sess.run(b)
fopt = f(train_X, Wopt, bopt)
```

```
('Epoch:', '0001', 'cost=', '0.21532')
(' Wtemp is', '-0.6964', 'btemp is', '-0.1715'
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0051', 'cost=', '0.01696')
(' Wtemp is', '0.1656', 'btemp is', '-0.7954')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0101', 'cost=', '0.00264')
(' Wtemp is', '0.5047', 'btemp is', '-0.9280')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0151', 'cost=', '0.00083')
(' Wtemp is', '0.6247', 'btemp is', '-0.9750')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0201', 'cost=', '0.00060')
(' Wtemp is', '0.6672', 'btemp is', '-0.9916')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0251', 'cost=', '0.00057')
(' Wtemp is', '0.6823', 'btemp is', '-0.9975')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0301', 'cost=', '0.00057')
(' Wtemp is', '0.6876', 'btemp is', '-0.9996')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0351', 'cost=', '0.00056')
(' Wtemp is', '0.6895', 'btemp is', '-1.0003')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0401', 'cost=', '0.00056')
(' Wtemp is', '0.6901', 'btemp is', '-1.0006')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0451', 'cost=', '0.00056')
(' Wtemp is', '0.6904', 'btemp is', '-1.0007')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0501', 'cost=', '0.00056')
(' Wtemp is', '0.6905', 'btemp is', '-1.0007')
(' Wref is', '0.7000', 'bref is', '-1.0000')
('Epoch:', '0551', 'cost=', '0.00056')
(' Wtemp is', '0.6905', 'btemp is', '-1.0007')
(' Wref is', '0.7000', 'bref is', '-1.0000')
```

Linear Regression by TensorFlow

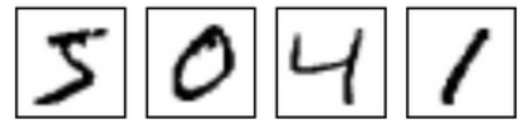
- in LinReg.ipynb
- Plot

```
# Plot Results
plt.figure(2)
plt.plot(train_X[0, :], ref_Y[0, :], 'ro', label='Original data')
plt.plot(train_X[0, :], train_Y[0, :], 'bo', label='Training data')
plt.plot(train_X[0, :], fopt[0, :], 'k-', label='Fitted Line')
plt.axis('equal')
plt.legend(loc='lower right')
```



MNIST dataset

- **MNIST Dataset**
  - Handwritten digits, which has a training set of 60,000 examples and a test set of 10,000 examples.
  - Includes 28 x 28 gray-scaled image and labels for each image.



- **For TensorFlow,**
  - we will use `tensorflow.examples.tutorials.mnist`
  - https://www.tensorflow.org/versions/master/tutorials/mnist/beginners/index.html
  - You can find how to download, extract, and load MNIST dataset using tensorflow pkg in MNIST.ipynb

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

universitätbonn ais

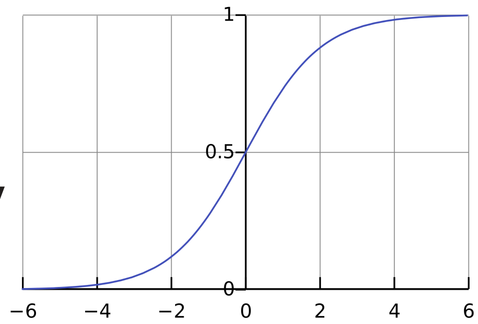
[http://yann.lecun.com/exdb/mnist/]

Logistic (Softmax) Regression

- **Logistic regression**
  - uses the same basic formula of the linear regressin, but instead of the continuous $\mathbf{y} \in \mathbb{R}^N$, it is regressing for the probability of a categorical outcome.
  - finds a linear boundary between class concepts.

- **Binary Logistic Regression**
  - One outcome continous variable and two states of that variable, either 0 or 1.
  - Logistic (softmax) function:

$$\sigma(z) = \frac{exp(z)}{exp(z) + 1} = \frac{1}{1 + exp(-z)}$$



  - The function is used to model the probability of input z being a part of either binary class 0 or 1.

Logistic (Softmax) Regression

- **Logistic regression**
  - assumes that the relation between a multi-dimensional input instance and its probability is linear.

$$P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + \mathbf{b})$$

  - Training is to maximize the probability of the class y given the data X by choosing appropriate weights w and b.

$$\mathbf{w}^*, \mathbf{b}^* = \arg\min_{\mathbf{w},\mathbf{b}} l(\mathbf{X}, \mathbf{y}, \mathbf{w}, \mathbf{b})$$

  - It does so by minimizing its loss, the negative logarithm of the class probability

$$l(\mathbf{X}, \mathbf{y}, \mathbf{w}, \mathbf{b}) = -\ln \prod_{\mathbf{x} \in \mathbf{X}, y \in \mathbf{y}} P(y = 1|\mathbf{x})^y P(y = 0|\mathbf{x})^{1-y}$$

Assignment 2

- **Training a Logistic Regression Classifier with MNIST dataset using TensorFlow**
  - Load MNIST dataset
  - Create Graph for Logistic Regression
  - Implement the loss function using `tf.nn.softmax()` and `tf.matmul()`, `tf.reduce_sum()`, `tf.reduce_mean()`, `tf.log()`
  - Define an optimizer using tf.train.GradientDescentOptimizer().minimize()
  - Define a node to calculate prediction accuracy using `tf.argmax()`
  - Training the graph using mini-batch learning using `mnist.train.next_batch()`
  - Compute average loss, training accuracy, test accuracy
  - Chainging **learning rate**, **training iteration**, **batch size** to achieve your best result.
  - Draw loss vs iteration graph with your best parameters.