MAE 598: Design Optimization
Homework 3

Tanner Bitz

October 12, 2018

1 Problem 1

1.1 Problem Statement

Vapor-liquid equilibrium data are correlated using two adjustable parameters A_{12} and A_{21} per binary mixture. For low pressures, the equilibrium relation can be formulated as:

$$\begin{aligned} p =& x_1 \exp\left(A_{12}\left(\frac{A_{21}x_2}{A_{12}x_1 + A_{21}x_2}\right)^2\right)p_1^{sat} \\ &+ x_2 \exp\left(A_{21}\left(\frac{A_{12}x_1}{A_{12}x_1 + A_{21}x_2}\right)^2\right)p_2^{sat} \end{aligned} \tag{1}$$

Here the saturation pressures are given by the Antoine equation

$$\log_{10}(p^{sat}) = a_1 - \frac{a_2}{T + a_3} \tag{2}$$

where $T = 20°C$ and $a_{1,2,3}$ for a water - 1,4 dioxane system is given below.

	a_1	a_2	a_3
Water	8.07131	1730.63	233.426
1,4 dioxane	7.43155	1554.679	240.337

The following table list the measure data. Recall that in a binary system $x_1 + x_2 = 1$.

x_1	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
p	28.1	34.4	36.7	36.9	36.8	36.7	36.5	35.4	32.9	27.7	17.5

Estimate A_{12} and A_{21} using data from Table 1.1.

1. Formulate the least square problem
2. Solve using your own gradient descent or Newton's implementation
3. Solve using MATLAB function "lsqnonlin" or "lsqcurvefit"

1.2 Least Square Problem Formulation

Problem 1

Formulation :

$$p = x_1 \exp\left(A_{12}\left(\frac{A_{21}x_2}{A_{12}x_1 + A_{21}x_2}\right)^2\right) p_1^{sat} + x_2 \exp\left(A_{21}\left(\frac{A_{12}x_1}{A_{12}x_1 + A_{21}x_2}\right)^2\right) p_2^{sat}$$

The goal is to fit A_{12} and A_{21} with least squares

$$\min_{A_{12}, A_{21}} \sum_{i=1}^{N} \left(y_i - p(x_i, A_{12}, A_{21})\right)^2$$

subject to $x_1 + x_2 = 1$

$$\log_{10}(p_{sat}) = a_1 - \frac{a_2}{T + a_3}, \quad a_1, a_2, a_3 \text{ in Table}$$

$$T = 20$$

All of the constraints can be solved and substituted into objective functions

Saturation Pressures

$$p_1^{sat} = p_{water}^{sat} = 10^{\left(8.07131 - \frac{1730.63}{20 + 233.426}\right)} = 17.4733$$

$$p_2^{sat} = p_{1,4\ dioxane}^{sat} = 10^{\left(7.43155 - \frac{1554.679}{20 + 240.337}\right)} = 28.8241$$

Binary

$$x_1 + x_2 = 1$$

$$x_2 = 1 - x_1$$

$$\min_{A_{12},A_{21}} \sum_{i=1}^{N} \left(y_i - \left[x_{1,i} \exp\left(A_{12} \left(\frac{A_{21} X_{2,i}}{A_{12} X_{1,i} + A_{21} X_{2,i}} \right)^2 \right) p_1^{sat} + X_{2,i} \exp\left(A_{21} \left(\frac{A_{12} X_{1,i}}{A_{12} X_{1,i} + A_{21} X_{2,i}} \right)^2 \right) p_2^{sat} \right] \right)^2$$

$$\boxed{\min_{A_{12},A_{21}} \sum_{i=1}^{N} \left(y_i - \left[X_{1,i} \exp\left(A_{12} \left(\frac{A_{21}(1-X_{1,i})}{A_{12} X_{1,i} + A_{21}(1-X_{1,i})} \right)^2 \right) p_{wat}^{sat} + (1-X_{1,i}) \exp\left(A_{21} \left(\frac{A_{12} X_{1,i}}{A_{12} X_{1,i} + A_{21}(1-X_{1,i})} \right)^2 \right) p_{dxn}^{sat} \right] \right)^2}$$

This is a nonlinear problem

so we will try to

$$\min_{a} \frac{1}{2} f(a)^T f(a)$$

where

$$f(a) = \begin{bmatrix} y_1 - p(x_1, a) \\ \vdots \\ y_n - p(x_n, a) \end{bmatrix}$$

The gradient

$$\frac{\partial F}{\partial a} = J^T f(a)$$

$$\begin{bmatrix} \frac{\partial f_1}{\partial a_1} & \frac{\partial f_1}{\partial a_2} & \cdots & \frac{\partial f_1}{\partial a_p} \\ \vdots & & & \\ \frac{\partial f_n}{\partial a_1} & \cdots & \cdots & \frac{\partial f_n}{\partial a_p} \end{bmatrix} f(a)$$

$$\boxed{a_{k+1} = a_k - \left(J^T J + \lambda I \right)^{-1} J^T f(a)}$$

1.3 Gradient Descent Implementation

Here's my code:

```
#!/usr/bin/python3
import numpy as np



# Antoine Equation Constants
a1_wat = 8.07131
a2_wat = 1730.63
a3_wat = 233.426

a1_dxn = 7.43155
a2_dxn = 1554.679
a3_dxn = 240.337

T = 20 # deg C


# Calc saturation pressure
def AntoineEq(a1, a2, a3, T):
    return 10**(a1 - a2/(T + a3))

psat_wat = AntoineEq(a1_wat, a2_wat, a3_wat, T)
psat_dxn = AntoineEq(a1_dxn, a2_dxn, a3_dxn, T)

# Data
x1 = np.arange(0, 1.1, 0.1)
x2 = np.ones(x1.shape) - x1
y = np.array([28.1, 34.4, 36.7, 36.9, 36.8, 36.7, 36.5, 35.4, 32.9, 27.7, 17.5])

# Objective function subfunctions
exp1inner = lambda xone, xtwo, A12, A21: A12 * (A21 * xtwo / (A12 * xone + A21 * xtwo))
exp2inner = lambda xone, xtwo, A12, A21: A21 * (A12 * xone / (A12 * xone + A21 * xtwo))
yerror = lambda e1inner, e2inner, xone, xtwo, y: (y - (
            xone * np.exp(e1inner) * psat_wat + xtwo * np.exp(e2inner) * psat_dxn))

# Gradient functions
dfdA12 = lambda p1, p2, x1, x2, A12, A21: p1*x1*np.exp((A12*A21**2*x2**2)/(A12*x1 + A21*
dfdA21 = lambda p1, p2, x1, x2, A12, A21: p2*x2*np.exp((A12**2*A21*x1**2)/(A12*x1 + A21*


# Define objective function and it's gradient xi in x:
def objfun(x):
    A12 = x[0, 0]
    A21 = x[1, 0]
    fa = np.zeros((len(x1), 1))
    for i in range(0, len(x1)):
        e1inner_i = exp1inner(x1[i], x2[i], A12, A21)
        e2inner_i = exp2inner(x1[i], x2[i], A12, A21)
        fa[i, 0] = yerror(e1inner_i, e2inner_i, x1[i], x2[i], y[i])
    return 1/2*fa.transpose().dot(fa)

def g(x):
```

```
    A12 = x[0, 0]
    A21 = x[1, 0]
    JaTrans = np.zeros((2,len(x1)))
    fa = np.zeros((len(x1), 1))
    for i in range(0, len(x1)):
        JaTrans[0, i] = dfdA12(psat_wat, psat_dxn, x1[i], x2[i], A12, A21)
        JaTrans[1, i] = dfdA21(psat_wat, psat_dxn, x1[i], x2[i], A12, A21)
        e1inner_i = exp1inner(x1[i], x2[i], A12, A21)
        e2inner_i = exp2inner(x1[i], x2[i], A12, A21)
        fa[i, 0] = yerror(e1inner_i, e2inner_i, x1[i], x2[i], y[i])
    return JaTrans.dot(fa)

def H(x):
    A12 = x[0, 0]
    A21 = x[1, 0]
    JaTrans = np.zeros((2, len(x1)))
    for i in range(0, len(x1)):
        JaTrans[0, i] = dfdA12(psat_wat, psat_dxn, x1[i], x2[i], A12, A21)
        JaTrans[1, i] = dfdA21(psat_wat, psat_dxn, x1[i], x2[i], A12, A21)
    return JaTrans.dot(JaTrans.transpose())


# Initialize Problem
prob = {'eps': 1e-3,
        'method': 'gradientDescent',
        'a': 1,
        't': 0.01,
        'b': 0.5,
        'itermax': 10000
        }

# Newton's Method
l = 0.01
a = np.array([[2],[1.5]])
e = 0.001;
for i in range(0, 10000):
    a = a + e*np.linalg.inv(H(a)+l*np.eye(2)).dot(g(a))

print(a)
```

The solution:
A12 = 1.9584
A21 = 1.6892

Note: This is the same answer as lsqnonlin, so the fit shown in Figure 1 also applies to this answer.

1.4 Use MATLAB - lsqnonlin

My function to plug into lsqnonlin

```
function f = mae598_desopt_hw3_p1(a)
    % a(1) = A12
    % a(2) = A21

    %Calc psat for water, 1,4 dioxane
```

```
    a1_wat = 8.07131;
    a2_wat = 1730.63;
    a3_wat = 233.426;
    a1_dxn = 7.43155;
    a2_dxn = 1554.679;
    a3_dxn = 240.337;
    T = 20;
    psat_wat = 10^(a1_wat - a2_wat/(T+a3_wat));
    psat_dxn = 10^(a1_dxn - a2_dxn/(T+a3_dxn));

    % Data
    x1 = 0:0.1:1;
    x2 = ones(size(x1))-x1;
    y = [28.1, 34.4, 36.7, 36.9, 36.8, 36.7, 36.5, 35.4, 32.9, 27.7, 17.5];
    f = zeros(length(x1),1);

    %Calculate f
    for i = 1:length(x1)
        e1inner = a(1)*(a(2)*x2(i)/(a(1)*x1(i) + a(2)*x2(i)))^2;
        e2inner = a(2)*(a(1)*x1(i)/(a(1)*x1(i) + a(2)*x2(i)))^2;
        f(i) = x1(i)*exp(e1inner)*psat_wat + x2(i)*exp(e2inner)*psat_dxn - y(i);
    end
end
```

My code to run lsqnonlin:

```
%% lsqnonlin
a0 = [1, 1];
options = optimoptions(@lsqnonlin, 'Algorithm', 'trust-region-reflective');
a = lsqnonlin(@mae598_desopt_hw3_p1, a0, [], [], options)
```

The solution:
A12 = 1.9584
A21 = 1.6892

Code to plot the fitted curve:

First, here's the function:

```
function f = mae598_desopt_hw3_p1_fun(x)
    A12 = 1.9584;
    A21 = 1.6892;

    %Calc psat for water, 1,4 dioxane
    a1_wat = 8.07131;
    a2_wat = 1730.63;
    a3_wat = 233.426;
    a1_dxn = 7.43155;
    a2_dxn = 1554.679;
    a3_dxn = 240.337;
    T = 20;
    psat_wat = 10^(a1_wat - a2_wat/(T+a3_wat));
    psat_dxn = 10^(a1_dxn - a2_dxn/(T+a3_dxn));


    e1inner = A12*(A21*x(2)/(A12*x(1) + A21*x(2)))^2;
    e2inner = A21*(A12*x(1)/(A12*x(1) + A21*x(2)))^2;
    f = x(1)*exp(e1inner)*psat_wat + x(2)*exp(e2inner)*psat_dxn;
```

```
end
```

Here's the scipt to evaluate and plot:

```
%% Check lsqnonlin solution with plot
x1 = 0:0.1:1;
x2 = ones(size(x1))-x1;
y = [28.1, 34.4, 36.7, 36.9, 36.8, 36.7, 36.5, 35.4, 32.9, 27.7, 17.5];

j = 1;
for i = 0:0.005:1
    x_blah(j) = i;
    x = [i, 1-i];
    f(j) = mae598_desopt_hw3_p1_fun(x);
    j = j + 1;
end

close all
figure(1)
plot(x1, y, 'ro');
hold on
plot(x_blah, f);
hold off
legend('Data', 'Fitted Curve')
title('Fitting with lsqnonlin')
xlabel('x1')
ylabel('y')
```

Figure 1 shows the plot to validate

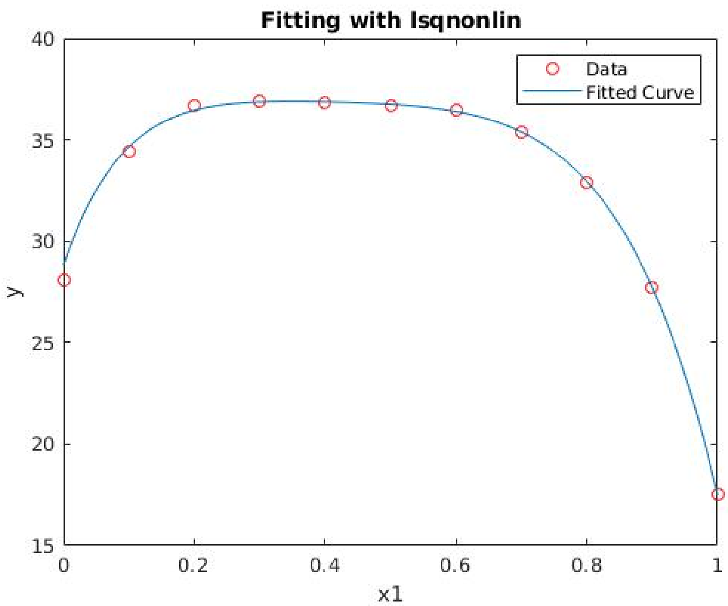


Figure 1: lsqnonlin evaulation plot

2 Problem 2

2.1 Problem Statement

Download the data homework3data.mat. The data contains a set of topologically optimal brackets. Each row of X represents a bracket structure and y the angle of the point load. See figure. Build a predictive model using y as the input and X the output through the MATLAB *Neural Net Fitting* App under *Math, Statistics, and Optimization* section of the App tab. How will you tell if your predictions are good?

Notes: When you use the Nerual Net Fitting app, make sure to check the matrix rows box because in our data, each rows is a data point, i.e., we have 100 data points.

2.2 Solution

The code that I used to complete this task is:

```
%% Problem 2
load homework3data.mat
neuralnet = fitnet();
neuralnet.divideParam.trainRatio = 0.7;
neuralnet.divideParam.valRatio = 0.15;
neuralnet.divideParam.testRatio = 0.15;

train(neuralnet, X', y')
```

Figure 2 shows the error histogram from the trained neural net.

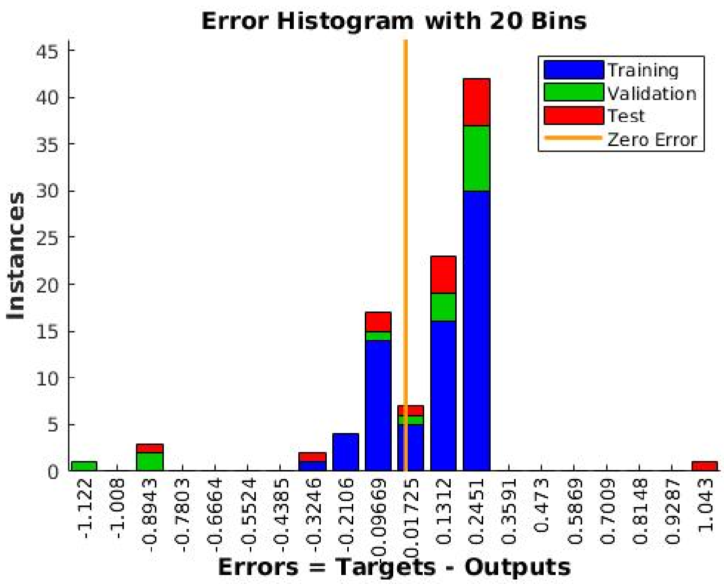


Figure 2: Error histogram from trained neural net

The goal of this neural net is to take in pixel data and determine the angle at which the load is applied. Based on our error histogram we see the most error from our predicted model is about +- 1 degree, however there are very few of these instances. Most of the instances of error are around zero degrees, and the single most instances occurs at 0.245 degrees, which is a small error. From this we can conclude that our neural net did a good job training the predicted model as there is very little error in the validation set.

3 Problem 3

3.1 Problem Statement

Logistic regression is commonly used to approximate systems with binary outputs, e.g., the result of an election, the outcome of a drug, the purchase of a product, or many other discrete decision making of human beings. The mathematical form of a logistic regression model is as follows:

$$p(y; x, \theta) = \frac{1}{1 + \exp\left(-y\theta^T x\right.} \tag{3}$$

where y is the output that takes either 1 or 1, x are the input variables (covariates), θ are the unknown parameters. Given a dataset $D = \{(x_i, y_i)\}_{i=1}^N$ the likelihood of θ can be written as

$$L(\theta, D) = \prod_{i=1}^{N} \frac{1}{1 + exp(-y_i\theta^T x_i)} \tag{4}$$

Is the maximum likelihood estimate of θ unique?

Hint: To obtain the maximum likelihood estimate of , one needs to minimize the **negative log** -likelihood function. To show that the solution is unique, you need to show that the negative log-likelihood function has a positive definite Hessian.

3.2 Solution

Problem 3

$$p(y_i; x, \theta) = \frac{1}{1 + \exp(-y\theta^T x)}$$

Assuming X is iid, the liklihood is:

$$L(\theta, D) = \prod_{i=1}^{N} \frac{1}{1 + \exp(-y\theta^T x)} = \frac{1}{\prod_{i=1}^{N} [1 + \exp(-y\theta^T x)]}$$

where

$$D = \{(x_i, y_i)\}_{i=1}^{N}$$

Taking the negative log, to get the negative log likelihood function gives:

$$\ell(\theta, D) = -\left(\cancel{\ln(1)}^{0} - \ln\left[\prod_{i=1}^{N} [1 + \exp(-y_i\theta^T x_i)]\right]\right)$$

$$= \sum_{i=1}^{N} \ln(1 + \exp(-y_i\theta^T x_i))$$

To find the max likelihood of θ, we need to minimize $\ell(\theta, D)$, so take derivative of ℓ w.r.t. θ

$$\frac{\partial \ell}{\partial \theta} = \sum_{i=1}^{N} \frac{1}{1 + \exp(-y_i\theta^T x_i)} \left(-y_i x_i \left(\exp(-y_i \theta^T x_i)\right)\right)$$

$$= \sum_{i=1}^{N} \frac{-y_i x_i \left(\exp(-y_i \theta^T x_i)\right)}{1 + \exp(-y_i \theta^T x_i)} = 0$$

variable substitute

$r = \exp(-y_i \theta^T x_i)$, $\quad -y_i x_i = a$

$$= \sum_{i=1}^{N} a \left(\frac{r}{1+r}\right) = \frac{\partial l}{\partial \theta}$$

$$\frac{\partial^2 l}{\partial \theta^2} = \sum_{i=1}^{N} a \frac{\partial}{\partial r}\left(\frac{r}{1+r}\right) \cdot \frac{\partial r}{\partial \theta}$$

$$= \sum_{i=1}^{N} \frac{(y_i x_i)^2 \exp(-y_i \theta^T x_i)}{\left(1 + \exp(-y_i \theta^T x_i)\right)^2}$$

$$\frac{\partial}{\partial r}\left(\frac{r}{1+r}\right) = \frac{1}{(1+r)^2}$$

$$\frac{\partial r}{\partial \theta} = -y_i x_i \exp(-y_i \theta^T x_i)$$

y_i is either -1 or 1, so $y_i^2 = 1$

$$\frac{\partial^2 l}{\partial \theta^2} = \sum_{i=1}^{N} \frac{(x_i)^2 \exp(-y_i \theta^T x_i)}{\left(1 + \exp(-y_i \theta^T x_i)\right)^2}$$

x_i^2 is always positive, the exp function is always positive, therefore

$$\frac{\partial^2 l}{\partial \theta^2} > 0$$

Because the Hessian is positive definite, then l is strictly convex, so there is only one θ_{max} and it is unique.

4 Problem 4

4.1 Problem Statement

Please go through this tutorial on creating a metamodel using a deep convolutional neural network to predict Youngs modulus of sandstone structures. There is an instruction on installing Keras and Tensorflow. Please attach your results.

Note: To open jupyter notebook, go to the command line, change directory to the downloaded folder, and type in jupyter notebook. This should open the notebook. Then run each block of the code to get the results.

4.2 Solution

Young's Modulus Prediction

October 11, 2018

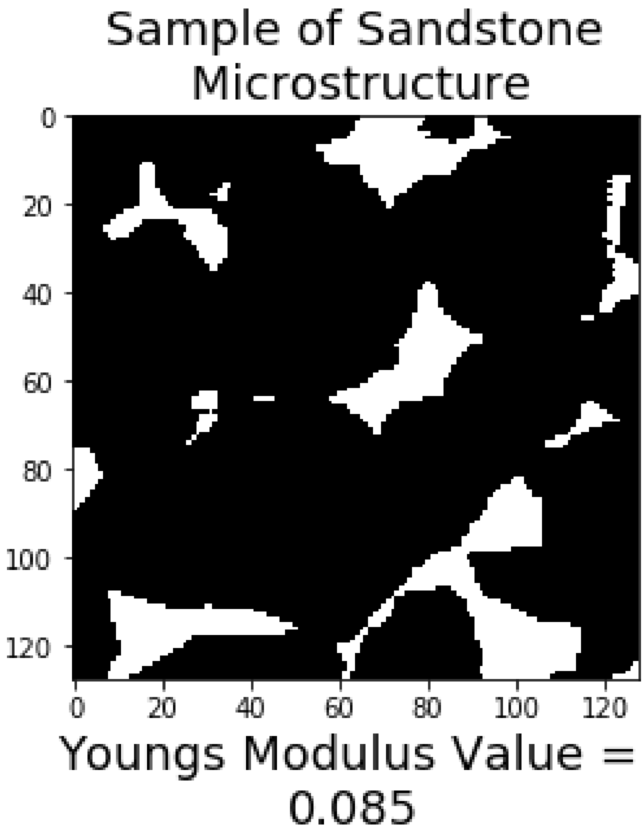
In [1]:
```
import numpy as np
import scipy.io as sio
from sklearn.metrics import mean_squared_error
from keras.models import Sequential, load_model
from keras.layers import Dense, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras import backend as K
import matplotlib.pyplot as plt
np.random.seed(1337)  # for reproducibility
%matplotlib inline
```

```
Using TensorFlow backend.
```

In [2]:
```
# input image dimensions
img_rows, img_cols = 128, 128
# size of pooling area for max pooling
pool_size = (2, 2)
# convolution kernel size
kernel_size = (3, 3)

# Import the data
WB = np.array(sio.loadmat('sandstone_data.mat')['Data'])
Y_data = np.array(sio.loadmat('sandstone_data.mat')['L'])
# Normalize the data
X_data = np.reshape(WB,(len(WB),1,img_rows,img_cols))
Y_data = (Y_data-min(Y_data))/(max(Y_data)-min(Y_data))

# data splitting
X_train = X_data[:600]
Y_train = Y_data[:600]

X_val = X_data[600:700]
Y_val = Y_data[600:700]

X_test = X_data[700:]
Y_test = Y_data[700:]
```

```
# show image sample
axes = plt.gca()
plt.imshow(X_train[10].reshape(img_rows,img_cols),'gray')
axes.set_title('Sample of Sandstone\n Microstructure',fontsize=18)
axes.set_xlabel('Youngs Modulus Value = \n{:.2}'.format(Y_train[0][0]),fontsize=18)
```

Out[2]: Text(0.5,0,'Youngs Modulus Value = \n0.085')



In [3]:
```
# Adjust data shape for different Keras version
if K.image_dim_ordering() == 'th':
    X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
    X_val = X_val.reshape(X_val.shape[0], 1, img_rows, img_cols)
    X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
    X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)
    X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
# Tensorflow only take float32 data type
X_train = X_train.astype('float32')
X_val = X_val.astype('float32')
X_test = X_test.astype('float32')

# print out the data information
print('X_train shape:', X_train.shape)
print(X_train.shape[0], 'train samples')
print(X_val.shape[0], 'validate samples')
print(X_test.shape[0], 'test samples')
```

```
X_train shape: (600, 128, 128, 1)
600 train samples
100 validate samples
68 test samples
```

In [4]:
```
# CNN Model
model = Sequential()

# block 1
model.add(Convolution2D(24, (6, 6), padding='same', input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

# block 2
model.add(Convolution2D(48, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

# fully connected layers
model.add(Flatten())
model.add(Dense(100))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
# model compile
model.compile(loss='mse', optimizer='adam', metrics=['mae'])
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 128, 128, 24)      888
_________________________________________________________________
activation_1 (Activation)    (None, 128, 128, 24)      0
_________________________________________________________________
```

```
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 24)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 64, 64, 48)        10416
_________________________________________________________________
activation_2 (Activation)    (None, 64, 64, 48)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 48)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 49152)             0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               4915300
_________________________________________________________________
activation_3 (Activation)    (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
_________________________________________________________________
activation_4 (Activation)    (None, 1)                 0
=================================================================
Total params: 4,926,705
Trainable params: 4,926,705
Non-trainable params: 0
_________________________________________________________________
```

```
In [5]: # train the model
        from keras.callbacks import EarlyStopping
        early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=0, mode
        model.fit(X_train, Y_train, batch_size=50, epochs=100, verbose=1, validation_data=(X_va
                  callbacks=[early_stop], initial_epoch=0)
```

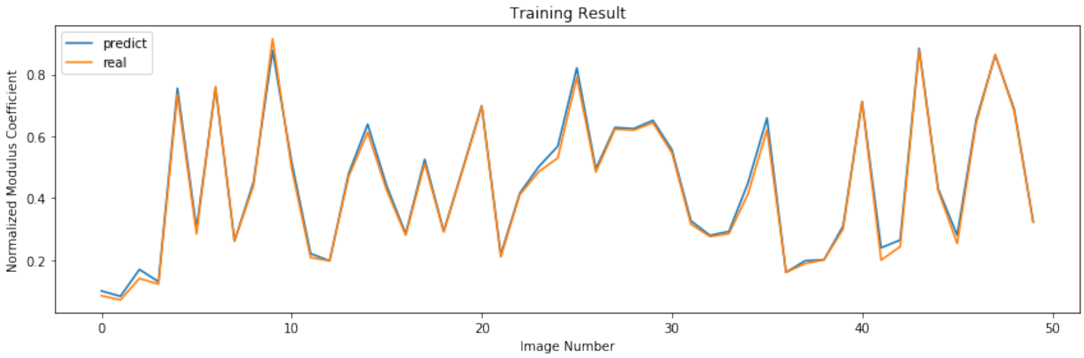
```
Train on 600 samples, validate on 100 samples
Epoch 1/100
600/600 [==============================] - 17s 29ms/step - loss: 0.1625 - mean_absolute_error:
Epoch 2/100
600/600 [==============================] - 16s 27ms/step - loss: 0.0537 - mean_absolute_error:
Epoch 3/100
600/600 [==============================] - 16s 27ms/step - loss: 0.0215 - mean_absolute_error:
Epoch 4/100
600/600 [==============================] - 15s 24ms/step - loss: 0.0051 - mean_absolute_error:
Epoch 5/100
600/600 [==============================] - 15s 25ms/step - loss: 0.0018 - mean_absolute_error:
Epoch 6/100
600/600 [==============================] - 15s 25ms/step - loss: 0.0011 - mean_absolute_error:
Epoch 7/100
600/600 [==============================] - 15s 25ms/step - loss: 6.6768e-04 - mean_absolute_er
Epoch 8/100
600/600 [==============================] - 15s 24ms/step - loss: 4.9390e-04 - mean_absolute_er
Epoch 9/100
```

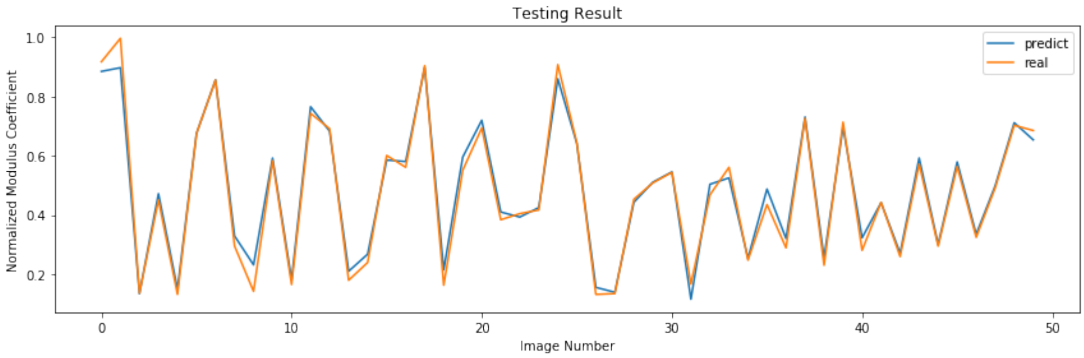
```
600/600 [==============================] - 17s 28ms/step - loss: 4.1674e-04 - mean_absolute_er
Epoch 10/100
600/600 [==============================] - 16s 27ms/step - loss: 3.6740e-04 - mean_absolute_er
```

```
Out[5]: <keras.callbacks.History at 0x7f44d45edd68>
```

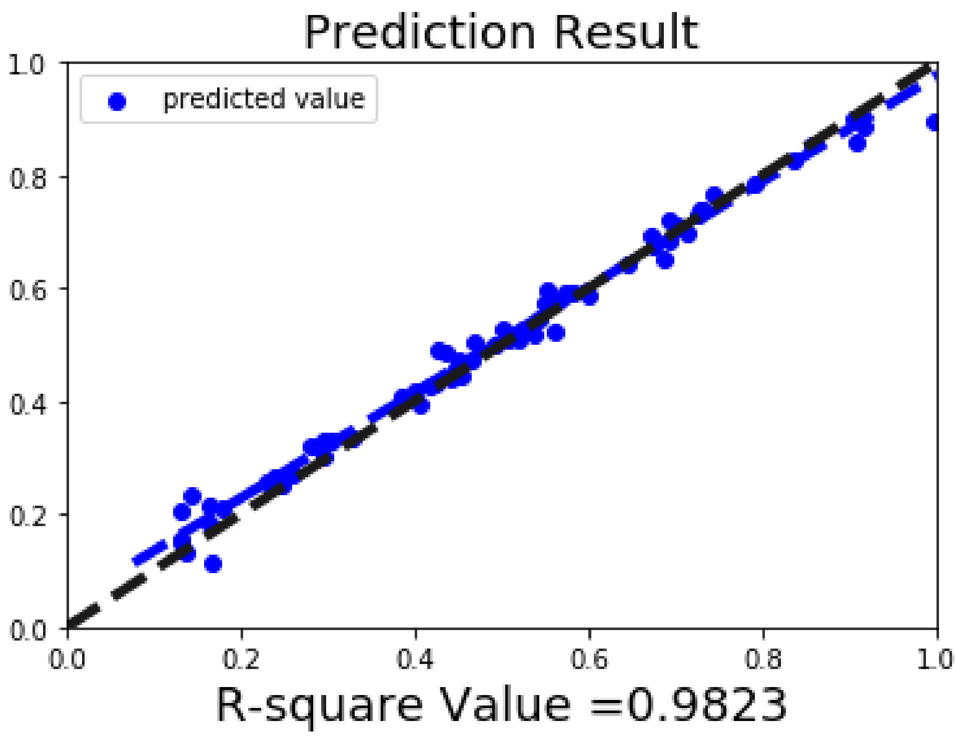
```
In [8]: # the number of points to show as comparison
        num_comp=50
        x=np.arange(num_comp)

        # define plot size
        fig = plt.figure(figsize=(12,8))
        ax1=fig.add_subplot(2,1,1)

        # prediction value by training set
        train_pred=model.predict(X_train)
        print('training mse:', mean_squared_error(Y_train, train_pred))

        ax1.plot(x,train_pred[0:num_comp], label='predict')
        ax1.plot(x,Y_train[0:num_comp],label='real')
        plt.legend()
        ax1.set_title('Training Result')
        ax1.set_xlabel('Image Number')
        ax1.set_ylabel('Normalized Modulus Coefficient')


        ax2=fig.add_subplot(2,1,2)

        # prediction value by testing set
        test_pred=model.predict(X_test)
        print('testing mse:', mean_squared_error(Y_test, test_pred))

        ax2.plot(x,test_pred[0:num_comp], label='predict')
        ax2.plot(x,Y_test[0:num_comp],label='real')
        plt.legend()
        ax2.set_title('Testing Result')
        ax2.set_xlabel('Image Number')
        ax2.set_ylabel('Normalized Modulus Coefficient')

        plt.tight_layout()
```

```
training mse: 0.0003691139357478827
testing mse: 0.0008954447495822719
```






```
In [9]: from sklearn.metrics import r2_score
        plt.scatter(Y_test.reshape(-1),test_pred,label='predicted value',color='blue')
        axes = plt.gca()
        m, b = np.polyfit(Y_test.reshape(-1), test_pred, 1)
        X_plot = np.linspace(axes.get_xlim()[0],axes.get_xlim()[1],100)
        plt.plot(X_plot, m*X_plot + b, '--',color='blue',linewidth=4)

        plt.plot([0, 1], [0, 1], ls="--", c=".1",linewidth=4)
        plt.legend()
        axes.set_title('Prediction Result',fontsize=18)
        axes.set_xlabel('R-square Value ={:.4}'.format(r2_score(Y_test.reshape(-1), test_pred.r
        plt.ylim(0.0,1.0)
        plt.xlim(0.0,1.0)

Out[9]: (0.0, 1.0)
```


Prediction Result
predicted value
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
R-square Value =0.9823

In [ ]: