IMPLEMENTING NEURAL NETWORKS IN PYTHON

LECTURE 6
SECTION 3
JUNE 11

INSTITUTE FOR APPLIED
COMPUTATIONAL SCIENCE
AT HARVARD UNIVERSITY

UNIVERSITY of
RWANDA

keras A PYTHON LIBRARY FOR NEURAL NETWORKS:

keras is a python library that provide intuitive api's for building neural networks very quickly.

Modules we need for basic neural networks

```
#keras model for feedforward neural networks
from keras.models import Sequential

#keras model for layers in feedforward networks
from keras.layers import Dense

#keras model for optimizing training objectives
from keras import optimizers
```

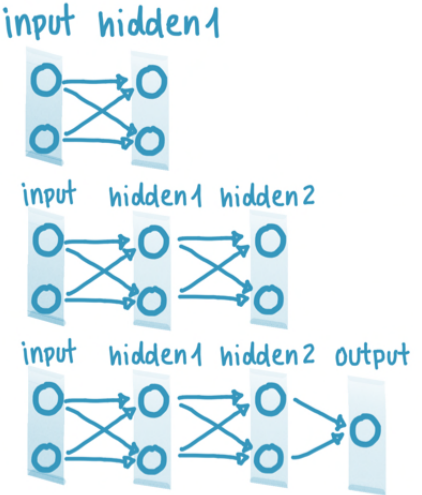
← for defining neural networks layer by layer.

← for connecting the nodes from each layer to every node in the next.

← for training neural networks

BUILDING A NEURAL NETWORK FOR REGRESSION:

```
#instantiate a feedforward model
model = Sequential()

#add layers sequentially

#input layer: 2 input dimensions
model.add(Dense(2, input_dim=2, activation='relu',
                kernel_initializer='random_uniform',
                bias_initializer='zeros'))
#hidden layer: 2 nodes
model.add(Dense(2, activation='relu',
                kernel_initializer='random_uniform',
                bias_initializer='zeros'))

#output layer: 1 output dimension
model.add(Dense(1, activation='linear',
                kernel_initializer='random_uniform',
                bias_initializer='zeros'))

#configure the model: specify training objective and training algorithm
sgd = optimizers.SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd,
              loss='mean_squared_error')
```



Stocastic gradient descent, learning rate= 0.01

Loss function for training

TRAINING A NEURAL NETWORK:

Fitting the Model

```
#fit the model and return the mean squared error during training
history = model.fit(X_train, Y_train, batch_size=20, shuffle=True, epochs=100, verbose=0)
```

compute the gradient using random samples of 20 data points at a time.

How many times we should go thru the data set 20 samples at a time

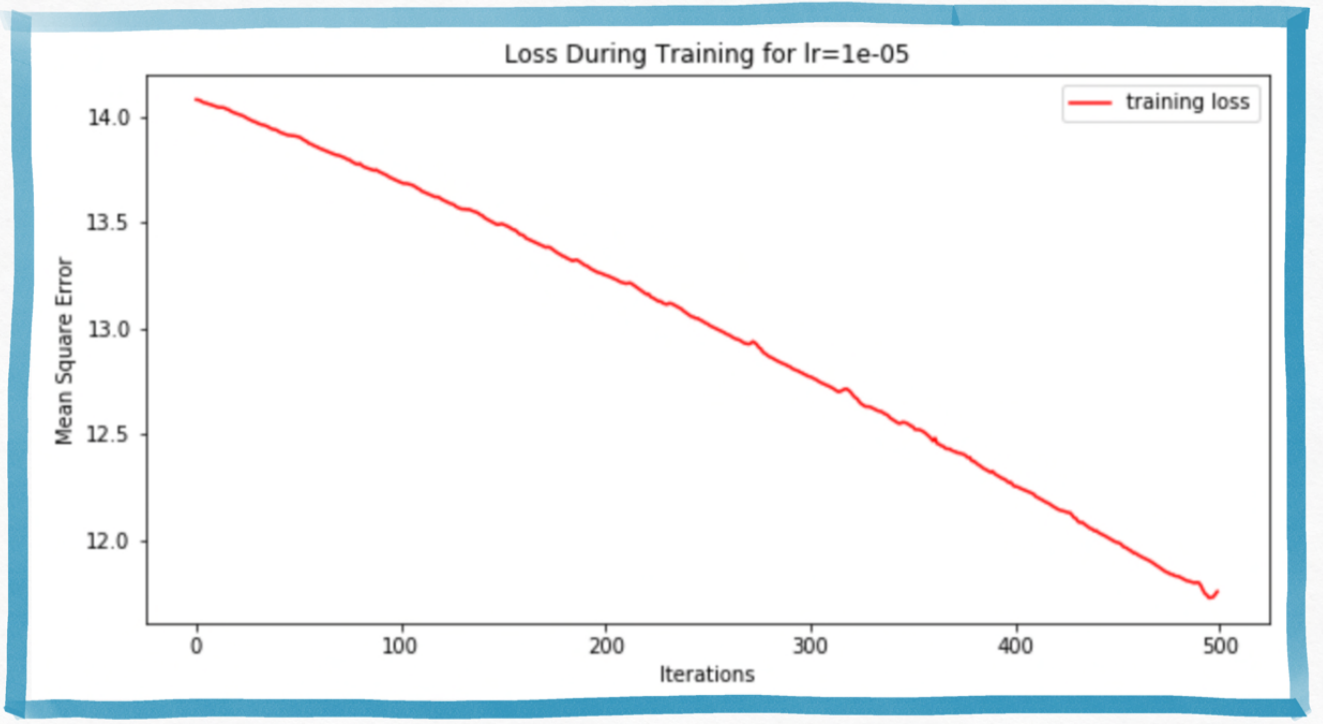
Visualizing the Trace plot

```
#fit the model and return the mean squared error during training
history = model.fit(X_train, Y_train, batch_size=20, shuffle=True, epochs=100, verbose=0)

# Plot the loss function and the evaluation metric over the course of training
fig, ax = plt.subplots(1, 1, figsize=(10, 5))

ax.plot(np.array(history.history['mean_squared_error']), color='blue', label='training accur
acy')

plt.show()
```

the loss function value is stored under the key 'mean_squared_error' in history.history

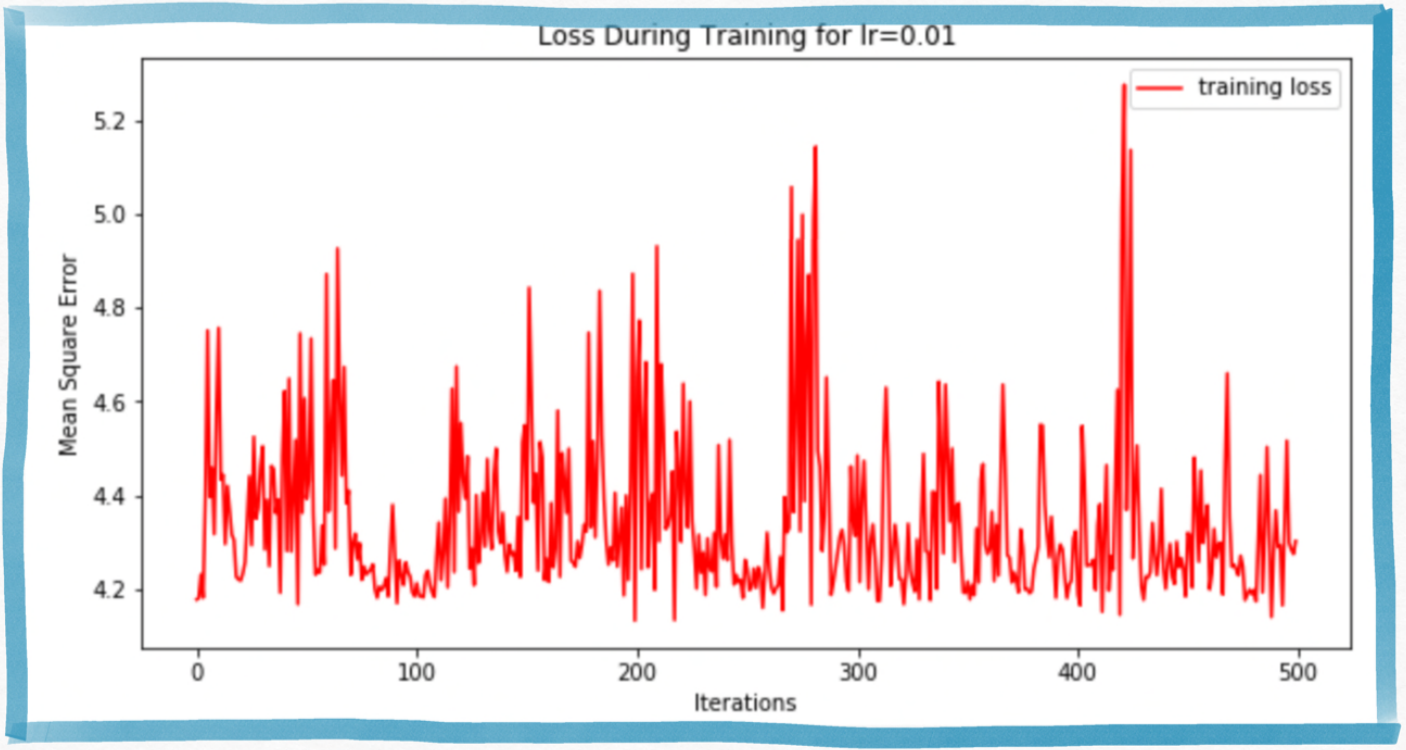
DIAGNOSING ISSUES USING THE TRACE PLOT:

Can you tell by looking at the traceplot that training ended at a stationary point? Is our choice of the learning rate appropriate?



DIAGNOSING ISSUES USING THE TRACE PLOT:

Can you tell by looking at the traceplot that training ended at a stationary point? Is our choice of the learning rate appropriate?



DIAGNOSING ISSUES USING THE TRACE PLOT:

Can you tell by looking at the traceplot that training ended at a stationary point? Is our choice of the learning rate appropriate?