Implementing Deep Neural Networks using Keras

Presented by:
Rubin Jose Peter

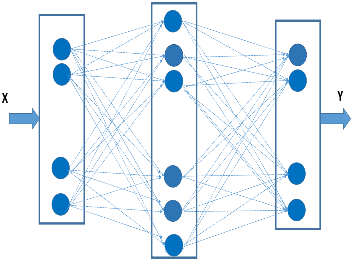
SPC Lab

March 9, 2019

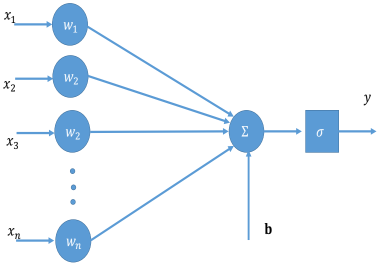
Overview

Deep Neural Networks

Neural Network



Single Neuron



- Can be used for classification & regression problems
- Training Phase:
  - Optimize the parameters of the network using **Training Data** and **Loss function**
- Testing Phase:
  - Predicts the outputs of the **Testing Data**
- Software to deploy DNNs : Keras, Tensorflow, PyTorch etc.

Installation: Keras & Tensorflow

- Anaconda
  - Open source distribution of Python programming language
  - Easily install packages from anaconda repository
  - Installation:
    https://www.anaconda.com/distribution/#download-section
- Install Tensorflow, Keras and other packages
- Commands
  - ≫ source /anaconda3/bin/activate root
  - ≫ conda create –name tensorflow python=*.*
  - ≫ conda activate tensorflow
  - ≫ pip install tensorflow
  - ≫ pip install keras
  - ≫ pip install scipy
  - ≫ pip install spyder
  - ≫ pip install h5py
  - ≫ pip install numpy

Keras

- Open-source neural-network library written in Python
- Running on top of other low level APIs like TensorFlow or Theano
- Contains commonly used neural-network building blocks
  - Dense Network (MultiLayer Perceptron)
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
  - Dropout, Batch normalization, Pooling layer
  - Different optimizers like sgd, adam etc
  - Activation functions like tanh, sigmoid, ReLU etc
- Can design neural networks with custom loss functions, layers
- Supports GPUs, clusters

Programming using Keras

- **Loading required packages**

```
import numpy as np
import matplotlib.pyplot as plt
import h5py
from keras.models import Sequential
from keras.layers import Dense
```

- **Model definition**

```
My_model.add(Dense(output_dim =  HiddenSize,
    activation = 'linear', input_dim=InpSize));

My_model.add(Dense(output_dim =  HiddenSize,
    activation = 'relu', input_dim=HiddenSize ));
```

Programming using Keras

- **Compiling the designed model**

```
My_model.compile(optimizer='adam',loss='mse',
               metrics =['accuracy']);
```

- **Training the neural network**

```
My_model.fit(X_train,Y_train,batch_size=1000,
nb_epoch = 20,shuffle =True,validation_split=.1)
```

- **Saving or loading a trained model**

```
My_model.save('SparseRecoveryModel.h5')
load_model('SparseRecoveryModel.h5')
```

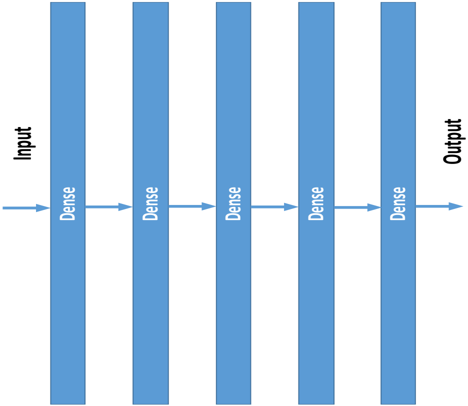
- **Evaluating the trained model**

```
My_model.predict(X_test)
```

Sequential Models

Sequential Models

- Layers are connected sequentially
- Does not support DNNs with parallel connections



```
import numpy as np
import matplotlib.pyplot as plt
import h5py

from keras.models import Sequential
from keras.layers import Dense
################################################################
filepath = 'OutputData.mat'
temp = {}
f = h5py.File(filepath,'r')
for k, v in f.items():
    temp[k] = np.array(v)

Y_train = np.transpose(temp['OutputData']);

filepath = 'InputData.mat'
temp = {}
f = h5py.File(filepath,'r')
for k, v in f.items():
    temp[k] = np.array(v)

X_train = np.transpose(temp['InputData']);
################################################################
```

Sequential Models

```
nLabel = np.size(Y_train,0);
InpSize = np.size(X_train,1);
OutSize = np.size(Y_train,1);
NeuronInHiddenLayer = InpSize*10;
###############################################
model = Sequential();

model.add(Dense(output_dim =NeuronInHiddenLayer,
                activation = 'linear',input_dim=InpSize));
model.add(Dense(output_dim =NeuronInHiddenLayer,
                activation = 'relu',input_dim=NeuronInHiddenLayer ));
model.add(Dense(output_dim =NeuronInHiddenLayer,
                activation = 'relu',input_dim=NeuronInHiddenLayer));
model.add(Dense(output_dim =OutSize,
                activation = 'linear',input_dim=NeuronInHiddenLayer))
###################################################

model.compile(optimizer='adam',loss='mse',metrics =['accuracy']);

history = model.fit(X_train,Y_train,batch_size=1000,
                    nb_epoch = 20,shuffle =True,validation_split=.1);

model.save('SparseRecoveryModel.h5')
```

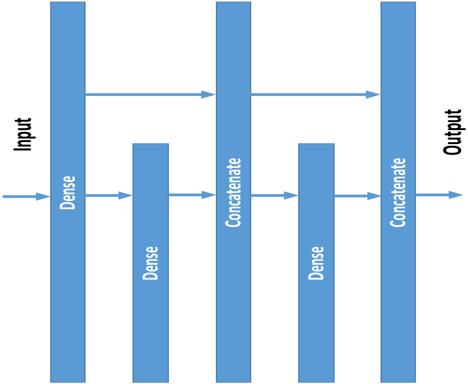
Sequential Models

```
import scipy.io as sio
from keras.models import load_model

model = load_model('SparseRecoveryModel.h5')

############################################
str2 = 'TestInputData' ;
SigTest = sio.loadmat(str2);
X_test = SigTest['TestInputData'];
##########################################


Y_pred = model.predict((X_test));


###########################################
str3 = 'TestOutputPred' ;
sio.savemat(str3, {'TestOutputPred':Y_pred});
############################################
```

Functional Models

Functional Models

- Provides more design flexibility
- Can design DNNs with parallel connections



Functional Models

```
import numpy as np
import h5py

from keras.models import Model
from keras.layers import Input,Dense,Concatenate
###############################################################
filepath = 'OutputData.mat'
temp = {}
f = h5py.File(filepath,'r')
for k, v in f.items():
    temp[k] = np.array(v)

Y_train = np.transpose(temp['OutputData']);

filepath = 'InputData.mat'
temp = {}
f = h5py.File(filepath,'r')
for k, v in f.items():
    temp[k] = np.array(v)

X_train = np.transpose(temp['InputData']);
##############################################################
```

Functional Models

```
npSize = np.size(X_train,1);
OutSize = np.size(Y_train,1);
NeuronInHiddenLayer = InpSize*10;
###################################################33

inputs = Input(shape=(InpSize,))
L1 = Dense(NeuronInHiddenLayer,activation='linear')(inputs)
L2 = Dense(NeuronInHiddenLayer,activation='relu')(L1)
L3 = Dense(NeuronInHiddenLayer,activation='relu')(L2)
L4 = Dense(NeuronInHiddenLayer,activation='relu')(L3)
L5 =  Concatenate()([L3,L4])
L6 = Dense(NeuronInHiddenLayer,activation='relu')(L5)
L7 = Dense(OutSize,activation='linear')(L6)
###################################################

My_Model = Model(inputs,L7)

My_Model.compile(optimizer='adam',loss='mse',metrics =['accuracy']);

history = My_Model.fit(X_train,Y_train,batch_size=1000,
  nb_epoch = 20,shuffle =True,validation_split=.1);

My_Model.save('SparseRecoveryModel.h5')
```

Custom Loss Function

Custom Loss Function

```
import numpy as np
import matplotlib.pyplot as plt
import h5py

from keras.models import Sequential
from keras.layers import Dense
import tensorflow as tf
###############################################################
filepath = 'OutputData.mat'
temp = {}
f = h5py.File(filepath,'r')
for k, v in f.items():
    temp[k] = np.array(v)

Y_train = np.transpose(temp['OutputData']);

filepath = 'InputData.mat'
temp = {}
f = h5py.File(filepath,'r')
for k, v in f.items():
    temp[k] = np.array(v)

X_train = np.transpose(temp['InputData']);
```

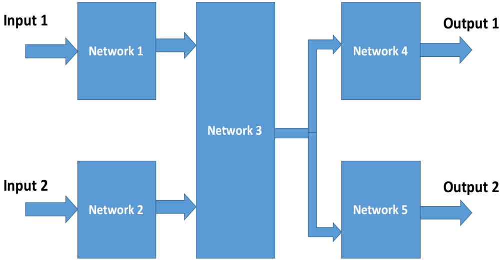
Custom Loss Function

```
#######################################################
def CustomLoss(yTrue,yPred):
    z = tf.square(tf.abs(yTrue-yPred))
    z = tf.reduce_mean(z);
    return(z)
######################################################
model = Sequential();
model.add(Dense(output_dim =NeuronInHiddenLayer,
                activation = 'linear',input_dim=InpSize));
model.add(Dense(output_dim =NeuronInHiddenLayer,
                activation = 'relu',input_dim=NeuronInHiddenLayer ));
model.add(Dense(output_dim =OutSize,
                activation = 'linear',input_dim=NeuronInHiddenLayer))

model.compile(optimizer='adam',loss=CustomLoss,metrics =['accuracy'])

history = model.fit(X_train,Y_train,batch_size=1000,
    nb_epoch = 20,shuffle =True,validation_split=.1);

model.save('SparseRecoveryModel.h5')
```

Multiple Input Multiple Output Models

Multiple Input Multiple Output Models

- Design a DNN with multiple inputs and outputs
- Can specify the loss function of each output



Multiple Input Multiple Output Models

```
class SparseNet:

        def generator_model(inputs):

                encoded1 = Dense(NeuronInHiddenLayer*20,
                   activation='linear')(inputs)
                encoded2 = Dense(NeuronInHiddenLayer*40,
                   activation='relu')(encoded1)
                encoded3 = Dense(OutSize,
                   activation='linear')(encoded2)
                return encoded3

        def Discriminator(inputs):
                x= Dense(100)(inputs)
                x= Activation('tanh')(x)
                x = Reshape((10,10,1), input_shape=(100,))(x)
                x = Conv2D(64, (5,5))(x)
                x= Activation('tanh')(x)
                x= Flatten()(x)
                x= Dense(1)(x)
                x= Activation('sigmoid')(x)
        return x
```

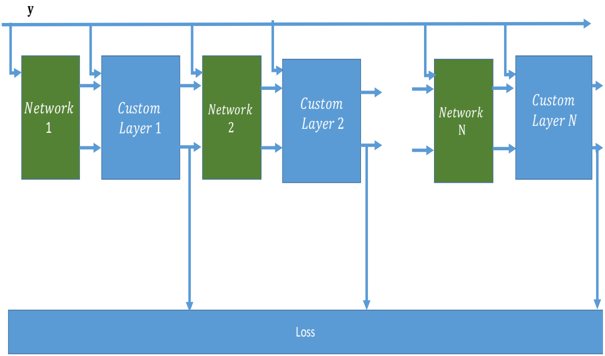
Multiple Input Multiple Output Models

```
inputs1 = Input(shape=(InpSize,))
inputs2 = Input(shape=(OutSize,))
### Generator Model #################################
SparseBranch = SparseNet.generator_model(inputs1)
SparseRec= Model(inputs=inputs1, outputs=SparseBranch)
#######Discriminator Model##############################
Discriminator = SparseNet.Discriminator(inputs2)
Disc = Model(inputs=inputs2, outputs=Discriminator)
############Creating Combined Model#####################333
SparseOp = SparseRec(inputs1)
DiscOp = Disc(SparseOp)
SparseVec= SparseNet.OutLayer1(SparseOp)
DiscOp = SparseNet.OutLayer2(DiscOp)
g = Model(inputs=inputs1, output=[SparseVec,DiscOp])

losses = {
        "OutLayer1": CustomLoss, "OutLayer2": "binary_crossentropy",
}
lossWeights = {"OutLayer1":.3 , "OutLayer2":.7}

g.compile(optimizer="adam", loss=losses, loss_weights=lossWeights,
        metrics=["accuracy"])
```

Deep Models with Custom Layers

Deep Models with Custom Layers

- Use custom layers to implement specific mathematical operations
- Suitable to unfold an iterative algorithm



Deep Models with Custom Layers

```python
class SparseNet:

  def SBL(y,alpha_0,sigma0):

############## Layer 1: Learned SBL##################################
   [mu_1,phi_1]=  Lambda(function = muEstimate)([y,alpha_0,sigma0])
   T1_1 = layers.Multiply()([mu_1, mu_1])
   C_1 = layers.Concatenate(axis=-1)([phi_1,T1_1])
   alpha_1= Dense(OutSize,activation='linear')(C_1)
############ Layer 2: Learned SBL ###################################
   [mu_2,phi_2]=  Lambda(function = muEstimate)([y,alpha_1,sigma0])
   T1_2 = layers.Multiply()([mu_2, mu_2])
   C_2 = layers.Concatenate(axis=-1)([phi_2,T1_2])
   alpha_2= Dense(OutSize,activation='linear')(C_2)
   [mu_3,phi_3]=  Lambda(function = muEstimate)([y,alpha_2,sigma0])
################ Layer 3: Learned SBL#####################33
   T1_3 = layers.Multiply()([mu_3, mu_3])
   C_3 = layers.Concatenate(axis=-1)([phi_3,T1_3])
   alpha_3= Dense(OutSize,activation='linear')(C_3)
   [mu_4,phi_4]=  Lambda(function = muEstimate)([y,alpha_3,sigma0])

   return [mu_2,mu_3,mu_4]
```

Deep Models with Custom Layers

```
def muEstimate(args):
    y,alpha,sigma = args
    alpha = tf.abs(alpha)
    sizeV = tf.shape(alpha)
    sigma = tf.abs(sigma)


    temp = tf.constant(1)
    temp =tf.cast(temp, tf.float32);

    InvalphaHalf =  tf.truediv(temp,alpha+.001);
    ..
    ..
    ..

    phi_D = tf.linalg.diag_part(phi)
    phi_D = tf.reshape(phi_D,[sizeV[0],OutSize])

    y = tf.reshape(y,[sizeV[0],InpSize,1])
    mu = tf.matmul(A_D_inv,y,transpose_a=False, transpose_b=False)

    mu = tf.reshape(mu,[sizeV[0],OutSize])
    return([mu,phi_D])
```

Implementing a Trained Model in MATLAB

Implementing a Trained Model in MATLAB

```
SBL = load_model('SBL.h5',custom_objects={'tf': tf,
    'OutSize':OutSize,'InpSize':InpSize,'Meas':Meas})
Nolayer = 11;

W = np.zeros([OutSize*2,OutSize,Nolayer])
B = np.zeros([OutSize*2,Nolayer])

for i in range(1,Nolayer+1):
    currSNR = i;
    print(i);
    str2 = 'InputData_' +str(currSNR)
    index = 6+(i-1)*4
    A = SBL.layers[index].get_weights()[0]
    b = SBL.layers[index].get_weights()[1]
    W[:,:,i-1] = A
    B[:,i-1] = b


str3 = 'WeightMatrix'
sio.savemat(str3, {'W':W});
str3 = 'BiasMatrix'
sio.savemat(str3, {'B':B});
```