Keras Tutorial

1. Installation using conda:
```
conda install -c hesi_m keras
conda update keras
```

2. To model using tensorflow.keras:
```
import keras as k
from keras.models import Sequential
from keras.layers import Dense, Activation
```

3. Defining a sequential Model NNet using Keras only:
```
model = Sequential([
    Dense(32, input_shape=(784,)), #32 hidden neurons in layer 1, input 784
    Activation('relu'),            #RELU activations
    Dense(10),                     #10 hidden neurons in layer2/output
    Activation('softmax'),
])
```

4. Alternatively define a sequential model and add layers:
```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
```

5. Implementing regularization parameters:
```
from keras.layers import regularizers
model.add(Dense(30, activation="relu",kernel_regularizer=regularizers.l2(0.001)))
```

6. Compile a model:
```
from keras.optimizers import SGD
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(optimizer=sgd, loss='categorical_crossentropy',
              metrics=['accuracy'])
```
   - Optimizer: https://keras.io/optimizers/
   - Loss: https://keras.io/losses/
     - multi-class classification problems: loss = 'categorical_crossentropy'
     - binary classification problems: loss = 'binary_crossentropy'
     - regression problems: loss = 'mse'
   - Metrics: https://keras.io/metrics/

7. Training (aka fitting):
```
M = model.fit(data, labels, epochs=10, batch_size=32)      #no validation
M = model.fit(data, labels, epochs=10, batch_size=10,
              validation_split=0.2)                        #20% validation
```

8. Scoring (aka evaluating):
```
score = model.evaluate(x_test, y_test, batch_size=128)    #print score: error and acc
```

9. Full Example:
```
import keras
from keras.models import Sequential
```

```
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
import numpy as np

# Generate dummy data
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)),
                                     num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)),
                                    num_classes=10)

model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd,
              metrics=['accuracy'])

#name it M for later
M = model.fit(x_train, y_train, epochs=20, batch_size=128, validation_split=0.2)


score = model.evaluate(x_test, y_test, batch_size=128)
```

10. Access training (history) results:

```
M.history                       #dictionary
len(M.history['acc'])           #dictionary keys: acc, loss, val_acc, val_loss
```

11. Plot loss/accuracy:

```
N = range(len(M.history["loss"]))
#plt.style.use("ggplot")        #grid plots
plt.figure()
plt.plot(N, M.history["loss"], label="train_loss")
plt.plot(N, M.history["val_loss"], label="val_loss")
plt.title("Loss/Validation Loss")
plt.xlabel("Epoch #")
plt.ylabel("Loss")
plt.legend()
```

12. Model Summary:

```
model.summary()
```

13. Detailed Configuration:

```
model.get_config()
```

Tensorflow Keras Tutorial

1. Installation using conda:

```
conda create -n tensorflow_env tensorflow
conda update tensorflow
```

2. Import library modules

```
import tensorflow as tf
from tensorflow.keras import layers
```

3. Sequential Model

```
#create a model:
model = tf.keras.Sequential()
```

4. Add sequential layers:

```
#Add the first hidden layer with 64 ReLUs and input of 32:
model.add(layers.Dense(64, activation='relu'), input_shape=(32,))

# Create a sigmoid layer:
layers.Dense(64, activation='sigmoid')
# Or:
layers.Dense(64, activation=tf.sigmoid)

# L1 regularization lambda=0.01 applied to the kernel matrix:
layers.Dense(64, kernel_regularizer=tf.keras.regularizers.l1(0.01))

# L2 regularization lambda=0.01 applied to the bias vector:
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# kernel initialized to a random orthogonal matrix:
layers.Dense(64, kernel_initializer='orthogonal')

# bias vector initialized to 2.0s:
layers.Dense(64, bias_initializer=tf.keras.initializers.constant(2.0))

#Add a softmax layers:
model.add(layers.Dense(10, activation='softmax'))
```

5. Compile:

```
model.compile(optimizer=tf.train.AdamOptimizer(0.001),
              loss='categorical_crossentropy', metrics=['accuracy'])

model.compile(optimizer=tf.train.RMSPropOptimizer(0.01),
              loss=tf.keras.losses.categorical_crossentropy,
              metrics=[tf.keras.metrics.categorical_accuracy])
```

Optimizers: https://www.tensorflow.org/api_docs/python/tf/train
Losses: https://www.tensorflow.org/api_docs/python/tf/keras/losses
Metrics: https://www.tensorflow.org/api_docs/python/tf/keras/metrics

6. Tensorized Datasets:

```
dataset = tf.data.Dataset.from_tensor_slices((data, labels))
dataset = dataset.batch(32).repeat()
val_dataset = tf.data.Dataset.from_tensor_slices((val_data, val_labels))
val_dataset = val_dataset.batch(32).repeat()
```

7. Fit (train):
   Same as Keras: https://www.tensorflow.org/api_docs/python/tf/keras/models/Model#fit

8. Evaluate/Predict:
```
model.evaluate(data, labels, batch_size=32)    # loss and metrics
Yhat = model.predict(data, batch_size=32)    # output last layer for data
```

9. Save/load weights
```
model.save_weights('./weights/my_model')
model.load_weights('./weights/my_model')
```

10. Save/load entire models:
```
model.save('my_model.h5')
model = tf.keras.models.load_model('my_model.h5')
```

Example

```
import tensorflow as tf
from tensorflow.keras import layers

# Generate dummy data
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)),
num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)),
num_classes=10)

#Tensorized Datasets:
dataset = tf.data.Dataset.from_tensor_slices((x_train, y_train))
dataset = dataset.batch(32).repeat()

#create a model:
model = tf.keras.Sequential()

# Create a sigmoid layer:
layers.Dense(64, activation='sigmoid')
# Or: layers.Dense(64, activation=tf.sigmoid)

# L1 regularization lambda=0.01 applied to the kernel matrix:
layers.Dense(64, kernel_regularizer=tf.keras.regularizers.l1(0.01))

# L2 regularization lambda=0.01 applied to the bias vector:
layers.Dense(64, bias_regularizer=tf.keras.regularizers.l2(0.01))

# kernel initialized to a random orthogonal matrix:
layers.Dense(64, kernel_initializer='orthogonal')

# bias vector initialized to 2.0s:
layers.Dense(64, bias_initializer=tf.keras.initializers.constant(2.0))

#Add a softmax layers:
model.add(layers.Dense(10, activation='softmax'))

#Compile:
model.compile(optimizer=tf.train.AdamOptimizer(0.001), loss='categorical_crossentropy', metrics=['acc
uracy'])

model.fit(x_train, y_train, epochs=20, batch_size=128)

model.evaluate(x_test, y_test, batch_size=32)    # loss and metrics

Yhat = model.predict(x_test, batch_size=32)     #see how the testing data is categorized
```

CNN Keras Tutorial

1. Import libraries

```
import numpy as np, os
import matplotlib.pyplot as plt
import matplotlib.image as mp
import tensorflow as tf
import keras as k
```

Alternatively, import functions (attributes) from Keras:

```
from keras.models import Model
from keras.layers import Input, Dense, Dropout
from keras.layers import Reshape, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical
```

2. Load the notMNIST data

```
trainX = np.load('notMNIST_small_trainX.npy')/255
trainY = np.load('notMNIST_small_trainY.npy')/1
```

3. Converting a label vector to one-hot-encoded array:

```
labels = [0,1,1,2,2,3,4,5]
Y = k.utils.to_categorical(labels, 6)
```

4. Whiten data

```
Xm   =    np.mean(trainX ,axis=1    ,keepdims=True)
#3d vect: np.mean(array3d,axis=(1,2),keepdims=True)

Xstd =    np.std(trainX ,axis=1    ,keepdims=True)
#3d vect: np.std(array3d,axis=(1,2),keepdims=True)

Xnan      = np.where(Xstd==0)
Xstd[Xnan]= 1
Xm  [Xnan]= 0
trainX   -= Xm
trainX   /= Xstd
```

5. Split and Reshape:

```
percentV =0.2
percentT =0.2
m = trainX.shape[0]
ix = list(range(m))
np.random.shuffle (ix)                #randomize the DB index
X = trainX[ix,:]
Y = trainY[ix,:]

m_train = np.floor(m*(1-percentV-percentT)).astype(int) #cut off for training
m_val   = m_train + np.floor(m*percentV).astype(int)    #cut off for val data

X_train = X[       :m_val,:].reshape(m_val,28,28,1)     #graysc image are MxNx1
X_test  = X[  m_val:,     :].reshape(m-m_val,28,28,1)

Y_train = Y[       :m_val,:]
Y_test  = Y[  m_val:,     :]
```

6. Display an image:
```
ix = 10
plt.imshow(X_train[ix,:,:,0] ,cmap='gray')
```

7. Creating a Sequential Model:
```
model = k.Sequential()
```

8. Add a 2D convolutional layer:
```
model.add(k.layers.Conv2D(32, (3, 3),        #32 filters each 3x3
          padding='same',                    #
          input_shape=X_train[0].shape,   #for 1st layer. don't include batch
          data_format="channels_last",       #28x28x1 channel = 1
          name='conv1_l1_32x3x3'))                  #give each component a name
```

`strides = (1,1)` by default. To change use `stride = (a,b)`

9. Add a ReLU following 2D conv
```
model.add(k.layers.Activation('relu'))
```

Alternatively,
```
model.add(k.layers.Conv2D(32, (3, 3), activation = 'relu')
```

10. Add a Pooling Layer:
```
model.add(k.layers.MaxPooling2D(pool_size=(2, 2)))
```

Pooling layers: MaxPooling2D(), AveragePooling2D(), see: https://keras.io/layers/pooling/

11. When done with convolutional blocks, flatten:
```
model.add(Flatten())
```

12. Feed to Dense NNet:
```
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

13. Compile:
```
rmspr = keras.optimizers.rmsprop(lr=0.0001, decay=1e-6)
model.compile(loss='categorical_crossentropy',
              optimizer=rmspr,
              metrics=['accuracy'])
```

14. Train (aka Fit):
```
model.fit(X_train, Y_train,
          batch_size       =128,
          epochs           =300,
          validation_split =0.2,          #20% validation
          shuffle=True)
```

15. Real-time Data Augmentation and training:

```
 datagen = ImageDataGenerator(
     featurewise_center=False,               # set input mean to 0 over the dataset
     samplewise_center=False,                # set each sample mean to 0
     featurewise_std_normalization=False,    # divide inputs by std of the dataset
     samplewise_std_normalization=False,     # divide each input by its std
     zca_whitening=False,                    # apply ZCA whitening
     zca_epsilon=1e-06,                      # epsilon for ZCA whitening
     rotation_range=0,              # random rotate (degrees, 0 to 180)
     width_shift_range=0.1,         # random horiz shift (percent of width)
     height_shift_range=0.1,        # random horiz shift (percent of height)
     shear_range=0.,                # random shear
     zoom_range=0.,                 # set range for random scale
     channel_shift_range=0.,        # set range for random channel shifts
     # set mode for filling points outside the input boundaries
     fill_mode='nearest',
     cval=0.,                       # val used for fill_mode = "constant"
     horizontal_flip=True,
     vertical_flip=False,
     rescale=None,                  # set rescaling factor before transformation
     preprocessing_function=None,   # preprocess function
     data_format=None,
     validation_split=0.0)

datagen.fit(X_train)

model.fit_generator(datagen.flow(X_train, Y_train, batch_size =128),
                    epochs =300, validation_split =0.2)     #20% validatin
```

16. Test:

```
scores = model.evaluate(X_test, Y_test, verbose=1)
```