Lab 11. Convolutional Neural Network

Intro to Machine Learning

Fall 2018, Innopolis University

Today's Plan

- Why and What is Keras?
- Installing Keras and TensorFlow
- Fundamentals of Keras
- Understanding Keras Sequential Model
- Building a CNN in Keras (for image recognition) and classification of MNIST using CNN
- Home work

What is Keras

Keras is a high-level python API which can be used to quickly build and train neural networks using either Tensorflow or Theano as back-end.

- Currently, Keras is one of the fastest growing libraries for deep learning.

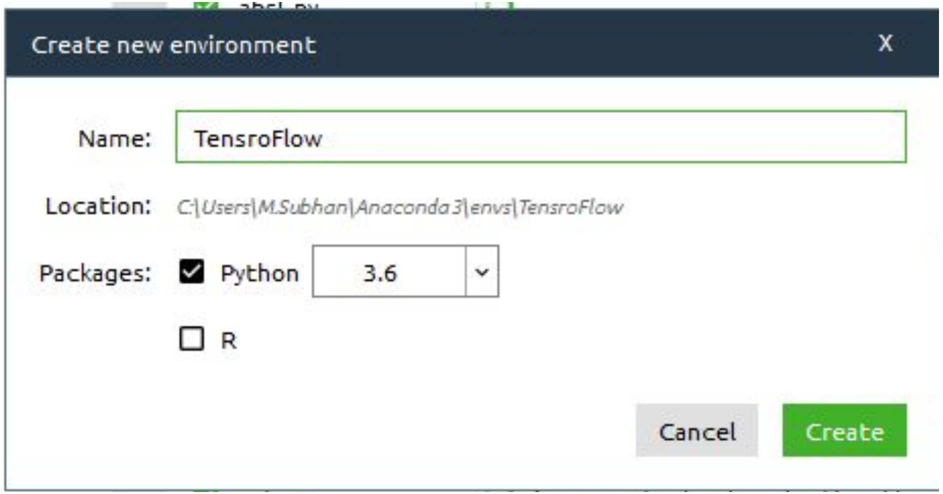
Why Keras

- Keras is being hailed as the future of building neural networks. Here are some of the reasons for its popularity:

- **Light-weight and quick**: Keras is designed to remove boilerplate code. Few lines of Keras code will achieve so much more than native Tensorflow code. You can easily design both CNN and RNNs and can run them on either GPU or CPU.

- **Emerging possible winner**: Keras is an API which runs on top of a back-end. This back-end could be either Tensorflow or Theano. Microsoft is also working to provide CNTK as a back-end to Keras. Currently, the world of Neural Network is very fragmented and evolving very fast.

Creating Environment



Open Anaconda Navigator

Go to Environments tab.

Create New Environment

Installing TensorFlow and Keras

pip install tensorflow Or **conda install -c conda-forge tensorflow**

OR

● To install the TensorFlow library, go to ***https://pypi.python.org/pypi/tensorflow/1.4.0/*** and look for a file named ***tensorflow-1.4.0-cp35-cp35m-win_amd64.whl***.

C:\>pip install C:\Keras\tensorflow-1.4.0-cp35-cp35m-win_amd64.whl

Installing TensorFlow and Keras

- First, install **h5py (pip install h5py)** for saving and restoring models and other dependencies such as **pip install pillow**
  - https://pypi.python.org/pypi/Keras/2.1.4/
- Look for file Keras-2.1.4-py2.py3-none-any.whl.
- C:\>pip install C:\Keras\Keras-2.1.4-py2.py3-none-any.whl

Installing TensorFlow and Keras

- To verify that Python, TensorFlow and Keras have been successfully installed, open a command shell and enter "python" to launch the Python interpreter.

- You'll see the ">>>" Python prompt. Then enter the commands.

- If you see the responses right, congratulations, you're ready to start writing machine learning code using Keras and TensorFlow.

```
C:\>python
>>> import tensorflow as tf
>>> tf.__version__
'1.4.0'
>>> import keras as K
Using TensorFlow backend.
>>> K.__version__
'2.1.4'
>>> exit()
C:\>
```

Fundamentals of Keras

- The main data structure in Keras is the **model** which provides a way to define the complete graph.

- Keras has two distinct ways of building models:

- **Sequential models:** This is used to implement simple models. You simply keep adding layers to the existing model.

- **Functional API:** Keras functional API is very powerful and you can build more complex models using it, models with multiple output, directed acyclic graph etc.

NN for MNIST Image Classification

- Load the Important libraries/packages.

➢ Import tensorflow as tf

Load the dataset.

```
## Loading MNIST Dataset from Keras split it into Training and test
mnist = tf.keras.datasets.mnist
# mnist is a dataset of 28x28 images of handwritten digits and their labels
(x_train, y_train),(x_test, y_test) = mnist.load_data()
# unpacks images to x_train/x_test and labels to y_train/y_test
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-dat
11493376/11490434 [==============================] - 2s 0us/step
```

NN for MNIST Image Classification

- Normalize the Data

```
## Normalize Data
x_train = tf.keras.utils.normalize(x_train, axis=1)
# scales data between 0 and 1
x_test = tf.keras.utils.normalize(x_test, axis=1)
# scales data between 0 and 1
```

NN for MNIST Image Classification

- Model

```
## Build a CNN Model
model = tf.keras.models.Sequential()
# a basic feed-forward model
model.add(tf.keras.layers.Flatten())
# takes our 28x28 and makes it 1x784
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
# a simple fully-connected layer, 128 units, relu activation
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
# a simple fully-connected layer, 128 units, relu activation
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
# our output layer. 10 units for 10 classes. Softmax for probability distribution
```

CNN for MNIST Image Classification

- Model Parameters and fit the Model

```
model.compile(optimizer='adam',  # Good default optimizer to start with
              loss='sparse_categorical_crossentropy',  # how will we calculate our "error." Neural network aims to minimize loss
              metrics=['accuracy'])  # what to track
```

```
## Train the Model
Emodel.fit(x_train, y_train, epochs=3)
```

```
Epoch 1/3
60000/60000 [==============================] - 10s 174us/step - loss: 0.2657 - acc: 0.9219
Epoch 2/3
60000/60000 [==============================] - 6s 106us/step - loss: 0.1097 - acc: 0.9654
Epoch 3/3
60000/60000 [==============================] - 7s 117us/step - loss: 0.0740 - acc: 0.9767

<tensorflow.python.keras.callbacks.History at 0x6a0d9e8>
```

Different Loss Functions

- Categorical cross entropy

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- Binary cross entropy

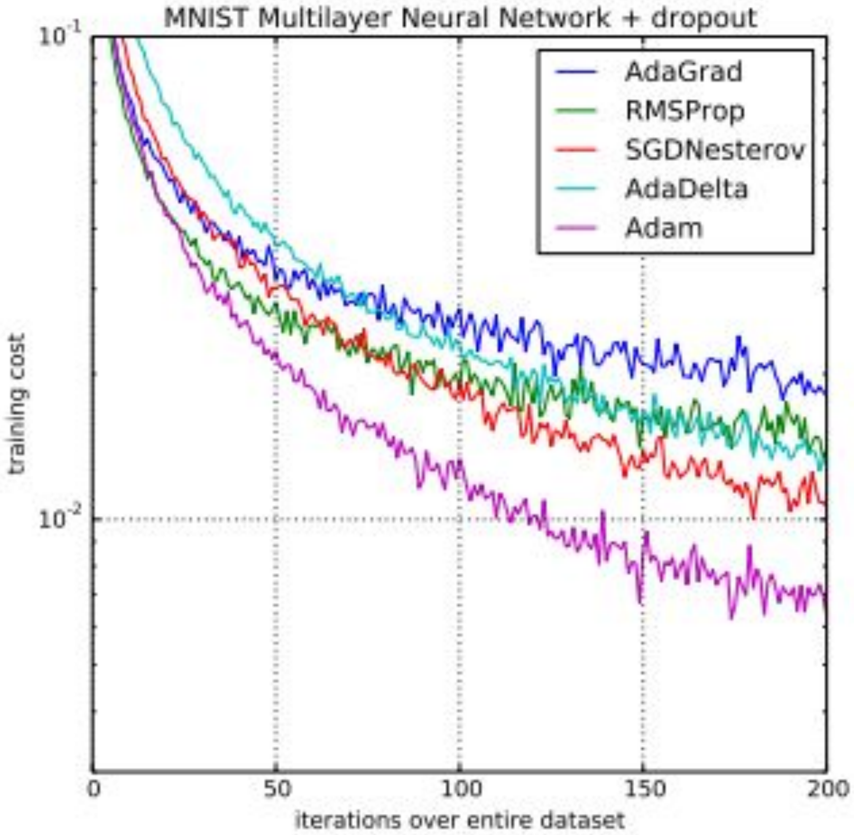
$$\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) = -\frac{1}{N}\sum_{i}^{N}[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)]$$

Optimizor

- Adam (adaptive moment estimation)

Default settings for the tested machine learning problems are alpha=0.001, beta1=0.9, beta2=0.999 and epsilon=10−8

http://ruder.io/optimizing-gradient-descent/



CNN for MNIST Image Classification

- Evaluate the Model

```
## Evaluate the Model
val_loss, val_acc = model.evaluate(x_test, y_test)
# evaluate the out of sample data with model
print(val_loss)
# model's loss (error)
print(val_acc)
# model's accuracy
```

```
10000/10000 [==============================] - 1s 131us/step
0.09341078321505338
0.9728
```

```
## Predict
predictions = model.predict(x_test)
print(predictions)
```

CNN for MNIST Image Classification

- Evaluate the Model

```
## Show the Predicted iamge
import numpy as np
import matplotlib.pyplot as plt
print(np.argmax(predictions[0]))
plt.imshow(x_test[0],cmap=plt.cm.binary)
plt.show()
```

```
7

<Figure size 640x480 with 1 Axes>
```

Image Recognition using CNN

1. Collecting the Dataset
2. Importing Libraries
3. Importing Image Datasets
4. Rescaling Data Set
5. Saving and reloading the datasets , Splitting the Dataset
6. Building the CNN
7. Testing

Image Recognition using CNN

○Collecting the Dataset

https://www.microsoft.com/en-us/download/confirmation.aspx?id=54765

○Importing Libraries and Packages

```
## Import Libraries for Reading List of Images
import numpy as np
import matplotlib.pyplot as plt
import os
import cv2    ## To do Some Image Operations.
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
```

Image Recognition using CNN

○Read and show image

```
DATADIR = "PetImages"
CATEGORIES = ["Dog", "Cat"]
for category in CATEGORIES:
    path = os.path.join(DATADIR,category)  # create path to dogs and cats
    for img in os.listdir(path):           # iterate over each image per dogs and cats
        img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
        plt.imshow(img_array, cmap='gray')
        plt.show()

        break  # we just want one for now so break
    break
```

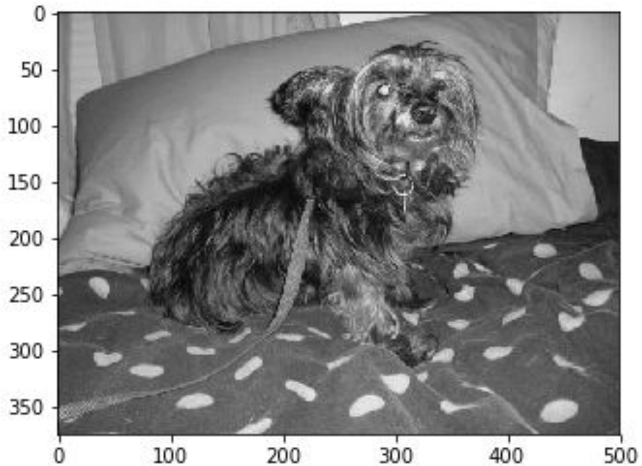


Image Recognition using CNN

○Resize the images

```
## Resize the image
IMG_SIZE = 50
new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))
plt.imshow(new_array, cmap='gray')
plt.show()
```

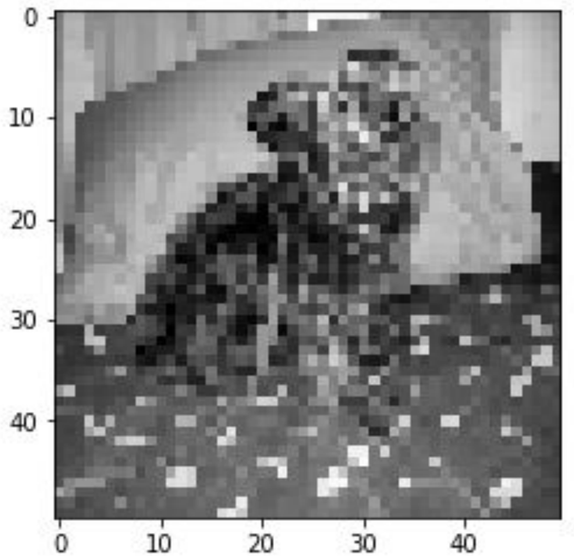


Image Recognition using CNN

○Building training data

```
## Building our training data
training_data = []
def create_training_data():
    for category in CATEGORIES:  # do dogs and cats
        path = os.path.join(DATADIR,category)  # create path to dogs and cats
        class_num = CATEGORIES.index(category)  # get the classification  (0 or 1). 0=dog 1=cat
        for img in tqdm(os.listdir(path)):  # iterate over each image per dogs and cats
            try:
                img_array = cv2.imread(os.path.join(path,img) ,cv2.IMREAD_GRAYSCALE)  # convert to array
                new_array = cv2.resize(img_array, (IMG_SIZE, IMG_SIZE))  # resize to normalize data size
                training_data.append([new_array, class_num])  # add this to our training_data
            except Exception as e:  # in the interest in keeping the output clean...
                pass
create_training_data()
print(len(training_data))
```

```
100%|██████████████████████████████████████| 12501/12501 [01:34<00:00, 131.73it/s]
100%|██████████████████████████████████████| 12501/12501 [01:52<00:00, 111.35it/s]
```

```
24946
```

Image Recognition using CNN

○Append Features and their Class labels

```
## Randomly Shuffle the training data
import random
random.shuffle(training_data)
#for sample in training_data[:10]:
#    print(sample[1])
## Append Features and their Class labels
X = []
y = []
for features,label in training_data:
    X.append(features)
    y.append(label)
print(X[0].reshape(-1, IMG_SIZE, IMG_SIZE, 1))
X = np.array(X).reshape(-1, IMG_SIZE, IMG_SIZE, 1)
```

Image Recognition using CNN

○Let's save the data to play with and to reload to the python to train the model

```
## Save the Data
import pickle
pickle_out = open("X.pickle","wb")
pickle.dump(X, pickle_out)
pickle_out.close()
pickle_out = open("y.pickle","wb")
pickle.dump(y, pickle_out)
pickle_out.close()
```

```
## Load the Data
pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)

pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
```

Image Recognition using CNN

○Let's start building CNN model: The basic CNN structure is as follows:

Convolution -> Pooling -> Convolution -> Pooling -> Fully Connected -> Output

```
## Import Libraries for Tensorflow and Keras
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import pickle ## Loading the dataset we just saved
```

```
## Read Dataset
pickle_in = open("X.pickle","rb")
X = pickle.load(pickle_in)
pickle_in = open("y.pickle","rb")
y = pickle.load(pickle_in)
X = X/255.0  ## Normalize Data (Since its a image data so
## min value will be 0 and max will be 255)
```

```python
## Creating CNN Model
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X.shape[1:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  #This converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

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

model.fit(X, y, batch_size=32, epochs=3, validation_split=0.3)
```

```
Train on 17462 samples, validate on 7484 samples
Epoch 1/3
17462/17462 [==============================] - 1253s 72ms/step - loss: 0.6874 - acc: 0.5635 - val_loss: 0.6596 - val_acc: 0.603
2
Epoch 2/3
17462/17462 [==============================] - 1302s 75ms/step - loss: 0.6487 - acc: 0.6243 - val_loss: 0.6475 - val_acc: 0.643
5
Epoch 3/3
```

Image Recognition using CNN

● **Building CNN:** This is most important step for our network. It consists of three parts -

○ **Convolution**: The primary purpose of Convolution is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.

○ **Polling**: Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information.

○ **Flattening:** After pooling comes flattening. Here the matrix is converted into a linear array so that to input it into the nodes of our neural network.

Summary

● The same concept can be applied to a diverse range of objects with a lot of training data and appropriate network. You can change the dataset with the images of your friends and work upon the network to make a Face Recognition Classifier.

●However there are many APIs available which can be automatically embedded into your application. They have been trained on a large dataset.

Homework

There is no specific template for today's homework. **You need to write a rigorous technical report on optimizing the network.** The important parameters you need to tune are:

Number of Hidden Layers and Units --- Activation Function --- Number of Epochs --- Batch size --- Different Loss Functions (categorical_crossentropy, sparse_categorical_crossentropy, binary_crossentropy) --- Different Optimization Methods (Stochastic gradient descent (sgd) and Adam).

You are required to write a report (sported with error and Accuracy graphs on above parameters) on which of the following method is appropriate for tuning the parameters with detailed illustration on MNIST dataset. Methods used to find out Hyperparameters are:

1): Manual Search

2): Grid Search
(https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/)