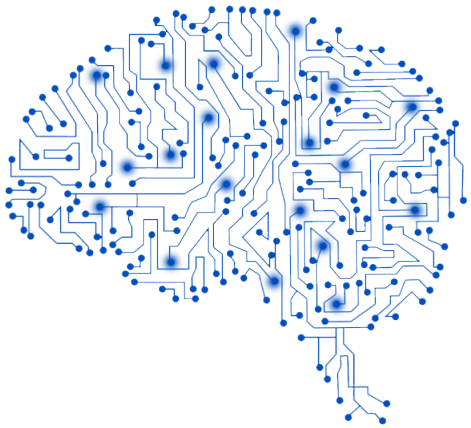

Deep Learning with Python Keras

Exercises

Manual by Themistoklis Diamantopoulos

Deep Learning with Python Keras
Notes for a tutorial compiled by Themistoklis Diamantopoulos

For more information check website:
`https://thdiaman.github.io/deeplearning/`
or check the following QR code:

1. Image Recognition

1.1 Solution using Convolutional Neural Network

1.1.1 Training

```
# Import basic libraries and keras
import os
from keras import backend as K
from keras.layers import Conv2D, MaxPooling2D
from keras.models import Sequential, load_model
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Dropout, Flatten, Dense

# Parameters of the model
img_width, img_height = 32, 32
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
nb_train_samples = 1000
nb_validation_samples = 400
epochs = 50
batch_size = 16

# Use the image data format of Tensorflow
input_shape = (img_width, img_height, 3)

# Check if there is a pre-trained model
if not os.path.exists('model.h5'):
    # Create a neural network with 3 convolutional layers and 2 dense layers
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(32, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))

    model.add(Flatten())
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='sigmoid'))

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

    # Perform augmentation
    train_datagen = ImageDataGenerator(rescale=1. / 255, shear_range=0.2,
                                       zoom_range=0.2, horizontal_flip=True)
    test_datagen = ImageDataGenerator(rescale=1. / 255)

    # Train the model
    train_generator = train_datagen.flow_from_directory(train_data_dir,
        target_size=(img_width, img_height), batch_size=batch_size,
            class_mode='categorical')
```

```
    validation_generator = test_datagen.flow_from_directory(
        validation_data_dir,
        target_size=(img_width, img_height), batch_size=batch_size,
            class_mode='categorical')
    model.fit_generator(train_generator, steps_per_epoch=nb_train_samples //
        batch_size,
        epochs=epochs, validation_data=validation_generator,
        validation_steps=nb_validation_samples // batch_size)

    # Save the model
    model.save('model.h5')
else:
    # Load the model from disk
    model = load_model('model.h5')

test_datagen = ImageDataGenerator(rescale=1. / 255)
validation_generator = test_datagen.flow_from_directory(validation_data_dir,
    target_size=(img_width, img_height), batch_size=batch_size, class_mode='
        categorical')
score = model.evaluate_generator(validation_generator, steps=
   nb_validation_samples // batch_size)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

1.1.2 Testing

```
# Import basic libraries and keras
import numpy as np
from scipy.misc import imresize
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.metrics import classification_report
from keras import backend as K
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator, load_img

# Parameters of the model
img_width, img_height = 32, 32
nb_test_samples = 400
batch_size = 16

# Load the model from disk
model = load_model('model.h5')

# Use the image data format of Tensorflow
input_shape = (img_width, img_height, 3)

# Run on test set
def run_on_test_set(test_set_path):
    test_datagen = ImageDataGenerator(rescale=1. / 255)
    test_generator = test_datagen.flow_from_directory(test_set_path, shuffle
        = False,
        target_size=(img_width, img_height), batch_size=batch_size,
            class_mode='categorical')
    predictions = model.predict_generator(test_generator, steps=
        nb_test_samples // batch_size)
```

```
    truevalues = [0] * 100 + [1] * 100 + [2] * 100 + [3] * 100
    predictedvalues = [np.argmax(p) for p in predictions]
    print(classification_report(truevalues, predictedvalues))

# Classify an image and optionally show it
def classify_image(image_path, plot = True):
    img = load_img(image_path)
    test_x = imresize(img, size=(img_height, img_width)).reshape(input_shape)
    test_x = test_x.reshape((1,) + test_x.shape)
    test_x = test_x / 255.0
    prediction = model.predict(test_x)
    predictedclass = np.argmax(prediction)
    predictedvalue = "airplane"
    if predictedclass == 1:
        predictedvalue = "automobile"
    elif predictedclass == 2:
        predictedvalue = "ship"
    elif predictedclass == 3:
        predictedvalue = "truck"

    if plot:
        img=mpimg.imread(image_path)
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.imshow(img)
        ax.set_title("This is " + predictedvalue)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        plt.show()

    return predictedvalue

run_on_test_set('data/test')
classify_image('data/test/airplanes/airplane.359.png')
```

1.2 Solution using Bottleneck Features on VGG16

1.2.1 Training

```
# Import basic libraries and keras
import os
import keras
import numpy as np
from keras import applications
from keras.models import Sequential, load_model
from keras.layers import Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator

# Parameters of the model
img_width, img_height = 32, 32
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
nb_train_samples = 1000
nb_validation_samples = 400
epochs = 50
batch_size = 20
```

```
# Generator used to load the data
datagen = ImageDataGenerator(rescale=1. / 255)

# Load the VGG16 network from disk
try:
    model = load_model("../ImageRecognition/vgg16_pretrained_imagenet.h5")
except:
    model = applications.VGG16(include_top=False, weights='imagenet')
    model.save("../ImageRecognition/vgg16_pretrained_imagenet.h5")

# Extract bottleneck features
if not os.path.exists('bottleneck_features_train.npy'):
    generator = datagen.flow_from_directory(train_data_dir, shuffle=False,
        target_size=(img_width, img_height), batch_size=batch_size,
            class_mode='categorical')
    bottleneck_features_train = model.predict_generator(generator,
        nb_train_samples // batch_size, verbose = 1)
    np.save(open('bottleneck_features_train.npy', 'wb'),
        bottleneck_features_train)

if not os.path.exists('bottleneck_features_validation.npy'):
    generator = datagen.flow_from_directory(validation_data_dir, shuffle=
        False,
        target_size=(img_width, img_height), batch_size=batch_size,
            class_mode='categorical')
    bottleneck_features_validation = model.predict_generator(generator,
        nb_validation_samples // batch_size, verbose = 1)
    np.save(open('bottleneck_features_validation.npy', 'wb'),
        bottleneck_features_validation)

# Load bottleneck features
train_data = np.load(open('bottleneck_features_train.npy', 'rb'))
trainquarter = nb_train_samples // 4
train_labels = np.array([0] * trainquarter + [1] * trainquarter + [2] *
    trainquarter + [3] * trainquarter)
train_labels = keras.utils.to_categorical(train_labels, 4)

validation_data = np.load(open('bottleneck_features_validation.npy', 'rb'))
validquarter = nb_validation_samples // 4
validation_labels = np.array([0] * validquarter + [1] * validquarter + [2] *
    validquarter + [3] * validquarter)
validation_labels = keras.utils.to_categorical(validation_labels, 4)

# Use the pretrained features network and add a connected network on top
if not os.path.exists('bottleneck_fc_model.h5'):
    # Create a neural network with 2 dense layers
    model = Sequential()
    model.add(Flatten(input_shape=train_data.shape[1:]))
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4, activation='sigmoid'))

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

    # Train the model
```

```
    model.fit(train_data, train_labels, epochs=epochs, batch_size=batch_size,
              verbose=1, validation_data=(validation_data, validation_labels)
                 )

    # Save the model
    model.save('bottleneck_fc_model.h5')
else:
    # Load the model from disk
    model = load_model('bottleneck_fc_model.h5')

score = model.evaluate(validation_data, validation_labels, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

1.2.2 Testing

```
# Import basic libraries and keras
import os
import numpy as np
from scipy.misc import imresize
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.metrics import classification_report
from keras import backend as K
from keras import applications
from keras.models import load_model
from keras.preprocessing.image import ImageDataGenerator, load_img

# Parameters of the model
img_width, img_height = 32, 32
nb_test_samples = 400
batch_size = 20

# Load the models from disk
try:
    vgg_model = load_model("vgg16_pretrained_imagenet.h5")
except:
    vgg_model = applications.VGG16(include_top=False, weights='imagenet')
    vgg_model.save("vgg16_pretrained_imagenet.h5")
model = load_model('bottleneck_fc_model.h5')

# Use the image data format of Tensorflow
input_shape = (img_width, img_height, 3)

# Run on test set
def run_on_test_set(test_set_path):
    test_datagen = ImageDataGenerator(rescale=1. / 255)
    if not os.path.exists('bottleneck_features_test.npy'):
        test_generator = test_datagen.flow_from_directory(test_set_path,
           shuffle=False,
            target_size=(img_width, img_height), batch_size=batch_size,
               class_mode='categorical')
        bottleneck_features_test = vgg_model.predict_generator(test_generator
           ,
            nb_test_samples // batch_size, verbose = 1)
```

```
        np.save(open('bottleneck_features_test.npy', 'wb'),
            bottleneck_features_test)
    else:
        bottleneck_features_test = np.load(open('bottleneck_features_test.npy
            ', 'rb'))
    predictions = model.predict_classes(bottleneck_features_test)
    truevalues = [0] * 100 + [1] * 100 + [2] * 100 + [3] * 100
    print(classification_report(truevalues, predictions))

# Classify an image and optionally show it
def classify_image(image_path, plot = True):
    img = load_img(image_path)
    test_x = imresize(img, size=(img_height, img_width)).reshape(input_shape)
    test_x = test_x.reshape((1,) + test_x.shape)
    test_x = test_x / 255.0
    bottleneck_features_test_x = vgg_model.predict(test_x)
    prediction = model.predict(bottleneck_features_test_x)
    predictedclass = np.argmax(prediction)
    predictedvalue = "airplane"
    if predictedclass == 1:
        predictedvalue = "automobile"
    elif predictedclass == 2:
        predictedvalue = "ship"
    elif predictedclass == 3:
        predictedvalue = "truck"

    if plot:
        img=mpimg.imread(image_path)
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.imshow(img)
        ax.set_title("This is a " + predictedvalue)
        ax.axes.get_xaxis().set_visible(False)
        ax.axes.get_yaxis().set_visible(False)
        plt.show()

    return predictedvalue

run_on_test_set('data/test')
classify_image('data/test/airplanes/airplane.359.png')
```

2. Text Classification

2.1 Solution using Fully Connected Neural Network

2.1.1 Training

```
# Import basic libraries and keras
import os
import json
import keras
import numpy as np
import keras.preprocessing.text as kpt
from keras.layers import Dense, Dropout
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, load_model
from dataload import load_data

# Load input data and labels (0 to 4)
train_x, train_y = load_data()

# Use the 3000 most popular words found in our dataset
max_words = 3000

# Tokenize the data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_x)
dictionary = tokenizer.word_index
# Save tokenizer dictionary to file
if not os.path.exists('dictionary.json'):
    with open('dictionary.json', 'w') as outfile:
        json.dump(tokenizer.word_index, outfile)

# For each text, change each token to its ID in the Tokenizer's word_index
allWordIndices = []
for text in train_x:
    words = kpt.text_to_word_sequence(text)
    wordIndices = [dictionary[word] for word in words]
    allWordIndices.append(wordIndices)

# Create matrix with indexed texts and categorical target
train_x = tokenizer.sequences_to_matrix(np.asarray(allWordIndices), mode='
    binary')
train_y = keras.utils.to_categorical(train_y, 5)

# Check if there is a pre-trained model
if not os.path.exists('model.h5'):
    # Create a neural network with 3 dense layers
    model = Sequential()
    model.add(Dense(512, input_shape=(max_words,), activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(256, activation='sigmoid'))
    model.add(Dropout(0.5))
    model.add(Dense(5, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics
        =['accuracy'])
```

```
    # Train the model
    model.fit(train_x, train_y, batch_size=32, epochs=10, verbose=1,
        validation_split=0.1, shuffle=True)

    # Save the model
    model.save('model.h5')
else:
    # Load the model from disk
    model = load_model('model.h5')
```

2.1.2 Testing

```
# Import basic libraries and keras
import json
import numpy as np
from keras.models import load_model
import keras.preprocessing.text as kpt
from keras.preprocessing.text import Tokenizer

# Load the dictionary and the model
with open('dictionary.json', 'r') as dictionary_file:
    dictionary = json.load(dictionary_file)
model = load_model('model.h5')

classes = ['business', 'entertainment', 'politics', 'sport', 'tech']

tokenizer = Tokenizer(num_words=3000)
while 1:
    print("\n\nRelevant types: " + ", ".join(classes))
    text = input('Input a sentence to evaluate its type, or press enter to
        quit: ')
    if len(text) == 0:
        break
    # Make the prediction
    words = kpt.text_to_word_sequence(text)
    wordIndices = [dictionary[word] for word in words if word in dictionary]
    testdata = tokenizer.sequences_to_matrix([wordIndices], mode='binary')
    pred = model.predict(testdata)[0]
    category = np.argmax(pred)
    print("The category is %s (confidence: %.2f%%)" % (classes[category], 100
        * max(pred)))
```

2.2 Solution using Convolutional Neural Network

2.2.1 Training

```
# Import basic libraries and keras
import os
import json
import keras
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, load_model
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Dense, Embedding, Conv1D, GlobalMaxPooling1D
```

```
from dataload import load_data

# Load input data and labels (0 to 4)
train_x, train_y = load_data()

# Use the 3000 most popular words found in our dataset
max_words = 3000

# Tokenize the data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_x)
dictionary = tokenizer.word_index
# Save tokenizer dictionary to file
if not os.path.exists('dictionary.json'):
    with open('dictionary.json', 'w') as outfile:
        json.dump(tokenizer.word_index, outfile)

# For each tweet, change each token to its ID in the Tokenizer's word_index
sequences = tokenizer.texts_to_sequences(train_x)
train_x = pad_sequences(sequences, maxlen=300)
train_y = keras.utils.to_categorical(train_y, 5)

# Check if there is a pre-trained model
if not os.path.exists('cnn_model.h5'):
    # Create a neural network with 3 dense layers
    model = Sequential()
    model.add(Embedding(3000, 64, input_length=300))
    model.add(Conv1D(filters=100, kernel_size=2, padding='valid', activation=
        'relu', strides=1))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(256, activation='relu'))
    model.add(Dense(5, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
        accuracy'])

    # Train the model
    model.fit(train_x, train_y, batch_size=32, epochs=10, verbose=1,
        validation_split=0.1, shuffle=False)

    # Save the model
    model.save('cnn_model.h5')
else:
    # Load the model from disk
    model = load_model('cnn_model.h5')
```

2.2.2 Testing

```
# Import basic libraries and keras
import json
import numpy as np
from keras.models import load_model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences

# Load the model, either 'cnn_model.h5' or 'lstm_model.h5'
model = load_model('cnn_model.h5')
```

```
# Load the dictionary and the model
with open('dictionary.json', 'r') as dictionary_file:
    dictionary = json.load(dictionary_file)

classes = ['business', 'entertainment', 'politics', 'sport', 'tech']

tokenizer = Tokenizer(num_words=3000)
tokenizer.word_index = dictionary
while 1:
    print("\n\nRelevant types: " + ", ".join(classes))
    text = input('Input a sentence to evaluate its type, or press enter to
       quit: ')
    if len(text) == 0:
        break
    # Make the prediction
    sequences = tokenizer.texts_to_sequences([text])
    padded_sequences = pad_sequences(sequences, maxlen=300)
    pred = model.predict(padded_sequences, verbose=0)[0]
    category = np.argmax(pred)
    print("The category is %s (confidence: %.2f%%)" % (classes[category], 100
        * max(pred)))
```

2.3 Solution using Recurrent Neural Network

2.3.1 Training

```
# Import basic libraries and keras
import os
import json
import keras
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential, load_model
from keras.layers import Dense, Embedding, LSTM
from keras.preprocessing.sequence import pad_sequences
from dataload import load_data

# Load input data and labels (0 to 4)
train_x, train_y = load_data()

# Use the 3000 most popular words found in our dataset
max_words = 3000

# Tokenize the data
tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_x)
dictionary = tokenizer.word_index
# Save tokenizer dictionary to file
if not os.path.exists('dictionary.json'):
    with open('dictionary.json', 'w') as outfile:
        json.dump(tokenizer.word_index, outfile)

# For each tweet, change each token to its ID in the Tokenizer's word_index
sequences = tokenizer.texts_to_sequences(train_x)
train_x = pad_sequences(sequences, maxlen=300)
train_y = keras.utils.to_categorical(train_y, 5)
```

```
# Check if there is a pre-trained model
if not os.path.exists('lstm_model.h5'):
    # Create a neural network with 3 dense layers
    model = Sequential()
    model.add(Embedding(3000, 64, input_length=300))
    model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(5, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['
        accuracy'])

    # Train the model
    model.fit(train_x, train_y, batch_size=32, epochs=10, verbose=1,
        validation_split=0.1, shuffle=True)

    # Save the model
    model.save('lstm_model.h5')
else:
    # Load the model from disk
    model = load_model('lstm_model.h5')
```

2.3.2 Testing

Same as Testing for Solution using Convolutional Neural Network

References

This chapter contains any references used to create this tutorial. In specific, it contains sources for the different source code parts of each section/example of this tutorial.

Image Recognition

- Building powerful image classification models using very little data: https://blog.keras.io/buildi ng-powerful-image-classification-models-using-very-little-data.html
- Object classification using CNN & VGG16 Model (Keras and Tensorflow): https://www.slidesh are.net/LalitJain29/object-classification-using-cnn-vgg16-model-keras-and-tensorflow

Text Classification

- Classifying Tweets with Keras and TensorFlow: https://vgpena.github.io/classifying-tweets-with-keras-and-tensorflow/
- Practical Neural Networks with Keras: Classifying Yelp Reviews: http://www.developintelligen ce.com/blog/2017/06/practical-neural-networks-keras-classifying-yelp-reviews/