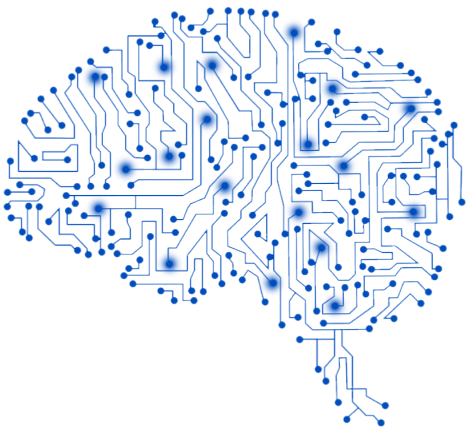

Deep Learning with Python Keras

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. Optical Character Recognition

1.1 Solution using Fully Connected Neural Network

1.1.1 Training

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

# Read the MNIST data and split to train and test
f = np.load('mnist.npz')
x_train, y_train = f['x_train'], f['y_train']
x_test, y_test = f['x_test'], f['y_test']
f.close()

# Optionally plot some images
#import matplotlib.pyplot as plt
#fig = plt.figure()
#for i in range(9):
#   plt.subplot(3,3,i+1)
#   plt.tight_layout()
#   plt.imshow(x_train[i], cmap='gray', interpolation='none')
#   plt.title("Digit: {}".format(y_train[i]))
#   plt.xticks([])
#   plt.yticks([])

# Reshape from (num_samples, 28, 28) to (num_samples, 784)
x_train = x_train.reshape(x_train.shape[0], 784)
x_test = x_test.reshape(x_test.shape[0], 784)
# Change type from int to float and normalize to [0, 1]
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# Optionally check the number of samples
#print(x_train.shape[0], 'train samples')
#print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices (transform the problem to
    multi-class classification)
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# 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, activation='relu', input_shape=(784,)))
    model.add(Dropout(0.2))
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(num_classes, activation='softmax'))
    model.summary()
    model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
       metrics=['accuracy'])

    # Train the model
    model.fit(x_train, y_train, batch_size=128, epochs=20, verbose=1,
              validation_data=(x_test, y_test))

    # Save the model
    model.save('model.h5')

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

# Get loss and accuracy on validation set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

1.1.2 Testing

```
# Import basic libraries and keras
import numpy as np
from keras.models import load_model

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

# Read the MNIST data and get test
f = np.load('mnist.npz')
x_test, y_test = f['x_test'], f['y_test']
f.close()

# Process the images as in training
x_test = x_test.reshape(x_test.shape[0], 784)
x_test = x_test.astype('float32')
x_test /= 255

# Make predictions
predictions = model.predict_classes(x_test, verbose=0)
correct_indices = np.nonzero(predictions == y_test)[0]
incorrect_indices = np.nonzero(predictions != y_test)[0]
print("Correct: %d" %len(correct_indices))
print("Incorrect: %d" %len(incorrect_indices))

# Optionally plot some images
#import matplotlib.pyplot as plt
#plt.figure()
#for i, correct in enumerate(correct_indices[:9]):
#     plt.subplot(3,3,i+1)
#     plt.tight_layout()
#     plt.imshow(x_test[correct].reshape(28,28), cmap='gray', interpolation='
    none')
#     plt.title("Predicted {}, Class {}".format(predictions[correct], y_test[
    correct]))
#
#plt.figure()
#for i, incorrect in enumerate(incorrect_indices[:9]):
#     plt.subplot(3,3,i+1)
#     plt.tight_layout()
#     plt.imshow(x_test[incorrect].reshape(28,28), cmap='gray', interpolation
    ='none')
#     plt.title("Predicted {}, Class {}".format(predictions[incorrect], y_test
    [incorrect]))
```

1.2 Solution using Convolutional Neural Network

1.2.1 Training

```
# Import basic libraries and keras
import os
import keras
import numpy as np
from keras.utils import np_utils
from keras.models import Sequential, load_model
from keras.layers import Dense, Dropout, Activation, Flatten, Convolution2D,
    MaxPooling2D

# Read the MNIST data and split to train and test
f = np.load('mnist.npz')
x_train, y_train = f['x_train'], f['y_train']
x_test, y_test = f['x_test'], f['y_test']
f.close()

# Change depth of image to 1
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
# Change type from int to float and normalize to [0, 1]
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

# Optionally check the number of samples
#print(x_train.shape[0], 'train samples')
#print(x_test.shape[0], 'test samples')

# Convert class vectors to binary class matrices (transform the problem to
    multi-class classification)
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

# Check if there is a pre-trained model
if not os.path.exists('cnn_model.h5'):
    # Create a neural network with 2 convolutional layers and 2 dense layers
    model = Sequential()
    model.add(Convolution2D(32, 3, 3, activation='relu', input_shape
        =(28,28,1)))
    model.add(Convolution2D(32, 3, 3, activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(10, activation='softmax'))

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

    # Train the model
    model.fit(x_train, y_train, batch_size=32, epochs=10, verbose=1,
        validation_data=(x_test, y_test))

    # Save the model
    model.save('cnn_model.h5')

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

# Get loss and accuracy on validation set
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

1.2.2 Testing

```
# Import basic libraries and keras
import numpy as np
from keras.models import load_model

# Load the model
model = load_model('cnn_model.h5')

# Read the MNIST data and get test
f = np.load('mnist.npz')
x_test, y_test = f['x_test'], f['y_test']
f.close()

# Process the images as in training
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
x_test = x_test.astype('float32')
x_test /= 255

# Make predictions
predictions = model.predict_classes(x_test, verbose=0)
correct_indices = np.nonzero(predictions == y_test)[0]
incorrect_indices = np.nonzero(predictions != y_test)[0]
print("Correct: %d" %len(correct_indices))
print("Incorrect: %d" %len(incorrect_indices))
```

```
# Optionally plot some images
#import matplotlib.pyplot as plt
#plt.figure()
#for i, correct in enumerate(correct_indices[:9]):
#     plt.subplot(3,3,i+1)
#     plt.tight_layout()
#     plt.imshow(x_test[correct].reshape(28,28), cmap='gray', interpolation='
    none')
#     plt.title("Predicted {}, Class {}".format(predictions[correct], y_test[
    correct]))
#
#plt.figure()
#for i, incorrect in enumerate(incorrect_indices[:9]):
#     plt.subplot(3,3,i+1)
#     plt.tight_layout()
#     plt.imshow(x_test[incorrect].reshape(28,28), cmap='gray', interpolation
    ='none')
#     plt.title("Predicted {}, Class {}".format(predictions[incorrect], y_test
    [incorrect]))
```

2. Image Recognition

2.1 Solution using Convolutional Neural Network

2.1.1 Training

```python
# 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 = 150, 150
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
nb_train_samples = 2000
nb_validation_samples = 800
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(1, 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='binary')
    validation_generator = test_datagen.flow_from_directory(
        validation_data_dir,
         target_size=(img_width, img_height), batch_size=batch_size,
             class_mode='binary')
    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='
        binary')
score = model.evaluate_generator(validation_generator, steps=
    nb_validation_samples // batch_size)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

2.1.2 Testing

```
# Import basic libraries and keras
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 = 150, 150
nb_test_samples = 800
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='binary')
    predictions = model.predict_generator(test_generator, steps=
        nb_test_samples // batch_size)
    truevalues = [0] * 400 + [1] * 400
    predictedvalues = [0 if p < 0.5 else 1 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)
    predictedvalue = "cat" if prediction < 0.5 else "dog"

    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/dogs/dog.1440.jpg')
```

2.2 Solution using Bottleneck Features on VGG16

2.2.1 Training

```
# Import basic libraries and keras
import os
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 = 150, 150
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
nb_train_samples = 2000
nb_validation_samples = 800
epochs = 50
batch_size = 16

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

# Load the VGG16 network from disk
try:
    model = load_model("vgg16_pretrained_imagenet.h5")
except:
    model = applications.VGG16(include_top=False, weights='imagenet')
    model.save("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=None)
    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=None)
    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'))
train_labels = np.array([0] * (nb_train_samples // 2) + [1] * (
   nb_train_samples // 2))

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

# 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(1, 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])
```

2.2.2 Testing

```python
# 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 = 150, 150
nb_test_samples = 800
batch_size = 16

# 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=None)
        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(bottleneck_features_test)
    truevalues = [0] * 400 + [1] * 400
    predictedvalues = [0 if p < 0.5 else 1 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
    bottleneck_features_test_x = vgg_model.predict(test_x)
    prediction = model.predict(bottleneck_features_test_x)
    predictedvalue = "cat" if prediction < 0.5 else "dog"

    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/dogs/dog.1440.jpg')
```

3. Text Classification

3.1 Solution using Fully Connected Neural Network

3.1.1 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

# Load input data
training = np.genfromtxt('15000tweets.csv', delimiter=',', skip_header=1,
    usecols=(1, 3), dtype=None)

# Get tweets and sentiments (0 or 1)
train_x = [str(x[1]) for x in training]
train_y = np.asarray([x[0] for x in training])

# 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
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 tweets and categorical target
train_x = tokenizer.sequences_to_matrix(np.asarray(allWordIndices), mode='
    binary')
train_y = keras.utils.to_categorical(train_y, 2)

# 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(2, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics
        =['accuracy'])

    # Train the model
    model.fit(train_x, train_y, batch_size=32, epochs=5, 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')
```

3.1.2 Testing

```
# Import basic libraries and keras
import json
from keras.models import load_model
from keras.preprocessing.text import Tokenizer, text_to_word_sequence

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

tokenizer = Tokenizer(num_words=3000)
while 1:
    text = input('Input a sentence to evaluate its sentiment, or press enter
        to quit:')
    if len(text) == 0:
        break
    # Make the prediction
    words = 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]
    print("The sentiment is %s (confidence: %.2f%%)" % ("negative" if pred[0]
         > pred[1] else "positive", 100 * max(pred)))
```

3.2 Solution using Convolutional Neural Network

3.2.1 Training

```
# Import basic libraries and keras
import os
import json
import numpy as np
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

# Load input data
training = np.genfromtxt('15000tweets.csv', delimiter=',', skip_header=1,
    usecols=(1, 3), dtype=None)

# Get tweets and sentiments (0 or 1)
train_x = [str(x[1]) for x in training]
train_y = np.asarray([x[0] for x in training])

# 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)

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

    # Train the model
    model.fit(train_x, train_y, batch_size=32, epochs=5, verbose=1,
        validation_split=0.1, shuffle=True)

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

3.2.2 Testing

```
# Import basic libraries and keras
import json
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)

tokenizer = Tokenizer(num_words=3000)
tokenizer.word_index = dictionary
while 1:
    text = input('Input a sentence to evaluate its sentiment, 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)
    print("The sentiment is %s (confidence: %.2f%%)" % ("positive" if pred >
       0.5 else "negative", 100 * max(pred, 1 - pred)))
```

3.3 Solution using Recurrent Neural Network

3.3.1 Training

```
# Import basic libraries and keras
import os
import json
import numpy as np
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

# Load input data
training = np.genfromtxt('15000tweets.csv', delimiter=',', skip_header=1,
   usecols=(1, 3), dtype=None)

# Get tweets and sentiments (0 or 1)
train_x = [str(x[1]) for x in training]
train_y = np.asarray([x[0] for x in training])

# 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)

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

    # Train the model
    model.fit(train_x, train_y, batch_size=32, epochs=5, 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')
```

3.3.2 Testing

Same as Testing for Solution using Convolutional Neural Network

4. Text Generation

4.1 Solution using Recurrent Neural Network

4.1.1 Training

```
# Import basic libraries and keras
import os
import json
from keras import layers
from keras.layers import LSTM
from keras.models import Model
from keras.models import load_model
from stories import get_stories, vectorize_stories

# Set parameters
EMBED_HIDDEN_SIZE = 50
SENT_HIDDEN_SIZE = 100
QUERY_HIDDEN_SIZE = 100
BATCH_SIZE = 32
EPOCHS = 40

# Load input data
train = get_stories('qa1_single-supporting-fact_train.txt')
test = get_stories('qa1_single-supporting-fact_test.txt')

# Create vocabulary
vocab = set()
for story, q, answer in train + test:
    vocab |= set(story + q + [answer])
vocab = sorted(vocab)

# Create index of words {word: id}
# Reserve 0 for masking via pad_sequences
vocab_size = len(vocab) + 1
word_idx = dict((c, i + 1) for i, c in enumerate(vocab))
# Get maximum length of sequences
story_maxlen = max(map(len, (x for x, _, _ in train + test)))
query_maxlen = max(map(len, (x for _, x, _ in train + test)))

# Save vocabulary and lengths to file
if not os.path.exists('dictionary.json'):
    with open('dictionary.json', 'w') as outfile:
        json.dump(word_idx, outfile)
if not os.path.exists('lengths.json'):
    with open('lengths.json', 'w') as outfile:
        json.dump({'story_maxlen': story_maxlen, 'query_maxlen': query_maxlen
           }, outfile)

# Vectorize the stories
x, xq, y = vectorize_stories(train, word_idx, story_maxlen, query_maxlen)
tx, txq, ty = vectorize_stories(test, word_idx, story_maxlen, query_maxlen)

# Check if there is a pre-trained model
if not os.path.exists('rnn_model.h5'):
    # Create a neural network for the stories
    sentence = layers.Input(shape=(story_maxlen,), dtype='int32')
    encoded_sentence = layers.Embedding(vocab_size, EMBED_HIDDEN_SIZE)(
        sentence)
    encoded_sentence = layers.Dropout(0.3)(encoded_sentence)

    # Create a neural network for the questions
    question = layers.Input(shape=(query_maxlen,), dtype='int32')
    encoded_question = layers.Embedding(vocab_size, EMBED_HIDDEN_SIZE)(
        question)
    encoded_question = layers.Dropout(0.3)(encoded_question)
    encoded_question = LSTM(EMBED_HIDDEN_SIZE)(encoded_question)
    encoded_question = layers.RepeatVector(story_maxlen)(encoded_question)

    # Combine the two networks
    merged = layers.add([encoded_sentence, encoded_question])
    merged = LSTM(EMBED_HIDDEN_SIZE)(merged)
    merged = layers.Dropout(0.3)(merged)
    preds = layers.Dense(vocab_size, activation='softmax')(merged)
    model = Model([sentence, question], preds)
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics
        =['accuracy'])

    # Train the model
    model.fit([x, xq], y, batch_size=BATCH_SIZE, epochs=EPOCHS,
        validation_split=0.05)

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

model.summary()
score = model.evaluate([tx, txq], ty, batch_size=BATCH_SIZE, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

4.1.2 Testing

```
# Import basic libraries and keras
import json
import numpy as np
from keras.models import load_model
from stories import get_stories, tokenize
from keras.preprocessing.sequence import pad_sequences

# Load test data
test_stories = get_stories('qa1_single-supporting-fact_test.txt')

# Load model, word index and lengths
model = load_model('rnn_model.h5')
with open('dictionary.json', 'r') as dictionary_file:
    word_idx = json.load(dictionary_file)
with open('lengths.json', 'r') as lengths_file:
    lengths = json.load(lengths_file)
    story_maxlen, query_maxlen = lengths["story_maxlen"], lengths["
        query_maxlen"]

# Get the word index of a question in a safe way
def best_effort_word_idxs(words):
    idxs = []
    for word in words:
        forms = [word, word.lower(), word[0].upper() + word[1:].lower()]
        for form in forms:
            if form in word_idx:
                idxs.append(word_idx[form])
    return idxs

while True:
    # Get a random story
    n = np.random.randint(0, 1000)
    story = test_stories[n][0]
    storystr = ' '.join(word for word in story)
    storymlstr = '\n'.join(storystr.split(' . '))[:-2]
    print(60 * '-')
    print('Story:\n' + storymlstr + '\n')

    # Request for a question
    print('Allowed vocabulary for questions:\n' + ' , '.join(word_idx.keys())
        )
    question = input('Enter your question (or press Enter to exit): ')
    if question == '':
        break

    # Tokenize story and question
    x = [word_idx[w] for w in story]
    xq = best_effort_word_idxs(tokenize(question)) #[word_idx[w] for w in
        tokenize(question)]
    if len(xq) < 1:
        print("Question is not valid")
        print(60 * '-')
        input("Press Enter to continue...\n")
        continue

    # Vectorize story and question
    tx, txq = pad_sequences([x], maxlen=story_maxlen), pad_sequences([xq],
        maxlen=query_maxlen)

    # Predict and print the result
    pred_results = model.predict(([tx, txq]))
    val_max = np.argmax(pred_results[0])
    for key, val in word_idx.items():
        if val == val_max:
            k = key
    print("Answer is: %s (confidence %.2f%%)" %(k, 100 * pred_results[0][
        val_max]))
    print(60 * '-')
    input("Press Enter to continue...\n")
```

5. Game Playing

5.1 Catch

```
import os
import matplotlib.pyplot as plt
from keras.optimizers import sgd
from keras.layers.core import Dense
from keras.models import Sequential, load_model

from qcatch import Catch
from qlearning import train, test

plt.ion()
plt.show()

# parameters
max_memory = 500 # Maximum number of experiences we are storing
batch_size = 1 # Number of experiences we use for training per batch
grid_size = 10 # Size of the playing field

# Check if there is a pre-trained model
if not os.path.exists('model.h5'):
    model = Sequential()
    model.add(Dense(100, input_shape=(grid_size**2,), activation='relu'))
    model.add(Dense(100, activation='relu'))
    model.add(Dense(3)) # There are 3 actions: [move_left, stay, move_right]
    model.compile(sgd(lr=.1), "mse")
    model.summary()

    # Train by playing the game - change visualize to True to also visualize
        the training
    env = Catch(grid_size)
    epochs = 5000
    model = train(model, epochs, env, max_memory, batch_size, verbose=1,
        visualize=False)

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

# Play game
env = Catch(grid_size)
test(model, env)
```

5.2 Maze

```
# Import basic libraries and keras
import os
import numpy as np
import matplotlib.pyplot as plt
from keras.layers.core import Dense
from keras.models import Sequential, load_model
from keras.layers.advanced_activations import PReLU

from qlearning import qtrain
from qmaze import Qmaze, play_game

plt.ion()
plt.show()

maze =  np.array([
    [ 1.,  0.,  1.,  1.,  1.,  1.,  1.],
    [ 1.,  1.,  1.,  0.,  0.,  1.,  0.],
    [ 0.,  0.,  0.,  1.,  1.,  1.,  0.],
    [ 1.,  1.,  1.,  1.,  0.,  0.,  1.],
    [ 1.,  0.,  0.,  0.,  1.,  1.,  1.],
    [ 1.,  0.,  1.,  1.,  1.,  1.,  1.],
    [ 1.,  1.,  1.,  0.,  1.,  1.,  1.]
])

# Check if there is a pre-trained model
if not os.path.exists('model.h5'):
    model = Sequential()
    model.add(Dense(maze.size, input_shape=(maze.size,)))
    model.add(PReLU())
    model.add(Dense(maze.size))
    model.add(PReLU())
    model.add(Dense(4)) # num of actions
    model.compile(optimizer='adam', loss='mse')

    # Train by playing the game - change visualize to True to also visualize
        the training
    model = qtrain(model, maze, n_epoch=1000, max_memory=8*maze.size,
        data_size=32, visualize = False)

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

# Play game
play_game(model, Qmaze(maze), (0, 0), True)
```

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.

OCR

- MNIST in Keras: https://github.com/wxs/keras-mnist-tutorial/blob/master/MNIST%20in%2 0Keras.ipynb
- A simple 2D CNN for MNIST digit recognition: https://towardsdatascience.com/a-simple-2d-cnn-for-mnist-digit-recognition-a998dbc1e79a

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/

Text Generation

- Question answering on the Facebook bAbi dataset using recurrent neural networks and 175 lines of Python + Keras: http://smerity.com/articles/2015/keras_qa.html

Neural Doodle & Style Transfer

- Neural Style Transfer In Keras: https://markojerkic.com/style-transfer-keras/

Game Playing

- Deep reinforcement learning: where to start: https://medium.freecodecamp.org/deep-reinforce ment-learning-where-to-start-291fb0058c01
- Deep Reinforcement Learning for Maze Solving: http://www.samyzaf.com/ML/rl/qmaze.html