Neural Nets with Keras

Intel-Unesp/NCC - Machine Learning and High Performance Computing Team

Julio Antonio do Amaral - *julioamaral@ncc.unesp.br ou julioa@ita.br*

July 10, 2017

Overview

- Introduction to Keras: The Python Deep Learning Library
- Sequential Model
- Model Class API
- Sequential Model: Recognizing Handwritten Digits (Single and MLP Nets)
- Optimizers in Keras
- How to Avoid Overfitting!
- Introduction to Convolutional Nets

Keras:The Python Deep Learning library

Keras is a high-level neural networks API, written in Python and capable of running on top of either TensorFlow, CNTK or Theano.

Keras is compatible with: Python 2.7-3.6.

Sequential Model

The initial building block of Keras is a model, and the simplest model is called sequential.

A sequential Keras model is a linear pipeline (a stack) of **neural networks layers**.

An example of a Neural Net in Keras:

The code below defines a single (dense) layer with 12 artificial neurons, and 8 input variables (**features**):

```
from keras.models import Sequential
model = Sequential()
model.add(Dense(12, input_dim=8, kernel_initializer='random_uniform'))
```

Sequential Model

Kernel Initializer

Each neuron can be initialized with specific weights.

Keras provides a few choices, the most common of which are:

- `'random_uniform'`: `Weights are initialized to uniformly random small values in (-0.05, 0.05)`
- `'random_normal'`: `Weights are initialized according to a Gaussian, with a zero mean and small standard deviation of 0.05.`
- `zero`: `All weights are initialized to zero.`

Find out more in: https://keras.io/initializations/.

Sequential Model

Compilation: Before training a model it is necessary to configure the learning process. It is done by compile method.

This method receives three arguments:

1. An Optimizer such as rmsprop or adam;
2. Objective Function - "loss function". This is the objective that the model will try to minimize;
3. A list of metrics: metrics=['accuracy'].

Sequential Model

Compilation

- *For a multi-class classification problem*

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

...
...
```

Sequential Model

Compilation

- *For a binary classification problem*

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

Sequential Model

Compilation

- *For regression problem*

```
...
...
model.compile(optimizer='rmsprop',
              loss='mse')
...
...
```

Sequential Model

Training

- *Keras runs over Numpy!*
- *For training a model use the fit function.*

For binary classification:

```
...
...
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels, epochs=10, batch_size=32)
```

Sequential Model

Training

For categorical classification:

```
...
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
targets = keras.utils.to_categorical(labels, num_classes=10)
model.fit(data, targets, epochs=10, batch_size=32)
...
```

Sequential Model

Example from https://keras.io/getting-started/sequential-model-guide/

Multilayer Perceptron (MLP) for multi-class *softmax* classification:

Inserting Headers

```
...
...
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
...
...
```

Sequential Model

Creating Dummy Data

```
...
...
import numpy as np
x_train = np.random.random((1000, 20))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1))
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1))
...
...
```

Sequential Model

Creating a MLP Neural Net

```
...
model = Sequential()
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'))
...
```

Sequential Model

Compiling the Neural Net Model

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

Sequential Model

Training/Evaluating the Neural Model

```
...
model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
...
```

Model Class API

In the functional API, you can create a Model via:

```
from keras.models import Model
from keras.layers import Input, Dense

a = Input(shape=(32,))
b = Dense(32)(a)
model = Model(inputs=a, outputs=b
...
)
```

Note that his model include all layers required in the computation of a given b

Model Class API

In the case of multi-input multi-output models, we can use:

```
...
...
model = Model(inputs=[a1, a2], outputs=[b1, b3, b3])
...
...
```

Recognizing Handwritten Digits!

In this section we build a network that can recognize Handwritten Digits.

For this, it will be used the MNIST Database (http://yann.lecun.com/exdb/mnist/), a set made up of 60.000 examples and a test set of 10.000 examples.

This training examples are annotated by humans with the correct answer. If the handwritten digit is the number three, then three is simply the label associated with that example.

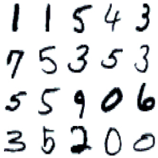

Figure: MNIST Digits

Recognizing Handwritten Digits!

Step 1 - Spliting the Data Set

```
...
...
(X_train, y_train), (X_test, y_test) = mnist.load_data()
...
RESHAPED = 784
...
...
```

X_train has 60000 samples of 28x28, reshaped in 60000x784 inputs

Recognizing Handwritten Digits!

Step 2 - Modeling the Net

```
...
...
model = Sequential()
model.add(Dense(NB_CLASSES, input_shape=(RESHAPED,)))
model.add(Activation('softmax'))
...
...
```

The input layer has a neuron for each picel in the image (784 neurons) and the final layer has a single neuron with softmax activation - a generalization of sigmoid functions

Recognizing Handwritten Digits!

Step 2 - Compiling the Net

```
...
...
model.compile(loss='categorical_crossentropy',
              optimizer=OPTIMIZER,
              metrics=['accuracy'])
...
...
```

The compiled model now can be executed by Keras over Theano or TensorFlow

Recognizing Handwritten Digits!

Step 2 - Training the Net

```
...
...
history = model.fit(X_train, Y_train,
batch_size=BATCH_SIZE, epochs=NB_EPOCH,
verbose=VERBOSE, validation_split=VALIDATION_SPLIT)
...
...
```

Remember!:

epochs:the number of times the model is exposed to the training set;
batch_size:number of training instances observed before the optimizer performs a weight update.

Recognizing Handwritten Digits!

Step 2 - Evaluating the Model

```
...
...
score = model.evaluate(X_test, Y_test, verbose=VERBOSE)
print("Test score:", score[0])
print('Test accuracy:', score[1])
...
...
```

Now we can evaluate the neural model on the test set that contains new ***unseen*** *examples.*

Note that the training set and the test set are rigorously separated. Learning is a process intended to generalize unseen observationsand not to memorize what is already know!

Recognizing Handwritten Digits!

Step 2 - Run the Source Code

Code: NeuralNetpy - https://github.com/julioaamaral/ml/blob/master/NeuralNet.py

Results got with 50 epochs: loss: 0.3289; accuracy:0.9087

Check it out and try to improve them! - Test this code with others parameters!

Hidden Layers

Step 2 - MultiLayer Perceptron in Keras

```
...
...
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
model.compile(loss='categorical_crossentropy',
...
...
```

Recognizing Hanwritten Digits!

Step 2 - Run the Source Code

Code: MLP.py - https://github.com/julioaamaral/ml/blob/master/MLP.py

Results got with 50 epochs: loss: ? accuracy:?

Check it out and try to improve them! - Test this code with others parameters!

Hidden Layers

Step 2 - MultiLayer Perceptron (with Dropout) in Keras

```
...
...
model = Sequential()
model.add(Dense(N_HIDDEN, input_shape=(RESHAPED,)))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(N_HIDDEN))
model.add(Activation('relu'))
model.add(Dropout(DROPOUT))
model.add(Dense(NB_CLASSES))
model.add(Activation('softmax'))
model.summary()
...
...
```

Recognizing Hanwritten Digits!

Step 2 - Run the Source Code

Code: MLP_Dropout.py - https://github.com/julioaamaral/ml/blob/master/MLP_Dropout.py

Results got with 50 epochs: loss: ? accuracy:?

Check it out and try to improve them! - Test this code with others parameters!

Testing different optimizers in Keras

```
from keras.optimizers import SGD, RMSprop, Adam
...
OPTIMIZER = RMSprop() # optimizer
```

Testing different optimizers in Keras

Hands On! Run the code **MLP_Dropout.py** using differents optimizers!

Code address: https://github.com/julioaamaral/ml/blob/master/MLP_Dropout.py

Regularization for Avoid Overfitting

keras offers different types of regularization. Take a look!

- L1 - also know as **lasso**
- L2 - also know as **ridge**
- Elastic net regularization

Note that the same ideia of regularization can be applied indepentently to the weights, to the model, and to the activation.

```
...
...
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01)))
...
...
```

Predicting Output

When a net is trained, it can be used for prediction. In Keras we use the following method:

```
...
...
#for a given input x
predictions = model.predict(X)
...
...
```

An Introduction to Convolutional Nets

Convolutional neural networks (also called ConvNet) leverage spatial information and are therefore very well suited for classifying images.

These nets use an ad hoc architecture inspired by biological data taken from physiological experiments doneon the visual cortex.

Note that our vision system is based on multiple cortex levels; each one applied on recognizing more and more structured information. From single pixels to sophisticated elements such as objects, faces, etc.

Deep Convolutional Neural Nets - DCNN

A ***Deep Convolutional Neural Net*** consist of many neural layers. Two different types convolutional and pooling as typically used alternated.

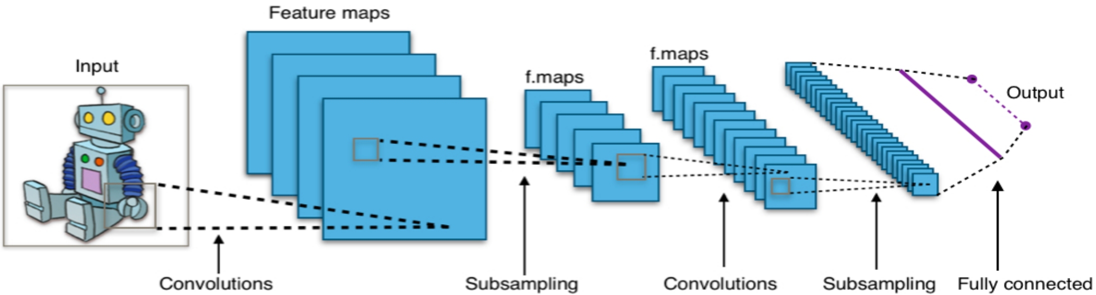

Figure: ConvNet

Modeling ConvNets

In Keras, to add a convolutional layer with dimensionality of the output 32 and kernel with 3x3, we use:

```
...
...
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(256, 256, 3))
...
...
```

or

```
...
...
model = Sequential()
model.add(Conv2D(32, kernel_size=3, input_shape=(256, 256, 3))
...
...
```

Modeling ConvNets

Note that the input (image) in the last example has shape (256,256).

With TensorFlow this input, consider three channels (RGB), is represented as (256,256,3) and with Theano, (3,256,256).

Modeling ConvNets

Max-Pooling

One common choice for *Pooling* Layer is max-pooling, which simply outputs the maximum activation as observed in the region. In Keras, to define a max-pooling layer of size 2 x 2, we code:

```
...
...
model.add(MaxPooling2D(pool_size = (2, 2)))
...
...
```

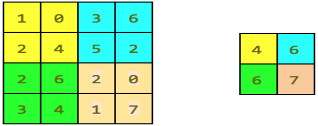

Figure: MaxPooling Filter

An example of DCNN - LeNet

Yann le Cun proposed a family of ConvNets named LeNet trained for recognizing MNIST handwritten characters with robustness to simple geometric transformations and to distortion.

The key intuition here is to have low-layers alternating convolution operations with max-pooling operations.

The convolution operations are based on carefully chosen local receptive fields with shared weights for multiple feature maps.

Higher levels are fully connected layers based on a traditional MLP with hidden layers and softmax as the output layer. *Take a look at: Convolutional Networks for Images, Speech, and Time-Series, by Y. LeCun and Y. Bengio, brain theory neural networks, vol. 3361, 1995*

LeNet in Keras

To define LeNet code, we use a convolutional 2D module, which is:

```
...
keras.layers.convolutional.Conv2D(filters,
                                  kernel_size,
                                  padding='valid')
...
```

In addition, we use a MaxPooling2D layer:

```
...
keras.layers.pooling.MaxPooling2D(pool_size=(2, 2),
                                  strides=(2, 2))
...
```

Coding the LeNet - Layers

Inserting modules:

```
from keras import backend as K
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.datasets import mnist
from keras.utils import np_utils
from keras.optimizers import SGD, RMSprop, Adam
import numpy as np
import matplotlib.pyplot as plt
...
...
```

Coding the LeNet - Layers

Defining the Net:

```
...
#define the ConvNet
class LeNet:
    @staticmethod
    def build(input_shape, classes):
        model = Sequential()
        # CONV => RELU => POOL
        model.add(Convolution2D(20, kernel_size=5, padding="same",
        input_shape=input_shape))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
...
```

Coding the LeNet - Layers

Defining the Net:

```
# CONV => RELU => POOL
model.add(Conv2D(50, kernel_size=5, border_mode="same"))
model.add(Activation("relu"))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
# Flatten => RELU layers
model.add(Flatten())
model.add(Dense(500))
model.add(Activation("relu"))
```

...

Coding the LeNet - Layers

Defining the Net:

```
...
        # a softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))
    return model
```

Coding the LeNet - Layers

Topology of the Net:

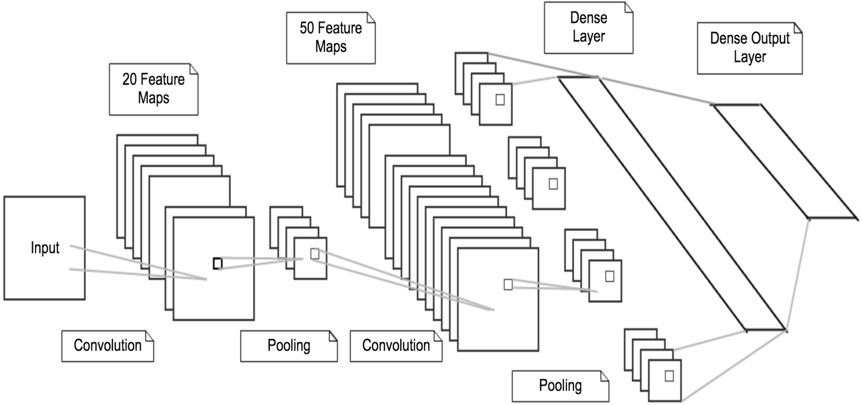

Figure: LeNet

LeNet - HandsOn!

Hands On! Run the code **LeNet.py**!

Code: LeNet.py - https://github.com/julioaamaral/ml/blob/master/LeNet.py

Results got with 20 epochs: loss: ? accuracy:?

Check it out and try to improve them! - Test this code with others parameters!

References

- HAYKIN, S. Neural Netwoks, NJ:Prentice Hall, 2009
- Keras Documentation - https://keras.io/