Deep Learning with Keras

- Perceptron
- Generative Adversarial Networks and WaveNet
- Word Embeddings
- Recurrent Neural Network — RNN

Keras API

Keras has a modular, minimalist, and easy extendable architecture. Francois Chollet, the author of Keras, says:

The library was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

Keras defineds high-level neural networks running on top of either TensorFlow or Theano

- Modularity: A model is either a sequence or a graph of standalone modules that can be combined together like LEGO blocks for building neural networks. Namely, the library predefines a very large number of modules implementing different types of neural layers, cost functions, optimizers, initialization schemes, activation functions, and regularization schemes.
- Minimalism: The library is implemented in Python and each module is kept short and self-describing.
- Easy extensiblity: The library can be extended with new functionalities, as we will describe in Chapter 7, Additional Deep Learning Models.

What is tensor?

Keras uses either Theano or TensorFlow to perform very efficient computations on tensors. But what is a tensor anyway? A tensor is nothing but a multidimensional array or matrix. Both the backends are capable of efficient symbolic computations on tensors, which are the fundamental building blocks for createing neural networks.

Composing models in Keras

There are two ways to composing models in Keras. They are as follows:

- Sequential composition
- Functional composition

Regular dense

snippet.python

```
keras.layers.core.Dense(units, activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None,
activity_regularizer=None, kernel_constraint=None,
bias_constraint=None)
```

Recurrent neural networks - simple, LSTM, and GRU

snippet.python

```
keras.layers.recurrent.Recurrent(return_sequences=False,
go_backwards=False, stateful=False, unroll=False, implementation=0)

keras.layers.recurrent.SimpleRNN(units, activation='tanh',
use_bias=True, kernel_initializer='glorot_uniform',
recurrent_initializer='orthogonal', bias_initializer='zeros',
kernel_regularizer=None, recurrent_regularizer=None,
bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, recurrent_constraint=None,
bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)

keras.layers.recurrent.GRU(units, activation='tanh',
recurrent_activation='hard_sigmoid', use_bias=True,
kernel_initializer='glorot_uniform',
recurrent_initializer='orthogonal', bias_initializer='zeros',
kernel_regularizer=None, recurrent_regularizer=None,
bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, recurrent_constraint=None,
bias_constraint=None, dropout=0.0, recurrent_dropout=0.0)

keras.layers.recurrent.LSTM(units, activation='tanh',
recurrent_activation='hard_sigmoid', use_bias=True,
kernel_initializer='glorot_uniform',
recurrent_initializer='orthogonal', bias_initializer='zeros',
unit_forget_bias=True, kernel_regularizer=None,
recurrent_regularizer=None, bias_regularizer=None,
activity_regularizer=None, kernel_constraint=None,
recurrent_constraint=None, bias_constraint=None, dropout=0.0,
recurrent_dropout=0.0)
```

Convolutional and pooling layers

snippet.python

```
keras.layers.convolutional.Conv1D(filters, kernel_size, strides=1,
padding='valid', dilation_rate=1, activation=None, use_bias=True,
kernel_initializer='glorot_uniform', bias_initializer='zeros',
kernel_regularizer=None, bias_regularizer=None,
activity_regularizer=None, kernel_constraint=None,
bias_constraint=None)

keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1),
padding='valid', data_format=None, dilation_rate=(1, 1),
activation=None, use_bias=True, kernel_initializer='glorot_uniform',
bias_initializer='zeros', kernel_regularizer=None,
bias_regularizer=None, activity_regularizer=None,
kernel_constraint=None, bias_constraint=None)

keras.layers.pooling.MaxPooling1D(pool_size=2, strides=None,
padding='valid')

keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None,
padding='valid', data_format=None)
```

Regularization

- kernel*regularizer: Regularizer function applied to the weight matrix - bias*regularizer: Regularizer function applied to the bias vector
- activity_regularizer: Regularizer function applied to the output of the layer (its activation)

In addition is possible to use Dropout for regularization and that is frequently a very effective choice

snippet.python

```
keras.layers.core.Dropout(rate, noise_shape=None, seed=None)
```

Where: - rate: It is a float between 0 and 1 which represents the fraction of the input units to drop - noise_shape: It is a 1D integer tensor which represents the shape of the binary dropout mask that will be multiplied with the input - seed: It is a integer which is used use as random seed

Batch normalization

snippet.python

```
keras.layers.normalization.BatchNormalization(axis=-1, momentum=0.99,
epsilon=0.001, center=True, scale=True, beta_initializer='zeros',
gamma_initializer='ones', moving_mean_initializer='zeros',
moving_variance_initializer='ones', beta_regularizer=None,
```

```
gamma_regularizer=None, beta_constraint=None, gamma_constraint=None)
```

An overview of optimizers

Optimizers include SGD, RMSprop, and Adam.

Using TensorBoard and Keras

Keras provides a callback for saving your training and test metrics, as well as activation histograms for the diffrerent layers in your model:

snippet.python

```
keras.callbacks.TensorBoard(log_dir='./logs', histogram_freq=0,
write_graph=True, write_images=False)
```

Saved data can then be visualized with TensorBoard launched at the command line:

snippet.shell

```
tensorboard --logdir=/full_path_to_your_logs
```

Using Quiver and Keras

snippet.python

```
pip install quiver_engine

from quiver_engine import server     server.launch(model)
```

Pooling layers

Max-pooling

snippet.python

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

Average pooling

LeNet code in Keras

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

snippet.python

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

Here, `filters` is the number of convolution kernels to use (for exampel, the dimensionality of the output), `kernel_size` is an integer or tuple/list of two integers, specifying the width the same value for all spatial dimensions), and `padding='same'` means that padding is used. There are two options: `padding='valid'` means that the convolution is only computed where the input and the filter fully overlap, and therefore the output is smaller than the input, while `padding='same'` means that we have an output that the same size as the input, for which the area around the input is padded with zeros.

In addition, we use a `MaxPooling2D` module:

snippet.python

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

Here, `pool_size=(2,2)` is a tuple of two integers representing the factors by which the image is vertically and horizontally downsacled. So (2,2) will halve the image in each dimension, and `strides=(2,2)` is the stride used for processing.

Now, let us review the code. First we import a number of modules:

snippet.python

```
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
```

Then we define the LeNet network:

snippet.python

```
#define the ConvNet
class LeNet:
    @staticmethod
    def build(input_shape, classes):
         model = Sequential()
         # CONV => RELU => POOL
```

snippet.python

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

관련 문서

- Keras