Python For Data Science Cheat Sheet

Keras

Learn Python for data science Interactively at www.DataCamp.com



Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2, size=(1000,1))
>>> 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)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the train test split module of sklearn.cross validation.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                                   cifar10,
                                   imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load data()
>>> (x train2,y train2), (x test2,y test2) = boston housing.load data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x train4,y train4), (x test4,y test4) = imdb.load data(num words=20000)
>>> num classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                     input dim=8,
                     kernel initializer='uniform',
                     activation='relu'))
>>> model.add(Dense(8,kernel initializer='uniform',activation='relu'))
>>> model.add(Dense(1, kernel initializer='uniform', activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> 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(10,activation='softmax'))
```

>>> model.add(Dense(64,activation='relu',input dim=train data.shape[1])) >>> model.add(Dense(1))

>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten

Convolutional Neural Network (CNN)

```
>>> model2.add(Conv2D(32,(3,3),padding='same',input shape=x train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool size=(2,2)))
>>> mode12.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool size=(2,2)))
>>> mode12.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Also see NumPy & Scikit-Learn

Preprocessing

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x train4 = sequence.pad sequences(x train4, maxlen=80)
>>> x test4 = sequence.pad sequences(x test4, maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to categorical
>>> Y train = to categorical(y train, num classes)
>>> Y test = to categorical(y test, num classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model selection import train test split
>>> X train5, X test5, y train5, y test5 = train test split(X,
                                                       test size=0 33.
                                                       random state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x train2)
>>> standardized X = scaler.transform(x train2)
>>> standardized X test = scaler.transform(x test2)
```

Inspect Model

```
Model output shape
>>> model.output shape
>>> model.summary()
                                      Model summary representation
>>> model.get config()
                                      Model configuration
>>> model.get weights()
                                     List all weight tensors in the model
```

Compile Model

```
MLP: Binary Classification
>>> model.compile(optimizer='adam',
                   loss='binary crossentropy',
                   metrics=['accuracy'])
MLP: Multi-Class Classification
>>> model.compile(optimizer='rmsprop',
                   loss='categorical crossentropy',
                   metrics=['accuracy'])
MLP: Regression
>>> model.compile(optimizer='rmsprop',
                   loss='mse',
                   metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x train4.
             y Train4,
             batch size=32,
             epochs=15,
             verbose=1,
             validation data=(x test4, y test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x test,
                                 y_test,
batch size=32)
```

Prediction

```
>>> model3.predict(x test4, batch size=32)
>>> model3.predict classes(x test4,batch size=32)
```

Save/Reload Models

```
>>> from keras.models import load model
>>> model3.save('model file.h5')
>>> my model = load model('my model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early stopping monitor = EarlyStopping(patience=2)
>>> model3.fit(x train4,
             y train4,
             batch size=32,
             epochs=15,
             validation data=(x test4, y test4),
             callbacks=[early_stopping_monitor])
```

DataCamp Learn Python for Data Science Interactively



Python For Data Science Cheat Sheet

NumPv Basics

Learn Python for Data Science Interactively at www.DataCamp.com



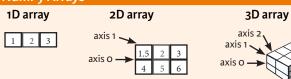
NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention: >>> import numpy as np



NumPy Arrays



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

>>> np.zeros((3,4))	Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)	
>>> d = np.arange(10,25,5)	Create an array of evenly
	spaced values (step value)
>>> np.linspace(0,2,9)	Create an array of evenly
	spaced values (number of samples)
>>> e = np.full((2,2),7)	Create a constant array
>>> f = np.eye(2)	Create a 2X2 identity matrix
>>> np.random.random((2,2))	Create an array with random values
>>> np.empty((3,2))	Create an empty array

1/0

Saving & Loading On Disk

```
>>> np.save('my array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my array.npy')
```

Saving & Loading Text Files

>>>	np.loadtxt("myfile.txt")
>>>	<pre>np.genfromtxt("my_file.csv", delimiter=',')</pre>
>>>	<pre>np.savetxt("myarray.txt", a, delimiter=" ")</pre>

Data Types

>>> np.int64	Signed 64-bit integer types
>>> np.float32	Standard double-precision floating point
>>> np.complex	Complex numbers represented by 128 floats
>>> np.bool	Boolean type storing TRUE and FALSE values
>>> np.object	Python object type
>>> np.string_	Fixed-length string type
>>> np.unicode_	Fixed-length unicode type

Inspecting Your Array

>>>	a.shape	Array dimensions
>>>	len(a)	Length of array
>>>	b.ndim	Number of array dimensions
>>>	e.size	Number of array elements
>>>	b.dtype	Data type of array elements
>>>	b.dtype.name	Name of data type
>>>	b.astype(int)	Convert an array to a different type

Asking For Help

>>> np.info(np.ndarray.dtype)

Array Mathematics

Arithmetic Operations

>>> g = a - b array([[-0.5, 0., 0.],	Subtraction
[-3., -3., -3.]]) >>> np.subtract(a,b)	Subtraction
>>> b + a array([[2.5, 4., 6.],	Addition
[5. , 7. , 9.]]) >>> np.add(b,a)	Addition
>>> a / b array([[0.66666667, 1. , 1.], [0.25 , 0.4 , 0.5]]	
>>> np.divide(a,b)	Division
>>> a * b array([[1.5, 4., 9.], [4., 10., 18.]])	Multiplication
>>> np.multiply(a,b)	Multiplication
>>> np.exp(b)	Exponentiation
>>> np.sqrt(b)	Square root
>>> np.sin(a)	Print sines of an array
>>> np.cos(b)	Element-wise cosine
>>> np.log(a)	Element-wise natural logarithr
>>> e.dot(f) array([[7., 7.],	Dot product
[7., 7.]])	

Comparison

>>> a == b array([[False, True, True],	Element-wise comparison
<pre>[False, False, False]], dtype=bool) >>> a < 2 array([True, False, False], dtype=bool)</pre>	Element-wise comparison
	Array-wise comparison

Aggregate Functions

>>> a.sum()	Array-wise sum
>>> a.min()	Array-wise minimum value
>>> b.max(axis=0)	Maximum value of an array row
>>> b.cumsum(axis=1)	Cumulative sum of the elements
>>> a.mean()	Mean
>>> b.median()	Median
>>> a.corrcoef()	Correlation coefficient
>>> np.std(b)	Standard deviation

Copying Arrays

>>> h = a.view()	Create a view of the array with the same data
>>> np.copy(a)	Create a copy of the array
>>> h = a.copy()	Create a deep copy of the array

Sorting Arrays

>>> a.sort()	Sort an array
>>> c.sort(axis=0)	Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Subsetting

>>> a[2]

>>> b[1,2]

>>> a[0:2]

>>> b[:1]

array([1, 2])

array([2., 5.])

>>> b[0:2,1]

>>> c[1,...]

>>> a[: :-1]

>>> a[a<2]

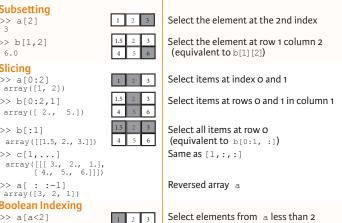
array([1])

Fancy Indexing

array([3, 2, 1]) **Boolean Indexing**

6.0 Slicina

```
Also see Lists
```



Select elements (1,0), (0,1), (1,2) and (0,0)

```
Select a subset of the matrix's rows
and columns
```

Array Manipulation

array([1, 2, 3, 10, 15, 20])

[7., 7., 0., 1.]])

[array([1]),array([2]),array([3])]

>>> np.hstack((e,f))
array([[7., 7., 1., 0.],

>>> np.column stack((a,d))

>>> np.vstack((a,b)) array([[1. , 2. , 3.], [1.5, 2. , 3.], [4. , 5. , 6.]])

>>> np.r [e,f]

array([[1, 10], 2, 15], [3, 20]])

Splitting Arrays

>>> np.hsplit(a,3)

>>> np.vsplit(c,2)

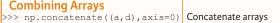
>>> np.c [a,d]

>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]

array([4. , 2. , 6. , 1.5]) >>> b[[1, 0, 1, 0]][:,[0,1,2,0]]

array dimensions array dimensions
ne array but don't change data

Adding/ Kemoving Elements	
>>> h.resize((2,6))	Return a new array with shape (2,6)
>>> np.append(h,g)	Append items to an array
>>> np.insert(a, 1, 5)	Insert items in an array
>>> np.delete(a,[1])	Delete items from an array



DataCamp Learn Python for Data Science Interactively

Stack arrays verti	cally (row-wise)
--------------------	------------------

Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)

Create stacked column-wise arrays

Create stacked	column-w	ise arrays
----------------	----------	------------

Split the array horizontally at the 3rd	
index	

Split the array vertically at the 2nd in	dex