DEEP NEURAL NETWORKS WITH KERAS

Testing different Deep Neural Network parameters for classification of rock data samples

ABSTRACT

In this tutorial, I discuss how to create Artificial Neural Networks for multi-class classification using the Keras Deep Learning Library. This is a Python library for Deep Learning. It wraps the numerical libraries Theano and Tensor Flow. I am going to show how to create multi-layer neural networks and how to play with the key parameters in order to improve the multi-classification results. I apply the networks to a test file including a rock sample data set consisting of various rock types that are characterized by different chemical composition.

TABLE OF CONTENT

Introduction

The expression of “Deep Learning” is commonly used when talking about multilayer Artificial Neural Networks (ANN). Current applications of Deep Neural Networks (DNN) concern image and speech recognition, text recognition in images for real-time translation, drug discovery and toxicity prediction and many other applications in health industry. Furthermore, there is a growing number of applications in hydrocarbon industry, including exploration, reservoir characterization, monitoring during oil production and oil field development (Dell’Aversana, 2018a). The first type of multi-layer ANN architecture is the multilayer perceptron (MLP). Three layers form the simplest type: one input layer, one hidden layer, and one output layer (Rosenblatt, 1957; von Neumann, 1958; Widrow, 1960). Each unit in the hidden layer is connected to all units in the input layers, and the output layer is fully connected to the hidden layer (Dense Multi-Layer ANN). In case there is more than one hidden layer, such a network is also called a Deep Artificial Neural Network. The learning workflow of MLP is schematically the following (Raschka and Mirjalili, 2017): 1) Forward propagation of the input patterns (training data vector) from the input layer through the network to generate an output. 2) Error calculation by comparing network output with desired output. 3) Error back-propagation, derivative calculation with respect to each weight in the entire network, and model update. We iterate the above steps many times (epochs) until we obtain a proper convergence (error reduction below the desired threshold).

The number of layers represents just one of the “hyper-parameters” of a DNN. The complexity of a network is given also by the number of neurons, connections and weighs. Every individual weight represents a parameter to be learnt. Of course, the complexity of the training depends on the number of those weights.

In this tutorial, I will show how to write a code in Python for creating a modular DNN. I use “Keras”, a Python library for Deep Learning that wraps the numerical libraries Theano and Tensor Flow. My goal is to show how we can test the crucial hyper-parameters of the neural network, trying to optimize the classification results in terms of accuracy. This is calculated using a labelled test-data set, through which we can check quantitatively the performance of our DNN classifier. In the test, I use an open source data set, the same that I have used in previous tutorial works (Dell’Aversana, 2018b).

The tutorial is divided in several sections explaining the Python code through many comments aimed at clarifying the main significance of the coding process. In such a way, everybody is able to reproduce the code itself and to test it with its own data. It is possible to use the same data used in this tutorial. One example of test data set can be found at the following link: https://www.researchgate.net/publication/333245171_rock_samples.

This is extracted from a public data set (Mamani et al., 2010; web link to the entire original data: http://georoc.mpchmainz.gwdg.de/georoc/webseite/Expert_Datasets.htm). Each section of the tutorial is written in a schematic way[1], and it is focused on the code[2]. Additional explanations can be found in Dell'Aversana (2018b). A basic knowledge of Python language and its main libraries is required. It is assumed that the user has properly installed all the libraries used in the tutorial.

[1] Do not take into account the sequential number of “In” and “Out” in the following. These have been edited several times, so they do not follow the original sequential order. I left these numbers just to separate more clearly the different steps of the workflow.

[2] Be careful to respect the correct indentation when copy and paste the code below. Otherwise, there will be an error message such as “syntax error” or “indentation error”.

Importing libraries

In [4]:

```
#Here, I import the library suite necessary for the following part of the
#tutorial.
#I will import additional libraries when necessary.
import numpy
import pandas
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
```

```
Using TensorFlow backend.
```

Input data file

The same test file used here can be download at the following link:
https://www.researchgate.net/publication/339146134_test_file

Alternatively, ask me directly by writing an email to dellavers@tiscali.it

In [27]:

```
#My input file is in Excel, thus I import the module for reading the
#excel file:
#This file can be downloaded from the following link:
#https://www.researchgate.net/publication/333245171_rock_samples
import xlrd
#We can access to the different sheets of the Excel file.
#We will take the sheet with our data set.
workbook = xlrd.open_workbook('zz.xlsx')
```

```
workbook
workbook.sheet_names()
#Remark: here, 'zz.xlsx' is our input file consisting of several rock
#samples characterized by different distribution of chemical features
#(oxides).
```

Out[27]:

```
['Table DR2']
```

In [28]:

```
#Now I convert my input file from xlsx to csv format, using a function of
#pandas library.
import pandas as pd
data_xls = pd.read_excel('zz.xlsx', 'Table DR2', index_col=None)
#data_xls.to_csv('zz.xlsx.csv', encoding='utf-8')
data_xls.to_csv('zz.csv')
#Remark: here, "zz.csv" is just a simple name given to our csv file.
```

Inizializing the process and reading the data

In [ ]:

```
#Next, I need to initialize the random number generator to a constant
#value (7).
#It ensures that the stochastic process of training a neural network
#model can be reproduced.
```

In [19]:

```
# fix random seed for reproducibility
seed = 7
numpy.random.seed(seed)
```

In [6]:

```
#After these preliminary operations, we can start the tutorial core.
#Let us start by reading the data.
```

In [29]:

```
import pandas as pd
#I define a function for reading the input file.
```

```
df_rock = pd.read_csv('zz.csv')
df_rock.head()
```

Out[29]:

Unnamed: 0	Sample	Rock_type	SiO2	TiO2	Al2O3	Fe2O3	MnO	MgO	CaO	Na2O	K2O	P2O5
0	SAR-00-08	andesite	60.4	1.133	16.5	5.85	0.074	2.25	5.20	5.06	2.89	0.557
1	SAR-00-07	andesite	60.5	0.828	16.5	4.97	0.075	1.97	4.35	4.76	2.99	0.343
2	SAR-00-13	andesite	60.6	1.015	16.4	5.59	0.082	2.24	4.82	5.00	3.00	0.473
3	COTA-05-15	andesite	58.9	1.222	16.9	6.56	0.082	2.88	5.64	4.76	2.41	0.482
4	COTA-05-06	andesite	59.5	1.169	16.7	6.09	0.077	2.54	5.24	4.58	2.70	0.490

In [8]:

```
#Let us plot the number of rows and columns of our file.
```

In [30]:

```
df_rock.shape
```

Out[30]:

(107, 13)

In [31]:

```
#We have 107 rows (one is for the header) and 10 chemical features
#(oxides) in 10 columns.
#If you explore the data set, you can see that it is characterized by
#5 rock types.
#These represent our classification target.
#Our objective is to classify these different rocks using their chemical
#compositions as features.
```

Data analysis and visualization

In [12]:

```
#Let us analyze the data through a suite of cross-plots.
#I use the seaborn library for that purpose.
```

In [34]:

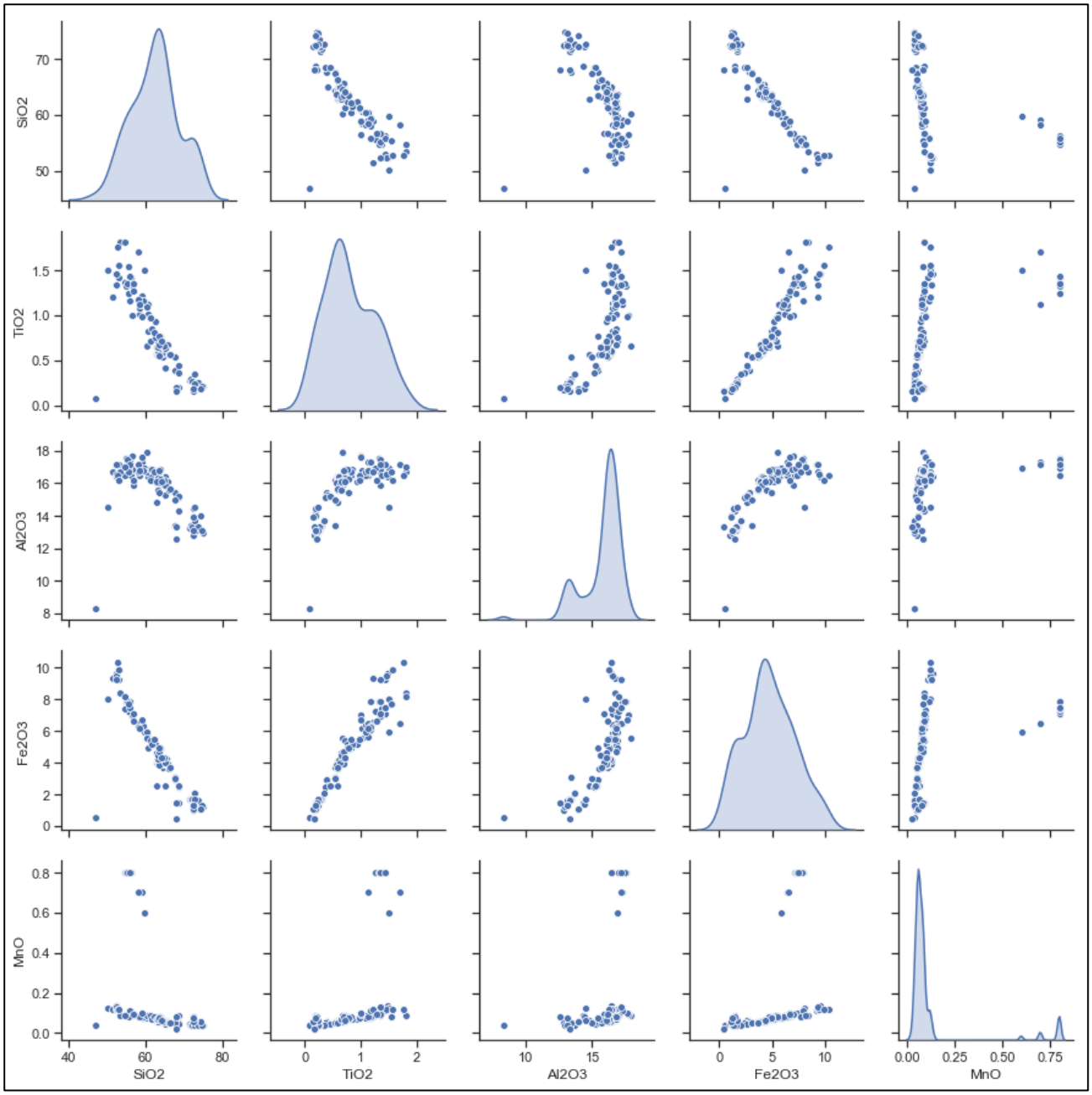
```
import seaborn as sns; sns.set(style="ticks", color_codes=True)
```

```
#g = sns.pairplot(df_rock[["SiO2","TiO2","Al2O3","Fe2O3","MnO","MgO",
#"CaO","Na2O","K2O","P2O5"]], diag_kind="kde")
g = sns.pairplot(df_rock, diag_kind="kde", height=2)
#Remark: in case of warning, run again this section.
```

In [35]:

```
#Next, I zoom in some selected images, creating a limited suite of
#cross-plots.
import seaborn as sns; sns.set(style="ticks", color_codes=True)
g = sns.pairplot(df_rock[["SiO2","TiO2","Al2O3","Fe2O3","MnO"]],
                 diag_kind="kde")
```

```
#g = sns.pairplot(df_rock, diag_kind="kde", height=4)
```



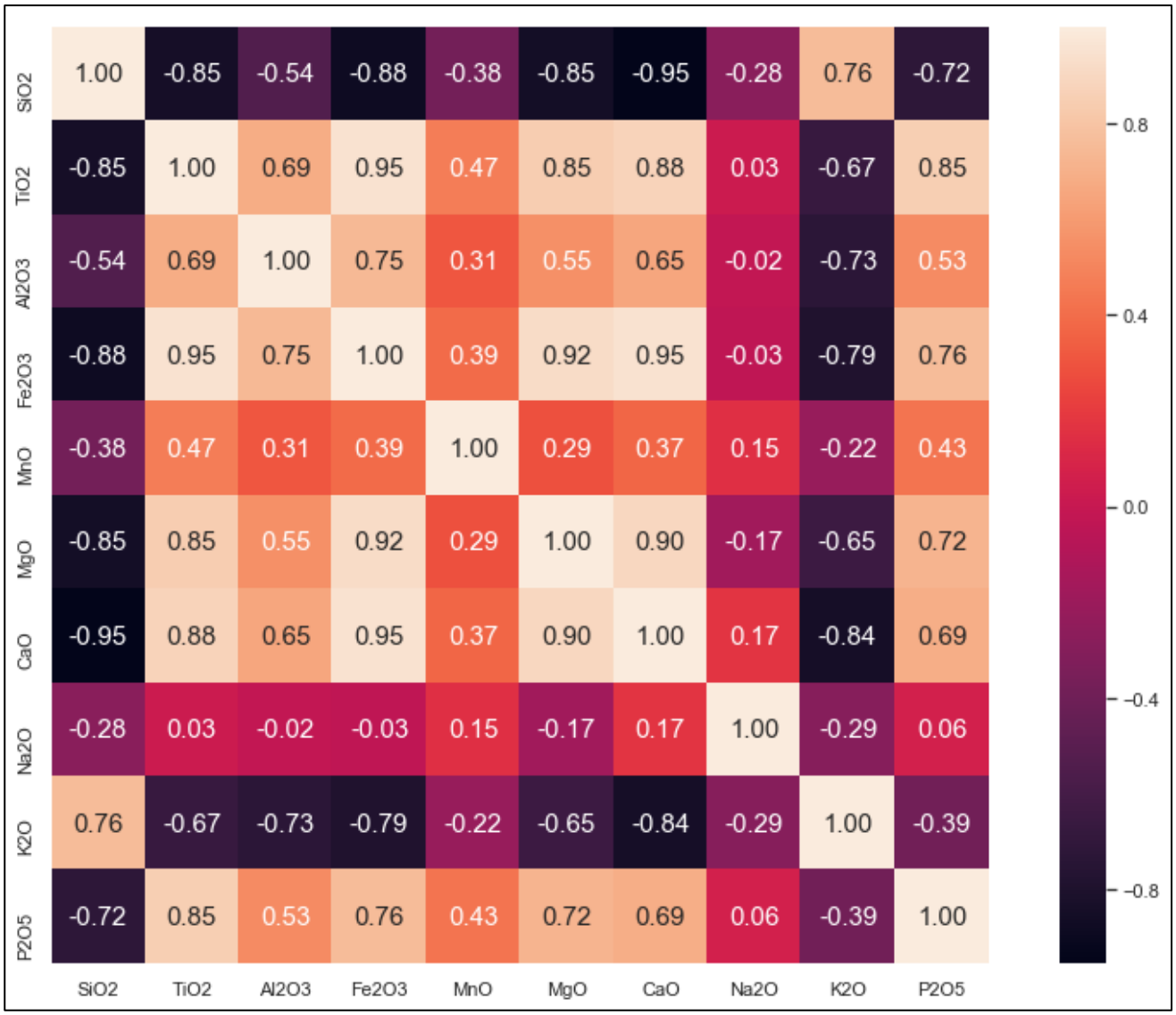
In [16]:

```
#For visualizing the important characteristics of our dataset,
#I use the pairplot function imported from the seaborn library.
#I can display a matrix of the cross-correlation values between the
#chemical oxides.
```

In [36]:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
cols = ['SiO2', 'TiO2','Al2O3', 'Fe2O3', 'MnO','MgO', 'CaO', 'Na2O',
        'K2O', 'P2O5']
cm = np.corrcoef(df_rock[cols].values.T)
sns.set(font_scale=1.0)
sns.set(rc={'figure.figsize':(11.7,8.27)})
hm = sns.heatmap(cm,
cbar=True,
annot=True,
square=True,
fmt='.2f',
annot_kws={'size': 15},
yticklabels=cols,
xticklabels=cols)
plt.tight_layout()
#plt.savefig('images/10_04.png', dpi=300)
plt.show()
```



In [37]:

```
#The numbers in the matrix above represent the cross-correlation
#coefficients.
```

Preparing the data for the Deep Neural Network (DNN)

In [38]:

```
#Next, I prepare the data for my neural network.
```

In [39]:

```
#I assign the 10 features to a NumPy array X.
#Furthermore, I split the data into a training and a testing sub-data
#sets.
```

```
#I diplay the labels of my data (rock types), just for checking.
```

In [40]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
X, y = df_rock.iloc[:, 3:].values, df_rock.iloc[:, 2].values
X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.3,
stratify=y,
random_state=0)
le = LabelEncoder()
```

In [42]:

```
print(y)
```

```
['andesite' 'andesite' 'andesite' 'andesite' 'andesite' 'andesite'
 'andesite' 'andesite' 'andesite' 'andesite' 'andesite' 'andesite'
 'andesite' 'andesite' 'andesite' 'andesite' 'andesite' 'andesite'
 'andesite' 'andesite' 'andesite' 'andesite' 'andesite' 'andesite'
 'andesite' 'andesite' 'andesite' 'basaltic andesite' 'basaltic andesite'
 'basaltic andesite' 'basaltic andesite' 'basaltic andesite'
 'basaltic andesite' 'basaltic andesite' 'basaltic andesite'
 'basaltic andesite' 'basaltic andesite' 'basaltic andesite'
 'basaltic andesite' 'basaltic andesite' 'basaltic andesite'
 'basaltic andesite' 'basaltic andesite' 'basaltic andesite'
 'basaltic andesite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite
'
 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite'
 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite' 'dacite'
 'dacite' 'dacite' 'dacite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite'
 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite'
 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite'
 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite' 'rhyolite'
 'rhyolite' 'trachyandesite' 'trachyandesite' 'trachyandesite'
 'trachyandesite' 'trachyandesite' 'trachyandesite' 'trachyandesite'
```

```
 'trachyandesite' 'trachyandesite' 'trachyandesite' 'trachyandesite'
 'trachyandesite' 'trachyandesite' 'trachyandesite']
```

In [43]:

```
#Now I encode the string classes into integers.
```

In [44]:

```
encoder = LabelEncoder()
encoder.fit(y_train)
encoded_y = encoder.transform(y_train)
#convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_y)
#Remark: in Informatics, dummy data is benign information that does not
#contain any useful data,but serves to reserve space where real data is
#nominally present.
#Dummy data can be used as a placeholder for both testing and operational
#purposes.
#Notice that I am encoding the classes referred to the training sub-set,
#that I want to use to train the network.
```

Defining the first, simple structure of the Neural Network

In [45]:

```
#Now, I define the "structure" of my neural network.
#I start with a very simple network formed by just one hidden layer.
#Then I will improve it by adding other hidden layers.
```

In [48]:

```
#Here, I define baseline model
#Below is a function that will create a baseline neural network for my
#classification problem.
#It creates a simple fully connected network with one hidden layer that
#contains 20 neurons.
#The input layer contains 10 neurons because my input file has 10
#features (10 different oxides).
#Instead, the output layer contains 5 neurons because we have 5 rock
#types (5 different classes).
```

```
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='relu'))
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [49]:

```
#Additional remarks about the baseline neural network model:
#1)
#You can notice that I use different activation functions:
#these are "relu" and "softmax".
#In the following, I will test various activation functions, including
#"Sigmoid",
#for the different hidden layers.
#2)
#I use cross-entropy as the loss function.
#A friendly introduction to cross-entropy can be found at the following
#link:
#https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/
```

In [50]:

```
#Here, I define my neural network estimator, assigning some key
#hyper-parameters.
#I fix the number of iterations in the training phase (epochs).
estimator = KerasClassifier(build_fn=baseline_model,
                            epochs=100, batch_size=5, verbose=0)
```

In [51]:

```
#Next, I check the performance of my estimator using a K-fold test.
#This is applied on my labelled training data (X_train).
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
```

```
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
WARNING:tensorflow:From C:\Users\ag12859\AppData\Local\Continuum\anaconda
3\lib\site-packages\tensorflow\python\framework\op_def_library.py:263: co
locate_with (from tensorflow.python.framework.ops) is deprecated and will
be removed in a future version.

Instructions for updating:

Colocations handled automatically by placer.

WARNING:tensorflow:From C:\Users\ag12859\AppData\Local\Continuum\anaconda
3\lib\site-packages\tensorflow\python\ops\math_ops.py:3066: to_int32 (fro
m tensorflow.python.ops.math_ops) is deprecated and will be removed in a
future version.

Instructions for updating:

Use tf.cast instead.

Accuracy: 46.25% (23.78%)
```

In [52]:

```
#The result is not very good. Accuracy is quite far from being 100%,
#standard deviation (std) is 23.78%.
#Let us improve the neural network by adding a second hidden layer.
#Thus, we build a deep neural network, by definition
#(that is a network with more than 1 hidden layer).
```

Improving the Neural Network by adding hidden layers

In [54]:

```
#Improving the model
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='relu'))
    model.add(Dense(20, input_dim=20, activation='relu'))
    #new hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
```

```
    return model
```

In [55]:

```
#Here, I define again my neural network estimator.
#This is redundant, but this step allows me to change, eventually,
#some parameter (ex: N. epochs).
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [56]:

```
#Next, I check again the performance of my estimator using a K-fold test.
#This is applied on my labelled training data (X_train).
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

Accuracy: 68.93% (25.97%)

In [57]:

```
#Good! Accuracy improved; std not too much.
#Let us try to add another inner layer:
```

In [59]:

```
#Improving the model with an additional hidden layer.
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='relu'))
    model.add(Dense(20, input_dim=20, activation='relu'))#hidden layer
    model.add(Dense(20, input_dim=20, activation='relu'))#hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [60]:

```
#I re-do the same steps as above ...
```

In [61]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [62]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 72.32% (14.53%)
```

In [63]:

```
#Good! It's improving.
#Now, let us try to increase the number of epochs.
#This will take a few minutes of additional computation times.
```

Improving the Neural Network by increasing the number of epochs and hidden layers

In [64]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=200,
                            batch_size=5, verbose=0)
```

In [65]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 79.64% (13.86%)
```

In [66]:

```
#OK.
```

```
#Let us try to add another hidden layer...
```

In [68]:

```
#Improving the model with an additional hidden layer
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='relu'))
    model.add(Dense(20, input_dim=20, activation='relu'))#hidden layer
    model.add(Dense(20, input_dim=20, activation='relu'))#hidden layer
    model.add(Dense(20, input_dim=20, activation='relu'))#hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [69]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [70]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 76.79% (18.02%)
```

In [71]:

```
#Not so good.
#I try to increase the number of neurons, but using 100 epochs.
```

Improving the Network by increasing the number of neurons and decreasing the number of layers

In [72]:

```
#Improving the model with additional neurons, but reducing the N. of
#hidden layers
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='relu'))
    model.add(Dense(50, input_dim=50, activation='relu'))#hidden layer
    model.add(Dense(50, input_dim=50, activation='relu'))#hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [73]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [74]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 81.07% (20.38%)
```

In [75]:

```
#Good! It seems to work. Accuracy increased, even though std increased.
#Let us increase the number of neurons further.
```

In [76]:

```
#Improving the model with additional neurons and the same N. of
#hidden layers
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='relu'))
```

```
    model.add(Dense(100, input_dim=100, activation='relu'))#hidden layer
    model.add(Dense(100, input_dim=100, activation='relu'))#hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [77]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [78]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

Accuracy: 83.39% (15.20%)

In [79]:

```
#Good.
```

Testing different activation functions

In [80]:

```
#Let us try to use a different activation function
```

In [81]:

```
#Using Sigmoid activation function.
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='sigmoid'))
    model.add(Dense(100, input_dim=100, activation='sigmoid'))
    #hidden layer
```

```
    model.add(Dense(100, input_dim=100, activation='sigmoid'))
    #hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [82]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [83]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 71.79% (15.67%)
```

In [84]:

```
#We can see that the sigmoid function allows an accuracy lower than the
#RELU function using the same ANN structure.
```

In [85]:

```
#What happens if we use sigmoid with a different ANN architecture?
```

Some final tests

In [ ]:

```
#Using Sigmoid activation function with less neurons.
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='sigmoid'))
    model.add(Dense(100, input_dim=25, activation='sigmoid'))
```

```
    #hidden layer
    model.add(Dense(100, input_dim=25, activation='sigmoid'))
    #hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [ ]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [87]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 79.46% (13.77%)
```

In [88]:

```
#Not so good.
```

In [89]:

```
#Let us try to add many additional hidden layers with 50 neurons
#with both RELU and Sigmoid functions.
```

In [90]:

```
#Using Sigmoid activation function with less neurons.
def baseline_model():
    # create model
    model = Sequential()
    model.add(Dense(20, input_dim=10, activation='sigmoid'))
    model.add(Dense(50, input_dim=50, activation='relu'))
    #hidden layer
    model.add(Dense(500, input_dim=50, activation='sigmoid'))
```

```
    #hidden layer
    model.add(Dense(50, input_dim=50, activation='relu'))#hidden layer
    model.add(Dense(50, input_dim=50, activation='sigmoid'))#hidden layer
    model.add(Dense(50, input_dim=50, activation='relu'))#hidden layer
    model.add(Dense(500, input_dim=50, activation='sigmoid'))#hidd. layer
    model.add(Dense(50, input_dim=50, activation='relu'))#hidden layer
    model.add(Dense(50, input_dim=50, activation='sigmoid'))#hidden layer
    model.add(Dense(5, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model
```

In [ ]:

```
estimator = KerasClassifier(build_fn=baseline_model, epochs=100,
                            batch_size=5, verbose=0)
```

In [ ]:

```
kfold = KFold(n_splits=10, shuffle=True, random_state=seed)
#I print the mean accuracy and its standard deviation
results = cross_val_score(estimator, X_train, dummy_y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100,
                                     results.std()*100))
```

```
Accuracy: 63.75% (15.07%)
```

In [ ]:

```
#Performance has deteriorated. Complicating the network does not mean
#improving it!
#That is interesting: considering that our classification problem is
#relatively simple,we do not need to complicate our network too much.
#If we exaggerate with the number of hidden layers and using different
#activation functions, we risk to deteriorate the results.
```

Final remarks

In this tutorial I showed how to create simple Artificial Neural Networks (Dense, full connected Deep ANN's). I showed how to play with them, by testing different hyper-parameters and checking the results, in terms of accuracy and standard deviations. We have seen that key variables are the number of hidden layers, the number of neurons, the activation function, the number of iterations and so forth. However, an important message is that complicating the network does not imply that it improves. Every classification (or regression) problem requires its own network architecture. If you are interested in this subject, you can play with this code using your own input data. In summary, you should have learnt how to manipulate the key parameters of a dense deep neural network for a simple classification problem. If you desire to explore the same type of classification problem from a wider perspective, I suggest to read my open-source paper at the following link: https://www.researchgate.net/publication/328020065_Machine_Learning_for_rock_classification_based_on_mineralogical_and_chemical_composition._A_tutorial.

References

Dell'Aversana, P., 2018a. A Global Approach to Data Value Maximization. Integration, Machine Learning and Multimodal Analysis. Cambridge Scholars Publishing.

Dell'Aversana, 2018b. Machine Learning for rock classification based on mineralogical and chemical composition. A tutorial. Published on Research Gate in October 2018. DOI: 10.13140/RG.2.2.32886.04168.

Mamani, M., Wörner, G., and Sempere, T., 2010. Geochemical variations in igneous rocks of the Central Andean orocline (13°S to 18°S): Tracing crustal thickening and magma generation through time and space. – GSA Bulletin; January/February 2010; v. 122; no. 1/2; p. 162–182; doi: 10.1130/B26538.1.

Raschka, S. and Mirjalili, V., 2017. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow, 2nd Edition.

Rosenblatt, F., 1957. The Perceptron, a Perceiving and Recognizing Automaton. Cornell Aeronautical Laboratory.

Simard, P. Y., Steinkraus, D. and. Platt, J. C, 2003. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. IEEE, 2003, p.958.

von Neumann, J., 1958. The Computer and the Brain. New Haven/London: Yale University Press. Widrow, B., 1960. Adaptive "Adaline" neuron using chemical "memistors". Number Technical Report 1553-2. Stanford Electron. Labs., Stanford, CA, October 1960.

WEB references

https://machinelearningmastery.com/multi-class-classification-tutorial-keras-deep-learning-library/
https://www.cambridgescholars.com/a-global-approach-to-data-value-maximization
https://www.researchgate.net/publication/328020065_Machine_Learning_for_rock_classification_based_on_mineralogical_and_chemical_composition_A_tutoria
https://www.researchgate.net/publication/330384396_Comparison_of_different_Machine_Learning_algorithms_for_lithofacies_classification_from_well_logs

Data

I have used the public data set available on the GEOROC (Geochemistry of Rocks of the Oceans and Continents) web site (http://georoc.mpchmainz. gwdg.de/georoc/). For a wider application to classification of these data, see my paper at the link below: https://www.researchgate.net/publication/333245171_rock_samples

Credits

Please, use the correct citation of this tutorial (Name of the author (Paolo Dell'Aversana), title of the tutorial, publication date, DOI), if you intend to use it and to publish your results.
In [ ]:

Remark: original codes modified by Paolo Dell'Aversana and re-adapted for this tutorial addressed to analysis of rock samples.

```
#Copyright (c) 2017 SEBASTIAN RASCHKA (mail@sebastianraschka.com)
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
```

```
#DISCLAIMER: this tutorial is written for scientific and educational
#purposes. The above code could not work properly with future versions of
#the #Python libraries here used.
```

```
#I don't take any responsibility for any improper use of this tutorial.
#Paolo Dell'Aversana - February 2020.
```