dnn_iris

July 26, 2020

0.1 preparing data and module

[1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
import numpy as np
```

0.2 Checking data and preparing functions to be used

[2]:
```
def relu(x):
    return np.maximum(x, 0)
```

[3]:
```
def softmax(x):
  if x.ndim == 2:
    x = x.T
    x = x - np.max(x, axis=0)
    y = np.exp(x) / np.sum(np.exp(x), axis=0)
    return y.T

  x = x - np.max(x)
  return np.exp(x) / np.sum(np.exp(x))
```

[4]:
```
def cross_entropy_error(y, t):
  if y.shape != t.shape:
    raise ValueError
  if y.ndim == 1:
    return - (t * np.log(y)).sum()
  elif y.ndim == 2:
    return - (t * np.log(y)).sum() / y.shape[0]
  else:
    raise ValueError
```

[5]:
```
def forward(x, W1, W2, b1, b2):
    return softmax(np.dot(relu(np.dot(x, W1) + b1), W2) + b2)
```

[6]:
```
def validate_with_test_data(X, y, W1, W2, b1, b2, title=''):
  print(f'{title}{(forward(X, W1, W2, b1, b2).argmax(axis=1) == y.
 ↪argmax(axis=1)).sum()}/{150 - TRAIN_DATA_SIZE}')
```

0.3 Checking data

[7]:
```
iris = load_iris()
```

[8]:
```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
```

```
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more …
```

[9]:
```
print(f'names={iris.target_names}')
```

```
names=['setosa' 'versicolor' 'virginica']
```

[10]:
```
print(f'shape={iris.data.shape}')
```

```
shape=(150, 4)
```

[11]:
```
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df.describe()
```

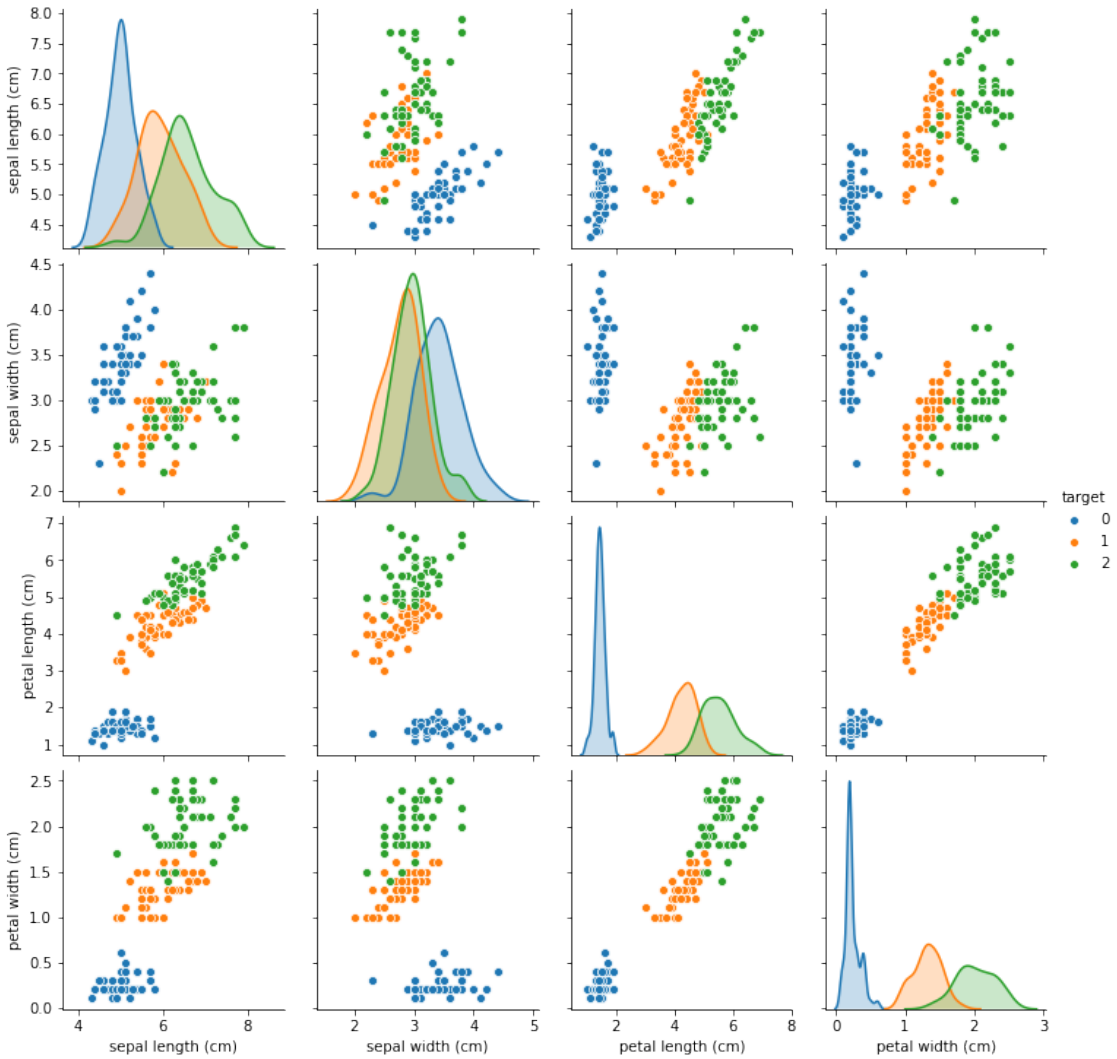
[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	\
count	150.000000	150.000000	150.000000	
mean	5.843333	3.057333	3.758000	
std	0.828066	0.435866	1.765298	
min	4.300000	2.000000	1.000000	
25%	5.100000	2.800000	1.600000	
50%	5.800000	3.000000	4.350000	
75%	6.400000	3.300000	5.100000	
max	7.900000	4.400000	6.900000	

	petal width (cm)	target
count	150.000000	150.000000
mean	1.199333	1.000000
std	0.762238	0.819232
min	0.100000	0.000000
25%	0.300000	0.000000
50%	1.300000	1.000000
75%	1.800000	2.000000
max	2.500000	2.000000

[12]:
```
sns.pairplot(df, hue="target")
```

[12]: <seaborn.axisgrid.PairGrid at 0x7fdff3fd1f90>



0.4 Split the data.

[13]:
```
X = pd.DataFrame(iris.data, columns=iris.feature_names)
y = iris.target.reshape(len(iris.target), 1)
y_one_hot = OneHotEncoder(sparse=False).fit_transform(y)
X_train, X_test, y_train, y_test = train_test_split(X, y_one_hot, test_size=0.
 ↪2, random_state=1, stratify=y_one_hot)
```

[14]:
```
X_test.head(5)
```

[14]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
7	5.0	3.4	1.5	0.2
109	7.2	3.6	6.1	2.5
98	5.1	2.5	3.0	1.1
111	6.4	2.7	5.3	1.9
143	6.8	3.2	5.9	2.3

[15]:
```
y_test[:5]
```

[15]:
```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

0.5 Building a neural network

[16]:
```
INPUT_SIZE = 4
HIDDEN_SIZE = 8
OUTPUT_SIZE = 3
LEARNING_RATE = 0.1
TRAIN_DATA_SIZE = 120
EPOCH = 1001
```

[17]:
```
W1 = np.random.randn(INPUT_SIZE, HIDDEN_SIZE) / np.sqrt(INPUT_SIZE) * np.sqrt(2)
W2 = np.random.randn(HIDDEN_SIZE, OUTPUT_SIZE)/ np.sqrt(HIDDEN_SIZE) * np.
 ↪sqrt(2)
b1 = np.zeros(HIDDEN_SIZE)
b2 = np.zeros(OUTPUT_SIZE)
```

[18]:
```
validate_with_test_data(X_test, y_test, W1, W2, b1, b2, "Previous Learning : ")
```

```
Previous Learning : 10/30
```

[19]:
```
for i in range(EPOCH):
    # Forward Propagation with Data Storage
    y1 = np.dot(X_train, W1) + b1
```

```
    y2 = relu(y1)
    train_y = softmax(np.dot(y2, W2) + b2)

    # Loss function calculation
    L = cross_entropy_error(train_y, y_train)

    # The loss is shown every 100 times.
    if i % 100 == 0:
      validate_with_test_data(X_test, y_test, W1, W2, b1, b2, f"{i}th time : ")
      print(f'Loss : {L}')

    # Slope calculation
    a1 = (train_y - y_train) / TRAIN_DATA_SIZE
    b2_gradient = a1.sum(axis=0)
    W2_gradient = np.dot(y2.T, a1)
    a2 = np.dot(a1, W2.T)
    a2[y1 <= 0.0] = 0
    b1_gradient = a2.sum(axis=0)
    W1_gradient = np.dot(X_train.T, a2)

    # Update parameters
    W1 = W1 - LEARNING_RATE * W1_gradient
    W2 = W2 - LEARNING_RATE * W2_gradient
    b1 = b1 - LEARNING_RATE * b1_gradient
    b2 = b2 - LEARNING_RATE * b2_gradient
```

```
0th time : 10/30
Loss : 3.670254393487529
100th time : 25/30
Loss : 0.32884822634051114
200th time : 30/30
Loss : 0.23671587038150818
300th time : 30/30
Loss : 0.16185501097222296
400th time : 30/30
Loss : 0.13512303370646908
500th time : 30/30
Loss : 0.11637827218126823
600th time : 30/30
Loss : 0.10676239227122082
700th time : 30/30
Loss : 0.09959917232482286
800th time : 30/30
Loss : 0.09550748535840696
900th time : 30/30
Loss : 0.0913751367244493
1000th time : 30/30
```

```
Loss : 0.0884907784731897
```

[20]:
```
L = cross_entropy_error(forward(X_train, W1, W2, b1, b2), y_train)
print(f'Loss : {L}')

validate_with_test_data(X_test, y_test, W1, W2, b1, b2, f"Last : ")
```

```
Loss : 0.09205039482727378
Last : 30/30
```