11753 Computational Intelligence
Master in Intelligent Systems
Universitat de les Illes Balears

Handout #1: **Feed-forward Neural Networks (FFNN)**

T1. Follow the next indications to build an FFNN to solve the multi-class classification problem related to the *ic_lab1* dataset available at the course web page:

(a) Load the *ic_lab1* dataset using the following source code:

```python
import numpy as np
data = np.loadtxt('ic_lab1.txt')
X = data[:,:-1]
y = data[:,-1]
```

Listing 1: Loading of the *ic_lab1* dataset.

(b) Determine the number of classes. For instance, you can use the following source code:

```python
import numpy as np
M = len(np.unique(y))
```

Listing 2: Use of numpy.unique to find the number of classes.

(c) Normalize the dataset and split it into the train and test sets using the function *train_test_split* of *scikit-learn* as follows:

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_ = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_, y, test_size=.3, random_state=100)
```

Listing 3: Train and test splitting function.

for a 70% of the dataset devoted to training and 30% for testing (random_state = <int value> ensures a reproducible splitting [useful for debugging purposes]).

(d) Make use of the indications in slides #4-6 of the lecture notes on *Keras (& Tensorflow)* to **define a neural network with one single hidden layer** with *nh* neurons. Let us assume that you store your network into the *model* object.

Remember that for a multi-class problem you need **as many output neurons as classes** M. (You can check diverse combinations of activation functions, as it is done in the source file *keras_examples.py* available in the course web page.)

(e) **Define a training strategy**, i.e. an optimizer (e.g. *RMSprop*), a loss function (e.g. *categorical cross entropy*) and a metric (e.g. *accuracy*), in accordance to the indications of the slides #7-9, and **compile your model**. (You can check other, more complex strategies suggested in the source file *keras_examples.py* available in the course web page.)

(f) Using slide #10 as a guide, **fit your model** using the training set, for a sufficient number of epochs, batch size and validation set. Remember that, previous to fitting a multi-class model, you need to transform your ground truth using *one-hot encoding*:

```python
from keras.utils import to_categorical
y_train_ = to_categorical(y_train)
```

Listing 4: One-hot enconding Keras function.

(g) Run your model for $nh = 3$ and $nh = 4$ and compare the resulting performances by means of suitable metrics and plots: loss and metric values, classification map and contour map. To this end, you can use slides #11-12 and the following source code (also available in *nn_evaluation.py* at the course web page):

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score
from keras.utils import to_categorical
```

```

def to_label(M, y):

  if M == 2:
      # round predictions
      y_ = [round(x[0]) for x in y]
  else:
      # take max output
      y_ = np.argmax(y, axis=1)

  return y_

def plot_class(c, X, y):

  i = np.where(y == c)[0]
  plt.scatter(X[i,0],X[i,1])

def show_class_map_(model, X, y):

  if X.shape[1] != 2:
      print('This dataset is not two-dimensional ...')
      return

  M = len(np.unique(y))

  # plot the contour map
  plt.figure()
  for c in range(M):
    plot_class(c, X, y)
  plt.axis('equal')

  # plot the decision boundaries
  ax = plt.gca()
  xlim = ax.get_xlim()
  ylim = ax.get_ylim()

  # create the grid to evaluate the model at discrete feature space points
  xx = np.linspace(xlim[0], xlim[1], 100)
  yy = np.linspace(ylim[0], ylim[1], 100)
  YY, XX = np.meshgrid(yy, xx)
  xy = np.vstack([XX.ravel(), YY.ravel()]).T
  ZZ_ = model.predict(xy)
  ZZ  = to_label(M, ZZ_)

  # plot the boundaries
  for c in range(M):
      ax.contour(XX, YY, ZZ_[:,c].reshape(XX.shape), levels=[0.5], alpha=0.5, linestyles=['--'])

  plt.xlabel('x1')
  plt.ylabel('x2')
  plt.axis('equal')
  plt.title('contour map')
  plt.show()

  # plot the classification map
  plt.figure()
  plt.imshow(ZZ.reshape(XX.shape).T, origin='lower', extent=(xlim[0], xlim[1], ylim[0], ylim[1]), cmap='RdYlGn')
  plt.colorbar()
  for c in range(M):
  plot_class(c, X, y)
  plt.xlabel('x1')
  plt.ylabel('x2')
  plt.axis('equal')
  plt.title('classification map')
  plt.show()

def do_show(model, history, X_train, y_train, X_test, y_test):

  loss = history.history['loss']
  val_loss = history.history['val_loss']
  accuracy = history.history['accuracy']
```

```
76   val_accuracy = history.history['val_accuracy']
77 
78   plt.figure()
79   plt.plot(loss, label='train')
80   plt.plot(val_loss, label='validation')
81   plt.title('loss function')
82   plt.legend()
83   plt.show(block=False)
84 
85   plt.figure()
86   plt.plot(accuracy, label='train')
87   plt.plot(val_accuracy, label='validation')
88   plt.title('accuracy')
89   plt.legend()
90   plt.show(block=False)
91 
92   M = len(np.unique(y_train))
93 
94   # evaluation: train set
95   y_pred_ = model.predict(X_train)
96   y_pred = to_label(M, y_pred_)
97   cm = confusion_matrix(y_train, y_pred)
98   print(cm)
99   print('accuracy = %f' % (accuracy_score(y_train, y_pred)))
100 
101   y_train_ = to_categorical(y_train)
102   score = model.evaluate(X_train, y_train_)
103   print("Train loss:", score[0])
104   print("Train accuracy:", score[1])
105 
106   show_class_map_(model, X_train, y_train)
107 
108   # evaluation: test set
109   y_pred_ = model.predict(X_test)
110   y_pred = to_label(M, y_pred_)
111   cm = confusion_matrix(y_test, y_pred)
112   print(cm)
113   print('accuracy = %f' % (accuracy_score(y_test, y_pred)))
114 
115   y_test_ = to_categorical(y_test)
116   score = model.evaluate(X_test, y_test_)
117   print("Test loss:", score[0])
118   print("Test accuracy:", score[1])
```

Listing 5: Python functions to examine the performance of a neural network model. (Also available in *nn_evaluation.py*)

Although you should try to understand every line of the previous source code, pay special attention to:

- function *to_label*, at lines 6-15, which transforms the network output (= one neuron per class) into a numerical class label;
- line 51, which, for plotting the boundary for class c (against the other classes), we have to choose the output c of the prediction of the network for every grid point of the map *ZZ_*, and we select a level of 0.5 for drawing the boundary (= *levels* parameter);
- lines 101 and 115, which perform the one-hot encoding of the data prior to evaluating the network through the method *evaluate* of Keras models; and
- lines 95 and 109, which transform the network output into a numerical class label after a network prediction.

T2. (a) Load the *iris* dataset and implement a solution for the inherent 3-class, 4-feature classification problem using the following **four neural network configurations**:

  i. one single hidden layer with 4, 7 and 10 neurons
  ii. two hidden layers with, respectively, 4 and 3 neurons

(b) Use the following source code for evaluating the performance of each model using n-fold cross validation:

```
from sklearn.model_selection import KFold
def nfoldcv_keras(model, X, y):

  M = len(np.unique(y))
  n_folds = 5
```

```python

results = np.zeros(n_folds)
kf = KFold(n_splits=n_folds, shuffle=True)
for k, (train, test) in enumerate(kf.split(X)):
  # training
  X_train, y_train = X[train], y[train]
  y_train_ = to_categorical(y_train)
  model.fit(X_train, y_train_, epochs=100, batch_size=10,
            validation_split=0.2, verbose=0)
  # testing
  X_test, y_test = X[test], y[test]
  y_pred_ = model.predict(X_test)
  y_pred = to_label(M, y_pred_)
  results[k] = accuracy_score(y_test, y_pred)

return results
```

Listing 6: n-fold cross validation of Keras models. (Also available in *nn_evaluation.py*)

You can next e.g. show a box-plot: