ISTA 331 SPECIAL TOPICS ASSIGNMENT: INTRO TO NEURAL NETWORKS

1. Introduction

Neural networks, or more properly artificial neural networks, are one of the most interesting machine learning methods to be studied in recent decades. The concept behind them originated in the 1950s, but wasn't developed very far. More recently, a few new concepts, an advance in algorithms, and a dramatic increase in computational power have combined to change that.

Neural networks of various types have been found to be extremely powerful, if sometimes mysterious, tools for a wide variety of tasks. In this assignment, you'll define and train a simple neural network for a classification task: classifying seeds by species according to various measurements.

1.1. Getting started: install Keras.

For this assignment we will use Keras, an easy-to-use Python neural network API that uses one of several other modules as a backend. You probably don't have this installed. See `https://keras.io/#installation` for detailed installation instructions.

If you are using `conda`, use

```
conda install tensorflow
conda install -c anaconda keras
```

to install TensorFlow (the recommended backend) and Keras.

If you are not using `conda`, use

```
pip install tensorflow
pip install keras
```

1.2. The multilayer perceptron.

The type of network we will train in this assignment is one of the most basic form, called a *multilayer perceptron*. This is a form of network that consists of several fully-connected (a.k.a. 'dense') layers, with at least one hidden layer, followed by an output layer. The activation of the output layer is a softmax function, which we first encountered with multiclass logistic regression. The softmax function is a way to translate "scores" produced by the final hidden layer into estimated class probabilities.

1.3. Keras and specifying network architectures.

While not conceptually too complicated, the details of implementing training by stochastic gradient descent are messy enough that we will use off-the-shelf packages instead of trying to write it from scratch.

There are several popular frameworks for neural networks in Python, including PyTorch, TensorFlow, and Keras – the last of these is the one we'll use.

There are three steps to building a network in Keras:

(1) Initializing the network and specifying layers

(2) "Compiling" the network, specifying an optimizer, a loss function, and a metric for measuring accuracy
(3) Training the network using the `fit` method.

In practice, these steps look like:

(1) Initialize the network by calling `Sequential()` and assigning the new model to a variable. (Below I'm calling this variable `model`.) One by one, add layers with `add`, e.g.:

```
model.add(Dense(size, activation = func_name, input_shape = (n_predictors,)))
model.add(Dense(size, activation = func_name)
...
model.add(Dense(n_classes, activation = 'softmax'))
```

You can have as many layers as you want, with however many neurons as you want. Notice the two restrictions: the input size for the first layer should be the number of predictors you have, and the final layer should have `'softmax'` activation with a number of neurons equal to the number of classes.
(2) Compiling the network prepares it for training. We're going to use SGD optimization and a loss function called cross-entropy. You can use the following code directly:

```
model.compile(optimizer='sgd',
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

If you attempt to train the model before doing this, you'll get a runtime error.
(3) Call `model.fit(train_X, train_y, batch_size = N, epochs = T)` to train for `T` epochs with a batch size of `N`. By default this will print a running summary of the training process. If you want to suppress this, pass `verbose = False`.

1.4. **Model evaluation.** As usual, we evaluate the model by testing it on some examples we have held out for testing. Keras networks come with an `evaluate` method which takes a testing `X`, a testing `y`, and a batch size, and returns the value of the loss function on the testing set and the prediction accuracy on the testing set.

2. Instructions

Create a Python script called `network.py`.

Do the following imports:

```
import numpy as np
import pandas as pd
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
```

Implement the functions specified below. Your `main` function must run when I type `python network.py` at the terminal.

2.1. **Documentation.** Your script must contain a header docstring containing your name, ISTA 331, the date, and a brief description of the module. Each function must contain a docstring. Each function docstring should include a description of the function's purpose, the name, type, and purpose of each parameter, and the type and meaning of the function's return value.

2.2. **Grading.** Your submission will be human-graded. I'll run the python script with `python network.py` and look at the output produced, and then read the code. Make sure that the script runs! I will deduct points if I have to chase down syntax errors to get output from your code, or add a call to `main`, etc. Code should be clear and concise, but you will only lose style points if your code is a real mess. Include inline comments to explain tricky lines and summarize sections of code.

2.3. **Collaboration.** Collaboration is **not** permitted on this assignment. However, you may use your own notes, documentation for `sklearn`, `numpy`, and `pandas`, and any resources available on the D2L site (including videos, worksheets, Jupyter notebooks, etc.).

3. Function specifications

- `get_data`: This function takes no arguments and loads the data frame from `seeds.csv`. The data frame should look like the following.

```
In [7]: seeds.head()
```

Out[7]:

	area	perimeter	compactness	length	width	asymmetry	groove_length	class
0	15.26	14.84	0.8710	5.763	3.312	2.221	5.220	1
1	14.88	14.57	0.8811	5.554	3.333	1.018	4.956	1
2	14.29	14.09	0.9050	5.291	3.337	2.699	4.825	1
3	13.84	13.94	0.8955	5.324	3.379	2.259	4.805	1
4	16.14	14.99	0.9034	5.658	3.562	1.355	5.175	1

  Then, use `np.random.choice` to select 105 rows to be the training set. Once you have split the data frame into `train` and `test` frames, Keras requires that you do a little bit more work to get the data frames prepped.
  - Our target variable is `class`, so drop this from the data frame to get `X`. Then, assign `X = X.values` for each `X` to save only the values in `x` as a numpy array – Keras doesn't want a pandas DataFrame.
  - Extract the variable `class` for `y`, and subtract 1 from it so that it has values `0, 1, 2` instead of `1, 2, 3`. Again, Keras is picky: it needs the `y` to be a one-hot encoded binary array. To get this, you can use something like:

```
train_y = keras.utils.to_categorical(train_y)
test_y = keras.utils.to_categorical(test_y)
```

    where `train_y` and `test_y` are your `y` Series.

  Return the training `X`, training `y`, testing `X`, and testing `y`.
- `setup_network`: This function takes a list of integers and a string representing an activation function (one of `'sigmoid'`, `'tanh'`, or `'relu'`; see the Jupyter notebook/video for details) and initializes a neural network model. The integers represent the sizes of `Dense` network layers. Build the network by adding `Dense` layers of the specified sizes and activation function; remember to specify the input size of the first layer. Finish by adding by a `Dense` layer of size 3, with 'activation = 'softmax''. Compile the network with the following code:

```
model.compile(optimizer='sgd',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

  After you have initialized the network, added layers, and compiled the model, return the model.

- `train_network`: This function takes a network that has been created by `setup_network`, a training `X`, a training `y`, and a number of epochs. Train the network using the `X` and `y` data, a batch size of `16`, and the given number of epochs. You can return `None`.
- `test_network`: This function takes a trained network, a testing `X`, and a testing `y`. Call the `evaluate` method from the network and pass it the testing `X` and testing `y`, and a batch size of 1. Return the training accuracy.
- `main`: Get the training and testing `X` and `y` by calling `get_data`. Then set up, train, and test a network. You may experiment with the network architecture, as well as the number of training epochs – I leave these parameters up to you.

  Print the following output: the list of integers you used for the network architecture, the activation function you chose, the number of training epochs, and the testing accuracy. The output should look something like this:

  ```
  Network architecture: [list of integers]
  Activation function: [function name]
  Number of epochs: [epochs]
  Test accuracy: [xxxxxx]
  ```

  where everything in brackets is replaced with the actual values you used/found. This is human-graded so don't worry about getting the formatting exactly like this.

  Your goal is to produce a network that will reliably get around 90% accuracy or better in the testing process, while using as few training epochs as you can manage. A small portion of your grade will depend on this accuracy and the number of epochs you use to get there. (I will test your code several times – it's tough to get reproducible results out of this even by setting the random seed. So don't worry if your process occasionally produces lower accuracy.)