University of Maine
School of Computing and Information Science

COS 470/570: Introduction to Artificial Intelligence

# Programming Assignment: : Artificial Neural Networks

**Spring 2019**

**Assigned: 3/14/2019** **Due: 3/26/2019**

In this homework assignment, you will play around with a neural network tool called Keras. Keras is an interface to one of two different neural network backends, TensorFlow or Theano; it uses whichever one is installed on the user's computer. All of these allow you to work with them in Python.

## Part I: Python

The first thing you need to do is install Python on your computer. You can find a wealth of information about Python on the Web, especially at Python.org. There, you can download a version of Python for you particular operating system. I suggest downloading Python 3 (I think 3.7.2 is the current version at this time.)

You may also be able to install Python using package manager, such as `apt-get`, etc. On the Macintosh, I use the `homebrew` package manager, which works extremely well (see brew.sh) for lots of things, including not only Python, but also Emacs and SBCL ("Now he tells me!").

If you don't know Python, you'll need to learn at least the basics, although you won't need a lot to use Keras at first. A good place to start is "The Python Tutorial" (docs.python.org/3/tutorial/index.html).

I'm not going to assign any particular problems for you to do; however, you might consider re-doing the Lisp practice problems in Python, where possible. (You could write the Search assignment in Python, too, if you're looking for something to do over break! ⌣̈)

## Part II: Keras

Once you have Python installed, you need to install Keras and one of TensorFlow or Theano on your computer.

If you want a pretty complete way of installing deep learning packages as well as an easy way to maintain different Python "environments" for different purposes, check out Anaconda. (Please, no Nicki Minaj or Sir Mix-A-Lot jokes.) It's available at anaconda.org, and comes with a very nice package manager itself (conda). It also allows you to set up multiple Python "environments" that can have different things available.

Once you have Anaconda set up, you can use conda to install Keras and either TensorFlow of Theano.

If you don't want to use Anaconda, or if you do and can't get conda to play nice with you, then you can install the programs another way. You can find Keras, along with installation instructions, at keras.io. From the installation link, you can find links to how to install either TensorFlow or Theano. I don't have a strong preference for one or the other.

You may also find that your Python package manager (PIP) or your operating system's package manager already has the ability to install Keras; if so, then that's the way to go. This *ought* to be the case for PIP. To see if it's defined for you, type:

```
pip search keras
```

That is how you would do it on Linux or a Mac, anyway (using the Terminal app on a Mac, or, on a Mac or Linux, from a shell inside of Emacs via `m-x shell` or `m-x term` or `m-x eshell`). On a Windows machine, well. . . I don't know; you're the expert!

## Part III: Simple NN

### Tutorial

There's a very easy tutorial to get your first neural network up and running at:

machinelearningmastery.com/tutorial-first-neural-network-python-keras

Do the tutorial to create a network for the Pima dataset and run it.

### Create your own

I'll put a version of the well-known "Iris" dataset on the course website. This dataset, dating from 1936, relates four parameters of irises (petal length and width, sepal length and width) to their species (*Iris setosa*, *I. versicolor*, or *I. virginica*). I've manipulated this so that the dataset is shuffled and in a form amenable to a "softmax" output. A softmax output layer is one that contains $n$ neurons, one for each possible category of output, and whose activations sum to 1: thus, each neuron's activation can be viewed as the probability that the input is a member of the corresponding category. The format of the iris data is a CSV file:

```
sepal-length, sepal-width, petal-length, petal-width, I.setosa, I.versicolor, I.virginica
```

So this data item:

```
6.6, 3.0, 4.4, 1.4, 0, 1, 0
```

would represent an *Iris versicolor* with sepal length and width of 6.6 cm and 3.0 cm, and with petal length and width of 4.4 cm and 1.4 cm.

Your job is to create **two** neural networks to classify irises based on values of the four input parameters. You can use as many layers and neurons per layer you like for each; a simple single-hidden-layer network is sufficient, although you can experiment with other kinds to see how your accuracy changes. Although you *could* use any type of layer, I would suggest sticking to the "dense" layers used in the tutorial. The output layer, however, *must* be a softmax layer with 3 neurons, one for each type of iris. Vary the number of layers and/or the number of nodes per layer between the two neural networks you construct.

I'll split the dataset into two files, `iris_train.csv` and `iris_test.csv` to use for training the net and testing it, respectively.

## Turn in...

Turn in a file containing:

- A *brief* (1–2 page) description of the networks you created and a discussion of what their differences meant for the accuracy of their answers. Relate any insights or hypotheses you may have about why what you observed happened.
- Your Python programs for the two networks.
- Output from the two programs[1]
  - Your output must include a line with the accuracy of your for test data.
  - It must also include the results of predicting the output given the inputs for the test data.

  For example, my output looks like this:

```
(keras) [rmt@rturner-2 NN]$ python iris.py
Using TensorFlow backend.
Epoch 1/150
132/132 [==============================] - 0s - loss: 0.6367 - acc: 0.6667
Epoch 2/150
132/132 [==============================] - 0s - loss: 0.6365 - acc: 0.6667
...
Epoch 150/150
132/132 [==============================] - 0s - loss: 0.3273 - acc: 0.7828
15/15 [==============================] - 0s

acc: 71.11%

Predictions (based on test dataset):
prediction=[ 0.11855328  0.44186708  0.43957967]
prediction=[ 0.98168379  0.01319122  0.00512492]
prediction=[ 0.09259017  0.44522405  0.46218583]
prediction=[ 0.10822026  0.44351041  0.44826928]
prediction=[ 0.09259017  0.44522405  0.46218583]
prediction=[ 0.09259017  0.44522405  0.46218583]
```

[1] I'd strongly suggest running the program, then getting a screen shot of the tail-end of the "Epoch..." things, the accuracy, and the predictions; as you will note when you run it, lots of control characters are embedded in the output, which makes printing or viewing any captured output problematic.

```
  prediction=[ 0.10330817  0.44415826  0.45253354]
  prediction=[ 0.09259017  0.44522405  0.46218583]
  prediction=[ 0.99175453  0.00614701  0.00209844]
  prediction=[ 0.09259017  0.44522405  0.46218583]
  prediction=[ 0.09259017  0.44522405  0.46218583]
  prediction=[ 0.09259017  0.44522405  0.46218583]
  prediction=[ 0.09259017  0.44522405  0.46218583]
  prediction=[ 0.12725127  0.44022453  0.43252417]
  prediction=[ 0.09259017  0.44522405  0.46218583]
```

In this case, the accuracy of the network was 71%, and (for example) the prediction for the last of the tests is that it is type 3, with a probability 0.46 (it actually is a type 3).