GALWAY-MAYO INSTITUTE OF TECHNOLOGY

Department of Computer Science & Applied Physics

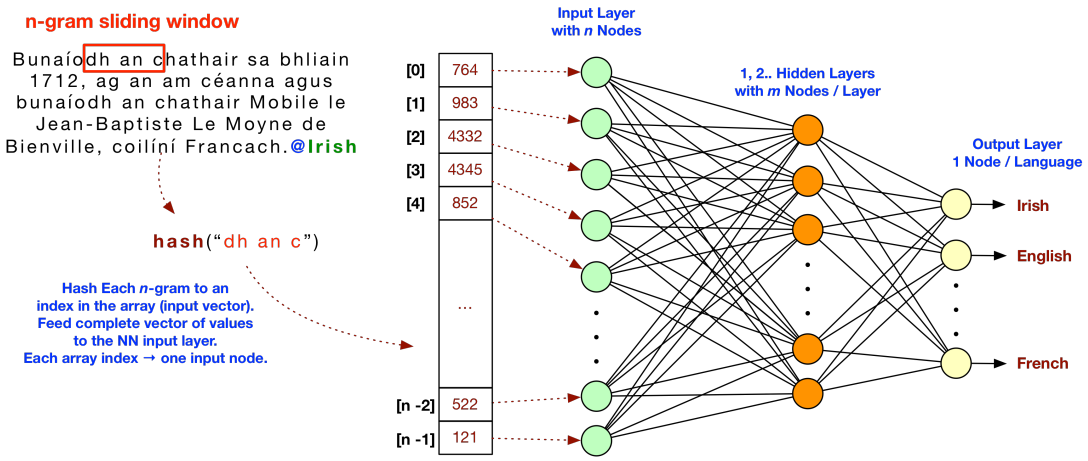
B.Sc. Software Development – Artificial Intelligence (2020)
SUPPLEMENTARY ASSIGNMENT

A Language Detection Neural Network with Vector Hashing

This assignment is offered in lieu of the written examination and constitutes the remaining 50% of the total marks for this module.

1. Overview

One of the limiting features of multi-layer perceptron is that the number of input neurons is fixed, causing issues when dealing with variable-length input sources such as text. Once way to circumvent this problem is to hash the inputs to a fixed-size vector and then map each of the vector indices to a node in the input layer of a neural network. When using text as a source, a common practice is to first decompose the text into a set of *n*-grams and then hash each of these. You are required to use the Encog 3.2 API to develop a multilayer neural network capable of detecting the language of text in a file. The text should be parsed, converted into *n*-grams and then hashed to a vector before being fed as input to a neural network. The process is shown below:



An *n*-gram, also called a shingle, *l*-mer or *k*-mer is a contiguous substring of text of size *n*. Their use is an example of a ***divide-and-conquer*** technique and *n*-grams have been applied to a wide variety of computing problems including text similarity measurement, text subject determination and language classification.

Consider the text "*object oriented programming is good fun for me*". This string can be decomposed into the following set of 5-grams: {"*objec*", "*t_ori*", "*ented*", "*_prog*", "*rammi*", "*ng_is*", "*_good*", "*_fun_*", "*for_m*", "*e____*"}. A set of *n*-grams can be formed from discrete "chunks" of non-overlapping characters or by tiling across a string using a fixed offset. For example, using an offset of 1, the following set of 5-grams will be formed: {"*objec*", "*bject*", "*ject_*", "*ect_o*" , "*ct_or*" , "*t_ori*"…}. Note that spaces are included in the *n*-gram set of a text

and that the total number of possible *n*-grams in an *n*-spectrum is $\sum^n$, where $\sum$ is the number of symbols in the alphabet for the text, e.g. the full *n*-spectrum for 5-gram lower-case words in English has 26^5 = 11,881,376 elements. In practical terms, an *n*-spectrum will not be fully realized, as the vast majority of *n*-gram combinations will not be present in text. It should also be clear that the frequency of a 2-gram appearing in a text will be much greater than that of a 5-gram. For example, if we only include lower-case characters in English, each character will appear with a probability of 1/26 = 0.038. A 2-gram will therefore occur with a probability of 0.038 * 0.038 = 0.001444 and a 5-gram with a probability of 0.038 * 0.038 * 0.038 * 0.038 * 0.038 = 0.038^5 = 7.92×10^{-8}.

You should train the neural network using the ***WiLI benchmark dataset*** (available on Moodle), consisting of 11,750 lines of text in 235 languages, with the text and language name delimited by an '@' symbol. The language name should be used as the label for each row of training data and places an upper bound on the number of nodes in the output layer.

```
быдыра, яву, губгъуэ яшъху, къуаджэхэр тIысху. ШIыпIэ хуиту здэпсоуфынхэр нэхъри нэхъ макIэ мэхъухэр. Фактхэмк Iэ Индиэмрэ Шри-Ланкамрэ хывхэр хуиту здэпсоуфын шIыпIу къэнэжьахэр къэралхэм я паркхэм (цIэрыIуэ къэрал парк Казиранга индиэ штат Ассамым де хыв гуп щопсо 1000 хуэдиз я бжыгъэр хъуурэ). Непалымрэ Бутанымрэ я Iохур тIэкIу нэхъыфIу щыт.@Kabardian
هاورد اتوود كيلى (بالإنجليزية: Howard Atwood Kelly) (20 فبراير 1858 - 12 يناير 1943) هو طبيب أمراض نساء أمريكي. بيعد مؤسس طب النساء كتخصص ذي كيان.@Egyptian Arabic
மூர்த்தி திருநாமம் :- அகிலாண்டேசுவரர் அம்பாள் பெயர் :- அகிலாண்டேசுவரி தலப்பெயர் :- கோயில்தும், வைகை தீர்த்தம் :- அக்நிதீர்த்தம் தலவிருட்சம் :- திருக்கொன்றை@Tamil
```

Each character of text can be represented by a Unicode UTF-16 number and by a **char** type in Java.

2. Minimum Requirements

- Use the package name ***ie.gmit.sw***. The application must be deployed and runnable using the specification in Section 3.
- Create a console-based ***menu-driven UI*** to input parameters to the application, e.g. vector size, *n*-gram characteristics, number of epochs. The UI should report the topology structure, the training time, and test statistics (sensitivity and specificity). The UI should also allow a text file to be specified with live data to be classified. Do not use a GUI!
- The application should be able to read in the text from a file specified through the menu and identify the language classification computed by an ***Encog 3.2 neural network***.
- The specified training, test and live data files should be parsed and processed using *n*-grams and hashed to a ***fixed-size feature vector*** of a specified size. Use the formula ***hashCode() % n*** to compute the vector index for an *n*-gram. The value at that index should be incremented for each "hit". The vector of values should be mapped to the set of input nodes in the neural network on a **1:1** basis, i.e. each vector index should map to one node.
- The neural network should be trained and tested using ***5-fold cross-validation***. The menu should report the sensitivity and specificity after testing. Use the formulae
    - **sensitivity (sn)** = TP (TP + FN)
    - **specificity (sP)** = TN / (TN + FP)

  Note that TP, TN, FP and FN can be represented as **double** variables and incremented using the actual (Y) and expected (Yd) outputs.
- Do NOT include the **Encog API**, the **WiLI benchmark dataset** or any **training data** with your submission. You can assume that *Encog 3.2* is already on the *CLASSPATH* and that the file *wili-2018-Small-11750-Edited.txt* is located in the current directory, i.e. refer to it using the **"./wili-2018-Small-11750-Edited.txt"** syntax.

You are free to asset-strip any online resources for functionality provided that you modify any code used and cite the source both in the README and inline as a code comment above the

relevant section of the programme. You are not free to re-use whole components and will only be given marks for work that you have undertaken yourself. Please pay particular attention to how your application must be packaged and submitted and the scoring rubric provided. Marks will only be awarded for features described in the scoring rubric.

3. Deployment and Delivery

Please **read the following carefully** and pay particular attention to the files that you are required to submit and those that you should not include with your assignment:

- ***The project must be submitted by midnight on Thursday 21st May 2020***. Before submitting the assignment, you should review and test it from a command prompt on ***a different computer*** to the one that you used to program the project.
- The project must be submitted as a Zip archive ***(not a 7z, rar or WinRar file)*** using the Moodle upload utility. You can find the area to upload the project under the "*A Language Detection Neural Network with Vector Hashing - (50%) Assignment Upload*" link on Moodle. Only the contents of the submitted Zip will be considered. ***Do not add comments to the Moodle assignment upload form.***
- The name of the Zip archive should be <*id*>.zip where <*id*> is your GMIT student number.
- The Zip archive should have the structure shown below. Do NOT submit the assignment as an Eclipse project.

Marks	Category
language-nn.jar	A Java archive containing your API and runner class with a ***main()*** method. You can create the JAR file using the following command from inside the "bin" folder of the Eclipse project: **jar –cf language-nn.jar *** The application should be executable from a command line as follows: **java –cp ./language-nn.jar ie.gmit.sw.Runner**
src	A directory that contains the packaged **source code** for your application.
README.txt	A text file detailing the **design and main features** of your application. Marks will only be given for features that are described in the README.

4. Marking Scheme

Marks for the project will be applied using the following criteria:

Element	Marks	Description
Structure	10	*All-or-nothing.* The packaging and deployment are both correct. The JAR file executes perfectly without any manual intervention when executed with the command **java –cp ./language-nn.jar ie.gmit.sw.Runner** and accesses the text resource as **./wili-2018-Small-11750-Edited.txt**.
README	25	All **features and their design rationale** are fully documented. Your README should clearly explain your rationale for the following: • The *n*-gram or other hashing approach used by your application. • The overall neural network topology. • The size of the hashing feature vector. • The number of hidden layers and the number of nodes in each layer. • The activation functions used in each layer. As this assignment is being given in lieu of the written examination, your README must address each of the items above clearly and comprehensively.

User Interface	7	A command-line menu-driven user interface allows users to pass in any parameters required by the application. The UI should report the topology structure, the training time, and test statistics (sensitivity and specificity). The UI should also allow a text file to be specified with live data to be classified.
Parsing & Hashing	20	The specified training / test / live data files should be parsed and processed using *n*-grams and hashed to a fixed-size feature vector.
Neural Network Configuration	5	The neural network should be configured using the Encog 3.2 API.
Training Time	18	The neural network should be trainable to > 98% accuracy in a short amount of time using 5-fold cross validation. Full marks will be given if the training can be done in < 3 mins, 10 marks for < 7 mins and 0 marks after 10 mins.
Confusion Matrix	10	The sensitivity and specificity of the neural network should be computed for the test data across all 5 cross-validations.
Extras	5	Only **relevant extras** that have been fully documented in the README.

You should treat this assignment as a project specification. Any deviation from the requirements will result in some or all marks not being earned for a category. Each of the categories above will be scored using the following criteria:

Range	Expectation
0–39%	Not delivering on basic expectations. Programme does not meet the minimum requirements.
40-60%	Meets basic expectations.
61–70%	Tending to exceed expectations. A high quality assignment with additional functionality added.
80-90%	Exceeding expectations. Any mark over 70% requires evidence of independent learning and cross-fertilization of ideas, i.e. your project must implement a set of features using advanced concepts not covered in depth during the module.
90-100%	Exemplary. Your assignment can be used as a teaching aid with little or no editing.