ROBERT GORDON UNIVERSITY ABERDEEN

CM3604 – Deep Learning

Academic Year	**2024/2025**
Semester	**Semester 1**
Module Number	**CM3604**
Module Title	**Deep Learning**
Assessment Method	**Coursework Element 2 – Group**
Deadline (time and date)	**17th November 2024 23:59**
Submission	**Assessment Dropbox in the Module Study Area in CampusMoodle.**
Word Limit (see Assessment Word Limit Statement)	**N/A**
Module Co-ordinator	**Prasan Yapa**

What knowledge and/or skills will I develop by undertaking the assessment?

The primary goal with this task is to give you hands-on experience implementing a neural network language model using recurrent neural networks. Understanding how these neural models work will help you understand not just language modelling, but also systems for many other applications such as machine translation. You may form a group of 3 to 4 members for this task and only one member should make the submission.

On successful completion of the assessment students will be able to achieve the following Learning Outcomes:

1. Build a neural model by selecting relevant functions and parameters using a state-of-the-art python framework.
2. Implement, test and transfer deep architectures for a given data science application.

Please also refer to the Module Descriptor, available from the module Moodle study area.

What is expected of me in this assessment?

Task(s) – content

Please use Python 3.5+ and PyTorch 1.0+ for this project. The dataset for this project is the Text collection by Matt Mahoney. This is a dataset taken from the first 100M characters of Wikipedia. Only 27-character types are present (lowercase characters and spaces); special characters are replaced by a single space and numbers are spelled out as individual digits (20 becomes two zero). A larger version of this benchmark (90M training characters, 5M dev, 5M test) was used in Mikolov et al. (2012).

The framework code you are given consists of several files. *lm_classifier.py* contains the driver for Part 1. It calls *train_rnn_classifier*, which learns an RNN classifier model on the classification data. *lm.py* contains the driver for Part 2, and calls *train_lm* on the raw text data. *models.py* contains skeletons in which you will implement these models and their training procedures. *utils.py* contains all the required utility logic to the framework.

Task(s) – format

Part 1: RNNs for Classification (40 Marks)

In this first part, you will do a simplified version of the language modelling task: binary classification of fixed-length sequences to predict whether the given sequence is followed by a consonant or a vowel. You will implement the entire training and evaluation loop for this model.

Data *train-vowel-examples.txt* and *train-consonant-examples.txt* each contain 5000 strings of length 20, and *dev-vowel-examples.txt* and *dev-consonant-examples.txt* each contain 500. The task is to predict whether the first letter following each string is a vowel or a consonant. The consonant file (for both train and test) contains examples where the next letter (in the original text, not shown) was a consonant, and analogously for the vowel file.

Getting started Run: *python lm_classifier.py*

What is expected of me in this assessment?

This loads the data for this part, learns a *FrequencyBasedClassifier* on the data, and evaluates it. This classifier gets 71.4% accuracy, where random guessing gets you 50%. *lm_classifier.py* contains the driver code, and the top of *models.py* contains the skeletal implementation for this classifier.

Q1 (30 Marks) Implement an RNN classifier to classify segments as being followed by consonants or vowels. This will require defining a PyTorch module to do this classification, implementing training of that module in *train_rnn_classifier*, and finally completing the definition of *RNNClassifier* appropriately to use this module for classification. In your report, you should: (1) Describe your model and implementation. (2) Report accuracy.

Network structure: The inputs to your network will be sequences of character indices. You should first embed these using a *nn.Embedding* layer and then feed the resulting tensor into an RNN. Two effective types of RNNs to use are *nn.GRU* and *nn.LSTM*. Finally, you should take the output of the RNN (the last hidden state) and use it for classification (optionally put it through feedforward layers and then a SoftMax layer). You can make your own *nn.Module* that wraps the embedding layer, RNN, and classification layer.

Code structure: First, you need a function to go from the raw string to a PyTorch tensor of indices. Then loop through those examples, zero your gradients, pick up an example, compute the loss, run backpropagation, and update parameters with your optimizer. You should implement this training in *train_rnn_classifier*.

Using RNNs: LSTMs and GRUs can be a bit trickier to use than feedforward architectures. First, these expect input tensors of dimension [sequence length, batch size, input size]. You can use the *batch_first* argument to switch whether the sequence length dimension or batch dimension occurs first. If you're not using batching, you'll want to pad your sentence with a trivial 1 dimension for the batch. *unsqueeze* allows you to add trivial dimensions of size 1, and *squeeze* lets you remove these. Second, an LSTM takes as input a pair of tensors representing the state, *h* and *c*. Each is of size [num layers * num directions, batch size, hidden size]. To start with, you probably want a 1-layer RNN just running in the forward

direction, so once again you should use *unsqueeze* to pad things out. GRUs are similar but only have one hidden state.

Tensor manipulation: *np.asarray* can convert lists into numpy arrays easily. *torch.from_numpy* can convert numpy arrays into PyTorch tensors. *torch.FloatTensor(list)* can convert from lists directly to PyTorch tensors. *.float()* and *.int()* can be used to cast tensors to different types. For these and other commands, read the error messages you get out and check the documentation.

General tips: As always, make sure you can overfit a very small training set as an initial test. If not, you probably have a bug. Then scale up to train on more data and check the development performance of your model. Consider using small values for hyperparameters so things train quickly. With only 27 characters, you can get away with small embedding sizes for these, and small hidden sizes for the RNN may work better than you think!

Q2 (10 Marks) What happens if you limit the amount of context the model uses? Try a few different values for the number of context characters used (up to a max of 20, which is the default in the dataset). What trends do you observe?

Part 2: Implementing a Language Model (60 Marks)

In this section, you will implement an RNN language model. This should build heavily off what you did for Part 1, though new ingredients will be necessary, particularly during training using *text8-100k.txt* and *text8-dev.txt*.

Getting started Run: *python lm.py*

This loads the data, instantiates a *UniformLanguageModel* which assigns each character an equal $\frac{1}{27}$ probability, and evaluates it on the development set. This model achieves a total log probability of -1644, an average log probability (per token) of -3.296, and a perplexity of 27. Note that exponentiating the average log probability gives you $\frac{1}{27}$ in this case, which is

the inverse of perplexity. As a probability, perplexity can be interpreted as a measure of the "branching factor."

Q3 (60 Marks) Implement an RNN language model. This will require defining a PyTorch module to handle language model prediction, implementing training of that module in *train_lm*, and finally completing the definition of *RNNLanguageModel* appropriately to use this module for classification. Your network should take indexed characters as input, embed them, put them through an RNN, and make predictions from the final layer outputs. In your report, you should: (1) Describe your model and implementation. (2) Report accuracy.

Chunking the data: Unlike classification, language modelling can be viewed as a task where the same network is predicting words at many positions. Your network should process a chunk of characters at a time, simultaneously predicting the next character at each index in the chunk. You'll have to decide how you want to chunk the data. Given a chunk, you can either initialize the RNN state with a zero vector (not quite right), "burn in" the RNN by running on a few characters before you begin predicting or carry over the end state of the RNN to the next state.

Start of sequence: In general, the beginning of any sequence is represented to the language model by a special start-of-sequence token. This means that the inputs and outputs of a language model are slightly different, since an LM will never output the start-of-sequence character but may need to read it as input.

Evaluation: The model should be evaluated on perplexity and likelihood.

Submission: You should submit the following files separately.

1. A PDF of your answers to the questions.
2. *models.py*. **Do not modify or upload any other source files.**

Make sure that the following commands work before you submit:

python lm_classifier.py --model RNN

What is expected of me in this assessment?
python lm.py --model RNN These commands should run without errors.

How will I be graded?	
A grade will be provided for each criterion on the feedback grid which is specific to the assessment. The overall grade for the assessment will be calculated using the algorithm below.	
A	At least 50% of the feedback grid to be at Grade A, at least 75% of the feedback grid to be at Grade B or better, and normally 100% of the feedback grid to be at Grade C or better.
B	At least 50% of the feedback grid to be at Grade B or better, at least 75% of the feedback grid to be at Grade C or better, and normally 100% of the feedback grid to be at Grade D or better.
C	At least 50% of the feedback grid to be at Grade C or better, and at least 75% of the feedback grid to be at Grade D or better.
D	At least 50% of the feedback grid to be at Grade D or better, and at least 75% of the feedback grid to be at Grade E or better.
E	At least 50% of the feedback grid to be at Grade E or better.
F	Failing to achieve at least 50% of the feedback grid to be at Grade E or better.
NS	Non-submission.

ROBERT GORDON UNIVERSITY ABERDEEN

Feedback grid

GRADE	A	B	C	D	E	F
DEFINITION / CRITERIA (WEIGHTING)	**EXCELLENT** Outstanding Performance	**COMMENDABLE/VERY GOOD** Meritorious Performance	**GOOD** Highly Competent Performance	**SATISFACTORY** Competent Performance	**BORDERLINE FAIL**	**UNSATISFACTORY** Fail
RNN Classifier **(30%)** **Grade:** ☐	Excellent use of an optimal network structure, proper usage of embeddings selecting optimal tensor dimensions and feeding the resulting tensor into the selected network, proper usage of feedforward layers and activation functions, demonstrating backpropagation and computing the loss, excellent showcase of hyper-parameter settings and excellent error/exception handling.	Very good use of an optimal network structure, proper usage of embeddings selecting optimal tensor dimensions and feeding the resulting tensor into the selected network, very good usage of feedforward layers and activation functions, demonstrating backpropagation and computing the loss, very good showcase of hyper-parameter settings and commendable error/exception handling.	Good use of an optimal network structure, proper usage of embeddings selecting optimal tensor dimensions and feeding the resulting tensor into the selected network, good usage of feedforward layers and activation functions, demonstrating backpropagation and computing the loss, good showcase of hyper-parameter settings and error/exception handling.	Satisfactory use of an optimal network structure, usage of embeddings selecting tensor dimensions and feeding the resulting tensor into the selected network, satisfactory usage of feedforward layers and activation functions, demonstrating backpropagation and computing the loss, showcase of hyper-parameter settings and error/exception handling.	Lacks understanding on the use of an optimal network structure, usage of embeddings selecting tensor dimensions and feeding the resulting tensor into the selected network, improper usage of feedforward layers and activation functions, poorly demonstrating backpropagation and computing the loss, hyper-parameter settings and error/exception handling.	No or very limited evidence in using an optimal network structure, usage of embeddings selecting tensor dimensions and feeding the resulting tensor into the selected network, poor usage of feedforward layers and activation functions, no evidence in demonstrating backpropagation and computing the loss, hyper-parameter settings and error/exception handling.
Limiting Context **(10%)** **Grade:** ☐	Excellent understanding of using different values and combinations for the number of context characters, limiting the amount of context the base model uses.	Very good understanding of using different values and combinations for the number of context characters, limiting the amount of context the base model uses.	Highly competitive understanding of using different values and a few combinations for the number of context characters, limiting the amount of context the base model uses.	Satisfactory usage of different values and a very few combinations for the number of context characters, limiting the amount of context the base model uses.	Lacks understanding on the usage of different values and combinations for the number of context characters, limiting the amount of context the base model uses.	No or very limited evidence in understanding the usage of different values and combinations for the number of context characters, limiting the amount of context the base model uses.

GRADE	A	B	C	D	E	F
DEFINITION / CRITERIA (WEIGHTING)	**EXCELLENT** Outstanding Performance	**COMMENDABLE/VERY GOOD** Meritorious Performance	**GOOD** Highly Competent Performance	**SATISFACTORY** Competent Performance	**BORDERLINE FAIL**	**UNSATISFACTORY** Fail
RNN Language Model **(40%)** **Grade:** ☐	Outstanding performance and understanding to process a chunk of characters at a time, predicting the next character at each index in the chunk, initializing the RNN state with a zero vector, and the manipulation of a special start-of-sequence token.	Very good understanding to process a chunk of characters at a time, predicting the next character at each index in the chunk, initializing the RNN state with a zero vector, and the manipulation of a special start-of-sequence token.	Highly competitive understanding to process a chunk of characters at a time, predicting the next character at each index in the chunk, initializing the RNN state with a zero vector, and the manipulation of a special start-of-sequence token.	Competitive performance to process a chunk of characters at a time, predicting the next character at each index in the chunk, initializing the RNN state with a zero vector, and the manipulation of a special start-of-sequence token.	Lacks understanding to process a chunk of characters at a time, predicting the next character at each index in the chunk, initializing the RNN state with a zero vector, and the manipulation of a special start-of-sequence token.	No or very limited evidence to process a chunk of characters at a time, predicting the next character at each index in the chunk, initializing the RNN state with a zero vector, and the manipulation of a special start-of-sequence token.
Model Evaluation **(20%)** **Grade:** ☐	Outstanding performance and understanding to evaluate the language model on perplexity and likelihood, execution of commands without errors, and documentation.	Very good understanding to evaluate the language model on perplexity and likelihood, execution of commands without errors, and documentation.	Good understanding to evaluate the language model on perplexity and likelihood, execution of commands without errors, and documentation.	Satisfactory understanding to evaluate the language model on perplexity and likelihood, execution of commands without errors, and documentation.	Lacks understanding to evaluate the language model on perplexity and likelihood, execution of commands without errors, and documentation.	No or very limited evidence to evaluate the language model on perplexity and likelihood, execution of commands without errors, and documentation.

Coursework received late, without valid reason, will be regarded as a non-submission (NS) and one of your assessment opportunities will be lost.

ROBERT GORDON UNIVERSITY ABERDEEN

What else is important to my assessment?

What is plagiarism?

"Plagiarism is the practice of presenting the thoughts, writings or other output of another or others as original, without acknowledgement of their source(s) at the point of their use in the student's work. All materials including text, data, diagrams or other illustrations used to support a piece of work, whether from a printed publication or from electronic media, should be appropriately identified and referenced and should not normally be copied directly unless as an acknowledged quotation. Text, opinions or ideas translated into the words of the individual student should in all cases acknowledge the original source" (RGU 2022).

What is collusion?

"Collusion is defined as two or more people working together with the intention of deceiving another. Within the academic environment this can occur when students work with others on an assignment, or part of an assignment, that is intended to be completed separately" (RGU 2022).

For further information please see Academic Integrity.

What is the Assessment Word Limit Statement?

It is important that you adhere to the Word Limit specified above. The Assessment Word Limit Statement lists what is included and excluded from the word count, along with the penalty for exceeding the upper limit.

What if I'm unable to submit?

- The University operates a Fit to Sit Policy which means that if you undertake an assessment then you are declaring yourself well enough to do so.
- If you require an extension, you should complete and submit a Coursework Extension Form. This form is available on the RGU Student and Applicant Forms page.
- Further support is available from your Course Leader.

What else is important to my assessment?

What additional support is available?

- RGU Study Skills provide advice and guidance on academic writing, study skills, maths and statistics and basic IT.
- RGU Library guidance on referencing and citing.
- The Inclusion Centre: Disability & Dyslexia.
- Your Module Coordinator, Course Leader and designated Personal Tutor can also provide support.

What are the University rules on assessment?

The University Regulation 'A4: Assessment and Recommendations of Assessment Boards' sets out important information about assessment and how it is conducted across the University.