Third deep learning project

(Language model using recurrent neural network)

Supervised by: prof. Hazem Abbas

Team members:

Zeyad Tarek Mohamed

Mahmoud Adel Khorshed

Ahmed Salama

Queen’s university, School of computing

April 5, 2022

**Introduction**

A language model is nothing more than a probability distribution over words or word sequences. In practice, a language model predicts the likelihood of a given word sequence being "valid." In this situation, validity does not refer to grammatical validity at all. It implies that it is like how humans talk (or, more precisely, write) — which is what the language model learns.

**Problem Formulation**

In this problem, we want to predict what’s the next word after we write a sentence, we want to build a language model which can predict what's the next word and what's the probability of this word being the next word, The model will consider the context of the last 50 words of a particular sentence and predict the next possible word. We will be using methods of natural language processing, language modeling, and deep learning. We will start by analyzing the data followed by the pre-processing of the data. We will then tokenize this data and convert them into sequences and finally, we'll build and train some deep learning models and compare their performance with each other.

**Data**: dirty text file have **119112** words and **7490** unique words.

**Input**: **118632\*50** training sequences. (**118632** here represent the number of samples and **50** represent the number of input words)

**Output**: **118632** training sequences. (Each value represents that output word).

**Data mining function:**

1. Load and read the text data.
2. Analyzing and pre-processing the data.
3. Build and train the models
4. Classification and prediction
5. Get insights from the results.

**Challenges:**

1. These types of problems require a huge amount of data and what we have here is a small one.
2. This is an open-ended problem so we have a lot of model architectures that we may want to try and choose the best one.
3. Non-accurate results due to a lack of information.
4. We use recurrent models with a lot of recurrent units to memorize the information, so we need huge number of resources, especially **RAM**, at least **(16 GIGABYTE of RAM).**

**Impact:**

Solving kind of this problem will help people a lot when they search for something on the internet and they don’t remember the exact context of the sentence, also when they send an email or while they’re using chats, that will save much time.

**Experimental protocol**

1. Load and read the data.
2. Understand the nature of this data
3. Pre-processing the text data
   1. Remove punctuation from the text file.
   2. Remove non-alphabetic words
   3. Tokenize the text data.
   4. Convert each token to numerical value.
   5. Organize the numerical data into 118,632 training sequences each sequence has 51 values.
   6. Save 50 value from each sequence in a variable and they will represent the input data and save the last numerical value from each sequence in a variable, and they will represent the output data.
   7. Convert the output feature to one hot encoded data.
4. Start building the models
5. Train each model.
6. Plotting the performance graph
7. Start predicting in the context of last 100 words.
8. Get insights from the results.

**Code Explanation**

This code ran on Kaggle environment using **Nvidia Tesla P100 GPUs** to train the deep learning models and **16GB of RAM**,

Note: less than these resources the model may crash because I used 1000 units in each recurrent layer.

So please try to run it on Kaggle environment.

First cell: I imported all required packages.

You can see what each package will do for us.

Text

Description automatically generated

Second cell:

I Started loading and reading the data and I clean the text file; I removed the punctuation, and I removed the non-alphabetic words, then I took **101** words from that file to use them as test data, then I convert the training data and test data to lower case.

Text

Description automatically generated

Third cell:

Here I continued pre-processing the text data, I tokenized the text data using keras tokenizer and I convert each **token** (word) to **ID** (numerical value).

Graphical user interface, text, application

Description automatically generated

**Note:** The number of **IDs** is equal to the number of words in the text file.

Fourth Cell:

I just saw how many unique words in the text file.

Graphical user interface, text, application

Description automatically generated

Fifth cell (important step):

I organized the **IDs** into sequences each sequence has **51** values.

Graphical user interface, text, application

Description automatically generated

Sixth cell:

I stored the **50** values in each sequence in one variable, and they will represent the input data, and the last word will be the predicted word and I stored them in another variable.

Text

Description automatically generated

Seventh cell:

Convert output words to one hot encoded data because one hot encoded output is better with multi-class classification problem.

Graphical user interface, text

Description automatically generated with medium confidence

Eighth cell:

Function to plot the validation loss and training loss over epochs and validation accuracy and training accuracy over epochs for one model at time, and it takes the history of the model as parameter.

Text

Description automatically generated Text

Description automatically generated with medium confidence

Nineth cell:

Function to predict the probabilities of all unique words and take the three highest (and maybe the correct next word will be one of these three words.), and it will print to probable 3 next word and their probabilities.

Note: more explanation in the test part.

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

**Start building, training and testing the models.**

I built six recurrent models, to test them on the test data and choose the best one.

All the models have common things and I’ll explain them in the next page.

* **Embedding layer:** Represent words as semantically meaningful dense real-valued vectors and uses a distributed representation for words so that different words with similar meanings will have a similar representation. and its shape will be **(the number of unique words \* input length)**.
* **Dense layer:** dense layer after recurrent layer to add more trainable parameters and to improve the accuracy.
* **Output layer:** will have 7410 neurons (number of unique words) each neuron represents the probability of the next word (highest probability will be the next word)
* All models have **Adam optimizer with nesterov momentum** and learning rate **0.001**, and I used it instead of **Adam optimizer with momentum**, because **Nadam** will reach to the optimal minimum faster than **Adam optimizer**.
* All models have **categorical\_crossentropy** as **loss function method** and **categorical\_accuracy** to calculate the accuracy since the output is **one-hot-encoded data**.
* All models will be optimized using **50 epochs** and **200 batch size** to make the training process faster.
* All models will be trained on **GPU**.
* In the training process there’s an **early stopping monitor** to ensure that the validation accuracy increases after every epoch otherwise it will stop after **5 epochs**.
* Each model will be saved in the **h5 file** format, and this file will be the best trained model with **best validation accuracy** to use it again in the prediction step.
* The data will be divided into the **training data (80% of the whole data)** and the **validation data (20% of the whole data)** just before the training process.
* The difference between each model is, that is each model have different recurrent neural networks between the embedding layer and the dense layer.
* The test data used to test all models is: input data: but at my age i can hardly get to the city and therefore you should come oftener to the piraeus for let me tell you that the more the pleasures of the body fade away the greater to me is the pleasure and charm of conversation do not then deny my request but make our house your resort and keep company with these young men we are old friends and you will be quite at home with us i replied there is nothing which for my part i like better cephalus than conversing with aged men for i regard them .....

And the output word is **‘as’**

**Building the first model**

In the first model I used a simple recurrent neural network in order to see if this kind of problems can be handled by the simplest recurrent neural networks or not.

**Total params: 2,505,518**

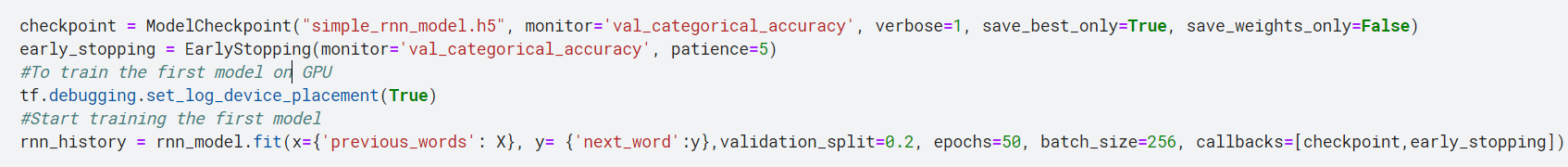
**Trainable params: 2,505,518**

**Non-trainable params: 0**

Diagram

Description automatically generated

**Training the first model**



Total time taken to finish the training process was: 15 minutes and 32 seconds.

**Building the second model**

In this model I used a LSTM network with 1000 units followed by GRU network with 736 units because they have gates to forget data and memorize the others also, they are better in resisting the vanishing gradient and gradient explosion problems.

**Total params: 9,462,230**

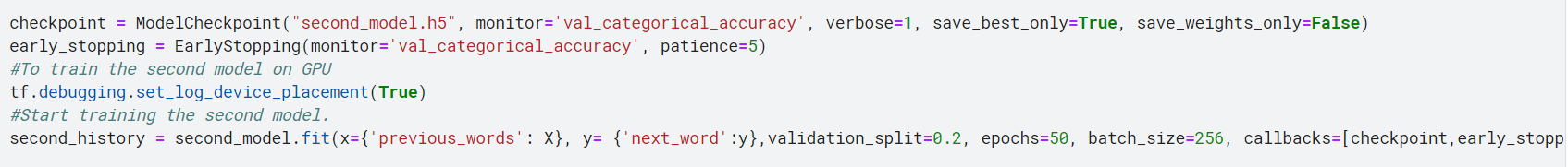
**Trainable params: 9,462,230**

**Non-trainable params: 0**

**Diagram

Description automatically generated**

**Training the second model**



Total time taken to finish the training process was: 15 minutes and 31 seconds.

**Building the third model**

In this model I used a GRU network followed by LSTM network to see if the change in orders will improve accuracy or not.

**Total params: 9,690,454**

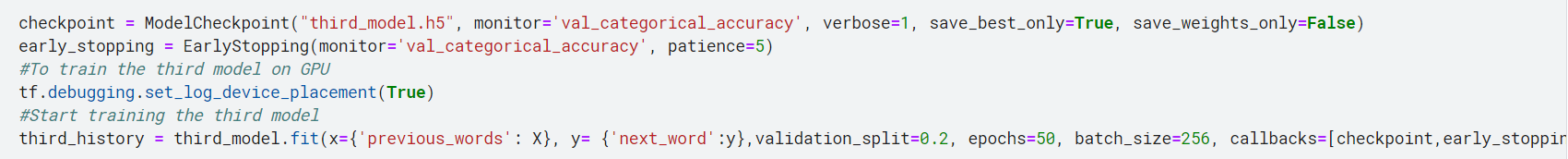
**Trainable params: 9,690,454**

**Non-trainable params: 0**

**Diagram

Description automatically generated**

**Training the third model**



Total time taken to finish the training process was: 23 minutes and 25 seconds.

**Building the fourth model**

In this model I used a Bidirectional LSTM network with 1000 units in order to have and memorize the sequence information in both directions, backwards (future to past) and forward (past to future), in order to improve the accuracy.

**Total params: 9,990,518**

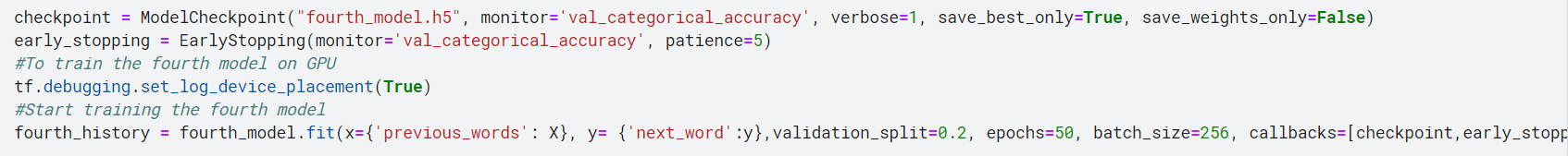
**Trainable params: 9,990,518**

**Non-trainable params: 0**

Diagram

Description automatically generated

**Training the fourth model**



Total time taken to finish the training process was: 34 minutes and 10 seconds.

**Building the fifth model**

In this model I used a Bidirectional GRU network with 1000 units instead of Bidirectional LSTM because it’s more efficient than LSTM network, and to see if that will improve the accuracy or not.

**Total params: 7,894,518**

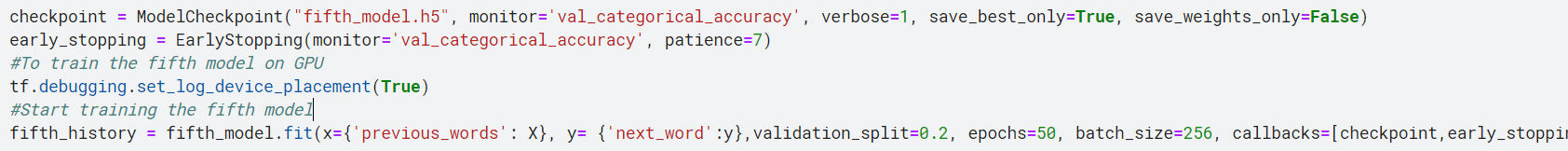
**Trainable params: 7,894,518**

**Non-trainable params: 0**

**Diagram, table

Description automatically generated**

**Training the fifth model**



Total time taken to finish the training process was: 16 minutes and 2 seconds

**Building the last model**

Since the bidirectional models was the best ones so, I tried to make a more complex model by using a bidirectional GUR network with 1000 units followed by bidirectional LSTM network with 1000 units, to see if more complex models can handle this kind of problems or not or it will be the same as simple ones.

**Total params: 23,942,390**

**Trainable params: 23,942,390**

**Non-trainable params: 0**

**Table

Description automatically generated**

**Training the last model**



Total time taken to finish the training process was: 30 minutes and 40 seconds.

**Results and comparisons**

**First model results**

*Best training loss: 4.4681*

*Best validation loss: 5.8874*

*Best training accuracy: 0.1791*

*Best validation accuracy: 0.16037*

**Performance curve and convergence curve for the first model**

**A picture containing shape

Description automatically generatedChart

Description automatically generated with medium confidence**

Results intuition: Here we find that the simplest model not the best in handling kind of this problems, but it can deal with it.

**Start testing the first model**

**Graphical user interface, text, application

Description automatically generated**

The output will be:

Correct word is [as]

the predict next word will be one of these three words **(the higher probability the higher chance to be the next word)**

the predicted next word is **[and]** with the largest probability **0.11699999868869781**

the next word could be **[in]** with the second largest probability **0.04899999871850014**

and it could be **[to]** with the third largest probability **0.03500000014901161**

That means the next word will be **“and”** although the correct word is **“as”.**

I’ll add more comments in all **results and comparisons** sections.

**Second model results**

*Best training loss: 6.0692*

*Best validation loss: 6.3443*

*Best training accuracy: 0.0586*

*Best validation accuracy: 0.06634*

**Performance curve and convergence curve for the second model**

**Line chart

Description automatically generated with low confidenceChart, line chart

Description automatically generated**

Results intuition: one of the worst models you may use for this kind of problems, the models start overfitting from the first epoch even there’s a dropout layer between GRU layer and the dense layer.

**Start testing the second model**

**Graphical user interface, text, application, Word

Description automatically generated**

The output will be:

Correct word is **[as]**

the predict next word will be one of these three words **(the higher probability the higher chance to be the next word)**

the predicted next word is **[the]** with the largest probability **0.035999998450279236**

the next word could be **[and]** with the second largest probability **0.029999999329447746**

and it could be **[of]** with the third largest probability **0.026000000536441803**

**Third model results**

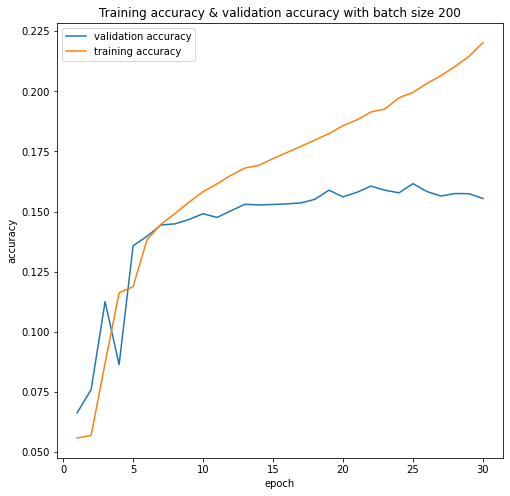
*Best training loss: 4.1857*

*Best validation loss: 5.7769*

*Best training accuracy: 0.2202*

*Best validation accuracy: 0.16159*

**Performance curve and convergence curve for the third model**

**Chart, line chart

Description automatically generated**

Results intuition: From this model results I found that the order of layers can flip the game although they are the same layers but what I did is I reverse the order and that’s make the accuracy and the loss much better than the previous model.

**Start testing the third model**

**Chart

Description automatically generated with low confidence**

The output will be:

Correct word is **[as]**

the predict next word will be one of these three words **(the higher probability the higher chance to be the next word)**

the predicted next word is **[in]** with the largest probability **0.08299999684095383**

the next word could be **[and]** with the second largest probability **0.050999999046325684**

and it could be **[to]** with the third largest probability **0.03799999877810478**

**Fourth model results**

*Best training loss: 3.4581*

*Best validation loss: 5.6837*

*Best training accuracy: 0.2803*

*Best validation accuracy: 0.16479*

**Performance curve and convergence curve for the fourth model**

**Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated**

Results intuition: Using one bidirectional LSTM with 2000 units was better choice than GRU layer with 1000 unit followed by LSTM layer with 1000 and it got better accuracy.

**Start testing the fourth model**

**Graphical user interface, Word

Description automatically generated with medium confidence**

The output will be:

Correct word is **[as]**

the predict next word will be one of these three words **(the higher probability the higher chance to be the next word)**

the predicted next word is **[to]** with the largest probability 0.07500000298023224

the next word could be **[in]** with the second largest probability 0.05299999937415123

and it could be **[and]** with the third largest probability 0.04899999871850014

**Fifth model results**

*Best training loss: 3.0848*

*Best validation loss: 5.6880*

*Best training accuracy: 0.3509*

*Best validation accuracy: 0.17069*

**Performance curve and convergence curve for the fifth model**

**Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated**

Results intuition: This trial was one of the best trials after the next model, it got the best training accuracy, and it was faster than previous trail in training time

**Start testing the fifth model**

**Graphical user interface, text

Description automatically generated**

The output will be:

**Sixth model results**

*Best training loss: 3.4577*

*Best validation loss: 5.6933*

*Best training accuracy: 0.3061*

*Best validation accuracy: 0.1713*

**Performance curve and convergence curve for the sixth model**

**Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated**

Results intuition: Actually, merging bidirectional GRU with bidirectional LSTM give us the best intuition which the best method to solve this problem is to use complex bidirectional models to save the patterns from future to past and from past to future and this model was the best model compared to all previous models.

**Start testing the sixth model**

**Graphical user interface

Description automatically generated**

The output will be: Correct word is **[as]**

the predict next word will be one of these three words **(the higher probability the higher chance to be the next word)**

the predicted next word is **[and]** with the largest probability **0.07900000363588333**

the next word could be **[to]** with the second largest probability **0.05299999937415123**

and it could be **[that]** with the third largest probability **0.05000000074505806**

**All models’ results comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training loss | Validation loss | Training accuracy | Validation accuracy |
| First model | 4.4681 | 5.8874 | 17.91% | 16.04% |
| Second model | 6.0692 | 6.3443 | 5.86% | 6.634% |
| Third model | 4.1857 | 5.7769 | 22.02% | 16.159% |
| Fourth model | 3.4581 | 5.6837 | 28.03% | 16.497% |
| Fifth model | 3.0848 | 5.6880 | %35.09 | %17.09 |
| Last model | 3.4577 | 5.6933 | 30.61% | 17.31% |

Why these results and outputs has low accuracy?

I found that we don’t have to have large validation or training accuracy to tell if the model is good or not, because we don’t have one word to be the only word to come after specific sentence.

Ex: we have the test sentence that we used to test our models and the correct value was **‘as’** and it was the predicted word, but it doesn’t have to be **‘as’**, it can be **‘and’** or **‘to’** like some models decided as long as it does not conflict with the English grammars.

Ex 2: test data: deep neural ….,

The predicted word in this example could be **‘models’** or **‘networks’** so it doesn’t have to be one specific value.

And that what gave us low training and categorical accuracy.

**Conclusion**

Can we increase the accuracy of our models and predicted the exact correct word?

Yes, we can do that by following some steps:

1. Increase the number of sequences, The more data you train, the better results and accuracy you will get, but we’ll need more computational resources.
2. Use more complex recurrent models and take in consideration the overfitting problem.
3. Use pre-trained models like GPT-3.

**REF**

<https://towardsdatascience.com/the-beginners-guide-to-language-models-aa47165b57f9>

https://towardsdatascience.com/next-word-prediction-with-nlp-and-deep-learning-48b9fe0a17bf