IDS 594: MLOPs

Final Project

Recurrent Neural Network Deployment

Babandeep Singh & Blake Weston

Project Goal and Possible Alternatives

- This project revolves around creating a web application which is able to read a book and generate the next paragraph of the book worth 200 words given 5 input words.
- The web application is deployed on the Ubuntu instance on SaaS platforms such as Amazon Web Services (AWS).
- The Deep learning model is a Recurrent Neural Network (RNN) that remembers each information through time. It is also useful in time series prediction only because of the feature to remember previous inputs as well.
- This learning can be further improved and utilized in automatic replies to emails, chats, writing articles, etc. To summarize, Natural Language Generation tasks to further differentiate between human and machine generated text.

Methodologies (overview of steps)

Building the Model: Build the RNN model in python using Torch library.

Web Design: Update html code to format the local site where the model is implemented.

Ubuntu: Implement the model in Ubuntu.

AWS: Connect the instance using SSH and deployment

Language and Tools

We have used python 3.8 which is an open-source scripting language. There are many open- source tools such as R, MATLAB which are equally efficient in implementing such practices. Python was chosen since we are most familiar with this language's syntax and packages.

For the web application, we used the flask application because Flask provides simplicity, flexibility and fine-grained control. It is un-opinionated (it lets you decide how you want to implement things). Alternatively, Django was an alternative because Django provides an all-inclusive experience: you get an admin panel, database interfaces, an ORM, and directory structure for your apps and projects out of the box. Since this project does not require inter-database dependencies, the Django capabilities were out of scope.

Model

We have used a deep learning method (RNN), whereas simpler Markov N-gram models could be used. Even though Markov models are simpler, they suffer from scarcity (curse of dimension) and high perplexity problems. To eradicate such dependencies, RNNs were used to remember and generate coherent text.

Deployment

We deployed our model on EC2 instances offered by AWS. Other platforms such as Heroku, Google Cloud Platform, etc. were possibilities. All these platforms are either costly or able to deploy simpler Machine learning models, even though they are simpler to deploy against EC2 instances.

Pros

Recurrent Neural Networks

- Stores each iteration throughout time. Ability to train itself based on historical runs.
- RNN can process inputs of any length.
- Model size does not increase as input size increases. This is important because it does not make the computation more expensive or perform slower[1].

Ubuntu

- Open Source: Many users can provide strengthened security and troubleshooting when issues arise.
- Frequently updated software with extensive user-community. Active user base makes it easy to find support for any issue. More secure from input from many experts who all work on ensuring maximum security.
- Long Term Support for each release and dedicated professional support.
- Lightweight

AWS

- One of the most trusted and widely used cloud computing services in the world.
- Scalable capacity: as projects increase, you can purchase additional space. Not 'use it or lose it' storage tiers.
- Quick set-up and application deployment.
- Highly flexible, and AWS reduces cost as you only pay for what you need.

Cons

Recurrent Neural Networks

- Training RNN models can be difficult and slow, due to the recurrent nature.
- Prone to issues such as exploding and gradient vanishing[1].

Ubuntu

- Limited functionality due to limited range of applications.
- Linux shell can be user-unfriendly.

[1] https://www.educba.com/recurrent-neural-networks-rnn/

AWS

- Susceptible to cloud computing glitches, albeit rare.
- EC2 limitations: The service only comes with a prescribed set of limitations in terms of volume, images, and snapshots.
- Setup cost is high as there are many preliminary steps that need to be taken.

Model Development

The dataset selected was Harry Potter and Sorcerer's Stone book. Due to the nature of enriched vocabulary and a coherent story line, Harry Potter and the Sorcerer's Stone book is selected as our dataset. While processing the file, we removed special characters, extra white spaces, and any extra new lines to reduce dimensionality and redundancy. After cleaning the data, we generated 5-grams. Recursively, inputting 4 words and getting the next word and sliding over to the next window from the top 5 probable next 5-grams window.

The RNN module is implemented using PyTorch. The neural network consists of a single embedding layer and LSTM layer followed by a fully connected dense layer. We have used Stochastic Gradient Descent (SGD) as our optimizer and cross entropy loss to learn weights of the model. Few parameters used were embedding size of 300, batch size of 64, Long Short-Term memory of size 64 and for 500 epochs, with a learning rate of 1 was selected; large learning rate allowed us to learn faster, at the cost of arriving on a sub-optimal final set of weights.

Web Application

We utilized Flask Application methodology. We made a web page, which consists of Introduction and an input text box with a generate button to generate the next 200 words.

Deployment Procedure

1. Create instance.
    a. Select ubuntu free tier.
    b. Select t2. Free tier
    c. save the .pem file in the to be deployed model.
2. Download putty & puttygen to connect with the ubuntu.
3. Open puttygen
    a. Load the .pem file.
    b. Save the private key in the same folder.
4. On AWS
    a. Connect the instance.
    b. get/copy the hostname from SSH client and username.
5. Download WinSCP [https://winscp.net/eng/download.php] and open it.
    a. Paste the hostname and username in the prompt from step 4.
    b. Go to “Advanced” -> “authentication” -> add .ppk file from step 3 -> ok.
    c. Login

d. Before dragging and dropping make sure the application is running at host = 0.0.0.0 port = 8080 -> "*app.run(host='0.0.0.0', port = 8080)"*
e. Upload the files into ubuntu machine by dragging and dropping all the essential files [see appendix 1]

6. Installing necessary libraries {open putty downloaded in step 2}
    a. Add host name from step 4.
    b. In saved sessions distinct name
    c. Go to ssh -> authentication -> add .ppk file.
    d. Go back to -session -> save it.
    e. Click open -> login as username.
    f. This will install python to run the program and essential libraries for python ***Sudo apt-get update && sudo apt-get install python3-pip.***
7. Making the server accessible
    a. Go to security groups from the left drop down list under network and security.
    b. Create a new security group.
        i. Go to inbound rules.
            1. Add new rule.
            2. Make it for Type: *Any traffic*, Source: *Anywhere*.
            3. Create security groups.
8. Connecting to the security group
    a. Go to network interfaces.
    b. Change security groups {via right click}.
    c. Search the access group created and add it and Save.
9. Open command line through the putty and login
10. Install libraries for the app pip3 install -r requirements.txt.
    a. Torch installation -> *pip install torch==1.7.1+cu92 torchvision==0.8.2+cpu torchaudio==0.7.2 -f https://download.pytorch.org/whl/torch_stable.html*
11. Now run the "appEC2.py" in putty. *Python3 appEC2.py*
    a. *If it shows error, install all the libraries. Update the text file. Through re-uploading the edited files.*
12. Open the hostname + ":8080".
13. The model should have been deployed at the server.

Example: ec2-3-141-36-81.us-east-2.compute.amazonaws.com:8080

Deployment Experience and lessons

While There are many self-hosted and managed deployment technologies available, we experienced deploying with an efficient deployment technology EC2. After a few hiccups on connecting and understanding the command line SSH technique, we were able to pick up the essence of the Deployment procedure. Another take away from the EC2 is, it can change and selecting storage size in comparison to our other deployment selection. Heroku platform. Heroku was able to deploy models conditioned on the GitHub regulation of file size and is not a standalone solution for bigger deployments.

Code Artifacts

Complete model is saved as model.py which contains data cleaning, RNN module, training and generating text code snippets. Since the model is trained on GPU for 500 epochs, it is time consuming to training section of the code is commented out after saving the required files for text generation. There are a total 3 pickle files, model weights, indexing of words to numbers, and numbers to back words.

Flask application is saved as appEC2.py which contains an HTML link for fewer details of the model and implementation.

Conclusion

In this study project we were able to deploy a neural network on a self-hosted platform. Along with learning to build a RNN model, we were able to weigh in implementation and cost benefits of 2 different deployment platforms. We tried to deploy on EC2 and Heroku platform; however, Heroku fall short of spacing issues and EC2 was able to deploy the required models for our application. Although there were 11 major steps in deploying the model on EC2, it was flexible to accommodate models from simple spam and ham to text generation.

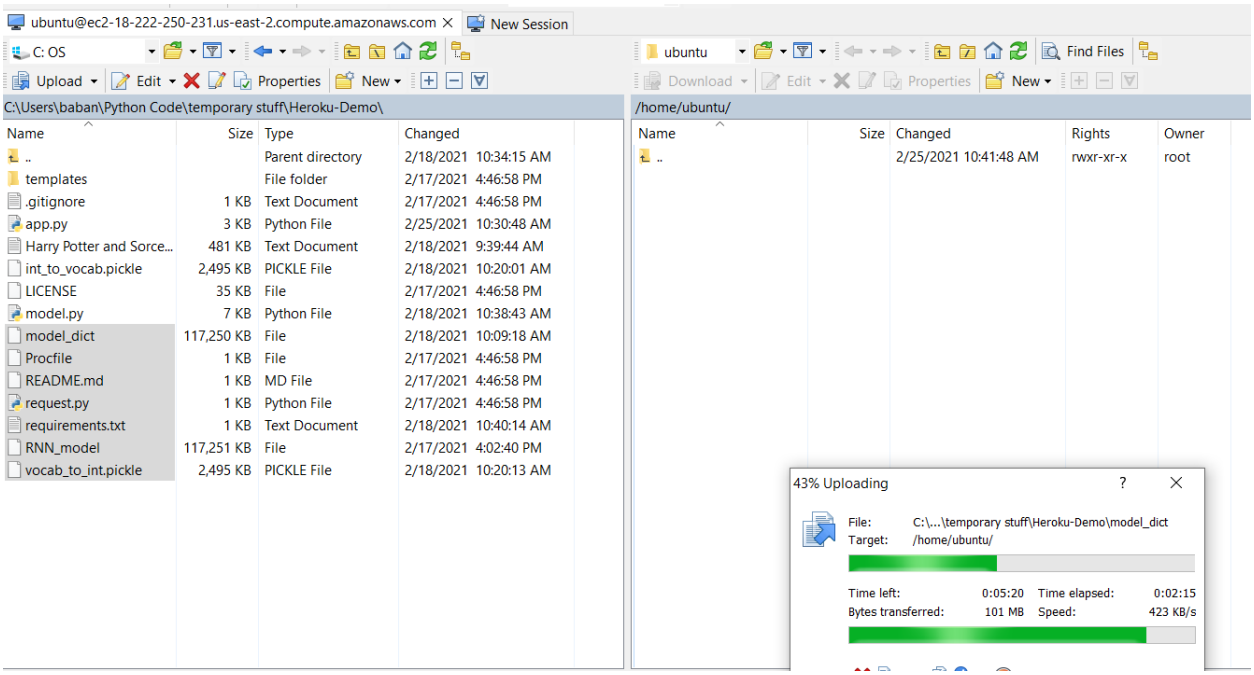
Future Scope of the project

This approach can help in learning the coherent structure of the book, which can further be used to generate the next chapter/ paragraph based on the gravity they trained and expected. This project and web application can also be used to generate textual replies, emails, autocomplete features while writing an article.

Appendix

Appendix 1

WinSCP UI showing the file transfers which are submitted in the zip file.



Appendix 2

Putty tool connection UI used for command line connection and running of the program.