A Study on WaveGlow Text-to-Speech System for Offline Deployability

By

Saishradha Mohanty

Under the guidance of

Mr. Harmandeep Singh Matharu

(Vice President of Engineering at Mihup)

Introduction

Machine Learning has emerged as a buzz word for the past few years, the reason being the high amount of data production by applications, the increase in computation power in the past few years and the development of better algorithms. The ML technology that has bloomed in these recent years is **Speech Recognition Technology**. Speech is the most natural form of communication for us --- it's second nature to us, and we are fascinated by machines that can understand us. Our machines have started to recognize our speech and they're getting better and better at communicating with us. Amazon and Google have developed voice assistants and devices like Amazon Alexa and Google Home that are greatly influencing the way we shop, how we search, how we interact with our devices and even each other. Since all these devices need internet support, in this internship, we studied the necessary factors for the offline deployment of Automatic Speech Recognition model and obstacles in the process.

Acknowledgment

My experience at Mihup Communication Pvt Ltd, Kolkata provided me an opportunity to be a part of a team working in the hot topic of Automatic Speech Recognition. I got a chance to witness as well as learn about the state-of-the-art technologies being used to achieve such a feat. I would like to express my gratitude towards Harmandeep Sir, my mentor, for guiding me throughout the internship. I thank Mr. Pushpak and Ms. Neha Sharma for getting me this opportunity. Finally, I would like to thank my family and my teachers for all their efforts in making me a capable person.

About Mihup

Mihup is an AI and ML-based startup built on Automatic Speech Recognition for both online and offline platforms in Indian languages. It has an AI foundation for scale and reliability in Natural Language Processing(NLP) and Information Retrieval(IR). It offers flexibility to support customer-specified features, content domains. Media and Entertainment, consumer electronics, IoT, retail and e-commerce, automotive, BFSI are among the industry segments can be benefited from Mihup's capabilities.
Mihup, short for '**May I help you please**', started as an offline platform in the year 2014, where one could ask questions (via SMS) and get an instant answer and has made its way into '**The top 50 most promising start-ups in the world**' within a span of just 4 years. Mihup's domain knowledge of Hinglish (a mix of Hindi and English) and deep research on accents and dialects across languages, and currently working with English, Hindi, Bengali, Hinglish, and Benglish (mix of Bengali and English) makes it a potential competitor to tech giants like Google.

Table of contents

List of Figures

Problem Definition

In today's busy life, people prefer technology that will save time and will accomplish the required task with fewer efforts. One way of getting work done with fewer efforts is by going handsfree. To solve a problem that involves another person, the best way to solve it is by having a conversation. Similarly, by communicating with our devices we can get things done. This idea has been exploited by the newly emerging technologies popularly known as **Automatic Speech Recognition(ASR) and Text-To-Speech(TTS)**.

Voice is the future. The world's technology giants are clamoring for vital market share, with both Google and Amazon placing voice-enabled devices at the core of their strategy. Since a huge amount of data is required to achieve this feat, most of these voice-enabled devices are deployed for online platforms. But online platforms leave customers vulnerable to privacy issues, bcoz of which Mihup is focussed on the offline deployment of these models. Also, many times it is not possible to access the net, for which it is important to have devices working in offline mode. Through this work, we:

- Try to build offline deployable TTS systems from already existing models and test their usability.
- Study the important factors observed that need to be taken into account while the offline deployment of these online models.

A brief on TTS

Text to speech, abbreviated as TTS, is a form of speech synthesis that converts text into spoken voice output. Text to speech system was first developed to aid the visually impaired by offering a computer-generated spoken voice that would "read" text to the user.

TTS is composed mainly of two components: **Natural Language Processing(NLP)** and **Digital Signal Processing(DSP).** The figure below illustrates the components of a TTS system.

Natural Language Processing (NLP)

It forms the front-end of the TTS system. This component mainly deals with **text analysis**. Ultimately it outputs phonemes(text) in a form readable by the speech synthesizer (DSP component).

Digital Signal Processing (DSP)

It forms the back-end of the TTS system. This component deals mainly with the **speech synthesis** from the phonemes obtained from the text analyzer. This component is responsible for the generation of appropriate waveforms in accordance with the input taken by the text analyzer. It ultimately outputs the speech form of the text read by the front end.

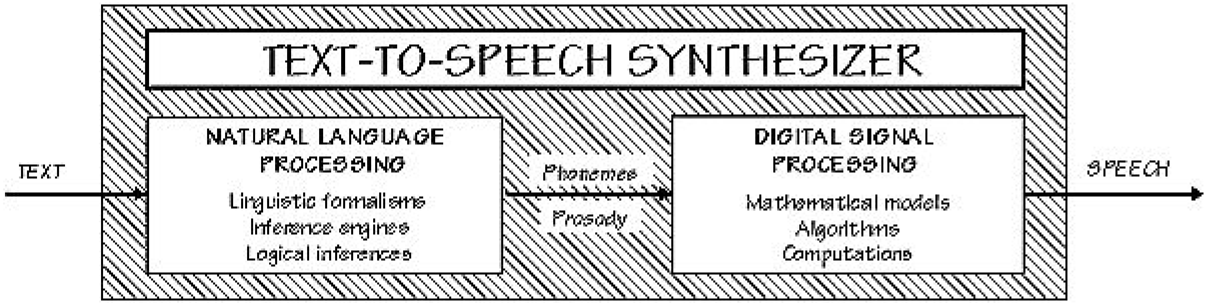

Fig. 1 Text to Speech System

Approach

1. Process Flow

The approach would be to first hand-pick some already existing online TTS models built on the advanced framework. And it is necessary to choose the appropriate frameworks for offline deployment of the chosen model keeping in mind the compatibility of the framework in which the model has been built and the framework chosen for deployment. Since TTS has many different parts, it is necessary to choose which part of the TTS has to be modified for the deployment. Then the required part is rebuilt in accordance with the previous framework and the new framework is trained and tested to check its accuracy in order to check its usability if deployed. Usability can be checked either by the accuracy or by the stability of the corresponding loss obtained while training the model. If the obtained output from the model is not above some threshold (or up to the mark as needed) or the loss value of the model keeps on varying continuously even after many epochs of training, either it is remodeled or discarded.

2. Choosing and Training TTS Model

In this internship, we mainly focus on the speech synthesis part of the TTS that deals with speech waveforms. The main task of the speech synthesizer is to take in the phonemes and generate a mel-spectrogram conditioned on the pronunciation of those phonemes.
The **Mel-Frequency Cepstrum** (or **Mel-spectrogram**) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear Mel scale of frequency. In simple terms, Mel-spectrogram represents the variation of frequency and power of the voice signal produced with time. Here, the **Mel scale** is a graph plotted between pitch value in terms of mels instead of dB vs frequency, depicted in fig. 3.1. It is a perceptual scale of pitches judged by listeners to be equal in distance from one another.
The part of the speech synthesizer that generates waveforms based on these Mel-spectrograms is known as the **vocoder**. The figures below illustrate the Mel scale, Mel-Spectrogram and the i/o of a vocoder clearly.

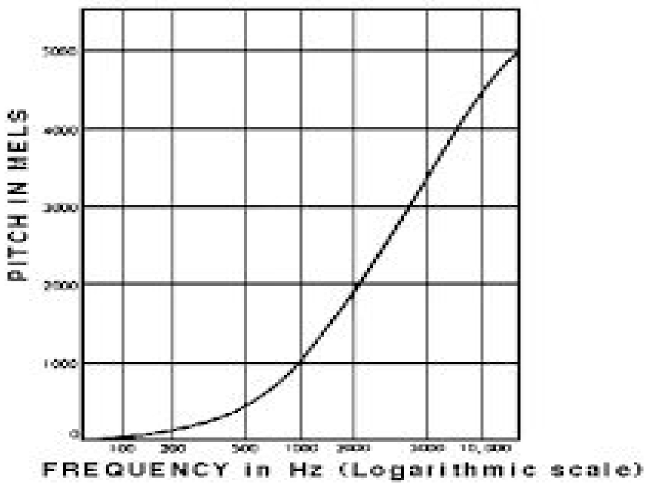

Fig. 2.1 The Mel scale

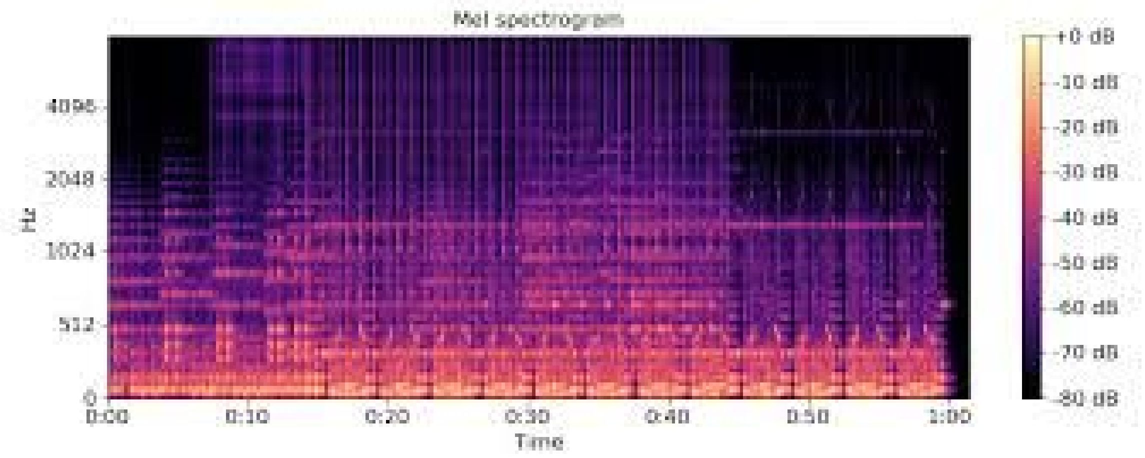

Fig. 2.2 The Mel-Spectrogram

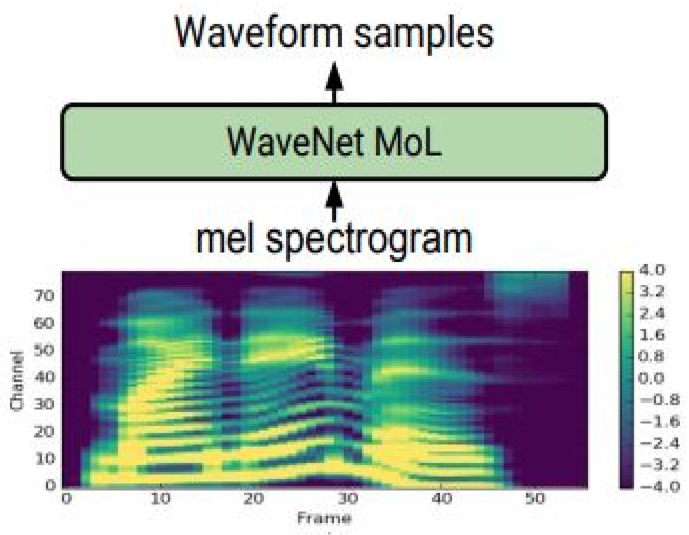

Fig. 2.3 The I/O for a vocoder

Fig. 2.3 represents the input and output of a vocoder. A vocoder takes in Mel-Spectrogram as its input and the generated waveform samples are the speech samples generated in accordance with the information depicted by Mel-Spectrogram. In the above fig. WaveNet MoL is a vocoder, basically, in place of WaveNet MoL, any deep neural network can be placed that can function as a vocoder.

2.1 WaveGlow

WaveGlow is one of the recently developed Speech Synthesis model built by **NVIDIA Corporation.** This model is built in **PyTorch** and has been tested for online platforms for now. This model is a combination of 2 previously existing well known and significantly used speech synthesis models, WaveNet developed by **DeepMind** and **Glow** developed by **OpenAI** from Google**.** It is a **vocoder**. The aim was to remodel this system in accordance with TensorFlow Lite, the framework that was chosen for offline deployment. Shown in fig. 3.1.2 is the architecture of the WaveGlow model.

2.1.1 Why choose WaveGlow?

The main reason for choosing this ASR model was because of its more promising results as compared to the other well-known models - WaveNet and Glow and that it combines the most important concepts of both WaveNet and Glow into the same model.

Secondly, most of the other Speech Synthesis models make use of the Concatenative Approach as opposed to the Deep Learning approach. The concatenative synthesis approach majorly fails to capture the natural variations in speech and the nature of the automated techniques for segmenting the waveforms sometimes results in audible glitches in the output. Hence, there’s a tendency to shift towards building the models using Deep Learning methods which can address the above two limitations of the former method.

Fig. 2.1.1 The WaveGlow symbol

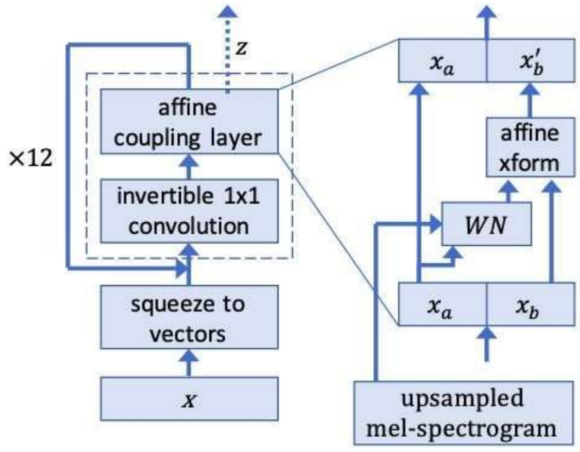

Fig. 2.1.2 The WaveGlow Architecture

2.2 WaveGlow-Tensorflow

2.2.1 Why choose TensorFlow over PyTorch?

PyTorch is not developed (or preferable) for production, therefore to ensure the deployability of the model on any platform, it was necessary to rebuild it using a different framework. For deployment purposes of Machine Learning models, Tensorflow is one of the most widely used frameworks. So, the first choice for a different framework was Tensorflow. Moreover, **for deploying the model offline**, **TensorFlow Lite** was the considered platform, hence, going for Tensorflow was the best choice.

2.2.2 Approach for diving into the TensorFlow version

We first refer to an already existing WaveGlow model in Tensorflow. The reason for choosing an already existing model rather than rebuilding it from scratch is it is more optimized, yields more consistent and accurate results with better utilization of time and resources for the online model. For this project, we chose to go for the latest version of TensorFlow, **TensorFlow Nightly-2.0**.

The corresponding model chosen was initially written in Tensorflow v1.12, many of whose functions were incompatible for Tensorflow Nightly, hence the corresponding incompatible functions of the old version were replaced with corresponding functions in the new version of Nightly 2.0.
This model written in the new version of TensorFlow was then tested on the LJ-Speech dataset, which was also used for training of the original model. Testing was done to check whether this model works in a similar way as the original model for the same dataset while keeping the architecture the same as before. The results obtained after training and testing were exactly the same as the previous model, which gave a green signal for proceeding with the redesigning of the model.

2.2.3 Approach for redesigning for TensorFlow platform

Now, the model had to be redesigned so that it could be made compatible with the TensorFlow Lite platform and could be operated offline.
The foremost factor to be kept in mind while building any offline model is the **size of the model**. More about the important factors for the offline version has been discussed in the later section. The **online TensorFlow model was of the order of 300 GB** which is way more than the size or the applications supported by TensorFlow Lite. **For Lite**, the **application is expected to be of the order of max 50-60 MB** for it to work in a satisfactory manner.
As shown in fig. 3.1.2, the invertible 1x1 convolution architecture and the affine coupling layer are the two main components that play a significant role in deciding the size of the model. Like normal CNNs, since this invertible 1x1 CNN also consists of convolution layers, these layers contribute significantly to the size of the model. **Higher the number of layers, higher the size of the model**. Similarly, the affine coupling layer is also comprised of different layers for coupling, which also adds to the size of the model, the higher the number of layers, the higher the size of the model. However, the affine coupling layer is quite small as compared to the Convolution layer**,** therefore, here, **the number of layers in CNN dominantly decides the size of the model**.
The following figure shows the steps to choose the number of layers for CNN and the coupling layer.

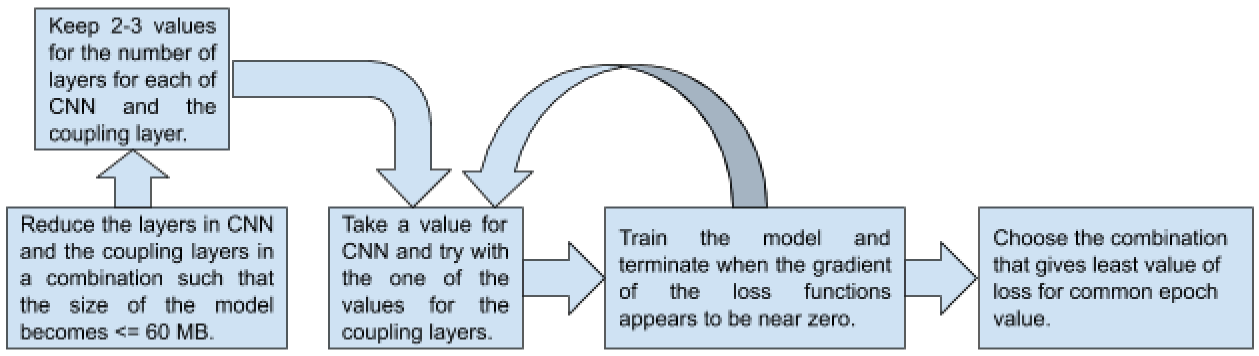
Keep 2-3 values for the number of layers for each of CNN and the coupling layer.
Reduce the layers in CNN and the coupling layers in a combination such that the size of the model becomes <= 60 MB.
Take a value for CNN and try with the one of the values for the coupling layers.
Train the model and terminate when the gradient of the loss functions appears to be near zero.
Choose the combination that gives least value of loss for common epoch value.

Fig. 2.2.3.1 Steps for choosing the number of CNN layers and coupling layers

After deciding the number of layers, the model was put to training over LJ-Speech datasets, keeping the number of iterations at 5,000,000. It took around 2-3 days for the model to be trained.

2.2.4 Problems with this model

The problem with this model was that even after training over 5,00,000 epochs, the loss function had failed to stabilize around a particular value. To check whether the loss could be reduced any further, the learning rate was decreased by almost a 10th of the original value. In that case, also, after a few epochs, the loss was varying in a range of almost 0.4 to 0.5 around the loss value of 2.3, which is very huge after so many epochs of iterations. The model was further trained for 2,00,000 more iterations, still, the loss value was varying by the same amount. Researching further, it was concluded that the **WaveGlow Tensorflow model doesn't converge for the GPU version.** Hence, there was a need to shift to a different architecture.

2.3 WaveGlow-Chainer

2.3.1 Why choose Chainer?

One of the main reasons to choose Chainer after Tensorflow over any other framework is because of its excellent Better GPU & GPU data center performance. Recently, Chainer became the world champion for GPU data center performance, beating even Tensorflow. **It can be run on multiple GPUs with very little effort.**
Also, Chainer is leveraged mainly for speech recognition, machine translation, and sentiment analysis. It supports various network architectures, like CNNs, fast-forward, nets, and RNNs, and has some significant advantages over its competitors:

1. It's much faster than other leading Python frameworks.
2. It's super flexible and intuitive.
3. Existing networks can be modified at runtime, unlike with Tensorflow, where the framework has to be defined and fixed before training itself.

2.3.2 Approach for redesigning the network

The approach followed is exactly the same as that followed while redesigning for the TensorFlow version, just that now it is being designed for the Chainer version.

2.3.3 Problems with this model

One problem with using Chainer is that even for lightweight networks, the training takes a very significant amount of time. Where **TensorFlow takes just 3- 4 days for training this model, Chainer takes almost 3 weeks to train the same model.**

3. Major bottlenecks encountered while building a model for low-level platforms

1. **Size of the model** - For offline platforms like the likes of Tensor Lite, the application to be run must be if the size of MBs, max 80-90 MBs is preferable, irrespective of the complexity of the application.

2. **A tradeoff between the number of layers and accuracy** - In cases similar to those discussed in section 2.2.3, while decreasing the number of layers, there might be a very high possibility of a negative impact accuracy of the model.
Before even checking for the accuracy of the model, first, it is important to check for the stability of the loss value around a particular value. If the loss value isn't stable, the first focus should be on stabilizing it, if that doesn't get stabilized in any way, it is preferable to do changes to the architecture or to discard the model altogether.
If the loss value is behaving properly, i.e is stabilized already, then we should go for accuracy, the number of layers or changes to the architecture must be done in a way such that accuracy doesn't go very low.
3. **Accuracy vs Clarity of Speech heard by the user - Accuracy, on the whole, doesn't exactly decide the usability of the model**. Because it is the quality of the output that ultimately decides the usability. Some models might have very accuracy, yet they won't be having an output audio quality that can be understood by the user comfortably. Whereas, there might be models having moderate accuracy values, yet would have the output audio quality that might be comfortably understood by the user. Hence, it can be concluded that **though accuracy is an important factor in deciding whether the model can be released for user experience or not, the quality of the audio output plays a more important role in deciding the usability of the model**.

Results and Discussions

1. WaveGlow-Tensorflow
The input vs The output

This link contains some audio input files and the produced output files after training the model for 40,000, 3,50,000, 5,00,000 and, 7,00,000 iterations.

1.1. It can be observed that the output obtained by testing the model obtained from 40,000 iterations is full of noise, this is due to very high loss value at that point in time.

1.2. The output obtained by testing the model obtained from 1,00,000 iterations is somewhat better than the previous output, as a few words from the original audio input can be heard to be present in this output, still, the output has a large amount of noise.

1.3. Moving ahead, it can be seen that the output obtained by testing the model obtained from 5,00,000 is exactly the same as that obtained from the previous model. This is because the loss value hasn't converged yet.

1.4 In order to check whether the loss value converges after few more iterations, we train the model further to 7,00,000 iterations, and it is very clear from that the output obtained by testing the model obtained from 7,00,000 is exactly the same as that obtained from the previous model. This is because the loss value still hasn't converged yet.

- ★ In general, if consider a neural network trained well, the loss value starts converging well after a few epochs, like, the model's loss value should have almost converged by the 5,00,000th epoch.
- ★ This unlikely observation of non-convergence of the loss value after such a significant amount of iterations gave rise to the doubt on the credibility of the chosen framework itself. And after some research, it was found out that the **WaveGlow-Tensorflow model doesn't converge for the NCWH (Number of batches, Number of input channels, Weight of the mel-spectrogram matrix, Height of the mel-spectrogram matrix) version, which is supported by GPU.**

2. WaveGlow-Chainer

Following are some graphs obtained after training the new model for 500000 iterations:

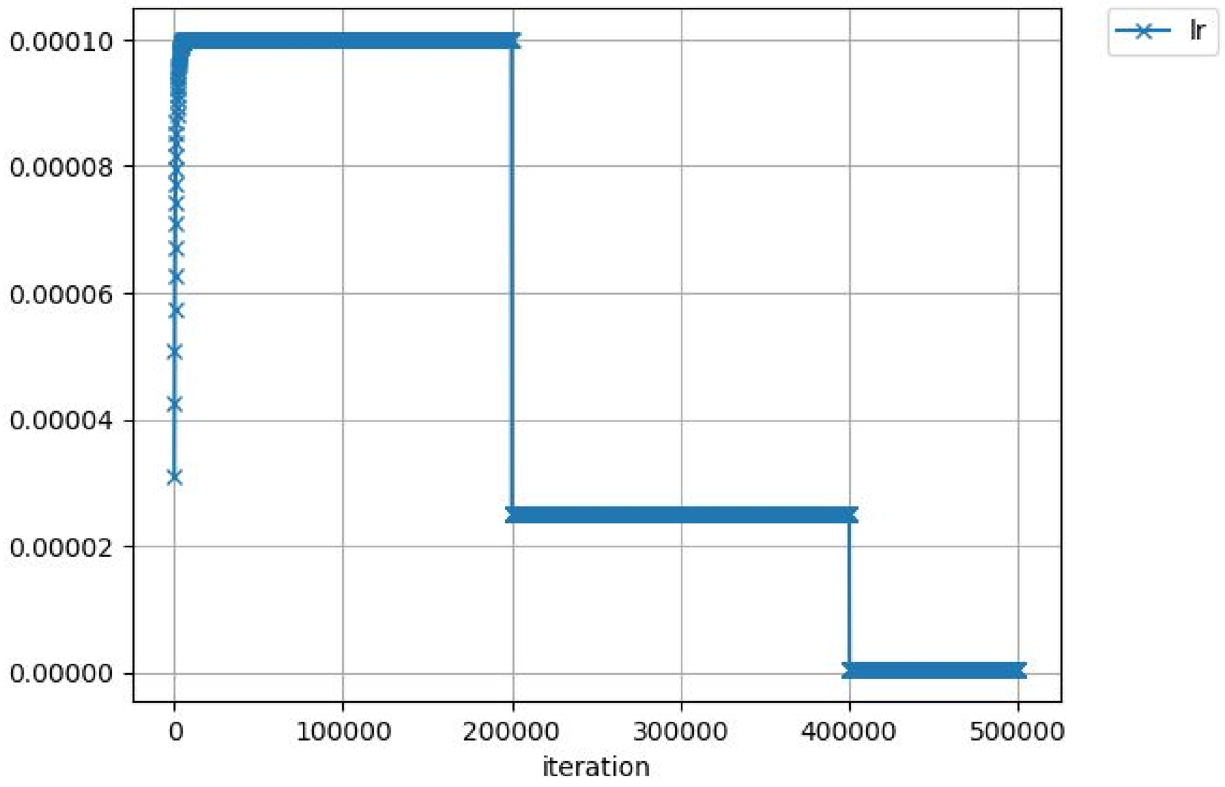

Fig. 3.1 The learning rate of the model (represented by lr)

Fig. 3.1 shows the learning rate of the model plotted against the number of iterations of training. As clear from the fig., after every 2,00,000 iterations, the learning rate is annealed i.e reduced further in a hope that the loss value will further go down if it has saturated for the previous learning rate value. At 2,00,000th epoch, the lr value is reduced by almost 5 times, whereas at the 4,00,000th epoch, the lr value is reduced by only 2 times, by less factor that the previous time. The reason for that will be clear from the next graph.

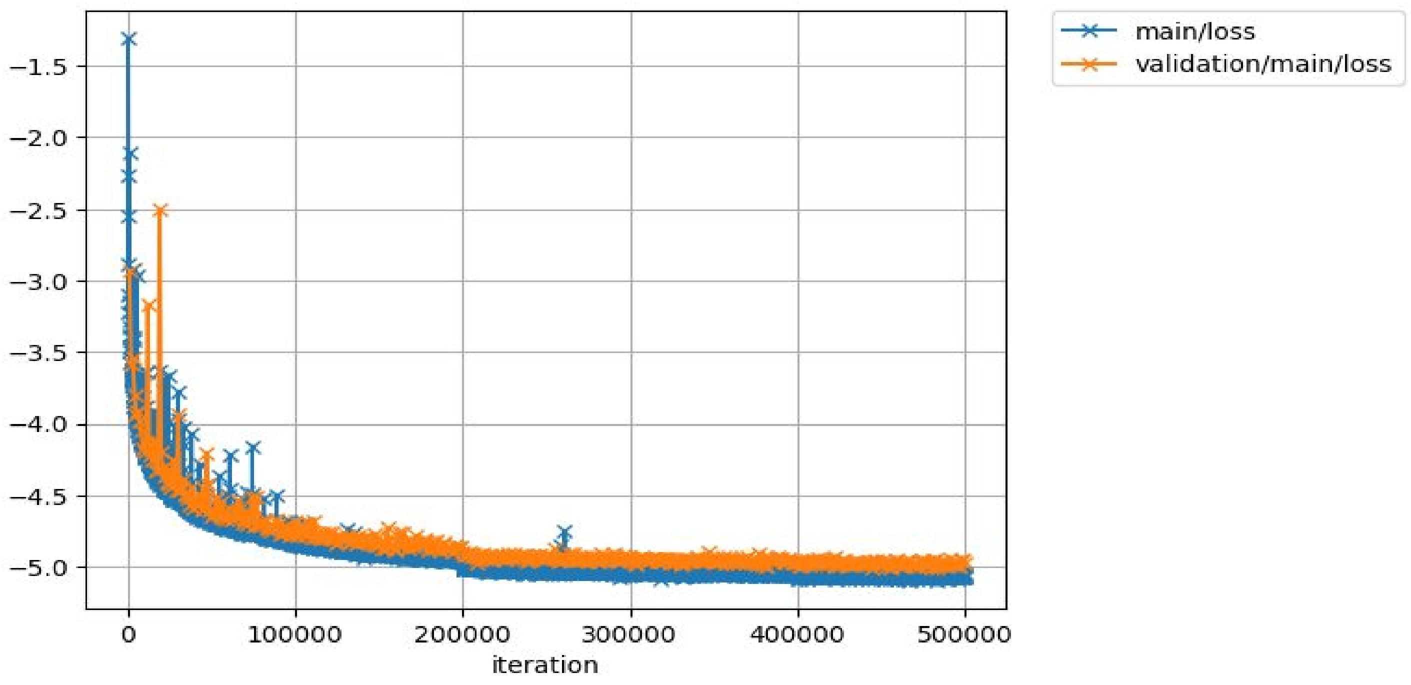

Fig. 3.2 The training and validation loss

Fig. 3.2 clearly shows that well before touching the 2,00,000th iteration, the loss value has become constant. Since there was a possibility that the learning rate had proved to be high for the area near the local minima, the decision to reduce the learning rate was made. Though this would reduce the step size to reach the local minima of the and result in an increment of the training time for obtaining the model close to the required one, annealing did prove to be a beneficial step. After annealing, the loss is seen to drop by some value approving the fact that now a smaller value of the learning rate would help in leading to the local minima of the loss function.
After very few iterations of training after the 2,00,000th iteration, the loss value became constant again. The prime suspect was again the comparatively large size of the learning rate as compared to the area surrounding the local minima. So, the learning rate was reduced again, but this time, only by twice (lesser than the previous time). It is because, already the value of lr is very small, and as it is clearly visible from the image that there is a very fewer change in the loss value after first annealing. If the learning rate is reduced again by good amount, the step size will become so small that the gradient dip will almost become negligible and a very high amount of time and resources will be used unnecessarily for the complete training of the model.
After the second annealing, the dip in the loss value was very negligible and the loss remained constant throughout all the iterations from 4,00,000 iterations to 5,00,000 iterations. Hence, we decided to test the model (assuming no further dip in the loss value possible).

The input vs The output

The model was now tested to check the quality of the output produced.

1. **Case I**:

Input: Any audio file.
Output: The same input audio file but the speech in the file is reconstructed artificially by the neural network.
Results: This link contains some audio input files and the produced output files after training the model for 39,000, 40,000, 1,25,000, 3,50,000, 5,00,000 iterations.
Discussions:

1.1. It can be seen that the audio obtained after 39,000 iterations is very blur. The reason can be stated as high loss value as seen in the loss graph corresponding to this iterations value.

1.2. The output obtained after 40,000 iterations seems to have improved by a noticeable amount. The reason being a noticeable loss drop from 39,000 to 40,000 iterations.

1.3. As we move on to the output obtained from 1,25,000th iteration, the output appears to be much clearer than the previous two outputs. The reason being the same, a significant loss drop from 40,000 to 1,25,000 iterations.

1.4. On the 1,25,000th iteration, the output appears to be much clearer than the previous outputs, because of a significant loss drop from 1,25,000 to 3,25,000 iterations.

1.5. The output obtained on the 5,00,000th iteration is significantly clearer than the first three outputs, because of the significant difference in the loss value. But the output appears to be similar to the output obtained from the 3,25,000 iteration. This is due to the very negligible loss drop during that period.

Though the output obtained from the last iteration is quite better than the previous ones, still the output quality doesn't seem to be satisfying for the user, as more audio clarity is expected even if the audio is not exactly the same as the input audio. **Hence, the possible reasons for this low audio quality even after training for a good amount of time can be thought as**:

1. The chosen values for the number of layers maybe be improper, or a different combination of those layers might give improved results.
2. Due to a very small size of the model, it might need more iterations of training as after 4,00,000 iterations the loss gradient becomes almost zero, decreasing in an extremely slow manner.

★ The training of the model up to 5,00,000 iterations took around 3 weeks when trained using 2 NVIDIA GPUs parallelly. This shows that the training process has to be done very precisely in order to obtain satisfactory results and takes a very significant amount of time.

★ Due to constraints on time, the model could not be improved further by making a new set of experimentations.

2. **Case II**:

Input: Any mel-spectrogram file.

★ The purpose of enabling the model to take in mels as input was to be able to directly plug in this vocoder with an already existing text analysis system to form the complete TTS system.

Output: The audio file corresponding to the input features reconstructed artificially by the neural network.

Results: This link contains some Mel spectrogram files as input files and the produced files produced after training the model for 39,000, 40,000, 1,25,000, 5,00,000 iterations.

Discussions: Previously, the model used to take audio files as input and has been designed such that it takes in the input audio files, converts them into mel-spectrogram and then undergoes training.

Now, we decided to test the model by giving directly Mel spectrogram files as the input. So, only the code for taking in the input was changed such that it would now take in mel-spectrogram as the input.

After testing the previously trained models obtained after 39,000, 40,000, 1,25,000, 3,50,000, 5,00,000 iterations with a difference that now, they take mel-spectrogram files as input, it is seen that no clear output is obtained. The whole audio sounds like noise as opposed to what was seen in the previous case where the output obtained was clear to some extent. From this, **it was concluded that the model will produce outputs in the expected direction only for the kind of inputs over which the training has been done.**

The model was trained over audio files as input, even by using the trained model with a changed the code for input for the trained model for testing, it was the same as doing testing with an untrained model, that's why the obtained output files contained only noise.

Learning and Outcomes

This internship mainly provided me with experience and insight into the currently hot topic of Automatic Speech Recognition and Speech Synthesis. Also, I got a chance to witness the development and deployment of Machine Learning Models in practical scenarios and bottlenecks faced while training, testing and deploying the model for user experience.

Hence, the outcome appears like that in small ML projects where Training takes 90% of the time and is the slower process out of training and testing and once trained properly, testing can be done in no time, whereas while deploying the models, other than just training, testing can also take up a significant amount of time, since the output has to be such that it receives satisfactory feedback from the user, similarly, the issues faced during deployment also take-up an equivalent amount of time as the former two processes.

Special Acknowledgement

I would also like to add that the concepts learned during the **Machine Learning Course taught by Dr. Y. Kalidas at IIT Tirupati** proved to be beneficial here as they saved my time from digging into the very basics of ML and hence, allowed me to progress further with my project.

References

[1] WaveGlow: A Flow-based Generative Network for Speech Synthesis by Ryan Prenger, Rafael Valle, Bryan Catanzaro.
[2] Glow: Generative Flow with Invertible 1x1 Convolutions by Diederik P. Kingma, Prafulla Dhariwal.
[3] WaveNet: A Generative Model for Raw Audio by Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu.
[4] Towards Understanding the Invertibility of Convolutional Neural Networks by Anna C. Gilbert, Yi Zhang, Kibok Lee, Yuting Zhang, Honglak Lee.
[5] Introduction to Chainer: A Flexible Framework for Deep Learning, a seminar by Seiya Tokui.
[6] Glow: Generative Flow with Invertible 1x1 Convolutions lecture by D. Kingma and P.Dhariwal, OpenAI.