Inter-University Accelerator Center

Analysis and Control of Tandem Accelerator Parameters for Stable Pulsed Beam Operation using Deep Learning

Shivi Gupta

Department of Electrical Engineering,
Indian Institute of Technology, Kanpur

September 3, 2022

Abstract

Particle accelerators being very complex systems, require tuning a large number of parameters, specific to specific operations. In the pulsed beam operation of a tandem accelerator, some of these parameters are needed to be controlled, i.e. locked at a certain value. Presently, this is done manually by tuning the other parameters by hand. Automating this process can have advantages in terms of more reliability and less time consumption. Further, even if the operation is kept manual, estimation of most significant parameters will also prove useful.

Reinforcement learning (RL) is a machine learning solution for control problems, used popularly in physical and dynamical systems. Leveraging ideas inspired from RL algorithms such as Imitation Learning, Behavioral Cloning and Teacher Forcing, in this work, I present a universal model to control one of the beam parameters of tandem accelerator for the stable pulsed beam operation. Furthermore, I infer from this model, the most significant parameters for this application. The model achieves mean absolute errors as low as 0.4% in predicting parameters for phase control. Not much work has been done in the fields of Machine Learning for particle accelerators, therefore this work can be a good starting point for more robust deep learning models which can be deployed practically in particle accelerators.

1 Bibliography

First of all, I want to thank Prof. Avinash Pandey, director, IUAC for introducing me to this beautiful facility and giving me the opportunity of pursuing the research internship. I would also like to thank my project supervisor, Dr. Bhuban K Sahu for giving me such a novel and exciting venture to carry out. Next, I want to thank the other people of the group - Abhishek sir for helping me in every step of the process and explaining all the mechanisms and Ashish sir for his guidance.

I am the most grateful to my brother, Shubh Gupta, being a researcher himself for giving me good inputs and ideas at times when I felt extremely lost. Finally, I am also ever grateful to my constantly supporting family and friends who are always there for me. I hope my efforts in this novel pursuit have been of assistance to the accelerator community, and can be a good starting point for more research in Machine Learning for Particle Accelerators.

Index

Bibliography, 2

Introduction, 4-5
- Reinforcement Learning
- Imitation Learning
- Behavioral Cloning

Dataset, 6

Method, 7-9
- Processing the Dataset
- Finding Most Significant Parameters
- Predicting Next Step Actions

Results, 10
- Most Significant Parameters
- Predictions of Independent Parameters

Conclusions and Future Work, 11

Appendix, 11

References, 12

2 Introduction

The control of charged particle accelerators is bit complex as it involves tuning of a number of electro- magnetic elements to deliver the accelerated beam on target. The pulsed mode operation of the accelerator is much more complex as the arrival of the beam needs to be time synchronized with the master clock/trigger and needs to be monitored. A 15UD Pelletron accelerator at IUAC, New Delhi is being operational for more than 3 decades to deliver dc as well as pulsed heavy ion beams to various university users for conducting experiments in nuclear physics, material science etc. During the project period the operational data was being collected for pulsed beam operation to understand the factors affecting the time synchronization of the accelerated beam particles by monitoring the arrival phase of the beam using phase meter which is locked at a particular phase with respect to master clock based on the initial tuning parameters of the beam.

The automation of this process will not only make it easier for the accelerator functioning, but also more robust and less prone to errors and will furthermore reduce the time taken to set up the parameters. Completely automating the process will require two steps: first, tuning the beam to a certain phase and second, keeping it constant. The methods presented here are for the second step, since tuning the beam from scratch would be a more complex problem. Though it should be possible to extend similar ideas for the first step as well.

The problem statement narrows down to the following. We wish to keep the phase of the AC voltage constant (at a certain value) by tuning all the other parameters of the accelerator. One way of automating this control process is to use Deep Learning algorithms, since they create non linear models to process large amounts of data to make predictions. Further reinforcement learning algorithms, a category of deep learning are algorithms which predict the next set of actions to be carried out given the current 'state' of the 'environment'.

2.1 Reinforcement Learning

In reinforcement learning (RL), an Agent interacts with an environment and in turn receives feedback (rewards or punishment). Through this process, the agent is able to learn from the environment and produce actions in such a way that its expected reward is maximised. At every timestep, t, the agent observes the state of the environment (S_t) and the reward received (R_t) to produce the next action, A_t. The action taken leads to a new state, S_{t+1} and reward R_{t+1} which are fed back into the agent like shown in figure 1.

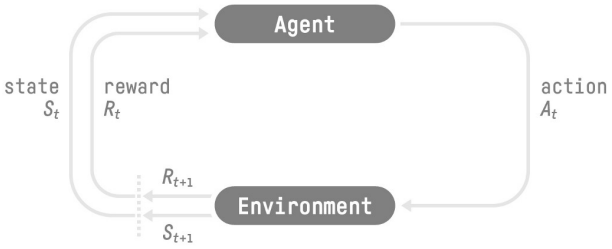

Figure 1: An illustration of the RL process (reference)

RL algorithms are algorithms which explore the different ways of designing reward functions and agents' policy. Policy (π_θ) refers to a function which gives actions for a given state to be taken by the agent, where θ are the parameters learned by the function. There are a vast multitude of algorithms used in different applications, such as Deep Q Learning (DQN), Deep Deterministic Policy Gradient (DDPG) for the continuous action space, Actor and Critic methods and so on.

Most of these algorithms require interacting and training on the environment. This can be carried out in the particle accelerator context if a reliable simulation is present. Since in my work, it was not, I have used an RL algorithm which trains on data already recorded by an expert supervisor.

2.2 Imitation Learning

A class of RL algorithms, Imitation Learning uses 'expert data' or demonstrations to train so that it predicts actions similar to what an expert would do. Imitation Learning includes algorithms such as Behavioral Cloning and Reverse RL. It is helpful in the cases where it is difficult to specify a reward function or when the rewards are very sparse.

The components of imitation learning are the following:

1. an environment which is a Markov Decision Process with a probability distribution $P(s'|s,a)$ and a reward function $R(s,a)$
2. Expert demonstrations or trajectories $\tau^* = (S_0^*, A_0^*), (S_1^*, A_1^*)...$ (The expert is said to be working on the optimal policy, π^*.
3. Agent's policy π, which is an estimate learned to be as close as possible to π^*

Different imitation learning algorithms have different learning algorithms, models and loss function. The algorithm used in this work is the behavioral cloning method.

2.3 Behavioral Cloning

In Behavioral Cloning, we require the demonstration data from an expert supervisor, from which the agent learns the action to be taken given a state. It makes use of supervised learning. The algorithm is (reference):

1. Collection of expert demonstrations (τ^* trajectories)
2. Treat the demonstrations as i.i.d. state-action pairs: (S_0^*, A_0^*), (S_1^*, A_1^*) ...
3. Learn π_θ policy using supervised learning by minimizing the loss function $\mathrm{L}(A^*, \pi_\theta(S))$

Extending this algorithm to our scope, Expert demonstrations comprise of the data collected over the past runs of the accelerator, hand tuned by a human expert. To make the state action pairs (S_i^*, A_i^*), we parse the data and select all two consecutive entries such that the phase becomes closer to the locked value in the second entry. In such cases, the first entry corresponds to the state S_i^* and the second entry, which represents what the human expert tuned the parameters to, becomes the action A_i^*. For the policy, we train a deep neural network on the loss function L as the mean absolute error.

A more detailed explanation is given in the following sections.

3 Dataset

The data collected and used in this work can be found here.

The dataset collected contains logs of beam operation in the pulsed beam mode, controlled by an expert supervisor. Each row corresponds to a different timestep, one after the other in an ascending order. There are four files, two each for a different beam type (^{19}F and ^{28}Si). Each file corresponds to different charge states and different MeV energies. The column titled 'Phase Meter-Phase' is the phase parameter that is to be locked. The value that the parameter is locked to (ϕ_o), in each file is taken to be the circular mean, i.e. there is a seperate locked value for each file. Circular mean takes into account the circular nature of the parameter, in the range $[0, 360]$. Some other information including the locked phase from the data is given in the table below.

File Name	Beam Type	Charge State(s)	Locked Phase
28Si_23112021_1	^{28}Si	+6, +11	9.0977
28Si_23112021_2	^{28}Si	+6, +11	7.3030
19F_15112021	^{19}F	+6, +9	2.0791
19F_15112021_2	^{19}F	+6, +9	3.8302

Table 1: Details of the data used in the analysis

Next, phase difference for every corresponding data entry is calculated, which is the difference between the phase in that timestep to the locked value ($\phi - \phi_o$). All the columns are normalised to the range $[1, 10]$ (minmax normalization).

$$X_{\text{normalized}} = 10 \cdot \left(\frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \right)$$

where X_{min} and X_{max} are across all the four sets, for particular columns. For the rest of the report, we will assume the number of parameters (except the phase and phase difference) to be n.

4 Method

In the following analysis, we have carried out two tasks-

1. Finding the most significant parameters that affect changes in phase
2. Predicting what the parameters should be changed to, in the next time step so that phase attains a value closer to the locked phase

For the above problems, we design a prediction model which takes as inputs the current timestep's parameters as well as the phase difference (input dimension $= n + 1$), and predicts the parameters for the next timestep (output dimension $= n$). In the behavioral cloning terms, the input (labelled X) is the state, S and the output (labelled y') is the action, A. The function f learned by the model is, thus the policy π which is an estimate for the expert's policy π^*.

Because of the way the dataset is processed, the model works to minimize the phase difference between the locked value of the phase (ϕ_o) and the current observed value (ϕ). An overview of this is shown in figure 2.

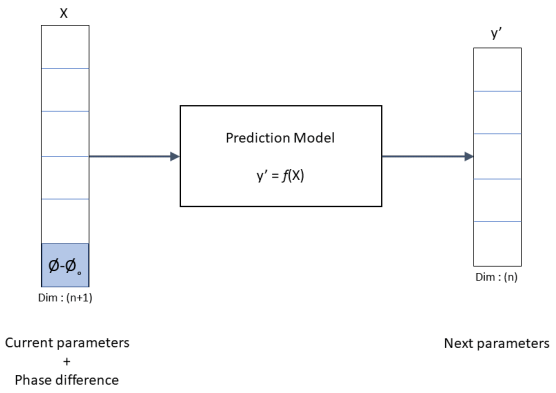

Figure 2: A schematic of the model workflow

In the following, we analyse two different prediction models for the two different tasks mentioned before: a fully connected neural network to find the most significant parameters and an LSTM network for predictions. For both of these, a train and a test dataset is created.

4.1 Processing the dataset

According to the Behavioral Cloning method, we require an expert data, which the model will learn from. In the dataset as mentioned previously, a human supervisor toggles the different n parameters to lock the phase closest to ϕ_o. To create the training data for the prediction model, we require inputs (X) and ground truth labels (y). Here, these (X_i, y_i) pairs correspond to the state-action pairs (S_i^*, A_i^*) which are the expert demonstrations in the algorithm.

We parse the dataset to find all such pairs of consecutive timesteps, such that the second timestep has a smaller absolute value of phase difference than the first one. The first row becomes X_i and the second row without the phase difference becomes y_i. Figure 3 gives an illustration of this process.

	Parameter #1	Parameter #2	...	Parameter #n	Ø-Ø₀
Appended to X					3.2
Appended to y'					2.1

Figure 3: Illustrating the data processing

After creating these input and output datasets, it is then shuffled and divided into train and test datasets in the ratio of 0.33.

4.2 Finding Most Significant Parameters

To find the most significant parameters, we take the prediction model to be a shallow, fully connected neural network with one hidden layer and one output layer. The hidden layer has the dimension $n+1$, with a *ReLu* activation and *he_normal* kernel initializer. The output layer has the dimension n. Optimizer used is Adam with *mae* loss function. This is trained for 300 epochs with a batch size of 16. The model is shown in figure 4.

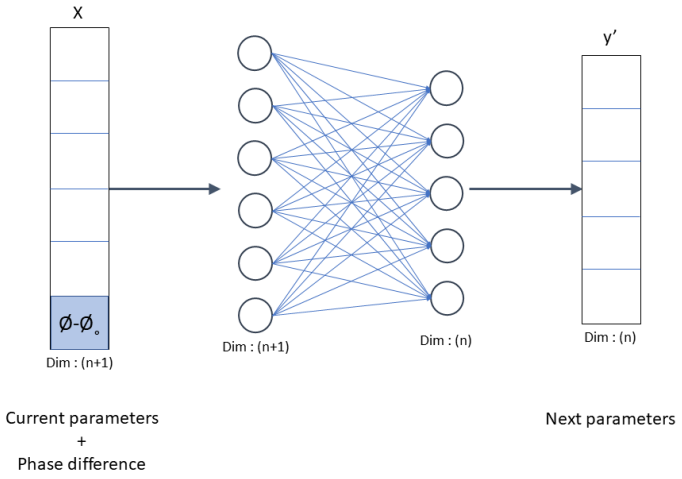

Figure 4: Neural Network model

After training the neural network on the training dataset, we estimate the significance of each parameter by calculating the importance score using the weights learned by the hidden layer. The importance score for each parameter is simply the absolute sum of the weights assigned to that parameter by each neuron.

$$I^i = \sum_{j=0}^{m} |W_j^i|$$

where I^i is the importance score for the ith parameter, m is the number of neurons in the hidden layer and W_j^i is the weight assigned by jth neuron to that parameter. This importance score is comparable for each parameter because they have all been normalized to [0,10]. A high importance score would mean that the parameter is highly significant. The results are given in the section "Results" below.

4.3 Predicting Next Step Actions

Since a simple fully connected neural network is not robust for long term predictions, timeseries models such as LSTMs and RNNs can be more stable. These models are popularly used in sequential data for predictions. One example for the applications of these is text prediction, requiring some form of long term stability to make meaningful sentences. Since this work also requires predictions to be made for next timesteps, as well as stability, using an LSTM (Long and Short Term Memory) model as the prediction model will be useful.

Input to an LSTM layer must be multiple timesteps, that is a window of data. So to create X and y datasets, as done previously, we first find all the consecutive rows which show a decrease in absolute phase difference. Following that, in X, we append not just the single row, but w rows in which $w-1$ are the previous timesteps. Here, w is the window size.

The model consists of a hidden LSTM layer of dimension $n+1$ with *ReLu* activation and *He_noral* initializer and a dense output layer of dimension n. Optimizer used is Adam with *mae* loss function. This is trained for 300 epochs with a batch size of 16. Schematic of this is given in figure 5.

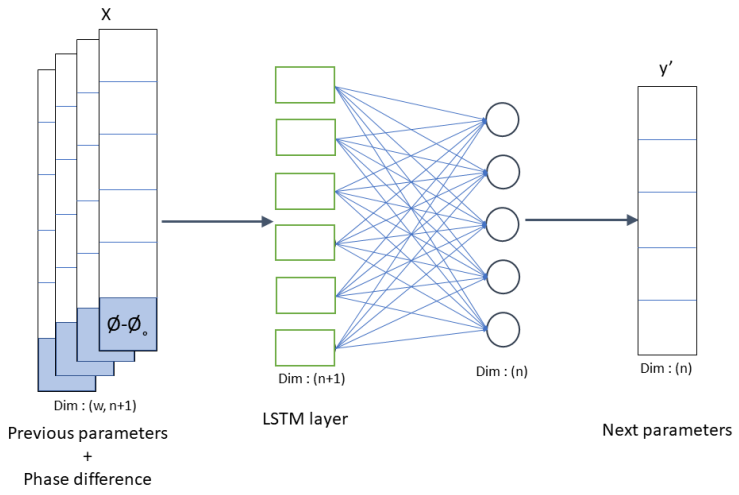

Figure 5: LSTM model

Here, the window size, w is a hyperparameter and tester for four different values. The results are shown in the next section.

5 Results

5.1 Most Significant Parameters

The fully connected neural network model trained to give a mean absolute error (mean and standard deviation is reported) of mae= 0.0498 ± 0.0046 over 10 independent runs. This amounts to a percentage error of about 0.5%.

Using the hidden layer of the neural network model mentioned above, importance scores were calculated for all n parameters. The histograms for this is given in figure 6. The highest importance score was achieved by the parameter "ES_011(Y-VRN)" with $I = 8.5155$. Further, some parameters show a distinctly poor importance score, meaning very little significance with phase locking.

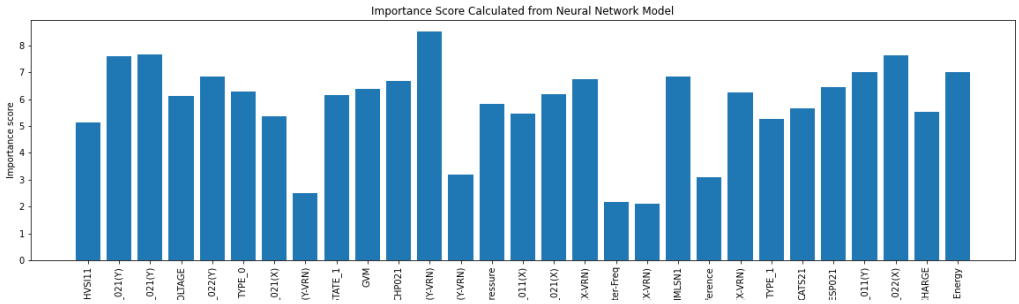

Figure 6: Histogram for the importance scores of each parameter

This data can give insights into the relationships between these parameters and the phase. It can further aid in manual tuning.

5.2 Predictions of Independent Parameters

The LSTM model is analysed for 4 different window sizes: 5, 10, 15 and 20. For each case, the model is trained for 5 independent runs and the means and standard deviations are calculated. Figure 7 shows these with the error bars corresponding to standard deviations. The baseline model corresponds to the neural network model used in the previous analysis.

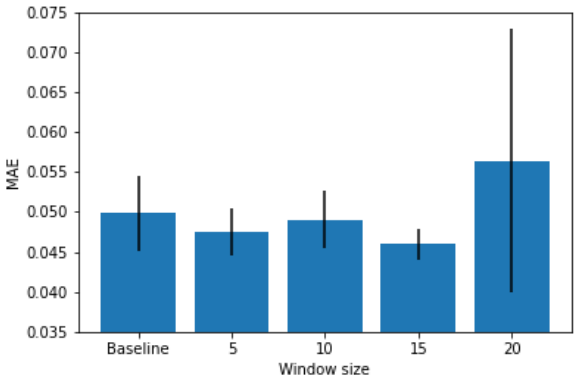

Figure 7: Mean MAE for varying window sizes

The lowest error achieved corresponds to a window size of $w = 15$, with an error of MAE= 0.046 ± 0.002. There is a slight decrease over the baseline. This would still be a more robust model for long term predictions because of the nature of LSTM.

6 Conclusions and Future Work

Through this analysis, we have created models to find significant parameters as well as a prediction method to be used in the control of the phase parameter of a tandem accelerator in the stable pulsed beam operation. The models trained on the data showed extremely good accuracy (loss of about 0.4%) and is an indicator that black box neural networks have a possibility to work well in these applications. Further, this accuracy was achieved only on the basis of pre collected data without any dynamic environment. With the presence of an environment, stronger algorithms can be devised with more robust predictions. While the prediction model may not be ready to be directly deployed for applications, it is a good starting point for the creation of more complex models, indicated by the good accuracy achieved even by a seemingly simple model.

Further, the significance of parameters found can be interpreted to correlate with current understanding of the relationships between the parameters. This can assist the current researchers and experts working currently on the manual tuning of the machinery.

7 Appendix

The GitHub link containing the data used, models and the codes can be found here.

References

[1] Auralee Edelen et al. *Opportunities in Machine Learning for Particle Accelerators*. 2018. DOI: 10.48550/ARXIV.1811.03172. URL: https://arxiv.org/abs/1811.03172.

[2] Jason John et al. "Real-time artificial intelligence for accelerator control: A study at the Fermilab Booster". In: *Physical Review Accelerators and Beams* 24 (Oct. 2021). DOI: 10.1103/PhysRevAccelBeams.24.104601.

[3] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. DOI: 10.48550/ARXIV.1509.02971. URL: https://arxiv.org/abs/1509.02971.

[4] Xiaoying Pang, Sunil Thulasidasan, and Larry Rybarcyk. *Autonomous Control of a Particle Accelerator using Deep Reinforcement Learning*. 2020. DOI: 10.48550/ARXIV.2010.08141. URL: https://arxiv.org/abs/2010.08141.