CS230

Dazzler Laser Shaper Parameter Prediction

Jack Hirschman
Department of Applied Physics
Stanford University
jhirschm@stanford.edu

1 Problem Description

The next generation of SLAC's Linac Coherent Light Source, the LCLS-II, is promising exciting new discoveries in never-before-reached spatio-temporal resolutions for physical and biological systems. Active beam manipulation is required to fully exploit the flexibility of this new generation of XFEL. A possible solution for this is a hardware-based machine learning (ML) implementation of real-time photoinjector laser manipulation that will not only enable active experimental control of x-ray pulse characteristics but could also increase the operational capacity of future accelerator facilities.

Spatio-temporal shaping of the photoinjector laser is the preeminent strategy for sufficiently tailoring the electron bunch phase-space. This is quite challenging as the required ultraviolet wavelengths, being produced via up-conversion of infrared lasers, makes this laser non-trivial to shape.

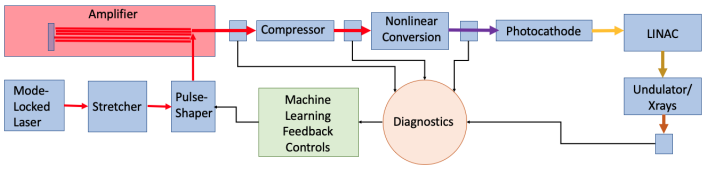


Figure 1: Block diagram of preliminary ML-based feedback system design for photoinjector shaping.

An overview of this system is shown in figure 1. Here the laser is generated from a mode-locked laser which goes through a stretcher and then through a pulse-shaper (which is controlled by the ML block in green). The output of the pulse-shaper goes through an amplifier followed by diagnostics equipment, then a compressor, then diagnostics equipment, then nonlinear conversion, then diagnostics, before hitting the photocathode. At the photocathode a distribution of electrons based on the shape of the laser pulse (in purple at this step) is accelerated via the linear accelerator (LINAC) followed by the undulators in the XFEL where the output xrays go through diagnostics.

This full design is well beyond the scope of the class (in fact, this is my PhD project). However, the design is broken down into several much smaller tasks. The first of which involves inputting particular laser shapes into the amplifier via the pulse-shaper. For this, we want to be able to feed particular shapes given a known shape from the mode-locked laser. **Therefore, we need an ML algorithm to predict the correct parameters for the pulse-shaper to arrive at particular output shapes from the pulse-shaper that would feed into the amplifier.**

Our pulse-shaper is the Fastlite Dazzler [2], which allows for spectrum and phase modulation of an input laser pulse. Since the target application for this inference network is eventually hardware, specifically field programmable gate arrays (FPGAs), there is an additional goal to keep the networks as simple as possible while also achieving high accuracy. FPGAs can efficiently compute parallel

,

operations. Deep networks thus throw away this advantage. Therefore, the architectures explored here will need to remain with few layers.[1]

2 Data Description

The Dazzler takes as input a laser pulse which can either be described as an electric field or by an intensity and phase profile in either time, frequency, or wavelength. The laser is represented by a Gaussian pulse centered at 800 nm with flat phase.

The Dazzler has a set of 9 parameters that control the transfer function. Depending on the transfer function, the output shape from the Dazzler will be altered. These parameters are in three sets:

- Input laser parameters: central wavelength (λ_0), pulse width ($\delta\lambda_0$)
- Amplitude modulation parameters: hole position (λ_1), hole width ($\delta\lambda_1$), hole depth (k)
- Phase modulation parameters: delay (a_1), 2^{nd} order dispersion (a_2), 3^{rd} (a_3), 4^{th} (a_4)

The first set are based on the input waveform and are constant (same laser being used, so these will not be a part of the ML output). The second set amplitude modulation, and the third describe phase modulation. The amplitude and phase parameters are all chosen at random between limits listed in the Dazzler manual. They are then normalized to be between 0 and 1 for the network.

The output from the Dazzler is what will be fed into the neural network as input. The network will then output the transfer function parameters needed to get that particular output shape.

Given the pandemic, we have not been able to collect real data. As a part of this project, I generated all the data synthetically based on simulations. The data generation code is include in the code files. Figure 2 shows an example of an input waveform, transfer function, and output waveform in the WD for spectrum and phase.

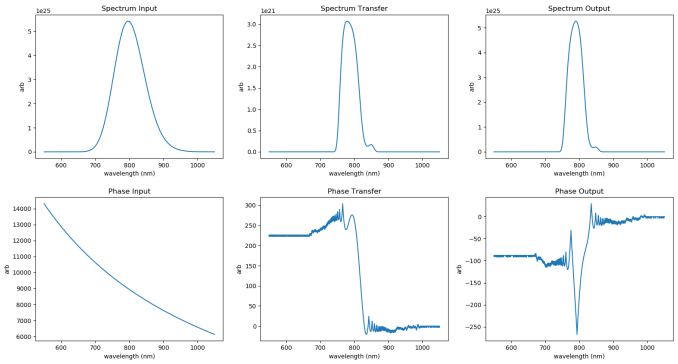


Figure 2: Example with parameters: $\lambda_0 = 800$ nm, $\delta\lambda_0 = 104$ nm, $\lambda_1 = 837$ nm, $\delta\lambda_1 = 94$ nm, $k = 0.75$, $a_1 = 1323$ fs, $a_2 = -773110$ fs^2, $a_3 = 118259$ fs^3, $a_4 = 55920569$ fs^4.

3 Model and Evaluation

The driving end-use of the model is in hardware, implemented on an FPGA. Therefore, the model must remain constrained in size (both in layer size and number of layers). Too large a depth also leads to increased latency in the data pipeline. As such the model is kept fairly simple. The first versions of the model were a three layer neural network. This particular model was informed by Boscolo and Finot [1] who present a neural network capable of retrieving the nonlinear propagation properties of a short-pulse traveling through a fiber given the pulse shape at the output of the fiber. Their network also consists of three hidden layers of fourteen neurons each and takes as input the temporal and spectral intensity profiles from a numerical simulation of the propagation.

[1] As a note, in the final system (not covered in this class) the training can be done offline, and the weights will be loaded from RAM into the FPGA for the inference (we are not, as of now, doing hardware-based training).

For mine, both a wavelength domain (WD) input and a time domain (TD) input have been tested. The inputs are 20,000 elements with the first 10,000 elements representing either intensity (for TD) or spectrum (for WD) and the latter 10,000 elements representing phase (either in TD or WD). The output layer is 7 units where each unit is a parameter input to the Dazzler. The activation function for the output layer is a sigmoid, producing an output from 0 to 1, which aligns with the normalized versions of the Dazzler parameters. The number and size of the hidden layers were the main elements being tuned for this model.

The model is trained on 15000 training examples. The total input size during training is thus 20000×15000, which is a fairly large input.

The cost function uses a weighted mean squared error (MSE) between Y and Y predicted (see eq 1). This cost function should give a good indication of how far-off the predicted Dazzler parameters are from the true values. The weight allows for adjustment of which Dazzler parameters affect the cost function the greatest. The weights were manually tuned across different runs.

Adam optimization was used for training with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e - 8$.

For evaluation of the model's performance, multiple applications of MSE were used. First a bar plot of the MSE for each Dazzler parameter is plotted in order to see which Dazzler parameters contribute most to the error. Next these parameters are converted back into their non-normalized form and are fed back into the Dazzler simulation model in order to produce a waveform. This is done for both the predicted parameters and the true parameters. The output of the dazzler simulation produces an electric field, an intensity profile in the TD, a phase profile in the TD, a spectrum profile in the WD, and a spectral phase profile in the WD. For each of these the MSE is calculated between the true and the predicted versions. This latter analysis is useful because it reveals how far off the real waveform shapes are rather than just how far off the parameters are.

$$\text{Cost} = \frac{1}{7m} \sum \text{weights} \times (Y - \hat{Y})^2; \tag{1}$$

Models were trained for anywhere from 800 to 2000 epochs with a learning rate ranging from 0.0001 to 0.001. Figure 3 shows an example cost curve.

4 Results

The main changes from one model to the next included altering number of layers, size of layers, weight parameters for cost function, number of epochs, and input data domain (wavelength or time). Before diving into the specifics of the results, a few trends will be discussed. First, the TD input data to the network seemed a lot less conducive for learning. Upon inspection of the cost over epochs for several different TD runs, the cost was not converging. This largely seemed to be a product of the randomly generated TD plots being a bit "wild" in shape. Next, in general it is very difficult to know what Bayes error is for this model. For some of the simple shapes, a human can do some estimation of the parameters but really won't be able to get a clear picture. As such, obtaining a threshold for acceptable error is not straightforward. In fact, the error seemed pretty low for the predicted Dazzler parameters and for the numerical comparisons of the waveform shapes. However, when visually inspecting the waveforms, the predicted and true were very different. Ways to address this in future versions of the model are discussed in the Conclusions.

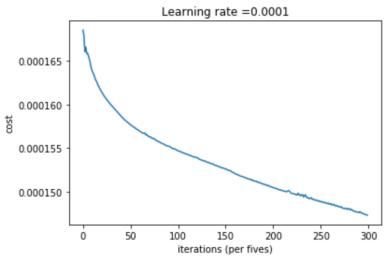


Figure 3: Cost optimization.

Another note is that the train and dev sets performed almost identically. Both of these data sets were synthetically generated from the same simulation, so it makes sense that they have similar results. As a result, bias really seems to be the driving factor in the error (and not variance). As such, most of the model adjustments came from exploring larger models (even though these may not be practical for FPGA implementation in our system).

Out of the models tested, the results for eight of them are presented below. The relevant information for each is presented first:

Model 1: 4 layer network; n_1: 50, n_2: 25, n_3: 15, n_4: 7; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 1$, depth : 1, $a_1 : 0.5$, $a_2 : 0.9$, $a_3 : 0.9$, $a_4 : 0.5$; Adam Optimization; Xavier Initialization; Input in WD

Model 2: 4 layer network; n_1: 50, n_2: 25, n_3: 15, n_4: 7; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 1$, depth : 1, a_1: 0.2, a_2:1, a_3:1, a_4:0.2; Adam Optimization; Xavier Initialization; Input in WD

Model 3: 4 layer network; n_1: 50, n_2: 25, n_3: 15, n_4: 7; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 2$, depth : 1, a_1: 0.2, a_2:5, a_3:5, a_4:0.2; Adam Optimization; Xavier Initialization; Input in WD

Model 4: 4 layer network; n_1: 25, n_2: 15, n_3: 7, n_4: 7; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 2$, depth : 1, a_1: 0.2, a_2:5, a_3:5, a_4:0.2; Adam Optimization; Xavier Initialization; Input in WD

Model 5: 4 layer network; n_1: 15, n_2: 15, n_3: 7, n_4: 7; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 2$, depth : 1, a_1: 0.2, a_2: 5, a_3: 5, a_4: 0.2; Adam Optimization; Xavier Initialization; Input in WD

Model 6: 4 layer network; n_1: 15, n_2: 15, n_3: 7, n_4: 7; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 2$, depth: 1, a_1: 0.2, a_2: 5, a_3: 5, a_4:0.2; Adam Optimization; Xavier Initialization; Input in TD

Model 7: 6 layer network; n_1: 25, n_2: 20, n_3: 10, n_4: 7, n_5: 7, n_6: 7 ; learning rate = 0.001, number of epochs = 800; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 2$, depth : 1, a_1: 0.2, a_2: 100, a_3: 100, a_4: 0.2; Adam Optimization; Xavier Initialization; Input in WD

Model 8: 6 layer network; n_1: 25, n_2: 20, n_3: 10, n_4: 7, n_5: 7, n_6: 7 ; learning rate = 0.001, number of epochs = 2000; $\beta_1 = 0.9$; $\beta_2 = 0.999$; $\epsilon = 1e-08$; weight parameters = $\lambda_1 : 1$, $\Delta\lambda_1 : 2$, depth : 1, a_1: 0.2, a_2: 100, a_3: 100, a_4: 0.2; Adam Optimization; Xavier Initialization; Input in WD

The results for each of these models are shown in tables 1 and 2 for the training and dev sets, respectively. Additionally, the MSE for the Dazzler parameters are shown for two of the models in figure 4.

	Models							
	1	2	3	4	5	6	7	8
Electric Field MSE	3.98e-01	3.97e-01	4.01e-01	4.03e-01	3.99e-01	3.42e-01	3.91e-01	3.70e-01
Intensity (TD) MSE	8.70e-02	9.45e-02	8.98e-02	9.79e-02	1.00e-01	9.35e-02	9.62e-02	8.52e-02
Phase (TD) MSE	3.47e-05	7.39e-05	5.06e-05	6.64e-05	5.65e-05	2.01e-05	1.10e-04	1.91e-04
Spectrum (WD) MSE	5.87e-03	5.32e-03	6.28e-03	7.91e-03	8.88e-03	2.22e-02	8.41e-03	1.36e-02
Phase (WD) MSE	5.52e-01	5.33e-01	5.45e-01	4.83e-01	5.29e-01	4.36e-01	5.54e-01	5.97e-01

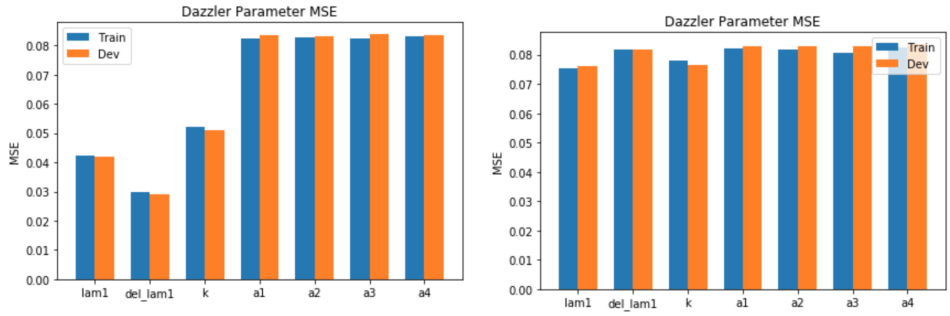
Table 1: Mean squared error (MSE) for 8 models for training data set. Details of the 8 models are shown in text. MSE is shown for electric field, intensity (in TD), phase (in TD), spectrum (in WD), and phase (in WD).

	Models							
	1	2	3	4	5	6	7	8
Electric Field MSE	2.38e-01	2.38e-01	2.41e-01	2.41e-01	2.39e-01	2.04e-01	2.34e-01	2.21e-01
Intensity (TD) MSE	5.22e-02	5.64e-02	5.40e-02	5.86e-02	6.03e-02	5.58e-02	5.76e-02	5.10e-02
Phase (TD) MSE	2.08e-05	4.37e-05	3.03e-05	4.00e-05	3.37e-05	1.16e-05	6.49e-05	1.14e-04
Spectrum (WD) MSE	3.52e-03	3.17e-03	3.77e-03	4.70e-03	5.30e-03	1.34e-02	5.00e-03	8.14e-03
Phase (WD) MSE	3.32e-01	3.21e-01	3.29e-01	2.92e-01	3.20e-01	2.62e-01	3.34e-01	3.60e-01

Table 2: Mean squared error (MSE) for 8 models for dev data set. Details of the 8 models are shown in text. MSE is shown for electric field, intensity (in TD), phase (in TD), spectrum (in WD), and phase (in WD).

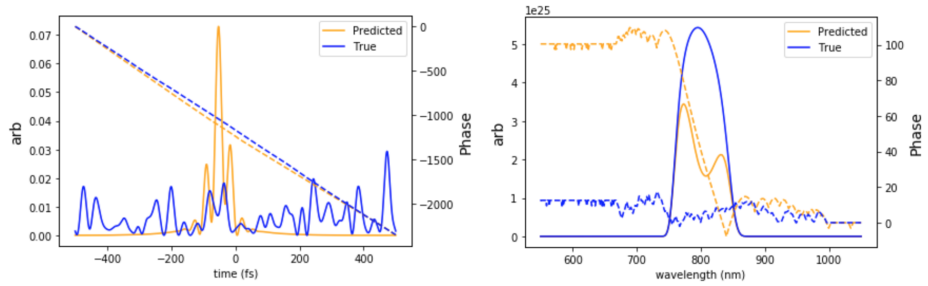
The results show that there is not a significant difference in performance between training and dev, and the model even is performing slightly better on dev (which means it has not properly learned yet!). This is an issue to be addressed and more training needs to be done. Furthermore, there errors seem pretty low, but in reality there are large portions of the waveforms that are 0 and thus the averaging

in the evaluation metric makes the error look better than it is. To get a better handle on how bad this error is see figure 5. These show the predicted and true TD and WD plots. The error for the TD intensity on this one is 4.97e-06 and the error for the WD spectrum is 1.04e-06. These are very small error values compared to those in the tables yet the predicted versus true waveforms are vastly different. The same rationale applied to figure 4 where the error appears low. This seems to imply that different training methods should be used, including alteration to the model and changes to the cost function used for training.



(a) (b)

Figure 4: MSE for Dazzler parameters for training and dev sets for model 1 (a) and for model 8 (b).



(a) (b)

Figure 5: Plots of true and predicted waveforms for Model 8. (a) shows the TD plots for intensity (solid) and phase (dashed), with respective scales on left and right. (b) shows the WD plots for spectrum (solid) and phase (dashed), with respective scales on left and right.

5 Conclusions

While the models presented here were able to capture some features of the problem, overall the models failed to learn the mapping. The error was too large and the comparison of the waveforms shows just how far-off it is. For this reason, the bulk of testing focused on reducing bias (did not use a test set and did not try drop-out, etc). As this is on-going research, the next steps will be to simplify the input parameter space to the model. This will include separating amplitude and phase modulation into two separate models and testing those individually. Additionally, shrinking the input to the network would prove beneficial. For instance, if using WD could artificially restrict the input to ± 100 nm from the central wavelength. Additionally, may try altering the cost function to use the waveforms instead of the Dazzler parameters directly. Could do this by, during training, feeding the predicted Dazzler parameters into the Dazzler simulation code to get the waveforms. This will lengthen training but could prove beneficial.

Boscolo and Finot were able to solve a similar problem. This particular problem here does have a larger input shape space than from their work; however, I think this problem is still solvable!

References

[1] Sonia Boscolo and Christophe Finot. "Artificial neural networks for nonlinear pulse shaping in optical fibers". In: *Optics & Laser Technology* 131 (2020), p. 106439. ISSN: 0030-3992. DOI: https://doi.org/10.1016/j.optlastec.2020.106439. URL: http://www.sciencedirect.com/science/article/pii/S0030399220310720.

[2] *Dazzler user manual volume I: installation, setup & operation*. Fastlite. 2020.