Using Neural Networks for Rapid Switching Between Beam Parameters in an FEL

Auralee Edelen
Fermilab and Colorado State University

Working with: Jonathan Edelen, Sandra Biedron, Stephen Milton, Peter van der Slot
Motivation: Switching Between User Requests

- FEL facilities support a wide variety of scientific endeavors (e.g. imaging protein structures\(^1\), understanding processes like photosynthesis\(^2\), origin of material properties\(^3\))


Motivation: Switching Between User Requests

- FEL facilities support a wide variety of scientific endeavors (e.g. imaging protein structures¹, understanding processes like photosynthesis², origin of material properties³)
- Need to accommodate requests for a wide variety of photon beam characteristics

---

Motivation: Switching Between User Requests

- FEL facilities support a wide variety of scientific endeavors (e.g., imaging protein structures\(^1\), understanding processes like photosynthesis\(^2\), origin of material properties\(^3\))

- Need to accommodate requests for a wide variety of photon beam characteristics

- May switch as often as every few days

- Have save/restore settings, but these are discrete, and there can be some drift in the machine


\(\text{e.g. the Linac Coherent Light Source} \quad \text{(image: lcls.slac.standford.edu)}\)
Motivation: Switching Between User Requests

• FEL facilities support a wide variety of scientific endeavors (e.g. imaging protein structures\(^1\), understanding processes like photosynthesis\(^2\), origin of material properties\(^3\))

• Need to accommodate requests for a wide variety of photon beam characteristics

• May switch as often as every few days

• Have save/restore settings, but these are discrete, and there can be some drift in the machine

• Time spent tuning = reduced scientific output for a given operational budget


Motivation: Switching Between User Requests

- FEL facilities support a wide variety of scientific endeavors (e.g. imaging protein structures\(^1\), understanding processes like photosynthesis\(^2\), origin of material properties\(^3\))

- Need to accommodate requests for a wide variety of photon beam characteristics

- May switch as often as every few days

- Have save/restore settings, but these are discrete, and there can be some drift in the machine

- Time spent tuning = reduced scientific output for a given operational budget

Would be nice to have a tool that can quickly give suggested settings for a given photon beam request, is valid globally, and can adapt to changes over time

---


Starting Smaller: A Case Study

Compact, THz FEL design based on previously operational TEU-FEL

3 – 6 MeV electron beam
200 – 800 $\mu$m photon beam

Previously operated at University of Twente in the Netherlands

Was going to be re-built at CSU: have simulation from design studies
Starting Smaller: A Case Study

This is an appealing system for an initial study because it has a small number of machine components, yet it exhibits non-trivial beam dynamics.

Compact, THz FEL design based on previously operational TEU-FEL

3 – 6 MeV electron beam
200 – 800 μm photon beam

Previously operated at University of Twente in the Netherlands
Was going to be re-built at CSU: have simulation from design studies
How to get the right wavelength?
How to get the right wavelength?

**FEL output is related to beam parameters at the entrance of the undulator**

Roughly speaking:

- Beam *energy* determines FEL wavelength
- Beam *size* ($\beta$) and *divergence* ($\alpha$) need to be set to minimize beam losses
- Beam *emittance* ($\varepsilon$) impacts FEL gain
- $\alpha, \beta, \varepsilon$ are defined in the position-momentum phase space of the beam

Simple analytic case:

$$\lambda_r = \frac{\lambda_u}{2\gamma^2} \left( 1 + \frac{K^2}{2} \right)$$

*(in reality the FEL process is more complicated)*
How to get the right wavelength?

Quadrupole electromagnets are used to match the beam into the undulator

- Focus in one transverse plane and defocus in the other
- A pair provides net focusing
- In principle only affects $\alpha, \beta$
  (but beam self-fields can thwart this $\Rightarrow$ also affects $\varepsilon$)
Photoinjector determines initial beam properties and accelerates the beam

- Electrons generated via photoelectric effect (laser incident on cathode)
- Beam energy dominated by RF power setting (acceleration in cavity)
- Solenoid compensates for strong beam self-fields (improves emittance)
- Bucking coil minimizes magnetic field on the cathode (improves emittance)
End goal: get the right beam parameters at the undulator entrance
First: Learn a Model from Physics-Based Simulation

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes beam self-fields (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
First: Learn a Model from Physics-Based Simulation

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes beam self-fields (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → couldn’t just run a lot of interactions with controller on a cluster
First: Learn a Model from Physics-Based Simulation

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes beam self-fields (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → couldn’t just run a lot of interactions with controller on a cluster

Decided to learn a neural network model from simulation:
- faster-executing than physics-based simulation
- can update with measured data
- would be nice to have a differentiable model
First: Learn a Model from Physics-Based Simulation

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes beam self-fields (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for windows → couldn’t just run a lot of interactions with controller on a cluster

Decided to learn a neural network model from simulation:
- faster-executing than physics-based simulation
- can update with measured data
- would be nice to have a differentiable model

More broadly: machine time is expensive, mistakes can be costly, and simulations don’t always match the machine well
- Sample efficiency matters a lot (both with slow sim and machine)
- Learning a machine model using simulation results and updating it with existing measurements can aid controller development
First: Learn a Model from Physics-Based Simulation

Simulation in PARMELA

- Standard particle tracking code (numerical)
- Includes beam self-fields (computationally expensive)
- Load EM field maps for cavities, solenoid, bucking coil
- Unfortunately: distribution restricted, source code not available, and compiled for Windows → couldn’t just run a lot of interactions with controller on a cluster

Decided to learn a neural network model from simulation:
- faster-executing than physics-based simulation
- can update with measured data
- would be nice to have a differentiable model

More broadly: machine time is expensive, mistakes can be costly, and simulations don’t always match the machine well

→ Sample efficiency matters a lot (both with slow sim and machine)
→ Learning a machine model using simulation results and updating it with existing measurements can aid controller development
Don’t always have a good physics-based model for particle accelerators, so what's in the data archive of a real facility?

Noisy data + tuning around roughly optimal settings
Don’t always have a good physics-based model for particle accelerators, so what’s in the data archive of a real facility?

Noisy data + tuning around roughly optimal settings
Get Training Data from Simulation

- beam parameters $p$
- settings $s$

Train Forward and Inverse NN Models

- all samples
- Leave out one energy range for validation

Don’t always have a good physics-based model for particle accelerators, so what’s in the data archive of a real facility? *Noisy data + tuning around roughly optimal settings*
Get Training Data from Simulation

- Optimizer
- Physics Simulation
- settings
- beam parameters $p$

repeat for different target energies

Train Forward and Inverse NN Models

- all samples
- Forward Model
- Leave out one energy range for validation

Don’t always have a good physics-based model for particle accelerators, so what’s in the data archive of a real facility?

Noisy data + tuning around roughly optimal settings

Bulk beam parameter estimation relies on good statistics $\rightarrow$ only train on those outputs when transmission $> 90\%$
Don’t always have a good physics-based model for particle accelerators, so what’s in the data archive of a real facility?

*Noisy data + tuning around roughly optimal settings*

Bulk beam parameter estimation relies on good statistics → only train on those outputs when transmission > 90%

Want to use the existing data to initialize control policy → model not invertible, but can pre-train policy with converged settings
Don’t always have a good physics-based model for particle accelerators, so what’s in the data archive of a real facility? 

*Noisy data + tuning around roughly optimal settings*

Bulk beam parameter estimation relies on good statistics → *only train on those outputs when transmission > 90%*

Want to use the existing data to initialize control policy → *model not invertible, but can pre-train policy with converged settings*
Don’t always have a good physics-based model for particle accelerators, so what's in the data archive of a real facility?

*Noisy data + tuning around roughly optimal settings*

Bulk beam parameter estimation relies on good statistics $\rightarrow$ *only train on those outputs when transmission > 90%*

Want to use the existing data to initialize control policy $\rightarrow$ *model not invertible, but can pre-train policy with converged settings*
Training the Control Policy (v0)

- *First: just want to switch to roughly correct settings*
- *Then, two options: efficient local tuning algorithms we already use, or online model/controller updating*
Training the Control Policy (v0)

- **First**: just want to switch to roughly correct settings
- **Then**, two options: efficient local tuning algorithms we already use, or online model/controller updating

---

**NN Control Policy Update**

- **Batch of** $p_t$
- **Policy** $s'$
- **Forward Model** $(frozen)$ $p'$
- **Cost**: $C(p_t, p', s')$
- **Add** $(s', p')$ to database $D$

**Cost:**
- difference between $p'$ and $p_t$
- penalize loss of transmission
- penalize higher magnet settings
Training the Control Policy (v0)

- *First: just want to switch to roughly correct settings*
- *Then, two options: efficient local tuning algorithms we already use, or online model/controller updating*

**NN Control Policy Update**

- Batch of $p_t$
- Policy $s' \xrightarrow{}$ Forward Model $p'$
- $C(p_t, p', s')$
- add $(s', p')$ to database $D$

**Cost:**
- difference between $p'$ and $p_t$
- penalize loss of transmission
- penalize higher magnet settings

Every $n^{th}$ iteration, take batch of $s', p'$ sampled from $D$, run through physics simulation, and update the model.
Training the Control Policy (v0)

- First: just want to switch to roughly correct settings
- Then, two options: efficient local tuning algorithms we already use, or online model/controller updating

![Diagram of NN Control Policy Update]

- Batch of $p_t$ → Policy → $s'$ → Forward Model (frozen) → $p'$ → cost $C(p_t, p', s')$ → add $(s', p')$ to database $D$

Cost: difference between $p'$ and $p_t$
- penalize loss of transmission
- penalize higher magnet settings

Every $n^{th}$ iteration, take batch of $s', p'$ sampled from $D$, run through physics simulation, and update the model

Then test policy directly on simulation
Initial Model and Policy

Training data from simulation:
• output from each iteration of Nelder-Mead, L-BFGS
• 12 beam energies between 3.1 – 6.2 MeV (7195 samples)
Initial Model and Policy

Training data from simulation:
- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)

Example of what the training data looks like
(quads in this case)
Initial Model and Policy

Training data from simulation:
• output from each iteration of Nelder-Mead, L-BFGS
• 12 beam energies between 3.1 – 6.2 MeV (7195 samples)

Model: 50-50-30-30 tanh nodes in hidden layers
- 8 inputs (rf power, rf phase, sol. strength, quads)
- 8 outputs (\(\alpha_{xy}, \beta_{xy}, \varepsilon_{xy}, E, N_p\))
- 5.7-MeV run used for validation set

Bulk beam parameter estimation relies on good statistics
\(\rightarrow\) only train on those outputs when transmission > 90%

Example of what the training data looks like
(quads in this case)
Initial Model and Policy

Training data from simulation:
- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)

Model: 50-50-30-30 tanh nodes in hidden layers
- 8 inputs (rf power, rf phase, sol. strength, quads)
- 8 outputs ($\alpha_{xy}, \beta_{xy}, \varepsilon_{xy}, E, N_p$)
- 5.7-MeV run used for validation set

Bulk beam parameter estimation relies on good statistics
\rightarrow only train on those outputs when transmission > 90%

First study: focus on target $\alpha, \beta$ for a given energy
\rightarrow don’t allow variation in gun settings beyond known optima
\rightarrow exclude emittance in cost

Example of what the training data looks like
(quads in this case)
Initial Model and Policy

Training data from simulation:
- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)

Example of what the training data looks like
(quads in this case)

Bulk beam parameter estimation relies on good statistics
→ only train on those outputs when transmission > 90%

First study: focus on target $\alpha, \beta$ for a given energy
→ don’t allow variation in gun settings beyond known optima
→ exclude emittance in cost

Model: 50-50-30-30 tanh nodes in hidden layers
- 8 inputs (rf power, rf phase, sol. strength, quads)
- 8 outputs ($\alpha_{xy}, \beta_{xy}, \varepsilon_{xy}, E, N_p$)
- 5.7-MeV run used for validation set

Policy: 30-30-20-20 tanh nodes in hidden layers
- inputs/outputs opposite the above (except $N_p$)
- random target energies, $\alpha_{xy} = 0, \beta_{xy} = 0.106$
- exclude 4.8 – 5.2 MeV range for validation
Initial Model and Policy

Training data from simulation:
- output from each iteration of Nelder-Mead, L-BFGS
- 12 beam energies between 3.1 – 6.2 MeV (7195 samples)

Model: 50-50-30-30 tanh nodes in hidden layers
- 8 inputs (rf power, rf phase, sol. strength, quads)
- 8 outputs ($\alpha_{xy}$, $\beta_{xy}$, $\varepsilon_{xy}$, $E$, $N_p$)
- 5.7-MeV run used for validation set

Bulk beam parameter estimation relies on good statistics → only train on those outputs when transmission > 90%

First study: focus on target $\alpha$, $\beta$ for a given energy → don’t allow variation in gun settings beyond known optima → exclude emittance in cost

Policy: 30-30-20-20 tanh nodes in hidden layers
- inputs/outputs opposite the above (except $N_p$)
- random target energies, $\alpha_{xy} = 0$, $\beta_{xy} = 0.106$
- exclude 4.8 – 5.2 MeV range for validation

- weights/biases updated with AdaMax
- batch size of 200
- implemented in Theano and lasagne

Example of what the training data looks like (quads in this case)
Initial Model and Policy Performance

First study: focus on target Twiss parameters and don’t allow variation in gun settings beyond known optima

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Train MAE</th>
<th>Train STD</th>
<th>Train Max</th>
<th>Val. MAE</th>
<th>Val. STD</th>
<th>Val. Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_x$ [rad]</td>
<td>0.018</td>
<td>0.042</td>
<td>0.590</td>
<td>0.067</td>
<td>0.091</td>
<td>0.482</td>
</tr>
<tr>
<td>$\alpha_y$ [rad]</td>
<td>0.022</td>
<td>0.037</td>
<td>0.845</td>
<td>0.070</td>
<td>0.079</td>
<td>0.345</td>
</tr>
<tr>
<td>$\beta_x$ [m/rad]</td>
<td>0.004</td>
<td>0.009</td>
<td>0.287</td>
<td>0.008</td>
<td>0.012</td>
<td>0.130</td>
</tr>
<tr>
<td>$\beta_y$ [m/rad]</td>
<td>0.005</td>
<td>0.011</td>
<td>0.357</td>
<td>0.012</td>
<td>0.017</td>
<td>0.189</td>
</tr>
</tbody>
</table>
Initial Model and Policy Performance

First study: focus on target Twiss parameters and don’t allow variation in gun settings beyond known optima

Summary of Model Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Train MAE</th>
<th>Train STD</th>
<th>Train Max</th>
<th>Val. MAE</th>
<th>Val. STD</th>
<th>Val. Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_x$ [rad]</td>
<td>0.018</td>
<td>0.042</td>
<td>0.590</td>
<td>0.067</td>
<td>0.091</td>
<td>0.482</td>
</tr>
<tr>
<td>$\alpha_y$ [rad]</td>
<td>0.022</td>
<td>0.037</td>
<td>0.845</td>
<td>0.070</td>
<td>0.079</td>
<td>0.345</td>
</tr>
<tr>
<td>$\beta_x$ [m/rad]</td>
<td>0.004</td>
<td>0.009</td>
<td>0.287</td>
<td>0.008</td>
<td>0.012</td>
<td>0.130</td>
</tr>
<tr>
<td>$\beta_y$ [m/rad]</td>
<td>0.005</td>
<td>0.011</td>
<td>0.357</td>
<td>0.012</td>
<td>0.017</td>
<td>0.189</td>
</tr>
</tbody>
</table>

Controller ability to reach $\alpha_{x,y} = 0$ and $\beta_{x,y} = 0.106$ in one iteration
Initial Model and Policy Performance

First study: focus on target Twiss parameters and don’t allow variation in gun settings beyond known optima

Summary of Model Performance

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Train MAE</th>
<th>Train STD</th>
<th>Train Max</th>
<th>Val. MAE</th>
<th>Val. STD</th>
<th>Val. Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_x$ [rad]</td>
<td>0.018</td>
<td>0.042</td>
<td>0.590</td>
<td>0.067</td>
<td>0.091</td>
<td>0.482</td>
</tr>
<tr>
<td>$\alpha_y$ [rad]</td>
<td>0.022</td>
<td>0.037</td>
<td>0.845</td>
<td>0.070</td>
<td>0.079</td>
<td>0.345</td>
</tr>
<tr>
<td>$\beta_x$ [m/rad]</td>
<td>0.004</td>
<td>0.009</td>
<td>0.287</td>
<td>0.008</td>
<td>0.012</td>
<td>0.130</td>
</tr>
<tr>
<td>$\beta_y$ [m/rad]</td>
<td>0.005</td>
<td>0.011</td>
<td>0.357</td>
<td>0.012</td>
<td>0.017</td>
<td>0.189</td>
</tr>
</tbody>
</table>

Controller ability to reach $\alpha_{x,y} = 0$ and $\beta_{x,y} = 0.106$ in one iteration

Example of Model Performance on Validation Set

What this means: for a given energy, the controller will immediately reach the desired beam size to within about 10% and the beam will be close to a waist, requiring minimal further tuning (assuming no drift...)

Presently working on the next steps for the complete study

- Including minimization of emittance + more freedom with injector settings
  - Requires finer start-to-end adjustments, so more simulation data was needed
  - Larger network needed to capture relationships accurately in model

- Need to see how well it does with machine drift
  - e.g. deviation between settings and real values, deviation in responses

- Need to compare with other methods
  - Online optimization methods used in accelerators
  - Try comparing with some model-free RL benchmarks (e.g. TRPO)

- Have plans for trying this approach on an operational machine

- Other tweaks:
  - Specify change in setting rather than setting
  - Weights of cost function should be tuned
Conclusion

• *Initial study for fast switching between beam energies while preserving* $\alpha$, $\beta$ *looks encouraging*

• *Continuing with more complete study*

• *Will be interesting to see how this might scale to a larger accelerator system*