Training Ebru's 2D Dynamic Scheduling Model

- Benchmark: Ebru's Python implementation
- **Attempt 1**: Replicate Ebru's results using (modified) Han's code under simple settings
- **Attempt 2**: Adding batch normalization for input layer
- **Attempt 3**: Adding batch normalization for all layers
- **Attempt 4**: Adding shape constraints (and removing batch normalization)

Benchmark: Ebru's Python implementation

Ebru's test instance:

- 2 dimensions, 17 hours, 204 time steps
- Reference policy: all-zero control

Running Ebru's Python implementation gives the following results:

- Start loss: ~ 10,500
- End loss: ~ 50
- Total steps: 2,000
- Sample average of trained $V^{\mathrm{NN}}(0, X_0)$: ~ 108 (using $X_0 \sim \mathrm{Uniform}(0, 1)$)

I checked above with Ebru for correctness. I'll try to replicate these results using Han's code.

Remark 1: Ebru's later Julia implementation contains more advanced "engineering tricks". However, I'm not familiar with Julia, so I'll start with her Python code for now.

Remark 2: To train Ebru's model myself, I:

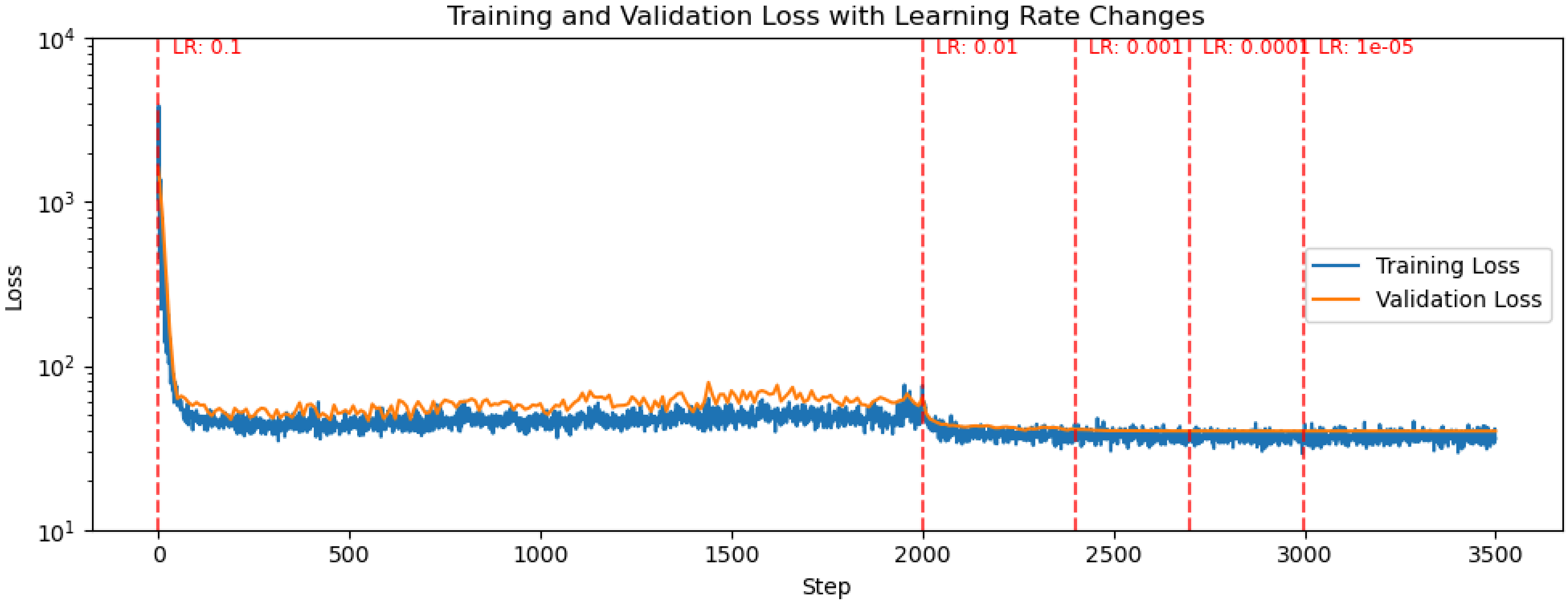
- Rewrote Han's code to use PyTorch instead of TensorFlow (for GPU compatibility)
- Modify the code to allow randomized X_0 (required by Ebru's model)
- Check the updated code using Han's test cases (for correctness of the code)

Attempt 1: Training Ebru's model using Han's code under simple settings

I started with training Ebru's model using the following settings:

Hyperparameters	**Values**
Neural network architecture	MLP
Number of hidden layers	4
Number of nodes per layers	100
Activation function	ReLU
Precision	float64
Optimizer	Adam
Batch size (training)	256
Batch size (validation)	512
Number of iterations	TBD (manual adjustment)
Learning rate schedule	Piecewise decay (manual)
Learning rates	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$

Observations: (using Dawei's plotting approach)



Loss goes from 10,500 to 40, very similar to Ebru's results.

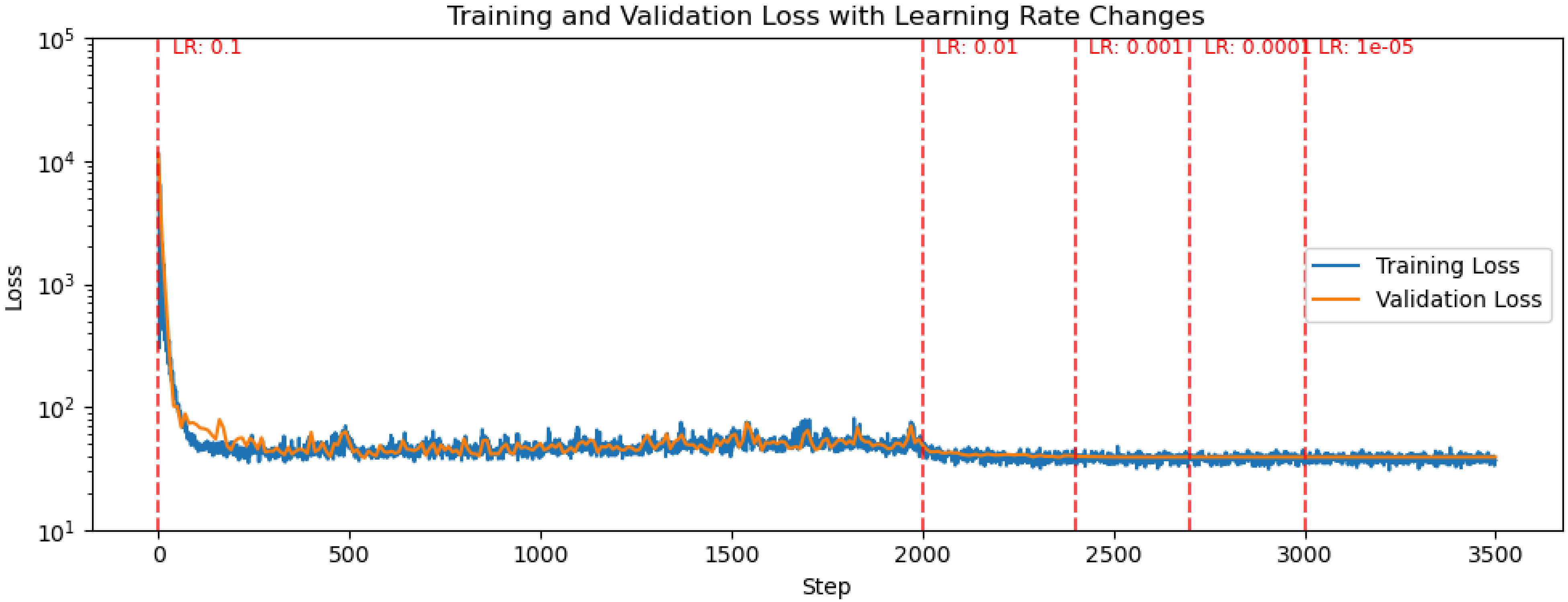
Sample average of trained $V^{\mathrm{NN}}(0, X_0)$ is ~ 75: lower than Ebru's ~ 108.

Attempt 2: Adding batch normalization for input layer

I observed that Ebru's code used batch normalization for the input layer. Thus, I tested the following:

Hyperparameters	**Values**
Neural network architecture	MLP
Number of hidden layers	4
Number of nodes per layers	100
Activation function	ReLU
Batch normalization	Input layer only
Precision	float64
Optimizer	Adam
Batch size (training)	256
Batch size (validation)	512
Number of iterations	TBD (manual adjustment)
Learning rate schedule	Piecewise decay (manual)
Learning rates	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$

Observations:



Loss still goes from 10,500 to 40, similar to Attempt 1.

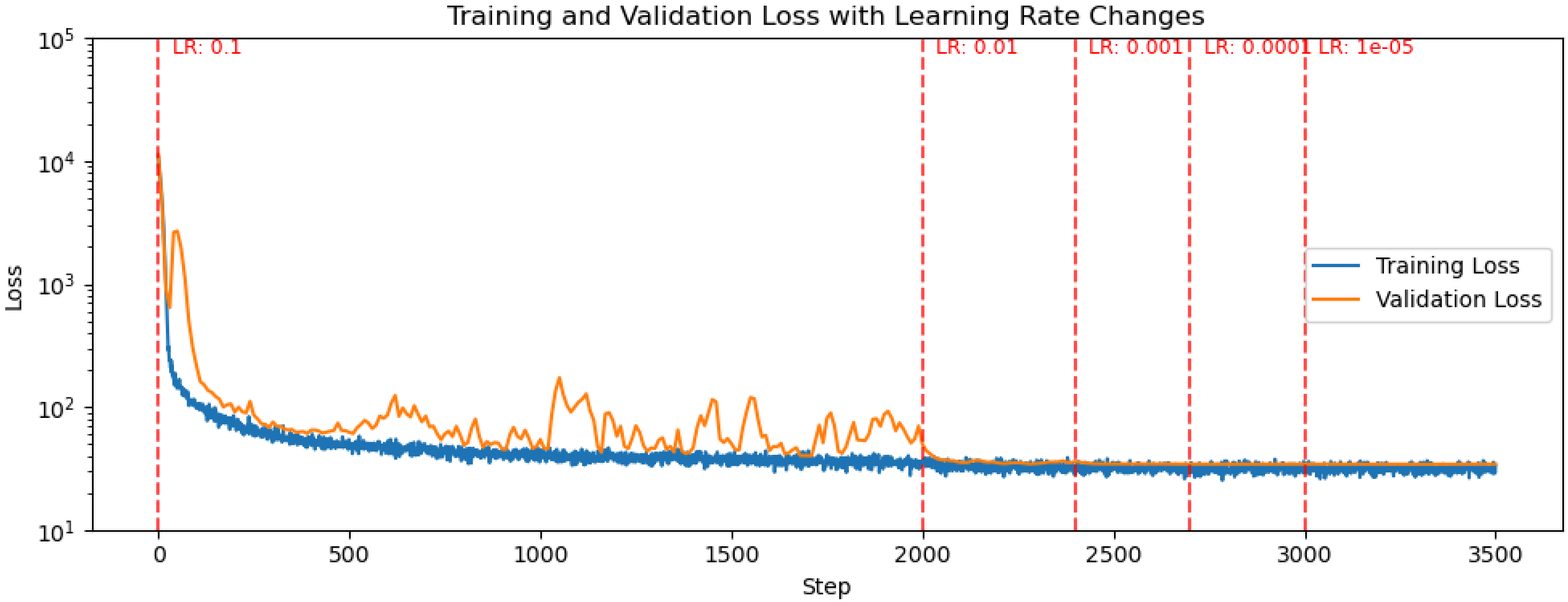
Sample average of trained $V^{\mathrm{NN}}(0, X_0)$ is still ~ 75, similar to Attempt 1.

Attempt 3: Adding batch normalization for all layers

Recall that Dawei used batch normalization for all layers. Thus, I tested the following:

Hyperparameters	**Values**
Neural network architecture	MLP
Number of hidden layers	4
Number of nodes per layers	100
Activation function	ReLU
Batch normalization	All layers
Precision	float64
Optimizer	Adam
Batch size (training)	256
Batch size (validation)	512
Number of iterations	TBD (manual adjustment)
Learning rate schedule	Piecewise decay (manual)
Learning rates	$10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$

Observations:



Loss still goes from 10,500 to 40, but converges slower. Validation loss deviates from training loss.

Sample average of trained $V^{\mathrm{NN}}(0, X_0)$ is ~ 10, worse than Attempt 1.

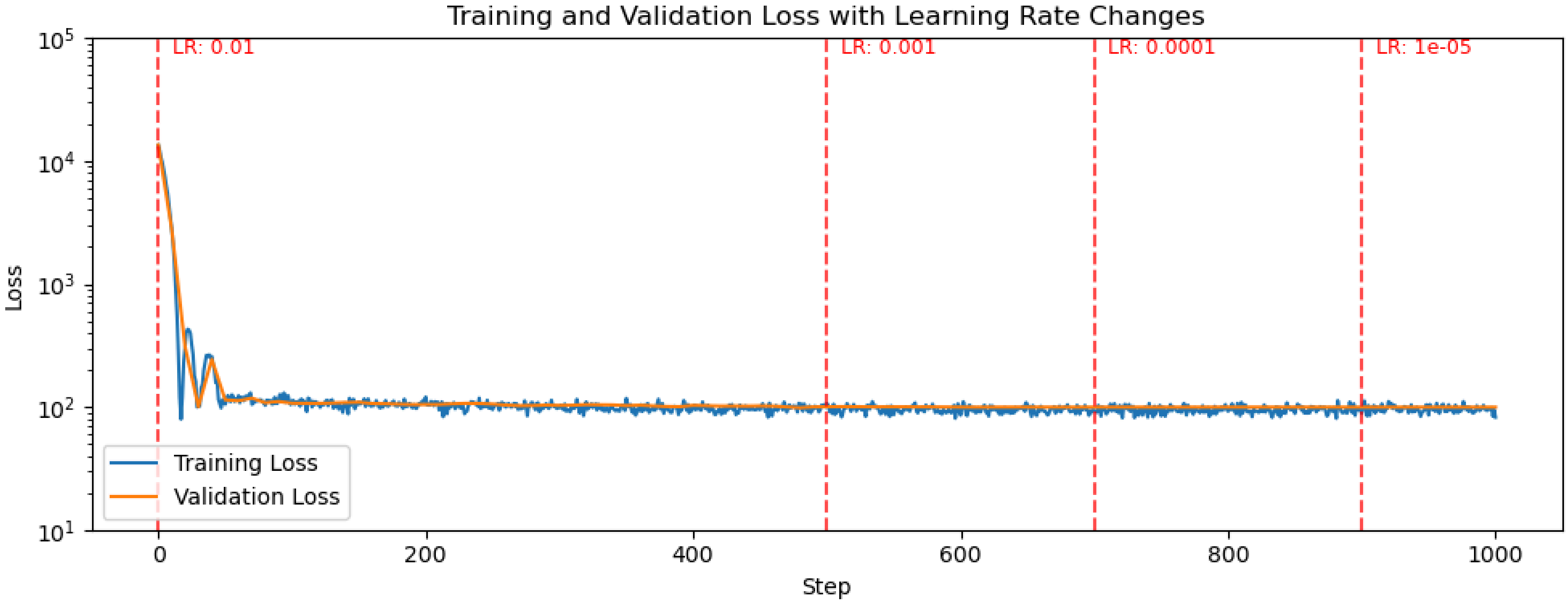
Attempt 4: Adding shape constraints

In a previous discussion, Ebru mentioned that shape constraints helped her training a lot.

Thus, I tested the following:

Hyperparameters	**Values**
Neural network architecture	MLP
Number of hidden layers	4
Number of nodes per layers	100
Activation function	ReLU
Batch normalization	None
Shape constraints	0.5 for negative derivatives
Precision	float64
Optimizer	Adam
Batch size (training)	256
Batch size (validation)	512
Number of iterations	TBD (manual adjustment)
Learning rate schedule	Piecewise decay (manual)
Learning rates	$10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$

Observations:



Loss still goes from 13,600 to 100, and converges faster.

Sample average of trained $V^{\mathrm{NN}}(0, X_0)$ is ~ 109, very close to Ebru's ~ 108.

Remarks and Next Steps

- I was able to replicate Ebru's results by using shape constraints.
- Interetingly, Ebru's Python code did not use shape constraints, but only used batch normalization for the input layer. I'll discuss with Ebru look deeper into the difference in the two implementations.