Scheduler Variations

Hans C. Suganda

8th January 2022

Contents

0.1 Rationale

Previously, it was established that as the weights of the neural network approached a local minima in the loss function, the learning rate must be reduced in order not to "overshoot" the local minima and cause stability issues. This stability issue seems more prominent in "larger" neural network architectures containing more nodes and hidden layers. Here a learning rate scheduler is implemented which allows the learning rate to change with the number of training iterations. As the number of training iterations increase, the learning rate reduces based on a variety of pre-defined functions. This was implemented in the hopes of accelearting training of the neural network at lower training iterations where stability is not likely to be an issue but then reducing the learning rate when the neural network has been sufficiently trained and stability issues are likely to occur.

0.2 Algorithm Changes

Generally, only a minor piece of the code must be modified, namely where the learning rate scheduler was defined.

0.2.1 Exponential LR

The initial modification sets the learning rate to exponential decay. The code is shown below,

```
#Determine the Learning Rate
lr_schedule = keras.optimizers.schedules.ExponentialDecay(
    initial_learning_rate=5e-4,
    decay_steps=250000,
    decay_rate=0.06,
    name=None)
```

0.2.2 Polynomial LR

Another type of scheduling scheme that `keras` allows is the polynomial decay. The code is shown below,

```
#Determine the Learning Rate
lr_schedule = keras.optimizers.schedules.PolynomialDecay(
    initial_learning_rate=5e-4,
    decay_steps=250000,
    end_learning_rate= 3e-5,
    power=1,
    cycle=False,
    name=None)
```

0.2.3 Inverse LR

The last scheme that is used in this short investigation is inverse decay. The code is shown below,

```
#Determine the Learning Rate
lr_schedule = keras.optimizers.schedules.InverseTimeDecay(
    initial_learning_rate=5e-4,
    decay_steps=250000,
    decay_rate=15.66666,
    staircase=False,
    name=None)
```

0.3 Choice of Parameters

To facilitate a direct comparison to previous results, the number of maximum training iterations here is kept identical to the number of training iterations in the previous work. Therefore, the maximum number of training iterations is 250000. In this investigation, the learning rate should be somewhat comparable to the previous learning rate. Since the learning rate of the previous work is $1e-4$, the initial learning rate here is chosen to be $5e-4$. The final learning rate after 250000 training iterations is chosen to be $3e-5$. These choices are somewhat arbitrary, but the decision was made with the idea that the initial learning rate should be higher to facillitate faster convergence but the "final" learning rate should be lower to avoid stability issues.

0.3.1 Exponential LR

The exponential decay learning rate function introduces 3 main variables. The true learning rate represented as `lr` is defined below,

```
lr = initial_learning_rate * decay_rate ^ (step / decay_steps)
```

The current training iteration number is represented by the variable `step`. The exponential decay means that the learning rate will decay by `decay_rate` every `decay_steps`. If `decay_steps` is set to the maximum number of training iterations 250000, then `step/decay_steps` when `step` is the last possible step at 250000 yields 1, therefore,

$$\frac{lr_f}{lr_i} = d_r$$

wherein lr_f represents final learning rate, lr_i represents initial learning rate and d_r represents decay rate. This expression is true given the assumptions above. Substituting for initial and final learning rates,

$$d_r = \frac{3 \times 10^{-5}}{5 \times 10^{-4}} = 0.06$$

0.3.2 Polynomial LR

The polynomial decay learning rate function introduces 4 main variables. The true learning rate represented as `lr` is defined below,

```
lr = ((initial_learning_rate - end_learning_rate) * (1 - step / decay_steps) ^ (power)) +
    end_learning_rate
```

Initially when the training iteration count is 0, the variable `step` is also 0, and the learning rate `lr` is equals to the variable `initial_learning_rate`. When the neural network is at its last training iteration, `1-step/decay_steps` is zero. And the variable `lr` is equals to the variable `end_learning_rate`. Without pre-calculations, we can simply substitute, the initial and final learning rates into variables `initial_learning_rate` and `end_learning_rate`. The variable `power` is somewhat arbitrarily decided and for this, we have decided to go with 1.

0.3.3 Inverse LR

The inverse decay learning rate function introduces 3 main variables. The true learning rate represented as `lr` is defined below,

```
lr = initial_learning_rate / (1 + decay_rate * step / decay_step)
```

By simple algebraic manipulations assuming at the last training iteration when `step` is the same as `decay_step`,

$$\frac{lr_i}{lr_f} = 1 + d_r$$

$$\frac{lr_i}{lr_f} - 1 = d_r$$

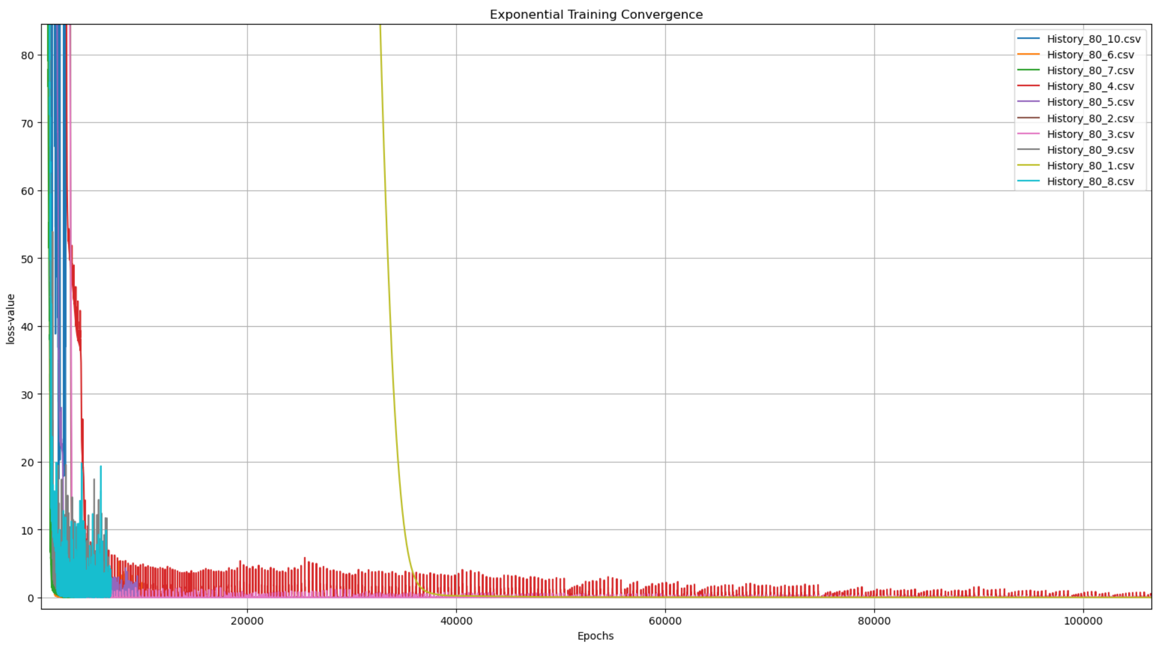
wherein lr_i represents initial learning rate, lr_f represents final learning rate, d_r represents decay rate, s_t represents step, and d_{st} represents decay step. Substituting for the desired initial and final learning rates,

$$dr = \frac{5 \times 10^{-4}}{3 \times 10^{-5}} - 1 = 15.6666$$

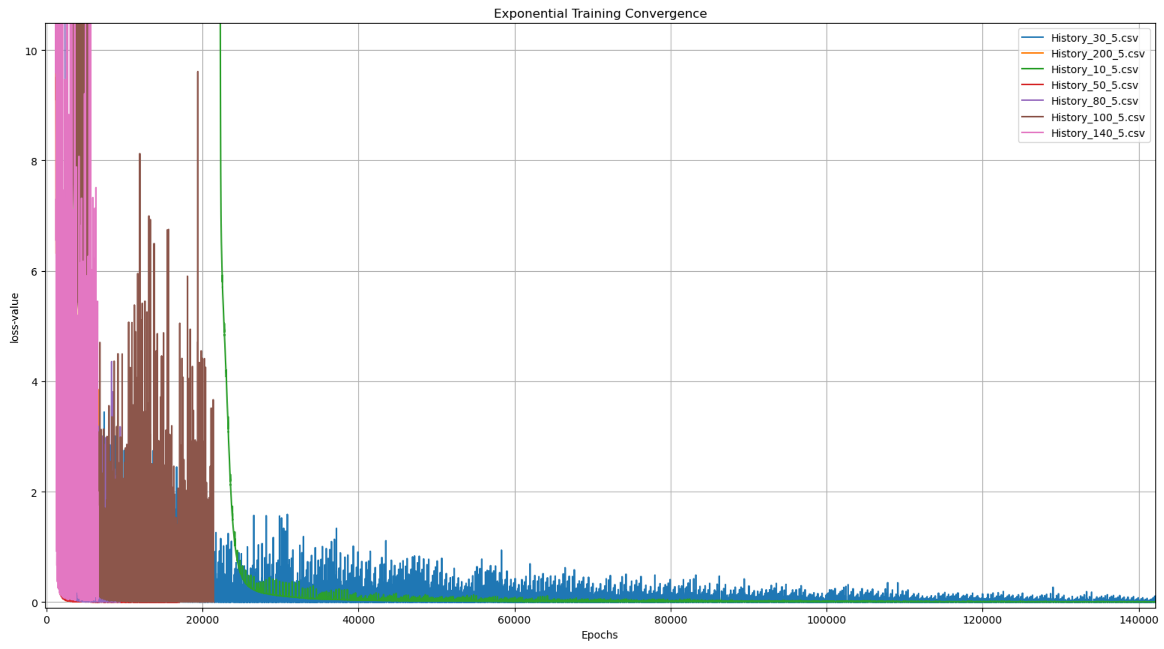
0.4 Results

0.4.1 Exponential LR

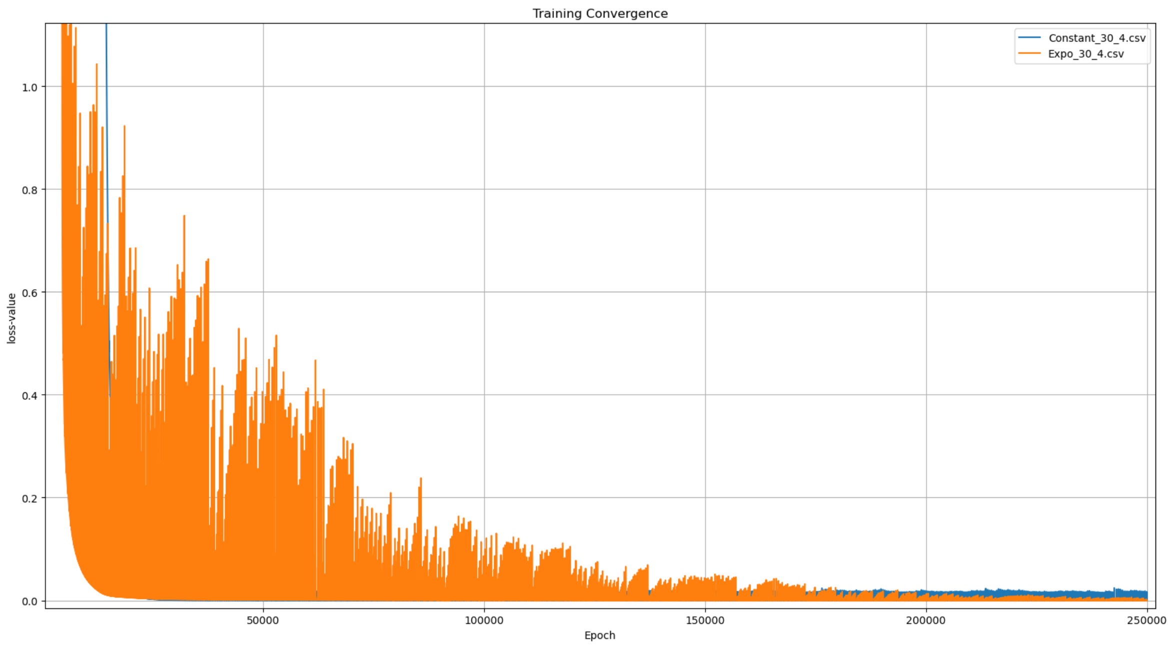
Plots Across Constant Width



Plots Across Constant Depth

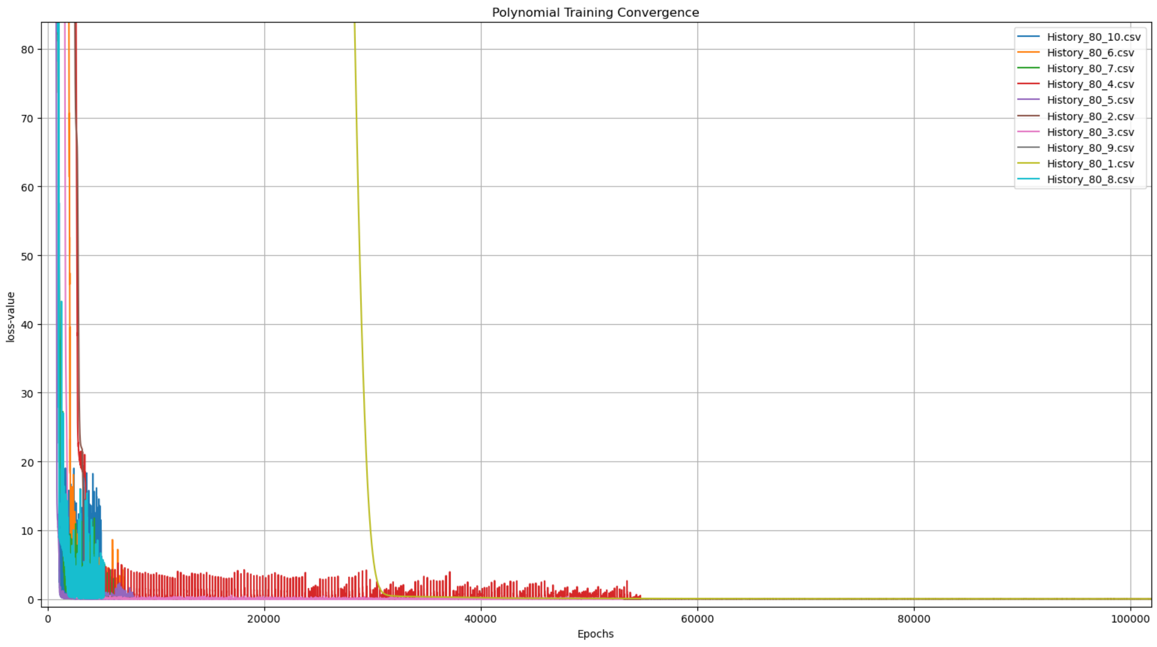


Comparison to Fixed LR

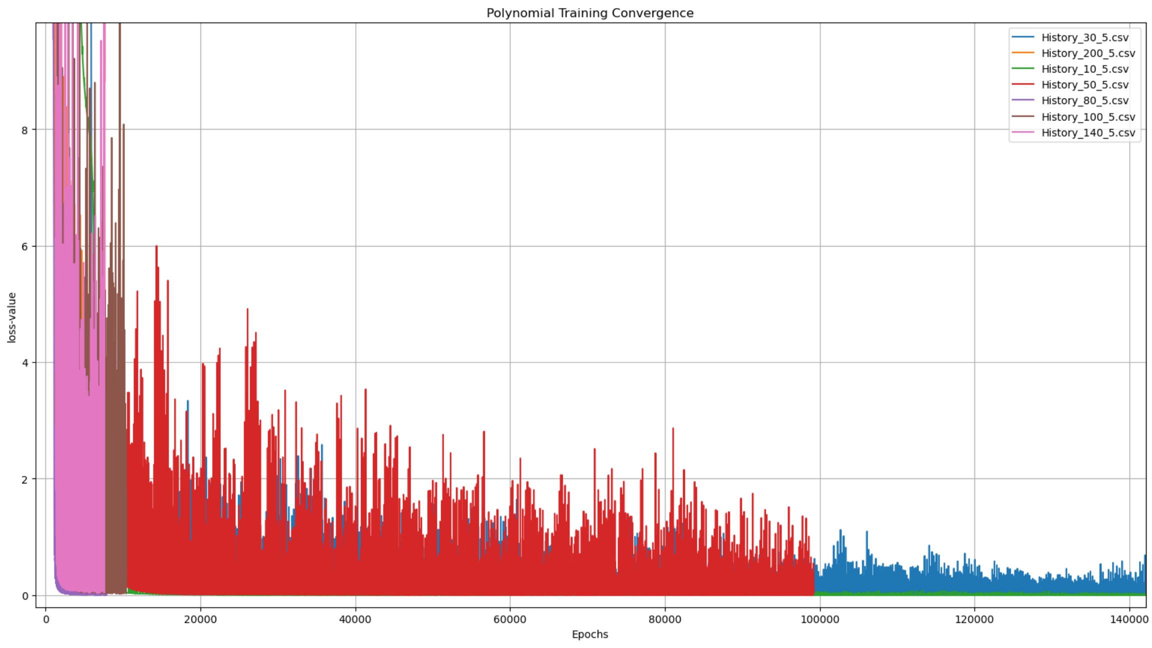


0.4.2 Polynomial LR

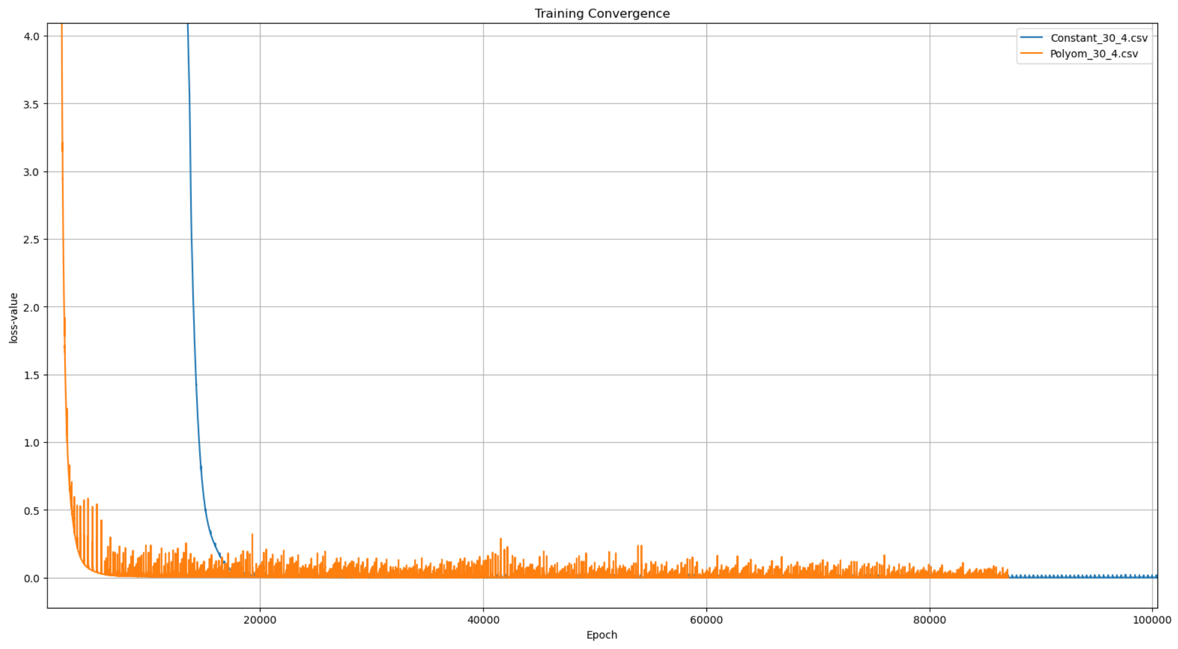
Plots Across Constant Width



Plots Across Constant Depth

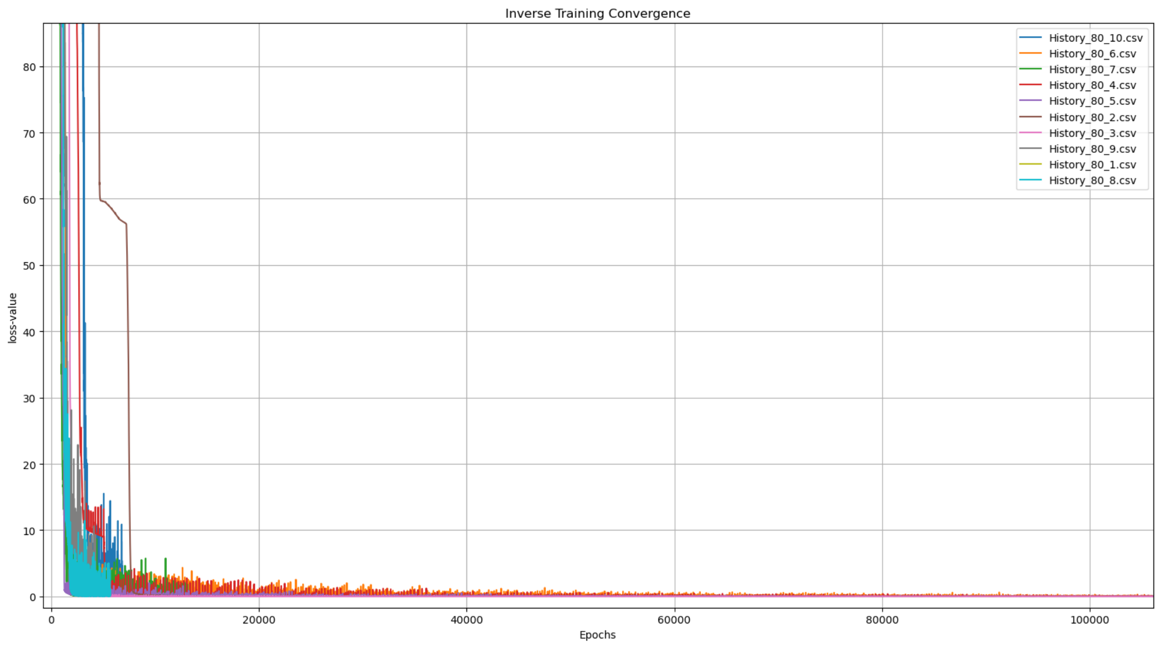


Comparison to Fixed LR

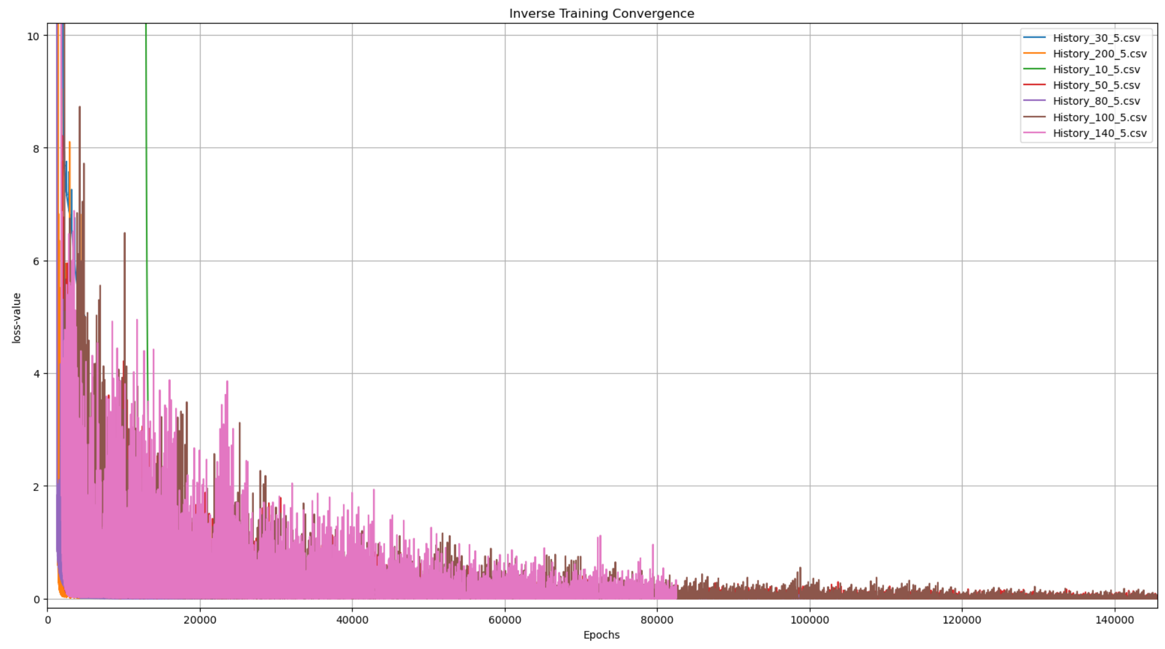


0.4.3 Inverse LR

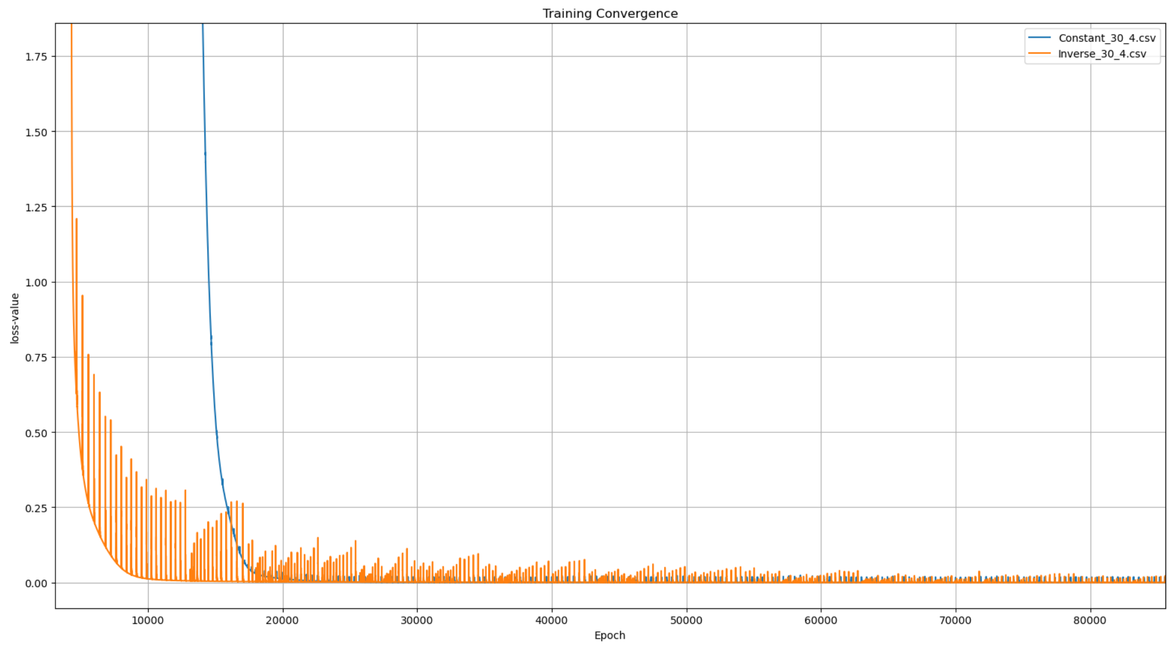
Plots Across Constant Width



Plots Across Constant Depth



Comparison to Fixed LR



0.5 Analysis

Broadly speaking, having the learning rate scheduler set-up to have a higher initial learning rate and decay to a smaller value improves the training covergence while simulatenously reduces stability issues for some of the scheduler. From observing the plots, it seems that the inverse scheduler is the most stable, followed by the exponential scheduler and finally followed by the polynomial scheduler. The polynomial scheduler has an order $n = 1$. Therefore, the polynomial scheduler is a linar function. The results can be explained well, a scheduler that ”more quickly” reduces the scheduling rate is numerically more stable compared to an algorithm that didn’t.

0.6 Further Development

This investigation establishes the relationship between learning rate, stability, and trainig convergence. Since the governing equation used may change, it is more useful not to make learning rate dependent on training iterations but instead have learning rate dependent on the magnitude of the weight gradients.

More investigations need to be conducted in the numerical instability of the algorithm. It seems that the numerical instability is periodic. Why is this the case?