Edit Share

GAN training methods

exploring the effects of different hyperparameters on GAN training

Ariel Keslassy

Interactive link: https://wandb.ai/arielkes/GAN-training/reports/GAN-training-methods--VmlldzoyNzkzODQz?accessToken=b12yacrjoo7p593w09g68mpkmk5pl9zua5zbnwqs6fepkpxne03ds5kiqpru8pvy

General setting:

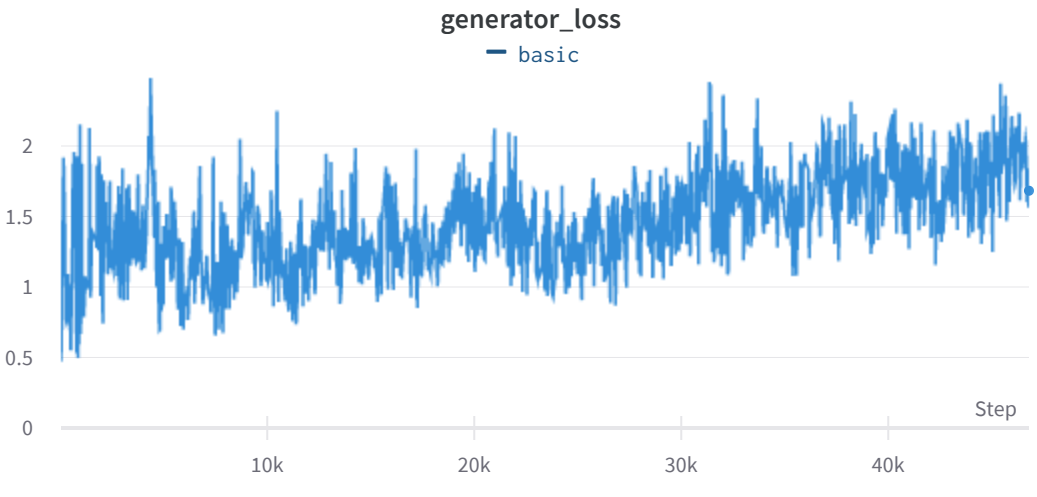
- Task: generating hand-written digits from MNIST dataset.
- Models:
  - Generator:
    - input: noise vectors of size (batch_size, 100)
    - output: images of size (batch_size, 1, 28, 28)
    - architecture: alternating linear, activation and dropout layers
  - Discriminator:
    - input: images of size (batch_size, 1, 28, 28)
    - output: prediction vectors of size (batch_size, 1)
    - architecture: alternating linear, activation and dropout layers
  - Hyperparameters:
    - gen_layers: number of linear layers in the generator

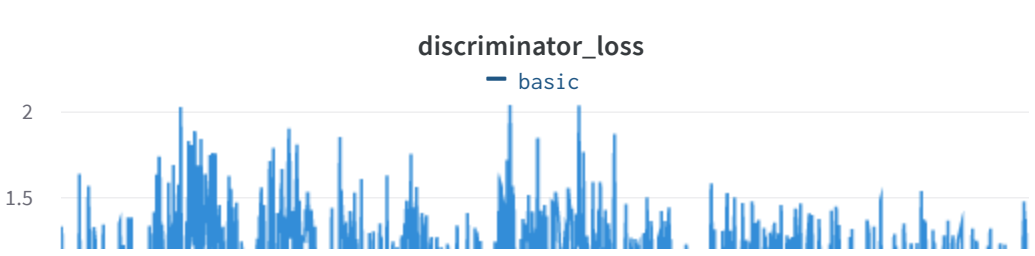
- dis_layers: number of linear layers in the discriminator
- activation: type of activation function, one in [relu, leaky_relu, elu, gelu, relu6]
- gen_dropout: dropout probability in the generator
- dis_dropout: dropout probability in the discriminator
- gen_lr: generator learning rate
- dis_lr: discriminator learning rate

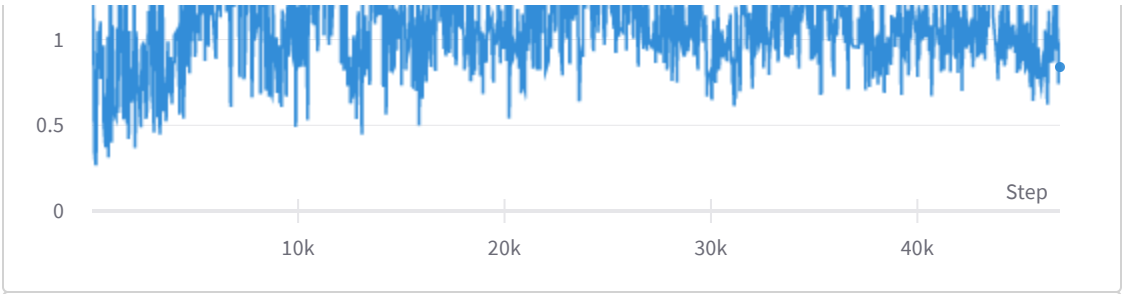
Experiments:

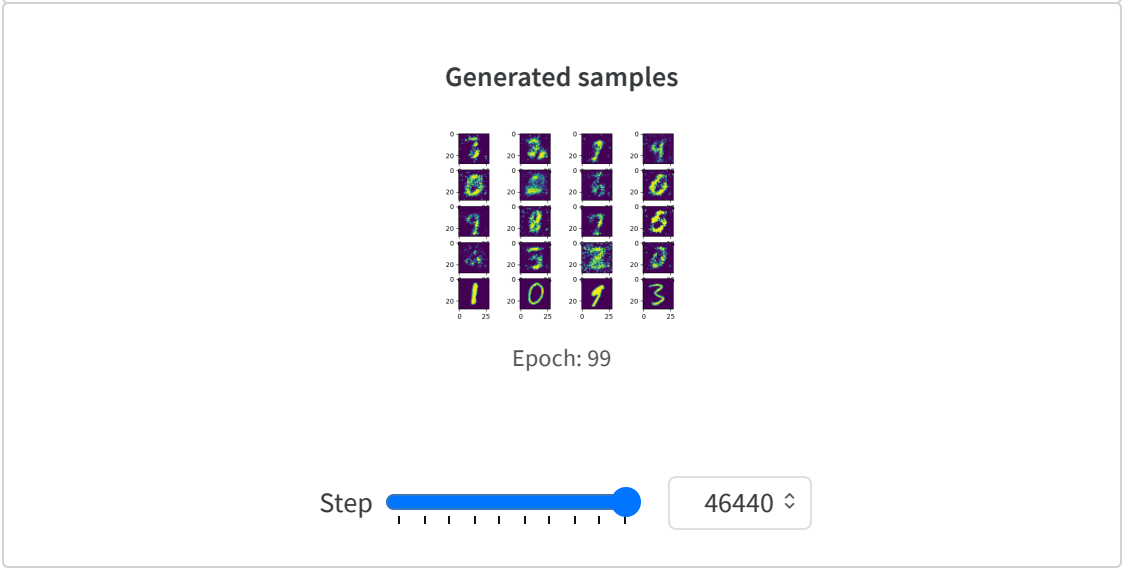
Basic experiment for comparison

- configuration: gen_layers = 3, dis_layers = 2, activation = ReLU, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results (the bottom row of the samples are real data, for comparison):



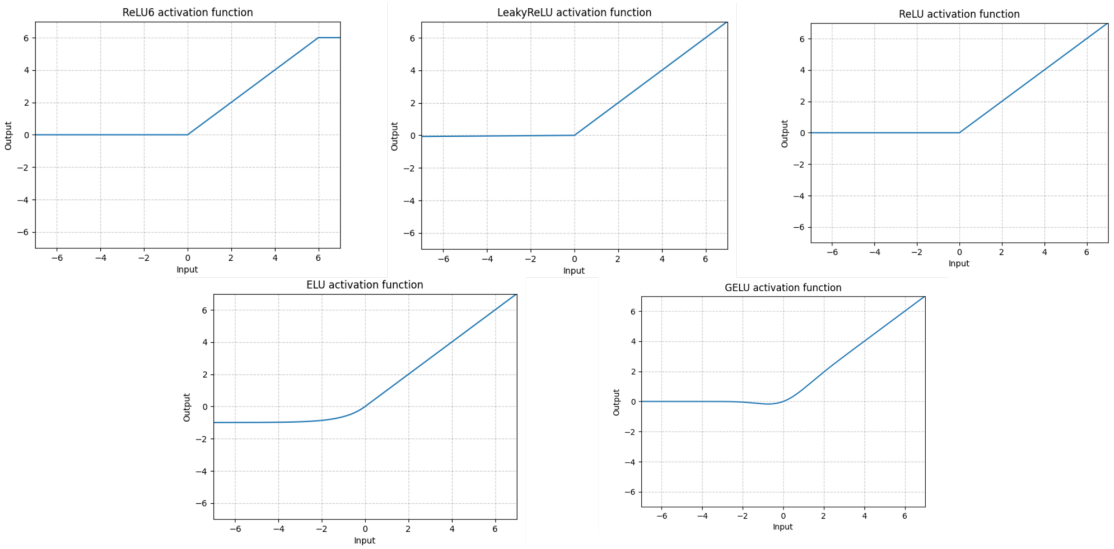




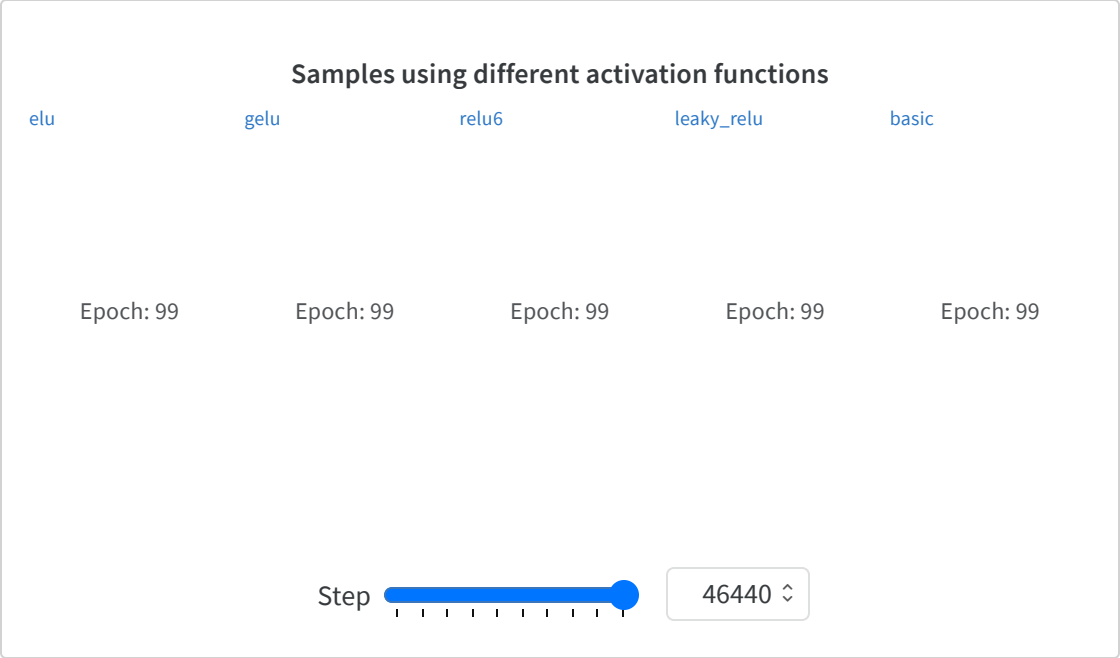


Activation functions:

- configuration: gen_layers = 3, dis_layers = 2, activation = every option, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- the activation functions used:



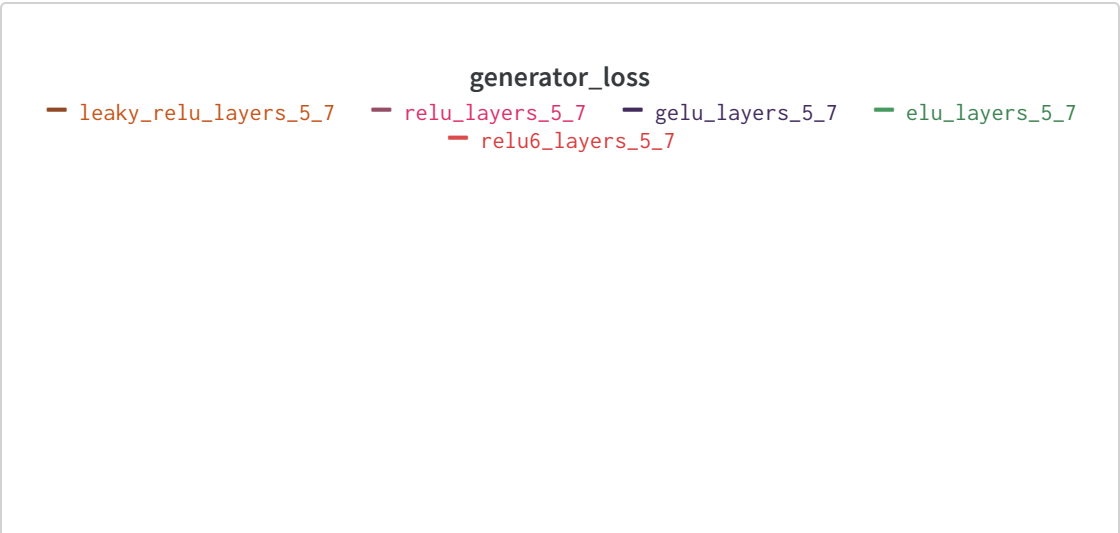
- results:



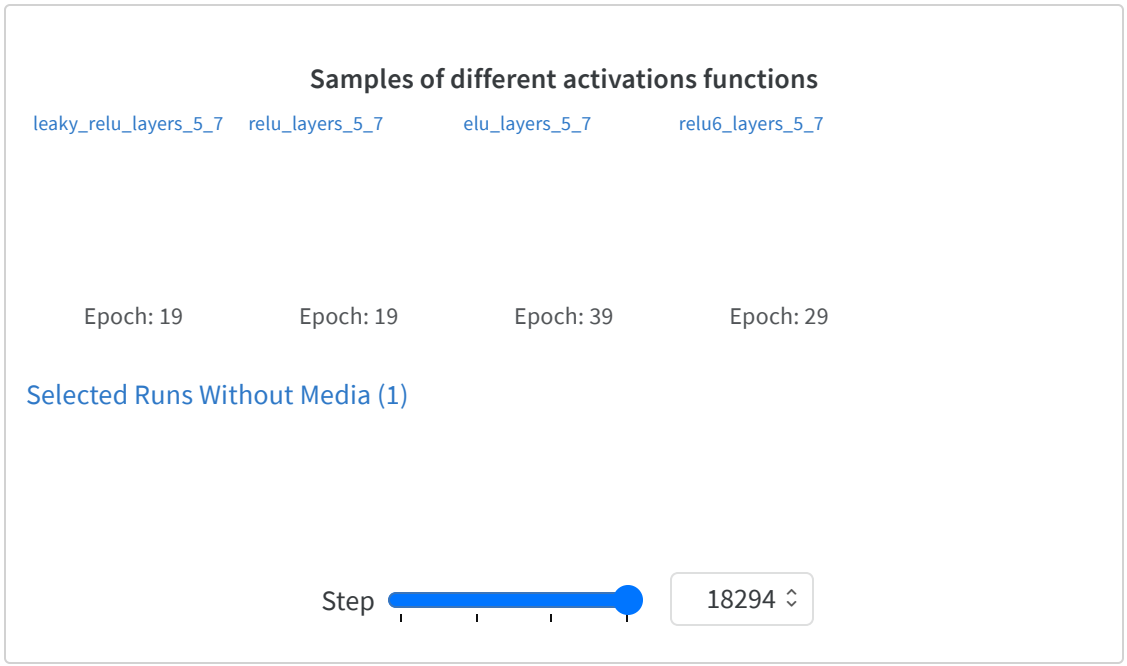
Based on our samples, ELU seems to have the best results, and GELU and Leaky ReLU also performed well. The worst performance was from ReLU.

Activation functions with larger networks:

- configuration: gen_layers = 7, dis_layers = 5, activation = every option, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:



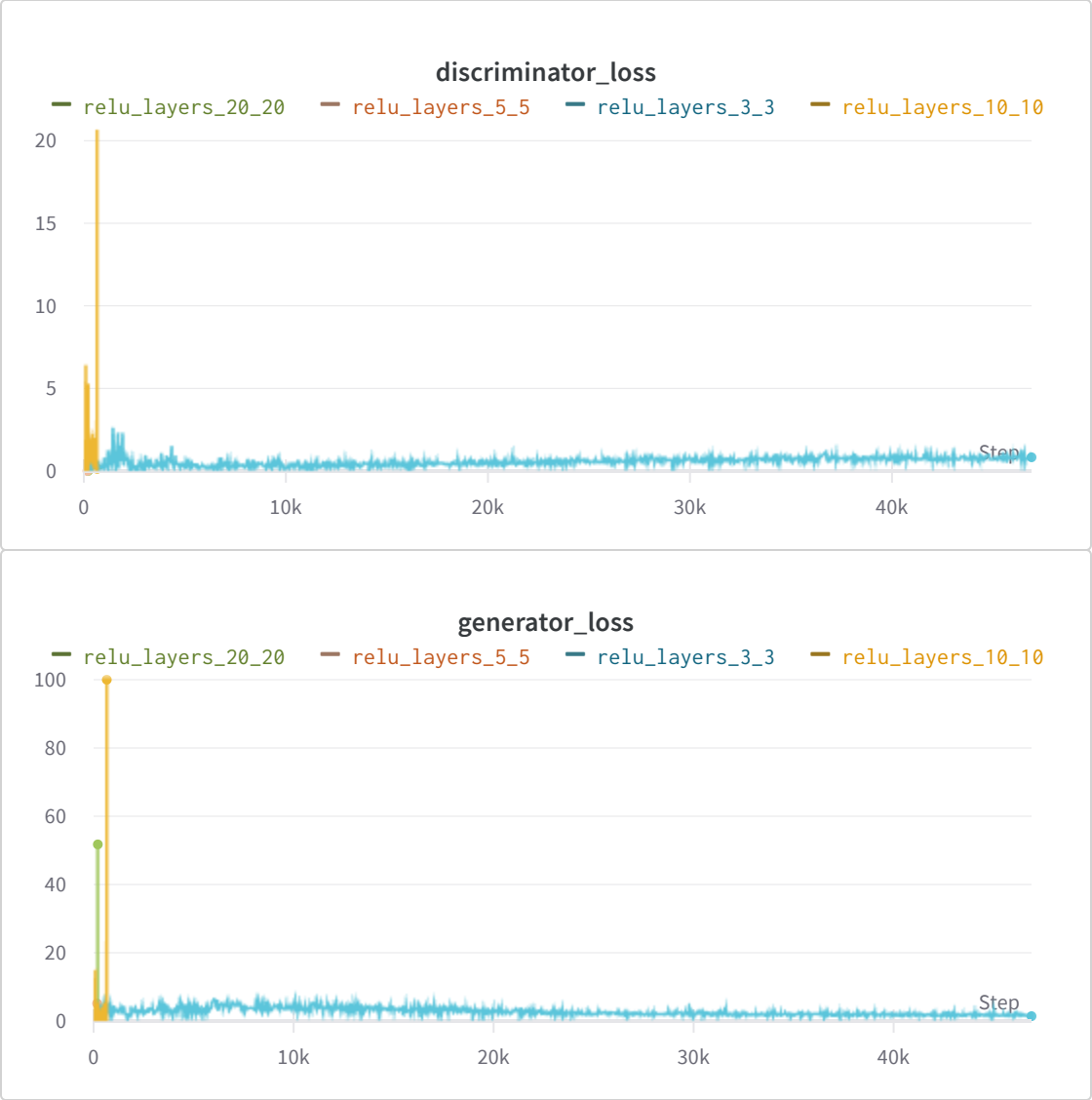
These runs failed as the discriminators' loss quickly approached zero, giving very small gradients to the generator and thus not letting the generator improve (vanishing gradient problem). This result emphasizes these models' sensitivity to the number of layers, and especially the difference between the discriminator's and the generator's depths.



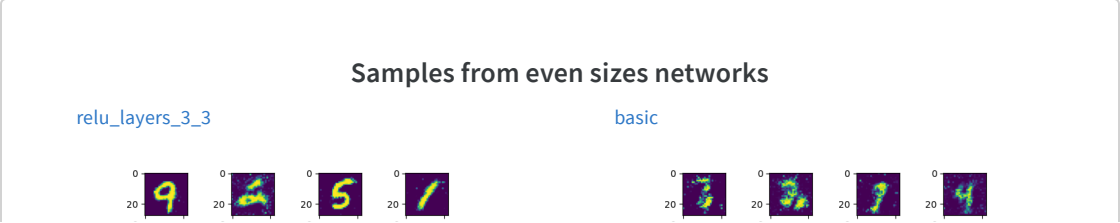
Other network sizes:

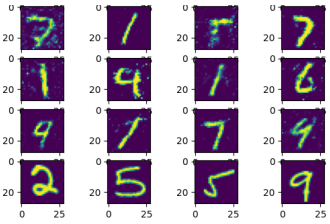
Even sizes

- configuration: gen_layers $\in [3, 5, 10, 20]$, dis_layers = $\in [3, 5, 10, 20]$, activation = relu, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:

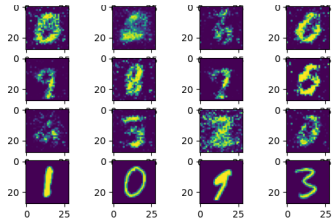


The smallest model (3, 3) was the only one able to converge. Qualitative analysis (below) shows that it performed better the the basic model with uneven depths (2, 3)





Epoch: 99

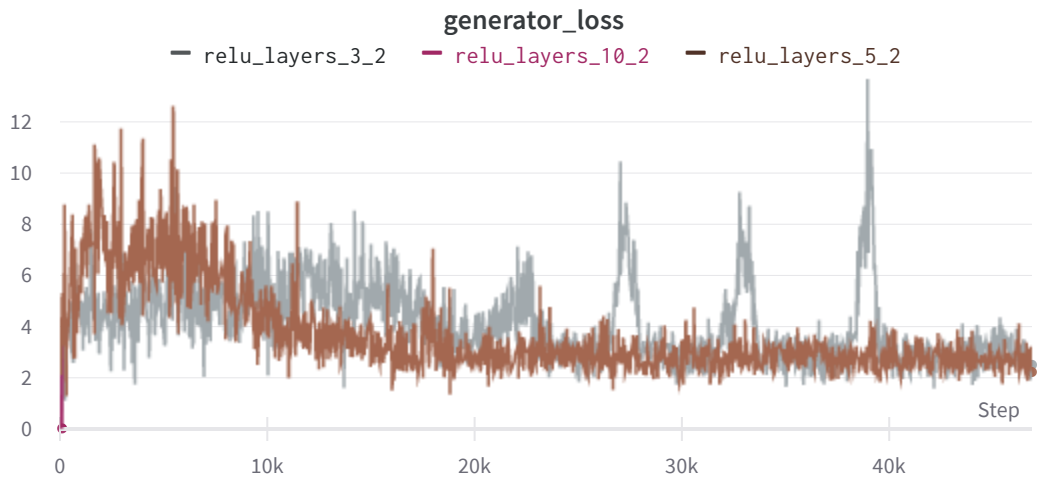


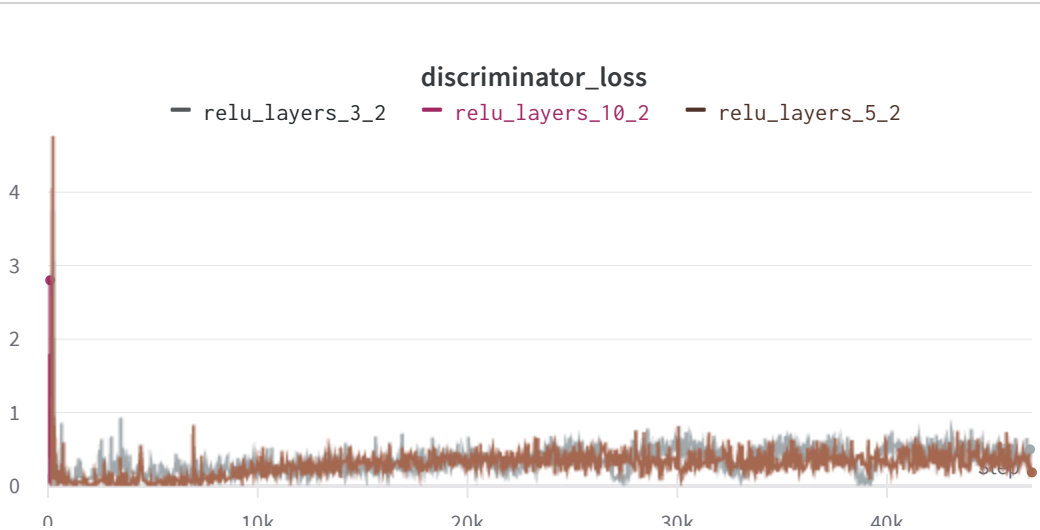
Epoch: 99

Step 46440

Uneven sizes - larger discriminator

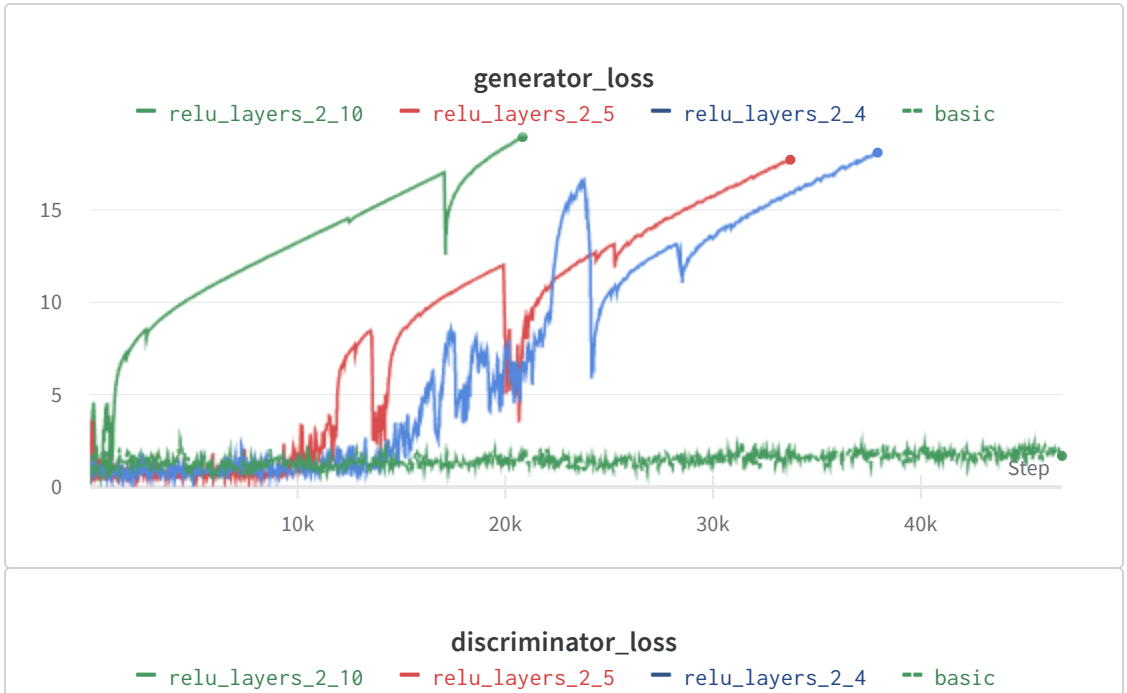
- configuration: gen_layers = 2, dis_layers = $\in [3, 5, 10]$, activation = relu, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:

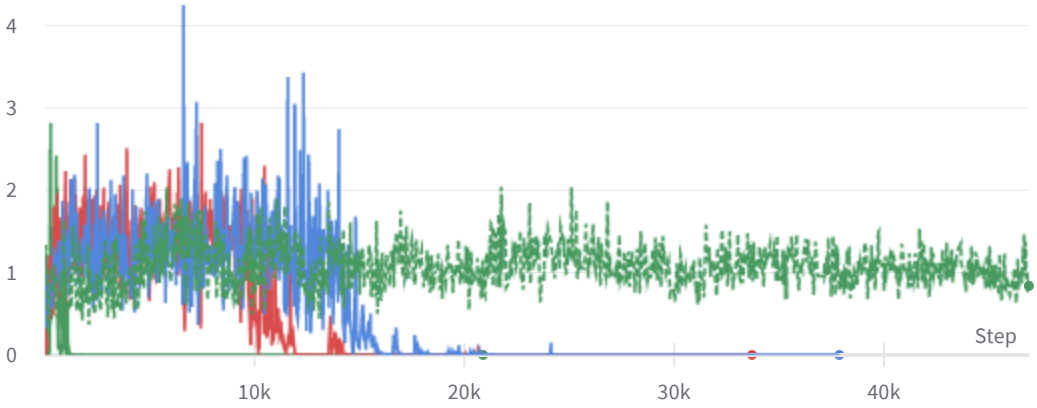


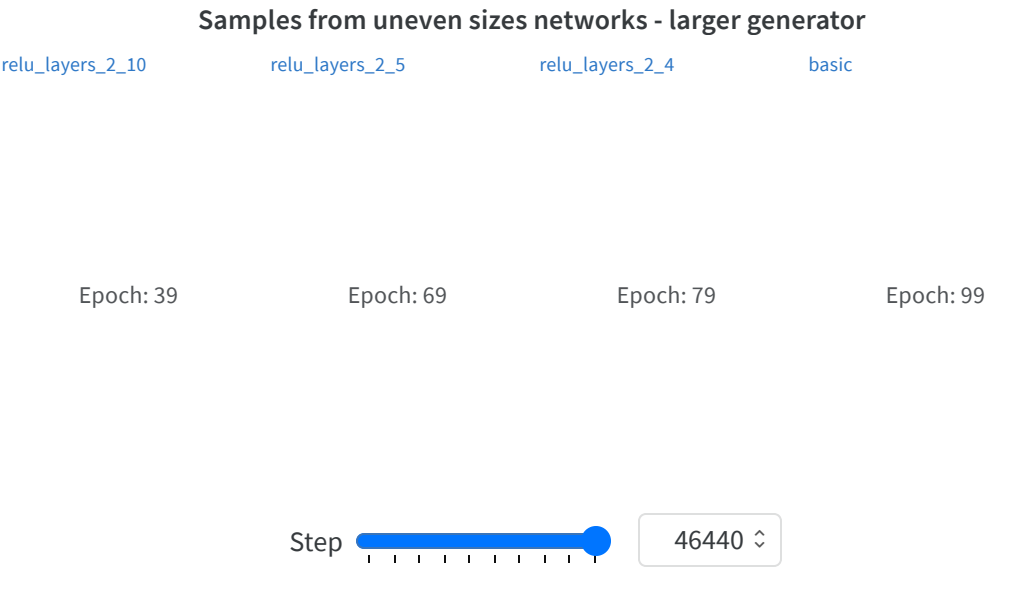


Uneven sizes - larger generator

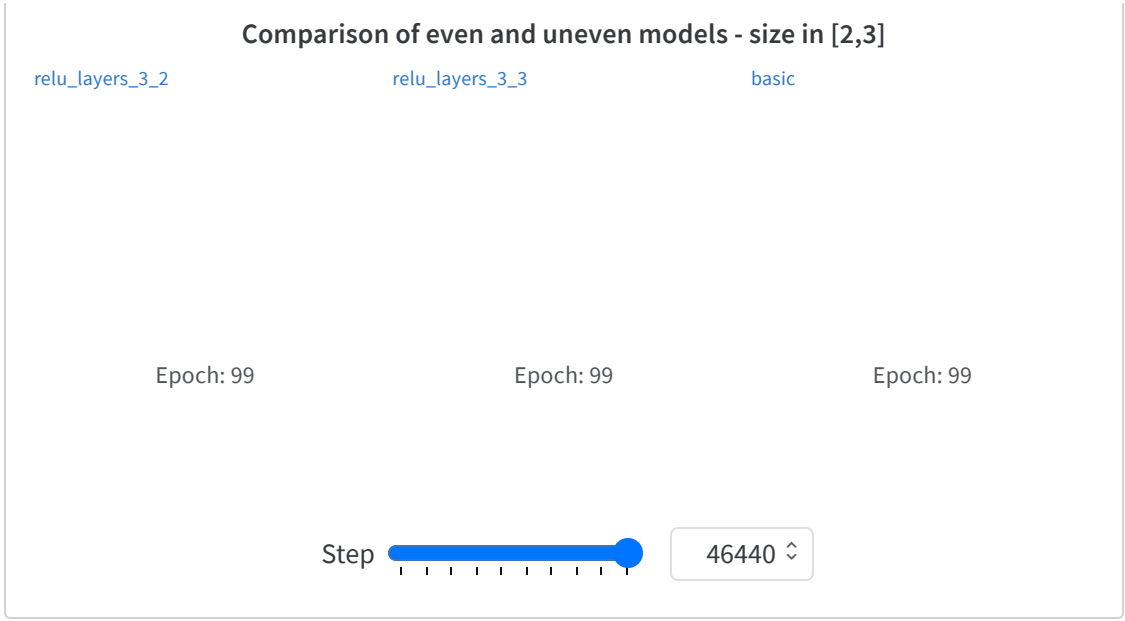
- configuration: gen_layers $\in [3, 4, 5, 10]$, dis_layers = 2, activation = relu, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:

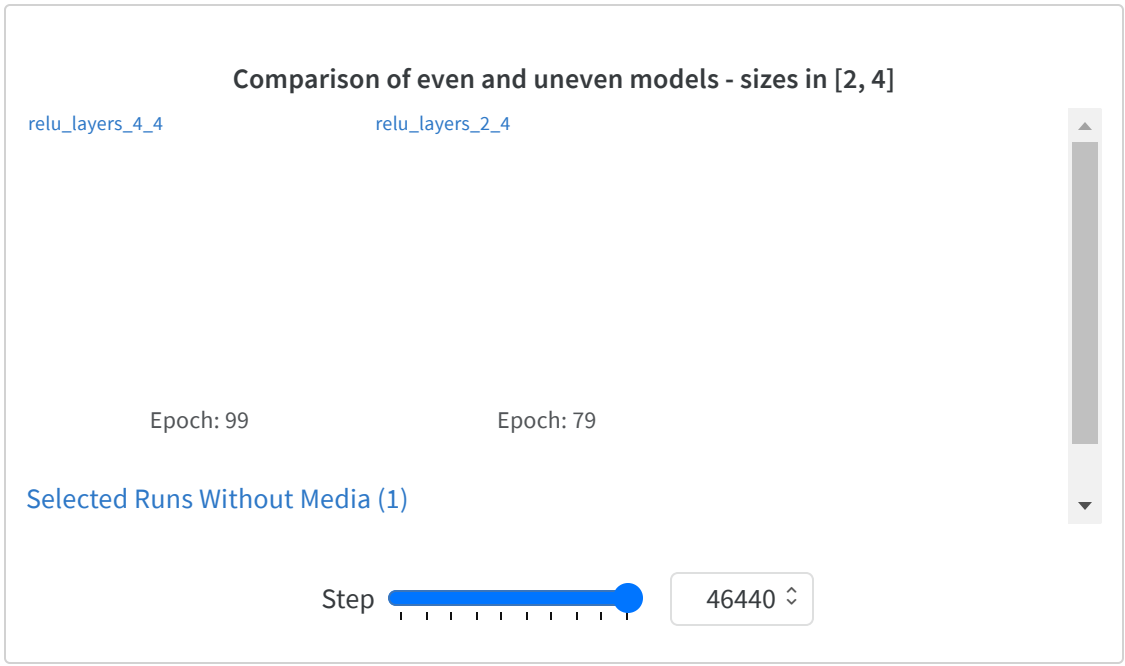






Again, too large models weren't able to converge.
A model with larger discriminator (5, 2) got high quality results, but was mostly able to generate a single digit (1), a problem named "mode collapse".
Generally, among the uneven models, it seems that models with larger discriminator were able to get better performance overall comparing to the models with larger generator
We can now compare the even and uneven models with the approximately same sizes:





Our examples suggests that even sized models perform better

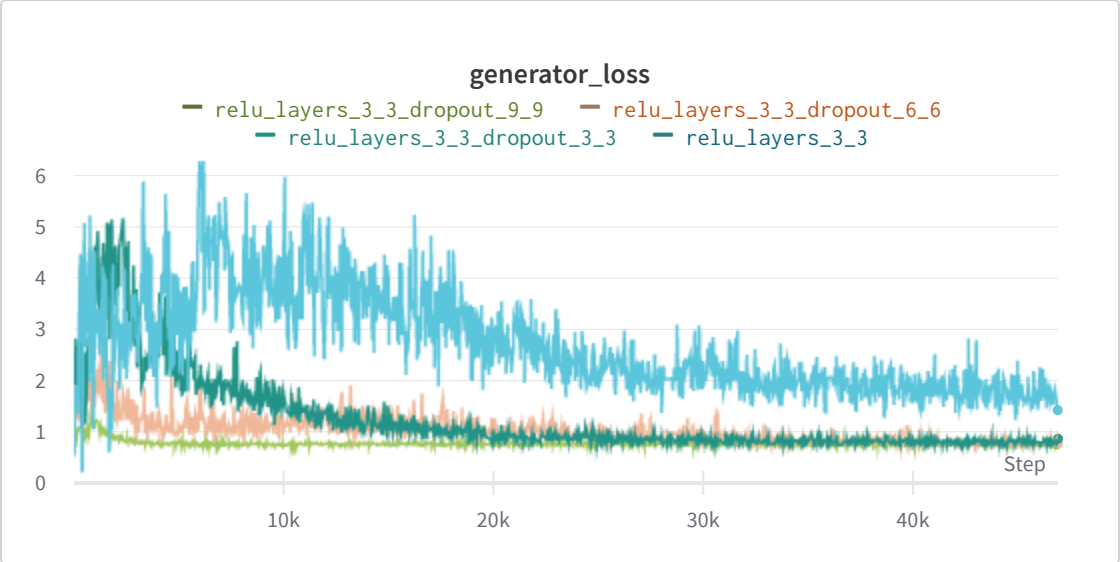
Using dropout:

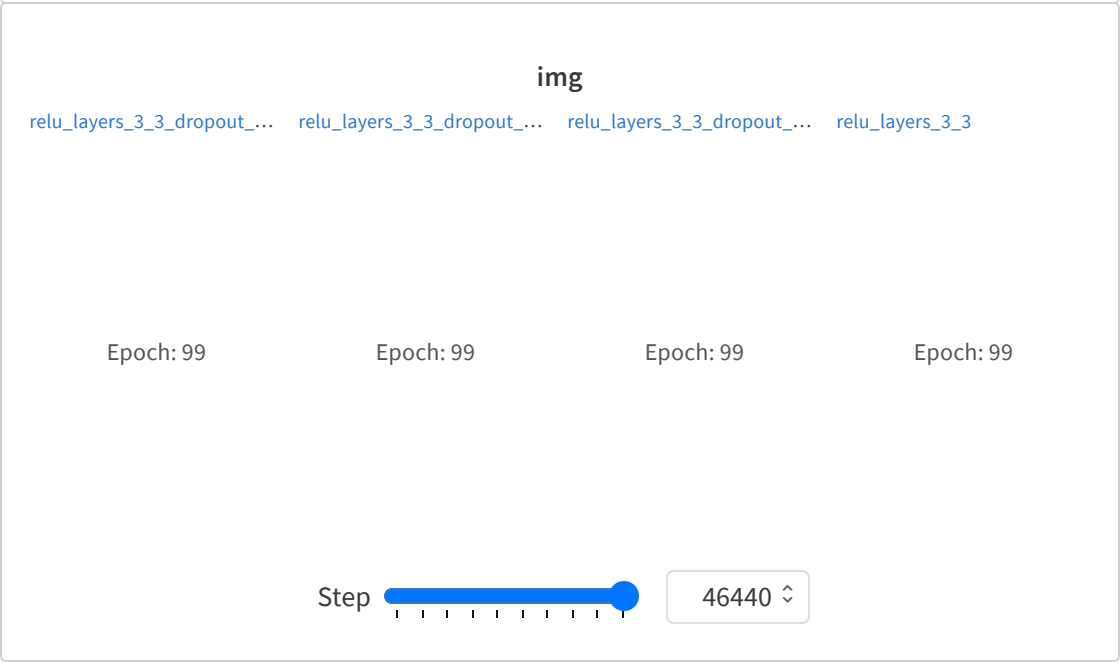
Different dropout probability - even sizes (3, 3)

- configuration: gen_layers = 3, dis_layers = 3, activation = relu, gen_dropout $\in [0.0, 0.3, 0.6, 0.9]$, dis_dropout $\in [0.0, 0.3, 0.6, 0.9]$, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs

- results:

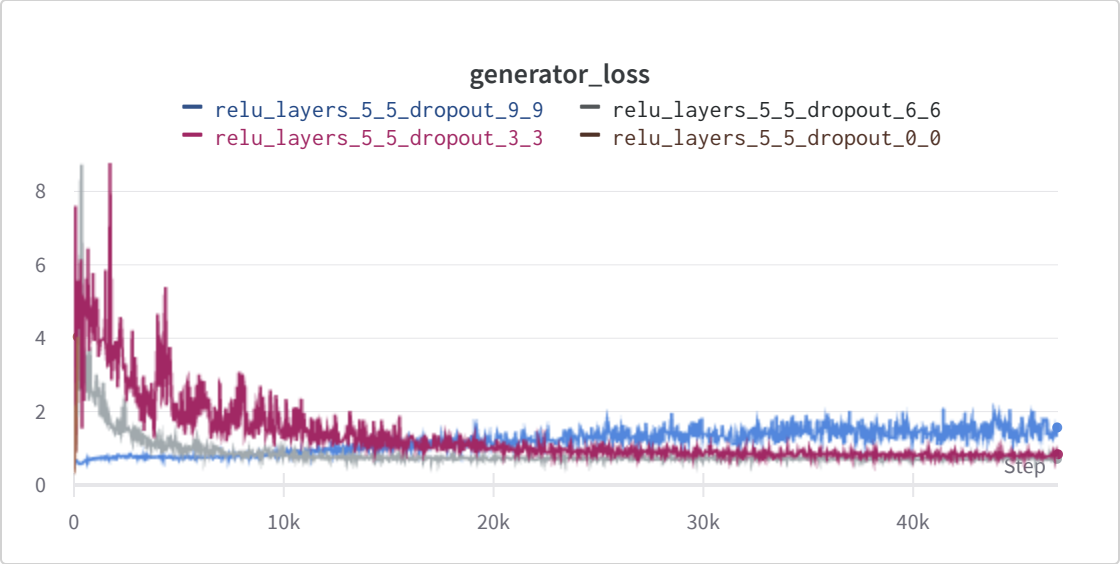

discriminator_loss
relu_layers_3_3_dropout_9_9
relu_layers_3_3_dropout_6_6
relu_layers_3_3_dropout_3_3
relu_layers_3_3
2.5
2
1.5
1
0.5
0
Step
10k
20k
30k
40k

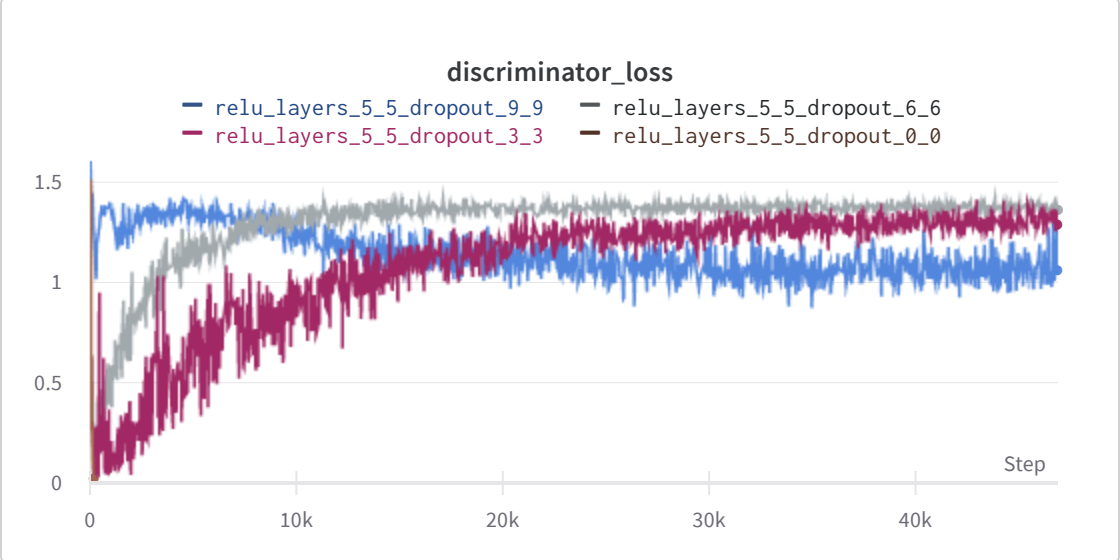

generator_loss
relu_layers_3_3_dropout_9_9
relu_layers_3_3_dropout_6_6
relu_layers_3_3_dropout_3_3
relu_layers_3_3
6
5
4
3
2
1
0
Step
10k
20k
30k
40k

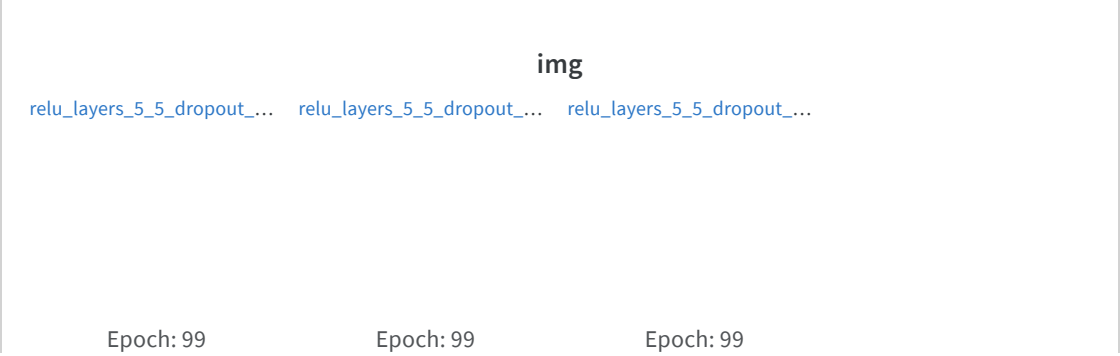

img
relu_layers_3_3_dropout_...
relu_layers_3_3_dropout_...
relu_layers_3_3_dropout_...
relu_layers_3_3
Epoch: 99
Epoch: 99
Epoch: 99
Epoch: 99
Step
46440

Different dropout probability - even sizes (5, 5)

- configuration: gen_layers = 3, dis_layers = 3, activation = relu, gen_dropout $\in [0.0, 0.3, 0.6, 0.9]$, dis_dropout $\in [0.0, 0.3, 0.6, 0.9]$, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:





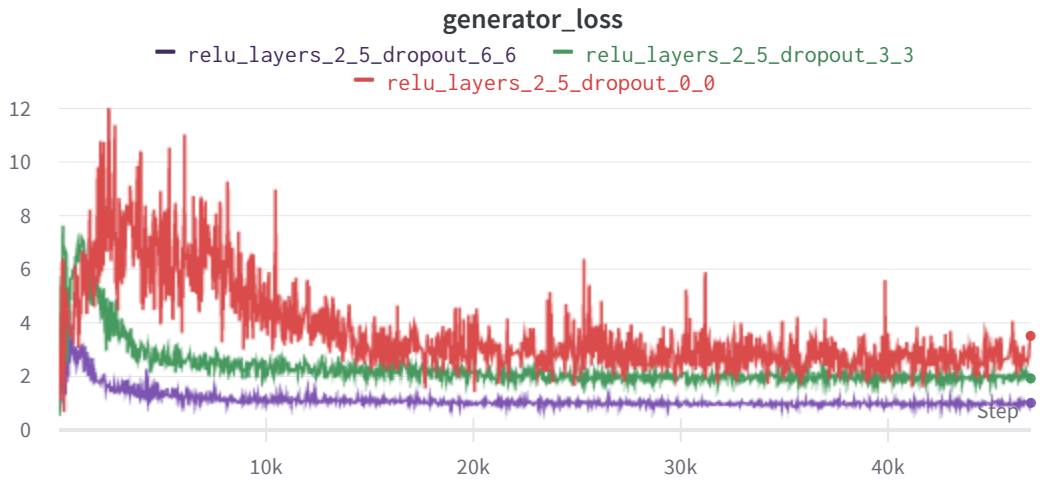


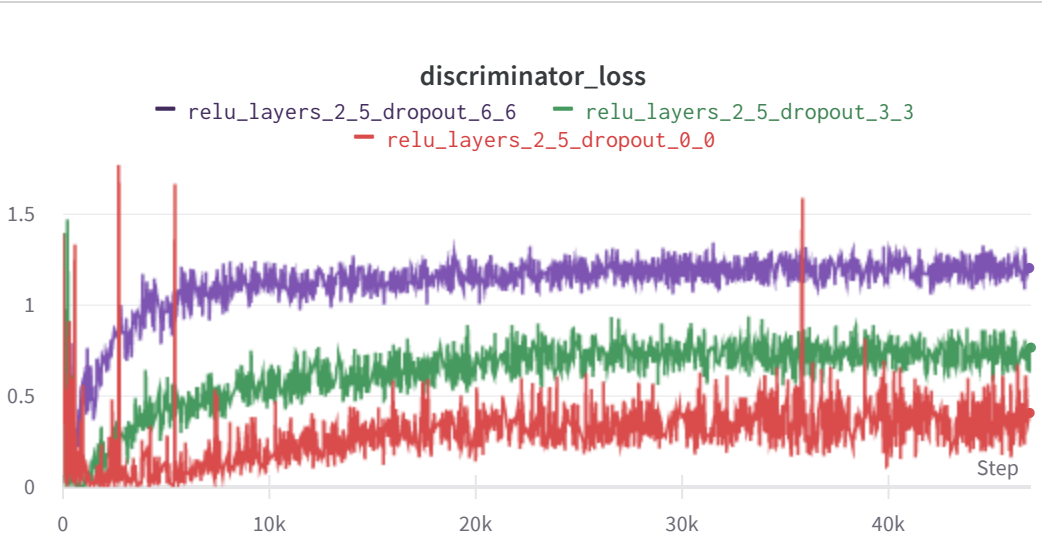
Selected Runs Without Media (1)

Step 46440

Different dropout probability - uneven sizes (2, 5)

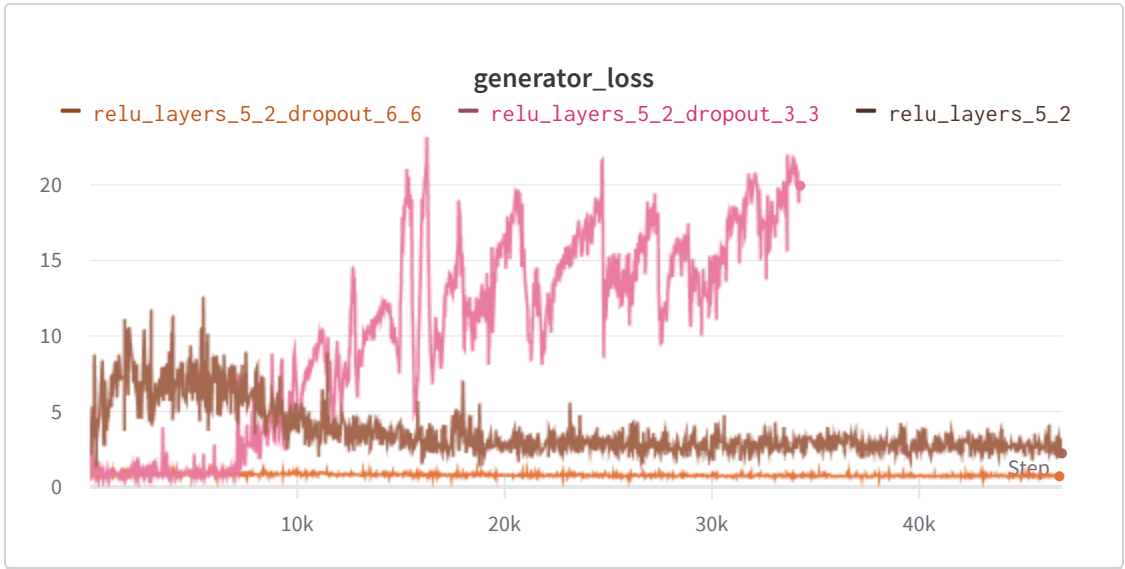
- configuration: gen_layers = 2, dis_layers = 5, activation = relu, gen_dropout $\in [0.0, 0.3, 0.6]$, dis_dropout $\in [0.0, 0.3, 0.6]$, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:

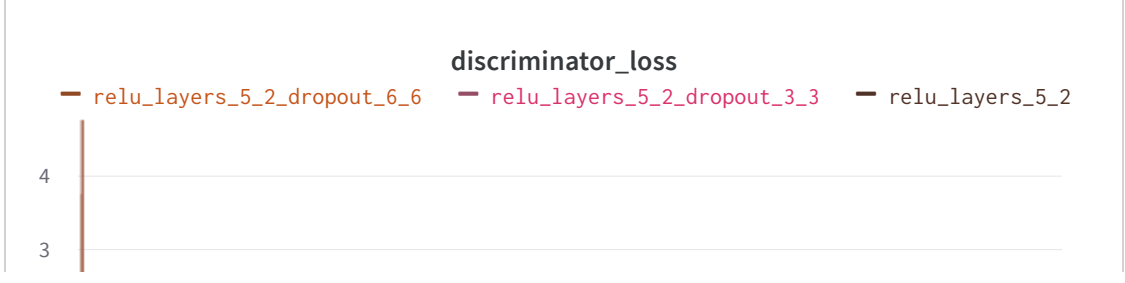


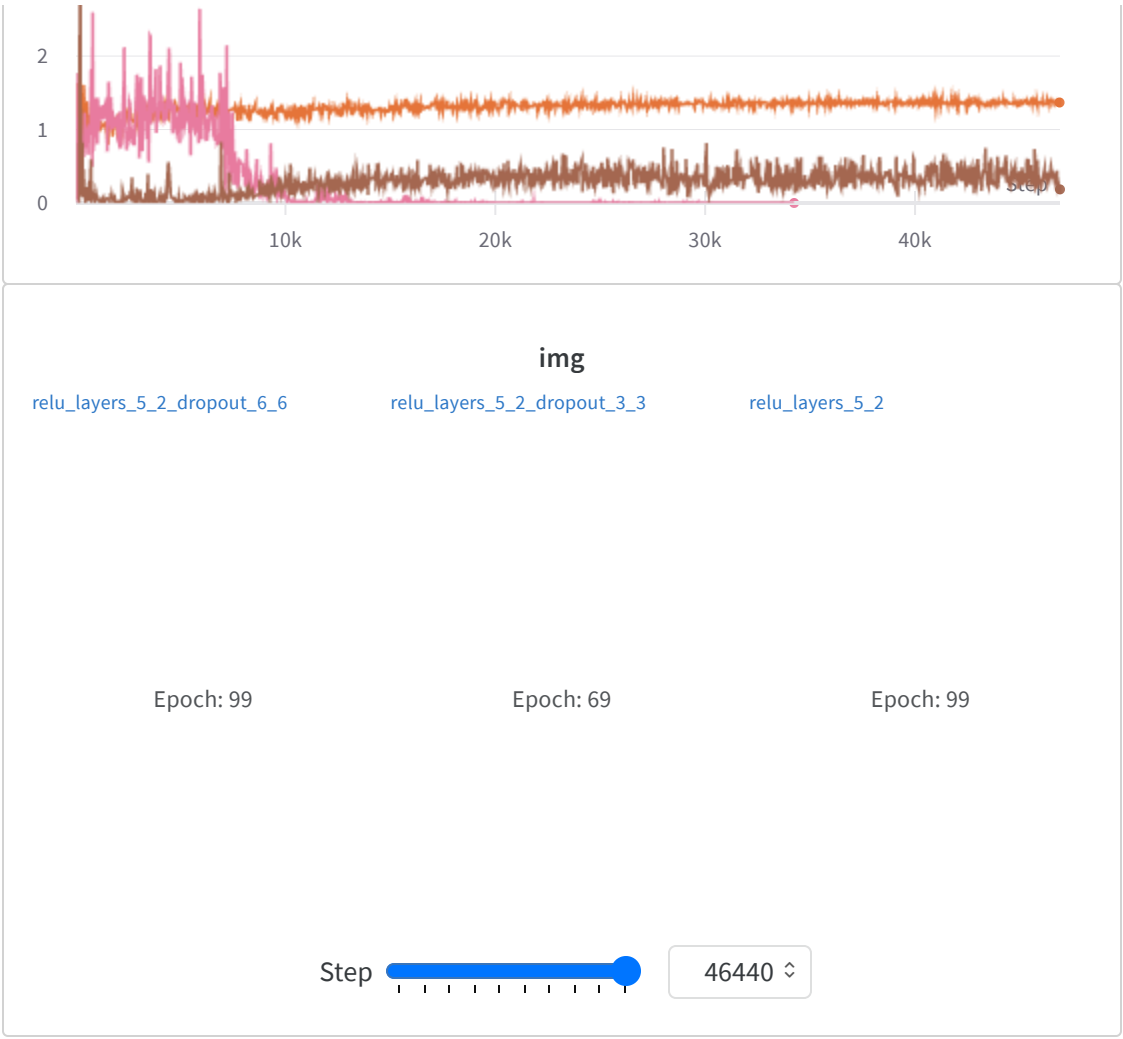


Different dropout probability - uneven sizes (5, 2)

- configuration: gen_layers = 5, dis_layers = 2, activation = relu, gen_dropout $\in [0.0, 0.3, 0.6]$, dis_dropout $\in [0.0, 0.3, 0.6]$, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:







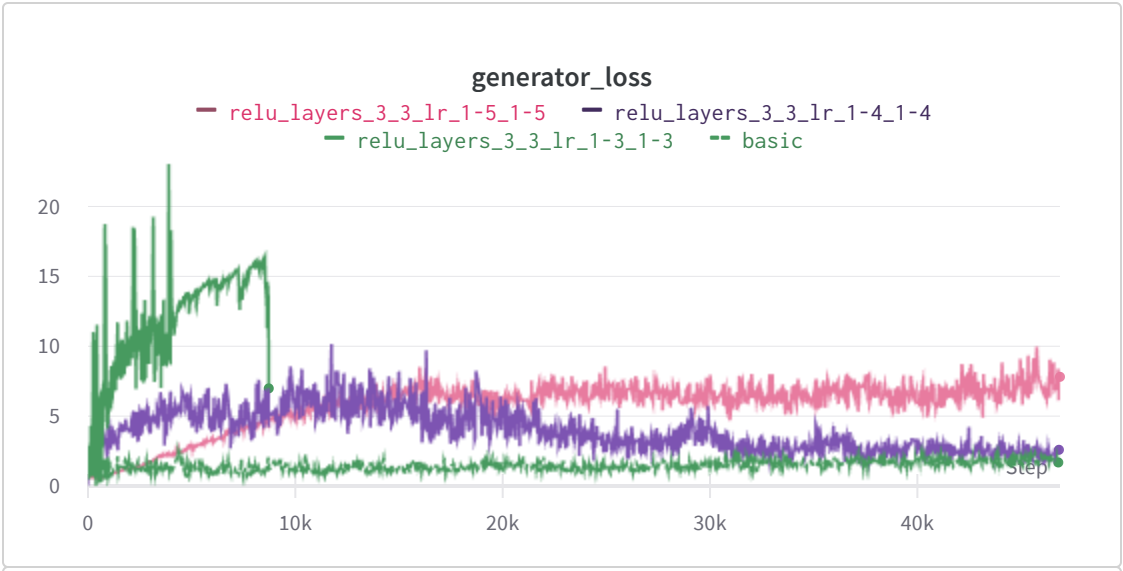
- Generally, dropout help architectures to converge, and may even reach lower loss.
  specifically, it seems that higher dropout probability helps to avoid "mode collapse".
- Regarding the quality of the generated data, higher dropout seems to produce lower quality results.

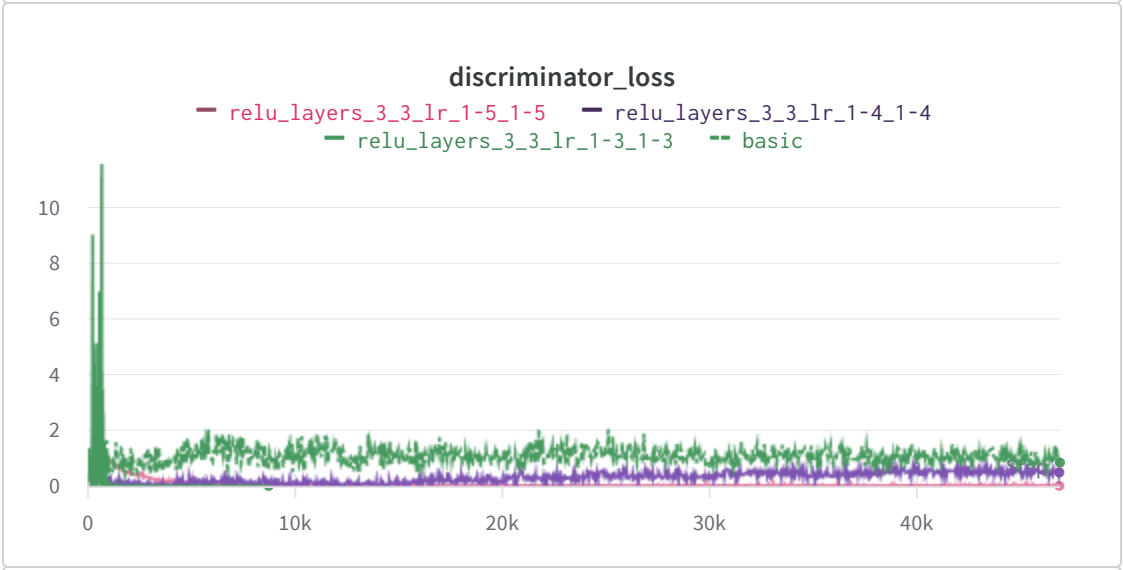
Learning rate:

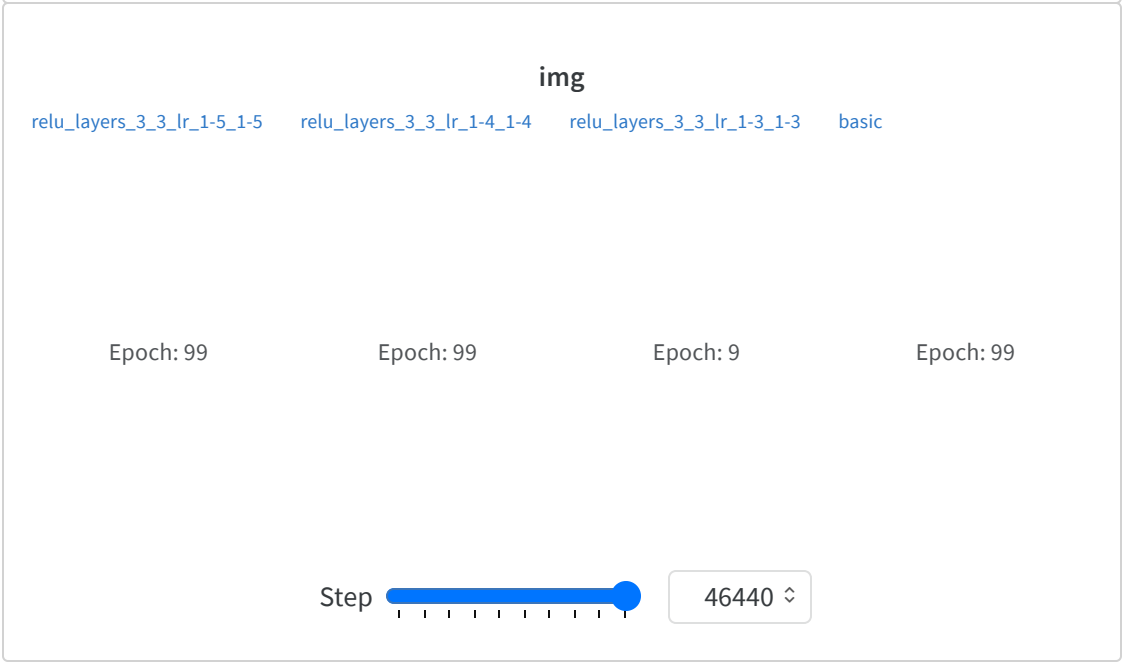
Same learning rates for the discriminator and the generator

- configuration: gen_layers = 3, dis_layers = 3, activation = relu, gen_dropout =0.0, dis_dropout = 0.0, gen_lr $\in [0.001, 0.0001, 0.0002, 0.00001]$, dis_lr $\in [0.001, 0.0001, 0.0002, 0.00001]$, 100 epochs

- results:

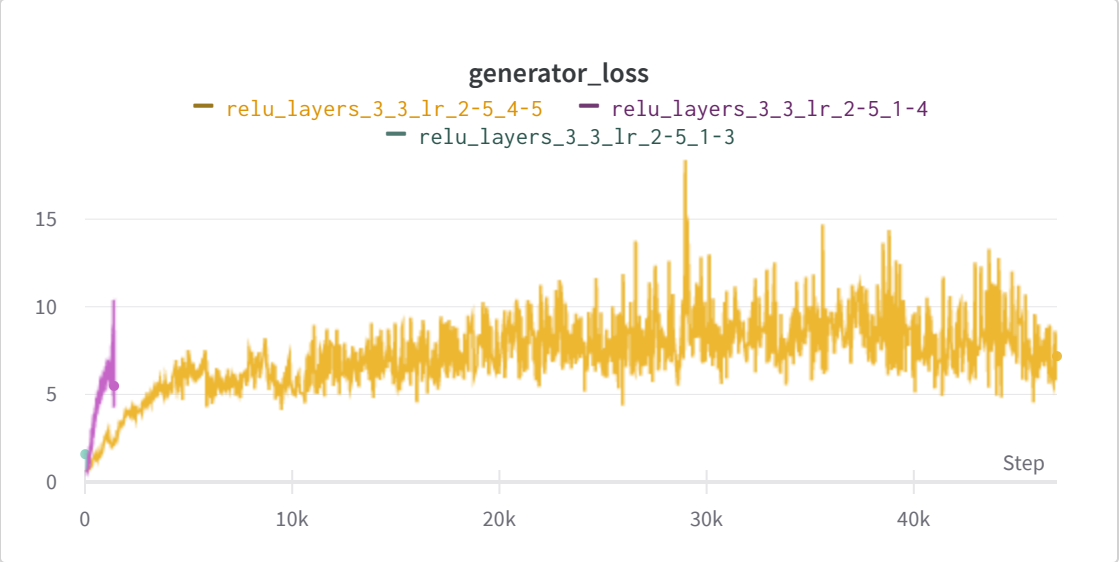

generator_loss
relu_layers_3_3_lr_1-5_1-5
relu_layers_3_3_lr_1-4_1-4
relu_layers_3_3_lr_1-3_1-3
basic
20
15
10
5
0
0
10k
20k
30k
40k
Step

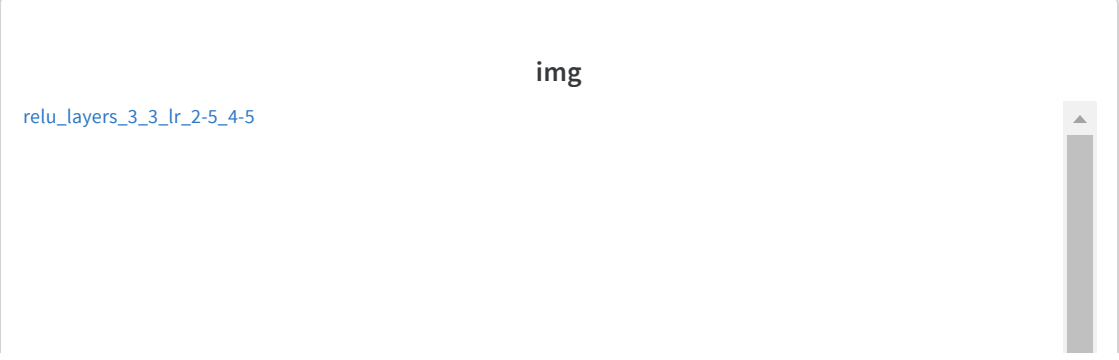

discriminator_loss
relu_layers_3_3_lr_1-5_1-5
relu_layers_3_3_lr_1-4_1-4
relu_layers_3_3_lr_1-3_1-3
basic
10
8
6
4
2
0
0
10k
20k
30k
40k
Step

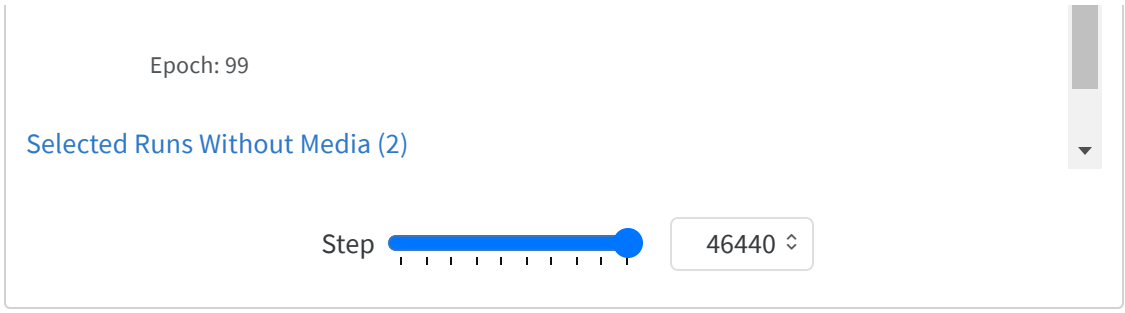

img
relu_layers_3_3_lr_1-5_1-5
relu_layers_3_3_lr_1-4_1-4
relu_layers_3_3_lr_1-3_1-3
basic
Epoch: 99
Epoch: 99
Epoch: 9
Epoch: 99
Step
46440

Different learning rates for the discriminator and the generator: Greater discriminator learning rate

- configuration: gen_layers = 3, dis_layers = 3, activation = relu, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr = 0.00002, dis_lr $\in [0.001, 0.0001, 0.00004]$, 100 epochs
- results:

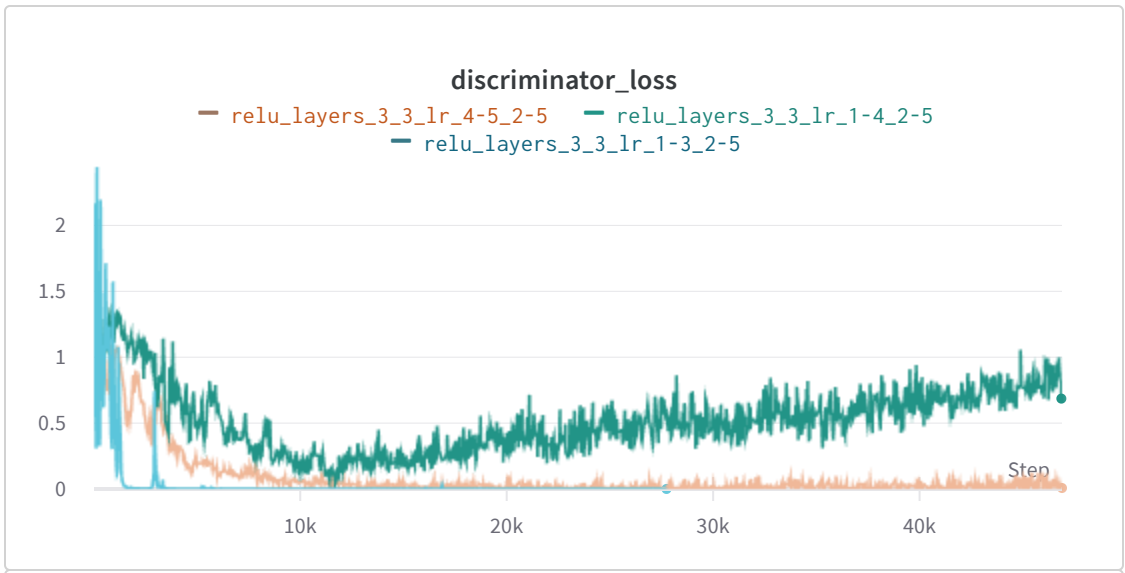


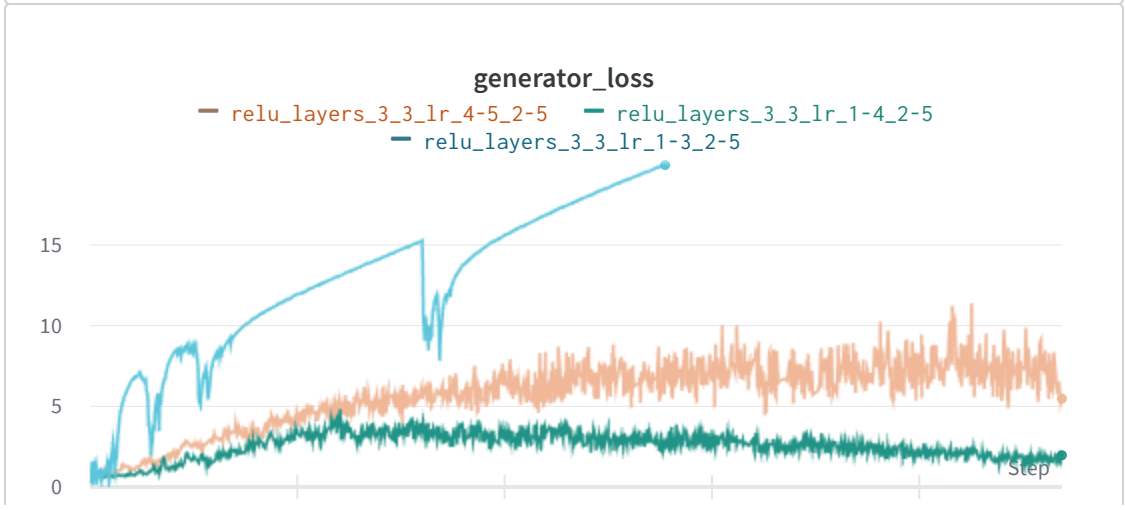




Different learning rates for the discriminator and the generator: Greater generator learning rate

- configuration: gen_layers = 3, dis_layers = 3, activation = relu, gen_dropout = 0.0, dis_dropout = 0.0, gen_lr $\in [0.001, 0.0001, 0.00004]$, dis_lr = 0.00002, 100 epochs
- results:

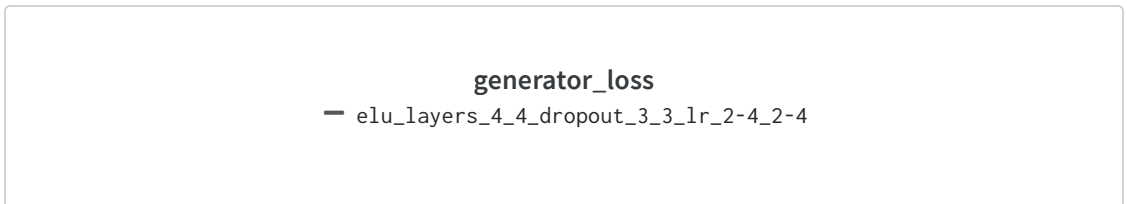


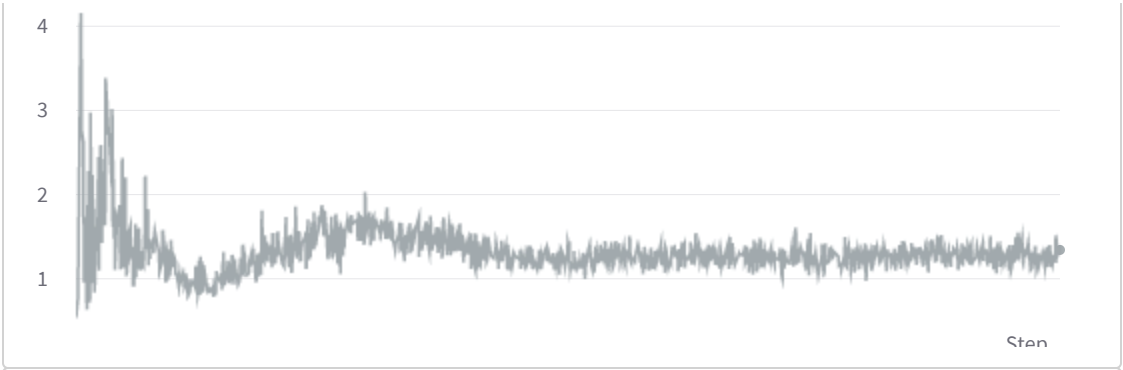


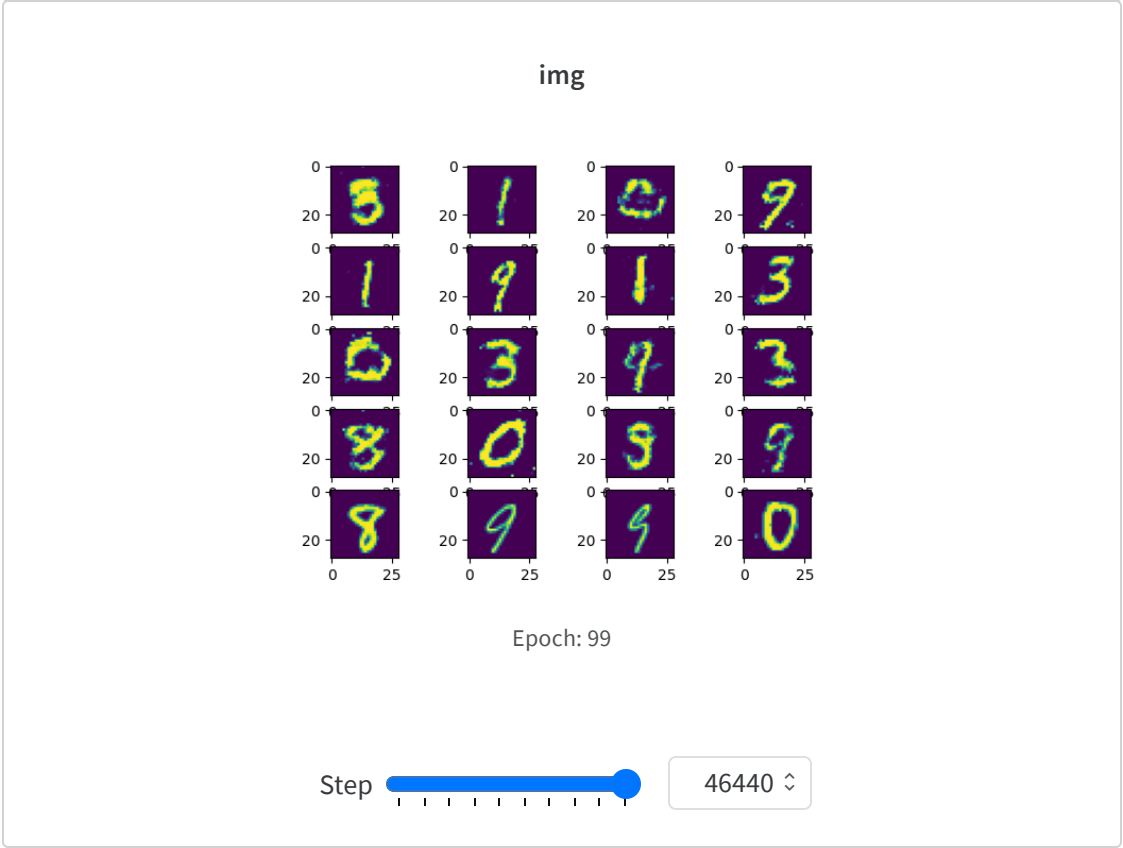
From our experiments, it seems that magnitude of 10^{-4} works the best. Generally, different learning rates for the discriminator and the generator performed worse than the same learning rate for both networks (unlike common practice, see TTUR). Among the different learning rates for both networks, having a larger learning rate in the generator worked better than the opposite setting

Combining methods based on previous experiments:

- configuration: gen_layers = 4, dis_layers = 4, activation = elu, gen_dropout = 0.3, dis_dropout = 0.3, gen_lr = 0.0002, dis_lr = 0.0002, 100 epochs
- results:







Created with ❤️ on Weights & Biases.

https://wandb.ai/arielkes/GAN-training/reports/GAN-training-methods--VmlldzoyNzkzODQz