1a. As a benchmark, I first gathered the settings used in the previous tests, which were provided in the repo.

The initial settings were set to:

```
model_args = {
        "number_of_hidden_layers": 1,
        "input_size": 2,
        "hidden_size": 10,
        "activation": torch.nn.Sigmoid(),
}
trainer_args = {
        "optimizer": torch.optim.SGD,
        "lr": 0.1,
}
```

This setting did pretty badly. Remembering from the first HW that adjusting the activation function had enabled me to improve significantly, my first change was setting the activation function to torch.nn.ReLU() . Quite interestingly, I kept getting nan as the output of the loss function. I tried with torch.nn.LeakyReLU() but kept getting nans. I tried debugging but couldn't figure out the reason for this. So, I reverted to the initial setup. Then, I first played around with the model size: number_of_hidden_layers, input_size, which did not affect loss much. I played around with the learning rate but still could not get it to perform significantly better. In the meantime, I adjusted the initialization of weights; I initially used torch.nn.init.uniform_, I changed that to torch.nn.init.normal_, which did show a significant improvement. I also tried adjusting the bias terms, which also led to significant improvement. I was not, however, able to pass tests. I was quite far from it. While I was staring at the parameters, thinking why using ReLU was leading to nan, it came to my mind that in the first HW, the optimizer choice was Adam, and I did not ever have the issue of nan. I changed the optimizer to Adam and adjusted the activation function to ReLU; boom, the loss got much better. This change definitely had the most impact by far. I could still not pass the tests, but I was getting there. After this adjustment, I played around with the learning rate and model architecture (dimensions); for the final touch, I had to decrease the model size by setting the hidden_layer size to 8. (while I'm writing this, I tried setting the hidden layer size to 4 and got 100% acc; it was around 99.9% with 8).

I did some research trying to understand why using SGD led to nans, whereas using Adam worked well. I checked the algorithms for both optimizers; well, SGD, I knew from lectures that Adam is a little more involved. I did not go into much detail on how Adam works, but after reading some comments, I realized that Adam uses estimates to adapt the learning rate, preventing large updates that can cause exploding gradients. This is what was likely happening when I was using ReLU with SGD, with a learning rate of 0.05, which is considered high. I tried adjusting the learning rate, but when I used a learning rate that was 100,000 times smaller, ReLU with SGD actually converged faster than ReLU with Adam. When I used a small learning rate for Adam, it did much worse. I think this leads me to an intuition that slightly larger learning rates are better for Adam and smaller ones for SGD.

Overall, changing the activation function and the learning rate, and somewhat the size/architecture (I think up to a point size does not matter too much, like having a layer of size 4 or 8 did not matter too much. They both did well) had the most effect which what I have also realized in HW1. However, I tend to believe that the most important thing is to know the context of your problem really well and also the tools that you are using. The tools should not be used as black boxes. For example, for the first HW understanding the idea of pretraining led me to get 100% validation accuracy. Here, I was dramatically failing using SGD, then I realized the differences between SGD and Adam, and that led me to adjust the lr for SGD, which eventually improved the model as it converged faster.

1b. I tried exactly the same parameters that I have used for the adddataset, initially I failed the tests, but then I realized that the code provided in the save_model_without_deleting module was using quite a few examples(100) and epochs(100). I increased the number of examples to 1000 and epochs to 200, and I was able to pass the test with the parameters I used for add dataset. Interestingly, however, with the optimizer set to SGD and a learning rate of 0.0000005, the add dataset model was able to converge faster(it reached 100% after 120 epochs). The same setting for the multiply dataset did much worse. It was not able to pass the test.