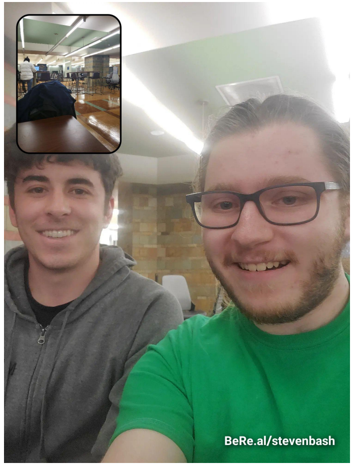
E96 MNIST Final Project: Single Digit Image Classifier



Steven Bash and Jacob Apoian

Design Process

- Started without the skeleton
- We figured out
  - How to import libraries
  - How to create neural network class and forward functions
  - How to import the training/testing data
  - How to create our train/test sets
  - Instantiate model and train it
  - We decided to use Adam as the optimizer

And we achieved 10% accuracy 🥳

Design Process

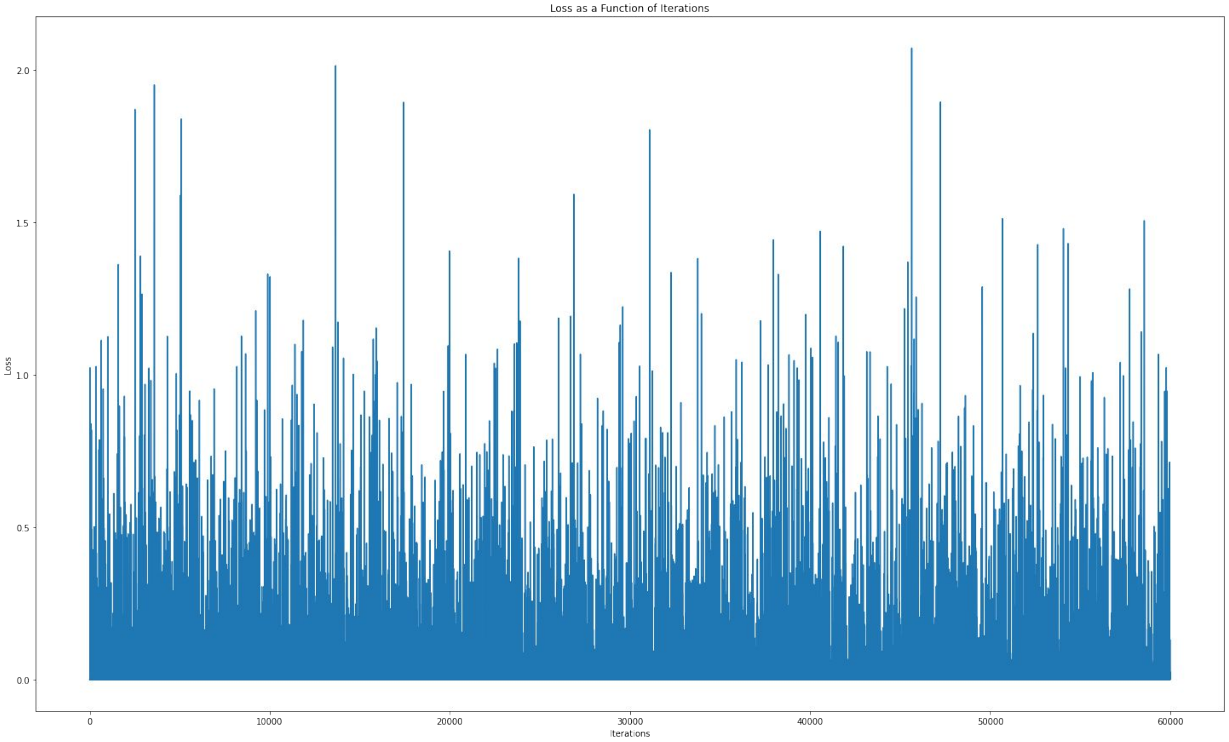
- Then we tried merging our with the skeleton
    - Ran into bugs
    - But, we figured out why we got 10% accuracy
- In the training loop, we wrote model.zero_grad() instead of optimizer.zero_grad()
    - Essentially, although we had chosen an optimizer, we weren’t training our model
- So, we continued to work on our original code and didn’t end up using the skeleton
- Then, we wanted to create an even better model
    - For all tests, we used 10 epochs and trained with the first 60000 of the MNIST data set

Iterations

1. All FC NN with Adam using F.nll_loss()

9744/10000 = .9744

We started with 4 fully connected layers and using the Nll_loss loss function

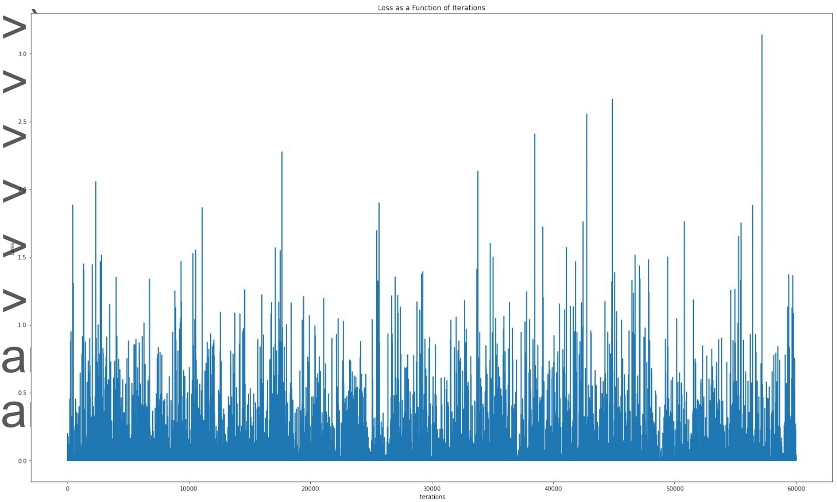


2. Then we changed the loss function to nn.CrossEntropyLoss()

Still All FC NN with Adam

9768/10000 = .9768

```
tensor(0.0110, grad_fn=<NllLossBackward0>)
tensor(2.2650e-07, grad_fn=<NllLossBackward0>)
tensor(0.0916, grad_fn=<NllLossBackward0>
tensor(0.0396, grad_fn=<NllLossBackward0>
tensor(0.0017, grad_fn=<NllLossBackward0>
tensor(0.6452, grad_fn=<NllLossBackward0>
tensor(0.0002, grad_fn=<NllLossBackward0>
tensor(0.0670, grad_fn=<NllLossBackward0>
tensor(1.1921e-08, grad_fn=<NllLossBackwa
tensor(4.5299e-07, grad_fn=<NllLossBackwa
```



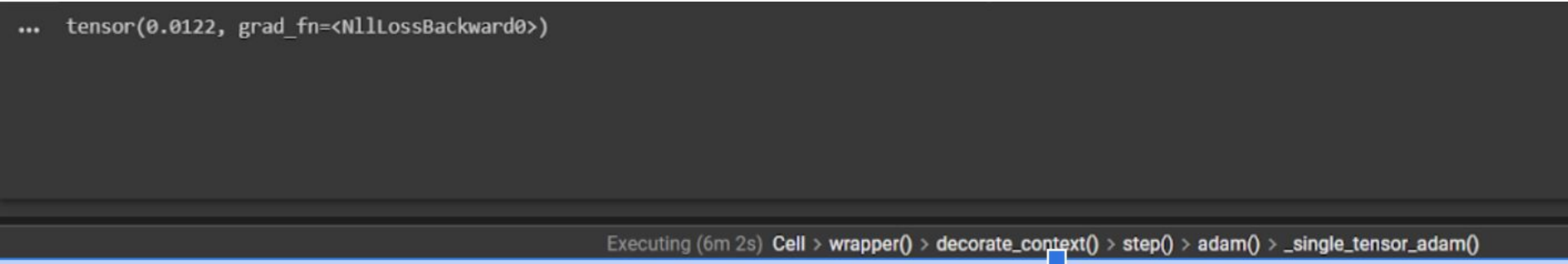
Trying we tried to get fancy!

Maxpool originally 2, increased to 14 (lowered dimension from 9216 to 64)

Running at 10 epochs with increased maxpool took:

1334.447 seconds, 22 minutes

Originally, a single epoch took: ~5 minutes * 10 iterations = 50 minutes



```
...  tensor(0.0122, grad_fn=<NllLossBackward0>)
```

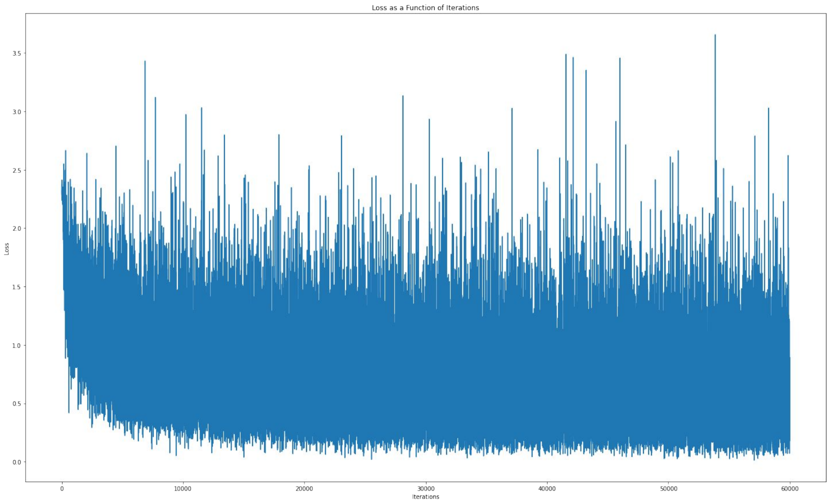
```
Executing (6m 2s)  Cell > wrapper() > decorate_context() > step() > adam() > _single_tensor_adam()
```

3. Forward function with 2 FC, 2 CL, and 2 dropout layers

8118/10000 = 0.8118

48424/60000 = 0.8070666666666667

```
tensor(1.1661, grad_fn=<NllLossBackward0>)
tensor(0.7710, grad_fn=<NllLossBackward0>)
tensor(0.2555, grad_fn=<NllLossBackward0>)
tensor(0.9164, grad_fn=<NllLossBackward0>)
tensor(0.6896, grad_fn=<NllLossBackward0>)
tensor(0.3064, grad_fn=<NllLossBackward0>)
tensor(0.6938, grad_fn=<NllLossBackward0>)
tensor(0.2539, grad_fn=<NllLossBackward0>)
tensor(0.8421, grad_fn=<NllLossBackward0>)
tensor(0.4648, grad_fn=<NllLossBackward0>)
```



Design Process

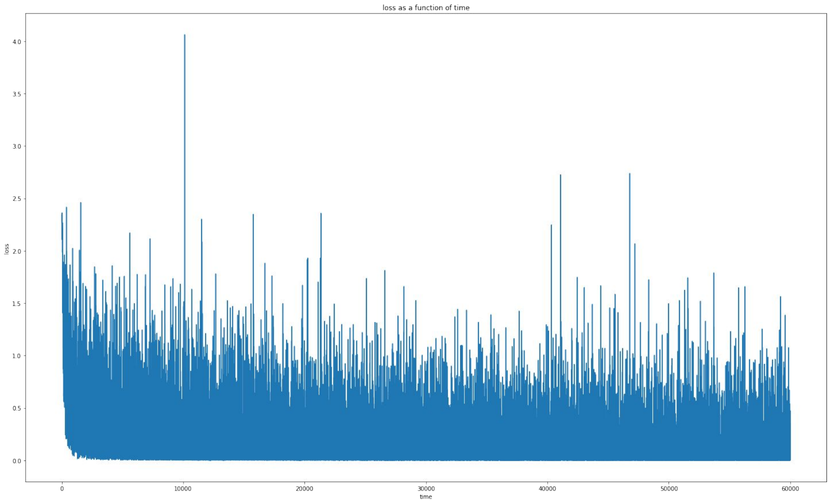
- Through trial and error we changed the amount and type of layers used
  - We didn't record data for every trial
- We ran into many issues with matching the shape of the forward function with the shape of the model
  - Unsuccessfully tried to add a single CL and a pool layer
  - Successfully tried a dropout layer with .5 rate

4. FC layers and dropout layer set to .5

9650/10000 = 0.965

58712/60000 = 0.9785333333333334

```
tensor(0.0906, grad_fn=<NllLossBackward0>)
tensor(0.0393, grad_fn=<NllLossBackward0>)
tensor(0.0035, grad_fn=<NllLossBackward0>)
tensor(0.0037, grad_fn=<NllLossBackward0>)
tensor(0.5758, grad_fn=<NllLossBackward0>)
tensor(0.0310, grad_fn=<NllLossBackward0>)
tensor(0.0160, grad_fn=<NllLossBackward0>)
tensor(0.0547, grad_fn=<NllLossBackward0>)
tensor(0.5861, grad_fn=<NllLossBackward0>)
tensor(0.0102, grad_fn=<NllLossBackward0>)
```

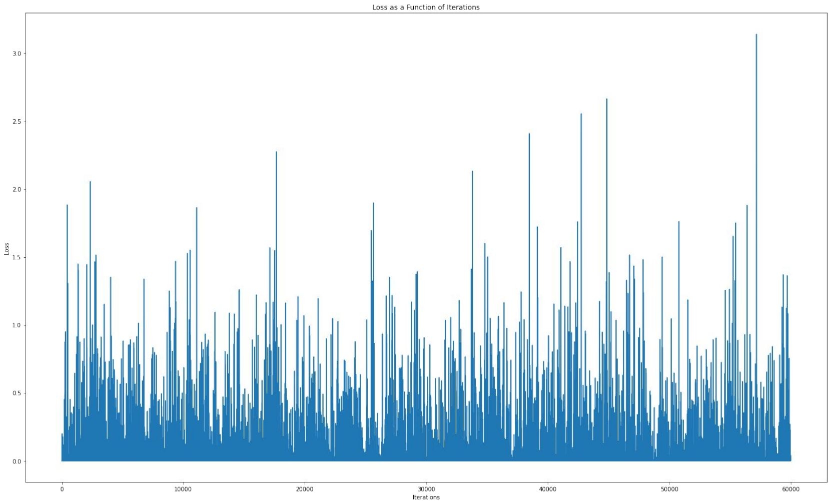


Final Result

- We chose our most accurate model

- Testing accuracy: 97.68%
- Training accuracy. We didn’t save it :/

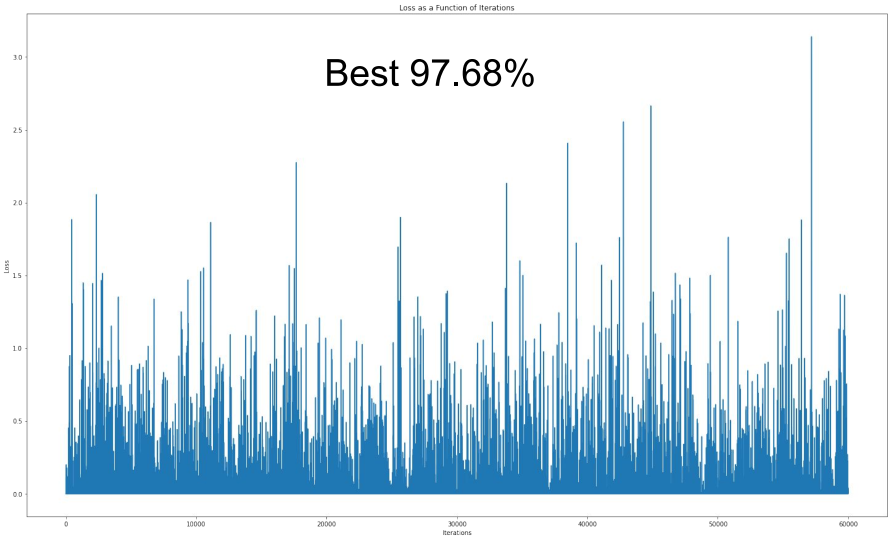
- Using FC NN only & Adam optimizer
- Using nn.CrossEntropyLoss() loss function



Best vs Worst

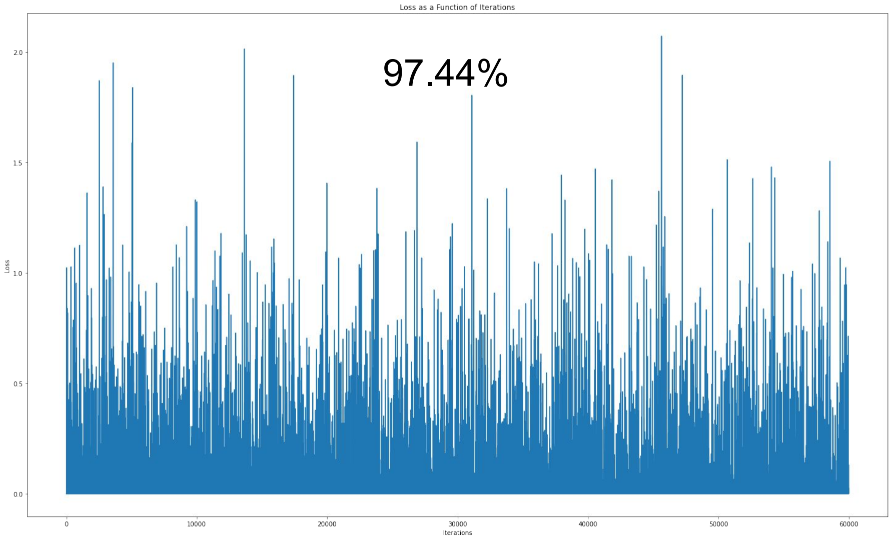
```
## 97.68% accuracy - fully connected only

tensor(0.0110, grad_fn=<NllLossBackward0>)

tensor(2.2650e-07, grad_fn=<NllLossBackward0>)

tensor(0.0916, grad_fn=<NllLossBackward0>)

tensor(0.0396, grad_fn=<NllLossBackward0>)

tensor(0.0017, grad_fn=<NllLossBackward0>)

tensor(0.6452, grad_fn=<NllLossBackward0>)

tensor(0.0002, grad_fn=<NllLossBackward0>)

tensor(0.0670, grad_fn=<NllLossBackward0>)

tensor(1.1921e-08, grad_fn=<NllLossBackward0>)

tensor(4.5299e-07, grad_fn=<NllLossBackward0>)
```

```
## 81.18% accuracy - FC, convolution and dropout layers

tensor(1.1661, grad_fn=<NllLossBackward0>)

tensor(0.7710, grad_fn=<NllLossBackward0>)

tensor(0.2555, grad_fn=<NllLossBackward0>)

tensor(0.9164, grad_fn=<NllLossBackward0>)

tensor(0.6896, grad_fn=<NllLossBackward0>)

tensor(0.3064, grad_fn=<NllLossBackward0>)

tensor(0.6938, grad_fn=<NllLossBackward0>)

tensor(0.2539, grad_fn=<NllLossBackward0>)

tensor(0.8421, grad_fn=<NllLossBackward0>)

tensor(0.4648, grad_fn=<NllLossBackward0>)
```



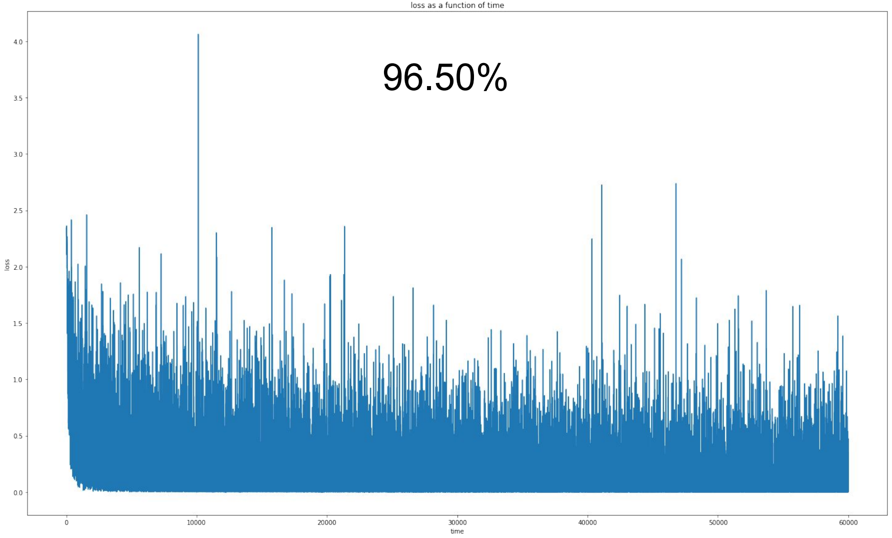
Loss as a Function of Iterations

Best 97.68%



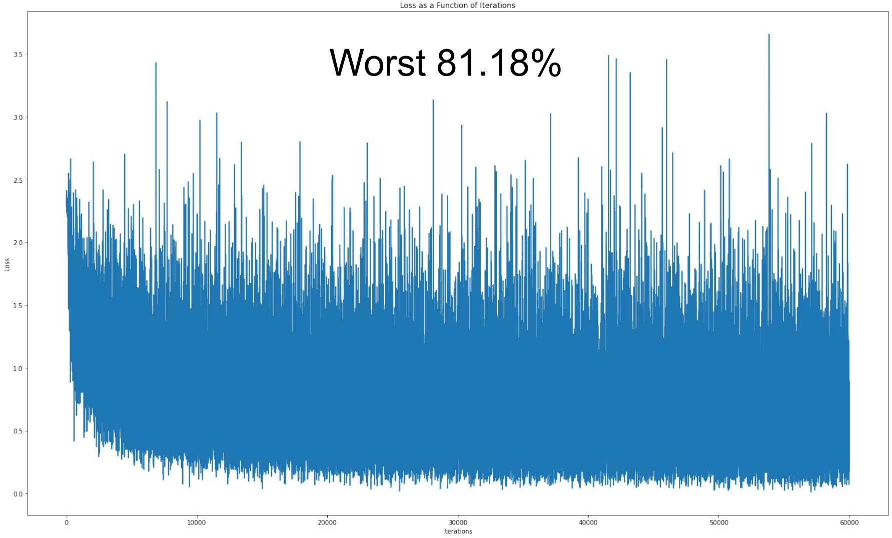
Loss as a Function of Iterations

97.44%



loss as a function of time

96.50%



Loss as a Function of Iterations

Worst 81.18%