Vpython trace analysis on MNIST data

1. Dataset MNIST

- Available in torchvision.dataset library
- Is downloaded using this statement:

```
dataset = datasets.MNIST('data', train=True, download=True, transform=transform)
```

2. Code

Adapted and modified from these sources:
- https://mlfromscratch.com/neural-network-tutorial/#/
- https://github.com/pytorch/examples/blob/main/mnist/main.py

Some notes:
- Each program uses a certain number images to train. E.g. 10, 20, …, 100, 200, …, 1000, 2000, …, 5000. While a given number of images are used to train and test, all available images are loaded at the beginning of the program.
- Every statement (loading full dataset, defining model, hyper-params, declare model object, etc.) is included in the X except for importing libraries
- There are 60000 images for training and 10000 for testing. However, we used a maximum of 5000 train images for this analysis to keep the simplicity. **In addition,** the trace prediction result (discussed in section 3 and 4) shows a stable trace outcome (trendline aligns exactly with the real trace + test RMSE is 0) so there's no need to gather trace from more images than that.
- Only 10 images are used for testing in every program
- Train in 1 epoch. Hyper-parameters are set to either default or popular choice:

```
optimizer = optim.Adadelta(model.parameters(), lr=1)
scheduler = StepLR(optimizer, step_size=1, gamma=0.7)
```

- Simple neural network model.
    - Model A uses 2 fully connected layers as described below:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(784, 32)
        self.linear2 = nn.Linear(32, 10)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = self.linear1(x)
        x = torch.sigmoid(x)
        x = self.linear2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

- Model B uses 3 fully connected layers as described below:

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.flatten = nn.Flatten()
        self.linear1 = nn.Linear(784, 128)
        self.linear2 = nn.Linear(128, 32)
        self.linear3 = nn.Linear(32, 10)

    def forward(self, x):
        x = torch.flatten(x, 1)
        x = self.linear1(x)
        x = torch.sigmoid(x)
        x = self.linear2(x)
        x = torch.sigmoid(x)
        x = self.linear3(x)
        output = F.log_softmax(x, dim=1)
        return output
```

3. Trace analysis result

Trendline plotted aligns exactly with the collected operation numbers

View in GG Sheets (Row 2, 40)

4. Test result

After rounding up the prediction outcome (by a very small fraction, less than 0.001 to be specific), most predictions match very closely with the actual values.

As a result, all RMSE values are nearly 0.

View in GG Sheets (Row 160)

Test cases Prediction

	row_num	pred_pop	pred_push	pred_grow	pred_shrink	actual_pop	actual_push	actual_grow	actual_shrink
	26	31105378	62207476	3711599	104.000000000015	31105378	62207476	104	3711599
	97	31148262	62287848	3713800	388.000000000014	31148262	62287848	388	3713800
	150	31180274	62347844	3715443	600.000000000014	31180274	62347844	600	3715443
	373	31314966	62600280	3722356	1492.00000000001	31314966	62600280	1492	3722356
	642	31477442	62904788	3730695	2568.00000000001	31477442	62904788	2568	3730695
	1234	31835010	63574932	3749047	4936.00000000001	31835010	63574932	4936	3749047
2 layers	4880	34037194	67702204	3862073	19520	34037194	67702204	19520	3862073
	7601	35680678	70782376	3946424	30404	35680678	70782376	30404	3946424
	7899	35860670	71119712	3955662	31596	35860670	71119712	31596	3955662
	11890	38271234	75637524	4079383	47560	38271234	75637524	47560	4079383
	26090	46848034	91711924	4519583	104360	46848034	91711924	104360	4519583
	33333	51222806	99911000	4744116	133332	51222806	99911000	133332	4744116
	53011	63108318	122186496	5354134	212044	63108318	122186496	212044	5354134
	RMSE	2.07E-09	1.31E-08	2.58E-10	2.61E-11				
	26	31112020	62219162	156.000000000015	3711720	31112020	62219162	156	3711720
	97	31168607	62324526	582.000000000014	3714063	31168607	62324526	582	3714063
	150	31210848	62403178	900.000000000014	3715812	31210848	62403178	900	3715812
	373	31388579	62734110	2238.00000000001	3723171	31388579	62734110	2238	3723171
	642	31602972	63133306	3852.00000000001	3732048	31602972	63133306	3852	3732048
	1234	32074796	64011834	7404.00000000001	3751584	32074796	64011834	7404	3751584
3 layers	4880	34980658	69422498	29280	3871902	34980658	69422498	29280	3871902
	7601	37149295	73460462	45606	3961695	37149295	73460462	45606	3961695
	7899	37386801	73902694	47394	3971529	37386801	73902694	47394	3971529
	11890	40567628	79825338	71340	4103232	40567628	79825338	71340	4103232
	26090	51885028	100898138	156540	4571832	51885028	100898138	156540	4571832
	33333	57657699	111646750	199998	4810851	57657699	111646750	199998	4810851
	53011	73341065	140848902	318066	5460225	73341065	140848902	318066	5460225
	RMSE	6.53E-09	5.06E-09	2.58E-10	4.07E-11				

5. Run time comparison

View in GG Sheets (Row 82 – train cases, Row 139 – test cases)

We can say vPython took a lot more time to run the same program. Not to mention >1.1 GB of trace is produced in running each program.