Bob Stutchbury
Math 5750/6880: Mathematics of Data Science
Project #3 Final Report
November 4, 2025

https://github.com/kerbs23/F25_math_for_data_science/tree/main/projects/Project3

1 Fashion-MNIST image classification using sklearn

I went through and went through and, after establishing a base case that is very similar to the default settints, changed a bunch of the settings one at a time, looking at their scores and how long they took to converge, as well as a large one based off a preliminary run. I gave them all 10,000 iterations to try to converge to a tolerance of .001, and let it spin for a while. Once they finished, I tested them on the test set and got the final score, as reported on the following page.

I used this to train a final model with 6 hidden layers, of [841, 841, 841, 500, 300, 69 layers respectively. This took about 4 minutes to train on the web machines, and resulted in a test accuracy of .891, the best of any of the models (but not remarkably better then the other, smaller, faster iterations of the model) It resulted in the following confusion matrix:

T-shirt"/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle-boot
840	3	18	27	4	0	100	0	8	0
5	965	0	21	4	0	4	0	1	0
18	0	801	16	93	1	69	1	1	0
30	8	12	891	33	0	22	1	3	0
4	0	87	31	825	0	52	0	1	0
0	1	0	0	0	954	0	24	3	18
137	2	94	29	69	0	659	0	9	1
0	0	0	0	0	24	0	954	0	22
10	0	6	7	5	8	7	6	950	1
0	0	0	0	0	9	1	32	1	957

hidden˙layers	neurons	activation	solver	learning˙rate	learning˙rate˙init	early˙stopping	training˙time	accuracy
1	”100”	relu	sgd	constant	0.001	False	251.09755682945251	0.8807
2	”100”	relu	sgd	constant	0.001	False	198.17222666740417	0.8721
4	”100”	relu	sgd	constant	0.001	False	168.52051615715027	0.8735
8	”100”	relu	sgd	constant	0.001	False	174.95713806152344	0.869
16	”100”	relu	sgd	constant	0.001	False	274.832635641098	0.8581
1	”20”	relu	sgd	constant	0.001	False	155.7820644378662	0.852
1	”50”	relu	sgd	constant	0.001	False	263.3890690803528	0.8704
1	”200”	relu	sgd	constant	0.001	False	379.7335412502289	0.8862
1	”500”	relu	sgd	constant	0.001	False	730.3306760787964	0.8921
1	”100”	logistic	sgd	constant	0.001	False	627.0255396366119	0.88
1	”100”	tanh	sgd	constant	0.001	False	282.7163245677948	0.8773
1	”100”	relu	adam	constant	0.001	False	41.16398096084595	0.8756
1	”100”	relu	lbfgs	constant	0.001	False	85.86618614196777	0.8745
1	”100”	relu	sgd	invscaling	0.001	False	15.668065309524536	0.788
1	”100”	relu	sgd	adaptive	0.001	False	288.13006234169006	0.8798
1	”100”	relu	sgd	constant	0.1	False	18.386075735092163	0.8704
1	”100”	relu	sgd	constant	0.01	False	67.6326973438263	0.8824
1	”100”	relu	sgd	constant	0.0001	False	521.4695470333099	0.8814
1	”100”	relu	sgd	constant	0.001	True	28.59324288368225	0.8783

2 Pytorch

Looks like I get a lot more fine-grained control of how each part of the model works. Stack together all the different bits into a class, then run the training stuff back and forth over it in a loop. Then, serialize the model and save it to the disk, so it can be re-loaded for testing.

It looks like there is a specific data structure that is functionally a matrix but in such a way that it can be GPUificated. For all purposes appears to be the same as the numpy arrays we were using with sklearn.

These are managed by two things: a dataset which is defined as a class that has some sort of setup, a way to return the length of the dataset, and some way to load in one item of the data. This is done for us with Fashion-MNIST, but they give an example of how to set up a custom one. Then, dataloaders work over the datasets to efficently load in little random chunks for the training.

NeuralNets are built as a part of a class, with an init section where they are defined with a series of steps including a head layer and the hidden layers and a forward section where you call the init functions in the "forward" order Then, you use the different parts of the nn package to set up phe different operations, such as layers, functions, ect It seems like the main organizational task is understanding the different layers' functions, some paramater tuning, and making sure the size of the layers match up Then, you set the layers up to define the forward pass by stacking up the init ed functions (methods? I dont really know python this way that well)

3 Fashion-MNIST image classification using PyTorch

There was just so many options on the sklearn stuff, I wasn't sure what to try and test, and I am also dissatisfied with the one observation conclutions in that section. Like come on were data scientists we ought to know better So, I defined 3 models, and ran them all 8 times overnight to get a better Idea of how the different architecture works. I initially used Nevstrov to solve it, but after learning about convergence issues with Stochastic stuff and GD (specifically, step size being dependant on iteration) I opted to switch to ADAM, and hopefully those issues are not present in that method either. I had a stopping rule whereby if we had not gotten more then .0001 below the previous best for 30 iterations, we would stop.

model	count	success avg	success stdev	loss avg	loss stdev	run time avg	run time stdev
simple	6	0.8917	0.0023	0.5799	0.0160	24.9662	1.4367
big	6	0.8911	0.0032	0.5466	0.0363	40.9640	2.2112
image	6	0.9187	0.0017	0.5779	0.0317	681.7904	16.1777

Note: 95% student's t for n=6 is 2.4.

The simple network is 3 layers of simple linear/relu functions. The performance is unremarkable compared to the sk results, and it trained much faster (likely because we are now on a GPU)

The big network is 6 layers of linear/relu, I just tried to make the simple one twice as "long". It does not preform significantly different from the simple network, but took about 15 seconds longer to converge.

The image network is like the simple network, but with 2 alternating sections of convolution (learnable interaction between adjacent pixels) and maxpool (deterministic dim reduction) added to the front. Took much longer to converge, but did statistically better than the other models.