INFO111 Part A and B

SID: 520474211, UniKey: jhal4273

March 31, 2023

1 GITHUB

Please use this public repository to access all files mentioned:
https://github.com/Jefry1217/INFO1111

1.1 Level A Demonstration

- Creating Tensors with different shapes, size, and values
- Vector manipulation of tensors of various types
- Matrix manipulation of tensors of various types

The demonstrations can be found in the python file "part_a.py" located in the git repository. Run the file and read the output to see what I've learnt.

1.2 Learning Approach

Pytorch involves neural networks and artificial intelligence which is quite a complicated topic, however there are abundant resources to learn from online. I started by googling "what are pytorch tensors" and trying to gain some preliminary information, however quickly went to YouTube as there are lots of videos explaining what they are and what you can do with them. I came across a short series that introduces you to pytorch and tensor manipulation, that also came with a python file attached that you could run in jupyter notebooks to see how the code was working and experiment with yourself. It was from that source where I learnt most the basics of creating tensors and tensor manipulation. Once I knew the basics, I experimented with a few things myself, such as how to create a tensor filled with random or roughly even amounts of 1's or 0's, representing a boolean tensor, and the errors with floating point numbers that came up when making a tensor of integers by converting from float to int types.

1.3 Challenges and Difficulties

As of part A, there aren't a lot of challenges in finding information and understanding what I have to do, as there are a lot of resources online, and the material is not too difficult to understand conceptually, especially after having done linear algebra. However this is not the case for part B, where there are many challenges. First, finding a problem to solve with pytorch is extremely difficult, as the problems become quite difficult very quickly when dealing with photos and non clean data. Understanding how loss functions, learning rates, momentum, epochs, and training batches all work together is very difficult and requires quite a lot of knowledge to get correct. Along with this, finding a dataset to work with is proving to be extremely difficult. Many datasets need large amounts of cleaning before they can be used, which I don't have the time or skills for, and the easy datasets to use already have fully written out solutions online, so it is difficult to try and find and solve my own problem. I have tried making my own dataset, and training a my NN on that, however it hasn't worked out so far.

1.4 Learning Sources

Learning Source 1: https://pytorch.org/tutorials/beginner/introyt/tensors_deeper_tutorial.html
Contribution to Learning 1: Great source that outlines how to create tensors of different shapes and sizes, and how to perform mathematical operations on them Learning Source 2: https://www.youtube.com/watch?v=r7
Contribution to Learning 2: Video explanation of creating tensors and doing some manipulations with them, as well as a jupyter notebook python file to download and run through yourself.

Application artifacts

I created a python file "part_a.py" that shows all the different types of tensors that can be created, how to create them, and mathematical operations you can do with them. Running the python file and reading the output should give enough information to gain a decent understanding of tensors and how they can be created and manipulated.

The following page and a half is very similar to the python file "part_a.py" located in my git repository. However here it is easier to read each section, and has to code used to create the output from the python file, where running the python file gives the same descriptions (but harder to read because not in latex format), and the actual output from the commands shown in this section.

1 Creating tensors

Tensors are created by using $torch.type(dimensons)$, where *type* determines the default value of the values inside the tensor. For example, here is how we create various tensors with dimensions 2x4. Dimensions 2x4 means an array of 2 arrays with 4 values in each.

torch.empty(2, 4)
torch.ones(2, 4)
torch.rand(2, 4)

2 Controlling randomness of tensors

The seed can be set for random tensor generation to control the randomness. The seed is only set for the next tensor, therefore the seed must be set each time before a tensor is created if you wish to have them be the same using the random function. The following two tensors will be the same if run in code.

torch.manual_seed(1000)
torch.rand(2, 3)
torch.manual_seed(1000)
torch.rand(2, 3)

3 Names of tensors

Tensors of 1 dimension are sometimes referred to as vectors: torch.rand(3)
torch.rand(5)
Tensors of 2 dimensions are sometimes referred to as matrices: torch.rand(2, 3) torch.rand(4, 5)

Tensors of higher dimensions are all just called tensors. In theory there isn't a limit to how many dimensions you can create a tensor with, however they get very big very quickly. As each dimension is nested inside of each other. Dimensions in tensors can be thought of as just nested arrays.

torch.rand(3, 4, 5)
torch.rand(2, 2, 4, 3)
torch.rand(2, 5, 3, 2, 3)

4 Mathematical operations of tensors and a constant

Tensors of any size can be multiplied by a constant, and as a result all the values in the tensor will be multiplied by that constant.

Multiplying a random tensor by 3 will give all values between 0 and 3.
torch.rand(2, 3, 2) * 3

Similarly, Tensors of any size can be added by a constant, and as a result all the values in the tensor will be added by that constant Adding 10 to a random tensor will give all values between 10 and 11. torch.rand(2, 3) + 10

The same idea goes for division and subtraction.

5 Mathematical operations of two tensors

If tensors are of the same size, they can be added, multiplied,divided, and subtracted from each other. The mathematical operation will simply apply to every pair of values that are in the same position in the tensor.
Subtracting two random tensors will give all values between -1 and 1
torch.rand(2, 2) - torch.rand(2, 2)
Multiplying two random tensors will give all values between 0 and 1

torch.rand(2, 3) * torch.rand(2, 3)

Adding two random tensors that have been multiplied first by 5 and 3 respectively will give a tensor with all values between 0 and 15

(torch.rand(3, 3) * 5) + (torch.rand(3, 3) * 3)

6 Tensors of different default types

So far all values in the tensors have been 32bit floating point numbers, as this is the default, however we can specify a range of different types when the tensor is created using the $dtype = torch.type$ argument

Here is how to make a tensor of zeroes as 32bit ints, then a tensor of ones as 64bit ints

torch.empty(2, 2, dtype=torch.int32)

torch.ones(2, 2, dtype=torch.int64)

There is a chance each of the 0s in the first tensor will be cast to a very large negative number, instead of 0. This is because of how floating point numbers are stored. 0 as a float will generally be slightly above or slightly below 0. Therefore adding something like 0.1 to all values before changing to int would fix this, leading us to our next part...

7 Changing tensor value types after creation

We can also change the type of a tensor after creating using *tensor.to*.

Here is how to make a tensor of actual 0 int values:

(torch.empty(2,2) + 0.1).to(torch.int32)

Now we'll make a tensor with random int16 values from 0 to 9.

First we'll create a random tensor and multiply it by 10

t = torch.rand(2, 2) * 10

And now we'll change it's type to int16.

t = t.to(torch.int16)

We can use the same process to create a tensor where roughly half the values will be True, roughly half the values will be False, however it depends on the initial random generation.

We will create a random tensor, add 0.5 to it, change to int, then change to bool.

torch.rand(2, 2) + 0.5).to(torch.int32).to(torch.bool)

*2. Level B: Basic Application

*2.1. Level B Demonstration I created a NN model with pytorch that guesses the model of a tesla car given it's 0-60 time, price, speed, and range. I created my own database and learnt how to create a neuralnet that would accept 4 integer values as entries, and output a single integer value representing the model of the car. I had to experiment with different loss functions, epoch numbers, data set values, learning rates, and momentum.

*2.2 Application artifacts I created my own which lists 5000 different entries of a Tesla car, each with attributes "0-60", "price", "speed", "range_km", "model". Each entry has randomly generated values which lie within specifications determined by the model of the car. Using pytorch, I created a neuralnet that given the 4 attributes besides model, it guesses which model of car it is.

I started of by creating the dataset through randomly generated values which have limits depending on the model chosen. If the attributes such as "range" and "price" were exactly the same each model, there wouldn't be enough diverse data, and the model would be over fitting, memorizing the training data. Line 5-58 in the python file shows how the data is created.

After creating the data, I create the neuralnet which has an input layer with 4 input features (speed, range, 0-60, price), a hidden layer with 7 features, and an output layer with 1 output feature (the model type). The hidden layer having 7 features was purely based off of a lot of testing to see what produced the most accurate results.

I set the loss function to MSELoss, as after testing with some others, this suited the best. Other functions had problems such as Cross Entropy Loss which caused all my loss on every epoch to be -0.0, and a custom loss function which was simply not as accurate as MSELoss.

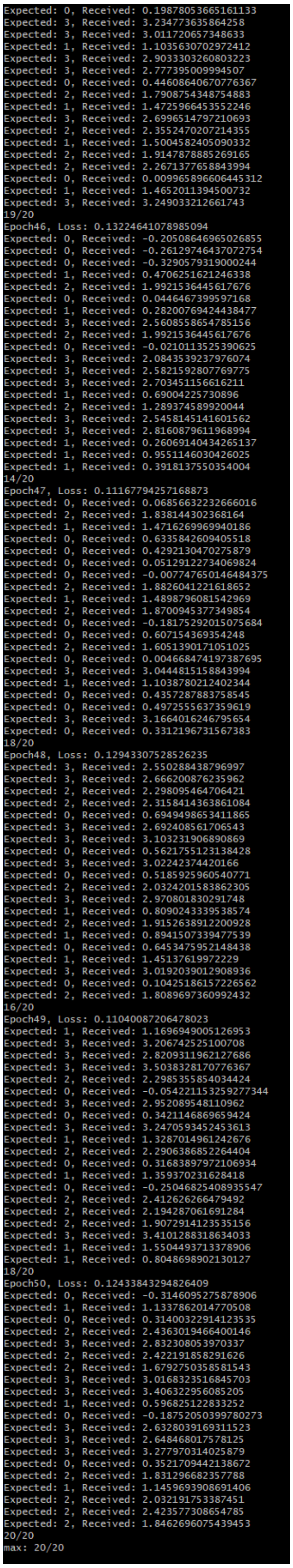
Similarly the optimizer, learning rate, and momentum was chosen after testing multiple, and RMSprop was the most accurate, as well as a learning rate of 0.00005, and default momentum. The learning rate is very low, however this is because the data is quite simply. Too high of a learning rate will cause the output to be exactly the same every time.

I chose 50 epochs, and 2000 datapoints in each epoch after testing various combinations. In each epoch, the first 2000 datapoints of the dataset I created was used.

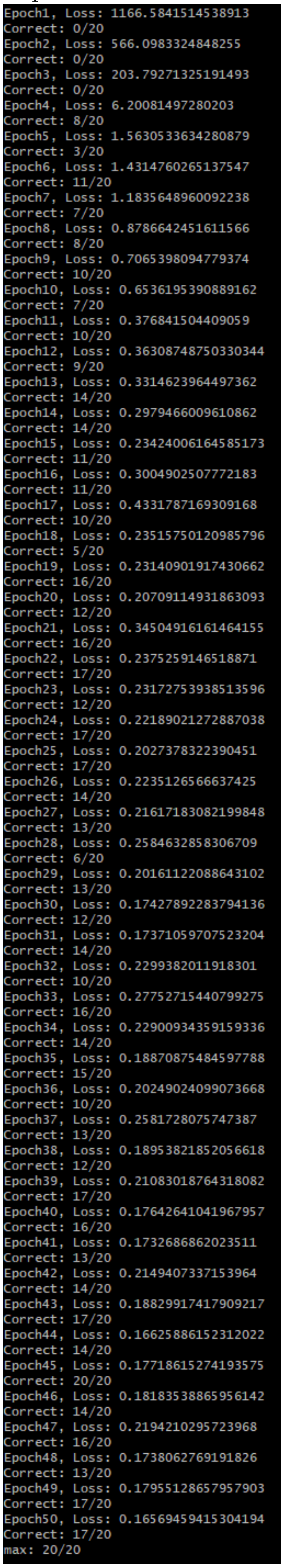
The result of all of this is a quite accurate NN that can predict the model of a Tesla car given the four inputs, however there is much variability between each time the code is run. Through every epoch, a test was run on 20 random data points out of the 5000 in the data set, and the number of correct was printed, as well as the Loss from that Epoch. Out of the 50 epochs, I kept track of every test that achieved greater than 10/20, greater than 15/20, and also the maximum result achieved. This allows me to see how well the training went throughout the 50epochs.

Most of the time the NN does very well, showing obvious improvements over multiple epochs, and getting many above 15 results, and around 30 above 10 results. However, every now and then, the loss converges very quickly and the NN training doesn't work very well, outputting the same value every time. This could be to do with the randomness of generating the data set.

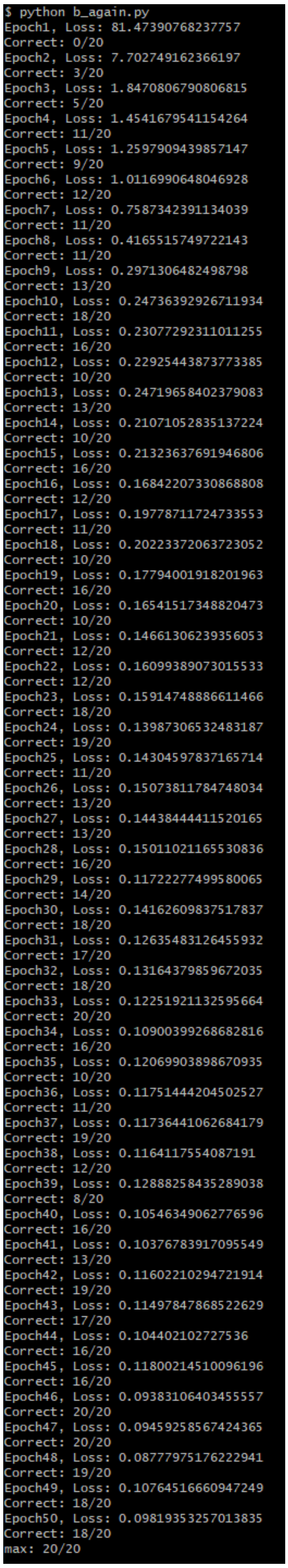
Here are some good examples:



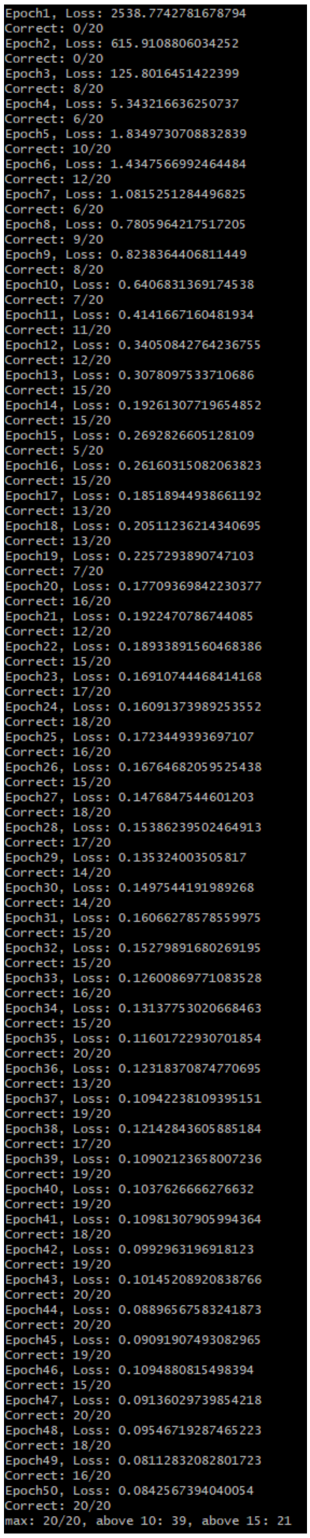
```
Expected: 0, Received: 0.19878053665161133
Expected: 3, Received: 3.234773635864258
Expected: 3, Received: 3.011720657348633
Expected: 1, Received: 1.1035630702972412
Expected: 3, Received: 2.9033303260803223
Expected: 3, Received: 2.777395009994507
Expected: 0, Received: 0.44608640670776367
Expected: 2, Received: 1.790875434875488З
Expected: 1, Received: 1.4725966453552246
Expected: 3, Received: 2.6996514797210693
Expected: 2, Received: 2.3552470207214355
Expected: 1, Received: 1.5004582405090332
Expected: 2, Received: 1.9147878885269165
Expected: 2, Received: 2.2671377658843994
Expected: 0, Received: 0.009965896606445312
Expected: 1, Received: 1.4652011394500732
Expected: 3, Received: 3.249033212661743
19/20
Epoch46, Loss: 0.13224641078985094
Expected: 0, Received: -0.20508646965026855
Expected: 0, Received: -0.26129746437072754
Expected: 0, Received: -0.3290579319000244
Expected: 1, Received: 0.4706251621246338
Expected: 2, Received: 1.9921536445617676
Expected: 0, Received: 0.0446467399597168
Expected: 1, Received: 0.28200769424438477
Expected: 3, Received: 2.5608558654785156
Expected: 2, Received: 1.9921536445617676
Expected: 0, Received: -0.0210113525390625
Expected: 3, Received: 2.0843539237976074
Expected: 3, Received: 2.5821592807769775
Expected: 3, Received: 2.703451156616211
Expected: 1, Received: 0.69004225730896
Expected: 2, Received: 1.289374589920044
Expected: 3, Received: 2.5458145141601562
Expected: 3, Received: 2.8160879611968994
Expected: 1, Received: 0.26069140434265137
Expected: 1, Received: 0.9551146030426025
Expected: 1, Received: 0.3918137550354004
14/20
Epoch47, Loss: 0.11167794257168873
Expected: 0, Received: 0.06856632232666016
Expected: 2, Received: 1.838144302368164
Expected: 1, Received: 1.4716269969940186
Expected: 0, Received: 0.6335842609405518
Expected: 0, Received: 0.4292130470275879
Expected: 0, Received: 0.05129122734069824
Expected: 0, Received: -0.007747650146484375
Expected: 2, Received: 1.8826041221618652
Expected: 1, Received: 1.4898796081542969
Expected: 2, Received: 1.8700945377349854
Expected: 0, Received: -0.18175292015075684
Expected: 0, Received: 0.607154369354248
Expected: 2, Received: 1.6051390171051025
Expected: 0, Received: 0.004668474197387695
Expected: 3, Received: 3.0444815158843994
Expected: 1, Received: 1.1038780212402344
Expected: 0, Received: 0.4357287883758545
Expected: 0, Received: 0.4972555637359619
Expected: 3, Received: 3.1664016246795654
Expected: 0, Received: 0.3312196731567383
18/20
Epoch48, Loss: 0.12943307528526235
Expected: 3, Received: 2.550288438796997
Expected: 3, Received: 2.666200876235962
Expected: 2, Received: 2.298095464706421
Expected: 2, Received: 2.3158414363861084
Expected: 0, Received: 0.6949498653411865
Expected: 3, Received: 2.692408561706543
Expected: 3, Received: 3.103231906890869
Expected: 0, Received: 0.5621755123138428
Expected: 3, Received: 3.02242374420166
Expected: 0, Received: 0.5185925960540771
Expected: 2, Received: 2.0324201583862305
Expected: 3, Received: 2.970801830291748
Expected: 1, Received: 0.8090243339538574
Expected: 2, Received: 1.9152638912200928
Expected: 1, Received: 0.8941507339477539
Expected: 0, Received: 0.6453475952148438
Expected: 1, Received: 1.45137619972229
Expected: 3, Received: 3.0192039012908936
Expected: 0, Received: 0.10425186157226562
Expected: 2, Received: 1.8089697360992432
16/20
Epoch49, Loss: 0.11040087206478023
Expected: 1, Received: 1.1696949005126953
Expected: 3, Received: 3.206742525100708
Expected: 3, Received: 2.8209311962127686
Expected: 3, Received: 3.5038328170776367
Expected: 2, Received: 2.2985355854034424
Expected: 0, Received: -0.054221153259277344
Expected: 3, Received: 2.952089548110962
Expected: 0, Received: 0.3421146869659424
Expected: 3, Received: 3.2470593452453613
Expected: 1, Received: 1.3287014961242676
Expected: 2, Received: 2.2906386852264404
Expected: 0, Received: 0.31683897972106934
Expected: 1, Received: 1.359370231628418
Expected: 0, Received: -0.25046825408935547
Expected: 2, Received: 2.412626266479492
Expected: 2, Received: 2.194287061691284
Expected: 2, Received: 1.9072914123535156
Expected: 3, Received: 3.4101288318634033
Expected: 1, Received: 1.5504493713378906
Expected: 1, Received: 0.8048698902130127
18/20
Epoch50, Loss: 0.12433843294826409
Expected: 0, Received: -0.3146095275878906
Expected: 1, Received: 1.1337862014770508
Expected: 0, Received: 0.31400322914123535
Expected: 2, Received: 2.4363019466400146
Expected: 3, Received: 2.832308053970337
Expected: 2, Received: 2.422191858291626
Expected: 2, Received: 1.6792750358581543
Expected: 3, Received: 3.0168323516845703
Expected: 3, Received: 3.406322956085205
Expected: 1, Received: 0.596825122833252
Expected: 0, Received: -0.18752050399780273
Expected: 3, Received: 2.6328039169311523
Expected: 3, Received: 2.648468017578125
Expected: 3, Received: 3.277970314025879
Expected: 0, Received: 0.3521709442138672
Expected: 2, Received: 1.831296682357788
Expected: 1, Received: 1.1459693908691406
Expected: 2, Received: 2.032191753387451
Expected: 2, Received: 2.423577308654785
Expected: 2, Received: 1.8462696075439453
20/20
max: 20/20
```



```
Epoch1, Loss: 1166.5841514538913
Correct: 0/20
Epoch2, Loss: 566.0983324848255
Correct: 0/20
Epoch3, Loss: 203.79271325191493
Correct: 0/20
Epoch4, Loss: 6.20081497280203
Correct: 8/20
Epoch5, Loss: 1.5630533634280879
Correct: 3/20
Epoch6, Loss: 1.4314760265137547
Correct: 11/20
Epoch7, Loss: 1.1835648960092238
Correct: 7/20
Epoch8, Loss: 0.8786642451611566
Correct: 8/20
Epoch9, Loss: 0.7065398094779374
Correct: 10/20
Epoch10, Loss: 0.6536195390889162
Correct: 7/20
Epoch11, Loss: 0.376841504409059
Correct: 10/20
Epoch12, Loss: 0.36308748750330344
Correct: 9/20
Epoch13, Loss: 0.3314623964497362
Correct: 14/20
Epoch14, Loss: 0.2979466009610862
Correct: 14/20
Epoch15, Loss: 0.23424006164585173
Correct: 11/20
Epoch16, Loss: 0.3004902507772183
Correct: 11/20
Epoch17, Loss: 0.4331787169309168
Correct: 10/20
Epoch18, Loss: 0.23515750120985796
Correct: 5/20
Epoch19, Loss: 0.23140901917430662
Correct: 16/20
Epoch20, Loss: 0.20709114931863093
Correct: 12/20
Epoch21, Loss: 0.34504916161464155
Correct: 16/20
Epoch22, Loss: 0.2375259146518871
Correct: 17/20
Epoch23, Loss: 0.23172753938513596
Correct: 12/20
Epoch24, Loss: 0.22189021272887038
Correct: 17/20
Epoch25, Loss: 0.2027378322390451
Correct: 17/20
Epoch26, Loss: 0.2235126566637425
Correct: 14/20
Epoch27, Loss: 0.21617183082199848
Correct: 13/20
Epoch28, Loss: 0.2584632858306709
Correct: 6/20
Epoch29, Loss: 0.20161122088643102
Correct: 13/20
Epoch30, Loss: 0.17427892283794136
Correct: 12/20
Epoch31, Loss: 0.17371059707523204
Correct: 14/20
Epoch32, Loss: 0.2299382011918301
Correct: 10/20
Epoch33, Loss: 0.27752715440799275
Correct: 16/20
Epoch34, Loss: 0.22900934359159336
Correct: 14/20
Epoch35, Loss: 0.18870875484597788
Correct: 15/20
Epoch36, Loss: 0.20249024099073668
Correct: 10/20
Epoch37, Loss: 0.2581728075747387
Correct: 13/20
Epoch38, Loss: 0.18953821852056618
Correct: 12/20
Epoch39, Loss: 0.21083018764318082
Correct: 17/20
Epoch40, Loss: 0.17642641041967957
Correct: 16/20
Epoch41, Loss: 0.1732686862023511
Correct: 13/20
Epoch42, Loss: 0.2149407337153964
Correct: 14/20
Epoch43, Loss: 0.18829917417909217
Correct: 17/20
Epoch44, Loss: 0.16625886152312022
Correct: 14/20
Epoch45, Loss: 0.17718615274193575
Correct: 20/20
Epoch46, Loss: 0.18183538865956142
Correct: 14/20
Epoch47, Loss: 0.2194210295723968
Correct: 16/20
Epoch48, Loss: 0.1738062769191826
Correct: 13/20
Epoch49, Loss: 0.17955128657957903
Correct: 17/20
Epoch50, Loss: 0.16569459415304194
Correct: 17/20
max: 20/20
```



```
$ python b_again.py
Epoch1, Loss: 81.47390768237757
Correct: 0/20
Epoch2, Loss: 7.702749162366197
Correct: 3/20
Epoch3, Loss: 1.8470806790806815
Correct: 5/20
Epoch4, Loss: 1.4541679541154264
Correct: 11/20
Epoch5, Loss: 1.2597909439857147
Correct: 9/20
Epoch6, Loss: 1.0116990648046928
Correct: 12/20
Epoch7, Loss: 0.7587342391134039
Correct: 11/20
Epoch8, Loss: 0.4165515749722143
Correct: 11/20
Epoch9, Loss: 0.2971306482498798
Correct: 13/20
Epoch10, Loss: 0.24736392926711934
Correct: 18/20
Epoch11, Loss: 0.23077292311011255
Correct: 16/20
Epoch12, Loss: 0.22925443873773385
Correct: 10/20
Epoch13, Loss: 0.24719658402379083
Correct: 13/20
Epoch14, Loss: 0.21071052835137224
Correct: 10/20
Epoch15, Loss: 0.21323637691946806
Correct: 16/20
Epoch16, Loss: 0.16842207330868808
Correct: 12/20
Epoch17, Loss: 0.19778711724733553
Correct: 11/20
Epoch18, Loss: 0.20223372063723052
Correct: 10/20
Epoch19, Loss: 0.17794001918201963
Correct: 16/20
Epoch20, Loss: 0.16541517348820473
Correct: 10/20
Epoch21, Loss: 0.14661306239356053
Correct: 12/20
Epoch22, Loss: 0.16099389073015533
Correct: 12/20
Epoch23, Loss: 0.15914748886611466
Correct: 18/20
Epoch24, Loss: 0.13987306532483187
Correct: 19/20
Epoch25, Loss: 0.14304597837165714
Correct: 11/20
Epoch26, Loss: 0.15073811784748034
Correct: 13/20
Epoch27, Loss: 0.14438444411520165
Correct: 13/20
Epoch28, Loss: 0.15011021165530836
Correct: 16/20
Epoch29, Loss: 0.11722277499580065
Correct: 14/20
Epoch30, Loss: 0.14162609837517837
Correct: 18/20
Epoch31, Loss: 0.12635483126455932
Correct: 17/20
Epoch32, Loss: 0.13164379859672035
Correct: 18/20
Epoch33, Loss: 0.12251921132595664
Correct: 20/20
Epoch34, Loss: 0.10900399268682816
Correct: 16/20
Epoch35, Loss: 0.12069903898670935
Correct: 10/20
Epoch36, Loss: 0.11751444204502527
Correct: 11/20
Epoch37, Loss: 0.11736441062684179
Correct: 19/20
Epoch38, Loss: 0.1164117554087191
Correct: 12/20
Epoch39, Loss: 0.12888258435289038
Correct: 8/20
Epoch40, Loss: 0.10546349062776596
Correct: 16/20
Epoch41, Loss: 0.10376783917095549
Correct: 13/20
Epoch42, Loss: 0.11602210294721914
Correct: 19/20
Epoch43, Loss: 0.11497847868522629
Correct: 17/20
Epoch44, Loss: 0.104402102727536
Correct: 16/20
Epoch45, Loss: 0.11800214510096196
Correct: 16/20
Epoch46, Loss: 0.09383106403455557
Correct: 20/20
Epoch47, Loss: 0.09459258567424365
Correct: 20/20
Epoch48, Loss: 0.08777975176222941
Correct: 19/20
Epoch49, Loss: 0.10764516660947249
Correct: 18/20
Epoch50, Loss: 0.09819353257013835
Correct: 18/20
max: 20/20
```



```
Epoch1, Loss: 2538.7742781678794
Correct: 0/20
Epoch2, Loss: 615.9108806034252
Correct: 0/20
Epoch3, Loss: 125.8016451422399
Correct: 8/20
Epoch4, Loss: 5.343216636250737
Correct: 6/20
Epoch5, Loss: 1.8349730708832839
Correct: 10/20
Epoch6, Loss: 1.4347566992464484
Correct: 12/20
Epoch7, Loss: 1.0815251284496825
Correct: 6/20
Epoch8, Loss: 0.7805964217517205
Correct: 9/20
Epoch9, Loss: 0.8238364406811449
Correct: 8/20
Epoch10, Loss: 0.6406831369174538
Correct: 7/20
Epoch11, Loss: 0.4141667160481934
Correct: 11/20
Epoch12, Loss: 0.34050842764236755
Correct: 12/20
Epoch13, Loss: 0.3078097533710686
Correct: 15/20
Epoch14, Loss: 0.19261307719654852
Correct: 15/20
Epoch15, Loss: 0.2692826605128109
Correct: 5/20
Epoch16, Loss: 0.26160315082063823
Correct: 15/20
Epoch17, Loss: 0.18518944938661192
Correct: 13/20
Epoch18, Loss: 0.20511236214340695
Correct: 13/20
Epoch19, Loss: 0.2257293890747103
Correct: 7/20
Epoch20, Loss: 0.17709369842230377
Correct: 16/20
Epoch21, Loss: 0.1922470786744085
Correct: 12/20
Epoch22, Loss: 0.18933891560468386
Correct: 15/20
Epoch23, Loss: 0.16910744468414168
Correct: 17/20
Epoch24, Loss: 0.16091373989253552
Correct: 18/20
Epoch25, Loss: 0.1723449393697107
Correct: 16/20
Epoch26, Loss: 0.16764682059525438
Correct: 15/20
Epoch27, Loss: 0.1476847544601203
Correct: 18/20
Epoch28, Loss: 0.15386239502464913
Correct: 17/20
Epoch29, Loss: 0.135324003505817
Correct: 14/20
Epoch30, Loss: 0.1497544191989268
Correct: 14/20
Epoch31, Loss: 0.16066278578559975
Correct: 15/20
Epoch32, Loss: 0.15279891680269195
Correct: 15/20
Epoch33, Loss: 0.12600869771083528
Correct: 16/20
Epoch34, Loss: 0.13137753020668463
Correct: 15/20
Epoch35, Loss: 0.11601722930701854
Correct: 20/20
Epoch36, Loss: 0.12318370874770695
Correct: 13/20
Epoch37, Loss: 0.10942238109395151
Correct: 19/20
Epoch38, Loss: 0.12142843605885184
Correct: 17/20
Epoch39, Loss: 0.10902123658007236
Correct: 19/20
Epoch40, Loss: 0.1037626666276632
Correct: 19/20
Epoch41, Loss: 0.10981307905994364
Correct: 18/20
Epoch42, Loss: 0.0992963196918123
Correct: 19/20
Epoch43, Loss: 0.10145208920838766
Correct: 20/20
Epoch44, Loss: 0.08896567583241873
Correct: 20/20
Epoch45, Loss: 0.09091907493082965
Correct: 19/20
Epoch46, Loss: 0.1094880815498394
Correct: 15/20
Epoch47, Loss: 0.09136029739854218
Correct: 20/20
Epoch48, Loss: 0.09546719287465223
Correct: 18/20
Epoch49, Loss: 0.08112832082801723
Correct: 16/20
Epoch50, Loss: 0.0842567394040054
Correct: 20/20
max: 20/20, above 10: 39, above 15: 21
```

Here is a bad example that happens every now and then:

```
Epoch1, Loss: 4.571485781998779
Correct: 6/20
Epoch2, Loss: 2.7488707733210176
Correct: 2/20
Epoch3, Loss: 2.5253272129818796
Correct: 6/20
Epoch4, Loss: 2.320881602421403
Correct: 10/20
Epoch5, Loss: 2.1342436934188007
Correct: 5/20
Epoch6, Loss: 1.9654056225568055
Correct: 7/20
Epoch7, Loss: 1.8143525319285692
Correct: 5/20
Epoch8, Loss: 1.6810675980085508
Correct: 5/20
Epoch9, Loss: 1.5655317016157788
Correct: 5/20
Epoch10, Loss: 1.4677164405303313
Correct: 1/20
Epoch11, Loss: 1.3875799053540687
Correct: 8/20
Epoch12, Loss: 1.3250711648254656
Correct: 6/20
Epoch13, Loss: 1.2800959385801107
Correct: 4/20
Epoch14, Loss: 1.2524739755764605
Correct: 6/20
Epoch15, Loss: 1.2417528189346194
Correct: 3/20
Epoch16, Loss: 1.2437523884177208
Correct: 8/20
Epoch17, Loss: 1.2423153546303511
Correct: 4/20
Epoch18, Loss: 1.2435467304140329
Correct: 3/20
Epoch19, Loss: 1.242523192256689
Correct: 6/20
Epoch20, Loss: 1.2433906008303166
Correct: 4/20
Epoch21, Loss: 1.2426627342179417
Correct: 3/20
Epoch22, Loss: 1.2432835458889604
Correct: 5/20
Epoch23, Loss: 1.2427596141919495
Correct: 4/20
Epoch24, Loss: 1.2432072461098433
Correct: 5/20
Epoch25, Loss: 1.2428280089125037
Correct: 3/20
Epoch26, Loss: 1.2431521505787968
Correct: 4/20
Epoch27, Loss: 1.2428769285157324
Correct: 8/20
Epoch28, Loss: 1.2431120652630925
Correct: 2/20
Epoch29, Loss: 1.2429122208356858
Correct: 1/20
Epoch30, Loss: 1.2430829733386637
Correct: 7/20
Epoch31, Loss: 1.2429375114813448
Correct: 7/20
Epoch32, Loss: 1.2430617038458587
Correct: 4/20
Epoch33, Loss: 1.2429558972418309
Correct: 2/20
Epoch34, Loss: 1.243046266414225
Correct: 11/20
Epoch35, Loss: 1.2429693207517267
Correct: 5/20
Epoch36, Loss: 1.243034945063293
Correct: 5/20
Epoch37, Loss: 1.2429790171012283
Correct: 7/20
Epoch38, Loss: 1.243026770733297
Correct: 3/20
Epoch39, Loss: 1.2429860492646694
Correct: 4/20
Epoch40, Loss: 1.2430207990556956
Correct: 6/20
Epoch41, Loss: 1.2429911229982973
Correct: 6/20
Epoch42, Loss: 1.2430165502279997
Correct: 7/20
Epoch43, Loss: 1.2429948050379753
Correct: 4/20
Epoch44, Loss: 1.2430133819803595
Correct: 5/20
Epoch45, Loss: 1.2429975412264467
Correct: 2/20
Epoch46, Loss: 1.243011104889214
Correct: 3/20
Epoch47, Loss: 1.2429994952380656
Correct: 5/20
Epoch48, Loss: 1.2430094039440156
Correct: 9/20
Epoch49, Loss: 1.2430009444579482
Correct: 4/20
Epoch50, Loss: 1.2430081730633975
Correct: 6/20
max: 11/20, above 10: 1, above 15: 0
```