

# Lab 6: Implementing forward, deriving all unit tests

4/25/2023

50 Possible Points

Attempt 1



4/24/2023

Next Up: Review Feedback

Attempt 1 Score:  
N/A

Add Comment

## Unlimited Attempts Allowed

### ▼ Details

In this lab, you will start to implement the code for your from-scratch deep neural network library. You will implement and write unit tests for forward propagation and develop numeric test cases on paper for the backpropagation unit tests.

In particular, you will implement:

- the `__init__` methods of all classes
- `forward()` for all concrete Layer classes that will need it.
- `forward()` for the network class.
- Unit tests of the methods above.

During this week's lab, you are implementing the forward pass. You should implement the `__init__` methods of all the classes, the Network's `set_input` and `forward` methods, and the layer class's `clear_grad`, `set`, `randomize`, and `forward` methods. Each layer should maintain the instance variables `output`, which represents the output of that layer.

The `accumulate_grad` and `backward` methods will be implemented in next week's lab.

You will write unit tests for your forward direction, with paper derivations to support the values expected in these layers.

You will also write a unit test for an entire network, using one of the numeric examples you developed in lab 3.

Finally, you will develop numeric backpropagation tests for the backward direction, which will support the backpropagation work you will do in next

week's lab. You do not need to implement backpropagation or unit tests in Python this week.

# Overview of the Gradient Tape Backpropagation Library

In this lab, you will develop the forward pass of the from-scratch gradient tape backpropagation library. This library will be capable of representing and training arbitrary networks, including the best networks available today such as Inception-v3 and transformer architectures.

Because this library is built on Pytorch, it will be optimized for running on the GPU. But it may not be optimized to reuse memory or communicate between multiple GPUs effectively.

The main thing you will learn from implementing this lab is how backpropagation works which is a central topic in training the world's best neural networks today.

To use your library, the client will be responsible for adding each Layer object to the Network object in the order they will be executed. The Network will then be responsible for maintaining these layers in its gradient tape and evaluating the network in the forward and backward directions.

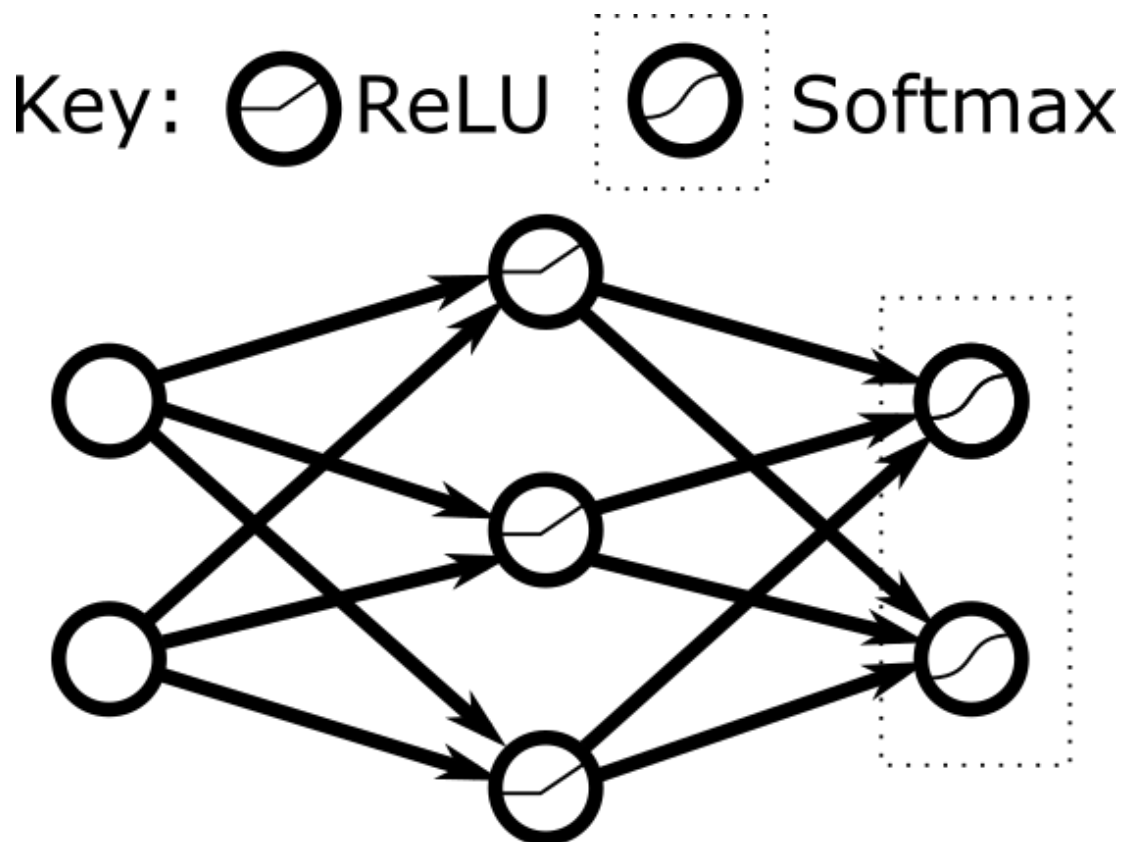
The client will also be responsible for tying each layer to the layers from which it draws its inputs. During forward propagation, each layer will derive its inputs from the layers that precede it. During backward propagation, each layer will send its input gradients for accumulation in the output gradients of the layers that precede it. So the client needs to give each layer the references it needs to communicate with the previous layers.

In Spring 2023, this was also discussed in Lab 2, and hopefully the motivation for writing Lab 2 in the way we suggested you do it is clearer now. Consider looking at your Lab2 code to see if it meets part of the requirements for this week's lab.

We will treat both inputs and learnable parameters as a single `Input` type. Doing this allows us to use the same code to update learnable parameters as we use to take backpropagation to a previous layer. All layers will collect gradients, so there is no need to distinguish between learnable parameters and inputs in this sense. But you will need some sort of `train=True` parameter

to mark those inputs whose output values should train during backpropagation.

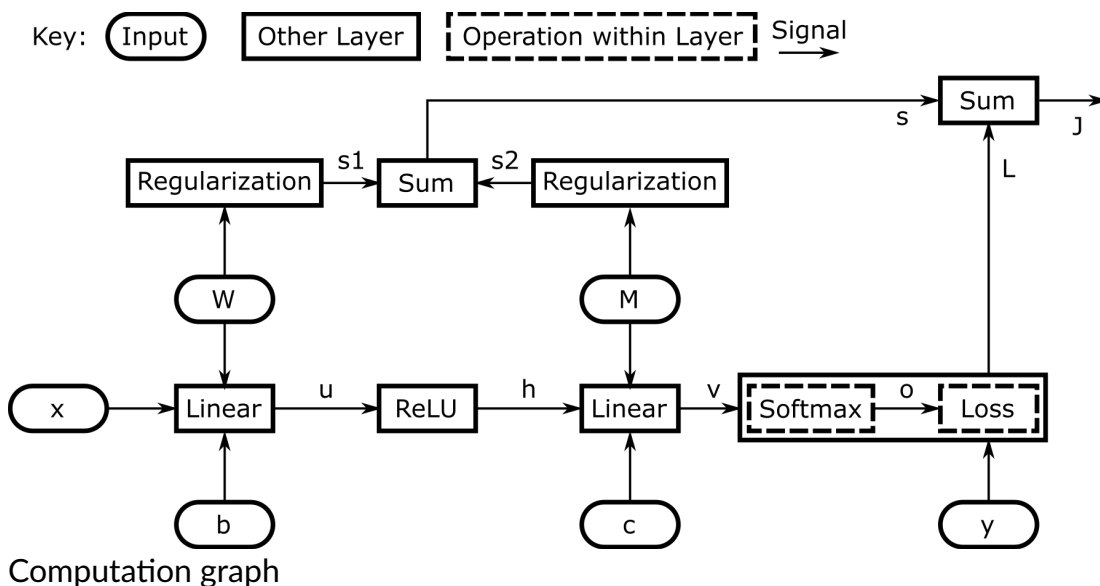
Your library should support a simple network with a single hidden layer:



Node diagram of a simple network

It will also support adding arbitrary layers, as the setup of the layers is left to the client.

Your library should allow the client to build this computation graph:



You are encouraged to adapt the design to meet your own needs. For example, you can split the Softmax and Loss into two layers as long as you hack the computation in the backward direction to use the simple gradients described in last week's lab.

But please do check with your instructor if you are consider major changes to the design. It is important that your design allow for parallel computations and accumuations from multiple sources, so, for example, embedding regularazation into the linear layers would not be approved.

## Provided code

To get started, copy the provided code to your working directory:

```
cp -r /data/cs3450/mlp_from_scratch/week4/ .
```

This will create a week3 folder (week 4 of the from-scratch sequence, starting with week 1 being the autograd on linear week) containing:

- **client.py** – You will NOT use this file this week. You will use this file in future weeks to load the training data.
- **network.py** – This holds the Network class, which, as mentioned previously, is responsible for maintaining the gradient tape and organizing the forward and backward passes.
- **layer.py** – This holds the Layer class and all of its subclasses. As mentioned previously, each layer class should maintain references to the layers that come before it. During forward propagation, each layer should derive its own output based on the output of its input layers and during backpropagation, each layer will (in future weeks) determine its input gradients from its output gradients and current inputs, and send these backward for the previous layer to accumulate into its output.
- **test\_sum.py** – This is a complete example of a unit test. You should make separate files for each of the layers you are testing with the unit tests for that layer. Having separate test files for each class makes isolating the problems for each class easier.

You can edit these python files through jupyter notebook. You are welcome to convert client.py to client.ipynb if you wish, which may make saving your output easier.

You only need to implement the methods introduced at the start of the lab.

Use good object-oriented design practices in your implementation. In Python, it is good practice to read instance variables directly (e.g. as in

`print(my_layer.output)` or `self.output = -self.input.output`) but it is poor practice to change another classes's instance variables (e.g., as in `self.input.output = -self.output`). Reduce duplicate code by putting code in a place within the hierarchy where it can be written once and called from several places.

You are encouraged to exercise some design thinking while completing the code-base. Although method stubs are provided, the parameters for the methods are not. This is so that you have some flexibility in how you define the inputs and the outputs to your layers. You are even welcome to discuss with your instructor changes to where method stubs are located if you think this may improve your code base. A fair bit of structure is required to give you a sense for how the gradient tape library will function. But our goal is for you to be able to make the library your own.

## Forward-pass unit tests

Your tests should cover the forward method for each layer.

Derive at least one numeric example for each layer for the forward direction. Write your derivations on paper.

See `test_sum.py` for an example.

To execute the unit test, include this code at the bottom of the unit-test file:

```
if __name__ == '__main__':  
    unittest.main(argv=[''], exit=False)
```

Then execute the code with `python test_sum.py` or run the code as a Jupyter notebook.

For the softmax and cross-entropy layers, it can be helpful to use faked values such as  $e$  (the base of the natural logarithm) or  $\log(2)$  so that the numbers are simpler to work on paper. For some of the matrix examples, working something out on paper can be necessary to provide a truly parallel test that doesn't depend on your PyTorch implementation.

For other layers, you can write the tests without writing anything on paper, and there is no need to include paper derivations for these layers.

These derivations must be submitted on paper – see more details below.

In addition to your unit tests, we strongly encourage you to write assertions

for the shapes and perhaps types of the inputs to a layer with the layer methods as early in the process as these assertions can be tested. For example, the linear layer benefits greatly from confirming that its three inputs have the correct shapes. But other layers benefit as well. These assertions will help you to catch errors both in your client code and in your library code.

## Full-network unit test

Write a file `test_network.py` which tests a complete network using the values you derived in Lab 2.

This test should include at least one hidden layer and at least two nodes in each layer, including the output. Your client should include a ReLU function as the activation in the hidden layer but does not need to include any activation function or loss on the output layer and does not need to include regularization. If your Lab 2 examples do not meet these requirements, please make a new example that does and include the derivations in your paper submissions for this lab.

Set the learnable parameters and inputs according to your example. Confirm that when your code runs, it produces the correct outputs for your numeric example.

## Numeric examples for backpropagation unit tests

Prepare numeric examples to test the step method and the backward method for each layer. You should write at least one numeric example for each method implementation.

Since you will be writing these numeric examples, you need some idea of how the `backward`, `accumulate_grad`, and `step` will be implemented in next week's lab.

The multi-variate chain rule states that if  $x = f(a, b, c)$  and  $a$ ,  $b$ , and  $c$  are all functions of  $y$ , then  $\frac{\partial x}{\partial y} = \frac{\partial x}{\partial a} \frac{\partial a}{\partial y} + \frac{\partial x}{\partial b} \frac{\partial b}{\partial y} + \frac{\partial x}{\partial c} \frac{\partial c}{\partial y}$ . Because of the multi-variate chain rule, we can compute the gradient of a layer as the sum of the gradients from all layers that receive that layer's output directly. This motivates our implementation of the `backward` and `accumulate_grad` methods:

The `backward` method should perform back-propagation through the single layer. At the time `backward` is called, you may assume that `self.grad` already contains  $\frac{\partial J}{\partial \text{output}}$ , that is, the derivative of the objective function  $\partial \text{input}$  for each input to this layer. For example, the `Layer Linear` class would receive  $\frac{\partial J}{\partial \mathbf{z}}$  and compute  $\frac{\partial J}{\partial \mathbf{W}}$ ,  $\frac{\partial J}{\partial \mathbf{b}}$ , and  $\frac{\partial J}{\partial \mathbf{x}}$ . These gradients are for an arbitrary `Linear` layer, not just the first layer of the example network used for the backpropagation derivations. Although  $\frac{\partial J}{\partial \mathbf{x}}$  is not needed in our entire network as  $\mathbf{x}$  is not a learnable parameter, it would be needed, for example, when doing style transfer to update the input image or when computing the gradients for the second linear layer.

The `backward` method then accumulates the input gradients into the  $\frac{\partial J}{\partial \text{output}}$  weights gradients of the previous layers using the `accumulate_grad` method of each layer, to prepare for backpropagation to continue into the earlier layers.

The `accumulate_grad` method simply adds the gradient it receives to this layer's `self.grad`. The `clear_grad` simply sets `self.grad` to zero. Both `accumulate_grad` and `clear_grad` should ensure that the grad does not change size.

The `step` method performs a single step of stochastic gradient descent, updating the weights of the current layer based on a `step_size` parameter it takes and the current gradients. (This method does not need to support more advanced learning algorithms like Adam or Adagrad.)

You may even wish to write tests to confirm backpropagation is computed successfully by your network. If you do this, ensure your examples also include enough examples of `step` and `accumulate_grad` that you are comfortable with your coverage of those as well. Each test should test only a single layer, with input layers feeding that layer and receiving its gradients. The method `np.testing.assert_allclose` may again be useful for testing that a pytorch tensor has the values you expect. Again see the examples in `test_sum.py`

These derivations must be submitted on paper – see more details below.

## In-lab milestone

Consider working on the backpropagation equations first and demoing these during lab.

You are encouraged to try to complete this lab during the lab period if

possible.









## Submission Requirements

For this lab, you must submit:








- On paper:
  - Forward pass: Numeric examples supporting those layers that need them
  - Backward pass: Numeric examples for testing the `step()` method and the `backward()` method for each layer
- To Canvas: (Submit each file individually)
  - All of your `.py` and `.ipynb` files:
    - `network.py`
    - `layers.py`
    - `test_network.py`
    - `test_sum.py`
    - `test_....py` (any other test files you write)
    - And any other python or `.ipynb` files you use that are needed by the files above.

The paper portion of the lab MUST be turned in on paper. You are encouraged to submit this during lab or at the start of class on Monday. You may also slip it under your instructor's office door before the deadline.

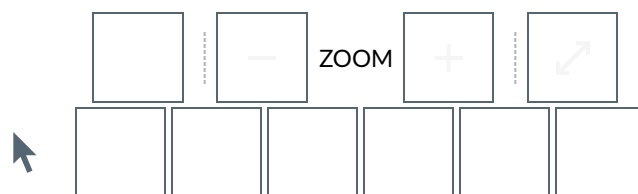
Thanks to Grant Fass for including the code for running unit tests within a Jupyter notebook.

	File Name	Size	
	<u><a href="#">test_mseloss.py</a></u>	1.17 KB	
	<u><a href="#">test_input.py</a></u>	1.07 KB	
	<u><a href="#">network.py</a></u>	3.23 KB	
	<u><a href="#">test_sum.py</a></u>	1.43 KB	



File Name	Size	
 <u>test_network.py</u>	2.1 KB	✓
 <u>test_softmax.py</u>	1.46 KB	✓
 <u>test_relu.py</u>	1.01 KB	✓
 <u>test_linear.py</u>	1.37 KB	✓
 <u>test_layer.py</u>	1.43 KB	✓
 <u>test_regu...zation.py</u>	1.18 KB	✓
 <u>layers.py</u>	14.1 KB	✓

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122  
123  
124  
125  
126  
127  
128  
129  
130  
131  
132  
133  
134  
135  
136  
137  
138  
139  
140  
141  
142  
143  
144  
145  
146  
147  
148  
149  
150  
151  
152  
153  
154  
155  
156  
157  
158  
159  
160  
161  
162  
163  
164  
165  
166  
167  
168  
169  
170  
171  
172  
173  
174  
175  
176  
177  
178  
179  
180  
181  
182  
183  
184  
185  
186  
187  
188  
189  
190  
191  
192  
193  
194  
195  
196  
197  
198  
199  
200  
201  
202  
203  
204  
205  
206  
207  
208  
209  
210  
211  
212  
213  
214  
215  
216  
217  
218  
219  
220  
221  
222  
223  
224  
225  
226  
227  
228  
229  
230  
231  
232  
233  
234  
235  
236  
237  
238  
239  
240  
241  
242  
243  
244  
245  
246  
247  
248  
249  
250  
251  
252  
253  
254  
255  
256  
257  
258  
259  
260  
261  
262  
263  
264  
265  
266  
267  
268  
269  
270  
271  
272  
273  
274  
275  
276  
277  
278  
279  
280  
281  
282  
283  
284  
285  
286  
287  
288  
289  
290  
291  
292  
293  
294  
295  
296  
297  
298  
299  
300  
301  
302  
303  
304  
305  
306  
307  
308  
309  
310  
311  
312  
313  
314  
315  
316  
317  
318  
319  
320  
321  
322  
323  
324  
325  
326  
327  
328  
329  
330  
331  
332  
333  
334  
335  
336  
337  
338  
339  
340  
341  
342  
343  
344  
345  
346  
347  
348  
349  
350  
351  
352  
353  
354  
355  
356  
357  
358  
359  
360  
361  
362  
363  
364  
365  
366  
367  
368  
369  
370  
371  
372  
373  
374  
375  
376  
377  
378  
379  
380  
381  
382  
383  
384  
385  
386  
387  
388  
389  
390  
391  
392  
393  
394  
395  
396  
397  
398  
399  
400  
401  
402  
403  
404  
405  
406  
407  
408  
409  
410  
411  
412  
413  
414  
415  
416  
417  
418  
419  
420  
421  
422  
423  
424  
425  
426  
427  
428  
429  
430  
431  
432  
433  
434  
435  
436  
437  
438  
439  
440  
441  
442  
443  
444  
445  
446  
447  
448  
449  
450  
451  
452  
453  
454  
455  
456  
457  
458  
459  
460  
461  
462  
463  
464  
465  
466  
467  
468  
469  
470  
471  
472  
473  
474  
475  
476  
477  
478  
479  
480  
481  
482  
483  
484  
485  
486  
487  
488  
489  
490  
491  
492  
493  
494  
495  
496  
497  
498  
499  
500  
501  
502  
503  
504  
505  
506  
507  
508  
509  
510  
511  
512  
513  
514  
515  
516  
517  
518  
519  
520  
521  
522  
523  
524  
525  
526  
527  
528  
529  
530  
531  
532  
533  
534  
535  
536  
537  
538  
539  
540  
541  
542  
543  
544  
545  
546  
547  
548  
549  
550  
551  
552  
553  
554  
555  
556  
557  
558  
559  
560  
561  
562  
563  
564  
565  
566  
567  
568  
569  
570  
571  
572  
573  
574  
575  
576  
577  
578  
579  
580  
581  
582  
583  
584  
585  
586  
587  
588  
589  
590  
591  
592  
593  
594  
595  
596  
597  
598  
599  
600  
601  
602  
603  
604  
605  
606  
607  
608  
609  
610  
611  
612  
613  
614  
615  
616  
617  
618  
619  
620  
621  
622  
623  
624  
625  
626  
627  
628  
629  
630  
631  
632  
633  
634  
635  
636  
637  
638  
639  
640  
641  
642  
643  
644  
645  
646  
647  
648  
649  
650  
651  
652  
653  
654  
655  
656  
657  
658  
659  
660  
661  
662  
663  
664  
665  
666  
667  
668  
669  
670  
671  
672  
673  
674  
675  
676  
677  
678  
679  
680  
681  
682  
683  
684  
685  
686  
687  
688  
689  
690  
691  
692  
693  
694  
695  
696  
697  
698  
699  
700  
701  
702  
703  
704  
705  
706  
707  
708  
709  
710  
711  
712  
713  
714  
715  
716  
717  
718  
719  
720  
721  
722  
723  
724  
725  
726  
727  
728  
729  
730  
731  
732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755  
756  
757  
758  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809  
810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863  
864  
865  
866  
867  
868  
869  
870  
871  
872  
873  
874  
875  
876  
877  
878  
879  
880  
881  
882  
883  
884  
885  
886  
887  
888  
889  
890  
891  
892  
893  
894  
895  
896  
897  
898  
899  
900  
901  
902  
903  
904  
905  
906  
907  
908  
909  
910  
911  
912  
913  
914  
915  
916  
917  
918  
919  
920  
921  
922  
923  
924  
925  
926  
927  
928  
929  
930  
931  
932  
933  
934  
935  
936  
937  
938  
939  
940  
941  
942  
943  
944  
945  
946  
947  
948  
949  
950  
951  
952  
953  
954  
955  
956  
957  
958  
959  
960  
961  
962  
963  
964  
965  
966  
967  
968  
969  
970  
971  
972  
973  
974  
975  
976  
977  
978  
979  
980  
981  
982  
983  
984  
985  
986  
987  
988  
989  
990  
991  
992  
993  
994  
995  
996  
997  
998  
999  
1000



```
from unittest import TestCase
import layers
import numpy as np
import torch
import unittest

class TestMSELoss(TestCase):
    """
    Please note: I (Dr. Yoder) may have assumed different p
    than you use.
    TODO: Update these tests to work with YOUR definitions
    """
```

```
variables.  
    """  
    def setUp(self):  
        self.yhat = layers.Input((2,2))  
        self.y = layers.Input((2,2), train=False)  
  
        self.yhat.set(torch.tensor([[0.94,0.6],[0.51,0.21]])  
        self.y.set(torch.eye(2).float())  
  
        self.mse = layers.MSELoss(self.yhat,self.y)  
  
    def test_forward(self):  
        self.mse.forward()  
  
        np.testing.assert_allclose(self.mse.output,  
                                    ((np.array([[.94, .6],[.51  
np.eye(2))**2).mean(axis=0).mean()  
  
    def test_backward(self):  
        self.mse.forward()  
        self.mse.accumulate_grad(torch.tensor([[ -1]]).float
```

[< Previous](https://msoe.instructure.com/courses/13814/modules/items/551946)

<https://msoe.instructure.com/courses/13814/modules/items/551946>

[Try Again](#)[Next >](https://msoe.instructure.com/courses/13814/modules/items/564452)

<https://msoe.instructure.com/courses/13814/modules/items/564452>