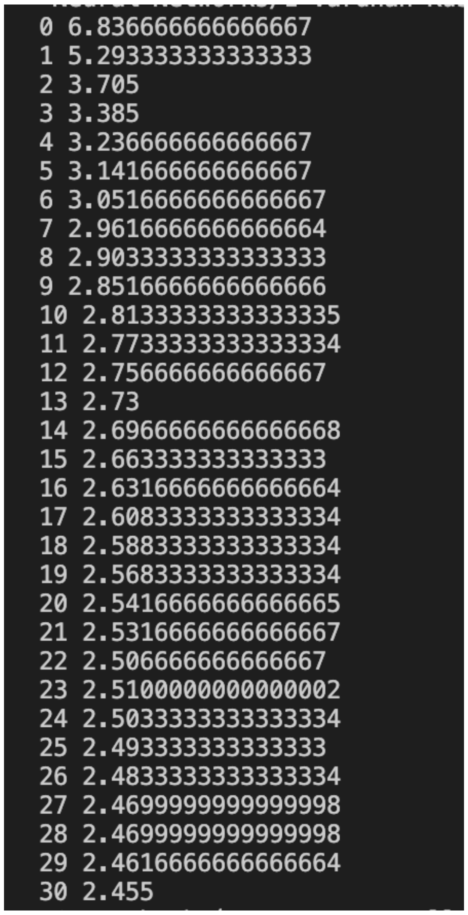
Implementation:

1. Pick a random number from [0,7) which will decide how we jitter the input
2. 0 means normal, the implementation that was present in blue - turn the 784 inputs into a 784x1 np.array
3. 1 means shift right: Go through lists of 28 in the dataset (one row), at each row, np. roll it over by 1. Add the rolled version to a final list that is later converted into an np.array and reshape into a 784x1 input array.
4. 2 means to shift left. Do the same as Step 3 however np.roll it over by -1.
5. 3 means to shift up. This time when we go through the lists of 28, we are not adjusting them, we are simply adding them to a final list that is a 28x28 list of lists. Turn that into an np.array and then np.roll it by -1 on axis = 0 (vertical shifting)
6. 4 means to shift down. Do the same as Step 5 however np.roll it by 1 still on the same axis
7. 5 means rotate to the right 15 degrees. Very similar to Step 5 in creating the list of lists structure. Then use scipy.ndimage.rotate() with reshape=False (very important because this ensures that we don't change the size of the dataset) to rotate the array 15 degrees and then convert it into an np.array and normalize it.
8. 6 means to shift to the left 15 degrees. Do the same as Step 7 but scipy.ndimage.rotate() by -15 degrees.

Comparing:

Whoa. While the code was running, I was almost certain that my outcome from "jittering" would be worse. As seen in the screenshot below, 30 epochs started off very similarly in both networks (distorted and undistorted), but at about the 8th or 9th epoch diverged greatly. Undistorted continued to decrease at an almost constant rate, and got as low as 0.97% inaccuracy on the training set as it was training. The distorted network? While it decreased, it was MUCH slower. It got down to about 2.81% inaccuracy pretty rapidly and in a similar pace, but after that by the end of 30 epochs only got to 2.45% inaccuracy on the training set while it trained. However, this is not where the story ends. I then ran both networks on the TEST set. Here is where I was shocked. The undistorted network got 3.21% inaccuracy, which is pretty good, but wildly higher than the inaccuracy "predicted" while training (should have been sub 1%). As for the distorted network, well that got a 2.5% inaccuracy! Not a drastic improvement, but way closer to the "predicted" accuracy (2.45%) and still LOWER than the undistorted set. Cool stuff!

Note: Network architecture was identical for both cases - [784, 300, 100, 10]



```
0 6.836666666666667
1 5.293333333333333
2 3.705
3 3.385
4 3.236666666666667
5 3.141666666666667
6 3.0516666666666667
7 2.9616666666666664
8 2.9033333333333333
9 2.8516666666666666
10 2.8133333333333335
11 2.7733333333333334
12 2.756666666666667
13 2.73
14 2.6966666666666668
15 2.663333333333333
16 2.6316666666666664
17 2.6083333333333334
18 2.5883333333333334
19 2.5683333333333334
20 2.5416666666666665
21 2.5316666666666667
22 2.506666666666667
23 2.5100000000000002
24 2.5033333333333334
25 2.493333333333333
26 2.4833333333333334
27 2.4699999999999998
28 2.4699999999999998
29 2.4616666666666664
30 2.455
```