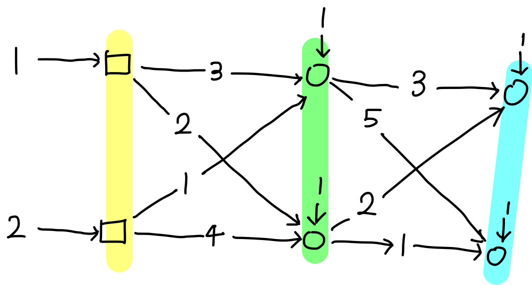
1



in | hidden | output

$$\begin{array}{c|c|c|c|c}
\text{in} & & \text{hidden} & & \text{output} \\
1 & \underbrace{3\times1 + 1\times2}_{5} + 1 = & 6 & \underbrace{3\times6 + 2\times11}_{40} + 1 = & 41 \\
2 & \underbrace{2\times1 + 4\times2}_{10} + 1 = & 11 & \underbrace{5\times6 + 1\times11}_{41} + 1 = & 42
\end{array}$$

weighted sum (1-1) | out (1-2) | weighted sum | out

$$v_{hidden} = \begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

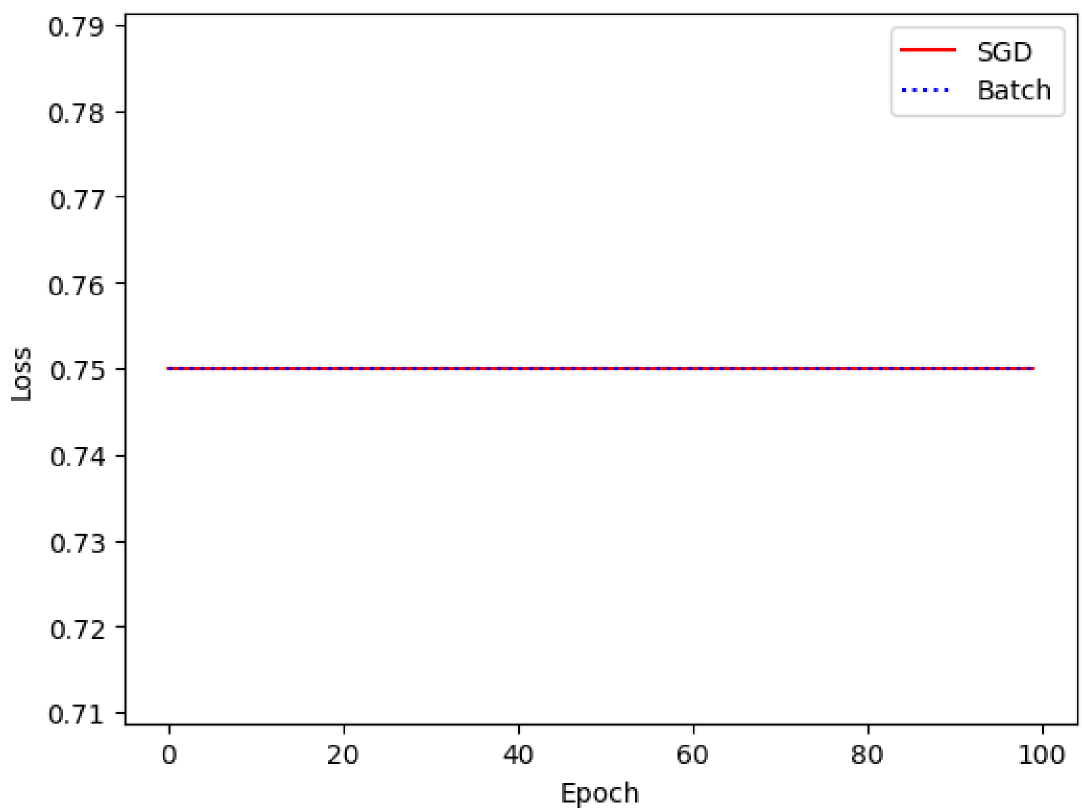
$$\begin{aligned}
v_{out} &= \begin{bmatrix} 3 & 2 \\ 5 & 1 \end{bmatrix} v_{hidden} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 3 & 2 \\ 5 & 1 \end{bmatrix}\left(\begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 3 & 2 \\ 5 & 1 \end{bmatrix}\begin{bmatrix} 3 & 1 \\ 2 & 4 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 3 & 2 \\ 5 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 13 & 11 \\ 17 & 9 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \end{bmatrix} + \begin{bmatrix} 6 \\ 7 \end{bmatrix}
\end{aligned}$$

homework1-2

March 9, 2025

2-1) Run the code using the code given in the next page without any change.

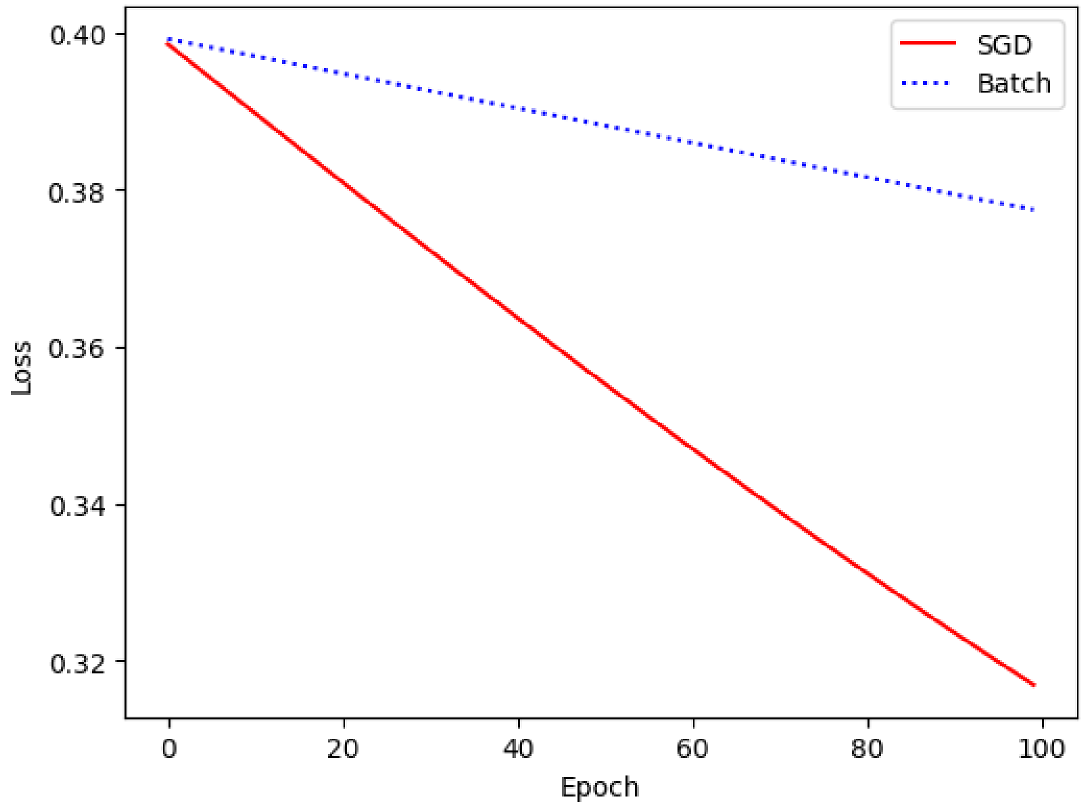
```
input(0,0), predicted output=1.0, desired output=0
input(0,1), predicted output=1.0, desired output=0
input(1,0), predicted output=1.0, desired output=0
input(1,1), predicted output=1.0, desired output=1
```



2-2) Using the code from 2-1, modify the initial weight as `W1 = [[0.68, 0.01, 0.73]]` and run the code.

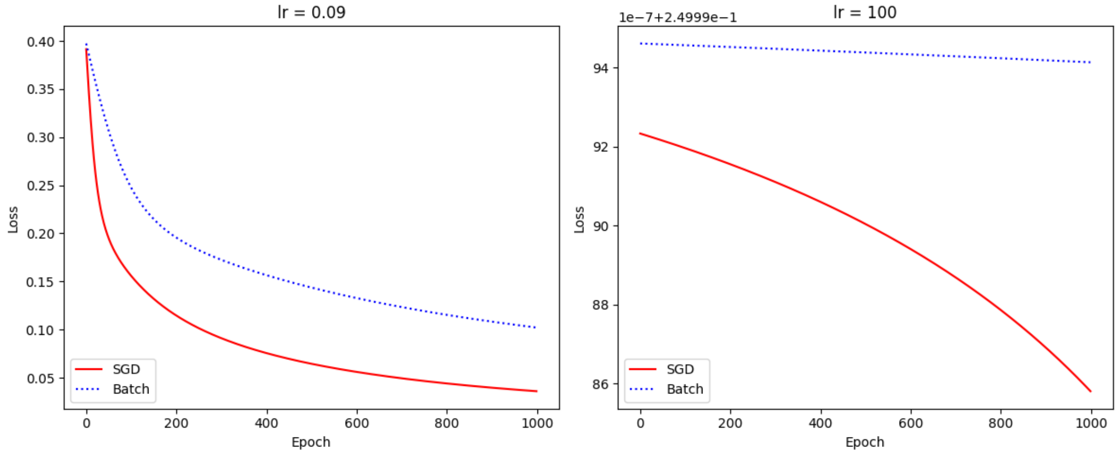
```
input(0,0), predicted output=0.5859495731959541, desired output=0
input(0,1), predicted output=0.5641976264287257, desired output=0
input(1,0), predicted output=0.7182932133223229, desired output=0
```

```
input(1,1), predicted output=0.6999334244241662, desired output=1
```



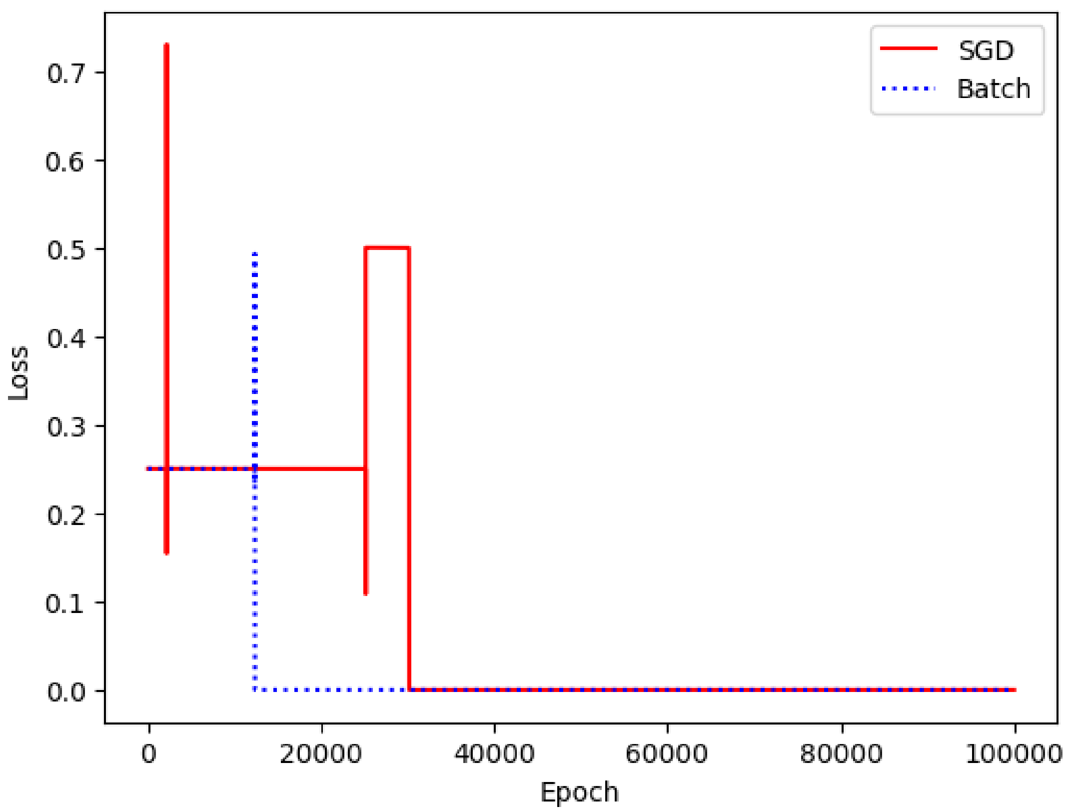
2-3) Using the code from 2-2, modify the learning rate to `lr = 0.9` and run the code. Then, modify the learning rate to `lr = 100` and run the code.

```
-----------------lr = 0.09-----------------
input(0,0), predicted output=0.020829578517260007, desired output=0
input(0,1), predicted output=0.20285630268480684, desired output=0
input(1,0), predicted output=0.20389394428823304, desired output=0
input(1,1), predicted output=0.7539260430488409, desired output=1
-----------------lr = 100-----------------
input(0,0), predicted output=9.442710248783476e-07, desired output=0
input(0,1), predicted output=1.171100244905025e-06, desired output=0
input(1,0), predicted output=2.2886046687843648e-06, desired output=0
input(1,1), predicted output=2.838363723947364e-06, desired output=1
```



2-4) (10 pts) Using the code from 2-3 (`lr=100`), modify the epochs to `epochs = 100000` and run the code.

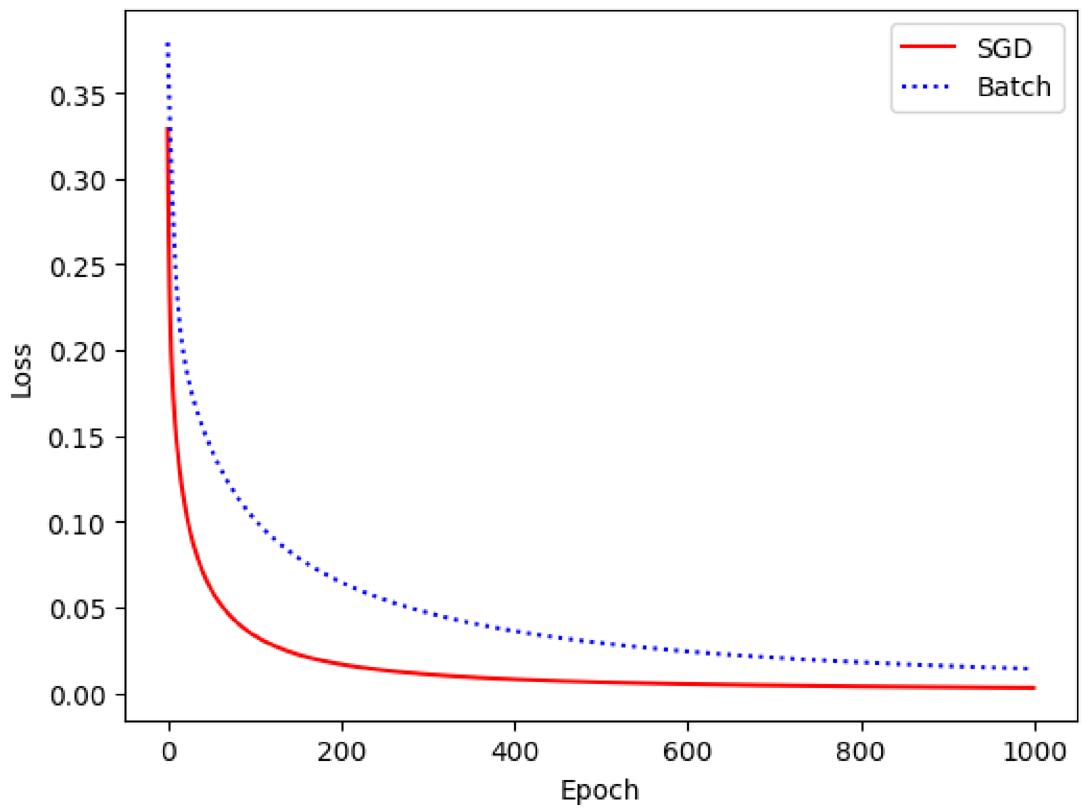
```
input(0,0), predicted output=1.9296590958690417e-14, desired output=0
input(0,1), predicted output=0.0004205020075436615, desired output=0
input(1,0), predicted output=1.2192186162009744e-07, desired output=0
input(1,1), predicted output=0.9996239160001456, desired output=1
```



2-5) Run the code using the parameters below. Now let's compare Batch (delta_batch) and SGD (delta_sgd) methods. Which one shows faster error convergence? (You can type and run a command `print(E1[-1],E2[-1])` to compare the final errors quantitatively.)

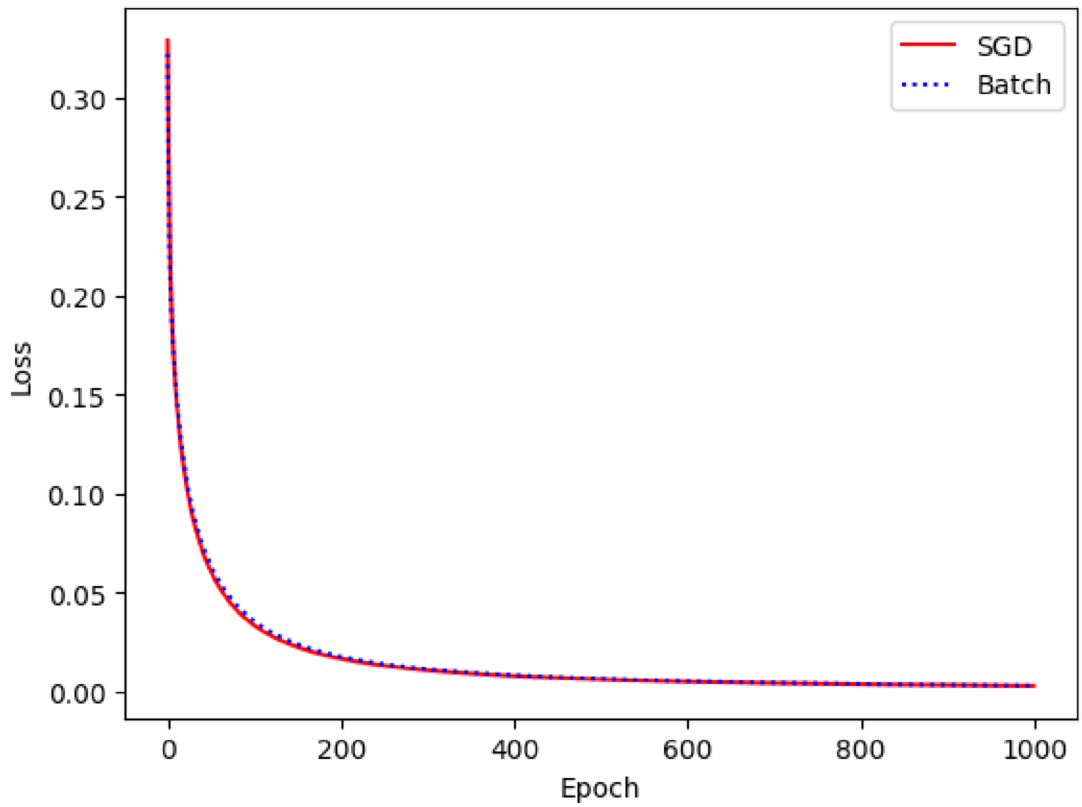
- `W1 = [[0.68 0.01 0.73]]`
- `lr = 0.9`
- `epochs=1000`

```
input(0,0), predicted output=0.00028582202096891256, desired output=0
input(0,1), predicted output=0.05844703939833359, desired output=0
input(1,0), predicted output=0.05864643215392242, desired output=0
input(1,1), predicted output=0.9311604874841752, desired output=1
```



SGD exhibits faster error convergence; however, this is a trivial result. The reason is that the SGD delta update takes a step that is `len(D)` times larger than the batch delta update in one epoch. If we modify the weight update rule from `W = W + dWavg` to `W = W + dWsum`, the convergence behavior becomes similar between the two methods.

```
input(0,0), predicted output=0.00028582202096891256, desired output=0
input(0,1), predicted output=0.05844703939833359, desired output=0
input(1,0), predicted output=0.05864643215392242, desired output=0
input(1,1), predicted output=0.9311604874841752, desired output=1
```


SGD
Batch
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Loss
0
200
400
600
800
1000
Epoch