Deep learning Ex1

Ariel Radomislensky 211991146

Question 1

1.

$$softmax_{[i]}(z + m \cdot \mathbf{1}) = \frac{\exp\,(z_i + m)}{\sum_j \exp\left(z_j + m\right)} = \frac{\exp(z_i)\exp(m)}{\exp(m)\sum_j \exp\left(z_j\right)} = \frac{\exp(z_i)}{\sum_j \exp\left(z_j\right)}$$
$$= softmax_{[i]}(z)$$

2.

for $c = 2$:

$$z = \begin{bmatrix} z_0 \\ z_1 \end{bmatrix}, \qquad softmax_{[0]}(z) = \frac{\exp(z_0)}{\exp(z_0) + \exp(z_1)} = \frac{1}{1 + \exp(z_1 - z_0)} = \sigma(z_0 - z_1)$$
$$softmax_{[1]}(z) = \frac{\exp(z_1)}{\exp(z_0) + \exp(z_1)} = \frac{1}{1 + \exp(z_0 - z_1)} = \sigma(z_1 - z_0)$$

3.

$$f(z) = \frac{1}{\pi}\arctan(z) + \frac{1}{2}$$

Question 3

1.

$$U = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, b_1 = \mathbf{-0.5}, w = \mathbf{1}, b_2 = \mathbf{-0.5}$$

```
import numpy as np

U = np.array([[1,-1],[-1,1]])
b1 = -0.5
w = np.array([1,1]).T
b2 = -0.1

X = [np.array([0,0]).T, np.array([0,1]).T,
np.array([1,0]).T, np.array([1,1]).T]
for x in X:
  h =  np.maximum(U.T@x + b1, np.zeros((2,)))
  print("For x = ", x, "hidden layer output is ", h)
  f = w.T@h + b2

  if(f>=0):
    print("For x = ", x, "output is 1")
  else:
    print("For x = ", x, "output is 0")
```

```
For x =  [0 0] hidden layer output is  [0. 0.]
For x =  [0 0] output is 0
For x =  [0 1] hidden layer output is  [0.  0.5]
For x =  [0 1] output is 1
For x =  [1 0] hidden layer output is  [0.5 0. ]
For x =  [1 0] output is 1
For x =  [1 1] hidden layer output is  [0. 0.]
For x =  [1 1] output is 0
```

verified using python:

2. No, it would not be possible, replacing the max function with the identity will effectively eliminate the hidden layer:
$$f(x) = w^T(U^Tx + b_1) + b_2 = w^TU^Tx + w^Tb_1 + b_2$$
$$w' = Uw, b' = w^Tb_1 + b_2$$
$$\Downarrow$$
$$f(x) = w'^Tx + b'$$
turning f into a linear classifier, and XOR cannot be linearly classified.
3. ReLU(x) can in fact be defined as max(x,0). Maybe the question meant weather we can set $b_1 = 0$? If so yes, it is still possible:

<table>
<tr>
<td>

```
import numpy as np

U = np.array([[1,-1],[-1,1]])
b1 = 0
w = np.array([1,1]).T
b2 = -0.1

X = [np.array([0,0]).T, np.array([0,1]).T,
np.array([1,0]).T, np.array([1,1]).T]
for x in X:
  h =  np.maximum(U.T@x + b1, np.zeros((2,)))
  print("For x = ", x, "hidden layer output is ", h)
  f = w.T@h + b2

  if(f>=0):
    print("For x = ", x, "output is 1")
  else:
    print("For x = ", x, "output is 0")
```

</td>
<td>

```
For x =  [0 0] hidden layer output is  [0. 0.]
For x =  [0 0] output is 0
For x =  [0 1] hidden layer output is  [1. 0.]
For x =  [0 1] output is 1
For x =  [1 0] hidden layer output is  [0. 1.]
For x =  [1 0] output is 1
For x =  [1 1] hidden layer output is  [0. 0.]
For x =  [1 1] output is 0
```

</td>
</tr>
</table>

Question 3

To use backward propagation, we need to calculate the gradients:

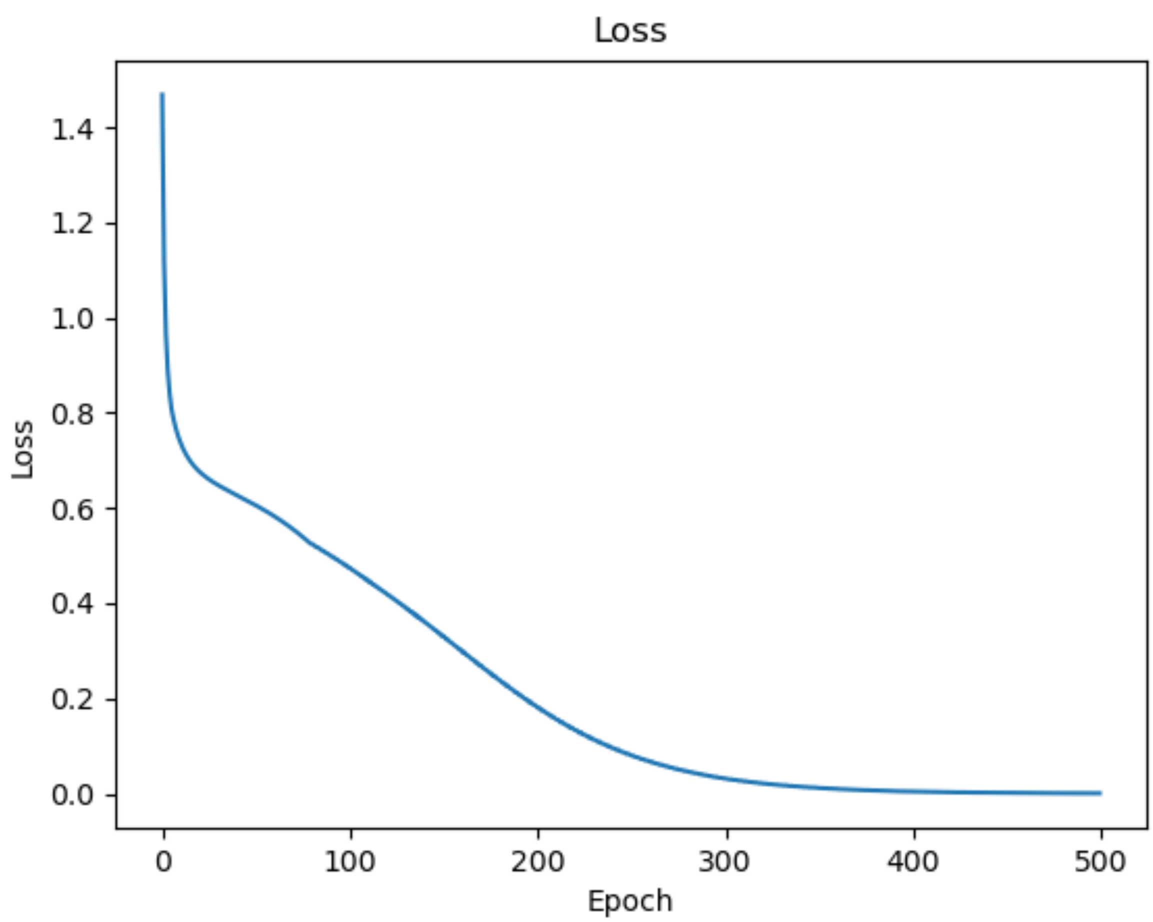
$$L = (y - \hat{y})^2$$
$$\hat{y} = f(x) = w^Th + b_2 = \mathrm{w}_1\mathrm{h}_1 + \mathrm{w}_2\mathrm{h}_2 + \mathrm{b}_2$$
$$h = ReLU(U^Tx + b_1) = ReLU(z) \Rightarrow h'(z) = \mathcal{U}(z)$$
$$z = U^Tx + b_1 = \begin{bmatrix} U_{1,1}x_1 + U_{2,1}x_2 + b_{1,1} \\ U_{1,2}x_1 + U_{2,2}x_2 + b_{1,2} \end{bmatrix}$$

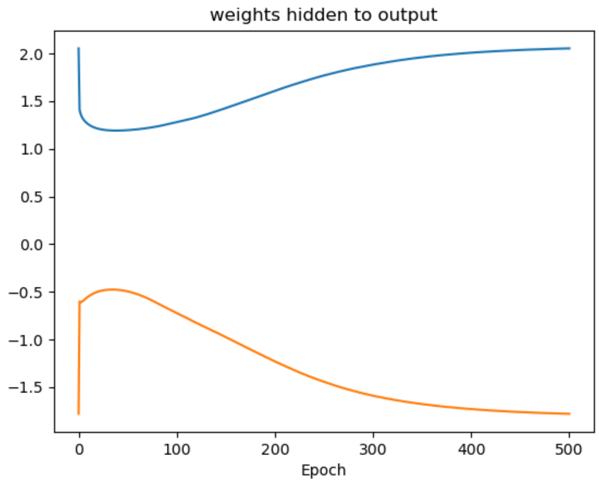
$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial w} = -2(y - \hat{y})h_i$$

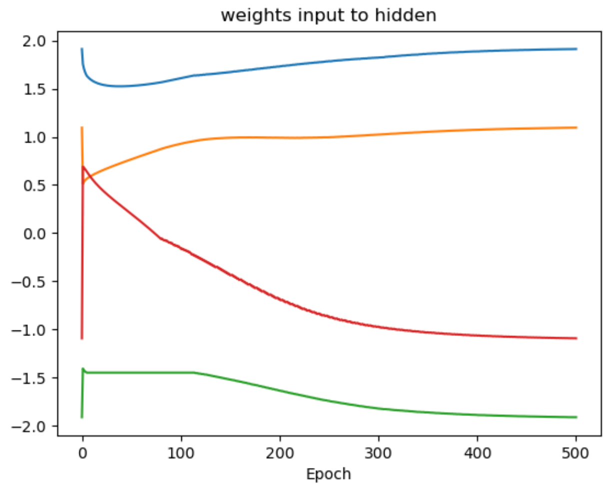
$$\frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial b_2} = -2(y - \hat{y})$$
$$\frac{\partial L}{\partial U_{ij}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_j}\frac{\partial h_j}{\partial z_j}\frac{\partial z_j}{\partial U_{ij}} = -2(y - \hat{y})\, w_j \mathcal{U}(z_j)\, x_i$$
$$\frac{\partial L}{\partial b_{1,i}} = \frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_i}\frac{\partial h}{\partial z_i}\frac{\partial z_i}{\partial b_{1,i}} = -2(y - \hat{y})\, w_i\, \mathcal{U}(z_i)$$

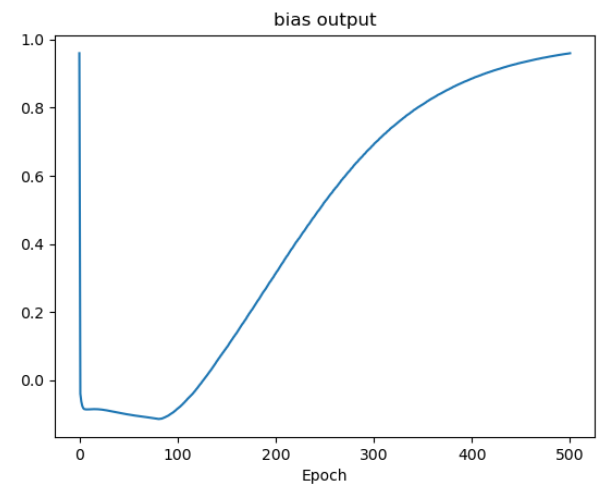
And implement in the code.
The code uses pandas for visualization, it can be easily installed via `pip install pandas`
To run the code, open a terminal window, cd into the directory where you stored the script, and run it with `python3 Q3.py`, the code will generate 5 png files.

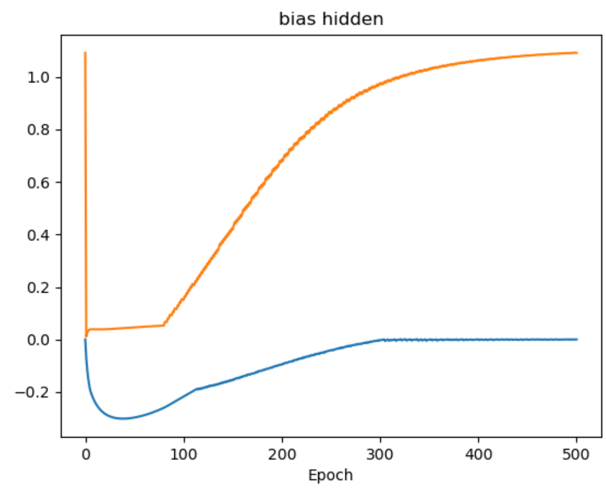
Here are the plots from my run with 500 epochs:











Challenges:

Initially I tried creating the weights and biases history by simply appending the numpy array to a list, I was surprised to see that the result was the same number. My mistake was appending the actual array which is static in memory instead of a copy of it.

Going from calculus to python code was challenging, I ended taking a matlab approach of representing everything as a matrix.