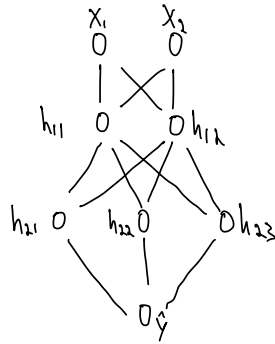


$$\begin{aligned}
 1.1 \quad & W_0 \in \mathbb{R}^{2 \times 2} \\
 & W_1 \in \mathbb{R}^{3 \times 2} \\
 & W_2 \in \mathbb{R}^{1 \times 3}
 \end{aligned}$$



a) weights: Input  $\rightarrow H1 = 2 \times 2 = 4$   
 $H1 \rightarrow H2 = 3 \times 2 = 6$   
 $H2 \rightarrow \text{Output} = 1 \times 3 = 3$

$$\text{total} = 4 + 6 + 3 = 13$$

$$b) \vec{h}_1 = W_0 \vec{x} = \begin{pmatrix} 0.1 & 0.4 \\ -0.5 & 0.6 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} \rightarrow \text{ReLU} \rightarrow \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}$$

$$\vec{h}_2 = W_1 \vec{h}_1 = \begin{pmatrix} 0.2 & 0.1 \\ 0.5 & 0.4 \\ 0.3 & 0.6 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.09 \\ 0.29 \\ 0.09 \end{pmatrix} \rightarrow \text{ReLU} \rightarrow \begin{pmatrix} 0.09 \\ 0.29 \\ 0.09 \end{pmatrix}$$

$$\hat{y} = W_2 \vec{h}_2 = (0.7 \ 0.4 \ 0.9) \begin{pmatrix} 0.09 \\ 0.29 \\ 0.09 \end{pmatrix} = 0.028 \rightarrow \text{ReLU} \rightarrow \underline{0.028}$$

$$c) \vec{h}_1 = W_0 \vec{x} = \begin{pmatrix} 0.1 & 0.4 \\ -0.5 & 0.6 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} \rightarrow \text{Sigmoid} \rightarrow \begin{pmatrix} (1 + e^{-0.5})^{-1} \\ (1 + e^{-0.1})^{-1} \end{pmatrix} = \begin{pmatrix} 0.6225 \\ 0.525 \end{pmatrix}$$

$$d) \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial W_2} = (\hat{y} - y) \vec{h}_2 = (0.028 - y) \begin{pmatrix} 0.09 \\ 0.29 \\ 0.09 \end{pmatrix}, \text{Shape: } 3 \times 1$$

$$1.2 \quad a) \vec{h}_1 = W_0 \vec{x} = \begin{pmatrix} 0.2 & 0.3 \\ 0 & -0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -0.1 \end{pmatrix} \rightarrow \text{ReLU} \rightarrow \begin{pmatrix} 0.5 \\ 0 \end{pmatrix}$$

$$\vec{h}_2 = W_1 \vec{h}_1 = \begin{pmatrix} 0.4 & 0 \\ 0 & 0.6 \\ -0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0 \\ -0.1 \end{pmatrix} \rightarrow \text{ReLU} \rightarrow \begin{pmatrix} 0.2 \\ 0 \\ 0 \end{pmatrix}$$

$$\hat{y} = W_2 \vec{h}_2 = (0.3 \ 0.5 \ 0) \begin{pmatrix} 0.2 \\ 0 \\ 0 \end{pmatrix} = 0.06$$

$$b) \vec{z}_1 = \begin{pmatrix} 0.5 \\ -0.1 \end{pmatrix}$$

$$\mu = \frac{1}{N} \sum_{i=1}^N z_i, \quad \sigma^2 = \frac{1}{N} \sum_{i=1}^N (z_i - \mu)^2$$

$$\mu = \frac{0.5 - 0.1}{2} = 0.2, \quad \sigma^2 = \frac{(0.5 - 0.2)^2 + (-0.1 - 0.2)^2}{2} = 0.09$$

$$\hat{z}_i = \frac{z_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \quad \text{where } \epsilon \approx 0$$

$$\hat{z}_1 = \begin{pmatrix} \frac{0.5 - 0.2}{\sqrt{0.09}} \\ \frac{-0.1 - 0.2}{\sqrt{0.09}} \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

$$\vec{h}_1 = \text{ReLU} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\vec{h}_2 = W_1 \vec{h}_1 = \begin{pmatrix} 0.4 & 0 \\ 0 & 0.6 \\ -0.2 & 0.1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0.4 \\ 0 \\ -0.2 \end{pmatrix} \rightarrow \text{ReLU} \rightarrow \begin{pmatrix} 0.4 \\ 0 \\ 0 \end{pmatrix}$$

$$\hat{y} = W_2 \vec{h}_2 = (0.3 \ 0.5 \ 0) \begin{pmatrix} 0.4 \\ 0 \\ 0 \end{pmatrix} = 0.12$$

Normalization helps stabilize the network.

(S) please see report at bottom of document.

2. a)  $y_i(w^T x_i - b) \geq 1 - \epsilon$

$$1(0.5(2) - 0.5(6) - 1) \geq 1 - \epsilon$$

$$(1 - 3 - 1) \geq 1 - \epsilon$$

$$-3 \geq 1 - \epsilon$$

$$\epsilon \geq 4$$

$$-1(0.5(6) - 0.5(2) - 1) \geq 1 - \epsilon$$

$$-1 \geq 1 - \epsilon$$

$$\epsilon \geq 2$$

$$1(0.5(4) - 0.5(6) - 1) \geq 1 - \epsilon$$

$$-2 \geq 1 - \epsilon$$

$$\epsilon \geq 3$$

$$-1(0.5(5) - 0.5(3) - 1) \geq 1 - \epsilon$$

$$-1(2.5 - 1.5 - 1) \geq 1 - \epsilon$$

$$\epsilon \geq 1$$

$$X = \begin{pmatrix} 2 & 6 \\ 6 & 2 \\ 4 & 6 \\ 5 & 3 \end{pmatrix}$$

no need to use

$$\max(0, 1 - y_i(w^T x - b))$$

Since  $1 - y_i(w^T x - b)$  is

always positive

All constraints hold

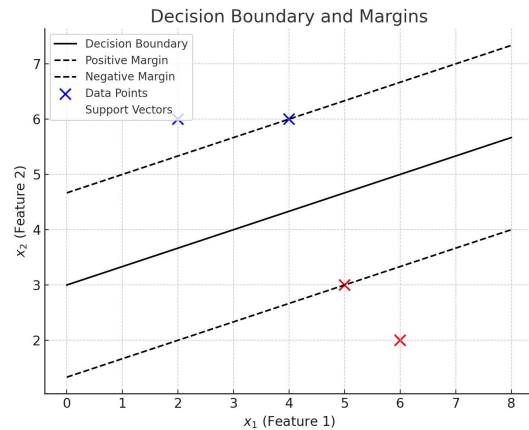
6)

```
from sklearn.svm import SVC
import numpy as np

X = np.array([[2, 6], [6, 2], [4, 6], [5, 3]])
y = np.array([1, -1, 1, -1])

model = SVC(kernel='linear', C=1.0)
model.fit(X, y)

w_star = model.coef_[0]
b_star = model.intercept_[0]
print('w*:', w_star)
print('b*:', b_star)
```



With the coding implementation, we get  $w = (-0.2, 0.6)$ ,  $b = 1.8$

Find objective value:  $y_i(w^T x_i - b^*)$

$$\begin{array}{llll}
 1((-0.2 \ 0.6) \begin{pmatrix} 2 \\ 6 \end{pmatrix} - 1.8) - ((-0.2 \ 0.6) \begin{pmatrix} 6 \\ 1 \end{pmatrix} - 1.8) & 1((-0.2 \ 0.6) \begin{pmatrix} 4 \\ 6 \end{pmatrix} - 1.8) & -1((-0.2 \ 0.6) \begin{pmatrix} 5 \\ 3 \end{pmatrix} - 1.8) \\
 -0.4 + 3.6 - 1.8 = 1.4 & 1.2 - 1.2 + 1.8 = 1.8 & -0.8 + 3.6 - 1.8 = 1 & 1 - 1.8 + 1.8 = 1 \\
 \epsilon = \max(0, 1 - 1.4) & \epsilon = \max(0, 1 - 1.8) & \epsilon = \max(0, 1 - 1) & \epsilon = \max(0, 1 - 1) \\
 \epsilon = 0 & \epsilon = 0 & \epsilon = 0 & \epsilon = 0
 \end{array}$$

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \epsilon_i$$

$$\frac{1}{2} \sqrt{-.2^2 + 0.6^2} + 0$$

$$\frac{1}{2} \sqrt{0.04 + 0.36}$$

$$\frac{1}{2} (0.4) = 0.2$$

High C means narrow margin, few misclassifications allowed.  
Low C means the opposite

c) Support vectors: (4, 6), (5, 3)

$$y_i(w^T x_i - b)$$

$$1((-0.2 \ 0.6) \begin{pmatrix} 4 \\ 6 \end{pmatrix} - 1.8) = -0.8 + 3.6 - 1.8 = 1 \rightarrow \text{Lies on the margin}$$

$$-1((-0.2 \ 0.6) \begin{pmatrix} 5 \\ 3 \end{pmatrix} - 1.8) = -1(-1 + 1.8 - 1.8) = 1 \rightarrow \text{Lies on the margin}$$

d) a) code for Gram:  $K = \text{np.dot}(X, X.T)$

b)  $\lambda$  values: [0, 0, 0.19995376, 0.19995376]

c) Done for us in code

d) Verification

$$w^* = \sum_{i=1}^N \lambda_i y_i x_i$$

first two  $\lambda_i$  values are zero, so we have

$$= (0.19995376)(1)(4, 6) + (0.19995376)(-1)(5, 3)$$

$$= (0.799815, 1.1997226) + (-0.999765, -0.599993)$$

$$w^* = (-0.19995, 0.5997931)$$

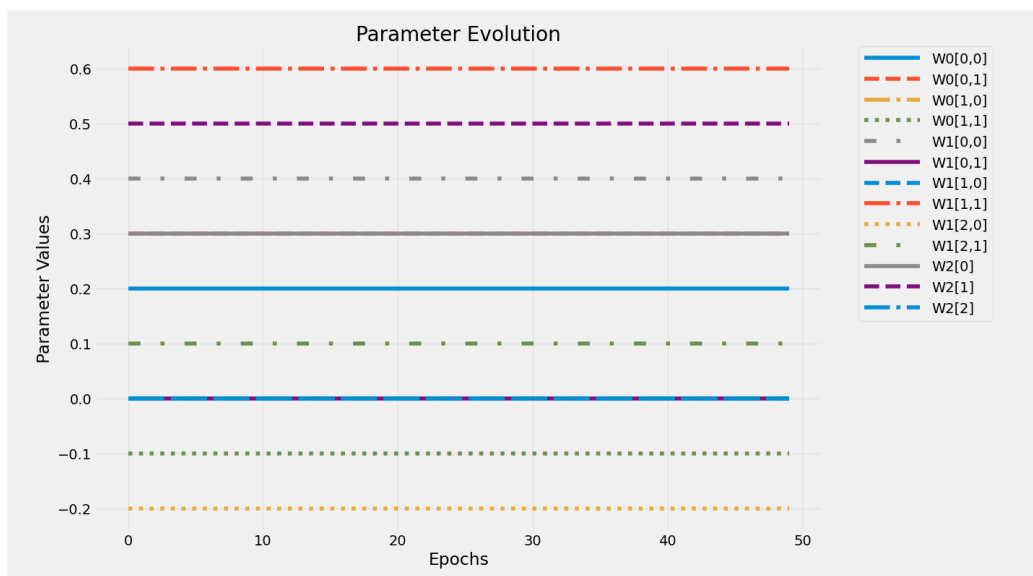
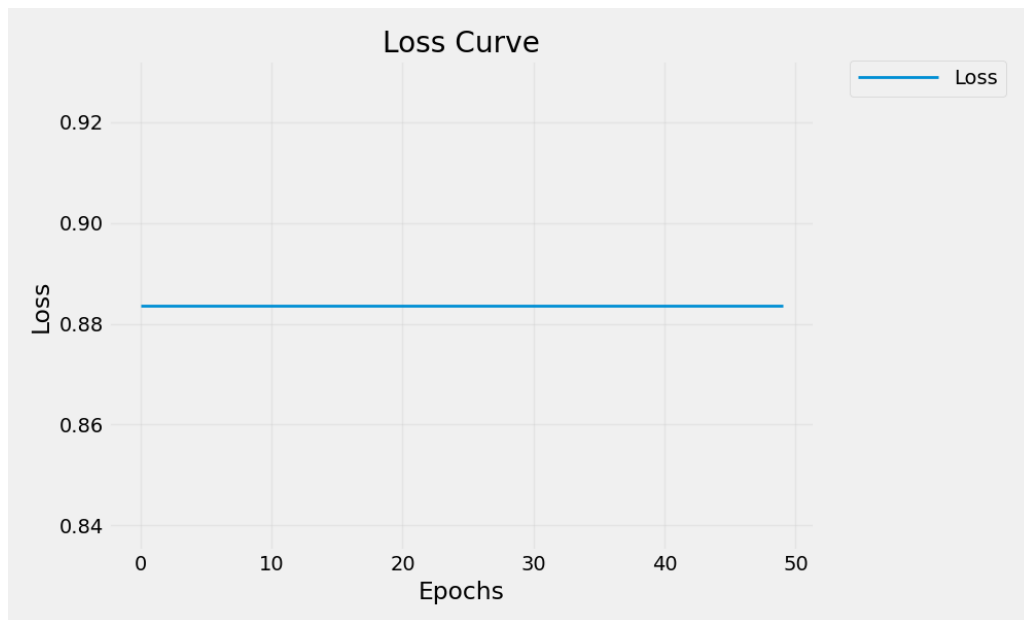
The values match. (Slight error due to rounding)

e) Taking the derivative of the Generalized Lagrangian w.r.t  $w$  gets us  $w = \sum_{i=1}^N \lambda_i y_i x_i$ , which links primal to dual. We can see from our  $\lambda_i$  values which vectors are support (nonzero  $\lambda$  values), and these will zero out the respective  $\lambda_i y_i x_i$  terms.

f) If we had a larger dataset, computing the Gram matrix would be slow as it is  $O(N^2)$ . It would also use  $O(N^2)$  memory. The linear kernel is a good, fast option. Full batch gradient descent would run slow. SGD would be a faster option.

i) Initial Run: Run the provided network as-is. Document the loss curve and explain why the network fails to converge.

The algorithm fails to converge initially because we are only marking the zero value weights as trainable. Since there is no bias term being used, and  $\text{ReLU}(0, \max(0)) = 0$ , these values will remain zero throughout the network. The gradient will also be zero due to this, so no learning can occur. Learning rate also appears to be a little low.

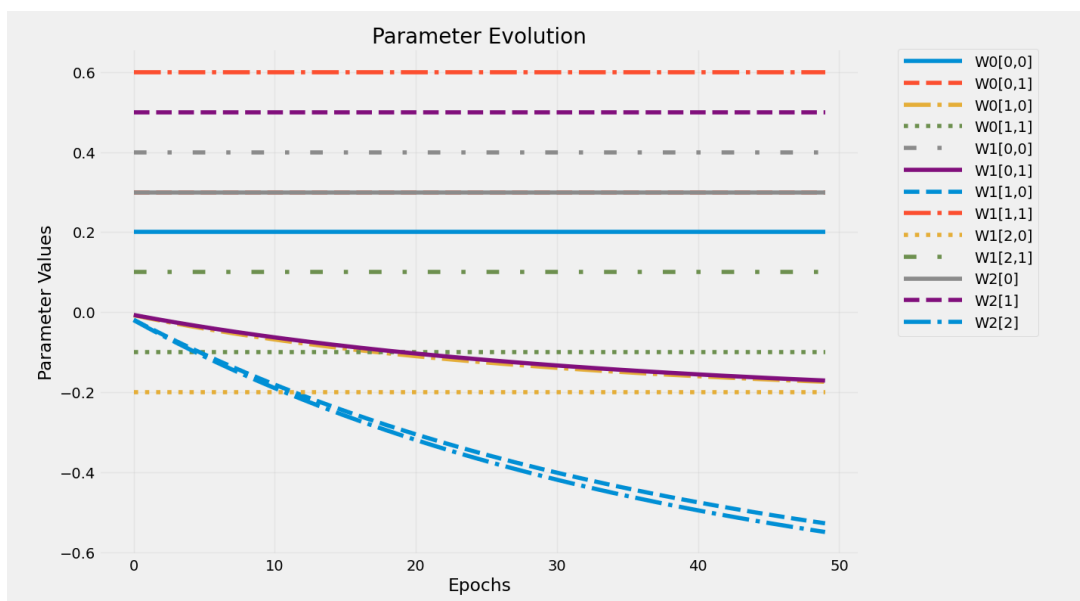
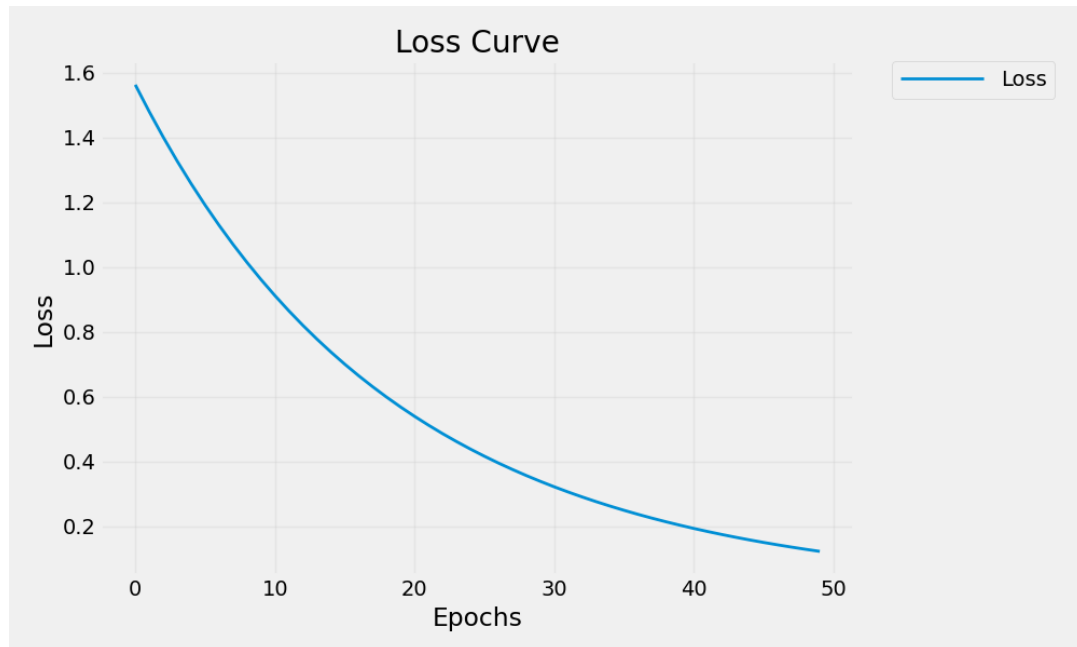


ii) Modify and Observe: Make one modification at a time. Document each noteworthy change, including plots showing the effects on loss and parameter evolution.

### 1. Adding bias

We can see that simply adding a 1 for the bias, so we do not have dead neurons and gradients everywhere, allows the network to improve its loss significantly. Only the selected weights from the masks are able to be learned from, however.

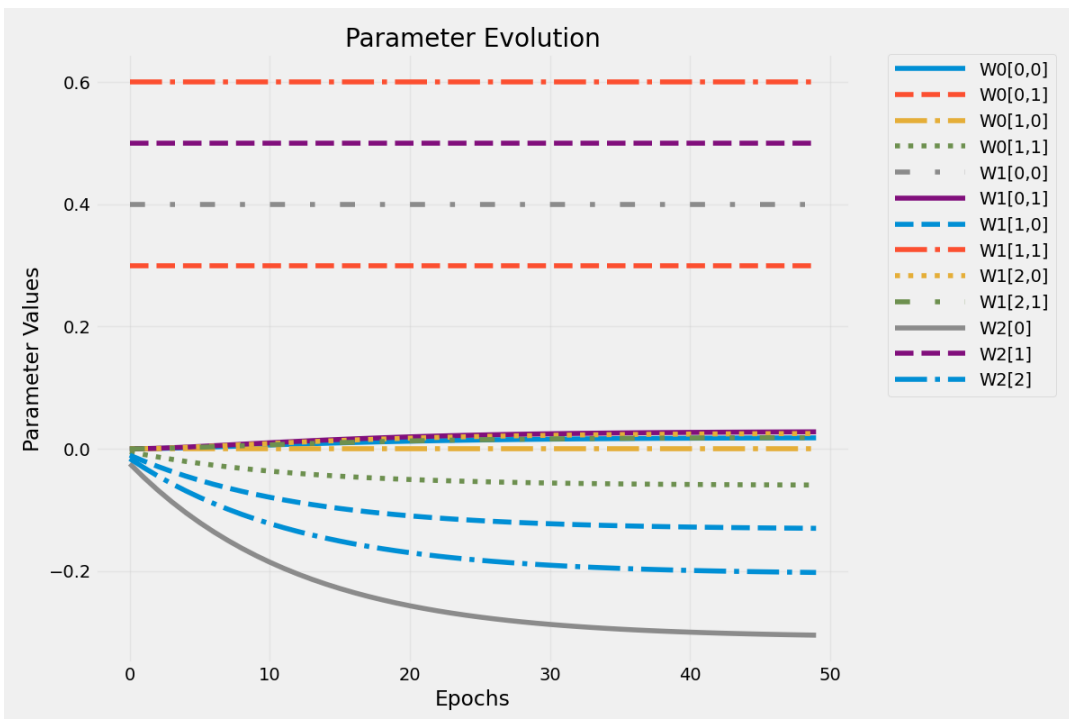
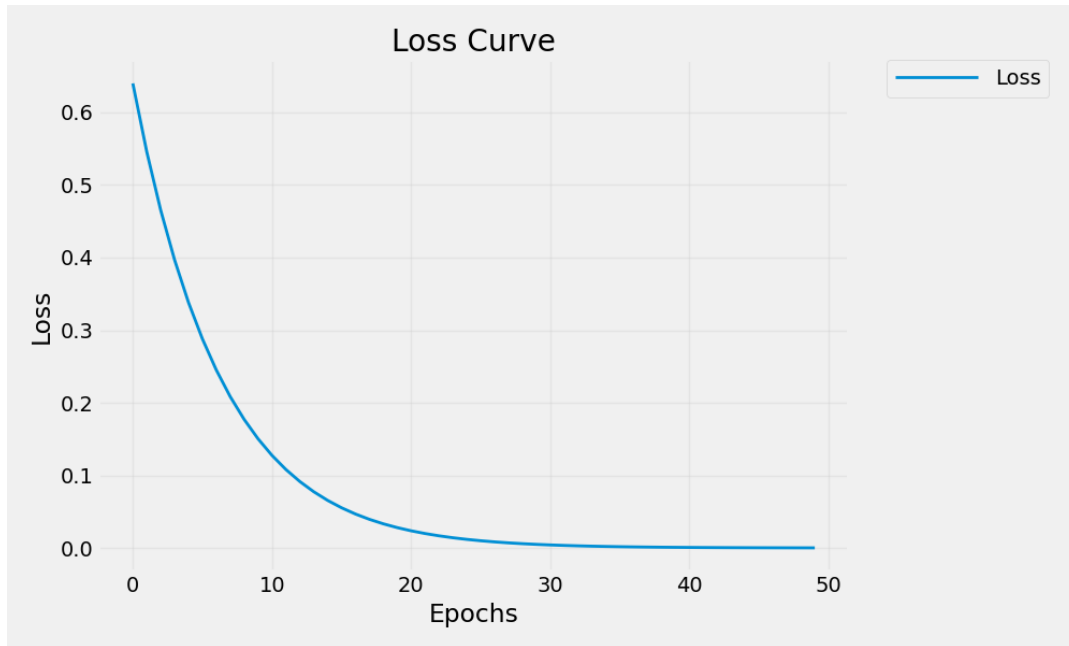
Epoch 50: Loss = 0.12237604707479477,  $y_{\text{pred}} = 1.3498228788375854$



## 2. Marking random weights as trainable

Since we're only using the zero weights as trainable, it would be a good idea to randomly mark some weights as trainable because we are ignoring the majority of them, and having the same starting value for each trainable weight may not be the best idea.

Epoch 50: Loss = 0.00014321059279609472, y\_pred = 1.0119670629501343

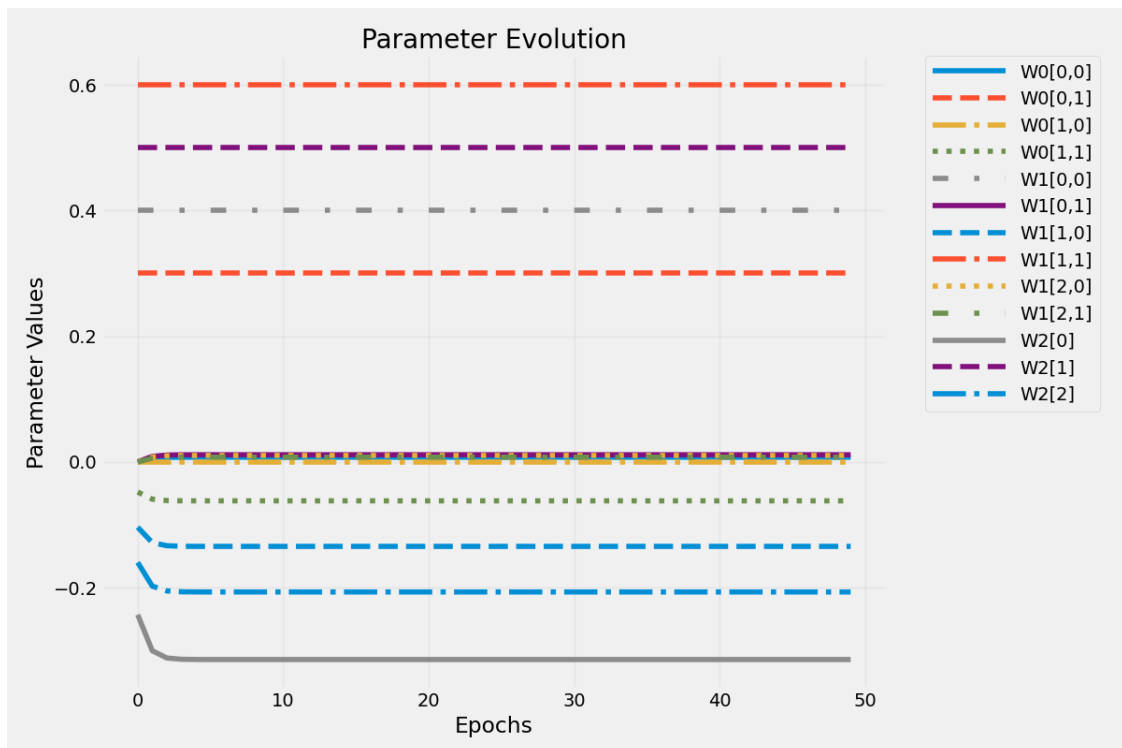
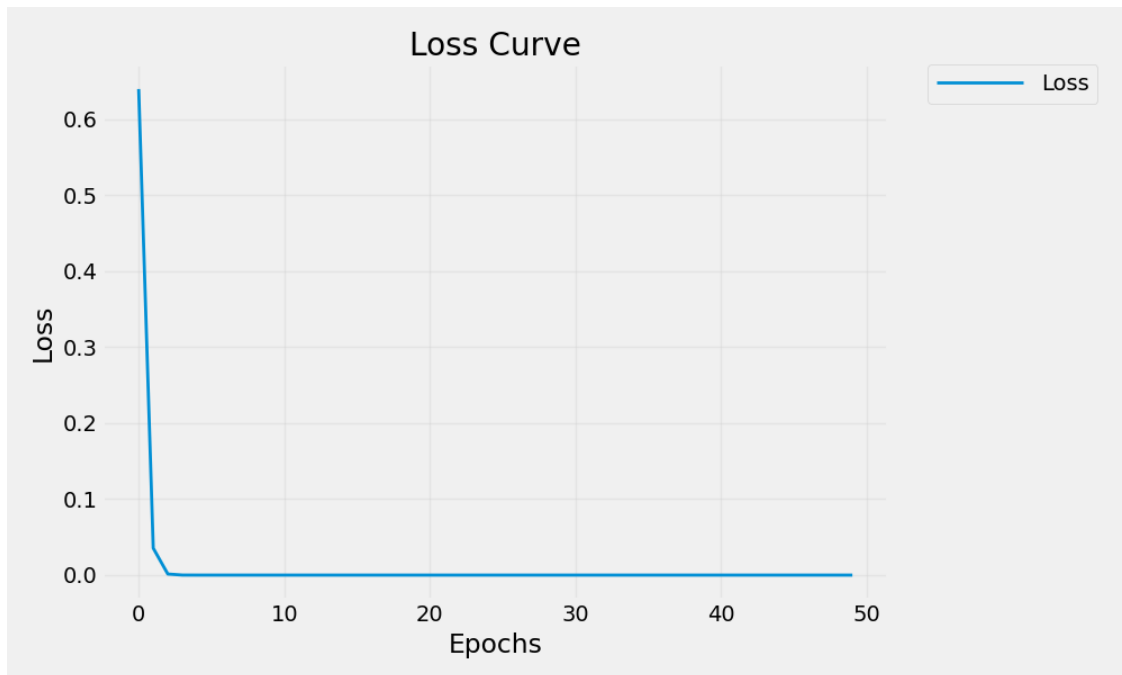




### 3. Learning rate

Tweaking the learning rate can sometimes help. In this case, it helped the network to converge to the optimal solution very quickly. It was updated from 0.01 to 0.1

Epoch 50: Loss = 0.0,  $y_{\text{pred}} = 1.0$



iii) Analyze: Compare the different modifications. Which ones improved convergence? Discuss how the initial zero values for weights led to degenerate learning, and explain how your modifications addressed these issues.

Adding bias, incrementing learning rate, and randomly marking random weights as trainable improved convergence. I tried using different activation functions, but they did not seem to help. Initial zero values led to degenerate learning because we only used zero value weights as trainable values. This, coupled with the lack of a bias term and ReLU activation, led to degenerate learning. My modifications addressed these by marking more of the weights as trainable (especially the nonzero ones) and adding a bias. These two changes help us produce nonzero output from each layer.