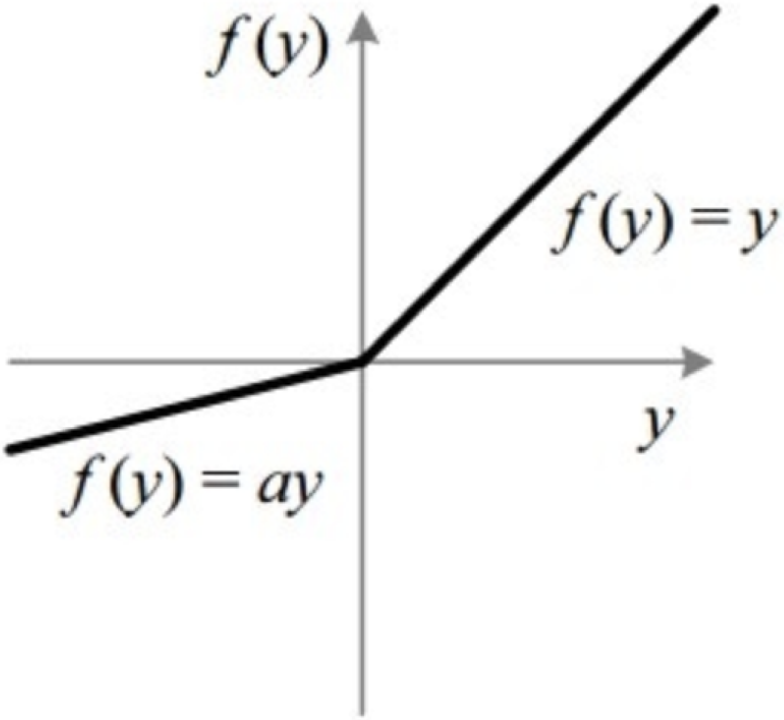
7.5a.



https://paperswithcode.com/method/leaky-relu#:~:text=Leaky%20Rectified%20Linear%20Unit%2C%20or,is%20not%20learnt%20during%20training.

Leaky ReLu pros are that it allows a small non-zero gradient when it's not active. This prevents neurons from becoming inactive during training. Cons are that it doesn’t always outperform ReLU, its performance depends on the dataset and architecture.

7.5b.

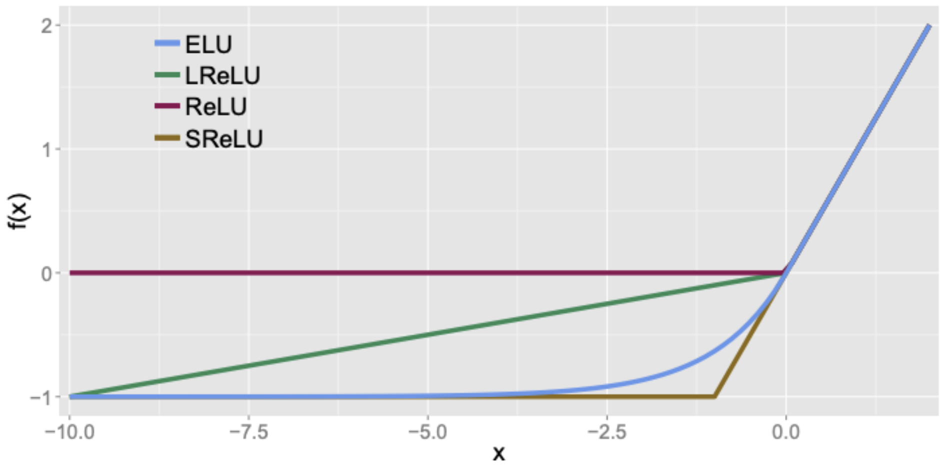
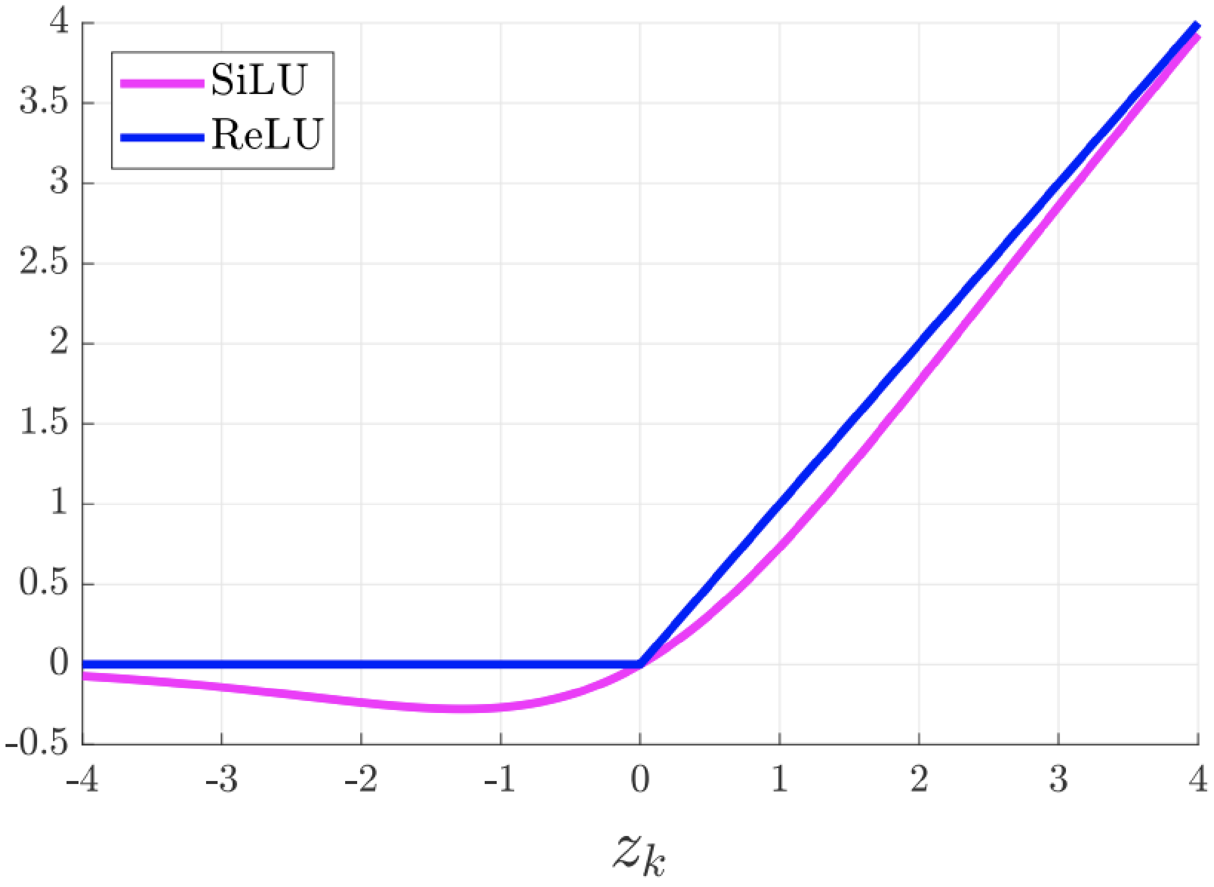


Figure 1: The rectified linear unit (ReLU), the leaky ReLU (LReLU, $\alpha = 0.1$), the shifted ReLUs (SReLUs), and the exponential linear unit (ELU, $\alpha = 1.0$).

https://paperswithcode.com/method/elu

Exponential linear unit also allow for non-zero gradients and provide more smooth outputs. The con would be its computational expense.

7.5c.



https://paperswithcode.com/method/silu

Sigmoid linear unit, it provides non-zero gradients almost everywhere, making training smoother and it is shown to outperform ReLU. Con is that it's also a computational expense similar to ELU.

7.8a. 34 weights (4*4 + 4*3 + 3*2) = 34

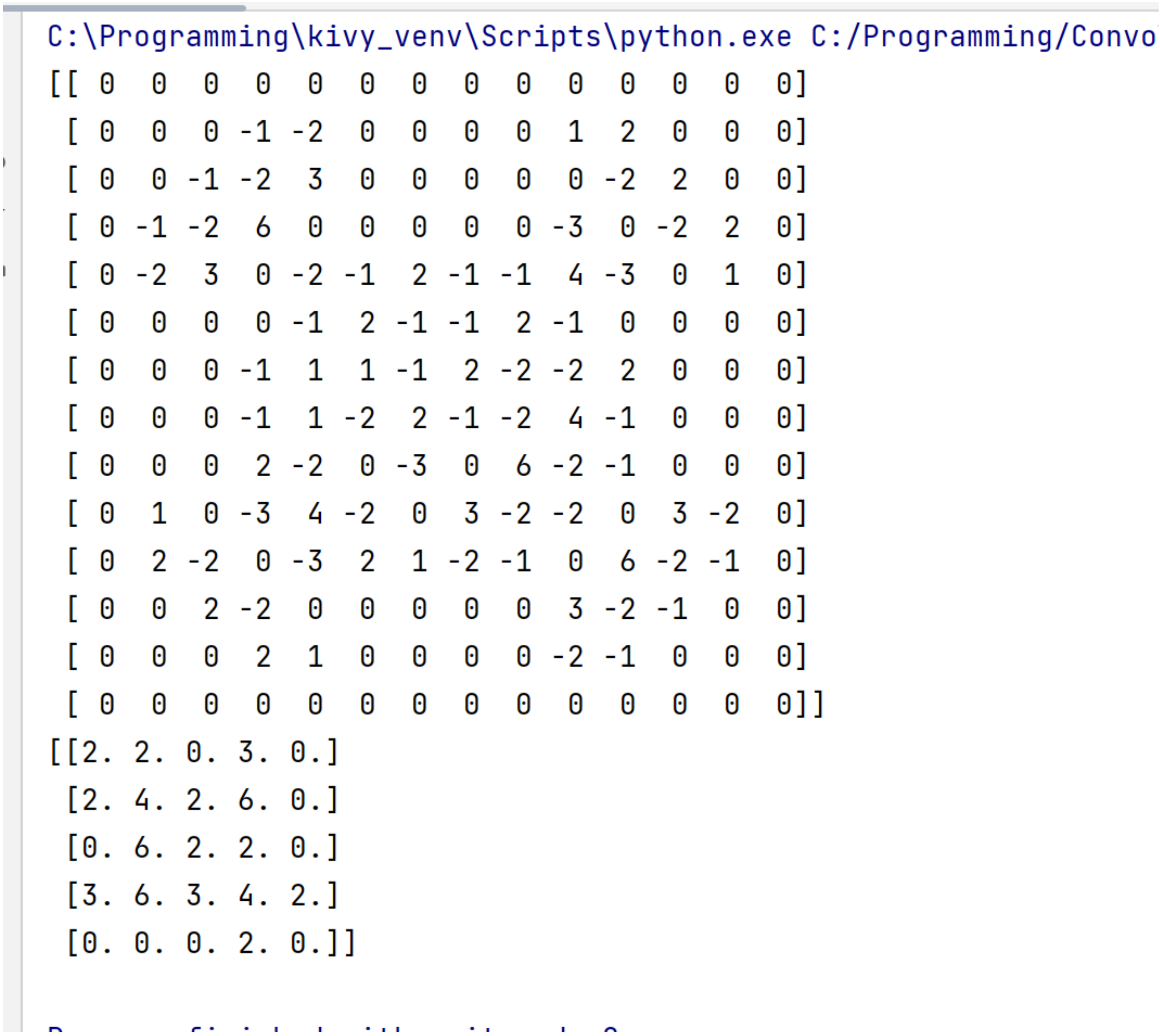
7.8b. $\sum^{L}_{i=1}=(n_{i-1}*n_{i})$ where i is the number of inputs and L is the number of layers and n is the number of neurons

7.9.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The softmax function is a generalization of the sigmoid function and it is used to convert a vector of K real numbers into a probability distribution of K probabilities.For multiclass classification it provides a normalized probability distribution and ensures the output is a probability distribution. It chooses the highest probability as the predicted class.

7.10



```
C:\Programming\kivy_venv\Scripts\python.exe C:/Programming/Convo
[[ 0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0 -1 -2  0  0  0  0  1  2  0  0  0]
 [ 0  0 -1 -2  3  0  0  0  0  0 -2  2  0  0]
 [ 0 -1 -2  6  0  0  0  0  0 -3  0 -2  2  0]
 [ 0 -2  3  0 -2 -1  2 -1 -1  4 -3  0  1  0]
 [ 0  0  0  0 -1  2 -1 -1  2 -1  0  0  0  0]
 [ 0  0  0 -1  1  1 -1  2 -2 -2  2  0  0  0]
 [ 0  0  0 -1  1 -2  2 -1 -2  4 -1  0  0  0]
 [ 0  0  0  2 -2  0 -3  0  6 -2 -1  0  0  0]
 [ 0  1  0 -3  4 -2  0  3 -2 -2  0  3 -2  0]
 [ 0  2 -2  0 -3  2  1 -2 -1  0  6 -2 -1  0]
 [ 0  0  2 -2  0  0  0  0  0  3 -2 -1  0  0]
 [ 0  0  0  2  1  0  0  0  0 -2 -1  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0]]
[[2. 2. 0. 3. 0.]
 [2. 4. 2. 6. 0.]
 [0. 6. 2. 2. 0.]
 [3. 6. 3. 4. 2.]
 [0. 0. 0. 2. 0.]]
```

Code in Folder

7.11a. 2*5*3*3 = 90 weights

7.11b. C*F*n*m