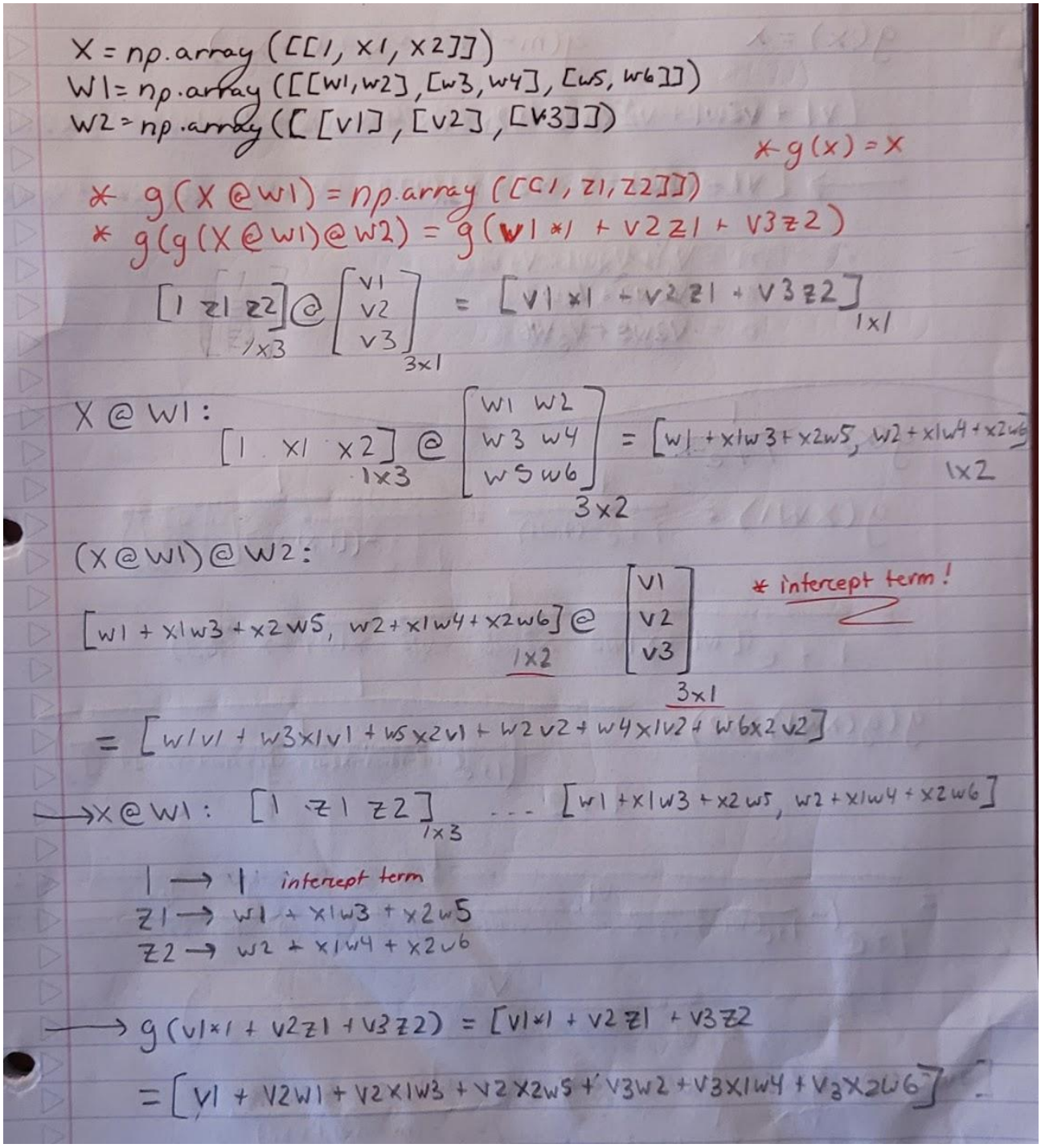
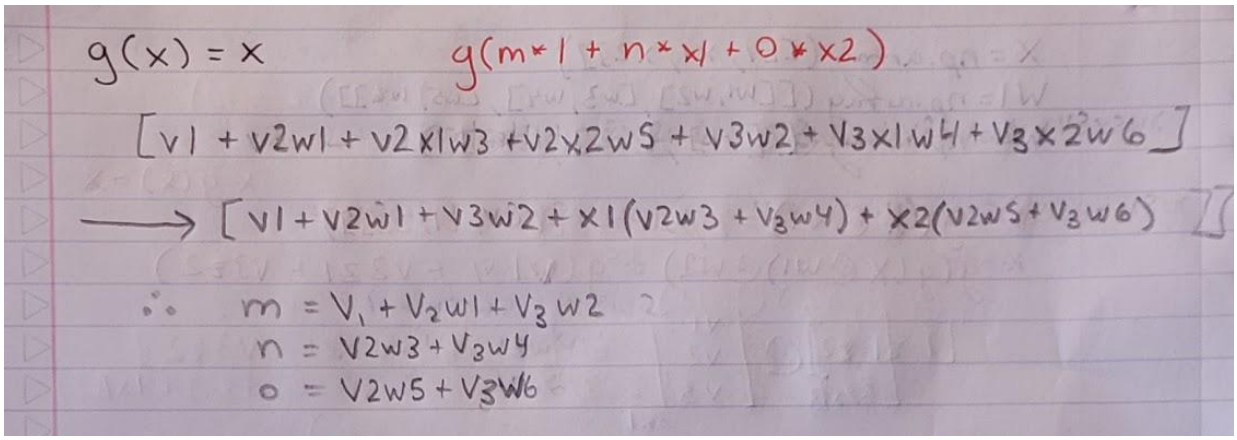
a.


X = np.array([[1, x1, x2]])
W1 = np.array([[w1, w2], [w3, w4], [w5, w6]])
W2 = np.array([[v1], [v2], [v3]])
* g(x) = x
* g(X @ W1) = np.array([[1, z1, z2]])
* g(g(X @ W1) @ W2) = g(v1*1 + v2 z1 + v3 z2)
[1 z1 z2] @ [v1 v2 v3] = [v1*1 + v2 z1 + v3 z2]
1x3 3x1 1x1
X @ W1:
[1 x1 x2] @ [w1 w2 w3 w4 w5 w6] = [w1 + x1w3 + x2w5, w2 + x1w4 + x2w6]
1x3 3x2 1x2
(X @ W1) @ W2:
[w1 + x1w3 + x2w5, w2 + x1w4 + x2w6] @ [v1 v2 v3]
1x2 3x1
* intercept term!
= [w1v1 + w3x1v1 + w5x2v1 + w2v2 + w4x1v2 + w6x2v2]
→ X @ W1: [1 z1 z2] ... [w1 + x1w3 + x2w5, w2 + x1w4 + x2w6]
1x3
1 → 1 intercept term
z1 → w1 + x1w3 + x2w5
z2 → w2 + x1w4 + x2w6
→ g(v1*1 + v2 z1 + v3 z2) = [v1*1 + v2 z1 + v3 z2
= [v1 + v2w1 + v2x1w3 + v2x2w5 + v3w2 + v3x1w4 + v3x2w6]

b.




c.

This shows that an MLP using linear activation functions is equivalent to a linear perceptron because the entire network can be expressed as a linear function. Though, it should be stated that using linear activation functions limits the expressive power of a neural network as it can only learn linear decision boundaries, while an MLP with one or more hidden layers and linear activation functions can approximate any continuous function (of course, given enough hidden units). In this case, using the linear activation function g(x) = x allows one to express the entire network as a linear function, which is equivalent to a linear perceptron. However, this approach, as briefly mentioned, limits the network's ability to model complex relationships between features and outputs. Any other complex function should be expressed by an MLP.

d.

$$g(x) = \frac{1}{1+e^{-x}} \; ; \; g(m\times 1 + n\times 1 + o\times 2) = \frac{1}{1+e^{-m\cdot 1 - n\times 1 - o\times 2}}$$

$$X @ W1 = \begin{bmatrix} w1 + x1w3 + x2w5 & w2 + x1w4 + x2w6 \end{bmatrix}_{1\times 2}$$

$$g(X @ W1) = \frac{1}{1+e^{(-[w1 - x1w3 - x2w5 \quad -w2 - x1w4 - x2w6])}} = \frac{1}{1+e^{-(X @ W)}}$$

$$g(g(X @ W1) @ W2) = \frac{1}{1+e^{-\left(\frac{1}{1+e^{-(X @ W1)}} \cdot W2\right)}}$$

$$= \frac{1}{1+e^{-\left(\frac{1}{1+e^{([-w1 - x1w3 - x2w5 \quad -w2 - x1w4 - x2w6])}} \cdot \begin{bmatrix} V1 \\ V2 \\ V3 \end{bmatrix}\right)}}$$

$$= \frac{1}{1+e^{\left(-\frac{V1}{1+e^{-(X @ W1)}} - \frac{V2}{1+e^{-(X @ W1)}} - \frac{V3}{1+e^{-(X @ W1)}}\right)}}$$

$$\because \; g(g(X @ W1) @ W2) = \frac{1}{1+e^{-m\times 1 - n\times 1 - o\times 2}}$$

$$-\frac{V1}{1+e^{-(X @ W1)}} - \frac{V2}{1+e^{-(X @ W1)}} - \frac{V3}{1+e^{-(X @ W1)}} = -m\times 1 - n\times 1 - o\times 2$$

$$\frac{-(V1 + V2 + V3)}{1+e^{-(X @ W1)}} = -m\times 1 - n\times 1 - o\times 2$$

$$* \; e^{A} = 1 + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \ldots + \sum_{n}^{\infty} \frac{1}{n!}A^n$$

* computationally expensive!

Yes, we'd need to use numerical methods to solve for m, n, and o.

e.

This now shows that an MLP using sigmoid activation functions is more expressive than a single sigmoid perceptron because the non-linear activation function allows for more complex mappings between the input features and the output. Using such an activation function allows the MLP to learn nonlinear decision boundaries and thus model more complex relationships within the data, rather than what a single sigmoid perceptron can do.

f.

As discussed prior, non-linearity is crucial for an activation function characteristic. This is an important property as the model must capture complex relationships between the input features and the output within the data. Without the nonlinear property, a neural network is intensely limited to learning only linear relationships, which cannot capture the true nature of relationships in many cases. Another important feature is differentiability. The activation function must be differentiable, as the backpropagation algorithm in neural networks uses the chain-rule, which requires differentiation, to train neural networks. This then allows one to compute gradients of the loss with respect to the network parameters and update them using gradient descent.