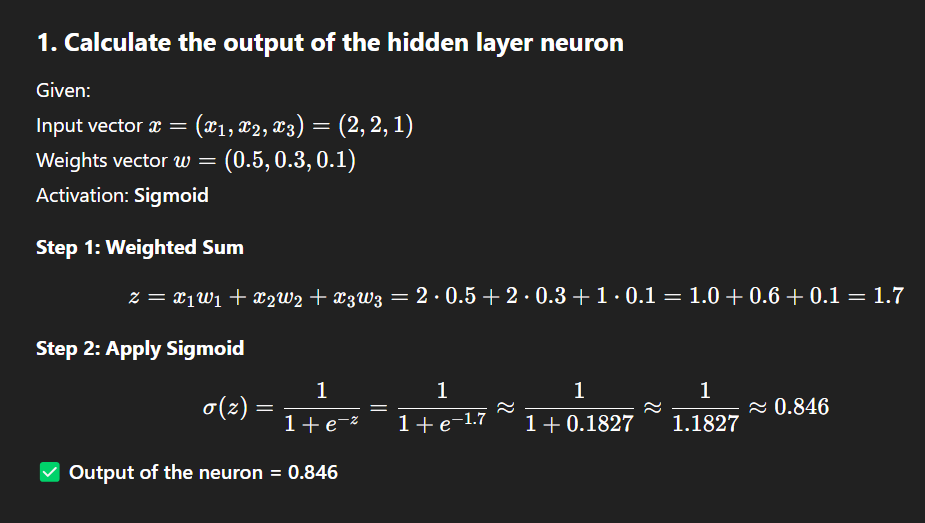
1Consider a 3 -dimensional input x = (x 1 ,x 2 ,x 3 ) = (2,2,1) fully connected with weights

(0.5,0.3,0.1) to one neuron which is in the hidden layer with sigmoid activation function.

Calculate the output of the hidden layer neuron.



2. Illustrate the limitation of a single layer perceptron with an example

A **single-layer perceptron** can only solve **linearly separable problems**.

**Example: XOR Problem**

Inputs:

| **x1** | **x2** | **Output** |
| --- | --- | --- |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

👉 XOR output is **not linearly separable**.  
A single-layer perceptron fails to classify it correctly because there's no straight line that can divide the output classes.

✅ To solve such problems, we need **multi-layer neural networks** (MLPs).

3. With an example of classification problem, explain the following terms:

i) Hyper parameters ii) Training set iii) Validation sets iv) Bias.

Let’s take **Iris Flower Classification** as an example:

* Features: sepal length, petal length, etc.
* Output: class of flower (Setosa, Versicolor, Virginica)

**i) Hyperparameters**

These are parameters that are **set before training**:

* Learning rate
* Number of layers
* Number of epochs
* Batch size
* Regularization terms (like dropout rate)

**ii) Training Set**

The portion of the data used to **train the model**.

* Example: 80% of Iris dataset

**iii) Validation Set**

Used to **tune hyperparameters** and **check for overfitting**.

* Example: 10% of Iris dataset, not seen by model during training.

**iv) Bias**

* Bias allows the activation function to **shift left or right**.
* Prevents the output from being zero when all inputs are zero.
* It’s a learnable parameter just like weights.

5. Compare overfitting and underfitting. How can it affect model generalization?

|  | **Overfitting** | **Underfitting** |
| --- | --- | --- |
| **Definition** | Model learns **noise** & performs badly on test data | Model doesn’t learn **patterns** even from training data |
| **Cause** | Too complex model, too many features, too many epochs | Model too simple, not enough features or epochs |
| **Effect** | High training accuracy, low test accuracy | Low accuracy on both training and test |
| **Fixes** | Regularization, more data, reduce complexity | Add complexity, train longer |

✅ **Model generalization** is best when neither overfitting nor underfitting occurs.

6. You are training a fully connected neural network with 4 hidden layers, each containing 9

hidden units. The input dimension is 15, and the output is a scalar. The network includes

bias terms for each neuron in both the hidden and output layers. Calculate the total number

of trainable parameters in the network.

**(Number of inputs to the layer + 1 bias)** × **Number of neurons in that layer**

**✅ 1. Input Layer → Hidden Layer 1**

* **Input dimension = 15**
* **Hidden layer 1 neurons = 9**
* Each of these 9 neurons is connected to **all 15 input features**.
* Each of these 9 neurons also has **1 bias**.

So, for each neuron:

* Weights = 15
* Bias = 1
* Total per neuron = 16

For 9 neurons:

(15+1)×9=16×9=144 parameters

**✅ 2. Hidden Layer 1 → Hidden Layer 2**

* Hidden layer 1 has 9 neurons
* Hidden layer 2 also has 9 neurons
* Each neuron in hidden layer 2:
  + Takes **9 inputs** (from previous layer)
  + Has **1 bias**

So:

(9+1)×9=10×9=90 parameters

**✅ 3. Hidden Layer 2 → Hidden Layer 3**

Same logic:

* Again 9 inputs and 9 neurons:

(9+1)×9=10×9=90 parameters

**✅ 4. Hidden Layer 3 → Hidden Layer 4**

Still 9 neurons feeding into 9 neurons:

(9+1)×9=10×9=90

**✅ 5. Hidden Layer 4 → Output Layer**

* Hidden Layer 4 has 9 outputs
* Output layer has only **1 neuron**
* That 1 neuron:
  + Takes 9 inputs
  + Has 1 bias

So:

(9+1)×1=10 parameters

**✅ Total Parameters = Sum of all**

144+90+90+90+10=424 trainable parameters

7. Design a neural network with two hidden layers having single neuron (using ReLU activation) and an output neuron to approximate a function f(x)=x2−4x+4 as accurately as possible in the range x [0,5]. ∈[0,5].

This function is a **parabola** and can be written as:

f(x)=(x−2)^2

So, the goal is to **approximate this function** using a **neural network with 2 hidden layers**, each containing **1 neuron**, and using **ReLU activation**.

**Network Structure:**

* **Input**: 1-dimensional (x)
* **Hidden Layer 1**: 1 neuron, **ReLU**
* **Hidden Layer 2**: 1 neuron, **ReLU**
* **Output Layer**: 1 neuron, linear (no activation)

**Step-by-step Network Flow:**

Let x∈[0,5]

Let:

* Layer 1: h1=ReLU(w1x+b1)
* Layer 2: h2=ReLU(w2h1+b2)
* Output: y^=w3h2+b3

Now, you train the weights w1,w2,w3w\_1, w\_2, w\_3w1​,w2​,w3​ and biases b1,b2,b3b\_1, b\_2, b\_3b1​,b2​,b3​ using gradient descent to minimize:

L=1/n∑i=1n(yi^−f(xi))^2

✅ This structure can **approximate quadratic functions** fairly well using piecewise linear segments from the ReLU.

You can try it using frameworks like **TensorFlow/Keras** or **PyTorch**.

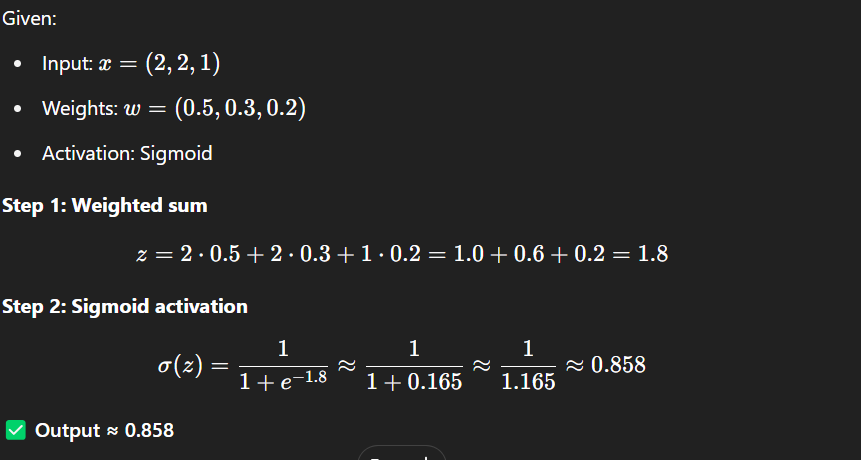
8. Specify the advantages of ReLU over sigmoid activation function.

| **feature** | **ReLU (Rectified Linear Unit)** | **Sigmoid** |
| --- | --- | --- |
| **Formula** | max⁡(0,x) | 1/1+e^−x ​ |
| **Speed** | Faster (no exponentials) | Slower due to exponentials |
| **Vanishing Gradient** | **No** (for x > 0) | **Yes**, gradients vanish as x → ±∞ |
| **Sparse Activation** | Yes (many neurons output 0) | No (always outputs ≠ 0) |
| **Convergence** | Faster training | Slower |
| **Biological realism** | Less biologically inspired | More biologically inspired |

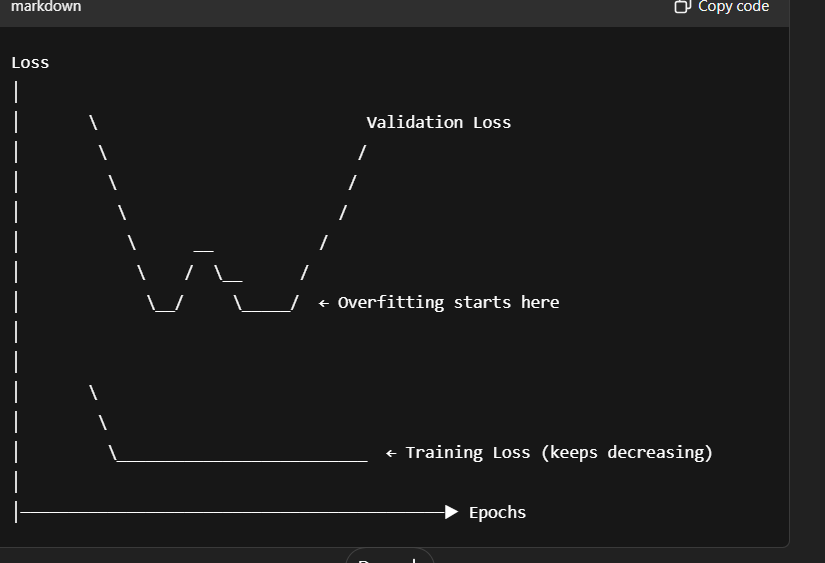
✅ **ReLU** is preferred for hidden layers because it:

* Avoids vanishing gradients
* Is computationally efficient
* Enables deep networks to train better

9. Suppose you have a 3-dimensional input x = (x1, x2, x3) = (2, 2, 1) fully connected with weights (0.5, 0.3, 0.2) to one neuron which is in the hidden layer with sigmoid activation function. Calculate the output of the hidden layer neuron.



10. Sketch the typical learning curves for the training and validation sets, for a setting where overfitting occurs at some point. Assume that the training set and the validation set are of the same size.



**✅ Explanation:**

* **Training Loss**:
  + Keeps **decreasing** steadily as the model fits better to the training data.
* **Validation Loss**:
  + **Decreases initially** (model learning useful features).
  + After a certain point, it **starts increasing** — this means the model is **overfitting** (memorizing training data and losing generalization).

**🧠 Overfitting Signs:**

* Very low training loss.
* Validation loss increases after some training.
* Validation accuracy may drop or stagnate.

12. Can a single-layer perceptron successfully represent the Boolean AND, OR, and XOR functions? Justify your answer, and if there are any limitations, suggest possible ways to overcome them.

| **Function** | **Linearly Separable?** | **Single-layer Perceptron?** |
| --- | --- | --- |
| AND | ✅ Yes | ✅ Yes |
| OR | ✅ Yes | ✅ Yes |
| XOR | ❌ **No** | ❌ **Not possible** |

**❗ Why XOR fails?**

* XOR cannot be separated by a single straight line in 2D input space.
* It needs a **non-linear boundary** → requires **hidden layer**.

**✅ Solution:**

Use a **multi-layer perceptron (MLP)** with **at least one hidden layer** using non-linear activations (ReLU, sigmoid, etc.).

13. What advantages does a deep feedforward network offer over shallow networks?

| **Feature** | **Deep Networks** |
| --- | --- |
| **Representation power** | Can represent complex functions better |
| **Efficiency** | Learn hierarchical features efficiently |
| **Generalization** | Better with enough data & regularization |
| **Parameter reuse** | Some architectures like CNNs benefit |
| **Performance** | Outperform shallow models in practice |

✅ Deep networks **decompose complex functions** into simpler layers — e.g., in images: edges → shapes → objects.

14. What is the purpose of a validation set in neural network training?

Validation set is used to:

* **Tune hyperparameters** (learning rate, hidden layers, etc.)
* **Monitor performance** and detect **overfitting**
* **Perform early stopping** to halt training when validation loss starts increasing

✅ It **simulates unseen data** to check how well the model generalizes.

15. Explain the terms overfitting and underfitting in the context of neural networks.

| **Aspect** | **Overfitting** | **Underfitting** |
| --- | --- | --- |
| **Definition** | Model learns **too much**, including noise | Model learns **too little**, misses patterns |
| **Training Loss** | Very low | High |
| **Validation Loss** | High after some epochs | High |
| **Cause** | Too complex model, too long training | Too simple model or not enough training |
| **Generalization** | Poor | Poor |

**✅ How to fix:**

* **Overfitting**: Use dropout, early stopping, regularization (L2), more data.
* **Underfitting**: Use a more complex model, longer training, better features.

16. Explain back propagation algorithm for neural network training.

**🔁 Backpropagation** is the algorithm used to **train neural networks** by **minimizing the loss** using **gradient descent**. It works in **two main phases**:

**✅ 1. Forward Pass:**

* Input is passed through the network layer by layer.
* Output is computed.
* Loss is calculated using a **loss function** (e.g., MSE, cross-entropy).

**🔙 2. Backward Pass (Backpropagation):**

* Gradients of the loss are **calculated layer by layer** using the **chain rule** of calculus.
* These gradients tell how much each weight contributed to the loss.
* Weights are **updated** using **gradient descent**:

wnew=wold−η⋅∂L/∂w

Where:

* η: learning rate
* ∂L/∂w​: gradient of loss w.r.t weight

**🎯 Goal:**

Minimize the loss by adjusting weights in the direction that reduces error.

17. How does bias and variance trade-off affect machine learning algorithms?

The **bias-variance trade-off** is about balancing **model complexity** and **generalization**.

| **Term** | **Description** |
| --- | --- |
| **Bias** | Error due to **simplifying** assumptions in the model. High bias → underfitting. |
| **Variance** | Error due to **model sensitivity** to small fluctuations in training data. High variance → overfitting. |

**🧠 Trade-off:**

* A **simple model** (high bias, low variance) → fails to capture patterns.
* A **complex model** (low bias, high variance) → captures noise.

**🎯 Goal:**

Find the **sweet spot** where the model performs well on **both training and unseen data** (generalization).

18. Explain the significance of loss function in a deep learning algorithm

A **loss function** measures **how well** a neural network is performing.

**✅ Role of Loss Function:**

* Quantifies the **difference between predicted output and actual output**.
* Guides the **backpropagation** process.
* Determines how **weights are updated** in training.

**📦 Common Loss Functions:**

* **MSE (Mean Squared Error)** – Regression
* **Cross-Entropy Loss** – Classification
* **Hinge Loss** – SVM

**🚀 Why it's important:**

Without a loss function, the model has **no objective** to optimize — it wouldn't know **how to improve**.