1. **Which of the following activation functions is not zero-centered?**
2. Sigmoid
3. Tanh
4. ReLU
5. Softmax

Sigmoid: This function outputs values between 0 and 1, which means it isn't zer o-centered.

Tanh: This function outputs values between -1 and 1, making it zero-centered.

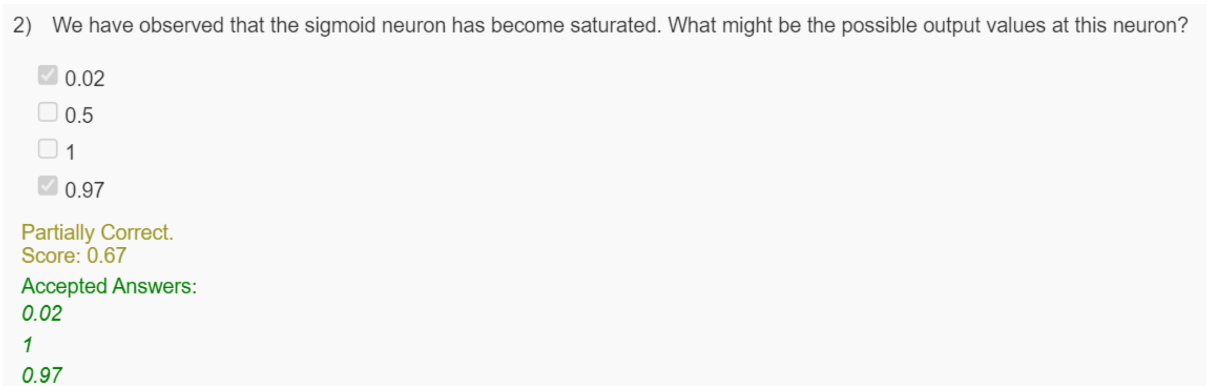
ReLU (Rectified Linear Unit): This outputs values as either 0 or positive, thus it isn't zero-centered.

Softmax: While not inherently zero-centered, it's typically used for output layers in classification tasks to produce pr obability distributions.

So, the ones that aren't zero-centered are **Sigmoid**, **ReLU**, and **Softmax**. Did you expect something different

A zerocentered function is one where the mean of its output is around zero. This can make optimization algorithms like gradient descent more efficient because i t tends to balance the gradients, avoiding large positive or negative biases. It hel ps in faster convergence during the training process.

For example, the **Tanh** activation function outputs values in the range of -1 to 1, which means its mean is approximately zero. On the other hand, function s like **Sigmoid** output values between 0 and 1, which shifts the mean away from zero.


2) We have observed that the sigmoid neuron has become saturated. What might be the possible output values at this neuron?

- [✓] 0.02
- [ ] 0.5
- [ ] 1
- [✓] 0.97

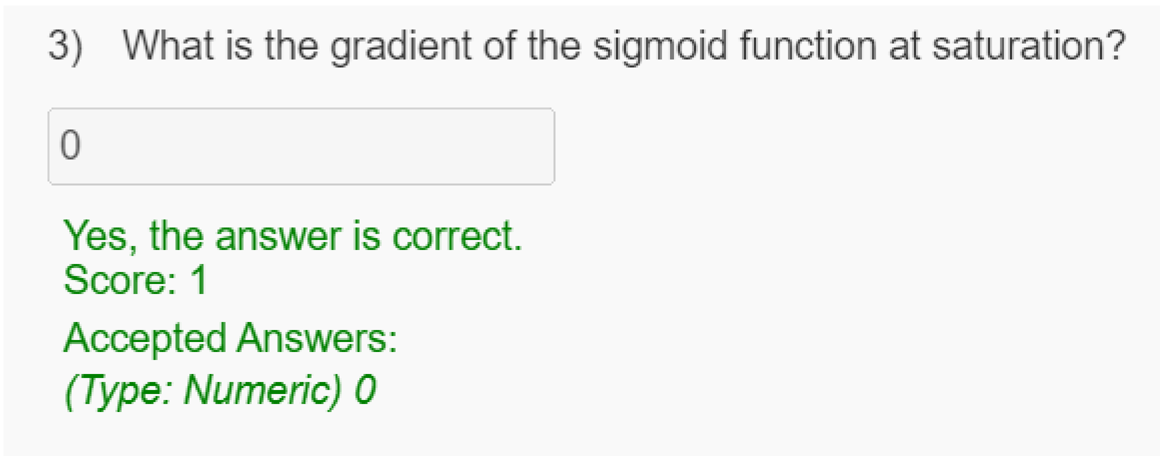
Partially Correct.
Score: 0.67

Accepted Answers:
0.02
1
0.97

when a sigmoid neuron becomes saturated, it means that the output values are p ushed towards the extreme ends of the sigmoid function's range, which is betwe

en 0 and 1. Saturation typically occurs when the neuron receives very large posi
tive or negative input values, causing the output to be close to 0 or 1.

The saturated output values are 0.02,
1 and 0.97. These values indicate that the sigmoid neuron is operating in the flat
 regions of the sigmoid curve, leading to very small gradients, which slows dow
n learning.

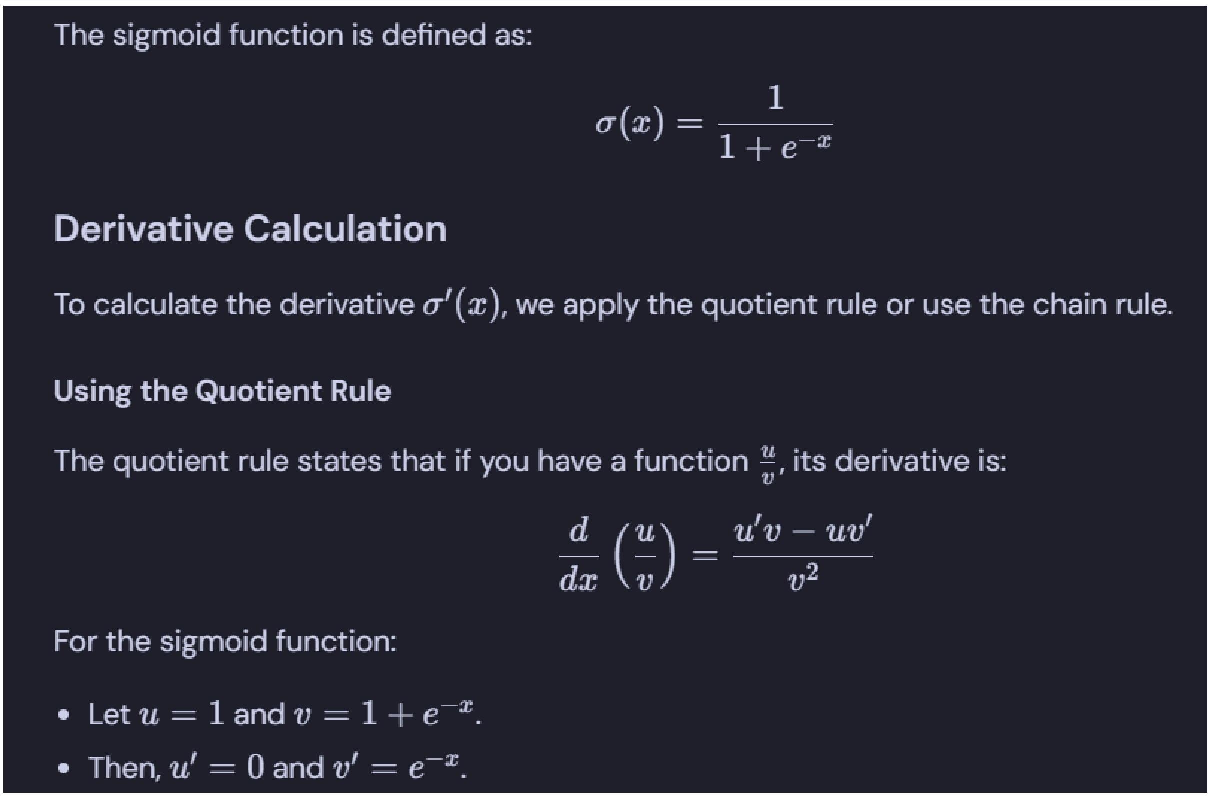


3) What is the gradient of the sigmoid function at saturation?

0

Yes, the answer is correct.
Score: 1

Accepted Answers:
(Type: Numeric) 0



The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Derivative Calculation

To calculate the derivative $\sigma'(x)$, we apply the quotient rule or use the chain rule.

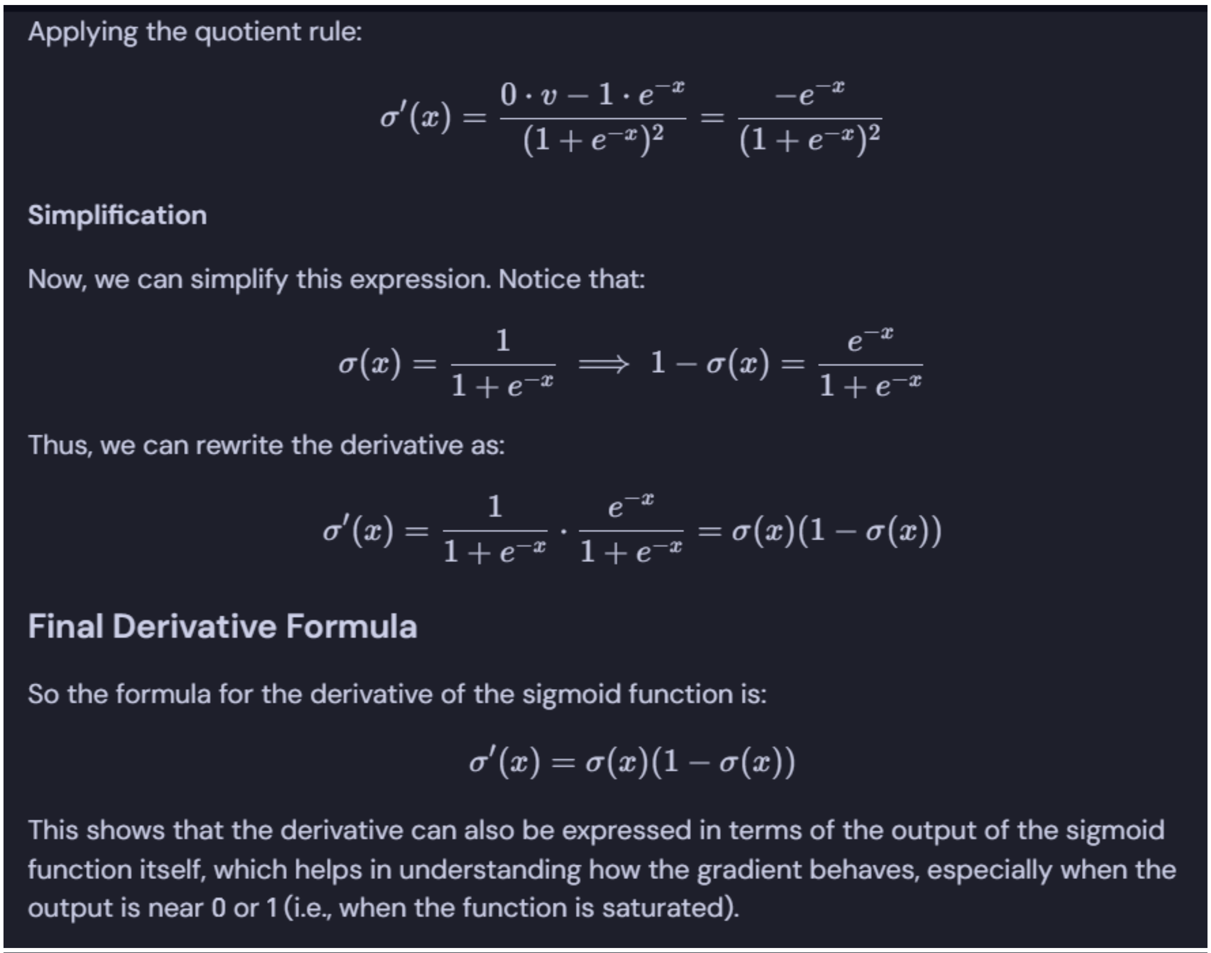
Using the Quotient Rule

The quotient rule states that if you have a function $\frac{u}{v}$, its derivative is:

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{u'v - uv'}{v^2}$$

For the sigmoid function:

- Let $u = 1$ and $v = 1 + e^{-x}$.
- Then, $u' = 0$ and $v' = e^{-x}$.



Applying the quotient rule:

$$\sigma'(x) = \frac{0 \cdot v - 1 \cdot e^{-x}}{(1+e^{-x})^2} = \frac{-e^{-x}}{(1+e^{-x})^2}$$

Simplification

Now, we can simplify this expression. Notice that:

$$\sigma(x) = \frac{1}{1+e^{-x}} \implies 1 - \sigma(x) = \frac{e^{-x}}{1+e^{-x}}$$

Thus, we can rewrite the derivative as:

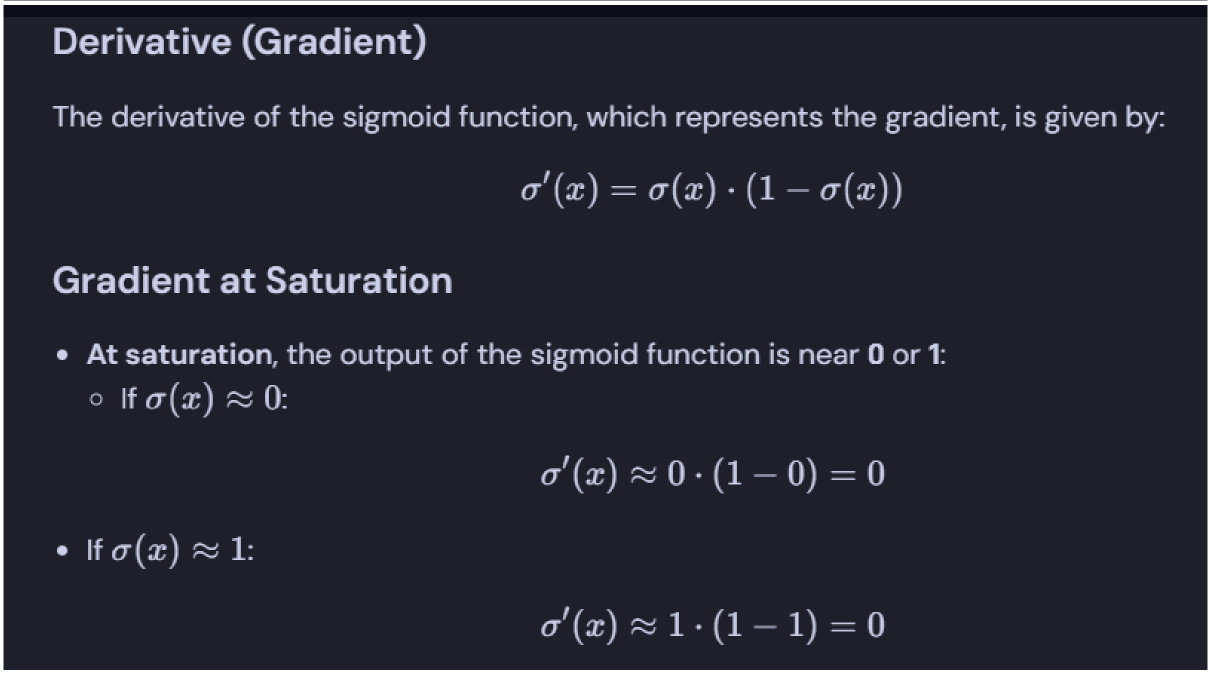
$$\sigma'(x) = \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} = \sigma(x)(1-\sigma(x))$$

Final Derivative Formula

So the formula for the derivative of the sigmoid function is:

$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

This shows that the derivative can also be expressed in terms of the output of the sigmoid function itself, which helps in understanding how the gradient behaves, especially when the output is near 0 or 1 (i.e., when the function is saturated).



Derivative (Gradient)

The derivative of the sigmoid function, which represents the gradient, is given by:

$$\sigma'(x) = \sigma(x) \cdot (1-\sigma(x))$$

Gradient at Saturation

- **At saturation**, the output of the sigmoid function is near **0** or **1**:
  - If $\sigma(x) \approx 0$:

$$\sigma'(x) \approx 0 \cdot (1-0) = 0$$

- If $\sigma(x) \approx 1$:

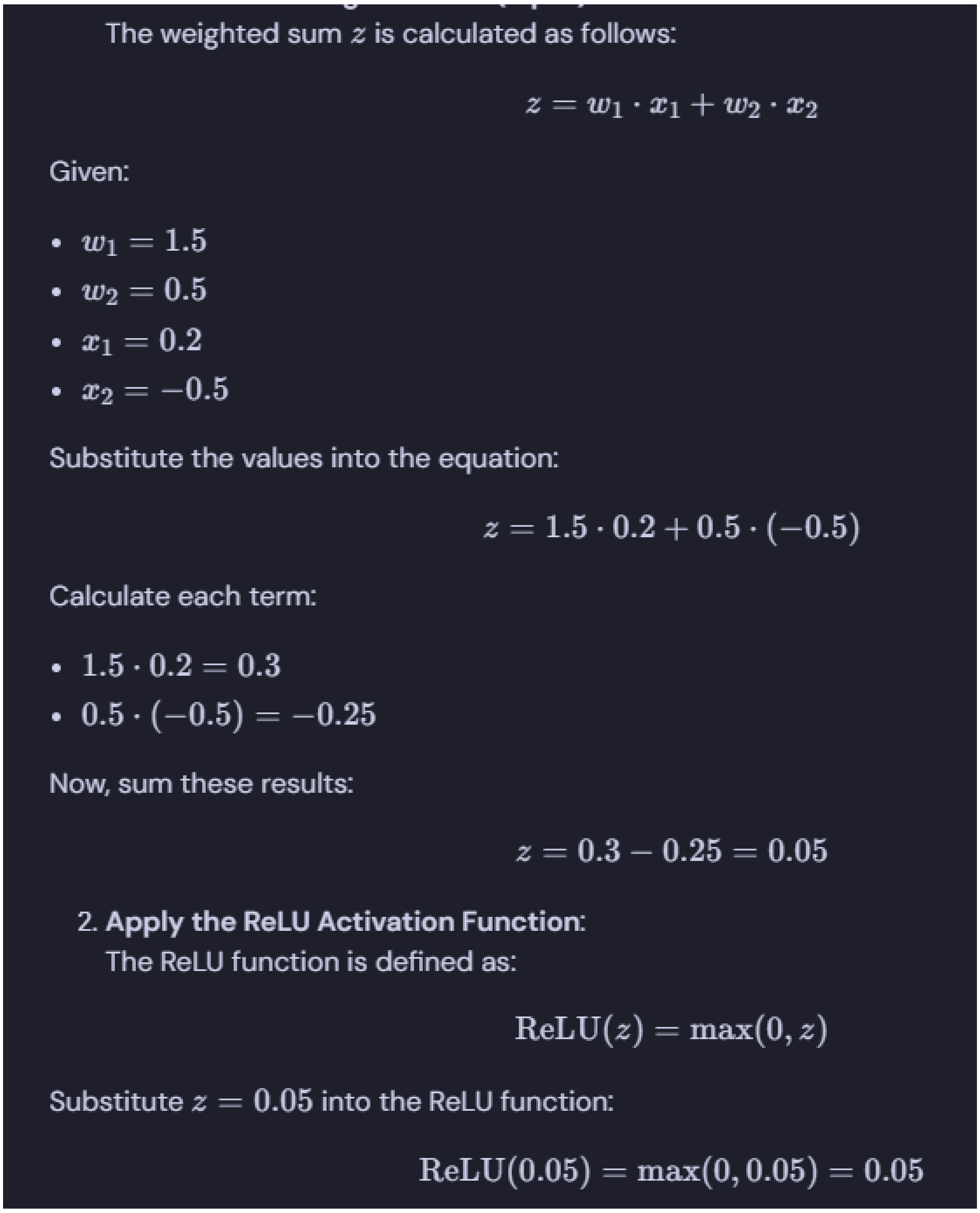
$$\sigma'(x) \approx 1 \cdot (1-1) = 0$$

4) Given a neuron initialized with weights $w_1 = 1.5$, $w_2 = 0.5$, and inputs $x_1 = 0.2$, $x_2 = -0.5$, calculate the output of a ReLU neuron.

0.05

Yes, the answer is correct.
Score: 1

Accepted Answers:
(Type: Numeric) 0.05



The weighted sum z is calculated as follows:

$$z = w_1 \cdot x_1 + w_2 \cdot x_2$$

Given:

- $w_1 = 1.5$
- $w_2 = 0.5$
- $x_1 = 0.2$
- $x_2 = -0.5$

Substitute the values into the equation:

$$z = 1.5 \cdot 0.2 + 0.5 \cdot (-0.5)$$

Calculate each term:

- $1.5 \cdot 0.2 = 0.3$
- $0.5 \cdot (-0.5) = -0.25$

Now, sum these results:

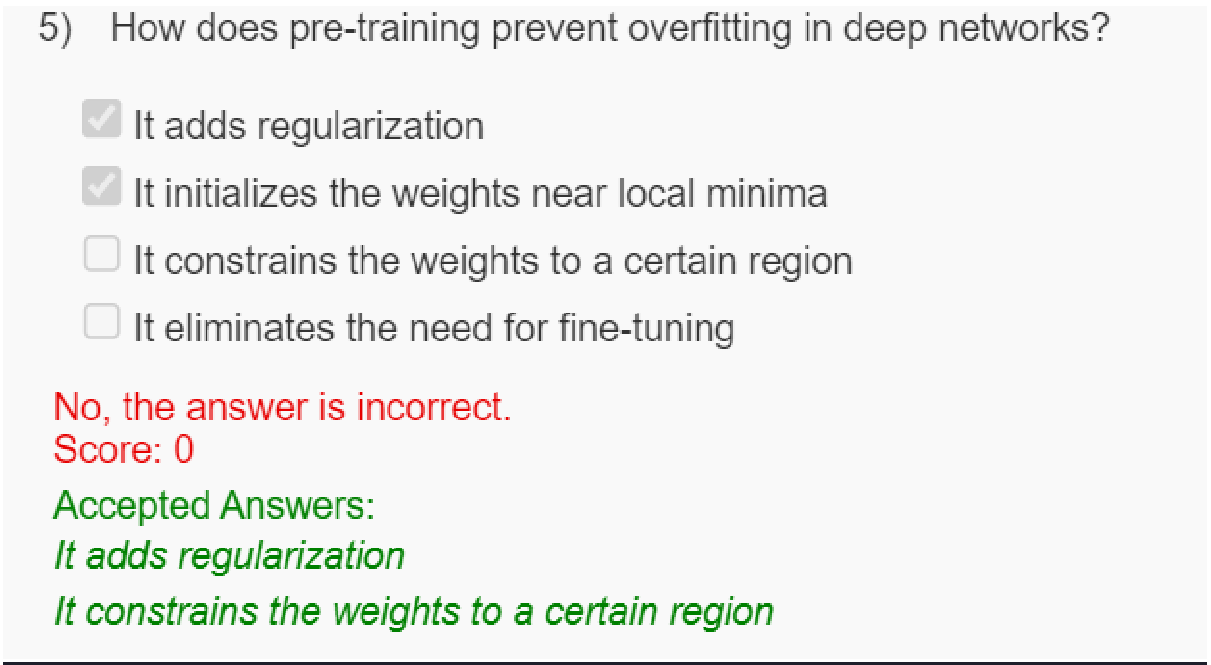
$$z = 0.3 - 0.25 = 0.05$$

2. **Apply the ReLU Activation Function**:
   The ReLU function is defined as:

$$\text{ReLU}(z) = \max(0, z)$$

Substitute $z = 0.05$ into the ReLU function:

$$\text{ReLU}(0.05) = \max(0, 0.05) = 0.05$$



5) How does pre-training prevent overfitting in deep networks?

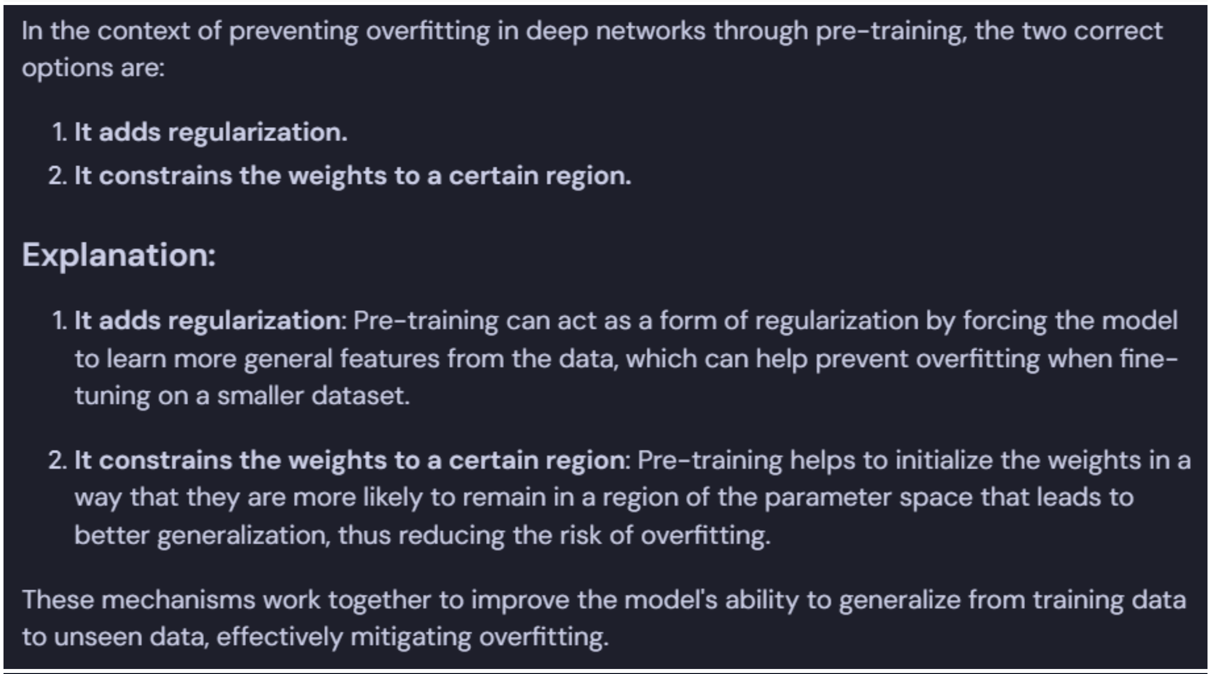
- [✓] It adds regularization
- [✓] It initializes the weights near local minima
- [ ] It constrains the weights to a certain region
- [ ] It eliminates the need for fine-tuning

No, the answer is incorrect.
Score: 0

Accepted Answers:

It adds regularization

It constrains the weights to a certain region



In the context of preventing overfitting in deep networks through pre-training, the two correct options are:

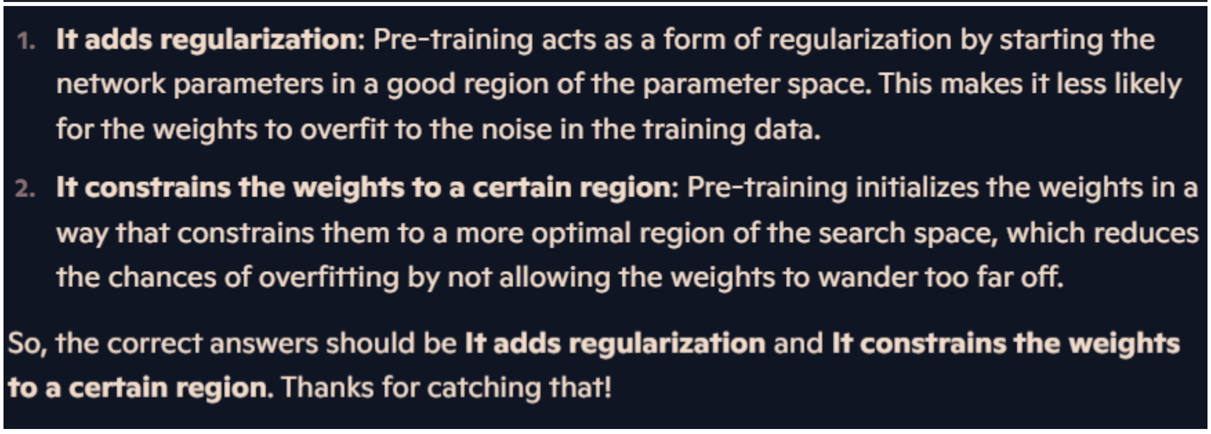
1. **It adds regularization.**
2. **It constrains the weights to a certain region.**

Explanation:

1. **It adds regularization**: Pre-training can act as a form of regularization by forcing the model to learn more general features from the data, which can help prevent overfitting when fine-tuning on a smaller dataset.

2. **It constrains the weights to a certain region**: Pre-training helps to initialize the weights in a way that they are more likely to remain in a region of the parameter space that leads to better generalization, thus reducing the risk of overfitting.

These mechanisms work together to improve the model's ability to generalize from training data to unseen data, effectively mitigating overfitting.

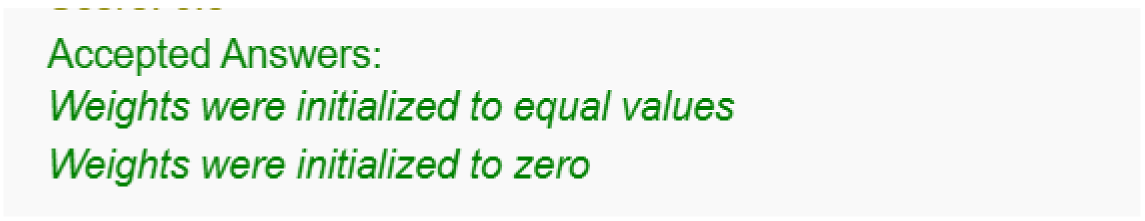


1. **It adds regularization:** Pre-training acts as a form of regularization by starting the network parameters in a good region of the parameter space. This makes it less likely for the weights to overfit to the noise in the training data.
2. **It constrains the weights to a certain region:** Pre-training initializes the weights in a way that constrains them to a more optimal region of the search space, which reduces the chances of overfitting by not allowing the weights to wander too far off.

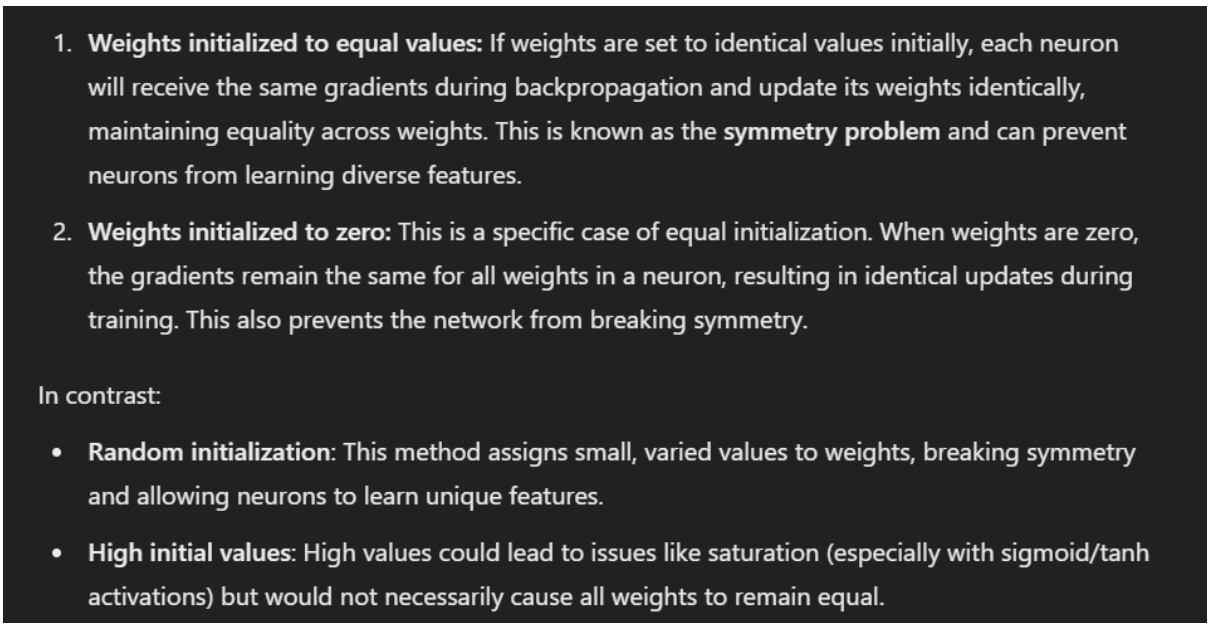
So, the correct answers should be **It adds regularization** and **It constrains the weights to a certain region**. Thanks for catching that!

We train a feed-forward neural network and notice that all the weights for a particular neuron are equal. What could be the possible causes of this issue?

1. Weights were initialized randomly
2. Weights were initialized to high values
3. Weights were initialized to equal values
4. Weights were initialized to zero



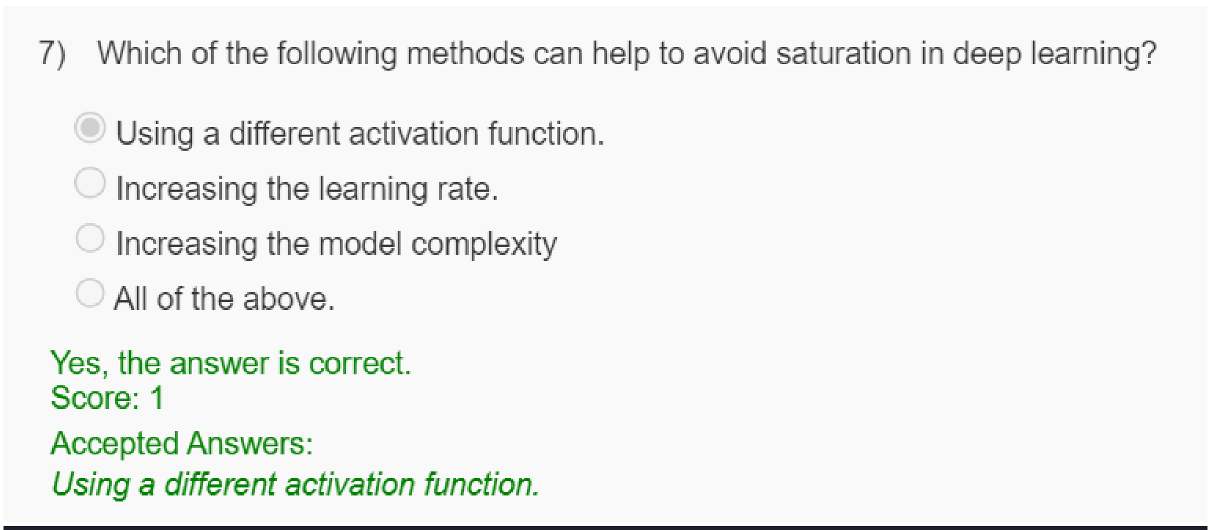
Accepted Answers:
Weights were initialized to equal values
Weights were initialized to zero



1. **Weights initialized to equal values:** If weights are set to identical values initially, each neuron will receive the same gradients during backpropagation and update its weights identically, maintaining equality across weights. This is known as the **symmetry problem** and can prevent neurons from learning diverse features.
2. **Weights initialized to zero:** This is a specific case of equal initialization. When weights are zero, the gradients remain the same for all weights in a neuron, resulting in identical updates during training. This also prevents the network from breaking symmetry.

In contrast:

- **Random initialization**: This method assigns small, varied values to weights, breaking symmetry and allowing neurons to learn unique features.
- **High initial values**: High values could lead to issues like saturation (especially with sigmoid/tanh activations) but would not necessarily cause all weights to remain equal.

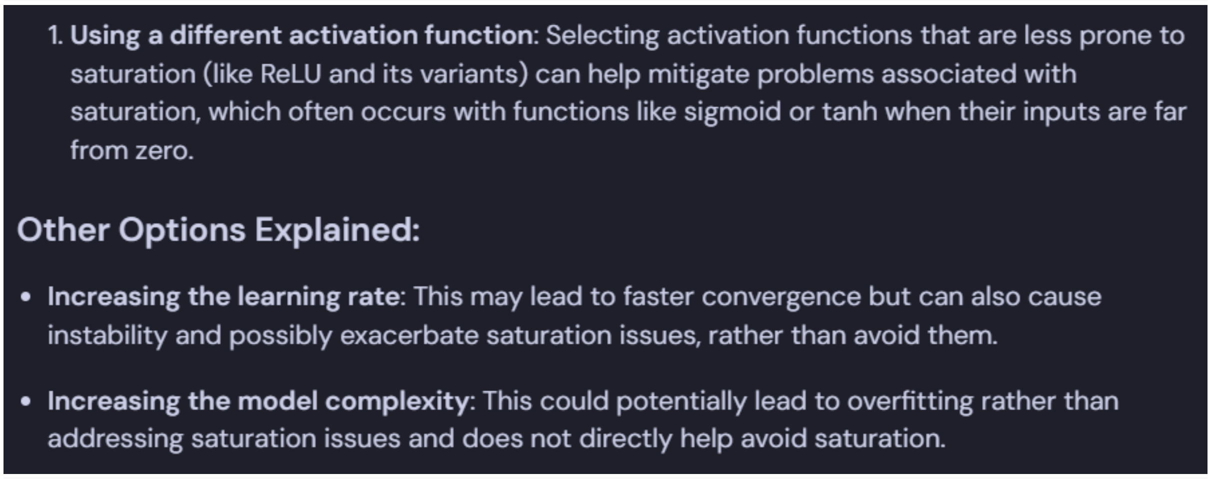


7) Which of the following methods can help to avoid saturation in deep learning?

- [✓ Using a different activation function.]
- Increasing the learning rate.
- Increasing the model complexity
- All of the above.

Yes, the answer is correct.
Score: 1

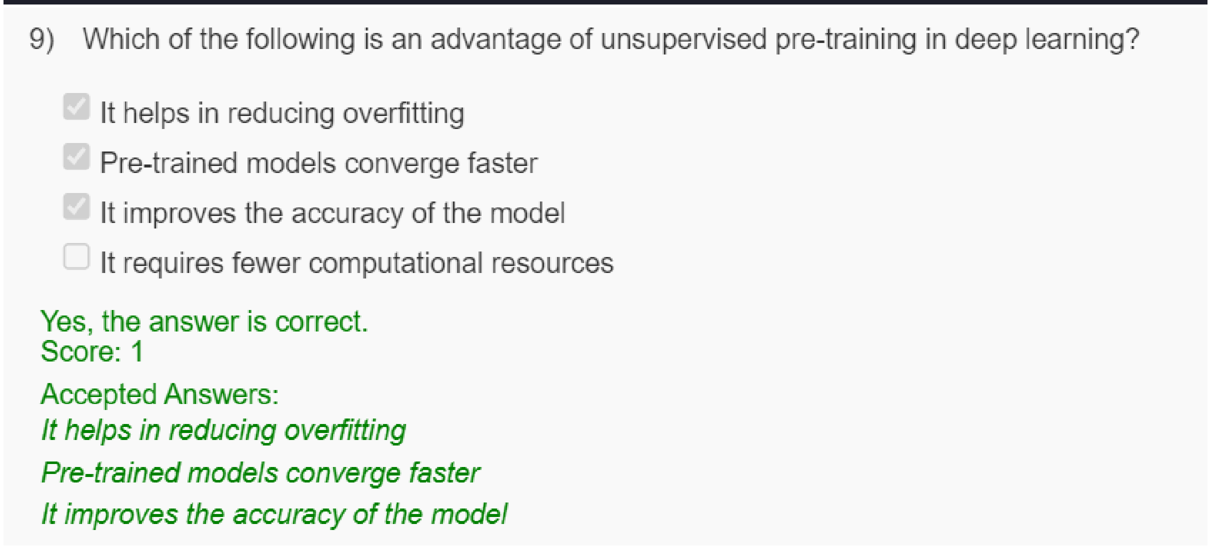
Accepted Answers:
Using a different activation function.



1. **Using a different activation function**: Selecting activation functions that are less prone to saturation (like ReLU and its variants) can help mitigate problems associated with saturation, which often occurs with functions like sigmoid or tanh when their inputs are far from zero.

Other Options Explained:

- **Increasing the learning rate**: This may lead to faster convergence but can also cause instability and possibly exacerbate saturation issues, rather than avoid them.
- **Increasing the model complexity**: This could potentially lead to overfitting rather than addressing saturation issues and does not directly help avoid saturation.

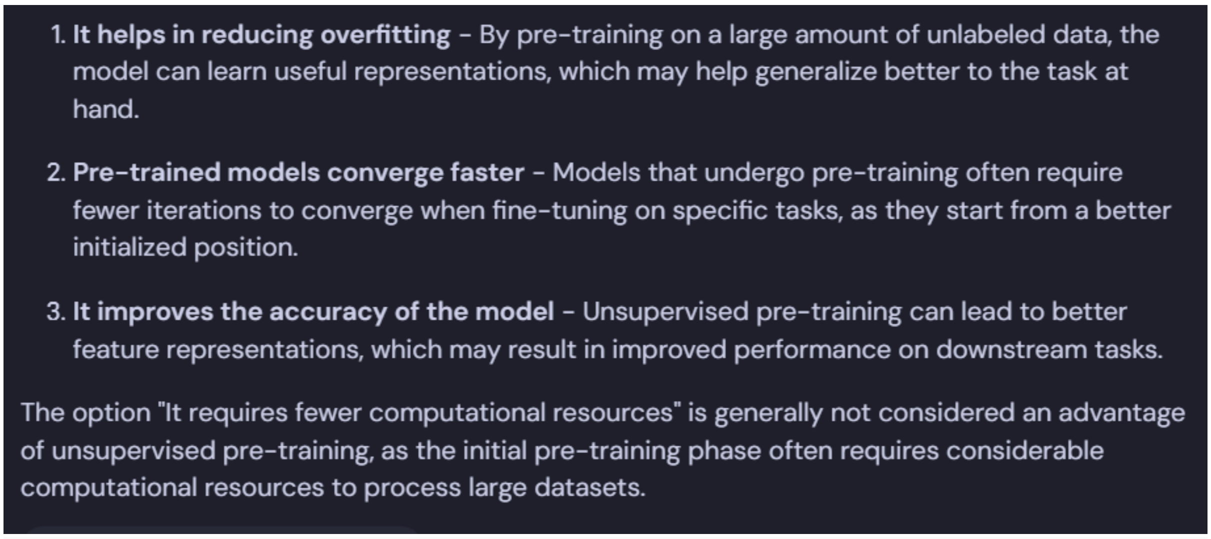


9) Which of the following is an advantage of unsupervised pre-training in deep learning?

- [✓ It helps in reducing overfitting]
- [✓ Pre-trained models converge faster]
- [✓ It improves the accuracy of the model]
- It requires fewer computational resources

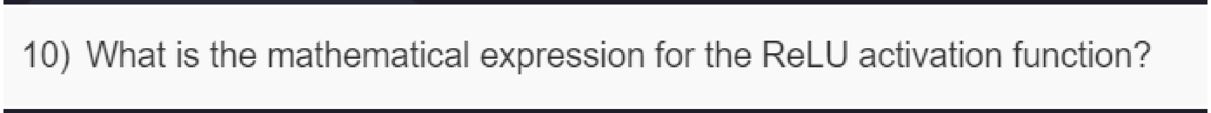
Yes, the answer is correct.
Score: 1

Accepted Answers:
It helps in reducing overfitting
Pre-trained models converge faster
It improves the accuracy of the model

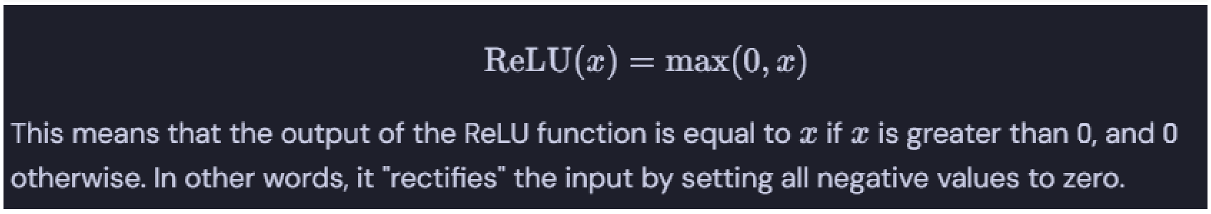


1. **It helps in reducing overfitting** - By pre-training on a large amount of unlabeled data, the model can learn useful representations, which may help generalize better to the task at hand.
2. **Pre-trained models converge faster** - Models that undergo pre-training often require fewer iterations to converge when fine-tuning on specific tasks, as they start from a better initialized position.
3. **It improves the accuracy of the model** - Unsupervised pre-training can lead to better feature representations, which may result in improved performance on downstream tasks.

The option "It requires fewer computational resources" is generally not considered an advantage of unsupervised pre-training, as the initial pre-training phase often requires considerable computational resources to process large datasets.



10) What is the mathematical expression for the ReLU activation function?

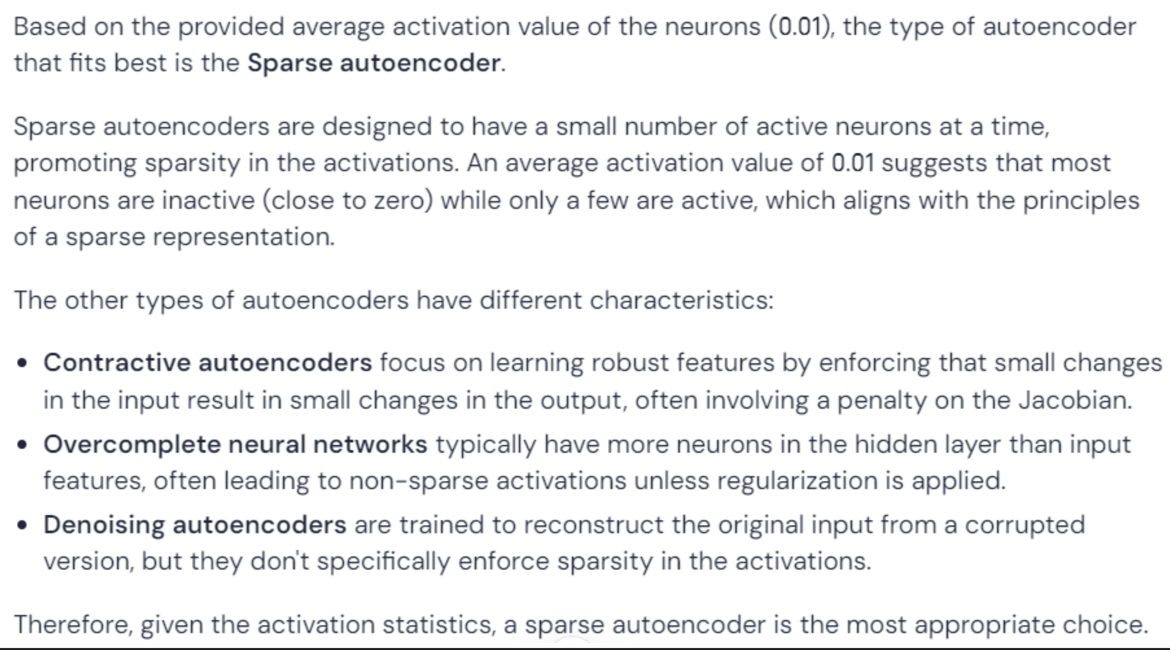


$$\mathrm{ReLU}(x) = \max(0, x)$$

This means that the output of the ReLU function is equal to x if x is greater than 0, and 0 otherwise. In other words, it "rectifies" the input by setting all negative values to zero.

Q. We are given an autoencoder A. The average activation value of neurons in this network is 0.01. The given autoencoder is:

- Contractive autoencoder
- Overcomplete neural network
- Denoising autoencoder
- Sparse autoencoder



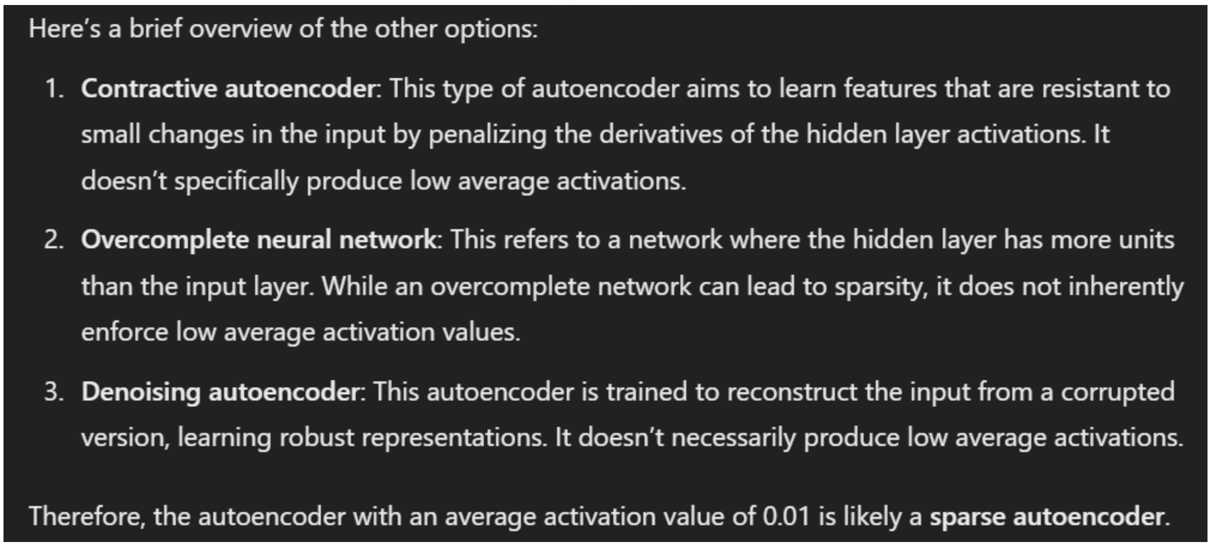
Based on the provided average activation value of the neurons (0.01), the type of autoencoder that fits best is the **Sparse autoencoder**.

Sparse autoencoders are designed to have a small number of active neurons at a time, promoting sparsity in the activations. An average activation value of 0.01 suggests that most neurons are inactive (close to zero) while only a few are active, which aligns with the principles of a sparse representation.

The other types of autoencoders have different characteristics:

- **Contractive autoencoders** focus on learning robust features by enforcing that small changes in the input result in small changes in the output, often involving a penalty on the Jacobian.
- **Overcomplete neural networks** typically have more neurons in the hidden layer than input features, often leading to non-sparse activations unless regularization is applied.
- **Denoising autoencoders** are trained to reconstruct the original input from a corrupted version, but they don't specifically enforce sparsity in the activations.

Therefore, given the activation statistics, a sparse autoencoder is the most appropriate choice.



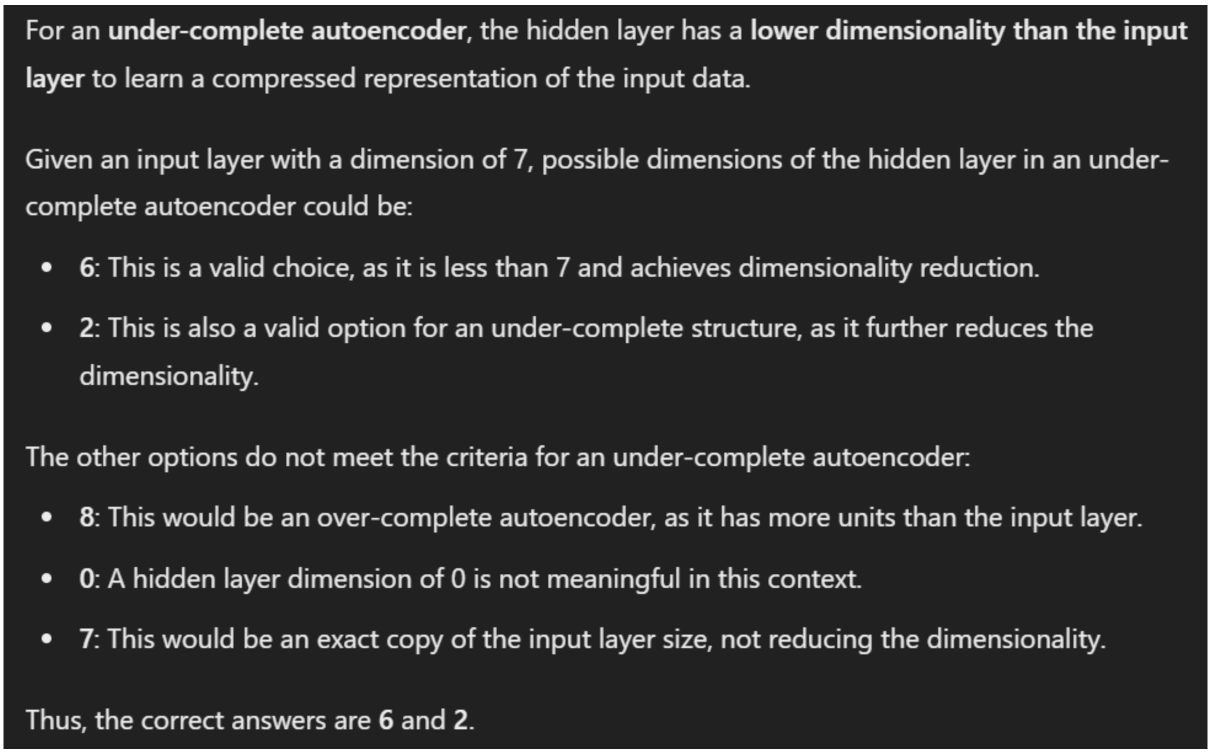
Here's a brief overview of the other options:

1. **Contractive autoencoder**: This type of autoencoder aims to learn features that are resistant to small changes in the input by penalizing the derivatives of the hidden layer activations. It doesn't specifically produce low average activations.
2. **Overcomplete neural network**: This refers to a network where the hidden layer has more units than the input layer. While an overcomplete network can lead to sparsity, it does not inherently enforce low average activation values.
3. **Denoising autoencoder**: This autoencoder is trained to reconstruct the input from a corrupted version, learning robust representations. It doesn't necessarily produce low average activations.

Therefore, the autoencoder with an average activation value of 0.01 is likely a **sparse autoencoder**.

If an under-complete autoencoder has an input layer with a dimension of 7, what could be the possible dimension of the hidden layer?

- 6
- 8
- 0
- 7
- 2


For an **under-complete autoencoder**, the hidden layer has a **lower dimensionality than the input layer** to learn a compressed representation of the input data.

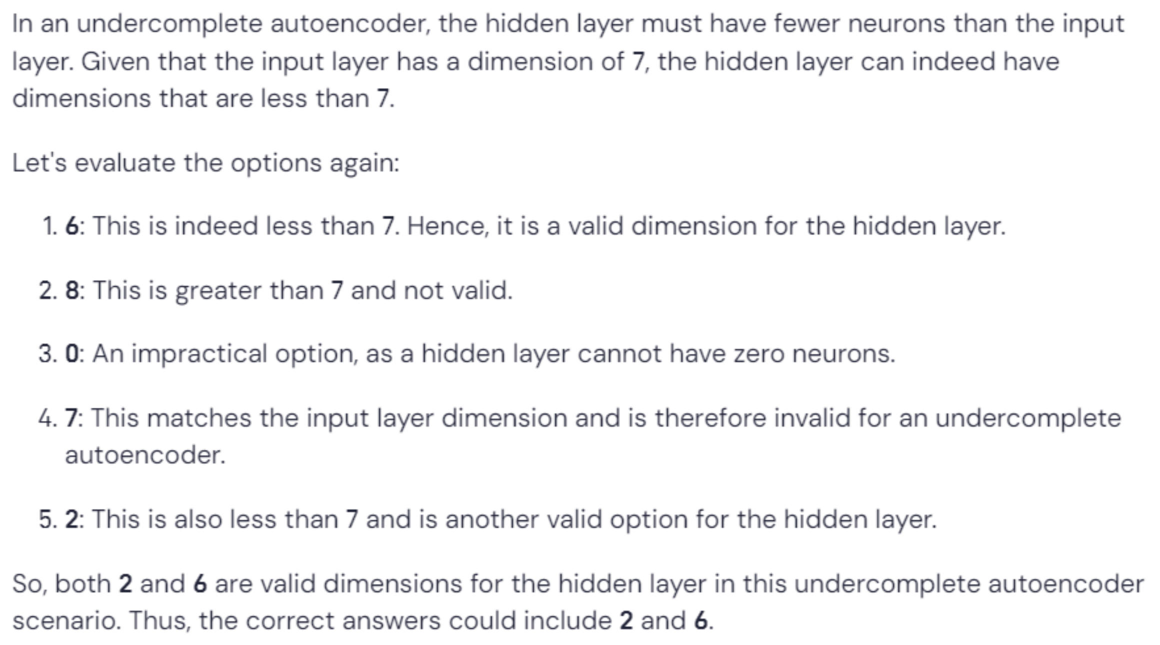
Given an input layer with a dimension of 7, possible dimensions of the hidden layer in an under-complete autoencoder could be:

- **6**: This is a valid choice, as it is less than 7 and achieves dimensionality reduction.
- **2**: This is also a valid option for an under-complete structure, as it further reduces the dimensionality.

The other options do not meet the criteria for an under-complete autoencoder:

- **8**: This would be an over-complete autoencoder, as it has more units than the input layer.
- **0**: A hidden layer dimension of 0 is not meaningful in this context.
- **7**: This would be an exact copy of the input layer size, not reducing the dimensionality.

Thus, the correct answers are **6** and **2**.


In an undercomplete autoencoder, the hidden layer must have fewer neurons than the input layer. Given that the input layer has a dimension of 7, the hidden layer can indeed have dimensions that are less than 7.

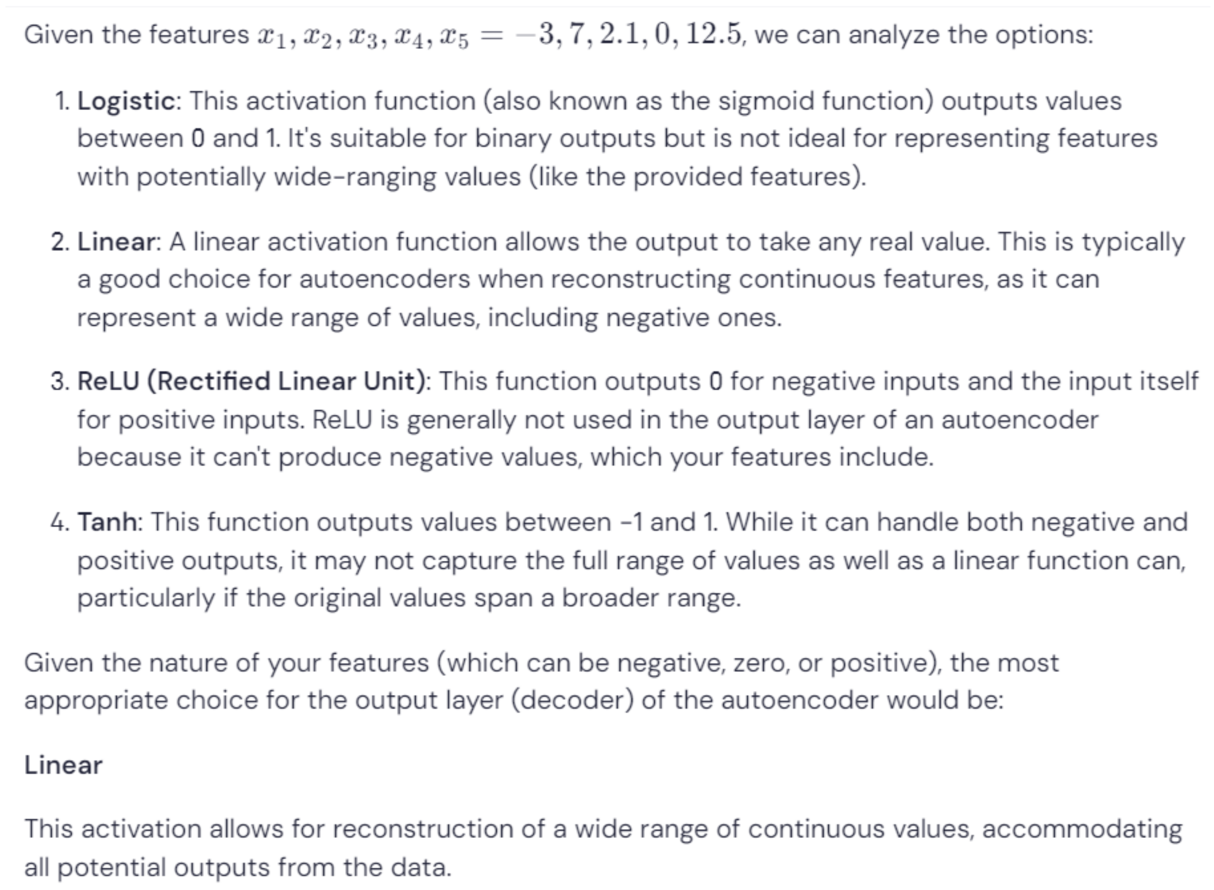
Let's evaluate the options again:

1. **6**: This is indeed less than 7. Hence, it is a valid dimension for the hidden layer.
2. **8**: This is greater than 7 and not valid.
3. **0**: An impractical option, as a hidden layer cannot have zero neurons.
4. **7**: This matches the input layer dimension and is therefore invalid for an undercomplete autoencoder.
5. **2**: This is also less than 7 and is another valid option for the hidden layer.

So, both **2** and **6** are valid dimensions for the hidden layer in this undercomplete autoencoder scenario. Thus, the correct answers could include **2** and **6**.

Suppose for one data point we have features x1, x2, x3, x4, x5 = -3, 7, 2.1, 0, 12.5, then which of the following functions should we use on the output layer (decoder)?

- Logistic
- Linear
- ReLU
- Tanh



Given the features $x_1, x_2, x_3, x_4, x_5 = -3, 7, 2.1, 0, 12.5$, we can analyze the options:

1. **Logistic**: This activation function (also known as the sigmoid function) outputs values between 0 and 1. It's suitable for binary outputs but is not ideal for representing features with potentially wide-ranging values (like the provided features).
2. **Linear**: A linear activation function allows the output to take any real value. This is typically a good choice for autoencoders when reconstructing continuous features, as it can represent a wide range of values, including negative ones.
3. **ReLU (Rectified Linear Unit)**: This function outputs 0 for negative inputs and the input itself for positive inputs. ReLU is generally not used in the output layer of an autoencoder because it can't produce negative values, which your features include.
4. **Tanh**: This function outputs values between -1 and 1. While it can handle both negative and positive outputs, it may not capture the full range of values as well as a linear function can, particularly if the original values span a broader range.

Given the nature of your features (which can be negative, zero, or positive), the most appropriate choice for the output layer (decoder) of the autoencoder would be:

Linear

This activation allows for reconstruction of a wide range of continuous values, accommodating all potential outputs from the data.

What is/are the primary advantages of Autoencoders over PCA?

- Autoencoders are less prone to overfitting than PCA.
- Autoencoders are faster and more efficient than PCA.
- Autoencoders can capture nonlinear relationships in the input data.
- Autoencoders require fewer input data than PCA.

1. **Autoencoders can capture nonlinear relationships in the input data**: Unlike PCA, which is a linear method that projects data onto a lower-dimensional space through linear combinations, autoencoders can learn complex nonlinear mappings, allowing them to capture intricate patterns in the data effectively. This is a significant advantage when the data has nonlinear structures.

The other statements are not necessarily true or require further clarification:

2. **Autoencoders are less prone to overfitting than PCA**: This statement is not accurate. Autoencoders can be prone to overfitting, especially if they are complex and the dataset is small. In contrast, PCA, a linear method, has

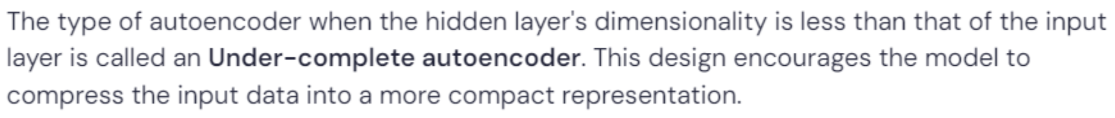
fewer parameters to optimize and can be less susceptible to overfitting in certain cases.

3. **Autoencoders are faster and more efficient than PCA**: This statement is generally false. Autoencoders, particularly deep neural networks, can be computationally intensive and may take longer to train compared to PCA, which is computationally efficient for linear projections.
4. **Autoencoders require fewer input data than PCA**: This statement is misleading. The amount of input data needed depends on the complexity of the autoencoder architecture and the data distribution. In many cases, autoencoders may require a significant amount of data to generalize well, especially if they have many layers or parameters.
5.

Question 5:

What type of autoencoder is it when the hidden layer's dimensionality is less than that of the input layer?

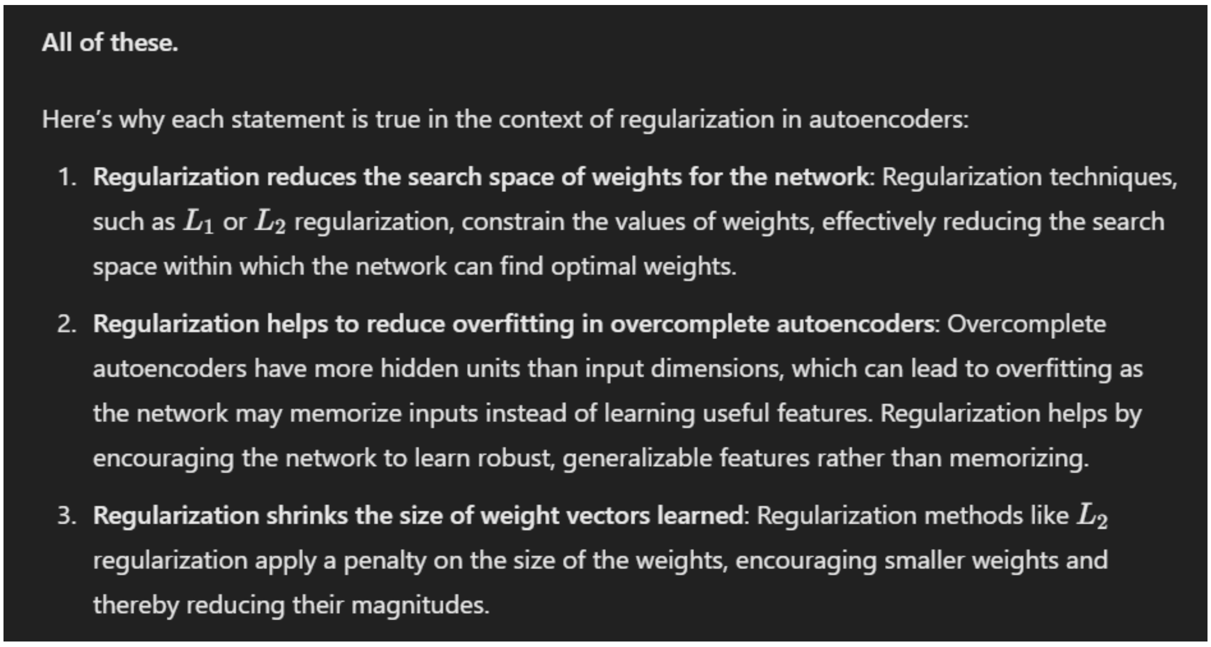
- Under-complete autoencoder
- Complete autoencoder
- Overcomplete autoencoder
- Sparse autoencoder


The type of autoencoder when the hidden layer's dimensionality is less than that of the input layer is called an **Under-complete autoencoder**. This design encourages the model to compress the input data into a more compact representation.

Question 6:

Which of the following statements about regularization in autoencoders is always true?

- Regularisation reduces the search space of weights for the network.
- Regularisation helps to reduce the overfitting in overcomplete autoencoders.
- Regularisation shrinks the size of weight vectors learned.
- All of these.


All of these.

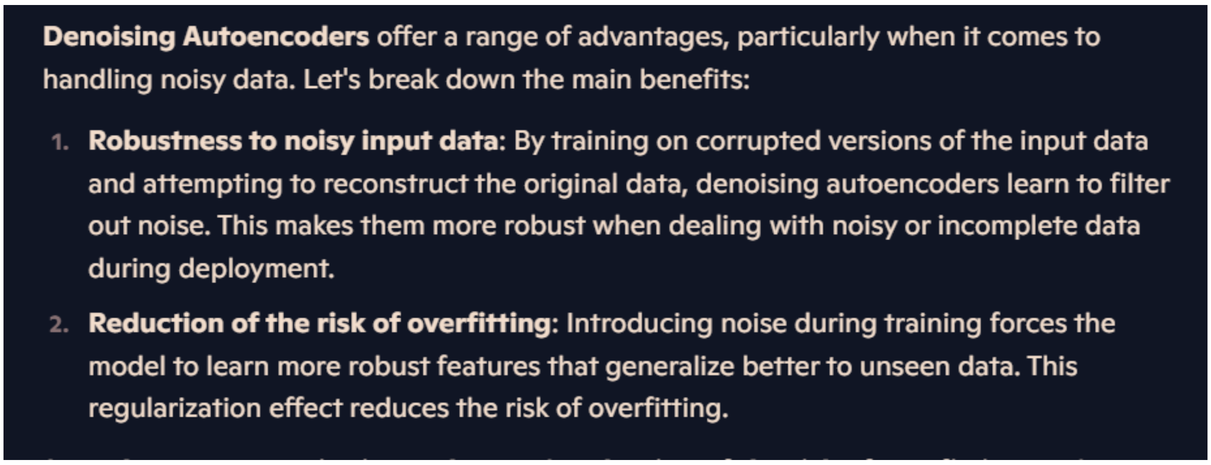
Here's why each statement is true in the context of regularization in autoencoders:

1. **Regularization reduces the search space of weights for the network**: Regularization techniques, such as L_1 or L_2 regularization, constrain the values of weights, effectively reducing the search space within which the network can find optimal weights.
2. **Regularization helps to reduce overfitting in overcomplete autoencoders**: Overcomplete autoencoders have more hidden units than input dimensions, which can lead to overfitting as the network may memorize inputs instead of learning useful features. Regularization helps by encouraging the network to learn robust, generalizable features rather than memorizing.
3. **Regularization shrinks the size of weight vectors learned**: Regularization methods like L_2 regularization apply a penalty on the size of the weights, encouraging smaller weights and thereby reducing their magnitudes.

Question 7:

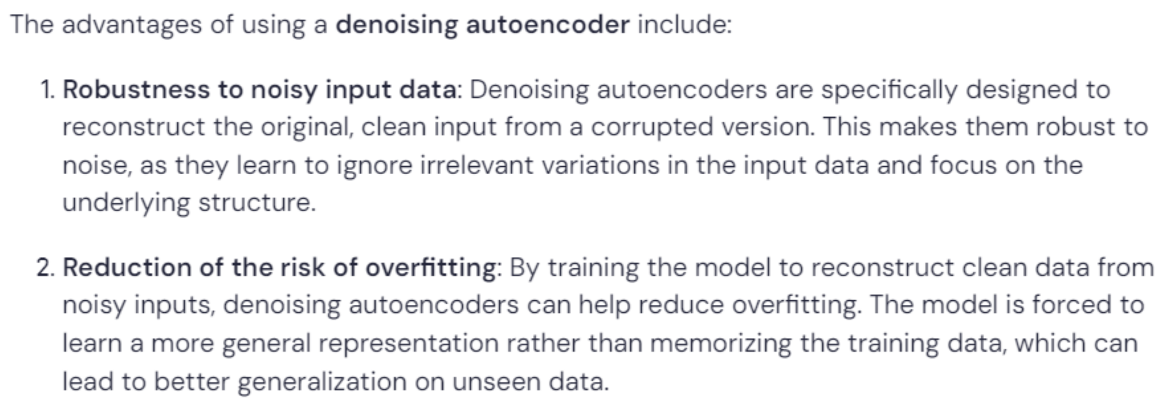
What are the advantages of using a denoising autoencoder?

1. Robustness to noisy input data
2. Reduction of the risk of overfitting
3. Faster training time
4. It promotes sparsity in the hidden layer



Denoising Autoencoders offer a range of advantages, particularly when it comes to handling noisy data. Let's break down the main benefits:

1. **Robustness to noisy input data:** By training on corrupted versions of the input data and attempting to reconstruct the original data, denoising autoencoders learn to filter out noise. This makes them more robust when dealing with noisy or incomplete data during deployment.
2. **Reduction of the risk of overfitting:** Introducing noise during training forces the model to learn more robust features that generalize better to unseen data. This regularization effect reduces the risk of overfitting.



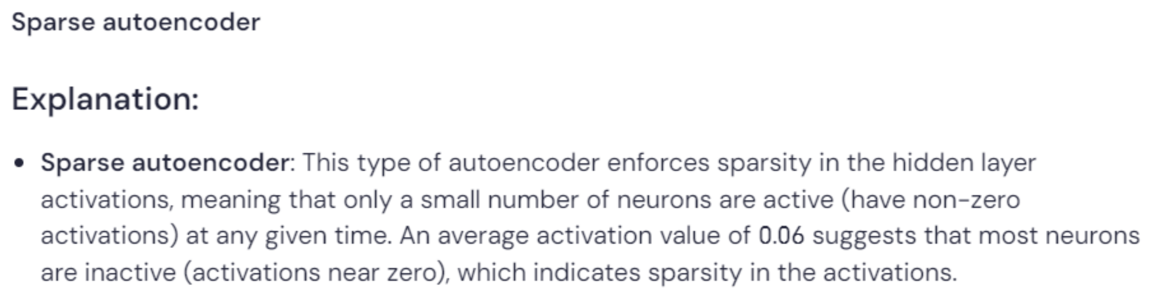
The advantages of using a **denoising autoencoder** include:

1. **Robustness to noisy input data**: Denoising autoencoders are specifically designed to reconstruct the original, clean input from a corrupted version. This makes them robust to noise, as they learn to ignore irrelevant variations in the input data and focus on the underlying structure.

2. **Reduction of the risk of overfitting**: By training the model to reconstruct clean data from noisy inputs, denoising autoencoders can help reduce overfitting. The model is forced to learn a more general representation rather than memorizing the training data, which can lead to better generalization on unseen data.

We are given an autoencoder A. The average activation value of neurons in this network is 0.06. The given autoencoder is:

1. Contractive autoencoder
2. Overcomplete neural network
3. Sparse autoencoder
4. Denoising autoencoder

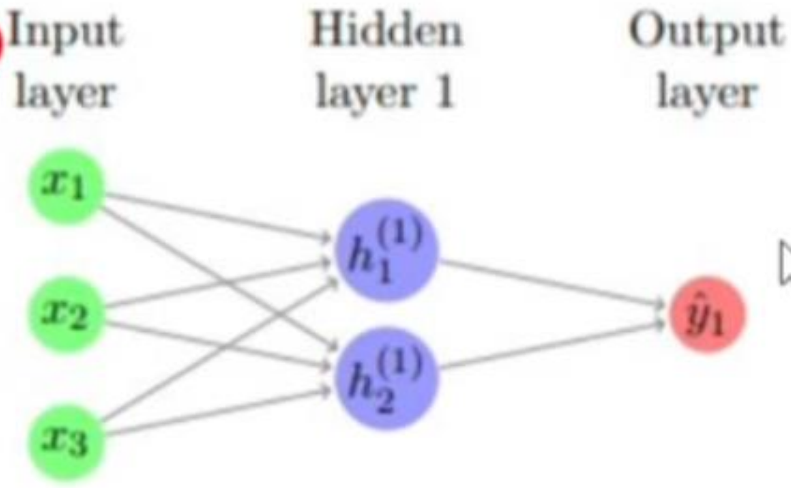


Sparse autoencoder

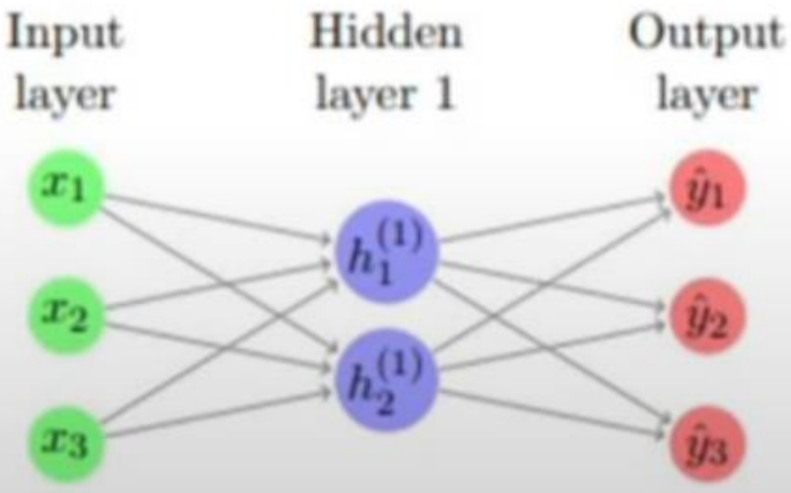
Explanation:

- **Sparse autoencoder**: This type of autoencoder enforces sparsity in the hidden layer activations, meaning that only a small number of neurons are active (have non-zero activations) at any given time. An average activation value of 0.06 suggests that most neurons are inactive (activations near zero), which indicates sparsity in the activations.

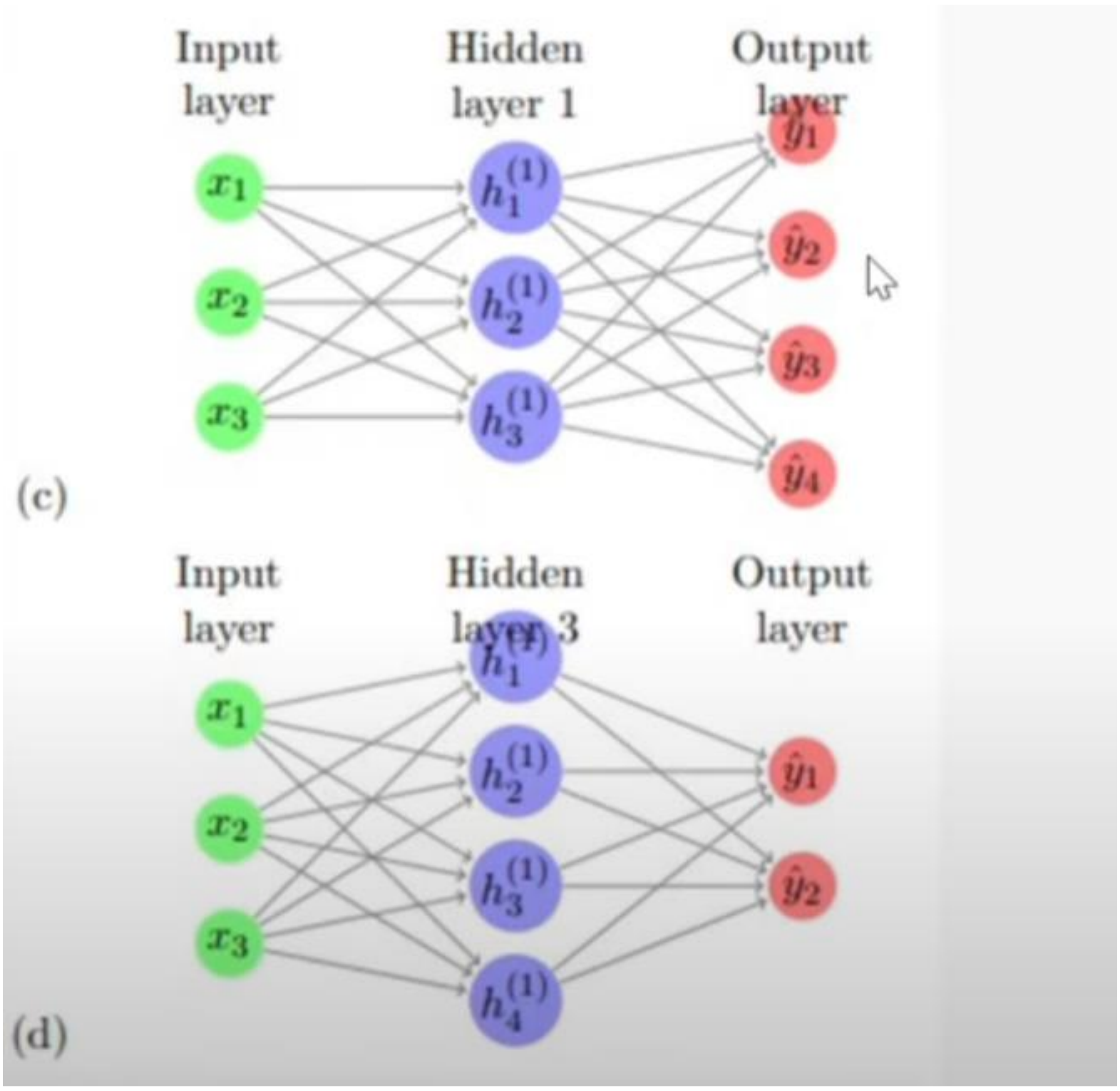
1) Which of the following networks represents an autoencoder?



(a)



(b)



Answer (B)

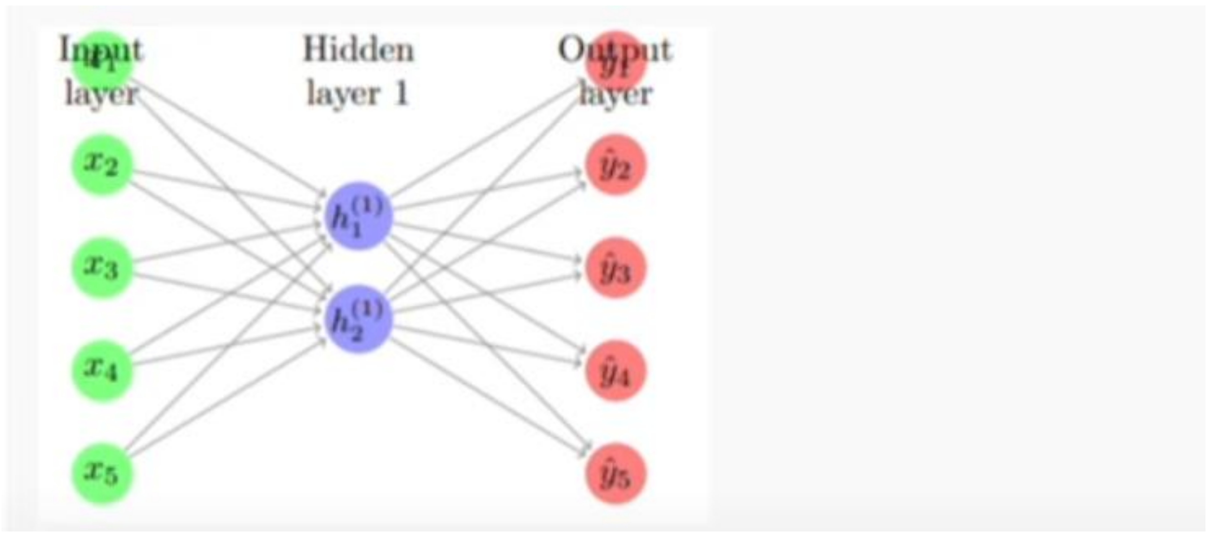
What is the primary reason for adding corruption to the input data in a denoising autoencoder?

- To increase the complexity of the model.
- To improve the model's ability to generalize to unseen data.
- To reduce the size of the training dataset.
- To increase the training time.

The correct answer is that adding corruption to the input data primarily aims **to improve the model's ability to generalize to unseen data**.

Question 7:

We are using the following autoencoder with linear encoder and linear decoder. The eigenvectors associated with the covariance matrix of our data X is (V1, V2, V3, V4, V5). What are the representations most likely to be learned by our hidden layer H? (Eigenvectors are written in decreasing order to the eigenvalues associated with them)



The representations most likely learned by the hidden layer H in a linear autoencoder, given the eigenvectors $(V1, V2, V3, V4, V5)$ associated with the covariance matrix of data X, will primarily correspond to:

The first few eigenvectors $V1$ and $V2$.

These eigenvectors represent the directions of maximum variance in the data, meaning the hidden layer captures the most significant features of the data along these dimensions.

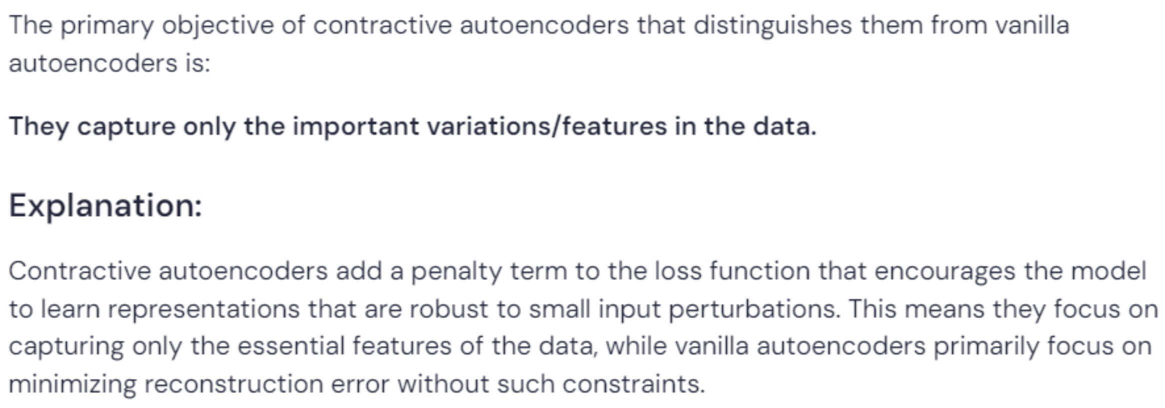
2. **What is the purpose of a decoder in an autoencoder?**

- To reconstruct the input data
- To generate new data
- To compress the input data
- To extract features from the input data

- **Answer: To reconstruct the input data.**

What is the primary objective of contractive autoencoders that distinguishes them from vanilla autoencoders?

- They learn a low-dimensional representation of the input data
- They minimize the reconstruction error between the input and the output
- They capture only the important variations/features in the data
- They maximize the mutual information between the input and the output



The primary objective of contractive autoencoders that distinguishes them from vanilla autoencoders is:

They capture only the important variations/features in the data.

Explanation:

Contractive autoencoders add a penalty term to the loss function that encourages the model to learn representations that are robust to small input perturbations. This means they focus on capturing only the essential features of the data, while vanilla autoencoders primarily focus on minimizing reconstruction error without such constraints.

Question 10:

What are the possible applications of autoencoders?

- Data Compression
- Extraction of important features
- Reducing noise
- All of these

Autoencoders can be applied for:

- **Data Compression**: Reducing the dimensionality of data while preserving important information.
- **Extraction of important features**: Learning useful representations for various tasks.
- **Reducing noise**: Denoising input data by reconstructing clean versions from noisy inputs.

Thus, the correct answer is **All of these**.

The question is:

1. Suppose we build a neural network for a 5-class classification task. Suppose for a single training example, the true label is [0 1 0 0 1] while the predictions by the neural network are [0.25 0.3 0.2 0.1 0.2]. What would be the value of cross-entropy loss for this example? (Answer up to two decimal places, Use base 2 for log-related calculations)

To calculate the cross-entropy loss for the given example, we can use the formula for cross-entropy loss for categorical data:

$$L = -\sum_{i=1}^{N} y_i \cdot \log_2(p_i)$$

Where:

- y_i is the true label (in this case, a one-hot encoded vector).
- p_i is the predicted probability for each class.
- N is the number of classes.

Given:

- True label: $y = [0, 1, 0, 0, 1]$
- Predictions: $p = [0.25, 0.3, 0.2, 0.1, 0.2]$

Step-by-step calculation:

Using the cross-entropy loss formula, we'll only consider the terms where $y_i = 1$:

- The true label vector indicates that the true classes are the second class and the fifth class. Therefore, we need to consider two terms: y_2 and y_5.

$$L = -\left(y_2 \cdot \log_2(p_2) + y_5 \cdot \log_2(p_5)\right)$$

Substituting the values:

$$L = -\left(1 \cdot \log_2(0.3) + 1 \cdot \log_2(0.2)\right)$$

Now, we compute the logarithms.

Now, we compute the logarithms.

1. $\log_2(0.3)$:

$$\log_2(0.3) \approx -1.737 \quad \text{(using a calculator)}$$

2. $\log_2(0.2)$:

$$\log_2(0.2) \approx -2.3219 \quad \text{(using a calculator)}$$

Now substituting these values back into the equation for loss:

$$L = -(-1.737 + -2.3219) = 1.737 + 2.3219 = 4.0589$$

Thus, rounding to two decimal places, the cross-entropy loss is:

$$\boxed{4.06}$$

Question 7:

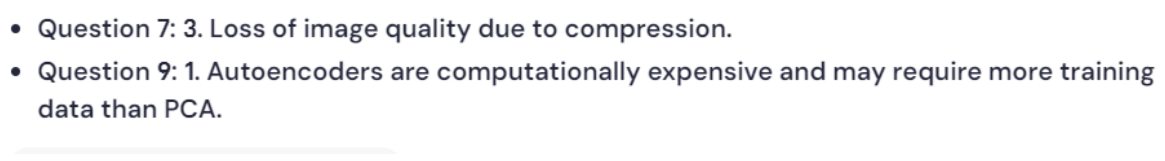
Which of the following problems prevents us from using autoencoders for the task of image compression?

1. Images are not allowed as input to autoencoders
2. Difficulty in training deep neural networks
3. Loss of image quality due to compression
4. Auto encoders are not capable of producing image output

Question 9:

Which of the following is a potential disadvantage of using autoencoders for dimensionality reduction over PCA?

1. Autoencoders are computationally expensive and may require more training data than PCA.
2. Autoencoders are bad at capturing complex relationships in data
3. Autoencoders may overfit the training data and generalize poorly to new data.
4. Autoencoders are unable to handle linear relationships between data.



- Question 7: 3. Loss of image quality due to compression.
- Question 9: 1. Autoencoders are computationally expensive and may require more training data than PCA.

Which of the following statements about overfitting in overcomplete autoencoders is true?

1. Reconstruction error is very low while training
2. Reconstruction error is very high while training
3. Network fails to learn good representations of input
4. Network learns good representations of input

Reconstruction error is very low while training.

Explanation:

In overcomplete autoencoders, where the number of hidden units exceeds the number of input features, the model can easily fit the training data very well. This leads to a low reconstruction error during training, but the downside is that it often results in the model overfitting to the training data, which means it may not generalize well to unseen data.

Question 4:

Which of the following statements about regularization in autoencoders is always true?

1. Regularisation reduces the search space of weights for the network.
2. Regularisation helps to reduce the overfitting in overcomplete autoencoders.
3. Regularisation shrinks the size of weight vectors learned.
4. All of these.

All of these.

Explanation:

- **Regularization reduces the search space of weights for the network**: This is true as regularization methods (like L1 or L2 regularization) impose constraints on the weights, effectively limiting their values and thus reducing the search space.
- **Regularization helps to reduce the overfitting in overcomplete autoencoders**: This is accurate because applying regularization techniques can help mitigate overfitting by preventing the model from adapting too closely to the training data.
- **Regularization shrinks the size of weight vectors learned**: This is also true, especially in the case of L2 regularization, which penalizes large weights and encourages smaller values.