Answer 1

(a) The Universal Approximation Theorem (UAT) states that a neural network with at least one hidden layer and a non-linear activation function can approximate any continuous function, meaning it can model any input-output relationship and minimize error for any dataset or loss function with enough neurons and training.
(b) The claim is wrong because:

- **Efficiency**: A single-layer network might need too many neurons, making it slow and hard to train.
- **Better Features**: More layers help the network learn better features from the data.
- **Easier to Train**: Deeper networks can be optimized better for complex tasks.

So, more layers are often better in practice.

Answer 2

(a) The main advantage of using a CNN over an MLP is that CNNs are specifically designed for image data. They can automatically detect important features like edges, shapes, and patterns, making them more efficient for image classification.
If Alice uses an MLP, she might face difficulties because MLPs don't handle spatial relationships in images well, leading to poorer performance and requiring many more parameters to learn effectively.
(b) While the convolution operation is indeed linear, the key difference lies in how CNNs process spatial relationships in the data. Convolutional layers preserve local patterns and are more effective at capturing features like edges and textures by taking into account the spatial structure of the image. MLPs, on the other hand, treat all inputs equally and don't consider the spatial arrangement, which leads to a loss of crucial information in images. Therefore, CNNs are better suited for image classification tasks despite both operations being linear between activations.

Answer 3

Since the problem is convex, there's only one minimum to find. Momentum can still help by speeding up how quickly we get to the minimum, especially if the path has some flat areas or changes a lot. But in this case, it's not necessary, because regular gradient descent can already do a good job of finding the minimum without much trouble.

Answer 4

PyTorch needs the loss tensor to be a scalar for backpropagation because it calculates the gradients with respect to a single value. The goal of backpropagation is to find how much each parameter contributed to the total loss, and this is only possible if the loss is a single number. If the loss were a tensor with multiple values, PyTorch wouldn't know how to assign responsibility for each part of the loss to the parameters.

Answer 5

The loss is computed as the sum of squared differences between the predicted output and the ground truth:

$$\text{loss} = \sum (W * x - y)^2$$

To compute the gradient of the loss with respect to the weights W, we apply the chain rule:

$$\frac{\partial \text{loss}}{\partial W} = 2 \sum (W * x - y) \cdot \frac{\partial (W * x)}{\partial W}$$

Since the gradient of $W * x$ with respect to W is x, the final expression is:

$$\frac{\partial \text{loss}}{\partial W} = 2 \sum (W * x - y) \cdot x$$

This gives us the gradient, which represents how the weights should be updated during training.

Answer 6

(a) It is generally unnecessary to add positional embeddings to an RNN model because RNNs naturally process data in sequence, carrying information step by step. However, it is technically possible to add positional embeddings, though it may not provide much benefit as RNNs already handle sequential order through their recurrent structure.
(b) To add positional embeddings to an RNN, the main change needed is to modify the input. You would add positional embeddings to the input sequence before feeding it into the RNN. This involves summing the positional information with the input data at each time step. The RNN itself does not need any structural changes, but you would need to ensure the positional embeddings are calculated and combined correctly with the inputs.

Answer 7

(a) Each pixel shows how much a word in the English sentence is linked to its translation in French.
(b) Rows with only one non-zero (bright) pixel indicate that a word in the French sentence has a clear, direct translation from a single word in the English sentence, meaning the model is confident about the one-to-one translation for that word.
(c) Rows with several non-zero (bright) pixels indicate that a word in the French sentence depends on multiple words in the English sentence for its translation. This often happens when translating phrases or when word meanings are spread across multiple words in the source language.
(d) Rows with one non-zero pixel have a white pixel because the model is highly confident about the direct translation for that word. In contrast, rows with multiple non-zero pixels have gray pixels because the attention is spread across multiple words, meaning the model is less certain and distributes its focus over several words.

Answer 8

The main difference between a basic GAN and WGAN is how they measure the difference between real and generated data. Basic GANs use a loss function based on binary cross-entropy, which can lead to issues like mode collapse and unstable training. WGAN, on the other hand, uses the Wasserstein distance to measure the difference, which leads to more stable training and helps avoid mode collapse.
The advantage of WGAN is that it provides better gradients for the generator, leading to smoother and more reliable learning.

Answer 9

(a) We rely on the fact that the KL-divergence term $D_{KL}(q(x_{1:T}|x_0)\|p_\theta(x_{1:T}|x_0))$ is non-negative and equals zero when q is equal to p_θ. Ignoring it assumes that the approximate posterior q is already close to the true posterior p_θ, so optimizing just the ELBO term is sufficient.
(b) We can't compute the KL-divergence term directly because it requires knowing the true distribution $p_\theta(x_{1:T}|x_0)$, which is often intractable. The distribution involves complex integrals or sums that are computationally expensive or impossible to solve exactly.
(c) We ignore the term $-D_{KL}(q(x_T|x_0)\|p_\theta(x_T))$ during training because it focuses on the final state, which is often less crucial during earlier stages of training. It simplifies optimization by focusing on the more immediate terms that impact the earlier steps, which contain more training information.

Answer 10

(a)

- **Vanishing gradients** occur when the gradients become very small during backpropagation, causing the model's weights to update very slowly or stop updating altogether. This often happens in deep networks when the gradient diminishes as it moves backward through layers, making it difficult for earlier layers to learn.
- **Exploding gradients** happen when the gradients grow too large during backpropagation, causing extremely large updates to the weights, which can make the model unstable or lead to divergence during training.

(b) Vanishing Gradients Example:
Consider a simple feedforward neural network with 3 layers and sigmoid activations. Assume the gradient of the loss with respect to the output layer is 0.1. During backpropagation, each layer applies the chain rule and the sigmoid's derivative (which is less than 1) reduces the gradient. If the derivative of the sigmoid is 0.1 at each layer, then after passing through 3 layers, the gradient becomes:

$$0.1 \times 0.1 \times 0.1 = 0.001$$

This very small gradient causes the weights in the earlier layers to update extremely slowly, leading to vanishing gradients.

Exploding Gradients Example:
Now, consider a deep network with ReLU activations. If the initial gradient is 1.5 and each layer's weight matrix has a large value, say 2, then after passing through 3 layers, the gradient becomes:

$$1.5 \times 2 \times 2 \times 2 = 12$$

This large gradient causes the weights to update excessively, leading to instability and potentially causing the training to diverge.

(c)

- **MLP (Multi-Layer Perceptron)**: Use **ReLU activation functions**. ReLU helps combat vanishing gradients because its derivative is 1 for positive inputs, preventing gradients from shrinking as they pass through layers. This ensures that gradients do not vanish in deep networks.
- **CNN (Convolutional Neural Network)**: Apply **batch normalization**. Batch normalization helps stabilize the gradient flow by normalizing the input to each layer, ensuring that the activations remain within a reasonable range. This reduces the chances of both vanishing and exploding gradients, leading to more stable training.
- **RNN (Recurrent Neural Network)**: Use **LSTM (Long Short-Term Memory)** units. LSTMs are designed to combat vanishing gradients by maintaining a memory cell that carries information across many timesteps without being affected by gradient decay. This structure helps the model retain long-term dependencies, reducing the impact of vanishing gradients.