Q1:

The relationship between the latent dimension h and classification accuracy is not strictly linear or monotonic. Instead, it typically follows a pattern where accuracy increases up to a certain point and then may plateau or even decrease as h continues to grow. Here's a breakdown of why this happens:

1. **Underfitting (Low h)**:

When the latent dimension is too small, the model may not have enough capacity to capture the underlying structure of the data. This leads to underfitting, where the learned latent space is not expressive enough to represent the data well. As a result, classification accuracy tends to be low.

2. **Optimal (h):**

As h increases, the model gains more capacity to learn meaningful representations of the data. The latent space becomes more expressive, allowing the classifier to better distinguish between classes. Classification accuracy improves up to a certain point, where the latent dimension is large enough to capture the relevant features but not so large as to overfit.

3. **Overfitting (High h):**

When h becomes too large, the model may start to overfit the training data. The latent space becomes overly complex, capturing noise or irrelevant details in the data rather than generalizable features. This can lead to a decrease in classification accuracy on unseen data, as the model loses its ability to generalize.

Expected Trend:

- **Classification accuracy increases up to a certain (h):** As the latent dimension grows, the model's capacity to learn useful features improves, leading to higher accuracy.

- **Accuracy plateaus or decreases after a certain (h):** Beyond a certain point, increasing (h) does not provide additional benefits and may harm performance due to overfitting.

Practical Considerations:

- The optimal value of (h) depends on the complexity of the dataset and the model architecture. For simpler datasets, a smaller (h) may suffice, while more complex datasets may require larger (h).

- Regularization techniques (e.g., dropout, weight decay) can help mitigate overfitting when using larger latent dimensions.

Q2:

The Universal Approximation Theorem (UAT) states that a feedforward neural network with a single hidden layer, containing a finite number of neurons, can approximate any continuous function on a compact subset of $\{R\}^n$ to arbitrary precision, provided the activation function is non-linear and not a polynomial. This means that, in theory, a single-hidden-layer MLP can model any continuous function, given enough neurons in the hidden layer.

How UAT Supports the Claim:

- The UAT guarantees that a single-hidden-layer MLP can achieve optimal error for any dataset and loss criterion, as long as the function being approximated is continuous and the network has sufficient capacity (i.e., enough neurons in the hidden layer).

- This implies that, in principle, a single-hidden-layer MLP can solve any regression or classification problem, provided it has enough neurons and is trained properly.

Why the Conclusion is Wrong:

While the UAT guarantees that a single-hidden-layer MLP can approximate any function, it does not imply that such a network is **practical** or **efficient** for all tasks. Here are the key reasons why using more than one hidden layer is often beneficial:

1. Efficiency in Representation:

- A single-hidden-layer MLP may require an **exponentially large number of neurons** to approximate certain functions, making it **computationally expensive** and **prone to overfitting.**

- Deep networks (with multiple hidden layers) can represent complex functions more compactly and efficiently by learning hierarchical features. Each layer can build on the

abstractions learned by the previous layer, leading to better generalization with fewer parameters.

2. Generalization Performance:

- Deep networks often generalize better than shallow networks because they can learn hierarchical representations of the data, which are more aligned with the underlying structure of many real-world problems (e.g., images, speech, text).

- A single-hidden-layer MLP may struggle to capture such hierarchical structures without an impractically large number of neurons.

3. Empirical Evidence:

- In practice, deep networks (with multiple hidden layers) consistently outperform shallow networks on a wide range of tasks, including regression, classification, and more complex tasks like image recognition and natural language processing.

- This empirical success of deep learning is a strong indication that deeper architectures are often necessary for real-world problems.

4. Optimization Challenges:

- While the UAT guarantees the existence of a solution, it does not guarantee that such a solution can be **found efficiently using gradient-based optimization methods.**

- Deep networks, despite being harder to train, often converge to better solutions because they can leverage the hierarchical structure of the data.

5. Task Complexity:

-For simple tasks, a single-hidden-layer MLP might suffice. However, for complex tasks , deeper networks are often necessary to achieve good performance.

Conclusion:

While the UAT theoretically supports the idea that a single-hidden-layer MLP can approximate any function, it does not account for practical considerations like efficiency, generalization, and optimization. In practice, deeper networks (with multiple

hidden layers) are often necessary to achieve good performance on complex tasks, making the conclusion that "there's never really a reason to use MLPs with more than one hidden layer" incorrect.

Q3:

(a) Advantages of CNN over MLP for Image Classification:

Main Advantage of CNN:

CNNs exploit the **spatial structure** of images using convolutional layers, which learn local patterns (e.g., edges, textures) efficiently. This makes them **parameter-efficient**, **translation-invariant**, and capable of learning **hierarchical features**.

Difficulties with MLP:

1. **Computational Inefficiency**: MLPs require fully connected layers for each pixel, leading to excessive parameters.

2. **Loss of Spatial Information**: MLPs flatten images, ignoring spatial relationships.

3. **Poor Generalization**: MLPs struggle with variations in object position, scale, or orientation.

4. **Overfitting**: High parameter counts increase overfitting risk.

(b) Is Convolution Just Linear?

Alice's claim is **incorrect**. While convolution is linear, CNNs are non-linear due to:

1. Non-linear Activations: Functions like **ReLU** introduce non-linearity.

2. **Pooling** Layers: Operations like max pooling are non-linear.

3. Hierarchical Learning: CNNs learn complex features through layers, unlike MLPs.

Why Alice is Wrong:

CNNs combine linear convolutions with non-linear activations and pooling, enabling them to model complex, hierarchical patterns in images, which MLPs cannot do effectively.

Conclusion:

CNNs are better for image classification due to their spatial efficiency and ability to learn hierarchical features. Alice's argument overlooks the non-linear components that make CNNs powerful.

Q4:

If **linear averaging** of accumulated gradients were used instead of **Exponential Moving Average (EMA)**, the efficiency of the optimization algorithm would likely be compromised. Here's why:

1. **Slower Adaptation**:

   Linear averaging would make the algorithm less responsive to recent changes in the gradient landscape. In stochastic optimization, where gradients can be noisy and the loss landscape can shift, this would lead to slower convergence.

2. **Increased Influence of Outdated Information**:

   Older gradients, which may no longer be relevant to the current optimization state, would have the same weight as recent gradients. This could introduce inefficiencies, especially in non-convex optimization problems where the gradient direction changes frequently.

3. **Loss of Momentum-Like Behavior**:

   EMA acts like a form of momentum, smoothing out noise in the gradients and helping the optimizer move more consistently toward the minimum. Linear averaging lacks this momentum-like behavior, making the optimization process less stable and potentially slower.

4. **Poor Handling of Noisy Gradients**:

   In stochastic optimization, gradients can be noisy. EMA effectively filters out this noise by emphasizing recent gradients, while linear averaging would retain the noise, leading to less efficient updates.

Q5:

PyTorch requires the loss tensor to be a scalar for backpropagation (loss.backward()) because:

Gradients are derivatives of a scalar function, and backpropagation computes gradients of the loss with respect to parameters.

The chain rule used in backpropagation applies to scalar outputs, making it efficient and straightforward.

Neural networks are trained by minimizing a single scalar loss (e.g., mean squared error, cross-entropy), which represents the overall error.

Computing gradients for a scalar loss is computationally efficient and avoids the complexity of handling multiple outputs (e.g., Jacobian matrices).

If the loss were not a scalar, **PyTorch would need additional mechanisms to handle multiple outputs**, increasing complexity and computational cost. This design ensures practicality and aligns with the mathematical foundations of optimization.

Q6:

(a) What is Inductive Bias?

Inductive bias refers to the set of assumptions or prior knowledge built into a machine learning model that guides its learning process and influences its predictions. It helps the model generalize from training data to unseen data by restricting the hypothesis space (the set of possible solutions the model can learn). Without inductive bias, a model would have no preference for one solution over another and would struggle to generalize effectively.

(b) Inductive Bias of CNNs

Convolutional Neural Networks (CNNs) have strong inductive biases tailored for image data. Two key biases are:

1. **Local Receptive Fields**:
    - CNNs assume that important patterns (e.g., edges, textures) are local and can be detected by small filters applied to small regions of the input. This bias reflects the spatial locality of features in images.
2. **Translation Invariance**:

- CNNs assume that the presence of a feature (e.g., an edge or object) is important regardless of its position in the image. This is achieved through weight sharing in convolutional filters, which allows the same filter to detect features anywhere in the input.

Other biases include:

- **Hierarchical Feature Learning**: CNNs assume that complex features can be built from simpler ones in a hierarchical manner (e.g., edges → textures → object parts).
- **Pooling**: CNNs assume that precise spatial information is less important than the presence of features, which is reflected in pooling layers that downsample feature maps.

(c) Pros and Cons of Inductive Bias

Pros:

1. **Improved Generalization**:

   Inductive bias helps models generalize better to unseen data by focusing on relevant patterns and ignoring irrelevant ones. Also small shifts in an input image doesn't really affect the outcome.

2. **Faster Convergence**:
   - Models with appropriate inductive biases require less data and fewer iterations to learn effectively, as they are guided toward plausible solutions.
3. **Reduced Overfitting**:
   - By restricting the hypothesis space, inductive bias prevents models from memorizing noise or irrelevant details in the training data.
4. **Computational Efficiency**:
   - Inductive bias reduces the complexity of the model, making training and inference faster and more memory-efficient.

Cons:

1. **Risk of Underfitting:** If the inductive bias is too strong or misaligned with the data, the model may fail to capture important patterns, leading to poor performance.

2. **Limited Flexibility**: Models with strong inductive biases may struggle to adapt to tasks or data that do not conform to their assumptions.
3. **Bias-Variance Trade-off**: Inductive bias reduces variance (overfitting) but can increase bias (underfitting) if the assumptions are incorrect.

Q7:

Part (a): Computational Time Complexity of the Attention Layer

The attention mechanism computes the output as:

$$\boldsymbol{Softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$$

1. Matrix Multiplication (QK^T) :

- Q and K are both matrices of size $\left(\{R\}^{\{n\ x\ d\}} \right)$

- The product (QK^T) results in a matrix of size R^{nxn}.

- Time Complexity: ($O(n^2\ d)$).

1. **Scaling and SoftMax**:
    - Scaling by $\frac{1}{\sqrt{d}}$ is $O(n^2)$
    - Applying the SoftMax function is also $O(n^2)$.
2. **Matrix Multiplication with V*V***:
    - The result of the SoftMax is an n×*n* matrix.
    - Multiplying this by *V* (which is n×*d*) results in an n×*d* matrix.
    - **Time Complexity**: $O(n^2 d)$.

Total Time Complexity: The dominant terms are $O(n^2 d)$ from the matrix multiplications.

(b): Bob's Proposed Change

Bob suggests changing the attention operation to:

$$\boldsymbol{Softmax}\left(\frac{Q}{\sqrt{d}}\right)K^T V$$

1. **Matrix Multiplication $\boldsymbol{K^T V}$**:
    - K is R^{dxn} and V is R^{nxd}.
    - The product $K^T V$ results in a matrix of size R^{dxd}.
    - **Time Complexity**: $O(nd^2)$.
2. **Scaling and Softmax**:
    - Scaling Q by $\frac{1}{\sqrt{d}}$ is $O(nd)$.
    - Applying the Softmax function to Q is $O(nd)$.
3. **Matrix Multiplication with $\boldsymbol{K^T V}$**:
    - The result of the Softmax is an n×d matrix.
    - Multiplying this by ($K^T \cdot V$) (*K* which is dx*d*) results in an n×d matrix.
    - **Time Complexity**: $O(nd^2)$.

Total Time Complexity: The dominant term is $O(nd^2)$.

Why Does This Help Reduce Computational Time Complexity?

- **Original Complexity**: $O(n^2 d)$.
- **Bob's Complexity**: $O(nd^2)$.

Given that d<<, : $O(nd^2)$.is significantly smaller than $O(n^2 d)$ This reduction is because Bob's approach avoids the n2*n*2 term by first reducing the dimensionality through $K^T V$

Does This Preserve the Function of the Original Attention Formulation?

No, it does not preserve the exact function. The original attention mechanism computes pairwise interactions between all tokens in the sequence, which is crucial for capturing the full context. Bob's approach reduces the dimensionality early, potentially losing some of these interactions and thus altering the attention mechanism's behaviour. This could impact the model's ability to capture complex dependencies in the data.

Q8:

(a) What does each pixel in the map represent?

Each pixel in the attention map represents the **attention weight** between a specific **source token (English)** and a **target token (French)**. The intensity of the pixel (e.g., white, gray, or black) indicates the strength of the attention weight:

- **Bright (white) pixels**: High attention weights, meaning the model is strongly focusing on this source token when generating the corresponding target token.
- **Gray pixels**: Moderate attention weights, indicating some focus but not as strong.
- **Dark (black) pixels**: Low or negligible attention weights, meaning the model is ignoring this source token for the current target token.

(b) What is the meaning of having rows with only one non-zero pixel?

A row with only one non-zero pixel indicates that the model is **strongly attending to a single source token** when generating the corresponding target token. This typically happens when the target and the source are almost in one to one alignment

(c) What is the meaning of rows that have several non-zero pixels?

Rows with several non-zero pixels indicate that the model is **attending to multiple source tokens** when generating the corresponding target token. This typically happens when:

- The target token depends on **context from multiple source tokens** (e.g., idiomatic expressions, reordering, or grammatical differences between languages).
- The translation requires **contextual understanding** rather than a direct word-for-word translation.

(d) Explain why only rows with one non-zero pixel have white pixels, while the rest have only grey pixels.

- **Rows with one non-zero pixel (white pixel)**: These rows represent **strong, focused attention** on a single source token. The attention mechanism assigns a high weight (close to 1) to one source token and negligible weights

(close to 0) to the rest. The high weight is represented by a **white pixel**, indicating a clear and unambiguous alignment.

- **Rows with several non-zero pixels (gray pixels)**: These rows represent **distributed attention** across multiple source tokens. The attention mechanism assigns moderate weights (less than 1) to several tokens, resulting in **grey pixels**. This indicates that the model is considering multiple tokens to generate the target token, but no single token dominates the attention.

Q9:

(a) What is the mathematical basis we rely on when we ignore the KL-divergence term?

The mathematical basis for ignoring the KL-divergence term during training is rooted in the **Evidence Lower Bound (ELBO)** formulation. The ELBO is derived from the **variational inference** framework, which approximates the true posterior distribution $p\theta(x1{:}T \mid x0)$ with a simpler distribution $q(x1{:}T \mid x0)$.

The ELBO is given by:

$$log\ p\theta(x0) \geq Eq\left[\frac{log\ p\theta(x0{:}T)}{q(x1{:}T \mid x0)}\right]$$

The KL-divergence term $DKL(q(x1{:}T \mid x0) \mid\mid p\theta(x1{:}T \mid x0))$ measures the difference between the approximate posterior q and the true posterior $p\theta$. However, during training, we focus on **maximizing the ELBO**, which indirectly minimizes the KL-divergence. Since the KL-divergence term is non-negative, dropping it does not affect the validity of the lower bound. The ELBO still provides a valid objective for training the model.

(b) Why can't we compute the KL-divergence term?

The KL-divergence term $DKL(q(x1{:}T \mid x0) \mid\mid p\theta(x1{:}T \mid x0))$ cannot be computed directly because:

1. **Intractability of the True Posterior**: The true posterior $p\theta(x1{:}T \mid x0)$ is often intractable, especially in complex models like diffusion models or deep generative models. It involves integrating over all possible latent states, which is computationally infeasible.
2. **Complexity of $q(x1{:}T \mid x0)$**: The approximate posterior $q(x1{:}T \mid x0)$ is designed to be simpler than the true posterior, but it may still involve complex dependencies that make the KL-divergence difficult to compute analytically.

Instead of computing the KL-divergence directly, we optimize the ELBO, which avoids the need to evaluate the intractable true posterior.

(c) Why do we ignore the term $-\boldsymbol{DKL}(\boldsymbol{q}(\boldsymbol{xT} \mid \boldsymbol{x0}) \mid\mid \boldsymbol{p\theta}(\boldsymbol{xT}))$ in training?

The term $-DKL(q(xT \mid x0) \mid\mid p\theta(xT))$ is ignored during training because:

1. **Role of xT**: In diffusion models, xT represents the final state of the forward process, which is typically a simple distribution (e.g., Gaussian noise). The term $DKL(q(xT \mid x0) \mid\mid p\theta(xT))$ measures the difference between the distribution of xT given x0 and the prior distribution $p\theta(xT)$. Since xT is designed to be independent of x0 and close to the prior, this term is often negligible.
2. **Focus on the Reverse Process**: The primary goal of training is to learn the **reverse process** (from xT to x0), which is captured by the other terms in the ELBO. The term $-DKL(q(xT \mid x0) \mid\mid p\theta(xT))$ does not contribute significantly to learning the reverse process and can be safely ignored without affecting the model's performance.

Q10:

(a) Explain why Bob is wrong.

Bob's wrong because a **standard autoencoder** (like the one trained in the wet assignment) is not inherently a generative model. Here's why:

1. **Latent Space Structure**:

- In a standard autoencoder, the latent space (the output of the encoder) is not structured or regularized. The encoder maps input data to latent vectors in an arbitrary way, and the decoder learns to reconstruct the input from these latent vectors.

- Since the latent space is not constrained, randomly sampling latent vectors and feeding them to the decoder does not guarantee meaningful or realistic outputs. The decoder is only trained to reconstruct inputs from the training set, not to generate new, realistic samples.

2. **No Probabilistic Framework**:

- A standard autoencoder does not model the probability distribution of the data. It simply learns a deterministic mapping from inputs to latent vectors and back.

- Without a probabilistic framework, there is no way to ensure that randomly sampled latent vectors correspond to valid data points in the input space.

3. **Overfitting**:

- Standard autoencoders are prone to overfitting, especially if the latent space is high-dimensional. The decoder may learn to reconstruct training data perfectly but fail to generalize to new, unseen latent vectors.

(b) Variational Autoencoders (VAEs)

Alice's excitement about using autoencoders as generative models is well-founded, as **Variational Autoencoders (VAEs)** address the limitations of standard autoencoders and enable generative modeling.

Key Differences from Standard Autoencoders:

1. **Probabilistic Latent Space:**

- In VAEs, the latent space is modeled as a probability distribution (typically Gaussian). This allows for structured and continuous latent spaces, enabling meaningful sampling.

- In contrast, standard autoencoders use deterministic latent vectors.

2. **Regularization via KL-Divergence:**

- VAEs introduce a **KL-divergence term** in the loss function to regularize the latent space. This term encourages the learned latent distributions to match a prior distribution (usually a standard Gaussian).

3. **Generative Capability**:

- By sampling latent vectors from the prior distribution $(p(z))$ and feeding them to the decoder, VAEs can generate new, realistic data points.

- This is because the latent space is structured and regularized, ensuring that sampled latent vectors correspond to valid regions of the input space.

How VAEs Enable Generative Modeling:

1. **Sampling from the Prior:**

   After training, VAEs allow sampling from the prior distribution ($p(z)$) (e.g., a standard Gaussian). These samples are fed to the decoder to generate new data points.

2. **Continuous Latent Space**:

   The probabilistic nature of the latent space ensures that small changes in the latent vector result in smooth changes in the generated output. This enables interpolation and exploration of the data manifold.

3. **Trade-off Between Reconstruction and Regularization**:

   The KL-divergence term in the VAE loss ensures that the latent space is well-structured, while the reconstruction term ensures that the generated outputs are faithful to the training data.