Dry Assignment

1)

Assuming only the latent dimension changes in each experiment, the classification accuracy would first increase up to a certain value of h, and then it would start to decrease.

Firstly, when starting with low value of h, the latent space would lack sufficient capacity to capture significant features of the data. Then, as h increases, the model would be able to represent more complicated features, which would lead to better class separability in the latent space, and improve the classification accuracy.

At a certain point, h would become too large, and the latent space might capture unnecessary details instead of focusing on important features. The model might also memorize training data rather than learning it which would lead to overfitting. Since the model encodes too many features which are not necessary and maybe even overfits, the class separation in the latent space becomes less effective, which reduces classification accuracy.

To conclude, h would start low, increase up to a certain value, and then decrease.

2)

a)

The universal approximation theorem states that a neural network, with a single hidden layer, and a finite number of neurons, can estimate any continuous function to a very accurate precision, given an appropriate activation function.

This basically means that a neural network with enough capacity can learn almost any pattern, and represent any continuous one-dimensional function, within a specified range, to any level of accuracy we would like to achieve.

Therefore, according to the UAT, MLP with one hidden layer can model any function well enough to minimize error for any dataset and loss function. Therefore, since regression tasks involve estimating a continuous function f according to samples we train, the friend's claims are supported, since technically we can achieve optimal error over any dataset and loss criteria with only one hidden MLP layer.

b)

Even though it is technically possible to achieve the accuracy we want using only one layer, that does not mean that this is the optimal solution. If the task is complex, and we only use one layer, the MLP would need a huge number of neurons to capture complicated patterns, therefore the model would be very large and computationally expensive to store and to train.

Moreover, shallow networks tend to struggle with capturing complicated patterns and might end up fitting the training data too much which would result in overfitting.

Therefore, even though it is technically possible to use only one layer, it has many disadvantages and is not always the best way.

3)

a)

In the case of an image classification task, over a large, annotated dataset, the main advantage of CNN over MLP is parameter efficiency.

CNNs use convolutional layers that share weights across the image, which means that the same filter scans different parts of the image- which significantly reduces the number of parameters.

On the other hand, MLP treats the image as a flat vector, and uses fully connected layers, where is pixel is connected to a different neuron, and therefore they use a lot of parameters.

Therefore, MLP requires significantly more parameters than CNNs, which might make them impractical, especially when the dataset is large.

Moreover, because MLP treats the image as a flat vector, it loses all the spatial features such as shapes and textures, which CNNs are able to capture efficiently.

The main difficulty Alice might encounter if she chooses to use MPL is the high memory and computational cost required to store and process all the parameters. Moreover, since there are many parameters, the MLPs might memorize the training image instead of learning the patterns. As a result, they would not be very good as classifying unseen images- meaning they would overfit the data and the generalization abilities of the model would be poor.

b)

The claim Alice made that convolution operation is linear is true, but the conclusion she reached that there wouldn't be a difference between using CNNs and MLPs is false.

CNNs include activation functions- such as ReLu, which are non-linear, after each convolutional layer. These functions introduce non-linearity to the model, which enables it to learn more complex patterns, that purely linear models such as MPL with only linear layers, could not learn.

Additionally, as mentioned before CNNs preserve spatial relationships in the data and use weight sharing, which makes them more efficient at detecting features such as textures and edges. In contrast MLPs are not able to capture spatial relationships as they treat images as flat vectors.

Therefore, there is a difference between using CNNs and MLPs, and her conclusion is incorrect.

4)

EMA is a type of moving average, which assigns greater weight to recent data points while gradually decreasing the weight of the older data points. This allows the optimizer that uses EMA to react faster to changes in the loss landscape.

In contrast, linear averaging gives equal weight to all gradients regardless of when they were obtained. Thus, old gradients influence updates just as much as newer gradients.

As a result, optimizers that use EMA prioritize recent gradients, which allows them to adjust more efficiently to changes in the gradient direction.

On the other hand, optimizers using linear averaging would adapt more slowly because new gradients are not given greater significance than older ones. This would make the optimizer less responsive to new trends, which as a result would reduce its efficiency.

Because EMA gives more weight to recent gradients while still taking past ones into account, it helps the optimizer adapt smoothly to changes during training and keeps the learning process stable. On the other hand, linear averaging treats all gradients equally, no matter when they were computed. This can cause the optimizer to rely too much on outdated information, making it slower to respond to new trends in the data. As a result, training may converge more slowly, or in some cases not at all, especially when the learning environment changes.

Therefore, if we replace EMA with linear averaging, the optimization algorithm would be less efficient, because it would adapt more slowly to new update**s** and might struggle to converge to the optimal solution in a reasonable timeframe.

5)

As we learned in tutorial 5, backpropagation is basically a forward pass followed by reverse mode automatic differentiation through the computational graph of a neural network.

PyTorch requires the loss tensor to be a scalar in order to run loss.backward() method because it uses the chain rule to compute gradients, which passes gradients from a single scalar loss back to every model parameter. When the loss is a scalar, each parameter receives one gradient value.

However, if loss.backward() were called on a non-scalar loss like a vector, PyTorch would have to compute and store a full Jacobian matrix (containing the gradients of multiple outputs with respect to all parameters) which is computationally expensive and usually unnecessary. That's why PyTorch expects you to reduce vector losses (for example using .sum()) before calling .backward().

6)

a)

The inductive bias of a learning algorithm refers to the assumptions that the algorithm makes in order to make predictions on unseen data based on its training data.

The inductive bias of an algorithm helps it navigate the trade-off between fitting the training data with perfect accuracy (overfitting) and generalizing well to data it has not encountered (underfitting).

b)

One of the inductive biases in CNNs is that in an image nearby pixels are more closely related than distant ones. Based on this assumption and to take advantage of it, convolution layers use small filters (kernels) that slide over local regions of the picture. This helps the network first pick up on simple features like edges or textures, which can then be combined into more complex patterns in the deeper layers.

Another assumption is that the network should still recognize an object even if it shifts a little bit in the image. CNNs handle this using weight sharing in convolution layers as well as max pooling, which help the model focus on the most important features while being less sensitive to small movements. Max pooling selects the maximum element from the region of the feature map covered by the filter. Therefore, after max-pooling layer the output would be a feature map which contains the most significant features of the previous feature map. This allows CNNs to recognize objects even when their position slightly changes.

c)

One of the advantages of using inductive bias is that it can make the models more efficient. For example, in the case of CNNs that we discussed in the previous answer, weight sharing- where the same filter scans different parts of the image- significantly reduces the number of parameters. This lowers the computational cost, making the model more efficient.

Another advantage of using inductive biases is that it can lead to better generalization if the model's assumptions match the real data distribution.

One disadvantage of using inductive biases is that it might lead to underfitting. If the model has too much inductive bias, it might overlook patterns that don't fit its assumptions, preventing it from learning key relationships in the data.

Another disadvantage is that the flexibility of the model might be compromised. A model with strong bias is less adaptable to diverse data, because its predefined assumptions limit the range of patterns it can learn.

One case where we would prefer <u>more inductive bias</u> is when data is limited. In such cases, a model with strong inductive bias can generalize better, as it does not need to learn everything from scratch.

However, when we have a large dataset, it is better to <u>minimize inductive bias</u>. With enough training data, a model with less assumptions is able to learn patterns directly from the data, often achieving higher accuracy.

For instance, vison transformers (ViTs) have minimal inductive bias, as they learn relationships between image patches directly from the data, rather than assuming spatial locality like CNNs. Therefore, this greater flexibility allows ViTs to outperform CNNs on large datasets.

7)

1)

First, we need to compute QK^T. Both matrices are of size $(n \times d)$. Therefore,

the time complexity of computing QK^T, is $O(n^2d)$. Then, we need to divide by $\sqrt{d}$ and then apply softmax which takes $O(n^2)$. Afterwards, we multiple the result by the matrix V which has the dimensions $(n \times d)$, so it takes $O(n^2d)$

Overall, the time complexity is $O(n^2d)$

2)

Using Bob suggestion, we would first compute K^TV which takes $O(nd^2)$ time. We would then apply softmax on $\frac{Q}{\sqrt{d}}$ which takes $O(nd)$ and then compute multiply the two results which takes $O(nd^2)$

Thus, computing it as Bob suggests would take $O(nd^2)$. Since d<<n, this time complexity is better than the original time complexity which took $O(n^2d)$.

However, even though Bob's method is more efficient, it does not preserve the original behavior of the attention formulation. In the standard attention formulation, we calculated QK^T, which result in $(n \times n)$ matrix capturing the attention score between every pair of tokens. These scores reflect how much each token attends to every other token in the sequence.

Bob's version replaces this token-to-token interaction with a global transformation K^TV, which removes the pairwise interactions between tokens. As a result, the model loses the ability to capture contextual dependencies and cannot adjust attention based on the specific query Q. This significantly limits the expressiveness and effectiveness of the attention mechanism.

8)

a)

Each pixel in the map represents the attention weight that was assigned to that pixel by the model when translating the sentence “The agreement on the European Economic Area was assigned in august 1992” from English to French. The rows in the map correspond to the French (target) sentence words and the columns correspond to the English (source) sentence. The brightness of the pixel represents how much attention the model gives to a certain word in the source when generating the target word, the brighter the pixel is- the more the model focuses on the specific English word the pixel represents when generating the corresponding French word.

b)

When a row has only one non-zero entry, it means that the model assigns all of its attention to a single English word when generating a certain French word. This means that the word translates directly from English to French, and it doesn’t require additional context for the translation. For example, 1992 is the same in both languages, regardless of the context of the sentence, so it can be directly translated and thus in its row it’s the only pixel that is bright.

c) Rows with multiple non-zero pixels indicate that the model distributes its attention across several source words when generating a target word. This distribution reflects the model's need to consider multiple words to accurately translate a particular word or phrase. This means that the translation of the word requires more context

d) The reason why rows with only one non-zero pixel appear white while the rest are gray comes down to how attention weights are distributed. In attention maps, each row represents a target word, and the values sum to 1. When all the attention is focused on a single source word, that pixel gets the highest possible intensity (white). However, when attention is spread across multiple words, the weights are divided, making each individual pixel dimmer (gray).

This happens because the attention mechanism uses a softmax function, which normalizes the weights into a probability distribution. As a result, strong one-to-one alignments produce bright white pixels, while more distributed attention results in gray pixels.

9)

1) We can express the log-likelihood $log p_\theta(x_0)$ as the sum of a KL divergence and the ELBO:

$$log p_\theta(x_0) = D_{KL}\big(q(x_{1:T}|x_0)\big|p_\theta(x_{1:T}|x_0)\big) + E_q\left[\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right]$$

Since the KL divergence is always non-negative:

$$D_{KL}\big(q(x_{1:T}|x_0)\big|p_\theta(x_{1:T}|x_0)\big) \geq 0$$

the ELBO serves as a lower bound on the true log-likelihood:

$$log p_\theta(x_0) = D_{KL}\big(q(x_{1:T}|x_0)\big|p_\theta(x_{1:T}|x_0)\big) + E_q\left[\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \geq E_q\left[\log\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right]$$

This means that maximizing the ELBO indirectly maximizes $log p_\theta(x_0)$, even if we don't compute the KL term directly.

2)

We can't compute the KL-divergence term it involves the reverse process $p_\theta(x_{1:T}|x_0)$, which is complicated to compute and not explicitly defined. Calculating it would require going through all possible steps in the reverse sequence, which is impractical in deep models. Therefore, we instead focus on optimizing ELBO as it is something we are able to compute, and as we proved in the previous answer maximizing the ELBO also increases the overall log-likelihood.

3)

We can ignore the term $-D_{KL}(q(x_T|x_0)||p_\theta(x_T))$ because it doesn't involve any learnable parameters and therefore doesn't affect gradient-based optimization. Theis KL term compares the final noisy state to a fixed prior distribution, that is usually set as $p_\theta(x_T) = N(0, I)$ and is independent of the model parameters θ. Since it doesn't contribute anything to the gradients, we can ignore it.

10)

a)

Bob claims that since the decoder learns to turn latent vectors into digit images, we can simply feed it random numbers to generate new, realistic-looking handwritten digits.

However, Bob is incorrect because not all random vectors will generate meaningful digits. If we sample latent vectors without considering the learned distribution, the decoder may produce unrealistic or nonsensical images instead of valid handwritten digits. For meaningful generation, the sampled latent vectors should come from the same distribution as those learned by the encoder during training.

b)

Variational Autoencoders (VAEs) are generative models that create new data similar to the input they are trained on. They consist of encoders which learn significant patterns from input data, and decoders which use those learnt patterns to reconstruct the image.

The difference between VAEs and standard autoencoders is that they learn a probabilistic latent space instead of encoding inputs into fixed latent vectors. In a standard autoencoder, the encoder maps each input to a single point in latent space, which limits its ability to generate new data. On the other hand, VAEs output a distribution, which includes mean and variance, for each input, allowing for random sampling from the latent space. This is achieved through the reparameterization trick, which ensures that sampling remains differentiable for training. Additionally, VAEs introduce a KL divergence loss, which forces the learned latent space to follow a smooth, continuous distribution.

These modifications enable VAEs to function as generative models, allowing them to produce entirely new, realistic samples by sampling from the learned latent space rather than simply reconstructing existing data.