University of Oslo Fall, 2024
Department of Mathematics Exams

Author: Jonas Semprini Næss

TEK5040 - Deep Learning for Autonomous Systems

Exam 2023

Task 1 - Field Of View

Answer:

$k = 2$ gives a field of view (FOV) 7 which is less than the required FOV 8. $k = 3$ gives a FOV 9 which is greater than the required FOV. Therefore, the minimum k that gives a larger FOV than 8 is 3.

Task 2 - Self-Attention and Transformers

Answer:

1. Characteristics of the Attention Modules

- **Encoder Attention (Self-Attention in Encoder):**
  - Operates solely within the input sequence.
  - Computes attention for each token by considering all other tokens in the input.
  - Captures contextual relationships between input tokens.
  - Outputs a sequence of transformed input embeddings.
- **Decoder Attention (Self-Attention in Decoder):**
  - Operates within the output sequence (partial or full translations).
  - Uses a *causal mask* to prevent tokens from attending to future tokens, ensuring autoregressive generation.
  - Outputs a sequence of context-aware embeddings for the output tokens.
- **Encoder-Decoder Attention:**
  - Allows the decoder to attend to the encoder's output sequence.
  - Focuses on relevant parts of the input sequence for generating each output token.
  - Combines encoder output embeddings (keys, values) with the decoder's output embeddings (queries).

2. Computational Complexity of Self-Attention

The self-attention mechanism involves three main operations:

1. Compute the *query* (Q), *key* (K), and *value* (V) matrices from input embeddings. Each has dimensions $n \times d$, where n is the sequence length and d is the embedding size.

2. Compute the attention scores:

$$A = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)$$

   Multiplying $Q(n \times d)$ with $K^\top(d \times n)$ results in a matrix of size $n \times n$. This operation requires $\mathcal{O}(n^2 d)$ multiplications.

$$\text{Output} = AV$$

   Multiplying $A(n \times n)$ with $V(n \times d)$ also requires $\mathcal{O}(n^2 d)$ multiplications.

Overall Complexity: The dominant term in these operations is $\mathcal{O}(n^2 d)$ from matrix multiplications. Thus, the computational complexity of self-attention is:

$$\mathcal{O}(n^2 d)$$

3. Reducing Computational Complexity

A few methods to reduce the complexity of self-attention are:

- **Sparse Attention:**
  - Compute attention only for a subset of tokens using local windows or predefined sparsity patterns.
  - This reduces complexity to $\mathcal{O}(nd^2)$ or $\mathcal{O}(n \log n \cdot d)$, depending on the sparsity structure.
- **Low-Rank Decomposition:**
  - Approximate the attention score matrix $QK^\top$ using a low-rank factorization.
  - Complexity is reduced to $\mathcal{O}(n \cdot r \cdot d)$, where $r \ll n$ is the rank of the approximation.
- **Efficient Variants (e.g., Linformer, Performer):**
  - **Linformer:** Projects keys and values into a lower-dimensional space to reduce the size of attention matrices.
  - **Performer:** Uses kernel-based approximations to compute attention in $O(n \cdot d)$.

Task 3 - Q-Learning

Answer:

Sampling Operation (Line 4)

In SARSA, the next action A' is sampled using the current policy (e.g., ϵ-greedy). To handle the case where the policy is unknown, Q-learning modifies this step by assuming the agent acts optimally. Instead of sampling A', the agent selects the action A^* that maximizes the Q-value for the next state S':

$$A^* = \arg\max_{a \in \mathcal{A}} Q(S', a)$$

where the action is selected with probability $1 - \epsilon$, or any other action otherwise with probability $\frac{\epsilon}{K-1}$ where K is the number of classes (i.e Greedy-epsilon policy). The Soft-epsilon policy on the other hand is simply to choose any action with probability $\frac{\epsilon}{K}$.

Modified Update Equation (Line 8)

The Q-value update equation for Q-learning becomes:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_{a \in \mathcal{A}} Q(S', a) - Q(S, A) \right]$$

where:

- R is the observed reward.
- $\max_{a \in \mathcal{A}} Q(S', a)$ represents the maximum estimated Q-value of the next state, assuming an optimal policy.

3. Deep Q-Learning

In Deep Q-Learning, the Q-table $Q(S, A)$ is replaced by a neural network that approximates $Q(S, A)$.

Inputs to the Neural Network

The input to the network is the state S:

- For discrete states, the input can be a one-hot encoded vector.
- For continuous states, the raw state vector (e.g., position, velocity) is used.

Outputs from the Neural Network

The network outputs a vector of size $|\mathcal{A}|$, where each element corresponds to the Q-value $Q(S, A)$ for a specific action A.

4. Challenges in Continuous Action Spaces

In continuous action spaces, the action A is not discrete but can take infinitely many values. This poses the following challenges:

1. **Argmax Operation:** In Q-learning, finding the optimal action requires solving:

   $$\max_{a \in \mathcal{A}} Q(S, a)$$

   For continuous actions, this is a non-trivial optimization problem since $\mathcal{A}$ is infinite.

2. **Network Output Size:** For discrete actions, the network outputs one Q-value per action. For continuous actions, approximating $Q(S, A)$ for all possible actions is computationally infeasible.

Common Solutions

To address these challenges, methods like *Deep Deterministic Policy Gradient (DDPG)* are often used. These approaches directly parameterize the policy to predict the optimal continuous action instead of estimating Q-values.

Task 4 - Generative Models

Answer:

1. Why the GAN Training Objective is Different

In supervised learning (e.g., classification or regression), the training objective minimizes a loss function, such as cross-entropy or mean squared error, that measures the difference between predictions and ground truth. This directly maps inputs to outputs under a well-defined metric.

In contrast, Generative Adversarial Networks (GANs) aim to generate samples from the data distribution by solving a two-player adversarial game:

- The **generator** G creates samples $G(z)$, where z is sampled from a noise distribution p_z.
- The **discriminator** D attempts to distinguish real samples $x \sim p_{\text{data}}$ from fake samples $G(z)$.

The adversarial objective is:

$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))].$$

This adversarial structure fundamentally differs from supervised learning, where objectives focus on minimizing direct prediction error.

2. GAN-Based Procedure for Super-Resolution

To address the problem of improving the resolution of low-resolution (LR) images using Generative Adversarial Networks (GANs), we can apply a **Conditional GAN (cGAN)**. This approach is suitable when we have **paired data**, meaning each low-resolution image has a corresponding high-resolution (HR) image. In this case, the goal is to generate HR images from LR inputs by conditioning the generator on the LR image.

Model Architecture

- **Generator:** The generator G takes a pair consisting of a noise vector z and a low-resolution image c as input. It generates a high-resolution image $G(z, c)$ conditioned on the low-resolution image.
- **Discriminator:** The discriminator D attempts to distinguish between real high-resolution images x_{real} and fake high-resolution images $x_{\text{fake}} = G(z, c)$, where c is the corresponding low-resolution image.

Training Procedure

In the training phase, both the generator and the discriminator are optimized through the following adversarial game:

- The **discriminator** is trained to:

  $$\max_D \mathbb{E}_{x_{\text{real}},c} \left[\log D(x_{\text{real}}, c)\right] + \mathbb{E}_{z,c} \left[\log(1 - D(G(z, c), c))\right],$$

  where x_{real} represents the real HR image, and $G(z, c)$ represents the generated HR image conditioned on c.

- The **generator** is trained to:

  $$\max_G \mathbb{E}_{z,c} \left[\log D(G(z, c), c)\right].$$

  This encourages the generator to produce images that are indistinguishable from the real HR images as evaluated by the discriminator.

Loss Functions

The loss functions for both the generator and the discriminator are designed to maximize the likelihood of real HR images and minimize the likelihood of fake HR images:
- The discriminator maximizes the conditioned likelihood of real data:

$$\mathcal{L}_{\text{D, real}} = \log D(x_{\text{real}}, c),$$

and minimizes the conditioned likelihood of fake data:

$$\mathcal{L}_{\text{D, fake}} = \log(1 - D(G(z, c), c)).$$

- The generator, on the other hand, aims to maximize the likelihood of fake data being classified as real:

$$\mathcal{L}_{\mathrm{G}} = \log D(G(z, c), c).$$

Additionally, a **content loss** can be added to ensure that the generated HR images are similar to the ground truth HR images, usually measured by the Mean Squared Error (MSE) between the generated and actual HR images:

$$\mathcal{L}_{\text{content}} = \|G(z, c) - x_{\mathrm{HR}}\|^2.$$

The final loss for the generator is a combination of the adversarial loss and the content loss:

$$\mathcal{L}_{\text{generator}} = \mathcal{L}_{\mathrm{G}} + \lambda \mathcal{L}_{\text{content}},$$

where λ is a hyperparameter that balances the importance of adversarial loss and content loss.

Inference: Super-Resolution for New Images

After training, the generator can be used to improve the resolution of a new low-resolution image c_{new} by providing a noise vector z and the low-resolution input image:

$$x_{\text{HR, generated}} = G(z, c_{\text{new}}).$$

The output is a high-resolution image that aims to closely match the ground truth high-resolution image corresponding to c_{new}.

Comparison with Other Methods

Compared to other methods, Conditional GANs provide the advantage of generating high-quality images by directly learning to map low-resolution images to high-resolution images. However, GANs are prone to training instability, and additional regularization or techniques like Wasserstein GANs may be required to stabilize training.

3. Outline of Denoising Diffusion Model (DDM)

A Denoising Diffusion Model (DDM) is an alternative to GANs that generates data through a sequential denoising process.

Forward Process (Diffusion)

- Gaussian noise is added to data over T steps, gradually transforming the data into pure noise.
- This process is modeled as:

  $$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t}\epsilon,$$

  where ϵ is Gaussian noise.

Reverse Process (Denoising)

- A neural network is trained to reverse the noise addition by predicting the noise at each step.
- The process reconstructs the data step-by-step, starting from noise.

Training Objective

The network minimizes the error between the predicted noise $\epsilon_\theta(x_t, t)$ and the actual noise ϵ:

$$\mathcal{L}_{\text{DDM}} = \mathbb{E}_{x,\epsilon,t}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right].$$

Application to Super-Resolution

To apply DDMs for super-resolution:

- Condition the reverse process on the low-resolution input image.
- Gradually denoise the input to generate the high-resolution output.

4. Comparison of GANs and DDMs

Method	**Objective**	**Key Characteristics**
GANs	Adversarial game between generator and discriminator.	High-quality outputs, but prone to instability.
Denoising Diffusion Models	Gradual noising and denoising with explicit likelihood estimation.	More stable training, but computationally intensive.

Tabell 1: Comparison of GANs and Denoising Diffusion Models.

Task 5 - Zero-Shot Learning

Answer:

Training the System

1. **Data Preparation:**
   - Use the given dataset of images for cars, buses, and vans.
   - Pair each image with its corresponding word vector $(\vec{w}_{\text{car}}, \vec{w}_{\text{bus}}, \vec{w}_{\text{van}})$.
2. **Model Design:**

- Train an image encoder (e.g., a CNN) to map images to a feature space, resulting in image features $\vec{f}_{\text{image}}$.
- Train a text encoder to process word vectors into the same feature space, resulting in text features $\vec{f}_{\text{text}}$.

3. **Training Objective:**

- Use a compatibility function (e.g., cosine similarity) to align image features ($\vec{f}_{\text{image}}$) with corresponding word vector features ($\vec{f}_{\text{text}}$).
- Minimize the distance between matching image-word pairs and maximize the distance for non-matching pairs using the following loss function:

$$\mathcal{L} = -\sum_{(I,w)} \log \frac{\exp(\text{sim}(\vec{f}_{\text{image}}, \vec{f}_{\text{text}}))}{\sum_{w'} \exp(\text{sim}(\vec{f}_{\text{image}}, \vec{f}_{\text{text'}}))}$$

Classifying a New Image (Zero-Shot)

1. **Input:**

- Given a new image of a truck, extract its feature representation using the trained image encoder:

$$\vec{f}_{\text{truck-image}} = \text{Encoder}_{\text{image}}(\text{truck-image}).$$

2. **Word Vector for Truck:**

- Use the word vector for "truck" ($\vec{w}_{\text{truck}}$) and process it through the text encoder to obtain:

$$\vec{f}_{\text{truck-text}} = \text{Encoder}_{\text{text}}(\vec{w}_{\text{truck}}).$$

3. **Prediction:**

- Compute similarity scores between the truck image features and all text features ($\vec{f}_{\text{car-text}}, \vec{f}_{\text{bus-text}}, \vec{f}_{\text{van-text}}, \vec{f}_{\text{truck-text}}$).
- Assign the label with the highest similarity score:

$$\text{Label} = \arg\max_{w} \text{sim}(\vec{f}_{\text{truck-image}}, \vec{f}_{\text{text}}).$$

- If "truck" has the highest similarity score, the image is classified as a truck.

Conclusion

This approach enables the system to recognize new classes, such as "truck," without requiring explicit training in truck images, utilizing the alignment between the features of the image and the semantic representations of the word vectors.

Task 6 - Meta Learning

Answer:

The provided support set (S) and query set (Q) in the episode are not sensible for the one-shot learning exercise. Below are the reasons:

1. Mismatch Between Classes in S and Q:

- The support set S contains labeled examples of four classes: Lion, Tiger, Bear, and Wolf.
- The query set Q contains an unlabeled example of a class (*Deer*) that is not present in S.
- In one-shot learning, the query set (Q) is meant to test the model's ability to correctly classify examples of the classes present in the support set (S). Since *Deer* is not in the support set, the model cannot learn to classify it using the provided support examples.

2. Purpose of One-Shot Learning:

- One-shot learning is designed to classify unseen examples based on a small number of labeled examples for each class in the support set.
- The query set must include examples from the same classes as those in the support set for the exercise to be valid.

3. Correction:

To make this episode sensible, the query set (Q) should include unlabeled examples of Lion, Tiger, Bear, or Wolf, as these are the classes represented in the support set.

Conclusion:

The support set and query set pair provided is not sensible because the query set contains a class (*Deer*) that is not represented in the support set. For a valid one-shot learning setup, the query set must be composed of examples from the same set of classes present in the support set.

Examples of Support Set-Query Set Pairs

1. Sensible Support Set-Query Set Pair

- **Support Set (S):**

  $S = \{$Lion image, 'Lion'; Tiger image, 'Tiger'; Bear image, 'Bear'; Wolf image, 'Wolf'$\}$

- **Query Set (Q):**

  $$Q = \{\text{Unlabeled Lion image; Unlabeled Tiger image; Unlabeled Bear image}\}$$

- **Explanation:** The query set contains examples from the same classes represented in the support set. This adheres to the intended use case of one-shot learning: classify query samples into one of the support set classes.

2. Sensible but Somewhat Non-Sensible Support Set-Query Set Pair

- **Support Set (S):**

  $$S = \{\text{Lion image, 'Lion'; Tiger image, 'Tiger'; Bear image, 'Bear'; Wolf image, 'Wolf'}\}$$

- **Query Set (Q):**

  $$Q = \{\text{Unlabeled Lion image; Unlabeled Tiger image; Unlabeled Giraffe image}\}$$

- **Explanation:** While most of the query set (Q) is sensible (Lion and Tiger), the inclusion of Giraffe makes this pair partially non-sensible. Giraffe is not represented in the support set, making it impossible for the model to classify it accurately.

Task 7 - Bayesian Deep Learning

Answer:

1. Formulas for Bayesian and Maximum Likelihood Training Criteria

Mathematical Elaboration on the Bayesian Approach

The Bayesian framework involves calculating the posterior distribution of parameters w given a dataset $D = (X, Y)$. Bayes' theorem provides this posterior as:

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)}$$

Below is a detailed explanation for both the *continuous* and *discrete* cases, particularly focusing on the marginal likelihood $p(D)$.

1. Continuous Case

For continuous parameters w, the marginal likelihood $p(D)$ is expressed as an integral:

$$p(D) = \int p(D|w)p(w)\,dw$$

where:

- $p(D|w)$: The likelihood of observing the data D given w.
- $p(w)$: The prior distribution, representing prior beliefs about w.
- $p(D)$: The marginal likelihood, also called the evidence, ensures normalization of the posterior $p(w|D)$.

Thus, the posterior $p(w|D)$ becomes:

$$p(w|D) = \frac{p(D|w)p(w)}{\int p(D|w)p(w)\,dw}$$

Prediction: When making predictions for a new input x^*, the predictive distribution is obtained by marginalizing over w:

$$p(y^*|x^*, D) = \int p(y^*|x^*, w)p(w|D)\,dw$$

2. Discrete Case

For discrete parameters w, the marginal likelihood $p(D)$ is expressed as a summation:

$$p(D) = \sum_w p(D|w)p(w)$$

where:

- $p(w)$ is a discrete probability mass function (PMF) for w.
- $p(D|w)$ gives the probability of the data D for each parameter w.

The posterior $p(w|D)$ then becomes:

$$p(w|D) = \frac{p(D|w)p(w)}{\sum_w p(D|w)p(w)}$$

Prediction: Similarly, the predictive distribution is:

$$p(y^*|x^*, D) = \sum_w p(y^*|x^*, w)p(w|D)$$

Maximum Likelihood Training Criterion: The model parameters w are estimated by maximizing the likelihood of the observed data:

$$w^* = \arg\max_w p(D|w) = \arg\max_w \prod_{i=1}^{N} p(y_i|x_i, w)$$

This does not involve a prior over w.

2. Difference Between Bayesian and Maximum Likelihood Approaches

Bayesian Approach:

- Incorporates prior knowledge through $p(w)$.
- Provides a distribution over w rather than point estimates.
- Accounts for uncertainty by integrating over $p(w|D)$.

Maximum Likelihood Approach:

- Focuses on finding the single most likely set of parameters w^*.
- Does not incorporate prior knowledge or uncertainty in w.
- Simpler but prone to overfitting in limited data scenarios.

3. Why Approximate Methods Are Used in Bayesian Learning

Exact Bayesian Learning Challenges:

- Computing the posterior $p(w|D)$ often involves intractable integrals due to the marginal likelihood $p(D)$.
- Prediction requires marginalizing over w, which involves high-dimensional integrals.

Approximation Techniques:

- Methods like Variational Inference or Markov Chain Monte Carlo (MCMC) approximate the posterior.
- These methods trade computational feasibility for exactness while retaining uncertainty estimation benefits.

Task 8 - Bayes by Backprop

Answer:

We are given the Evidence Lower Bound (ELBO) and tasked with calculating $\ln p(w^s, D)$ and explaining why we cannot calculate gradients of $\mathcal{L}(\lambda)$ directly.

1. Calculating $\ln p(w^s, D)$

In the expression

$$\mathcal{L}(\lambda) = \frac{1}{S}\sum_{s=1}^{S}\left[\ln p(w^s, D) - \ln q(w^s, \lambda)\right],$$

we need to compute the term $\ln p(w^s, D)$, where:

$$p(w^s, D) = p(D|w^s)p_0(w^s)$$

is the joint likelihood, assuming a prior distribution $p_0(w)$ for the model parameters w and a likelihood function $p(D|w)$ for the data D. Therefore, $\ln p(w^s, D)$ is computed as:

$$\ln p(w^s, D) = \ln p(D|w^s) + \ln p_0(w^s)$$

The first term $\ln p(D|w^s)$ is the log-likelihood of the data given the model parameters w^s, and the second term $\ln p_0(w^s)$ is the log-prior of the model parameters w^s.

2. Why Can't We Calculate Gradients Directly?

The issue with directly calculating the gradients of $\mathcal{L}(\lambda)$ with respect to λ lies in the presence of the sampling operation. In the ELBO, we are sampling w^s from the variational distribution $q(w^s, \lambda)$ (parameterized by λ) and calculating the objective based on these samples. The gradients of $\mathcal{L}(\lambda)$ would require computing how the log-likelihood term $\ln p(w^s, D)$ changes with respect to λ, but because the samples w^s are drawn from a distribution, we cannot differentiate through this sampling process directly.

This leads to a problem because the sampling is non-differentiable. Thus, we cannot calculate the gradients of the ELBO with respect to λ as it is written.

3. Method to Resolve the Problem

REINFORCE

One solution to this issue is to use the **REINFORCE** algorithm, a policy gradient method, which allows for differentiating through samples. The REINFORCE estimator provides an unbiased estimate of the gradient of the ELBO by introducing a **score function** that allows for the gradient computation in the presence of sampling.

The idea is to express the gradient of the ELBO with respect to λ as:

$$\nabla_\lambda \mathcal{L}(\lambda) \approx \frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda \ln q(w^s, \lambda)\left[\ln p(w^s, D) - \ln q(w^s, \lambda)\right]$$

This formulation enables the gradient estimation by applying the **likelihood ratio trick** in the REINFORCE method, which allows us to compute the gradients with respect to the parameters λ even in the presence of sampling.

Re-parameterization Trick

To mitigate this issue, the re-parameterization trick can be applied, assuming that the variational distribution $q(w^s|\lambda)$ is continuous, such as a Gaussian distribution with mean $\mu(\lambda)$ and covariance $\Sigma(\lambda)$. We can express w^s as:

$$w^s = \mu(\lambda) + \Sigma(\lambda)^{1/2}\epsilon^s,$$

where $\epsilon^s \sim \mathcal{N}(0, I)$ is a standard normal noise variable independent of λ.

Using this re-parameterization, the objective function becomes:

$$\mathcal{L}(\lambda) = \frac{1}{S}\sum_{s=1}^{S}\left[\ln p(\mu(\lambda) + \Sigma(\lambda)^{1/2}\epsilon^s, D) - \ln q(\mu(\lambda) + \Sigma(\lambda)^{1/2}\epsilon^s, \lambda)\right].$$

Now, since ϵ^s is independent of λ, we can compute the gradient of the ELBO with respect to λ:

$$\nabla_\lambda \mathcal{L}(\lambda) = \frac{1}{S}\sum_{s=1}^{S}\nabla_\lambda\left[\ln p(\mu(\lambda) + \Sigma(\lambda)^{1/2}\epsilon^s, D) - \ln q(\mu(\lambda) + \Sigma(\lambda)^{1/2}\epsilon^s, \lambda)\right].$$

This reformulation allows us to compute the gradient of the ELBO with respect to λ and use gradient-based optimization techniques, such as stochastic gradient descent, for training the variational parameters.

Summary

- $\ln p(w^s, D)$ is computed as $\ln p(w^s) + \ln p(D|w^s)$, using the prior $p(w^s)$ and the likelihood $p(D|w^s)$.
- We cannot calculate the gradients directly due to the non-differentiable sampling operation.
- The **REINFORCE** method can be used to resolve this by estimating the gradient through the likelihood ratio trick.
- The re-parameterization trick transforms the non-differentiable sampling operation into a differentiable one by expressing the latent variable w^s as a deterministic function of λ and independent noise. This allows for the application of backpropagation, enabling efficient gradient computation and optimization of the variational parameters λ.

Note:

Intuitively, the **re-parameterization trick** provides more informative gradients by exposing the dependence of the sampled latent variables w^s on the variational parameters λ. Specifically, w^s is expressed as a deterministic function of λ, such as $w^s = \mu(\lambda) + \Sigma(\lambda)^{1/2}\epsilon^s$, where $\epsilon^s \sim \mathcal{N}(0, I)$ is a noise term independent of λ. This explicit dependency enables more efficient and stable gradient estimation during optimization.

In contrast, the **REINFORCE gradient estimator** only depends on the relationship between the density function $\log q(w^s|\lambda)$ and its parameters λ. It computes gradients based on the observed likelihood of the sample w^s, without making the explicit dependence of w^s on λ transparent. This results in noisier gradient estimates, as the REINFORCE estimator does not directly account for the continuous relationship between λ and w^s.

Task 9 - Inverse Reinforcement Learning

Answer:

The objective function for Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) is defined as:

$$\sum_{i=1}^{M} \log p\left(\tau_i\right)$$

where the probability of each trajectory $p(\tau_i)$ is given by:

$$p\left(\tau_i\right) = \frac{1}{Z} \exp\Big(R_\psi\left(\tau_i\right)\Big)$$

and Z is the partition function:

$$Z = \sum_{\tau \in \mathcal{D}_{\text{all}}} \exp\Big(R_\psi(\tau)\Big)$$

Here, $R_\psi(\tau_i)$ is the total reward of a trajectory τ_i, computed by a neural network with parameters $\boldsymbol{\psi}$.

Why Exact Calculation is Not Suitable

The calculation of the objective function $\sum_{i=1}^{M} \log p(\tau_i)$ involves computing the partition function Z, which requires summing over all possible trajectories $\mathcal{D}_{\text{all}}$. This becomes intractable in real-world problems because:

- **Exponential Growth in the Number of Trajectories:** The number of possible trajectories grows exponentially with the number of states and actions, making it impossible to compute the partition function exactly.
- **High Computational Cost:** The sum over all trajectories, $\sum_{\tau \in \mathcal{D}_{\text{all}}}$, involves a combinatorially large number of terms, which is computationally expensive and impractical for large state-action spaces.

Therefore, exact computation of the objective function is unsuitable for many real-world problems due to the high complexity involved in summing over all possible trajectories.

Guided Cost Learning (GCL) and Sampling Approach

Guided Cost Learning (GCL) proposes a sampling-based approach to address the issue of calculating the partition function. In GCL, instead of summing over all possible trajectories, the trajectories are sampled from a **probability distribution** that is guided by the current reward function. Specifically, the probability distribution through which the trajectories are sampled is:

$$p(\tau) = \frac{\exp(R_\psi(\tau))}{Z}$$

This sampling approach avoids the need to explicitly compute Z, as it samples trajectories in a way that approximates the distribution defined by the reward function. This allows GCL to efficiently estimate the reward function by focusing on trajectories that are likely under the current model, rather than considering all possible trajectories.

In Summary

- The exact calculation of the objective in MaxEnt IRL is impractical because it involves computing the partition function Z, which requires summing over all possible trajectories. This is computationally expensive and infeasible for large-scale problems.
- Guided Cost Learning (GCL) mitigates this problem by using a **sampling-based approach**, where trajectories are sampled according to the probability distribution $p(\tau) = \frac{\exp(R_\psi(\tau))}{Z}$, thus bypassing the need to compute Z directly and making the process more scalable.

Task 10 - Behaviour Cloning & Reinforcement Learning

Answer:

Advantages of Reinforcement Learning over Behavior Cloning:

- **Generalization to Unseen States:** Behavior cloning directly mimics expert actions from a dataset, limiting its ability to generalize to states not present in the training data. In contrast, reinforcement learning (RL) learns through exploration, enabling it to discover and handle new or unseen states effectively.
- **No Dependency on Expert Demonstrations:** Behavior cloning requires high-quality, annotated expert demonstrations to train. RL eliminates this dependency by learning optimal actions through interaction with the environment, making it suitable for scenarios where expert data is unavailable or costly to obtain.

Negative Consequences of RL's Trial and Error Approach:

- **Inefficiency in Learning:** RL often requires a large number of interactions with the environment to discover optimal actions, which can be computationally expensive and time-consuming.

- **Risk of Negative Outcomes:** The trial-and-error process may lead to suboptimal or harmful behaviors in environments, especially during early stages of learning where the agent is exploring.

Mitigation Strategies:

- **Reward Shaping:** By designing appropriate rewards, we can guide the agent toward beneficial behaviors and reduce the exploration of harmful actions.
- **Imitation Learning:** Incorporating expert demonstrations into the training process (e.g., behavior cloning) can accelerate learning, allowing the agent to start from a better initial policy.

Task 11 - PointNet

Answer:

Permutation invariance in **PointNet** is achieved through the use of a *symmetric function*, specifically a max-pooling operation. The process is as follows:

1. **Input Representation:** A point cloud is represented as an unordered set of 3D points, $\{x_1, x_2, \ldots, x_N\}$, where each point x_i may have features such as coordinates (x, y, z) or additional attributes.
2. **Feature Extraction:** PointNet processes each point independently using shared multi-layer perceptrons (MLPs) to extract local features, resulting in a set of feature vectors:
$$\{\phi(x_1), \phi(x_2), \ldots, \phi(x_N)\},$$
where $\phi(x_i)$ denotes the learned feature of point x_i.
3. **Symmetric Aggregation:** To ensure the network is invariant to the order of the points, PointNet applies a symmetric aggregation function $f(\cdot)$, typically max-pooling:
$$g = \text{MAX}(\phi(x_1), \phi(x_2), \ldots, \phi(x_N)),$$
where g is a global feature vector summarizing the entire point cloud.
4. **Permutation Invariance:** The max-pooling operation guarantees that the output g remains unchanged regardless of the order of the input points, as it only considers the maximum value across dimensions.

Task 12 - Edge Function

Answer

Edge Function and Aggregation in PointNet

1. **Edge Function ($\boldsymbol{h}_{\boldsymbol{\Theta}}(\cdot)$):** In PointNet, the edge function $\boldsymbol{h}_{\boldsymbol{\Theta}}$ is simplified to operate only on the features of the concerned node itself. It does not explicitly consider the features of neighboring nodes. For a node i:

$$\boldsymbol{h}_{\boldsymbol{\Theta}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi_{\boldsymbol{\Theta}}(\boldsymbol{x}_i),$$

where $\phi_{\boldsymbol{\Theta}}$ is a shared multi-layer perceptron (MLP) applied to each point individually. Here, $\boldsymbol{x}_j$ is ignored.

2. **Aggregation Operation ($\square$):** PointNet uses **max-pooling** as its symmetric aggregation function:

$$\square_{j \in \mathcal{E}_i} \boldsymbol{h}_{\boldsymbol{\Theta}} \left(\boldsymbol{x}_i, \boldsymbol{x}_j \right) = \text{MAX} \left\{ \phi_{\boldsymbol{\Theta}}(\boldsymbol{x}_k) \mid k \in \mathcal{E}_i \cup \{i\} \right\}.$$

This max-pooling operation ensures permutation invariance, as it outputs the same result regardless of the order of points.

Disadvantage of the Edge Function in PointNet

The edge function in PointNet is **limited** because it does not explicitly encode relationships between neighboring points. Specifically:

- It ignores pairwise interactions between points ($\boldsymbol{x}_i$ and $\boldsymbol{x}_j$), leading to a loss of local geometric context.
- This can reduce performance for tasks requiring detailed spatial relationships (e.g., understanding edges or surfaces in point clouds).

This limitation was addressed in follow-up architectures like PointNet++ or graph neural networks (GNNs), which include local neighborhood information to improve feature representation.

Task 13 - Multi-Object Tracking

Answer:

Inputs of G_t

The graph neural network G_t in multi-object tracking takes the following inputs:

- **Node Features ($\boldsymbol{x}_i$)**: Each node in the graph represents a detection, and its feature vector may include:
  - Bounding box coordinates or center positions of the detection.
  - Appearance features extracted from an object detector or a convolutional neural network (CNN).

  - Motion features such as velocity or predicted trajectory (if available).

- **Edge Features ($\boldsymbol{e}_{ij}$)**: Edges represent possible associations between detections. The feature vector for an edge may include:
  - Spatial relationships (e.g., distances or overlap between bounding boxes).
  - Temporal relationships (e.g., predicted positions of objects from the previous frame).
  - Appearance similarity metrics (e.g., cosine similarity of appearance embeddings).

Outputs of G_t

The graph neural network G_t outputs:

- **Updated Node Features ($\boldsymbol{x}_i'$)**: After passing through the network, the node features are updated to include context from neighboring nodes. This reflects the object's identity or refined representation for association.
- **Edge Weights or Scores (s_{ij})**: The edge weights represent the likelihood of association between two detections. These scores are used to:
  - Establish associations between objects across frames (e.g., using a bipartite matching algorithm like the Hungarian algorithm).
  - Filter unlikely associations based on thresholded scores.
- **Edge Classification (y_{ij})**: For each edge, G_t outputs a binary variable y_{ij} where:
  - $y_{ij} = 1$: The edge is retained, indicating a valid association between two detections.
  - $y_{ij} = 0$: The edge is discarded, rejecting the association.

  This binary classification determines whether two detections are connected in the tracking graph.
- **Updated Node Features ($\boldsymbol{x}_i'$)**: Optionally, the node features may be updated to include contextual information from neighboring nodes for subsequent processing.

Task 14 - Single Object Tracking

Answer:

Forward Passes in MDNet

In MDNet, a sliding window of size 127×127 slides across the search image of size 255×255 to generate a 17×17 score map. Since the output grid is 17×17, the number of forward passes required is:

$$\text{Number of forward passes} = 17 \times 17 = 289$$

Thus, **289 forward passes** would be needed to generate a score map of size 17×17.

END

Exercise on Forward Passes in MDNet

A Siamese neural network is used for single object tracking, where the target image is of size $127 \times 127 \times 3$ and the search image is of size $512 \times 512 \times 3$. The network outputs a score map of size 32×32 in a single forward pass.

If the network is implemented using a sliding window approach where each forward pass processes a 127×127 window from the search image, how many forward passes are needed to process the entire 512×512 search image and generate the 32×32 score map?

Solution

1. **Calculate the number of windows:** The sliding window moves across the image with a stride of 127 pixels. Thus, the number of windows in each dimension is:

$$\text{Number of windows in width} = \frac{512 - 127}{127} + 1 = 4$$

$$\text{Number of windows in height} = \frac{512 - 127}{127} + 1 = 4$$

2. **Total forward passes:** The total number of forward passes is the product of the number of windows in width and height:

$$\text{Total forward passes} = 4 \times 4 = 16$$

Thus, **16 forward passes** are needed to process the entire 512×512 search image and generate the corresponding 32×32 score map.

Exam 2022

Task 1 - Field Of View

Answer:

$k = 3$ gives a field of view (FOV) 7 which is less than the required FOV 8. $k = 4$ gives a FOV 8 which is exactly the required FOV. Therefore, the minimum $k = 4$.

Task 2 - Self-Attention

Answer:

Sub-question 1: Output Sequence Corresponding to New Input Order

The output sequence corresponding to the new input order (x_2, x_3, x_1) is:

$$(z_2, z_3, z_1).$$

Explanation: In a self-attention layer, the output for each input token is computed as:

$$z_i = \sum_{j=1}^{n} \alpha_{ij} x_j, \quad \text{where } \alpha_{ij} = \text{softmax}\left(\frac{q_i^\top k_j}{\sqrt{d_k}}\right),$$

where:

- q_i, k_j are the query and key vectors for tokens i and j,
- d_k is the dimensionality of the key vectors,
- α_{ij} is the attention weight between tokens i and j.

Since self-attention computes outputs based on relative token interactions, reordering the input sequence results in a corresponding reordering of the output sequence. Therefore, if the input sequence changes to (x_2, x_3, x_1), the output sequence becomes (z_2, z_3, z_1).

Sub-question 2: Remedy for High Computational Complexity

One major disadvantage of self-attention is its high computational complexity, which scales quadratically with the sequence length n due to the computation of the attention matrix:

$$\mathcal{O}(n^2 \cdot d),$$

where d is the embedding dimension.

To address this, **sparse attention mechanisms** can be applied, which restrict the computation of attention weights to a subset of tokens, reducing the complexity to:

$$\mathcal{O}(k \cdot n \cdot d),$$

where $k \ll n$ is the number of tokens attended by each token.

To address the quadratic computational complexity of self-attention, we can consider only a local neighborhood of each input element, rather than the entire sequence, when calculating the self-attention. This reduces the number of computations and hence the complexity.

Disadvantage of the Remedy: This approach reduces the field of view of each self-attention layer. To cover the entire sequence, several self-attention layers must be stacked. Consequently, the maximum path length (i.e., the number of operations between an output element and an input element) depends on the length of the input sequence. This could lead to inefficiencies in capturing long-range dependencies.

Task 3 - Policy Gradients

Answer:

Why optimizing $R(\tau)$ directly is unlikely to be successful

Optimizing $R(\tau)$ with respect to θ by back-propagating through the policy network is infeasible because:

1. **Non-differentiability**: The sampling operations involved in selecting actions (a_t) and state transitions (s_{t+1}) are not differentiable. This prevents gradients of $R(\tau)$ from being propagated through the stochastic components of the environment.
2. **Sparse rewards**: $R(\tau)$ is typically dependent on rewards received only at the end of the trajectory or sparsely during it, leading to high variance in gradients.

Effective Loss to Backpropagate

The effective loss to backpropagate through the policy network is:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{T} \log \pi_\theta \left(a_t \mid s_t\right) R(\tau).$$

This is derived from the policy gradient formulation:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta \left(a_t \mid s_t\right) R(\tau) \right],$$

where the log-probability of actions is scaled by the total reward $R(\tau)$.

Interpretation of the Gradients

The gradients of the effective loss, $-\log \pi_\theta(a_t \mid s_t) R(\tau)$, resemble those in the cross-entropy loss used in supervised learning, where the log-probabilities of the correct class are maximized.

- Here, $R(\tau)$ serves as the weight or importance of the sampled trajectory, scaling the contribution of each action to the gradient.
- Intuitively, this encourages the policy to assign higher probabilities to actions that lead to higher rewards, similar to how supervised learning increases probabilities for correct predictions.

Task 4 - Generative Adverserial Networks

Answer:

1. Approach for Training a GAN

To train a GAN for generating an image corresponding to a given description word, we can use a **Conditional GAN (cGAN)** framework. The key components and steps are as follows:

- **Generator (G):** Takes as input a noise vector z and the word vector w (representing the description). It generates an image $\hat{x}$:

$$G(z, w) \to \hat{x}.$$

- **Discriminator (D):** Takes as input either a real image x and its associated word vector w, or a generated image $\hat{x}$ with the same word vector w. It predicts whether the image is real or fake:

$$D(x, w) \to [0, 1].$$

- **Loss Functions:**
  - **Generator Loss:** Encourages G to produce realistic images paired with the given word vector w:

$$\mathcal{L}_G = -\mathbb{E}_{z \sim p_z(z), w \sim p_w(w)} \left[\log D(G(z, w), w) \right].$$

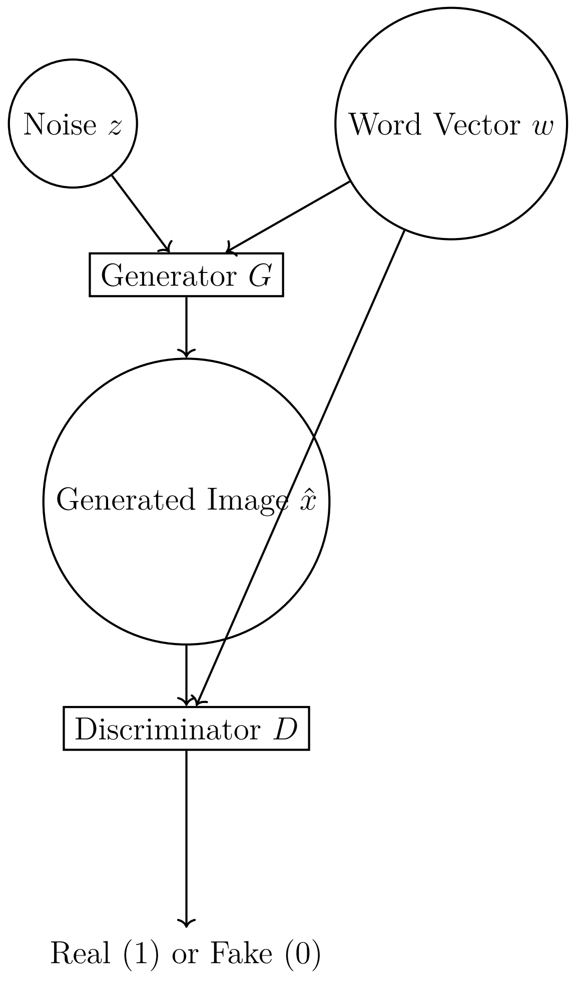
  - **Discriminator Loss:** Helps D distinguish between real and fake image-word pairs:

$$\mathcal{L}_D = -\mathbb{E}_{x \sim p_{data}(x), w \sim p_w(w)} \left[\log D(x, w) \right] - \mathbb{E}_{z \sim p_z(z), w \sim p_w(w)} \left[\log(1 - D(G(z, w), w)) \right].$$

- **Training:** Alternate between updating D and G using their respective loss functions. Both G and D are conditioned on the word vector w.

Diagram of cGAN Framework

The diagram below illustrates the conditional GAN framework:



2. Generating Diverse Images

To generate a diverse set of images for a given description word w:

- Sample multiple noise vectors z from the noise distribution $p_z(z)$ while keeping w fixed:
$$\hat{x}_i = G(z_i, w), \quad z_i \sim p_z(z), \quad i = 1, 2, \ldots, N.$$
- The stochastic nature of z allows the generation of diverse outputs.

3. Advantage of Diffusion Models (DDM) over GANs

A key advantage of Denoising Diffusion Models (DDMs) over GANs in this task:

- **Mode Coverage:** DDMs are better at covering the entire data distribution and generating diverse samples without collapsing into limited modes.

- **GAN Limitation:** GANs are prone to mode collapse, where they might generate only a subset of possible outputs for a given word vector w.
- **Reason for DDM's Advantage:** DDMs explicitly model the data distribution by iteratively reversing a noise process, which encourages better diversity and quality.

Task 5a - Meta Learning

Answer: Answer:

What is k-class-n-shot learning?

k-class-n-shot learning is a type of few-shot learning where the goal is to classify data into k classes, with the constraint that only n labeled examples (shots) are available per class during training.

Example of a 5-class, 1-shot learning problem

In a 5-class, 1-shot learning problem:

- There are 5 classes, each with only 1 labeled example in the support set.
- The task is to classify unlabeled examples (query set) into one of these 5 classes.

Example: Consider a dataset of animals with the following classes:

(1) Dog

(2) Cat

(3) Elephant

(4) Tiger

(5) Zebra

During training and testing, for each task:

- The **support set** contains **one image per class** (e.g., one image of a dog, one of a cat, etc.).
- The **query set** contains unlabeled images from the same 5 classes, which need to be classified.

Description of Support and Query Sets

Meta-Training Phase

- **Support Set (S):** Contains 5 labeled examples, one for each of the 5 classes. For example, $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_5, y_5)\}$, where y_i represents the class label (Dog, Cat, etc.).
- **Query Set (Q):** Contains several unlabeled images sampled from the same 5 classes.

Meta-Testing Phase

- **Support Set (S):** Again contains 5 labeled examples (1 per class) from previously unseen classes. For example, these classes might be Whale, Shark, Dolphin, Penguin, and Seal.
- **Query Set (Q):** Contains unlabeled images from these new classes, requiring the model to classify based on the support set.

Key Point

This formulation is common in *meta-learning* methods like Prototypical Networks, where the model learns a distance metric to classify query images relative to the support set.

Task 5b - Active Learning

Answer:

In active learning, the most "valuable" samples refer to those that are expected to provide the most information to improve the model. These samples are selected based on uncertainty, diversity, or representativeness, as they are expected to maximize the model's performance gain with minimal labeled data.

Common Approaches to Predict the Most Valuable Samples

There are several common strategies used to predict valuable samples:

- **Uncertainty Sampling:** The model selects samples where it is most uncertain. This is typically done by measuring the model's prediction confidence. Common uncertainty measures include:
  - **Least Confidence:** Select the sample for which the model has the lowest prediction confidence.
  - **Margin Sampling:** Select the sample whose top two predicted classes have the smallest margin between their probabilities.
  - **Entropy Sampling:** Select the sample with the highest entropy, which indicates the model's uncertainty in predicting the correct class.

- **Query by Committee (QBC):** A committee of models is trained on the labeled data, and the sample with the highest disagreement among the models is considered valuable. The rationale is that a sample where the models disagree the most is likely to improve the model's understanding when labeled.
- **Diversity Sampling:** This approach aims to select samples that represent the underlying data distribution. By choosing samples that are diverse and spread across different regions of the input space, the model can learn from a wide variety of situations.
- **Expected Model Change:** This strategy selects the samples that are expected to cause the greatest change in the current model. Samples that can lead to significant changes in the model's parameters are prioritized for labeling.

The overall goal is to efficiently select samples that will improve the model's performance with the fewest labels.

Task 6a - Bayesian Deep Learning

Answer:

How can we use the variance in safety-critical applications?

In safety-critical applications, such as autonomous driving or medical diagnosis, understanding the uncertainty in predictions is crucial. The variance of the predicted output $\hat{y}$, given an input $\hat{x}$, can be used to quantify the confidence of the model's prediction.

- **High Variance:** If the variance is high, it indicates that the model is uncertain about its prediction. In such cases, the system can take precautionary measures, such as slowing down or requesting additional data, to avoid taking risky actions based on an uncertain prediction.
- **Low Variance:** If the variance is low, it means the model is confident in its prediction. This can help to prioritize decisions and actions based on the model's confidence.

By evaluating the variance, we can decide when to trust the model's prediction and when to request further data or human intervention, which is particularly important in domains where errors can have severe consequences.

Why is it hard to evaluate the integral in practical deep learning?

The integral that needs to be evaluated is of the form:

$$\boldsymbol{I} = \int \boldsymbol{F}(\boldsymbol{w})\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})\, d\boldsymbol{w}$$

where $\boldsymbol{F}(\boldsymbol{w})$ is a function of the weights $\boldsymbol{w}$, and $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ is the posterior distribution of the weights given the data $\mathcal{D}$.

In practical deep learning, evaluating this integral is challenging because:

- **Complexity of the Posterior Distribution:** The posterior distribution $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ is typically highly complex and high-dimensional, especially for deep neural networks. Exact analytical solutions are usually not available, and approximations are needed.
- **Intractability of the Integral:** In most cases, the posterior $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ does not have a closed-form expression, making it difficult to compute the integral directly. This requires the use of approximate methods such as Markov Chain Monte Carlo (MCMC) or Variational Inference, which can be computationally expensive.
- **Monte Carlo Approximation:** Since the exact integral is intractable, methods like Monte Carlo sampling are often used. This requires drawing samples from the posterior distribution $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ and estimating the integral as an average over these samples. However, in high-dimensional settings, obtaining a sufficient number of samples can be computationally expensive.

Thus, the main difficulty lies in the high-dimensional and complex nature of the posterior distribution $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$, which makes evaluating the integral computationally prohibitive in large-scale deep learning models.

Task 6b - Variational Inference

Answer:

Variational inference (VI) is a method used to approximate complex integrals, such as the one described in Question 6(a). In VI, we approximate the true posterior distribution $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ with a simpler, tractable distribution $q(\boldsymbol{w})$, and then minimize the difference between the true posterior and the approximating distribution.

Mathematical Formulation

The integral we need to estimate is:

$$\boldsymbol{I} = \int \boldsymbol{F}(\boldsymbol{w})\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})\, d\boldsymbol{w}$$

where $\boldsymbol{F}(\boldsymbol{w})$ is a function of the weights $\boldsymbol{w}$ and $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ is the true posterior distribution.

To apply variational inference, we introduce a variational distribution $q(\boldsymbol{w})$ and aim to approximate the true posterior with this distribution. The main idea in VI is to minimize the Kullback-Leibler (KL) divergence between the true posterior $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ and the variational distribution $q(\boldsymbol{w})$.

Steps in Variational Inference

1. **Define the Evidence Lower Bound (ELBO):** The goal of variational inference is to maximize the Evidence Lower Bound (ELBO), which is derived from the log marginal likelihood of the observed data:

   $$\log \mathcal{P}(\mathcal{D}) = \mathbb{E}_{q(\boldsymbol{w})}[\log \frac{\boldsymbol{p}(\mathcal{D}, \boldsymbol{w})}{q(\boldsymbol{w})}] = \mathbb{E}_{q(\boldsymbol{w})}[\log \boldsymbol{p}(\mathcal{D}, \boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w})}[\log q(\boldsymbol{w})]$$

   where $\boldsymbol{p}(\mathcal{D}, \boldsymbol{w}) = \boldsymbol{p}(\mathcal{D} \mid \boldsymbol{w})\boldsymbol{p}(\boldsymbol{w})$ is the joint distribution of data and weights.

   The ELBO is the lower bound on the log marginal likelihood and can be written as:

   $$\text{ELBO} = \mathbb{E}_{q(\boldsymbol{w})}[\log \boldsymbol{p}(\mathcal{D} \mid \boldsymbol{w})] - \mathbb{E}_{q(\boldsymbol{w})}[\log q(\boldsymbol{w})]$$

2. **Approximate the Posterior:** To estimate the integral, we replace the true posterior distribution $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ with a variational distribution $q(\boldsymbol{w})$, which is parameterized by variational parameters. A common choice for $q(\boldsymbol{w})$ is a Gaussian distribution, but in deep learning, more flexible distributions are used.

3. **Optimization:** We then maximize the ELBO with respect to the variational parameters of $q(\boldsymbol{w})$, effectively minimizing the KL divergence between the true posterior and the variational distribution. The optimization can be done using gradient-based methods, which are feasible in deep learning settings.

4. **Monte Carlo Estimation:** After training the variational distribution $q(\boldsymbol{w})$, the integral $\boldsymbol{I}$ can be estimated by sampling from $q(\boldsymbol{w})$ and computing the expectation:

   $$\boldsymbol{I} \approx \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{F}(\boldsymbol{w}^{(i)}) \quad \text{where} \quad \boldsymbol{w}^{(i)} \sim q(\boldsymbol{w})$$

   The Monte Carlo method allows us to approximate the integral by averaging over the samples drawn from $q(\boldsymbol{w})$.

Conclusion

By using variational inference, we approximate the posterior $\boldsymbol{p}(\boldsymbol{w} \mid \mathcal{D})$ with a simpler distribution $q(\boldsymbol{w})$, and then use Monte Carlo sampling to estimate the integral $\boldsymbol{I}$. This allows us to calculate quantities such as the variance of the prediction in a computationally feasible way, which would otherwise be intractable for complex deep learning models.

Task 7a - Inverse Reinforcement Learning

Answer

Maximum Entropy Inverse Reinforcement Learning

Objective Function

The goal of Maximum Entropy Inverse Reinforcement Learning (MaxEnt IRL) is to find a reward function $\boldsymbol{r}_\psi(\cdot,\cdot)$, parameterized by $\boldsymbol{\psi}$, that explains the demonstrated trajectories $\mathcal{D} = \{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2, \ldots, \boldsymbol{\tau}_M\}$ under the principle of maximum entropy. This principle ensures that the learned reward function does not make unnecessary assumptions about the expert's behavior.

The probability of a trajectory $\boldsymbol{\tau}_i$ under MaxEnt IRL is given by:

$$\boldsymbol{P}_\psi(\boldsymbol{\tau}_i) = \frac{1}{Z_\psi} \exp\Big(\boldsymbol{R}_\psi(\boldsymbol{\tau}_i)\Big),$$

where:

- $\boldsymbol{R}_\psi(\boldsymbol{\tau}_i) = \sum_{(s,a)\in\boldsymbol{\tau}_i} \boldsymbol{r}_\psi(s,a)$ is the total reward of trajectory $\boldsymbol{\tau}_i$,
- $Z_\psi = \sum_{\boldsymbol{\tau}} \exp\Big(\boldsymbol{R}_\psi(\boldsymbol{\tau})\Big)$ is the partition function, summing over all possible trajectories $\boldsymbol{\tau}$ to normalize the distribution.

The objective function of MaxEnt IRL is to maximize the likelihood of the expert demonstrations $\mathcal{D}$ under the learned distribution:

$$\mathcal{L}(\boldsymbol{\psi}) = \sum_{\boldsymbol{\tau}_i\in\mathcal{D}} \log \boldsymbol{P}_\psi(\boldsymbol{\tau}_i).$$

Substituting $\boldsymbol{P}_\psi(\boldsymbol{\tau}_i)$, we get:

$$\mathcal{L}(\boldsymbol{\psi}) = \sum_{\boldsymbol{\tau}_i\in\mathcal{D}} \boldsymbol{R}_\psi(\boldsymbol{\tau}_i) - \log Z_\psi.$$

Gradient of the Objective

To optimize the reward parameters $\boldsymbol{\psi}$, the gradient of the objective is computed as:

$$\nabla_\psi \mathcal{L}(\boldsymbol{\psi}) = \mathbb{E}_{\boldsymbol{\tau}\sim\boldsymbol{P}_\psi}\Big[\nabla_\psi \boldsymbol{R}_\psi(\boldsymbol{\tau})\Big] - \sum_{\boldsymbol{\tau}_i\in\mathcal{D}} \nabla_\psi \boldsymbol{R}_\psi(\boldsymbol{\tau}_i),$$

where:

- The first term represents the expected reward gradient under the model's trajectory distribution,
- The second term represents the reward gradient over the expert's demonstrated trajectories.

Key Insight

MaxEnt IRL uses the maximum entropy principle to resolve ambiguity in the reward function by favoring the distribution that maximizes entropy while matching the observed expert behavior. This avoids overly confident reward estimates when multiple reward functions explain the data equally well.

Explanation of the Components

- **Trajectory Probability**: Trajectories are assigned probabilities based on their total reward using a softmax function. This ensures that high-reward trajectories are more likely but doesn't discard lower-reward ones completely.
- **Likelihood Objective**: The objective function maximizes the log-likelihood of the expert demonstrations, ensuring that the learned model aligns closely with the observed behavior.
- **Partition Function**: The partition function Z_ψ normalizes the trajectory probabilities, making them valid probabilities. However, it involves summing over all possible trajectories, which is computationally expensive and typically approximated during optimization.
- **Gradient Formulation**: The gradient consists of two terms: one aligns the model's behavior with the expert data, and the other ensures consistency across the distribution of trajectories. This makes the optimization robust.
- **Maximum Entropy Principle**: By maximizing entropy, the model avoids overfitting to specific expert demonstrations, maintaining generality in the reward function.

Task 7b - Behaviour Cloning

Answer:

Weaknesses of Behavior Cloning

Behavior cloning (BC) involves learning a policy by imitating expert demonstrations. While effective in some scenarios, it has the following weaknesses:

1. **Distribution Shift:** Behavior cloning assumes that the data distribution during training matches that during deployment. However, in real-world control tasks, the agent's actions can lead to states not present in the training set. This results in compounding errors as the agent deviates from the expert trajectory.
2. **Lack of Exploration:** BC relies solely on expert demonstrations and does not explore the environment. Consequently, it cannot recover from states outside the demonstrated trajectories or optimize performance beyond the expert's capabilities.

Reinforcement Learning as an Approach to Control Learning

Reinforcement learning (RL) involves learning an optimal policy by interacting with the environment and maximizing cumulative rewards. A student's claim that RL is the best approach has both advantages and disadvantages:

Arguments in Favor of Reinforcement Learning

1. **Exploration and Generalization:** RL actively explores the environment, allowing the agent to discover new and optimal strategies beyond those demonstrated by experts. This capability is essential for tasks where expert data is unavailable or suboptimal.
2. **Adaptation to New Environments:** RL agents can adapt to dynamic or previously unseen environments by updating their policy through further interaction, making RL more versatile than behavior cloning in such scenarios.

Arguments Against Reinforcement Learning

1. **Sample Inefficiency:** RL often requires a large number of interactions with the environment to converge to an optimal policy, especially in complex tasks. This can be computationally expensive and impractical for real-world applications with physical constraints.
2. **Reward Engineering:** Designing a suitable reward function is challenging, especially in tasks with sparse or ambiguous rewards. Poorly designed rewards can lead to suboptimal or undesired behaviors, making RL less effective.

Key Insight

While RL is powerful due to its ability to learn directly from interactions and optimize beyond expert performance, it is not universally superior. Combining RL with approaches like imitation learning or inverse reinforcement learning can often mitigate its weaknesses and lead to more efficient and robust solutions.

Task 8a - 3D Processing (PointCloud)

Answer:

The likely dimension of the feature vectors of the initial nodes in the graph neural network is **4**. This is because:

- Each point in the point cloud is described by four attributes:
  - x, y, z: The spatial coordinates, representing the 3D position of the point.
  - r: The reflectivity, a scalar property of the point.

- Therefore, the feature vector for each node in the graph is the concatenation of these attributes, resulting in a **4-dimensional vector** $[x, y, z, r]$.

Justification:

- **Input Requirements of GNNs:** Each node in a graph neural network requires a feature vector that encodes the properties of the corresponding data point. For a point cloud, the natural raw input features are the spatial coordinates and reflectivity.
- **Completeness of Representation:** The combination of (x, y, z) and r fully captures the characteristics of each point in the point cloud, making 4 the dimensionality of the input feature vector.

Task 8b - 3D Processing (PointCloud, Edge Function)

Answer:

1. Dropping the Aggregation Operation

When the edge function is defined as $\boldsymbol{h}_\Theta\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = \boldsymbol{h}_\Theta\left(\boldsymbol{x}_i\right)$, the aggregation operation $\square$ is effectively dropped, and the feature update depends only on the node's current feature vector $\boldsymbol{x}_i$.

Disadvantage:

- The network loses the ability to incorporate information from neighboring nodes ($\mathcal{E}_i$) during the update process.
- As a result, the node features are not influenced by the local graph structure or the features of neighboring nodes, severely limiting the expressiveness of the graph neural network.
- The network reduces to a fully independent node-wise function, which is equivalent to applying a multilayer perceptron (MLP) to each node without considering its graph connectivity.

2. Using the Edge Function $h_\Theta\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right) = h_\Theta\left(\boldsymbol{x}_j\right)$

Advantage:

- The update rule depends solely on the features of neighboring nodes ($\boldsymbol{x}_j$), allowing the network to explicitly aggregate information from the local neighborhood. This can improve the network's ability to capture and propagate structural and contextual information across the graph.

Disadvantage:

- The updated feature vector $\boldsymbol{x}_i'$ does not consider the features of the central node ($\boldsymbol{x}_i$) itself. This omission may result in the loss of essential node-specific information, potentially degrading the network's performance on tasks where the individual node's characteristics are critical.

Task 8c - Multi-Object Tracking

Answer:

1. Initial Features in G_t:

- **Node Features:** These represent properties of individual detections. Examples include:

- Spatial coordinates (e.g., x, y, z).
- Visual embeddings (e.g., derived from a CNN).
- Velocity or other kinematic information.
- Reflectivity, if available.

Node features encode information about the object represented by each node.
- **Edge Features:** These represent relationships or similarities between pairs of detections. Examples include:

- Spatial proximity.
- Similarity of visual features.
- Temporal differences between detections.

Edge features quantify the likelihood of two nodes (detections) belonging to the same object or track.

2. Importance of Edge Features in G_t and G_s:

- **In G_t:**

- The task involves associating detections across frames. Edge features are critical for modeling relationships between nodes, such as:
  – Motion consistency.
  – Appearance similarity.
  – Temporal proximity.
- Without explicit edge features, the network cannot effectively evaluate whether two detections are likely to correspond to the same object over time, which is the core of solving the association problem.

- **In G_s:**

- The task is segmentation of point clouds, where each node corresponds to a point.
- Node features (e.g., spatial coordinates and reflectivity) are sufficient for learning local geometric patterns.
- The relationships between nodes (e.g., adjacency in space) are implicitly captured by the graph structure itself, making explicit edge features unnecessary.

Task 8d - Single Object Tracking

Answer:

1. Architecture and Computational Differences

Siamese Neural Network:

- A Siamese neural network compares the target object (template patch) with candidate regions in the current frame.
- The network processes both the template and candidate patches in a shared feature extraction branch, and computes their similarity directly in a single forward pass.
- This architecture allows for parallel processing and avoids frame-by-frame fine-tuning.

MDNet (Multi-Domain Network):

- MDNet is designed for robustness through domain adaptation. It includes a domain-specific classifier for each tracking instance.
- At runtime, MDNet fine-tunes its domain-specific layers using samples from the current video sequence.
- This fine-tuning step is computationally expensive, requiring multiple forward and backward passes for optimization.

2. Efficiency in Inference

Siamese Neural Network:

- Once trained, the Siamese network does not require online fine-tuning during tracking.
- It directly computes similarities for all candidate regions using pre-trained weights, making the tracking process significantly faster.

MDNet:

- The online fine-tuning step in MDNet requires iterative gradient updates, which slows down inference.
- Additionally, MDNet processes candidate regions sequentially, further increasing runtime.

3. Parallelization and Search Mechanism

Siamese Neural Network:

- Candidate patches are extracted and compared to the template in parallel, leveraging GPU acceleration.
- The similarity computation is efficient and enables quick evaluation of multiple regions simultaneously.

MDNet:

- Candidate patches are classified individually using the domain-specific classifier, which is a sequential process.
- The sequential nature of this evaluation inherently limits parallelization and increases latency.

4. Adaptability vs. Speed Trade-Off

Siamese Neural Network:

- The absence of online fine-tuning sacrifices some adaptability to appearance changes of the target object but significantly boosts speed.
- Robustness to variations is achieved by training on diverse datasets, which generalizes well to unseen scenarios.

MDNet:

- Online fine-tuning allows MDNet to adapt to specific appearance changes of the target object within a video sequence, enhancing robustness.
- This adaptability comes at the cost of slower runtime due to frequent optimization steps.

Conclusion

A fully connected Siamese neural network is faster than MDNet because:

- It avoids computationally expensive online fine-tuning.
- It processes candidate regions in parallel, leveraging efficient similarity computation.
- Its architecture is inherently optimized for speed, focusing on direct matching rather than iterative adaptation.

While MDNet may achieve higher robustness in some scenarios due to its fine-tuning capabilities, the Siamese network's speed makes it a practical choice for real-time single object tracking.

Exam 2021

Task 1 - Field Of View

Answer:

Task 2 - Self-Attention

Answer:

Self-Attention vs Recurrent Neural Networks (RNNs)

Advantages of Self-Attention Networks Over RNNs

- **Parallelizability:** Self-attention networks allow for parallel computation, as opposed to the sequential processing of RNNs. This significantly reduces the time required for training and inference, especially for long sequences.
- **Shorter Input-to-Output Path:** The input-to-output mapping in self-attention involves fewer operations, as all input elements are treated equally. This improves gradient flow and facilitates learning.

Disadvantages of Self-Attention Networks Compared to RNNs

- **Loss of Positional Information:** The self-attention mechanism is permutation-invariant, making it unable to inherently capture the order of elements in a sequence.
- **Quadratic Computational Complexity:** The memory and computational cost grow quadratically with the sequence length due to the pairwise attention score computation.

Remedies for the Disadvantages

- **Loss of Positional Information:** Introduce **positional encoding**, which adds explicit positional information to the input embeddings.
- **Quadratic Computational Complexity:** Use efficient attention mechanisms such as:
  - **Local or Sparse Attention:** Focus on a neighborhood of each input element to reduce the number of pairwise computations.
  - **Low-Rank Approximations:** Techniques like Linformer or Performer approximate the attention matrix, reducing complexity to linear in sequence length.

Task 3 - Policy Gradients

Answer:

Policy Gradients in Reinforcement Learning

Algorithm Based on the Formulation

The reinforcement learning algorithm based on the importance sampling formulation is **Trust Region Policy Optimization (TRPO)** or **Proximal Policy Optimization (PPO)**. These algorithms leverage importance sampling to estimate and optimize the expected reward while ensuring stable training.

Differentiation and Proof

We start with the importance sampling-based expected reward formulation:

$$U(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau}\sim\boldsymbol{\theta}_{\text{old}}}\left[\frac{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})}\boldsymbol{R}(\boldsymbol{\tau})\right].$$

Step 1: Differentiate $U(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}}\mathbb{E}_{\boldsymbol{\tau}\sim\boldsymbol{\theta}_{\text{old}}}\left[\frac{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})}\boldsymbol{R}(\boldsymbol{\tau})\right].$$

Since the expectation is taken with respect to $\boldsymbol{\theta}_{\text{old}}$, which is independent of $\boldsymbol{\theta}$, we can move the gradient operator inside the expectation:

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau}\sim\boldsymbol{\theta}_{\text{old}}}\left[\nabla_{\boldsymbol{\theta}}\left(\frac{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})}\boldsymbol{R}(\boldsymbol{\tau})\right)\right].$$

Step 2: Differentiate the fraction

The only term dependent on $\boldsymbol{\theta}$ is $\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})$. Thus:

$$\nabla_{\boldsymbol{\theta}} \left(\frac{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})} \boldsymbol{R}(\boldsymbol{\tau}) \right) = \frac{\nabla_{\boldsymbol{\theta}} \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})} \boldsymbol{R}(\boldsymbol{\tau}).$$

Substituting this, we have:

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{\theta}_{\text{old}}} \left[\frac{\nabla_{\boldsymbol{\theta}} \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})} \boldsymbol{R}(\boldsymbol{\tau}) \right].$$

Step 3: Apply the log-derivative trick

Using the log-derivative trick:

$$\nabla_{\boldsymbol{\theta}} \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \log \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \cdot \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}),$$

we substitute:

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{\theta}_{\text{old}}} \left[\frac{\nabla_{\boldsymbol{\theta}} \log \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \cdot \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})} \boldsymbol{R}(\boldsymbol{\tau}) \right].$$

Step 4: Simplify the fraction

When $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}$, we know:

$$\frac{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta})}{\boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}_{\text{old}})} = 1.$$

Thus:

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}} = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{\theta}_{\text{old}}} \left[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \cdot \boldsymbol{R}(\boldsymbol{\tau}) \right].$$

Step 5: Compare with $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

From the original policy gradient expression:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\tau} \sim \boldsymbol{\theta}} \left[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{P}(\boldsymbol{\tau} \mid \boldsymbol{\theta}) \cdot \boldsymbol{R}(\boldsymbol{\tau}) \right].$$

At $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}$, the two gradients are identical:

$$\nabla_{\boldsymbol{\theta}} U(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \Big|_{\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}}.$$

Conclusion

The importance sampling-based estimator $U(\boldsymbol{\theta})$ provides the same gradient as the original objective $J(\boldsymbol{\theta})$ at $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}$, ensuring consistent optimization.

Alternatively:

One can expand the expectation and prove that $J(\theta) = U(\theta)$ and hence the gradients. In this case 4 points each for steps 1,2 ,and 3

$$\begin{aligned}
\nabla_\theta U(\theta) &= \nabla_\theta E_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau \mid \theta)}{P\left(\tau \mid \theta_{\text{old}}\right)} R(\tau) \right] \\
&= \nabla_\theta \int \frac{P(\tau \mid \theta)}{P\left(\tau \mid \theta_{\text{old}}\right)} R(\tau) P\left(\tau \mid \theta_{\text{old}}\right) d\tau \\
&= \nabla_\theta \int P(\tau \mid \theta) R(\tau) d\tau \\
&= \nabla_\theta E_{\tau \sim \theta}[R(\tau)] \\
&= \nabla_\theta J(\theta)
\end{aligned}$$

Task 4 - Generative Adversarial Networks

Answer:

Question 1: Why is it unlikely that the student would succeed with a conditional GAN?

A conditional GAN requires paired data, where the input image (e.g., sonar) and the target image (e.g., optical) correspond to the same location. However, in this scenario, such pairs are unavailable because it is difficult to find sonar and optical images of the exact same seafloor location. The lack of paired data prevents the conditional GAN from learning the desired mapping effectively.

Question 2: Essential steps for a better approach

A better approach would involve using **CycleGAN**, which does not require paired data and is designed for unpaired image-to-image translation. The essential steps are:

1. Architecture

- Use two generators:

$$G : \text{Sonar} \to \text{Optical}, \quad F : \text{Optical} \to \text{Sonar}.$$

- Use two discriminators:

$$\begin{aligned}
D_{\text{Optical}} &: \text{Distinguishes real optical images from generated ones,} \\
D_{\text{Sonar}} &: \text{Distinguishes real sonar images from generated ones.}
\end{aligned}$$

2. Loss Functions

- **Adversarial Loss**: Encourages G and F to generate realistic optical and sonar images, respectively:

$$\mathcal{L}_{\text{GAN}} = \mathbb{E}[\log D_{\text{Optical}}(\text{Optical})] + \mathbb{E}[\log(1 - D_{\text{Optical}}(G(\text{Sonar})))].$$

- **Cycle Consistency Loss**: Ensures that transforming an image to the other domain and back results in the original image:

$$\mathcal{L}_{\text{Cycle}} = \mathbb{E}[\|F(G(\text{Sonar})) - \text{Sonar}\|_1] + \mathbb{E}[\|G(F(\text{Optical})) - \text{Optical}\|_1].$$

3. Training

Minimize the combined loss:

$$\mathcal{L}_{\text{Total}} = \mathcal{L}_{\text{GAN}} + \lambda \mathcal{L}_{\text{Cycle}},$$

where λ balances the adversarial and cycle consistency terms.

4. Data Preparation

- Collect unpaired sonar and optical images of the seafloor.
- Normalize and preprocess the datasets.

5. Evaluation

Use qualitative and quantitative metrics (e.g., FID score) to assess the quality of the generated optical images.

Conclusion

Using CycleGAN, the model can learn the mappings between sonar and optical domains without requiring paired data.

Task 5a - Meta Learning

Answer:

Objective Function in Meta-Learning

The objective function for meta-learning is given as:

$$\theta^\star = \arg\max_\theta \frac{1}{N} \sum_{n=1}^{N} \sum_{x,y \in \mathcal{D}_{\text{query}}^{(n)}} p_\theta \left(y \mid x, \mathcal{D}_{\text{support}}^{(n)} \right),$$

where:

- $\mathcal{D}_{\text{support}}^{(n)}$: A small labeled dataset used to "condition" or adapt the model to the task n.
- $\mathcal{D}_{\text{query}}^{(n)}$: A small dataset used to evaluate the model's performance after conditioning on the support set.
- $p_\theta(y \mid x, \mathcal{D}_{\text{support}}^{(n)})$: The likelihood of predicting label y for input x, given the task-specific adaptation from the support set.

The objective encourages learning a parameterization θ that enables the model to perform well across many tasks, as defined by their support and query sets.

Meta-Learning with Siamese Networks

Siamese networks are a type of meta-learning model where the objective is to learn a metric space that generalizes across tasks. In the context of the above function:

Siamese networks are trained using pairs of data examples $(x_{\text{support}}^{(n)}, x_{\text{query}}^{(n)})$ and their corresponding labels $(c_{\text{support}}^{(n)}, c_{\text{query}}^{(n)})$, where the n-th support set $\mathcal{D}_{\text{support}}^{(n)} = (x_{\text{support}}^{(n)}, c_{\text{support}}^{(n)})$ and the n-th query set $\mathcal{D}_{\text{query}}^{(n)} = (x_{\text{query}}^{(n)}, c_{\text{query}}^{(n)})$. The overall meta-training set is $\mathcal{D}_{\text{train}} = \{\mathcal{D}_{\text{support}}^{(n)} \mid n = 1, 2, \ldots, N\}$, and the meta-test set is $\mathcal{D}_{\text{test}} = \{\mathcal{D}_{\text{query}}^{(n)} \mid n = 1, 2, \ldots, M\}$.

In Siamese networks, the network learns to match the support set example $x_{\text{support}}^{(n)}$ with the query set example $x_{\text{query}}^{(n)}$ based on their labels $c_{\text{support}}^{(n)}$ and $c_{\text{query}}^{(n)}$. The network is trained to produce a high similarity score when $c_{\text{support}}^{(n)} = c_{\text{query}}^{(n)}$ and a low similarity score when the labels are different. This can be framed as a binary classification problem where the goal is to predict whether the query image belongs to the same class as the support image.

The learning objective is based on maximizing the conditional probability of the query label $y = c_{\text{query}}^{(n)}$ given the support set $\mathcal{D}_{\text{support}}^{(n)}$ and the query input $x_{\text{query}}^{(n)}$. The training objective function aligns with the meta-learning objective function:

$$\theta^\star = \arg\max_\theta \frac{1}{N} \sum_{n=1}^{N} \sum_{x,y \in \mathcal{D}_{\text{query}}^{(n)}} p_\theta\left(y \mid x, \mathcal{D}_{\text{support}}^{(n)}\right)$$

Where the network maximizes the conditional probability $p_\theta(y \mid x, \mathcal{D}_{\text{support}}^{(n)})$ by conditioning the query label y on both the support set examples $\mathcal{D}_{\text{support}}^{(n)} = (x_{\text{support}}^{(n)}, c_{\text{support}}^{(n)})$ and the query input $x_{\text{query}}^{(n)}$. During training, the network adjusts the parameters θ to minimize the dissimilarity between $x_{\text{support}}^{(n)}$ and $x_{\text{query}}^{(n)}$ when they belong to the same class, and maximize the dissimilarity when they belong to different classes. This is exactly the same as maximizing the conditional probability of the class label $y = c_{\text{query}}^{(n)}$, conditioned on the support set $\mathcal{D}_{\text{support}}^{(n)}$, which is the objective of meta-learning with Siamese networks.

Therefore, the objective function of Siamese networks is directly aligned with the given meta-learning objective function because the network is trained to maximize the probability of the query label y, conditioned on the support set, which matches the objective function in the meta-learning framework.

1. **Support Set Usage**:
    - During meta-training, the support set is used to define task-specific embeddings. Each class in the support set is embedded into the feature space using a shared encoder (parameterized by θ).
    - For a given query sample x, the distances or similarities between x and the embeddings of the support set are used to predict the label y.
2. **Query Set Usage**:
    - The query set evaluates how well the model has adapted to the task using the support set. The predictions are made based on the learned metric space and compared to the true labels in the query set.
3. **Objective Connection**:
    - The likelihood $p_\theta(y \mid x, \mathcal{D}^{(n)}_{\text{support}})$ in the objective function corresponds to the prediction made by comparing the query sample x to the support set $\mathcal{D}^{(n)}_{\text{support}}$ in the learned metric space.
4. **Training Process**:
    - For each task, the support and query sets are sampled, and the Siamese network computes similarities between the query samples and the support embeddings.
    - The network parameters θ are optimized to maximize the likelihood of the correct labels in the query set, as per the given objective function.

Summary

The above objective aligns with meta-learning using Siamese networks because the support set provides task-specific context, and the query set evaluates the model's performance in adapting to that context. The likelihood $p_\theta(y \mid x, \mathcal{D}^{(n)}_{\text{support}})$ is computed by the Siamese network using the learned metric space, fulfilling the meta-learning objective.

Task 5b - Self-Supervised Learning

Answer:

In self-supervised learning, the **pretext task** and the **downstream task** serve distinct roles:

- **Pretext Task**: The pretext task is a self-imposed task designed to train a model on unlabeled data. The goal of the pretext task is to learn useful representations of the data by solving a task that is easy to define but not necessarily related to the final goal. The model is trained on this task using the data itself to generate supervisory signals. Examples include predicting the next word in a sentence, predicting missing parts of an image, or solving puzzles like predicting the rotation angle of an image. The learned representation from the pretext task captures important features or patterns that can generalize well to other tasks.

- **Downstream Task**: After training on the pretext task, the learned representation is transferred to a **downstream task**, which is the actual task of interest. The downstream task typically involves labelled data, and the learned representation from the pretext task is used to improve the performance on this task. Common downstream tasks include image classification, object detection, or sentiment analysis, depending on the domain. The idea is that the pretext task helps the model learn useful features that are transferable and beneficial for the downstream task.

Thus, the pretext task acts as a proxy for learning useful data representations, and the downstream task is the target application where those learned representations are utilized to perform the actual task.

6a - Bayesian Deep Learning

Answer:

In Bayesian deep learning, we aim to estimate the posterior distribution of the model parameters $\mathbf{w}$ given the observed data $\mathbf{X}$ and $\mathbf{Y}$. According to Bayes' theorem, the posterior distribution $p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y})$ can be expressed as:

$$p(\mathbf{w} \mid \mathbf{X}, \mathbf{Y}) = \frac{p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{Y} \mid \mathbf{X})}$$

Where:

- $p(\mathbf{w})$ is the **prior distribution**, reflecting our belief about the model parameters $\mathbf{w}$ before observing any data.
- $p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w})$ is the **likelihood distribution**, representing the probability of observing $\mathbf{Y}$ given the inputs $\mathbf{X}$ and the parameters $\mathbf{w}$.
- $p(\mathbf{Y} \mid \mathbf{X})$ is the **marginal likelihood** (or evidence), which normalizes the posterior and ensures the probabilities sum (or integrate) to one.

The marginal likelihood can be written as:

$$p(\mathbf{Y} \mid \mathbf{X}) = \int p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})\, d\mathbf{w} \quad \text{(Continuous case)}$$

For discrete model parameters $\mathbf{w}$, the integral is replaced by a sum:

$$p(\mathbf{Y} \mid \mathbf{X}) = \sum_{\mathbf{w}} p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w}) \quad \text{(Discrete case)}$$

Why exact Bayesian inference is difficult in deep learning

Performing exact Bayesian inference is challenging due to the following reasons:

1. **Intractable integral:** The integral (or sum) over $\mathbf{w}$ in the marginal likelihood $p(\mathbf{Y} \mid \mathbf{X})$ is generally intractable. In deep learning, the parameter space $\mathbf{w}$ is high-dimensional, and exact computation of the marginal likelihood is computationally expensive or infeasible.
2. **Complex likelihood:** The likelihood function $p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w})$ is typically highly non-linear in $\mathbf{w}$, which complicates both analytical and numerical evaluation of the posterior.
3. **Non-conjugate priors:** In most deep learning applications, the prior $p(\mathbf{w})$ and the likelihood $p(\mathbf{Y} \mid \mathbf{X}, \mathbf{w})$ are not conjugate distributions. This means the posterior distribution does not have a closed-form solution, requiring approximate inference methods.

Due to these challenges, approximate inference techniques, such as variational inference or Markov chain Monte Carlo (MCMC), are often used to estimate the posterior distribution in Bayesian deep learning.

Task 6b - Evidence Lower Bound

Answer:

Black Box Variational Inference (BBVI) and Bayes by Backprop

Black Box Variational Inference (BBVI)

Black Box Variational Inference (BBVI) is a flexible method for approximating complex posterior distributions in probabilistic models. The key steps are:

- Approximate the true posterior $p(\mathbf{w} \mid \mathcal{D})$ with a variational distribution $q(\mathbf{w} \mid \boldsymbol{\lambda})$, parameterized by $\boldsymbol{\lambda}$.
- The Evidence Lower Bound (ELBO) is optimized to approximate the true posterior. The ELBO is given by:

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\lambda})} \left[\ln p(\mathcal{D} \mid \mathbf{w}) + \ln p(\mathbf{w}) - \ln q(\mathbf{w} \mid \boldsymbol{\lambda})\right]$$

- Monte Carlo sampling is used to estimate the expectations in the ELBO.
- Stochastic Gradient Descent (SGD) is applied to optimize the variational parameters $\boldsymbol{\lambda}$.
- The method is “black box” because it doesn’t require analytical forms for the likelihood or the posterior.

Bayes by Backprop

Bayes by Backprop is a specific implementation of BBVI for Bayesian neural networks. The main steps are:

- Approximate the posterior distribution over network weights $p(\mathbf{w} \mid \mathcal{D})$ using a variational distribution $q(\mathbf{w} \mid \boldsymbol{\lambda})$, typically a Gaussian distribution.
- The ELBO is maximized, where:

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{w}|\boldsymbol{\lambda})} \left[\ln p(\mathcal{D} \mid \mathbf{w}) + \ln p(\mathbf{w}) - \ln q(\mathbf{w} \mid \boldsymbol{\lambda})\right]$$

- Monte Carlo samples are drawn from $q(\mathbf{w} \mid \boldsymbol{\lambda})$ to estimate the expectation.
- The reparameterization trick is used to enable backpropagation through the random variables in the variational distribution:

$$\mathbf{w} = \boldsymbol{\mu} + \boldsymbol{\sigma} \cdot \epsilon$$

  where $\epsilon \sim \mathcal{N}(0, I)$, and $\boldsymbol{\mu}, \boldsymbol{\sigma}$ are the mean and standard deviation of the variational distribution.
- The gradients of the ELBO with respect to $\boldsymbol{\lambda}$ (parameters of the variational distribution) are computed using backpropagation and stochastic gradient descent (SGD).

Connection between BBVI and Bayes by Backprop

Bayes by Backprop is a specific case of BBVI applied to neural networks. Both methods aim to approximate the true posterior by optimizing the ELBO, but Bayes by Backprop:

- Uses the reparameterization trick to backpropagate through stochastic nodes.
- Applies this approach to Bayesian neural networks, allowing for uncertainty estimation in the weights.
- Optimizes the variational parameters using SGD, similar to BBVI.

Task 7a - Inverse Reinforcement Learning

Answer:

Four challenges in Inverse Reinforcement Learning (IRL):

1. **Ambiguity in Reward Function:** In IRL, the reward function is learned from observed behavior, but multiple reward functions can lead to similar behavior. This means there is ambiguity in recovering a unique reward function that explains the observed demonstrations.

2. **Sample Efficiency:** IRL often requires a large amount of demonstration data to effectively learn the reward function, which can be impractical in environments where obtaining demonstrations is expensive or time-consuming.

3. **Ill-Defined Problem:** The problem of inverse reinforcement learning is often ill-posed, as there may not be a unique or well-defined solution for the reward function given the observed demonstrations, especially in complex environments.

4. **Non-Optimal Expert Demonstrations:** Expert demonstrations may not always be drawn from an optimal policy. If the expert's behavior is suboptimal, it becomes challenging for the IRL algorithm to infer the true underlying reward function that would have generated the expert's actions.

Task 7b - Generative Adversarial Imitation Learning

Answer:

GAN Entity	**Equivalent GAIL Entity**
Generator (G)	Policy π_θ
Discriminator (D)	Critic or Value function D_ϕ
Real data samples x^{real}	Expert trajectory or action $\mathcal{D}_{\text{expert}}$
Fake data samples x^{fake}	Generated trajectory or action $\mathcal{D}_{\text{generated}}$
G maximizes probability of x^{fake}	Policy maximizes likelihood of generated actions/trajectories
D maximizes probability of x^{real}	Critic maximises the probability of expert actions/trajectories
D minimizes probability of x^{fake}	Critic minimises the probability of generated actions/trajectories

Tabell 2: Equivalence between GAN and GAIL entities and operations.

In GAIL, the discriminator (D) works by distinguishing between expert trajectories τ_e and generated trajectories τ_p. The behavior of D depends on how the objective function J changes with the probability distribution $P(\tau_p)$. If J is a monotonically increasing function of $P(\tau_p)$, D behaves as described in the table: it maximizes the probability of expert trajectories $P(\tau_e)$ and minimizes the probability of generated trajectories $P(\tau_p)$. If J is monotonically decreasing, the roles of the probabilities are reversed: D would maximize the probability of generated trajectories $P(\tau_p)$ and minimize the probability of expert trajectories $P(\tau_e)$. This reverse behavior is also valid but less commonly used.

Task 8a - 3D Processing (GNN)

Answer:

In the context of graph neural networks (GNNs) for point cloud segmentation, each point in the point cloud is treated as a node in the graph. Given that the point cloud consists of 100 points, each point corresponds to one node.

Since the student is designing a network for segmentation, and each point cloud contains 100 points, the number of nodes in the graph for each layer of the GNN will likely be 100. This is because each node in the graph corresponds to a point in the point cloud, and this holds for all layers of the network unless there is downsampling or other operations altering the number of nodes.

Thus, the number of nodes in the graph for any given layer is 100.

Task 8b - 3D Processing (MLPs)

Answer:

Using a series of identical Multi-layer Perceptrons (MLPs) independently operating on the input points for point cloud segmentation has several disadvantages compared to a graph convolution-based system:

- **Loss of Spatial Structure**: MLPs treat each point independently, which means they do not take into account the relationships or spatial dependencies between the points in the point cloud. This is a significant disadvantage, as the structure of the point cloud (e.g., local neighborhoods) is crucial for effective segmentation. In contrast, graph convolutional networks (GCNs) can explicitly capture spatial relationships between points through graph edges, allowing the network to learn the structure of the point cloud.
- **Limited Neighborhood Information**: In MLP-based approaches, each point is processed in isolation, so the model cannot directly aggregate information from neighboring points unless explicitly designed to do so through pooling or other mechanisms. Graph convolutions, on the other hand, aggregate features from neighbouring nodes (points) in the graph, which helps the network learn more meaningful representations of the point cloud.
- **Scalability Issues**: In MLP-based systems, each point is processed independently, which can lead to scalability issues when dealing with large point clouds, as the system cannot exploit any locality or dependencies between points. In contrast, graph-based methods efficiently process point clouds by exploiting local structures and relationships, making them more scalable for large datasets.
- **Inefficient Feature Aggregation**: MLPs apply the same operation independently to each point, which means that they may not efficiently combine features of neighbouring points in a way that reflects the global structure of the point cloud. Graph convolution layers aggregate information from neighboring points in a structured manner, leading to more robust and meaningful feature representations for segmentation tasks.

In summary, the MLP-based approach lacks the ability to capture local dependencies and the spatial structure of the point cloud, making it less effective for tasks like segmentation where such information is vital. Graph convolutions, by explicitly modeling relationships between points, offer a more powerful and scalable solution for point cloud segmentation.

Task 8c - Multi-Object Tracking

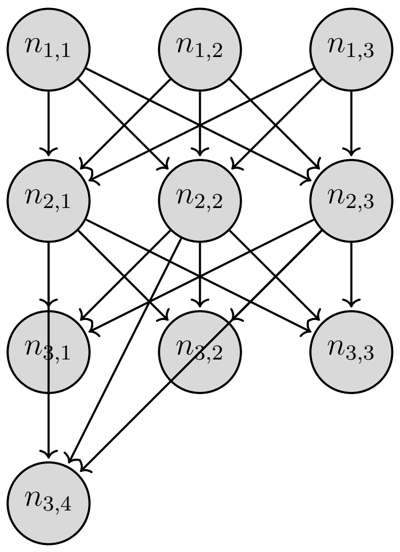
Answer:

Multi-Object Tracking: Graph Representation and Solution

Graph Representation

The graph can be represented as follows:

- **Nodes**: Each detection at each time step is represented as a node. We have:
  - 3 nodes at $t = 1$, 3 nodes at $t = 2$, and 4 nodes at $t = 3$.
- **Edges**: Edges represent possible associations between detections across consecutive time steps. These edges connect nodes from $t = 1$ to $t = 2$ and from $t = 2$ to $t = 3$.
  - Edges from nodes in $t = 1$ to nodes in $t = 2$ (3x3 = 9 possible edges).
  - Edges from nodes in $t = 2$ to nodes in $t = 3$ (3x4 = 12 possible edges).



Figur 1: Graph structure for multi-object tracking across three time steps $t = 1, 2, 3$.

Initial Node and Edge Features Calculation

- **Node Features**:
  - Detection Features: For each node, extract features such as position (coordinates), velocity, or appearance (e.g., color, shape).
  - Temporal Features: Add temporal information, such as time step indices, to represent the relative time for each detection.
- **Edge Features**:
  - Spatial/Distance Features: Calculate the distance between detections across consecutive time steps (Euclidean distance between positions of detections).
  - Motion Features: Consider the motion model (velocity or direction) between detections at consecutive time steps. The change in position from $t = 1$ to $t = 2$, and from $t = 2$ to $t = 3$, can be encoded as edge features.

Procedure to Solve the Association Problem

- **Define the Optimization Problem**:
  - The goal is to maximize the likelihood of the associations between detections at consecutive time steps, ensuring that each object is tracked across frames.
- **Graph Neural Network (GNN) Model**:
  - Use a GNN to compute the node and edge representations iteratively.
  - The GNN can aggregate the node features based on neighboring nodes and edges, updating the features to learn associations between detections across time steps.
- **Tracking**:
  - After computing the node and edge representations, use a suitable method (e.g., Hungarian algorithm, max-flow, or neural network-based classifiers) to solve the association problem and assign detections from one frame to the next.
- **Final Tracking Output**:
  - The output will be the correct associations between detections over the three time steps, solving the multi-object tracking problem.

Task 8d - Single Object Tracking

Answer:

For single-object tracking, a Siamese neural network is used to compare a reference image (the object to track) and a candidate image (the current frame or search region). The network outputs a similarity score indicating how likely it is that the object in the candidate image matches the reference.

Why the Network Should Be Fully Convolutional:

1. **Translation Equivariance:**
   - To perform tracking correctly, the network must be translation-equivariant, meaning it should detect the object at any location in the image, regardless of its position.
   - Fully convolutional networks (FCNs) satisfy this requirement because they apply the same filters across the entire image, maintaining spatial relationships. This allows the network to compare the reference object to all possible locations in the candidate image, even if the object moves across frames.
2. **Efficient Comparison:**

- In a Siamese network for tracking, the network compares the reference image to the candidate image at various positions. An FCN allows this by sliding over the candidate image and performing comparisons at each location. This makes the network capable of locating the object regardless of where it appears in the frame.

3. **End-to-End Learning:**

   - The fully convolutional architecture enables end-to-end learning of similarity features, allowing the network to learn to distinguish the object from the background effectively.
   - Additionally, the score map produced by the network can be traced back to the input image, enabling accurate localization of the object.

In summary, a fully convolutional network is essential for single-object tracking with Siamese networks, as it ensures translation equivariance, efficient search across the image, and correct localization of the object within the frame.

Exam 2020

Task 2 - Recurrent Neural Network attention

Answer:

Disadvantages of Introducing Attention in Recurrent Neural Networks (RNNs)

The attention mechanism can help RNNs handle long sequences more effectively by attending to previous state values. However, it comes with several potential disadvantages:

1. **Increased Computational Complexity:** The attention mechanism requires computing alignment scores between the current state and all previous states, which increases computation time and memory usage, especially for long sequences.
2. **Higher Memory Requirements:** Storing all previous states in memory to compute attention scores adds significant overhead. This can make the method infeasible for resource-constrained environments or very long sequences.
3. **Risk of Overfitting:** Attention mechanisms often introduce additional parameters (e.g., for scoring functions), which increases the model's capacity. This can lead to overfitting, especially when the dataset is small.
4. **Loss of Temporal Structure:** While attention focuses on salient parts of the sequence, it may disregard the temporal order or structure of the input sequence, which could be crucial for tasks that depend on sequential dependencies.

5. **Interpretability Challenges:** Although attention weights provide some interpretability, the dynamic focus on past states can complicate attributing decisions to specific parts of the sequence, especially compared to simpler RNNs.
6. **Potential Latency Issues:** Attention requires processing all previous states at every time step, which can introduce latency in real-time applications where low processing time is critical.

These disadvantages highlight the trade-offs involved when incorporating attention mechanisms in RNNs.

Task 3 - Proximal Policy Optimization

Answer:

Proximal Policy Optimization (PPO) improves stability and sample efficiency in reinforcement learning through the following mechanisms:

1. **Clipped Surrogate Objective:** PPO uses the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$$

and optimizes the objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right],$$

where $\hat{A}_t$ is the advantage estimate. Clipping prevents large, destabilizing updates.

2. **Multiple Updates:** PPO reuses the same batch of data for several gradient steps, improving sample efficiency compared to single-pass methods.
3. **Entropy Regularization:** An entropy bonus encourages exploration, avoiding premature convergence to suboptimal deterministic policies.
4. **Trust Region Approximation:** PPO approximates Trust Region Policy Optimization (TRPO) with simpler first-order optimization, balancing stability and ease of implementation.

These features make PPO stable, sample-efficient, and broadly applicable.

Task 4 - Generative Adversarial Networks

Answer:

To build an application that predicts how a person might look at a desired output age, using the three inputs (current face image, current age, and desired output age), and leveraging GANs, the following approach can be used:

Approach

CycleGan

To predict how a person might look at a desired age using CycleGAN, the process is as follows:

1. **Dataset Preparation:** Divide the dataset into unpaired sets of images for each pair of ages or age groups (e.g., 20–25 and 50–55).
2. **CycleGAN Training:** Train a CycleGAN for each pair of age sets. The two generators in the CycleGAN learn mappings between the two domains, while the cycle consistency loss ensures identity preservation:
$$\mathcal{L}_{\text{cycle}} = \mathbb{E}_{x\sim X}\|F(G(x)) - x\|_1 + \mathbb{E}_{y\sim Y}\|G(F(y)) - y\|_1.$$
3. **Application:** Use the input image, current age, and desired age to select the appropriate CycleGAN model. Pass the input image through the trained generator for the target age to produce the transformed output.
4. **Granularity Trade-Off:** Age groups (e.g., 20–25) can be used instead of specific ages to increase training data per model, at the cost of coarser granularity in predictions.

This approach uses CycleGAN's capability for unpaired image-to-image translation to perform realistic and identity-preserving age transformations.

StarGan

1. **Dataset Preparation:** Divide the dataset into subsets based on age ranges (e.g., 0–10, 11–20, etc.). This allows the network to learn transformations between different age domains.
2. **GAN Architecture for Unpaired Data:** Use **StarGAN** or **CycleGAN**, as the dataset likely lacks paired examples of the same person at different ages.
   - **StarGAN:** A single generator $G(x, c_{\text{target}})$ is conditioned on both the input image x and the target domain label c_{target}, representing the desired age range.
   - The discriminator D distinguishes between real and generated images and predicts the domain label of the input.
3. **Loss Functions:**
   - **Adversarial Loss:** Ensures realism in the generated image.
$$\mathcal{L}_{\text{adv}} = \mathbb{E}[\log D(x)] + \mathbb{E}[\log(1 - D(G(x, c_{\text{target}})))].$$
   - **Domain Classification Loss:** Ensures that generated images match the target age group.
$$\mathcal{L}_{\text{cls}} = -\mathbb{E}[\log D_{\text{domain}}(c_{\text{target}})].$$

- **Reconstruction Loss:** Ensures identity preservation by reconstructing the original image when transforming back to the original domain.

$$\mathcal{L}_{\text{rec}} = \mathbb{E}\|G(G(x, c_{\text{target}}), c_{\text{original}}) - x\|_1.$$

4. **Training Process:** Train StarGAN on unpaired subsets of the dataset, allowing it to learn mappings between all age domains.

5. **Application Pipeline:**

   a) For a given input image, current age, and target age:
      - Identify the target age domain.
      - Use the generator G to produce the transformed image $G(x, c_{\text{target}})$, representing the person's appearance at the desired age.

   b) Post-process to ensure smoothness and realism if needed.

Why StarGAN is Suitable:

- Handles transformations across multiple domains (age ranges) with a single model.
- Does not require paired data, making it practical for the given dataset.
- Preserves identity while achieving age transformations.

This approach leverages StarGAN's multi-domain capability to output realistic age-progressed or age-regressed images.

Task 5 - Learning Concepts (Meta-Learning)

Answer:

Meta-Learning Techniques and Optimization Objective

Meta-learning aims to optimize a model's ability to adapt to new tasks quickly with limited data. The given objective function describes the optimization of a model's parameters θ to maximize the likelihood of the model's predictions on query sets, given the support sets for each task. Below are two meta-learning techniques that align with this objective function.

1. Prototypical Networks

Prototypical Networks are a meta-learning technique that works with the given objective function by learning a metric space where each class in a task is represented by a prototype. Here's how this method works:

1. **Task Setup:** For each task n, the dataset is split into a support set $\mathcal{D}^{(n)}_{\text{support}}$ and a query set $\mathcal{D}^{(n)}_{\text{query}}$, where the support set contains labeled data and the query set contains unlabeled data.

2. **Prototype Computation:** For each class c in the support set, compute the prototype $\mathbf{p}_c$, which is the mean of all support set samples belonging to class c:

$$\mathbf{p}_c = \frac{1}{|\mathcal{D}^{(n)}_{\text{support},c}|} \sum_{\mathbf{x} \in \mathcal{D}^{(n)}_{\text{support},c}} f_\theta(\mathbf{x}),$$

   where f_θ is the embedding function parameterized by θ.

3. **Query Prediction:** For each query sample $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}^{(n)}_{\text{query}}$, the model computes a probability distribution over classes by comparing the embedded query point $f_\theta(\mathbf{x})$ to each prototype using a softmax function:

$$p_\theta(\mathbf{y} \mid \mathbf{x}, \mathcal{D}^{(n)}_{\text{support}}) = \frac{\exp(-d(f_\theta(\mathbf{x}), \mathbf{p}_{\mathbf{y}}))}{\sum_c \exp(-d(f_\theta(\mathbf{x}), \mathbf{p}_c))},$$

   where d is a distance metric, such as Euclidean distance.

4. **Optimization:** The parameters θ are optimized to maximize the log-likelihood of the query labels given the support set:

$$\theta^* = \arg\max_\theta \frac{1}{N} \sum_{n=1}^{N} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}^{(n)}_{\text{query}}} \log p_\theta(\mathbf{y} \mid \mathbf{x}, \mathcal{D}^{(n)}_{\text{support}}).$$

Why Prototypical Networks Align with the Objective Function

- The meta-learning task distribution is represented by the summation over tasks n, reflecting the model's optimization across multiple tasks.
- The support and query sets are directly aligned with the structure of $\mathcal{D}^{(n)}_{\text{support}}$ and $\mathcal{D}^{(n)}_{\text{query}}$, where the model is trained using the support set to predict labels on the query set.
- The probabilistic prediction $p_\theta(\mathbf{y} \mid \mathbf{x}, \mathcal{D}^{(n)}_{\text{support}})$ is computed based on the learned embeddings, aligning with the given objective function.

2. Model-Agnostic Meta-Learning (MAML)

Model-Agnostic Meta-Learning (MAML) is another meta-learning technique that also fits the given objective function, particularly with a focus on fast adaptation. Here's how it works:

1. **Task Setup:** Similar to Prototypical Networks, each task n consists of a support set $\mathcal{D}^{(n)}_{\text{support}}$ and a query set $\mathcal{D}^{(n)}_{\text{query}}$.

2. **Model Initialization:** MAML aims to find an initialization of model parameters θ_0 that can be quickly adapted to new tasks with a few gradient steps. The model is trained on a meta-objective that encourages fast adaptation.

3. **Inner Loop (Task-Specific Update):** For each task, update the model parameters using the support set $\mathcal{D}_{\text{support}}^{(n)}$:

$$\theta_n' = \theta_0 - \alpha \nabla_\theta \mathcal{L}_{\text{task}}(\theta_0, \mathcal{D}_{\text{support}}^{(n)}),$$

   where α is the learning rate and $\mathcal{L}_{\text{task}}$ is the loss function for the task.

4. **Outer Loop (Meta-Update):** After adapting the model to each task, evaluate the model on the query set $\mathcal{D}_{\text{query}}^{(n)}$ and compute the meta-gradient:

$$\mathcal{L}_{\text{meta}} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_{\text{task}}(\theta_n', \mathcal{D}_{\text{query}}^{(n)}).$$

   The model parameters θ_0 are updated to minimize this meta-loss:

$$\theta_0^* = \arg\min_{\theta_0} \mathcal{L}_{\text{meta}}.$$

Why MAML Aligns with the Objective Function

- MAML's goal is to optimize the model parameters θ_0 such that after a few gradient updates using the support set, the model can quickly perform well on the query set, which is exactly what the objective function describes.
- The objective function maximizes the probability of the model's predictions on the query set, which corresponds to MAML's goal of improving performance on query tasks after adaptation.
- Like Prototypical Networks, the method uses both support and query sets and is optimized across multiple tasks, aligning with the structure of the objective function.

Few-Shot Learning in Meta-Learning

Meta-learning algorithms for few-shot learning focus on training models to quickly adapt to new tasks. The general approach is to learn a model that can make predictions on novel tasks by utilizing prior knowledge gained from a meta-training process that involves a variety of tasks. Some important techniques in few-shot learning include:

- **Learning a good initialization**: MAML is a meta-learning technique that learns a good initialization of model parameters that can be quickly adapted to new tasks. This helps in few-shot learning by minimizing the number of examples required to fine-tune the model.

- **Metric-based learning**: Prototypical Networks, for example, learn a metric space where each class is represented by a prototype, making it easy to classify new examples with minimal data. This approach can be directly applied to few-shot classification tasks.
- **Transfer learning**: This approach involves leveraging previously learned knowledge from related tasks, which is then transferred to the current task with minimal adjustment. Few-shot learning can be seen as an extension of transfer learning, where the model is expected to adapt quickly based on prior experience.

Challenges in Few-Shot Learning

Few-shot learning presents several challenges, including:

- **Overfitting**: With so few labeled examples, models are prone to overfitting, learning noise or irrelevant patterns from the small data.
- **Class imbalance**: In few-shot learning tasks, classes may have a significant imbalance, where some classes may have more examples than others, making the learning task harder.
- **Generalization**: Few-shot models need to generalize well from a limited amount of data. Ensuring that the model captures essential features of a task rather than memorizing specific examples is a key challenge.

Conclusion

Prototypical Networks, Model-Agnostic Meta-Learning (MAML) and Few-Shot Learning are meta-learning techniques that align with the objective function. Prototypical Networks use a metric learning approach with prototypes, while MAML focuses on fast adaptation by updating the model's parameters for each task. Both methods optimize the model across a distribution of tasks and predict labels based on support-query set pairs, making them well-suited for the objective described.

Task 6a - Bayesian Deep Learning (ELBO)

Answer:

Why ELBO is Easier to Work With Than the KL Divergence

In variational inference, we aim to approximate the posterior distribution $p(\mathbf{w} \mid \mathcal{D})$ with a variational distribution $q(\mathbf{w})$. This is achieved by minimizing the Kullback-Leibler (KL) divergence between the two distributions, defined as:

$$\mathrm{KL}(q(\mathbf{w}) \| p(\mathbf{w} \mid \mathcal{D})) = -\mathbb{E}_{q(\mathbf{w})} \ln \frac{p(\mathbf{w} \mid \mathcal{D})}{q(\mathbf{w})}.$$

However, directly minimizing the KL divergence is difficult due to the intractability of the posterior $p(\mathbf{w} \mid \mathcal{D})$, which involves the normalization constant $p(\mathcal{D}) = \int p(\mathbf{w}, \mathcal{D})\, d\mathbf{w}$.

To overcome this, we use the **Evidence Lower Bound (ELBO)**, which is defined as:

$$\text{ELBO} = -\mathbb{E}_{q(\mathbf{w})} \ln \frac{p(\mathbf{w}, \mathcal{D})}{q(\mathbf{w})}.$$

The ELBO provides a lower bound on the log marginal likelihood of the data $\log p(\mathcal{D})$. By maximizing the ELBO, we indirectly minimize the KL divergence between the variational distribution $q(\mathbf{w})$ and the posterior $p(\mathbf{w} \mid \mathcal{D})$.

The reason the ELBO is easier to work with than the KL divergence is that it avoids the direct computation of the posterior, which is intractable. Instead, the ELBO involves the joint distribution $p(\mathbf{w}, \mathcal{D})$, which can be optimized more efficiently, especially by using variational approximations or Monte Carlo methods. This makes ELBO a more computationally feasible objective for variational inference.

Task 6b - Bayesian Deep Learning (Gradient ELBO)

Answer:

Why Paul's Approach Fails and a Better Solution

Paul's approach to approximating the ELBO, $\hat{\mathcal{L}}(\lambda)$, involves sampling $\mathbf{w}^s$ from $q(\mathbf{w}, \lambda)$ and attempting to compute the gradient of the resulting expression. However, this approach is flawed because the term $\ln q(\mathbf{w}^s, \lambda)$ involves the sampled variable $\mathbf{w}^s$, making it nontrivial to differentiate with respect to λ.

The issue arises because $\mathbf{w}^s$ is drawn randomly from $q(\mathbf{w}, \lambda)$, which means the log-probability $\ln q(\mathbf{w}^s, \lambda)$ is not directly differentiable with respect to λ. This makes computing the gradient of $\hat{\mathcal{L}}(\lambda)$ problematic.

A more successful procedure would involve using the **reparameterization trick**. This method expresses $\mathbf{w}^s$ as a deterministic function of λ and a noise variable ϵ, often drawn from a simple distribution (such as Gaussian). This allows the gradient to be computed with respect to λ by backpropagating through the noise, making the procedure differentiable. Thus, Paul could compute the gradient using the **reparameterization trick** or **Monte Carlo estimation with the pathwise derivative** to obtain a gradient estimate for λ and successfully optimize the ELBO.

Task 7 - Deep Learning for Control

Answer:

Challenges in Deep Learning for Mobile Robot Control

Compounding Errors

Approach

Compounding errors can be mitigated using *reinforcement learning (RL)*, where the agent continuously learns optimal policies through interaction with the environment. *DAGGER (Dataset Aggregation)* combines expert demonstrations with learned policies. Another approach is using techniques like those in *NVIDIA's self-driving system*, where simulations are used for training and real-world fine-tuning.

Limitations

These methods require extensive interaction with the environment, particularly with immature policies, increasing the risk of *catastrophic events*. Simulations and expert policies may not generalize well to real-world scenarios, and approaches like NVIDIA's are limited in dynamic or unpredictable environments.

Complex Reward Functions

Approach

Inverse reinforcement learning (IRL) is used to learn reward functions by observing expert demonstrations. This allows the system to infer the reward structure from the expert's behavior, useful for complex or unknown reward functions.

Limitations

IRL is an ill-defined problem, computationally complex, and expert demonstrations may be suboptimal, leading to subpar reward functions and poor performance.

High Risk in Exploration in Unknown Environments

Approach

Safe exploration can be achieved by restricting exploration to known good policies or using *simulators* to test actions safely before deployment, which reduces risk during exploration.

Limitations

Limited exploration may lead to *compounding errors* because the agent may not experience enough scenarios to generalize. Simulators often have a *reality gap*, where policies learned in simulation do not transfer well to real-world environments.

Task 8a - 3D Processing (Regular - 3D Convolutions)

Answer:

Difference Between Regular and 3D Convolutions

In regular convolution operations (e.g., in networks like ResNet), the input data is typically 3D (height, width, and depth for an image or feature map), and 2D convolution filters are applied across the spatial dimensions (height and width). The depth of the filter is usually fixed to the number of channels or feature maps, but the filter itself operates over the spatial dimensions only.

In contrast, **3D convolutions** involve a filter that operates across all three dimensions of the input data (height, width, and depth), making them suitable for 3D data such as volumetric data, videos, or medical imaging. The filter in 3D convolution has a depth dimension, which is used to capture information across both spatial dimensions (height and width) and the temporal or depth dimension (e.g., time in videos, or depth in medical scans).

Thus, while both types of convolutions involve 3D inputs, the key difference is that **3D convolutions use filters with a third dimension** (depth), while standard convolutions apply 2D filters and treat depth as a set of independent channels.

Task 8b - 3D Processing (PointCloud, MLP)

Answer:

Dimensions of MLP Outputs for a Point Cloud

Given the problem setup, each point in the point cloud has a dimension of 3 (representing its 3D coordinates). The Multi-layer Perceptron (MLP) is applied in parallel over every point in the point cloud. The MLP output for each point is a vector of dimension 256.

Explanation:

- **Input:** Each point in the point cloud has a dimension of 3 (representing its 3D coordinates).
- **MLP Operation:** The MLP processes each of the 100 points in the point cloud independently and outputs a 256-dimensional vector for each point.
- **Output for each point:** For each of the 100 points, the MLP produces a vector of dimension 256.
- **Output for the whole point cloud:** Since there are 100 points and the MLP outputs a 256-dimensional vector for each point, the output for the whole point cloud is a matrix of dimensions 100×256.

Answer: The dimensions of the MLP outputs for the whole point cloud are 100×256.

Task 8c - 3D Processing (Permutation Invariance)

Answer:

Achieving Permutation Invariance in MLP Outputs

To achieve permutation invariance in the network, the student needs to ensure that the order of the points in the point cloud does not affect the output classification (Hence a symmetric operation). One effective approach is to aggregate the MLP outputs using a **permutation-invariant function**.

Common approaches to achieve permutation invariance:

- **Summing the outputs:** One simple approach is to sum the output vectors of all points in the point cloud. This will result in a single 256-dimensional vector, where the order of the points does not matter.

$$\mathbf{h}_{\text{sum}} = \sum_{i=1}^{100} \mathbf{h}_i$$

  where $\mathbf{h}_i$ is the 256-dimensional MLP output for the i-th point.

- **Averaging the outputs:** Another common approach is to average the output vectors:

$$\mathbf{h}_{\text{avg}} = \frac{1}{100} \sum_{i=1}^{100} \mathbf{h}_i$$

  This results in a 256-dimensional vector, again without regard to the order of the points.

- **Max pooling:** Alternatively, the student could apply max pooling across the MLP outputs. The max pooling operation selects the maximum value from each of the 256 dimensions across all 100 points:

$$\mathbf{h}_{\text{max}} = \max(\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_{100})$$

  This operation preserves permutation invariance by taking the maximum value for each feature dimension.

Answer: To achieve permutation invariance, the student can combine the MLP outputs by using functions like summation, averaging, or max pooling across the outputs of all 100 points. These methods aggregate the information in a way that does not depend on the order of the points in the point cloud.

Task 8d - 3D Processing (Geometric Transformations)

Answer:

Improving Robustness Against Geometric Transformations

To improve the robustness of the network against geometric transformations of the inputs (such as rotations, translations, or scalings), the student can consider the following techniques:

1. Data Augmentation

The student can apply various geometric transformations (such as rotations, translations, and scalings) to the point clouds during training. This approach helps the network learn to classify point clouds regardless of their geometric variations. Examples of transformations to apply include:

- **Rotation**: Rotate the point cloud around different axes.
- **Translation**: Shift the points in space.
- **Scaling**: Scale the point cloud by a factor.

By exposing the network to a wide variety of geometric transformations, the model can become more invariant to these transformations at test time.

2. Equivariant Networks (Spatial Transformer Networks)

Group Equivariant Convolutional Networks (G-CNNs): These networks are designed to be equivariant to transformations such as rotations, translations, and reflections. In such networks, the architecture is modified to handle transformations by applying convolution operations that respect these transformations.

PointNet and PointNet++: These architectures explicitly learn a permutation-invariant function and can be extended to be invariant to geometric transformations by using transformations in the form of spatial normalization or by applying specific transformations during the learning process.

3. Learning Invariant Features

Transformation-Invariant Networks: The student can incorporate methods that explicitly aim to extract invariant features from the point cloud. For example, features can be extracted in a way that the learned representation does not depend on the geometric transformation applied to the input point cloud.

Network Architecture: The network can be designed to focus on key local geometric features that are less sensitive to transformation, using methods like local feature extraction or hierarchical feature learning (as seen in PointNet++).

4. Point Cloud Alignment

Prior to feeding the point cloud into the network, alignment techniques can be applied to standardize the input, such as:

- **Procrustes Analysis**: Aligning the point cloud by rotating and translating it to a canonical pose.
- **Principal Component Analysis (PCA)**: Aligning the point cloud by aligning it with the principal axes.

Answer: To improve the robustness of the network against geometric transformations of the inputs, the student can use data augmentation, incorporate equivariant network architectures like G-CNNs or PointNet++, learn transformation-invariant features, or apply point cloud alignment techniques. These approaches help the network generalize across geometric transformations such as rotations, translations, and scalings.

Task 8e - 3D Processing (3D processing Network Architecture)

Answer:

3D Object Tracking Network Architecture

For the 3D object tracking task, the student can extend a Siamese fully convolutional network (FCN) by using a 3D variant to process point cloud data and capture spatial relationships over time. The proposed architecture is as follows:

1. Siamese Network Architecture

The Siamese network consists of two identical sub-networks that share weights, allowing the model to compare two inputs. In this case, the inputs are consecutive point clouds, and the task is to determine the similarity between them.

2. Point Cloud Representation

Each point cloud is represented as a set of 3D points $\mathbf{x} = \{\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_n}\}$, where $\mathbf{x_i} \in \mathbb{R}^3$ is a 3D coordinate of a point in the cloud. Raw points can be processed directly using methods like PointNet or PointNet++.

3. 3D Convolutional Layers

Use 3D convolutional layers to capture spatial features from the point cloud. 3D convolution kernels are applied over the input point cloud data to learn local spatial patterns and object representations. PointNet or PointNet++ can be adapted for this task, using shared Multi-Layer Perceptrons (MLPs) followed by symmetric aggregation functions.

4. Feature Extraction

The Siamese sub-networks share 3D convolutional layers to extract features from the two consecutive point clouds. The output is a global feature vector representing the object in 3D space.

5. Comparison Layer

A distance metric (e.g., Euclidean distance or cosine similarity) compares the feature vectors of the two point clouds, predicting the object's movement or position change between frames.

6. Tracking Strategy

The network predicts the transformation (e.g., translation or rotation) of the object between two consecutive point clouds, allowing for continuous tracking.

7. Post-Processing

The output is used to update the object's position in subsequent frames.

8. Loss Function

A contrastive loss or triplet loss function can be used to train the network, maximizing similarity between matching point clouds and minimizing similarity between non-matching point clouds.

9. Additional Enhancements

Incorporate temporal consistency with recurrent layers like LSTM or GRU to capture temporal dependencies. Data augmentation strategies like random rotations or translations can also improve generalization.

Final Architecture:

- Input: Two consecutive point clouds $(\mathbf{X}_1, \mathbf{X}_2)$.
- Sub-network 1 and 2: Shared 3D convolutional layers (PointNet or PointNet++) to extract features.
- Comparison Layer: Metric learning layer (e.g., Euclidean distance or cosine similarity).
- Tracking: Output transformation (e.g., translation or rotation).
- Loss Function: Contrastive loss or triplet loss.

Exam 2019

Task 1 - Semantic Segmentation

Answer:

An undesirable property of accuracy in this example is that it fails to account for class imbalance. The background class (non-lane pixels) will likely dominate, allowing the model to achieve high accuracy by simply predicting background for most pixels, even if it doesn't segment the lane lines correctly.

Alternative metrics:

- **Intersection over Union (IoU):** Measures the overlap between the predicted and ground truth segmentation for each class, providing a better indication of segmentation quality, especially for smaller classes like lane lines.
- **Dice Similarity Coefficient (DSC):** Similar to IoU but emphasizes the balance between precision and recall, useful for imbalanced classes.
- **Precision:** Measures the accuracy of predicted positive pixels (lane lines) out of all predicted positive pixels.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

- **Recall:** Measures the ability to detect all actual positive pixels (lane lines) out of all actual positives.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

- **F1 Score:** The harmonic mean of precision and recall, providing a balanced measure of both.

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Task 2 - Image Captioning with Attention

Answer:

Content-based Attention in RNN for Image Captioning

To apply content-based attention in a Recurrent Neural Network (RNN) for generating captions for images, you can use the feature map extracted from the pretrained CNN (dimension $h \times w \times c$) and apply attention mechanisms to focus on different parts of the image at each time step.

Approach

1. Extract Feature Maps: Use the pretrained CNN to extract the feature maps of the image, which have dimensions $h \times w \times c$. These feature maps represent the image's visual content.

2. Attention Mechanism: At each time step t, compute a context vector $\mathbf{c}_t$ using an attention mechanism that allows the model to focus on specific spatial regions of the feature map. This involves:

- **Query Vector:** The hidden state $\mathbf{h}_t$ of the RNN (e.g., LSTM or GRU) at time step t is used as the query for the attention mechanism.
- **Key and Value Vectors:** Each spatial location (i, j) in the feature map is treated as a key $\mathbf{k}_{ij}$ and a value $\mathbf{v}_{ij}$, where the keys and values are derived from the feature map. For instance, you can flatten the $h \times w$ spatial dimensions into a vector for each location in the image.

3. Attention Weights Calculation: Compute the attention scores (or weights) α_{ij} for each spatial location (i, j) by measuring the similarity between the query $\mathbf{h}_t$ and the keys $\mathbf{k}_{ij}$. A common choice is to use the dot product followed by a softmax operation:

$$\alpha_{ij} = \frac{\exp(\mathbf{h}_t^T \mathbf{k}_{ij})}{\sum_{i',j'} \exp(\mathbf{h}_t^T \mathbf{k}_{i'j'})}$$

4. Context Vector: Compute the context vector $\mathbf{c}_t$ as a weighted sum of the values $\mathbf{v}_{ij}$ based on the attention weights α_{ij}:

$$\mathbf{c}_t = \sum_{i,j} \alpha_{ij} \mathbf{v}_{ij}$$

5. Generate Caption: Concatenate the context vector $\mathbf{c}_t$ with the hidden state $\mathbf{h}_t$ of the RNN and pass it through a fully connected layer followed by a softmax to predict the next word in the caption.

Summary

At each time step, the attention mechanism allows the RNN to focus on different regions of the image, providing a dynamic, content-based focus on relevant parts of the image as the caption is generated. This helps the model to attend to specific objects or regions in the image corresponding to the current word being predicted in the caption.

Task 3 - Greedy Policy Update

Answer:

No, it is not possible that $v_{\pi'}(s) < v_\pi(s)$ for any state s. This follows from the Policy Improvement Theorem, which states that if a policy π' is derived by greedily choosing the action that maximizes the action-value function $q_\pi(s, a)$ of another policy π, i.e.,

$$\pi'(s) = \arg\max_a q_\pi(s, a),$$

then the new policy π' will either perform better or be at least as good as the original policy π in terms of the state-value function.

Formally, the state-value under the new policy π' is given by:

$$v_{\pi'}(s) = \mathbb{E}_{\pi'}[q_\pi(s, a)],$$

and since π' chooses actions that maximize $q_\pi(s, a)$, we have:

$$v_{\pi'}(s) \geq v_\pi(s).$$

Thus, the state-value under π' is greater than or equal to that under π for all states s, meaning that $v_{\pi'}(s) \geq v_\pi(s)$ for all s. Therefore, it is not possible for $v_{\pi'}(s)$ to be less than $v_\pi(s)$ for any state s.

Task 4 - Proximal Policy Optimazation

Answer:

The hyperparameter ϵ in the clipping function of the Proximal Policy Optimization (PPO) algorithm controls the range within which the policy update is allowed to vary. Specifically, it ensures that the new policy does not deviate too much from the old policy by clipping the probability ratio $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ when it exceeds $1 + \epsilon$ or $1 - \epsilon$. This clipping helps prevent large, unstable policy updates, improving training stability and avoiding destructive updates. The value of ϵ typically ranges between 0.1 and 0.3.

Task 5 - Meta Learning

a.)

Answer:

In meta-learning, particularly with the Matching Network architecture, the structure of a training data sample is different from regular training. A meta-training sample consists of a **support set** and a **query set**.

- **Support set**: A set of labeled examples that the model will use to learn the task. In this case, the support set would contain a few labeled images of various classes (e.g., {truck, train, motorcycle}).
- **Query set**: A set of examples for which the model will make predictions, based on the support set. The query set can consist of a few images, each labeled with the class it belongs to.

Thus, a meta-training data sample for Tom would consist of:

- A support set $\mathcal{D}_{\text{support}}$, which contains labeled images (I, c).
- A query set $\mathcal{D}_{\text{query}}$, which also contains a few images of the same classes but is used for the model to make predictions based on the support set.

In essence, each meta-training sample for the Matching Network consists of a pair $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}})$.

b.)

Answer

Each data sample of the meta-test set consists of a support set and a query set.

- **Support set**: The support set consists of three image examples, each from a distinct class drawn from the set {truck, train, motorcycle}. These images have known class labels.
- **Query set**: The query set contains a single image, which has an unknown class label, also drawn from the set {truck, train, motorcycle}.

Thus, each data sample for Bob in the meta-test set consists of the pair $(\mathcal{D}_{\text{support}}, \mathcal{D}_{\text{query}})$, where $\mathcal{D}_{\text{support}}$ contains three labeled images from the set {truck, train, motorcycle}, and $\mathcal{D}_{\text{query}}$ contains one image with an unknown class label from the same set.

c.)

Answer:

The Matching network outputs a probability vector $[p_1, p_2, p_3]$, where each p_i measures the probability (or similarity, measured through cosine distance) that the query image belongs to the i-th class from the set $\{\text{truck}, \text{train}, \text{motorcycle}\}$. Specifically:

- p_1 is the probability that the query image belongs to the truck"class.
- p_2 is the probability that the query image belongs to the train"class.
- p_3 is the probability that the query image belongs to the "motorcycle"class.

These probabilities are computed based on the similarity between the query image and the images in the support set, with the Matching network using these similarities to determine the most likely class for the query image.

Task 6 - Processing 3D-Scenes (3D Processing)

a.)

Answer:

A major problem with applying 3D-convolutions on voxelized point clouds is the high memory and computational cost. Voxelization involves converting the point cloud into a grid of 3D voxels, which can result in large, sparse grids that requires substantial memory storage and computational resources. Additionally, the resolution of the voxel grid can significantly impact both the accuracy and efficiency of the model. High-resolution voxel grids may lead to prohibitively large memory requirements, while low-resolution grids may fail to capture important details, resulting in poor performance.

b.)

Answer:

PointNet achieves permutation invariance by applying a symmetric function, such as max pooling, to aggregate the features of individual points in the point cloud. Specifically, each point is processed independently through a shared multi-layer perceptron (MLP), generating a feature vector for each point. Then, a max pooling operation is applied across all points, which selects the maximum feature value for each dimension, ensuring that the output is invariant to the order of points in the input.

c.)

Answer:

- **Regular Convolution**: When the aggregation operation is a summation and the edge function is $h_\theta(\mathbf{x}_i, \mathbf{x}_j) = \theta \cdot \mathbf{x}_j$, the edge convolution becomes regular convolution, where the aggregation is a weighted sum of neighboring features.
- **PointNet**: When there is no aggregation (i.e., point-wise operations), and the edge function is $h_\theta(\mathbf{x}_i, \mathbf{x}_j) = h_\theta(\mathbf{x}_i)$, it becomes the operations in PointNet, where each point is processed independently, and the final aggregation is done using max pooling.

This approach mathematically shows how both regular convolution and PointNet are special cases of edge convolution.

Task 7 - Multi-Object Tracking

a.)

Answer:

The four basic operations in multiple object visual tracking (MOT) in which deep learning can be employed are:

1. **Object Detection**: Deep learning models, such as Convolutional Neural Networks (CNNs), can be used to detect and localize objects in each frame, providing bounding box coordinates for potential objects.
2. **Feature Extraction**: Deep learning can be employed to extract discriminative features (e.g., appearance features) of objects, which helps in distinguishing between objects with similar appearance over time.
3. **Data Association**: Deep learning techniques, such as Recurrent Neural Networks (RNNs) or Transformer networks, can be used to match and associate detected objects across frames, helping to track the same object through occlusions or overlaps.
4. **Trajectory Prediction**: Deep learning-based models like Long Short-Term Memory (LSTM) networks can be used to predict the future positions of objects based on their past trajectories, improving the robustness of tracking in challenging scenarios.

b.)

Answer:

A Siamese network can be simpler and more efficient for object tracking compared to MDNet because:

- It only requires running the network once for each input image, whereas MDNet requires separate classifiers for each object.
- There is no runtime training in the Siamese network, unlike MDNet, which needs online updates during tracking.

c.)

Answer:

To address the issue of different scales in Siamese networks, a common method is **multi-scale training**. This approach involves:

1. **Image Pyramid**: Generating an image pyramid where the tracked object is represented at multiple scales (by resizing the image).
2. **Siamese Network Processing**: The network is trained on these different scales, so it learns to match the object regardless of its size in the frame.
3. **Feature Fusion**: The features from different scales are combined to produce a more robust representation for tracking, which can then be used to predict the object's position across scales.

This technique improves the network's ability to handle scale variation and increases tracking accuracy.

Task 8 - Generative Adversarial Networks

Answer:

Conditional GAN for Image Colorization

To formulate the problem of colorizing grayscale images as a conditional GAN (cGAN) problem, we proceed as follows:

- **Generator:** The generator network G takes a grayscale image x_0 as input and generates a color image $G(x_0)$, where G is conditioned on the input x_0 (the grayscale image).
- **Discriminator:** The discriminator network D distinguishes between real color images x_i from the dataset and the generated color images $G(x_0)$, conditioned on the grayscale input x_0.
- **Objective:** The generator G is trained to produce realistic color images that fool the discriminator, while the discriminator is trained to correctly differentiate real color images from generated ones. The objective for training is to minimize the adversarial loss:
$$\mathcal{L}_{\text{GAN}}(G, D) = \mathbb{E}_{x_0,x_i}[\log D(x_0, x_i)] + \mathbb{E}_{x_0}[\log(1 - D(x_0, G(x_0)))]$$
where x_0 is the input grayscale image, and x_i is the corresponding real color image.

Thus, the cGAN is conditioned on the grayscale image x_0, and the goal is to generate plausible colorized images from it.

Task 9 - Evidence Lower Bound

a.)

Answer:

Evidence Lower Bound (ELBO) in KL-Divergence

The Evidence Lower Bound (ELBO) is related to the Kullback-Leibler (KL) divergence between a variational distribution $q(\mathbf{w})$ and the true posterior $p(\mathbf{w} \mid \mathcal{D})$ in variational inference. It is derived from the following relationship:

$$\text{KL}(q(\mathbf{w}) \| p(\mathbf{w} \mid \mathcal{D})) = \mathbb{E}_{q(\mathbf{w})}\left[\log \frac{q(\mathbf{w})}{p(\mathbf{w} \mid \mathcal{D})}\right]$$

Using the factorization of the joint distribution and re-arranging terms, the ELBO is given by:

$$\text{ELBO} = \mathbb{E}_{q(\mathbf{w})}\left[\log p(\mathbf{w}, \mathcal{D}) - \log q(\mathbf{w})\right]$$

This ELBO serves as a lower bound to the log-marginal likelihood $\log p(\mathcal{D})$, and its maximization indirectly minimizes the KL divergence between $q(\mathbf{w})$ and $p(\mathbf{w} \mid \mathcal{D})$, enabling efficient approximate inference.

b.)

Answer:

Maximizing ELBO and its Relation to $q(\mathbf{w})$ and $p(\mathbf{w} \mid \mathcal{D})$

The Evidence Lower Bound (ELBO) provides a lower bound on the log-marginal likelihood $\log p(\mathcal{D})$. By maximizing the ELBO, we indirectly minimize the Kullback-Leibler (KL) divergence between the variational distribution $q(\mathbf{w})$ and the true posterior distribution $p(\mathbf{w} \mid \mathcal{D})$.

$$\text{KL}(q(\mathbf{w}) \| p(\mathbf{w} \mid \mathcal{D})) = \log p(\mathcal{D}) - \text{ELBO}$$

Since the KL divergence is non-negative, we have:

$$\text{KL}(q(\mathbf{w}) \| p(\mathbf{w} \mid \mathcal{D})) \geq 0$$

Maximizing the ELBO increases $\log p(\mathcal{D})$, which leads to a decrease in the KL divergence. A decrease in KL divergence implies that $q(\mathbf{w})$ gets closer to $p(\mathbf{w} \mid \mathcal{D})$. Thus, by maximizing the ELBO, the variational distribution $q(\mathbf{w})$ becomes a better approximation of the true posterior distribution $p(\mathbf{w} \mid \mathcal{D})$, ultimately improving the accuracy of approximate inference.

Task 10 - Deep Learning for Control

a.)

Answer:

Main Motivation for Inverse Reinforcement Learning

The main motivation for Inverse Reinforcement Learning (IRL) is to infer the reward function of an environment by observing the behavior of an expert. In traditional reinforcement learning, the reward function is predefined, and the agent learns the optimal policy. In IRL, however, the goal is to learn the reward function from demonstrations of an expert, allowing the agent to mimic the expert's behavior without explicitly providing the reward function.

b.)

Answer:

Guided Cost Learning (GCL) Pseudocode

Algorithm 1 Guided Cost Learning Algorithm

```
1: **Initialize:** Randomly initialize policy network $\pi(\mathbf{a}|\mathbf{s})$ and reward network $r(\mathbf{s}, \mathbf{a})$
2: **For each episode:**
3:     Sample trajectory $\tau = (\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \ldots, \mathbf{s}_T)$ using the policy $\pi$
4:     Compute the total cost of the trajectory $C(\tau) = \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$
5:     **Optimize the reward network:**
6:         Use GCL loss to update the reward function $r(\mathbf{s}, \mathbf{a})$ based on the observed expert demonstrations
7:     **Optimize the policy network:**
8:         Update the policy network $\pi$ using gradients from the reward network:
9:             $\nabla_\pi J(\pi) = \nabla_\pi \log \pi(\mathbf{a}_t|\mathbf{s}_t) \cdot \nabla_\pi r(\mathbf{s}_t, \mathbf{a}_t)$
```

c.)

Answer:

Generative Adversarial Imitation Learning (GAIL) Pseudocode

Algorithm 2 Generative Adversarial Imitation Learning (GAIL) Algorithm

```
1: **Initialize:** Randomly initialize policy network $\pi(\mathbf{a}|\mathbf{s})$ and discriminator network $D(\mathbf{s}, \mathbf{a})$
2: **For each episode:**
3:     Sample trajectory $\tau = (\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \ldots, \mathbf{s}_T)$ using the policy $\pi$
4:     **Train discriminator network $D$:**
5:         Update the discriminator using expert demonstrations $\tau_E$ and generated trajectories $\tau_\pi$:
6:             $L_D = \mathbb{E}_{\tau\sim\tau_E}[\log D(\mathbf{s}_t, \mathbf{a}_t)] + \mathbb{E}_{\tau\sim\tau_\pi}[\log(1 - D(\mathbf{s}_t, \mathbf{a}_t))]$
7:     **Train policy network $\pi$:**
8:         Update the policy network using the discriminator $D$:
9:             $L_\pi = \mathbb{E}_{\tau_\pi}[\log \pi(\mathbf{a}_t|\mathbf{s}_t) \cdot D(\mathbf{s}_t, \mathbf{a}_t)]$
```

Exam 2018

Task 2 - Data Efficiency

Answer:

Training Neural Networks with Less Data

To train neural networks effectively with less data, common methods include:

1. **Data Augmentation**: Generate additional training samples by applying transformations such as flipping, rotating, cropping, and changing brightness.
2. **Transfer Learning**: Use a pre-trained network on a similar task and fine-tune it on the smaller dataset.
3. **Regularization Techniques**: Apply dropout or weight decay to prevent overfitting and improve generalization.
4. **Few-shot Learning**: Use techniques like meta-learning to enable the model to learn from very few examples.
5. **Synthetic Data Generation**: Create artificial data through simulations or generative models to expand the training set.

Task 3 - Self-supervised learning

Answer:

Using Assumptions for Self-Supervised Learning

Self-supervised learning leverages inherent assumptions or constraints in the data to generate pseudo-labels, enabling training without explicit annotations. For depth and camera motion estimation, some common assumptions include:

1. **Photometric Consistency:** Assume that pixel intensities remain consistent between frames of a video, enabling depth and motion estimation by minimizing pixel reconstruction errors.
2. **Epipolar Geometry:** Utilize geometric constraints between two views of the same scene to infer depth and camera motion.
3. **Temporal Smoothness:** Assume that motion changes smoothly over consecutive frames, aiding in regularizing predictions.
4. **Object Boundaries:** Assume that depth discontinuities align with object edges detected in the scene, improving depth map estimation.

These assumptions act as surrogate supervision, allowing deep neural networks to be trained effectively with minimal or no annotated data.

Task 4 - Tracking

a.)

Answer:

Tracking Network Design and Training

Design

- **Base Network:** Use a pretrained object detection network as the backbone and include a *shared base network* that learns general features for tracking.
- **Specific Layers:** Attach a separate *last layer* for each training video-object pair to specialize for specific objects during training.
- **Input Representation:** Input consists of random crops from video frames around the predicted target location. These crops are classified as *target* or *not target* based on IoU thresholds with the ground truth bounding box.

Training

1. **Two-stage Training Process:**
    - Train the *shared base network* and video-specific last layers on a dataset of videos with bounding box annotations.
    - Optimize shared features across all videos while specializing last layers for each video-object pair.
2. **Loss Function:** Use a classification loss (e.g., binary cross-entropy) to differentiate *target* crops from *background* crops.

Inference

1. **Initialize the Tracker:**
    - Fix the *shared base network*.
    - Attach a new last layer for the target object and train it using the first frame and sampled crops.
2. **Tracking Process:** For each subsequent frame:
    - Extract features using the shared base network.
    - Classify crops using the initialized last layer to predict the target's new location.

This method integrates general feature extraction with video-specific specialization, enabling accurate and adaptive object tracking throughout the video.

b.)

Answer:

Problem with General Object Tracking Networks

A typical issue with general object tracking networks is **drift**—gradual loss of the target due to confusion with similar objects or accumulated errors. For example, the tracker might mix up visually similar objects, such as one person with another. This can occur because:

- Objects are treated as the same class in the pretrained detection network (when transfer learning is used).
- Few similar objects exist in each frame during training, reducing discrimination ability.

Alleviation Strategies

1. **Training with Similar Objects:** Explicitly include visually similar objects as object/non-object samples during training to improve discrimination.
2. **Distraction-Aware Inference:** Integrate mechanisms to account for and handle distractors during inference.
3. **Online Update Mechanism:** Adapt the last layer of the network using new frames during tracking to better distinguish between similar objects.
4. **Re-detection Modules:** Employ re-detection to recover from tracking failures when confusion arises.

These methods enhance robustness, enabling the tracker to maintain accuracy in challenging conditions and prevent target confusion.

Task 5 - 3D Segmentation

Answer:

Scenarios Where 3D Deep Learning Techniques Work Better than Multi-View Segmentation

While multi-view segmentation techniques often deliver strong results in 3D image datasets, there are scenarios where other 3D deep learning techniques can perform better:

- **Volumetric Data with Rich Spatial Information:** For 3D point clouds or voxel grids, 3D Convolutional Neural Networks (CNNs) can directly capture spatial context across the entire volume, leading to better performance compared to multi-view methods, which may lose fine-grained details from a single perspective.

- **Unstructured Point Cloud Data:** In cases where data is in the form of unstructured 3D points (such as LIDAR scans), methods like PointNet or PointNet++ are specifically designed to handle unordered and irregularly spaced points, making them more effective than multi-view segmentation, which assumes a structured set of views.
- **Complex Object Shape Recognition:** For 3D object segmentation in complex scenes, techniques like 3D U-Net or other volumetric methods can leverage the entire 3D space of an object, providing more accurate segmentations of intricate shapes that may not be well-represented in a limited set of 2D views.
- **Small Object Detection:** When objects are small or occupy a small portion of the scene, 3D methods that can process local context more efficiently (such as 3D CNNs) can better capture small features compared to multi-view methods, which may dilute the information across multiple perspectives.
- **Occlusion Handling:** For scenes with occluded objects, 3D methods can infer the structure of occluded regions by leveraging volumetric information, whereas multi-view segmentation may struggle due to missing information from certain perspectives.

In these cases, 3D-specific techniques that work directly on volumetric data or unstructured points offer significant advantages over multi-view segmentation.

Task 6 - Optimization

Answer:

Why Not Use Second-Order Derivatives in Neural Network Optimization?

While second-order derivatives (curvature information) can theoretically improve the optimization process by providing more accurate updates to the parameters, there are several reasons why we typically do not calculate them in practice:

- **High Computational Cost:** Calculating second-order derivatives requires computing the Hessian matrix (the matrix of second derivatives), which is computationally expensive. For large neural networks with millions of parameters, the computation and storage of the Hessian is prohibitively expensive and can be very slow.
- **Memory Requirements:** The Hessian matrix grows quadratically with the number of parameters. For deep neural networks, this results in immense memory requirements, making it infeasible to store and compute the second-order derivatives in practice.
- **Complexity in Handling Large Networks:** Neural networks, especially deep ones, have many parameters, and the curvature (second-order) information is often highly sparse. Storing and utilizing this information efficiently is challenging, and methods like full-batch second-order optimization are not scalable to modern deep networks.

- **Inaccuracies in Empirical Hessians:** In practice, computing second-order information from the data can be inaccurate, especially with noisy data or when the network has many parameters. This can lead to poor approximations and ineffective optimization.
- **Alternative Methods:** There are effective alternatives to second-order methods, such as first-order methods (like SGD) and approximations like Adam or L-BFGS. These methods are much more computationally efficient and still achieve good performance without the need for expensive second-order derivatives.

In summary, while second-order methods have the potential to improve optimization, their computational cost, memory requirements, and implementation complexity make them impractical for large-scale neural networks. First-order methods like stochastic gradient descent are preferred for their efficiency and scalability.

Task 7 - Batch Normalisation

Answer:

Batch Normalization and Running Averages

During training, batch normalization (BN) computes the mean and variance of each mini-batch to normalize activations. However, during inference, we use precomputed statistics since mini-batch statistics are not available.

The two common approaches are:

- **Stored Running Averages:** A moving average of the mean and variance is maintained during training and used during inference to normalize activations.
- **Global Statistics:** The mean and variance computed over the entire training set are used during inference.

These approaches ensure stable normalization during inference and avoid issues with mini-batch variability.

Task 8 - Bidirectional RNNs

Answer:

The motivation behind bidirectional RNNs is to capture context from both past and future input sequences. By processing the input data in both forward and backward directions, bidirectional RNNs can leverage information from both temporal ends, improving performance for tasks where future context is as important as past context, such as in sequence labeling or machine translation.

Task 9 - External memory

Answer:

Content-Based Addressing in Memory Networks

In content-based addressing, we match a query vector $\mathbf{q}$ against the key vectors $\mathbf{k}_1, \mathbf{k}_2, \mathbf{k}_3$ of the memory cells M_1, M_2, M_3. The matching scores are denoted as $\alpha_1, \alpha_2, \alpha_3$, respectively. The procedure for returning a d-dimensional vector with memory contents is as follows:

Hard Addressing

In hard addressing, we select the memory cell corresponding to the maximum matching score. The steps are:

$$i = \arg\max(\alpha_1, \alpha_2, \alpha_3)$$

Return the memory content M_i associated with the selected memory cell M_i.

Soft Addressing

In soft addressing, we compute a weighted sum of the memory contents based on the matching scores. The output vector is:

$$\mathbf{m} = \sum_{i=1}^{3} \alpha_i M_i$$

where α_i are the matching scores, typically normalized to sum to 1.

Task 10 - RL value functions

Answer:

To calculate the state-value function $v^\pi(s)$, we use the following formula based on the policy π:

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) \cdot q^\pi(s, a)$$

Given:

$$\pi(a_1|s) = 0.2, \quad \pi(a_2|s) = 0.8, \quad q^\pi(s, a_1) = -10, \quad q^\pi(s, a_2) = 10$$

We can substitute these values into the formula:

$$v^\pi(s) = \pi(a_1|s) \cdot q^\pi(s, a_1) + \pi(a_2|s) \cdot q^\pi(s, a_2)$$

$$v^\pi(s) = 0.2 \cdot (-10) + 0.8 \cdot 10$$

$$v^\pi(s) = -2 + 8 = 6$$

Thus, the state-value function is $v^\pi(s) = 6$.

Task 11 - RL policy gradient

Answer:

The gradient policy update rule:

$$\theta \leftarrow \theta + \alpha \sum_{t=1}^{\tau} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$$

can be interpreted as follows:

- $\nabla_\theta \log \pi_\theta(a_t|s_t)$ indicates the direction in which the probability of taking action a_t in state s_t increases the most.
- If the return G_t is positive, this term increases the probability of the taken action, reinforcing successful actions.
- If the return G_t is negative, the term decreases the probability of the action, discouraging unsuccessful actions.
- Thus, the policy is updated to favor actions that lead to good outcomes and avoid those leading to bad ones.

Task 12 - Word Embedding

Answer

Word Vector Representation and Softmax Output

Let $\mathbf{x} \in \mathbb{R}^{V \times 1}$ be the one-hot encoded vector of the input word, where V is the vocabulary size. The word vector $\mathbf{h} \in \mathbb{R}^{d \times 1}$ is obtained by applying a weight matrix W to $\mathbf{x}$, i.e.,

$$\mathbf{h} = W^T \mathbf{x}.$$

Next, we compute $\mathbf{z} = U\mathbf{h}$, where U is another weight matrix. The elements of $\mathbf{z}$ represent scores for each possible output word. The probability of selecting the output word $w_o = y$ is then given by the softmax function:

$$P(w_o = y) = \frac{\exp(z(y))}{\sum_{y=1}^{V} \exp(z(y))},$$

where $z(y)$ is the y-th element of $\mathbf{z}$. The i-th row of the matrix W corresponds to the word vector representation of the i-th word in the vocabulary.

Task 13 - Loss Functions

Answer:

Equivalence of Negative Sample Loss (NSL) and Noise Contrastive Estimation (NCE)

The Negative Sample Loss (NSL) is equivalent to Noise Contrastive Estimation (NCE) when the noise distribution is uniform and the data distribution is modeled using a softmax function. Both methods aim to distinguish between real data samples and negative (noise) samples, and they become equivalent under the assumption of uniform noise.

Task 14 - Bayesian deep learning

Answer:

General Closed Forms

Maximum Likelihood (ML)

$$\hat{w}_{ML} = \arg\max_{w} P(D|w)$$

or equivalently in terms of the log-likelihood:

$$\hat{w}_{ML} = \arg\max_{w} \log P(D|w)$$

Maximum A-Posteriori (MAP)

$$\hat{w}_{MAP} = \arg\max_{w} \left(\log P(D|w) + \log p(w)\right)$$

Bayesian Deep Learning (Bayesian Inference)

$$p(w|D) = \frac{p(D|w)p(w)}{p(D)}$$

where the marginal likelihood $p(D)$ is given by:

$$p(D) = \int p(D|w)p(w)dw$$

and the expected value of a function $f(w)$ under the posterior is:

$$\mathbb{E}[f(w)] = \int f(w)p(w|D)dw$$

Task 15 - Evidence Lower Bound

Answer:

The first term encourages data to be explained correctly with model parameters drawn from the auxiliary distribution. The second term requires that the distance between the auxiliary distribution and the posterior distribution would be minimized.

Task 16 - Guided Cost Learning

Back-propagation Through the Cost Function in Guided Cost Learning (GCL)

In Guided Cost Learning (GCL), we back-propagate through the neural network implementing the cost function by computing gradients for both expert demonstrations and system-generated trajectories. Specifically, we back-propagate a positive value for expert demonstrations and a negative value for system-generated trajectories.

Basic Steps for Calculating Back-propagated Quantities

1. **Initialize Cost Function**: Start by initializing the cost function c_θ, where θ represents the parameters of the cost function. This can be done with random or pre-defined values.
2. **Initialize Policy**: Initialize the policy network q_0, which will generate the system trajectories.
3. **Gather Expert Demonstrations**: Collect the expert demonstration data D_{demo}, which consists of the trajectories observed from the expert.
4. **Run Policy and Gather Trajectories**: Run the policy q_0 to generate system trajectories D_{samp}.
5. **Feed Expert Demonstration through Cost Function**: Pass the expert trajectories through the cost function and compute the backpropagation term as $\frac{1}{N}$, where N is the number of expert demonstration trajectories.
6. **Feed System Trajectories through Cost Function**: Pass the system-generated trajectories through the cost function and compute the backpropagation term as $-\frac{w_j}{Z}$, where w_j is the weight assigned to a trajectory and Z is a normalization factor.
7. **Update Cost Function**: Update the cost function c_θ based on the gradients computed from expert and system trajectories.
8. **Optimize Policy**: Optimize the policy network using reinforcement learning with the updated cost function, so that the policy improves based on the feedback from the cost function.
9. **Repeat**: Repeat steps 4 to 8 to continuously improve the policy and the cost function.

Task 17 - Dialogue Systems

Answer:

Issues in Dialogue System Training Addressable by Reinforcement Learning

1. **Handling Long-Term Dependencies**: Dialogue systems often struggle with maintaining context over long conversations. Reinforcement learning can help optimize responses based on long-term rewards, enabling the system to remember and refer to earlier parts of the conversation, improving the overall flow and coherence.
2. **Reward Optimization**: Defining a clear objective for dialogue systems can be challenging, as traditional supervised learning approaches rely on predefined labels. Reinforcement learning can help optimize a dialogue system by rewarding it for desirable actions, such as providing informative, engaging, or contextually appropriate responses, thereby aligning its behavior with the intended goals.
3. Non differentiable evaluation metrics relevant to longer term goals of the dialogue.

Reinforcement Learning Components in Dialogue Systems

- **Agent**: The dialogue system (e.g., chatbot).
- **State**: The current conversation context or history (e.g., previous user inputs, system responses).
- **Policy**: The strategy that the dialogue system uses to choose responses (e.g., selecting a response based on the current state).
- **Action**: The response or message generated by the dialogue system.
- **Reward**: A scalar value representing the quality of the system's response (e.g., user satisfaction, engagement, or successful completion of the task).

To reduce the variance of the gradient estimate in reinforcement learning-based dialog generation, we can use *reward normalization* or *baseline subtraction*. By subtracting a baseline from the reward, we reduce the variability of the reward signal, leading to more stable updates.

A common baseline is the average reward over the generated samples, denoted as b.

The modified equation would be:

$$\nabla_\theta L(\theta) \approx (r(w_s) - b)\nabla_\theta \log p_\theta(w_s)$$

Where b is the baseline (e.g., average reward) used to reduce the variance of the gradient estimate. This helps make the training more stable and efficient.

Task 18 - Sequence Modelling

Answer:

Maximum Path Lengths in Sequence Modelling:

- **Recurrent Network (RNN):**
  The maximum path length in an RNN is $O(n)$, where n is the sequence length, as each step depends on the previous one.
- **Stacked Convolution Network (Contiguous Kernels):**
  For a stacked convolutional network with kernel size k, the maximum path length is $O(n/k)$, as the kernel slides over the sequence, and there are approximately n/k possible applications.
- **Self-Attention Network:**
  In a self-attention network, each vector can attend to all others, so the maximum path length is $O(1)$, as attention is computed in parallel for all pairs of vectors.