Comprehensive Practice Problem Set

AIML ZG511: Deep Neural Networks

(Sessions 1 – 8)

Based on Course Handout & Past Exam Patterns

Instructions

This document contains practice questions and detailed solutions covering the first half of the course (Contact Sessions 1–8). These questions mimic the style of past Mid-Semester and Regular exams, focusing on:

- **Mathematical Derivations** (Backpropagation, Gradients)
- **Numerical Calculations** (Perceptron updates, CNN output sizes, Parameter counting)
- **Conceptual Understanding** (Activations, Regularization, Architectures)
- **Design Problems** (Logic gates, Network architecture)

1 Session 1 & 2: Introduction and Perceptron

Q1. Perceptron Learning Algorithm (Calculation)

Consider a perceptron with two inputs x_1, x_2 and a bias b. The initial weights are $w_1 = 0.5, w_2 = -0.5, b = 0.1$. The learning rate is $\eta = 0.2$. The activation function is a step function: $y = 1$ if $z \geq 0$, else $y = 0$. Perform one weight update step for the input sample $X = [1, 1]$ with target $T = 0$.

Solution:

1. **Compute Net Input (z):**

$$z = w_1x_1 + w_2x_2 + b = (0.5)(1) + (-0.5)(1) + 0.1 = 0.1$$

2. **Compute Prediction (y):** Since $z = 0.1 \geq 0$, the output $y = 1$.

3. **Compute Error:**

$$\text{Error} = T - y = 0 - 1 = -1$$

4. **Update Weights:**

$$\begin{aligned} w_1^{\text{new}} &= w_1 + \eta(T-y)x_1 = 0.5 + 0.2(-1)(1) = 0.5 - 0.2 = 0.3 \\ w_2^{\text{new}} &= w_2 + \eta(T-y)x_2 = -0.5 + 0.2(-1)(1) = -0.5 - 0.2 = -0.7 \\ b^{\text{new}} &= b + \eta(T-y) = 0.1 + 0.2(-1) = 0.1 - 0.2 = -0.1 \end{aligned}$$

Final Weights: $w_1 = 0.3, w_2 = -0.7, b = -0.1$.

Q2. Logic Gates (Design)

Design a single perceptron to implement the **NOR** gate logic. Specify the weights and bias. Inputs are bipolar $(+1/-1)$ and Output is bipolar $(+1/-1)$.

x_1	x_2	y
-1	-1	+1
-1	+1	-1
+1	-1	-1
+1	+1	-1

Solution: We need a line separating $(-1, -1)$ from the other three points. Let the equation be $w_1x_1 + w_2x_2 + b = 0$. By inspection or solving inequalities:

- For $(-1, -1) \to +1$: $-w_1 - w_2 + b > 0$
- For $(+1, +1) \to -1$: $w_1 + w_2 + b < 0$

Since the output is high only when both inputs are low, the weights must be negative. Let $w_1 = -1, w_2 = -1$.

- $(-1, -1) \to 1 + 1 + b > 0 \Rightarrow 2 + b > 0$
- $(-1, 1) \to 1 - 1 + b < 0 \Rightarrow b < 0$

We can choose $b = -1$. **Weights:** $w_1 = -1, w_2 = -1, b = -1$. Check: $(-1, -1) : 1 + 1 - 1 = 1 > 0$ (+1) ✓
$(-1, 1) : 1 - 1 - 1 = -1 < 0$ (-1) ✓
$(1, 1) : -1 - 1 - 1 = -3 < 0$ (-1) ✓

Q3. XOR Problem (Concept)

Explain why a single-layer perceptron cannot solve the XOR problem.

Solution: A single-layer perceptron defines a linear decision boundary (a straight line in 2D) given by $w^T x + b = 0$. The XOR problem input space consists of four points: $(0, 0)$ and $(1, 1)$ belonging to class 0, and $(0, 1)$ and $(1, 0)$ belonging to class 1. If you plot these points, it is geometrically impossible to draw a single straight line that separates the class 0 points from the class 1 points. Therefore, the XOR problem is not **linearly separable**, and a single perceptron cannot learn it.

Q4. Activation Functions (Concept)

Why is the step function generally not used in Deep Learning, and what replaced it?

Solution: The step function has a derivative of zero everywhere (except at $z = 0$ where it is undefined). This makes it impossible to use **Gradient Descent** or Backpropagation, as the gradients would not flow through the network to update weights. It has been replaced by differentiable non-linear functions like **Sigmoid**, **Tanh**, and **ReLU** (Rectified Linear Unit), which allow for the computation of gradients.

Q5. Biological vs. Artificial Neurons (Concept)

List two similarities and two differences between Biological Neurons and Artificial Neurons.

Solution: Similarities: 1. Both receive inputs from multiple sources (Dendrites vs. Input layer/Previous layer). 2. Both sum the inputs and fire/activate based on a threshold or activation function (Soma/Axon vs. Activation). **Differences:** 1. Biological neurons signal using discrete spikes (Action Potentials), while artificial neurons typically output continuous numerical values. 2. Biological synapses effectively have separate processing capabilities and are chemical in nature; artificial weights are simple scalar multipliers.

Q6. Perceptron Convergence (Theory)

Under what condition is the Perceptron Learning Algorithm guaranteed to converge?

Solution: The Perceptron Learning Algorithm is guaranteed to converge and find a solution in a finite number of steps if and only if the training dataset is **linearly separable**. If the data is not linearly separable, the weights will oscillate forever.

Q7. Weight Initialization (Concept)

Why should weights not be initialized to zero?

Solution: If all weights are initialized to zero, every neuron in a hidden layer will receive the same input and compute the same output. Consequently, during backpropagation, they will all receive the same gradient updates and evolve identically. This **symmetry** prevents the network from learning diverse features.

Q8. Bias Term (Concept)

What is the geometric interpretation of the bias term b in $w_1x_1 + w_2x_2 + b = 0$?

Solution: Geometrically, the bias term shifts the decision boundary (the line or hyperplane) away from the origin. Without a bias term ($b = 0$), the decision boundary is forced to pass through the origin $(0, 0)$, which severely restricts the model's ability to fit data that is not centered at the origin.

Q9. Epoch vs Batch (Concept)

Differentiate between an Epoch and a Batch in training.

Solution:

- **Epoch:** One complete pass of the learning algorithm through the entire training dataset.
- **Batch:** A subset of the training dataset used to compute the gradient and update weights once.

If a dataset has 1000 samples and batch size is 100, one epoch consists of 10 weight updates (iterations).

Q10. Learning Rate (Concept)

What happens if the learning rate η is too large or too small?

Solution:

- **Too Large:** The algorithm may overshoot the global minimum, causing the loss to diverge or oscillate efficiently without converging.
- **Too Small:** The convergence will be extremely slow, and the model might get stuck in a local minimum (though less of an issue with convex problems).

2 Session 3 & 4: Linear Neural Networks

Q11. MSE Derivative (Derivation)

Given the loss function for a single training example $L = \frac{1}{2}(\hat{y} - y)^2$ where $\hat{y} = w^T x + b$. Derive the gradient $\frac{\partial L}{\partial w}$.

Solution: Using the Chain Rule:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w}$$

1. $\frac{\partial L}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}} \left(\frac{1}{2}(\hat{y} - y)^2 \right) = (\hat{y} - y)$
2. $\frac{\partial \hat{y}}{\partial w} = \frac{\partial}{\partial w}(w^T x + b) = x$

Substituting back:

$$\frac{\partial L}{\partial w} = (\hat{y} - y)x$$

Q12. Binary Cross Entropy (Derivation)

Derive the derivative of the Binary Cross Entropy loss $L = -[y \ln(\hat{y}) + (1 - y) \ln(1 - \hat{y})]$ with respect to the output $\hat{y}$.

Solution:

$$L = -y \ln(\hat{y}) - (1 - y) \ln(1 - \hat{y})$$

$$\frac{\partial L}{\partial \hat{y}} = -y \left(\frac{1}{\hat{y}} \right) - (1 - y) \left(\frac{1}{1 - \hat{y}} \cdot (-1) \right)$$

$$\frac{\partial L}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{1 - y}{1 - \hat{y}}$$

$$\frac{\partial L}{\partial \hat{y}} = \frac{-y(1 - \hat{y}) + \hat{y}(1 - y)}{\hat{y}(1 - \hat{y})} = \frac{-y + y\hat{y} + \hat{y} - y\hat{y}}{\hat{y}(1 - \hat{y})} = \frac{\hat{y} - y}{\hat{y}(1 - \hat{y})}$$

Q13. Softmax Calculation (Calculation)

Given the logits (raw outputs) from a network $Z = [2.0, 1.0, 0.1]$, calculate the Softmax probabilities.

Solution: Softmax formula: $S(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ 1. Calculate exponentials: $e^{2.0} \approx 7.389$ $e^{1.0} \approx 2.718$ $e^{0.1} \approx 1.105$ 2. Sum of exponentials: $\Sigma = 7.389 + 2.718 + 1.105 = 11.212$ 3. Calculate probabilities: $P_1 = 7.389/11.212 \approx 0.659$ $P_2 = 2.718/11.212 \approx 0.242$ $P_3 = 1.105/11.212 \approx 0.099$ Check sum: $0.659 + 0.242 + 0.099 = 1.0$.

Q14. Stochastic vs Batch Gradient Descent (Concept)

Compare Stochastic Gradient Descent (SGD) and Batch Gradient Descent (BGD).

Solution:

- **Batch GD:** Computes the gradient using the *entire* dataset. Steps are smooth and deterministic but computationally expensive for large data.
- **Stochastic GD:** Computes the gradient using a *single* sample. It is much faster per iteration but the path to convergence is noisy (high variance).
- **Mini-Batch GD:** A compromise using k samples.

Q15. Overfitting (Concept)

In the context of linear regression, what does overfitting look like, and how can we prevent it?

Solution: In linear regression (specifically polynomial regression), overfitting occurs when the curve is too complex (high-degree polynomial) and passes through every noise point in the training data, resulting in wild oscillations. **Prevention:** 1. Regularization (L1 Lasso, L2 Ridge). 2. Using more training data. 3. Feature selection (reducing dimensionality).

Q16. Computational Graph (Graph/Concept)

Draw a computational graph for $J = (w \cdot x - y)^2$ and trace the backward pass for $\frac{\partial J}{\partial w}$.

Solution: Forward: 1. Node A: $u = w \cdot x$ 2. Node B: $v = u - y$ 3. Node C: $J = v^2$ **Backward:** 1. $\frac{\partial J}{\partial v} = 2v = 2(w \cdot x - y)$ 2. $\frac{\partial v}{\partial u} = 1 \implies \frac{\partial J}{\partial u} = \frac{\partial J}{\partial v} \cdot 1 = 2(w \cdot x - y)$ 3. $\frac{\partial u}{\partial w} = x \implies \frac{\partial J}{\partial w} = \frac{\partial J}{\partial u} \cdot x = 2(w \cdot x - y)x$

Q17. Sigmoid Derivative (Derivation)

Show that if $\sigma(z) = \frac{1}{1+e^{-z}}$, then $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

Solution: $\sigma(z) = (1 + e^{-z})^{-1}$ Using chain rule: $\sigma'(z) = -1(1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1+e^{-z})^2}$ Rewrite: $\frac{e^{-z}}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})} \cdot \frac{e^{-z}}{(1+e^{-z})}$ Note that $\frac{e^{-z}}{1+e^{-z}} = \frac{1+e^{-z}-1}{1+e^{-z}} = 1 - \frac{1}{1+e^{-z}} = 1 - \sigma(z)$ Thus, $\sigma'(z) = \sigma(z)(1 - \sigma(z))$.

Q18. Categorical Cross Entropy (Calculation)

Given target vector $y = [0, 1, 0]$ (one-hot) and predicted probabilities $\hat{y} = [0.1, 0.7, 0.2]$. Calculate the Categorical Cross Entropy loss.

Solution: $L = -\sum_i y_i \ln(\hat{y}_i)$ $L = -[0 \cdot \ln(0.1) + 1 \cdot \ln(0.7) + 0 \cdot \ln(0.2)]$ $L = -\ln(0.7) \approx -(-0.356) = 0.356$

Q19. Output Dimensions (Concept)

If you have 10 input features and perform multi-class classification for 5 classes, what are the dimensions of the weight matrix W and bias vector b?

Solution: Inputs $x \in \mathbb{R}^{10}$, Output $z \in \mathbb{R}^5$. Equation: $z = W^T x + b$ (or $Wx + b$ depending on convention). Assuming standard $z = W^T x + b$ where output is a row, or $z = xW + b$ in frameworks: If input is column vector (10×1): W must be 5×10 to produce 5×1. Bias b is 5×1. (Common notation: W is 10×5 if x is row vector 1×10).

Q20. Vanishing Gradient (Concept)

Why is the sigmoid function problematic for deep networks?

Solution: The maximum derivative of the sigmoid function is 0.25 (at $z = 0$). For large positive or negative inputs, the derivative is close to 0 (saturating). When chaining many layers, these small gradients multiply ($\approx 0.25 \times 0.25 \dots$), causing the gradient to vanish exponentially. This stops weights in early layers from updating.

3 Session 5: Deep Feedforward Neural Networks (MLP)

Q21. Parameter Counting (Calculation)

Consider an MLP with:

- Input layer: 10 neurons
- Hidden Layer 1: 20 neurons
- Hidden Layer 2: 15 neurons
- Output Layer: 5 neurons

Calculate the total number of trainable parameters (weights + biases).

Solution:

1. Input $\to$ H1: weights $(10 \times 20) + \text{bias } 20 = 200 + 20 = 220$
2. H1 $\to$ H2: weights $(20 \times 15) + \text{bias } 15 = 300 + 15 = 315$
3. H2 $\to$ Output: weights $(15 \times 5) + \text{bias } 5 = 75 + 5 = 80$

Total Parameters $= 220 + 315 + 80 = 615$.

Q22. XOR with MLP (Design)

Design a minimal MLP (specify nodes and weights) to solve the XOR problem. XOR: $(0,0) \to 0, (0,1) \to 1, (1,0) \to 1, (1,1) \to 0$.

Solution: Structure: 2 Inputs, 2 Hidden neurons (ReLU or Step), 1 Output. Let hidden neurons be h_1 (OR gate logic) and h_2 (NAND gate logic). Output y computes AND logic on h_1 and h_2. **Hidden Layer:** 1. h_1 (OR): weights $[1,1]$, bias -0.5. Output 1 if $x_1 + x_2 \geq 0.5$. 2. h_2 (NAND): weights $[-1,-1]$, bias 1.5. Output 1 if $-x_1 - x_2 + 1.5 \geq 0$. **Output Layer:** 3. y (AND): weights $[1,1]$, bias -1.5 taking inputs from h_1, h_2. **Verification for (1,1):** h_1: $1 + 1 - 0.5 = 1.5 \to 1$. h_2: $-1 - 1 + 1.5 = -0.5 \to 0$. y: $1(1) + 1(0) - 1.5 = -0.5 \to 0$. Correct.

Q23. Backpropagation Update (Calculation)

A simple network: Input $x = 1$, Hidden h (weight w_1, sigmoid), Output y (weight w_2, linear). Target $t = 1$. Initial $w_1 = 0.5, w_2 = 1.0$. Learning rate $\eta = 1$. Calculate w_1 update.

Solution: Forward: $z_1 = w_1 x = 0.5(1) = 0.5$ $h = \sigma(0.5) \approx 0.622$ $y = w_2 h = 1.0(0.622) = 0.622$ Loss $L = \frac{1}{2}(y-t)^2 = \frac{1}{2}(0.622-1)^2$ Backward: $\frac{\partial L}{\partial y} = y - t = -0.378$ $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} h = -0.378 \times 0.622 = -0.235$ $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \cdot w_2 \cdot \sigma'(z_1) \cdot x$ $\sigma'(0.5) = 0.622(1 - 0.622) = 0.235$ $\frac{\partial L}{\partial w_1} = (-0.378)(1.0)(0.235)(1) = -0.089$ New $w_1 = 0.5 - (1)(-0.089) = 0.589$.

Q24. ReLU Benefits (Concept)

Why is ReLU preferred over Tanh/Sigmoid for hidden layers?

Solution: 1. **No Vanishing Gradient (Positive side):** The gradient is exactly 1 for all $x > 0$, allowing deep networks to train. 2. **Computational Efficiency:** Computing $\max(0, x)$ is much faster than exponentials (e^x). 3. **Sparsity:** It outputs true zeros for negative inputs, leading to sparse activations which can help generalization.

Q25. Universal Approximation Theorem (Theory)

State the Universal Approximation Theorem.

Solution: A feedforward neural network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$ to arbitrary accuracy, provided the activation function is non-constant, bounded, and continuous (non-linear).

Q26. Dropout (Concept)

How does Dropout act as a regularization technique?

Solution: Dropout randomly sets a fraction of neurons to zero during each training iteration. This prevents neurons from co-adapting (relying too much on specific other neurons) and forces the network to learn robust, redundant features. It essentially trains an ensemble of sub-networks.

Q27. Initialization Methods (Concept)

Why is "He Initialization" used with ReLU?

Solution: ReLU activations are not centered at zero and halves the variance of the output (since half inputs become 0). He initialization sets the variance of weights to $2/n$ (where n is inputs) to compensate for this, ensuring the signal magnitude remains consistent across layers.

Q28. Depth vs Width (Concept)

What is the benefit of a "Deep" network over a "Wide" network?

Solution: Deep networks can learn hierarchical features (edges $\rightarrow$ shapes $\rightarrow$ objects) and compositional functions more efficiently. A deep network can represent certain complex functions with polynomial parameter count, whereas a shallow (wide) network might require exponentially many parameters to represent the same function.

Q29. Forward Propagation Vectorization (Concept)

Write the vectorized equation for the activation of layer l given inputs from layer $l-1$.

Solution:

$$A^{[l]} = g(Z^{[l]}) = g(W^{[l]}A^{[l-1]} + b^{[l]})$$

Where $W^{[l]}$ is the weight matrix, $A^{[l-1]}$ is the activation vector from previous layer, and g is the activation function.

Q30. Momentum (Concept)

How does Momentum help SGD?

Solution: Momentum accumulates the past gradients (velocity) to smooth out the updates. It helps dampen oscillations in steep valleys and accelerates traversal through flat plateaus, leading to faster convergence compared to standard SGD.

4 Session 6: Convolutional Neural Networks

Q31. Output Dimension Calculation (Calculation)

Input image size: $32 \times 32 \times 3$. Convolution Layer: 10 filters of size 5×5, Stride $S = 1$, Padding $P = 0$. Calculate the output volume dimensions.

Solution: Formula: $O = \frac{W-F+2P}{S} + 1$ $W = 32, F = 5, P = 0, S = 1$. $O = \frac{32-5+0}{1} + 1 = 27 + 1 = 28$. Since there are 10 filters, the depth is 10. **Output:** $28 \times 28 \times 10$.

Q32. CNN Parameter Counting (Calculation)

Using the configuration in Q31, calculate the total number of trainable parameters.

Solution: Each filter has dimensions $5 \times 5 \times 3$ (depth matches input depth). Parameters per filter $= (5 \times 5 \times 3) + 1(\text{bias}) = 75 + 1 = 76$. Total filters $= 10$. Total parameters $= 76 \times 10 = 760$.

Q33. Pooling Layer (Calculation)

Input: 4×4 matrix.

$$\begin{bmatrix} 1 & 2 & 5 & 6 \\ 3 & 4 & 7 & 8 \\ 1 & 1 & 2 & 2 \\ 0 & 0 & 9 & 1 \end{bmatrix}$$

Apply Max Pooling with 2×2 filter and Stride 2.

Solution: Divide into regions: Top-Left: $\max(1, 2, 3, 4) = 4$ Top-Right: $\max(5, 6, 7, 8) = 8$ Bot-Left: $\max(1, 1, 0, 0) = 1$ Bot-Right: $\max(2, 2, 9, 1) = 9$ Result:

$$\begin{bmatrix} 4 & 8 \\ 1 & 9 \end{bmatrix}$$

Q34. Convolution vs Fully Connected (Concept)

Why are CNNs preferred over MLPs for image data? (Two reasons)

Solution: 1. **Parameter Sharing:** The same filter (weights) is used across the entire image, drastically reducing parameters compared to dense connections. 2. **Translation Invariance (Equivariance):** A feature (e.g., an edge) detected in the top-left corner is recognized by the same filter in the bottom-right corner. 3. **Local Connectivity:** CNNs exploit the spatial structure of images (neighboring pixels are related).

Q35. 1x1 Convolution (Concept)

What is the purpose of a 1×1 convolution?

Solution: It preserves the spatial dimensions ($H \times W$) but changes the depth (number of channels). It is used for dimensionality reduction (reducing filter count before expensive operations) and adding non-linearity (via activation) without changing the receptive field.

Q36. Receptive Field (Concept)

Define Receptive Field.

Solution: The receptive field is the specific region of the input space (image) that a particular feature (neuron) in a deeper layer is looking at or is affected by. As we go deeper, the receptive field increases, allowing the network to capture global context.

Q37. Padding Types (Concept)

Difference between 'Valid' and 'Same' padding.

Solution:

- **Valid:** No padding ($P = 0$). The spatial dimensions reduce after convolution.
- **Same:** Padding is added such that the output spatial dimension equals the input spatial dimension (assuming stride=1).

Q38. Channels (Concept)

If an input has 3 channels (RGB) and we apply a convolution with 64 filters, what is the depth of the output feature map?

Solution: The output depth is determined solely by the number of filters in the convolution layer. Output Depth = 64.

Q39. Stride Impact (Concept)

How does increasing stride affect the output and information?

Solution: Increasing stride (e.g., from 1 to 2) reduces the spatial dimensions of the output (downsampling). It reduces computational cost but may result in loss of fine-grained spatial information.

Q40. LeNet Architecture (Fact)

What was the primary innovation and application of LeNet-5?

Solution: LeNet-5 introduced the standard CNN architecture (Conv $\rightarrow$ Pool $\rightarrow$ Conv $\rightarrow$ Pool $\rightarrow$ FC). It was successfully applied to handwritten digit recognition (MNIST) on checks and mail.

5 Session 7: Deep CNN Architectures

Q41. AlexNet vs LeNet (Concept)

Name two major improvements AlexNet had over LeNet.

Solution: 1. **ReLU Activation:** Solved vanishing gradients, allowing faster training. 2. **Dropout:** Used in FC layers to prevent overfitting. 3. **Data Augmentation:** Used to artificially increase dataset size. 4. **GPU Utilization:** Trained on dual GPUs.

Q42. VGG Philosophy (Concept)

What is the key design philosophy of VGGNet regarding filter sizes?

Solution: VGG replaced large filters (e.g., 11×11, 5×5) with stacks of small 3×3 filters. A stack of two 3×3 convs has the same receptive field as a 5×5 but with fewer parameters and more non-linearities (activations), improving discriminative power.

Q43. ResNet Skip Connections (Derivation/Concept)

Write the equation for a Residual Block and explain how it solves the degradation problem.

Solution: Equation: $y = \mathcal{F}(x, \{W_i\}) + x$ Where x is input, $\mathcal{F}$ is the residual mapping (layers), and $+x$ is the skip connection. **Explanation:** It allows the gradient to flow directly through the identity shortcut ($+x$) during backpropagation. Even if the weights in $\mathcal{F}$ vanish, the gradient of x is 1, ensuring earlier layers still receive gradient updates. This allows training of very deep networks (100+ layers).

Q44. GoogLeNet Inception Block (Concept)

What is the motivation behind the Inception module?

Solution: Instead of choosing a single filter size (3×3 or 5×5) or pooling at each layer, the Inception module performs them all in parallel and concatenates the results. This allows the network to capture features at multiple scales simultaneously. It also uses 1×1 convolutions to reduce dimensionality before expensive 3×3 and 5×5 convolutions.

Q45. Global Average Pooling (Concept)

Why is Global Average Pooling (GAP) used instead of Flattening in modern architectures like NiN or ResNet?

Solution: GAP takes the average of each feature map to produce a single number per channel. 1. Drastically reduces parameter count (no heavy FC layers at the end). 2. Reduces overfitting. 3. More robust to spatial translations.

Q46. Parameter Calculation AlexNet (Calculation)

The first layer of AlexNet has input $227 \times 227 \times 3$, 96 filters, size 11×11, stride 4. Calculate output size and parameters.

Solution: Output Size: $\frac{227-11}{4} + 1 = \frac{216}{4} + 1 = 54 + 1 = 55$. Output: $55 \times 55 \times 96$. Parameters: $(11 \times 11 \times 3 \times 96) + 96(\text{bias}) = 34848 + 96 = 34944$.

Q47. Network in Network (NiN) (Concept)

What is the "mlpconv" layer in NiN?

Solution: NiN replaces the standard convolution layer with a "micro-network" structure. Practically, this is implemented as a normal convolution followed by one or more 1×1 convolutions. This increases the non-linearity of the local patches.

Q48. Auxiliary Classifiers (Concept)

GoogLeNet used "Auxiliary Classifiers" connected to intermediate layers. Why?

Solution: They were used to inject gradients into the middle of the network during training to combat the vanishing gradient problem in the deep Inception architecture. They also provided regularization. (They were removed in inference).

Q49. Depth Evolution (Concept)

Order these networks by depth (shallowest to deepest): AlexNet, VGG-19, ResNet-152, LeNet-5.

Solution: LeNet-5 (<10 layers) $\rightarrow$ AlexNet (8 layers) $\rightarrow$ VGG-19 (19 layers) $\rightarrow$ ResNet-152 (152 layers).

Q50. Computational Cost (Concept)

Which is computationally more expensive (FLOPS): A 3×3 conv with 64 inputs/64 outputs, or a 1×1 conv with 64 inputs/64 outputs?

Solution: 3×3 is more expensive. Cost $3 \times 3 \approx 9 \cdot C_{in} \cdot C_{out} \cdot H \cdot W$ Cost $1 \times 1 \approx 1 \cdot C_{in} \cdot C_{out} \cdot H \cdot W$ The 3×3 is roughly 9 times more expensive.

6 Session 8: Transfer Learning

Q51. Transfer Learning Definition (Concept)

Define Transfer Learning in the context of Deep Learning.

Solution: Transfer Learning is a technique where a model developed for a task is reused as the starting point for a model on a second, related task. For example, using weights from a model trained on ImageNet (1000 classes) to initialize a model for detecting dog breeds.

Q52. Feature Extraction vs Fine Tuning (Concept)

Differentiate between ”Feature Extraction” and ”Fine Tuning”.

Solution:

- **Feature Extraction:** We freeze the convolutional base (weights do not change) and only train the new classifier (dense layers) on top. The base acts as a fixed feature extractor.
- **Fine Tuning:** We unfreeze some or all of the top layers of the convolutional base and train them jointly with the new classifier, usually with a very low learning rate, to adapt high-level features to the new task.

Q53. Small Dataset Strategy (Concept)

If you have a very small dataset similar to the pre-trained data (e.g., ImageNet), what strategy should you use?

Solution: Use **Feature Extraction** (Freeze base, train linear classifier). Reason: Since the dataset is small, fine-tuning might lead to overfitting. Since the data is similar, the pre-learned features are already relevant.

Q54. Large Different Dataset Strategy (Concept)

If you have a large dataset that is very different from the pre-trained data, what strategy should you use?

Solution: Train the model from scratch or **Fine-tune the entire network**. Reason: Since the data is large, overfitting is less of a concern. Since the domain is different, the original features might not be useful, so re-training effectively allows the model to learn new domain-specific features.

Q55. Freezing Layers (Concept)

What does it mean to ”freeze” a layer in code (e.g., Keras/PyTorch)?

Solution: It means setting the ‘trainable‘ attribute of the layer to ‘False‘. During back-propagation, the gradients for the weights in this layer are not computed or applied, so the weights remain constant (retaining the pre-trained knowledge).

Q56. Pre-training Benefits (Concept)

Why does pre-training help even if the tasks are slightly different?

Solution: Early layers in CNNs learn low-level features like edges, curves, and textures. These features are **universal** to almost all visual tasks. Transferring these weights saves the model from relearning basic vision, leading to faster convergence and better performance.

Q57. Learning Rate in Fine Tuning (Concept)

Why should we use a lower learning rate during fine-tuning?

Solution: To prevent destroying the pre-trained weights. High learning rates might make large updates that wreck the useful features learned during the extensive pre-training phase. A small rate allows for gentle adaptation.

Q58. Catastrophic Forgetting (Concept)

What is Catastrophic Forgetting in the context of sequential transfer learning?

Solution: It is the phenomenon where a neural network forgets previously learned information (Task A) upon learning new information (Task B). In standard transfer learning, we usually don't care about performance on Task A anymore, but it's a concern in continuous learning.

Q59. Bottleneck Features (Concept)

What are "Bottleneck Features"?

Solution: They are the output of the final convolutional layer (before the fully connected layers) of the pre-trained network. In Feature Extraction, we often pass our dataset through the base once, save these "bottleneck features", and then train a simple classifier on these saved features to save computational time.

Q60. Domain Shift (Concept)

Can transfer learning fail? Give an example of negative transfer.

Solution: Yes, if the source and target domains are too dissimilar (Negative Transfer). For example, transferring weights from a model trained on ImageNet (natural images) to classify Chest X-Rays (grayscale, specific medical textures) might not be optimal compared to simpler initializations, or might require significantly more fine-tuning, as the high-level features (eyes, fur) are irrelevant to X-rays.