Deep Learning

Final Project Defense Instructions

Deadline: End of Week 11

Overview

The final project defense represents the culmination of your work throughout this course. You are required to present a comprehensive overview of both your midterm and end-term projects, demonstrating your understanding of deep learning concepts and your ability to apply them to practical problems.

Defense Components

The final defense consists of two parts: (1) a presentation showcasing your midterm and end-term projects, and (2) an oral examination covering key concepts from the course syllabus.

1 Presentation Requirements

1.1 Format and Length

Your presentation must be 10–15 slides in length. This should provide sufficient space to comprehensively cover both projects while maintaining focus and clarity. The presentation should be professionally formatted and include appropriate visualizations, diagrams, and results.

1.2 Content Structure

Your presentation should be organized to provide a coherent narrative of your learning journey throughout the course. The following structure is recommended:

Introduction (1–2 slides)

Begin with a brief overview that introduces both projects and explains how they relate to the course objectives. Highlight the main deep learning concepts and techniques you explored.

Midterm Project (4–6 slides)

Present your midterm project comprehensively. Include the problem statement and motivation, methodology and architecture design, implementation details and key technical decisions, experimental results with appropriate metrics and visualizations, and analysis of what worked well and what challenges you encountered.

End-Term Project (4–6 slides)

Present your end-term project with similar depth. Cover the problem formulation and objectives, architectural choices and integration of multiple course topics, implementation approach

and experimental design, results including ablation studies and comparisons, and insights gained from the project.

Integration and Learning Outcomes (1–2 slides)

Conclude by reflecting on how the two projects complemented each other, key concepts and skills you developed throughout the course, and how your understanding of deep learning evolved from midterm to end-term.

1.3 Technical Expectations

Your presentation should demonstrate technical depth and understanding. Include architectural diagrams showing model components and data flow, training curves and performance metrics with clear labels, ablation studies or comparative analyses, visualizations of learned representations (attention maps, embeddings, generated samples), and code snippets for critical implementation details where relevant.

Ensure all figures and tables are clearly labeled with captions and legends. Mathematical notation should be used appropriately to explain key concepts or formulations.

2 Oral Examination

Following your presentation, you will participate in an oral examination covering fundamental concepts from the course syllabus. This examination is designed to assess your theoretical understanding of deep learning principles and your ability to explain concepts clearly.

2.1 Examination Format

The oral examination will last approximately 10–15 minutes. You will be asked questions covering various topics from the course syllabus. Questions may range from basic definitions to more complex conceptual explanations and comparisons.

2.2 Topics for Examination

The examination may cover any topic from the course syllabus. Below is a comprehensive list of key concepts, definitions, and topics you should be prepared to discuss.

Week 1: Introduction to Deep Learning

Deep Learning: A subset of machine learning using neural networks with multiple layers to learn hierarchical representations of data. Distinguished from traditional machine learning by automatic feature learning.

Historical Evolution: From the Perceptron (1950s) through the AI winters to modern deep learning enabled by big data, GPU computing, and algorithmic advances (2010s onwards).

Linear Models vs Deep Models: Linear models learn linear decision boundaries ($y = Wx + b$), while deep models can learn complex non-linear mappings through composition of multiple layers with non-linear activations.

Biological Inspiration: Neural networks are loosely inspired by biological neurons. Artificial neurons receive weighted inputs, sum them, and apply activation functions, analogous to biological neuron firing.

Perceptron: A single-layer neural network that performs binary classification by computing a weighted sum of inputs plus a bias term, then applying an activation function: $y = f(w^T x + b)$.

Multi-Layer Perceptron (MLP): A feedforward neural network with one or more hidden layers between the input and output layers, capable of learning non-linear decision boundaries through layer composition.

Activation Functions: Non-linear functions applied to neuron outputs enabling networks to learn complex patterns. Common examples include sigmoid ($\sigma(x) = \frac{1}{1+e^{-x}}$), tanh ($\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$), and ReLU ($\text{ReLU}(x) = \max(0, x)$).

Major DL Applications: Computer vision (image classification, object detection), natural language processing (translation, sentiment analysis), speech recognition, robotics (perception and control), and more.

Week 2: Optimization & Training Deep Networks

Loss Functions: Functions that quantify the difference between predicted and actual outputs. Mean Squared Error (MSE) for regression: $\mathcal{L} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$. Cross-Entropy for classification: $\mathcal{L} = -\sum y_i \log(\hat{y}_i)$.

Gradient Descent: An optimization algorithm that iteratively updates parameters in the direction opposite to the gradient to minimize the loss function: $\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$.

Backpropagation: The algorithm for computing gradients of the loss function with respect to network parameters using the chain rule of calculus. It propagates error signals backward through the network layer by layer.

Stochastic Gradient Descent (SGD): Updates parameters using gradients computed on small random batches rather than the entire dataset, enabling faster training and better generalization.

Momentum: An optimization technique that accumulates a velocity vector in directions of persistent gradient reduction: $v_t = \beta v_{t-1} + \eta \nabla_\theta \mathcal{L}$, $\theta_t = \theta_{t-1} - v_t$. Helps overcome local minima and accelerates convergence.

Adam Optimizer: An adaptive learning rate method combining momentum and RMSProp. Maintains both first moment (mean) and second moment (variance) of gradients, automatically adjusting learning rates per parameter.

Vanishing Gradients: When gradients become extremely small during backpropagation through many layers, making it difficult to train deep networks. Common with sigmoid/tanh activations.

Exploding Gradients: When gradients become extremely large, causing unstable training and divergence. Can be mitigated using gradient clipping.

Weight Initialization: Proper initialization is critical for training. Xavier/Glorot initialization for sigmoid/tanh, He initialization for ReLU. Prevents vanishing/exploding gradients at initialization.

Normalization: Techniques like Batch Normalization that normalize layer inputs to stabilize training, enable higher learning rates, and reduce sensitivity to initialization.

Regularization: Techniques to prevent overfitting by constraining model complexity, including L2 regularization (weight decay), L1 regularization, and dropout.

Week 3: Convolutional Neural Networks (CNNs)

Convolution Operation: A mathematical operation that applies a learnable filter (kernel) across an input to extract local features. The output at position (i, j) is computed as: $(I*K)_{i,j} = \sum_m \sum_n I_{i+m,j+n} \cdot K_{m,n}$.

Kernels (Filters): Small matrices of learnable weights that slide across the input to detect specific patterns (edges, textures, shapes). Each kernel produces one feature map.

Stride: The step size by which the filter moves across the input. A stride of 1 means the filter moves one pixel at a time, while larger strides (e.g., 2) result in smaller output dimensions and reduced computation.

Padding: Adding zeros (or other values) around the input border to control output dimensions and preserve spatial information at boundaries. Valid padding uses no padding, while same padding maintains input dimensions.

Pooling: A downsampling operation that reduces spatial dimensions while retaining important features. Max pooling selects the maximum value in each window, while average pooling computes the mean. Provides translation invariance.

Feature Maps: The outputs produced by applying convolutional filters to input data. Each filter learns to detect specific patterns or features at different spatial locations.

LeNet: One of the earliest CNN architectures (1998) for digit recognition, consisting of convolutional layers, pooling layers, and fully connected layers.

AlexNet: The breakthrough CNN architecture (2012) that won ImageNet competition, introducing ReLU activations, dropout, and GPU training for deep networks.

VGG: An architecture emphasizing depth with small (3×3) convolutional filters stacked in sequences, demonstrating that network depth is crucial for performance.

CNN Inductive Biases: Built-in assumptions that help CNNs learn efficiently. Locality: nearby pixels are more related. Translation invariance: features are useful regardless of position. Weight sharing reduces parameters.

Week 4: Modern CNN Architectures & Transfer Learning

ResNet (Residual Networks): Introduced skip connections allowing gradients to flow directly through the network, enabling training of very deep networks (50-152+ layers). Uses residual blocks: $F(x) = H(x) - x$, learning residual mappings.

Skip Connections (Residual Connections): Direct pathways that add the input of a layer to its output: $y = F(x) + x$. Addresses vanishing gradient problem and enables identity mappings, allowing layers to learn refinements.

Inception: An architecture using parallel convolutions of different kernel sizes (1×1, 3×3, 5×5) concatenated together, capturing multi-scale features efficiently.

EfficientNet: A family of models that systematically scales network depth, width, and resolution using compound scaling, achieving state-of-the-art accuracy with fewer parameters.

Batch Normalization: Normalizes layer inputs by subtracting the batch mean and dividing by the batch standard deviation: $\hat{x} = \frac{x-\mu_B}{\sqrt{\sigma_B^2+\epsilon}}$, then applies learnable scale (γ) and shift (β) parameters. Stabilizes training and allows higher learning rates.

Transfer Learning: Using a model pre-trained on a large dataset (e.g., ImageNet) as a starting point for a new task. Leverages learned features, reducing training time and data requirements.

Fine-tuning: Adapting a pre-trained model to a new task by continuing training on new data, typically with lower learning rates. May freeze early layers and only train later layers.

Data Augmentation: Artificially expanding the training dataset by applying transformations (rotations, flips, crops, color jittering, mixup) to existing samples, improving model robustness and generalization.

Week 5: Regularization & Generalization in DL

Overfitting: When a model learns the training data too well, including noise and outliers, resulting in poor generalization to unseen data. The model has high training accuracy but low validation/test accuracy.

Underfitting: When a model is too simple to capture the underlying patterns in the data, resulting in poor performance on both training and validation sets. The model has insufficient capacity.

Dropout: A regularization technique that randomly deactivates neurons during training with probability p (typically 0.5). This prevents co-adaptation of neurons and reduces overfitting by creating an ensemble effect.

L2 Regularization (Weight Decay): Adds the sum of squared weights to the loss function: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_i \theta_i^2$. Encourages smaller weights and smoother models, preventing overfitting.

L1 Regularization: Adds the sum of absolute values of weights to the loss function: $\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \lambda \sum_i |\theta_i|$. Promotes sparsity by driving some weights to exactly zero.

Early Stopping: A regularization strategy that monitors validation performance during training and stops when performance begins to degrade, preventing overfitting to training data.

Bias-Variance Tradeoff: The balance between a model's ability to fit training data (low bias) and its sensitivity to training set variations (low variance). High bias leads to underfitting, while high variance leads to overfitting. Optimal models balance both.

Training Curves Interpretation: Plotting training and validation loss/accuracy over epochs. Diverging curves indicate overfitting. Both curves plateauing at high loss indicates underfitting. Converging curves suggest good generalization.

Week 6: Sequence Modeling & RNNs

Recurrent Neural Network (RNN): A neural network designed for sequential data that maintains a hidden state updated at each time step: $h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$.

Vanishing Gradient Problem: In deep or long-sequence networks, gradients can become extremely small during backpropagation, making it difficult to learn long-term dependencies. Occurs when repeated multiplication of gradients less than 1 causes exponential decay.

Exploding Gradient Problem: The opposite of vanishing gradients, where gradients become extremely large, causing unstable training. Can be mitigated using gradient clipping.

Long Short-Term Memory (LSTM): An RNN variant with gated mechanisms (forget gate, input gate, output gate) and a cell state that can maintain information over long sequences, addressing the vanishing gradient problem.

Gated Recurrent Unit (GRU): A simplified version of LSTM with fewer parameters, using reset and update gates to control information flow. Computationally more efficient than LSTM while maintaining similar performance.

Backpropagation Through Time (BPTT): The algorithm for training RNNs by unrolling the network through time and applying standard backpropagation across all time steps.

Sequence-to-Sequence (Seq2Seq): An architecture using an encoder RNN to process input sequences and a decoder RNN to generate output sequences, commonly used for machine translation.

Week 7: Transformers & Attention Mechanisms

Attention Mechanism: A technique that allows models to focus on relevant parts of the input when producing each output. Computes weighted combinations of input representations based on learned relevance scores.

Self-Attention: An attention mechanism where the query, key, and value all come from the same sequence, allowing each position to attend to all positions in the input.

Scaled Dot-Product Attention: Computed as $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$, where Q, K, V are query, key, and value matrices, and d_k is the key dimension.

Multi-Head Attention: Runs multiple attention mechanisms in parallel with different learned projections, allowing the model to attend to different representation subspaces simultaneously.

Transformer Architecture: A neural network architecture based entirely on attention mechanisms, consisting of encoder and decoder stacks with self-attention, cross-attention, and feed-forward layers.

Positional Encoding: Since Transformers have no inherent notion of sequence order, positional encodings are added to input embeddings to provide position information, typically using sinusoidal functions.

Encoder-Decoder Architecture: The encoder processes the input sequence to create contextual representations, while the decoder generates the output sequence autoregressively, attending to encoder outputs.

Masked Self-Attention: Used in the decoder to prevent positions from attending to subsequent positions, ensuring autoregressive generation during training.

Large Language Models (LLMs): Transformer-based models trained on massive text corpora. Encoder-only models (BERT) are used for understanding tasks, while decoder-only models (GPT) are used for generation.

Week 8: Generative Models (VAE, GANs)

Generative Model: A model that learns the underlying data distribution $p(x)$ and can generate new samples from this distribution, as opposed to discriminative models that learn $p(y|x)$.

Latent Variable Models: Models that represent data using unobserved (latent) variables that capture underlying factors of variation. Examples include VAEs and GANs.

Variational Autoencoder (VAE): A generative model that learns a latent representation by encoding inputs to a distribution (typically Gaussian) and decoding samples from this distribution. Trained using the Evidence Lower Bound (ELBO).

VAE Encoder-Decoder: The encoder (recognition network) maps inputs to latent distribution parameters (μ, σ), while the decoder (generative network) reconstructs inputs from latent samples.

Reparameterization Trick: A technique in VAEs that enables backpropagation through stochastic sampling by expressing the sample as $z = \mu + \sigma \odot \epsilon$ where $\epsilon \sim \mathcal{N}(0, I)$.

ELBO (Evidence Lower Bound): The objective function for VAEs: $\mathcal{L} = \mathbb{E}_{q(z|x)}[\log p(x|z)] - D_{KL}(q(z|x)||p(z))$. Balances reconstruction quality and regularization.

KL Divergence: A measure of difference between two probability distributions. In VAEs, it regularizes the latent space by penalizing deviation from a prior distribution: $D_{KL}(q(z|x)||p(z))$.

Generative Adversarial Network (GAN): A framework with two networks: a generator that creates fake samples from noise and a discriminator that distinguishes real from fake samples. Trained through adversarial competition.

Generator: The network in a GAN that takes random noise z as input and produces synthetic data samples $G(z)$ that should resemble real data from the training distribution.

Discriminator: The network in a GAN that classifies inputs as real or fake, providing feedback to improve the generator. Outputs a probability that the input is real.

Adversarial Training: The minimax game between generator and discriminator: $\min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}}[\log D(x$ $\mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]$.

GAN Training Challenges: Issues include mode collapse (generator produces limited variety), vanishing gradients (discriminator too strong), and training instability requiring careful hyperparameter tuning.

Mode Collapse: A failure mode in GANs where the generator produces limited variety of samples, failing to capture the full diversity of the data distribution.

Applications of Generative Models: Image synthesis and super-resolution, style transfer, data augmentation, anomaly detection (identifying outliers based on reconstruction error or discriminator scores), and creative applications.

Week 9: Deep Learning for Vision & Robotics

Object Detection: The task of identifying and localizing objects in images, typically producing bounding boxes with coordinates and class labels for multiple objects.

YOLO (You Only Look Once): A real-time object detection architecture that frames detection as a single regression problem, predicting bounding boxes and class probabilities directly from full images in one forward pass.

SSD (Single Shot Detector): A fast object detection method using multi-scale feature maps to detect objects of different sizes efficiently.

Faster R-CNN: A two-stage object detection architecture with a Region Proposal Network (RPN) that generates candidate bounding boxes, followed by classification and refinement.

Semantic Segmentation: Assigning a class label to every pixel in an image, creating a pixel-wise classification map. All pixels of the same class get the same label.

Deep Reinforcement Learning Overview: Combining deep neural networks with reinforcement learning to learn policies or value functions from high-dimensional observations (images, sensor data).

Value-Based Methods: RL approaches that learn value functions (Q-functions) and derive policies from them. Examples include DQN (Deep Q-Networks) which uses neural networks to approximate Q-values.

Policy-Based Methods: RL approaches that directly learn a policy mapping states to actions, optimized using policy gradient methods (e.g., REINFORCE, PPO, A3C).

CNNs in Robotics Perception: Convolutional neural networks process visual input from cameras for tasks like object recognition, pose estimation, grasp prediction, and scene understanding in robotic systems.

Real-Time Inference: The requirement for models to make predictions quickly enough for time-sensitive applications (autonomous vehicles, robotics). Involves model optimization and efficient architectures.

Deployment Constraints: Practical limitations including computational resources (embedded devices), memory, power consumption, and latency requirements that affect model design and implementation.

Week 10: Deployment, Ethics, and Final Presentations

Model Deployment: The process of taking a trained model and making it available for use in production environments, requiring consideration of inference speed, memory, and platform compatibility.

ONNX (Open Neural Network Exchange): An open format for representing deep learning models that enables interoperability between different frameworks (PyTorch, TensorFlow, etc.), facilitating deployment.

TensorRT: NVIDIA's high-performance deep learning inference optimizer and runtime that accelerates inference on NVIDIA GPUs through layer fusion, precision calibration, and kernel optimization.

Mobile Deployment: Deploying models on resource-constrained mobile devices (smartphones, edge devices) requiring techniques like quantization, pruning, and specialized architectures (MobileNet, EfficientNet).

Model Compression: Techniques to reduce model size and computational requirements while maintaining performance. Essential for deployment on resource-constrained devices.

Pruning: Removing unnecessary weights or entire neurons from a neural network based on importance metrics. Can be structured (removing entire filters) or unstructured (removing individual weights).

Quantization: Reducing the precision of model weights and activations from 32-bit floating point to lower bit representations (16-bit, 8-bit, or even binary). Post-training quantization or quantization-aware training.

Knowledge Distillation: Training a smaller "student" model to mimic a larger "teacher" model, transferring knowledge while reducing computational requirements.

Bias in AI: Systematic errors in AI systems that can lead to unfair outcomes for certain groups. Can arise from biased training data, problematic feature selection, or algorithmic design choices.

Fairness: The principle that AI systems should treat all individuals and groups equitably, without discrimination based on protected attributes (race, gender, age). Multiple mathematical definitions exist (demographic parity, equalized odds).

Data Privacy: Protecting sensitive information in training data and model outputs. Concerns include data leakage, membership inference attacks, and privacy-preserving techniques (differential privacy, federated learning).

Ethical Issues in AI: Broader concerns including transparency and explainability, accountability for AI decisions, potential for misuse, environmental impact of training large models, and societal implications of automation.

Industry Trends: Current directions including foundation models and transfer learning, multimodal learning (combining vision, language, audio), efficient architectures, and AutoML for automated model design.

Research Frontiers: Cutting-edge areas such as few-shot and zero-shot learning, neural architecture search, self-supervised learning, continual learning, and AI safety and alignment.

3 Defense Schedule

Early Defense Option: Students who complete their projects early may schedule defenses during Week 9 or Week 10. This option is encouraged for those who wish to receive earlier feedback and complete the course requirements ahead of schedule.

Standard Deadline: All final project defenses must be completed by the end of Week 11. This is a firm deadline, and students should plan accordingly to ensure adequate preparation time.

4 Evaluation Criteria

Your final project defense will be evaluated based on the following criteria:

4.1 Presentation Quality (40%)

Assessment considers clarity and organization of slides, completeness of technical content covering both projects, quality of visualizations and results presentation, depth of analysis and insights, and professionalism of delivery.

4.2 Technical Understanding (40%)

Evaluation focuses on demonstrated understanding of implemented methods, ability to explain architectural and design choices, depth of experimental analysis, integration of multiple course concepts, and quality of implementation and results.

4.3 Oral Examination (20%)

Grading is based on accuracy and completeness of answers to conceptual questions, clarity of explanations, ability to connect concepts across topics, and depth of understanding of fundamental principles.

5 Preparation Recommendations

To prepare effectively for your defense, review both midterm and end-term project implementations thoroughly, prepare clear explanations of your technical decisions and trade-offs, practice your presentation multiple times to ensure timing and flow, study all course material with particular attention to definitions and key concepts, be prepared to explain mathematical formulations and algorithms, and review your code to answer detailed implementation questions.

Consider preparing backup slides with additional details that you can reference during questions. Anticipate potential questions about your design choices, limitations, and alternative approaches.

Good luck with your final defense!

This is your opportunity to showcase the skills and knowledge you have developed throughout the course.