# Deep Learning Final Assignment: Parameter-Efficient Fine-tuning with LoRA

Course: Deep Learning
Instructor: Dr. Mahdi Eftekhari

Deadline: Weeks

## 1 Assignment Overview

This capstone assignment integrates multiple deep learning concepts covered throughout the course, focusing on parameter-efficient fine-tuning using Low-Rank Adaptation (LoRA) for question answering. You will implement a complete pipeline for adapting GPT-2 to the SQuAD dataset while exploring the theoretical foundations and practical implications of modern fine-tuning techniques.

- **Points:** 100
- **Estimated Time:** 15-20 hours

## 2 Learning Objectives

By completing this assignment, you will demonstrate mastery of:

1. **Transformer Architecture Understanding**: Deep comprehension of attention mechanisms, causal language modeling, and autoregressive generation
2. **Parameter-Efficient Transfer Learning**: Implementation and analysis of LoRA for reducing computational costs while maintaining performance
3. **Dataset Preprocessing**: Advanced tokenization strategies for sequence-to-sequence tasks with proper attention masking
4. **Training Pipeline Design**: End-to-end model training with custom data collators, loss functions, and evaluation metrics
5. **Model Evaluation**: Comprehensive assessment using domain-specific metrics (SQuAD F1, Exact Match)
6. **Inference and Generation**: Implementation of controlled text generation with sampling strategies

# 3 Theoretical Background

## 3.1 Low-Rank Adaptation (LoRA)

LoRA reduces the number of trainable parameters by constraining the weight updates to a low-rank decomposition:

$$W' = W + \Delta W = W + BA \tag{1}$$

where:

- $W \in \mathbb{R}^{d \times k}$ is the original pre-trained weight matrix
- $\Delta W = BA$ with $B \in \mathbb{R}^{d \times r}$, $A \in \mathbb{R}^{r \times k}$, and $r \ll \min(d, k)$
- Only matrices A and B are trained while W remains frozen

The rank $r$ and scaling factor $\alpha$ control the adaptation capacity:

$$h = Wx + \frac{\alpha}{r} BAx \tag{2}$$

## 3.2 Question Answering with Causal LMs

For extractive QA adapted to generative models, we format examples as:

```
Context: [context_text]
Question: [question_text]
Answer: [answer_text]<eos>
```

The loss function masks prompt tokens to focus learning on answer generation:

$$\mathcal{L} = -\sum_{t=T_{prompt}}^{T_{total}} \log P(x_t | x_{<t}) \tag{3}$$

# 4 Implementation Tasks

## 4.1 Task 1: Data Preprocessing (25 points)

Implement the `preprocess_function` that converts SQuAD examples into properly formatted training sequences:

**Requirements:**

- Create structured prompts with context, question, and answer
- Implement proper tokenization with attention to sequence length
- Design label masking to exclude prompt tokens from loss computation
- Handle edge cases (empty answers, truncation)

**Key Considerations:**

- How does prompt masking affect gradient flow?
- What are the implications of different maximum sequence lengths?
- How should you handle examples where answers don't fit after truncation?

### 4.2 Task 2: Custom Data Collation (20 points)

Implement the `QADataCollator` class for efficient batching:

**Requirements:**

- Dynamic padding to batch maximum length
- Proper attention mask generation
- Label padding with -100 for loss masking
- Memory-efficient tensor creation

**Analysis Questions:**

- Compare dynamic vs. static padding strategies
- Analyze memory usage patterns with different batch sizes
- Discuss trade-offs between padding strategies and computational efficiency

### 4.3 Task 3: LoRA Configuration and Training (30 points)

Set up and execute the LoRA fine-tuning pipeline:

**Requirements:**

- Configure LoRA parameters (rank, alpha, dropout)
- Implement parameter freezing for base model
- Set up training arguments with appropriate hyperparameters
- Monitor training metrics and convergence

**Experimental Analysis:** Design and conduct experiments varying:

- LoRA rank (r = 4, 8, 16, 32)
- Learning rates (1e-4, 2e-4, 5e-4)
- Target modules (attention only vs. attention + MLP)

Create plots showing:

- Training/validation loss curves
- Parameter count vs. performance trade-offs
- Computational cost analysis

### 4.4 Task 4: Inference and Evaluation (25 points)

Implement the complete evaluation pipeline:

**Requirements:**

- Design inference function with controllable generation
- Implement SQuAD metric computation (F1, Exact Match)
- Compare with baseline (non-fine-tuned) model
- Analyze failure cases and model limitations

**Generation Strategy Analysis:** Experiment with different decoding strategies:

- Greedy decoding
- Top-k sampling (k = 10, 25, 50)
- Nucleus sampling (p = 0.8, 0.9, 0.95)
- Temperature scaling (0.7, 1.0, 1.3)

Document the impact on answer quality and diversity.

# 5 Deliverables

## 5.1 1. Complete Implementation (40 points)

Submit fully functional code with all TODOs completed. Code must:

- Run without errors on provided test cases
- Include comprehensive documentation
- Follow clean coding practices
- Include appropriate error handling

## 5.2 2. Experimental Report (40 points)

Submit a 6-8 page technical report including:

**Section 1: Methodology** (10 points)

- LoRA configuration rationale
- Hyperparameter selection process
- Data preprocessing design decisions

**Section 2: Experimental Results** (20 points)

- Quantitative results table with confidence intervals
- Training dynamics analysis with plots
- Ablation study results
- Statistical significance testing

**Section 3: Analysis and Discussion** (10 points)

- Model performance interpretation
- Failure case analysis with examples
- Computational efficiency discussion
- Comparison with full fine-tuning

## 5.3 3. Theoretical Questions (20 points)

Answer the following questions with mathematical rigor:

1. **LoRA Mathematics** (8 points): Derive the forward and backward pass equations for a LoRA-adapted linear layer. Show how gradients flow through the low-rank matrices during backpropagation.

2. **Loss Function Analysis** (6 points): Explain why masking prompt tokens in the loss function is crucial for QA fine-tuning. Discuss potential issues if the entire sequence contributed to loss.

3. **Computational Complexity** (6 points): Compare the time and space complexity of LoRA fine-tuning vs. full fine-tuning. Calculate the parameter reduction factor for your specific configuration.

# 6 Evaluation Criteria

## 6.1 Technical Implementation (40%)

- **Correctness**: Code runs without errors and produces expected outputs
- **Completeness**: All required components implemented
- **Efficiency**: Reasonable computational and memory usage
- **Code Quality**: Clean, documented, well-structured code

## 6.2 Experimental Rigor (30%)

- **Methodology**: Sound experimental design with appropriate controls
- **Analysis**: Thorough interpretation of results with statistical analysis
- **Visualization**: Clear, informative plots and tables
- **Reproducibility**: Results can be replicated from provided code

## 6.3 Theoretical Understanding (20%)

- **Mathematical Accuracy**: Correct derivations and explanations
- **Conceptual Depth**: Deep understanding of underlying principles
- **Critical Analysis**: Thoughtful discussion of limitations and implications

## 6.4 Communication (10%)

- **Clarity**: Well-written, organized report
- **Completeness**: All required sections addressed
- **Professional Presentation**: Proper formatting, citations, and figures

# 7 Common Pitfalls to Avoid

1. **Gradient Issues**: Ensure LoRA parameters have `requires_grad=True`
2. **Memory Leaks**: Properly manage GPU memory with gradient accumulation
3. **Label Misalignment**: Verify prompt masking corresponds to correct token positions
4. **Evaluation Bias**: Don't overfit to validation set during hyperparameter tuning
5. **Generation Loops**: Implement proper stopping criteria for text generation

# 8 Resources and References

## 8.1 Essential Papers

1. Hu et al. (2021). "LoRA: Low-Rank Adaptation of Large Language Models"
2. Rajpurkar et al. (2016). "SQuAD: 100,000+ Questions for Machine Reading Comprehension"
3. Vaswani et al. (2017). "Attention Is All You Need"

### 8.2 Technical Documentation

- Hugging Face Transformers: `https://huggingface.co/docs/transformers/`
- PEFT Library: `https://huggingface.co/docs/peft/`
- PyTorch Documentation: `https://pytorch.org/docs/`

### 8.3 Computational Resources

- You can use Google Colab free plan to train the model for at least 15 epochs(we don't need this much of training)

## 9 Submission Guidelines

1. **Code Submission**: Single Python file with complete implementation
2. **Report**: PDF format, 6-8 pages, IEEE conference style
3. **Results**: Include model checkpoints and evaluation outputs
4. **Reproducibility**: Requirements.txt and execution instructions

**File Structure:**

```
assignment_submission/
|-- lora_qa_finetuning.py
|-- report.pdf
|-- results/
|   |-- model_checkpoints/
|   |-- evaluation_results.json
|   `-- plots/
|-- requirements.txt
`-- README.md
```

## 10 Grading Rubric

This assignment represents the culmination of your deep learning education, integrating theoretical knowledge with practical implementation skills essential for modern AI research and development.

| **Component** | **Excellent (A)** | **Good (B)** | **Satisfactory (C)** | **Needs Improvement (D/F)** |
|---|---|---|---|---|
| Implementation | All components work perfectly, efficient code | Minor issues, mostly working | Some bugs, basic functionality | Major issues, incomplete |
| Experiments | Comprehensive analysis, multiple configurations | Good experiments, some analysis | Basic experiments, limited analysis | Insufficient experimentation |
| Theory | Deep understanding, accurate derivations | Good understanding, mostly correct | Basic understanding, some errors | Poor understanding, major errors |
| Report | Excellent writing, thorough analysis | Good writing, adequate analysis | Basic writing, superficial analysis | Poor communication, incomplete |

Table 1: Grading Rubric