Homework 1: Introduction to Deep Learning and Neural Networks

Course: CS4152- Deep Learning and Neural Networks CLO-1
Instructor: Dr. Jameel Ahmad
Fall 2025
Max Marks: 100
Submission Date: Nov 04, 2025

Work in group of 2 students

October 21, 2025

Overall Learning Objectives

- Develop foundational understanding of neural network mathematics and operations
- Gain practical implementation skills in Python/PyTorch/TensorFlow
- Understand optimization, regularization, and architecture design principles
- Build competency in both theoretical analysis and practical implementation

1 Part A: Mathematical Foundations (50 marks)

Objective: *Master the core mathematical operations and derivations in neural networks*

1. **Feedforward Computation**
   - Understand forward propagation through multi-layer networks
   - Apply activation functions ($\sigma(z) = \frac{1}{1+e^{-z}}$) to compute neuron outputs
   - Perform layer-wise computations with weights and biases
2. **Gradient Computation**
   - Calculate gradients using chain rule for parameter updates
   - Apply derivatives of activation functions and loss functions
   - Compute parameter gradients for backpropagation
3. **Optimization Methods**
   - Implement gradient descent algorithm
   - Understand learning rate impact on parameter updates
   - Apply optimization to minimize loss functions $L = (y - (wx + b))^2$
4. **Backpropagation Mechanics**
   - Derive update rules for different activation functions
   - Understand gradient flow through network layers
   - Analyze mathematical foundations of learning algorithms
5. **Activation Function Analysis**
   - Compare properties of sigmoid, tanh, ReLU, Leaky ReLU, ELU, Swish

- Compute derivatives for gradient calculations
- Understand gradient behavior for different activations

6. **Initialization Strategies**
   - Compare zero initialization vs. Xavier initialization
   - Understand impact on training convergence and stability
7. **Softmax and Multi-class Output**
   - Compute probability distributions from logits
   - Understand gradient calculations for classification outputs

2 Part B: Basic Neural Network Implementation (25 marks)

Objective: *Build foundational programming skills for neural networks*

1. **Perceptron Implementation**
   - Code basic neuron operations from scratch
   - Implement training algorithms for simple models
   - Understand the building blocks of neural networks
2. **Digit Classification System**
   - Design simple neural network architectures
   - Implement classification pipelines
   - Develop data preprocessing and model evaluation skills

3 Part C: Advanced Neural Network Applications (25 marks)

Objective: *Apply neural networks to real-world problems with advanced techniques*

Task 1: Regression - Function Approximation

- Implement feedforward networks for continuous value prediction
- Explore network architecture design (layers, neurons, activations)
- Apply and evaluate different optimization algorithms
- Implement and compare regularization techniques
- Develop model evaluation and performance metrics

Task 2: Classification - FashionMNIST

- Build Convolutional Neural Networks (CNNs) for image classification
- Implement data preprocessing and normalization pipelines
- Design CNN architectures with convolutional layers, pooling, dropout
- Compare optimizer performance (SGD, Adam, RMSprop)
- Analyze training dynamics and generalization

4 Advanced Learning Objectives

4.1 Optimization & Regularization Mastery

- Compare and contrast optimization algorithms (SGD, Adam, RMSprop)
- Understand momentum mechanisms in gradient updates
- Implement and evaluate L1/L2 regularization effects
- Apply dropout for improved generalization
- Analyze weight initialization impact on convergence

4.2 Hyperparameter Tuning & Validation

- Implement systematic hyperparameter search strategies
- Apply k-fold cross-validation techniques
- Understand computational trade-offs in parameter search
- Develop methodology for model selection and evaluation

4.3 CNN Architecture Understanding

- Comprehend convolutional filter operations and spatial hierarchies
- Understand role of pooling, batch normalization, and stride
- Analyze how architecture depth and width affect performance
- Develop intuition for feature learning in deep networks

4.4 Model Interpretation & Visualization

- Visualize learned filters and activation maps
- Interpret model behavior through feature analysis
- Understand limitations of visualization techniques
- Develop skills for model debugging and analysis

4.5 Professional Reporting & Reproducibility

- Document workflow and design decisions systematically
- Ensure code clarity and experimental reproducibility
- Analyze bias-variance tradeoffs in practical settings
- Develop extension strategies for more complex problems

Assessment Focus Areas

- **Theoretical Understanding** (Mathematical derivations, conceptual knowledge)
- **Practical Implementation** (Coding skills, architecture design)
- **Experimental Analysis** (Results interpretation, comparative studies)
- **Critical Thinking** (Design decisions, problem-solving approaches)
- **Professional Communication** (Clear reporting, proper documentation)

Note: These objectives align with Course Learning Outcome CLO-1, focusing on building comprehensive foundational knowledge in deep learning principles and practices.

5 Questions-Part-A [50 Marks]

For this part you will submit handwritten report as it requires mathematical understanding of Deep Neural Networks (DNN).

1. **(Feedforward Computation)** Consider a neural network with two inputs $x_1 = 0.5$, $x_2 = 0.3$, a single hidden layer with two neurons, and an output neuron. The weights and biases are given as follows:

   - Input to Hidden Layer:

   $$\begin{aligned} w_{11} &= 0.8, \quad w_{12} = -0.4, \\ w_{21} &= 0.2, \quad w_{22} = 0.9, \\ b_1 &= 0.1, \quad b_2 = -0.2 \end{aligned}$$

   - Hidden to Output Layer:

   $$w_{h1} = 0.5, \quad w_{h2} = -0.7, \quad b_o = 0.3$$

   Assume a sigmoid activation function $\sigma(z) = \frac{1}{1+e^{-z}}$. Compute the final output of the network.

2. **(Gradient Computation)** Consider a neuron with input $x = 0.7$, weight $w = -0.5$, bias $b = 0.2$, and a sigmoid activation function. Compute the gradient of the loss function L with respect to w if the true label is $y = 1$ and the loss function is Mean Squared Error (MSE).

3. **(Optimization)** Perform one step of gradient descent for a neuron with an input $x = 2$, weight $w = 1$, bias $b = -1$, and learning rate $\eta = 0.1$. The loss function is $L = (y - (wx + b))^2$, where the target output is $y = 3$.

4. **(Backpropagation)** Derive the backpropagation update rule for a neuron with a ReLU activation function.

5. **(Weight Initialization)** Compare the effect of using zero initialization versus Xavier initialization in a simple neural network.

6. **(Tanh Activation Function)** Compute the derivative of the Tanh activation function and use it to find the gradient of the loss function L with respect to weight w in a neuron where $a = \tanh(z)$ and $z = wx+b$.

7. **(Leaky ReLU Activation)** Consider a neuron with Leaky ReLU activation $f(z) = \max(0.01z, z)$. Derive its derivative and compute the gradient update for weight w.

8. **(Softmax Function)** Consider a neural network output layer using the softmax function. Given logits $z_1 = 2.0$, $z_2 = 1.5$, compute the softmax probabilities and their gradients.

9. **(ELU Activation)** Given an ELU activation function $f(z) = \alpha(e^z - 1)$ for $z < 0$, compute its derivative and derive the backpropagation update.

10. **(Swish Activation)** Compute the derivative of the Swish activation function $f(z) = z\sigma(z)$ and derive the gradient computation.

6 Programming Exercises-Part B [25 marks]

In this first homework, you are going to write your own simple feedforward neural network code using Python and NumPy (the standard numeric library for Python). you will start by implementing just a simple neuron, or perceptron, then you define the training algorithm for this simple model. The second part consists in defining a simple neural network to perform digits classification.

Important Instructions for Submissions:

Generally, in the homeworks, you will either need to complete a part of Python code or answer questions in text cells. Code and text cells where you should write your answers are marked by STARTCODE and ENDCODE or STARTTEXT and ENDTEXT tags, respectively. Do not change, move, or remove these tags; otherwise, your answers will not be valid. Each cell that includes a [TO COMPLETE] part is placed between these placeholders.

Python Notebook(.ipynb) to complete and practice is given at
https://colab.research.google.com/drive/1ay39qpNOssbpEoKAEnzuW6FcMaTLNJOu?usp=sharing
You can download it and then self-practice it on your own. Attach the code and your results (graphs) in your final submission. Add a shared link to your Colab and attach a clean copy of the results to your report with a brief description.

For a good professional report write-up, you can use LaTeX instead of MS Word and complete your write-up on www.overleaf.com or using any TeX editor.

7 Programming Exercises-Part C [25 marks]

In this homework, you will also implement and test simple neural network models for solving **supervised learning problems**. The assignment is divided into two tasks:

- **Task 1: Regression** — Function approximation using feedforward neural networks.
- **Task 2: Classification** — Image recognition using FashionMNIST dataset.

In both cases, you are required to explore:

- Advanced optimizers (SGD with momentum, Adam, RMSprop)
- Regularization techniques (L1, L2, Dropout, Early Stopping)
- Weight initialization schemes
- Hyperparameter tuning via grid/random search with cross-validation

The goal is to improve convergence of stochastic gradient descent and promote generalization. The homework should be implemented in **Python using PyTorch**.

Sample Input Data and Implementation Hints

Regression Task (Function Approximation)

You may approximate a continuous function such as:

$$y = \sin(x) + 0.1\epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

Sample input data can be generated using:

```python
import torch, numpy as np
x = torch.linspace(-2*np.pi, 2*np.pi, 200).unsqueeze(1)
y = torch.sin(x) + 0.1*torch.randn(x.size())
```

Use a simple feedforward neural network with two hidden layers to learn the mapping.

Classification Task (FashionMNIST)

Use the FashionMNIST dataset available in `torchvision.datasets`. Example:

```python
from torchvision import datasets, transforms
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Normalize((0.5,), (0.5,))])
trainset = datasets.FashionMNIST(root='./data', train=True,
                                 download=True, transform=transform)
```

Implement a CNN with at least two convolutional layers, ReLU activations, and dropout.

Questions

Part 1 — Regression Task

1. What function or dataset did you use for the regression problem, and why?
2. Describe your network architecture (layers, neurons, activation functions).
3. Which loss function and optimizer were used, and why?
4. How did you evaluate model performance (e.g., MSE, RMSE, R^2)?
5. What regularization methods improved convergence or reduced overfitting?

Part 2 — Classification Task (FashionMNIST)

1. What preprocessing steps did you apply before training?
2. Describe your CNN architecture, kernel sizes, and feature map dimensions.
3. Which optimizer achieved the best accuracy? Compare Adam vs. SGD.
4. How did dropout or weight decay affect training stability?
5. What was your final accuracy and confusion matrix on the test set?

Part 3 — Optimization and Regularization

1. Compare the performance of SGD, Adam, and RMSprop optimizers.
2. Explain how momentum influences gradient updates.
3. How did L1 and L2 regularization affect learned weights?
4. What impact did dropout have on training and validation accuracy?
5. Why is weight initialization critical for deep learning convergence?

Part 4 — Hyperparameter Tuning and Cross-Validation

1. Which hyperparameters did you tune and how (grid or random search)?
2. How did you implement k-fold cross-validation?
3. Report the best-performing hyperparameter set and corresponding results.
4. Discuss computational trade-offs between exhaustive and random search.
5. What challenges did you face during tuning, and how were they resolved?

Part 5 — Convolutional Neural Networks

1. Why are CNNs effective for image classification tasks?
2. What is the role of pooling and batch normalization layers?
3. How do convolutional filters learn spatial hierarchies?
4. Explain the role of stride, padding, and filter size in your model.
5. How would deeper or wider CNNs affect generalization?

Part 6 — Visualization and Interpretability

1. How did you visualize learned filters and activation maps?
2. Compare feature maps from early vs. deep layers.
3. What did weight histograms reveal about training dynamics?
4. How can maximizing activation gradients help interpret model focus?
5. What are possible limitations of such visualization techniques?

Part 7 — Report and Implementation Understanding using LATEX, IEEE Format 2-column, 10 points, Time New Romans

1. Summarize your workflow and major design decisions.
2. How did you ensure code clarity and reproducibility?
3. What are your main takeaways regarding bias–variance tradeoff?
4. If given more time, what improvements or extensions would you make?
5. How could this experiment generalize to datasets like CIFAR-10?

While FashionMNIST is excellent for beginners, numerous other image datasets offer varying complexity, domains, and challenges for Convolutional Neural Network (CNN) training and evaluation.

8 Popular Image Classification Datasets to Choose from –other than FashionMNIST

8.1 1. Standard Academic Datasets

Dataset	**Domain**	**Classes**	**Key Features**
CIFAR-10	Object Recognition	10	32×32 color images
CIFAR-100	Object Recognition	100	Fine-grained categories
SVHN	Street View House Numbers	10	Real-world digit recognition
STL-10	Object Recognition	10	96×96 higher resolution

Table 1: Standard academic datasets for CNN classification

8.2 2. Large-Scale Datasets

- **ImageNet** (1,000 classes, 1.2M images)
  - Large-scale visual recognition challenge
  - High diversity and real-world complexity
  - Standard benchmark for state-of-the-art models
- **Places365** (365 scene categories, 1.8M images)
  - Scene understanding and classification
  - Environmental context recognition
- **Open Images Dataset** (19,958 classes, 9M images)
  - Extremely diverse object categories
  - Multi-label classification challenges

8.3 3. Specialized Domain Datasets

8.3.1 Medical Imaging

- **CheXpert** (Chest X-rays)
- **ISIC** (Skin lesion classification)
- **BraTS** (Brain tumor segmentation)
- **Retinopathy datasets** (Diabetic retinopathy)

8.3.2 Remote Sensing

- **UC Merced Land Use** (21 land use classes)
- **NWPU-RESISC45** (45 scene classes)
- **BigEarthNet** (Multi-label satellite imagery)

8.3.3 Agricultural and Environmental

- **PlantVillage** (Plant disease detection)
- **iNaturalist** (Species identification)
- **DeepWeeds** (Weed species classification)

9 Dataset Complexity Spectrum

9.1 Beginner-Level Datasets

- **MNIST** (Handwritten digits) - 10 classes, 28×28 grayscale
- **FashionMNIST** - 10 fashion categories, 28×28 grayscale
- **Kuzushiji-MNIST** - Japanese characters, 10 classes

9.2 Intermediate-Level Datasets

- **CIFAR-10/100** - Object recognition with color complexity
- **SVHN** - Real-world digit recognition with noise
- **Caltech-101/256** - Object categories with viewpoint variation

9.3 Advanced-Level Datasets

- **ImageNet** - Large-scale, fine-grained classification
- **FGVC Aircraft** - Fine-grained aircraft model recognition
- **Stanford Cars** - Fine-grained vehicle classification
- **CUB-200-2011** (Birds) - Fine-grained bird species identification

10 Domain-Specific Applications

10.1 Autonomous Vehicles

- **BDD100K** - Diverse driving scenarios
- **Cityscapes** - Urban street scene understanding
- **KITTI** - Autonomous driving benchmarks

10.2 Facial Recognition and Analysis

- **CelebA** - Celebrity facial attributes
- **LFW** (Labeled Faces in the Wild) - Unconstrained face recognition
- **FFHQ** (Flickr-Faces-HQ) - High-quality facial images

10.3 Document Analysis

- **RVL-CDIP** - Document classification
- **IAM Handwriting** - Handwritten text recognition
- **Tobacco3482** - Document image classification

11 Considerations for Dataset Selection

11.1 Technical Factors

- **Image Resolution**: MNIST (28×28) vs. ImageNet (variable, often 224×224+)
- **Color Channels**: Grayscale vs. RGB vs. multi-spectral
- **Dataset Size**: From thousands (CIFAR) to millions (ImageNet) of images
- **Class Balance**: Balanced vs. imbalanced distributions
- **Annotation Quality**: Clean vs. noisy labels

11.2 Practical Considerations

- **Computational Requirements**: Memory, storage, and processing power
- **Licensing**: Academic use vs. commercial restrictions
- **Preprocessing Needs**: Normalization, augmentation requirements
- **Benchmark Availability**: Established baselines for comparison

12 Emerging Trends and Datasets

12.1 Few-Shot Learning

- **miniImageNet** - Few-shot learning benchmark
- **tieredImageNet** - More challenging few-shot classification
- **Omniglot** - Character recognition with many classes, few examples

12.2 Multi-Modal Datasets

- **MS-COCO** - Object detection with captions
- **Visual Genome** - Dense image annotations with relationships
- **Conceptual Captions** - Image-text pairs for multi-modal learning

12.3 Adversarial Robustness

- **ImageNet-C** - Common corruption robustness benchmark
- **ImageNet-A** - Natural adversarial examples
- **RobustBench** - Standardized adversarial robustness evaluation

13 Implementation Considerations

13.1 Data Loading Frameworks

- PyTorch: `torchvision.datasets`
- TensorFlow: `tf.keras.datasets` and `tensorflow_datasets`
- Custom datasets via `Dataset` classes and data loaders

13.2 Preprocessing Pipelines

- Standard normalization: ImageNet stats vs. dataset-specific
- Data augmentation: Rotation, flipping, color jittering
- Resizing strategies: Center cropping, random cropping, padding

14 Conclusion

The choice of dataset should align with:

- Learning objectives and student level
- Available computational resources
- Domain interests and applications
- Desired complexity and challenge level
- Benchmarking and comparison needs

Starting with FashionMNIST and progressing to CIFAR-10, then to specialized domains provides a structured learning path for CNN education.