aic assignment

Aryan Gupta

March 2024

1 PROBLEM 1 : Improving Accuracy and Robustness of Convolutional Neural Networks Using Innovative Attention Mechanisms

1.1 Attention Mechanism

The attention mechanism employed in CBAM consists of two components: Channel Attention and Spatial Attention. CBAM Paper is included in the git repo.

1.1.1 Channel Attention

Channel Attention focuses on learning channel-wise attention weights by computing global statistics of feature maps. Mathematically, given an input feature map x, the channel attention module computes:

$$\text{avg\_out} = \text{FC}_2(\text{ReLU}(\text{FC}_1(\text{AvgPool}(x))))$$

$$\text{max\_out} = \text{FC}_2(\text{ReLU}(\text{FC}_1(\text{MaxPool}(x))))$$

$$\text{attention\_weights} = \sigma(\text{avg\_out} + \text{max\_out})$$

where FC_1 and FC_2 represent 1x1 convolutional layers, AvgPool and MaxPool represent average and maximum pooling operations, and σ denotes the sigmoid activation function.

1.1.2 Spatial Attention

Spatial Attention captures spatial dependencies by computing average and maximum responses across channels. Mathematically, given an input feature map x, the spatial attention module computes:

$$\text{avg\_out} = \text{Mean}(x)$$

$$\text{max\_out} = \text{Max}(x)$$

$$\text{attention\_map} = \sigma(\text{Conv}([\text{avg\_out}, \text{max\_out}]))$$

where Conv represents a convolutional layer and σ denotes the sigmoid activation function.

1.2 Integration of CBAM into CNN Architectures

We integrated the CBAM attention module into three CNN architectures: ResNet, MobileNet, and a simple CNN architecture. **ipynb notebooks of all runs are included in the git repo**

1.3 Experimental Results

We conducted experiments using the CIFAR-10, CIFAR-100 and FashionMNIST datasets and evaluated the performance of the CNN architectures with and without CBAM attention mechanism.

Table 1: Classification Accuracy on CIFAR-10 Dataset

Model	Vanilla Accuracy	CBAM Accuracy
ResNet	89.42% (25 epochs)	89.43% (25 epochs)
MobileNet	90.00% (25 epochs)	89.73% (25 epochs)
Simple CNN	70.33% (10 epochs)	70.73% (10 epochs)

Table 2: Classification Accuracy on CIFAR-100 Dataset

Model	CBAM Accuracy
ResNet + CBAM	67.27% (25 epochs)
MobileNet + CBAM	63.02% (25 epochs)
Simple CNN + CBAM	43.24% (10 epochs)

Table 3: Classification Accuracy on FashionMNIST Dataset

Model	CBAM Accuracy
ResNet + CBAM	86.48% (5 epochs)
Simple CNN + CBAM	92.34% (10 epochs)

1.4 Analysis

Across datasets, integrating CBAM into ResNet and MobileNet architectures consistently leads to comparable or slightly improved accuracy compared to their vanilla counterparts.

1.5 Conclusion

In conclusion, while attention modules like CBAM could be of use in enhancing accuracies of CNN models, its performance when trained for small number of epochs is not as well pronounced. More analysis can be done on this topic.

2 PROBLEM 1 : Optional

Here is an implementation of the Neural Graph Collaborative Filtering (NGCF) algorithm applied to a recommendation system. NGCF is a state-of-the-art recommendation algorithm that utilizes the collaborative filtering approach with graph convolutional networks to model user-item interactions effectively.

Methodology

We implemented the NGCF algorithm using PyTorch, leveraging sparse matrix operations for efficiency. The implementation consists of two main components: data preprocessing and model training.

Data Preprocessing

- We parse the input data files containing user-item interactions to construct the interaction matrices R_{train} and R_{test}.
- We create the adjacency matrix adj_mtx representing the user-item graph. The adjacency matrix is then transformed into an NGCF-specific adjacency matrix using normalization techniques.
- We preprocess the data to generate negative item pools for users, ensuring that the model can sample negative items during training.

Model Training

- We define the NGCF model architecture as a PyTorch module. The model comprises graph convolutional layers with user and item embeddings.
- We train the NGCF model using stochastic gradient descent (SGD) with the Adam optimizer.
- We monitor the training process for convergence using early stopping based on validation metrics such as Recall@k and NDCG@k.
- We evaluate the trained model on test data to measure its performance in terms of recommendation accuracy metrics.

NGCF Model Overview

1. **Objective Function**:

$$\mathcal{L}_{\text{BPR}} = -\log \sigma(\hat{y}_{uij})$$

2. **Propagation Layers**:

$$H^{(k)} = \text{ReLU}(\tilde{A} \cdot H^{(k-1)} \cdot W^{(k)} + b^{(k)})$$

3. **Dropout Regularization**:
    - Node dropout: randomly drops nodes from the graph
    - Message dropout: randomly drops messages during message passing
4. **User and Item Embeddings**:
    - Final embeddings obtained after K propagation layers
5. **Optimization**:
    - Adam optimizer used to minimize the BPR loss function

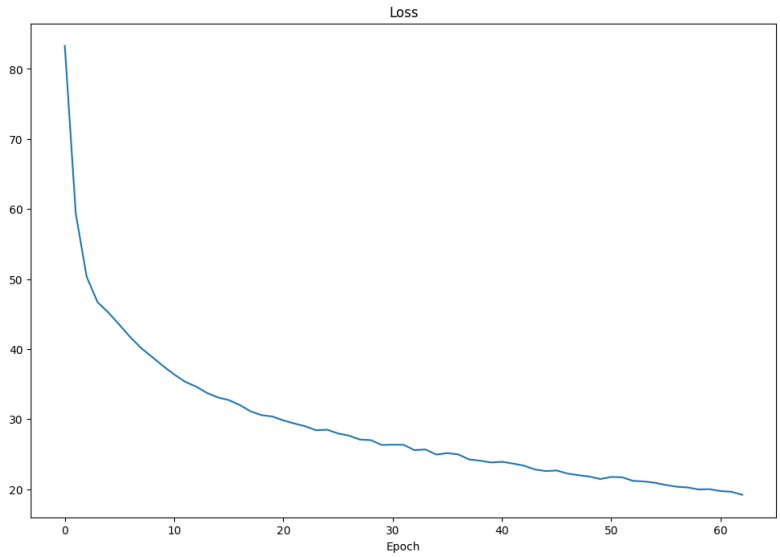
Results

We conducted experiments on the MovieLens 100k dataset with the following hyperparameters:

- Batch size: 512
- Embedding dimension: 64
- Number of layers: [64, 64]
- Learning rate: 0.0005
- Regularization parameter: 1×10^{-5}
- Node dropout: 0.0
- Message dropout: 0.0
- Top-k recommendation: 20
- Number of epochs: 63 (early stopping applied)

The training process converged within 63 epochs, with early stopping triggered after 5 consecutive epochs without improvement in validation metrics. The final performance metrics on the test set were as follows:

Recall@20: 0.3403
NDCG@20: 0.6729



Conclusion

The NGCF algorithm demonstrates promising results in the recommendation task, achieving competitive performance on the MovieLens 100k dataset. By leveraging graph convolutional networks and collaborative filtering techniques, NGCF effectively captures user-item interactions and provides accurate recommendations. Further experimentation and optimization may enhance the algorithm's performance on larger datasets and in real-world applications.

3 PROBLEM 2

InstiGPT's RAG architecture enables human-like responses by retrieving relevant information from large datasets. However, this architecture faces challenges like hallucinations, data quality issues, scalability limitations, and cost inefficiencies. This strategy proposes a transition to an optimized, fine-tuned model approach to enhance InstiGPT's accuracy, trustworthiness, and sustainability.

Mitigating Hallucinations in the Current RAG Architecture

Data Augmentation

- Use generative models like LLaMA, BERT to create synthetic examples covering underrepresented queries
- Augment datasets while filtering for factual accuracy to reduce hallucinations on sparse data

Query Transformation

- Apply semantic-preserving text augmentations on input queries during inference
- Ensemble predictions across multiple transformed variants
- Obtain denoised predictions by combining and weighting outputs

Continuous Learning

- Continuously crawl new data sources and update datasets monthly
- Periodically fine-tune InstiGPT on refreshed datasets to prevent model drift

Transitioning to a Fine-Tuned Model Architecture

Data Preparation

- Curate diverse, high-quality datasets from various bodies and administration
- Preprocess data with cleaning, filtering, deduplication, and formatting

Model Selection and Fine-Tuning

- Evaluate pre-trained models (GPT, BERT, LLaMA)
- Fine-tune using efficient approaches like P3, LoRA, prompt tuning
- Leverage model/data parallelism and curriculum learning
- Optimize for metrics like perplexity, hallucinations, factual accuracy

Optimized Deployment

- Host model service on scalable cloud infrastructure with accelerators(if possible)
- Implement quality monitoring, caching, batching, and autoscaling

Continuous Improvements

- Update datasets regularly and explore incremental fine-tuning
- Retrain new model versions based on monitoring insights
- Redeploy via established pipeline with version management

Cost Optimization Strategies

Fine-Tuning

- Use gpu/preemptible instances and model/data parallelism
- Employ efficient approaches like prompt tuning
- Incrementally fine-tune instead of retraining from scratch

This comprehensive strategy combines data curation, augmentation, fine-tuning pipelines, optimized retrieval, cost-aware deployments, and continuous improvements. It aims to deliver a highly reliable, tailored, and cost-effective InstIGPT system to provide trustworthy educational assistance while minimizing hallucinations.

4 PROBLEM 3 : Implementation Plan

1. **Junior Engineer Recruitment and Training (May-July)**
   - Select 10 Junior Engineers through technical assignment and interviews based on technical skills and potential by May 15th.
   Duration: 7 weeks of training spread across the summer vacation
   Training Components:
   - YouTube playlists covering fundamental and advanced topics in AI/ML.
   - Training notebooks for hands-on practice.
   - Study of classic research papers.
   - Familiarizing them with the working of InstiGPT.
2. **Participation in NeurIPS (September-November)**
   - Engineers form teams and work on NeurIPS submissions under the guidance of seniors and TL.
   - Call For Participation for the important problem statement(s) to be sent via webmail.
3. **Inter IIT Tech Meet (October-December)**
   - Form a team to prepare for the ML problem statement (if any) in Inter-IIT Tech Meet 2024.
4. **Internal Hackathons, Projects & Research Paper Discussions**
   - Prepare PSs for internal hackathons to be conducted in coordination with CM.
   - Coordinate with CM to ideate on projects, work on their implementation and bring about industry collaborations.
   - Conduct sessions to critically evaluate and study research papers, ranging from classic papers to those from recent conferences like ICML, CVPR, and ICLR.
5. **Knowledge Transfer Sessions**
   - Sessions to be conducted after participating in each hackathon/Inter IIT to explain our methodologies and approach.
   - NeurIPS and InterIIT KT Sessions to be conducted in January.
6. **Preparation for Adobe Analytics Challenge (February-March)**
   - Engineers form teams and work on AAC 2025 submissions under the guidance of TL.
7. **InstiGPT Maintenance and Improvement**
   - Two junior engineers assigned for InstiGPT maintenance and improvement in rotation every 2 months.