Command Classification with CNN and GloVe Embeddings

This project implements a Convolutional Neural Network (CNN) model for classifying natural language queries into command categories. It uses pre-trained GloVe word embeddings for text representation.

Project Overview

The system takes natural language queries as input and classifies them into appropriate command categories. It's designed for command classification in CAD systems like Rhino 3D, helping users find the right commands efficiently.

Key Features

- **Text Classification CNN**: Implements a TextCNN architecture optimized for command recognition.
- **Pre-trained Word Embeddings**: Uses GloVe embeddings for rich semantic word representation.
- **Balanced Dataset Handling**: Ensures proper representation of all commands in training/testing.
- **Performance Visualization**: Includes tools to visualize model performance and confusion matrices.
- **Filter Size Optimization**: Experimentally determined optimal CNN filter sizes for command queries.

Project Structure

- **CMPSC_497_MID_TERM_PROJECT_TEXT_CLASSIFICATION_USING_CNN.ipynb**: Main notebook with detailed annotations of all steps.
- **train.json**: JSON file containing the training dataset.
- **test.json**: JSON file containing the test dataset.

Model Architecture

The model follows this pipeline: - 1. Text preprocessing (lowercase, tokenization, removing special characters) - 2. Word embedding lookup using pre-trained GloVe vectors (100-dimensional) - 3. Multiple parallel convolutions with filter sizes [3, 4] for n-gram pattern recognition - 4. Max-over-time pooling to capture the most important features - 5. Concatenation of pooled features from different filter sizes - 6. Dropout (0.6) for regularization and preventing overfitting - 7. Fully connected layer for final classification

Model Configuration and Hyperparameters

The model is configured with the following carefully tuned hyperparameters:

```
# Hyperparameters
batch_size = 64          # Number of samples processed per batch
num_filters = 100        # Number of convolutional filters per size
filter_sizes = [3, 4]    # Sizes of convolutional filters for n-gram detection
dropout_rate = 0.6       # Dropout probability for regularization
learning_rate = 0.001    # Learning rate for optimizer
num_epochs = 10          # Number of complete passes through the training dataset
```

Optimization Strategy

We use Adam optimizer with L2 regularization to prevent overfitting:

```
# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate, weight_decay=1e-5)
```

Regularization Approach

Our experiments showed that combining multiple regularization techniques yielded the best results:

- **High Dropout Rate**: A dropout rate of 0.6 was found to be optimal for our dataset size, helping prevent co-adaptation of neurons.
- **L2 Weight Regularization**: We added L2 penalty (weight_decay=1e-5) to constrain the model's weights and prevent overfitting.
- **Filter Size Limitation**: Using only [3, 4] filter sizes instead of [3, 4, 5] reduced model complexity while maintaining high accuracy.

This configuration achieves an optimal balance between model complexity and generalization ability, with only ~2% difference between training and test accuracy.

Word Embeddings and Fine-tuning

We use pre-trained GloVe embeddings but importantly keep fine-tuning enabled to allow the embeddings to adapt to Rhino-specific semantic relationships:

```
# Initialize with GloVe embeddings
if embedding_matrix is not None:
    self.embedding.weight = nn.Parameter(torch.tensor(embedding_matrix, dtype=torch.float))
    self.embedding.weight.requires_grad = True  # Fine-tune the embeddings
```

This approach gives us the best of both worlds:

- Starting with rich semantic representations from general-domain text.
- Adapting these representations for CAD-specific terminology like "extrude", "surface", "curve", etc.
- Improving recognition of domain-specific relationships between concepts in Rhino 3D.

Performance

The model achieves excellent accuracy on command classification tasks:

- **Training Accuracy**: ~95%
- **Test Accuracy**: ~93%
- **Minimal overfitting gap**: (~2%)
- **Fast convergence**: within 5-7 epochs

Sample training results:

```
Epoch 10/10:
Train Loss: 0.1793, Train Acc: 95.33%
Test Loss: 0.2181, Test Acc: 93.40%
```

Dataset

The dataset contains natural language queries about commands, split to ensure:

- Each command has proportional representation in both train and test sets.
- For commands with ≥20 examples: 17 for training, 3 for testing.
- For commands with <20 examples: 85%/15% train/test split.
- Random shuffling for unbiased distribution.

The `split_dataset.py` script handles this balanced division:

```
# Split data: 17 for training, 3 for testing per command
```

```
for command, items in command_to_queries.items():
    # Shuffle to ensure random selection
    random.shuffle(items)

    # Get train and test samples
    if len(items) >= 20:
        train_samples = items[:17]
        test_samples = items[17:20]
    else:
        train_count = int(len(items) * 0.85)
        train_samples = items[:train_count]
        test_samples = items[train_count:]
```

Requirements

- Python 3.8+
- PyTorch 1.9+
- NLTK 3.6+
- gensim 4.0+ (for GloVe embeddings)
- scikit-learn 1.0+
- matplotlib, seaborn (for visualization)
- tqdm (for progress tracking)

Results and Insights

Our experiments revealed:

- Filter sizes [3, 4] perform better than [3, 4, 5] for command classification.
- Dropout of 0.6 provides optimal regularization.
- Training converges quickly (5-7 epochs) with minimal overfitting.
- Commands with similar functionality show expected confusion patterns in the confusion matrix.

Authors

- Deebak Tamilmani

License

This project is licensed under the MIT License - see the LICENSE file for details.

Acknowledgments

- This project is part of the CMPSC 497 Mid-Term Project.
- The TextCNN architecture is inspired by Kim's paper "Convolutional Neural Networks for Sentence Classification".
- GloVe embeddings provided by Stanford NLP Group.