Financial News Data Preprocessing Pipeline

This pipeline outlines the steps for transforming raw financial news data into a structured, model-ready format. It ensures consistency, quality, and suitability for time-series modeling using financial articles.

---

1. Input Data Structure

Raw financial news articles are stored in JSON format, organized by month:

```
2018_MM/
├── article_1.json
├── article_2.json
└── ...
```

Each JSON file represents one article with metadata and full text.

---

2. Loading and Cleaning Articles

Articles are parsed from all folders. Malformed or HTML-only files are excluded. Valid articles are cleaned by removing HTML tags, URLs, digits, special characters, and normalizing whitespace.

---

3. Article Filtering

Articles are retained only if they meet quality thresholds:

- Minimum of 20 tokens
- Must include body content
- Duplicates removed

This ensures relevance and consistency across entries.

---

4. Date Normalization and Grouping

The `published` field is extracted and formatted as `YYYY-MM-DD`. Articles are grouped into a dictionary by date for efficient access:

```
{ "YYYY-MM-DD": [article_1, article_2, ..., article_N] }
```

---

5. Sliding Window Construction

A 10-day sliding window approach identifies periods with sufficient article density. A valid window must include at least 10 articles per day across all 10 days.

---

6. Generating Training Examples

Each valid window forms one training sample:

- Ten articles randomly sampled per day
- Tokenized and truncated to 512 tokens
- Saved as a single example, assigned a unique ID

---

7. Splitting and Saving
Examples are shuffled and divided into training (80%), validation (10%), and test (10%) sets. Each example is saved as:

```
example<ID>/
├── 2020-01-01.txt
├── 2020-01-02.txt
└── ...
```

Each file contains 10 articles for that day.

This storage format facilitates efficient data loading during model training and downstream experiments.

Financial Market Prediction System: Comprehensive Technical Specification

This system processes financial news alongside market data to generate predictive embeddings for stock price movements. It employs a dual-stream architecture that fuses textual and numerical insights through multiple processing stages.

Data Dimensions & Architecture Overview

- **Temporal Scope**: 9-day input window to predict metrics for day 10
- **Stocks Analyzed**: 5 tech companies (AAPL, AMZN, GOOGL, META, NFLX)
- **Financial Metrics**: 6 per stock (Open, High, Low, Close, Adj Close, Volume)
- **News Processing**: 10 articles per day, limited to 128 tokens per article

---

Processing Pipeline

1. Data Collection & Preparation

- **Stock Data**: Loaded from CSV file with multi-index structure (metrics × stocks)
- **News Articles**: Organized in date-based folders with 10 articles per day
- **Date Alignment**: Business day detection with nearest-date matching for weekend gaps
- **Data Splitting**: Chronological partitioning into train/validation/test sets

2. News Processing Branch

- **Tokenization**: Each article tokenized with FinBERT tokenizer (up to 512 tokens)
- **Token Selection**: For efficiency, 128 tokens sampled from each article
- **Embedding Extraction**:
    1. Pass selected tokens through FinBERT
    2. Extract 768-dimensional token-level embeddings
    3. Apply PCA to reduce embedding dimensions from 768 → 400
    4. Result: `[batch, 9 days, 10 articles, 128 tokens, 400 features]`
- **Article Aggregation**:
    1. Average across 10 articles per day → `[batch, 9, 128, 400]`
    2. Apply token-level attention (token_attention layer)
    3. Compute weighted sum of token embeddings → `[batch, 9, 400]`
    4. Reduce dimensions with linear projection to 128 features → `[batch, 9, 128]`

3. Financial Metrics Processing Branch

- **Input**: 6 metrics × 5 stocks per day for 9 days → `[batch, 9, 6, 5]`
- **Convolutional Processing**:
    1. Rearrange dimensions to `[batch, metrics, stocks, days]`
    2. Apply separate Conv1D layers for each metric (`kernel_size=3`)
    3. Generate 64 hidden features per metric
    4. Combine features from all metrics → `[batch, 9, 384]`

4. Feature Fusion

- **Concatenation**: Join news (128d) and metrics (384d) features → `[batch, 9, 512]`
- **Dimension Reduction**: Linear projection to 312 dimensions with tanh activation
- **Final Output**: `[batch, 9, 312]` temporal embeddings representing unified market state

5. Memory Optimization Techniques

- **Batch Processing**: Max batch size of 4 examples for lower GPU memory usage
- **Day-by-Day Processing**: Each day processed separately to save memory
- **PCA Reduction**: Early dimensionality reduction (768 → 400) to decrease memory footprint
- **Tensor Offloading**: Moving tensors to CPU after processing to free GPU memory

---

Component Architecture

PCAReducer

- Uses scikit-learn PCA to reduce dimensions while preserving ~92% variance
- Handles batch conversion between PyTorch tensors and NumPy arrays
- Maintains original tensor shapes by only reducing the last dimension

MemoryEfficientNewsProcessor

- Processes one day at a time to minimize memory usage

- Uses attention mechanism to weight tokens by importance
- Applies weighted summation to reduce from token-level to article-level

FinancialConvNet

- Creates separate convolutional layers for each financial metric
- Processes time series data while maintaining the day-by-day structure
- Applies dimension reduction after metric-specific processing

CombinedFinancialModel

- Orchestrates the complete pipeline:
    1. Applies PCA reduction to FinBERT embeddings
    2. Processes news articles through attention mechanism
    3. Processes financial metrics through CNN
    4. Combines both streams and reduces dimensions
    5. Outputs temporal embeddings for downstream prediction

Dataset Statistics

=== Dataset Summary ===
Train split:
• Examples: 345 • News tensor: `torch.Size([345, 9, 10, 128, 768])`
• Metrics tensor: `torch.Size([345, 9, 6, 5])`
• Targets tensor: `torch.Size([345, 30])`
• Embeddings tensor: `torch.Size([345, 9, 312])`

The final embeddings represent a unified temporal representation that captures both semantic meaning from financial news and technical patterns from stock metrics, enabling accurate prediction of market movements.

TinyBERT with LoRA for Stock Market Prediction: Comprehensive Technical Analysis

---

1. **System Architecture Overview**

The **TinyBERT_v2** system implements a sophisticated machine learning pipeline that combines natural language processing with financial time series analysis to predict stock prices. The core innovation is using a compressed language model (TinyBERT) with parameter-efficient fine-tuning (LoRA) to forecast multiple stock metrics simultaneously.

Key System Components

Data Processing Pipeline

The system begins with a preprocessing stage that:

- **Extracts chronological data**: Identifies files named with dates (YYYY-MM-DD) from example folders
- **Creates temporal sequences**: Organizes 9 consecutive days of data for each example
- **Aligns with target data**: Matches the 10th day with corresponding stock price data
- **Handles format conversion**: Remaps between CSV storage format and model-expected format

Model Architecture

- **Base model**: TinyBERT (4-layer, 312D hidden size) — a distilled BERT variant with ~14.5M parameters
- **Fine-tuning approach**: Low-Rank Adaptation (LoRA) with `rank=8`, `alpha=16`
- **Projection head**: Multiple fully-connected layers (312→256→128→30) with dropout regularization
- **Improved version**: Additional normalization, batch normalization, and residual connections

---

2. **Data Flow and Tensor Dimensions**

Input Data Format

- **Embedding tensors**: Shape `[N, 9, 312]`
  - N: Variable batch size (typically 4,000–6,000 examples)
  - 9: Sequence length (9 consecutive trading days)
  - 312: Embedding dimension (aligned with TinyBERT's hidden size)

Target Data Format

- **Stock metrics tensor**: Shape `[N, 30]`
  - 30 = 6 metrics × 5 stocks
  - Metrics: Open, High, Low, Close, Adjusted Close, Volume
  - Stocks: AAPL, AMZN, GOOGL, META, NFLX

Data Transformation

The system handles a critical format remapping between CSV storage and model processing:

- This transformation is implemented in the `remap_targets_to_model_format()` function, which handles the tensor reorganization across an arbitrary batch size.

---

3. **Model Implementation Details**

TinyBERT with LoRA

The core model adapts TinyBERT using LoRA to significantly reduce trainable parameters:

- **LoRA implementation**:
  - Focuses modifications on query and value projection matrices in attention heads

    - Creates parallel "update path" alongside frozen pre-trained weights
    - Uses multiplication with low-rank matrices: `W + ΔW = W + (A×B)×scaling`
    - Reduces trainable parameters from ~14.5M to ~1.8M (87% reduction)

- **Forward pass operations**:

    - Processes input embeddings `[batch_size, 9, 312]` through TinyBERT
    - Uses uniform attention mask to attend to all time steps
    - Applies pooling to obtain `[batch_size, 312]` representation
    - Projects through multiple fully-connected layers to final output

- **Parameter dimensions**:

    - LoRA matrices: A∈ℝ^(in_features×r), B∈ℝ^(r×out_features) where r=8
    - Projection layers: 312×256 (79,872 params), 256×128 (32,896 params), 128×30 (3,870 params)
    - **Total trainable parameters**: ~1.8M (vs ~14.5M if fully fine-tuning)

Improved Model Architecture

The second cell implements an enhanced version with additional components:

- **Data normalization**:

    - MinMax scaling to range [-1, 1] for better stability
    - Fits parameters during first forward pass, then applies consistent transformation

- **Architecture improvements**:

    - Batch normalization after each linear layer to stabilize training
    - Higher dropout rate (0.2 vs 0.1) to improve generalization
    - HuberLoss instead of MSE for robustness against outliers

- **Directional accuracy tracking**:

    - Novel metric capturing correct prediction of price movement direction (up/down)
    - Implemented at both overall and per-stock level
    - Critical for trading strategy evaluation

---

4. **Training Implementation**

Training Process

- **Optimization strategy**:

    - Base version: AdamW optimizer (`lr=3e-5`, `weight_decay=0.01`)
    - Improved version: AdamW with stronger regularization (`lr=1e-5`, `weight_decay=0.05`)

- **Learning rate scheduling**: ReduceLROnPlateau or CosineAnnealing

- **Gradient clipping**: 1.0 (improved version only)

- **Loss calculation**:
    - Focused loss on closing prices (highest business relevance)
    - Secondary tracking of full loss across all metrics
    - Directional accuracy calculation for up/down movement prediction
- **Training loop operations**:
    - Batch processing with `size=16`
    - Validation after each epoch
    - Early stopping with `patience=10 epochs`
    - Model saving based on validation loss

---

5. **Performance Metrics**

- **Standard regression metrics**:
    - Mean Squared Error (MSE): Primary optimization target
    - R^2 Score: Measuring explained variance proportion
    - Huber Loss: More robust to outliers than MSE
- **Financial-specific metrics**:
    - Directional Accuracy: % of correctly predicted up/down movements
    - Per-stock breakdown of all metrics
    - Scaling to handle high-value stocks
- **Visualization**:
    - Training and validation loss curves
    - MSE progression over epochs
    - R^2 progression over epochs
    - Directional accuracy tracking (improved version only)

---

6. **Critical System Workflows**

Data Loading and Preparation

- Load pre-processed embeddings tensors (`.pt` files)
- Load corresponding stock price CSV files
- Match targets with embeddings based on dates
- Create TensorDatasets and DataLoaders with custom collate function
- Apply remapping to correct target format during collation

Model Training Flow

- Initialize model (original or improved version)
- Set up optimizer, scheduler, and trainer
- Execute training loop with validation

- Generate test predictions
- Calculate and save evaluation metrics
- Generate performance visualizations

Prediction Generation

- Process embeddings in batches through trained model
- Extract closing price predictions from full output
- Calculate performance metrics on test set
- Save predictions and metrics to disk for downstream use

Results Storage

- Full predictions tensor (all metrics, all stocks)
- Close-only predictions tensor (focused on closing prices)
- Per-stock metrics in CSV format
- Overall metrics in CSV format
- Directional accuracy metrics in separate CSV
- Visualization plots in PNG format

---

7. **Key Improvements in Enhanced Version**

- **Model architecture improvements**:
    - Input normalization for better numerical stability
    - Batch normalization to stabilize training
    - Residual connections to improve gradient flow
- **Training optimizations**:
    - HuberLoss for robustness against outlier prices
    - Gradient clipping to prevent explosive gradients
    - Higher weight decay (0.05 vs 0.01) for better regularization
    - CosineAnnealing scheduler instead of ReduceLROnPlateau
- **Evaluation enhancements**:
    - Directional accuracy tracking for practical trading applications
    - Per-stock directional accuracy breakdown
    - Scaling of high-value stocks for stable metrics
    - Handling of potential numerical instabilities in R^2 calculation
- **Implementation upgrades**:
    - Sequential ordering in validation/test for directional metrics
    - Explicit tensor detachment and cloning for memory efficiency
    - More comprehensive error handling and edge cases