ALCNet

Adaptive Layer Condensation Networks

Learning Dynamic Compression Ratios for Hierarchical Feature Selection

Author: Ananda Jana

Indian Institute of Science Education and Research
Thiruvananthapuram (IISER TVM), Kerala, India

PyPI Package: https://pypi.org/project/alcnet/

Installation:

```
pip install alcnet
```

Version 0.1.1 — February 2026

1 Overview

ALCNet is a neural network architecture where compression ratios at each layer are learned as trainable parameters rather than fixed before training. This enables task-adaptive architecture optimization without expensive neural architecture search.

1.1 Key Features

- **Learnable Compression Ratios:** Target sparsity $\rho^{(i)} \in (0, 1)$ at each layer adapts during training
- **Task-Adaptive:** Simple tasks learn aggressive compression, complex tasks preserve features
- **Differentiable:** End-to-end training with backpropagation
- **No NAS Required:** Eliminates expensive architecture search
- **Interpretable:** Learned compression ratios reveal task complexity

2 Available Methods

Discovering Available Methods

Use this code to explore all available methods:

```
import alcnet

# List all available classes and functions
print("Available in alcnet:")
print(dir(alcnet))

# Check specific class methods
from alcnet import ALCNet, ALCNetTrainer, CompressionAnalyzer

print("\nALCNet methods:")
print([m for m in dir(ALCNet) if not m.startswith('_')])

print("\nALCNetTrainer methods:")
print([m for m in dir(ALCNetTrainer) if not m.startswith('_')])

print("\nCompressionAnalyzer methods:")
print([m for m in dir(CompressionAnalyzer) if not m.startswith('_')])
```

3 Core Classes and Methods

3.1 ALCNet - Main Model Class

Class: ALCNet

Creates adaptive layer condensation network with learnable compression ratios.

Methods:

- `__init__(layer_sizes, initial_ratios=None, initial_alpha=1.0)`
  Purpose: Initialize network with specified architecture
  Parameters:
  - `layer_sizes` - List of layer dimensions [input, hidden1, ..., output]
  - `initial_ratios` - Initial compression ratios (optional)
  - `initial_alpha` - Sharpness parameter (default: 1.0)
- `forward(x)`
  Purpose: Forward pass through network
  Returns: output, sparsity_loss, ratio_loss
- `get_compression_ratios()`
  Purpose: Get current learned compression ratios for all layers
  Returns: Dictionary mapping layer names to ratio values
- `predict(x)`
  Purpose: Prediction without auxiliary losses
  Returns: Softmax probabilities

3.2 ALCNetLayer - Single Layer

Class: ALCNetLayer

Single layer with learnable compression ratio and soft filtering.

Methods:

- `__init__(input_size, output_size, initial_ratio=0.5)`
  Purpose: Create layer with learnable compression
- `get_compression_ratio()`
  Purpose: Get current compression ratio for this layer
  Returns: Float value in (0, 1)
- `forward(x)`
  Purpose: Forward pass with soft filtering
  Returns: filtered_activations, selection_scores, ratio_loss

3.3 ALCNetTrainer - Training Utilities

Class: ALCNetTrainer

Handles training loop with built-in loss management and tracking.

Methods:

- `__init__(model, optimizer, lambda_sparse=0.001, lambda_ratio=0.01, device='cpu')`
  Purpose: Initialize trainer with model and hyperparameters
  Parameters:
  - `model` - ALCNet instance
  - `optimizer` - PyTorch optimizer
  - `lambda_sparse` - Weight for sparsity loss
  - `lambda_ratio` - Weight for ratio matching loss
  - `device` - 'cpu' or 'cuda'
- `train_epoch(train_loader)`
  Purpose: Train for one epoch
  Returns: average_loss, accuracy
- `validate(val_loader)`
  Purpose: Validate model on validation set
  Returns: validation_loss, validation_accuracy
- `fit(train_loader, val_loader=None, epochs=50, verbose=True)`
  Purpose: Train model for multiple epochs with automatic tracking
  Returns: Training history dictionary
- `save_model(path)`
  Purpose: Save model checkpoint with optimizer state and history
- `load_model(path)`
  Purpose: Load model checkpoint and restore training state

3.4 CompressionAnalyzer - Analysis Tools

Class: CompressionAnalyzer

Visualization and analysis of learned compression patterns.

Methods:

- `__init__(model, device='cpu')`
  Purpose: Initialize analyzer with trained model
- `compute_feature_survival(dataloader, max_batches=10)`
  Purpose: Compute survival probability for input features
  Returns: Dictionary mapping layers to survival probabilities
- `plot_compression_evolution(history)`
  Purpose: Plot how compression ratios evolved during training
  Returns: matplotlib Figure object

- `plot_training_curves(history)`
  Purpose: Plot loss and accuracy curves
  Returns: matplotlib Figure object

- `plot_layer_comparison(compression_ratios_dict)`
  Purpose: Compare learned compression across different tasks
  Returns: matplotlib Figure object

- `get_summary_stats(history)`
  Purpose: Get summary statistics from training
  Returns: Dictionary with accuracy, compression metrics

4 Usage Examples

4.1 Basic Usage

```python
import torch
from alcnet import ALCNet, ALCNetTrainer
import torch.optim as optim

# Create model with learnable compression
model = ALCNet([784, 256, 128, 64, 10])

# Setup optimizer
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Create trainer
trainer = ALCNetTrainer(model, optimizer,
                        lambda_sparse=0.001,
                        lambda_ratio=0.01)

# Train
history = trainer.fit(train_loader, val_loader, epochs=50)

# Check learned compression ratios
print(model.get_compression_ratios())
# Output: {'layer_1': 0.3245, 'layer_2': 0.2891, 'layer_3': 0.1987}
```

4.2 Analysis and Visualization

```python
from alcnet import CompressionAnalyzer

# Create analyzer
analyzer = CompressionAnalyzer(model)

# Get statistics
stats = analyzer.get_summary_stats(history)
print(f"Best accuracy: {stats['best_val_acc']:.2f}%")
print(f"Avg compression: {stats['avg_compression']:.3f}")

# Generate plots
analyzer.plot_training_curves(history)
analyzer.plot_compression_evolution(history)

# Compute feature survival
survival = analyzer.compute_feature_survival(dataloader)
```

4.3 Utility Functions

```python
from alcnet import set_seed

# Set random seed for reproducibility
set_seed(42)

# Get device
from alcnet.utils import get_device, count_parameters

```

```
device = get_device()  # Returns 'cuda' or 'cpu'
num_params = count_parameters(model)
print(f"Model has {num_params:,} parameters")
```

5 Quick Reference

Task	Code
Create model	`model = ALCNet([784, 256, 128, 64, 10])`
Get compression	`model.get_compression_ratios()`
Train model	`trainer.fit(train_loader, val_loader, epochs=50)`
Make prediction	`predictions = model.predict(x)`
Save model	`trainer.save_model('model.pth')`
Analyze results	`analyzer.get_summary_stats(history)`
Plot curves	`analyzer.plot_training_curves(history)`

For more information:

PyPI: https://pypi.org/project/alcnet/
GitHub: https://github.com/anandajana/alcnet

Contact: captainajunited@gmail.com