**Universal Spectral CF Performance Analysis**

**Current Performance**

* **ML-100K NDCG@20**: 0.3880
* **Target Performance**: 0.45+ (13% improvement needed)

**Key Issues Identified**

**1. Eigendecomposition Problems**

# Current: Using similarity-based Laplacian

similarity = normalized @ normalized.t()

laplacian = torch.eye(self.n\_users, device=self.device) - normalized\_sim

**Issues:**

* Cosine similarity may not capture complex user/item relationships
* Standard Laplacian eigendecomposition loses important spectral information
* No consideration of graph structure beyond simple similarity

**Solutions:**

* Use degree-normalized adjacency matrix: D^(-1/2) \* A \* D^(-1/2)
* Consider personalized PageRank or other graph-based similarities
* Implement multi-hop similarities (A², A³) for richer representations

**2. Filter Design Limitations**

# Current: Overly complex filter ensemble

filter\_response = torch.exp(-torch.abs(result).clamp(max=10.0)) + 1e-6

**Issues:**

* Exponential activation may suppress important frequencies
* Complex polynomial combinations without theoretical justification
* No frequency analysis to understand what's being filtered

**Solutions:**

* Use simpler, interpretable filters (e.g., low-pass, band-pass)
* Implement learnable cutoff frequencies based on data characteristics
* Add frequency analysis to understand spectral properties

**3. Training Methodology Issues**

# Current: Simple MSE loss

loss = torch.mean((predicted\_ratings - target\_ratings) \*\* 2)

**Issues:**

* MSE loss doesn't handle ranking well for recommendation
* No negative sampling or contrastive learning
* Training on dense matrices is computationally expensive

**Solutions:**

* Implement BPR (Bayesian Personalized Ranking) loss
* Add negative sampling for better ranking learning
* Use pointwise ranking losses with proper weighting

**4. Model Architecture Problems**

# Current: Simple linear combination

predicted = sum(w \* score for w, score in zip(weights, scores))

**Issues:**

* Linear combination may not capture complex interactions
* No non-linearity between different views
* Personalization layers are too shallow

**Solutions:**

* Add deeper neural architectures for view fusion
* Implement attention mechanisms between views
* Use non-linear transformations between spectral and spatial domains

**Specific Recommendations**

**A. Eigendecomposition Improvements**

def compute\_improved\_eigen(self):

# Use normalized adjacency instead of similarity

adj = self.adj\_tensor

degree = adj.sum(dim=1) + 1e-8

deg\_inv\_sqrt = torch.pow(degree, -0.5)

norm\_adj = adj \* deg\_inv\_sqrt.unsqueeze(0) \* deg\_inv\_sqrt.unsqueeze(1)

# Use largest eigenvalues (not smallest of Laplacian)

eigenvals, eigenvecs = torch.linalg.eigh(norm\_adj)

# Take largest k eigenvalues

eigenvals = eigenvals[-k:]

eigenvecs = eigenvecs[:, -k:]

return eigenvals, eigenvecs

**B. Better Loss Function**

class BPRLoss:

def \_\_init\_\_(self, model, config):

self.model = model

self.opt = torch.optim.Adam(model.parameters(), lr=config['lr'])

def train\_step(self, users, pos\_items, neg\_items):

self.opt.zero\_grad()

user\_embeddings = self.model.get\_user\_embeddings(users)

pos\_scores = self.model.get\_item\_scores(user\_embeddings, pos\_items)

neg\_scores = self.model.get\_item\_scores(user\_embeddings, neg\_items)

# BPR loss

loss = -torch.log(torch.sigmoid(pos\_scores - neg\_scores)).mean()

loss.backward()

self.opt.step()

return loss.item()

**C. Improved Filter Design**

class AdaptiveSpectralFilter(nn.Module):

def \_\_init\_\_(self, filter\_order=6):

super().\_\_init\_\_()

# Learnable cutoff frequencies

self.low\_cutoff = nn.Parameter(torch.tensor(0.1))

self.high\_cutoff = nn.Parameter(torch.tensor(0.8))

self.filter\_strength = nn.Parameter(torch.tensor(1.0))

def forward(self, eigenvalues):

# Normalize eigenvalues to [0, 1]

normalized\_eigs = eigenvalues / (eigenvalues.max() + 1e-8)

# Simple but effective band-pass filter

low\_pass = torch.sigmoid(self.filter\_strength \* (self.low\_cutoff - normalized\_eigs))

high\_pass = torch.sigmoid(self.filter\_strength \* (normalized\_eigs - self.high\_cutoff))

return low\_pass + high\_pass + 0.1 # Ensure non-zero response

**D. Model Architecture Improvements**

class ImprovedSpectralCF(nn.Module):

def forward(self, users):

# Get base collaborative filtering

base\_scores = self.adj\_tensor[users]

# Apply spectral filtering with non-linear fusion

spectral\_features = []

if hasattr(self, 'user\_filter'):

user\_filtered = self.apply\_user\_spectral\_filter(users)

spectral\_features.append(user\_filtered)

if hasattr(self, 'item\_filter'):

item\_filtered = self.apply\_item\_spectral\_filter(users)

spectral\_features.append(item\_filtered)

# Non-linear fusion instead of linear combination

if spectral\_features:

combined\_spectral = torch.cat(spectral\_features, dim=-1)

# Add MLP for better fusion

fused\_features = self.fusion\_mlp(combined\_spectral)

final\_scores = base\_scores + fused\_features

else:

final\_scores = base\_scores

return final\_scores

**Performance Bottlenecks**

**1. Computational Efficiency**

* Eigendecomposition is computed every time → Cache and reuse
* Dense matrix operations → Use sparse operations where possible
* Batch processing inefficiencies → Optimize batch sizes

**2. Hyperparameter Issues**

* Too many filter types → Focus on 2-3 proven effective filters
* Complex polynomial orders → Start with simpler filters (order 3-4)
* Learning rates may be suboptimal → Use learning rate scheduling

**3. Data Preprocessing**

* No feature normalization → Normalize user/item features
* No handling of popular item bias → Add popularity debiasing
* No data augmentation → Consider graph augmentation techniques

**Quick Wins for Immediate Improvement**

1. **Replace MSE with BPR loss** - Should give 2-3% NDCG improvement
2. **Simplify filter ensemble** - Use only user + item views with simple filters
3. **Fix eigendecomposition** - Use proper normalized adjacency matrix
4. **Add negative sampling** - Critical for ranking performance
5. **Implement proper evaluation** - Ensure training items are excluded correctly

**Expected Performance Gains**

* **BPR Loss**: +0.02-0.03 NDCG
* **Better Eigendecomposition**: +0.01-0.02 NDCG
* **Simplified Filters**: +0.01-0.02 NDCG (reduced overfitting)
* **Negative Sampling**: +0.02-0.04 NDCG
* **Architecture Improvements**: +0.01-0.03 NDCG

**Total Expected**: ~0.42-0.45 NDCG@20 (reaching your target!)

**Next Steps**

1. Implement BPR loss as highest priority
2. Simplify to user+item filters only
3. Fix eigendecomposition methodology
4. Add proper negative sampling
5. Validate each change incrementally