Biolord Training and Prediction Pipeline

This notebook contains:

1. Required setup and packages
2. Data preparation with GO features
3. Training pipeline
4. Prediction code verification
5. Suggested prediction code based on findings

Note: Paths and configurations need adjustment for HPC environment.

```
# Required packages
!pip install scanpy
!pip install biolord
import scanpy as sc
import biolord
import numpy as np
import gc
import os
import torch
import psutil
from google.colab import drive

# Mount drive
drive.mount('/content/drive')

# Clear memory
gc.collect()

print(f"Available RAM: {psutil.virtual_memory().available / 1024**3:.1f} GB")
```

Data Preparation

Creating GO features and preparing data with perturbation attributes

```
def get_gene_id_mapping():
    """
    Get mapping between gene symbols and NCBI gene IDs for human genes
    """
    if not os.path.exists('gene_info.gz'):
        print("Downloading gene info...")
        url = "https://ftp.ncbi.nlm.nih.gov/gene/DATA/GENE_INFO/Mammalia/Homo_sapiens.gene_info.gz"
        response = requests.get(url)
        with open('gene_info.gz', 'wb') as f:
            f.write(response.content)

    # Read gene info file to get symbol to ID mapping
    symbol_to_id = {}
    with gzip.open('gene_info.gz', 'rt') as f:
        next(f)  # Skip header
        for line in f:
            fields = line.strip().split('\t')
            gene_id = fields[1]
            symbol = fields[2]
            synonyms = fields[4].split('|')

            # Map both main symbol and synonyms
            symbol_to_id[symbol] = gene_id
            for syn in synonyms:
                if syn:
                    symbol_to_id[syn] = gene_id

    return symbol_to_id

def create_go_features():
    """
    Create GO features for perturbations with proper gene ID mapping
    """
```

```
        # Load your dataset to get perturbation names
        print("Loading dataset...")
        adata = sc.read("NormanWeissman2019_filtered_prepared.h5ad")

        # Get unique perturbations
        pert1 = set(adata.obs['perturbation_1'].unique())
        pert2 = set(adata.obs['perturbation_2'].unique())
        all_perts = list(pert1.union(pert2) - {'control'})

        print(f"Total unique perturbations: {len(all_perts)}")

        # Get gene symbol to ID mapping
        print("Getting gene ID mapping...")
        symbol_to_id = get_gene_id_mapping()

        # Map our perturbation genes to IDs
        pert_to_id = {}
        unmapped_genes = []
        for gene in all_perts:
            if gene in symbol_to_id:
                pert_to_id[gene] = symbol_to_id[gene]
            else:
                unmapped_genes.append(gene)

        print(f"Mapped {len(pert_to_id)} genes to NCBI IDs")
        print(f"Unmapped genes: {len(unmapped_genes)}")
        if unmapped_genes:
            print("First few unmapped genes:", unmapped_genes[:5])

        # Download files if they don't exist
        if not os.path.exists('go.obo'):
            !wget http://purl.obolibrary.org/obo/go.obo

        if not os.path.exists('gene2go.gz'):
            !wget https://ftp.ncbi.nlm.nih.gov/gene/DATA/gene2go.gz

        # Create gene-GO associations
        gene_to_go = {}
        print("Reading gene2go file...")
        with gzip.open('gene2go.gz', 'rt') as f:
            next(f)  # Skip header
            for line in f:
                fields = line.strip().split('\t')
                if len(fields) > 2:
                    tax_id = fields[0]
                    if tax_id == '9606':  # Human genes only
                        gene_id = fields[1]
                        go_id = fields[2]
                        if gene_id in set(pert_to_id.values()):
                            if gene_id not in gene_to_go:
                                gene_to_go[gene_id] = set()
                            gene_to_go[gene_id].add(go_id)

        # Create feature matrix
        print("Creating GO feature matrix...")
        all_go_terms = set()
        for go_terms in gene_to_go.values():
            all_go_terms.update(go_terms)

        go_features = pd.DataFrame(0, index=all_perts, columns=list(all_go_terms))

        for gene in all_perts:
            if gene in pert_to_id:
                gene_id = pert_to_id[gene]
                if gene_id in gene_to_go:
                    for go_term in gene_to_go[gene_id]:
                        go_features.loc[gene, go_term] = 1

        # Save features
        print("Saving GO features...")
        go_features.to_csv('go_features.csv')

        return go_features

    # Run GO features creation
    go_features = create_go_features()
```

Data Preparation with GO Features

Incorporating GO features into the dataset

```
def prepare_training_data():
    """
    Prepare training data with GO features
    """
    # Load data
    print("Loading data...")
    adata = sc.read("NormanWeissman2019_filtered_prepared.h5ad")

    # Load GO features
    print("Loading GO features...")
    go_features = pd.read_csv('go_features.csv', index_col=0)

    print("Creating cell-level GO features...")
    # Create GO feature matrix
    go_matrix = np.zeros((adata.n_obs, go_features.shape[1]))

    print(f"Processing {adata.n_obs} cells...")
    for idx, row in enumerate(adata.obs.itertuples()):
        if idx % 10000 == 0:
            print(f"Processed {idx} cells...")

        pert1 = row.perturbation_1
        pert2 = row.perturbation_2

        # Add GO features for both perturbations
        if pert1 in go_features.index:
            go_matrix[idx] += go_features.loc[pert1].values
        if pert2 in go_features.index:
            go_matrix[idx] += go_features.loc[pert2].values

    # Add GO features to adata
    print("Adding GO features to adata...")
    adata.obsm['go_features'] = go_matrix

    # Save prepared data
    print("Saving prepared data...")
    adata.write('data_with_go_features.h5ad')

    print("Data preparation complete!")
    print(f"Final data shape: {adata.shape}")
    print(f"GO features shape: {adata.obsm['go_features'].shape}")

    return adata

# Run data preparation
adata = prepare_training_data()
```

Training Pipeline

Model configuration and training with both categorical perturbations and ordered GO features Note: Paths may need adjustment for HPC environment

```
def train_model():
    """
    Train biolord model with GO features and perturbations
    """
    # Load data in backed mode
    print("Loading data in backed mode...")
    adata = sc.read('/content/drive/MyDrive/biolord_data/data_with_go_features.h5ad', backed='r')

    # Get training data only
    print("\nPreparing training data...")
    train_mask = (adata.obs['partition'] == 'training')

    # Create temporary filtered dataset
    temp_filename = "temp_train_data.h5ad"
    print("Creating filtered training dataset...")
```

```
        adata_train = sc.AnnData(
            X=adata.X[train_mask],
            obs=adata.obs[train_mask],
            var=adata.var,
            obsm={'go_features': adata.obsm['go_features'][train_mask]}
        )

        # Save filtered data
        adata_train.write(temp_filename)
        del adata  # Free memory
        gc.collect()

        # Load filtered data
        print("Loading filtered data...")
        adata_train = sc.read(temp_filename)

        # Setup model with both ordered and categorical attributes
        print("\nSetting up biolord...")
        biolord.Biolord.setup_anndata(
            adata_train,
            ordered_attributes_keys=['go_features'],
            categorical_attributes_keys=['perturbation_1', 'perturbation_2']
        )

        # Initialize model
        model = biolord.Biolord(
            adata_train,
            n_latent=16,
            module_params={
                'decoder_width': 256,
                'decoder_depth': 1,
                'gene_likelihood': 'normal',
                'reconstruction_penalty': 1e2,
                'unknown_attribute_penalty': 1e1,
                'n_latent_attribute_categorical': 16
            }
        )

        # Training parameters
        trainer_params = {
            'decoder_lr': 1e-3,
            'decoder_wd': 1e-4,
            'attribute_nn_lr': 1e-2,
            'attribute_nn_wd': 4e-8,
            'step_size_lr': 45,
            'cosine_scheduler': True,
            'scheduler_final_lr': 1e-5,
            'n_epochs_warmup': 0
        }

        # Train model
        print("\nTraining model...")
        model.train(
            max_epochs=5,
            batch_size=1024,
            early_stopping=False,
            plan_kwargs=trainer_params,
            enable_checkpointing=False,
            enable_model_summary=False,
            num_sanity_val_steps=0,
            logger=False
        )

        # Save model
        save_path = "/content/drive/MyDrive/biolord_go_model"
        print(f"\nSaving model to {save_path}...")
        model.save(save_path)

        # Clean up
        if os.path.exists(temp_filename):
            os.remove(temp_filename)

        print("Training complete and model saved!")
        return model

    # Run training
    model = train_model()
```

Prediction Code Verification

Testing prediction code logic and structure before full implementation Note: This verifies the code structure without requiring full memory load

```
def test_prediction_logic(n_test_perts=5):
    """
    Test prediction code logic with minimal data processing
    Just to verify the structure works
    """
    try:
        print("Starting prediction logic test...")
        print(f"Available RAM: {psutil.virtual_memory().available / 1024**3:.1f} GB")

        # Load data references only (not full data)
        print("\nLoading data structure...")
        adata = sc.read('/content/drive/MyDrive/biolord_data/data_with_go_features.h5ad', backed='r')

        # Get only perturbation information
        test_mask = (adata.obs['partition'] == 'test')
        test_perts = adata[test_mask].obs[['perturbation_1', 'perturbation_2']].drop_duplicates()
        print(f"\nTotal test perturbation combinations: {len(test_perts)}")
        print("Sample combinations:")
        print(test_perts.head(n_test_perts))

        # Check prediction structure
        print("\nChecking prediction structure...")
        print("Required attributes:")
        print("- Categorical attributes present:", all(x in adata.obs.columns for x in ['perturbation_1', 'perturbation_2']))
        print("- GO features present:", 'go_features' in adata.obsm)
        print("- Gene expression matrix shape:", adata.shape)

        # Verify model structure
        print("\nChecking model structure...")
        model_path = "/content/drive/MyDrive/biolord_go_model/model.pt"
        if os.path.exists(model_path):
            state_dict = torch.load(model_path, map_location='cpu')
            if 'model_state_dict' in state_dict:
                print("Model contains:")
                print("- latent_codes:", 'latent_codes.embedding.weight' in state_dict['model_state_dict'])
                print("- decoder:", any('decoder' in k for k in state_dict['model_state_dict'].keys()))

        print("\nVerifying compute_prediction_adata requirements:")
        print("- Target attributes available:", ['perturbation_1', 'perturbation_2'])
        print("- Features for prediction:", adata.var_names[:5], "...")

        del adata
        gc.collect()

        return True

    except Exception as e:
        print(f"\nError in logic test: {str(e)}")
        print("\nDetailed error information:")
        import traceback
        traceback.print_exc()
        return False

# Run test
print("Starting prediction logic verification...")
success = test_prediction_logic()
```

Suggested Prediction Code for HPC Environment

This code reflects biolord's architecture requirements and handles:

- Full training data loading for latent optimization
- Both categorical perturbations and ordered GO features
- Proper prediction structure based on biolord API

Note: This code is for HPC execution where memory constraints are not an issue, it doesn't work on colab and has not been tested.

```
def make_predictions():
    """
    Generate predictions using trained biolord model.

    This code:
    1. Properly handles full training data requirement for latent optimization
    2. Manages both categorical (perturbations) and ordered (GO features) attributes
    3. Uses biolord's compute_prediction_adata for predictions
    4. Follows biolord's architecture requirements
    """
    try:
        print("Starting prediction pipeline...")

        # Load full dataset (needed for latent optimization and reference)
        print("\nLoading data...")
        adata = sc.read('/content/drive/MyDrive/biolord_data/data_with_go_features.h5ad', backed='r')

        # Get train/test data
        print("\nPreparing data splits...")
        train_mask = (adata.obs['partition'] == 'training')
        adata_train = adata[train_mask].copy(filename="temp_train.h5ad")
        adata_train = sc.read("temp_train.h5ad")

        # Get initial states (control cells) from test set
        print("\nPreparing source data (initial states)...")
        source_mask = (adata.obs['partition'] == 'test') & (adata.obs['perturbation_1'] == 'control')
        adata_source = adata[source_mask].copy(filename="temp_source.h5ad")
        adata_source = sc.read("temp_source.h5ad")

        print("\nSetting up model...")
        biolord.Biolord.setup_anndata(
            adata_train,
            ordered_attributes_keys=['go_features'],
            categorical_attributes_keys=['perturbation_1', 'perturbation_2']
        )

        print("\nLoading model...")
        model_path = "/content/drive/MyDrive/biolord_go_model"
        model = biolord.Biolord.load(model_path, adata=adata_train)

        print("\nAnalyzing test data...")
        test_perts = adata_source.obs[['perturbation_1', 'perturbation_2']].drop_duplicates()
        print(f"Found {len(test_perts)} unique perturbation combinations to predict")

        print("\nGenerating predictions...")
        predictions = model.compute_prediction_adata(
            adata=adata,                # Full data as reference
            adata_source=adata_source,  # Test data as source
            target_attributes=['perturbation_1', 'perturbation_2'],
            add_attributes=['go_features']
        )

        print("\nSaving predictions...")
        predictions.write('/content/drive/MyDrive/biolord_data/predictions.h5ad')

        # Clean up
        for f in ["temp_train.h5ad", "temp_test.h5ad"]:
            if os.path.exists(f):
                os.remove(f)

        print("\nPrediction pipeline complete!")
        print(f"Predictions saved for {len(predictions)} cells")
        return predictions

    except Exception as e:
        print(f"\nError occurred: {str(e)}")
        print("\nDetailed error information:")
        import traceback
        traceback.print_exc()
        return None

# Note: This code is designed for HPC execution
print("Prediction code ready for HPC execution")
```

Suggested Prediction Code for HPC Environment with validation metrics

This code reflects biolord's architecture requirements and includes:

- Full training data loading for latent optimization
- Both categorical perturbations and ordered GO features
- Proper prediction structure based on biolord API
- Validation metrics from biolord paper:
  - R^2 score for prediction accuracy
  - Normalized MSE for perturbation effects
  - Gene-level correlation analysis

```
def make_predictions():
    """
    Generate and evaluate predictions using trained biolord model.

    This code:
    1. Properly handles full training data requirement for latent optimization
    2. Manages both categorical (perturbations) and ordered (GO features) attributes
    3. Uses biolord's compute_prediction_adata for predictions
    4. Implements validation metrics from biolord paper
    """
    try:
        print("Starting prediction pipeline...")

        # Load full dataset (needed for latent optimization and reference)
        print("\nLoading data...")
        adata = sc.read('/content/drive/MyDrive/biolord_data/data_with_go_features.h5ad', backed='r')

        # Get train/test data
        print("\nPreparing data splits...")
        train_mask = (adata.obs['partition'] == 'training')
        adata_train = adata[train_mask].copy(filename="temp_train.h5ad")
        adata_train = sc.read("temp_train.h5ad")

        # Get initial states (control cells) from test set
        print("\nPreparing source data (initial states)...")
        source_mask = (adata.obs['partition'] == 'test') & (adata.obs['perturbation_1'] == 'control')
        adata_source = adata[source_mask].copy(filename="temp_source.h5ad")
        adata_source = sc.read("temp_source.h5ad"))

        print("\nSetting up model...")
        biolord.Biolord.setup_anndata(
            adata_train,
            ordered_attributes_keys=['go_features'],
            categorical_attributes_keys=['perturbation_1', 'perturbation_2']
        )

        print("\nLoading model...")
        model_path = "/content/drive/MyDrive/biolord_go_model"
        model = biolord.Biolord.load(model_path, adata=adata_train)

        print("\nAnalyzing test data...")
        test_perts = adata_source.obs[['perturbation_1', 'perturbation_2']].drop_duplicates()
        print(f"Found {len(test_perts)} unique perturbation combinations to predict")

        print("\nGenerating predictions...")
        predictions = model.compute_prediction_adata(
            adata=adata,                # Full data as reference
            adata_source=adata_source,  # Test data as source
            target_attributes=['perturbation_1', 'perturbation_2'],
            add_attributes=['go_features']
        )

        print("\nEvaluating predictions...")
        from scipy import stats
        import numpy as np
        from sklearn.metrics import r2_score

        # Calculate metrics per perturbation combination
        results = {}
        for pert1 in test_perts['perturbation_1'].unique():
            for pert2 in test_perts['perturbation_2'].unique():
```

```
                # Get relevant cells
                mask_pred = (predictions.obs['perturbation_1'] == pert1) & (predictions.obs['perturbation_2'] == pert2)
                mask_true = (adata_source.obs['perturbation_1'] == pert1) & (adata_source.obs['perturbation_2'] == pert2)

                if mask_pred.sum() > 0 and mask_true.sum() > 0:
                    # Get expression values
                    pred_exp = predictions[mask_pred].X
                    true_exp = adata_source[mask_true].X

                    # 1. R² score (from biolord paper)
                    r2 = r2_score(true_exp, pred_exp)

                    # 2. Normalized MSE (from biolord paper)
                    control_exp = adata_source[adata_source.obs['perturbation_1'] == 'control'].X
                    mse = np.mean((true_exp - pred_exp) ** 2)
                    baseline_mse = np.mean((true_exp - np.mean(control_exp, axis=0)) ** 2)
                    nmse = mse / baseline_mse if baseline_mse != 0 else float('inf')

                    # 3. Pearson correlation per gene
                    correlations = [stats.pearsonr(true_exp[:, i], pred_exp[:, i])[0]
                                    for i in range(true_exp.shape[1])]
                    mean_correlation = np.mean(correlations)

                    results[(pert1, pert2)] = {
                        'r2_score': r2,
                        'normalized_mse': nmse,
                        'mean_gene_correlation': mean_correlation
                        }

            # Calculate aggregate metrics
            avg_metrics = {
                'mean_r2': np.mean([v['r2_score'] for v in results.values()]),
                'mean_nmse': np.mean([v['normalized_mse'] for v in results.values()]),
                'mean_correlation': np.mean([v['mean_gene_correlation'] for v in results.values()])
            }

            print("\nPrediction Results:")
            print(f"Average R² score: {avg_metrics['mean_r2']:.3f}")
            print(f"Average normalized MSE: {avg_metrics['mean_nmse']:.3f}")
            print(f"Average gene correlation: {avg_metrics['mean_correlation']:.3f}")

            print("\nSaving predictions and metrics...")
            predictions.uns['evaluation_metrics'] = {
                'per_perturbation': results,
                'aggregate_metrics': avg_metrics
            }
            predictions.write('/content/drive/MyDrive/biolord_data/predictions.h5ad')

            # Clean up
            for f in ["temp_train.h5ad", "temp_test.h5ad"]:
                if os.path.exists(f):
                    os.remove(f)

            print("\nPrediction pipeline complete!")
            print(f"Predictions saved for {len(predictions)} cells")
            return predictions

        except Exception as e:
            print(f"\nError occurred: {str(e)}")
            print("\nDetailed error information:")
            import traceback
            traceback.print_exc()
            return None

    # Note: This code is for HPC execution
    print("Prediction code ready for HPC execution")
```