Machine Learning for Omics Integration - Day 2 Notes

Unsupervised Multi-Omics Integration

Course Notes

`r Sys.Date()`

Convert to PDF: `pandoc Notes_day2.Rmd -o Notes_day2.pdf --pdf-engine=xelatex -V mainfont="Helvetica Neue" -V monofont="Monaco" -V mathfont="TeX Gyre Termes Math" -V fontsize=12pt -V geometry:margin=1in -V linestretch=1.2`

Continuation of Day 1 laboratory session (lab 2)

Key Challenges with Binary Data

Problem with Binary/Sparse Data

- **Binary data** (e.g., mutation data with mostly 0s and 1s) presents significant challenges
- **DIABLO** has known difficulties handling such sparse binary datasets
- **Recommendation**: Check mixOmics discussion forum for best practices and workarounds

Day 2 lecture ## Pre-analysis Strategy: MOFA

Why Run MOFA First?

- **Purpose**: Understanding relationships between different omics layers before integration
- **Benefit**: Provides insights into data structure and inter-omics correlations
- **Strategy**: Use MOFA as exploratory analysis prior to supervised methods

Unsupervised Machine Learning Philosophy

The "Fishing Expedition" Approach

- **Concept**: Unsupervised ML is like a "fishing expedition"
- **No prior hypothesis**: We don't understand the biological hypothesis beforehand
- **Discovery-driven**: Let the data reveal patterns and relationships

  Key Reference: *"A hypothesis is a liability"* - article published in Genome Biology
  [Link to be added]

MOFA: Multi-Omics Factor Analysis

Overview

- **MOFA & MOFA+**: Leading examples of unsupervised multi-omics integration
- **Methodological approach**: Hybrid of PLS/CCA and Bayesian methods
- **Applications**: Widely used for discovering latent factors across omics layers

Core Concept: Factor Analysis

What is Factor Analysis?

- **Central idea**: All observed data (gene expression, methylation, mutations) are generated by a few **latent variables** (factors/vectors)
- **Goal**: Learn these hidden factors from observed data
- **Process**: Start from observed data □ infer hidden factors that explain the patterns

Mathematical Foundation: Matrix Factorization **Concept**: Decomposing the original data matrix into multiple component matrices

$$X_{ij} = U_{ik} \times V_{kj}$$

Where: - **X**: Original data matrix (samples × features) - **U**: Factor loadings matrix (samples × factors)
- **V**: Factor weights matrix (factors × features) - **k**: Number of latent factors

MOFA Mathematical Framework

Multi-View Factor Model For multiple omics datasets, MOFA extends the basic factor model:

$$X_{ij}^{(m)} = \sum_{k=1}^{K} Z_{ik} \cdot W_{kj}^{(m)} + \epsilon_{ij}^{(m)}$$

Where: - **X^(m)**: Data matrix for omics type m (samples × features) - **Z**: Shared latent factor matrix (samples × factors) - **W^(m)**: Factor loadings for omics m (factors × features) - **K**: Number of latent factors - **ε^(m)**: Noise term for omics m

Bayesian Formulation MOFA uses a **Bayesian approach** with prior distributions:

Factor Prior:

$$Z_{ik} \sim \mathcal{N}(0, 1)$$

Loading Prior (with sparsity):

$$W_{kj}^{(m)} \sim \mathcal{N}(0, (\alpha_k^{(m)})^{-1})$$

Precision Prior (Automatic Relevance Determination):

$$\alpha_k^{(m)} \sim \text{Gamma}(a_0, b_0)$$

Likelihood Functions **For continuous data (e.g., gene expression):**

$$X_{ij}^{(m)} | Z, W^{(m)} \sim \mathcal{N}\left(\sum_{k=1}^{K} Z_{ik} W_{kj}^{(m)}, (\tau^{(m)})^{-1}\right)$$

For count data (e.g., RNA-seq):

$$X_{ij}^{(m)} | Z, W^{(m)} \sim \text{Poisson}\left(\exp\left(\sum_{k=1}^{K} Z_{ik} W_{kj}^{(m)}\right)\right)$$

For binary data (e.g., mutations):

$$X_{ij}^{(m)} | Z, W^{(m)} \sim \text{Bernoulli}\left(\sigma\left(\sum_{k=1}^{K} Z_{ik} W_{kj}^{(m)}\right)\right)$$

Where σ is the sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$

Variational Inference MOFA uses **variational Bayes** to approximate the posterior distribution:

Objective Function (ELBO):

$$\mathscr{L} = \mathbb{E}_q[\log p(X, Z, W, \alpha)] - \mathbb{E}_q[\log q(Z, W, \alpha)]$$

Where: - **p(X, Z, W, α)**: Joint probability of data and parameters - **q(Z, W, α)**: Variational approximation to posterior - **ELBO**: Evidence Lower BOund (maximized during training)

Key Advantages of MOFA

1. **Missing Value Compensation**: Values missing in one omics layer can be compensated by information from other omics layers
2. **Variance Explanation**: Uses R^2 to quantify how much variance is explained by the model
3. **Cross-omics Discovery**: Identifies shared and unique factors across different data types

Applications

scNMT Study

- **Example application**: Single-cell multi-omics integration
- **Reference**: scNMT paper [Link to be added]
- **Demonstrates**: Practical utility in real biological datasets

MOFA vs MOFA+: Detailed Comparison

MOFA (Multi-Omics Factor Analysis)

Core Methodology

- **Statistical Framework**: Bayesian factor analysis with group sparsity
- **Key Innovation**: Handles multiple omics datasets simultaneously
- **Mathematical Foundation**:
  - Assumes data is generated from a low-dimensional latent space
  - Uses variational inference for model fitting
  - Incorporates automatic relevance determination (ARD) for factor selection

MOFA Architecture

```
Input: Multiple omics matrices (RNA-seq, ATAC-seq, Methylation, etc.)
↓
Latent Factor Model: Z (samples × factors)
↓
Factor Loadings: W_m (factors × features) for each omics m
↓
Reconstruction: X_m = Z × W_m + noise
```

Key Features of MOFA

1. **Factor Interpretability**: Each factor can be interpreted biologically
2. **Sparsity**: Automatically selects relevant features and factors
3. **Uncertainty Quantification**: Provides confidence intervals for estimates
4. **Missing Data Handling**: Naturally accommodates missing observations

MOFA+ (MOFA Plus)

Major Improvements Over MOFA

- **Scalability**: Handles much larger datasets (>10,000 samples)
- **GPU Acceleration**: Faster computation using GPU implementations
- **Enhanced Flexibility**: Better handling of different data types and structures
- **Improved Convergence**: More robust optimization algorithms

New Features in MOFA+

1. **Stochastic Variational Inference**: Enables mini-batch processing
2. **Non-Gaussian Likelihoods**: Better modeling of count data, binary data
3. **Smoothness Constraints**: For spatial/temporal data integration
4. **Transfer Learning**: Pre-trained models can be applied to new datasets

When to Use MOFA vs MOFA+

Aspect	MOFA	MOFA+
Dataset Size	< 5,000 samples	> 5,000 samples
Data Types	Continuous/Gaussian	Mixed data types
Computational Resources	CPU-friendly	Requires GPU for large data
Interpretability	High (simpler model)	High (with more complexity)
Development Status	Mature, stable	Active development

MOFA/MOFA+ Workflow

1. Data Preprocessing

```
# Example preprocessing steps
- Log-transformation for count data
- Feature filtering (highly variable features)
- Normalization across samples
- Quality control checks
```

2. Model Training

```
# Basic MOFA model setup
MOFAobject <- create_mofa(data_list)
model_opts <- get_default_model_options(MOFAobject)
model_opts$num_factors <- 10
train_opts <- get_default_training_options(MOFAobject)
MOFAmodel <- run_mofa(MOFAobject, model_opts, train_opts)
```

3. Model Analysis

- **Factor inspection**: Which factors explain most variance?
- **Feature loadings**: Which genes/features drive each factor?
- **Sample scores**: How do samples project onto factors?
- **Variance decomposition**: How much variance per omics layer?

Biological Interpretation of MOFA Results

Factor Types

1. **Shared Factors**: Active across multiple omics layers
    - Often represent fundamental biological processes
    - Examples: Cell cycle, differentiation states, stress responses
2. **Specific Factors**: Active in single omics layer
    - Capture omics-specific technical or biological variation
    - Examples: RNA processing effects, chromatin accessibility patterns

Downstream Analysis

- **Pathway Enrichment**: Gene set analysis on factor loadings
- **Cell Type Identification**: Factor scores as features for clustering
- **Temporal Analysis**: Factor dynamics across time points
- **Clinical Association**: Correlate factors with phenotypes

Advantages of MOFA Approach

Over Traditional Methods

1. **vs PCA**: Handles multiple data types simultaneously
2. **vs CCA**: No requirement for paired samples across all omics
3. **vs Concatenation**: Accounts for different scales and noise levels
4. **vs Individual Analysis**: Identifies shared regulatory mechanisms

Statistical Benefits

- **Dimensionality Reduction**: From thousands of features to ~10-50 factors
- **Noise Reduction**: Separates signal from technical noise
- **Integration**: Leverages complementary information across omics
- **Flexibility**: Accommodates different experimental designs

Limitations and Considerations

MOFA Limitations

1. **Linear Assumptions**: May miss non-linear relationships
2. **Factor Interpretation**: Requires biological expertise
3. **Hyperparameter Tuning**: Number of factors needs careful selection
4. **Computational Complexity**: Can be slow for very large datasets

Best Practices

- **Factor Number Selection**: Use model selection criteria (ELBO, cross-validation)
- **Feature Selection**: Pre-filter for highly variable features
- **Batch Effects**: Address technical confounders before analysis
- **Validation**: Replicate findings in independent cohorts

Case Study Applications

Single-Cell Multi-Omics (scNMT-seq)

- **Data**: Single-cell RNA, DNA methylation, chromatin accessibility
- **Findings**: Identified developmental trajectories and regulatory relationships
- **Impact**: Revealed cell-type-specific regulatory mechanisms

Cancer Multi-Omics

- **Data**: Gene expression, copy number, methylation, mutation
- **Findings**: Discovered pan-cancer molecular subtypes
- **Clinical Relevance**: Biomarkers for treatment stratification

Population Studies

- **Data**: Multi-omics across large population cohorts
- **Findings**: Environmental and genetic factors affecting molecular profiles
- **Applications**: Precision medicine and risk prediction

Model Evaluation

R^2 (R-squared)

- **Definition**: Proportion of variance in the data explained by the model
- **Range**: 0-1 (or 0-100%)
- **Interpretation**: Higher R^2 indicates better model fit and factor explanatory power

Mathematical Formulation of R^2 For each omics layer m and factor k, the variance explained is:

$$R^2_{mk} = \frac{\text{Var}(\text{Predicted}_{mk})}{\text{Var}(\text{Observed}_m)}$$

Total variance explained across all factors for omics m:

$$R_m^2 = \sum_{k=1}^{K} R_{mk}^2$$

Overall model R² (across all omics):

$$R_{\text{total}}^2 = \frac{1}{M} \sum_{m=1}^{M} R_m^2$$

MOFA-Specific Evaluation Metrics

Variance Decomposition The total variance in the data can be decomposed as:

$$\text{Var}(X^{(m)}) = \text{Var}_{\text{explained}} + \text{Var}_{\text{noise}}$$

Where:

$$\text{Var}_{\text{explained}} = \text{Var}\left(\sum_{k=1}^{K} Z_{ik} W_{kj}^{(m)}\right)$$

$$\text{Var}_{\text{noise}} = \text{Var}(\epsilon_{ij}^{(m)})$$

Factor-Specific Variance Contribution For factor k in omics m:

$$\text{Contribution}_{mk} = \frac{\text{Var}(Z_{ik} W_{kj}^{(m)})}{\text{Var}(X_{ij}^{(m)})}$$

Evidence Lower Bound (ELBO) The ELBO objective function can be decomposed as:

$$\mathscr{L} = \underbrace{\mathbb{E}_q[\log p(X|Z, W)]}_{\text{Reconstruction}} - \underbrace{D_{KL}(q(Z)\|p(Z))}_{\text{Factor Regularization}} - \underbrace{D_{KL}(q(W)\|p(W))}_{\text{Loading Regularization}}$$

Where: - **Reconstruction term**: How well the model reconstructs the original data - **KL divergence terms**: Regularization preventing overfitting - **D_KL**: Kullback-Leibler divergence measuring difference between distributions

Model Selection Criteria

ELBO-Based Model Selection Compare models with different numbers of factors K:

$$\text{Best K} = \arg\max_{K} \mathcal{L}(K)$$

Cross-Validation for Factor Selection **K-fold CV procedure:** 1. Split data into K folds 2. For each fold i: Train MOFA on remaining folds, test on fold i 3. Compute CV error:

$$\text{CV Error} = \frac{1}{K}\sum_{i=1}^{K} ||\hat{X}^{(i)} - X^{(i)}||^2$$

4. Select number of factors that minimize CV error

Factor Activity Measure Sparsity of factor k in omics m:

$$\text{Activity}_{mk} = \frac{\text{Number of non-zero } W_{kj}^{(m)}}{\text{Total number of features in omics m}}$$

Biological Validation Metrics

Gene Set Enrichment For factor k, compute enrichment p-value:

$$p_{\text{enrichment}} = P(\text{overlap} \geq \text{observed}|\text{random})$$

Using hypergeometric distribution:

$$P(X = x) = \frac{\binom{K}{x}\binom{N-K}{n-x}}{\binom{N}{n}}$$

Where: - **N**: Total genes in background - **K**: Genes in pathway - **n**: Genes associated with factor - **x**: Overlap between factor and pathway

Factor Reproducibility Correlation between factors across independent datasets:

$$\text{Reproducibility} = \text{cor}(Z_{\text{dataset1}}, Z_{\text{dataset2}})$$

Good reproducibility: correlation > 0.7

Sidenote: read MEFISTO method paper published in nature methods

Artificial Neural Networks for Multi-Omics Integration

Challenges of Deep Learning in Life Sciences

Major Limitations

- **Difficult application to real life science projects**: Most biological data (NGS, tabular omics data) doesn't naturally fit deep learning architectures
- **Lack of sufficient data**: Life sciences typically have small sample sizes compared to image/text domains
- **Simpler methods often outperform**: Traditional ML methods (LASSO, Random Forest) frequently achieve better results
- **Interpretability issues**: Black-box nature conflicts with biological understanding needs

Exceptions Where DL Excels

- **Single-cell omics**: Large cell numbers provide sufficient data
- **Medical imaging**: Microscopy, radiology, pathology images
- **Sequence analysis**: Protein/DNA sequence prediction tasks

General Principles of Artificial Neural Networks (ANN)

Mathematical Foundation

Artificial Neural Networks are **mathematical functions** $Y = f(X)$ with a special architecture characterized by:

$$f(X) = f^{(L)}(f^{(L-1)}(...f^{(1)}(X)))$$

Where each layer l computes:

$$h^{(l)} = g(W^{(l)} \cdot h^{(l-1)} + b^{(l)})$$

Components: - $W^{(l)}$: Weight matrix for layer l - $b^{(l)}$: Bias vector for layer l
- $g(\cdot)$: Activation function (non-linear) - $h^{(l)}$: Hidden layer activations

Network Architecture

Basic Structure

```
Input Layer → Hidden Layer(s) → Output Layer
     ↓              ↓                ↓
  Features     Weighted Sum    Predictions
               + Activation
```

Layer-by-Layer Process

1. **Input Layer**: Raw features (gene expression, methylation, etc.)
2. **Hidden Layer(s)**:
   - Weighted sum: $z = \sum_i w_i x_i + b$
   - Activation function: $a = g(z)$
3. **Output Layer**: Final predictions or classifications

Activation Functions

Common Activation Functions

Function	Formula	Range	Use Case
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0, 1)	Binary classification output
ReLU	$\text{ReLU}(x) = \max(0, x)$	[0, ∞)	Hidden layers (most common)
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1, 1)	Hidden layers (zero-centered)
Softmax	$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	(0, 1)	Multi-class output

Universal Approximation Theorem Multi-layer neural networks with sufficient neurons can approximate any continuous function to arbitrary accuracy.

Deep Learning Architectures for Multi-Omics

1. Autoencoders

Architecture

```
Input → Encoder → Latent Space → Decoder → Reconstruction
 (X)       ↓          (Z)           ↓            (X̂)
```

Mathematical Formulation **Encoder**: $Z = f_{\text{enc}}(X; \theta_{\text{enc}})$ **Decoder**: $\hat{X} = f_{\text{dec}}(Z; \theta_{\text{dec}})$ **Loss Function**: $L = ||X - \hat{X}||^2 + \lambda R(\theta)$

Multi-Omics Application

- **Input**: Concatenated omics data [RNA-seq | Methylation | Proteomics]
- **Latent space**: Compressed representation capturing shared patterns
- **Applications**: Dimensionality reduction, missing data imputation, batch correction

2. Multi-Modal Deep Learning

Early Fusion (Concatenation)

```
RNA-seq     →  \
Methylation →   } → [Concatenated Vector] → Deep Network → Output
Proteomics  →  /
```

Late Fusion (Decision-Level)

```
RNA-seq     → Network₁ →  \
Methylation → Network₂ →   } → Fusion Layer → Output
Proteomics  → Network₃ →  /
```

Training Neural Networks

Backpropagation Algorithm

Forward Pass For each layer l:

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}$$

$$a^{(l)} = g(z^{(l)})$$

Backward Pass (Gradient Descent for Error Minimization) **Loss gradient**: $\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial z^{(l)}}\frac{\partial z^{(l)}}{\partial W^{(l)}}$

Weight update: $W^{(l)} \leftarrow W^{(l)} - \eta\frac{\partial L}{\partial W^{(l)}}$

Loss Functions for Omics

Regression Tasks **Mean Squared Error**: $L_{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

Classification Tasks **Cross-Entropy**: $L_{CE} = -\sum_{i=1}^{n} y_i \log(\hat{y}_i)$

Multi-Omics Deep Learning Applications

1. Cancer Subtyping

- **Input**: Gene expression + Methylation + Copy number + Mutation data
- **Architecture**: Multi-modal autoencoder → Clustering in latent space
- **Output**: Cancer subtypes with clinical relevance

2. Drug Response Prediction

- **Input**: Cell line omics profiles + Drug molecular features
- **Architecture**: Deep neural networks with multi-omics fusion
- **Output**: IC50 values or binary response predictions

3. Single-Cell Multi-Omics Integration

- **Input**: scRNA-seq + scATAC-seq + scProteomics
- **Architecture**: Variational autoencoders with modality-specific encoders
- **Output**: Integrated cell embeddings for trajectory analysis

Limitations and Alternatives

When NOT to Use Deep Learning

- **Small datasets**: < 1000 samples typically insufficient
- **High interpretability requirements**: Use linear models instead
- **Simple relationships**: Traditional ML often sufficient and faster

Better Alternatives for Small Data

- **MOFA/MOFA+**: Factor analysis approaches
- **Kernel methods**: SVM with multiple kernel learning
- **Ensemble methods**: Random Forest, XGBoost
- **Network-based methods**: Graph-based integration approaches

Gradient Descent and Backpropagation

Mathematical Formulation of Gradient Descent

Gradient descent is the cornerstone optimization algorithm for training neural networks. The objective is to minimize the loss function $L(\theta)$ by iteratively updating parameters θ in the direction of steepest descent.

Basic Gradient Descent Update Rule:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

Where: - θ_t are the parameters at iteration t - η is the learning rate (crucial hyperparameter) - $\nabla_\theta L(\theta_t)$ is the gradient of the loss function with respect to parameters

Backpropagation Algorithm

Backpropagation computes gradients efficiently using the chain rule of calculus. For a neural network with layers $l = 1, 2, ..., L$:

Forward Pass:

$$z^{(l)} = W^{(l)} h^{(l-1)} + b^{(l)}$$

$$h^{(l)} = g(z^{(l)})$$

Backward Pass:

$$\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}}$$

For the output layer ($l = L$):

$$\delta^{(L)} = \frac{\partial L}{\partial h^{(L)}} \odot g'(z^{(L)})$$

For hidden layers ($l = L-1, L-2, ..., 1$):

$$\delta^{(l)} = [(W^{(l+1)})^T \delta^{(l+1)}] \odot g'(z^{(l)})$$

Parameter Gradients:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} (h^{(l-1)})^T$$

$$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$$

Numeric Implementation Considerations

1. Learning Rate Selection: - Too large: oscillations, divergence - Too small: slow convergence - Common strategies: learning rate schedules, adaptive methods (Adam, RMSprop)

2. Gradient Descent Variants:

Stochastic Gradient Descent (SGD):

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L_i(\theta_t)$$

Mini-batch Gradient Descent:

$$\theta_{t+1} = \theta_t - \eta \frac{1}{|B|} \sum_{i \in B} \nabla_\theta L_i(\theta_t)$$

Adam Optimizer:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Learning Rate as Critical Hyperparameter

The learning rate η significantly impacts training dynamics:

Adaptive Learning Rate Strategies: - **Step decay:** $\eta_t = \eta_0 \cdot \gamma^{\lfloor t/s \rfloor}$ - **Exponential decay:** $\eta_t = \eta_0 \cdot e^{-\lambda t}$ - **Cosine annealing:** $\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})(1 + \cos(\frac{\pi t}{T}))$

Learning Rate Scheduling for Multi-Omics: Given the complexity and high dimensionality of omics data, careful learning rate management is crucial: - Start with higher rates for initial feature learning - Reduce rates as fine-tuning progresses - Use validation loss plateaus to trigger rate reductions

Practical Implementation for Multi-Omics Integration

Challenge-Specific Considerations:

1. **High Dimensionality:** Use gradient clipping to prevent exploding gradients
2. **Batch Normalization:** Helps stabilize training with different omics scales
3. **Regularization:** L1/L2 penalties added to loss function:

$$L_{total} = L_{data} + \lambda_1 ||W||_1 + \lambda_2 ||W||_2^2$$

Example: Multi-Modal Loss Function

$$L_{total} = L_{genomics} + L_{proteomics} + L_{metabolomics} + L_{integration}$$

Where each component can have different learning rates:

$$\theta_{genomics} = \theta_{genomics} - \eta_g \nabla L_{genomics}$$

$$\theta_{proteomics} = \theta_{proteomics} - \eta_p \nabla L_{proteomics}$$

ANN from Scratch: Problem Formulation

Key Insight: Neural Networks are About Derivatives, Not Black Boxes

> **Core Principle**: Neural networks are fundamentally about **gradients and derivatives** - they are interpretable mathematical functions, not mysterious black boxes.

Mathematical Prerequisites for Understanding ANNs

Essential Calculus Knowledge **Time Investment for Learning Derivatives:**

Background	Learning Timeline	Focus Areas
No Calculus	2-3 weeks intensive study	Chain rule, partial derivatives, gradients
Basic Calculus	3-5 days focused review	Multivariate calculus, chain rule applications

Strong Math Background	1-2 days	Review notation, vector calculus

Core Derivative Concepts for ANNs **1. Chain Rule (Most Critical)**

$$\frac{d}{dx}[f(g(x))] = f'(g(x)) \cdot g'(x)$$

Example for Neural Networks:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

2. Partial Derivatives For function $f(x, y)$:

$$\frac{\partial f}{\partial x} = \text{derivative with respect to x, treating y as constant}$$

3. Vector Gradients

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right]$$

Minimal ANN Implementation (≈10 Lines of Code)

Simple Neural Network from Scratch:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Network parameters
np.random.seed(42)
weights_input_hidden = np.random.uniform(-1, 1, (2, 3))  # 2 inputs, 3 hidden ne
weights_hidden_output = np.random.uniform(-1, 1, (3, 1))  # 3 hidden, 1 output

# Training data (XOR problem)
```

```
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

# Training loop
for epoch in range(10000):
    # Forward pass
    hidden_layer = sigmoid(np.dot(X, weights_input_hidden))
    output_layer = sigmoid(np.dot(hidden_layer, weights_hidden_output))

    # Backward pass (the derivatives in action!)
    output_error = y - output_layer
    output_delta = output_error * sigmoid_derivative(output_layer)

    hidden_error = output_delta.dot(weights_hidden_output.T)
    hidden_delta = hidden_error * sigmoid_derivative(hidden_layer)

    # Update weights (gradient descent)
    weights_hidden_output += hidden_layer.T.dot(output_delta) * 0.1
    weights_input_hidden += X.T.dot(hidden_delta) * 0.1

print("Final predictions:", output_layer.round(3))
```

Why Understanding Derivatives Matters for Omics

1. Interpretability

- **Gradient analysis** reveals which genes/features most influence predictions
- **Sensitivity analysis** shows how changes in expression affect outcomes
- **Feature importance** derived from partial derivatives

2. Debugging and Optimization

- **Vanishing gradients**: When derivatives → 0, learning stops
- **Exploding gradients**: When derivatives → ∞, training becomes unstable
- **Learning rate tuning**: Based on gradient magnitude

3. Multi-Omics Specific Applications **Gradient-Based Feature Selection:**

$$\text{Importance}_i = \left|\frac{\partial L}{\partial x_i}\right|$$

Cross-Omics Gradient Analysis:

$$\frac{\partial L}{\partial \text{Gene}_i} \text{ vs } \frac{\partial L}{\partial \text{Protein}_j}$$

Learning Path Recommendations

Week 1-2: Calculus Fundamentals

- **Khan Academy**: Derivatives and Chain Rule
- **3Blue1Brown**: "Essence of Calculus" YouTube series
- **Focus**: Chain rule, partial derivatives, gradients

Week 3: Applied to Neural Networks

- **Implement**: Simple ANN from scratch (above code)
- **Understand**: How backpropagation uses chain rule
- **Practice**: Calculate gradients by hand for 2-layer network

Week 4: Multi-Omics Applications

- **Gradient-based interpretation** of trained models
- **Feature importance** analysis using derivatives
- **Cross-omics gradient** correlation analysis

Quick Self-Assessment

Can you compute these derivatives?

1. $\frac{d}{dx}[\sigma(wx+b)]$ where $\sigma(x) = \frac{1}{1+e^{-x}}$
2. $\frac{\partial}{\partial w}[(y - \sigma(wx+b))^2]$
3. Chain rule for: $\frac{\partial L}{\partial w_1}$ in a 3-layer network

If yes: You're ready for ANN implementation **If no**: Spend 1-2 weeks on calculus review

The “10 Lines” Philosophy

Key Point: Once you understand derivatives, neural networks become surprisingly simple: - **Forward pass**: Matrix multiplications + activation functions - **Backward pass**: Chain rule application - **Weight update**: Gradient descent

The complexity isn’t in the math - it’s in: - Architecture design - Hyperparameter tuning - Data preprocessing - Biological interpretation

10 Lines of Code for ANN:

```
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)

# Network parameters
np.random.seed(42)
weights_input_hidden = np.random.uniform(-1, 1, (2, 3))  # 2 inputs, 3 hidden ne
weights_hidden_output = np.random.uniform(-1, 1, (3, 1))  # 3 hidden, 1 output

# Training data (XOR problem)
X = np.array([[0,0], [0,1], [1,0], [1,1]])
y = np.array([[0], [1], [1], [0]])

# Training loop
for epoch in range(10000):
    # Forward pass
    hidden_layer = sigmoid(np.dot(X, weights_input_hidden))
    output_layer = sigmoid(np.dot(hidden_layer, weights_hidden_output))

    # Backward pass (the derivatives in action!)
    output_error = y - output_layer
    output_delta = output_error * sigmoid_derivative(output_layer)

    hidden_error = output_delta.dot(weights_hidden_output.T)
```

```
    hidden_delta = hidden_error * sigmoid_derivative(hidden_layer)

    # Update weights (gradient descent)
    weights_hidden_output += hidden_layer.T.dot(output_delta) * 0.1
    weights_input_hidden += X.T.dot(hidden_delta) * 0.1

print("Final predictions:", output_layer.round(3))
```

Autoencoders vs MOFA: Comparative Analysis

Autoencoder Fundamentals

Definition: Autoencoders are **unsupervised neural networks** designed to learn compressed representations of input data.

Core Objective: Make the encoder and decoder reconstruct the input as perfectly as possible through a compressed bottleneck layer.

Architecture Comparison

```
Autoencoder:    Input → Encoder → Bottleneck → Decoder → Output (≈ Input)
MOFA:        Multi-Omics → Factor Analysis → Latent Factors → Reconstruction
```

Mathematical Relationship to MOFA

Autoencoder Loss Function:

$$L_{AE} = ||X - \text{Decoder}(\text{Encoder}(X))||^2$$

MOFA Reconstruction:

$$L_{MOFA} = ||X^{(m)} - ZW^{(m)}||^2 + \text{regularization}$$

Key Similarity: Both methods aim to summarize high-dimensional data (thousands of genes/features) into a low-dimensional representation (2-50 factors/latent dimensions).

When Autoencoders Are Useful vs Not Useful

Autoencoders Excel At:

1. **Large datasets**: > 10,000 samples (more data than typical genomics)
2. **Non-linear relationships**: Can capture complex patterns MOFA might miss
3. **Image-like data**: Spatial genomics, microscopy images
4. **Missing data imputation**: Learn to fill gaps in multi-omics datasets

Autoencoders Limitations in Genomics:

1. **Small sample sizes**: Most genomics studies < 1,000 samples
2. **Interpretability**: Black-box nature vs MOFA's interpretable factors
3. **Overfitting risk**: Too many parameters for genomics datasets
4. **Better alternatives exist**: MOFA, PCA, factor analysis often superior

MOFA vs Autoencoder Trade-offs

Aspect	MOFA	Autoencoder
Sample size	Works with < 500 samples	Needs > 1,000 samples
Interpretability	High (factor meanings)	Low (black box)
Multi-omics handling	Native support	Requires concatenation
Missing data	Natural handling	Requires preprocessing
Non-linear patterns	Limited (linear model)	Excellent
Computational cost	Moderate	High (GPU needed)
Biological insight	High	Limited

Practical Recommendation

For typical genomics projects: **Use MOFA first** - Provides interpretable biological insights - Handles small sample sizes well - Better suited for multi-omics integration - Established in genomics literature

Consider autoencoders when: - Dataset > 5,000 samples - Strong non-linear relationships suspected - Working with spatial/imaging data - Need missing data imputation

Critical Insight: Context Matters

> **"Autoencoder dimension reduction: somewhat useful, but there are better techniques"**

This reflects the reality that **simpler methods often outperform complex ones** in life sciences, especially with limited data.

Deep learning limitations in critical applications: - **Life-death situations**: Medical diagnostics require interpretability - **Safety-critical domains**: Earthquake prediction, drug discovery - **Regulatory approval**: Black-box models difficult to validate

The "right tool for the job" principle: Match method complexity to: - Data size - Interpretability requirements
- Domain constraints - Validation needs