POLITECNICO
MILANO 1863

Multidisciplinary Project (5 CFU)

A Reproducible PyTorch Implementation of Gated Linear Networks

Online Learning, Benchmarking, and a TCGA-BRCA Case Study

Instructors:
Prof. Marcello Restelli

Authors:
Federico Lolli <10720843@polimi.it>

A.Y. 2025-2026

Abstract

Gated Linear Networks (GLNs) replace end-to-end backpropagation with context-gated geometric mixing and purely local convex updates. This report's primary contribution is a clean, modular, and reproducible PyTorch implementation of GLNs, designed to make the architecture easy to inspect, extend, and benchmark under controlled training regimes. The current implementation supports both paper-faithful local online learning (OGD) and batch optimization for ablations. Validation is provided on MNIST and on TCGA-BRCA tumor vs. normal classification (a setting with far more features than samples). While TCGA confirms that high accuracy is attainable under local OGD, it should be considered a partial failure from an interpretability perspective: Integrated Gradients and saliency do not yield robust gene-level rankings aligned with known breast cancer markers, suggesting that fixed random half-space gating can be incompatible with biomarker-style explanations even when predictive performance is strong.

Contents

1. Introduction

Gated Linear Networks (GLNs) are a class of neural networks introduced by J. Veness *et al.* [1] that replace global backpropagation with local convex learning and a contextual gating mechanism. Despite their theoretical appeal (online learning, modularity, and a natural notion of "active-path" explanations), available implementations are often hard to reproduce due to outdated dependencies and vague experimental settings.

This work provides:

i. A clean, modular GLN implementation in PyTorch[1] designed for extension and controlled experimentation, following best practices for reproducible research and GPU acceleration.
ii. An empirical study on a real high-dimensional biomedical task (TCGA-BRCA mRNA-based tumor vs. normal classification [3]).

Beyond predictive performance, an analysis of interpretability is performed using Saliency attribution and Integrated Gradients [4] methods, revealing important constraints when applying explainability techniques to GLNs.

1.1. Project Objective

Implement the GLN architecture from first principles in PyTorch (including geometric mixing and contextual gating), with emphasis on a reproducible and extensible codebase, and evaluate it on a real-world high-dimensional genomic classification task (TCGA-BRCA tumor vs. normal), focusing on both predictive performance and interpretability.

1.2. Motivation

Two needs motivate this project. First, GLNs are typically presented as a practically attractive alternative to deep networks because each unit optimizes a local convex objective, enabling efficient online learning without end-to-end gradient credit assignment. Second, genomic classification operates in an extreme $p \gg n$ regime (tens of thousands of genes and relatively few samples), where reproducibility and interpretability are as important as raw accuracy. A modular, well-specified implementation is required to assess whether GLNs' theoretical properties translate into reliable gains on real biomedical data.

1.3. Problem Statement

Given TCGA Breast Invasive Carcinoma (BRCA) transcriptomic profiles with 20k mRNA expression features per sample, learn a binary classifier that distinguishes tumor from normal tissue. The core challenges are (i) very high input dimensionality, (ii) class imbalance, and (iii) the requirement to understand which biological features drive decisions rather than only achieving high test accuracy.

1.4. Report Structure

The report is split into two parts:

- **Part I** introduces the theory behind GLNs (local learning, geometric mixing, and half-space contextual gating) and describes the PyTorch implementation choices that enable reproducible experiments and architectural variations.
- **Part II** presents the TCGA-BRCA experimental setup and results, then focuses on interpretability: attribution with Integrated Gradients and Saliency attribution methods, followed by a discussion of the observed accuracy-interpretability tension in high-dimensional genomics.

[1] de-facto standard for the deep learning research community, see A. Paszke *et al.* [2]

Part I: Gated Linear Networks — Theory and Implementation

Focus: Understanding the GLN architecture and building it from first principles.

2. Theoretical Foundation

Gated Linear Networks (GLNs) are fundamentally based on two different learning mechanics [1]:

i. **geometric mixing**: a logit-space mixture that maps probability features back to a probability while keeping the Bernoulli log-loss convex in the mixing weights
ii. **contextual gating**: a discrete selection mechanism that chooses among multiple expert weight vectors based on side information.

Stacking such gated geometric-mixing units results in a deep architecture that admits an online training rule with purely local updates and strong theoretical guarantees.

2.1. Probability Features and Stable Logits

A key architectural constraint in [1] is that each layer consumes *probability features* rather than unconstrained real-valued activations. In the paper's formulation, these initial probabilities are provided by a *base model* (or *base transformation*) that maps the raw input into $(0, 1)$; the GLN then composes gated geometric-mixing units on top of this probabilistic interface.

In this work, following the practical choice suggested by DeepMind's reference implementation [5], the base transform is a feature-wise sigmoid. To make this mapping invariant to affine shifts in location and scale, the raw vector $\boldsymbol{x} \in \mathbb{R}^p$ is first standardized using training-set statistics:

$$\boldsymbol{x}' = \frac{\boldsymbol{x} - \boldsymbol{\mu}}{\boldsymbol{s}},$$

where $\boldsymbol{\mu}$ and $\boldsymbol{s}$ are per-feature mean and standard deviation estimated on the training set. The base probabilities are then obtained as

$$\boldsymbol{p} = \mathrm{clip}(\sigma(\boldsymbol{x}'), \varepsilon, 1 - \varepsilon) \in (0, 1)^p,$$

with σ the logistic sigmoid and ε a small constant to prevent numerical issues at the boundaries. This stage provides the bounded probabilistic features required by the GLN units and reduces immediate sigmoid saturation due to arbitrary input scales.

The network then operates on $\boldsymbol{p}$ through the stable logit transform

$$\sigma_\varepsilon^{-1}(p) = \mathrm{logit}(\mathrm{clip}(p, \varepsilon, 1 - \varepsilon)) = \ln\left(\frac{p}{1 - p}\right).$$

2.2. Geometric Mixing

Let $\boldsymbol{p} \in (0, 1)^d$ be the vector of probability features for a neuron (after optional bias concatenation), and let $\boldsymbol{w} \in \mathbb{R}^d$ be its mixing weights. The geometric-mixing prediction is

$$\hat{p} = \sigma\big(\boldsymbol{w}^\top \sigma_\varepsilon^{-1}(\boldsymbol{p})\big).$$

The same prediction can be written more intuitively as a product-of-experts by moving to odds space. Define the odds of each input feature as o_i and the combined odds as o:

$$o_i := \frac{p_i}{1 - p_i}, \qquad\qquad o := \prod_i o_i^{w_i}$$

Converting back to a probability gives

$$\hat{p} = \frac{o}{1+o} = \frac{\prod_i p_i^{w_i}}{\prod_i p_i^{w_i} + \prod_i (1-p_i)^{w_i}}.$$

Given Bernoulli target $y \in \{0, 1\}$, the local log-loss is

$$\ell(y, \hat{p}) = -y \ln \hat{p} - (1-y) \ln(1-\hat{p}).$$

Under geometric mixing, ℓ is convex in $\boldsymbol{w}$ [1]. Moreover, the gradient has a particularly simple closed form:

$$\nabla_w \ell = (\hat{p} - y)\sigma_\varepsilon^{-1}(\boldsymbol{p}).$$

This is the algebraic reason GLNs admit stable, layer-local Online Gradient Descent (OGD) updates without backpropagation.

2.3. Contextual Gating via Random Half-Spaces

Geometric mixing alone is a generalized linear model in logit space. Capacity is increased by equipping each neuron with a **table** of expert weights indexed by a discrete context.

In the implemented GLN, each neuron j samples K random half-spaces over a context vector $\boldsymbol{z} \in \mathbb{R}^q$:

$$h_{\{j,k\}}(\boldsymbol{z}) = \left[\boldsymbol{v}_{\{j,k\}}^\top \boldsymbol{z} \geq b_{\{j,k\}}\right], \quad k \in \{0, 1, ..., K-1\},$$

where $\boldsymbol{v}_{\{j,k\}}$ is sampled from a standard normal distribution and normalized to unit length, and $b_{\{j,k\}}$ is sampled from a standard normal distribution. The resulting bit-vector $\boldsymbol{h}_j \in \{0, 1\}^K$ is packed into an integer index

$$c_{j(\boldsymbol{z})} = \sum_{\{k=0\}}^{\{K-1\}} 2^k h_{\{j,k\}}(\boldsymbol{z}) \in \{0, ..., 2^K - 1\}.$$

Each neuron therefore stores a weight tensor

$$W_j \in \mathbb{R}^{2^K \times d}, \quad (W_j)_{\{c,:\}} = \boldsymbol{w}_{\{j,c\}},$$

and uses $\boldsymbol{w}_{\{j,c_{j(\boldsymbol{z})}\}}$ as its active expert on input $\boldsymbol{z}$. In this work, the context is taken to be the input itself (transformed as seen in Section 2.1), and it is treated as fixed side information during local updates.

2.4. Bias as a Probability Feature

To provide an intercept term while preserving the probability-feature interface, the implementation uses a learnable *bias probability* $b \in (0, 1)$ appended as an additional input coordinate (and similarly appended between hidden layers). The bias is parameterized by an unconstrained scalar $r \in \mathbb{R}$ via $b = \sigma(r)$ and then concatenated to the feature vector, yielding an always-valid Bernoulli feature.

3. PyTorch Implementation

The implementation, provided alongside this document, closely follows the decomposition suggested by J. Veness *et al.* [1], while exposing two complementary training modes (Section 3.2).

3.1. Tensorization and Indexing

For a layer with `size` neurons, input dimension d, and context dimension K, the layer stores

$$W \in \mathbb{R}^{\text{size} \times 2^K \times d}$$

Given a batch of contexts, the half-space tests produce a boolean tensor of shape $(\text{size}, \text{batch}, K)$, which is packed into indices $c \in \{0, ..., 2^K - 1\}$ per neuron and sample. The corresponding expert vectors $W[j, c] \in \mathbb{R}^d$ are gathered efficiently to compute the forward pass in a fully vectorized manner.

3.2. Two Training Regimes

The codebase supports two distinct ways of fitting the same architecture.

3.2.1. (A) Paper-faithful Online OGD (Local Learning)

In the online regime, each layer performs local OGD updates on its active expert weights, treating the inputs from the previous layer as fixed probability features. For a neuron with active expert weights $\boldsymbol{w}_t$ at time t, local feature vector $\boldsymbol{p}_t$ and learning rate η_t, the update implemented is

$$\boldsymbol{w}_{\{t+1\}} = \Pi_W(\boldsymbol{w}_t - \eta_t(\hat{p}_t - y_t)\sigma_\varepsilon^{-1}(\boldsymbol{p}_t)),$$

where Π_W is the Euclidean projection onto a compact convex set W.

In this implementation, W is chosen as a coordinate-wise hypercube $[w_{\min}, w_{\max}]^d$ (defaulting to $[0, 1]^d$), so projection is an inexpensive clamp operation. The learning rate schedule is configurable, and includes the paper-standard decay

$$\eta_t = \frac{\eta_0}{\sqrt{t}},$$

which yields an $O\left(\sqrt{T}\right)$ regret bound for OGD under standard assumptions (convex loss, bounded gradients, bounded domain). In practice, the implementation also provides a vectorized variant of the update: within a batch, gradients corresponding to samples that activate the same (neuron, context) expert are aggregated before applying the step, trading exact per-sample updates for speed.

3.2.2. (B) Batch Adam (End-to-end Optimization for Ablations)

For empirical comparisons and controlled experiments, the same GLN can be trained in a batch regime using standard end-to-end optimization. In this mode, the empirical risk is minimized:

$$\min_\theta \left(\frac{1}{n}\right) \sum_{\{i=1\}}^{n} \ell\left(y_i, f_{\theta(\boldsymbol{p}_i)}\right),$$

where θ collects all learnable parameters (in particular, the expert weight tables and bias parameters) and f_θ is the compositional forward map of the stacked gated layers.

Optimization is performed with Adam in the experiments reported in this work, but the implementation is agnostic to the specific choice of optimizer: any PyTorch-compatible optimizer (e.g., SGD, AdamW, RMSprop, Adagrad) can be plugged in to update the expert weight tables and bias parameters. Since contextual gating is discrete, the resulting objective is piecewise smooth: for a fixed set of active experts, gradients flow to the selected weights, while the gating boundaries remain fixed (hyperplanes are not trained). To stabilize training across optimizers, weights are optionally clamped to a finite range after each optimizer step.

The two regimes should be interpreted as complementary: online OGD matches the theoretical GLN learning rule and supports regret-style guarantees, whereas batch Adam intentionally relaxes the local-learning principle to enable ablations and stress-tests in the high-dimensional settings.

3.3. Accelerator Compatibility and Vectorized Execution

All computations are expressed as standard PyTorch tensor operations and run unchanged on any backend supported by PyTorch's `device` interface (CPU, CUDA, MPS). Inputs, weight tables, gating hyperplanes, and intermediate tensors are created on the selected device, and both the forward pass

(gating, expert selection, geometric mixing) and the optional batched OGD update are implemented without Python-level loops, relying instead on batched indexing/gather and fused elementwise ops.

3.4. Reproducibility Benchmark (MNIST)

Results on the MNIST dataset [6] are provided as an external reproducibility benchmark. The dataset is small enough to enable quick end-to-end runs and structured to evaluate (i) multiclass training, (ii) non-trivial generalization after short training, and (iii) interpretable saliency patterns consistent with digit strokes.

The benchmark suite lives in `gated-linear-networks/benchmarks`. The most reliable reproduction instructions (installation, CLI options, and output paths) are given in the benchmark README in that directory.

The benchmark trains a `MulticlassGLN` via a one-vs-all decomposition. Two training modes are available: paper-faithful online OGD (default) and batch backpropagation. The run summarized here uses online OGD and then generates a 2×5 saliency grid over the ten digit classes (see Figure 2). The hyperparameter configuration and the per-class test accuracies are summarized in the tables below.

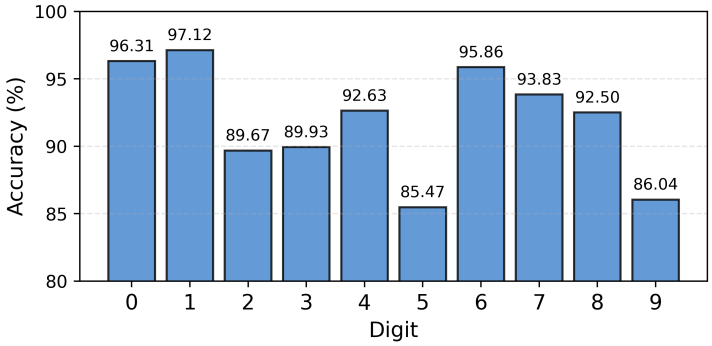

Figure 1: Barplot of the MNIST per-class test accuracies

Param.[2]	Value
E	1
M	1
η_t	0.01
L_1, L_2	50, 25
K	6
$t \mapsto \eta_t$	sqrt

Table 1: MNIST benchmark hyperparameters used for the run.

On this configuration, the overall test accuracy was **92.03%**. Per-class accuracies are reported in Table 1; the most challenging classes in this run were 5 and 9, while 1 achieved the highest accuracy.

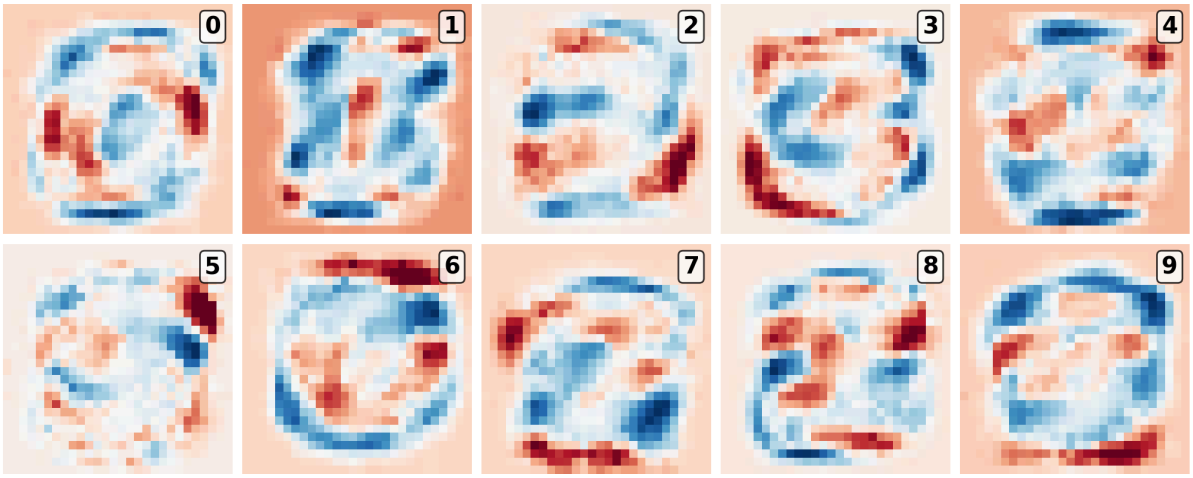

Figure 2: MNIST signed saliency maps per digit class computed from the GLN's collapsed active weights. The patterns are qualitatively consistent with reference visualizations (salient strokes align with digit morphology), but are not expected to match exactly across runs due to random half-space partitions, seed sensitivity, and small differences in preprocessing and learning-rate schedules.

[2]Parameters description can be found in the Appendix Section I

Part II: Application to Genomic Data

Focus: Applying GLN to real cancer data and investigating what the model learns.

4. The TCGA-BRCA Dataset

This work uses TCGA Breast Invasive Carcinoma (BRCA) data and frames it as a binary classification problem: given a vector of mRNA expression measurements, predict whether a tissue sample is **tumor** or **normal**. At a high level, each sample is represented by a high-dimensional feature vector (one feature per gene), which matches the setting described in the TCGA BRCA reference study [3]. Concretely, the files used in this project are obtained through the cBioPortal ecosystem [7].

4.1. Data Source and Acquisition

Data are retrieved from the public cBioPortal DataHub repository [8], specifically from the curated study `brca_tcga_pan_can_atlas_2018`. To make acquisition reproducible and lightweight, the loader downloads only the two required expression matrices directly via HTTP from GitHub (rather than cloning the full repository):

- tumor samples: `data_mrna_seq_v2_rsem.txt`
- normal samples: `normals/data_mrna_seq_v2_rsem_normal_samples.txt`

Each file is a tab-separated *expression matrix*, whose rows index genes and whose columns index samples. Each entry $e_{g,s}$ represents the (RSEM-based) expression estimate of gene g in sample s [9]. Downloaded artifacts are cached locally to make reruns deterministic and to avoid repeated network transfers.

4.2. Dataset Construction and Cleaning

The loader parses the two tab-separated tables, removes metadata columns, and transposes the matrices so that each row is a sample and each column is a gene.

Tumor and normal files do not always contain exactly the same gene list. Therefore, the final feature set keeps only genes that appear in both sources. Denoting by G_{tumor} and G_{normal} the gene sets in the two tables, the final feature set is the intersection $G^* = G_{\text{tumor}} \cap G_{\text{normal}}$. Genes with missing names are discarded and duplicate gene columns are removed.

The final design matrix is $X \in \mathbb{R}^{n \times p}, \quad p = |G^*|$ and labels are defined as

$$y_i = 0(\text{normal}), \quad y_i = 1(\text{tumor}),$$

resulting in $\boldsymbol{y} \in \{0, 1\}^n$.

4.3. Splitting Protocols (Hold-out and Stratified k-fold)

The dataset container exposes two splitting schemes used across experiments:

- a reproducible hold-out split with configurable test fraction (default: 80/20) and seed-controlled shuffling;
- stratified k-fold cross-validation (default: $k = 5$), where stratification approximately preserves the class proportions in each fold.

These protocols are essential in the high-dimensional regime (many more features than samples): performance estimates can vary substantially with the particular split, and stratification reduces the risk of misleading results when classes are imbalanced.

5. Experimental Results

This section summarizes the empirical behavior of GLNs on TCGA-BRCA tumor vs. normal classification, with two complementary goals: (i) assess predictive performance in a high-dimensional setting and (ii) characterize how sensitive the method is to optimization choices, which is a prerequisite for any subsequent interpretability analysis.

In particular, the two training regimes introduced in Section 3.2 are contrasted: paper-faithful local online OGD versus end-to-end batch optimization (Adam/backpropagation). For both regimes, representative learning dynamics are reported and performance is compared against standard baselines.

5.1. Hyperparameter Configurations

The experiments in this section are run under a small set of controlled hyperparameter choices, with the goal of making training-regime comparisons interpretable rather than exhaustively tuned. The common configuration used for the cross-validation comparison and for the representative training-curve runs is summarized in Table 2.

Two choices are deliberate. First, several runs use a single epoch (one pass through the data) to directly probe the paper's online-learning claim: meaningful performance should emerge quickly, without relying on repeated passes through a limited cohort. Second, batch size is set to 1 in the most direct comparisons because it maximizes the number of parameter updates per observed sample and makes the distinction between local OGD and backpropagation most visible in the loss/accuracy trajectories. Larger batches (e.g., 10) are additionally included to show how conventional minibatching smooths optimization in the batch regime, even though it departs from the strictly online setting.

5.2. Training Dynamics: Local OGD vs. Backpropagation

On TCGA-BRCA, the separability of the task leads most regimes to achieve high test accuracy, but the training dynamics differ substantially across update rules and learning-rate schedules.

In the local online OGD setting (Figure 3, subplots A and B), training loss is highly intermittent, with frequent sharp spikes throughout training. This behavior is consistent with the online nature of the update and with the discretization induced by contextual gating: each minibatch (here, a single sample) triggers a potentially large step on the weights of the currently active experts, and small changes in the active-path mixture can translate into abrupt changes in loss.

Learning-rate control is particularly consequential under local updates. The decay schedule (subplot A) yields higher and more stable final accuracy than the constant step-size variant (subplot B), suggesting that step-size decay is important for mitigating the variance introduced by sample-wise updates and context-dependent expert selection.

In contrast, batch backpropagation with batch size 1 (subplots C and D) exhibits a higher loss at the beginning of training but converges more smoothly. The same qualitative pattern holds for the learning-rate schedule: decay (subplot C) outperforms a constant step size (subplot D), indicating that even when optimization is end-to-end, the high-dimensional setting benefits from reducing the effective step size over time.

Finally, batch backpropagation with batch size 10 (subplots E and F) produces the smoothest curves and the fastest convergence in terms of minibatches, at the cost of giving up the strictly online update principle. This contrast motivates the two-regime implementation: local online OGD is retained to test the theoretical learning rule under realistic data, while the batch regime enables controlled ablations and comparisons against standard optimization practice.

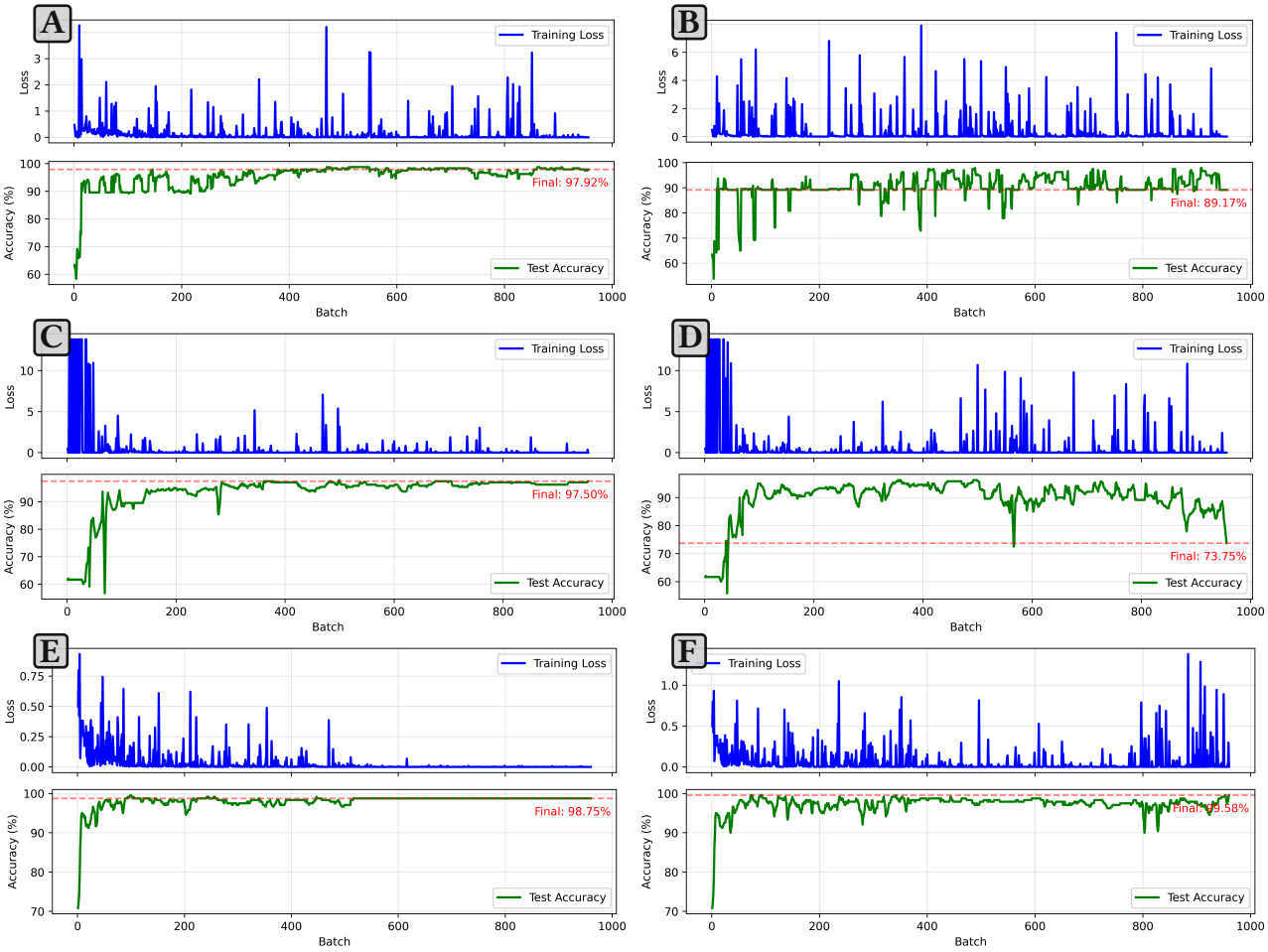

Figure 3: Training curves on TCGA-BRCA. Each subplot shows loss (top) and test accuracy (bottom) as a function of minibatches. The two columns correspond to learning-rate schedules (decay A C E vs. constant B D F), while the three rows correspond to training regimes: (i) local online OGD, single pass (A, B); (ii) batch backpropagation with batch size 1 (online backprop), single pass (C, D); (iii) batch backpropagation with batch size 10 (standard minibatching), trained for 10 epochs (E, F).

5.3. Baseline Comparisons

To contextualize GLN performance, the experimental pipeline includes two baselines trained on the same splits used for the GLN runs:

i. Logistic regression as a strong convex model in high-dimensional settings
ii. A MLP as a standard nonlinear reference.

The logistic regression baseline is implemented with `sklearn` and uses feature-wise standardization (via `StandardScaler`) followed by `LogisticRegression` with the `lbfgs` solver and an L_2 penalty (default inverse regularization strength $C = 1$). The MLP baseline matches the GLN hidden-layer sizes for a controlled comparison, uses ReLU nonlinearities and a sigmoid output, and is trained end-to-end with Adam and a linear learning-rate schedule. Inputs are standardized using training-set statistics (as in the GLN pipeline's first standardization step).

Results for these baselines are reported in the cross-validation comparisons in Figure 4.

5.4. Cross-Validation Results

Single hold-out splits can overestimate performance when p is large and n is moderate. For this reason, models are also evaluated with stratified k-fold cross-validation (default: $k = 5$) repeated for n times (default: $n = 3$) to assess variability across splits and seeds.

Figure 4 summarizes these results via accuracy boxplots. Two trends are noteworthy:

i. Baseline methods exhibit relatively tight distributions, reflecting the stability of convex (logistic regression) or well-regularized (MLP) training.

ii. GLN variants display larger accuracy dispersion across folds, especially with local OGD, reflecting sensitivity to split-induced shifts and hyperparameter choices. This instability reduces confidence in attribution results: feature attributions are meaningful only when the learned predictors are stable across reasonable resamplings.

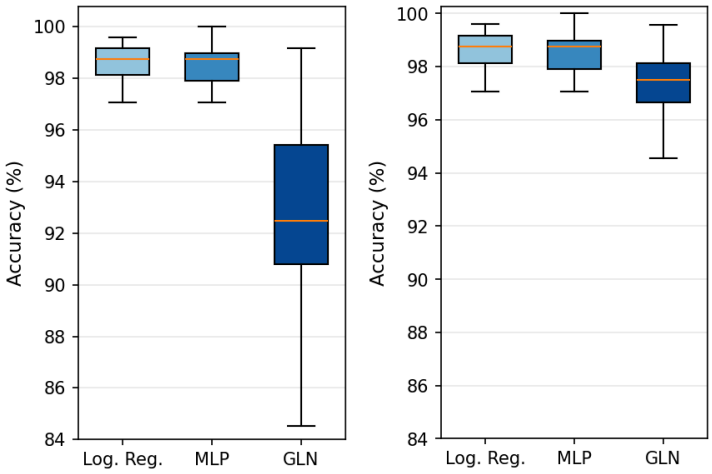

Figure 4: Cross-validation comparison across model families. Boxplots report accuracy distributions across repeated stratified k-fold splits. Left: GLNs trained with local online OGD. Right: GLNs trained with end-to-end backpropagation. In both panels, GLN variants are compared against a logistic regression baseline and a feed-forward MLP baseline (trained with standard backpropagation).

Param.[3]	Value
E	1
M	1
L_i	20, 40, 20
K	4
η_t	0.01
$t \mapsto \eta_t$	const
s	4

Table 2: Common hyperparameters used for the TCGA-BRCA experiments reported in this section. Parameters not listed here are left at their codebase defaults; the exact CLI options and default values are documented in the repository README.

6. Interpretability Analysis

This section examines whether the GLN predictor learned on TCGA-BRCA supports meaningful gene-level explanations. The motivating use case is biomarker discovery: beyond high classification accuracy, an attribution method should highlight genes that align with known tumor biology and remain stable under reasonable analysis choices.

6.1. The Promise of Interpretability

GLNs are often described as naturally interpretable because each neuron computes a convex geometric mixture and (in the local regime) updates only the parameters on its active path. For a fixed input, the model selects a discrete set of experts through contextual gating and combines probability features through logit-space mixing. In principle, one could identify the active experts and attribute the final logit to the input genes through gradient-based credit assignment.

[3]Parameters description can be found in the Appendix Section I

In a genomics setting, this would ideally produce a ranked list of genes whose expression shifts the model towards tumor or normal. The remainder of this section evaluates whether this expectation holds in practice.

6.2. Integrated Gradients Method

Integrated Gradients (IG) [4] is used as the primary attribution method because it addresses common failure modes of raw gradients (notably saturation) by integrating sensitivity along a path from a baseline input $\boldsymbol{x}_0$ to the actual input $\boldsymbol{x}$.

In this work, the baseline is the dataset mean. IG is approximated with 50 steps along the straight-line path from $\boldsymbol{x}_0$ to $\boldsymbol{x}$, and global importance is computed by aggregating the mean absolute attribution across samples.

6.3. Expected vs. Observed Results

The expectation is that a model achieving high predictive performance should, at least to some extent, assign high importance to genes that are widely reported as breast-cancer relevant (e.g., ESR1, PGR, GATA3, TP53, BRCA1/2, ERBB2).

Figure 5 shows the genes with highest average IG magnitude for an OGD-trained model. Many top-ranked genes are not canonical breast-cancer markers and include genes with unclear or indirect relevance to the task. More importantly, the right panel shows that a curated list of known cancer genes is not highly ranked by IG: only a small subset appears in the top 1,000 features, while many fall in the lower half of the full gene set.

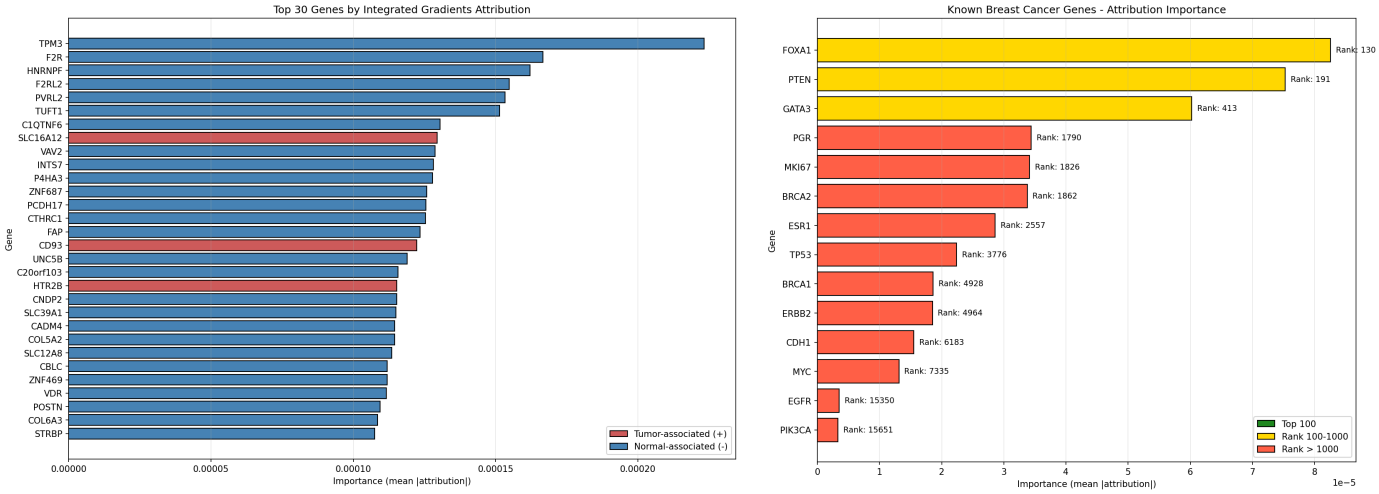

Figure 5: Integrated Gradients feature attribution on TCGA-BRCA (OGD-trained GLN). Left: top-ranked genes by mean absolute IG attribution across samples. Right: ranks of a curated set of known breast cancer genes under the same scoring. High predictive accuracy does not translate into a biologically aligned gene ranking.

6.4. Architectural Modifications Tested

To assess whether the mismatch between accuracy and gene rankings is primarily an artifact of a single attribution choice, two controlled variations are considered.

First, IG is compared against vanilla saliency (raw input gradients). Saliency measures local sensitivity at the input point and is known to be unstable under saturation and small perturbations; IG provides a more global attribution by construction.

Across these variations, the qualitative conclusion is consistent: the model can separate tumor from normal reliably, but gene-level attribution is not robust enough to be treated as biomarker discovery.

6.5. Root Cause Analysis

The observed behavior is best explained as an architectural limitation.

6.5.1. Primary Cause: Random Fixed Hyperplanes

The most consequential limitation is architectural. Contextual gating is implemented through random half-space tests with non-learnable parameters `ctx_V` and `ctx_b`. They are sampled once and remain fixed throughout training.

As a result, each neuron's decision is conditioned on the sign pattern of random projections of the input, and learning occurs only in the expert weight tables indexed by those sign patterns. In this setting, the model does not learn that a specific gene is directly important; it learns that certain random linear combinations of genes define useful partitions of the input space. Importance is therefore distributed across many arbitrary mixtures, which makes single-gene attribution intrinsically fragile.

6.6. Alternative Attribution: Saliency

Vanilla saliency provides an additional lens on the model but does not resolve the core issue. Figure Figure 6 shows that saliency-based rankings differ substantially from IG and can emphasize different gene sets and directions. This sensitivity is consistent with the theoretical limitation discussed above: when predictions are mediated by context-dependent expert selection and by saturating nonlinear preprocessing, local gradients can change sharply across nearby inputs.

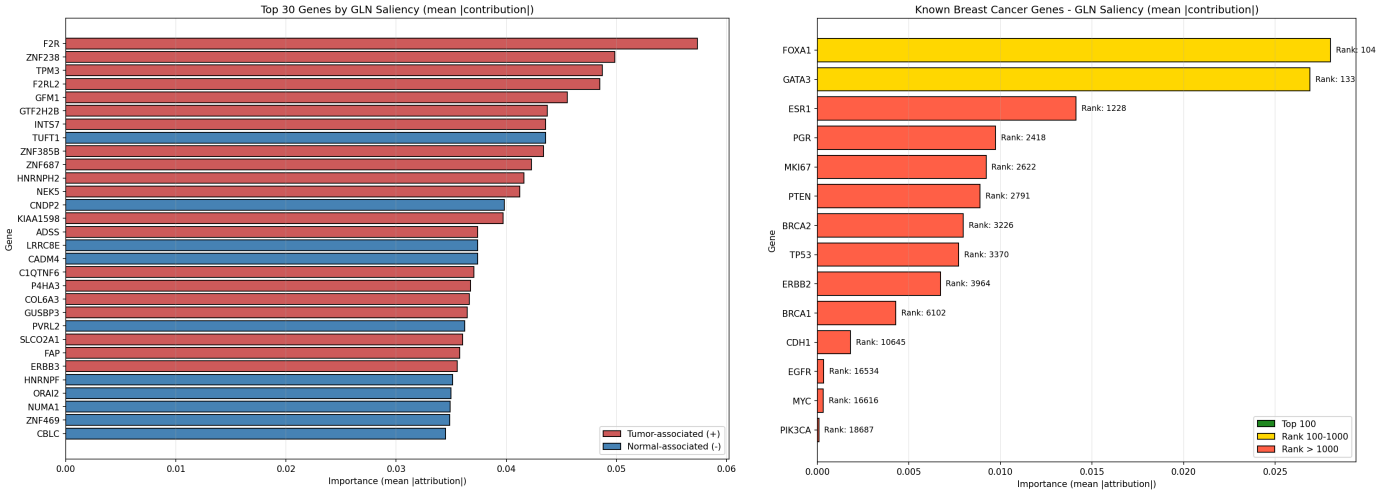

Figure 6: Vanilla saliency attribution on TCGA-BRCA (OGD-trained GLN). Left: top-ranked genes by mean absolute gradient contribution. Right: ranks of known breast cancer genes under saliency scoring. Compared to IG, saliency yields a more local and often less stable notion of importance, reinforcing that gene-level rankings are not robust under this architecture.

7. Conclusion

The empirical study in Part II shows that GLNs can be trained successfully on TCGA-BRCA under the paper-faithful local online OGD regime and can achieve high classification accuracy in a high-dimensional setting. When compared against strong baselines, performance is competitive but does not inherently surpass convex linear models on this specific task, which is consistent with the separability of tumor vs. normal expression profiles.

The interpretability analysis highlights a more fundamental limitation. Although GLNs expose a notion of active-path computation and use strictly local updates, this structure does not automatically yield reliable gene-level explanations on genomic data. Under both Integrated Gradients and saliency, top-ranked genes do not consistently align with widely reported breast cancer markers, and known

genes may appear far down the ranking. This mismatch is best understood as architectural: contextual gating relies on fixed random half-space tests, so predictive signal is routed through a discrete partition defined by arbitrary linear combinations of genes. In consequence, attribution methods that attempt to assign credit to individual coordinates can be unstable or misleading even when predictive performance is strong.

7.1. Summary of Contributions

i. A modular PyTorch implementation of GLNs with contextual gating and geometric mixing, supporting both paper-faithful local online OGD and batch optimization for ablations.
ii. A reproducibility benchmark on MNIST, including configuration reporting and qualitative saliency visualizations.
iii. An empirical evaluation on TCGA-BRCA with controlled splits and baseline comparisons against logistic regression and an MLP.
iv. An interpretability study on OGD-trained GLNs using Integrated Gradients and saliency, documenting the observed accuracy–interpretability tension and attributing it to fixed random gating.

Appendix

I. Glossary

$\boldsymbol{E}$ number of passes through the training data
$\boldsymbol{M}$ number of samples per minibatch
$\boldsymbol{L_1}, \boldsymbol{L_2}, \boldsymbol{L_i}$ ... layer sizes (number of neurons per layer)
$\boldsymbol{K}$ context dimension (number of half-space tests per neuron)
$\boldsymbol{\eta_t}$ learning rate (depends on learning schedule)
$\boldsymbol{t \mapsto \eta_t}$ learning rate schedule (e.g., "decay" for $\eta_t = \frac{\eta_0}{\sqrt{t}}$, "constant" for $\eta_t = \eta_0$)
$\boldsymbol{s}$ random seed for reproducibility (affects weight initialization and half-space sampling)

Bibliography

[1] J. Veness *et al.*, "Gated Linear Networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[2] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, 2019.

[3] The Cancer Genome Atlas Network, "Comprehensive molecular portraits of human breast tumours," *Nature*, vol. 490, no. 7418, pp. 61–70, Oct. 2012.

[4] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic Attribution for Deep Networks," in *International Conference on Machine Learning (ICML)*, 2017.

[5] *DeepMind's GitHub Repository*. [Online]. Available: https://github.com/google-deepmind/deepmind-research/tree/master/gated_linear_networks

[6] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, 1998.

[7] E. Cerami *et al.*, "The cBio Cancer Genomics Portal: An Open Platform for Exploring Multidimensional Cancer Genomics Data," *Cancer Discovery*, vol. 2, no. 5, 2012.

[8] *cBioPortal DataHub*. [Online]. Available: https://github.com/cBioPortal/datahub

[9] B. Li and C. N. Dewey, "RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome," *BMC Bioinformatics*, vol. 12, no. 1, 2011.