UNIVERSITY OF CALIFORNIA
SANTA CRUZ

SIMS: SCALABLE, INTERPRETABLE MACHINE LEARNING FOR SINGLE-CELL ANNOTATION

A dissertation submitted in partial satisfaction of the
requirements for the degree of

MASTERS OF SCIENCE

in

APPLIED MATHEMATICS AND SCIENTIFIC COMPUTING

by

Julian Lehrer

June 2022

The Dissertation of Julian Lehrer
is approved:

Professor Vanessa Jonsson, Chair

Professor Mircea Teodorescu

Mohammed Mostajo-Radji

Professor Daniele Venturi

Dean Peter Biehl
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures **iv**

List of Tables **vii**

Abstract **viii**

Acknowledgments **x**

0.1 Introduction . . . 1
0.1.1 Single-cell RNA-sequencing . . . 2
0.1.2 Determining Cell Types . . . 5
0.2 Learned Classifiers with Machine Learning . . . 8
0.2.1 Metric Calculation and Understanding Class Distribution . . . 11
0.2.2 Hyperparameters and the Regularization . . . 13
0.2.3 Deep Learning . . . 16
0.2.4 Self-Supervised Learning . . . 18
0.2.5 A Deep Learning Architecture for Transcriptomics . . . 19
0.2.6 Interpretability . . . 22
0.3 SIMS: Scalable, Interpretable Machine Learning for Single-Cell . . . 23
0.3.1 Concept and API . . . 23
0.3.2 Results . . . 29
0.3.3 Interpretability Analysis . . . 40
0.3.4 Generalization Capability . . . 44
0.3.5 Ablative Studies . . . 46
0.3.6 Benchmarking . . . 51
0.3.7 Inference on Tissue Cultures . . . 54
0.3.8 Discussion . . . 56
0.4 Conclusion and Future Work . . . 59

Bibliography **61**

List of Figures

0.1 General workflow of a single-cell RNA-seq experiment, from [HETL17]. . 4
0.2 Example of a feedforward neural network, with 10 input features. In the classification case, this induces a distribution for a binary classification problem with two hidden outputs. 17
0.3 TabNet architecture. Based on a transformer, TabNet uses sparse feature masks for interpretability by sample, as well as globally by aggregating weights across multiple samples. 21
0.4 The *SIMS* pipeline. First, (*a*) normalized input data from expert annotations is input in the DataModule class. Next, the neural network (*b*) is defined with the chosen optimizer and training parameters. Live training statistics (*c*) can be viewed to understand training, validation and test performance. Finally, we can use the feature masks to make interpretable predictions (*d*) on unseen data. 25
0.5 Example of Metric Tracking via the SIMS package on wandb.ai for the Human Cortical Dataset. All metrics shown here are tracked by default by SIMS. 29
0.6 UMAP and label visualizations of the dental data [KSK$^+$20] and retinal data [LLS$^+$19]. (a) UMAP projection of dental data colored by cell type. (b) Distribution of labels for dental data. (c) UMAP projection of retina data colored by cell type. (d) Distribution of labels for the retina dataset. 31
0.7 UMAP and label visualizations of the Allen Brain Institute human cortical data. (a) UMAP projection colored by cell subtype. (b) Distribution of subtype labels. (c) UMAP projection colored by cell type. 33
0.8 UMAP and label visualizations of the Allen Brain Institute mouse cortical data. (a) UMAP projection of the colored by cell subtype. (b) Distribution of subtype labels. (c) UMAP projection colored by main cell type. 34
0.9 UMAP and label distributions of human human cortical data from [BAML$^+$20]. (a) the UMAP projection of the human cortical data colored by cell subtype. (c) the distribution of these subtype labels. (c) UMAP projection colored by cell supertype. 35

0.10 Principal component visualizations of the three benchmark datasets from cortical tissue. The left column visualizes the data listed projected onto the first two principal components, where each point is colored by its major phenotype group. The right column ranks the first ten principal components for the dataset, ordered by the proportion of the linear variation explained by each. 37
0.11 Feature weights aggregated over all test samples for the dental and retina models. (a) Distribution of global feature weights for the dental model trained on [KSK+20] data. (b) Matrix of normalized weights (input genes) across all samples on the test set. (c) Distribution of global feature weights for the retina model trained on [LLS+19]. (d) Matrix of normalized feature weights for all samples on the test set. 41
0.12 Feature weights for models trained on the Allen Brain Institute human and mouse cortical datasets. (a) Distribution of global feature weights for the human cortical model, trained on all brain regions. (b) Matrix of normalized weights (input genes) across all samples on the human cortical test set. (c) Distribution of global feature weights for the mouse cortical model. (d) Matrix of normalized feature weights for all samples on the mouse cortical test set. 42
0.13 Top genes aggregated over cell subtype for the Allen Brain Institute mouse cortical data with $C = 42$ cell types. Each row represents a class label, and the columns are normalized feature mask values. 43
0.14 Top genes aggregated over cell subtype for the Allen Brain Institute human cortical data with $C = 19$ cell types. Each row represents a class label, and the columns are normalized feature mask values. 44
0.15 Metric results for the SIMS pipeline trained on the Allen Brain Institute human MTG data and benchmarked on all available human brain tissue data. (a) Balanced and weighted accuracy. (b) Aggregated F1 and median F1 scores. 45
0.16 Metric results for the SIMS pipeline trained on the Allen Brain Institute mouse V1C (Visual Cortex Region 1) data and tested on all other mouse brain tissue data. (a) Balanced and weighted accuracy. (b) Aggregated F1 and median F1 scores. 46
0.17 Metrics for model trained on the Allen Brain Institute Human MTG data. Total number of cells in initial training set was $N = 47432$ and $M = 19$ classes. Each proportion p corresponds to a train/val/test split of pN cells. Data was stratified in the train/val/test split, and each split was determined with a deterministic seed for all runs. (a) Average of the median F1 score, across the final ten model epochs on the validation set for each ablative model. Each proportion p corresponds to a train/-val/test split of pN cells. (b) Validation loss as a function of the number of epochs trained. (c) Median F1 score as a function of the number of epochs trained. (d) Weighted accuracy as a function of epochs trained. 47

0.18 Metrics for the SIMS model trained on the Allen Brain Institute Human MTG data with smaller training proportions. The train/val/test splits were stratified when the dataset was large enough to do so, and each split was determined with a deterministic seed for all runs. (a) Average of the median F1 score, across the final ten model epochs on the validation set for each ablative model. Each proportion p corresponds to a train/-val/test split of pN cells. (b) Validation loss as a function of the number of epochs trained. (c) Median F1 score as a function of the number of epochs trained. (d) Weighted accuracy as a function of epochs trained. 49
0.19 Metrics for model trained on the Allen Brain Institute Mouse cortex data. Total number of cells in initial training set was $N = 73347$ and $M = 42$ classes. (a) Average of the median F1 score, across the final ten model epochs on the validation set for each ablative model. Each proportion p corresponds to a train/val/test split of pN cells. (b) Validation loss as a function of the number of epochs trained. (c) Median F1 score as a function of the number of epochs trained. (d) Weighted accuracy as a function of epochs trained. 50
0.20 Inferred cell types using the SIMS pipeline trained on inhibitory cortical neurons from the M.R. UCSF experiment. 55
0.21 Inferred cell types of organoid data from [BAML$^+$20] using the SIMS pipeline trained on primary data from the same reference. 56

List of Tables

0.1 Metric results for the SIMS pipeline on the test set. Precision, recall, and specificity were calculated using micro averaging. 39
0.2 Benchmarking results for SIMS vs scANVI 53

Abstract

SIMS: Scalable, Interpretable Machine Learning for Single-Cell Annotation

by

Julian Lehrer

Experiments in molecular and cellular biology today have become increasingly large and complex, with technological advances enabling high resolution, multi-modal omics measurements at the level of individual cells. The capacity to readily collect these datasets has contributed to unprecedented biological insight – and concurrently, a data deluge. Tasks such as cell annotation and cell state characterization increasingly necessitate automation, and while data driven methods aimed at inferring cell state from omics and image data are currently in development, a focus on robustness, scalability and interpretability are paramount. We present SIMS: an end-to-end modeling pipeline for discrete morphological prediction of single-cell data with minimal boilerplate and high accuracy. We perform several studies using SIMS, and show the underlying model performs well in a variety of data-adverse conditions. We show that SIMS performs well between tissue samples and outperforms one of the most popular cell type classification algorithms on several benchmark datasets. We also implement how classification outputs can be directly characterized as a combination of sparse feature masks, allowing for interpretability at the level of individual samples. Finally, we show some use case of SIMS for inference, and argue it will become a useful tool in the field of single-cell data analysis. All code is open source and available at github.com/jlehrer1/SIMS.

Acknowledgments

This thesis and my all my research at the University of California, Santa Cruz would not have been possible if it weren't for my wonderful lab members, mentors, professors, friends and family. Thank you to Yohei Rosen for first introducing me to research at the Braingeneers group. His fascination by cutting edge statistical and biological sciences inspired me to question everything and trust nothing. I would not be the inspired researcher I am today without his guidance throughout my undergraduate research career.

Thank you to Mircea Teodorescu for being the most supportive and kind principal investigator I could ask for. His passion for the pursuit of science is endlessly inspiring, and without his encouragement I would not be where I am today.

Mohammed Mostajo-Radji, for being dedicated to the cutting edge of neuroscience, inspiring this project and pushing me to go further than I thought possible. This thesis would not have been possible without his consistent support and guidance.

My advisor Vanessa Jonsson for teaching me about single-cell analysis, transcriptomics and machine learning; being a mentor, introducing me to new friends and colleagues who have provided endless valuable guidance. Lest I forget, helping me fix a massive bug in my code that stopped me in my tracks for weeks.

David Parks, whose knowledge of distributed computing allowed me to run my experiments seamlessly. His knowledge of machine learning & Python programming has been unboundedly helpful. Lucas Seninge for his deep knowledge of single-cell analysis.

To my friends family for supporting me throughout this process.

Finally, to all my lab mates for fostering a caring, supportive and rigorous environment. My time here has been wonderful, and I can't wait to continue.

0.1 Introduction

High-throughput NGS (Next-Generation Sequencing) systems have allowed for large scale collection of transcriptomic data at the resolution of individual cells. Within this data lies variability allowing us to characterize morphological characteristics such at cell type, cell state, infer trajectories of cell growth and estimate gene regulatory networks between cells [HETL17] [SCTS19]. These qualities are an important part of understanding how cells interact with one another, both for building better cellular models in vitro and understanding biological processes in vivo. While the size of single-cell data has increased massively [AST+17], the techniques for analysis of these characteristics has stayed fairly constant, following manual pipelines which often require domain experts for critical components [LT19]. Naturally, the development of software to automate manual analysis has become an important and popular research topic [LT19]. However, the performance of these automated classifiers often degrades as the number of cell types and cell subtypes increase, and the number of samples per label becomes small. The distribution of cell types is often asymmetric, with a majority class dominating a high percentage of cells. Additionally, technical variability between experiments can make robust classification between multiple tissue samples difficult. Finally, the high-dimensional nature of transcriptomic data makes analysis statistically and computationally intractable [KS05]. These conditions make applying classical models such as support vector machines difficult [AKJ04]. In response, generative neural networks have become a popular framework for their robustness to technical variability within

data, scalability, and ability to capture biological variability in the latent representation of the inputs. In this paper we present a new framework based on [AP21] and [Fal20], capable of robust performance with deep annotation and skewed label distributions, high accuracy with small and large datasets, and direct interpretability from the input features. We begin with a review of single-cell RNA-sequencing.

0.1.1 Single-cell RNA-sequencing

Standard methods such as microarrays and bulk RNA-seq analysis analyze the expression level of RNAs from large, and possibly mixed, groups of cells. Although this provides a lower-cost way to measure the averaged expression profile of a cell type, the critical differences in cells may be obscured, distorting downstream analyses and conclusions.

Current single-cell RNA-seq protocols, which we shall now refer to as RNA-seq, involve isolating single cells and their RNA. The first and most important step in single-cell RNA-seq is the isolation of individual cells. Next, poly[T]-primed RNA is converted to complementary cDNA by a process called reverse transcriptase. Often, the reverse transcription will additionally implement methods for barcoding the individual RNA molecules and nuclei to preserve cellular origin. These small amounts of cDNA are then *amplified*. The amplified and tagged cDNA from each of the cells is pooled and sequenced by an NGS system. Then, reverse transcription, amplification, library generation and sequencing allow the measurement of individual transcriptomes at a certain point in time for each cell [HETL17]. This recapitulates an *expression matrix*,

that is, an $n \times p$ matrix of n cells with p genes measured, where the ith row of the expression matrix represents the transcriptome for the ith measured cell at a snapshot in time.

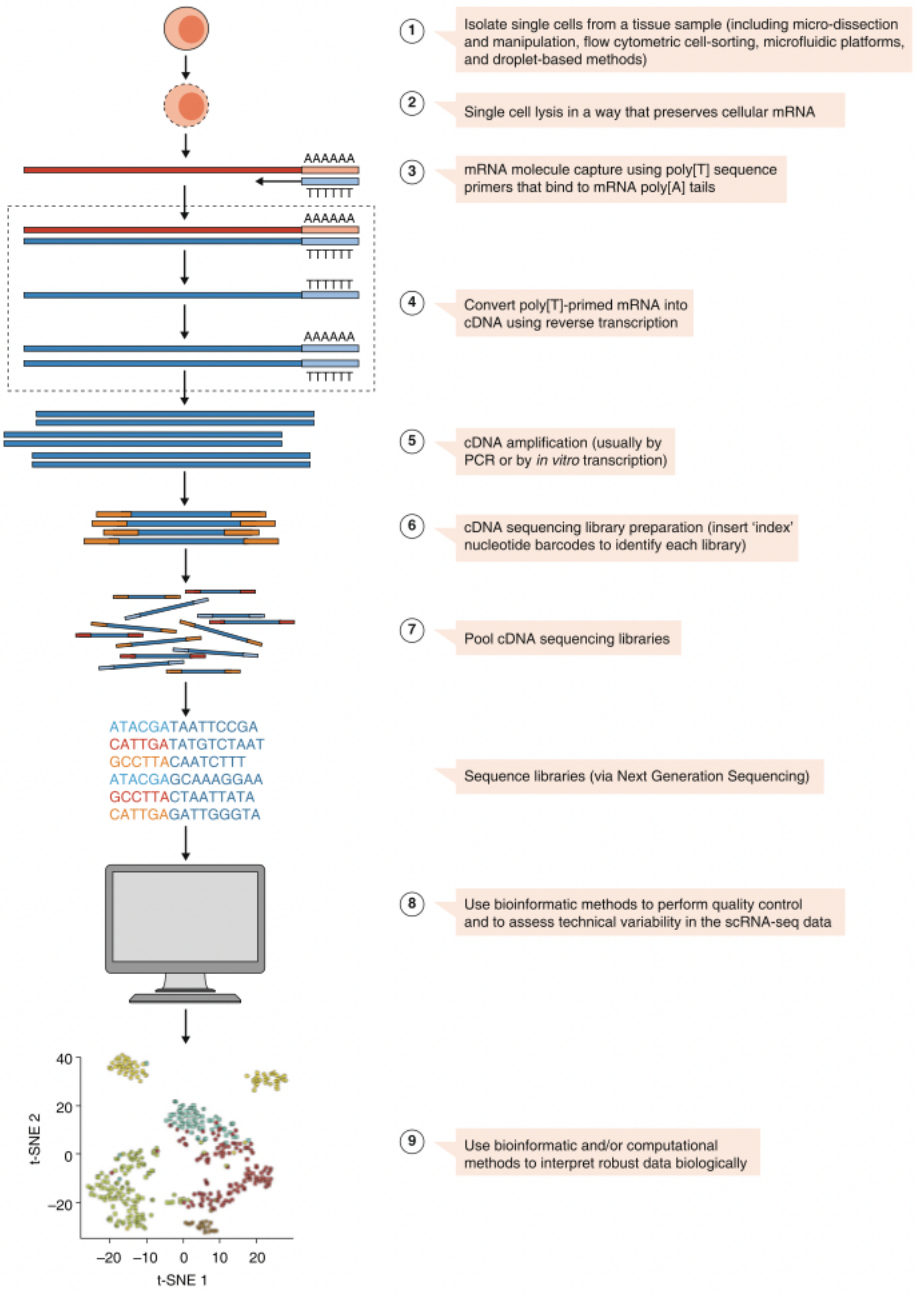


Figure 0.1: General workflow of a single-cell RNA-seq experiment, from [HETL17].

The expression matrix is the core component of our downstream analysis, where we focus on prediction of discrete characteristics, rather than continuous variation

in time or state. We choose cell type as the canonical label set of interest, although the methods presented generalize for any label set.

0.1.2 Determining Cell Types

The process for estimating cell populations is performed with a pipeline beginning with manual data prepreprocesing and often results in requiring a domain expert to help identify cell-specific marker genes [LT19]. Each entry in the expression matrix is first $\log(x+1)$ normalized. This normalization serves two purposes: firstly, distances between log-transformed expression values represent log-fold changes, the standard measure of comparing expression values [HETL17] [BHGMP22]. Secondly, the log-transformation mitigates the mean-variance relationship in the data. Often, although not always, we first discard the first m nonvariable genes where variability is calculated from a variety of statistical tests. This leaves an expression matrix of $n \times p - m$ cells. To make clustering computationally tractable and less affected by the curse of dimensionality, the data is projected to a lower dimensional subspace via principal components' analysis. Additionally, an autoencoder is often used to mitigate technical variation in the data and expose true biological variability.

Next, these first l principal components are clustered on via k-means or Jaccard-Louvain graph-based clustering. We then calculate the top *differentially expressed genes*, that is, the genes that vary the most within each cluster. The top differentially expressed genes within a cluster are called marker genes, and are used to determine cluster cell type. A reference atlas is a collection of one or more single-cell experiments done on

homogeneous cells in order to understand which differentially expressed genes are important for transcriptomically identifying the cell type. We then take the marker genes identified from the clusters in our dataset and compare them against the marker genes in the reference atlas. The atlas labels are then transferred over to our clusters; hence we have identified the cell types within our data. To visualize these assigned clusters, an algorithm called UMAP (uniform manifold approximation and projection) is run on the principal components' analysis projection of our input data. UMAP projects the PCA space into two dimensions, by creating a high-dimensional fuzzy graph that probabilistically connects points, and then attempts to recreate this graph in lower dimensions by minimizing an error function between the two. Although UMAP creates nice qualitative plots, understanding its limitations for inference is critical. The UMAP projection should not be used to validate any clustering via visual separation, but rather serve as a tool for visualization purposes only. This is because in the reconstruction of the low-dimensional embedding, clusters can seem to appear to the human eye that quantitatively do not exist in the ambient space [CBP21].

There are a few implicit assumptions in this standard clustering pipeline. Firstly, in the data preprocessing. By projecting our data to a lower dimensional subspace via principal components' analysis, we are assuming that the variation between genes is linear, since principal components' analysis finds a new basis in data space whose basis elements are orthogonal and pass through the most linearly variable directions. In the case of highly nonlinear variation, it may not be that PCA best captures this variation.

Additionally, each clustering algorithm has its own internal assumptions about the underlying structure of the data, which will be explored more later. Finally, and most importantly, we assume that the clusters capitulated by our chosen clustering method indeed represent cell type, a scientifically defined category, and not some undetermined group that has not been identified. Luckily, there has been much research into single-cell clustering, and although there is room for more rigor and testing of methods, it has indeed been shown that clusters found by graph-based methods indeed define cell type. [LT19]

More generally, this methodology requires a lot of preprocessing, which in turn requires a lot of computing power. Given a new dataset from a previously explored tissue, performing this entire procedure would lead to redundancy, and in the case of large enough datasets may not be computationally feasible. However, there is a more prominent reason for not using this pipeline, which is the case of transcriptomic change from in vitro experiments. As 3D tissue models called organoids become increasingly popular for modeling tissue growth conditions in the cerebral cortex and beyond, understanding cell growth within these models is important for validating their research use. The technical differences in growing cells in vitro, as well as sequencing technology and library preparation is called the *batch effect*. Batch effect can often change the transcriptomic profile of the expression matrix in subtle ways that affect the initial clustering analysis, and in turn make cell type annotation difficult [BAML$^+$20]. For this reason, we turn to a machine learning approach where we seek to learn the most salient genes for predicting cell type labels.

Specifically, we seek a function $f : c_i = \mathbb{R}^p \longrightarrow L$, where c_i is the ith transcriptome (that is, the gene expression for the ith cell) and L is the set of cell type labels, such that f has high *accuracy* while also selecting a sparse subset of vector elements c_i^k (genes) for prediction power. Since the model is only allowed a small subset of the genes to use for cell type labeling, we hope that only true biological variation is learned and not batch effect, which will not be salient for cell type labeling in general.

0.2 Learned Classifiers with Machine Learning

Consider our samples X with associated labels Y sampled from distribution D given by $\{(X_i, y_i)\}_{i=1}^N$. Our goal is to find a classifier f with small test error, that is, for all samples we want that

$$Error = \arg\min_f \mathbb{E}_{x,y\sim D} \left[f(x) \neq y \right]$$

is minimized. To do this, we select a model f parametrized by θ. Then, we sample N points from D, which we call the train set. Then, we sample two more datasets, known as the validation set and test set, respectively. We find parameters θ such that the train error is minimized, and then quantify our model's ability on unseen data via our test set. The validation set is an intermediate step, in which we validate our model against data as we vary so-called "hyperparameters", which will be discussed later.

In general, models such as logistic regression have parameters found by max-

imum likelihood, where we can find the maximum likelihood parameter by minimizing the negative log-likelihood function. For this paper, we'll consider models whose parameters can be found via gradient methods.

To use a gradient method, we first must construct a differentiable proxy of our train error [Nak21], called the loss function $L(f(x), y)$, as accuracy is given by an indicator function which is not differentiable with respect to the model parameters. This makes the assumption that

Minimizing training loss also minimizes training error

In general, this is not exactly true [Nak21]. It may be that the training error and training loss are not monotonic with respect to one another. In general though, this principle holds and we can find our parameters via the loss function.

Gradient methods find a function f_θ by the following principle:

Fixing the model structure and data, find parameters such that loss function averaged across all samples is minimized

We want that the loss function is minimized over all samples in the training set, that is, we seek to find

$$\arg\min_\theta L(f(X), Y) = \arg\min_\theta \frac{1}{N} \sum_{j=1}^{N} \ell(f(x_j), y_j)$$

By unlucky fate, the term "loss function" refers to both the loss over all training samples and the loss from a single training sample. We won't specify the difference here

unless needed, with the understanding that the general principle of minimization applies either way.

In its simplest form, gradient methods choose θ_0 as a random initialization in parameter space, and perform the following iteration for finding θ

$$\theta_{k+1} = \theta_k - \alpha \nabla_{\theta_k} L(f(X), y)$$

This iterative scheme for finding the parameters such that train loss is minimized is intuitive. Recall that the gradient with respect to a function's arguments gives the direction of steepest ascent, as those parameters are varied by a small amount. Then the negative gradient is the direction of steepest descent, and guides us along the surface to some local or global minima. Letting the loss be a differentiable landscape of our training error, we first select a random point along the loss surface and iteratively step with step size α towards the direction of steepest descent until we converge to a minima.

In practice, we may choose an adaptive step size α, or α may be a matrix that weights different components of the gradient according to some rules. Generally, this is the form of most modern-day optimizers for estimating machine learning models.

There is no guarantee, however, that L is convex. Therefore gradient descent may converge to a local optima whose magnitude relative to other minima is small, and therefore the loss converges to a model with suboptimal training error. With large N, the computation of ∇L is computationally intractable or expensive enough to cause the algorithm to be slow. For these reasons, we often employ stochastic gradient descent

(SGD), which updates θ by sampling a subset of samples from our training set called a *mini-batch*. SGD methods therefore take the form of

$$\theta_{k+1} = \theta_k - \alpha \frac{1}{M} \sum_{i=1}^{M} \nabla_{\theta_k} \ell(f(x_i), y_i)$$

We are estimating the true gradient of our loss function with a small number of samples. Empirically, this has several benefits. Firstly, computationally intractable model training suddenly becomes simple, since we can choose a minibatch size M such that gradient computation is possible on whatever hardware one uses. Additionally, although smaller batch sizes may find different local minima than larger gradient sizes, these minima still generalize well. The choice of batch size and its effects, however, are still an active area of research. [HLT19] [QK20] [SKYL17], and is both dependent on the training data and problem space. For vision models, we often choose a larger batch size, while for the SIMS cell classifier we found that smaller batch sizes between 8 and 32 performed most effectively. One entire pass through the training dataset via minibatches is known as an epoch, and this entire process is repeated until convergence.

0.2.1 Metric Calculation and Understanding Class Distribution

So far we've described the task of minimizing train error as a surrogate for minimizing test error, the ultimate goal of training robust and useful machine learning models. However, this description has neglected a particularly important part of the process: model validation. Often overlooked, quantifying how well our models perform is critical to understanding their benefits and limitations in practice. Here, we will focus

on metrics for multi-class classification.

The most common classification metric is accuracy, where we quantify the total number of correct classifications from the sample set. There are two common ways to calculate accuracy; macro-weighted calculates the accuracy per class and then takes the simple mean, while micro-weighted calculates the total number of predictions globally. Finally, weighted accuracy is calculated by taking the micro-weighted accuracy per class, and then averaging by class support.

Consider the case where the distribution of labels is highly skewed, and the majority class makes up nearly all the data points. In this situation, the model could obtain good accuracy by simply predicting the majority class. Clearly, this doesn't mean the model has learned the input-output mapping. In this case, weighted accuracy is preferred to account for class imbalanced [AAVPS13]. Additionally, we often use precision, recall (sensitivity) and specificity given by the following formulas.

For the binary case,

Precision: true positives / (true positives + false positives)

Recall: true positives / (true positives + false negatives)

Specificity: true negatives / (true negatives + false positives)

Precision measures the total number of correct positive predictions out of the total number of true positives. Equivalently, the class agreement of the data labels with the prediction made by the classifier [GBV20]. Recall measures the total number of

actual positives that were correctly identified, while specificity measures how good the model is at avoiding false positives.

For the multi-class case, where our label set Y contains $M \geq 2$ labels we have that

$$\text{Precision} = \frac{\sum_{i=1}^{M} tp_i}{\sum_{i=1}^{M}(tp_i + fp_i)} \tag{0.1}$$

and

$$\text{Recall} = \frac{\sum_{i=1}^{M} tp_i}{\sum_{i=1}^{M}(tp_i + fn_i)} \tag{0.2}$$

Finally, we may want to use balanced accuracy given by the arithmetic mean of sensitivity and specificity

$$\text{Balanced accuracy} = \frac{\text{sensitivity} + \text{specificity}}{2} \tag{0.3}$$

or the F-score, given by the geometric mean between precision and recall

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{0.4}$$

0.2.2 Hyperparameters and the Regularization

Training a model with the gradient methods described above seeks to minimize the train error. However, in practice this is unimportant. What we really care about is model generalization, that is, the ability to have low error on previously unseen examples. This is because a model with sufficient capacity could simply memorize the entire train set and recapitulate zero train error, therefore not estimating the function mapping between the samples X and the labels Y. Alternatively, since the training

set contains some irreducible noise, the model may be learning this noise and not the underlying relationship between the data and labels. In both cases, the phenomenon of having a large difference between training and testing error is known as *overfitting* and is a common problem when training machine learning models.

There are several ways to reduce overfitting, so our model generalizes well. With neural networks, we can easily reduce model capacity, i.e. reduce the number of parameters to be far fewer than the number of training samples such that the training error has a lower bound on the amount it can be minimized. This stops the model from interpolating, or memorizing, the training set [Nak21]. Reducing model capacity is somewhat underspecified, since the question of where we should remove capacity is dependent on the problem space. Therefore, is common to use *regularization*, which restricts the model's ability to reduce the loss function by penalizing the loss over all samples by a function of the model parameters.

Regularization is added in the following general form

$$\textit{Minimize } L(f_\theta(X), Y) + R(\theta), \textit{ where } R(\theta) \textit{ is function of the model parameters } \theta$$

The most common forms of R are the L_1 and L_2 loss, weighted by a coefficient c. For example, in the case of L2 loss we seek to minimize

$$\arg\min_\theta L(f_\theta(X), Y) + c\|\theta\|_2$$

Consider the simple case of linear regression given by $f_\theta(x) = \theta \cdot x = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$, that is, the estimate of an output y is the linear combination of all input features X. Keep in mind that linear regression is used to predict continuous

variables and does not induce a distribution over our label set like examples previously. Let $\ell(f(x), y) = (f(x) - y)^2$ be the mean squared error used to define our loss L. By minimizing the mean squared error loss plus the norm of the parameters θ, we are forced to select only a subset of weights with large magnitude, hence ignoring erroneous features by setting the weights to be small. If we use $R = c\|\theta\|_1$, some weights are sent to zero during training. Regardless, these two regularization terms have the same effect and help to make the difference between training and testing error as small as possible, hence reducing overfitting.

Since the parameter c must be known before using iterative gradient methods, c is known as a *hyperparameter*. More generally, hyperparameters refer to parameters that cannot be learned via optimization methods and must be set before the training process begins. Tuning these hyperparameters is an active area of research; simple methods such as grid search train a model for a range of hyperparameter values, and select the best one. In the case of having many hyperparameters where the combinatorial set is large we randomly sample sets of hyperparameters until we find ones that fit well, known as random search. More complex methods model the hyperparameters via a distribution, and use Bayesian optimization to iteratively select better ones in function space [FH19]. Regardless of the method chosen, hyperparameters are the reason we split our initial training data two sets: the *validation set* and the *test set*.

Consider iteratively retraining a model, selecting a hyperparameter from a range of values and measuring the performance on the test set. Although this methodology is intuitive, we may actually overfit to our validation set. This is because we're

iteratively (albeit manually or semi-manually) selecting hyperparameters such that error on our validation set is small. We are not truly testing the generalization ability of our model, rather biasing it to perform well on our test set by leaking information about how to perform well on the validation set. This constitutes the third split; the test set. We train our such that training loss is minimized, and then use the validation error as a surrogate for the test error such that we can tune hyperparameters. Only once we select a model that has satisfactory performance on both the training and test set can we test its ability on the test set.

0.2.3 Deep Learning

For this paper, our classifier f is found using *deep learning*. Although there is no formal definition of deep learning, the modern field is defined by a few fundamental principles. First, we choose a model architecture defined by a sequence of linear and nonlinear blocks, where the input data is a sample and the output is a probability distribution over the labels.

For the most straightforward neural network architecture, commonly called a multilayer perceptron (MLP), $f(x) = f_L(\dots f_2(f_1(x;\theta_1);\theta_2);\theta_L)$ where each layer outputs a value a_k such that $a_k = g_k(W_k a_{k-1} + b_k)$. W is a matrix of weights and b is the affine transformation, or *bias* and g_k is the *activation function* applied element-wise. The multilayer perceptron performs an affine transformation of the input x, so $f_1(x; W, b) = g_1(W_1 x + b_1)$. The activation function adds a nonlinearity to each compositional layer, otherwise the entire neural network would be a composition of

affine transformations and hence be linear.

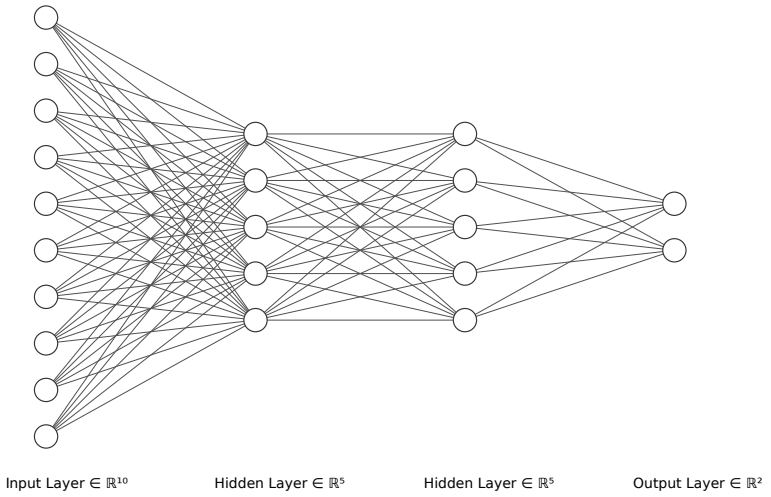


Figure 0.2: Example of a feedforward neural network, with 10 input features. In the classification case, this induces a distribution for a binary classification problem with two hidden outputs.

For vision systems, these nonlinear blocks are commonly convolutions with a fully connected layer for the classification head. For language models, we commonly have transformer blocks with a classification head. In all cases, the structure of our model is chosen based on the given task at home. Convolutional layers, by their construction, are useful for learning information when the input has gridlike topology, which images indeed have. Additionally, we may use other networks such as *recurrent* neural networks when the input is a sequence of time kind, such as an audio file or sentence.

We then construct a neural network $f(x, \theta)$ parametrized by θ. Neural networks output soft decisions, that is, a probability distribution induced over the labels whose final decision is simply the argmax over the label probabilities.

To iteratively train our neural network f, the only thing we require is the

gradient of the loss function L with respect to the model parameters θ. We then update our network by moving the weights with respect to the negative gradient in weight space with a fixed step size, or some variant thereof. We can think of this process as sequentially performing a forward pass and backwards pass. For each minibatch (subset of our training data), we pass the samples forward through the network and obtain a distribution over the label set. The loss function is computed with respect to the known label. Finally, we update our weights such that the loss decreases. We do this sequentially from the tail of the network to the head, in what's called a backwards pass.

0.2.4 Self-Supervised Learning

The task of predicting an output from an input, $f : X \longrightarrow Y$ is known as a *supervised learning task*. To do this well, we require many labeled examples to learn the mapping between each sample and the associated distribution over the labels. However, this data is expensive. In the case of images, it requires manual annotation via human insight. In the case of single-cell analysis, data must clustered via the pipeline described in 0.1, which requires cluster analysis by an expert via a reference atlas. Unlabeled data, however, is cheap and plentiful such as collected unlabeled images or large corpi of text without associated target labels (i.e. question vs answer).

Because of this disparity, machine learning researchers have developed a method for exploiting this unlabeled data to help improve supervised tasks, in a regime known as *self-supervised learning*. In self-supervised learning, we have the model learn a representation of the features in the intermediate layers of the neural network, known as

the latent space. These features are known as latent features. The general principle of self-supervised learning is that if a model can learn some feature representations of our overall input, then performing supervised tasks will be easier. There are two general ways to train a model to learn representations of the input without associated labels: binary masking and denoising.

With binary masking, we consider removing at random particular input features from each input and asking the model to fill them in. For example, we may have a transcriptome with P genes, and we randomly set $K << P$ of these to zero. The loss function is no longer one that measures the difference between the predicted label and the correct label, but rather of reconstruction error. Essentially, we measure if the network was correctly able to fill in the missing values with respect to the original unperturbed data. By doing this, the model will learn a latent representation of how the features relate to each other, making the downstream classification task easier.

0.2.5 A Deep Learning Architecture for Transcriptomics

Single-cell transcriptomics are tabular data, meaning that we can store the data in a spreadsheet-like format, where each row is a cell and the columns denote the total set of genes which we're measuring. Unstructured data typically encompasses images, video or text. This data is unstructured in the sense that it is not stored in a structured database format. In general, neural networks are used for this unstructured data since their representational power allows them to learn input features salient for supervised tasks, without these things being predefined by the user. For example, in the

case of image classification, the only prior needed in the architecture are convolutional layers. Then, the neural network can learn which pixels (i.e. inputs) in an image are relevant for classifying a particular class.

For this same reason, neural networks have typically not been used for structured data. We already have the inputs pre-defined, and unlike images we know a priori that the raw input is relevant for classification, and latent representations won't be as useful. But like all machine learning problems, this assumption is data dependent. In the case of transcriptomics, we know there is relevant high-dimensional geometry, evidenced by the success of dimensionality reduction and clustering for gathering ground truth labels in cell type specification [HETL17].

Therefore, ensemble decision trees, bootstrapped trees and gradient boosted trees have been the go-to for classification of tabular data, in particular when inputs are categorical and not continuous. But this does not mean neural networks are unused. In fact, they have several engineering tricks that make them attractive for supervised problems in general. Firstly, tree methods require data engineering and preprocessing. This means removing outliers, incorrect entries, and choosing a subset of features in the high-dimensional case to make computation tractable.

Neural networks, however, do not require this same data engineering. We can always make computation tractable, by simply choosing a minibatch size for gradient estimation that is computationally feasible on the given hardware. Additionally, the dataset does not need to be loaded into memory, since we only require a minibatch number of samples at each training step. In the case of neural network architectures

that utilize any sort of encoder and decoder, we can pretrain the model such that it is initialized with a latent representation of the data when asked to perform supervised tasks, which can improve performance significantly.

TabNet [AP21], a transformer-based neural network architecture built specifically for tabular data, is at the core of the SIMS pipeline.

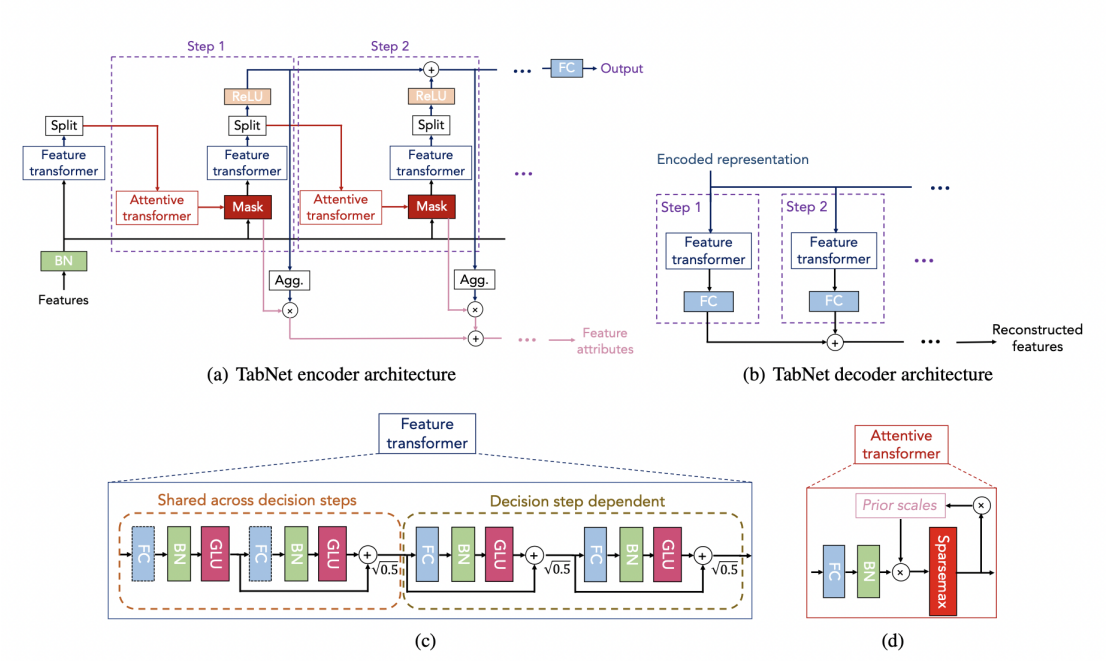


Figure 0.3: TabNet architecture. Based on a transformer, TabNet uses sparse feature masks for interpretability by sample, as well as globally by aggregating weights across multiple samples.

The architecture consists of an encoder composed of a feature transformer, an attentive transformer and a feature mask. The inputs are the raw features with no global normalization internally, although batch normalization is used for training stability and better convergence [STIM18]. The same p dimensional inputs are passed to each decision step of the encoder, which has N_{steps} decision steps. For feature selection at the ith step,

an element-wise multiplicative learnable mask M_i is used. This mask is learned via the attentive transformer, and sparsemax normalization [MA16] is used to induce sparsity in the feature mask. These sequential feature masks are then passed to fully-connected layers for the classification head, first normalized via batch normalization with a gated linear unit [Sha20] for the activation. In our case, we use the raw output of the fully connected classification layer, as [Fal20] loss functions handle logits.

0.2.6 Interpretability

A long-standing and open question in the field of deep learning is seeking a way to interpret the model outputs as an interpretable function of the inputs. In linear regression, for example, we can measure the exact change in the predicted output from a change in an input, since the output is directly a weighted linear combination of inputs.

The deep learning architecture TabNet, the model that underlies SIMS, allows interpretability by measuring weights of the sparse feature masks in the encoding layer. This allows us to understand which input features were used to make each sample prediction. Additionally, the weights can be aggregated over many samples to understand which features are used per class, as well as the average feature weights for the entire dataset.

0.3 SIMS: Scalable, Interpretable Machine Learning for Single-Cell

SIMS, Scalable, Interpretable Modeling for Single-Cell, is the framework we developed for this project. In its conception, we followed three guiding principles

a. Ease-of-use & development time

b. Interpretability

c. Generalizability

In the current state of single-cell transcriptomics, finding labeled examples from data requires much manual preprocessing, removal of technical variation (batch effect), clustering, and labeling with respect to a reference atlas. With the SIMS model, we seek a user interface so that the raw transcriptome from an experiment can be passed to a pretrained model and the cell type labels can be derived immediately with their associated labels. Additionally, this should require *no* manual preprocessing, normalization, correction of batch effect, or selecting variable genes on the user's part. Each lab will be able to train in-house models and save them in a "model zoo" for future use. To this end, we developed a three-point API for model training.

0.3.1 Concept and API

The SIMS pipeline serves as an end-to-end development tool for building robust and sparse single-cell classifiers for discrete cell type prediction. SIMS was developed

with ease-of-use as a guiding principle, freeing up the end user from data preparation in exchange for rapid results for cell type inference.

First, the user defines a data module as an extension of the data module from [Fal20], which as its input takes in a list of data files containing the expression matrices and the associated label files. This data module can handle an arbitrary number of arbitrarily-sized datasets, even when the input genes do not match. We do this by taking the intersection of features across multiple datasets, and only reading samples into memory from delimited files and HDF based files [Kor11] when needed for training. The label files are a delimited text file such that each row contains the associated label. The data module automatically numerically encodes the label set, computes the train, validation and test sets for training. Finally, we also calculate the proportion of each label, so we can weight samples inversely to this number in the loss function to account for class imbalance.

Next, we define the model. The model is a derivation of the model from [AP21] as described in the self-attention section. The user can specify which metrics to track, which variant of stochastic gradient descent to use (or any other gradient-based optimizer), as well as methods for adaptive calculation of step size.

Finally, we define the trainer as an extension of [Fal20], which encapsulates all actual model training. The trainer connects to wandb.ai for live metric tracking, loss of both the training and validation step at each model epoch. Additionally, the trainer automatically handles distributed computation across multiple GPU's, moving gradients between CPU and GPU and handling device-specific movement of data [IPK21].

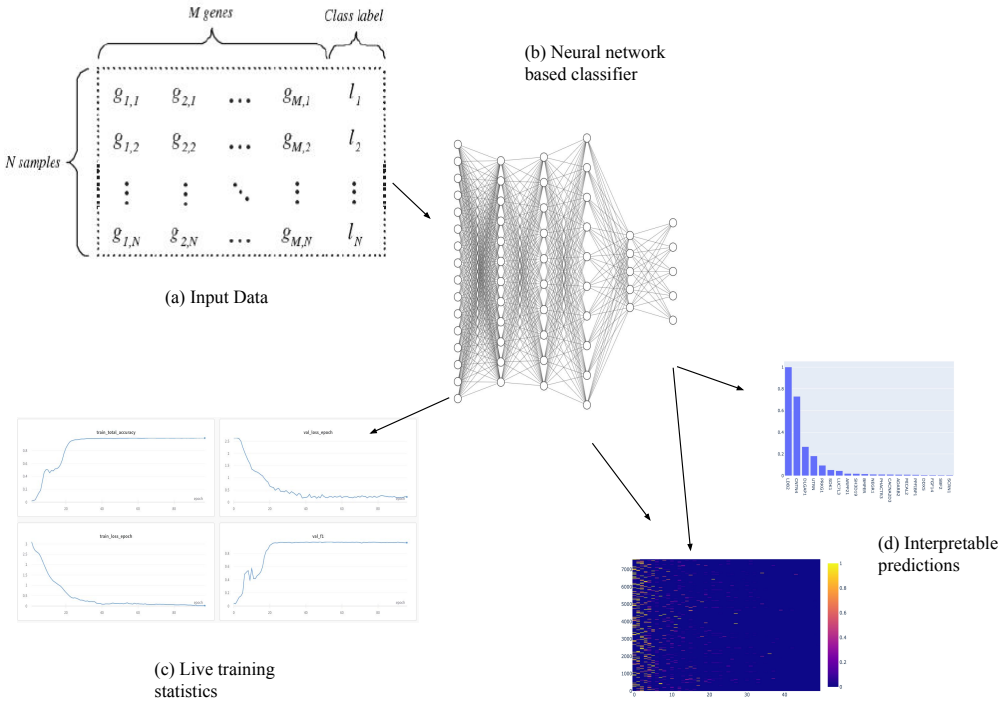


Figure 0.4: The *SIMS* pipeline. First, (a) normalized input data from expert annotations is input in the DataModule class. Next, the neural network (b) is defined with the chosen optimizer and training parameters. Live training statistics (c) can be viewed to understand training, validation and test performance. Finally, we can use the feature masks to make interpretable predictions (d) on unseen data.

SIMS is meant to be modular and fast to develop with. To this end, we allow training of multiple files of multiple data types and of arbitrary size. This is possible by reading in samples only as needed from delimited files. For HDF and files from binary data, the HDF distributed backend is used by default. This allows users to train models even when limited by memory constraints.

```
from scsims import generate_trainer
trainer, model, module = generate_trainer(
    datafiles=['data.h5ad'],
    labelfiles=['labels.csv'],
    class_label='Cell Type',
)
trainer.fit(model, datamodule=module)
```

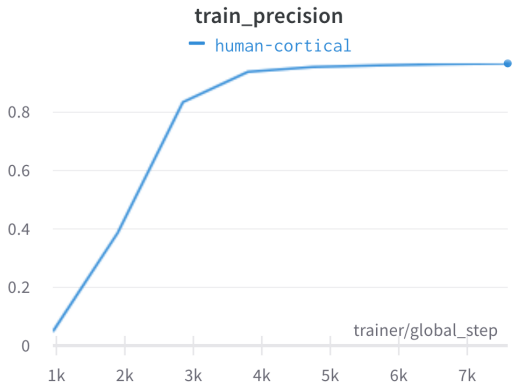
This is by far the simplest case. However, *SIMS* is built in a modular way and therefore can be easily extended to handle custom data, model and trainer requirements.

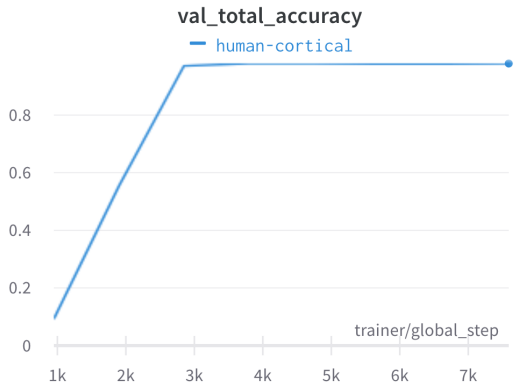
```
import pytorch_lighting as pl
from scsims import SIMSClassifier, DataModule
module = DataModule(
    datafiles=['data.h5ad'],
    labelfiles=['labels.csv'],
    class_label='Cell Type',
)
logger = pl.loggers.WandbLogger(
    project="Single-Cell Classifier",
    name=name,
)
```

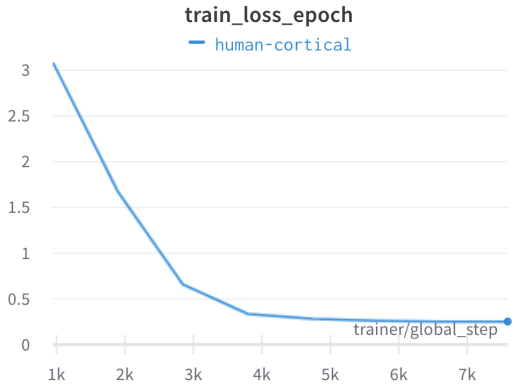
```
early_stopping_callback = pl.callbacks.EarlyStopping(
    monitor='val_loss',
    patience=4,
)
trainer = pl.Trainer(
    gpus=1,
    logger=wandb_logger,
    gradient_clip_val=0.5,
    callbacks=[
        early_stopping_callback,
    ]
)
model = SIMSClassifier(
    input_dim=module.input_dim,
    output_dim=module.output_dim,
)
trainer.fit(model, datamodule=module)
```

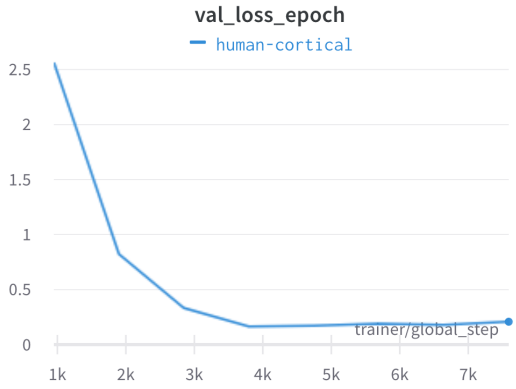
Unless specified otherwise, *SIMS* will also track over a dozen model training metrics for both the training and validation set on initial training. These metrics are calculated every mini-batch step, and aggregated appropriately at each epoch. Below, we show a select few metrics tracked for the training of the human cortical tissue model

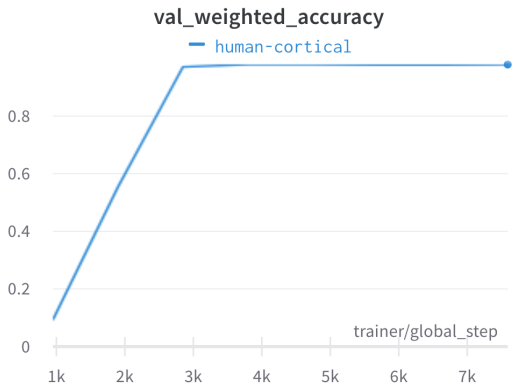
from the Allen Brain Institute, visualized on wandb.ai.

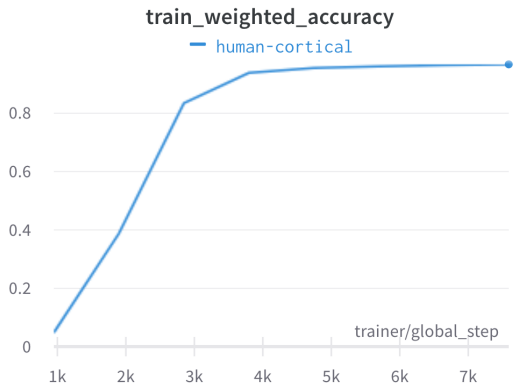












Example of Metric Tracking via the SIMS package on wandb.ai for the Human Cortical Dataset. All metrics shown here are tracked by default by SIMS.

This modular and high-level API allows end users to develop and deploy models quickly, and with minimal data preprocessing before the pipeline is used, as normalization is done within the SIMS pipeline and selecting highly variable genes via statistical tests [LT19] is handled by the learned sparse feature masks [AP21].

0.3.2 Results

We used the SIMS pipeline on multiple datasets with high accuracy and good generalization ability. Since transformer based architectures are usually data intensive to perform well, we hypothesized that models trained on larger single-cell experiments would have better test accuracy. Surprisingly, accuracy and median f1 were high even with datasets with around 50k cells. Additionally, as shown in 0.3.5, taking a small proportion of the initial dataset yielded little degradation in test accuracy. We initially benchmarked SIMS against human dental data, human cortical tissue from two datasets, human retinal tissue, and mouse cortical tissue.

Next, we visualize the distribution of the label sets across four of the datasets. Note that in biological tissue, there is often a dominant cell type and many minority classes. This is a problem for classification, since the model can achieve high unweighted accuracy by only predicting the majority classes – failing to predict rare cell types.

Dataset	Source	Technology	# Classes	# Cells
Human Cortical Model	Allen Brain Institute	SMART-Seq v4 and 10x v3 RNA-sequencing	19	47432
Mouse Cortical Model	Allen Brain Institute	SMART-Seq v4 and 10x chromium v2	42	73347
Human Retina Model	cells.ucsc.edu [LLS+19]	10x Chromium v2	13	16446
Human Dental Model	cells.ucsc.edu [KSK+20]	10x Chromium v2	9	41673
UCSF Cortical Model	cells.ucsc.edu [BAML+20]	10x Chromium v2	28	168697

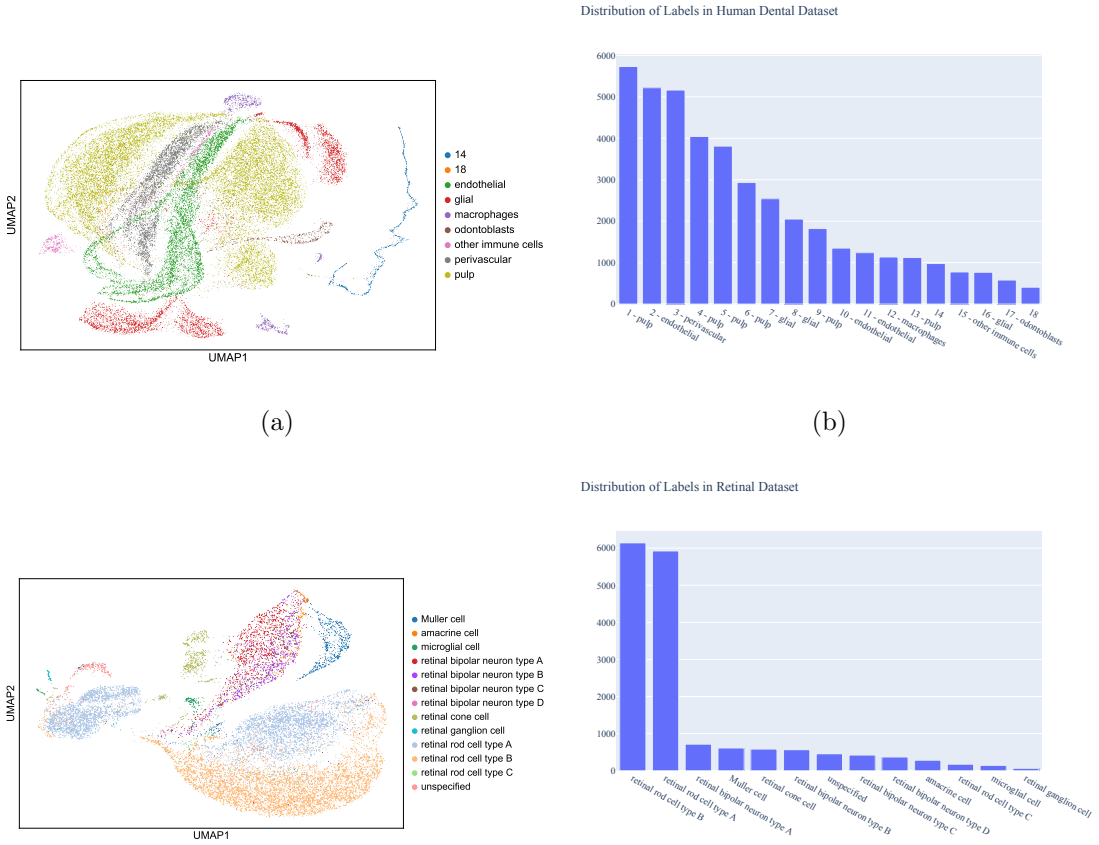
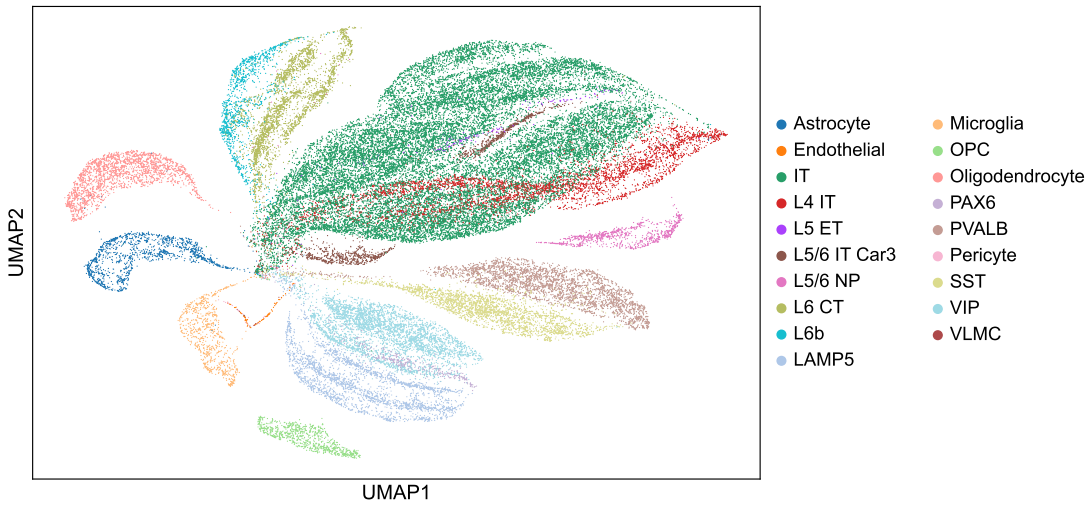


Figure 0.6: UMAP and label visualizations of the dental data [KSK+20] and retinal data [LLS+19]. (a) UMAP projection of dental data colored by cell type. (b) Distribution of labels for dental data. (c) UMAP projection of retina data colored by cell type. (d) Distribution of labels for the retina dataset.

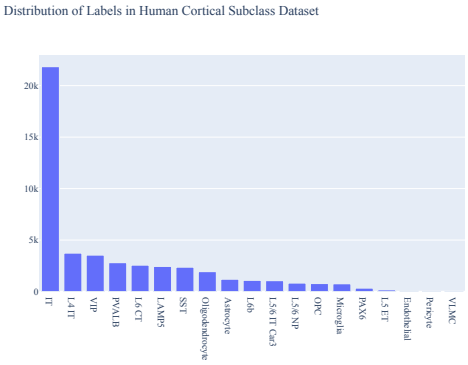
Note that the label sets are highly skewed and long-tailed, where the majority class encompasses nearly half of all samples and the minority classes can be sparse. For this reason, we compute the f1 score of each class, and calculation the median of this. Intuitively, this tells us that the model will perform about that well half the time, and

worse the other half. This is informative for when we have many rare cell types that could be ignored without much increase to the loss.

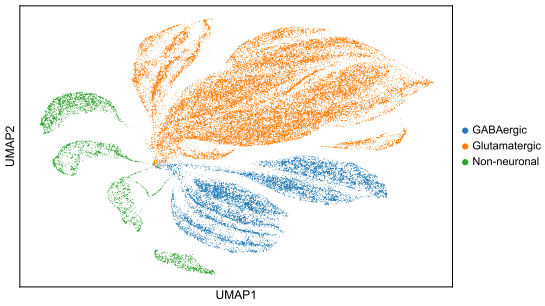
Below, we visualize the UMAP projection of the first fifty principal components of the input data, where we color each projected point by its assigned cluster via the pipeline described in 0.1.2.



(a)



(b)

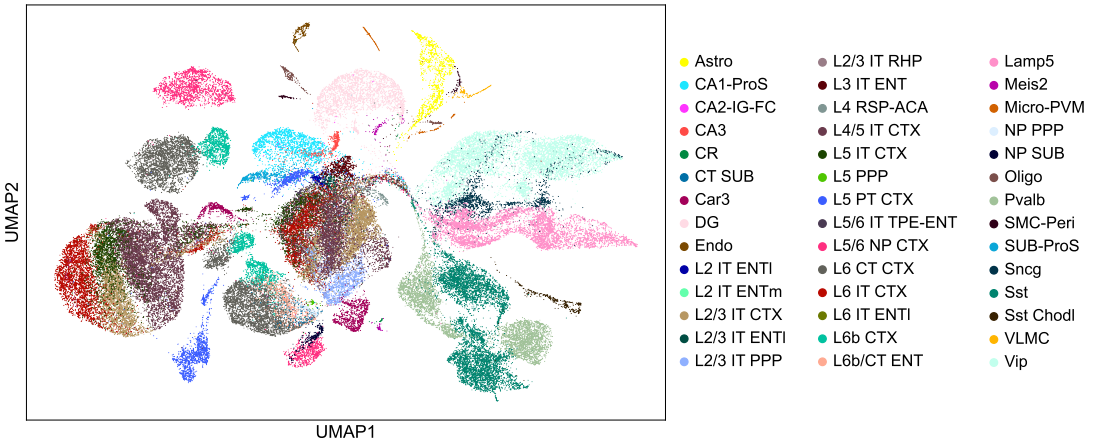


(c)

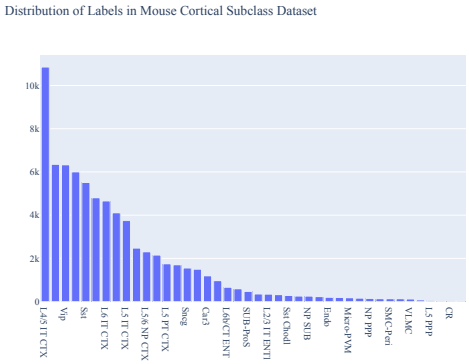
Figure 0.7: UMAP and label visualizations of the Allen Brain Institute human cortical data. (a) UMAP projection colored by cell subtype. (b) Distribution of subtype labels. (c) UMAP projection colored by cell type.

In the visualization of the human cortical data from the Allen Brain Institute, we can see clear separation in the projection by class. For the granular annotations

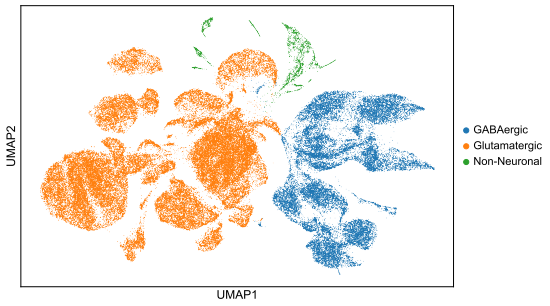
of cell subtype, there is little visual distinction in the projected data. Conversely, the larger cell phenotypes are visually distinct.



(a)

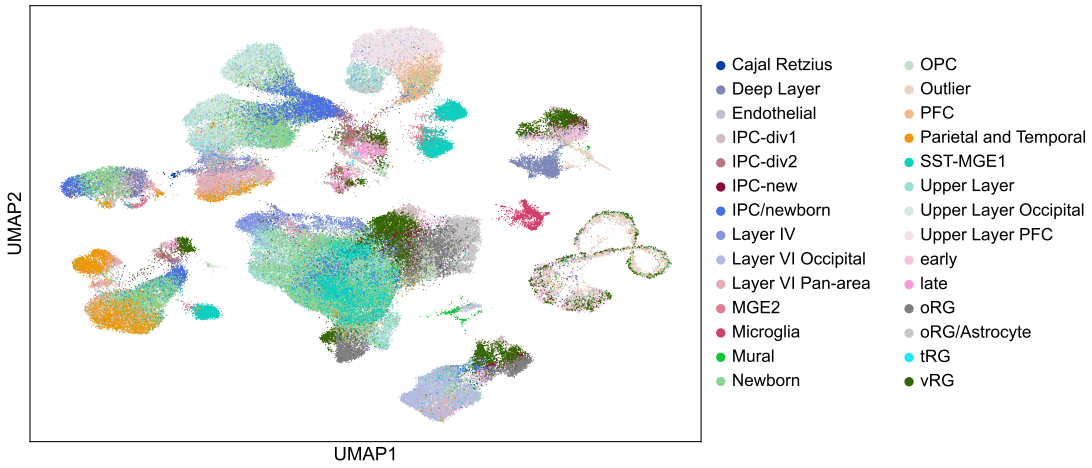


(b)

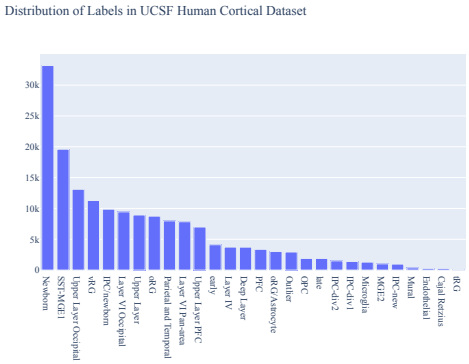


(c)

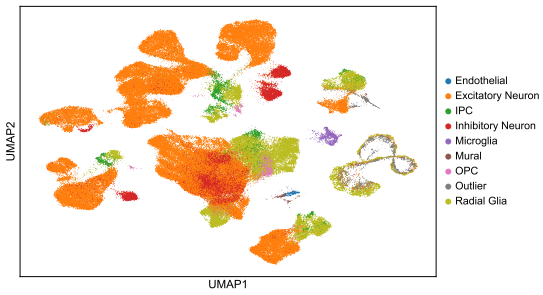
Figure 0.8: UMAP and label visualizations of the Allen Brain Institute mouse cortical data. (a) UMAP projection of the colored by cell subtype. (b) Distribution of subtype labels. (c) UMAP projection colored by main cell type.



(a)



(b)



(c)

Figure 0.9: UMAP and label distributions of human human cortical data from [BAML$^+$20]. (a) the UMAP projection of the human cortical data colored by cell subtype. (c) the distribution of these subtype labels. (c) UMAP projection colored by cell supertype.

Additionally, we projected the data onto the first and second principal components, and visualized the proportion of linear variation explained by the first thirty. Interestingly, the models with the highest accuracy also have a high percentage of vari-

ance explained by the first few principal components.

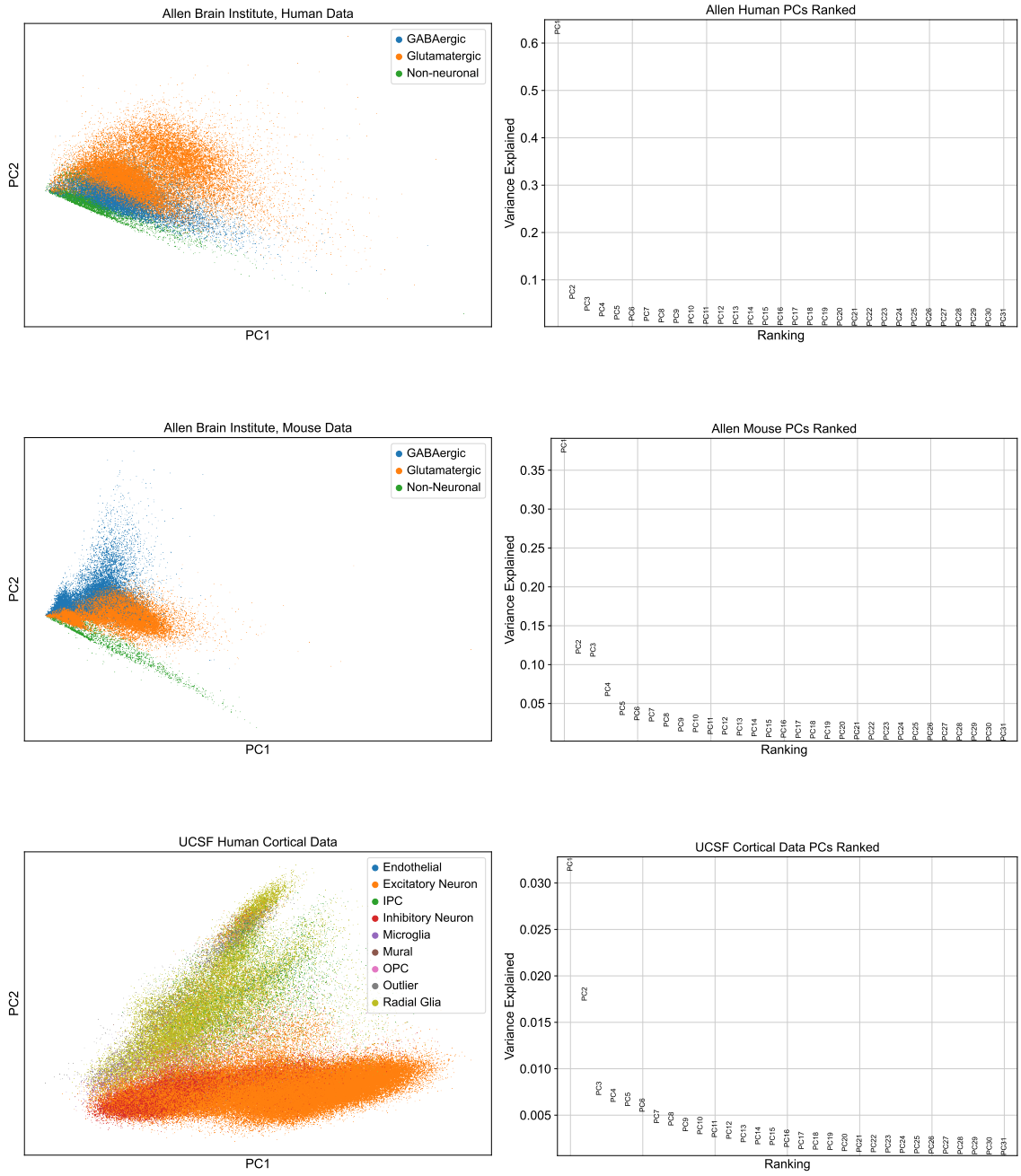


Figure 0.10: Principal component visualizations of the three benchmark datasets from cortical tissue. The left column visualizes the data listed projected onto the first two principal components, where each point is colored by its major phenotype group. The right column ranks the first ten principal components for the dataset, ordered by the proportion of the linear variation explained by each.

Interestingly, we noticed a relationship between the amount of linear variation explained by the first principal component and model performance. For data where the explained variance is high, the SIMS classifier performed more accurately, even when the granularity of annotation was high.

We trained each model remotely on a distributed compute cluster via GPU, so all calculations were done with 32 bit precision. Metric results are shown below.

Model	Accuracy	Weighted Accuracy	Median F1
Human Cortical Model	0.9667	0.9812	.986
Mouse Cortical Model	0.9686	0.9755	0.9659
Human Retina Model	0.857	0.92	0.9219
Human Dental Model	0.9556	0.9598	0.9733
UCSF Cortical Model	0.7354	0.7298	0.7245

Model	Precision	Recall	Specificity	Loss
Human Cortical Model	.9812	0.9812	0.999	0.2068
Mouse Cortical Model	0.9686	0.9686	0.9992	0.4153
Human Retina Model	0.9323	0.9223	0.922	0.2873
Human Dental Model	0.9801	0.9801	0.9975	0.2854
UCSF Cortical Model	0.7298	0.7298	0.9887	0.9097

Table 0.1: Metric results for the SIMS pipeline on the test set. Precision, recall, and specificity were calculated using micro averaging.

For all models, the Adam optimizer [KB14] was used, with a learning rate of $r = 0.01$ and a weight decay of $w = 1e - 3$. We used the cross-entropy loss function, and weighted samples inversely proportional to their frequency in the dataset. Model convergence was assumed when the absolute validation accuracy did not increase for 4

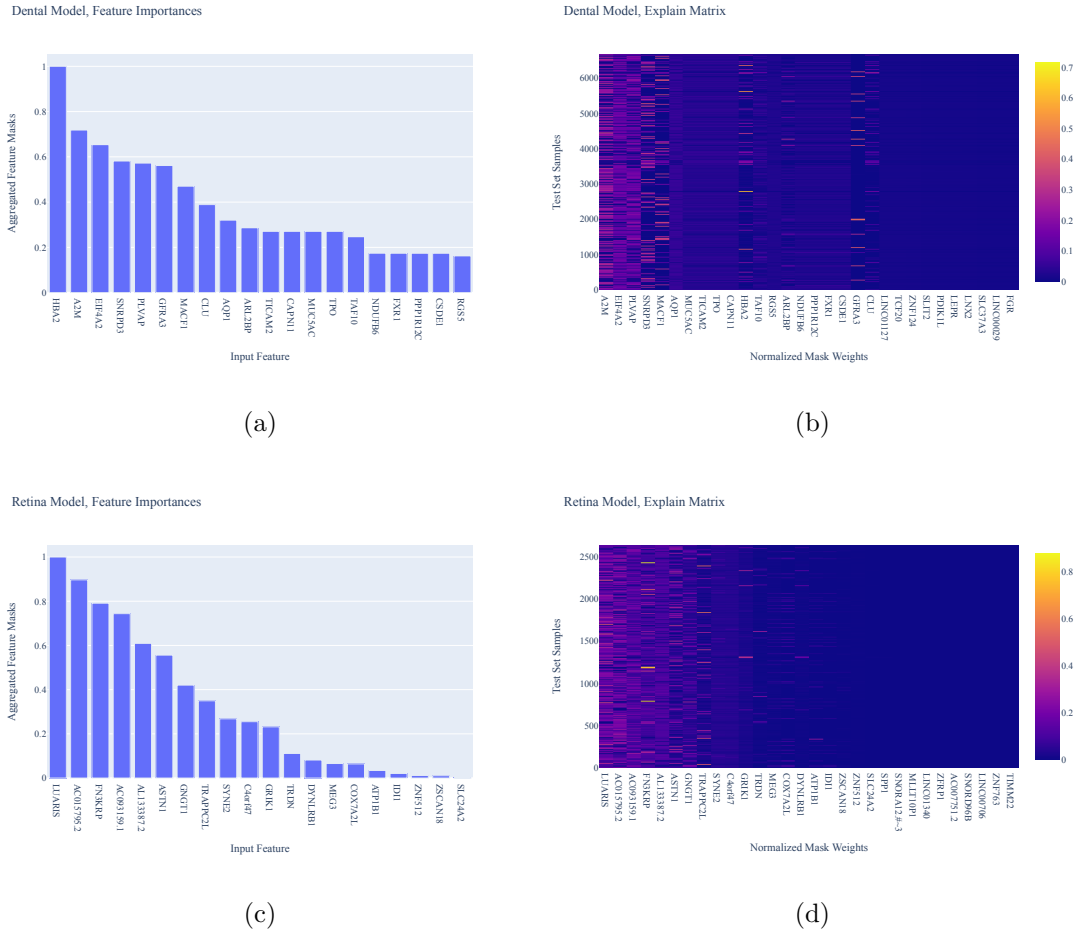
epochs. A learning rate optimizer was used such that $l \leftarrow 0.75l$ when the validation loss did not improve for twenty epochs. In all cases, models reached convergence by the early stopping criterion on validation accuracy before the maximum number of epochs (500) was reached. Gradient clipping was used to avoid exploding gradient values. Although we used a train, validation and test split for reducing overfitting via hyperparameter tuning bias, the only hyperparameter tuned was the learning rate, once, to avoid divergence in the loss. Training took less than 20 epochs for most models. For all models, we found model training to be consistent and had less than three cases of suboptimal convergence due to poor initialization. See 0.3.5 for more on loss convergence. The train, validation and test sets were stratified, meaning the distribution of labels is the same in all three (up to an error of one sample, when the number of samples for a given class was not divisible by three).

For all datasets, all models were trained using the most granular annotation available. As seen in Table 1, the SIMS pipeline performed well with varying levels of annotation granularity. Accuracy was the lowest on the UCSF cortical dataset with 28 classes. We now use sparse feature masks of [AP21] to visualize feature importance at a global level, sample level, and by class.

0.3.3 Interpretability Analysis

Unlike other deep learning methods commonly used for single-cell label prediction [CBP21], SIMS is one of the few that provides direct interpretability from the input features (0.2.5). Below, we have the top feature weights for the models trained on

our benchmarking data, on both a global basis and sample-wise basis for the test set.



(a) (b)

(c) (d)

Figure 0.11: Feature weights aggregated over all test samples for the dental and retina models. (a) Distribution of global feature weights for the dental model trained on [KSK+20] data. (b) Matrix of normalized weights (input genes) across all samples on the test set. (c) Distribution of global feature weights for the retina model trained on [LLS+19]. (d) Matrix of normalized feature weights for all samples on the test set.

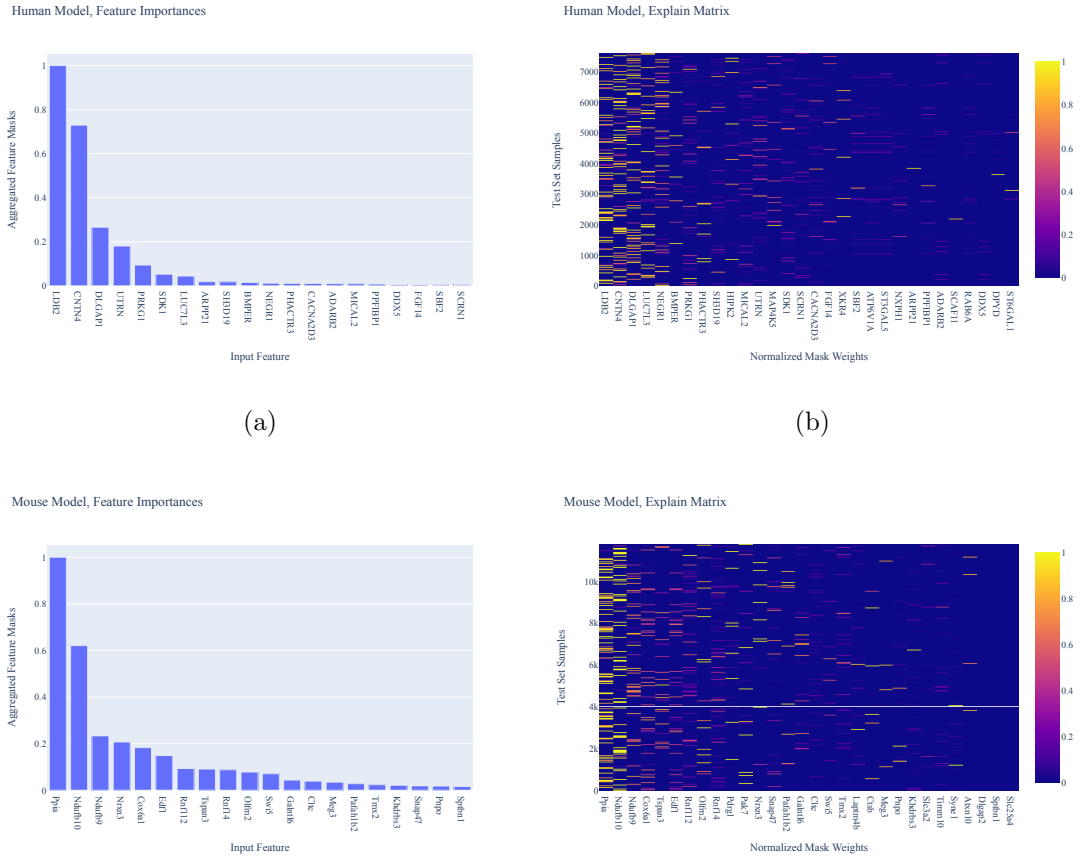


Figure 0.12: Feature weights for models trained on the Allen Brain Institute human and mouse cortical datasets. (a) Distribution of global feature weights for the human cortical model, trained on all brain regions. (b) Matrix of normalized weights (input genes) across all samples on the human cortical test set. (c) Distribution of global feature weights for the mouse cortical model. (d) Matrix of normalized feature weights for all samples on the mouse cortical test set.

For figures 11 and 12, we aggregated all the feature masks over all test samples. Then, we normalized by the largest weight since the relative scale between models isn't meaningful, but rather the proportion of each weight. Finally, we visualized the top

twenty features used. On the right, we have the so-called "explain matrix" [AP21] sorted by feature sum across all samples, for the test set. This allows us to visualize the fact that across different samples, different genes are being used for classification. Unlike [XLM+21], we have direct interpretability on the input features.

We then partitioned the sample-wise explain matrices by class, then averaged the rows. We aggregated this in a new table, and finally normalized each column (gene) by dividing by its maximum. We visualize the top 50 columns by norm. Although some genes may be used by only a few classes and therefore have small total column sum, figures 13 and 14 serve to show the different distributions of genes across classes.

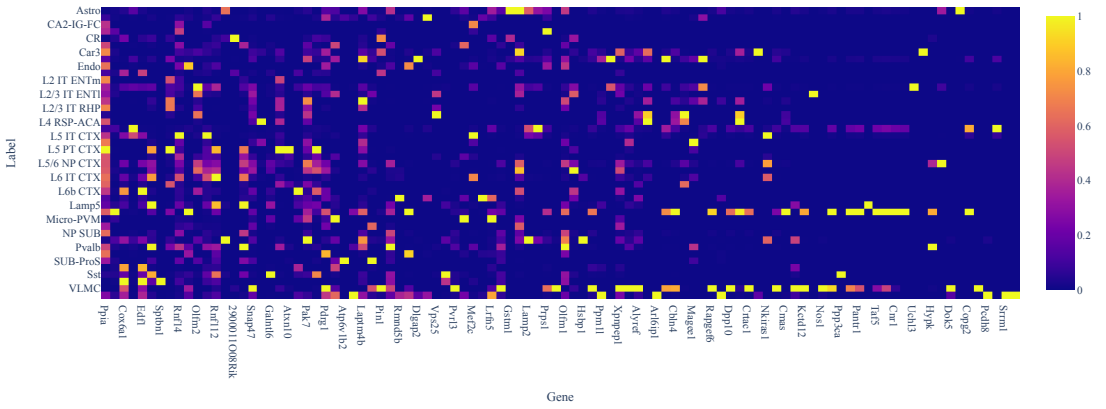


Figure 0.13: Top genes aggregated over cell subtype for the Allen Brain Institute mouse cortical data with $C = 42$ cell types. Each row represents a class label, and the columns are normalized feature mask values.

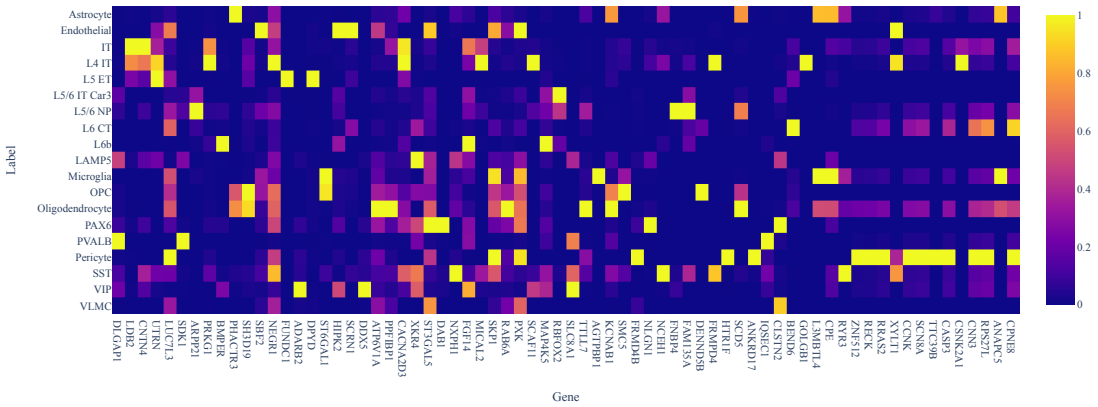


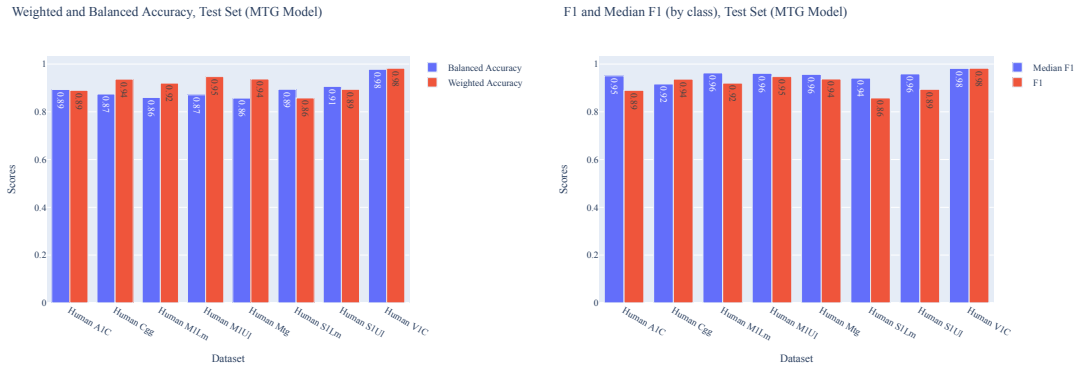
Figure 0.14: Top genes aggregated over cell subtype for the Allen Brain Institute human cortical data with $C = 19$ cell types. Each row represents a class label, and the columns are normalized feature mask values.

By calculating the weights across the sparse feature masks, we are able to measure the contribution of each input feature to the total classification process, while also promoting sparsity in downstream weights, allowing for a smaller and more computationally efficient model without sacrificing accuracy [FC18]. We now turn studying generalization capability between multiple tissue samples from multiple experiments.

0.3.4 Generalization Capability

In addition to performing well on the test set, we also want the SIMS model to perform well on data from different studies. Although the test set is a proxy for unseen data, the distribution is assumed to be the same as the data on which the model is trained [WLZ+16]. However, for data collected from different studies with technical variation, this assumption may not hold. An important test for real-world use case

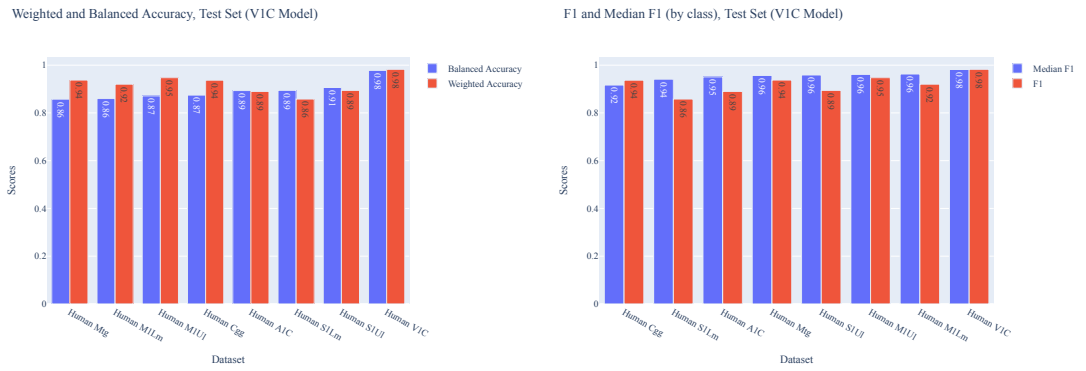
is the ability for a model to perform well on data with a potentially different input distribution. The Allen Brain Institute data comprises multiple tissue samples from multiple experiments, and in total samples tissue from several parts of the human and mouse brain. To test the ability of the SIMS model to generalize to other datasets, we first trained a model on the human middle temporal gyrus from the Allen data, and tested against all other tissue samples from the Allen human data.



(a) (b)

Figure 0.15: Metric results for the SIMS pipeline trained on the Allen Brain Institute human MTG data and benchmarked on all available human brain tissue data. (a) Balanced and weighted accuracy. (b) Aggregated F1 and median F1 scores.

In general, the model performs well when predicting labels from different tissue samples. For both models trained on MTG and V1C data respectively, weighted accuracy is lowest against the SILm tissue sample. The median F1 for both models is lowest against cGG tissue samples.



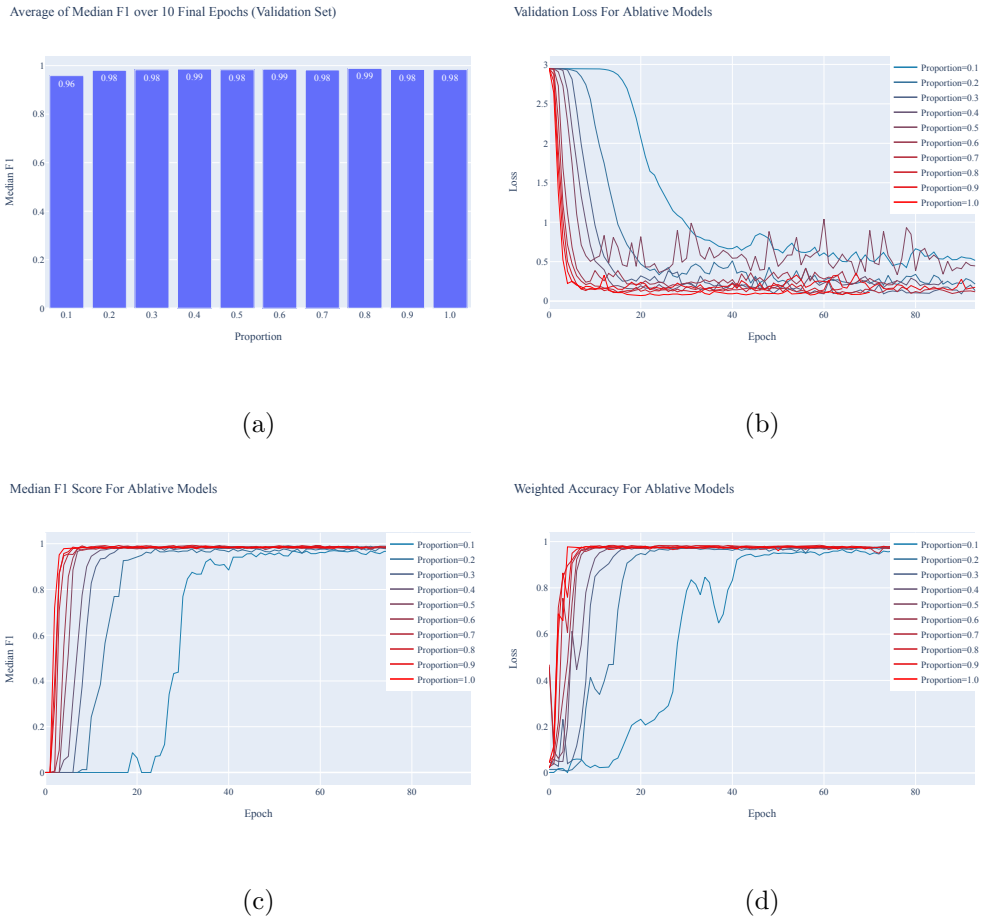
(a) (b)

Figure 0.16: Metric results for the SIMS pipeline trained on the Allen Brain Institute mouse V1C (Visual Cortex Region 1) data and tested on all other mouse brain tissue data. (a) Balanced and weighted accuracy. (b) Aggregated F1 and median F1 scores.

0.3.5 Ablative Studies

In machine learning, an ablation is the removal of a component of a machine learning system in order to test robustness to different conditions [MLdPM19]. By restricting particular parts of the modeling pipeline, glean insight into performance causality, guiding future model research and data experimentation. With SIMS, we performed an ablative study on model capacity as a function of dataset size. Since transformer-based architectures in computer vision and natural language tasks tend to require large training sets to obtain accurate results [LSB+21], we hypothesized that progressively restricting the size of the training set would lead to a fast drop-off in test accuracy. Instead, our results indicated the SIMS model yielded comparable train and test errors for up to a 90% data truncation when trained on the Allen Brain Institute

data.



(a) (b)

(c) (d)

Figure 0.17: Metrics for model trained on the Allen Brain Institute Human MTG data. Total number of cells in initial training set was $N = 47432$ and $M = 19$ classes. Each proportion p corresponds to a train/val/test split of pN cells. Data was stratified in the train/val/test split, and each split was determined with a deterministic seed for all runs. (a) Average of the median F1 score, across the final ten model epochs on the validation set for each ablative model. Each proportion p corresponds to a train/val/test split of pN cells. (b) Validation loss as a function of the number of epochs trained. (c) Median F1 score as a function of the number of epochs trained. (d) Weighted accuracy as a function of epochs trained.

Interestingly, there was only a 2% difference in median F1 score from the smallest training set to the largest training set, when tested against unseen samples from the same experiment. Since the difference in training time may be worth the trade-off in capability for some use cases, we tested how small of a dataset would be acceptable for good in-distribution performance. Below, we show that a sample proportion of $p = 0.09 \approx 4000$ cells with 19 cell types gives suitable performance on the test set of 900 cells. However, since the datasets were small, the splits could not be stratified.

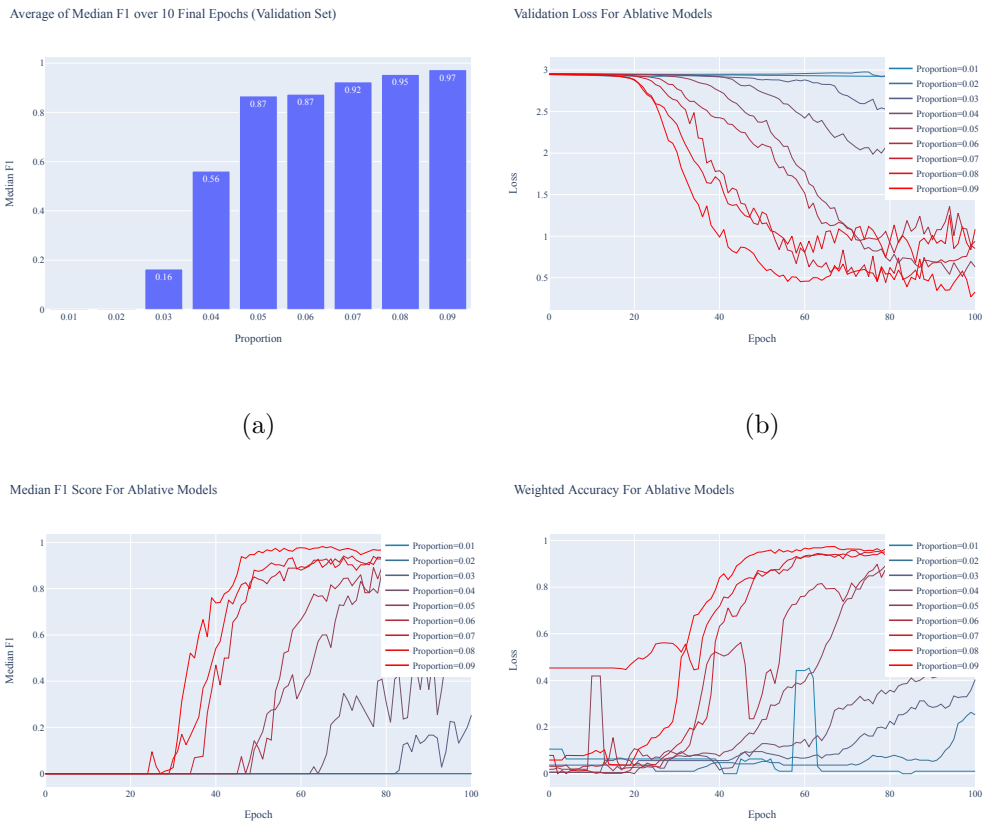


Figure 0.18: Metrics for the SIMS model trained on the Allen Brain Institute Human MTG data with smaller training proportions. The train/val/test splits were stratified when the dataset was large enough to do so, and each split was determined with a deterministic seed for all runs. (a) Average of the median F1 score, across the final ten model epochs on the validation set for each ablative model. Each proportion p corresponds to a train/val/test split of pN cells. (b) Validation loss as a function of the number of epochs trained. (c) Median F1 score as a function of the number of epochs trained. (d) Weighted accuracy as a function of epochs trained.

To test if this data efficiency holds for more granular annotations, we performed

the same experiment on the Allen Brain Institute Mouse data with $N = 42$ classes.

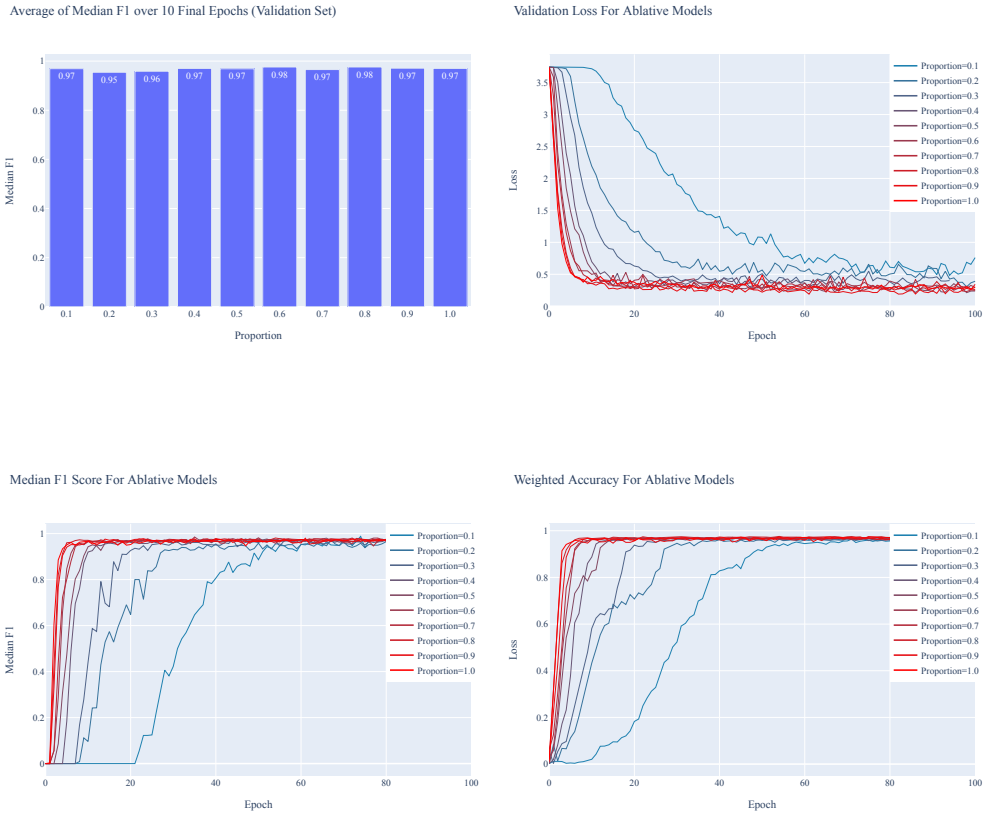


Figure 0.19: Metrics for model trained on the Allen Brain Institute Mouse cortex data. Total number of cells in initial training set was $N = 73347$ and $M = 42$ classes. (a) Average of the median F1 score, across the final ten model epochs on the validation set for each ablative model. Each proportion p corresponds to a train/val/test split of pN cells. (b) Validation loss as a function of the number of epochs trained. (c) Median F1 score as a function of the number of epochs trained. (d) Weighted accuracy as a function of epochs trained.

Even with increased annotation granularity, the [AP21] model performed well on unseen data from the same initial dataset. In both experiments, there is a monotonic relationship between convergence time and dataset size, whereas the size of the dataset

decreases the number of epochs until convergence increases. Overall, we found that SIMS performed well when classifying cells across multiple tissue types. Although these datasets were collected across multiple experiments, they are all from the Allen Brain Institute. In our future work, we will benchmark model's trained on data from one lab and validate against other data, in order to test robustness against other sources of technical variation. We now measure how SIMS performed when compared with another popular single-cell analysis pipeline, the scVI and scANVI tools [XLM$^+$21] [LRC$^+$18].

0.3.6 Benchmarking

Automated cell type identification has many current tools available [AMC$^+$19]. Here, we focus on tools written in Python [VRD09], as we find Python integrates best with the current existing tools. Moana [WY18] uses a support vector machine with a linear kernel to define multiple separating hyperplanes for classification. Other methods such at scPred [AHSJ$^+$19] use a support vector machine with a radial kernel for improved results. Simpler algorithms such as [DHB$^+$21] use a priori known markers to assign cell types to differentially expressed genes for each cell. Actinn, the first neural-network based classifier in single-cell [MP20] uses a fully-connected feedforward neural network on PCA space for classification. In [AMC$^+$19], they find that this method underperforms current benchmarks, likely due to the strong sparsity in single-cell data not being integrated as a prior into the feedforward architecture. Both scANVI [LRC$^+$18] [XLM$^+$21] and cell-BLAST [CWL$^+$20] use generative neural networks to learn compressed representations of the inputs, and use Bayesian modeling to generate posterior

probabilities for each cell. The most popular of these methods in the current single-cell pipelines is scANVI, due to its ease of use and ability to both build joint embeddings and calculate differentially expressed genes.

To validate the SIMS pipeline as a valid method for supervised label classification we benchmarked the same datasets under the exact same train, validation and test splits against the scANVI method. To be as unbiased as possible, we trained both the initial scVI model to generate the latent cell embedding and the scANVI semi-supervised classification model for 150 epochs with early stopping on validation accuracy, in order to guarantee convergence without overfitting. In both cases, the model did stop before 150 epochs with the stopping criterion.

	scANVI	SIMS
Accuracy	0.913	**0.976**
Median F1	0.0	**0.9834**

(a) Metrics on test set, mouse cortical dataset from the Allen Brain Institute dataset. $C = 42$ classes.

	scANVI	SIMS
Accuracy	0.974	**0.9812**
Median F1	0.9819	**0.9868**

(b) Metrics on test set, human cortical dataset from the Allen Brain Institute dataset. $C = 19$ classes.

	scANVI	SIMS
Accuracy	0.5687	**0.7354**
Median F1	0.0	**0.7245**

(c) Metrics on test set, human cortical dataset from the UCSF Bhaduri dataset [BAML$^+$20]. $C = 28$ classes.

	scANVI	SIMS
Accuracy	**0.93123**	0.9223
Median F1	**0.9375**	0.9223

(d) Metrics on test set, retina dataset from [LLS$^+$19]. $C = 13$ classes.

Table 0.2: Benchmarking results for SIMS vs scANVI

The pancreas dataset was scaled so values were not raw counts. Therefore, we weren't able to use the scANVI pipeline. We note that SIMS outperforms scANVI in all but one case, but the discrepancy in accuracy and median F1 of less than one percent is likely due to noise from the train and test splits. Additionally, in the case of deep annotation in the Bhaduri UCSF data [BAML$^+$20], SIMS outperforms by nearly 17%. In the two final cases, SIMS outperforms scANVI by approximately five and one

percent. Therefore, SIMS is on par with or exceeding [XLM+21], with the additional benefits of interpretability as described in 0.2.5 and 0.2.4.

0.3.7 Inference on Tissue Cultures

A primary use case for the both 3D organoid models [PBA+19] and 2D human induced pluripotent stem cells [BCS+20] is characterizing the growth of human organs modeled in vitro. In our goal to build a good 3D model of the human brain via organoids in vitro, an important step is making sure the distribution of cells is the same in both primary tissue sample and organoids.

However, the differences between cell feeding times in stem cell organoids changes the transcriptomic characterization of cells [BAML+20]. Although these changes are limited to small number of genes [VEN+22], these molecular changes make the clustering process difficult and therefore distinct cell types difficult to detect. Machine learning models, however, can perform inference on transcriptomic data. In the case of the SIMS model, we verified that no stress genes had nonzero weight in the feature masks. Below, we have experimental results from a UCSF in which cells from mouse embryos were transplanted into human cortical organoids. A priori, these cells were known to be of inhibitory neuron origin. Using the SIMS pipeline, we trained a model on a large dataset of inhibitory neurons, and used this model to infer cell type within the transplanted cells.

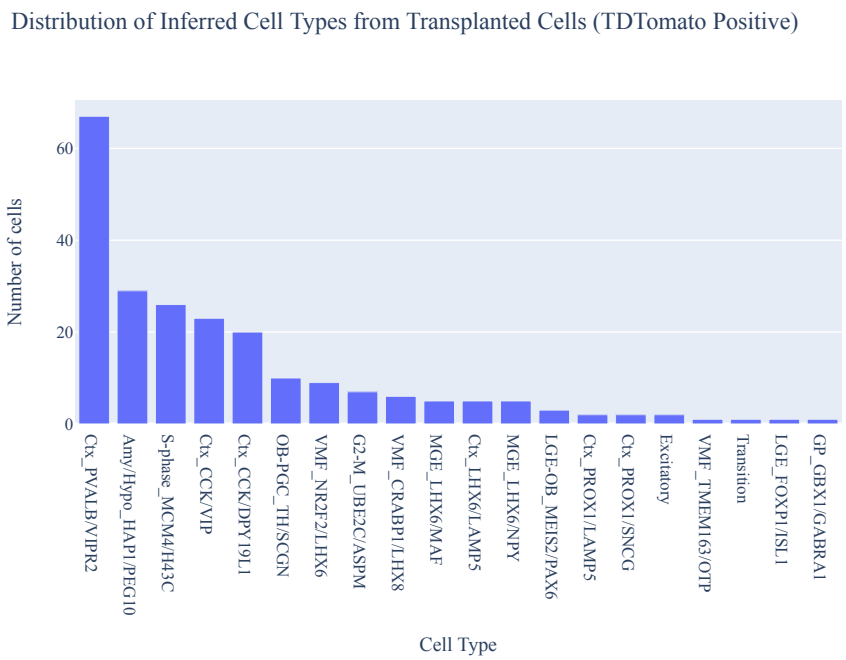


Figure 0.20: Inferred cell types using the SIMS pipeline trained on inhibitory cortical neurons from the M.R. UCSF experiment.

These results are expected biologically, as all cells transplanted were dissociated from mouse cortex. Additionally, the cells were known to be inhibitory. This means the model definitely misclassified the four cells predicted as excititory neurons.

In [BAML$^+$20], samples were taken from both primary and organoid tissues. Cell types for the primary tissue were inferred via the pipeline described in 0.1.2. However, due to cell stress in a subset of genes [BAML$^+$20] [VEN$^+$22], clustering was difficult to perform, and distinct cell subtypes were not able to be inferred. Using the explainability of the SIMS model [AP21], we were able to confirm that the genes associated with cells stress were not used for prediction. Therefore, we reason that inference using

the SIMS model is more robust to cell stress, since the model does not directly use that part of the transcriptome. We visualize the inferred cell types below.

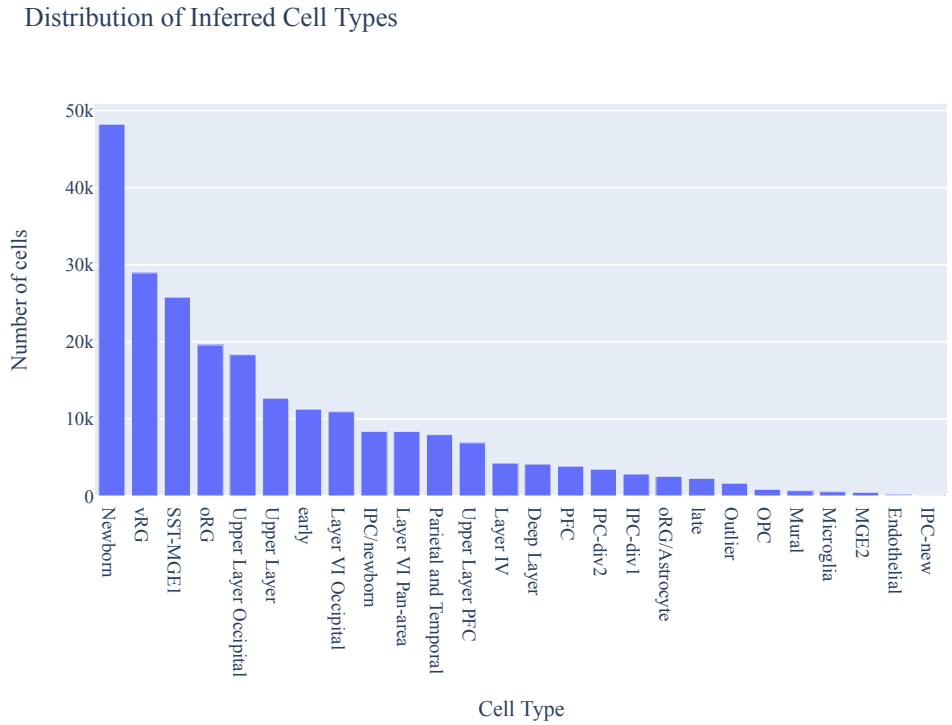


Figure 0.21: Inferred cell types of organoid data from [BAML+20] using the SIMS pipeline trained on primary data from the same reference.

The inferred cell types were concluded to be biologically likely from domain expert M.R.

0.3.8 Discussion

As biologists seek to understand the functionality and relation of individual cells both in vivo and in vitro, single-cell RNA-seq allows for transcriptomics at unprece-

dented resolution. By measuring mRNA levels in each cell, we are able to characterize the distribution of discrete and continuous morphological properties, allowing important insight into biological function. However, as these experiments increase in size, the need for an automated pipeline of key analyses is critical for three major reasons. Firstly, these computational steps are often require expert domain knowledge for cluster annotation, limiting rapid analyses for experimental prototyping. Secondly, the process is both manual and recursive. This leads to potential redundancy by repeating statistically challenging and automatable steps. Thirdly, as 2D dissociated tissue models [BCS+20] and 3D organoid models [PBA+19] become increasingly important for data-driven biological discovery, clustering and marker gene annotation from molecular data can be impeded by cell stress and technical variability [BAML+20] [VEN+22]. For this reason, interpretable and powerful models built on high-quality ground truth data will become increasingly important for phenotypic and morphological predictions. While current methods [XLM+21] [CWL+20] [WY18] [AMC+19] often rely on a priori known markers or variational autoencoders [KW19] for isolating biological variability and classification, we turn to an interpretable deep learning approach [AP21], focusing on both data efficiency, robustness, and ease-of-use. We call our method SIMS; scalable, interpretable machine learning for single-cell.

In the previous sections, we perform several cell classification tasks using the SIMS pipeline for single-cell label classification. Since cells between tissues have subtle molecular differences [BAML+20], we tested the model's ability to predict on different tissue types than the training set. Between tissue samples from both human cortical

tissue and mouse cortical tissue, we show that SIMS performs generally well at predicting cell types on data with possible distribution shift. Next, we performed several ablative studies using the same dataset, testing model capability as a function of dataset proportion. Surprisingly, we find that the SIMS classifier, without any external data normalization or preprocessing, performs well with less than ten percent of the original training set. This holds for samples from both human cortical data and mouse cortical data, and steep drop-offs in performance are only seen below six percent of the initial dataset size. We will continue to perform ablations against multiple other datasets, but we conclude that SIMS has high test accuracy and median F1 even with significantly smaller training datasets. Therefore, the SIMS model does not need to see many samples of each cell type to infer the salient genes for classification.

We also benchmarked against scANVI, another popular method for single-cell cell type classification [XLM$^+$21]. We found that in one case, SIMS performed slightly below scANVI in accuracy and median F1 score, but was less than a one-percent difference. However, in all other cases SIMS outperformed scANVI. In particular, the absolute increase was upwards of 17% when benchmarked against the UCSF human primary tissue data [BAML$^+$20], and five percent when benchmarked against the Allen Brain Institute mouse data. We conclude that the SIMS pipeline is valuable tool in the field of automated single-cell classification.

0.4 Conclusion and Future Work

Single-cell transcriptomics has provided incredible resolution and understanding of cell distribution within tissue samples at the level of individual cells. An important task in the pipeline of single-cell analysis is morphological classification, such as cell type or discretized cell type. We provide SIMS, a [VRD09] pipeline for quick and easily creation of classifiers for single-cell data. SIMS is meant for quick model development with minimal data preprocessing, so results can be inferred quickly and accurately for experiment design and inference. In addition to handling arbitrary number of files of arbitrary size with varying file types, SIMS has a high-level API for dataset creation and model training in as little as two lines of code. We show that the SIMS model is robust against difference tissue samples, and performs well with subsampled data and tissues with a high number of cell types.

In the future, we plan to implement and test several more processes. First, we are working on implementing self-supervised learning via binary masking as described in 0.2.4, which can reduce test error, training time, and increase accuracy [AP21]. We will provide these pretrained encoders for multiple data sources, so fine-tuning on a new labeled dataset becomes even faster. Additionally, the abundance of single-cell classification methods available [AMC+19], it is important to benchmark against a variety of datasets and a variety of other available methods. We plan to benchmark against all methods listed in [AMC+19], as a function of both dataset size and against multiple tissue types with a varying number of cell types. Additionally, more ablation

studies should be performed to test the SIMS pipeline's robustness against dataset size when the number of cell types is large, such as in the case of human brain tissue.

Bibliography

[AAVPS13] Roberto Alejo, JA Antonio, Rosa Maria Valdovinos, and J Horacio Pacheco-Sánchez. Assessments metrics for multi-class imbalance learning: A preliminary study. In *Mexican Conference on Pattern Recognition*, pages 335–343. Springer, 2013.

[AHSJ+19] Jose Alquicira-Hernandez, Anuja Sathe, Hanlee P Ji, Quan Nguyen, and Joseph E Powell. scpred: accurate supervised method for cell-type classification from single-cell rna-seq data. *Genome biology*, 20(1):1–17, 2019.

[AKJ04] Rehan Akbani, Stephen Kwek, and Nathalie Japkowicz. Applying support vector machines to imbalanced datasets. In *European conference on machine learning*, pages 39–50. Springer, 2004.

[AMC+19] Tamim Abdelaal, Lieke Michielsen, Davy Cats, Dylan Hoogduin, Hailiang Mei, Marcel JT Reinders, and Ahmed Mahfouz. A comparison of automatic cell identification methods for single-cell rna sequencing data. *Genome biology*, 20(1):1–19, 2019.

[AP21] Sercan O Arık and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *AAAI*, volume 35, pages 6679–6687, 2021.

[AST+17] Philipp Angerer, Lukas Simon, Sophie Tritschler, F Alexander Wolf, David Fischer, and Fabian J Theis. Single cells make big data: New challenges and opportunities in transcriptomics. *Current Opinion in Systems Biology*, 4:85–91, 2017.

[BAML+20] Aparna Bhaduri, Madeline G Andrews, Walter Mancia Leon, Diane Jung, David Shin, Denise Allen, Dana Jung, Galina Schmunk, Maximilian Haeussler, Jahan Salma, et al. Cell stress in cortical organoids impairs molecular subtype specification. *Nature*, 578(7793):142–148, 2020.

[BCS+20] Emily E Burke, Joshua G Chenoweth, Joo Heon Shin, Leonardo Collado-Torres, Suel-Kee Kim, Nicola Micali, Yanhong Wang, Carlo Colantuoni, Richard E Straub, Daniel J Hoeppner, et al. Dissecting transcriptomic signatures of neuronal differentiation and maturation using ipscs. *Nature communications*, 11(1):1–14, 2020.

[BHGMP22] A Sina Booeshaghi, Ingileif B Hallgrímsdóttir, Ángel Gálvez-Merchán, and Lior Pachter. Depth normalization for single-cell genomics count data. *bioRxiv*, 2022.

[CBP21] Tara Chari, Joeyta Banerjee, and Lior Pachter. The specious art of single-cell genomics. *bioRxiv*, 2021.

[CWL+20] Zhi-Jie Cao, Lin Wei, Shen Lu, De-Chang Yang, and Ge Gao. Searching large-scale scrna-seq databases via unbiased cell embedding with cell blast. *Nature communications*, 11(1):1–13, 2020.

[DHB+21] Sergii Domanskyi, Alex Hakansson, Thomas J Bertus, Giovanni Paternostro, and Carlo Piermarocchi. Digital cell sorter (dcs): a cell type identification, anomaly detection, and hopfield landscapes toolkit for single-cell transcriptomics. *PeerJ*, 9:e10670, 2021.

[Fal20] W Falcon. Pytorchlightning/pytorch-lightning. *Pytorch Lightning*, 2020.

[FC18] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[FH19] Matthias Feurer and Frank Hutter. Hyperparameter optimization. In *Automated machine learning*, pages 3–33. Springer, Cham, 2019.

[GBV20] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multiclass classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.

[HETL17] Ashraful Haque, Jessica Engel, Sarah A Teichmann, and Tapio Lönnberg. A practical guide to single-cell rna-sequencing for biomedical research and clinical applications. *Genome medicine*, 9(1):1–12, 2017.

[HLT19] Fengxiang He, Tongliang Liu, and Dacheng Tao. Control batch size and

learning rate to generalize well: Theoretical and empirical evidence. *Advances in Neural Information Processing Systems*, 32, 2019.

[IPK21] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. In *Programming with TensorFlow*, pages 87–104. Springer, 2021.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kor11] Sandeep Koranne. Hierarchical data format 5: Hdf5. In *Handbook of open source tools*, pages 191–200. Springer, 2011.

[KS05] Frances Y Kuo and Ian H Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52(11):1320–1328, 2005.

[KSK+20] Jan Krivanek, Ruslan A Soldatov, Maria Eleni Kastriti, Tatiana Chontorotzea, Anna Nele Herdina, Julian Petersen, Bara Szarowska, Marie Landova, Veronika Kovar Matejova, Lydie Izakovicova Holla, et al. Dental cell type atlas reveals stem and differentiated cell types in mouse and human teeth. *Nature communications*, 11(1):1–18, 2020.

[KW19] Diederik P Kingma and Max Welling. An introduction to variational autoencoders. *arXiv preprint arXiv:1906.02691*, 2019.

[LLS+19] Samuel W Lukowski, Camden Y Lo, Alexei A Sharov, Quan Nguyen, Lyujie Fang, Sandy SC Hung, Ling Zhu, Ting Zhang, Ulrike Grünert,

Tu Nguyen, et al. A single-cell transcriptome atlas of the adult human retina. *The EMBO journal*, 38(18):e100811, 2019.

[LRC+18] Romain Lopez, Jeffrey Regier, Michael B Cole, Michael I Jordan, and Nir Yosef. Deep generative modeling for single-cell transcriptomics. *Nature methods*, 15(12):1053–1058, 2018.

[LSB+21] Yahui Liu, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco Nadai. Efficient training of visual transformers with small datasets. *Advances in Neural Information Processing Systems*, 34, 2021.

[LT19] Malte D Luecken and Fabian J Theis. Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular systems biology*, 15(6):e8746, 2019.

[MA16] Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pages 1614–1623. PMLR, 2016.

[MLdPM19] Richard Meyes, Melanie Lu, Constantin Waubert de Puiseau, and Tobias Meisen. Ablation studies in artificial neural networks. *arXiv preprint arXiv:1901.08644*, 2019.

[MP20] Feiyang Ma and Matteo Pellegrini. Actinn: automated identification of cell types in single cell rna sequencing. *Bioinformatics*, 36(2):533–538, 2020.

[Nak21] Preetum Nakkiran. *Towards an Empirical Theory of Deep Learning*. PhD thesis, Harvard University, 2021.

[PBA+19] Alex A Pollen, Aparna Bhaduri, Madeline G Andrews, Tomasz J Nowakowski, Olivia S Meyerson, Mohammed A Mostajo-Radji, Elizabeth Di Lullo, Beatriz Alvarado, Melanie Bedolli, Max L Dougherty, et al. Establishing cerebral organoids as models of human-specific brain evolution. *Cell*, 176(4):743–756, 2019.

[QK20] Xin Qian and Diego Klabjan. The impact of the mini-batch size on the variance of gradients in stochastic gradient descent. *arXiv preprint arXiv:2004.13146*, 2020.

[SCTS19] Wouter Saelens, Robrecht Cannoodt, Helena Todorov, and Yvan Saeys. A comparison of single-cell trajectory inference methods. *Nature biotechnology*, 37(5):547–554, 2019.

[Sha20] Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

[SKYL17] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[STIM18] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander

Madry. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

[VEN+22] Abel Vertesy, Oliver L Eichmueller, Julia Naas, Maria Novatchkova, Christopher Esk, Meritxell Balmana, Sabrina Ladstaetter, Christoph Bock, Arndt von Haeseler, and Juergen A Knoblich. Cellular stress in brain organoids is limited to a distinct and bioinformatically removable subpopulation. *bioRxiv*, 2022.

[VRD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[WLZ+16] H Wang, Z Lei, X Zhang, B Zhou, and J Peng. Machine learning basics. *Deep learning*, pages 98–164, 2016.

[WY18] Florian Wagner and Itai Yanai. Moana: a robust and scalable cell type classification framework for single-cell rna-seq data. *BioRxiv*, page 456129, 2018.

[XLM+21] Chenling Xu, Romain Lopez, Edouard Mehlman, Jeffrey Regier, Michael I Jordan, and Nir Yosef. Probabilistic harmonization and annotation of single-cell transcriptomics data with deep generative models. *Molecular systems biology*, 17(1):e9620, 2021.