Identifying differential expression of neural circuitry using deep learning techniques

Jennifer Owens: jenowens Colton Swingle: cswingle

Abstract

Our project explores the differential expression of brain cells in *Drosophila Melanogaster*, a species of fruit fly. It is valuable for researchers to be able to identify a cell based on its gene expression patterns in order to better understand the relationship between gene expression levels and differential cell development in fruit flies. By developing a successful predictive model, we hope to not only provide scientists with a means to identify cells that can already be identified via other means using solely their gene expression matrices, but also to use gene expression matrices of yet-unidentified cells to understand their likely function in the *Drosophila* brain, as per their predicted circuit classification. To achieve this goal, we developed a fully connected deep neural network to predict a *Drosophila* brain cell's cell type from its gene expression patterns. Our model achieved an overall accuracy of 0.868 and an average F1-score of 0.791.

Introduction

The input to our algorithm is a gene expression matrix, which consists of the gene expression levels for approximately 17,000 fruit fly genes for a single cell. We then use a deep neural network to predict the cell's type from its gene expression matrix. There are 59 types of neuronal cells in our dataset.

Related Work

Single-cell RNA sequencing is a powerful genetic sequencing technique developed by Tang et al. in 2009.[1] The technique allows biologists to measure the gene expression levels at the single-cell level, providing much more detailed and granular gene expression analysis than previous techniques. Our dataset comes from a paper called "A Single-Cell Transcriptome Atlas of the Aging Drosophila Brain" that leverages this technique to develop a detailed dataset on tens of thousands of brain cells of gene expression matrices for over 20,000 brain cells.[2] There are several key papers that explore the relationship between gene expression and phenotype in cells. Chen et al.'s 2015 paper "Gene expression inference with deep learning" predicts expression of target genes based on known expressions of landmark genes,[3] and Guia et al.'s 2020 paper "DeepGx: Deep Learning Using Gene Expression for Cancer Classification" explores how to identify cell's cancer type based on their gene expression matrices.[4] While the methods used in these papers are not directly applicable to our dataset, they served as a useful reference and guide as we developed our architecture.

Dataset

Our dataset comes from the Stein Aerts Lab. They have performed single-cell RNA sequencing on the entire *Drosophila* brain, in order to create a *Drosophila* brain cell atlas.

Our dataset consists of 20,679 examples, each of which consists of a gene expression matrix for a single cell and the corresponding cell type. We used a 70-15-15 train/val/test split.

The data we downloaded from the lab was initially split into multiple files and was not very interpretable. The given features for gene expression were sparse and ranged from a majority of 0.0 up to 26,000. The targets were simply the name of the cell type that the gene expressions encoded for. In order for our model to learn, we first log transformed our features. We then used min-max standardization to normalize our data so it would range from 0 to 1. This normalization was fitted to our training data split and was reused to transform our validation and test data. We found that this technique greatly improved the model's ability to learn mapping from features to targets. In addition, we found every unique class of our targets and encoded them into integer classes to allow for one-hot encodings and faster load time.

In order to clean our initial dataset, we built a customizable script to sanitize it and produce model quality data. The script removed unknown cell types that the authors could not identify, and allowed us to remove targets that had support insufficient for a specified threshold. For this study, we disincluded classes with less than 50 examples because most of our classes had more than 500 examples and smaller classes provided little information for the model. The ability to clean the data and define with examples to keep allowed us greater flexibility in training the model and defining which cell types we were most confident in predicting.

Methods

Model Architecture

For our architecture, we chose a fully connected network, which we felt would be most appropriate for this task. While some studies on gene expression patterns rely on 1D convolutional networks, the features in our dataset (gene expressions) were sorted alphabetically and thus had no spatial relation to one another. The dimensionality of our input features was also large enough that we could not develop a human-interpretable understanding of the features from simple analysis. Thus, we decided the fully-connected model would be best for this task so that the model would be able to build increasingly sophisticated features from a relatively simple input that lacked a clear underlying structure. In addition, we learned very quickly that our model performed much better with usage of batch normalization--thus we included it in every layer except for the output. Dropout and L2 normalization was used to regularize the model but showed little effect in our final testing. Overall, our fully connected architecture performed above our expectations. With continued testing, we could try to group similar features together (either based on machine learning analyses or by mapping each gene's name to its location on the genome) in order to apply a convolutional layer before our fully connected layers to allow for spatial recognition and a decreased number of parameters for the model to tune. However, with 17,000 features, it was not realistic to perform this type of preprocessing given the timeline of the project.

Loss function and Optimizer

We used a cross-entropy loss function (which applies a log softmax and then calculates cross-entropy) and an Adam optimizer with L2 regularization.

Hyperparameters

We performed a large hyperparameter search in order to find good hyperparameters for testing. The search was conducted over the model learning rate, weight decay lambda value, number of hidden layers, number of hidden units, mini-batch size, and dropout rate. For these hyperparameters, we defined values that we could possibly use to tune, and randomly sampled from that distribution to train for 15 epochs. We optimized for the average validation accuracy score to find our best hyperparameters. Given our testing, we used our optimal hyperparameters to perform final testing on our test datasets.

Experiments/Results/Discussion

Our best hyperparameters were as follows:

Number of hidden layers: 2
Number of hidden units: 8192
Learning rate: 1e-6
Weight decay: 1e-3
Dropout: 0.2

We chose the hyperparameters that led to the best performance on the validation set, as measured by overall accuracy. The test set metrics are as described in the table below. "Average" for these metrics indicates the unweighted average across all classes for the given metric.

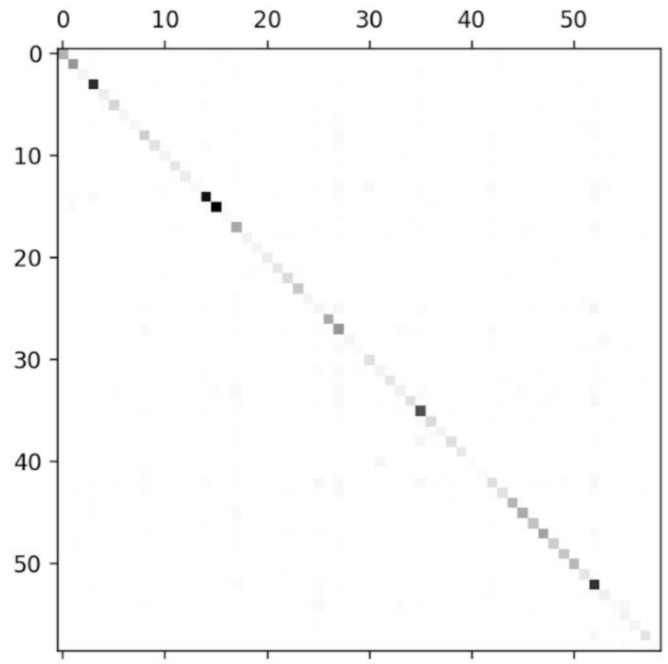
Test Metrics on Baseline and Final Model

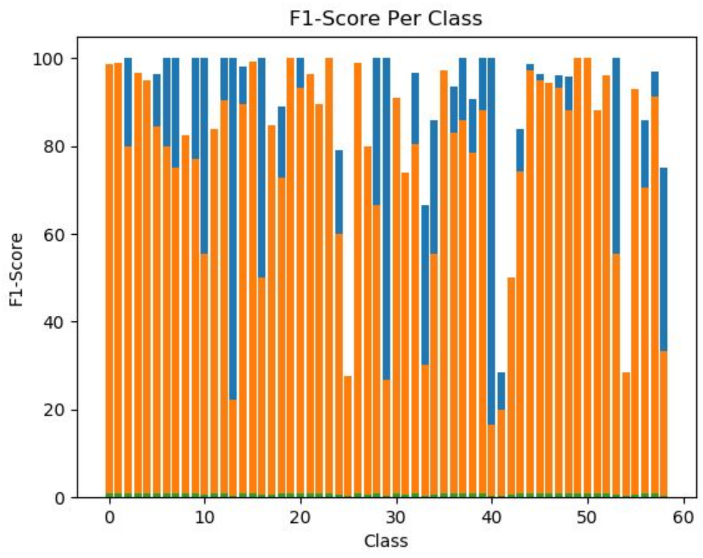
	Overall Accuracy	**Average Accuracy**	**Average Precision**	**Average Recall**	**Average F1 Score**	**Average Area under ROC**	**Average Area under PRC**
Baseline	0.867	0.797	0.828	0.797	0.804	0.897	0.695
Final	0.868	0.763	0.868	0.763	0.791	0.881	0.684

*Note: The identical values for overall accuracy and average precision, and for average accuracy and average recall, appear to be coincidental. None of our other runs had this outcome.

Although the improvement from our baseline (original) model was not very large, we still did see more consistent results across classes in the final model. There were many models trained during hyperparameter tuning with very poor performance--suggesting that the small improvement from the baseline model is due to good guesses for original hyperparameters rather than underlying issues with the model.We also visualized our results using a confusion matrix and a chart of per-class F1 scores, shown below:

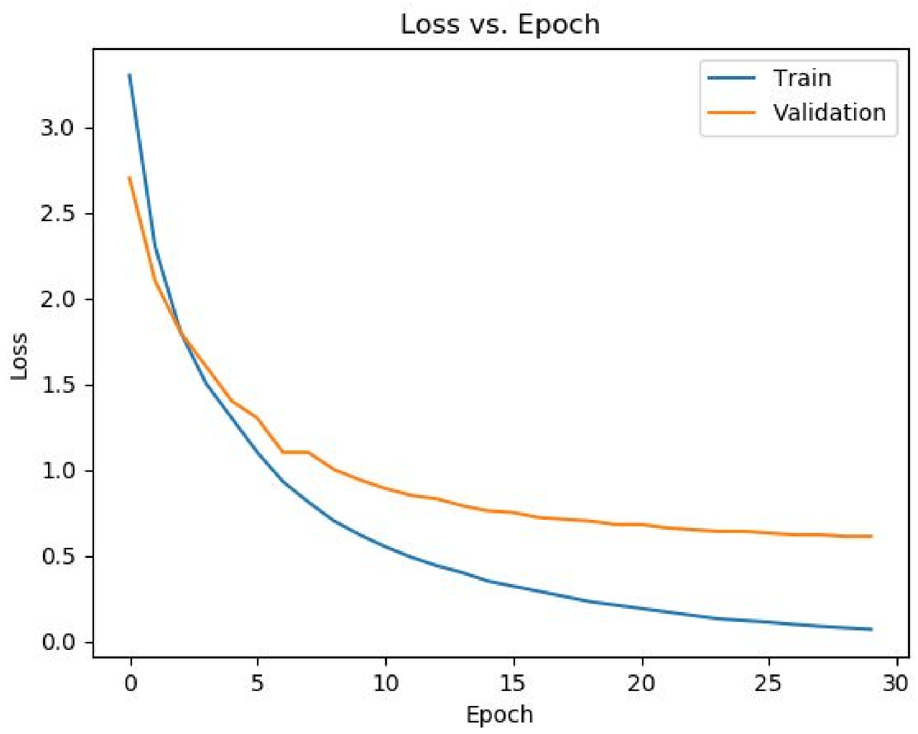
Confusion Matrix





The model performed extremely well on almost every class, as shown by the strong diagonal line in the confusion matrix and in the height of the bars in the chart depicting per-class F1 scores above. Nearly all of the classes in the test set achieved an F1 score of at least 0.7. More specifically, the minimum, first quartile, median, third quartile, and maximum F1 scores among the classes were 0.235, 0.714, 0.871, 0.942, and 1.0. The distribution for per-class accuracy was similar: 0.167, 0.686, 0.844, 0.947, and 1.0. We did not find a clear pattern for why a handful of classes got an F1 score of 0.2-0.3. All three of these low-performing classes had fewer examples than the typical class, so this is likely due at least in part to class imbalance,

however, many classes that were similarly small had very high F1 scores. More investigation into the nature of these cells from a scientific perspective may provide insight as to why these cells were more difficult to identify.
The loss curves for the model trained using the best hyperparameters are below.



As shown by the loss curves, the model trained quite smoothly for the over the thirty training epochs. While validation loss is higher than the training loss, it is still decreasing at the end of training, suggesting that the model is not overfitting.

Conclusions/Future Work

While our model performed much better than expected, there is still room for improvement. Future experiments could encode spatial information in the gene expression matrix by putting adjacent genes next to each other in the matrix, allowing for the use of 1D convolutions, or utilize data augmentation to reduce class imbalance. Future experiments could also explore model interpretability to make the results more generalizable to future scientific research.

Contributions
Jen did data exploration and initial preprocessing, incorporating metrics into model, analysis, and final report.
Colton did model architecture implementation, hyperparameter tuning, and final video slides.
References

1. https://www-nature-com.stanford.idm.oclc.org/articles/nmeth.1315
2. https://www-sciencedirect-com.stanford.idm.oclc.org/science/article/pii/S0092867418307207?via%3Dihub

3. https://www.biorxiv.org/content/10.1101/034421v1.full.pdf
4. https://ieeexplore-ieee-org.stanford.idm.oclc.org/abstract/document/9073128