Predicting Synaptic Activity Based on Axonal-Dendritic Proximity: A Machine Learning Model

Michael Popa
Department of Electrical and Computer Engineering
Rice University
Houston, TX, USA
map21@rice.edu

***Abstract*—Understanding when two neurons will synapse remains a significant challenge in the field of neuroscience. One major factor in predicting synapse formation is the proximity between the axon of a pre-synaptic neuron and the dendrite of a post-synaptic neuron. This is commonly referred to as the axonal-dendritic proximity, or the ADP. To explore this, data derived from neuronal activity of the visual cortex in mice was used to develop a machine learning model that would predict synapses based on an ADP. The development process involved preprocessing, feature engineering, hyperparameter tuning, and cross-validation. We demonstrate model stacking to be the best predictive method using linear Support Vector Machines, Adaptive Boosting, Gradient Boosting, K-Nearest Neighbors, and Random Forest as base learners, and a linear SVM as a meta learner. Through this, we find the most important feature in predicting a synapse is the depth of the dendritic ADP, in addition to the pre-synaptic neuron's receptive field vertical location. We hope this research can put forth a unique perspective to answering a long-standing difficult problem within the field.**

Keywords—axonal-dendritic proximity, machine learning

I. Introduction

The brain is considered to be the most complex organ in the human body. There are approximately 86 billion neurons that work together to control our thoughts, emotions, experiences, memories, and much more. These aspects of life are in large part the result of neurons forming connections with one another; signals that transmit from one neuron to another. These connections between neurons are what we now know as synapses. Understanding when two neurons will synapse is a major open question within neuroscience research.

To better understand synaptic activity, it's important to first understand the characteristics of a neuron. There are three basic parts of a neuron: (1) the cell body, also known as the soma, (2) the axon, and (3) dendrites. The soma is the most central part of a neuron, containing the nucleus which houses the cell's DNA and controls the cell's activities. The axon is a long thin fiber that acts as a cable, being responsible for the transmission of information. Specifically, the signal is an electrical impulse, which the axon projects away from the soma to the end of the neuron. At the end of the neuron (the axon terminal), these electrical signals can be converted into chemical signals (neurotransmitters), which will lead to the activation of other neurons. The dendrites are branch-like structures that protrude from the cell body and are responsible for the input of the neuron – they receive signals from other neurons and will send this information to the cell body. Neurons generally have many dendrites, which allow them to receive signals from multiple other neurons simultaneously. It's also important to note that the sizes of dendrites can vary, which can influence whether a signal will be received from another neuron. The neuron that sends the signal is known as the pre-synaptic neuron and the one receiving the signal is known as the post-synaptic neuron, and the specific site of communication of these two neurons is between the axon and the dendrite (at what is known as the synaptic gap, or synaptic cleft). These synapses can vary in terms of their size, as measured by the volume of the synaptic cleft, and a pair of neurons can have more than one synapse between them.

Understanding when two neurons will synapse depends on a number of factors. In this study, the focus was mainly on the proximity (physical distance) between the axon of a pre-synaptic neuron, and the dendrite of a post-synaptic neuron. This is what is known as the axonal-dendritic proximity, or ADP. This process is fundamental in neuroscience, as every synapse is the result of an ADP, but not every ADP results in a synapse. The question then becomes, what exactly causes an ADP to transform into a synapse?

To explore this question, an approach using machine learning will be used to predict when the formation of a synapse occurs from an ADP. The data that we will be using comes from neural activity in the visual cortex of mice while they are watching a movie. The neurons recorded are a specific type called cortical pyramidal neurons. In these neurons, the dendrites can be subdivided into three specific types: (1) apical, (2) basal, and (3) oblique. Apical dendrites typically originate from the top of the soma and extend upwards towards the cortical surface. They can have their branches divided into shaft and tuft, where the tuft branches upwards from the top of the shaft. Basal dendrites originate from the soma and span out laterally. Oblique dendrites originate from the apical shaft and span out laterally also.

The goal of this paper is to both accurately predict synapse formation through a machine learning algorithm as well as gain key insights into the underlying biological processes that lead to synaptic connectivity. Through this research, the hope is that a new solution will be put forth to an open problem within the field of neuroscience. The implications of our model's results may enhance our understanding of the brain and improve the

development of treatments for neurological disorders and diseases in future research.

II. Methods

A. MICrONS Dataset

The dataset that we used originated from the MICrONS collaboration and was provided by the Tolias Lab at Baylor College of Medicine. The data was created by two-photon excitation microscopy imaging of neuronal activity in the visual cortex of mice while they were watching a movie. The same chunk of brain tissue was extracted and imaged at high resolution with electron microscopy (EM). Every neuron was reconstructed in 3D, and synapses were automatically identified. This dataset contained both functional (neuronal activity) and morphological (neuron shape and structure) data and can be visually explored on the MICrONS Explorer website [1].

Two general types of information were included in the dataset: per-neuron features (for both pre- and post-synaptic neurons) and per-ADP features. For our per-neuron features, we had the three-dimensional coordinates of a neuron's nucleus, the brain area it belonged to, its reliability to repeated stimulus presentation, and the predictive performance of a deep learning model on withheld test trials. In addition, there were 512-dimensional feature weight vectors from the readout layer of the deep learning model, coordinates of the location, and 32-dimensional morphological embeddings that sought to capture the overall shape and structure of each neuron (the essential characteristics of a neuron's morphology). For our per-ADP features, we had the coordinates of axonal and dendritic ADPs, the distance between axonal and dendritic ADPs, the path length from the specific ADP coordinate to the pre- or post-neuron's soma, the compartment(s) where the ADP resided on the post-synaptic neuron, and whether or not at least one synapse was present at the ADP.

B. Implementation

The entire process was done using Python version 3.12.0 in Jupyter Notebook. In order to streamline the machine learning workflow, we used a pipeline that simplifies the ability to modify and compare models. This was done through the scikit-learn pipeline. The stages of our pipeline varied from model to model but generally included data preprocessing, feature selection, model training, and performance evaluation. We also reported the confusion matrix after each model to understand how many predictions were correct and the kinds of errors our model made.

C. Preprocessing

Our given data had 34 features and 185,832 observations. The dataset was unbalanced, as out of 185,832 potential synapses, only 1,366 synapses were present, and they all came from only 77 pre-synaptic neurons and 2,663 post-synaptic neurons.

To deal with the large number of data and the unbalanced nature of the set, we employed resampling techniques. Specifically, undersampling was used to reduce the number of instances in the majority class (in our case, the number of non-synapses) to make it comparable to the minority class (number of successful synapses) through the random removal of these majority-class instances. Undersampling was chosen as opposed to oversampling (SMOTE and ADASYN) due to computational efficiency. However, as potentially valuable data would be lost from the majority class, oversampling was seldom employed to analyze any possible changes in performance. Additionally, as feature engineering was performed throughout the project, the dimensionality of our training and test data would increase, therefore requiring the utilization of principal component analysis (PCA) for dimensionality reduction.

In order to stabilize model performance, as well as improve the interpretability and reproducibility of our approach, the data was normalized through the use of StandardScaler(), which transforms the dataset to have a mean of zero and a standard deviation of one. This method is less affected by outliers, as it does not constrain the range of values, allowing outliers to have an influence appropriate to their value. The values of many of our features also tended to follow a Gaussian distribution, which is another reason why StandardScaler() was used.

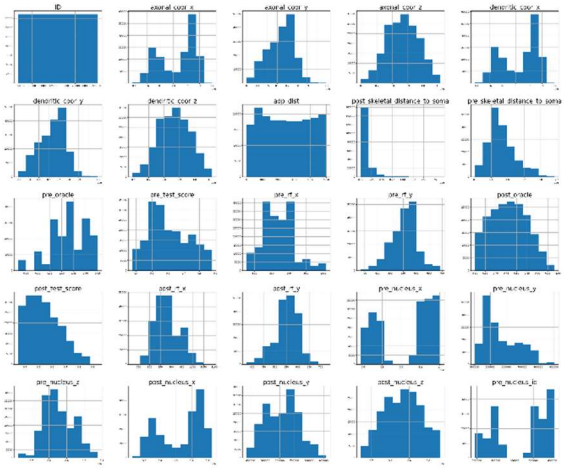

Fig. 1. Histograms of the values from each feature

Not all neurons had morphological embeddings, meaning that we had missing values in our dataset. To deal with this, we employed imputation methods. The main imputation method used was SimpleImputer() for mean/median/mode imputation. This method works well with both numerical and categorical data and is a fairly simple standardization method compatible with the scikit-learn pipeline.

D. Feature Engineering

Given our research goal, engineering new features was crucial for capturing the most important aspects of our data, leading to enhanced model performance. First, because we had categorical data values, we performed one-hot encoding to get numeric values. This was done to get the specific brain areas (primary, anterolateral, and rostrolateral visual areas) and to get the various compartments of a neuron. From there, several new features were engineered, which we split into the following categories: (1) distance features, (2) similarity features, (3) interaction features, (4) difference features, and (5) number count features.

TABLE I.

Description of each engineered feature category

Engineered Feature Category	***Description***
(1) Distance	Physical and functional distances between various aspects of the neurons, including their tuning functions and receptive field locations
(2) Similarity	The cosine similarities between various tuning patterns and receptive field locations
(3) Interaction	Reliability of neural activation changes with different distance metrics, brain areas, and receptive field location
(4) Difference	Different in predictive performance of the deep learning predictive model between pre- and post-synaptic neurons
(5) Number Count	The total number of synapses, total number of compartments, and the proportion of synapses in each compartment

Distance features can be crucial since the time it takes for a signal to travel will depend on the distance it must travel. If that distance is too high, the signal may degrade, and a synapse will fail to occur. Similarity features can give us a degree of likeness between different pre- and post-synaptic neurons based on their location and firing patterns. A higher level of similarity may or may not lead to synapse formation. Interaction features can provide insights into how the interaction between different aspects of neurons guides the formation of a synapse. The difference in test scores in the deep learning model could influence whether or not a synapse will form. The number count features could tell us which compartments will be more likely to lead to a synapse.[1]

E. Modeling

The machine learning task is to predict whether an ADP transforms into a synapse, so this means our problem is a binary classification problem. There was a focus on four classification modeling approaches: (1) Support Vector Machines, (2) Random Forests, (3) Model Stacking, and (4) Deep Learning.

Support Vector Machines (SVMs) identify the best hyperplane that separates the classes by maximizing the distance between the nearest data point(s) of both classes and the hyperplane. Random Forests (RF) are ensemble learning methods, meaning they use multiple learning algorithms to obtain better performance. They work by constructing a multitude of decision trees followed by recursively splitting the data into subsets based on the most significant feature at each node of the tree. Model stacking combines multiple machine learning models and uses a final model that combines the predictions of the other models. This can improve predictive performance by leveraging the strengths of each individual model. The focus in this paper was more on ensemble methods, such as Gradient Boosting, Adaptive Boosting, and Random Forests. Lastly, deep learning involves artificial neural networks with at least one hidden layer between the input and output layers that are capable of learning complex patterns in large datasets. Our focus was on feedforward neural networks with the Adam optimizer and cross-entropy loss.

Hyperparameter tuning was done through both GridSearchCV and RandomizedSearchCV (both with CV = 5). GridSearchCV performs an exhaustive search over a specified set of parameter values for an estimator. The parameters were optimized through a cross-validated grid search over a parameter grid. RandomizedSearchCV randomly samples a set of hyperparameters and calculates the score, returning the best set of hyperparameters that give the best score as an output. RandomizedSearchCV is more computationally efficient, as it does not need to try all parameter combinations but rather a randomly selected subset of them. Therefore, for computational efficiency, this project performed hyperparameter tuning using RandomizedSearchCV more than GridSearchCV. In addition to these two methods, there was a high degree of manual hyperparameter tuning, which involved a more hands-on approach and would lead to a better understanding of the model and its hyperparameters than the two automated methods. As for model stacking, the models were selected by defining two sets of random base models with the same meta learner and evaluate performance through cross-validation.

The model was trained on 80% of the data, while the remaining 20% was used to test the model (stabilizing the random state parameter ensured that our splits were reproducible). These training and test sets were internal, so once the model had been trained and tested using these sets, it was then tested on two new sets of unseen data to ensure it was not overfitting. Of these two new sets, the first was used to evaluate performance on a regular basis (public set), and the second set was our final test to check generalizability (private set). Additionally, to prevent overfitting, we used 5-fold cross-validation, which involved dividing the data into 5 subsets of equal size, using 4 of those 5 to train the model, then test on the last subset. This process was repeated a total of five times and a different subset was used as the test set each time. Finally, the cross-validation scores for each fold were saved then averaged, giving an overall measure of the model's performance across all folds. This way, we reduced the variance associated with a single trial and had a more robust measure of performance.

It's important to note that our performance evaluation metric was the Balanced Accuracy metric, which averages the sensitivity and specificity:

$$\text{Sensitivity} = \text{True Positive Rate} = \text{True Positive} / (\text{True Positive} + \text{False Negative})$$

$$\text{Specificity} = \text{True Negative Rate} = \text{True Negative} / (\text{True Negative} + \text{False Positive})$$

$$\text{Balanced Accuracy} = (\text{Sensitivity} + \text{Specificity})/2$$

The reason we focused on this metric is due to the heavily unbalanced nature of our dataset, as the number of ADPs that do not result in a synapse far exceeds those that do. This was in addition to our baseline model accuracy metric.

[1] For a detailed view for each engineered feature, please refer to the Jupyter Notebook file available at https://github.com/01MPopa/Neuron-Synapse-Prediction

Further, we employed the Receiver Operating Characteristic (ROC) curve to illustrate the performance of a binary classifier as its discrimination threshold was varied. It provides a visualization of the true positive rate (TPR) plotted against the false positive rate (FPR) at different binary classification thresholds. To interpret the plot, the closer the curve is to the top-left corner, the better the model's performance is. A perfect classifier would have a curve that passed through the top-left corner (100% sensitivity – no false negatives, and 100% specificity – no false positives). A classifier with poor performance, with no better accuracy than randomly guessing, would have a diagonal curve from the bottom left to the top right.

Lastly, to understand which features contributed to the model's success (or failure), we looked at feature importance scores. To find these, we performed permutation importance, which calculates the mean importance of each feature by repeatedly shuffling the values of each feature and measuring how much the model's accuracy decreases. We provided a bar plot of the results, showing each feature used in the model and its importance to model accuracy.

III. Results

To understand the results best, we will divide this section of the paper into two subsections, one focusing on predictive performance over time using various models and parameters, and another that will deal with interpretability and any promising findings of our study.

A. Prediction

To start, we used the reference model to better understand the binary classification problem at hand. This model is a simple logistic regression model that is only tested on two features: the distance between axonal ADP coordinates and dendritic ADP coordinates, and the similarity between feature weight vectors of the pre- and post-synaptic neurons. When not accounting for the imbalanced nature of the data, the model performs exceptionally well (~99.27%). However, the confusion matrix shows the model predicts that the neurons are never connected, making its balanced accuracy 0.5 – no better than random guessing. When implementing oversampling, we see the performance of the model drop to ~64.78%, but the balanced accuracy jumps to 72.19%, confirming the importance of resampling techniques. From there, the focus turned on the four aforementioned modeling classification techniques:

1. Support Vector Machines (SVMs)

Although oversampling did well, we opted to use undersampling from here on out due to its computational efficiency, which proved to be exceptionally useful during hyperparameter tuning. To start, a simple kernel SVM was chosen with its default parameters (RBF kernel) and trained on the same two features as the logistic regression. The baseline model accuracy metric would drop significantly, down to ~61.29%, and there would be a slight increase in the balanced accuracy metric, ~72.63%. However, a simple manual change from RBF to a linear kernel caused the accuracy metric to jump to ~65.64% but a drop in balanced accuracy down to ~71.89%. After engineering two new simple features, both calculating the Euclidean distance between (1) the coordinates of the axon and dendrites, (2) the nucleus of the pre- and post-synaptic neurons, we see an increase in performance in model and balanced accuracy, reaching ~68.17% and ~73.89% respectively. When testing on the public set, we got a score of ~70.12%.

Hyperparameter tuning was employed using GridSearchCV and a small level of manual experimentation. There was a focus on two parameters, gamma and C. The best gamma value turned out to be the default, 'scale,' while the optimal C value was 10, as opposed to the default of 1. This change would yield an increase of ~0.2% in performance. Note: RandomizedSearchCV was used as well, but we opted for GridSearchCV's parameters in fear of overfitting (RandomizedSearch returned a relatively high C value of 27).

Finally, after implementing a variety of new engineered features, many of which came from the one-hot encoded brain area features, we get our final best SVM model accuracy of ~69.16% and balanced accuracy of ~73.476%. Testing this on the public set, we reached a score of ~70.877%.

2. Random Forests

As the number of engineered features would exponentially increase, the worry of overfitting due to increased model complexity caused a transition towards ensemble methods, as they tend to be quite robust to overfitting.

Starting with a simple Random Forest algorithm, the only parameter changed was 'class_weight' to balanced due to our dataset imbalance. We used a small subset of features to test the model employing oversampling (SMOTE) for experimentation purposes. This would lead to failure, as the model did not manage to deal with the unbalanced nature of the dataset, making it practically useless. Combining oversampling with undersampling yielded the same results, poor balanced accuracy but incredible model accuracy.

All engineered features, with the exception of some, were used in the Random Forest algorithm. Doing so caused a drastic increase in performance, both in our internal sets as well as the public set. Furthermore, the hyperparameters best tuned to the model turned out to be the default ones. Our model accuracy reached ~73.06% with a balanced score of ~77.64%. On the public set, we reached an accuracy of ~74.295%, and a private set score of ~77.9%.

As ensemble methods seemed to perform quite well, there was now a drive to use multiple methods and combine the prediction performance, which led to model stacking.

3. Model Stacking

For optimal model stacking performance, we opted to train a diverse set of models, but with a major focus on ensemble methods for our base learners. Using diverse models can encourage model robustness and generalizability. From there, the predictions from each would be trained on a meta learner.

In our first attempt, we implemented six base models: linear SVM, Adaptive Boosting, Gradient Boosting, Logistic Regression, K-Nearest Neighbors, and Random Forests, with a meta learner of Logistic Regression. All parameters were default with exception to linear SVM with C = 10. Doing so resulted in

a model accuracy of ~66.9% but a balanced accuracy of ~73.44%.

Following this, a number of adjustments were made. First, as logistic regression did not tend to perform well throughout this project, it was replaced with K-Nearest Neighbors in the base model, and its place in the final estimator was taken by linear SVM w/ C=10. Second, various hyperparameters were tuned, resulting in KNN having K=1 neighbors and p=1 (Manhattan distance), RFs being limited to 10 trees, and XGBoost having a max tree depth of 7, a learning rate of 0.2, and a subsample of 0.5. Third, as the number of engineered features grew, we employed feature selection using Random Forests w/ 100 trees, as well as PCA, with the number of components being varied but worked best around 7 components. Additionally, StandardScaler() was replaced with MinMaxScaler(), as many new engineered features did not follow a Gaussian distribution, and SelectFromModel was implemented using Random Forests with 100 trees. These adjustments would result in a model accuracy of ~70.43% and a balanced accuracy of ~75.03%. On the public set, it would perform significantly worse, returning a score of ~67.23%.

However, the performance of our model would find a notable improvement in performance by: (1) the removal of PCA, (2) reverting MinMaxScaler() back to StandardScaler(), and (3) removing SelectFromModel(). From our internal sets, we now get a model accuracy of ~73.164% with a balanced accuracy of ~78.06%. Performing 5-fold cross-validation, we receive a mean CV score of ~77.26%. On our public set, we get a score of ~75.23%, but on our private set, we get a surprising score of ~79.61%. This would be our best performing model and resulted in the following ROC curve:

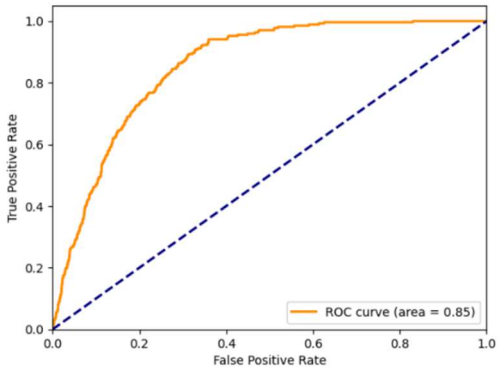

Fig. 2. Receiver Operating Characteristic for model stacking

4. Deep Learning

The last modeling technique focused on feedforward neural networks (FFNN). However, they proved to be difficult to tune and could often be misleading in the performance output. Because of this, time was spent focusing more on the aforementioned modeling techniques, and deep learning was used more for experimentation purposes.

The implementation was done using Tensorflow and Keras in Python. Our best performing FFNN involved three layers, with the input layer having 512 neurons, the hidden layer having only 16 neurons (with ReLU activation) and our output layer having a sigmoid activation. The internal set tested model accuracy would reach ~75.15% with a balanced accuracy of ~75.21%. However, the only way this was possible was through the use of 300 epochs, which led to overfitting. Our public set score was lower, being around ~73.83%, and our private set score would end up being ~76.13%.

B. Interpretability

To interpret the results, we will concentrate on the results of our best performing model: model stacking with linear SVM, KNN, RF, AdaBoost, and XGBoost with a linear SVM meta learner.

It's important to note that not all engineered features were used. Although we would have a total of 109 features in our DataFrame, only 88 proved to be useful. One group of features that were incredibly difficult to work with were the morphological embeddings, specifically those for the pre-synaptic neurons. Applying imputation techniques to those neurons was harder than expected. When the techniques would yield results, the engineered features using morphological embeddings would not improve model performance. One example of this would be the Euclidean distance between pre- and post-synaptic neuron morphological embeddings. The idea was that neurons with a similar shape/structure or morphology may be more likely to synapse than neurons with less similar structures. This feature would cause a drop in all evaluation metrics. The same would go for a feature that computes the cosine similarity between pre- and post-synaptic neuron morphological embeddings. Therefore, no features trained on our model would use morphological embeddings. Moreover, no ratio features were used for the same reasons. An example of a ratio feature omitted would be the ratio of each compartment type to the total number of compartments for each neuron. A number of other features were experimented with as well but failed to perform in our model.

As for important features, we can analyze our resulting permutation importance bar plot. This showed the most important feature to be the y value of the dendritic ADP coordinate, while the least important feature to be the x-coordinate of the readout location from the deep learning model of the pre-synaptic neuron. In other words, the x-coordinate of the region in the visual space that stimulates a pre-synaptic neuron. The finding that proves to be quite intriguing is that the second most important feature was the y-coordinate of the least important feature. Specifically, the y-coordinate of the region in the visual space that stimulates a pre-synaptic neuron.

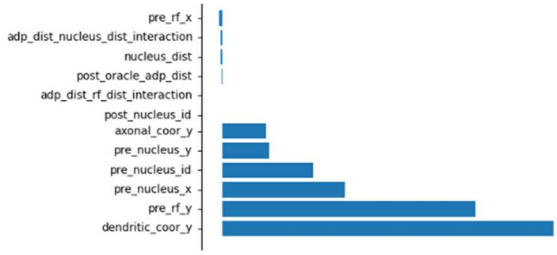

Fig. 3. The six least and six most important features from least (top) to most important (bottom)[2]

[2]See all complete figures at: https://github.com/01MPopa/Neuron-Synapse-Prediction

Our chosen parameters provide useful insights as well. For our SVM, a larger C provided better results, meaning our model emphasized classifying all points correctly through a smaller-margin hyperplane. The classes may be well-separated, with few instances of wrong class labels. The same reason could explain why KNN w/ K = 1 worked best as well. The Manhattan distance also worked better, implying there isn't a clear pattern of correlation between the features. This is confirmed using the correlation matrix[2]. Another surprising parameter was the use of only 10 trees in RF, implying that more estimators would cause overfitting to the training set. Conversely with XGBoost, having a slight increase in the maximum depth of a tree improved performance but did not overfit. This may be due to our subsample parameter being set to 0.5, which randomly samples half of the training data prior to growing trees, as well as our learning rate being lower to improve generalization. These parameters could explain why the model was able to generalize so well to the unseen data in the public and private set.

IV. Discussion

The use of a stacked machine learning model is a fairly novel approach to predicting synapse formation yet has the potential to advance our understanding of neuronal connectivity. Choosing a set of diverse models can introduce unique perspectives to the classification problem and may capture a wider range of relationships in the data that would not be found otherwise.

Given our results, it appears that the most crucial factor in determining whether an ADP will turn into a synapse is the depth of the post-synaptic neuron (the y-axis of the dendritic ADP). The intuition for this could lie in that certain tissue depths influence the likelihood of synaptic formation. Specifically, the y-coordinate of the dendritic ADP may indicate which layer of the brain the dendrite is in. Each layer of tissue has different functions (depending on the region), and neurons with similar functions would be more likely to synapse [2], [3]. This could be further explained by the second most important feature, which is the y-coordinate of the pre-synaptic neuron's receptive field. So, pre-synaptic neurons in certain vertical regions/areas of the visual field would be more likely to synapse with post-synaptic neurons at specific layers/depths of the brain. Additionally, the horizontal positioning of the synapses in the receptive field may be distributed uniformly. In other words, the horizontal position does not significantly affect the likelihood of synapse formation. We can infer this because the x-coordinate of the pre-synaptic neuron's receptive field is the least important feature.

Another crucial factor in determining ADP to synapse formation is the x- and y-axis of the nucleus of the pre-synaptic neuron (but not the z-axis). Many of our engineered features used the x- and y-axes of the pre-synaptic neuron's nucleus but not the z-axis, since multiple features did not have a z-axis (such as the receptive field). This would confirm the receptive field's importance, as it is primarily defined along the x and y axes. Additionally, it may be true that not only is the proximity between the axonal and dendritic ADP important for determining a synapse, but also the distance from the pre-synaptic neuron's nucleus to the post-synaptic dendrite, independent of the axon's length.

Lastly, it's important to note that the pre-synaptic neuron's nucleus x-coordinate is more important than its y-coordinate. A potential explanation comes from the flow or spread of neuronal connectivity. Concretely, the spread of synaptic activity may be more extensive in a horizontal direction. This could align with the idea of a cortical column, where the neurons that extend vertically within a column exhibit similar neuronal activity and are more likely to synapse [5], [6]. The horizontal position would assist in differentiating between these columns and predicting the spread of activation.

In future research, an investigation into the specific neurotransmitters housed in each neuron could drastically change outcomes. The type of neurotransmitter (excitatory or inhibitory), the concentration of these neurotransmitters, and the receptors on the dendrites all have a noteworthy impact on synaptic activity. Furthermore, understanding why synaptic activity is occurring (encoding, retrieval, plasticity, etc.) may lead to different ADP coordinates being more important than others. Our dataset involved encoding visual stimuli, but an analysis of data involving *retrieving* encoded stimuli could reveal a different set of important coordinates. Additionally, it would be beneficial to experiment more with the morphological embeddings to understand which aspects of neuronal structure are most significant for synapse formation. This can be done through clustering or dimensionality reduction methods (like PCA or t-SNE).

Acknowledgment

We thank the Andreas Tolias Lab (Baylor College of Medicine) for their invaluable work in collecting and processing the data, a rich dataset for our research. Additionally, the guidance and support of Dr. Genevera Allen (Rice University) was a source of inspiration throughout the project that significantly enhanced the quality of our work.

References

[1] MICrONS Consortium, J Alexander Bae, Mahaly Baptiste, Caitlyn A Bishop, Agnes L Bodor, Derrick Brittain, JoAnn Buchanan, Daniel J Bumbarger, Manuel A Castro, Brendan Celii, et al. Functional connectomics spanning multiple areas of mouse visual cortex. BioRxiv, pages 2021–07, 2021.

[2] T. C. Südhof, "The Cell Biology of Synapse Formation," *Journal of Cell Biology*, vol. 220, no. 7, 2021. doi:10.1083/jcb.202103052

[3] [1] T. C. Südhof, "Towards an understanding of synapse formation," *Neuron*, vol. 100, no. 2, pp. 276–293, 2018. doi:10.1016/j.neuron.2018.09.040

[4] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, 2016. doi:10.3389/fncir.2016.00023

[5] J. C. Horton and D. L. Adams, "The cortical column: A structure without a function," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 360, no. 1456, pp. 837–862, 2005. doi:10.1098/rstb.2005.1623

[6] T. Schwalger, M. Deger, and W. Gerstner, "Towards a theory of cortical columns: From spiking neurons to interacting neural populations of finite size," *PLOS Computational Biology*, vol. 13, no. 4, 2017. doi:10.1371/journal.pcbi.1005507