RETENTION BASED AUTOREGRESSIVE MODELS FOR MODELLING NEURAL DYNAMICS

BY

ABHINAV MURALEEDHARAN

A thesis submitted in conformity with
the requirements for the degree of
Masters in Engineering
Graduate Department of Institute for Aerospace Studies
University of Toronto

ABSTRACT

Retention based autoregressive models for modelling neural dynamics

Abhinav Muraleedharan

Masters in Engineering

Graduate Department of Institute for Aerospace Studies

University of Toronto

2023

Autoregressive models based on the Transformer architecture have achieved state of the art performance in various machine learning domains, ranging from natural language processing to multimodal foundation models. Transformers process sequential data using the attention mechanism, without any recurrent or convolutional layers. While the performance of transformer based models is impressive, they have three main drawbacks. First, transformers operate on a context window of finite size, and hence it cannot process sequences larger than the size of its context window. Second, the computational cost of attention mechanism is quadratic $\mathcal{O}(N^2)$ in the length (N) of the sequence. Third, the output layer of transformer, which predicts probabilities of next token in the sequence, cannot scale well for datasets of huge token vocabulary. Because of these limitations, transformer based models are not efficient in modelling and decoding neural dynamics where due to high sampling rates, the sequence length can be extremely large and since the possible firing patterns grow exponentially with respect to number of neurons we record from, the size of token vocabulary is in trillions of discrete tokens. In this work, we

present 'Retention based Autoregressive Models' which overcomes the limitations of attention based Transformer architectures. Retention based models can process sequences of variable length, and is not bounded by a limited context window. Unlike Transformers, the complexity of Retention mechanism is linear $\mathcal{O}(N)$ in the length of the sequence. We apply Retention mechanisms along with convolutional architectures to build an autoregressive generative model of neural dynamics, and for decoding behavior from observed neural spiking data. For generative modelling task, Retention based models offer fast inference when compared to transformer based models. For generative modeling tasks, Retention-based models provide quicker inference compared to transformer-based models. Employing Retention-based models enabled us to generate neural spike trajectories for 4096 neurons across 5000 timesteps in just 3.5 minutes on a standard desktop PC. In the context of decoding neural spikes into behavior, Retention-based models achieve an impressive R2 score of 76.9. Overall, Retention-based autoregressive models present a significant advancement, offering enhanced efficiency and scalability for complex sequence processing tasks in neural dynamics and beyond.

To my Grandmother,

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor, Prof. Prasanth Nair, for his guidance and support throughout the completion of my MEng project. Prof. Nair's insightful feedback and constructive criticism played a pivotal role in shaping the direction of my research. I thoroughly enjoyed our discussions on various ideas, and his mentorship greatly enriched my academic experience.

I would also like to extend my appreciation to my co-supervisor, Prof. Taufik Valiante, for his motivating presence and the engaging discussions we had during the course of this research. Prof. Valiante's expertise and enthusiasm for the subject matter inspired me to tackle challenges with a fresh perspective.

Furthermore, I am grateful to both Prof. Nair and Prof. Valiante for not only guiding me in my academic pursuits but also providing valuable career advice that will undoubtedly influence my future endeavors.

This acknowledgment would be incomplete without expressing my deepest gratitude to my parents. Their unwavering support, encouragement, and belief in my abilities have been the cornerstone of my academic journey. Their sacrifices and love have fueled my determination, and I am profoundly thankful for their presence in my life.

Thank you all for your invaluable contributions and support.

PUBLICATIONS

This work has not been published yet.

CONTENTS

1

INTRODUCTION

Hard problems inspire the creation of novel algorithms. These novel algorithms then find application in various contexts, distant from the original application which it was designed for. Among the hard problems that we face, understanding the human brain stands out as particularly challenging. The human brain is an intricate network, where countless neurons interact in complex ways, leading to thoughts, actions, and behaviors. In order to understand how the brain works, we need methods that can efficiently model neural activity and the relationship between neural activity and behavior. In this thesis, we develop efficient methods for learning neural dynamics and decoding behavior from neural spiking data. Although methods in this thesis are developed specifically for neural data, we believe that our approach would find application in diverse sequence modelling tasks in language, finance and engineering.

Machine learning techniques have played a pivotal role in modeling brain dynamics, and modeling the correlation between neural dynamics and behaviour [HMW+18, POC+18, KTK+19, SW20]. In [POC+18], Pandarinath et al. introduced LFADS, a method to infer firing rates of neurons from observed neural spiking activity. LFADS is a sequential variational autoencoder that uses recurrent neural networks to model neural population dynamics. More recently, transformer-based models [VSP+17, GZ22], have been applied to learn neural dynamics. In [YP21], Pandarinath et al introduced Neural Data Transformer, a transformer-based model to learn neural dynamics. While LFADS and NDT (Neural Data Transformers) were focused on learning neural dynamics from single trial recordings, Azabou et.al recently introduced POYO [AAG+23], a transformer-based model to learn neural dynamics from multi-session neural recordings. POYO uses Perceiver IO [JBA+21] architecture to process neural activity and infer behaviour.

While these methods differ in the approach taken to model neural activity, they all assume that the neural spiking activity is generated by a poisson process, characterized by instantaneous firing rate of neurons. In neuroscience literature, this assumption is called as the independent spike hypothesis [H+00]. The machine learning problem then reduces to

inferring instantaneous firing rates of the neurons from observed neural spiking data. While certain experiments support the independent spike hypothesis for certain regions of the cortex, it is unclear if this hold true for all regions of the cortex [SK93]. While the independent spike hypothesis has provided a useful framework for understanding and modeling neural activity, it is becoming increasingly clear that more sophisticated models are needed to fully capture the rich dynamics of the brain. A more comprehensive understanding of neural dynamics may require moving beyond the independent spike hypothesis and considering the complex interactions and dependencies between neurons. This would require building machine learning models that can predict the trajectory of firing pattern of a collection of neurons over a time window. In principle, autoregressive models using parametrized Transformer architecture can be used to model stochastic dynamics of a collection of neurons. However, challenges arise when directly applying transformer based models, which, despite their success in language modeling, face scaling difficulties with neural spiking data.

Unlike text data, neural recording probes sample on the order of kHz, which result in extremely long sequences. Moreover, the exponential growth of possible firing patterns with the number of recorded neurons leads to token vocabularies in the trillions. This work introduces "Retention-based Autoregressive Models" designed to overcome the shortcomings of attention-based Transformer architectures. Retention-based models offer the flexibility to process sequences of variable length without being confined by a limited context window. Unlike Transformers, the computational complexity of the Retention mechanism is linear, addressing the quadratic costs associated with attention mechanism. The application of Retention mechanisms, coupled with convolutional architectures, forms the basis for constructing an autoregressive generative model of neural dynamics and decoding behavior from observed neural spiking data. Notably, Retention-based models outperform Transformers in terms of speed and efficiency during generative modeling tasks. The ability to generate neural spike trajectories for a considerable number of neurons across numerous timesteps within a short time frame underscores the practical advantages of Retention-based models. In decoding neural spikes into behavior, these models exhibit an impressive R2 score of 76.9, showcasing their efficacy.

In summary, Retention-based autoregressive models represent a significant advancement, offering enhanced efficiency and scalability for complex sequence processing tasks in neural dynamics and beyond. This research contributes to bridging the gap between the capabilities of existing mod-

els and the intricate demands of understanding and interpreting neural dynamics.

2

PROBLEM STATEMENT

Imagine we are recording data from D neurons distributed across different regions of the brain. Let $x(t_i) \in \mathbb{R}^D$ denote the observed neural activity at timestep t_i and let y_i denote the observed behaviour of the animal at timestep t_i. From the time series dataset $\mathcal{D} = \{(x_i, y_i, t_i)\}_{i=1}^N$ of neural recordings, our goal is to construct:

- A predictive model of underlying brain dynamics
- A probabilistic model to predict behaviour of the organism at time $t+1$ given brain recordings until timestep t.

The probability of observing a sequence of neural recordings and behavior can be expressed as:

$$p(\{x_1, y_1\}, \{x_2, y_2\}, \{x_3, y_3\}, ..) = \lim_{N \to \infty} \prod_{i=1}^{N} p(\{x_i, y_i\} | \{x_1, y_1\}, \{x_2, y_2\}, ..\{x_{i-1}, y_{i-1}\}) \tag{2.1}$$

In the context of neural recordings, it is convenient to assume that the neural recording data and behavior can be modelled with separate probability distributions of the form:

$$\prod_{i=1}^{N} p_d(\{x_i\} | \{x_1\}, \{x_2\}, ..\{x_{i-1}\}) \tag{2.2}$$

$$\prod_{i=1}^{N} p_b(\{y_i\} | \{x_1\}, \{x_2\}, ..\{x_{i-1}\}) \tag{2.3}$$

Specifically, we assume that that the observed neural spiking data at timestep t_i is not dependent on the behavior variables in the preceding timesteps. Probability distributions of this nature have been extensively investigated in the field of language modeling. In conventional autoregressive frameworks, the approximation of conditional distributions often involves the utilization of parameterized models constrained by a finite context limit [VSP+17]. While autoregressive models of this kind have been extremely successful in generating plausible language [RNS+18], they still struggle to capture long-range dependencies due to the finite context length limit[Hah20]. Furthermore, the complexity of training and

inference of transformer-based models is quadratic $\mathcal{O}(N^2)$, where N is the context length of the transformer model. In this work, we introduce a new class of autoregressive models wherein the per iteration training complexity and inference time complexity is linear with respect to context length. This enables us to process substantially longer sequences of neural data, capturing intricate long-range dependencies that are crucial for understanding brain dynamics and behavior. The proposed model, which we term 'Retention based Autoregressive Models', is specifically designed to address the computational limitations of traditional transformer models in the realm of neural data analysis. In the next chapter, we describe the theory behind Retention based autoregressive models.

3

METHODS

In this chapter, we describe the theory behind Retention based autoregressive models and methods for training retention based models.

3.1 MOTIVATION

To model conditional distributions defined in eq(2.2) and eq(2.3) exactly, we require a method that can process sequences of variable input length. In the realm of neural spiking data, which is frequently recorded at a sampling rate expressed in kHz, we require a method that can efficiently scale with the length of the sequence Furthermore, the patterns of neural activity increase exponentially with respect to the number of neurons, and hence the number of discrete tokens required to represent neural spike pattern at time step t can be prohibitively large. For instance, if we are recording spike signals from 100 neurons, in total there are 2^{100} possible firing patterns. Existing Transformer based models cannot be directly applied to model conditional distributions of these kind, without placing an assumption on the nature of probability distribution.

To address these challenges, we introduce "Retention", a mathematical operation to map a sequence of vectors $\{x_i\}_{i=1}^{N}$ to real valued vector ζ_i of same dimension. Retention is inspired from Score-life programming [Mur23], a novel method to solve sequential decision making problems. In Score-life programming [Mur23], Muraleedharan et.al applied the insight that the binary expansion of a real number can be used to represent a sequence of discrete variables. After constructing the mapping between a sequence of discrete variables and real numbers in a bounded interval, functions can be directly defined on the real numbers. By defining functions using this approach, we can model non-trivial relationships between elements of a sequence. In prior work [Mur23], has showed that such functions have unique properties, which can be exploited in developing efficient methods for solving determinisitic reinforcement learning problems. In our work, we extend this insight to vector valued variables, which are typically encountered in deep learning settings.

3.2 RETENTION

Mathematically, retention is defined as an exponentially weighted sum of a sequence of discrete vectors. If the vectors are drawn from a continuous space, then we perform thresholding operation to discretize the vectors. Specifically, given a sequence of vectors $\{x_i\}_{i=1}^{N}$, $x_i \in \mathbb{R}^d$, Retention variable $\zeta_k \in [0,1)^d$ as:

$$\zeta_k = \sum_{j=1}^{k} 2^{-(\log M)j} \sum_{i=0}^{M-1} \sigma(w_i \odot x_{k-j+1} + b_i) \tag{3.1}$$

Here, $w_i, b_i \in \mathbb{R}^d$ are trainable parameters for the thresholding operation defined in inner summation. Given a vector $x_i \in \mathbb{R}^d$ as input, the inner summation operation acts like a smoothened step function, essentially discretizing elements of the vector to discrete values in the set: $\{0, 1, 2, , , M-1\}$. The outer summation operation, with an exponentially decaying factor maps the sequence of discrete vectors to a continuous real valued vector ζ_k. If the input data is discrete, or binary as in the case of neural spike signals, then the thresholding operation can be omitted, and the retention variable can be defined as:

$$\zeta_i = \sum_{k=1}^{i-1} 2^{-k}(x_{i-k}) \tag{3.2}$$

Given a sequence of discrete vectors $\{x_i\}_{i=1}^{N}$, retention variable ζ_i stores the discrete vectors in the binary expansion of ζ_i. If the vectors $\{x_i\}_{i=1}^{N}$ are continuous, then we perform a thresholding operation first to discretize the vectors and perform discounted sum of these discretized vectors.

Retention can also be defined for sequence of matrices $\{\mathbf{X}_i\}_{i=1}^{N}$ as:

$$\mathbf{1}_k = \sum_{j=1}^{k} 2^{-(\log M)j} \sum_{i=0}^{M-1} \sigma(\mathbf{W}_i \odot \mathbf{X}_{k-j+1} + \mathbf{B}_i) \tag{3.3}$$

Modelling Conditional Distributions with Retention Variables

Now, we can approximate the conditional distribution defined in eq(3) using retention variables. Specifically, the product of conditional distributions can now be approximated as:

$$\prod_{i=1}^{N} p_d(\{x_i\}|\{x_1\}, \{x_2\}, ..\{x_{i-1}\}) \approx \prod_{i=1}^{N} p_r(\{x_i\}|\zeta_i) \tag{3.4}$$

In this case, sequence of vectors $\{x_j\}_{j=1}^{i}$ is encoded in the binary representation of Retention variable ζ_i. Note that in this approach, a sequence of arbitrary length can be encoded within binary representation of ζ_i.

Generative Models for neural spiking data

Now, we apply Retention for generative modelling of neural spike patterns. Let $x_i \in \mathbb{R}$ denote the recording data from d neurons at time step i. To learn the dynamics of the brain from neural recordings in an unsupervised manner, we maximize the following likelihood:

$$\mathcal{L}(X, \theta) = -\sum_i log(p_r(\{x_i\}|\zeta_i; \theta)) \tag{3.5}$$

Here, $X = \{x_1, x_2, ...x_M\}$, is the dataset of neural recordings.

Note that in this approach, the context window is not bounded, and the complexity of learning the parametrized model $p_r(\{x_i\}|\zeta_i; \theta)$ is independent of the length of the context window.

Neural Spike to behavior model

To learn the correlation between neural dynamics and behavior, we follow a similar approach and approximate the conditional distribution defined in eq(2.4) with:

$$\prod_{i=1}^{N} p_b(\{y_i\}|\{x_1\}, \{x_2\}, ..\{x_{i-1}\}) \approx \prod_{i=1}^{N} p_b(\{y_i\}|\zeta_i) \tag{3.6}$$

We define the loss function associated with this approach as the negative log-likelihood of the observed behavioral outcomes given the estimated neural activity states. Formally, the loss function $\mathcal{L}$ is expressed as:

$$\mathcal{L}(X, Y, \phi) = -\sum_i \log p_b(\{y_i\}|\zeta_i; \phi)$$

We further assume that the conditional distribution is of the form:

$$p_b(\{y_i\}|\zeta_i; \phi) = \mathcal{N}(f_b(\zeta_i; \phi), \sigma^2 I_d) \tag{3.7}$$

After this assumption the loss function takes the form of Mean Squared Error loss given by:

$$\mathcal{L}(X, \theta) = -\sum_i ||f_b(\zeta_i; \phi) - y_i||_2 \tag{3.8}$$

3.3 ARCHITECTURE

For predicting neural dynamics and inferring behavior given spike data, we employ a convolutional network based on the UNet architecture [RFB15]. The UNet model has proven to be highly effective in various image segmentation tasks and is well-suited for our objective of decoding neural activity.

Our architecture comprises multiple key components designed to handle the intricacies of spike data and capture the underlying patterns in neural dynamics. The UNet structure consists of an encoder and decoder network, facilitating the extraction and reconstruction of features at different abstraction levels. This enables the model to learn hierarchical representations of the spatio-temporal input data, enhancing its ability to discern complex relationships within the neural activity.

For the autoregressive model, we use a standard UNet model with retention variable ζ_k at input layer. In our implementation, we computed the retention variable ζ_k online, during training. Intuitively, given a sequence of black and white images that represent neural firing patterns at various time steps, the retention layer convert the sequence of black and white images to a single grayscale image, which is then fed into the UNet architecture. Each pixel in the image correspond to a neuron or unit from which the neural spiking data is collected.

In a typical UNet model employed for medical image segmentation tasks, the output layer predicts the masked image corresponding to the input image. In our case, the output layer predicts the probability of different neurons firing at the next timestep. Since the UNet model is a fully convolutional network, the model can be trained on diverse set of input datasets, consisting of variable number of input neurons. Hence, we can utilize the architecture on learning from a large dataset of neural recordings collected using various experimental setups.

Our Architecture consists of a UNet [RFB15] block to process input images and fully connected layers for predicting three-dimensional coordinates of finger-tip from the final layer of UNet block. We used pretrained UNet block from the autoregressive generative modelling task and fully

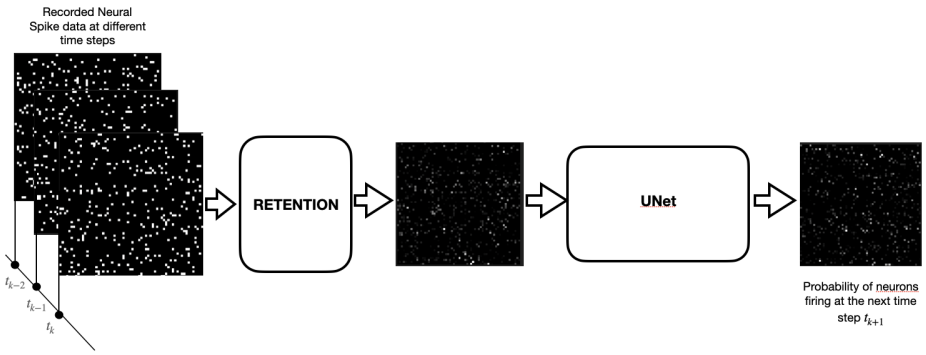


Figure 3.1: Architecture for Autoregressive Generative Model

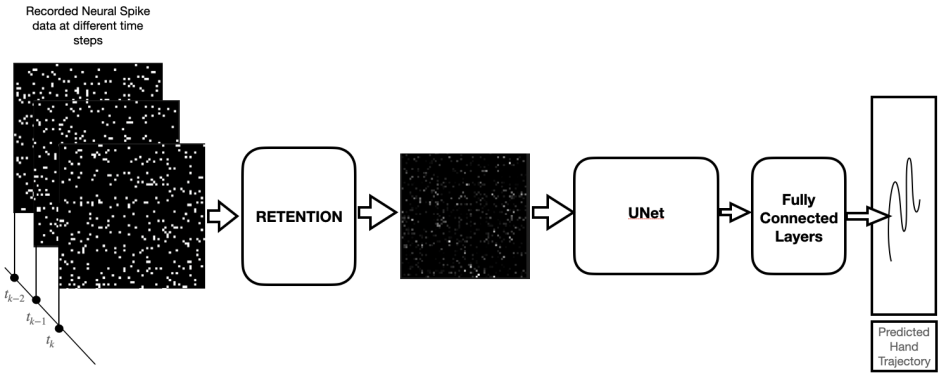


Figure 3.2: Architecture for decoding behavior from neural spiking data

connected layers consisting of 9 hidden layers. This approach, using a pretrained generative model along with

For predicting behaviour from observed neural spiking data, we add extra fully connected layers to the output of UNet model to predict the behaviour variable corresponding to input neural spiking data.

For instance, if we are interested in modelling the dynamics of human motor cortex and finding relationship between spike patterns in motor cortex and trajectory of finger motion, then our output layer should be three-dimensional to represent the three-dimensional motion of human finger. In addition to modelling kinematics of finger, the architecture can also be applied to other tasks such as predicting the probability of seizure given neural spike recordings.

3.4 TRAINING RETENTION BASED MODELS

In this section, we describe how retention based models are trained. Retention variables at different timesteps k are related to each other by a recursive relationship, which can be utilized for developing efficient training algorithms.

Recursive relationship between Retention Variables

Retention variable at any time-step k is given by:

$$\zeta_k = 2^{-(\log M)} \sum_{i=0}^{M-1} \sigma(w_i \otimes x_k + b_i) + 2^{-(\log M)} \zeta_{k-1} \tag{3.9}$$

If the variables x_k are discrete vectors, then the thresholding layer can be omitted and the relation simplifies to:

$$\zeta_k = 2^{-1} x_k + 2^{-1} \zeta_{k-1} \tag{3.10}$$

While training the model, using the recursive equation, we can update ζ_i in an online fashion, instead of pre-computing and storing the retention variables $\{\zeta_i\}_{i=1}^N$ at each time step i.

Online Computation of Retention Variable

Offline Computation of Retention Variable

If the input dataset is drawn from a discrete space, then the retention variables can be ${\zeta_i}_{i=1}^N$ can be pre-computed from the dataset ${x_i}_{i=1}^N$. After computation of the retention variables, parametrized models can be trained similar to existing supervised learning methods where batch of training data is used to compute gradients and update model weights at each iteration.

While online learning based method is memory efficient, we found that Offline learning based method is faster, to train on GPUs.

Algorithm 1 Online Learning

procedure TRAINMODEL(*Data*, *Epochs*)
    Initialize *Model*
    for $epoch = 1$ **to** *Epochs* **do**
        $\zeta = [0, 0, 0, ,]^d$
        for $i = 1$ **to** N **do**
            $\zeta_i, Targets \leftarrow Retention(x_i, \zeta)$
            $\zeta = \zeta_i$
            $Predictions \leftarrow$ FORWARDPASS$(Model, \zeta_k)$
            $Loss \leftarrow$ COMPUTELOSS$(Predictions, Targets)$
            Perform backpropagation to compute gradients
            Update *Model* parameters
        end for
        Evaluate model on validation data
        if performance improves **then**
            Update best model
        end if
    end for
    return Trained *Model*
end procedure

Algorithm 2 Offline Learning

procedure TRAINMODEL(*Data*, *Epochs*)
    Initialize *Model*
    for $epoch = 1$ **to** *Epochs* **do**
        for each *batch* in *Data* **do**
            $Inputs, Targets \leftarrow batch$
            $Predictions \leftarrow$ FORWARDPASS$(Model, Inputs)$
            $Loss \leftarrow$ COMPUTELOSS$(Predictions, Targets)$
            Perform backpropagation to compute gradients
            Update *Model* parameters
        end for
        Evaluate model on validation data
        if performance improves **then**
            Update best model
        end if
    end for
    return Trained *Model*
end procedure

4

RESULTS

In this chapter, we present results obtained from training and evaluating Retention-based models. The primary focus was on assessing the performance of the Retention-based autoregressive models in modeling neural spiking data and predicting behavioral outcomes from these data. The results are divided into two main sections: (1) Generative Modeling of Neural Spike Patterns and (2) Neural Spike to Behavior Decoding.

4.1 GENERATIVE MODELING OF NEURAL SPIKE PATTERNS

Dataset

The dataset used in this thesis was sourced from a detailed study that examined the role of the cortex in navigational decision-making in mice [TCA+22]. This study involved recording neuronal activity from the posterior cortex of mice engaged in a virtual navigation task, which was designed to reflect the challenges animals face in integrating sensation, planning, and action in dynamic environments. The dataset includes neural recordings from approximately 90,000 neurons in the mouse posterior cortex collected using two photon imaging technology [MMSS+99]. We sampled spike trajectories of 4096 neurons from the dataset and created images representing neural spikes at various time-steps.

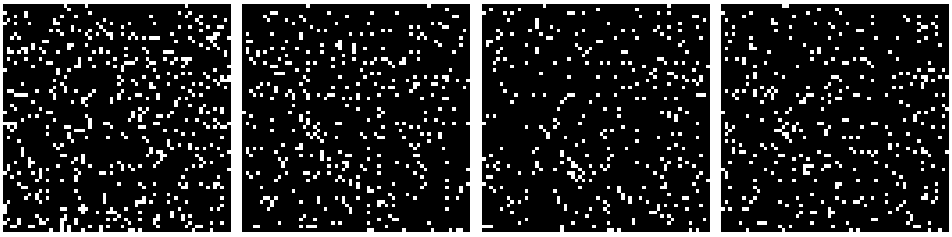
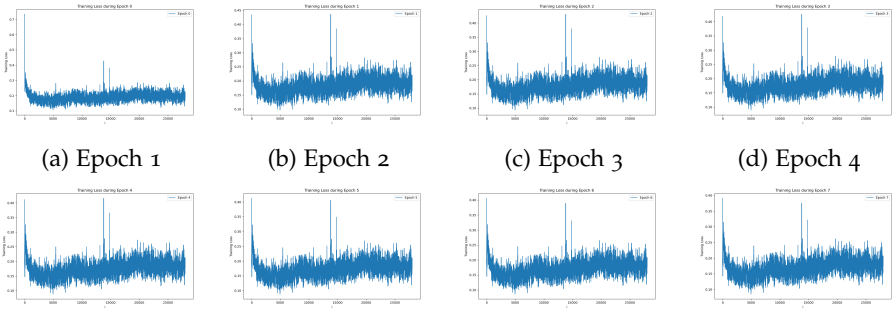


Figure 4.1: t=1 Figure 4.2: t=2 Figure 4.3: t=3 Figure 4.4: t=4

Figure 4.5: Neural Spiking Activity at various timesteps

Training

We trained the model on a single NVIDIA P100 GPU using Adam Optimizer [KB14]. For the specific task of generative modeling of neural dynamics, we employed binary cross entropy as the loss function. This loss function quantified the disparity between the predicted neural spiking patterns generated by the model and the actual observed neural spike pattern at the next timestep. We used an initial learning rate of 0.001 and trained the model for 37 epochs. On NIVIDIA P100 GPU, it took eight hours to complete model training. Within each epoch, we found that the training dynamics is a bit unstable (Figure 4.6). However, despite these minor fluctuations, the average loss over epochs consistently decreased, indicative of the model's capacity to learn and refine its predictive capabilities (Figure 4.7).



(a) Epoch 1 (b) Epoch 2 (c) Epoch 3 (d) Epoch 4

(e) Epoch 5 (f) Epoch 6 (g) Epoch 7 (h) Epoch 8

Figure 4.6: Training Loss over 8 Epochs

Model Performance

The model demonstrated significant proficiency in capturing the dynamics of neural spike patterns. Quantitatively, Retention-based model achieved an average log-likelihood of 0.07 after 34 epochs of training.

Qualitative Analysis

Visual inspection of the generated spike patterns revealed a high degree of similarity to the actual recorded data (See Figure 4.8). Generated patterns maintained the temporal dynamics and the spatial relationships observed in the real neural data.

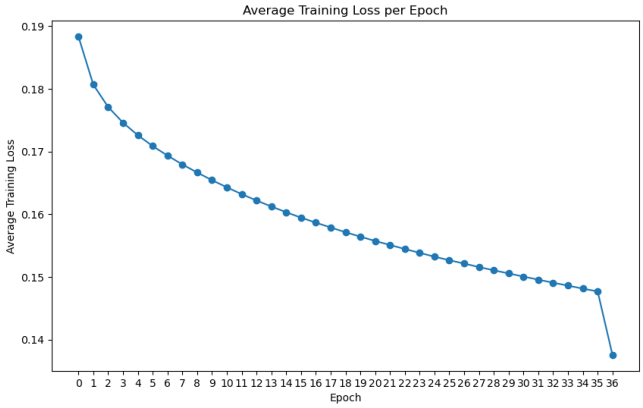


Figure 4.7: Average Training loss vs Epoch

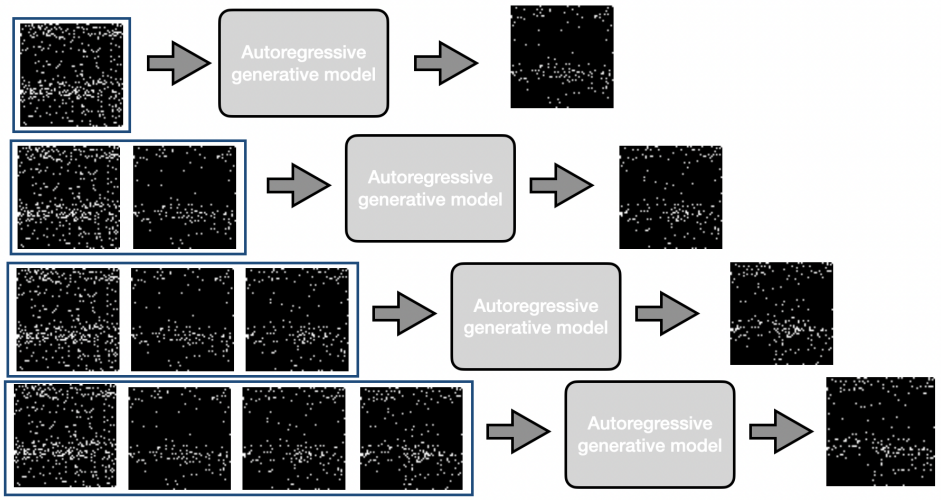


Figure 4.8: samples from generative model

Scalability

The model's performance remained stable even as the length of the input sequences increased, showcasing its capability to handle long sequence lengths efficiently, a key advantage over traditional sequence models. Examining the scalability of the model reveals its consistent performance across varying sequence lengths, a marked improvement over traditional sequence-based models. The model demonstrated this robustness in a practical experiment, generating 5000 images representing neuro pixel data in just 3.5 minutes. Notably, it maintained a constant inference time, regardless of the sequence length, showcasing its efficiency in handling extended sequences. Executed on a MacBook Air M1, these results emphasize the model's effective operation on standard commercial hardware, indicating its potential for scalable applications in processing extensive neural datasets.

4.2 NEURAL SPIKE TO BEHAVIOR DECODING

Dataset

For the neural decoding task, we used the dataset collected by Churchland et.al [CCK+10] for studying the relationship between activity of neurons in motor cortices and movement task in non-human primates. The data was recorded using electrode arrays implanted in the dorsal premotor cortex (PMd) and from surface and sulcal primary motor cortex (M1). The dataset includes neural spiking data from the primary cortex along with simultaneously recorded monkey finger position, cursor position, and target position. We projected the neural recording data to images of size 64x64 with individual neurons assigned to specific pixels in the image. Since the data is recorded from only 130 neurons, the black and white images representing neural activity looks sparse (See Figure 4.13).

Figure 4.9: t=1 Figure 4.10: t=2 Figure 4.11: t=3 Figure 4.12: t=4

Figure 4.13: Neural Spiking Activity at various timesteps

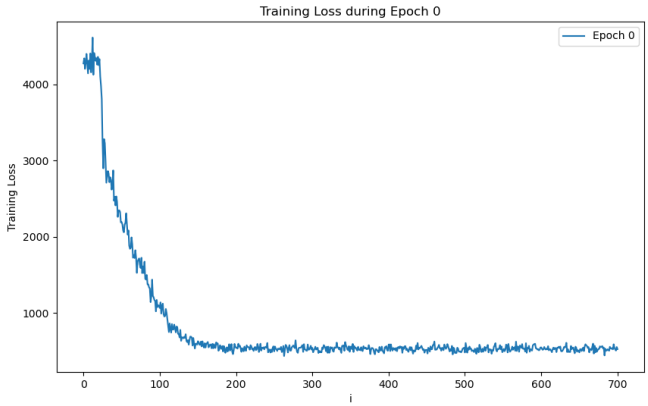


Figure 4.14: Training loss vs Iterations

Architecture

We added extra fully connected layers to the pretrained autoregressive model to compute finger-tip position from Retention Variables. Although the previous model was trained on a different neural spiking datset, we noticed that this architectural choice enables faster learning, when compared to random initialization for all weight parameters.

Training

We trained the model on Macbook Air M1 laptop for 700 iterations, with a batch size of 128. We used Adam Optimizer [KB14] with an initial learning rate of 0.001. The training loss converged to a value of 534, after around 700 iterations (Fig 4.14).

Model Performance

The performance of the model in decoding behavioral outcomes from neural spiking data was evaluated using R2 metric. Our model achieved an R2 score of 75.62 for the neural decoding task. This is not close to the current state of the art performance [AAG+23] which is around 95.82.

5

CONCLUSION

In this thesis, we have addressed the complex challenge of modeling neural dynamics and their relationship with behavior. Through this process, we have developed new scalable methods specifically designed for analyzing neural data. This development is informed by the limitations and capabilities of existing machine learning techniques, including LFADS, NDT, and POYO, which have been instrumental in advancing our understanding of neural dynamics.

The core contribution of our work is the introduction of a novel class of autoregressive models. These models are designed to effectively manage the high temporal resolution characteristic of neural spiking data, a challenge that traditional transformer-based models have struggled with, particularly in high-frequency contexts. Our proposed models showcase an enhanced capability to capture long-range dependencies in time series data. Moreover, they exhibit improved computational efficiency, a crucial advantage considering the complexity and scale of neural datasets.

Importantly, while our methods are developed with a focus on neural data, we recognize their potential applicability in other areas that involve sequence modeling, such as language processing, finance, and engineering. This adaptability underlines the broader relevance of our approach.

In summary, this thesis contributes to the field of neuroscience by providing new tools and perspectives for understanding brain dynamics and behavior. The methodologies we have developed, while tailored for neural data, hold promise for broader applications in various sequence modeling tasks, highlighting the potential for cross-disciplinary impact.

6

APPENDIX

Code

Importantly, while our methods are developed with a focus on neural data, we recognize their potential

Listing 6.1: Training Code

```
import os
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms.functional
from torch.utils.data import DataLoader
from torchvision.transforms import ToTensor
from u_net import UNet
from PIL import Image
from torchvision import transforms
import json
# Define U-Net architecture (same as before)

# Hyperparameters
learning_rate = 0.001
batch_size = 1
num_epochs = 10
# data folder:

data_folder = '/Users/abhinavmuraleedharan/MEng_project/
    MEng-project/code/v_1/data/raw_data/binary_image_data'
# Create U-Net model, loss function, and optimizer
#device = torch.device("cuda" if torch.cuda.is_available()
    else "cpu")
device = torch.device("mps")
model = UNet(n_channels=1, n_classes=1).to(device)
criterion = nn.BCEWithLogitsLoss()  # Binary Cross-Entropy
    loss for binary segmentation
```

```
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)


    # function for loading i th image:

    def load_image(i):
        image_filename = os.path.join(data_folder, f'image_{i}.png')
        if os.path.exists(image_filename):
        # Open the image and convert it to grayscale and then to a PyTorch tensor
            image = Image.open(image_filename).convert('L')  # 'L' mode is for grayscale
            transform = transforms.Compose([
            transforms.Resize((64, 64)),  # Resize the image if required
            transforms.ToTensor()  # Convert the image to a PyTorch tensor
            ])
            image = transform(image)
        else:
            print("Wrong filepath")
        image = image.view(1,1,64,64)
        return image


    batch_idx = 100

    train_losses = []
    # Training loop
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        x_input = torch.zeros(1, 1, 64, 64)
        for i in range(28000):
            # get inputs
            x_input = 2**(-1)*load_image(i) + 2**(-1)*x_input
            targets = load_image(i+1)
            inputs, targets = x_input.to(device), targets.to(device)

```

```
            # Zero the gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = model(inputs)

            # Calculate the loss
            loss = criterion(outputs, targets)

            # Backpropagation and optimization
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            loss_val = loss.item()

            # Print statistics every 10 batches
            if i % 100 == 0:
                print(f"Epoch {epoch + 1}/{num_epochs}, Image:
                    {i}, Loss: {running_loss / 100:.4f}")
            # train_losses.append(running_loss/100)
            train_losses.append({'epoch': epoch, 'i': i, '
                training_loss': loss.item()})
            running_loss = 0.0

            # save model when at every 2000 th i
            if i%1000 == 0:
                print("Saving Checkpoint:")
                checkpoint_path = f'checkpoint_epoch{epoch +
                    1}.pth'
                torch.save(model.state_dict(), checkpoint_path)
                losses_str = json.dumps(train_losses, indent=4)
                with open('training_log.txt', 'w') as file:
                    file.write(losses_str)



    print("Training finished!")
```

Listing 6.2: Training Code

```

    # training code with online computation of auxillary
        variables.
```

```
    # batch size of 1
    from u_net import UNet
    import torch
    import torch.nn as nn
    import torch.optim as optim
    import numpy as np
    import os
    from model import ExtendedUNet
    from torchvision import transforms
    from data import CustomDataset
    from torch.utils.data import DataLoader
    import json
    #cuda
    # Check for GPU #
    device = torch.device("cuda" if torch.cuda.is_available()
        else "cpu")
    # device = torch.device("mps")
    print(f"Using device: {device}")
    ########

    #load UNet model ####

    u_net_model = UNet(n_channels=1,n_classes=1)
    u_net_model.load_state_dict(torch.load('U_Net.pth',
        map_location=torch.device('cpu')))
    # u_net_model = torch.load('U_Net.pth', map_location=torch.
        device('cpu'))
    fc_layers = [64*64, 512,512,512,256, 128,64,32,16,8]  #
        Example sizes, adjust as needed
    output_size = 3  ###

    # instantiate model
    model = ExtendedUNet(u_net_model, fc_layers, output_size).
        to(device)

    #print number of parameters::

    num_params = sum(p.numel() for p in model.parameters())
    print(f"Number of parameters in the model: {num_params}")

    # Define lodd function and optimizer
    criterion = nn.MSELoss()
```

```
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    # Training loop
    num_epochs = 10

    # Define image transformations
    transform = transforms.Compose([
        transforms.Resize((64, 64)),
        transforms.ToTensor()
    ])

    # Create the dataset
    dataset = CustomDataset(img_dir='image_data/',
                            npy_file='Y_target.npy',
                            transform=transform)

    # Create the DataLoader
    data_loader = DataLoader(dataset, batch_size=128, shuffle=
        True)
    model = model.float()
    # Number of epochs
    num_epochs = 10  # You can modify this number based on your
         requirements
    print("Length of dataloader",len(data_loader))
    # Transfer model to GPU if available
    device = torch.device("cuda" if torch.cuda.is_available()
        else "cpu")
    model.to(device)
    train_losses_1 = []
    train_losses_2 = []
    # Training Loop
    i = 0
    for epoch in range(num_epochs):
        model.train()  # Set the model to training mode
        running_loss = 0.0
        i = 0
        for batch in data_loader:
            print("in training loop")
            # Get data
            images = batch['image'].to(device)
            numpy_data = batch['numpy_data'].to(device)

```

```
            # Zero the parameter gradients
            optimizer.zero_grad()

            # Forward pass
            outputs = model(images)

            # Compute the loss
            loss = criterion(outputs, numpy_data)

            # Backward pass and optimize
            loss.backward()
            optimizer.step()
            print(loss.item())
            train_losses_1.append({'epoch': epoch, 'i': i, '
                training_loss': loss.item()})
            if i%100 == 0:
                print("Saving Checkpoint:")
                checkpoint_path = f'checkpoint_epoch{epoch +
                    1}.pth'
                torch.save(model.state_dict(), checkpoint_path)
                losses_str = json.dumps(train_losses_1, indent
                    =4)
                with open('training_log_1.txt', 'w') as file:
                    file.write(losses_str)
            running_loss += loss.item()
            i = i + 1

        # Print statistics
        epoch_loss = running_loss / len(data_loader)
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss
            :.4f}")
        # append training loss
        train_losses_2.append({'epoch': epoch, 'i': i, '
            epoch_loss': epoch_loss})
        checkpoint_path = f'checkpoint_epoch{epoch + 1}.pth' #
            checkpoint path
        torch.save(model.state_dict(), checkpoint_path) # save
            model
        losses_str = json.dumps(train_losses_2, indent=4)
        with open('training_log_2.txt', 'w') as file:
            file.write(losses_str)

```

```
114     print("Training complete")
115
116     # Save the final trained model
117     torch.save(model.state_dict(), 'trained_model_final.pth')
```

BIBLIOGRAPHY

[AAG+23] Mehdi Azabou, Vinam Arora, Venkataramana Ganesh, Ximeng Mao, Santosh Nachimuthu, Michael J Mendelson, Blake Richards, Matthew G Perich, Guillaume Lajoie, and Eva L Dyer, *A unified, scalable framework for neural population decoding*, arXiv preprint arXiv:2310.16046 (2023).

[CCK+10] Mark M Churchland, John P Cunningham, Matthew T Kaufman, Stephen I Ryu, and Krishna V Shenoy, *Cortical preparatory activity: representation of movement or first cog in a dynamical machine?*, Neuron **68** (2010), no. 3, 387–400.

[GZ22] Nicholas Geneva and Nicholas Zabaras, *Transformers for modeling physical systems*, Neural Networks **146** (2022), 272–289.

[H+00] David Heeger et al., *Poisson model of spike generation*, Handout, University of Standford **5** (2000), no. 1-13, 76.

[Hah20] Michael Hahn, *Theoretical limitations of self-attention in neural sequence models*, Transactions of the Association for Computational Linguistics **8** (2020), 156–171.

[HMW+18] Daniel Hernandez, Antonio Khalil Moretti, Ziqiang Wei, Shreya Saxena, John Cunningham, and Liam Paninski, *Nonlinear evolution via spatially-dependent linear dynamics for electrophysiology and calcium data*, arXiv preprint arXiv:1811.02459 (2018).

[JBA+21] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Daniel Zoran, Andrew Brock, Evan Shelhamer, et al., *Perceiver io: A general architecture for structured inputs & outputs*, arXiv preprint arXiv:2107.14795 (2021).

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

[KTK+19] Georgia Koppe, Hazem Toutounji, Peter Kirsch, Stefanie Lis, and Daniel Durstewitz, *Identifying nonlinear dynamical systems via generative recurrent neural networks with applications to fmri*, PLoS computational biology **15** (2019), no. 8, e1007263.

[MMSS+99] ZF Mainen, M Maletic-Savatic, SH Shi, Y Hayashi, R Malinow, and K Svoboda, *Two-photon imaging in living brain slices*, Methods **18** (1999), no. 2, 231–239.

[Mur23] Abhinav Muraleedharan, *Beyond dynamic programming*, arXiv preprint arXiv:2306.15029 (2023).

[POC+18] Chethan Pandarinath, Daniel J O'Shea, Jasmine Collins, Rafal Jozefowicz, Sergey D Stavisky, Jonathan C Kao, Eric M Trautmann, Matthew T Kaufman, Stephen I Ryu, Leigh R Hochberg, et al., *Inferring single-trial neural population dynamics using sequential auto-encoders*, Nature methods **15** (2018), no. 10, 805–815.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox, *U-net: Convolutional networks for biomedical image segmentation*, Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer, 2015, pp. 234–241.

[RNS+18] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al., *Improving language understanding by generative pre-training*.

[SK93] William R Softky and Christof Koch, *The highly irregular firing of cortical cells is inconsistent with temporal integration of random epsps*, Journal of neuroscience **13** (1993), no. 1, 334–350.

[SW20] Qi She and Anqi Wu, *Neural dynamics discovery via gaussian process recurrent neural networks*, Uncertainty in Artificial Intelligence, PMLR, 2020, pp. 454–464.

[TCA+22] Shih-Yi Tseng, Selmaan N Chettih, Charlotte Arlt, Roberto Barroso-Luque, and Christopher D Harvey, *Shared and specialized coding across posterior cortical areas for dynamic navigation decisions*, Neuron **110** (2022), no. 15, 2484–2502.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, Advances in neural information processing systems **30** (2017).

[YP21] Joel Ye and Chethan Pandarinath, *Representation learning for neural population activity with neural data transformers*, arXiv preprint arXiv:2108.01210 (2021).

COLOPHON

This thesis was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić.

The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*".

Here you can insert things like "Figures were created with..."

[Insert version number/description, if you want]