Recurrent Neural Networks and Bayesian Methods for Neural Decoding

Vilde Ung

(Dated: May 31, 2023)

In this analysis, I apply recurrent neural networks and Bayesian methods to multiple spike train data in order to predict the position and velocity of an animal. The methods are applied to data sets collected from two different brain regions, the hippocampus and the motor cortex. For the recurrent neural networks, I tested using a simple recurrent network as well as the long short-term memory (LSTM). Additionally, I applied a naive Bayes model, which incorporates information about position or velocity to model the firing rate of neurons. The analysis shows that the LSTM in both cases performed well on the data sets, explaining 59% and 89% of the variance in the hippocampus and motor cortex data, respectively. Additionally, I find that the naive Bayes model performed better than the LSTM on the hippocampus data, explaining 60% of the variance in the data. However, it performed significantly worse for the motor cortex data, at best explaining 25% of the variance. I discuss the implications of these findings in terms of the neural code, and evaluate if the model assumptions of the naive Bayes model are appropriate.

The code used for this project is available at the GitHub repository: https://github.com/ungvilde/FYS5429

I. INTRODUCTION

An important problem in neuroscience is to understand the relationship between neural activity and behaviour, perception, and cognition. This is generally referred to as the *neural coding* problem. This problem is highly relevant for building brain-machine interfaces, where brain activity is used to control electronic devices, such as a a robotic limb, in real time. It is also relevant for understanding the principles of how the brain processes information. With technology for recording the activity of multiple individual neurons, we can begin to understand the information processing beyond the scale of single neurons. Machine learning methods are therefore relevant for helping us uncover the structure of neural computation at a network level, giving us better insight into brain function.

Neurons transmit information using so-called *action potentials*. Action potentials, also referred to as firings or spikes, involve a rapid change in the electric potential of the neuron that is propagated along the cell membrane. Exactly how action potentials encode relevant features of a stimulus is still an open question. We know that information is encoded in the temporal distribution of spikes, with a temporal resolution at the millisecond scale [1]. We also know that the rate at which spiking occurs correlates with the intensity of the stimulus. Additionally, synchronous neural activity is thought to encode information at an ensemble level [2]. Because of the interactions between stimulus effects and the complex dynamics between neurons, identifying firing patters and understanding their relationship to neural coding is an ongoing research issue.

Neural decoding is essentially a regression (or classification) problem, namely to find a functional mapping between recorded neural activity and target variables. Machine learning algorithms have repeatedly been shown to be excellent functional approximators, and have already been successfully applied to neuroscience problems [3].

In this project, I apply recurrent neural networks (RNNs) and Bayesian methods for predicting outcomes, based on recorded neural activity. These methods are different in many ways, but both have been shown to be useful for the purpose of neural decoding. With Bayesian methods, we can incorporate prior beliefs about how the brain encodes information. This allows us to test different hypotheses about the neural code, by comparing how well they can predict the outcome [4]. On the other hand, RNNs allow us to have a highly flexible relationship between neural activity and outcome variables, and different approaches can be used to incorporate time-dependencies. This can give better predictions [5], but the "black-box" functional mapping of neural networks might not reveal as much about the underlying functioning of the brain.

II. THEORY

A. A Very Brief Neuroscience Overview

In this analysis, I will apply machine learning methods to so-called *spike train data*. Spike trains are data points consisting of the times that a neuron was recorded to spike. When the activity of a neuron is recorded, we measure how the membrane potential, the difference in ion concentration between the inside and the outside of the cell membrane, changes with time. If the membrane potential passes a certain threshold, whereby an action potential is induced, then a spike is recorded at that time. Action potentials are generally thought to be uniform in both duration (approximately 1 ms) and strength, so all the relevant information can be stored in the recorded spike times. For an illustration of how spike trains are related to the membrane potential, see Figure 1.

Here I will analyse the activity of neurons located at two different regions of the brain: the motor cortex and the hippocampus. The motor cortex is a brain area as-

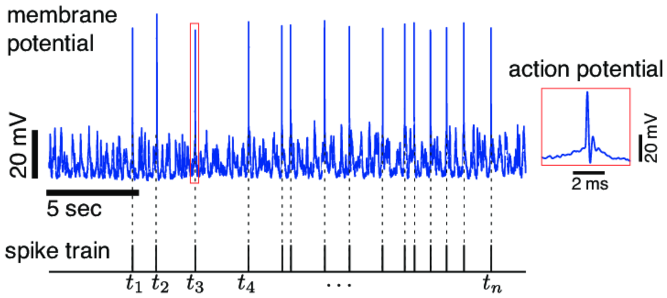

Figure 1. Illustration of the relationship between recorded spike times and the membrane potential of a neuron. The blue trace represents the value of the membrane potential in time. When the membrane potential exceeds a threshold value, a spike time, denoted by $t_1, t_2, \ldots, t_n$, is recorded. The right hand side of the figure shows the shape of a single action potential. The illustration is taken from [6].

sociated with, as the name would imply, planning and generating movement. Previous studies have successfully been able to control the movement of electronic devices based on recordings in the motor cortex [7].

The hippocampus is a brain region involved in learning and memory. Within the hippocampus there are so-called *place cells*. Place cells are thought to model the spacial environment that the animal is in: a given place cell is mostly quiet, but will fire maximally whenever the animal enters a particular region of the environment [8]. Recordings from place cells are therefore useful for decoding information about the animal's position.

Neurons in the brain are highly interconnected and exhibit noisy firing patterns. There will typically be a time-delay between the stimulus and neural activity, for example the time it takes the signal to reach the muscles and cause movement. Also, the previous activity of a neuron can affect its firing patterns into the future. For these reasons, it is relevant to capture the spiking pattern of many neurons and over a wide time range when doing neural decoding.

III. MATERIALS AND METHODS

A. Data Sets

Here, I will consider two data sets: one collected from the motor cortex (M1) of a macaque monkey [9] and one from the hippocampus (CA1) of a rat [10].

In the task for the M1 data set, the monkey controlled the movement of a cursor on a screen using a manipendulum. The target variable is the velocity of the cursor in the x- and y-direction. The data set contains 21 minutes of recorded activity from 164 neurons. In the task for the CA1 data set, the rat chased randomly placed targets on a square platform. The target variable is the rat's location on the (x, y)-plane. The data set contains the activity of 46 neurons recorded for 75 minutes.

Both data sets can be accessed through https://github.com/KordingLab/Neural_Decoding.

B. Recurrent Neural Networks

RNNs are useful for sequential data, such as time series data, because they can explicitly model temporal changes in the system. A key feature of RNNs is parameter sharing. A dense feedforward network needs to fit parameters for each time step, while RNNs can use the same model parameters across different time steps. This allows RNNs to learn from different examples and generalise. This improves the statistical strength of the model, and allows the model to identify important information regardless of exactly where in the time-series it occurs. The parameters are shared by allowing past outputs to influence the current output, i.e. a recurrent operation.

At each time t, the input $\mathbf{X}_t$ is projected to the hidden layer $\mathbf{H}_t$, which is recurrently connected to itself in time. First, we initialise the hidden state $\mathbf{H}_0$. Then the forward propagation of the recurrent network can be computed as

$$\begin{aligned} \mathbf{H}_t &= \mathrm{ReLU}(\mathbf{b} + \mathbf{W}\mathbf{H}_{t-1} + \mathbf{U}\mathbf{X}_t) \\ \hat{\mathbf{y}}_t &= \mathbf{c} + \mathbf{V}\mathbf{H}_t, \end{aligned}$$

where the model parameters are the bias vectors $\mathbf{b}$ and $\mathbf{c}$, and the weight matrices $\mathbf{W}$, $\mathbf{V}$ and $\mathbf{U}$. The ReLU-function is a non-linear activation function. This is done across all time steps $t \in \{1, 2, \ldots, K\}$. The predicted value $\hat{\mathbf{y}}_t = (\hat{y}_1^t, \hat{y}_2^t)$ is a vector containing the predictions of the target variables. There are two target variables because the position and velocity of the animal have two components.

The loss is computed as the total mean squared error (MSE) across all time steps. That is

$$\mathrm{MSE} = \frac{1}{K}\sum_{t=1}^{K}(y_1^t - \hat{y}_1^t)^2 + (y_2^t - \hat{y}_2^t)^2.$$

Here, $\mathbf{y}_t = (y_1^t, y_2^t)$ denote the observed values. To minimise loss, we apply the backpropagation algorithm through time (BPTT), which involves computing the gradient of the loss function iteratively by going backwards in time. This is a generalisation of the standard backpropogation algorithm.

However, a common problem with BPTT is vanishing or exploding gradients. This can make it difficult for the network to learn long-term dependencies, because the gradient of a long-term interaction has exponentially smaller or larger magnitude than the gradient of a short term interaction [11].

1. *Long Short-Term Memory*

One way to combat this problem, is using so-called *gated* RNNs. The Long Short-Term Memory (LSTM) model is an example of this type of network. The LSTM uses gated units to control how information flows through the different parts of the network. It is typically made up of a *forget gate*, an *input gate* and an *output gate*. These have their own weights and biases, and are computed as

$$\mathbf{f}_t = \sigma\left(\mathbf{b}_f + \mathbf{W}_f\mathbf{H}_{t-1} + \mathbf{U}_f\mathbf{X}_{t-1}\right) \qquad \text{(forget gate)}$$
$$\mathbf{i}_t = \sigma\left(\mathbf{b}_i + \mathbf{W}_i\mathbf{H}_{t-1} + \mathbf{U}_i\mathbf{X}_{t-1}\right) \qquad \text{(input gate)}$$
$$\mathbf{o}_t = \sigma\left(\mathbf{b}_o + \mathbf{W}_o\mathbf{H}_{t-1} + \mathbf{U}_o\mathbf{X}_{t-1}\right) \qquad \text{(output gate)}$$

Here, σ is the sigmoid function. The typical hidden units in RNNs are replaced by *cells*, which are recurrently connected to each other. The cell's internal state $\mathbf{S}_t$ is updated as

$$\mathbf{S}_t = \mathbf{f}_t \odot \mathbf{S}_{t-1} + \mathbf{i}_t \odot \tanh\left(\mathbf{b} + \mathbf{W}\mathbf{H}_{t-1} + \mathbf{U}\mathbf{X}_t\right),$$

where $\mathbf{W}, \mathbf{U}$ and $\mathbf{b}$ denote the weights and biases of the cell. The $\odot$ operator denotes element-wise multiplication. Here, we see that the cell state depends on how the forget gate affects the previous state of the cell $\mathbf{S}_{t-1}$, and the input gate determines how the input affects the cell state.

Finally, the output of the cell is computed as

$$\mathbf{H}_t = \mathbf{o}_t \odot \tanh\left(\mathbf{S}_t\right).$$

The predicted value is, similarly as before, computed linearly from the output as

$$\hat{\mathbf{y}}_t = \mathbf{c} + \mathbf{V}\mathbf{H}_t.$$

2. *Data Preprocessing*

The data sets contain the spike times of the different neurons and target values (velocity or position) in continuous time, and these values must be discretised into smaller time bins. This requires that we decide on the temporal resolution of the predicted values. That is, we partition the total duration of the experiment into smaller equally sized time intervals, and compute the average of the target and the input values within each of these intervals. For an experiment of duration T and time bins of length Δt, this gives us a total number of $T/\Delta t$ values.

The features of our model is the average firing rate of the neurons within a given time bin. For a given neuron, this is computed as

$$\frac{\text{Num. spikes in interval } (t, t+\Delta t]}{\Delta t}.$$

Furthermore, it has been shown that incorporating time-delay in the decoding scheme makes the predictions more robust [12]. We therefore use the firing rate from B time bins when making a prediction at time bin t, which can include time bins preceding and following t. Then, the input $\mathbf{X}_t$ of our decoder at time bin t will include N features from B time bins, such that $\mathbf{X}_t$ is a $B \times N$ matrix. The full data set $\mathbf{X}$ is a $K \times B \times N$ tensor, where $K = T/\Delta t - B + 1$.

C. Bayesian Methods

With Bayesian methods we use Bayes theorem to estimate the probability distribution of outcomes, given what we have observed. Here, we apply a naive Bayes model similar to the one that is presented in [8]. Zhang et al. estimated the movement of a rat based on recordings from place cells in the hippocampus, similar to what I do in this analysis. In the naive Bayes model, we assume conditional independence between the feature values to approximate the posterior probability.

Let n_i^t denote the number of spikes of neuron i within a given time bin t. Then $\mathbf{n}_t = (n_1^t, n_2^t, \ldots, n_N^t)$ is a vector of length N containing the spike counts of the recorded neurons in time bin t. Furthermore, let $\mathbf{y}_t$ be the target variable. Again, $\mathbf{y}_t$ is the x- and y-coordinates of either the location of the rat in the plane, or the velocity of the cursor, at time bin t.

It follows from Bayes' theorem that

$$P(\mathbf{y}_t|\mathbf{n}_t)P(\mathbf{n}_t) = P(\mathbf{n}_t|\mathbf{y}_t)P(\mathbf{y}_t).$$

The goal is to estimate the conditional probability $P(\mathbf{y}_t|\mathbf{n}_t)$, also referred to as the posterior distribution. The probability distribution of the target variables $P(\mathbf{y}_t)$, also called the prior, can be estimated directly from the data. That is

$$P(\mathbf{y}_t) = \frac{\text{Num. times in state } \mathbf{y}_t}{\text{Total possible states}}. \tag{1}$$

Alternatively, we can assume the prior to be approximately uniform, meaning that we assume it is equally likely for any outcome state to be observed. The probability distribution of the model evidence $P(\mathbf{n}_t)$ is independent of $\mathbf{y}_t$ and does not need to be directly computed, as it is effectively constant.

This leaves us with the task of evaluating the conditional probability $P(\mathbf{n}_t|\mathbf{y}_t)$. As a first step, we assume that the spike count of any given neuron is conditionally independent of the spike counts of other neurons, given the outcome $\mathbf{y}_t$. Then

$$P(\mathbf{n}_t|\mathbf{y}_t) = \prod_{i=1}^{N} P(n_i^t|\mathbf{y}_t).$$

Furthermore, we can model the spike count of a neuron with a Poisson distribution. Then

$$P(n_i^t|\mathbf{y}_t) = \frac{[f_i(\mathbf{y}_t)]^{n_i^t}}{n_i^t!}\exp\left(-f_i(\mathbf{y}_t)\right).$$

Here, $f_i(\mathbf{y}_t)$ is the expected spike count of neuron i when being in state $\mathbf{y}_t$, also referred to as the *tuning curve*. It models the relationship between the firing rate of neuron i and being in state $\mathbf{y}_t$. This model can be estimated by fitting a Poisson generalised linear model to the data. Then, the tuning curve can be modelled as

$$\log(f_i(\mathbf{y}_t)) = \beta_0 + \beta_1 y_1 + \beta_2 y_2, \tag{2}$$

where $\mathbf{y}_t = (y_1, y_2)$ and the β_j's are the model parameters. This will be referred to as a linear form tuning curve. Alternatively, we can model the tuning curve as

$$\log(f_i(\mathbf{y}_t)) = \beta_0 + \beta_1 y_1 + \beta_2 y_2 + \beta_3 y_1 y_2 + \beta_4 y_1^2 + \beta_5 y_2^2. \quad (3)$$

Here, we use a quadratic form of the covariates to get a more complex model of the tuning curve. This will be referred to as a quadratic form tuning curve.

Finally, we incorporate a continuity constraint, meaning that we use the state of the previous time step $\mathbf{y}_{t-1}$ to estimate the current state $\mathbf{y}_t$. This will reduce the chance of having erratic jumps in the predicted values. In effect, this means that we want to estimate the probability $P(\mathbf{y}_t|\mathbf{n}_t, \mathbf{y}_{t-1})$, i.e. the probability of being in state $\mathbf{y}_t$, given $\mathbf{n}_t$ spikes and having previously been in state $\mathbf{y}_{t-1}$. From Bayes' theorem, we have

$$P(\mathbf{y}_t|\mathbf{n}_t, \mathbf{y}_{t-1}) \propto P(\mathbf{y}_{t-1}|\mathbf{y}_t, \mathbf{n}_t) P(\mathbf{n}_t|\mathbf{y}_t) P(\mathbf{y}_t).$$

We assume that the previous state $\mathbf{y}_{t-1}$ is independent of the spike count at time bin t. Then this simplifies to

$$P(\mathbf{y}_t|\mathbf{n}_t, \mathbf{y}_{t-1}) \propto P(\mathbf{y}_{t-1}|\mathbf{y}_t) P(\mathbf{n}_t|\mathbf{y}_t) P(\mathbf{y}_t).$$

All that remains is to evaluate $P(\mathbf{y}_{t-1}|\mathbf{y}_t)$. We do this by modelling it as a 2-dimensional Gaussian distribution, i.e. a random walk. This means that the two states are unlikely to be very far apart in euclidean space. That is

$$P(\mathbf{y}_{t-1}|\mathbf{y}_t) \propto \exp\left(-\frac{||\mathbf{y}_t - \mathbf{y}_{t-1}||_2^2}{2\sigma^2}\right).$$

The variance σ^2 is estimated as the average squared distance between $\mathbf{y}_t$ and $\mathbf{y}_{t-1}$ in the data.

Finally, the predicted value can be found as the target value which is maximally likely, so that

$$\hat{\mathbf{y}}_t = \underset{\mathbf{y}_t}{\operatorname{argmax}} P(\mathbf{y}_t|\mathbf{n}_{t^*}, \mathbf{y}_{t-1}). \quad (4)$$

Note that in Eq. (4) we use the spike count of several time bins, denoted by t^*, not only the concurrent time bin t. That is because we include the spike counts from a specified number of preceding and/or following time bins, as well as the concurrent bin t.

1. Data Preprocessing

When applying the naive Bayes decoder, the data set $\mathbf{X}$ was a $K \times N$ matrix, where the element $x_{i,j}$ was the spike count of neuron j at time bin i. The spike count included the spikes of B time bins preceding and/or following time bin i, as well as the concurrent bin i. The target values $\mathbf{y}$ was a $K \times 2$ matrix containing the coordinates in the plane or the velocities of the cursor for each time bin.

The state space, which consisted of the (x, y)-plane for the CA1 data set, and the velocities (v_x, v_y) for the M1 data set, was discretised with a resolution of 100×100 values for the CA1 data, and 50×50 values for the M1 data.

IV. RESULTS

I applied a simple RNN model, an LSTM model, and a naive Bayes model to both the M1 data set and the CA1 data set. The neural networks were implemented using the Keras library [13]. The Python code used for Bayesian decoding is based on the Neural Decoding package from [14]. For training the neural networks, both data sets were centered and scaled to have unit variance with respect to the training data. The target values were centered.

To validate and test the methods, I did 10-fold cross validation. For each fold, the data set was split into three parts, of which 10% was a contiguous validation set, 10% was a contiguous testing set, and 80% was training data. The data set was split into contiguous parts to preserve the time series structure of the data. For the RNNs, I optimised the hyperparameters using Bayesian optimisation [15]. The hyperparameters I investigated were the number of epochs and the number of units in the network.

For the CA1 data set, I used a time resolution of $\Delta t = 200$ ms, with a time delay of 800 ms preceding activity and 1000 ms of following activity. That is, I used 4 time bins of preceding activity, the concurrent time bin, and 5 time bins of following activity, to predict the location of the rat, which is in all $B = 11$ time bins. For the M1 data set, the time resolution was $\Delta t = 50$ ms. For the neural network models I used 700 ms of activity, specifically 13 time bins of preceding activity and the concurrent bin.

I first investigated the simple RNN and LSTM methods. The networks were trained using the Adam optimiser [16]. I then plotted the predicted values to compare their predictive performance. Both methods appeared to perform very well on the M1 data set, but the results were not as good on the CA1 data set. I then applied the naive Bayes decoder to both data sets, and I observed that it performed better on the CA1 data set than on the M1 data set. Figure 2 and Figure 3 show the predicted values of the different decoders for the M1 and CA1 data sets, respectively.

Next, I did 10-fold cross validation and computed the coefficient of determination R^2 of each fold on a separate testing set. The R^2 score is computed as

$$R^2 = 1 - \frac{\sum_{i=1}^{K}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{K}(y_i - \bar{y})^2}.$$

Here, y_i is the ground truth of time bin i, $\hat{y}_i$ is the predicted value at time bin i, and $\bar{y}$ is the mean value of the target variable across time bins. Because the target values in this analysis have both x- and a y-components, I report the average R^2 score of the two target values. The R^2 score quantifies the proportion of variance that can be explained by the model, and it indicates a good fit when it is close to one.

When I applied the naive Bayes model, I investigated using a uniform prior as well as estimating the prior from

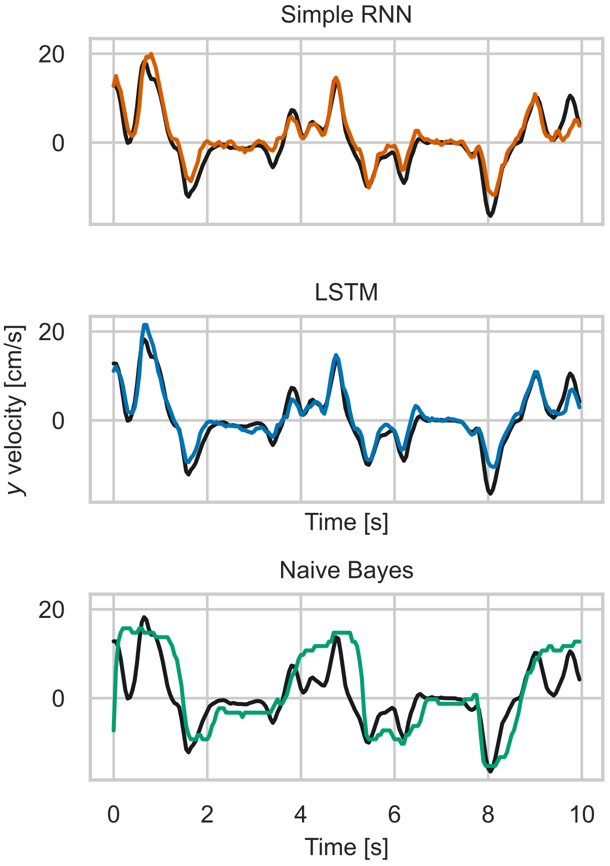

Figure 2. The predicted velocity values in the y-direction for the M1 data set. The black trace represents the observed values from a 10 s time interval. The colored trace represents the predicted values from the different methods, respectively the simple RNN, LSTM, and naive Bayes.

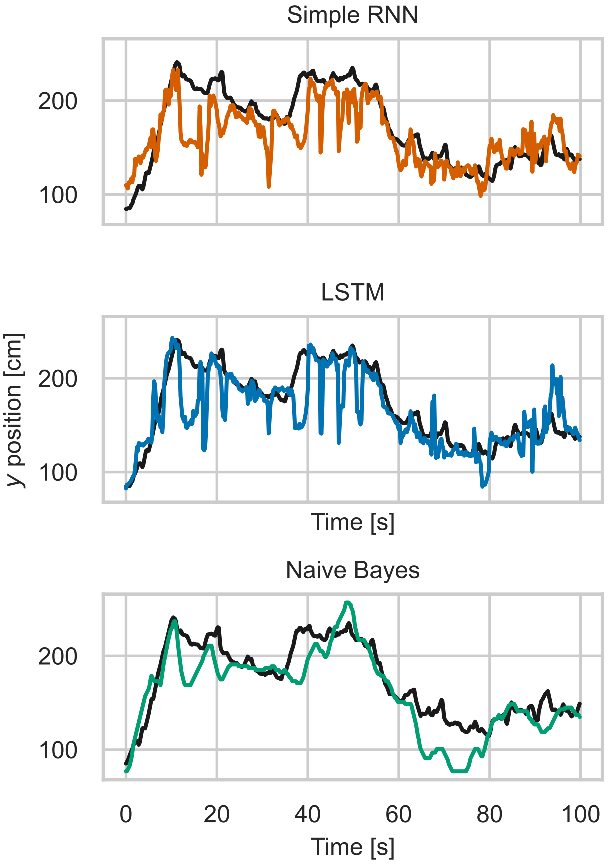

Figure 3. The predicted position values in the y-direction for the CA1 data set. The black trace represents the observed values from a 100 s time interval. The colored trace represents the predicted values from the different methods, respectively the simple RNN, LSTM, and naive Bayes.

the training data, as in Eq. (1). The validation results showed that there was no added benefit by estimating the prior from the data, as the the R^2 scores were worse. This is possibly because the prior distribution changes with time, so that the distribution estimated from the training data is not applicable to the validation data. To simplify the decoder and get the best performance, I used a uniform prior.

For the naive Bayes model, I was interested in seeing if the model improved when the covariates of the tuning curve (i.e. the x- and y-values of the position or velocity) were transformed into a second-order polynomial, compared to a straight-forward linear form. See Eq. (2) and Eq. (3) for a specification of the linear and quadratic form of the model, respectively. The advantages of using covariates with quadratic form when modelling the tuning curve has previously been discussed in [17], where they showed that the quadratic model performs well on spiking data.

The results from cross-validation show that the best decoder on the CA1 data set is the naive Bayes decoder with a quadratic model of the tuning curve. The R^2 score of the LSTM model was marginally worse than naive Bayes for the CA1 data set, but it had a smaller standard deviation. The best decoder on the M1 data set is LSTM, which was able to explain as much as 89% of the variance. For the M1 data set, the predictions of the naive Bayes decoder were significantly worse than the predictions of the neural networks. Interestingly, the naive Bayes decoder with quadratic form performed significantly better than the same model with linear form on both data sets, explaining more than 20% of the unexplained variance in the linear model. The full results from cross validation can be seen in Table I.

V. DISCUSSION

I have applied RNN architectures and Bayesian methods to spike train data to decode information about velocity and position. I used these methods because they are methodologically different in interesting ways, and

Table I. The mean R^2 scores of the models computed from 10-fold cross-validation, with the standard deviation. The results for each data set is shown.

Decoder	CA1
LSTM	0.59 ± 0.07
Simple RNN	0.53 ± 0.08
Naive Bayes (Quadratic)	0.60 ± 0.10
Naive Bayes (Linear)	0.38 ± 0.08
Decoder	M1
LSTM	0.89 ± 0.009
Simple RNN	0.86 ± 0.012
Naive Bayes (Quadratic)	0.26 ± 0.153
Naive Bayes (Linear)	0.0 ± 0.28

therefore have different strengths and weaknesses with respect to this analysis. On one hand, RNNs take in series of inputs and can accumulate information about inputs over time. They are therefore good at modelling time-dependencies and at finding a flexible functional mapping between spike trains and behaviour. On the other hand, Bayesian methods model physical and biological features of the system explicitly, and can therefore reveal relevant features underlying the behaviour, at the cost of being less flexible.

A. Predictive Performance

On both data sets, the LSTM was able to predict the target variables with reasonably good accuracy. The LSTM is the neural network that is best able to capture the long-ranging time-dependencies in the data to make predictions [11]. It is also a method that can find complex and nonlinear relationships of the features to make flexible predictions. My findings support the fact that, in general, RNN architectures are well-suited for sequential data with temporal dependencies, as is the case for both of these data sets. They are also in line with what other studies have found, namely that the LSTM generally is quite successful for movement decoding [18]. Although the LSTM performed well on both data sets, it was also the model that demanded the most computational resources for training.

Although the LSTM performed consistently well on both data sets, it was the naive Bayes model that had the best predictive performance on the CA1 data set. This is an interesting finding, as it indicates that long-ranging time-dependencies are not necessary for making predictions. The naive Bayes model only used spike counts from 2 s of activity for a given prediction, and this proved to be sufficient for outperforming deep learning methods. It was, however, necessary to use a quadratic form tuning curve, otherwise the model became too simplified. The quadratic form of the tuning curve increases the number of model parameters, and therefore allows for greater complexity and flexibility.

On the other hand, the naive Bayes method was significantly worse when applied to the M1 data set. This indicates that the covariates of the model and the model complexity are not sufficient for capturing the important features of the system. This finding further indicates that different brain regions might require different modelling approaches to best capture the relevant biological features for neural decoding.

Additionally, we should note that the CA1 data set contained recordings from fewer neurons than the M1 data set (46 neurons vs. 164 neurons). This can limit the ability of the complex models like LSTM and simple RNN to discover the relevant features of the system, implying that a simpler model like the naive Bayes can extract useful information more efficiently. Alternatively, incorporating regularisation for the neural networks could alleviate possibly this problem. In other words, there is a bias-variance trade-off between the different methods that depend on the model complexity and the size of the data set.

B. Model Assumptions

The naive Bayes model is based on two main assumptions: that the spike counts have a Poisson distribution, and that the cells fire independently. Although these assumptions are reasonable and useful for this analysis, they do not necessarily reflect the underlying biology exactly.

First of all, a Poisson model of the spike counts implies that the inter-spike intervals (the waiting time between successive spikes) should be approximately exponentially distributed. But we know that place cells tend to exhibit bursts of activity [8], which would violate the "memoryless" property of exponentially distributed inter-spike intervals. Additionally, we know that neurons have a *refractory period*, where the membrane potential needs time to equilibrate after spiking. This prevents the neuron from firing again for a short time period right after spiking. However, the Poisson model cannot capture this inability to have very short inter-spike intervals.

The independence assumption implies that the joint probability of any two neurons firing when the animal is in a given state is equal to the product of their individual probabilities. That is, for any two neurons

$$P(n_1^t, n_2^t|\mathbf{y}_t) = P(n_1^t|\mathbf{y}_t)P(n_2^t|\mathbf{y}_t).$$

However, because neurons are nodes in a highly connected network, we cannot be certain that their firing patterns are independent in reality. For example, if the recordings happen to collect the spike times from two place cells that represent the same region of space, their spike counts would be highly correlated. Additionally, some hypothesise that neurons that fire synchronously carry important information, especially in the cortex [2]. In this case, activity at the ensemble level could be an important feature of the system.

In their paper, Truccolo et al. [4] propose a framework for an alternative Poisson model, where history effects, stimulus effects, and ensemble activity all can be included in modelling the tuning curve. As a continuation of this analysis, it could have been interesting to apply their model on the M1 data set to see if it improved predictive performance. If it did, this might indicate that ensemble activity and/or history effects are a central component of the neural code. Still, because the firing patterns of neurons tend to be both sparse and very noisy (and neurons are therefore unlikely to fire together), independence is for practical purposes considered to be a reasonable approximation of the probability distribution.

Finally, I have included a continuity constraint in the naive Bayes model, which ensures that the predicted values are reasonably close to each other at subsequent time steps. This prevents the predictions from becoming erratic, and is especially useful when the firing rate of the cells is close to zero, which causes there to be little available information for the model to make accurate predictions. It does, however, come at the cost that erroneous predictions can get "stuck" and take longer to "recover" from, especially if the standard deviation σ is small. When looking at Figure 3 we can see that the naive Bayes predictions are more smooth than the predictions of the simple RNN and the LSTM.

VI. CONCLUSION

With improving methods for investigating the neural code, we can hope to better understand how the brain represents and processes information about the outside world. This analysis has shown that machine learning methods, both deep learning algorithms like RNNs and simpler models like the naive Bayes, can be useful for this purpose. The RNN methods in particular can be useful for making accurate predictions of outcome variables, which is highly relevant for developing technologies like brain-machine interfaces. RNNs can capture complex features and long-ranging time-dependencies in a flexible way. However, they require sufficiently large data sets to train, and training is somewhat expensive and time consuming. Additionally, neural networks can be difficult to interpret with respect to what information is physiologically relevant to the neural code. The naive Bayes model is simpler, in terms of having fewer model parameters, and captures explicitly what information is relevant to the neural code. However, this simplicity comes with the cost of flexibility: compared to RNNs, the naive Bayes model only performed well on the CA1 data, indicating that further improvements to the model might be needed to make it better suited to different kinds of brain regions. I have briefly discussed an alternative Poisson model of the tuning curve, which could possibly improve the predictive performance of the naive Bayes model by incorporating other relevant features. Overall, these results are in line with what other studies have found, and support the fact that machine learning methods are powerful and useful for neural decoding.

[1] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*, 6th ed., edited by L. F. Abbott (MIT Press, 2001).

[2] C. F. Stevens and A. Zador, Neural coding: The enigma of the brain, Current Biology **5**, 1370 (1995).

[3] J. I. Glaser, A. S. Benjamin, R. Farhoodi, and K. P. Kording, The roles of supervised machine learning in systems neuroscience, Progress in Neurobiology **175**, 126 (2019).

[4] W. Truccolo, U. T. Eden, M. R. Fellows, J. P. Donoghue, and E. N. Brown, A point process framework for relating neural spiking activity to spiking history, neural ensemble, and extrinsic covariate effects, Journal of Neurophysiology **93**, 1074 (2005).

[5] P. Szabó and P. Barthó, Decoding neurobiological spike trains using recurrent neural networks: a case study with electrophysiological auditory cortex recordings, Neural Computing and Applications , 2825 (2022).

[6] I. Park, S. Seth, A. Paiva, L. Li, and J. Principe, Kernel methods on spike train space for neuroscience: A tutorial, IEEE Signal Processing Magazine **30** (2013).

[7] M. D. Serruya, N. G. Hatsopoulos, L. Paninski, M. Fellows, and J. F. Donoghue, Instant neural control of a movement signal, Nature **416**, 141–142 (2002).

[8] K. Zhang, I. Ginzburg, B. L. McNaughton, and T. J. Sejnowski, Interpreting Neuronal Population Activity by Reconstruction: Unified Framework With Application to Hippocampal Place Cells, Journal of Neurophysiology **79**, 1017 (1998).

[9] J. I. Glaser, M. G. Perich, P. Ramkumar, L. E. Miller, and K. P. Kording, Population coding of conditional probability distributions in dorsal premotor cortex, Nature communications 10.1101/137026 (2018).

[10] K. Mizuseki, A. Sirota, E. Pastalkova, and G. Buzsáki, Theta oscillations provide temporal windows for local circuit computation in the entorhinal-hippocampal loop, Neuron **64**, 267 (2009).

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016) `http://www.deeplearningbook.org`.

[12] F. Liu, S. Meamardoost, R. Gunawan, T. Komiyama, C. Mewes, Y. Zhang, E. Hwang, and L. Wang, Deep learning for neural decoding in motor cortex, Journal of Neural Engineering **19**, 056021 (2022).

[13] F. Chollet *et al.*, Keras, `https://keras.io` (2015).

[14] J. I. Glaser, A. S. Benjamin, R. H. Chowdhury, M. G. Perich, L. E. Miller, and K. P. Kording, Machine learning for neural decoding, eNeuro **7**, 10.1523/ENEURO.0506-19.2020 (2020).

[15] J. Snoek, H. Larochelle, and R. P. Adams, Practical bayesian optimization of machine learning algorithms, Advances in neural information processing systems **25** (2012).

[16] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization (2017), 1412.6980.

[17] I. Park, S. Seth, A. Paiva, L. Li, and J. Principe, Kernel methods on spike train space for neuroscience: A tutorial, IEEE Signal Processing Magazine **30** (2013).

[18] J. A. Livezey and J. I. Glaser, Deep learning approaches for neural decoding across architectures and recording modalities, Briefings in Bioinformatics **22**, 1577 (2020).