NeuroView-RNN: It's About Time

CJ BARBERAN, Rice University, USA

SINA ALEMOHAMMAD, Rice University, USA

NAIMING LIU, Rice University, USA

RANDALL BALESTRIERO, Rice University, USA

RICHARD G. BARANIUK, Rice University, USA

Recurrent Neural Networks (RNNs) are important tools for processing sequential data such as time-series or video. Interpretability is defined as the ability to be understood by a person and is different from explainability, which is the ability to be explained in a mathematical formulation. A key interpretability issue with RNNs is that it is not clear how each hidden state per time step contributes to the decision-making process in a quantitative manner. We propose *NeuroView-RNN* as a family of new RNN architectures that explains how all the time steps are used for the decision-making process. Each member of the family is derived from a standard RNN architecture by concatenation of the hidden steps into a global linear classifier. The global linear classifier has all the hidden states as the input, so the weights of the classifier have a linear mapping to the hidden states. Hence, from the weights, NeuroView-RNN can quantify how important each time step is to a particular decision. As a bonus, NeuroView-RNN also offers higher accuracy in many cases compared to the RNNs and their variants. We showcase the benefits of NeuroView-RNN by evaluating on a multitude of diverse time-series datasets.

1 INTRODUCTION

Recurrent neural networks (RNNs) [19] are ubiquitous in deep learning, because their design enables them to process arbitrary length sequential data. For example, RNNs and their variants like the Gated Recurrent Unit (GRU) [9] and Long Short-Term Memory (LSTM) [19] have been core components in numerous applications, such as machine translation [8], image/video captioning [43, 46], and action recognition [14, 27]. There are other works in studying the dynamics of generalization, learning, and initializations of RNNs using neural tangent kernels [1, 2]. However, even though RNNs are powerful tools, they are challenging to interpret and explain.

From [15], they state that interpretability has many definitions. The definition of interpretability [15] that we will use is the ability to be understood by a person. In addition, the term, explainability is different from interpretability and will be defined as the ability to be explained in a mathematical formulation.

Deep learning interpretability has become an important topic since RNNs and their variants have become useful in several applications. The minimal amount of interpretability and explainability does not reveal why they have this performance on their tasks. With more interpretability with these models, it can explain why they are performing well on their respective tasks. Plus, if the model is not performing well, it is difficult to assess which specific parts of the signal contribute to the final decision. Since the model is opaque and hard to interpret, this creates a black-box effort for a practitioner.

There are works that have contributed to providing interpretability/explainability about how these RNNs perform. For instance, the work from [18] explores the connection of the input variables to the RNN in order to interpret the performance. Other works such as [17, 20, 23] focus on the interpretability of RNNs within their respective task, hence limiting the ability to interpret other tasks. [13, 20, 31] create their own type of RNN in order to provide interpretability in their application. The works of [3, 17, 23] use some metrics to provide interpretability with the RNN/LSTM that they use for their application.

With the works that created an interpretable RNN ([13, 20, 31]), the main issue is that the definition of interpretability is focused on their application, which would make it difficult to adapt to another application for interpretability. For instance, [20] creates a finite-automaton RNN for text classification. This work would be hard to adapt to other applications. Another issue is that with the works mentioned, they cannot provide a mathematical formulation that explains the prediction with the hidden states.

Contributions. We propose NeuroView-RNN (NV-RNN) as a novel general framework that provides enhanced interpretability and explainability to classification.

(1) We introduce the NV-RNN framework, which consists of a concatenation of all of the hidden states to be the input to the linear classifier. This allows a linear mapping to all of the hidden states to the classes. This linear mapping allows interpretability and explainability since the prediction for each class is in a dot product of the weights from the linear classifier with the hidden states.

(2) Influenced from the work of [4], we provide interpretability and explainability defined in the Introduction to showcase how NV-RNN, NV-GRU, and NV-LSTM can provide more understanding especially with applications and architectures of RNNs, GRUs, and LSTMs. This is in stark contrast to the work of [4] that used convolutional neural networks (CNNs) for image classification which cannot on its own be used in specific RNN architectures like bidirectional RNNs or variable length input.

(3) We have better performance on most of the datasets from Table 1 to show why this NV-RNN framework should be used. In addition, we have on par performance on some of the datasets from Table 1 and 2. The results show that we have the accuracy performance compared to typical RNNs and are able to interpret and explain the performance.

(4) We use numerous case studies about weight initialization, bidirectional GRUs, semantic analysis, video action recognition, and counterfactuals to show the benefit of NV-RNN.

2 BACKGROUND

2.1 Recurrent Neural Networks

Given an input sequence data $x = \{x_t\}_{t=1}^{T}$ of length T with data at time t, $x_t \in \mathbb{R}^m$, an RNN performs the following recursive block computation at each time step t

$$h^{(t)}(x) = \mathcal{F}_\theta(h^{(t-1)}(x), x_t) \in \mathbb{R}^n, \tag{1}$$

where $h^{(0)}(x) = 0$ and n is the number of parameters for the hidden state. $\mathcal{F}_\theta : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is the hidden time steps mapping with time-agnostic parameters θ. Each recurrent architecture has a different $\mathcal{F}_\theta$. For a simple RNN [12] we have this formulation,

$$\mathcal{F}_\theta(h^{(t-1)}(x), x_t) = h^{(t)}(x) = \phi\left(Wh^{(t-1)}(x) + Ux_t + b\right), \tag{2}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the activation function that act point-wise on a vector and $\theta = \text{vect}\left[\{W, U, b\}\right]$ contains the mapping parameters. In case of simple RNNs, we use the sigmoid function $\phi(\alpha) = \frac{1}{1+e^{(-\alpha)}}$. Other RNNs variants such as GRU [8] and LSTM [19] have a more complex mapping $\mathcal{F}_\theta$. See Supplementary Material Section A.1 for more details.

Bidirectional recurrent architectures [38], use two separated *forward* and *reverse* direction

$$h_f^{(t)}(x) = \mathcal{F}_{\theta_f}(h_f^{(t-1)}(x), x_t) \tag{3}$$

$$h_r^{(t)}(x) = \mathcal{F}_{\theta_r}(h_r^{(t+1)}(x), x_t), \tag{4}$$

where θ_f and θ_r are independent of each other and together form the network parameters $\theta = \{\theta_f, \theta_r\}$. The final hidden state is obtained by concatenation of each direction hidden states,

$$h^{(t)}(x) = \left[h_f^{(t)}(x)^\top, h_r^{(t)}(x)^\top\right]^\top \in \mathbb{R}^{2n}. \tag{5}$$

The output of a *many to one* recurrent architecture is generally a linear transform of the last hidden state T:

$$f_\theta(x) = Vh^{(T)}(x) \in \mathbb{R}^d. \tag{6}$$

Many-to-one recurrent architecture refers to when the input is a sequence of data but the output is decided at the end. They are used in applications like sentiment analysis or time-series classification when the input is a sequence and the output is to decide which class it is. For recurrent architectures with average pooling [40], the output is a linear transform of the sum of all hidden states:

$$f_\theta(x) = \sum_{t=1}^{t} Vh^{(t)}(x) \in \mathbb{R}^d. \tag{7}$$

3 NV-RNN: INTERPRETABLE AND EXPLAINABLE RECURRENT NEURAL NETWORK

NV-RNN is inspired by the work in [4] except in that work the authors only focused on 2D CNNs. The work in [4] focuses on the spatial filters of CNNs whereas our work is adapted to use for RNNs. This paper focuses on RNNs with their variants and the hidden states per time. [4] could only be used for CNN architectures and we adapt it to the RNN architecture which is different since CNNs focus on spatial features while RNNs focus on temporal features. In addition, the applications to RNNs have not been explored to specific RNN architectures like bidirectional RNN or varying input length which scenario is not present with CNNs and image classification. Therefore, adapting the work

from [4] is non-trivial and brings merit into interpretability and explainability in the definitions that we denoted in the introduction.

3.1 NV Architecture Description

Given the sequences of hidden states calculated in Equation 1 for any recurrent network, the output from NV-RNN is obtained first by acquiring the hidden states for all time steps

$$q^{(t)}(x) = \text{ReLU}\left(h^{(t)}(x)\right), \tag{8}$$

where $\text{ReLU}(\alpha) = \max(\alpha, 0)$. In the next step, the output is calculated using

$$f_\theta(x) = \sum_{t=1}^{T} \left(V^{(t)}\right)^\top q^{(t)}(x) = \sum_{t=1}^{T} f^{(t)}(x) \in \mathbb{R}^d. \tag{9}$$

This is equivalent to concatenate all the hidden states per time step into a large hidden state vector $Q(x)$, where

$$Q(x) = \left[q^{(1)}(x)^\top, q^{(2)}(x)^\top, \ldots, q^{(T)}(x)^\top\right]^\top \in \mathbb{R}^{nT}, \tag{10}$$

and concatenate all linear output weights into one large matrix

$$V = \left[V^{(1)}, V^{(2)}, \ldots, V^{(T)}\right] \in \mathbb{R}^{nT \times d}. \tag{11}$$

and calculate the output as the following

$$f_\theta(x) = V^\top Q(x) \in \mathbb{R}^d. \tag{12}$$

Where unlike commonly used recurrent architectures, the NV-RNN concatenates all of the hidden states as the input to the linear classifier. This is different from a typical RNN where the last hidden state is the input to the linear classifier. This concatenation does increase the size of the input but is needed for both interpretability/explainability and the performance in Table 1.

Figure 1 depicts our NV-RNN where it displays how the hidden states are aggregated in order to provide a classification decision. Note that this network is applied to many-to-one applications.

The use of new weights for the linear classifier at each time step limits the NV-RNN to datasets where all data sequences have the same length. In the case where the data sequences have variable length, then zero padding will be used. This interpretability especially in terms of which time steps are resonating with the class helps explain which time steps are contributing the most. The entries of V will state in a numeric manner how much each weight is contributing to classification.

3.2 Interpretable and Explainable Linear Classifier

Recall that the final input to the linear classifier is the concatenation of the hidden states from all the time steps. When training is complete, we look at the weights per class and since we have the linear relationship of the linear classifier to all the hidden states, we have this ordered mapping of the weights to the hidden states. This ordering is done by concatenating the first hidden state starting from the first time step and concluding with the final time step's hidden state. Hence, we inspect which of the hidden states per time are contributing to the decision-making process for every class. Recall that from Equation 12 we have NV-RNN denoted in that form. Then we can substitute the right hand side
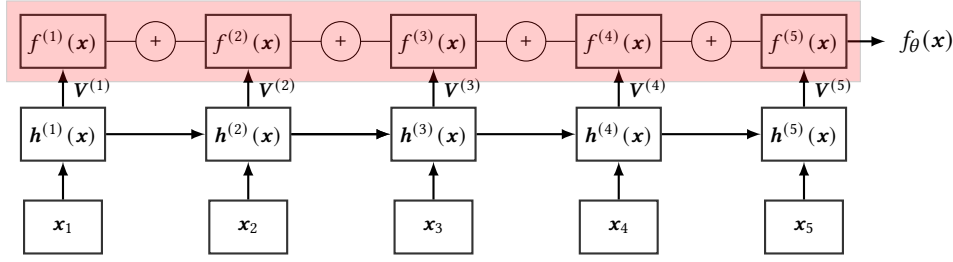
Fig. 1. Depiction of the NV-RNN framework. Every RNN can be converted to a NV model by having every hidden state concatenated to the linear classifier. The input to the linear classifier is the concatenation of all the hidden states. This provides a mapping to evaluate which time steps are the most relevant to each class.

to have in it in this new form (Equation 9). The classification depends on the concatenation of all the hidden states, Q, and the weights from the linear classifier. For each class of the output we get

$$[f_\theta(x)]_i = v_i^\top Q(x) \in \mathbb{R}, \quad \forall\, i \in [d], \tag{13}$$

where $v_i \in \mathbb{R}^{nT}$ is the ith row of V that corresponds to class i. Since each class output is obtained by inner product of $Q(x)$ and v_i, each class acquires a set of distinct learned v_i weights and we can interpret their values for each class. The goal is that in the training phase, the weights of V are generalized well for a test set and we can inspect which time steps are contributing the most by looking at the values. Each row of V corresponds to a class, so for every input that is sent, the softmax will choose the class with the highest score when each of the class's weights are multiplied by the time steps' hidden state activations.

In addition to interpretability, we provide explainability of the classification. For each class, their v explain the classification since $Q(x)$ is constant for each class and the determining factor are each class's v since the highest dot product from Equation 13 will determine which class is chosen.

This interpretability and explainability can provide more understanding of the training dynamics of NV-RNN. This provides additional analysis to why some models perform better than others. In Figures 2, 4, 5, and 7 we can see which time steps are critical based on their weight values from looking at each class's v.

3.3 Experimental Setup

We demonstrate the performance of NV-RNN, NV-GRU and NV-LSTM against other RNN, GRU, and LSTM architectures that are difficult to interpret/explain. The premise is that if we are on par or better than these models it is beneficial to use. The RNN models that are used besides RNN are GRU and LSTM. Also we monitor the effect of average pooling with RNNs, LSTMs, and GRUs in our comparison. The idea to benchmark against average pooling RNNs came from these works, [21, 47], since they have better performance than the standalone RNNs, LSTMS, and GRUs. Datasets that were used are the UCR repository [10], Large Movie Review [28], and UCF11 [25]. Do note that these tasks fall under the many-to-one regime for RNN classification. The UCR datasets had the input of the same length while the Large Movie Review dataset has varying input length. For UCF11, we use a sampler that we sample 50 sequential frames from each video. For the UCR and Large Movie Review datasets, we choose the best accuracy from each model where we vary the hidden state size to 32, 64, and 128. For UCF11, we use the hiddens state size of 32. All of the hyperparameters for the experiments are detailed in the Supplementary Material Section in A.2.

Table 1. The best performing NV-RNN, NV-GRU, and NV-LSTM models against the best performing RNN, GRU, and LSTM models. For most of the datasets, the NV-RNN, NV-GRU, and NV-LSTM models outperform especially on datasets that have more time steps.

Data set	RNN	GRU	LSTM	RNN-AVG	GRU-AVG	LSTM-AVG	NV-RNN	NV-GRU	NV-LSTM
Adiac	35.8%	37.08%	49.61%	10.23%	31.45%	16.87%	69.56%	68.28%	**74.68%**
BME	88.66%	94.66%	80%	84%	84.66%	84.66%	**99.3%**	98.66%	98.66%
CBF	60.66%	94.55%	90%	97.33%	98.66%	**99.77%**	97.77%	98.44%	98.55%
Chinatown	74.34%	97.37%	97.66%	**98.83%**	98.25%	98.54%	97.95%	97.08%	98.54%
Chlorine Concentration	58.17%	60.1%	57.73%	55.39%	57.05%	55.88%	**83.95%**	78.15%	72.39%
Fungi	49.46%	58.6%	68.81%	60.21%	58.6%	75.26%	96.77%	98.92%	**99.46%**
Ham	69.52%	68.57%	69.52%	74.28%	**81.9%**	80.95%	78.09%	80.95%	78.09%
Haptics	42.2%	41.55%	41.88%	33.76%	44.48%	41.88%	**46.42%**	46.1%	45.77%
Herring	67.18%	67.18%	68.75%	67.18%	65.62%	68.75%	68.75%	**73.43%**	68.75%
Insect Regular	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
Insect Small	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
InsectWingbeat	28.93%	49.34%	43.58%	28.48%	46.41%	39.94%	64.29%	64.54%	63.88%
Meat	48.33%	50%	50%	66.66%	86.66%	81.66%	**96.66%**	**96.66%**	**96.66%**
OliveOil	46.66%	50%	40%	40%	80%	40%	93.33%	93.33%	93.33%
Plane	89.52%	79.04%	95.23%	65.71%	98.09%	70.47%	99.04%	**100%**	99.04%
Rock	64%	74%	68%	56%	62%	60%	80%	76%	**82%**
SmoothSubspace	91.33%	89.33%	90.66%	90.66%	91.33%	86.66%	91.33%	**96%**	94%
Synthetic Control	**99.66%**	98.66%	98.33%	94.33%	95.66%	97.33%	**99.66%**	99.3%	99.3%
UMD	74.3%	99.3%	86.8%	75%	92.36%	72.22%	**100%**	**100%**	**100%**
Wine	59.25%	59.25%	62.96%	75.92%	79.62%	74.07%	**100%**	**100%**	**100%**

Table 2. The best performing NV-GRU models against the best performing non NV-GRU models with the Large Movie Review dataset. The dataset has sentences of reviews of variable length. NV-GRU performs close to on par despite zero padding.

Data set	Embedding	GRU	NV-GRU
Large Movie Review	Word2Vec	**91.87%**	90.12%
Large Movie Review	FastText	**91.46%**	89.56%
Large Movie Review	GloVe	**89.98%**	87.76%

Table 3. The best performing NV-CNN-RNN, NV-CNN-GRU, and NV-CNN-LSTM models against the best performing CNN-RNN, CNN-GRU, and CNN-LSTM models. The dataset is UCF11 where the task is video action recognition.

Data set	NV-CNN-GRU	NV-CNN-RNN	NV-CNN-LSTM	CNN-GRU	CNN-RNN	CNN-LSTM
UCF11	76.4%	74.3%	**78.3%**	69.3%	72.1%	72.4%

3.4 Results

We see from Table 1 that for most of the datasets, any of the NV-RNN, NV-GRU, and NV-LSTM models outperform the traditional RNN, GRU, and LSTM and its variants. For the exact accuracies among the different hidden state sizes look into the Supplementary Material Section A.3. It is interesting to note that with the datasets that have a smaller amount of timesteps, the RNNs will perform better compared to their gating variants. In addition, with the Adiac dataset, the results show that using average pooling on the hidden states is not the optimal solution. This is why NV-RNN was developed to have the linear classifier learn which of the time steps should have higher positive or negative weight values for the different classes. From Table 2 we apply NV-GRU to a dataset where the input has varying length and we perform on par. The reason is that the maximum length we set is 1000 and any reviews under that maximum length

will have zero padding. From Table 3, we outperform on another application called video action recognition. Note, that in this application, we have to use CNN filter units and RNN hidden state units as input to the linear classifier.

With the great performance, there is a cost associated with it. The cost is that the input for the linear classifier greatly increases. The input of the linear classifier size greatly increases since we concatenate all the hidden states from the input. This is in stark contrast from the input being the last hidden state or the averaging of the hidden states. Yet, even with the additional memory overhead, we can still run these NV-RNN, NV-GRU, and NV-LSTM models with an 8 GB GPU.

4 NV-RNN TIME ANALYSIS

4.1 Interpreting the Time Steps

Now with the experimental results showing that the NV-RNN, NV-GRU, and NV-LSTM models outperform RNNs, GRUs, and LSTMs among multiple datasets, the next step is to explain which time steps are pivotal for the classification portion. This is something that is lacking with the models that we compared. The weights from the linear classifier have a linear mapping between the different hidden states per time in order to directly observe which hidden states are responsible for each class's decision. Each class will have its own distribution of linear classifier weights. In the following section, we will observe different classes' linear weights for different datasets, different hidden state and weight initializations, and other case studies.

This interpretation is lacking from RNNs and their variants since the input to the linear classifier is the last hidden state. Then for the RNNs with average pooling, the notion of averaging the hidden states does not produce the best results as shown. Hence, by having all the hidden states as the input for the linear classifier, we learn the weights for each class to prioritize all the time steps. Each class will have a set of learnable weights that can be different from other classes. Figures 2, 4, and 5 show the weights from different NV-RNN models for different classes.

One dataset from Table 1 is the Chinatown dataset that has 2 classes and the number of time steps is 24. Figure 2 shows the NV-GRU weights for both classes. The weights for each class are drastically different from each other. This notion makes sense because for binary classification, the objective would be to have the weights drastically differ. The positive weights for class 0 become the negative weights for class 1. Figure 3 shows the individual hidden state weights for more granular information. Since this dataset only has 24 time steps, it is easy to view the individual weights for every hidden state as opposed to other datasets. With the input size being one, there is not a lot of activity with the hidden states for every time step. It is interesting that for every time step there seems to be a gradual change when looking at the nearby hidden states at neighboring time steps.

Another dataset from Table 4 is the Fungi dataset that has 18 classes and the number of time steps is 201. This dataset is very different from the Chinatown dataset, since we have multiple classes. Hence, one idea is to assess if there are weights from one class that is similar to another class and if there are weights from one class that is dissimilar to another class. To do this, we acquire the weights from each class and calculate the cosine similarity between all the classes. The cosine similarity is defined as $\frac{w_1^\top w_2}{||w_1||_2 ||w_2||_2}$ where w_1 is the weights of one class and w_2 is the weights of another class. In this scenario, class 5 is similar to class 16 while dissimilar to class 9. Figure 4 shows the weights for each of these classes. Based on the cosine similarity, it is easier to perceive why class 5 and class 16 are similar. Plus, it is easy to identify that class 5 and class 9 are dissimilar in their weights. Hence, if there is a prediction that was meant for class 5 but went to class 9, this notion makes sense by looking at the weights.

We show different weights from different classes of the same dataset, but now we want to assess if NV-RNN, NV-GRU, and NV-LSTM prioritize on different time steps. This is done by using the Chlorine Concentration dataset, which has 3 classes and 166 time steps. From Table 1, each of these NV models had different test accuracies. Figure 5 shows the different weights from these NV models for class 0. From Figure 5, all of these NV models have a similar positive trend for the last few time steps. However, the middle time steps are where each of these NV models starts to differ in prioritizing certain time steps. Hence, a benefit of the NV-RNN, NV-GRU, and NV-LSTM models is that you can interpret which time steps are critical based on the weight value.

We use three datasets on how to interpret the NV-RNN models after they are trained. In addition, they can also explain their predictions since for each class, the weight values are provided and linked to all of the hidden states. Hence, with each NV-RNN model there is a formal manner to explain the predictions since we can acquire the hidden state values for each time step and the weights for each class.
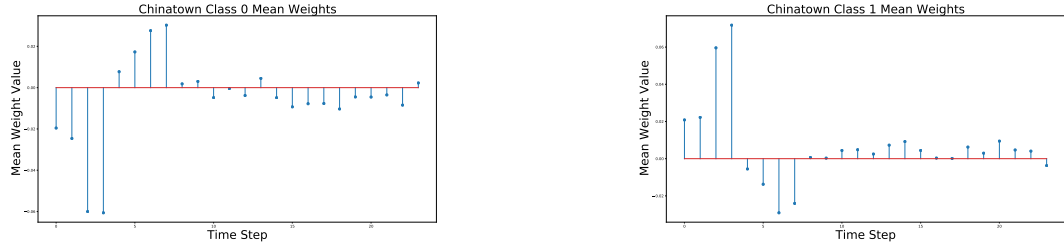

Fig. 2. (Left) NV-GRU weights for class 0. (Right) NV-GRU weights for class 1. For the Chinatown dataset, there are two classes and the weights for each class are drastically different. Hence each class's prediction focuses on different time steps. In regard to class 0, the most important time steps are 4, 5, 6, and 7. In regard to class 1, the most important time steps are 0, 1, 2, and 3.
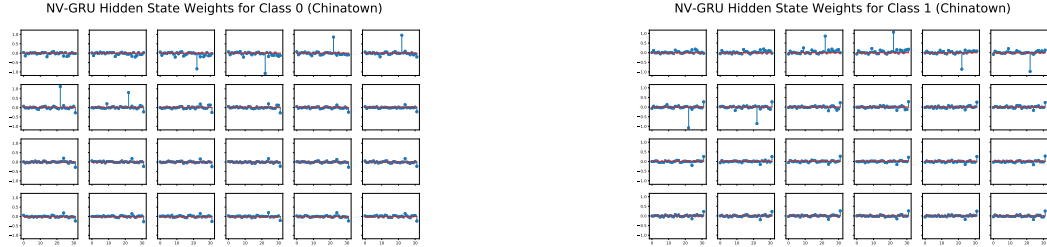
Fig. 3. (Left) NV-GRU hidden state weights for class 0. (Right) NV-GRU hidden state weights for class 1. For the Chinatown dataset, it has 24 time steps and the first hidden state starts at the top left and going from left to right where the last hidden state is in the bottom right.

4.2 Case Studies

We provide different case studies on how to use NV-RNN to inspect which time steps are prioritized within the application. With this insight, we observe how it provides additional understanding from the application.

Different Weight Initializations RNNs can be difficult to train. From [35], they state that early in the development of RNNs they would experience the vanishing gradient or exploding gradient problems. This led to the advent of GRUs
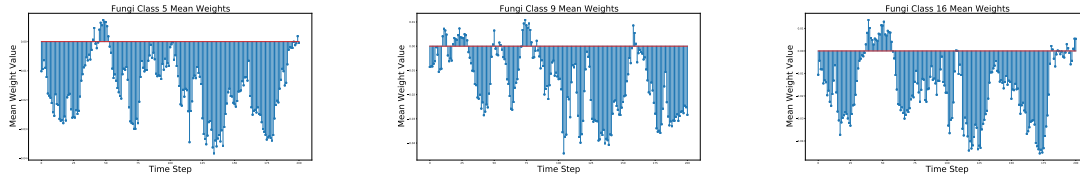
Fig. 4. (Left) NV-GRU weights for class 5. (Middle) NV-GRU weights for class 9. (Right) NV-GRU weights for class 16. For the Fungi dataset, there are a total of 18 classes. Compared to class 5's weights, class 9's weights are one of the most different based on the cosine similarity of the weights for each of these classes. For class 16, it is one of the most similar classes to class 5 based on the cosine similarity of the classes. Inspecting the NV-GRU weights from the linear classifier can aid in interpreting which classes are similar or dissimilar to each other.
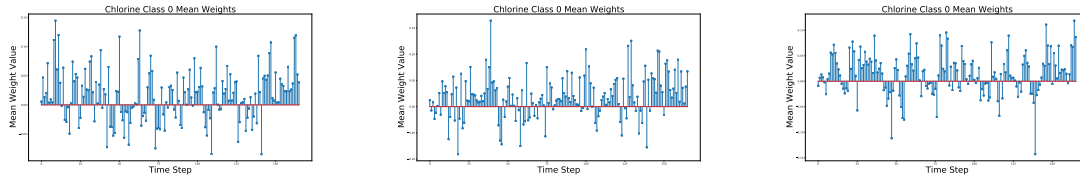

Fig. 5. (Left) NV-GRU weights for class 0. (Middle) NV-RNN weights for class 0. (Right) NV-LSTM weights for class 0. From Table 1, each of the NV models had different test accuracies and by inspecting each of the models, the prioritization of the time steps are different. The last time steps seem to have a similar trend for all of the NV models but the middle time steps have different prioritizations.

and LSTMs. Even with the new gating architectures of GRUs and LSTMs, they can experience issues of learning. This discovery leads others to figure out how to improve the performance. Others such as [22, 44] found that initializing the W weight matrix of the RNN can help in performance. Hence, we can use NV-RNN to inspect what is happening with the time step prioritization in regards to classification.

On the InsectWingbeat dataset, we perform an experiment to vary the weight initialization on the hidden-to-hidden matrix. This matrix is different from the V that is the linear classifier. We use the NV-GRU network to inspect how that can affect the decision-making process. The three different weight initializations are orthogonal, identity, and normal distributed. Figure 6 shows the three different weight initialization schemes and you can notice that the weight initialization scheme using a normal distribution is focusing on different time steps compared to the same NV-GRU model but with different weight initializations.

The two weight initializations, orthogonal and identity, look very similar in terms of time step prioritization with Figure 7. In Figure 7, it is the default weight initialization for NV-RNN, NV-GRU, and NV-LSTM, which is uniform initialization. Thus it is interesting that even with two different weight initializations, those weight initializations look similar to each other. Yet, the normal initialization is vastly different in regard to the time step prioritization. Plus with the NV-RNN model, it can explain why the performance is bad by looking at the time step weight prioritization. Hence, even if the NV-RNN model is performing in a sub-optimal manner, we can inspect why it is performing in that manner. Using a traditional RNN, GRU, or LSTM cannot provide this information.

Increasing the Depth (More Layers) [34] was one of the first works to take RNNs and make them deeper by stacking RNNs to increase the depth. In [34], unfortunately in their work, they did not show any interpretation towards how each depth aids in classification. There was one work [16] that used a deeper GRU for a chemical task. They
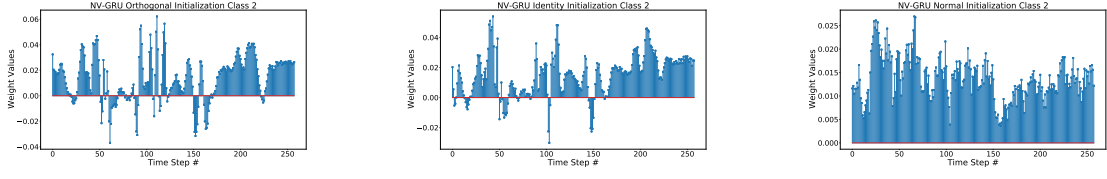
Fig. 6. (Left) Orthogonal. (Middle) Identity. (Right) Normal. With the same NV-GRU model but having 3 different weight initializations for W of the NV-GRU network. With different weight initializations, the distribution of the weights mapping to the time steps is shown to be different. Hence weight initialization is important because even with similar performance, by looking inside of the linear weights can provide insight into what the network is deciding for classification in regards to class 2.
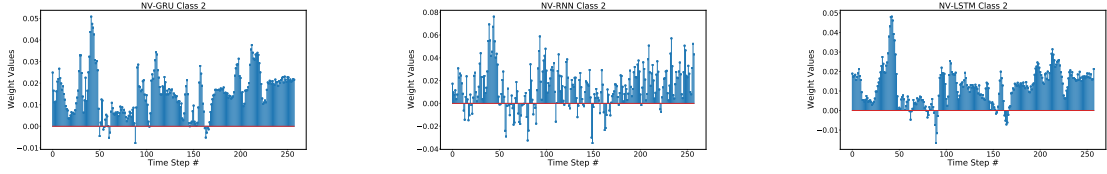

Fig. 7. (Left) NV-GRU weights across time. (Middle) NV-RNN weights across time. (Right) NV-LSTM weights across time. All three different NV-RNN models display the weights of time to show for this class which time steps are the most significant. NV-GRU and NV-LSTM seem to behave in a similar manner while NV-RNN has a different way of prioritizing the time steps.

provide interpretability through a mask but unfortunately it lacks sufficient detail in how each RNN in the deep RNN is aiding in classification. This is where NV-RNN can aid and show how each RNN (layer) is prioritizing the time steps in terms of classification.

From Table 1, we focused on one layer RNNs and NV-RNNs, but now we want to assess how increasing the depth of an NV-GRU model will show which time steps are prioritized. The question to inspect is if for every layer, will the time step prioritization be similar to the previous layers? Also, at a certain depth, will the time step prioritization be different from the previous layers? We can do this by looking at the weights of a NV-RNN model. In this analysis, we used the Rock dataset which has 2844 time steps and 4 classes. We have 4 different NV-GRU models where each one has varying depth. The first one starts at a depth of one and each additional model increases by an additional depth. Hence the last NV-GRU model has a depth of 4.

When comparing the test accuracy of the different depths of NV-GRU to GRU and average pooling GRU, the NV-GRU models would outperform the GRU and average pooling GRU. When testing the performance among different hidden states like 32, 64, and 128, NV-GRU would have a test performance of **76%**, **76%**, and **76%** for the depths of 2,3,4. This is the same performance as noted for a one-layer NV-GRU for the same dataset. For the GRU and average pooling GRU, the best test performance for depths of 2, 3, and 4 were 68%, 72%, and 66%. Thus, for different depths, NV-GRU is outperforming.

Figure 8 shows the time step weights for four different NV-GRU models with different depths. With NV-GRU, we can answer that question in a quantitative manner if the time step prioritization is similar among the different layers of the NV-GRU model. The answer is no and this makes sense since the input to the next layer is the previous layer's representation. For some time steps, there is some manner of consistency like with the NV-GRU models of 2, 3, or 4 layers where the beginning time steps are close to zero. Then the time step prioritization will vary towards the end of the time steps.

Bidirectional RNNs There are some RNNs that were developed that are bidirectional such that the RNN will learn in two directions. One direction is in a causal manner (forward) going from the beginning of the input to the end of it. While the other direction is in a non-causal manner (reverse) where it begins at the end of the input and ends at the beginning of the input. Equation 5 details how an RNN will use both the forward and reverse hidden states. Thus the aspect for NV-RNN is to adapt it to the bidirectional variant and to inspect the weights. By inspecting the weights, we can assess how the time steps are prioritized in either direction. One question to answer is if the time steps of either direction will be symmetrical to each other?

There are works from [26, 39] that have used bidirectional RNNs to aid performance in their respective task. For interpretability, they use the attention weights to show how a given input is being classified. Yet in their interpretability, they cannot provide how each direction in the RNN is aiding for the classification. This lack of directional interpretability is where NV-RNN will help and can answer the questions mentioned above.

For the bidirectional analysis, we use an NV-GRU model that allows the bidirectional nature. We use the Rock dataset which has 2844 time steps and 4 classes.

When comparing a bidirectional NV-GRU with a bidirectional GRU and average pooling GRU, the bidirectional NV-GRU outperformed in test accuracy. Among the different hidden state sizes of 32, 64, and 128, the best test accuracy for bidirectional NV-GRU is **76%** while for GRU is 72% and average pooling GRU is 66%. Even for the bidirectional variant, NV-GRU is outperforming.

Figure 9 shows the time step prioritizations for the bidirectional NV-GRU. With this NV-GRU model, we can assess if the time steps prioritized from both directions will be symmetrical. The answer is that there is evidence that we do not see this notion and we can see that in a visual and quantitative manner (Figure 9). Looking at the vertical columns for each direction, each direction does not have the same prioritization.

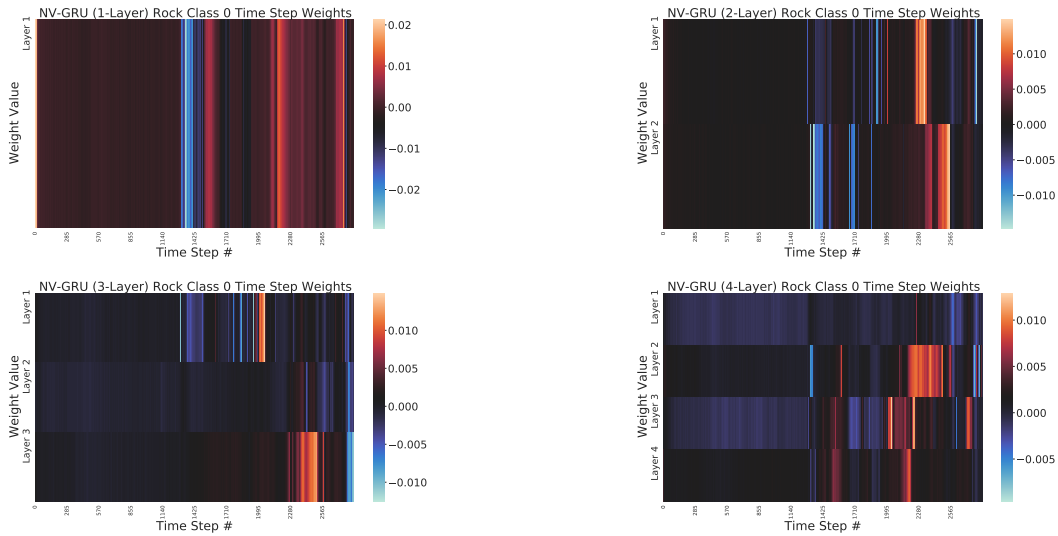Fig. 8. (Top Left) NV-GRU weights for class 0. (Top Right) NV-GRU (2 Layers) weights for class 0. (Bottom Left) NV-GRU (3 Layers) weights for class 0. (Bottom Right) NV-GRU (4 Layers) weights for class 0. By increasing the number of layers for the NV-GRU model, the prioritization of time steps is not consistently the same among all the layers. Each vertical column is the weight value at that time step.
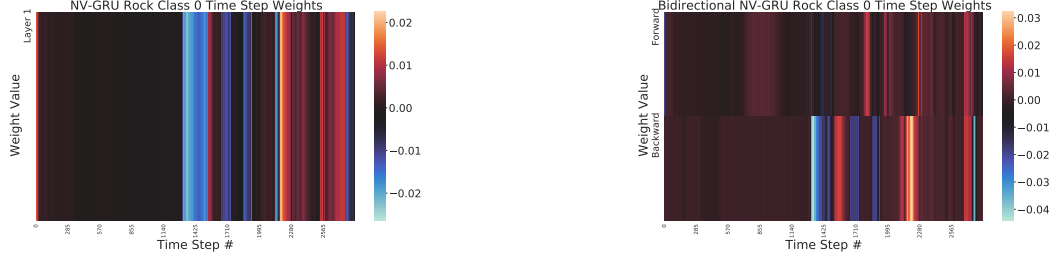
Fig. 9. (Left) NV-GRU weights for class 0. (Right) Bidirectional NV-GRU weights for class 0. The forward and reverse prioritizations of time steps looks very similar. Each vertical column is the weight value at that time step.

Sentiment Analysis The previous datasets that were shown consisted of time-series data. Now we will use a movie review dataset [28] with three embedding techniques, Word2Vec [30], Fasttext [6], and GloVe [36], to convert each word into an embedding vector. Since NV-RNN is versatile, we can inspect how the weights for each class are prioritized for text data. In this dataset, the task is sentiment classification where one class is positive sentiment and the second class is negative sentiment.

Other works have provided different forms of understanding the differences of these embedding techniques. [41] show the similarities of GloVe and a skip-gram Word2Vec in a mathematical manner. [33] use a couple of evaluation tasks like correlation to show the differences or similarities of these embedding techniques. [42] uses eigenvector analysis to compare the embedding vectors to assess how the words cluster together. Yet, all of these techniques cannot link the time steps to the class, which is what NV-RNN will show.

Table 2 shows the performance and in this scenario, NV-GRU performs on par. Note that for this dataset, there are reviews of variable length so for a GRU the last hidden state per review is the input to the linear classifier. However, for NV-GRU, padding has to be applied since the input for the linear classifier has to be of fixed size. Hence, it does explain the small drop in performance. As for padding, we only used the first 1,000 words of the review. With reviews that are smaller than 1,000 words, zero padding would be applied. Hence, there can be a good parameter to pad the reviews. Even with this disadvantage, NV-GRU is still on par and only loses one to two percent of test accuracy.

Figures 10, 11, and 12 display the weights for each class. Even though there are three different embedding techniques, Word2Vec, FastText, and GloVe, it seems that the prioritization for the time steps in each embedding looks to be quite similar in the broad general sense. For the negative sentiment class, it is interesting how with all the embedding inputs, the last time steps are negative. This does make sense since FastText is a continuation of Word2Vec except for the case that FastText will approximate words that are not in the dictionary of words that it had learned. It is interesting how the GloVe embedding looks to have similar broad time step prioritization since it is a matrix factorization technique compared to Word2Vec and FastText. There are very specific differences based on the magnitudes but in a general manner all the embedding techniques for the input will result in the same general trend for each class in terms of the weights.

From Figures 10 and 11, we provide another experiment to see which words are tied to the highest weights based on the time step index. To reduce the amount of reviews, we only look at the top 5 reviews for each class based on the pre-softmax score. In addition, we look at the top 10 weights per class. When inspecting the words for each class, we find pronouns and prepositions. However, one particular notion we inspect is particular words for each class. For the negative sentiment class, we observe the words, **bad**, **dodgy**, **unpleasant**, and **pointless** from the time steps with the

highest weights. For the positive sentiment class, we observe the words, **mature**, **happy**, **perfect**, and **top**. We would hope to observe this notion because if the class is centered around negative sentiments, then the words that should be linked to the class would be bad words. The same notion applies to the positive sentiment class. Also, we did observe that for the positive sentiment class, we did not notice any negative sentiment words like bad or terrible.

Fig. 10. (Left) NV-GRU weights for class 0. (Right) NV-GRU weights for class 1. For the Movie Review dataset, there are two classes and the weights for each class are drastically different. In addition, the input is the Word2Vec embedding of each word per time step. Compared to continuous time-series data, the prioritization of time steps is not as smooth compared to datasets with continuous data.
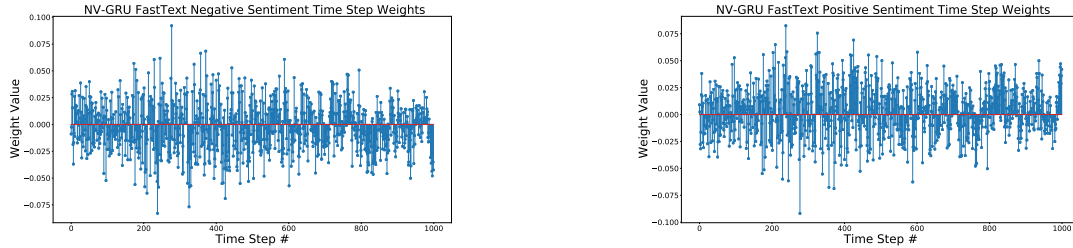
Fig. 11. (Left) NV-GRU weights for class 0. (Right) NV-GRU weights for class 1. For the Movie Review dataset, there are two classes and the weights for each class are drastically different. In addition, the input is the FastText embedding of each word per time step. Compared to continuous time-series data, the prioritization of time steps is not as smooth compared to datasets with continuous data.

Video Action Recognition In this application, the input to the network is a collection of images from a video. The task is to predict which of the actions is presented in the video where the dataset is UCF11 [25]. In Table 3, we show that the NV models are outperforming and we inspect the NV-CNN-GRU model. Note that in this scenario we are concatenating both the CNN filter units and the GRU hidden states.

There have been works such as [11, 29] to provide interpretability with video action recognition with CNN-LSTM models. Yet, one of the main issues is that they cannot explain which part, the CNN or LSTM is contributing to the classification. This is where NV-CNN-GRU is able to provide both interpretability and explainability.

In Figure 13, we see that the mean positive weights are towards the CNN filter units as opposed to the GRU hidden states. This is interesting to notice that most of the hidden state time step mean weights are negative. This does make sense since the model is sequential and the NV-GRU is depending on the CNN's features. In addition, the dataset sampler is sampling 50 sequential frames and most of the videos contain more than 50 frames. Thus, the impression is that the

NV-GRU GloVe Negative Sentiment Time Step Weights

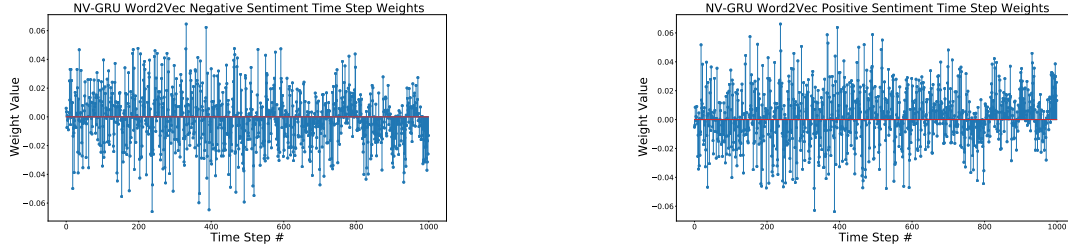NV-GRU GloVe Positive Sentiment Time Step Weights
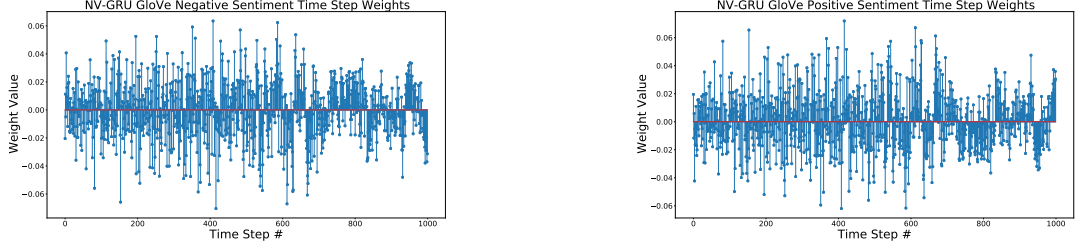
Fig. 12. (Left) NV-GRU weights for class 0. (Right) NV-GRU weights for class 1. For the Movie Review dataset, there are two classes and the weights for each class are drastically different. In addition, the input is the GloVe embedding of each word per time step. Compared to continuous time-series data, the prioritization of time steps is not as smooth compared to datasets with continuous data.

hidden states may not be providing as much useful information as the CNN. From this work, [24], there is evidence that video action recognition datasets tend to have visual bias so it does make sense that with the NV-CNN-GRU model, it has more positive weights towards the spatial information. Given this information, there could be future work to mitigate the CNN/GRU prioritization while retaining the accuracy. Ideas from this work, [32], could aid in mitigating the prioritization.

UCF11 Class 5 NV-CNN-GRU First Layer Weights

UCF11 Class 5 NV-CNN-GRU Second Layer Weights

UCF11 Class 5 NV-CNN-GRU Third Layer Weights

UCF11 Class 5 NV-CNN-GRU Time Step Weights
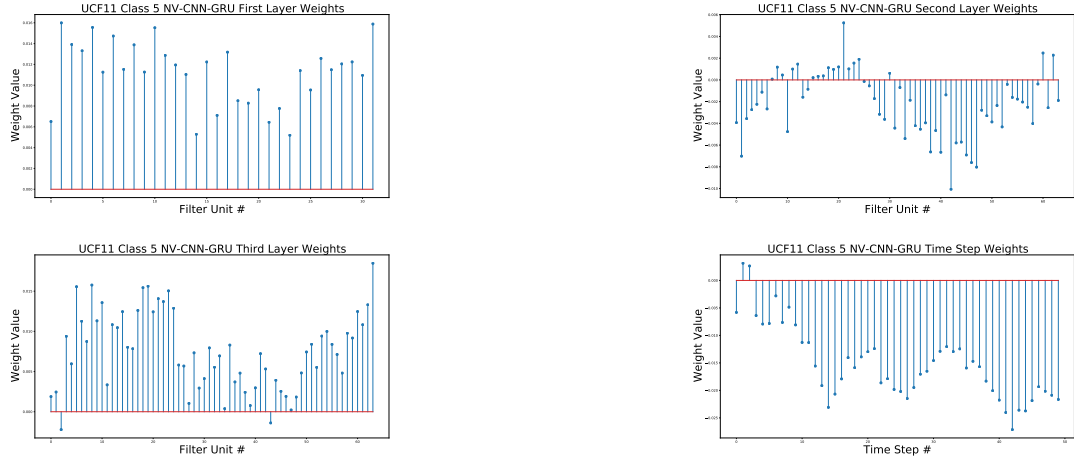
Fig. 13. Time step prioritization and filter unit prioritization (Class 5) for the NV-CNN-GRU model for UCF11. For this model, the positive mean weights are for the first layer of the CNN portion. Here the time step mean weights are negative.

Counterfactuals There have been works [5, 7, 37, 45] in using counterfactuals to understand how the RNN is performing. [5, 7, 37] use the counterfactuals for regression applications with RNNs. While [45] uses the counterfactuals for classification applications with RNNs.

From the interpretability and explainability that NV-RNN provides, we can create counterfactuals called *Time Analysis*. With each class having a unique set of weights per time step, we look at the mean hidden state weights per time step and set the top **K** time steps to zero. The idea is that the performance should drop if the information from the time-series

Table 4. Performance of the model with Time Analysis. From NV-GRU we inspect which time steps contribute to each class by their weight value. In the Chinatown dataset, we remove the top weights for each class and calculate the overall class accuracy. Immediately, the test accuracy drops but then will come back up.

Network	# of Time Steps	Test Accuracy
NV-GRU	0	96.4
NV-GRU	1	72.7
NV-GRU	5	43.87
NV-GRU	10	89.5

data related to the time step is omitted. it will make it harder for the classifier to perform adequately. We set the top $\mathbf{K}$ time steps to zero to evaluate the degradation of the per-class accuracy.

For Time Analysis, we utilize the information we learned from Figure 2, which shows that for each class there were about 5 top positive time steps. In Table 4, this confirms that if we set those time steps to 0, then the class accuracy will drop. The interesting aspect is that after we remove more time steps, the class accuracy will increase but never get to the level of the original test accuracy. Note that this accuracy is for one of the NV-GRU models that did not achieve the best accuracy listed in Table 1. The reason for this dip is that we are now eliminating the negative time steps so it will affect the classification decision. Additional experiments are in the Supplementary Material Section in A.4 where there are individual class performance results.

5 CONCLUSION

We propose a novel model, NV-RNN, as an alternative to traditional RNNs that has superior to on par performance to RNNs under multiple datasets. NV-RNN can provide interpretability and explain the prediction in a mathematical formulation. NV-RNN has the potential to be used in a wide array of different RNN applications. With its generic framework, it is used to show the connection between all of the hidden states and the classification. Thus, there are other scenarios within the scope of RNNs where it can provide additional understanding where typical RNNs are unable to provide. Overall, this type of interpretability and explainability is helpful to understand what is happening. In addition, with the concatenation of all the hidden states, it enables us to understand more and in most cases have higher performance.

REFERENCES

[1] S. Alemohammad, R. Balestriero, Z. Wang, and R. G. Baraniuk. 2020. Scalable Neural Tangent Kernel of Recurrent Architectures. *arXiv preprint arXiv:2012.04859* (2020).

[2] S. Alemohammad, Z. Wang, R. Balestriero, and R.G. Baraniuk. 2020. The recurrent neural tangent kernel. *arXiv preprint arXiv:2006.10246* (2020).

[3] L. Arras, J. Arjona-Medina, M. Widrich, G. Montavon, M. Gillhofer, K.R. Müller, S. Hochreiter, and W. Samek. 2019. Explaining and interpreting LSTMs. In *Explainable ai: Interpreting, explaining and visualizing deep learning*. Springer, 211–238.

[4] C. Barberan, R. Balestriero, and R. G Baraniuk. 2021. NeuroView: Explainable Deep Network Decision Making. *arXiv preprint arXiv:2110.07778* (2021).

[5] I. Bica, A. M. Alaa, J. Jordon, and M. van der Schaar. 2020. Estimating counterfactual treatment outcomes over time through adversarially balanced representations. *arXiv preprint arXiv:2002.04083* (2020).

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics* 5 (2017), 135–146.

[7] G. Chen, J. Li, J. Lu, and J. Zhou. 2021. Human Trajectory Prediction via Counterfactual Analysis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 9824–9833.

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[9] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).

[10] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Y. Chen, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and M. L. Hexagon. 2019. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[11] Y. Dong, H. Su, J. Zhu, and B. Zhang. 2017. Improving interpretability of deep neural networks with semantic information. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4306–4314.

[12] Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14, 2 (1990), 179–211. https://www.sciencedirect.com/science/article/pii/036402139090002E

[13] J.N. Foerster, J. Gilmer, J. Sohl-Dickstein, J. Chorowski, and D. Sussillo. 2017. Input switched affine networks: An rnn architecture designed for interpretability. In *International Conference on Machine Learning (ICML)*. PMLR, 1136–1145.

[14] H. Gammulle, S. Denman, S. Sridharan, and C. Fookes. 2017. Two stream lstm: A deep fusion framework for human action recognition. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 177–186.

[15] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal. 2018. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*. IEEE, 80–89.

[16] G. B. Goh, N. O. Hodas, C. Siegel, and A. Vishnu. 2017. Smiles2vec: An interpretable general-purpose deep neural network for predicting chemical properties. *arXiv preprint arXiv:1712.02034* (2017).

[17] C. Guan, X. Wang, Q. Zhang, R. Chen, D. He, and X. Xie. 2019. Towards a deep and unified understanding of deep neural models in nlp. In *International Conference on Machine Learning (ICML)*. PMLR, 2454–2463.

[18] T. Guo, T. Lin, and N. Antulov-Fantulin. 2019. Exploring interpretable LSTM neural networks over multi-variable data. In *International Conference on Machine Learning (ICML)*. PMLR, 2494–2504.

[19] S. Hochreiter and J. Schmidhuber. 1997. Long short-term memory. *Neural Computation (Neural Comput.)* 9, 8 (1997), 1735–1780.

[20] C. Jiang, Y. Zhao, S. Chu, L. Shen, and K. Tu. 2020. Cold-start and Interpretability: Turning Regular Expressions into Trainable Recurrent Neural Networks. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 3193–3207.

[21] C.-C. Kao, M. Sun, W. Wang, and C. Wang. 2020. A comparison of pooling methods on LSTM models for rare acoustic event classification. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 316–320.

[22] Q. V. Le, N. Jaitly, and G. E. Hinton. 2015. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941* (2015).

[23] A. Li, K.and Javer, E. E. Keaveny, and A.E.X. Brown. 2017. Recurrent neural networks with interpretable cells predict and classify worm behaviour. *BioRxiv* (2017), 222208.

[24] Y. Li, Y. Li, and N. Vasconcelos. 2018. Resound: Towards action recognition without representation bias. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 513–528.

[25] J. Liu, J. Luo, and M. Shah. 2009. Recognizing realistic actions from videos "in the wild". In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 1996–2003.

[26] F. Ma, R. Chitta, J. Zhou, Q. You, T. Sun, and J. Gao. 2017. Dipole: Diagnosis prediction in healthcare via attention-based bidirectional recurrent neural networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 1903–1911.

[27] Z. Ma and Z. Sun. 2018. Time-varying LSTM networks for action recognition. *Multimedia Tools and Applications (Multimed. Tools. Appl)* 77, 24 (2018), 32275–32285.

[28] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. http://www.aclweb.org/anthology/P11-1015

[29] L. Meng, B. Zhao, B. Chang, G. Huang, W. Sun, F. Tung, and L. Sigal. 2019. Interpretable spatio-temporal attention for video action recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*. 0–0.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[31] Y. Ming, P. Xu, H. Qu, and L. Ren. 2019. Interpretable and steerable sequence learning via prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (ACM SIGKDD)*. 903–913.

[32] J. Nam, H. Cha, S. Ahn, J. Lee, and J. Shin. 2020. Learning from failure: Training debiased classifier from biased classifier. *arXiv preprint arXiv:2007.02561* (2020).

[33] A. Nematzadeh, S. C Meylan, and T. L Griffiths. 2017. Evaluating Vector-Space Models of Word Representation, or, The Unreasonable Effectiveness of Counting Words Near Other Words.. In *CogSci*.

[34] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio. 2013. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026* (2013).

[35] R. Pascanu, T. Mikolov, and Y. Bengio. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*. PMLR, 1310–1318.

[36] J. Pennington, R. Socher, and C. D Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[37] J. Poulos and S. Zeng. 2021. RNN-based counterfactual prediction, with an application to homestead policy and public schooling. *Journal of the Royal Statistical Society: Series C (Applied Statistics)* 70, 4 (2021), 1124–1139.

[38] M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing (IEEE Trans. Signal Process)* 45, 11 (1997), 2673–2681.

[39] P. Schwab, G. C. Scebba, J. Zhang, M. Delai, and W. Karlen. 2017. Beat by beat: Classifying cardiac arrhythmias with recurrent neural networks. In *2017 Computing in Cardiology (CinC)*. IEEE, 1–4.

[40] L. Shen and J. Zhang. 2016. Empirical evaluation of RNN architectures on sentence classification task. *arXiv preprint arXiv:1609.09171* (2016).

[41] T. Shi and Z. Liu. 2014. Linking GloVe with word2vec. *arXiv preprint arXiv:1411.5595* (2014).

[42] J. Shin, A. Madotto, and P. Fung. 2018. Interpreting word embeddings with eigenvector analysis. In *32nd Conference on Neural Information Processing Systems (NuerIPS 2018), IRASL workshop*.

[43] J. Song, Y. Guo, L. Gao, X. Li, A. Hanjalic, and H. T. Shen. 2018. From deterministic to generative: Multimodal stochastic RNNs for video captioning. *IEEE Transactions on Neural Networks and Learning Systems (IEEE Trans. Neural Netw. Learn. Syst)* 30, 10 (2018), 3047–3058.

[44] S.S. Talathi and A. Vartak. 2015. Improving performance of recurrent neural network with relu nonlinearity. *arXiv preprint arXiv:1511.03771* (2015).

[45] S. Tonekaboni, S. Joshi, D. Duvenaud, and A. Goldenberg. 2019. Explaining time series by counterfactuals. (2019).

[46] C. Wang, H. Yang, C. Bartz, and C. Meinel. 2016. Image captioning with deep bidirectional LSTMs. In *Proceedings of the 24th ACM International Conference on Multimedia (Proc ACM Int Conf Multimed)*. 988–997.

[47] Y. Wang, J. Li, and F. Metze. 2019. A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 31–35.

A   SUPPLEMENTARY MATERIAL

A.1   Architecture descriptions

In this section, we provide the recursive block computation of other recurrent architecture variants that has been used in this paper. To convert each model to it's NeuroView version, we concatenate all hidden states $\boldsymbol{h}^{(t)}(\boldsymbol{x})$ and calculate the output using as described in Equation 12.

A.1.1   Gated Recurrent Unit (GRU).

$$\boldsymbol{r}^{(t)}(\boldsymbol{x}) = \phi(\boldsymbol{W}_{ir}\boldsymbol{x}_t + \boldsymbol{b}_{ir} + \boldsymbol{W}_{hr}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{hr}) \tag{14}$$

$$\boldsymbol{z}^{(t)}(\boldsymbol{x}) = \phi(\boldsymbol{W}_{iz}\boldsymbol{x}_t + \boldsymbol{b}_{iz} + \boldsymbol{W}_{hz}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{hz}) \tag{15}$$

$$\boldsymbol{n}^{(t)}(\boldsymbol{x}) = \tanh(\boldsymbol{W}_{in}\boldsymbol{x}_t + \boldsymbol{b}_{in} + \boldsymbol{r}^{(t)}(\boldsymbol{x}) \odot (\boldsymbol{W}_{hn}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{hn})) \tag{16}$$

$$\boldsymbol{h}^{(t)}(\boldsymbol{x}) = (1 - \boldsymbol{z}^{(t)}(\boldsymbol{x})) \odot \boldsymbol{n}^{(t)}(\boldsymbol{x}) + \boldsymbol{z}^{(t)}(\boldsymbol{x}) \odot \boldsymbol{z}^{(t-1)}(\boldsymbol{x}), \tag{17}$$

where $\phi(\alpha) = \frac{1}{1+e^{(-\alpha)}}$ is the sigmoid function and $\odot$ is the Hadamard product.

A.1.2   Long Short-Term Memory (LSTM) .

$$\boldsymbol{i}^{(t)}(\boldsymbol{x}) = \phi(\boldsymbol{W}_{ii}\boldsymbol{x}_t + \boldsymbol{b}_{ii} + \boldsymbol{W}_{hi}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{hi}) \tag{18}$$

$$\boldsymbol{f}^{(t)}(\boldsymbol{x}) = \phi(\boldsymbol{W}_{if}\boldsymbol{x}_t + \boldsymbol{b}_{if} + \boldsymbol{W}_{hf}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{hf}) \tag{19}$$

$$\boldsymbol{g}^{(t)}(\boldsymbol{x}) = \tanh(\boldsymbol{W}_{ig}\boldsymbol{x}_t + \boldsymbol{b}_{ig} + \boldsymbol{W}_{hg}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{hg}) \tag{20}$$

$$\boldsymbol{o}^{(t)}(\boldsymbol{x}) = \phi(\boldsymbol{W}_{io}\boldsymbol{x}_t + \boldsymbol{b}_{io} + \boldsymbol{W}_{ho}\boldsymbol{h}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{b}_{ho}) \tag{21}$$

$$\boldsymbol{c}^{(t)}(\boldsymbol{x}) = \boldsymbol{f}^{(t)} \odot \boldsymbol{c}^{(t-1)}(\boldsymbol{x}) + \boldsymbol{i}^{(t)} \odot \boldsymbol{g}^{(t)}(\boldsymbol{x}) \tag{22}$$

$$\boldsymbol{h}^{(t)}(\boldsymbol{x}) = \boldsymbol{o}^{(t)}(\boldsymbol{x}) \odot \tanh(\boldsymbol{c}^{(t)}(\boldsymbol{x})) \tag{23}$$

A.2   Hyperparameter Optimization

From Table 1, the results were taking the best model among different hidden states sizes which were 32, 64, and 128. The optimizer used is Adam. The learning rate is 0.001. The number of epochs is 1000.

For Large Movie Review dataset, we first preprocess the text by converting it to lowercase, removing all numbers, punctuation and special characters. Then for Word2Vec and Fasttext embedding method, we use dimensionality of 100 for word vectors. We count all words that appear at least once into our vocabulary, resulting in a vocabulary of size 122,762. Within the model training, the maximum distance between the current and predicted word within a sentence is 5. Both models are trained with the movie review dataset itself without any pre-training process. For GloVe embedding, we use a pre-trained model **GloVe.6B** whose embedding is trained on *Wikipedia 2014* and *Gigaword 5th Edition corpora* with 6 billion word tokens and 400,000 vocabulary size. We use the word embedding dimension of 100 so that it is consistent with other embedding methods.

In training the models in Table 2, the three different hidden state sizes used were 32, 64, and 128. The models were trained for 20 epochs. The optimizer used is Adam. The learning rate is 0.001. The batch size is 100.

In training the models in Table 3, the hidden state size is set to 32. The number of epochs is 100. The learning rates for NV-CNN-RNN, NV-CNN-GRU, and NV-CNN-LSTM were set to 0.001 while the learning rates for CNN-RNN, CNN-GRU, and CNN-LSTM were set to 0.0001. The CNN architecture used was a 3 layer CNN with each CNN layer having a

max-pooling layer placed after the CNN layer. The first CNN layer had an input of 3 channels and an output of 32 channels with a kernel size of 3 and padding of 1. The second CNN layer had an input of 32 channels and an output of 64 channels with a kernel size of 3 and a padding of 1. The third CNN layer had an input of 64 channels and an output of 64 channels with a kernel size of 3 and padding of 1. The activation function used is a ReLU. Every max pooling parameter would pool by a factor of 2. The input size for all the RNNs, GRUs, and LSTMs would be 28*28*64.

A.3 Ablation Studies

We conduct ablation studies for different hidden state dimension of 32, 64, 128 for all the models mentioned in Table 1. The results can be found in Table 5, Table 6 and Table 7.

Table 5. An ablation study of NV-RNN, NV-GRU, and NV-LSTM models with hidden state sizes of 32, 64, 128.

Data set	NV-GRU32	NV-GRU64	NV-GRU128	NV-RNN32	NV-RNN64	NV-RNN128	NV-LSTM32	NV-LSTM64	NV-LSTM128
Adiac	**68.28%**	64.45%	68.03%	68.79%	64.96%	**69.56%**	68.03%	71.35%	**74.68%**
BME	**98.66%**	**98.66%**	**98.66%**	98.66%	**99.33%**	98.66%	**98.66%**	98%	**98.66%**
CBF	95.66%	98%	**98.44%**	**97.77%**	96%	97.44%	98.11%	98.22%	**98.55%**
Chinatown	96.5%	**97.08%**	**97.08%**	**97.95%**	97.08%	**97.95%**	**98.54%**	**98.54%**	98.25%
Chlorine Concentration	74.58%	75.44%	**78.15%**	80.33%	82.21%	**83.95%**	68.82%	69.01%	**72.39%**
Fungi	**98.92%**	97.84%	96.23%	**96.77%**	96.23%	94.08%	98.92%	**99.46%**	98.38%
Ham	**80.95%**	78.09%	79.04%	78.09%	75.23%	**78.09%**	77.14%	77.14%	**78.09%**
Haptics	**46.1%**	45.77%	45.45%	**47.07%**	45.12%	46.42%	45.45%	**45.77%**	44.8%
Herring	71.87%	**73.43%**	68.75%	68.75%	68.75%	68.75%	68.75%	68.75%	67.18%
InsectRT	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
InsectST	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
InsectWingbeat	**64.54%**	64.39%	64.04%	63.98%	64.19%	**64.29%**	63.58%	**63.88%**	63.73%
Meat	91.66%	**96.66%**	95%	95%	93.33%	**96.66%**	**96.66%**	93.33%	93.33%
OliveOil	**93.33%**	**93.33%**	90%	90%	**93.33%**	**93.33%**	90%	**93.33%**	**93.33%**
Plane	98.09%	98.09%	**100%**	**99.04%**	98.09%	97.14%	97.14%	**99.04%**	**99.04%**
Rock	68%	70%	**76%**	76%	**80%**	72%	**82%**	74%	80%
SmoothSubspace	**96%**	94.66%	95.33%	90.66%	**91.33%**	90.66%	**94%**	90%	90%
Synthetic Control	**99.33%**	98.66%	98.66%	99.33%	98.33%	**99.66%**	**99.33%**	98.33%	99%
UMD	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**
Wine	**100%**	98.14%	**100%**	**100%**	**100%**	**100%**	**100%**	**100%**	96.29%

A.4 Additional Counterfactuals

Table 8 shows the results of using Time Analysis on the test dataset. By focusing on one class at a time, we can quantitatively assess how perturbing the time steps within the data can affect the performance. With one time step modified, the accuracy dropped by one percent. When we started to set more of the input data at certain time steps to zero, the accuracy would continue to drop.

One interesting observation is to see which time steps have a negative mean value and to set the most negative weights to zero. Table 9 displays the test accuracy for the negative Time Analysis. The test accuracy will remain the same, but when ten input data time steps are set to zero the test accuracy of that class has increased. Hence the interpretability of this model can provide these insights and practitioners can develop test cases for their data to understand what happens if you perturb the data to certain degrees.

Table 6. An ablation study of RNN, GRU, and LSTM models with hidden state sizes of 32, 64, 128.

Data set	GRU32	GRU64	GRU128	RNN32	RNN64	RNN128	LSTM32	LSTM64	LSTM128
Adiac	30.69%	36.82%	37.08%	31.2%	35.8%	33.75%	39.13%	49.61%	48.33%
BME	93.33%	94.66%	92.66%	77.33%	88.66%	74%	80%	76.66%	78%
CBF	76.44%	81.66%	94.55%	60.66%	56.66%	57.22%	90%	87.11%	84.55%
Chinatown	96.79%	97.37%	97.37%	74.34%	72.59%	72.01%	97.66%	97.66%	97.66%
Chlorine Concentration	58.07%	60.1%	59.74%	56.48%	57.16%	58.17%	56.71%	57.31%	57.73%
Fungi	43.54%	50.53%	58.6%	49.46%	47.31%	46.23%	45.16%	67.2%	68.81%
Ham	68.57%	67.61%	68.57%	68.57%	69.52%	69.52%	69.52%	69.52%	67.61%
Haptics	41.23%	41.55%	40.25%	37.33%	40.58%	42.2%	38.31%	41.23%	41.88%
Herring	65.62%	65.62%	67.18%	65.62%	67.18%	67.18%	68.75%	65.62%	67.18%
InsectRT	100%	100%	100%	100%	100%	100%	100%	100%	100%
InsectST	100%	100%	100%	100%	100%	100%	100%	100%	100%
InsectWingbeat	44.34%	48.98%	49.34%	26.61%	28.93%	27.87%	38.68%	43.58%	28.73%
Meat	45%	45%	50%	48.33%	48.33%	45%	50%	46.66%	41.66%
OliveOil	50%	40%	40%	46.66%	40%	40%	40%	40%	40%
Plane	68.57%	67.61%	79.04%	59.04%	62.85%	89.52%	87.61%	91.42%	95.23%
Rock	68%	74%	64%	64%	62%	60%	62%	68%	66%
SmoothSubspace	89.33%	89.33%	89.33%	91.33%	90.66%	90.66%	88.66%	90%	90.66%
Synthetic Control	98%	98.66%	97.33%	79%	98%	99.66%	96.33%	98.33%	98.33%
UMD	66.66%	88.88%	99.3%	74.3%	65.97%	66.66%	65.27%	86.8%	67.36%
Wine	59.25%	55.55%	55.55%	57.4%	59.25%	50%	62.96%	53.7%	59.25%

Table 7. An ablation study of RNN-AVG, GRU-AVG, and LSTM-AVG models with hidden state sizes of 32, 64, 128.

Data set	GRU-AVG32	GRU-AVG64	GRU-AVG128	RNN-AVG32	RNN-AVG64	RNN-AVG128	LSTM-AVG32	LSTM-AVG64	LSTM-AVG128
Adiac	15.85%	19.18%	31.45%	10.23%	9.46%	7.92%	7.67%	16.87%	9.2%
BME	84%	84%	84.66%	84%	72%	81.33%	64%	84.66%	84%
CBF	98.33%	98.66%	97.66%	97.33%	96.22%	96.66%	98%	99.77%	97.44%
Chinatown	98.25%	97.66%	98.25%	98.54%	98.83%	98.83%	98.54%	98.54%	98.54%
Chlorine Concentration	56.38%	56.87%	57.05%	55.39%	55.44%	55.07%	55.65%	55.88%	55.2%
Fungi	39.78%	48.92%	58.6%	40.86%	46.23%	60.21%	58.6%	58.06%	75.26%
Ham	75.23%	81.9%	77.14%	74.28%	73.33%	71.42%	80.95%	75.23%	77.14%
Haptics	38.63%	42.53%	44.48%	32.79%	33.11%	33.76%	34.74%	41.88%	39.61%
Herring	62.5%	65.62%	65.62%	67.18%	64.06%	59.37%	60.93%	64.06%	68.75%
InsectRT	100%	100%	100%	100%	100%	100%	100%	100%	100%
InsectST	100%	100%	100%	100%	100%	100%	100%	100%	100%
InsectWingbeat	26.26%	46.41%	44.69%	28.48%	23.03%	26.16%	32.47%	32.72%	39.94%
Meat	66.66%	68.33%	86.66%	66.66%	66.66%	65%	81.66%	66.66%	65%
OliveOil	40%	40%	80%	40%	40%	40%	40%	40%	40%
Plane	65.71%	93.33%	98.09%	65.71%	59.04%	62.85%	69.52%	70.47%	68.57%
Rock	52%	54%	62%	52%	56%	50%	50%	56%	60%
SmoothSubspace	91.33%	88%	88%	89.33%	90%	90.66%	86.66%	84%	86%
Synthetic Control	94.66%	92.33%	95.66%	88%	85.33%	94.33%	96%	95.66%	97.33%
UMD	66.66%	84.72%	92.36%	66.66%	75%	72.91%	70.83%	72.22%	72.22%
Wine	79.62%	72.22%	66.66%	75.92%	61.11%	66.66%	74.07%	72.22%	66.66%

Table 8. Performance of the model with Time Analysis with the Insect Wingbeat dataset. From NV-LSTM we inspect which time steps correspond to the most significant positive weights. From there we set the input data at those time steps to zero. As we set more of the time indices to zero, the test performance will start to decrease more and more.

Network	Class	# of Time Steps	Per Class Accuracy
NV-LSTM	0th	0	81.1
NV-LSTM	0th	1	80.0
NV-LSTM	0th	5	77.2
NV-LSTM	0th	10	70.0
NV-LSTM	0th	15	70.0
NV-LSTM	0th	20	68.3
NV-LSTM	0th	25	65.5
NV-LSTM	0th	30	61.1

Table 9. Performance of the model with Time Analysis with the Insect Wingbeat dataset. From NV-LSTM we inspect which time steps correspond to negative weights we can set the input data to those indices to zero. When we set at least 10 of them, we see that the class's test accuracy increases. From the previous table, setting the time indices from the positive weights would decrease the performance, but here we see the opposite effect.

Network	Class	# of Time Steps	Per Class Accuracy
NV-LSTM	0th	0	81.1
NV-LSTM	0th	1	81.1
NV-LSTM	0th	5	81.1
NV-LSTM	0th	10	82.2