A Biologically Inspired Approach to Learning Spatio-Temporal Patterns

Banafsheh Rekabdar
Computer Science and Engineering Department
University of Nevada, Reno
Reno, NV, 89557
Email: brekabdar@unr.edu

Monica Nicolescu
Computer Science and Engineering Department
University of Nevada, Reno
Reno, NV, 89557
Email: monica@cse.unr.edu

Mircea Nicolescu
Computer Science and Engineering Department
University of Nevada, Reno
Reno, NV, 89557
Email: mircea@cse.unr.edu

Richard Kelley
Computer Science and Engineering Department
University of Nevada, Reno
Reno, NV, 89557
Email: rkelley@cse.unr.edu

Abstract—This paper presents an unsupervised approach for learning and classifying patterns that have spatio-temporal structure, using a spike-timing neural network with axonal conductance delays, from a very small set of training samples. Spatio-temporal patterns are converted into spike trains, which can be used to train the network with spike-timing dependent plasticity learning. A pattern is encoded as a string of "characters," in which each character is a set of neurons that fired at a particular time step, as a result of the network being stimulated with the corresponding input. For classification we compute a similarity measure between a new sample and the training examples, based on the longest common subsequence dynamic programming algorithm to develop a fully unsupervised approach. The approach is tested on a dataset of hand-written digits, which include spatial and temporal information, with results comparable with other state-of-the-art supervised learning approaches.

I. INTRODUCTION

Learning and recognizing spatio-temporal patterns is an important problem for all biological systems. Gestures, movements, activities, all encompass both spatial and temporal information that is critical for implicit communication and learning. The main challenges that need to be addressed for such a learning problem are the number of training samples necessary for learning, finding a suitable encoding for the pattern and designing the learning and classification algorithms that would enable their recognition.

The majority of pattern recognition algorithms are supervised and require significant amounts of training data, which makes them unsuitable for a wide range of applications. In this paper we propose an unsupervised learning method that provides high accuracy, while relying on a very small set of training examples (5 per class). To encode spatio-temporal patterns, we propose a novel approach that uses a spike-timing neural network with axonal conductance delays. This means that a spike from a neuron to one of its connected neighbors is propagated after a particular time delay, not necessarily at the next time step. With these networks, the timing between neuronal firings strengthens or weakens the weight of a synapse between neurons through a spike-timing dependent plasticity learning (STDP) process [1]. This is different from artificial neural networks, in which synapses are updated based on the magnitude of the input. With STDP, repeated observations of a neuronal firing pattern leads to strengthening of some of the synapses in the network. As a result of STDP learning, a subset of neurons in the network will fire in a time-locked pattern (same time interval between firings) every time the pattern is observed. Our assumption is that similar patterns lead to similar neural firings. Therefore, to classify a new pattern we would need to find its similarity to previously seen patterns. The time-locked firing pattern, which constitutes the network’s response to a given stimulus, represents our basic (training or testing) model in the form of a spiking raster (a matrix representing which neurons fired throughout a given time interval). From this matrix, we build a new form of the model called model string as a string of "characters" or symbols, for which each character is the set of neurons that fired at a particular time step. To classify a new pattern, we build a similar model string and we use a similarity measure based on the longest common subsequence (LCS) dynamic programming algorithm for strings. While for strings the LCS algorithm uses exact matches between symbols, in our case the matches may not be exact: if two symbols in our model strings have a high overlap on the neurons that fired, that would still be considered a match. For the proposed approach we modified the LCS algorithm with a new symbol matching procedure, which takes the above consideration into account. For validation we use a dataset of hand-made digits, drawn with Gimp. The data contains both spatial and temporal information (i.e., how was the pattern being drawn). Our experimental results show that the proposed approach gives high accuracy, even with a very small set of training samples.

The remainder of the paper is structured as follows: Section II describes related work in learning spatio-temporal patterns, Section III presents our general approach, Section IV shows our experimental results and Section V concludes our paper.

II. PREVIOUS WORK

Learning of spatio-temporal patterns is a research problem that has been addressed in multiple domains. In computer vision, researchers have focused on learning human gestures or activities. Statistical methods for this problem include principal components analysis, hidden Markov models [2],
particle filters [3], and the condensation algorithm [4]. Finite state machines have also been used to model the temporal structure of human gestures [5]. In addition to such statistical methods, many vision systems use computer vision techniques in their processing pipeline [6] [7]; vision techniques used often include the analysis of shape, texture, color, motion cues, optical flow, segmentation, and contour modeling [8].

In robotics, multiple approaches have been proposed for learning trajectories from human's demonstration. Statistical modeling approaches include spline smoothing techniques [9], hidden Markov models [10], gaussian mixture models [11] and gaussian mixture regression [12]. Dynamical systems approaches to learning trajectories have used locally weighted regression [13], receptive field weighted regression [14] and locally weighted projection regression [15]. An important feature shared by all systems employing these methods is that they require large amounts of training data. In contrast with these approaches, we focus on methods that can learn from significantly small training sets.

Biologically inspired approaches, such as spike timing networks have been used in supervised approaches to learn spatiotemporal patterns [16], [17]. However, these have been shown only to learn very specific patterns, without the capability to generalize to classes of patterns. Spike timing has also been used with reservoir computing on supervised learning tasks [18]. Our methods emphasize unsupervised learning of features from training data and focuses on time-locked firing patterns to build representation that enable classification of temporal sequences.

### III. General Approach

Spike timing neural networks are suitable to model spatiotemporal patterns. In this section we describe our novel spike timing neural network formalism for pattern classification by using both the temporal and the spatial information in the pattern. The general strategy we employ is presented in Figure 1 and Figure 2. In the training phase, to train the network, the spatiotemporal patterns are mapped to a spike train, similar to the process described in [19] [20]. Next, the spike train is used to stimulate neurons in the network with axonal conductance delays. The STDP learning rule governs how the synaptic weights should update with each stimulation. The training phase ends after all the training samples have been given as input to the network the network and synaptic weights are updated according to STDP. We finally build a model string for each training sample with a procedure described in Section III-D. In the classification phase to classify a new, previously unseen, spatiotemporal pattern, we first map the input pattern into a spike train and then with the same procedure (Section III-D) create a model string for the input. The actual classification is based on selecting the closest training sample model string to the input model string as described in Section Section III-E.

#### A. Network Structure

The network has 320 neurons that are physically arranged in a 2D space. Neurons are connected to each other with a predefined probability distribution. Neurons are connected to 10% of other neurons in the network. We used a 2D Gaussian probability distribution function to randomly choose 10 neighbors for each neuron. The standard deviation of the Gaussian distribution is equal to 3. With this configuration, physically adjacent neurons in the 2D space have a higher probability of being connected with a synapse than two neurons that are physically not close. Each synaptic connection has a fixed conductance delay between 1 ms to 20 ms that is initialized randomly with a uniform distribution. Neurons can be either excitatory or inhibitory: 64 neurons from all 320 neurons are excitatory or inhibitory. We only stimulate excitatory neurons to feed an input to the network, based on [21]. Synaptic weights represent how strong a synaptic connection is and how strongly firing of a neuron would affect its synaptic neighbor. The weights for the synaptic connections are initialized to +6 for excitatory neurons and -5 inhibitory neurons, with 10 being the largest weight possible in our network. We adopted the Izhikevich
spiking neuron model [21] in our work.

\[
\begin{align*}
  v' &= 0.04v^2 + 5v + 140 - u - I \\
  u' &= a(bv - u) \\
  &\text{if } v \geq 30 \text{ mV}, \text{ then } v \leftarrow c, \ u \leftarrow u + d
\end{align*}
\]

In equations 1(a-c), \(v\) and \(u\) are membrane potential and recovery variables respectively, \(a\) is the time scale of the recovery variable, \(b\) is the sensitivity of the recovery variable, \(c\) is the after-spike reset value for the membrane potential and \(d\) is the after-spike reset for recovery variable. For excitatory neurons we set the parameters as \(a=0.02, b=0.2, c=-65, d=8\), and for inhibitory neurons we set the parameters as \(a=0.1, b=0.2, c=-65, d=2\).

B. Temporal Structure of Data

In order to experimentally evaluate how the proposed approach can classify spatiotemporal patterns, we have chosen a well-known test domain that has been extensively used in other similar approaches [18]. This dataset contains grayscale images of hand-written digits from 0 to 9, created with Gimp. The size of all images is equal to 16 x 16 pixels. Pixels with intensity values different than white are a part of the pattern, which means they are a part of the actual written digit in that 16 x 16 2D space. Spatial information in the images are easily stored, since pixel positions and their relations are known in the 2D image space. When a user is writing a digit, there is also a temporal information that captures which parts of the pattern were drawn first and how the hand moved to complete the write the digit. We use the full range of intensity values to also encode the relative temporal relationships between pixels as well as their spatial relations. The idea is similar to fade tapering. Pixel intensity values represent the relative temporal information: pixels with lower intensities are drawn before pixels with higher intensity values. Figure 3 describes the temporal structure in our hand-written digits dataset.

To feed our spatiotemporal patterns to the spike timing neural networks we need to define an association between the patterns and neuron activation. This association tells us which neuron to stimulate given a particular data element in the pattern that is currently being observed. In our work, we use a one to one association between neurons and pixels in the dataset. Since only excitatory neurons can be stimulated, the association only considers this type of neurons. Every pixel with an intensity value other than 255 (white) is considered a part of spatiotemporal pattern and the corresponding neurons would be stimulated at different times. To determine the timing of firing neurons, we sort pixels by their intensity values in increasing order and create a list of corresponding neurons. This list becomes a time-firing pattern and is used in training the network and in classification later on.

C. Training Spike Timing Networks

To train the network we present individual training samples to the network in 1 second intervals (without providing any label for each sample since our approach is unsupervised). For each training sample, a time-firing pattern is generated as described in Section III-B. For each neuron in the time-firing list, we stimulate the corresponding neuron for 1 ms before moving to the next neuron in the list. Thus, each training sample lasts a number of milliseconds equal to its temporal length. After firing each neuron during this process, we update the synaptic weights according to the Spike-timing Dependent Plasticity rule (STDP) [21][22]. According to STDP, the timing of spikes between a pre and postsynaptic neurons determines the amount of synaptic weight changes.

If a presynaptic spike arrives at postsynaptic neuron before
the latter fires, the strength of the connection is increased by $A_+ e^{-t/\tau}$. On the other hand, the synaptic weight is decreased by $A_- e^{-t/\tau}$ if the presynaptic spikes arrives the postsynaptic neurons just after the postsynaptic neuron fired. In our work we use $A_+ = 0.1$, $A_- = 0.12$ and $\tau_{au} = 20$ms, which are the same as [21]. Since for every data element in the pattern we need to spend 1 ms time after firing the corresponding neuron and the length of our data elements in the hand-written digit dataset is much less than 1000, we let the network run after stimulating all input neurons, and we let these firings propagate in the network while updating synaptic weights with STDP. To simulate a neuron, we simply apply to it an input current of 20 mA. During training, the input patterns are presented to the network by rotating through all five instances of each digit. We continue repeating the rotation until all input model strings become stable and do not change.

D. Modeling Data with Temporal Patterns of Firing Neurons

This section describes our approach to build model strings of the data from our trained neural network. Spiking neural networks, unlike standard three-layer networks have no particular output or input layers. However, to use this type of networks for classification, we need to define a computational approach to model the input data based on the behavior of the network. The core idea in spiking neural networks is the importance of temporal firing patterns. Therefore, any input to the network, can be modeled by capturing this timing of firing neurons while the input is presented to the network.

After training the network, we need to create model strings for each training sample in our dataset, which will be used for classification later on. This model string, represents how the neurons in the network fire when the particular input is applied to the network. We feed a training sample to the network by stimulating individual neurons corresponding to different pixels in the input. The pixel with highest intensity value will be fired at time 0, and pixels with lower intensity values will be fired in order of their intensities in 1ms intervals. For example, the neuron corresponding to a pixel with the second largest intensity, will be stimulated at 2ms and the third will be stimulated at 3ms until the entire pattern is exhausted. We then let the network propagate any existing activations, without any further stimulation, for 500 ms from the starting time. At each 1ms interval, we need to keep track of the fired neurons to create a model string for the input. If at any given time interval, no neurons were fired (either by our external stimulation or by receiving firing signals from previous neurons) we disregard that time interval in our model string.

More formally, a model string created for a given input to the trained network is a string in which the alphabet is the set of all possible subsets of the neurons in the network except the empty set. The ordering of characters, represent the temporal pattern of firing neurons for the given input. For example a model string such as $\{1, 2\} \{4, 14, 3\} \{5\}$ means that neurons 1 and 2 fired at the same time step, neurons 4, 14 and 3 fired at the same time (at a time-step after neurons 1 and 2), and finally neuron 5 fired after all of the mentioned neurons. It is important to note that we do not need to know the absolute time of the firing for each neuron, since we are interested in the order of neurons firing and not the absolute time of their firing. This is why we disregard the empty set of neurons as a character in our model string alphabet. Figure 4 shows the training algorithm of the spiking neural network. A model string in our approach can be a string of at most 500 characters. This is however is very unlikely to happen in practice for our dataset. We only have 256 pixels in our 16 x 16 images and most of the pixels are not a part of the pattern and receive no stimulation. 500 ms is a safe choice for this particular setting since it is improbable to have any neurons firing after this time span. This ending time is data dependent and it is directly related to the size of the network and the size of the input and should be chosen carefully for different applications.

E. Classification Algorithm

In order to classify a new pattern, we first compute a model string for that pattern with the same procedure described in Section III-D. This model string is a string of characters, where each character is a set of neurons that fired in response to stimulating the network with the new input pattern. Intuitively, if two spatiotemporal patterns are similar to each other, their model strings should also be similar. From the perspective of the spiking neural network, this means that similar firing patterns of neurons in the network is a sign of similar inputs. We create training models that contain a model string for each training sample. Therefore, for any given input we need to compare its model string with all the other model strings in the training dataset and choose the closest model string. The final class of the input would be the label of the closest training sample. The main challenge in classification with spiking neural networks is defining a suitable similarity measurement to compare two model strings with each other. Since a character in these strings is a set of neurons, unlike simple characters in which they are either equal or not, characters in our model strings can have different levels of similarities with each other. For instance a character such as $\{1, 2, 3\}$ is more similar to another character such as $\{2, 3, 4\}$ than $\{5, 6, 1\}$. Figure 5

![Fig. 4: The process of creating training and testing model strings.](image-url)
Fig. 5: Two model strings $a = (a_1 \ a_2 \ a_3)$ and $b = (b_1 \ b_2 \ b_3)$. Each character contains a set of neuron numbers which fired at the same time step. The values which are shown in edges present the Jaccard similarity between two characters, i.e. the Jaccard similarity for $a_1$ and $b_1$ equals to 0.33. Also the overall LCS similarity for $a$ and $b$ is 0.66.

shows the matching process and what the similarity values would be for two model strings $a$ and $b$.

To formally define a similarity measure for comparing two model strings, we first need to define a similarity measurement to compare individual characters (sets of firing neurons). A straightforward yet effective similarity measure between two sets is the Jaccard index \[23\]. Equation 2 shows the Jaccard similarity measure to compare to sets of neurons. This measure is normalized with respect to the size of two sets and can range from 0 to 1. Two sets for which one is a subset of the other have a similarity of 1 (maximum). If two sets do not have any intersection, then the similarity is 0 (minimum). This measure does not distinguish between the case that two sets are equal and two sets that one of them is a subset of the other.

If $A$ and $B$ are two sets of firing neurons corresponding to two patterns, the similarity measure between $A$ and $B$ is:

\[
sim(A, B) = 1 - \frac{|A \cap B|}{|A \cup B|} \tag{2}
\]

The main intuition behind the Jaccard similarity measure defined in Equation 2 is that two sets are similar if they have more members in common with each other. Following the same intuition, we can presume that two model strings are similar if they have more shared characters in them. However, the ordering of characters is also an important factor in building a model string because they represent the temporal firing patterns in the spiking neural network. Therefore a simple Jaccard similarity measure is not a good choice for comparing two model strings, since this measure ignores the ordering of elements in a sequence. In this work we use the Longest Common Subsequence algorithm (LCS) to define a measure of similarity between model strings. LCS is the longest subset of two strings, which preserves the order of the characters. The size of LCS is a good measure that captures how much two strings have in common while preserving the order. A normalization factor is needed to handle strings of different sizes. Equation 3 shows the similarity measure to compare two model strings. A minor modification to this measure is that characters are considered equal not only when they are exactly the same set of neurons, but also when their Jaccard measure is over a threshold. Therefore in this modified LCS we are actually matching similar characters with each other. If $A$ and $B$ are the model strings corresponding to two different patterns, the similarity measure between $A$ and $B$ is:

\[
sim_{LCS}(A, B) = \frac{|LCS(A, B)|}{\min(|A|, |B|)} \tag{3}
\]

where $|A|$ represents the length of string $A$.

If the similarity measure is zero, the two strings have no common substring, which shows they have nothing in common. A similarity measure of 1, shows that one of the strings is a substring of the other. This is the maximum similarity defined by LCS. Just like Jaccard, we cannot distinguish between the case where two model strings are exactly the same and the case where one string is a substring of the other. In this manner we obtain a similarity measure between the test samples and each of the training samples. For classification we choose the class of the sample from the training set that results in the greatest similarity measure.

The classification algorithm is as follows. First, we find the model string that represents the behavior of the network with the current input. This model is then compared with all the other model strings from training samples and the most similar one is chosen, and finally we assign the input to the class of the chosen training sample.

F. Parallelizing the Classification Algorithm

The basic approach to classification described above presents a number of opportunities for speedup. To achieve performance that would allow the system to be deployed on a robot interacting with humans in real time, we explored multiple strategies for parallelizing our code. In the case of the classification method described in section III-E, we found that relatively straightforward SIMD routines led to significant speedup of the classification process. In particular, we found that SIMD routines could be used to significantly improve performance for the computation of the similarity measure. This judicious use of SIMD routines led to a factor of seven improvement in the runtime of the classifier, bringing classification to a speed that is currently just on the edge of real-time operation - approximately 88 milliseconds per test data sample.

IV. Experimental Results

The domain we selected for our approach is recognizing handwritten digits. Our dataset, built as described in Section III-B consists of 5 training samples and 50 testing samples per digit. Figure 6 shows all training samples.

To validate the performance of our method, we defined and used the following quantitative measures: i) the success rate, which is the percentage of correctly classified test samples ii) the error rate, which is the percentage of wrongly classified test samples, and iii) the rejection rate, which is the percentage of test samples for which no classification can be made. A pattern is rejected by the network (no class is detected for that pattern) when the similarity measure between the pattern and any training sample is zero or there is a tie between maximum similarity classes. Table I and Table II show the results for
TABLE I: Classification results for each individual digit. SR: success rate, ER: error rate, RR: rejection rate

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR (%)</td>
<td>74</td>
<td>98</td>
<td>84</td>
<td>80</td>
<td>84</td>
<td>98</td>
<td>82</td>
<td>78</td>
<td>80</td>
<td>70</td>
</tr>
<tr>
<td>ER (%)</td>
<td>22</td>
<td>2</td>
<td>16</td>
<td>20</td>
<td>10</td>
<td>2</td>
<td>16</td>
<td>12</td>
<td>16</td>
<td>30</td>
</tr>
<tr>
<td>RR (%)</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>2</td>
<td>10</td>
<td>14</td>
<td>14</td>
<td>0</td>
</tr>
</tbody>
</table>

TABLE II: Classification results for all digits combined

<table>
<thead>
<tr>
<th></th>
<th>All Digits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Success rate (%)</td>
<td>82.5</td>
</tr>
<tr>
<td>Error rate (%)</td>
<td>14.6</td>
</tr>
<tr>
<td>Rejection rate (%)</td>
<td>3.3</td>
</tr>
</tbody>
</table>

As it is shown in Table I, the system performs very well on recognizing hand-written digits, with an overall accuracy of 82.5%. This is particularly significant, considering that we have only used 5 training samples. Table II and Figure 7 show the confusion matrix for the classification results. The columns show the actual classes and the rows represent the predicted class. The eleventh row represents the tie or unpredicted class.

As described in Section III-E, the classification phase in our approach requires selecting a threshold value to decide whether two sets of fired neurons that form two characters in our model strings either match or not. The effect of changing this threshold is shown in Table III. As it is shown in Table II, the best result was obtained by a threshold of 0.1 for Jaccard similarity. However, we can see that the performance does not degrade significantly by changing this parameter. Higher thresholds are forcing the system to consider sets of firing neurons that are very similar as a match and disregard the others. This limits the LCS measure and would result in shorter common substrings being detected by the classifier.

A. Comparison with Other Approaches

In order to evaluate the performance of our approach in the context of other machine learning methods, we compare our results with those given by support vector machines [24], regularized logistic regression [25] and ensemble neural networks [26]. To implement multi-class classification with SVM we used Libsvm [27]. The kernel, gamma and penalty parameters are polynomial, 1024 and 8 respectively. We implemented Regularized Logistic regression in Matlab, with the regularization parameter equal to 0.1. We used ensemble neural network instead of a single neural network, since the size of the feature set is too big in comparison to the number of training samples in the dataset (50 samples for all digits). In addition, based on [26], an ensemble neural network can generalize better rather than one. Our implementation of ensemble neural network is done in Matlab. Total number of neural neural networks in the ensemble is 15. All the networks in the ensemble have one hidden layer, but have a different number of hidden nodes: the first one has 4 hidden nodes, the second one has 6 hidden nodes, up to the last one which has 32 hidden nodes (the range of hidden nodes is between 4 to 32 with a step of 2). We set 0.02 and 0.005 for learning rate and threshold respectively in each neural network. The same training data has been used for all 15 networks. For training we used the Levenberg-Marquardt back propagation algorithm [28]. The activation functions used in the hidden and output layers are the *tansig* transfer function and the *purelin* activation functions respectively. Moreover, for selecting the best parameters we changed each parameter gradually to find the value that results in the best performance. In addition, we computed the average results of 10 trials for model generation. During classification we use majority voting to select the final output from the 15
TABLE IV: Result of comparison experiment (unit : %)

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>Logistic Regression</th>
<th>ENN</th>
<th>Proposed method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>86%</td>
<td>76.6%</td>
<td>55%</td>
<td>82.5%</td>
</tr>
</tbody>
</table>

The comparison results are shown in Table IV. Based on this table our unsupervised approach performs better than logistic regression and the ensemble neural networks. But its performance is slightly worse than SVM. The state-of-the-art methods that we selected for this comparison are all supervised learning approaches. A significant difference, however, is that the feature modeling proposed in our approach is fully unsupervised. Overall, based on these results, our approach can compete with and even outperform other state of the art supervised learning methods.

V. CONCLUSION

In this paper we propose an unsupervised spike timing neural network approach based on spike-timing dependent plasticity for classifying spatio-temporal patterns. We encoded patterns into spike trains, and train the network from a very small number of training samples. Each pattern is a string of characters. Each character is a set of neurons that fired in the same time step. During the classification phase, we compute the similarity value between a new sample and all training samples by using longest common subsequence and the Jaccard metric. The method is validated on a dataset of spatio-temporal hand written digit. We compared our results with three other machine learning methods; based on the results our method outperforms or is comparable with other state of the art supervised learning approaches.

ACKNOWLEDGMENT

This work has been supported by ONR Award N000141210860. Some of the computations in this paper were run on the High Performance Computing cluster at the University of Nevada, Reno.

REFERENCES