Democratic Liquid State Machines for Music Recognition

Leo Pape[1], Jornt de Gruijl[2], and Marco Wiering[2]

[1] Department of Physical Geography, Faculty of Geosciences, IMAU, Utrecht University; P.O. Box 80.115, 3508 TC Utrecht, Netherlands; l.pape@geo.uu.nl

[2] Intelligent Systems Group, Department of Information and Computing Sciences, Utrecht University; P.O. Box 80.089, 3508 TB Utrecht, Netherlands; jornt.degruijl@phil.uu.nl, marco@cs.uu.nl

cite as: Pape, L., J. de Gruijl and M. A. Wiering, 2008. Democratic liquid state machines for music recognition. In: Speech, Audio, Image and Biomedical Signal Processing using Neural Networks. Bookseries: Studies in Computational Intelligence, vol 83, B. Prasad and S.R.M. Prasanna (Eds.), Springer Berlin/Heidelberg.

1 Introduction

Liquid state machines [Maass et al., 2002] and echo state networks [Jaeger, 2001] are relatively novel algorithms that can handle problems involving time-series data. These methods consist of a liquid (or dynamic reservoir) that receives time-series data as input information, which causes the liquid to make transitions to dynamical states dependent on the temporal information stream. After a specific time-period the liquid is read out by a trainable readout network (such as a perceptron or multilayer perceptron) that learns to map liquid states to a particular desired output. Earlier recurrent neural networks that could also be used for handling time-series data such as Elman or Jordan networks suffer from the problem of vanishing gradients which makes learning long time-dependencies a very complex task [Hochreiter, 1998]. A more advanced recurrent neural network architecture is Long-Short Term Memory (LSTM) [Hochreiter and Schmidhuber, 1997]. LSTM is ideally suited for remembering particular events, such as remembering whether some lights were on or off [Bakker et al., 2003], but training the LSTM is more complicated than training a liquid state machine. The attractive property of a liquid state machine (LSM) is that the liquid which represents temporal dependencies does not have to be trained, which therefore makes the learning process fast and simple. Furthermore, the liquid state machine bears some similarities

to biological neural networks and can therefore profit from new findings in neurobiology.

In this paper we extend the liquid state machine for real-world time-series classification problems. Our democratic liquid state machine (DLSM) uses voting in two different complementory dimensions. The DLSM uses an ensemble of single-column LSMs and multiple readout periods to classify an input stream. Therefore instead of a single election on the classification, many votes are combined through majority voting. DLSM can therefore also be seen as an extension to bagging [Breiman, 1996] LSMs for time-series classification problems, since multiple votes in the time-dimension are combined. We argue that this multiple voting technique can significantly boost the performance of LSMs, since they can exhibit a large variance and small bias. It is well-known that using ensembles and majority voting can significantly reduce the variance of a classifier Ruta and Gabrys [2000].

In the present work we apply the LSM and its democratic extension on two different music recognition problems: composer identification and musical instrument classification. In the first problem the algorithm receives time-series input about the pitch and duration of notes of a particular classical music piece and the goal is to identify the composer of the piece. Here we will deal with two binary classification tasks. In the first task the goal is to distinguish between Beethoven and Bach as being the composer, and in another more complex task the goal is to distinguish between Haydn and Mozart. Furthermore, we will also study the performance of the LSM and the DLSM on a musical instrument classification problem where the goal is to distinguish between a flute and a bass guitar.

Outline. In Section 2 we describe neural networks, the LSM, the DLSM and the implementation we used. Then in Section 3 we describe the experimental results on the classical music recognition tasks. In Section 4 we describe the results on the task of musical instrument classification, and we conclude in Section 5.

2 Democratic Liquid State Machines

2.1 Feedforward Neural Networks

Artificial neural networks are based on the low-level structure of the human brain, and can mimic the capacity of the brain to learn from observations. Neural networks have several advantages compared to other pattern recognition techniques. Neural networks can learn non-linear relationships, do not require any a priori knowledge, can generalize well to unseen data, are able to cope with severe noise, can deal with high dimensional input spaces, and are fast in their usage (although not in training). A last important advantage of neural networks compared to algorithms based on computing distances between feature vectors is that no higher-level features need to be designed. Instead

a neural network learns to extract higher level features from raw data itself. However, not all types of pattern recognition problems are solved equally well by neural networks.

A feedforward neural network or multi-layer perceptron is a trainable differentiable function that maps an input x to an output y. It can be used both for regression and classification problems. A neural network uses a topology of neurons and weighted connections between the neurons. The output of a fully connected feedforward neural network with three layers (one hidden layer) given some input x is computed by first copying the input to the input layer of the neural network. After this the activations h_i of the hidden units are computed as follows:

$$h_i = f(\sum_j w_{ij}^h x_j + b_i^h), \tag{1}$$

where w_{ij}^h is the strength of the weighted connection between input j and hidden unit i, b_i^h is the bias of the hidden unit, and $f(.)$ is some differentiable activation function. Here we will use the commonly used sigmoid function: $f(z) = \frac{1}{1+exp(-z)}$.

After the values for all hidden units are computed we can compute the value y_i of output unit i:

$$y_i = \sum_j w_{ij}^o h_j + b_i^o, \tag{2}$$

where we use a linear activation function for the output units.

The weights w can be trained using gradient descent on the error function for a training example. The error function that is most commonly used is the squared error defined as: $E = \sum_i (d_i - y_i)^2$, where y_i is the i-th output of the neural network and d_i is the i-th target output for the example. Now we can use the derivative of the error function to each weight to change the weight with a small step:

$$\frac{\partial E}{\partial w_{ij}^o} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial w_{ij}^o} = -(d_i - y_i)h_j \tag{3}$$

The errors are all backpropagated to change the weights between inputs and hidden units:

$$\frac{\partial E}{\partial w_{ij}^h} = \sum_k \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial h_i}\frac{\partial h_i}{\partial w_{ij}^h} = \sum_k -(d_k - y_k)w_{ki}^o(1 - h_i)h_i x_j \tag{4}$$

The learning rules then become:

$$w_{ij} = w_{ij} - \alpha\frac{\partial E}{\partial w_{ij}}, \tag{5}$$

where α is known as the learning rate. The updates for the bias weights is very similar and are therefore not presented here.

2.2 Time-series Classification with Neural Networks

At present there exist a lot of pattern recognition methods that operate on snapshots of data, but these methods cannot always be applied to time-series data. Time-series data consist of sequences of snapshots that are typically measured at successive times and are usually spaced at uniform intervals. Often successive patterns in time-series show dependencies. Because these dependencies can span over patterns at different moments in time, they are referred to as temporal dependencies. When information on the class of time-series data is contained in the temporal dependencies, a pattern recognition method should be able to deal with dependencies in the temporal dimension. In the case of artificial neural networks it is of course possible to increase the number of input units, such that several snapshots containing temporal dependencies can be provided to the network at once. It is however not always a priori known over how many time steps the temporal dependencies span, which makes it difficult to determine how many patterns should be presented to the network at once. Besides, in case the dependencies span a very long time, the network becomes very large, which results in low learning speeds and increased chance of overfitting. Whereas neural networks without recurrent connections cannot use any information that is present in the dependencies between inputs that are presented at different time points, recurrent connections allow a network to keep some sort of short-term memory on previously seen inputs. However, using the short-term memory that is based on the activations of the units in the past is very difficult.

The reason that short-term memory based on previous activities in the network is really short, is that it is very difficult to retain a stable level of activity in the network. Once a certain level of activity is reached, connections with a large weight cause the activation to increase exponentially, and connections with a small weight cause the activations to decrease exponentially. Non-linear activation functions of the units prevent the activities from growing without bound, but the problem is that it is very difficult to train the network to make use of this sort of memory [Hochreiter and Schmidhuber, 1997]. For example in training algorithms based on error backpropagation, which are often used in non-recurrent neural networks, the contribution to the error of a particular weight situated lower in the network becomes more indirect. Consequently, when the error is propagated back over time, each propagation dilutes the error until far back in time it is too small to have any influence. This makes it very difficult to assess the importance of network states at times that lie far back in the past. In general, recurrent neural networks that are trained by backpropagation methods such as Realtime Recurrent Learning (RTRL) and Backpropagation Through Time (BPTT) [Williams and Zipser, 1989; Rumelhart et al., 1986] cannot reliably use states of the network that lie more than about 10 time steps in the past. In the literature, this problem is often referred to as the problem of the vanishing gradient [Bengio et al., 1994; Pearlmutter, 1995; Williams and Zipser, 1995; Lin et al., 1996; Hochreiter, 1998].

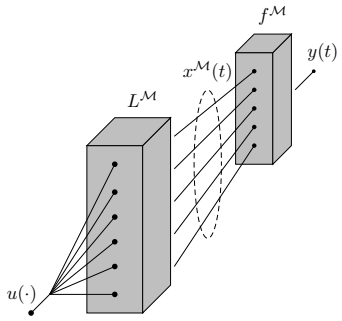

Fig. 1. The architecture of the liquid state machine. $u(\cdot)$ is the input, L^M the liquid, $x^M(t)$ the state of the liquid at time t, f^M the readout network, and $y(t)$ the output of the liquid state machine at time t.

2.3 The Liquid State Machine

Instead of training a recurrent neural network to learn to exploit memory over previous states, the liquid state machine [Maass et al., 2002] is based on the idea that due to their dynamics, recurrent neural networks can be employed as temporal filters. Just as a real liquid contains information in the form of waves and surface ripples on perturbations that were applied to it, the liquid filter of an LSM contains information on its present and past inputs at any moment. At certain times, the state of this liquid filter can be read out and presented to a separate readout mechanism. This readout mechanism can be trained to learn to extract useful information from the state of the liquid. The problems that arise when training recurrent neural networks disappear in this approach: the liquid itself is a recurrent neural network, but is not trained, while the readout mechanism does not need to implement any memory since the information on previous inputs is already reflected in the state of the liquid. Figure 1 shows the LSM architecture, connecting a liquid to a readout network.

Although the idea for the LSM originates from the dynamics of spiking recurrent neural networks, it can include any liquid that shows some kind of dynamical behavior. For a liquid to serve as a salient source of information on temporal dependencies in the input data, it should satisfy the separation property, which is the property to produce increasing divergence of internal states over time caused by different input patterns [Maass et al., 2002]. Furthermore, it should have a fading memory to prevent the state trajectories from becoming chaotic [Natschläger et al., 2004]. Whether these conditions are satisfied for a certain liquid, depends on the complexity and dynamics of the liquid. In general, more complex liquids with more degrees of freedom of the processes that govern their behavior, produce more diverging internal states, but also have a greater risk of showing chaotic behavior. The way in

which perturbations in the liquid are formed and propagated varies for different types of liquids and strongly determines how useful it is.

Recurrent neural networks with spiking units not only have a large variety of mechanisms and constants determining their interactions, but can also have local as well as non-local interactions. This means that they can effectively be used as high-dimensional liquid equivalents [Gupta et al., 2000]. Maass et al. [Maass et al., 2002] used a simplified model of the cerebral cortex, in which the individual neurons are organized in columns. These columns are all very similar and can be used to perform many complex tasks in parallel. Not only are these heterogeneous column-like structures found in the human cortex; they are also present in the cortex of other species. Implementations of these columns in the form of spiking neural networks, have already shown to perform very well; even in noisy environments their dynamics remain reliable [Vreeken, 2004].

The internal state of the liquid at a certain time reflects the perturbations that were applied to it in the past. The capability of the readout mechanism to distinguish and transform the different internal states of the liquid to match a given target, or the approximation property, depends on the ability to adjust to the required task. The readout mechanism should be able to learn to define its own notion of equivalence of dynamical states within the liquid, despite the fact that the liquid may never re-visit the same state. It is however argued by Maass et al. [2002] and more elaborately by Häusler et al. [2003] that the LSM operates by the principle of projecting the input patterns in a high-dimensional space. That means that very simple and robust (linear) readout mechanisms should be able to perform very well, given enough degrees of freedom in the liquid from which the readout mechanism can extract individualized states.

2.4 Democratic Liquid State Machines

Maass et al. [Maass et al., 2002] show that the separation property of the LSM can be improved by adding more liquid columns. In a multiple column configuration the input patterns are presented to each individual column. Because the columns consist of different randomly initialized neural circuitries, each column produces its own representation of the input patterns. In case of a single liquid column, the intrinsic complexity of this column determines the separation property, whereas the complexity of a column in case of multiple liquid columns becomes less important. This allows for a trade-off between the intrinsic complexity of the microcircuitry and the number of liquid columns.

An important difference between other multiple classifier systems and the multiple column LSM as described by Maass et al. [2002] is that although each liquid column extracts different features from the input patterns, no actual classification is being made at the level of the liquid. In the multiple column LSM the states of all liquid columns together are presented to a single readout

mechanism. Since each liquid column has its own intrinsic structure, each column produces its own kind of equivalence in its dynamic states. When a single readout mechanism is used, it should learn to define a notion of equivalence of the dynamic states for *all* liquid columns. This imposes certain requirements on the size of the readout mechanism (in terms of connections) and the complexity of its training algorithm. In order to decrease the complexity of the readout mechanism and to use the advantages of a combination function over multiple classifiers, a separate readout mechanism can be used for each individual liquid column. In this configuration each readout mechanism learns its own notion of equivalence of dynamic states in a separate liquid column. The classification results of each liquid-readout pair are now equivalent to classification results of multiple classifier systems, and can be combined using multiple classifier combination functions as for example is done in bagging [Breiman, 1996].

Improving performance using multiple classifiers does not only depend on the diversity between classifiers, but also on the function that is used to combine them. One of the simplest combination functions that has been studied thoroughly, is majority voting, which can be applied to classification tasks by counting the number of times each classifier assigns a certain class to an input pattern. The majority of the classification results is taken as the outcome of the majority voting combination function. In case more than one class gets the highest number of votes, there is no majority, and the outcome of the majority voting function is undecided. Benefits of majority voting have been studied for both odd and even numbers of classifiers, as well as dependent and independent errors [Ruta and Gabrys, 2000; Narasimhamurthy, 2005; Lam and Suen, 1997]. Even without the assumption of a binary classification task, independent errors or equally performing classifiers majority voting can substantially increase performance.

Apart from using multiple classifiers that each involve their own notion of equivalence over input patterns, the LSM allows for multiple classification results over time. As stated in Sect. 2.3, the liquid state contains information on present and past perturbations at any moment. Because the liquid should involve some kind of 'forgetting behavior' to prevent the state trajectories from becoming chaotic, additional information can be gathered by using the state of the liquid at different moments instead of just the final liquid state. In case the length of the time-series of input patterns is relatively large compared to the time-constants of the liquid, multiple readout moments can be used. The readout mechanism produces a classification using the liquid state at each readout moment. In this fashion the input data are divided in parts, and a classification is performed on each part. In case the activity in the liquid is not reset at the beginning of each part of the input data, the liquid still satisfies the requirements of the LSM. As with multiple classifier systems, the individual classification results at each readout moment can be combined using a certain combination function. Although the different classification results are by no means independent, majority voting can still be used to boost performance

over multiple results [Ruta and Gabrys, 2000]. Majority voting over the results of the same classifier but for different input patterns at different moments, can be used, at least in theory, to improve performance of the LSM.

Combining majority voting over multiple classifiers with majority voting at multiple times results in a strategy in which the classification of a time-series pattern is not determined by a single election, but by the result of multiple elections performed by different classifiers, each having their own generalization strategy. LSMs that combine majority voting over spatial (multiple classifiers) and temporal (multiple classification moments) dimensions are what we call democratic liquid state machines (DLSMs).

2.5 Implementation of the Liquid State Machine

As stated in Sect. 2.3, spiking recurrent neural networks can effectively be used as high-dimensional liquid equivalents. A spiking neural network tries to capture the behavior of real biological neural networks, but does not necessary implement all features and interactions of its biological counterpart. One model that is commonly used for networks of spiking units, is the integrate-and-fire model [Feng and Brown, 2000]. This is a general model that implements certain properties of biological neurons, such as spikes and membrane potentials, and describes in basic terms that the membrane potential of a unit slowly leaks away, and that a unit fires if the membrane potential reaches a certain value. This model is rather simple to understand and implement, and can be described by giving the differential equations that govern the behavior of the membrane potentials, spikes and some other variables over time. Although it is common practice in the field of LSMs to use spiking neural networks with a continuous-time dynamics, in the present work a spiking neural network is used in which the states and variables of the units are computed in discrete time. Therefore the equations that describe the behavior of such units are not given in differential equations, but are described in terms of the updates of the membrane potentials and activities of the units, that are computed each time step t.

The units in the present work are a simplified model of their more realistic biological counterparts. Spikes are modeled by short pulses that are communicated between units. Action potentials in biological neurons are all very similar. Therefore all spikes are modeled to have the same value, so the output y_i of unit i is 1 if a unit fires and 0 if it does not fire. Incoming spikes affect the membrane potential of a unit. How much each incoming spike decreases or increases the membrane potential is determined by the synaptic strength, or weight w_{ij}, of the connection between unit j and i. In the spiking neural network, a certain proportion p^i of the weights is set to a negative value to provide for inhibitory connections, while the remaining weights are set to positive values, representing excitatory connections. Because in biological neurons the effect of an incoming signal lasts longer than the duration of the pulse itself and the membrane potential in absence of any input slowly

returns to its resting potential, the accumulated input of a unit at each time step is added to a proportion of the membrane potential of the previous step. To keep the equations for the membrane potential simple, the resting potential for each unit is 0. The parameter that controls the speed at which the membrane potential returns to its resting potential is:

$$\tau = \left(\frac{1}{2}\right)^{\frac{1}{t^h}}, \tag{6}$$

where t^h is the number of time steps in which half of the membrane potential is left if no additional inputs are presented. The membrane potential μ_i, of unit i is computed each time step according to:

$$\mu_i(t) = \tau\mu_i(t-1) + \sum_j w_{ij}y_j(t-1) \tag{7}$$

Subsequently the output, or whether the unit fires or not, is computed by comparing it with a positive threshold value η:

$$y_i(t) = \begin{cases} 1 \text{ if } \mu_i(t) \geq \eta \\ 0 \text{ otherwise} \end{cases}, \tag{8}$$

where η has a value of 1 throughout all experiments. After sending an action potential, a biological neuron cannot fire again for a certain period, followed by a period in which it is more difficult to fire. This behavior is modeled in the artificial neurons by implementing an absolute and a relative refractory period. After sending an action potential, it is impossible for a unit to fire again for a number of t_i^a time steps, where t_i^a is the absolute refractory period of unit i. Combining this with equation 8, yields:

$$y_i(t) = \begin{cases} 1 \text{ if } \mu_i(t) \geq \eta \text{ and } (t - t_i^f) > t_i^a \\ 0 \text{ otherwise} \end{cases}, \tag{9}$$

where t_i^f is the last time unit i generated an action potential. When the absolute refractory period has ended, a relative refractory period is simulated by resetting the membrane potential μ_i to some negative value $-r_i$ which influences the relative refractory period of unit i, thus making it harder for the membrane potential to reach the positive threshold value η. To implement this relative refractory period, equation 7 is augmented with an additional term:

$$\mu_i(t) = \begin{cases} \tau\mu_i(t-1) + \sum_j w_{ij}y_j(t-1) \text{ if } (t - t_i^f) > t_i^a \\ -r_i \qquad\qquad\qquad\qquad\qquad \text{otherwise} \end{cases} \tag{10}$$

In case there is no additional input presented to the unit, the negative membrane potential slowly returns to the resting potential, just as it would for a positive membrane potential in case no additional inputs are presented. Equations 9 and 10 sum up the whole of the dynamics of spiking units of which the spiking neural network consists.

The structure of the liquid is an n-dimensional space, in which each unit is located at the integer points. The units in this space are connected to each other according to a stochastic process such that units that are close to each other have a higher chance to be connected than units that lie farther from each other. To create such a network with mainly locally connected units, the chance $p_i^c(j)$ for each unit i that it has a connection to unit j is computed according to:

$$p_i^c(j) = Ce^{(D(i,j)/\lambda)^2}, \tag{11}$$

where $D(i, j)$ is the Euclidean distance between units i and j. λ is the parameter that controls both the probability of a connection and the rate at which the probability that two units are connected decreases with increasing distance between the two. C is the parameter that controls just the probability that two units are connected, and can be used in combination with λ to scale the proportion between the rate at which the probability that unit i and j are connected decreases with increasing distance between i and j, and the actual probability of a connection. Maass et al. [Maass et al., 2002] found that the connectivity in biological settings can be approximated by setting the value of λ to 2, and C to a value between 0.1 and 0.4, depending on whether the concerning units are inhibitory or excitatory. However, that does not mean that other values for C and λ cannot yield a reasonable separation property of the liquid. In the present work λ is set to a value of 2 in all experiments, while the value of C was set depending on the task. The probability that a connection is inhibitory or excitatory is computed independent of the properties of the units it connects, so C was the same for all units.

When time-series data of input patterns are provided to the network, the state $y_i(t)$, of unit i in the network is computed at each discrete time step t, using the states of connected units at the previous time step. The incoming signals propagate through the network, and produce certain activities in the liquid units. The total activity of all units in the network at a certain time step contains information on the inputs that were provided during the previous time steps. It is not necessary to present the liquid state to the readout mechanism at each time step, because the liquid can preserve information on previously seen inputs over several time steps. The interval between successive readouts is denoted by t^v. At the beginning of each time-series pattern, the activity in the network is reset to zero. Because it takes some time before the activity has spread from the input units to the rest of the network, the collection of liquid states is started after a number of time steps. The time between reset and the first readout moment is called the warmup period t^w.

Because in a spiking neural network the units are not firing most of the time the activity in the network at the readout moments is sparse. That means that little information can be extracted from the activities at just a single time step. There are several ways to create representations of liquid states that overcome the problem of sparse activity. For example, the membrane potential μ can be used, which reflects more information about the activity

in the network than just the unit's activity. Another method is to take the number of times a unit fires during a certain period as measure of activity in the network.

As stated in Sect. 2.3, the LSM operates by projecting input patterns in a high-dimensional space, which makes the learning process less complicated [Vapnik, 1998]. Due to the high-dimensionality of the liquid, the projections of the original input patterns become linearly separable. In general, this means that the more degrees of freedom the liquid has, usually the easier it becomes for the readout mechanism to learn to predict the right targets. Because no theory exists at present that tells for which problems and liquids the task becomes linearly separable, we cannot be sure whether this is attained for a certain amount of units and parameter settings of the spiking neural network. Therefore the readout mechanism in the present work is a multilayer perceptron [Rumelhart et al., 1986], which can also learn non-linearly separable tasks.

At the end of intervals with length t^v the representations of the liquid states are presented to the multilayer perceptron. All units in the liquid other than the input units are used for readout, so the number of units in the first layer of the multilayer perceptron is equal to the number of units in the liquid minus the input units. The multilayer perceptron is trained by the supervised backpropagation algorithm [Rumelhart et al., 1986], with first-order gradient descent as explained in Sect. 2.1. In order to compute the error, a representation of the desired output has to be created. In the classification tasks studied in the present work, each output unit is assigned to a class. The desired activity of the unit that represents the class that corresponds to an input pattern is 1, while the desired activity for all other output units is 0. In this fashion the output unit that represents the class that corresponds to an input pattern learns to have a high activity, while the other output units have a low activity. For testing purposes the unit with the highest activity determines the class that is assigned by the network.

In the implementation of the DLSM in the present work, the time-series of input patterns is presented to several spiking neural networks, where each network has its own multilayer perceptron that serves as readout mechanism. Each perceptron is trained to learn a mapping from the states of the spiking network to classification outputs represented in the activation of the output units. After the training of all perceptrons is finished, the performance on novel data can be determined by multiple elections over the classification results of the multilayer perceptrons. First, the outcome at each readout moment is determined by applying majority voting over all readout mechanisms. Next, majority voting is applied on the results of the previous step for all readout moments. The outcome of the elections determine the class of a time-series of input patterns.

3 Classical Music Recognition

3.1 Learning Task

The performance of the DLSM can be investigated using a classification task involving time-series data in which the classes are determined by temporal dependencies. A real-world pattern recognition task that satisfies this requirement is the classification of music based on musical content. A set of musical pieces can be divided into different classes based on several criteria. The criterion used here is the composer. Two different binary classification tasks were constructed: a task that is quite easily solved, and a far more difficult task (even for human experts). The first task consists of the classification of Johann Sebastian Bach and Ludwig van Beethoven, using music from the 'Well-Tempered Clavier' by Bach, and the 'Piano Sonatas' by Beethoven obtained from several sources on the internet. The second task is the classification of Joseph Haydn and Wolfgang Amadeus Mozart, using files from the 'String Quartets' of both composers. The difficulty of the second task is demonstrated by the current identification statistics, which indicate an average performance of 53% for self-reported novices and a maximum of 66% for self-reported experts [Sapp and Liu, 2006]. The musical instruments for which the works are composed – keyboard instruments and string quartets – are the same in each of the two classification tasks, so recognition cannot be based on features that are specific for a certain musical instrument.

3.2 Input Representation

Whereas most pattern recognition systems dealing with sound first transform raw wave data into frequency bands, *pitch* representation rather than a representation involving frequency bands is used in the music classification task. Pitch representation is also used by the human brain when processing music. There are specific neurons that respond to specific pitches rather than frequencies, which are spatially ordered by pitch, not frequency, just as the keys on a piano [Pantev et al., 1989, 1991]. A musical representation that consists of information on the pitch of a note, its onset, its duration and its loudness, is the MIDI format. Because neural networks cannot deal with 'raw' MIDI files, the files were processed to time-series data with ordered pitch information.

A MIDI file consists of events. There are a large number of events that can be represented in such a file, most of which have to do with specific properties of electronic instruments, but are not relevant for the experiments. The events used here are: the onset and duration of a note, and its pitch. Each note is set to the same velocity value, and the beginning and duration of the notes are set to discrete time steps with intervals of $\frac{1}{32}$ note. This value is chosen because a note with a duration shorter than $\frac{1}{32}$ has little implication for the melodic line, and is more often used as ornament. Furthermore, all files are transposed to the same key, namely the key of C-major for pieces in a major scale, and

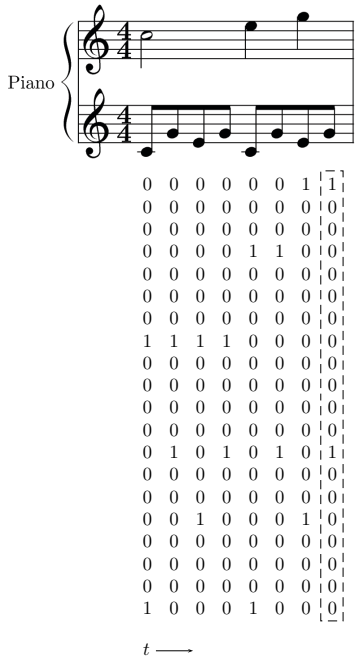

Fig. 2. Representation of MIDI file in staff (top), and part of the corresponding time-series data (bottom). The numbers within the dashed box represent an input pattern to the liquid filter at one time step.

A-minor for pieces in a minor scale, such that classes cannot be determined based on key. In a MIDI file a range of 128 different pitches can be represented, but because the data sets contain files composed for certain instruments, notes that cannot be played or are rarely played on such instruments, are filtered out. After preprocessing the MIDI files, the results are represented in the form of time-series data, in which the time is divided into discrete time steps of $\frac{1}{32}$ note. Each input pattern in the time-series data consists of a set of 72 numbers (corresponding to the 72 most-used keys on a piano starting at f2), which have a value of 1 if the corresponding note is 'on' and 0 if the corresponding note is 'off'. A simplified version of the process of converting MIDI files to time-series data with ordered pitches is depicted in Fig. 2.

3.3 Setup

The time-series data extracted from the MIDI files consisting of information on the presence of each of the 72 notes are presented to the liquid columns. The structure of each liquid column is a three-dimensional space, in which the units are located at the integer points. The time-series data are presented to

the leftmost row with dimensions 12×6 to represent the 12-note scale property of the music over 6 octaves ($12 \times 6 = 72$ notes). Each unit in a horizontal row of 12 units corresponds to a note in the 12-note scale, and each vertical row to one octave. For all notes that are played in the MIDI file at a certain time step, the output of the corresponding units is set to 1, while the output of the other units is set to 0. To create a three-dimensional space, 5 additional rows of the same size are placed next to the input row. The resulting three-dimensional space of units has dimensions $12 \times 6 \times 6$. In the experiments on music classification by composer the liquid units are prevented from having self-recurrent connections.

The interval between successive readouts t^v, is set to 128 for all experiments. To give an idea about the duration of the music that is presented to the liquid during this period: 128 time steps correspond to $\frac{128}{32}$ notes in the MIDI file (= 4 bars in $\frac{4}{4}$ measure). Furthermore, the warmup period t^w is chosen to be half as long as the interval between successive readouts.

As stated in Sect. 2.5, little information can be extracted from the sparse activity in the network at a single time step. Therefore not the information whether a unit is firing or not at a single time step, but the number of times a unit fires during a certain readout period is used as measure of activity in the network:

$$z_i(t) = \sum_{t'=t-t^s}^{t} y_i(t'), \tag{12}$$

where t is the time step at which the liquid state is read, $z_i(t)$ is a measure of activity of unit i, and t^s is the number of time steps in the readout period. Next, the results of equation 12 are scaled between 0 and 1 by dividing it by the maximum number of times a unit can fire during the readout period:

$$x_i(t) = \frac{z_i(t) t_i^a}{t^s} \tag{13}$$

The results of equation 13 for all units in the liquid other than the input units are presented to the readout mechanism. The readout mechanism for each liquid column is a three-layer perceptron. The number of units in the input layer is equal to the number of units in the spiking neural network other than the input units, which is $12 \times 6 \times 5 = 360$. In both classification tasks the number of classes is two, so there are two units in the output layer of the multilayer perceptron. The optimal number of units in the hidden layer is to be determined empirically. The training process of the multilayer perceptron is performed in batch-wise mode.

In the experiments the generalization capacity of an algorithm is tested, so the data are divided into a training set that is used for the training algorithm and a test set that is used to compute the performance of the selected method. Because the learning speed of several training algorithms can substantially be improved by using the same number of patterns in each class, the number of files in the training sets are such that an equal amount of

Table 1. Parameters for the spiking neural network

Parameter description	Symbol	Value
liquid dimensions		$12 \times 6 \times 6$
connectivity parameter	C	0.4
connectivity parameter	λ	2
proportion of inhibitory connections	p^i	0.35
weight between unit i and j	w_{ij}	0.4
firing threshold	η	1
half-time of the membrane potential	t^h	4
mean of Gaussian distribution of absolute refractory period	t^a	4
standard deviation of Gaussian distribution of t^a		2
relative refractory period	r_i	$\frac{\tau t_i^a}{10}$
readout interval	t^v	128
warmup period	t^w	$\frac{1}{2}t^v$
readout summation steps in Bach / Beethoven task	t^s	100
readout summation steps in Haydn / Mozart task	t^s	50

classification moments occurs for both classes. In the Bach / Beethoven classification task, Book I of the 'Well-Tempered Clavier' and 20 movements of the 'Piano Sonatas' are used as training data, resulting in approximately the same number of classification moments for both classes. Book II of the 'Well-Tempered Clavier' and 30 movements of the 'Piano Sonatas' are used as test data. For the Haydn / Mozart classification task, approximately two-third of the movements of Mozart is used as training set, together with 64 movements of Haydn's 'String Quartets' which results in the same amount of classification moments for both classes. The remaining movements of both composers are used as test set.

Both the multilayer perceptron used as readout mechanism, and the spiking neural network used as liquid column have a number of adjustable parameters. For some parameters the values can be inferred from existing literature, as described in Sect. 2.5, while other parameters have to be attuned to the task at hand. Several experiments are performed to find the right parameter settings of the neural networks for the learning tasks. To make a quick comparison between the results of different parameter settings, the average percentages of correctly classified liquid states over a total of 20 LSMs are compared. Table 1 shows the values that were found during the experiments for a number of parameters, together with the values of the parameters that are inferred from existing literature as described in Sect. 2.5. The parameter values for the readout networks that were found during the experiments are given in Table 2. Note that the number of epochs needed to train the networks was different in both classification tasks because the number of patterns in the training set is not the same.

Table 2. Parameters for the multilayer perceptron

Parameter description	Value
number of units in each layer	$360 \times 8 \times 2$
initial weight interval	$[-0.3, 0.3]$
initial bias interval	$[-0.1, 0.1]$
batch size	10
learning rate	0.00025
number of epochs in Bach / Beethoven task	3000
number of epochs in Haydn / Mozart task	2000

3.4 Experiments

A task that is difficult for humans does not always have to be difficult for computers as well, and vice versa, so to get an indication how difficult the composer identification task is for a machine learning algorithm a number of experiments were performed with some rather simple algorithms. In this fashion the performance of the DLSM can not only by compared to that of the LSM, but also to some less-sophisticated methods.

The first method that is tested is a very simple note-counting algorithm. All notes that are played are summed during the entire duration of the songs in the test set, and the result is compared to the average number of times these notes are used in the training set. The class with the smallest absolute difference is assigned as the outcome for each song. In Tables 3 and 4 the results for the note counting experiments are given.

The setup of the second method is more like the LSM, but instead of a spiking neural network as a liquid, readout samples are created by summing over the notes that are played within the same readout interval as the LSM, and scaling the results between 0 and 1. The other parameters have the same values as those that were found in the LSM and DLSM experiments, and all results are averaged over 20 repeated experiments. To see how this simple 'summation liquid' compares to an LSM with a spiking neural network liquid, majority voting over multiple readout moments (MV over time) and the spatial dimension (MV over multiple classifiers) was also tested. While it is straightforward to create different liquid columns for the multi-column LSM and the DLSM, summation can only be performed in one way, so only the initial weights and bias values of the readout networks were different for a comparison with the 'multiple liquid columns' experiments. The results are given in Tables 3 and 4. The 'average' columns gives the average over 20 experiments, the 'best' column the best out of 20 experiments, and the 'sd' column the standard deviation.

Next several experiments are performed to investigate the results of majority voting over different dimensions, with spiking neural networks as liquids. Just as in the previous setup the experiments for each parameter setting are repeated 20 times. In the first experiment (MV over time) majority voting

Table 3. Results for the Bach / Beethoven classification task

Note counting	Correct		
All samples	71.3%		
Summation over time	**Average**	**Best**	**SD**
All samples	73.4%	73.9%	0.3
MV over time (correct)	88.3%	90.8%	1.3
MV over time (undecided)	1.7%		0.7
MV over multiple classifiers (correct)	73.4%	73.6%	0.1
MV over multiple classifiers (undecided)	-		-
MV over multiple classifiers and time (correct)	87.8%	89.8%	0.9
MV over multiple classifiers and time (undecided)	1.2%		0.4
Spiking neural network liquid	**Average**	**Best**	**SD**
LSM (average correct over all samples)	73.3%	75.6%	1.3
MV over time (correct)	90.2%	93.5%	2.1
MV over time (undecided)	2.4%		1.5
MV over multiple liquid columns (correct)	77.6%	79.1%	0.8
MV over multiple liquid columns (undecided)	-		-
DLSM (correct)	**92.7%**	**96.7%**	**1.6**
DLSM (undecided)	1.5%		1.0

is applied over different readout moments of a single liquid column. In the second experiment (MV over multiple liquid columns) majority voting is applied over the time-averaged results of multiple liquid columns, and in the last experiment (DLSM) majority voting is applied over multiple liquid columns and over multiple readout moments. For majority voting over multiple liquid columns we tested multiple-column LSMs with increasing amounts of different liquid columns, up to 20 columns. As expected the performance of the multiple-column LSM improves when more liquid columns are added. The best results in the Bach / Beethoven classification task were obtained with 15 columns, while a DLSM with 8 liquid columns achieved the best results in the Haydn / Mozart classification task. The results of the experiments for the optimal number of liquid columns are given in Tables 3 and 4. Note that the 'undecided' row for majority voting over multiple liquid columns is missing in Table 3, because of an odd number of liquid columns.

For both types of liquids (simple summation and the spiking neural network) we find in most experiments a significant increase in performance for majority voting over time. While for the DLSM additional performance increase can be obtained by adding different liquid columns, this is of course not the case for the summation liquid (there is only one way to compute a sum). Another striking result is that the performance of a summation liquid is almost the same as that of a single-column LSM. This might be caused by the fact that a relatively simple liquid was used here compared to the more

Table 4. Results for the Haydn / Mozart classification task

Note counting	Correct		
All samples	54.9%		
Summation over time	**Average**	**Best**	**SD**
All samples	55.6%	55.8%	0.1
MV over time (correct)	57.3%	59.6%	1.2
MV over time (undecided)	5.1%		1.6
MV over multiple classifiers (correct)	55.6%	55.8%	0.1
MV over multiple classifiers (undecided)	0.4%		0.1
MV over multiple classifiers and time (correct)	57.0%	57.8%	0.9
MV over multiple classifiers and time (undecided)	5.9%		0.9
Spiking neural network liquid	**Average**	**Best**	**SD**
LSM (average correct over all samples)	56.0%	58.3%	1.1
MV over time (correct)	55.8%	66.1%	4.6
MV over time (undecided)	5.5%		2.3
MV over multiple liquid columns (correct)	57.4%	58.9%	1.0
MV over multiple liquid columns (undecided)	0.2%		0.6
DLSM (correct)	**63.5%**	**73.5%**	**4.5**
DLSM (undecided)	7.1%		2.6

sophisticated liquids used by others Maass et al. [2002]; Vreeken [2004], but the fact that even real water can effectively be applied as liquid column Fernando and Sojakka [2003] indicates that this is not always the case. Whereas the results of the previous experiments hardly encourage the use of an LSM, the strength of this method becomes clear when majority voting is applied to multiple readout moments and multiple different classifiers as is done in the DLSM. The DLSM achieves a significantly ($p = 0.01$) higher performance than the single-column LSM and other less-sophisticated algorithms.

When the results of the different composer identification tasks in Tables 3 and 4 are compared, it becomes immediately clear that the Bach / Beethoven task is more easy to solve than the Haydn / Mozart task, at least for the algorithms used here. Because no identification statistics for human beings on the Bach / Beethoven classification task are currently known to the authors we cannot say anything with certainty about the difficulty of the Bach / Beethoven classification task. Although both tasks might be equally difficult for the untrained human ear, the identification statistics of the 'Haydn / Mozart String Quartet Quiz' show a maximum of 66% correct movements for self-reported experts [Sapp and Liu, 2006], indicating that the task is very difficult even for the trained human ear. However, a direct comparison between the quiz and the results in Table 4 cannot be made because in the quiz no distinction is being made between train and test sets. The DLSM, with an average of 63.5% correct (and 7.1% undecided) on the test

has not been presented with any file in the test set before, while in case of the 'Haydn / Mozart String Quartet Quiz' the contesters might have heard a piece before, for several times, or might even be able to play (reproduce) a piece themselves. Furthermore the 'Haydn / Mozart String Quartet Quiz' gives only averages, so we cannot compare our best DLSM (73.5%) to the best contester in the quiz. Although no fair comparison can be made between the performance of the DLSM and the human expert we can conclude that the performance of the DLSM comes close to the performance of human experts even under less favorable circumstances.

4 Musical Instrument Recognition

4.1 Learning Task

Apart from the composer classification task, we also test the DLSM architecture on the more basal task of timbre recognition. Musical instruments generate tones that can potentially span many octaves and thus cover a wide range of frequencies, with relations between frequency bands largely determining the timbre. While in the composer classification tasks timbre information was left out of the input data representation, in the present task timbre is the most important factor used to distinguish between musical instruments. For this task we use two musical instruments: the bass guitar and the flute. These instruments differ significantly in sound (even to the untrained ear) and have a relatively small tonal overlap, providing a convenient focus for this experiment.

In an experiment using nine wind instruments and a piano in a pair-wise comparison classification paradigm, 98% of the piano-vs.-other cases were classified correctly[3] [Essid et al., 2004]. These results were obtained using extensive preprocessing on the input data, the need for which is decreased dramatically when using reservoir computing variants. Although the aforementioned results indicate that this task is fairly easy, there is still room for improvement.

4.2 Data Set and Input Representation

The majority of the samples used here are made especially for this classification task. The data set consists of single tones, two-tone samples and partial scales. For the samples made for this research three different bass guitars and three different flutes (one of which was an alto flute to extend the range of

[3] The piano is the closest match found to a bass guitar in musical instrument classification literature. Since the difference between a piano and a wind instrument is not only great, but also qualitatively similar (string instrument vs. wind instrument) to the difference between flute and bass, this seems a reasonable indication of the difficulty of the task.

the tonal overlap) that differ significantly in timbre are used. A small number of additional samples is taken from several CDs and can feature background noise such as roaring crowds and other instruments playing. The samples taken from CDs can also venture outside the tonal overlap.

All of the samples that are used in our timbre recognition experiments are 16-bit mono wave files with a 22 kHz sample rate. Most files are roughly one second in length, but some are longer. The data set consists of 234 wave files, with an equal number of bass guitar and flute fragments. Instead of presenting the samples to an LSM as raw wave data, the samples are transformed into the frequency domain by applying a Fast Fourier Transform (FFT). This yields an efficient representation of the input data and bears some similarities to audio processing in the human cochlea.

The length of the time window on which the FFT algorithm is applied is 5.8 ms, and the resolution of the FFT is set such that the raw wave data are converted into vectors of 64 frequency bands with their respective amplitudes. After applying the FFT the data is normalized between 0 and 1 for compatibility with the liquid's input units. The normalized FFT vectors are fed into the liquid's input units every 5 time steps, making 1 time step roughly equal to 1 ms and allowing the input to propagate through the liquid before the next is presented.

4.3 Setup

The model used for the musical instrument classification task is somewhat different from the one used for music recognition. For this task a two-dimensional rather than a three-dimensional liquid is sufficient. The liquid is organized in a grid of 64×5 units, including one layer of 64 input units, representing the 64-band frequency resolution of the FFT transform. The dynamics of the liquid used for the musical instrument recognition task are as described in equations 9 and 10, but no relative refractory period is implemented ($r = 0$). Readout periods are determined by cutting up samples into five fragments of roughly the same duration. Since most sound samples are approximately one second in length, readout periods would span some 200 ms, which translates to 40 input patterns.

Because the activity in the liquid is sparse, little information can be extracted from the activity in the network at one time step. Therefore not the information whether a unit fired at a single time step, but the membrane potential equivalent μ, is used as measure of activity in the network: $x_i(t) = \mu_i(t)$. The value of x_i for each non-input unit i in the liquid is presented to the readout mechanism.

For the readout function a three-layer perceptron is used. This neural network is trained using backpropagation with first-order gradient-descent. A special requirement has to be met in order to end the training period: the output for the correct class has to be bigger than the output for the incorrect class by at least a factor 1.5. If this requirement is not met after

Table 5. Parameters for the spiking neural network

Parameter description	Symbol	Value
liquid dimensions		64×5
connectivity parameter	C	0.9
connectivity parameter	λ	2
proportion of inhibitory connections	p^i	0.1
weight between unit i and j	w_{ij}	0.2
firing threshold	η	1
half-time of the membrane potential	t^h	4.5
absolute refractory period	t^a	15
relative refractory period	r	0
readout interval (see text)	t^v	~ 40
warmup period	t^w	t^v

Table 6. Parameters for the multilayer perceptron

Parameter description	Value
number of units in each layer	$256 \times 50 \times 2$
initial weight and bias intervals	$[-0.5, 0.5]$
learning rate	0.1
high / low output value ratio	1.5
maximum number of epochs	2000

2000 iterations, training is also stopped. The number of units in the input layer is equal to the number of the non-input units in the liquid: $64 \times 4 = 256$. In the musical instrument recognition task the number of classes is two, so the readout network has two output units.

To determine the parameter settings for the liquid and the readout mechanism for which the best generalization capacity is achieved, several experiments are performed. Each experiment is repeated 50 times with a different distribution of samples over the training and test sets. Of the 234 wave files in total, 34 are picked as a test set while the system is trained on the remaining 200 files. Both the test set and the training set consist of an equal amount of bass and flute samples. The best parameter settings for the liquid columns and the readout neural network as found in a number of experiments are given in Tables 5 and 6.

The optimal value of 50 units in the hidden layer of the multilayer perceptron is somewhat surprising. Networks with smaller numbers of hidden units are outperformed by this setting, as are those with larger numbers. Normally, such a large number would cause overfitting, but with the early stopping criterion – which causes the training to stop when the output for the correct outcome is at least 1.5 times as large as the incorrect classification – this problem is alleviated. The large number of hidden units combined with a

Table 7. Results for the musical instrument classification task

Summation over time	Average	Best	SD
All samples	85.0%	94.7%	5.4
MV over time (correct)	94.6%	100%	4.0
MV over multiple classifiers (correct)	85.3%	91.8%	4.6
MV over multiple classifiers (undecided)	0.6%		1.3
MV over multiple classifiers and time (correct)	95.9%	100%	3.4
MV over multiple classifiers and time (undecided)	0%		0
Spiking neural network liquid	**Average**	**Best**	**SD**
LSM (average correct over all samples)	88.4%	100%	5.2
MV over time (correct)	95.9%	100%	4.1
MV over time (undecided)	2.4%		1.5
MV over multiple liquid columns (correct)	91.9%	100%	4.8
MV over multiple liquid columns (undecided)	1.4%		2.6
DLSM (correct)	**99.1%**	**100%**	**2.6**
DLSM (undecided)	0.3%		0.3

rule that ensures generalization enables the system to take more features into account than it would otherwise be able to.

4.4 Experiments

Since the liquids that are used consist of a set of randomly connected nodes it is expected that the performance of individual LSMs will not be steady over the entire frequency range. We hypothesize that the DLSM having multiple different liquid columns, minimizes this problem and thus will outperform the single-column LSM.

A comparison between a single-column LSM in which the activity is read once per file, and several possibilities of applying majority voting over multiple readout moments and the results of multiple liquid columns is given in Table 7. For the results of both majority voting over multiple liquid columns and the DLSM, 10 combinations of a liquid and readout mechanism were tested. All experiments were repeated 10 times to get a more reliable average.

For comparison purposes, a simpler algorithm is also investigated. This model uses a readout network with 15 units in the hidden layer and a simple 64-band 'summation filter' replacing the liquid. The filter sums all the inputs per frequency band in a time span of 200 time steps, after which the readout function is called for classification and the filter is reset. This algorithm is also investigated in several majority voting settings. The results are given in Table 7.

For all setups, the classification accuracy for bass guitar samples is about the same as that for flute samples. Even though the simple summation algorithm performs well, it is outperformed by LSMs. And, as expected, LSMs in

the DLSM setting structurally outperform single-column LSMs. The improved classification accuracy on individual readout moments has a marked effect on classification of entire audio files, regardless of the type of sample (e.g. from CD, single tone, interval, etc.). The only times when the DLSM incorrectly classifies certain samples occur when there are no comparable samples in the training set.

The performance of individual LSMs is fairly good, but leaves enough room for improvement. As demonstrated in the experiments in this section applying majority voting variants can increase the classification accuracy. In the DLSM setting, a marked increase in classification accuracy is observed for all samples, effectively going from an average of 88% to an average of 99% correctly classified samples. This is comparable to results obtained by using extensive preprocessing of the input (98%, [Essid et al., 2004]) and exceeds the results that a simpler setup yields (96%, Sect. 4.4) significantly.

5 Conclusion

The experiments in the previous sections show that the DLSM outperforms a configuration in which majority voting is not applied over multiple liquid columns. The reason for this is that using ensembles of multiple classifiers can significantly boost performance, especially when there is a large variance between the individual classifiers. As the combination of a liquid column with a trained readout mechanism involves its own kind of equivalence over input patterns, the way in which such a combination decides the class of an input pattern is probably different for each liquid, and the variance of the error will also be high. The concept of majority voting over multiple classifiers is therefore highly applicable to the LSM.

Majority voting as applied in the temporal dimension in the DLSM minimizes the chance of judgment errors due to atypical liquid states. These can be caused by many things, such as a momentary lapse in the signal, a high amount of noise that drowns out the signal completely or a partial input pattern that resembles a property of another class. Since input samples will yield a sufficiently characteristic liquid state most of the time, using multiple readout moments reduces the chances of misidentification of an input pattern due to atypical periods in the input pattern. As becomes clear from the experiments, majority voting over the temporal dimension also seems to be a good strategy.

A possible improvement to the DLSM is to use liquids with different characteristics. As stated before, the DLSM operates by the principle that different liquid columns represent other aspects of the input data. The difference between liquid columns can not only be achieved by using columns that are initialized with different random seeds, but also by changing the parameters of the liquid columns. Maass et al. [Maass et al., 2002] for example, used multiple liquid columns with different values of the connection parameter λ, but

it is also possible to change other parameters or even the network topology to increase the difference between liquid columns.

Since the liquids we used in the experiments are randomly generated, the separation between liquid states of patterns belonging to different classes is not optimized. We are currently investigating optimization techniques such as reinforcement learning to optimize the separation property, which could reduce the needed size of the liquid and increase generalization performance.

References

Bakker, B., Zhumatiy, V., Gruener, G., Schmidhuber, J., 2003. A robot that reinforcement-learns to identify and memorize important previous observations. In: Proceedings of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2003). pp. 430–435.

Bengio, Y., Simard, P., Frasconi, P., March 1994. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks 5 (2), 157–166.

Breiman, L., 1996. Bagging predictors. Machine Learning 24 (2), 123–140.

Essid, S., Richard, G., David, B., 2004. Musical instrument recognition based on class pairwise feature selection. In: ISMIR Proceedings 2004. pp. 560–568.

Feng, J., Brown, D., 2000. Integrate-and-fire models with nonlinear leakage. Bulletin of Mathematical Biology 62, 467–481.

Fernando, C., Sojakka, S., 2003. Pattern recognition in a bucket. In: Proceedings of the 7$^{\text{th}}$ European Conference on Artificial Life. pp. 588–597.

Gupta, A., Wang, Y., Markram, H., 2000. Organizing principles for a diversity of GABAergic interneurons and synapses in the neocortex. Science 287, 273–278.

Häusler, S., Markram, H., Maass, W., 2003. Perspectives of the high dimensional dynamics of neural microcircuits from the point of view of low dimensional readouts. Complexity (Special Issue on Complex Adaptive Systems) 8 (4), 39–50.

Hochreiter, S., 1998. Recurrent neural net learning and vanishing gradient. International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems 6 (2), 107–116.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Computation 9 (8), 1735–1780.

Jaeger, H., 2001. The 'echo state' approach to analyzing and training recurrent neural networks. GMD report 148.

Lam, L., Suen, C. Y., 1997. Application of majority voting to pattern recognition: an analysis of its behaviour and performance. In: IEEE Transactions on Systems, Man, and Cybernetics. Vol. 27. pp. 553–568.

Lin, T., Horne, B., Tiňo, P., Giles, C., 1996. Learning long-term dependencies is not as difficult with NARX networks. In: Touretzky, D., Mozer, M.,

Hasselmo, M. (Eds.), Advances in Neural Information Processing Systems. Vol. 8. MIT Press, pp. 577–583.

Maass, W., Natschläger, T., Markram, H., 2002. Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Computation 14, 2531–2560.

Narasimhamurthy, A., 2005. Theoretical bounds of majority voting performance for a binary classification problem. In: IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol. 27. pp. 1988–1995.

Natschläger, T., Bertschinger, N., Legenstein, R., 2004. At the edge of chaos: Real-time computations and self-organized criticality in recurrent neural networks. In: Proceedings of Neural Information Processing Systems. Vol. 17. pp. 145–152.

Pantev, C., Hoke, M., Lutkenhoner, B., Lehnertz, K., 1989. Tonotopic organization of the auditory cortex: pitch versus frequency representation. Science 246, 486–488.

Pantev, C., Hoke, M., Lutkenhoner, B., Lehnertz, K., 1991. Neuromagnetic evidence of functional organization of the auditory cortex in humans. Acta Otolaryngol Supply 491, 106–115.

Pearlmutter, B., 1995. Gradient calculations for dynamic recurrent neural networks: a survey. In: IEEE Transactions on Neural Networks. Vol. 6. pp. 1212–1228.

Rumelhart, D., Hinton, G., Williams, R., 1986. Learning internal representations by error propagation. Parallel Distributed Processing 1, 318–362.

Ruta, D., Gabrys, B., December 2000. A theoretical analysis of the limits of majority voting errors for multiple classifier systems. Tech. Rep. 11, ISSN 1461-6122, Department of Computing and Information Systems, University of Paisley.

Sapp, C., Liu, Y., 2006. The Haydn / Mozart string quartet quiz. Center for Computer Assisted Research in the Humanities at Stanford University. Available on: `http://qq.themefinder.org/`.

Vapnik, V., 1998. Statistical learning theory. John Wiley and Sons, New York.

Vreeken, J., 2004. On real-world temporal pattern recognition using liquid state machines. Master's thesis, Utrecht University, University of Zürich.

Williams, R., Zipser, D., 1989. A learning algorithm for continually running fully recurrent neural networks. Neural Computation 1, 270–280.

Williams, R., Zipser, D., 1995. Gradient-based learning algorithms for recurrent networks and their computational complexity. In: Chauvin, Y., Rumelhart, D. (Eds.), Back-propagation: theory, architectures and applications. Lawrence Erlbaum Publishers, Hillsdale, N.J., pp. 433–486.