Neural Process Lett (2016) 44:103–124
DOI 10.1007/s11063-015-9478-6

Computation by Time

Florian Walter[1] · **Florian Röhrbein**[1] · **Alois Knoll**[1]

Published online: 17 November 2015

Abstract Over the last years, the amount of research performed in the field of spiking neural networks has been growing steadily. Spiking neurons are modeled to approximate the complex dynamic behavior of biological neurons. They communicate via discrete impulses called spikes with the actual information being encoded in the timing of these spikes. As already pointed out by Maass in his paper on the third generation of neural network models, this renders time a central factor for neural computation. In this paper, we investigate at different levels of granularity how absolute time and relative timing enable new ways of biologically inspired neural information processing. At the lowest level of single spiking neurons, we give an overview of coding schemes and learning techniques which rely on precisely timed neural spikes. A high-level perspective is provided in the second part of the paper which focuses on the role of time at the network level. The third aspect of time considered in this work is related to the interfacing of neural networks with real-time systems. In this context, we discuss how the concepts of computation by time can be implemented in computer simulations and on specialized neuromorphic hardware. The contributions of this paper are twofold: first, we show how the exact modeling of time in spiking neural networks serves as an important basis for powerful computation based on neurobiological principles. Second, by presenting a range of diverse learning techniques, we prove the biologically plausible applicability of spiking neural networks to real world problems like pattern recognition and path planning.

Keywords Spiking neural network · Neurobiological learning · Reservoir computing · Hierarchical learning · Neural coding · Neuromorphic hardware

✉ Florian Walter
florian.walter@tum.de

Florian Röhrbein
florian.roehrbein@in.tum.de

Alois Knoll
knoll@in.tum.de

1 Institut für Informatik VI, Technische Universität München,
85748 Garching bei München, Germany

1 Introduction

Neural networks are among the oldest and most widespread concepts in artificial intelligence and machine learning. Their early beginnings date back to the time around 1950, when McCulloch and Pitts published their well-known neuron model [47] and when the *Perceptron* was conceived by Rosenblatt [59]. At that time, it was already known that biological neurons communicate via discrete impulses, so-called *action potentials* or *spikes*. A detailed model of the neural dynamics governing the generation of these action potentials was derived by Hodgkin and Huxley in a seminal study of the squid giant axon [27]. However, due to the complex mathematics involved in computing the neural state updates, this model is no viable way for simulating neural networks at large scale even today. As a result, researchers developed phenomenological approximations. With computing power having become cheaper and cheaper over the last decades, these approximations where further refined and extended. In [42], Maass identifies three distinct generations of neural networks. Networks of the first generation are composed of neurons as introduced by McCulloch and Pitts and thus only produce digital output, which can be interpreted as the presence or absence of a spike. In the second generation, the digital threshold logic of the neuron was replaced by the more general concept of an activation function which computes continuous output based on the input provided by its presynaptic afferents. From a biological point of view, the real-valued output signal can be interpreted as a spike rate. The computational power of this concept was confirmed by proving that networks of the second generation are *universal function approximators* as long as they satisfy a small number of relatively weak requirements [28].

Neural networks of both generations have yielded remarkable results in such diverse applications as optical character recognition, natural language processing and robot control. Recently, especially *deep neural networks* and *deep learning* have become popular due to their unprecedented performance in supervised classification tasks [38,63]. However, while the main motivation of early artificial neural networks was the study of neural information processing in biological organisms, most current state-of-the-art neural architectures and algorithms hardly bear any similarities to their biological counterparts. This has motivated the development of a third generation of neural networks which captures the full temporal dynamics present in the nervous systems of biological organisms. Unlike in the previous generations, the internal processes responsible for the emission of spikes are modeled explicitly instead of just computing an abstract firing rate based on simplified activation functions. Available neuron models range from exact quantitative biological descriptions like the already mentioned Hodgkin-Huxley model to phenomenological approximations like the Leaky Integrate-and-Fire neuron [19] or the Izhikevich neuron [29]. One can prove that, independently of the actually chosen model, spiking neural networks are computationally more powerful than their predecessors from the first two generations [42]. More importantly, they build a bridge between the fields of machine learning, artificial intelligence and robotics on the one hand and neuroscience on the other. Since spiking neural networks no longer abstract from biological processes but try to reproduce them as closely as possible, new findings can be easily exchanged among the different disciplines. The EU-funded Human Brain Project [69] is specifically designed to promote these synergistic effects by integrating all available knowledge about the human brain into a large-scale simulation. The results are intended to advance not only neuroscience but also future computing technologies which are based on the principles of neural information processing.

From a biological point of view, the only means for neurons to transfer information is to emit spikes. However, every spike is just an impulse and does not carry any data. Experimental evidence suggests that the actual payload must therefore be encoded in the *timing* of the spikes [3]. This clearly points out a major drawback of neuron models from the first and second generation which convey only very limited timing information by indicating whether a spike has occurred at all or by computing a firing rate. Only spiking neural networks are able to produce precisely timed spikes as observed in experiments and to implement neurobiological mechanisms of learning which rely on the relative timing of subsequent spikes. As already stated by Maass, this opens up the possibility of "using *time* as a resource for computation and communication" [42].

While the biological plausibility and the superior computational power of spiking neural networks have motivated the development of a huge variety of different methods and technologies for neural information processing based on spikes, a detailed review of how spike timing and time in general can serve as a means of computation is still missing. In this paper, we aim to fill this gap by investigating the role of time from three different but coherent perspectives. Section 2 focuses on single neurons and gives an overview of algorithms for learning sequences of spikes and corresponding applications. In Sect. 3, we illustrate the role of time at network level and explain how carefully adjusted relative timing can yield higher-level cognitive features. When computing the output of a spiking neural network for the purpose of pure simulation, the execution speed is usually irrelevant in terms of the results. This changes when the network is attached to an external physical system like a robot which operates in real-time and in the real world. For this reason, Sect. 4 examines how neural simulation software and novel neuromorphic hardware support the execution of spiking neural networks at appropriate timescales. In Sect. 5, we finally conclude the paper by summarizing the principal results of our review.

2 How Spiking Neurons Learn: The Role of Time at the Neural Level

In spiking neural networks, every single neuron is a complex dynamical system. Unlike neural models which are based on threshold logic or activation functions, the output not only depends on the current input received from other neurons but also on an internal state which evolves over time. This enables spiking neurons to respond individually to differently timed sequences of input spikes. In the following, we investigate how these temporal dynamics enable single neurons to perform meaningful computation. The first subsection briefly introduces the foundations of modeling spiking neurons mathematically. Furthermore, different types of neural coding schemes are discussed to demonstrate how to encode information based on the timing of spikes. In the second subsection, we review selected learning algorithms which are specifically designed for spiking neural networks. By providing a means to control the neural response to patterns of input spikes, these algorithms make spiking neurons capable of performing tasks like classification or the generation of specific output patterns and thereby realize neural computation by time.

2.1 Biological and Mathematical Foundations

Before a learning algorithm can be applied to a spiking neuron, one must select the underlying neuron model and choose an appropriate neural code. The following paragraphs provide the required neuroscientific background.

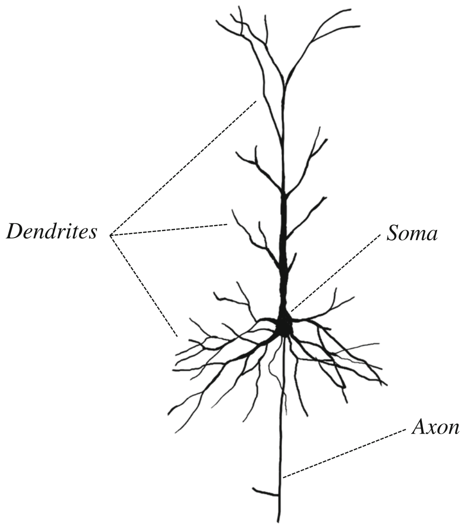

Fig. 1 Basic components of a neural cell. The drawing depicts a pyramidal cell from the cortex of a rat. Drawn and extended based on a figure from [11]

2.1.1 Mathematical Models for Spiking Neurons

Biological neurons have a branching spatial structure and an internal state which is determined by chemical processes. An example can be seen in Fig. 1 which depicts a drawing of a cortical pyramidal cell. Incoming spikes travel along the *dendrites* towards the *soma* and elicit a change of the neuron's *membrane potential*. As soon as this potential crosses a threshold, a spike is generated and propagated along the *axon*. Synapses between the axon and the dendrites of other neurons transmit the spikes between the cells. In spiking neural networks, this complex structure is usually simplified by assuming that every neuron is just a single point. Moreover, as already stated earlier, detailed conductance-based models like the one derived by Hodgkin and Huxley are often replaced by phenomenological approximations. One of the simplest and also most commonly used models of this type is the Leaky Integrate-and-Fire neuron, which is characterized by the following equations [19]:

$$\tau_m \frac{\mathrm{d}u}{\mathrm{d}t} = -u(t) \; + \; R\, I(t) \tag{1}$$

$$t^{(f)} : \quad u\left(t^{(f)}\right) = \vartheta \tag{2}$$

Equation 1 models the membrane potential u of the neuron which is thought of as an electrical circuit consisting of a capacitor C and a linear resistor R. The factor $I(t)$ denotes the electrical current and is composed of a leak term and an external driving input current. τ_m denotes the membrane time constant and is decisive for the temporal behavior of the neuron. The emission of spikes is described by Eq. 2 which states that the neuron fires at time $t^{(f)}$ when the membrane potential reaches a threshold value ϑ from below. Note that not all spiking neuron models are based on differential equations. For example, a model proposed by Thorpe simply computes a weighted sum of all incoming spikes and assumes that every neuron only emits at most one spike [62]. A new spike can only be generated after resetting the neuron state. Stochastic models describe neural dynamics using tools from probability theory in order to allow for the incorporation of noise.

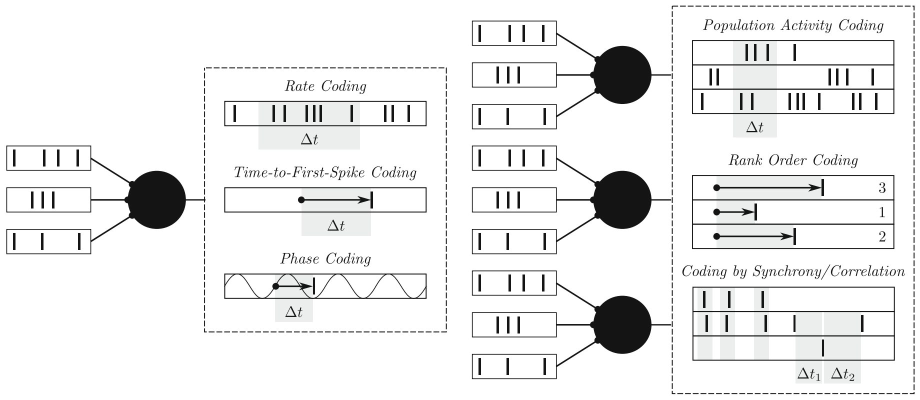
Rate Coding
Δt
Time-to-First-Spike Coding
Δt
Phase Coding
Δt
Population Activity Coding
Δt
Rank Order Coding
3
1
2
Coding by Synchrony/Correlation
Δt1 Δt2

Fig. 2 Schematic overview of selected neural codes. Neurons are depicted by the *black circles*. Input spike sequences from presynaptic afferents are indicated by the simplified spike raster plots on the *left* of the neurons. The actual coding schemes are shown on the *right*

2.1.2 Neural Coding

Based on experimental evidence and theoretical considerations, different types of potential neural codes have been conceived. However, the general problem of deciphering the neural code which is actually implemented in the brain still remains an open question [19]. In the following, we will provide an overview of commonly employed coding schemes based on the reviews by Gerstner et al. [19], Ponulak et al. [57] and Grüning et al. [22]. These reviews also provide extensive pointers to references discussing the problem at a more elaborated level of detail which is beyond the scope of this paper. The schema in Fig. 2 gives a graphical summary of the neural codes presented in the next two paragraphs.

When considering only a single neuron, *rate coding* is the possibly simplest way of encoding information with spikes. As mentioned before, rate codes do not necessarily require spiking neuron models but can also be implemented using second generation networks with continuous activation functions. This is not true for *Time-to-First-Spike* coding which relies on the precise timing of a single spike. All information is carried solely in the latency time of this spike with respect to some stimulus. *Phase coding* is a slightly modified approach where the measurement of the relative timing is not based on the onset of an event but on periodic background oscillations in the considered neural system. Thus, timing in this context refers to a phase shift. Oscillatory behavior has been observed in biological nervous systems in the form of rhythmical spike patterns emitted by populations of neurons. In a somewhat degenerated version of single-spike codes, the whole information is carried by the fact that a spike has occurred or not [24]. The time of spike emission is considered irrelevant.

While the codes presented above are equally suited if more than one neuron is involved, other coding schemes are explicitly defined for populations of neurons. *Population activity coding* is an augmented rate code for neural populations. Assuming that a single neuron receives synaptic input from all neurons belonging to a certain population, it can measure their overall activity, i. e. the fraction of neurons emitting spikes within a certain period of time. Like rate coding for single neurons, encoding information solely via the overall activity of a huge neuron population seems rather inefficient. *Rank order coding* therefore takes into account the firing order of a selected group of neurons. A drawback of this approach is that at most one spike per neuron is considered. Neural codes based on *synchrony* or *correlation*

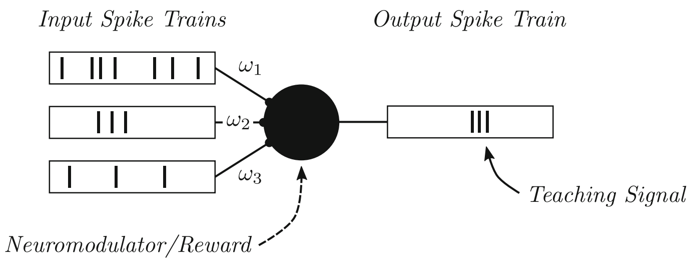

Fig. 3 Learning in spiking neural networks. In analogy to Fig. 2, the *black circle* denotes a single spiking neuron. The presynaptic input spike trains are depicted on the *left*. The parameters ω_i store the synaptic properties governing the spike transmission. The input is processed by the neuron and yields an output spike train which is shown on the *right*. In *unsupervised learning*, only the input spike trains are provided to the algorithm. *Reinforcement learning* relies on an additional reward signal to guide the adaption of network parameters. In *supervised* learning scenarios, the desired output spike train has to be provided explicitly as teaching input

encode information by the synchronous firing of selected neurons or by precisely defined correlations between the timing of their spikes, respectively. Finally, any sequence of spikes emitted by a single neuron or a population of neurons can be regarded as a neural code.

2.2 Learning Techniques for Spiking Neurons

A specific combination of a neuron model and a coding scheme defines a basic framework for neural information processing based on spikes. However, to achieve a desired behavior or to implement a certain functionality, additional parameters need to be set. Like in classical artificial neural networks, a graph-like network structure must be defined in order to specify the number of neurons and their connectivity. The flow of information within the network can be controlled by adjusting synaptic efficacies (i. e. the weights of the edges of the network graph) and other parameters like the delays of synaptic spike transmission. Positive efficacies indicate that a synapse is *excitatory*, which means that the membrane potential increases on spike arrival. Negative synaptic weights denote *inhibitory* synapses which decrease the membrane potential in response to incoming action potentials. Note that while it is easily possible for a model synapse to switch between excitatory and inhibitory mode by simply adapting the sign of the corresponding weight, the properties of biological synapses depend on physiological parameters and thus cannot alter between excitation and inhibition [34]. Although methods for automatically deriving or adapting the topology of neural networks have been proposed [13,52,72], the focus of research is clearly on algorithms for learning optimal values for the synaptic weights. Figure 3 illustrates the basic learning task for a single spiking neuron. Based on the incoming spike trains, the general goal of all learning algorithms for spiking neurons is to adjust the synaptic efficacies to make the neuron produce the desired output spike train. Over the last years, the research in machine learning and computational neuroscience has yielded a huge and diverse variety of different learning techniques, ranging from approaches specifically designed for a certain neuron model to more general algorithms only relying on basic properties which are common to all spiking neurons. Another important classification criterion is the biological plausibility. It describes whether the neural implementation of a learning algorithm is in accordance with data available from nervous systems in biological organisms. The next paragraphs provide an overview over the current state of the art in spike time-based neurobiological learning. Using the common notions from the field of learning, we will distinguish between methods for *unsupervised learning*, *reinforcement learning* and *supervised learning*.

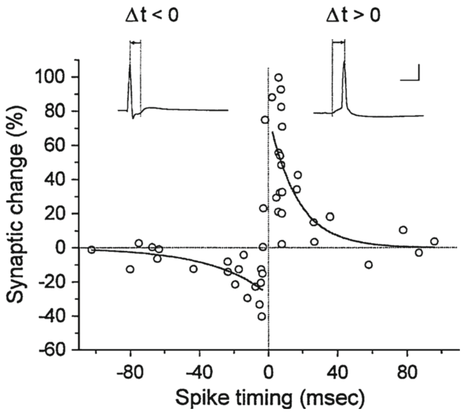

Fig. 4 Plot of a STDP learning window. The two curves which are fitted to the experimental data indicate how the relative timing between presynaptic and postsynaptic spikes relates to changes in the synaptic efficacy. From [2]. Reproduced with permission of Annual Review of Neuroscience, Volume 24 © by Annual Reviews, http://www.annualreviews.org

2.2.1 Unsupervised Learning

In unsupervised learning tasks, the system is only provided with a set of input values. There is neither information available on the desired output nor any kind of feedback signal. In classical machine learning, the main applications for unsupervised methods are clustering and pattern recognition. The absence of any kind of external supervision makes unsupervised learning a central mechanism in self-organizing distributed biological information processing. In the context of spiking neural networks, it enables a natural way of biologically plausible learning since every neuron in the network can adjust its synaptic efficacies solely based on information which is available locally. Two of the most basic neurobiological learning techniques of this class are *habituation* and *sensitization* [33]. Habituation describes a decay of synaptic efficacy caused by the repeated occurrence of a stimulus. On the behavioral level, an organism habituates when it becomes insensitive to an irrelevant stimulus. For example, people living near a freeway get used to the noise over time until they no longer care about it. *Sensitization*, the opposite phenomenon, can be triggered by unpleasant and harmful events and causes an increase of the neuron's synaptic efficacies. Therefore, it can also counteract previous habituation. Depending on the duration of a series of stimuli, both habituation and sensitization can yield short-term or long-term synaptic plasticity, which again highlights the importance of time.

The plasticity effects explained above mainly depend on the occurrence frequency of some input event rather than the precise timing of single spikes. The importance of the latter was anticipated by Hebb already in 1949 [25] when he stated that synaptic plasticity might be induced if the firing of a postsynaptic neuron is repeatedly causally related to the firing of a presynaptic neuron. More recent work in both theoretical [18] and experimental neuroscience [45] formalized and confirmed this early hypothesis with the discovery of *spike-timing dependent plasticity* (STDP) [65]. STDP adapts the efficacy of a synapse based on the relative timing of spike emission by its two adjacent neurons. This is illustrated in Fig. 4. If the postsynaptic neuron fires a spike after the presynaptic neuron within a certain time window, i. e. $\Delta t > 0$, the synaptic connection is strengthened. Otherwise, if $\Delta t < 0$, there is no causal relationship between the two spikes and the synapse is depressed. Based on the original *learning window* from Fig. 4, a huge number of different variations has been proposed [49]. For example, in anti-STDP non-causal spikes yield potentiation and causal spikes elicit synaptic depression. More recently, a study of a STDP learning rule which is based on triplets of spikes rather than pairs was published [54].

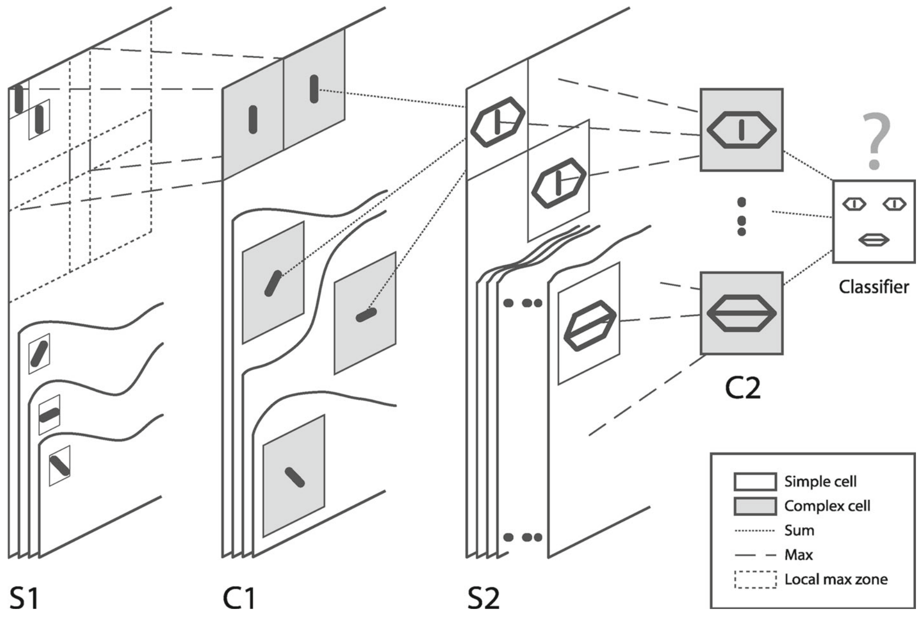

Fig. 5 Unsupervised learning of visual features based on STDP and spiking neurons. Input is processed by a feedforward neural network architecture with four hierarchical layers. Simple cells in the S1 layer perform edge detection based on a convolution operation on the input image. The complex cells in layer C1 compute a *max*-operation by only propagating the first spike received from the corresponding S1 cells. Analogously, S2 neurons detect visual features of intermediate complexity while layer C2 performs another *max*-operation. STDP is implemented by the synapses between layer C1 and layer S2. There are multiple instances of all layers in S1, C1 and S2 in order to achieve scale-invariance. The extracted visual features can serve as input for an image classification algorithm. Reprinted from [46]. This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

It is obvious that the simple learning rules from above alone not automatically give way to advanced cognitive capabilities. To fulfill their potential, they must be embedded in an appropriate neural network. The authors of [46] propose a layered feedforward architecture for unsupervised feature detection which is inspired by the visual ventral pathway as described in the HMAX model. Figure 5 provides a schematic overview. The first layer S1 contains neural cells performing edge detection in the visual input based on a convolution operation. There are four different types of cells, yielding a total of four retinotopic maps, each of which is sensitive to a different edge direction. On presentation of a new input stimulus, the cells with the best matching direction emit their spikes first and thus realize a time-to-first-spike code. Inhibition between the retinotopic maps ensures that only one spike is emitted for each position. All subsequent layers are comprised of Leaky Integrate-and-Fire neurons. The neurons in layer C1 perform a *max*-operation by only forwarding the earliest arriving spikes from predefined sampling regions. An STDP process which adapts the synaptic weights between cells in layers C1 and S2 makes individual S2 neurons sensitive to specific visual features of intermediate complexity. The learned weights are shared between the different retinotopic maps using a winner-take-all mechanism. In addition, the complete S1–C1–S2 pathway is instantiated multiple times to support processing at different scales. The neurons in layer C2 finally select the maximum input from all retinotopic maps and all scales, which enables position- and scale-invariant feature detection. When presented with a series of input images, STDP extracts common visual features in a self-organized manner while the inhibitory winner-take-all mechanisms prevent the system from storing redundant information.

In addition to the approach discussed above, many other STDP-based unsupervised learning methods have been conceived. In [5], a new population coding scheme is introduced which allows for an efficient clustering of data using a network of spiking neurons. The underlying learning rule was originally developed in [50] and realizes a winner-take-all mechanism by potentiating synapses which were active only a short time before the emission of a postsynaptic spike. In [10], the authors combine STDP and habituation to create a network with only three neurons which implements operant conditioning. Finally, anti-STDP is used in [55] for a spiking neuron-based implementation of the wavefront path planning algorithm.

2.2.2 Reinforcement Learning

The range of applications for unsupervised learning is narrowed down by the fact that there is no way of guiding the learning process towards a desired goal. Reinforcement learning overcomes this limitation by not only providing input data but also a *reward signal* which encodes feedback for the produced output. Since the reward is delivered dynamically in response to interactions with an external system, reinforcement learning is especially suited for applications like robotics which involve interactions with a physical environment. From a neural perspective, reinforcement learning adds a third factor to the learning process besides presynaptic and postsynaptic spike times [15]. In general, the synaptic efficacy change Δw can therefore be expressed as follows (from [64]):

$$\Delta w = R \cdot \mathrm{PI} \tag{3}$$

In the above equation, R denotes the received reward and PI the *plasticity induction* resulting from the spikes emitted by the presynaptic and postsynaptic neurons. A possible example for the choice of PI is the STDP learning window from Fig. 4. The main challenge of reinforcement learning at the neural level is to determine to which neurons and synapses the current reward applies. Compared to the duration of a spike or the length of the learning window, which are in the order of milliseconds, the delivery of the reward depends on the environment and can be delayed by several seconds. *Reward-modulated STDP* (R-STDP) solves this issue by storing spikes which are potential candidates for eliciting synaptic change in an *eligibility trace* as depicted in Fig. 6. The temporal contribution of related pairs of spikes to the trace is determined by an eligibility function. As soon as a reward signal is received, all synaptic weights are updated by computing the product of the reward and the current value of the corresponding synaptic eligibility trace. In [30], a biologically inspired implementation of R-STDP is proposed. Every synapse is assigned an additional synaptic tag which stores the eligibility trace and decays over time if no further spike pairs occur. In analogy to biological nervous systems, reward is emitted in form of the neuromodulator dopamine. The synaptic change is finally computed as a product of the current value of the synaptic tag and the dopamine concentration. Theoretical studies based on statistical modeling have proven the computational power of R-STDP which enables a neuron to learn classification tasks and spike sequences [40]. A study focusing more on exact modeling in accordance to experimental results is available in [12]. Importantly, the authors point out the problems involved in teaching a set of neurons to reproduce individual sequences of spikes when only a single global reward signal is available. The problem of credit assignment is completely neglected. Instead, a possible biological implementation of reward prediction is discussed theoretically.

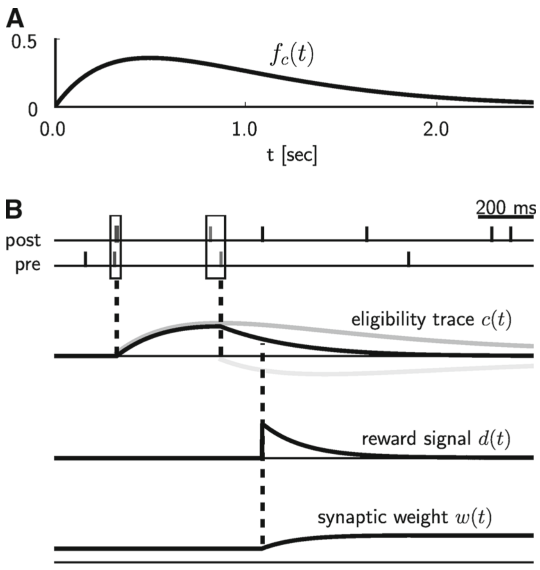

Fig. 6 Credit assignment in reward-modulated STDP through eligibility traces. The eligibility function $f_C(t)$ in the upper part determines how pairs of spikes contribute to the eligibility trace $c(t)$ below. The synaptic weights $w(t)$ are adjusted based on the eligibility trace and the reward signal $d(t)$. Reprinted from [40]. This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

A common way of making neurobiological reinforcement learning methods analytically tractable is to embed them into classical frameworks from the field of machine learning. This allows for mathematical deduction and analysis of new learning rules based on *Policy Gradient* methods and *Temporal Difference* learning. Algorithms of the former type adapt synaptic weights by computing the gradient of a function which estimates the expected reward [64]. In [14], a policy gradient algorithm for reinforcement learning in partially observable Markov decision processes is employed to derive a reinforcement learning rule for spiking neurons. Temporal difference (TD) learning is a more advanced approach since it not only tries to maximize the reward of the next action but the complete future expected reward. As in the case of Policy Gradient methods, this requires an approximation function. During runtime, this function is continuously updated. A neural implementation of such a so-called critic is developed in [15]. The authors introduce the resulting learning rule as TD-LTP to highlight that synaptic plasticity is modulated by a product of the TD error and synaptic long term potentiation. A comprehensive overview of further reinforcement learning algorithms is available in [64].

2.2.3 Supervised Learning

Supervised neural learning techniques provide the most direct control over the output of a neuron. In terms of Fig. 3, the shape of the output spike train produced in response to a given input can be fully controlled by a teacher. Up to now, all learning methods considered were defined locally for single neurons. The additional reward signal in reinforcement learning was defined globally at network scope. All learning rules considered so far thus worked independently from the connectivity of the neural network. In contrast, supervised learning methods must consider all output neurons individually in order to make them reproduce predefined sequences of spikes. This explains why many algorithms for supervised neural learning only work for single neurons. In the following, we will therefore state explicitly if a method supports neural networks with hidden neurons and more complex connectivity patterns. Like before, the focus will be on the learning of precisely timed spikes rather than firing rates.

Basic unsupervised neural learning based on STDP can be used to implement a simple mechanism for supervised learning of sequences of spikes. In *Supervised Hebbian Learning* [57], the emission of the desired spike trains is forced by current injection into the corresponding neurons. While this approach ensures that the neuron learns to emit spikes at the desired firing times through STDP, unwanted spiking is suppressed by the current injection and thus has no effect on the synaptic strengths. Convergence can consequently only be proofed on average when modeling the input and output via Poisson spike trains [39]. The *Remote Supervision Method* is a more advanced supervised learning algorithm which implements an adaption of the Widrow-Hoff learning rule from classical artificial neural network theory for spiking neural networks [56]. It employs relative spike timings to adjust the synaptic efficacies and works without currency injection (adapted from [56]):

$$\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t) = X_{dj}(t) + X_{ij}(t) \tag{4}$$

The synaptic weight change $\frac{\mathrm{d}}{\mathrm{d}t} w_{ij}(t)$ between the postsynaptic neuron i and the presynaptic neuron j is computed as a sum of an STDP process $X_{dj}(t)$ and an anti-STDP process $X_{ij}(t)$. While the former models the correlation between the desired output and the input spike train provided by the teacher, the latter models the anti-causal correlation between the presynaptic and postsynaptic spikes. The convergence of ReSuMe is theoretically proven. The original version only works for single layer networks. A gradient-based extension for multilayer networks with hidden neurons is derived in [66].

The vast majority of supervised learning algorithms for spiking neural networks is not directly based on STDP. A very recent method is the *Finite Precision* (FP) learning algorithm presented in [48]. Based on the assumption that in practice no infinitely accurate spike timing is required, every spike in the sequence to be learned may occur within a certain tolerance window of width ϵ. The actual process of learning is illustrated in Fig. 7. The topmost graph depicts the initial output of the neuron and the time windows for the desired spikes. The initial spike train contains three errors. The FP algorithm iteratively corrects one error after the other starting from the first one. This avoids the uncontrolled emergence of new errors due to "crosstalk" caused by the nonlinear neural dynamics. Weight updates are proportional to the neuron's postsynaptic potential. The authors were able to prove both the convergence and the stability of the algorithm and also demonstrated its applicability to networks with recurrent connections.

A quite different approach for feedforward networks with hidden layers is proposed in [4]. The *SpikeProp* algorithm is an adaption of the well-known backpropagation algorithm for spiking neural networks based on the Spike Response neuron model. It adapts synaptic weights by computing the gradient of a linearized version of the thresholded membrane potential and propagates the error back to the hidden network layers. A major drawback of the original SpikeProp algorithm is that it supports only a single spike per neuron. However, later extensions enable it to process sequences of spikes [57]. Besides SpikeProp, there is a very broad and diverse range of many other learning algorithms based on gradient descent. In particular, some of theme use stochastic neuron models. The approach discussed in [6] supports learning in recurrent networks by minimizing an upper bound on the Kullback-Leibler divergence which is used as a means of measuring the difference between statistically modeled spike trains.

All methods considered up to now can *encode* information in precisely timed sequences of spikes. For classification, it is already sufficient to emit a certain number of spikes to classify input signals. Although this *decoding* tasks seems to be simpler at first glance, it is pointed out in [23] that the optimal choice of the times when to emit the output spikes

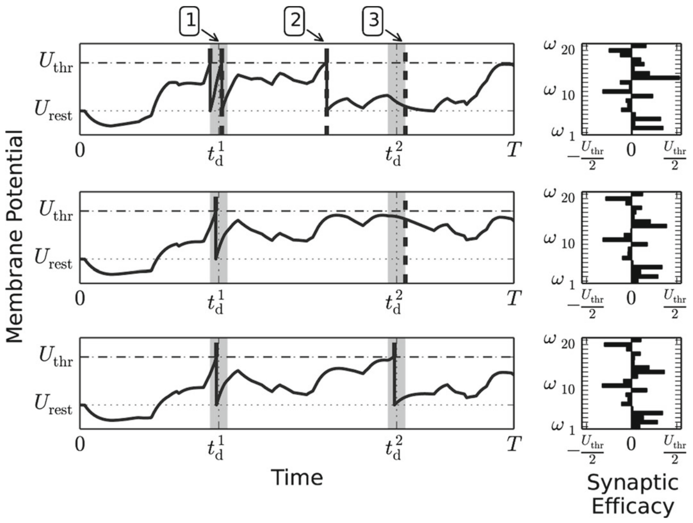

Fig. 7 Synaptic weight adaption with *Finite Precision* (FP) learning. At the beginning, the first error is corrected. This automatically also cancels the unwanted output spike in the *middle*. Finally, the synaptic weights are adjusted to add the second spike at the end of the sequence. Some intermediate states are skipped. Adapted and reprinted from [48] with permission from Elsevier

adds additional complexity to the learning problem. The *Tempotron* learning rule introduced in [24] enables a single neuron to recognize two distinct classes of input by the emission of a single spike. Like the algorithms before, the method is gradient-based. If no spike is produced for a positive sample synaptic weights are adjusted by adding a scaled value of the neuron's maximum membrane potential which occurs within the considered time window. False positives are corrected by subtracting this scaled potential. This procedure minimizes the deviation of the maximum membrane voltage occurring in response to a falsely classified input pattern from the neuron's firing threshold.

With the exception of the SpikeProp algorithm, all of the learning methods above were designed with biological plausibility in mind. However, also more abstract methods focusing solely on computational aspects have been conceived. The authors of [8] employ methods from linear algebra to define a vector space of spike trains. For a linear neuron model, synaptic weights can be computed via orthogonal projection. An iterative approximation of the projection operation can also be applied to Leaky Integrate-and-Fire neurons. Finally, a novel approach introduced in [73] transforms the problem of learning sequences of spikes to a classification problem which can be solved using the Perceptron learning rule for non-spiking artificial neural networks. The resulting *Perceptron-Based Spiking Neuron Learning Rule* is developed based on the Spike Response neuron model. An application of the method to other types of neurons requires individual mathematical derivations.

3 The Role of Timing at Network Level for Advanced Cognitive Capabilities

The previous section covered the role of time at the level of single neurons. In the following, we provide a more coarse-grained perspective by reviewing selected aspects of time in networks

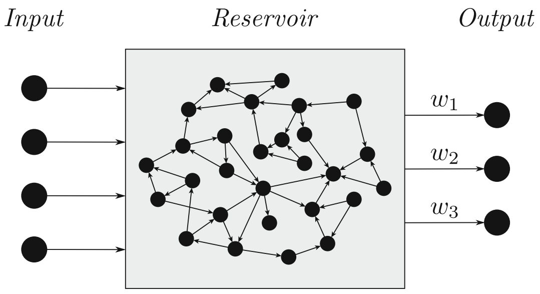

Fig. 8 Schematic illustration of the three main components of reservoir computing. Input units provide data to the recurrent network in the reservoir. The current state of the reservoir is accessed by another separate set of output units

of neurons. In particular, we present two different concepts which rely on temporal dynamics to perform complex computations. The first subsection introduces the concept of *Reservoir Computing* with a special focus on *Liquid State Machines*. An even higher level of granularity will be considered in the second subsection where we explore the interactions between different networks of neurons which operate at individual timescales.

3.1 Reservoir Computing

The additional computational power enabled by accurate modeling of neural dynamics was a central point in the last subsection. The fact that a dynamical system can compute more than a static structure was early recognized in neural networks research and led to the investigation of network architectures with recurrent connectivity. Independently from the underlying neuron model, recurrent edges add memory to the network and thus yield temporal dynamics at the network level. There is no need to say that the enhanced computational capabilities enabled by these dynamics come at the price of increased complexity in the learning phase. Algorithms for recurrent neural networks like *Backpropagation Through Time* [71] suffer from vanishing gradients and convergence problems [41]. *Reservoir Computing* is a new learning paradigm for recurrent neural networks which has the potential to overcome these limitations [43]. The basic principles and components of reservoir techniques are summarized in Fig. 8. The central element is the *reservoir*, a usually randomly generated recurrent neural network with fixed synaptic weights. External input is provided via separate input units which are connected to the reservoir based on some connectivity pattern. The weights of the synaptic connections emerging from these units are fixed, too. Another set of dedicated output units is driven by the internal dynamic state of the reservoir and produces the actual output. The synaptic weights w_i of these neurons are adjusted using some selected learning algorithm. Consequently, there is no need to perform complex learning in the recurrent reservoir network. The basic idea behind this approach is that the dynamics of the reservoir project the input into a high-dimensional space representation which allows for a simple mapping to the desired output.

The *Liquid State Machine* (LSM) is an instantiation of the reservoir paradigm specifically targeted at recurrent networks of spiking neurons as found in the brain [44]. It is based on the idea of a liquid which is perturbed by some external influence. The temporal dynamics caused by this perturbation form continuous states which carry information about both their past and their future behavior until they finally fade away. A basic description of the overall approach is given by the following two equations (from [44]):

$$x^M(t) = \left(L^M u\right)(t) \tag{5}$$

$$y(t) = f^M\left(x^M(t)\right) \tag{6}$$

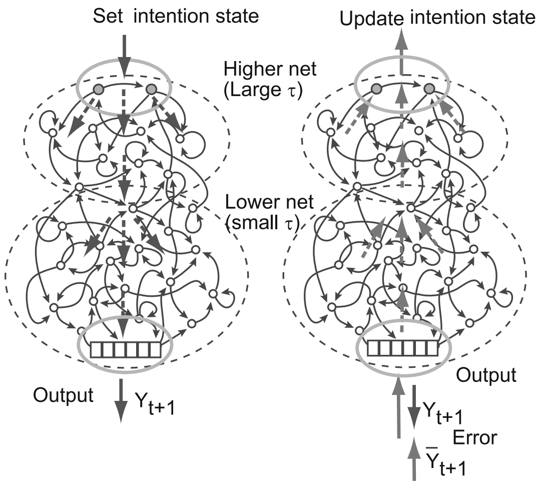

Fig. 9 Hierarchical learning in a *Multiple Timescales Recurrent Neural Network* (MTRNN). The network architecture is based on two complementary pathways for propagating intentions *top-down* (*left side*) and acquiring sensory information *bottom-up* (*right side*). While the *top-down* direction predicts, the *bottom-up* direction is responsible for learning. The high-level network runs at a slower timescale than the low-level network. © 2014 IEEE. Reprinted, with permission, from [68]

The first line describes the mapping of the input spike train $u(t)$ to the high-dimensional representation $x^M(t)$ by a *liquid filter* L^M. In Fig. 8, the filter corresponds to the neural network, i.e. the reservoir.[1] Equation 6 computes the output $y(t)$ by applying a *memoryless* readout map $f^M(t)$ to the current liquid state $x^M(t)$. The theory of LSMs introduces a new definition of computational power which is implemented by a concrete LSM if L^M fulfills the *Separation Property* (SP) and if $f^M(t)$ fulfills the *Approximation Property* (AP). Formal definitions of these properties are provided in [44]. In the same paper, the authors implement a liquid state machine with a random network of Integrate-and-Fire neurons. The synaptic efficacies of the output neurons are adapted using a modified perceptron rule.

Besides LSMs, there is a huge variety of other implementations of the reservoir computing paradigm. *Echo State Networks* were developed in parallel to LSMs but rely on rate-based neural networks of the second generation [41]. The output weights are typically computed using linear regression. In another LSM-based approach proposed in [53], the synaptic connections between the neurons of the liquid were modified via STDP which led to better results for real-world data. An extensive review of the vast amount of further work on reservoir computing methods is provided in [41].

3.2 Hierarchical Models of Learning and Cognition

In the reservoir computing paradigm discussed above, temporal dynamics are considered at the global scale of the underlying recurrent neural circuitry. However, processing all data at the same timescale might not always be appropriate. From a very high-level perspective, complex tasks like baking a cake are composed of many different subtasks like heating up the oven or fetching milk from the refrigerator. These subtasks are valid on a shorter timescale than the main task. In the following, we will study a conceptual framework presented in [68] which enables higher order cognitive capabilities through linked recurrent artificial neural networks operating at different dynamical timescales. The main components of this framework are depicted in Fig. 9. The schema illustrates that the approach is based on two complementary pathways. The first one is a *top-down intentional pathway* which generates action plans

[1] Note that although L^M is usually implemented as a spiking neural network, the theory of LSMs allows arbitrary implementations as long as they meet a set of formal mathematical requirements.

based on an intentional state and predicts the network output Y_{t+1} for the next timestep. The second pathway propagates perceptual information bottom-up and is responsible for correcting errors through learning. A neural implementation of this concept is defined by the *Recurrent Neural Network with Parametric Biases* (RNNPB). The top-down prediction computed by this network is given by the following update rule (from [68]):

$$(Y_{t+1}, X_{t+1}) = f\,(Y_t, X_t, W, \xi) \tag{7}$$

X_i denotes the internal network state and Y_i refers to the observable perceptual state. The prediction of these states at timestep $t+1$ depends on the synaptic weights and biases W and the intentional state ξ. Learning and thus also the bottom-up pathway are realized via backpropagation through time.

The key to a functional hierarchy with different timescales lies in the hierarchical connection of the RNNPBs defined above. The resulting state prediction for two concatenated networks is stated below (from [68]):

$$\begin{cases} (Y_{t+1}, X_{t+1}) = f^l\left(Y_t, X_t, W^l, \xi_t\right) \\ (\xi_{T+1}, X_{T+1}) = f^h\left(\xi_T, X_T, W^h, \xi^h\right) \end{cases} \tag{8}$$

The variable T denotes the slower timescale of the RNNPB at the higher level which predicts the intentional state ξ_t of the RNNPB at the lower level. Compared to the speed of the low-level network, the prediction for ξ_t changes slowly and thus provides a persistent context for the fast low-level dynamics. The discrete prediction steps in equation 8 are a major drawback of RNNPB networks. However, a continuous extension of the concept is available and the resulting hierarchies of networks are referred to as *Multiple Timescales Recurrent Neural Networks* (MTRNNs). In a series of real-world experiments with robots controlled by MTRNNs, the author of [68] was able to demonstrate the emergence of functional hierarchies. In particular, it was shown that a robot which had already been trained to perform primitive actions was able to learn new action sequences by only adapting the plans through solely changing the weights of the high-level network running at slow timescale. The fast low-level network which performed the primitive actions remained unchanged.

The concept discussed above is inspired by findings from brain research but implemented using classical artificial neural networks. Nevertheless, it highlights how carefully adjusted relative timing of neural dynamics can enable the emergence of higher cognition. Related work on the general concept of hierarchical task structuring based on different timescales was also considered in other contexts and for different applications. Under the assumption that temporal dependencies in the input data are structured hierarchically, the authors of [26] introduced *hierarchical recurrent networks*. Long-lasting context information is processed at a slower timescale than other data by incorporating synaptic delays in parts of the network. In addition to the neural implementation, a possible realization with hidden Markov Models was proposed. Different concurrent timescales also were considered for hierarchical reinforcement learning in [67].

4 Interfacing Neural Networks with Real-Time Applications

All aspects of the role of time in neural information processing considered so far were concerned with mathematical modeling and algorithmic properties. The positive results make spiking neural networks a promising tool for solving real-world problems. But simulating the neural dynamics for thousands or even millions of neurons requires huge computational power and is thus not easily feasible in real-time. Apart from the fact that it takes more time until the results are available, this does not pose any limitations in scenarios where the network input comes from an abstract dataset. However, real-time applications like neurorobotics [36] which integrate the neural simulation in a closed control loop are highly sensitive to the speed of the network dynamics. As demonstrated in Sect. 3, the execution speed of a neural circuit has huge influence on its computational properties. In this section, we therefore give a brief overview of selected techniques for executing spiking neural networks with a special focus on the supported timescales. While the first part covers simulators implemented in software, hardware-based approaches will be addressed in the second and the third part. Finally, we briefly explain factors that could contribute to neural processing at different timescales in one of the most widespread large-scale neural information processing systems—the mammalian brain.

4.1 Neural Network Simulators

The most direct approach of executing spiking neural networks is to implement a corresponding simulation on a standard general purpose computer. *NEST*, a widely used simulator for spiking neural networks, follows exactly this paradigm [51]. It supports parallel execution across many CPU cores and compute nodes via OpenMP and MPI. In [20], NEST is used to define a neural network with 10,000 Integrate-and-Fire neurons based on a connectivity structure proposed in [7]. Even on a modern quad-core CPU, simulating this network for a timespan of only 300 milliseconds requires simulation time in the order of seconds. From this it becomes clear that classical software simulations are no viable approach for real-time control tasks with large neural networks. On the other hand, if the control task is modeled in another simulation, e. g. a robot simulator, real-time execution is not required and the software-based approach allows for a fine-grained control over the relative timing between the dynamics of the neural network and the simulation of the control task.

NEST is not the only spiking neural network simulator available. Alternative implementations include *NEURON* [9] and *Brian* [21]. *CARLsim* [37] has support for GPU acceleration. Finally, *EDLUT* realizes an event-driven style of execution based on look-up tables to speed up the simulation [58].

4.2 Optimized Parallel Architectures

In NEST, the simulation of a neural network could be made real-time-capable by scaling up the number of CPUs. However, this increases the required physical space and the power consumption of the system, both of which is problematic especially for robotic applications. A possible explanation for the high requirements for CPU power imposed by spiking neural network simulators are the huge conceptual differences between the classical Von Neumann architecture found in standard CPUs and the massively parallel and distributed information processing in neural networks. The *SpiNNaker* architecture is specifically designed to support the execution of tasks of the latter type [31]. A single SpiNNaker chip comprises 18 ARM cores. Two of them are reserved for administrative tasks and fault tolerance while the remaining ones perform the actual neural computations [17]. The communication between the individual cores is handled by a dedicated network-on-chip. A special property of the SpiNNaker architecture is its integrated support for building large systems from a huge number of chips which are connected according to the toroidal topology illustrated in Fig. 10.

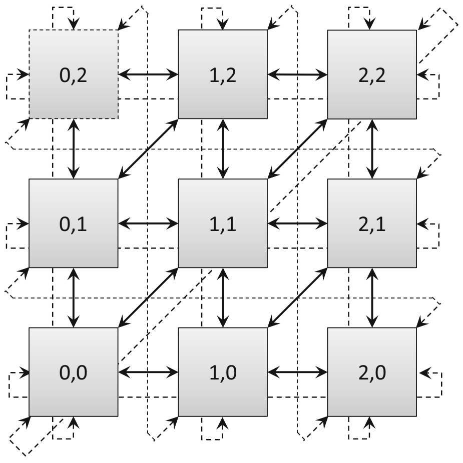

Fig. 10 Illustration of the physical topology of a SpiNNaker system [16]. Every node corresponds to a chip. *Dashed* connections wrap around the borders due to the toroidal topology. The execution of neural networks with arbitrary connectivity structure is achieved by superposition of a logical topology through adequate package routing. Reprinted from [70] with permission from Elsevier

The communication between different chips across the module is implemented in hardware, which means that every chip contains an own router.

The computation in a SpiNNaker system is driven by the asynchronous exchange of data packets based on the concept of *Address Event Representation* (AER). In the AER protocol, all information carried by a packet is encoded in its source and the time when it was generated. This clearly corresponds to neural communication via spikes. A single SpiNNaker chip is capable of simulating up to 20,000 neurons [31]. Note that although the topology of the SpiNNaker system is toroidal, the hardware is able to simulate neural networks of arbitrary connectivity. The topology of the simulated neural network is thus *virtualized* [17].

SpiNNaker is designed to execute spiking neural networks in real-time [17] and supports neural learning through online adaption of synaptic weights [32]. Together with its low energy consumption, this renders the system an ideal platform for the neural control of physical systems.

4.3 Neuromorphic Chips

Although the overall architecture of SpiNNaker is clearly inspired by neural principles, the neuron models are still evaluated on standard CPU cores. *Neuromorphic* chip designs go even one step further and also implement the neural dynamics in analogy to biological neurons. The principal feasibility of simulating neurons with electrical circuits was already mentioned at the beginning of this paper: Like many other neuron models, the Leaky Integrate-and-Fire neuron from Eq. 1 basically models an electrical circuit. Analog neuromorphic chips directly implement such circuits to simulate neural networks in continuous time.

The neuromorphic *HICANN* chip, which was designed as part of the BrainScaleS project, implements the exponential Integrate-and-Fire neuron model [60]. 384 of these chips are combined on a single wafer [61] and communicate via an event-based protocol. The neural dynamics run 10^3–10^5 times faster than real-time. While this is a remarkable advantage for theoretical studies, it remains unclear how such a fast timescale can be integrated in real-time control tasks. *Neurogrid* is another neuromorphic design which executes neural circuits in real-time [1]. But unlike the BrainScaleS system, it has no built-in support for synaptic plasticity. A recent review of current neuromorphic chip designs with a focus on supported learning techniques is available in [70].

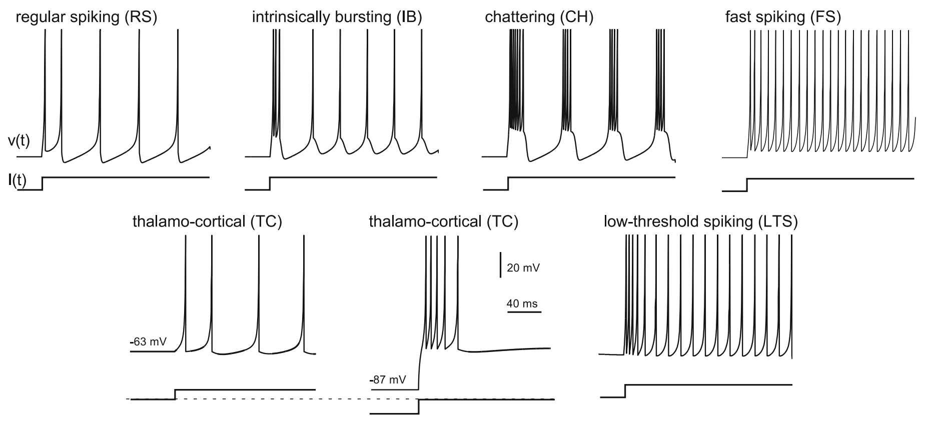

Fig. 11 Qualitative overview of different types of dynamics exhibited by cortical neurons Adapted and reprinted from [29]. Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com

4.4 Biological Neurons

Having investigated different types of specialized hardware and software, it is finally worth considering how biological neurons realize computation at different timescales. Since biological neurons are the actual subjects of study in the mathematical modeling and simulation of spiking neural networks, it is clear that networks composed of these neurons operate in real-time by definition. Simply put, the properties of biological neural dynamics are governed by the interaction of different types of ion channels [35]. This leads to various types of neurons which produce different outputs for a given input. Special synapses are able to change the behavior of the neuron channels and thus also the dynamics of the postsynaptic neuron [35]. A qualitative overview of different types of neural response is provided in Fig. 11. Although all these observations are far from proving the role of different temporal dynamics and their neural implementation in the human brain, they at least point to a possible biologically plausible implementation of advanced cognitive capabilities with spiking neural networks.

5 Conclusion and Outlook

The goal of this work was to review how temporal neural dynamics can enable meaningful computation in biologically inspired spiking neural networks. Starting by considering single neurons, we first studied how the detailed dynamical models employed in spiking neural networks allow for the encoding of information in the precise timing of spikes in Sect. 2. The second part of the section was dedicated to learning algorithms which enable spiking neurons to produce specific neural codes based on given sets of input spike trains. Examples like an algorithm for unsupervised visual feature learning clearly illustrated how powerful neurobiological implementations of complex algorithms can be enabled by spiking neural networks.

In Sect. 3, we analyzed the role of time at network level. One of the most important results in this section was the fact that recurrent neural networks as found in brains exhibit complex internal dynamics which can be used for computational purposes. Importantly, this is true for both artificial neural networks and spiking neural networks. Learning techniques based on the reservoir computing paradigm use these dynamics to project input into a complex high-dimensional space. Learning is achieved by training the weights of a set of dedicated output units. The Liquid State Machine was presented as a spiking neuron-based implementation of the reservoir concept. In the second part of the section, we showed how neural networks working concurrently on different timescales enable hierarchical learning and the emergence of higher-order cognitive functions. In MTRNNs, subnets executed with slow dynamics represent the current context for the networks at lower levels with faster dynamics.

How the temporal features discussed in the sections above can emerge during the execution of a neural network with a simulator or dedicated hardware was addressed in Sect. 4. The motivation for studying this topic was drawn from the field of neurorobotics, i. e. the research on robots controlled by biologically plausible models of nervous systems. It turned out that standard simulation software for spiking neural networks is not suited for closed-loop real-time control since the real-time simulation of larger networks requires powerful compute clusters. The SpiNNaker architecture solves this issue by enabling real-time execution of spiking neural networks through a highly efficient communication system. Neuromorphic chips which employ analog computing to simulate neural dynamics are a similar approach in this direction. However, the presented BrainScaleS system runs too fast for real-time applications. Finally, we gave a brief overview of different types of temporal dynamics exhibited by biological neurons in the brain.

In summary, our literature review on the role of time as an important aspect of information processing in neural networks indicates that temporal neural dynamics can perform meaningful computations. Interestingly, the importance of the factor time has also been recognized in classical machine learning where different concurrent timescales have been applied in reinforcement learning scenarios in order to separate different logical contexts [67]. Since spiking and recurrent neural networks are capable of producing a broad range of temporal dynamics, we therefore suggest that they are a very promising approach for realizing *computation by time*. The huge variety of neuron models, coding schemes and learning techniques is clear evidence of a well-established theoretical framework which supports both rigorous mathematical analysis and close correspondence to biology at the same time. Especially the latter aspect opens up new possibilities for an interdisciplinary exchange between neural information processing and neuroscience. But our review also shows that spiking neural networks are an interesting tool from an engineering point of view. Like analog neural networks of the second generation, they implicitly realize massively distributed and parallel processing. But in addition, the interneural communication via spikes enables efficient neuromorphic chips which simulate temporal neural dynamics in real-time or even faster. Considering the increasingly growing interest in these novel hardware architectures, spiking neural networks have a huge potential of becoming the standard programming language of future computing.

Acknowledgments The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007–2013) under grant agreement no 604102 (HBP).

References

1. Benjamin BV, Peiran Gao, McQuinn E, Choudhary S, Chandrasekaran AR, Bussat JM, Alvarez-Icaza R, Arthur JV, Merolla PA, Boahen K (2014) Neurogrid: a mixed-analog-digital multichip system for large-scale neural simulations. Proc IEEE 102(5):699–716
2. Bi G, Poo M (2001) Synaptic modification by correlated activity: Hebb's postulate revisited. Annu Rev Neurosci 24(1):139–166

3. Bohte SM (2004) The evidence for neural information processing with precise spike-times: a survey: natural computing. Nat Comput 3(2):195–206
4. Bohte SM, Kok JN, La Poutré H (2002) Error-backpropagation in temporally encoded networks of spiking neurons. Neurocomputing 48(1–4):17–37
5. Bohte SM, La Poutre H, Kok JN (2002) Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks. Neural Netw IEEE Trans 13(2):426–435
6. Brea J, Senn W, Pfister JP (2013) Matching recall and storage in sequence learning with spiking neural networks. J Neurosci 33(23):9565–9575
7. Brunel N (2000) Dynamics of sparsely connected networks of excitatory and inhibitory spiking neurons. J Comput Neurosci 8(3):183–208
8. Carnell A, Richardson D (2005) Linear algebra for time series of spikes. In: Proceedings of ESANN, pp 363–368
9. Carnevale N, Hines M (2015) NEURON for empirically-based simulations of neurons and networks of neurons: project homepage. http://www.neuron.yale.edu/neuron/
10. Cyr A, Boukadoum M, Thériault F (2014) Operant conditioning: a minimal components requirement in artificial spiking neurons designed for bio-inspired Robot's controller. Front Neurorobot 8(21)
11. Diamond MC (2001) Response of the brain to enrichment. Anais da Academia Brasileira de Ciências 73:211–220
12. Farries MA, Fairhall AL (2007) Reinforcement learning with modulated spike timing-dependent synaptic plasticity. J Neurophys 98(6):3648–3665
13. Floreano D, Mattiussi C (2001) Evolution of spiking neural controllers for autonomous vision-based robots. In: Goos G, Hartmanis J, van Leeuwen J, Gomi T (eds) Evolutionary robotics, vol 2217., From intelligent robotics to artificial life, lecture notes in computer scienceSpringer, Berlin, pp 38–61
14. Florian RV (2005) A reinforcement learning algorithm for spiking neural networks. In: Proceedings of the seventh international symposium on symbolic and numeric algorithms for scientific computing, SYNASC '05. IEEE Computer Society, Washington, DC, USA
15. Frémaux N, Sprekeler H, Gerstner W (2013) Reinforcement learning using a continuous time actor-critic framework with spiking neurons. PLoS Comput Biol 9(4):e1003,024
16. Furber S, Brown A (2009) Biologically-inspired massively-parallel architectures - computing beyond a million processors. In: Application of concurrency to system design, 2009 (ACSD '09). Ninth international conference on, pp 3–12
17. Furber SB, Lester DR, Plana LA, Garside JD, Painkras E, Temple S, Brown AD (2013) Overview of the spinnaker system architecture. Comput IEEE Trans 62(12):2454–2467
18. Gerstner W, Kempter R, van Hemmen JL, Wagner H (1996) A neuronal learning rule for sub-millisecond temporal coding. Nature 383(6595):76–78
19. Gerstner W, Kistler WM (2002) Spiking neuron models: single neurons, populations, plasticity. Cambridge University Press, Cambridge
20. Gewaltig MO, Morrison A, Plesser HE (2012) NEST by example: an introduction to the neural simulation tool NEST. In: Le Novère N (ed) Computational systems neurobiology. Springer, The Netherlands, pp 533–558
21. Goodman Dan FM, Brette R (2009) The brian simulator. Front Neurosci 3(2):192
22. Grüning A, Bohte SM (2014) Spiking neural networks: principles and challenges. In: ESANN 2014. 22nd European symposium on artificial neural networks, computational intelligence and machine learning. Bruges, April 23–25, 2014. i6doc.com, Louvain-La-Neuve
23. Gütig R (2014) To spike, or when to spike? Theor Comput Neurosci 25:134–139
24. Gütig R, Sompolinsky H (2006) The tempotron: a neuron that learns spike timing-based decisions. Nat Neurosci 9(3):420–428
25. Hebb DO (1949) The organization of behavior: a neuropsychological theory. Wiley, New York
26. Hihi SE, Bengio Y (1996) Hierarchical recurrent neural networks for long-term dependencies. In: Touretzky DS, Mozer MC, Hasselmo ME (eds) Advances in neural information processing systems 8. MIT Press, Cambridge, pp 493–499
27. Hodgkin AL, Huxley AF (1952) A quantitative description of membrane current and its application to conduction and excitation in nerve. J Physiol 117(4):500–544
28. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4(2):251–257
29. Izhikevich EM (2003) Simple model of spiking neurons. Neural Netw IEEE Trans 14(6):1569–1572
30. Izhikevich EM (2007) Solving the distal reward problem through linkage of STDP and dopamine signaling. Cereb Cortex 17(10):2443–2452
31. Jin X, Lujan M, Plana LA, Davies S, Temple S, Furber SB (2010) Modeling spiking neural networks on spinnaker. Comput Sci Eng 12(5):91–97

32. Jin X, Rast A, Galluppi F, Khan M, Furber S (2009) Implementing learning on the spinnaker universal neural chip multiprocessor. In: Leung C, Lee M, Chan J (eds) Neural information processing, vol 5863., Lecture notes in computer scienceSpringer, Berlin, pp 425–432
33. Kandel ER, Siegelbaum SA (2013) Cellular mechanisms of implicit memory storage and the biological basis of individuality. In: Kandel ER, Schwartz JH, Jessel TM, Siegelbaum SA, Hudspeth AJ (eds) Principles of neural science. McGraw-Hill, New York, pp 1461–1486
34. Kandel ER, Siegelbaum SA (2013) Synaptic integration in the central nervous system. In: Kandel ER, Schwartz JH, Jessel TM, Siegelbaum SA, Hudspeth AJ (eds) Principles of neural science. McGraw-Hill, New York, pp 210–235
35. Koester J, Siegelbaum SA (2013) Propagated signaling: the action potential. In: Kandel ER, Schwartz JH, Jessel TM, Siegelbaum SA, Hudspeth AJ (eds) Principles of neural science. McGraw-Hill, New York, pp 148–176
36. Krichmar J (2008) Neurorobotics. Scholarpedia 3(3):1365
37. Krichmar J (2015) CARLsim: GPU-accelerated spiking neural network simulator: project homepage. http://www.socsci.uci.edu/~jkrichma/CARLsim/index.html
38. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
39. Legenstein R, Naeger C, Maass W (2005) What can a neuron learn with spike-timing-dependent plasticity? Neural Comput 17(11):2337–2382
40. Legenstein R, Pecevski D, Maass W (2008) A learning theory for reward-modulated spike-timing-dependent plasticity with application to biofeedback. PLoS Comput Biol 4(10):e1000,180
41. Lukoševičius M, Jaeger H (2009) Reservoir computing approaches to recurrent neural network training. Comput Sci Rev 3(3):127–149
42. Maass W (1997) Networks of spiking neurons: the third generation of neural network models. Neural Netw 10(9):1659–1671
43. Maass W, Jaeger H, Steil J, Dominey PF, Schrauwen B (2015) Web portal for reservoir computing. http://organic.elis.ugent.be/
44. Maass W, Natschläger T, Markram H (2002) Real-time computing without stable states: a new framework for neural computation based on perturbations. Neural Comput 14(11):2531–2560
45. Markram H, Lübke J, Frotscher M, Sakmann B (1997) Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. Science 275(5297):213–215
46. Masquelier T, Thorpe SJ (2007) Unsupervised learning of visual features through spike timing dependent plasticity. PLoS Comput Biol 3(2):e31
47. McCulloch WS, Pitts W (1943) A logical calculus of the ideas immanent in nervous activity. Bull Math Biophys 5(4):115–133
48. Memmesheimer RM, Rubin R, Ölveczky BP, Sompolinsky H (2014) Learning precisely timed spikes. Neuron 82(4):925–938
49. Morrison A, Diesmann M, Gerstner W (2008) Phenomenological models of synaptic plasticity based on spike timing. Biol Cybern 98(6):459–478
50. Natschläger T, Ruf B (1998) Spatial and temporal pattern analysis via spiking neurons. Network 9(3):319–332
51. NEST Initiative (2015) NEST: project homepage. http://www.nest-initiative.org/
52. Nichols C, McDaid LJ, Siddique NH (2010) Case study on a self-organizing spiking neural network for robot navigation. Int J Neural Syst 20(06):501–508
53. Norton D, Ventura D (2006) Preparing more effective liquid state machines using hebbian learning. In: Neural networks, 2006. IJCNN '06. International joint conference on, pp 4243–4248
54. Pfister JP (2006) Triplets of spikes in a model of spike timing-dependent plasticity. J Neurosci 26(38):9673–9682
55. Ponulak F, Hopfield JJ (2013) Rapid, parallel path planning by propagating wavefronts of spiking neural activity. Front Comput Neurosci 7:98
56. Ponulak F, Kasiński A (2009) Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. Neural Comput 22(2):467–510
57. Ponulak F, Kasinski A (2011) Introduction to spiking neural networks: information processing, learning and applications. Acta Neurobiol Exp 71(4):409–433
58. Ros E, Carrillo R, Ortigosa EM, Barbour B, Agís R (2006) Event-driven simulation scheme for spiking neural networks using lookup tables to characterize neuronal dynamics. Neural Comput 18(12):2959–2993
59. Rosenblatt F (1958) The perceptron: a probabilistic model for information storage and organization in the brain. Psychol Rev 65(6):386–408

60. Schemmel J, Brüderle D, Grübl A, Hock M, Meier K, Millner S (2010) A wafer-scale neuromorphic hardware system for large-scale neural modeling. In: Circuits and systems (ISCAS), proceedings of 2010 IEEE international symposium on, pp 1947–1950
61. Schemmel J, Grubl A, Hartmann S, Kononov A, Mayr C, Meier K, Millner S, Partzsch J, Schiefer S, Scholze S, Schuffny R, Schwartz M (2012) Live demonstration: a scaled-down version of the BrainScaleS wafer-scale neuromorphic system. In: Circuits and systems (ISCAS), 2012 IEEE international symposium on, p 702
62. Schliebs S, Kasabov N (2014) Computational modeling with spiking neural networks. In: Kasabov N (ed) Springer handbook of bio-/neuroinformatics. Springer, Berlin, pp 625–646
63. Schmidhuber J (2015) Deep learning in neural networks: an overview. Neural Netw 61:85–117
64. Senn W, Pfister JP (2014) Reinforcement learning in cortical networks. In: Jaeger D, Jung R (eds) Encyclopedia of computational neuroscience. Springer, New York, pp 1–9
65. Sjöström PJ, Turrigiano GG, Nelson SB (2001) Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. Neuron 32(6):1149–1164
66. Sporea I, Grüning A (2012) Supervised learning in multilayer spiking neural networks. Neural Comput 25(2):473–509
67. Sutton RS, Precup D, Singh S (1999) Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artif Intell 112(1–2):181–211
68. Tani J (2014) Self-organization and compositionality in cognitive brains: a neurorobotics study. Proc IEEE 102(4):586–605
69. The Human Brain Project (2015) Project homepage. https://www.humanbrainproject.eu
70. Walter F, Röhrbein F, Knoll A (2015) Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks. Neural Netw
71. Werbos PJ (1990) Backpropagation through time: what it does and how to do it. Proc IEEE 78(10):1550–1560
72. Wysoski S, Benuskova L, Kasabov N (2006) On-line learning with structural adaptation in a network of spiking neurons for visual pattern recognition. In: Kollias S, Stafylopatis A, Duch W, Oja E (eds) Artificial neural networks—ICANN 2006, vol 4131., Lecture notes in computer scienceSpringer, Berlin, pp 61–70
73. Xu Y, Zeng X, Zhong S (2013) A new supervised learning algorithm for spiking neurons. Neural Comput 25(6):1472–1511