Sander M. Bohté
Machine Learning Group
Centrum Wiskunde & Informatica, Amsterdam
s.m.bohte@cwi.nl

Davide Zambrano
Machine Learning Group
Centrum Wiskunde & Informatica, Amsterdam
d.zambrano@cwi.nl

Adapting spiking neural networks

Understanding how neurons are able to efficiently encode information is a topic with applications ranging from more efficient neural network chips, to robot control, and also to future prosthetics that directly communicate with neurons. To understand how the brain is able to operate efficiently and asynchronously, Sander Bohté and Davide Zambrano describe models of biological neurons and examine how the spiking nature of neuronal communication relates this question.

A central goal of Artificial Intelligence (AI) is to develop algorithms that match the human ability to perceive, plan and act, be it vision, hearing, smelling, or touching. The human brain shows a remarkable ability to deal with the difficult task of making sense of the external world. This task is difficult in many ways: perception itself is by definition noisy and ambiguous, and muscles are notoriously hard to control as factors like fatigue, growth and atrophication alter the effect of motor commands given by the brain.

Loosely modeled after the neuronal networks of the brain, so-called deep neural networks have revolutionised AI in recent years, delivering breakthrough performance on such diverse tasks like image recognition, speech recognition, and superhuman performance playing Go and Chess: we have truly entered the age where computers are better at certain 'intelligent' tasks than humans. Here, we will give some intuition into the question of what these deep neural networks are, how they relate to the brain, and what more we can learn from the brain.

The human brain consists of an intricate web of billions of interconnected cells called 'neurons', where each neuron typically makes connections to up to $10{,}000$ other neurons. The study of neural networks in computer science aims to understand how such a large collection of connected elements can produce useful computations, such as vision and speech recognition, and also motor control, like using perception to catch a ball.

Artificial neural networks

Artificial neural networks (ANNs) are abstractions of the computation believed to be carried out by real neurons. A 'real' neuron receives pulses from many other neurons. These pulses are processed in a manner that may result in the generation of puls-

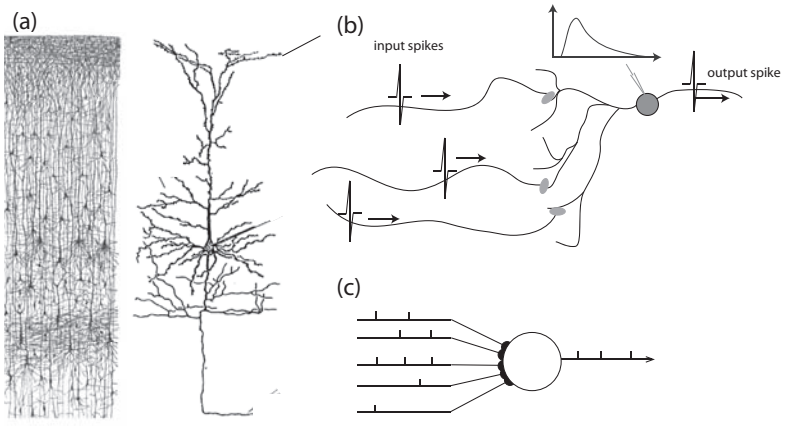

Figure 1 (a) Staining of just some of the neurons in a piece of cortex. A singly (pyramidal) neuron is shown as well. Notice the many synapses where the neuron receives input from other neurons. (b) Neurons communicate with each other using spikes, which each influence the internal state (typically the membrane potential) of the target neuron. (c) Abstracted representation of spike-based communication, where synapses between neurons are taken as 'weights'.

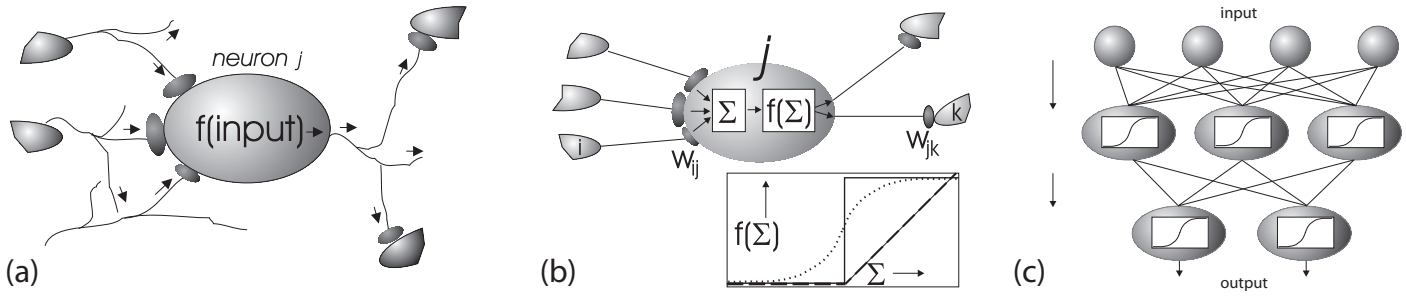

Figure 2 (a) A neuron computes output from inputs. (b) Artificial neuron modeling input–output computation. Inset: transfer functions $f(\Sigma)$. Plotted are examples of a binary (solid line), sigmoidal (dotted line) and ReLU (dashed line) transfer function. (c) Stringing neurons together to obtain a multilayer perceptron network.

es in the receiving neuron, which are then transmitted to other neurons (Figure 1b, c). The neuron thus 'computes' by transforming input pulses into output pulses.

ANNs try to capture the essence of this computation: as depicted in Figure 2, the rate at which a neuron fires pulses is abstracted to a scalar 'activity value', or *output*, assigned to the neuron. Directional connections determine which neurons are input to other neurons. Each connection has a weight, and the output of a particular neuron is a function of the sum of the weighted outputs of the neurons it receives input from. The applied function is called the *transfer function*, $F(\Sigma)$. Binary 'thresholding' neurons have as output a '1' or a '0', depending on whether or not the summed input exceeds some threshold. Sigmoidal neurons apply a sigmoidal transfer function, and have a real-valued output, and so-called 'rectified linear' neurons, or 'ReLU') neurons apply a rectified linear function (inset Figure 2b, solid respectively dotted and dashed line). Abstracted in the sigmoidal transformation function is the idea that real neurons communicate via firing rates: the rate at which a neuron generates action potentials (spikes). When receiving an increasing number of (positively weighted) spikes, a neuron is naturally more likely to emit an increasing number of spikes itself.

Neural networks are sets of connected artificial neurons. Remarkably, networks of such simple, connected computational elements can implement a range of mathematical functions relating input states to output states, where its computational power is derived from the connectivity pattern and clever choices for the values of the connection weights.

Learning rules for neural networks prescribe how to adapt the weights to improve performance given some task. An example of a neural network is the *multi-layer perceptron* (MLP, Figure 2c). Learning rules like *error backpropagation* [23] compute the gradient of each weight with respect to a pre-defined loss function that captures the cost of deviations from desired behavior. The weights in the network are adjusted along this gradient to minimize the loss, which enables the neural networks to learn and perform many tasks associated with intelligent behavior, like learning, memory, pattern recognition, and classification [1, 22].

Different types of neural networks have been developed over the last two decades. So-called convolutional neural networks take their inspiration from filters used in computer vision: a filter is specified as a matrix of weights which is convolved with an image to, for instance, detect edges in figures. The benefit of this approach is that a single small filter, like a 3×3, 4×4 or 5×5 matrix of weights, can be applied to the entire image: detecting edges thus needs only few parameters (the weights). Applying the same filter to the entire image also makes sense, as edges can be anywhere in the image.

Convolutional neural networks (CNNs) exploit this principle, having small filters that are convolved with the image, however, rather than handcrafted, the filters are learned. The convolutional neural networks, as illustrated in Figure 3a, moreover contains many filters in a layer, and a layer of filters connects to a next layer of filters, to learn filters-of-filters. In deep CNNs, many of these layers are used successively. The resultant filters-of-filters become sensitive to progressively complex features in the image, like from edges to lines, to features like noses, mouths and eyes, to faces. It was this type of neural network, developed already in the late 1980's by Yann LeCun, that achieved the breakthrough in

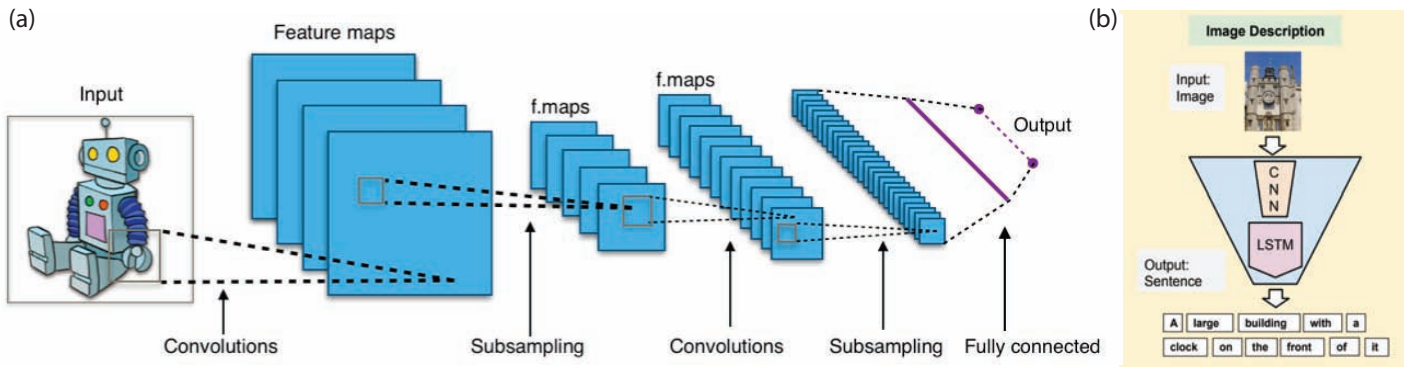

Figure 3 (a) Deep convolutional neural networks. Feature layers extract features from previous maps, while subsampling layer compress features maps to create more position-invariant features. (b) A neural network that writes captions to describe an image: an image is parsed by a CNN and the output of the CNN is parsed by an LSTM to select a sequence of words (taken from [8]).

object classification by Alex Krizhevsky and Geoffrey Hinton in 2012.

Convolutional neural networks are complemented with neural network structures capable of learning to maintain information-memory structures. Many tasks have a sequential nature where information has to be integrated and maintained to make the right inferences or choices: from reading a text to driving from home to work. While recurrent network structures can in principle (learn to) maintain relevant information, it was found in the late 1980's that such structures are notoriously hard to train. In 1998 then, Sepp Hochreiter and Jürgen Schmidhuber developed a memory structure with more tractable learning properties, so-called Long Short-Term Memory, or LSTM. Such networks, and variants thereof, are the workhorse of modern neural networks for sequential tasks. Figure 3b shows an impressive example of how convolutional neural networks and LSTMs are combined to create remarkably accurate captions for images.

Much of the magic of modern neural networks is enabled by the development of very powerful hardware for computing the matrix multiplications that underly the computations in neural networks. The star here are GPUs: initially developed for high-end gaming, it turned out that the massively parallel hardware was an excellent fit for computing neural networks. Only the last few years has dedicated AI hardware started to emerge, ranging from ultra-high performance tensor processing units (TPUs) developed by Google, to dedicated 'AI' blocks in cell-phone chips like Huawei's Kirin 970. Exploiting this powerful hardware has also become feasible by the development of high-level neural network frameworks, like Tensorflow and PyTorch, that make it easy to implement and train neural networks in an almost hardware agnostic manner.

Back to the brain

While deep neural networks are achieving huge successes, in many aspects they still pale in comparison to their biological source of inspiration, the brain. For example, the brain needs vastly fewer examples to learn tasks and is massively more energy efficient, while it's ability to control hundreds of flexible and variable muscles for motion remains unsurpassed. In particular, artificial neural networks operate according to a synchronised paradigm of computation, where in a single pass all neurons exchange their activation values and update their internal state. In contrast, the brain operates in an asynchronous fashion: real neurons only exchange information when they receive sufficient inputs, and they do so only rarely. Understanding how neurons are able to efficiently encode information is a topic with applications ranging from more efficient neural network chips, to robot control, and also to future prosthetics that directly communicate with neurons — neuroprosthetics. Thus, to understand how the brain is able to operate efficiently and asynchronously, we return to our models of biological neurons, and in particular examine how the spiking nature of neuronal communication relates this question.

The question of how neurons encode information in the spikes they emit is a hotly debated one in neuroscience. At the heart of the argument is the issue to what degree neurons respond in a stochastic manner to received inputs: on the one hand, many experimental findings show that neuronal firing is highly unreliable, and can be reasonably described as a rate-driven Poisson process. On the other hand, we know that individual spiking neurons can be highly reliable, emitting reproducible spikes at a very high time resolution [15]. Part of the reason for this finding seems to be that spike-time reliability is related to the temporal properties of the received signal; an example of this phenomenon is shown in Figure 4.

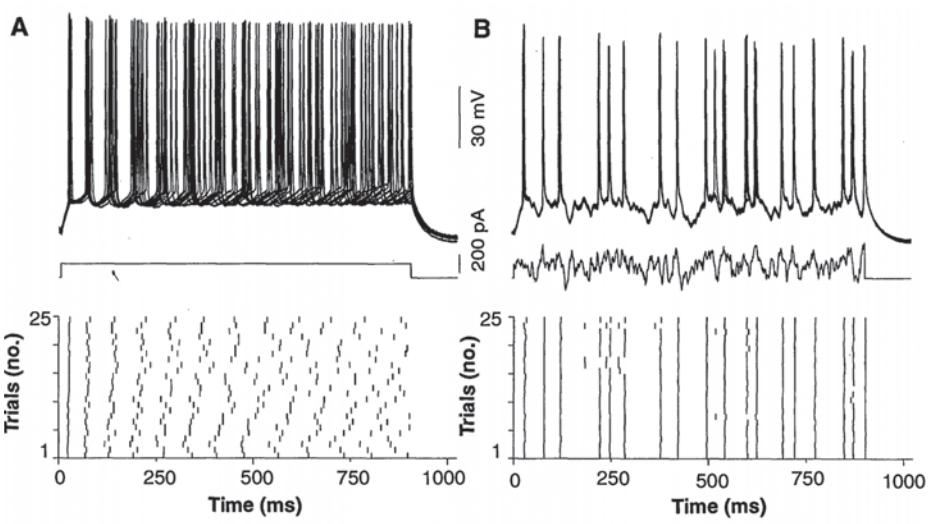

Figure 4 Reliability of firing of real spiking neurons when inject by either a constant (a) or fluctuating current (b). Top: overlapped voltage traces from 25 trials obtained from a single neuron repeatedly injected with a either a fixed or variable current profile (middle). Bottom: raster-plot of individual spike-times for each of the 25 trials. Note the dispersion in spike-times for a fixed current injection and the reliability of spike-timing for the fluctuating current profile. Graph taken from [15].

Spike times carrying significant information is attractive, as in theory it increases the amount of information carried by each spike. Information theoretic measurements on the entropy of in-vivo (real-world) neurons have also shown significant information in the precise spike timing [16].

Spiking neuron models

The prototypical model of a spiking neuron is the Hodgkin–Huxley model. Experimenting on the squid's giant axon, Hodgkin and Huxley [13] found that three ionic currents determined most of the axon's behavior: the sodium and potassium currents, and a leak current. Ion channels in the neuron's cell membrane control the flow of ions in a voltage-dependent manner, where the interior of the cell acts as a capacitor. This leads them to propose a relatively simple electrical circuit as a model of the neurons response to a current entering the cell (Figure 5a), where the current partly charges the capacitor and partly leaks through the ion-channels.

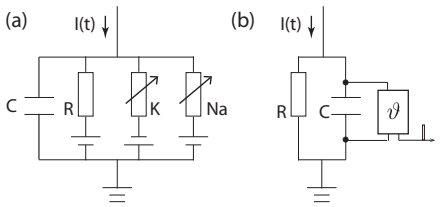

Figure 5 (a) Electrical circuit for the Hodgkin–Huxley neuron model. (b) Electrical circuit for the leaky integrate-and-fire neuron model.

In this model, mathematically, we split an applied current $I(t)$ into a current charging the capacitor I_{cap} and components I_k leaking trough the ion channels:

$$I(t) = I_{\text{cap}} + \sum_k I_k(t)$$

For a voltage u across the capacitor, we can substitute $I_{\text{cap}} = C du/dt$, rewriting:

$$C\frac{du}{dt} = -\sum_k I_k(t) + I(t).$$

For the leakage currents $I_k(t)$, Hodgkin and Huxley formulated differential equations for the three main components, the voltage dependent Na^+, K^+ channels and a generic leakage channel:

$$\sum_k I_k = g_{\text{Na}} m^3 h(u - E_{\text{Na}}) + g_{\text{K}} n^4 (u - E_{\text{K}}) + g_{\text{L}}(u - E_{\text{L}}),$$

where $E_{\text{Na}}, E_{\text{K}}$ and E_{L} are the respective reversal potentials; $g_{\text{Na}}, g_{\text{K}}$ and g_{L} are the respective maximum channel conductances. The variables m, h and n are the gating variables that control the Na^+ and K^+ channels, and that evolve as:

$$\frac{dm}{dt} = \alpha_m(u)(1-m) - \beta_m(u)m, \quad (1)$$

$$\frac{dn}{dt} = \alpha_n(u)(1-n) - \beta_n(u)m, \quad (2)$$

$$\frac{dh}{dt} = \alpha_h(u)(1-h) - \beta_h(u)m. \quad (3)$$

Hodgkin and Huxley then fitted the functions $\alpha(u)$ and $\beta(u)$ to the experimental data.

The Hodgkin and Huxley equations provide an accurate description of many dynamical responses of the squid axon, and by choosing different values for the various variables, many types of observed neural responses can be fitted. For example, a current injection may result in a moderate disturbance of the membrane potential, the generation of a single spike, or even trigger a burst of spikes persisting for much longer than the current injection.

Still, while the equations can be studied with the tools of mathematics, the behavior of such high-dimensional and non-linear equations is both hard to analyze and hard to visualize.

Leaky integrate-and-fire (LIF)

To study topics like memory and neural coding, simple phenomenological models are often preferred. Such models capture both the dynamics of the membrane potential as a function of impinging spikes and current injections while also prescribing the conditions for a neuron to generate an action potential.

Broadly, the transmission of a single spike from one neuron to another is mediated by *synapses* at the point where the two neurons interact. An input, or *presynaptic* spike arrives at the synapse, which in turn releases neurotransmitter which then influences the state, or *membrane potential* of the target, or *postsynaptic* neuron. When the value of this state crosses some threshold ϑ, the target neuron generates a spike, and the state is reset by a *refractory response*. The size of the impact of a presynaptic spike is determined by the type and efficacy (weight) of the synapse (this is illustrated in Figure 6).

The electrical circuit describing a Leaky-Integrate-and-Fire (LIF) neuron is a simple version of the Hodgkin-Huxley neuron: as illustrated in Figure 5b, it consists of a current $I(t)$ driving a capacitor C in parallel with a resistor R. The current again splits in a component that charges the capacitor and one that passes through the resistor: $I(t) = I_{\text{cap}} + I_R$. Substituting $I_R = u/R$ (Ohm's Law) and $C = q/u$, where u is the voltage and q is the charge, we get:

$$I(t) = \frac{u(t)}{R} + C\frac{du}{dt}.$$

With $\tau_m = RC$, we can rewrite this as:

$$\tau_m \frac{du}{dt} = -u(t) + RI(t),$$

where we identify $u(t)$ as the membrane potential of the neuron, and τ_m as the membrane time constant.

The neuron emits a spike when the membrane potential reaches a threshold ϑ from below. When this happens, the membrane potential is reset to a new value $u_r < \vartheta$. This combination of leaky integration of incoming current and a reset at the time of spiking characterises LIF neuron models.

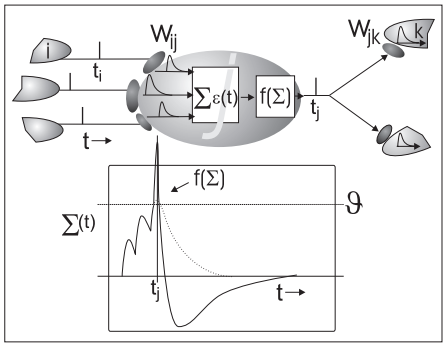

Figure 6 Impact of spikes on the potential of a target neuron.

Many different variations of LIF neurons can be created, including versions that include refractory effects at the time of spiking, where the threshold is stochastic and where the threshold is dynamic. Such more elaborate models of LIF neurons have been shown to predict neural behavior to a remarkable degree when compared to experimental data; a wonderful and accessible treatise on this topic has been written by Gerstner and Kistler [10].

Synaptic currents

So far, spiking neurons have been described in terms of their response to currents $I(t)$ injected into the neuron. In reality, these currents are caused by neurotransmitters arriving through synapses triggered by spikes from the sending neuron (presynaptic spikes).

Synapses are complicated beasts: broadly, neurotransmitters are released in quantiles, contained in small vesicles that release their content by fusing with the synapse' membrane. This process seems to be both stochastic and history-dependent: the amount of neurotransmitter released at a synapse in response to the arrival of a spike can vary dramatically.

Ignoring this complexity, we can model the current that a presynaptic spike at time t_j contributes to a postsynaptic neuron i as a post-synaptic current (PSC) with time course $\alpha(t - t_j)$ weighted by a particular weight, or 'synaptic efficacy w_{ij}. A neuron i thus receives as input currents:

$$I(t) = \sum_j w_{ij} \sum_{t_j} \alpha(t - t_j).$$

The most simple model for the postsynaptic current $\alpha(s)$ is a Dirac $\delta-$pulse, $\alpha(s) = q\delta(s)$, for a total current contribution q. More realistic models let the current α have a finite duration, for example an exponential decay with time constant τ_s:

$$\alpha(s) = \frac{q}{\tau_s}\exp\left(-\frac{s}{\tau_s}\right)\Theta(s),$$

where $\Theta(s)$ denotes the Heaviside step function.

Spike Response Model (SRM)

Wulfram Gerstner [11] developed the Spike

Response Model (SRM) as a non-linear integrate-and-fire model that expresses the membrane potential at time t as an integral over the past, as opposed to a formulation in terms of dynamical systems. Specifically, the membrane potential is modelled as a sum of (weighted) impinging post-synaptic potentials, $\epsilon(t)$, and refractory responses $\eta(t)$:

$$u_i(t) = \sum_{t_i} \eta(t - t_i) + \sum_j w_{ij} \sum_{t_j} \epsilon(t - t_i, t - t_j),$$

where ϵ and η are *response kernels*. The threshold, that determines when a neuron fires, can be dynamical: $\vartheta \longrightarrow \vartheta(t - t_i)$. Many phenomenological models include such dynamical thresholds to explain the spiking behavior of many different neurons. The main benefit of the SRM formulation is that in many ways, SRM formulations of LIF-neurons are much more easily interpretable. We will rely on this in our formulation of a spike-time-based neural code later on.

Neural networks

As an artificial neuron models the relationship between the inputs and the output of a neuron, artificial spiking neurons describe the input in terms of single spikes, and how such input leads to the generation of output spikes. For this, we need to relate spikes to information and computation.

As noted, the exact nature of neural coding by biological neurons is still unresolved in neuroscience and subject to much debate. A recent line of work suggests that spiking neurons may implement adaptive online analog-to-digital and digital-to-analog (AD/DA) conversion [2, 3, 4, 6, 26]: the key observation is that when a neuron spikes, the refractory reset removes a part of the internally computed analog voltage signal, which the spike, through the synapse, delivers to the next neuron. Young [26] recently demonstrated a direct correspondence between simple leaky integrate-and-fire (LIF) models and the AD/DA encoding/decoding scheme in electrical engineering called *asynchronous pulse sigma-delta modulation* (APSDM). The APSDM scheme however presumes a fixed dynamic range for the encoded analog values, as signals are 'chopped' into fixed-size pieces, and requires that a neuron fires at a very high firing rate to obtain a good signal approximation.

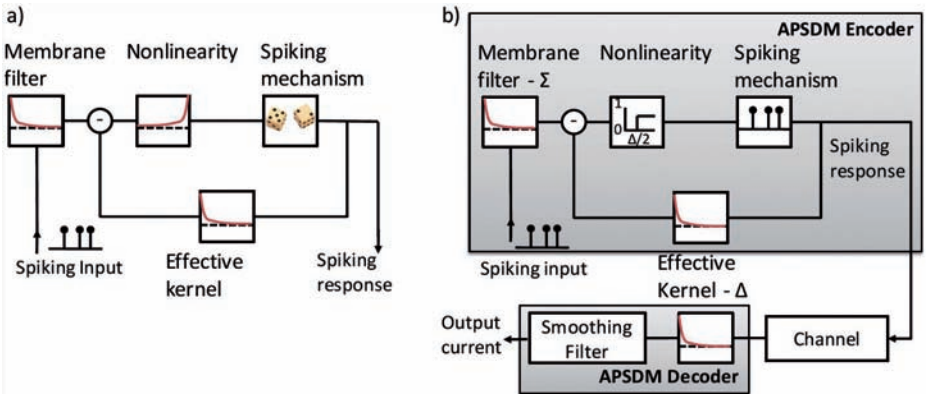

Figure 7 (a) Generalized leaky integrate-and-fire and (b) the asynchronous pulse sigma-delta modulation (APSDM) [26]. The APSDM scheme consists of an encoder (analog-to-digital), a channel, and a decoder (digital-to-analog), where signals are encoding using uni-polar pulses (spikes). A signal is first passed through a filter Σ and added to the internal state. Then a non-linearity is applied in the form of a thresholding function with threshold $\Delta/2$. When the threshold is exceeded, a pulse is sent to the decoder through the channel, while a response kernel Δ is subtracted from the internal state of the neuron. At the decoder, each pulse is decoded with a fixed response kernel and then smoothed. Note the close similarities between the APSDM scheme and the LIF neuron on the left.

Adaptive spike coding

We can combine the APSDM scheme with a spiking neuron model that dynamically adapts to the (varying) dynamic range of the computed internal activation value. Inspired by [5] and [3], we use a multiplicative model of adaptation to obtain a spiking neuron that is capable of encoding and decoding a wide dynamic range of activation values with a limited firing rate. We show that we can thus compose computationally efficient adaptive spiking neural networks through drop-in replacement of analog neurons in artificial neural networks (ANNs), which achieve identical performance to these ANNs without additional modifications.

A spiking neural network is defined by the relationship between spikes and the quantity that is computed in the neuron as the result of impinging spikes. With *adaptive spike-time coding*, weighted input spikes contribute linearly to the membrane potential, and when this sum of inputs reaches (positive) threshold, a spike is generated while a refractory reset is subtracted from the membrane potential. Since both refractory reset and the contribution of impinging spikes are temporally extended, intuitively, the (smoothed) sum of refractory resets corresponds to the signal conveyed to the next neuron; this process is illustrated in Figure 8a.

Adaptive spiking neuron using multiplicative adaptive spike-time coding

To create artificial spiking neural networks based on adaptive spike-time coding, we address the limited dynamic range of standard LIF or corresponding Spike Response Model (SRM) neurons. We note that it is the fixed size refractory resets that limit the dy-

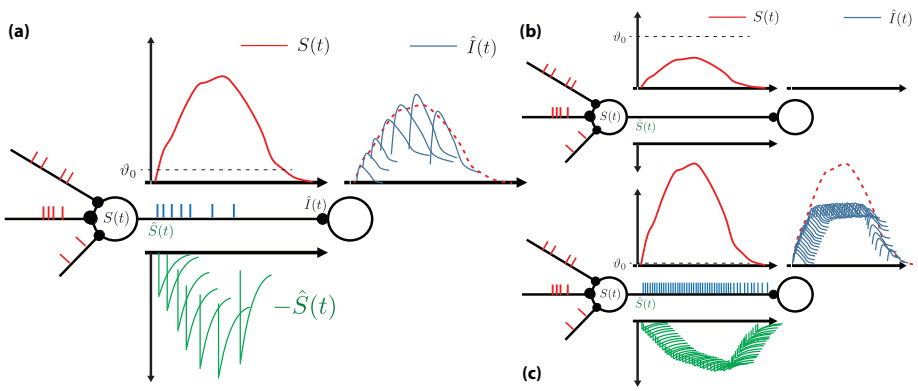

Figure 8 (a) Illustration of signal encoding with the ASN. $\hat{I}$ denotes the smoothed sum of (weighted) postsynaptic currents in the post-synaptic target neuron, proportionally approximating the encoded presynaptic signal $S(t)$. (b,c) Limited dynamic range: approximations fail when the signal $S(t)$ is too small relative to the neurons' threshold ϑ_0 (no spikes), or, (c) too large: then, due to absolute refractoriness and corresponding maximum firing rate, the 'high' parts of the signal $S(t)$ cannot be encoded.

namic range of the internal activation that a neuron can encode [3, 6]. Effectively, activation values that are either too small or too large relative to the threshold cannot be encoded. We use the solution proposed in [3] based on fast adaptation: by dynamically adjusting the threshold, the size of the refractory responses can be controlled and the dynamic range can be increased, drastically even when a multiplicative form of threshold adjustment is used. Such multiplicative adaptation effectively allows a neuron to assign a fixed 'budget' of spikes to a given dynamic range, also when that range changes drastically. Such a model of adaptation also explains various adaptive behaviour in real biological neurons [3, 5, 9].

We implement adaptive spike-time coding using multiplicative adaptation in an SRM [10]. A spiking neuron computes a smoothed internal activation value $S(t)$ on the input current:

$$S(t) = (\phi * I)(t),$$

where $\phi(t)$ is the (exponential) smoothing filter with time constant τ_{smooth} and $I(t)$ is the input current that the neuron receives. This current $I(t)$ can be injected directly into the spiking neuron (for inputs), or be the result of impinging (weighted) spikes causing post-synaptic currents (PSCs) (specified below). The spiking mechanism approximates the ReLU activation of $S(t)$ with $\hat{S}(t)$ using a sum of spike-triggered kernels $\eta(t-t_i)$:

$$\hat{S}(t) = \sum_{t_i} \eta(t-t_i), \tag{4}$$

where a spike is added in an online and incremental fashion when the difference between the input signal and the signal approximation exceeds a positive dynamic threshold $\vartheta(t)$ from below:

$$u(t) = S(t) - \hat{S}(t) > \vartheta(t), \tag{5}$$

where $u(t)$ denotes the neuron's membrane potential. Upon emitting a spike at t_i, the spike-triggered refractory response $\eta(t-t_i)$ is subtracted from $S(t)$ and added to $\hat{S}(t)$. The part of $S(t)$ larger than the minimal value of the threshold $\vartheta(t)$ is thus encoded as $\hat{S}(t)$ in a spike train t_i. It is decoded at the postsynaptic target neuron where the resultant postsynaptic currents are added as weighed versions of the refractory response $\eta(t)$. The resultant postsynaptic current in target neuron j, $I_j(t)$ induced by presynaptic spikes t_i from multiple presynaptic neurons i, is then computed as:

$$I_j(t) = \sum_i \sum_{t_i} w_{ij} \eta(t-t_i),$$

where w_{ij} is the weight between presynaptic neuron i and postsynaptic neuron j. The refractory response kernel $\eta(t)$ is adaptive and controlled through the dynamic threshold $\vartheta(t)$:

$$\eta(t-t_i) = \vartheta(t_i)\, \varkappa(t-t_i),$$

where $\vartheta(t_i)$ is the effective threshold at the time of spiking, $\varkappa(t-t_i)$ is a spike-triggered exponentially decaying response kernel shaping the refractory response due to the spike at t_i with normalised height $\varkappa(0) = 1$. Thus computed, the average of the sum of η kernels approximates the mean of the (rectified positive) signal $S(t)$.

We model the dynamic threshold $\vartheta(t)$ as multiplicative adaptation after [3]:

$$\vartheta(t) = \vartheta_0 + \sum_{t_i} m_f \vartheta(t_i)\, \gamma(t-t_i), \tag{6}$$

where ϑ_0 is the baseline threshold set to some (small) fixed value. A multiplicative factor m_f of fixed size regulates the threshold dynamics, where the ratio between ϑ_0 and m_f determines the asymptotic firing rate of the neuron for large activation values. The adaptation kernel $\gamma(t)$ is computed as a sum of exponentials: $\gamma(t) = \sum_n \gamma_n \exp(-t/\tau_{\gamma_n})$ with weights γ_n normalised to one such that $\gamma(0) = 1$. A few components are sufficient to mimic limited long-memory adaption as experimentally reported in e.g. [21]; here we use either one or two components. Note that the internal state of the neuron is fully determined by the two kernels $\eta(t)$ and $\gamma(t)$, and both of these kernels can be expressed as sum of exponentials: one for $\varkappa(t)$ and one or two for $\gamma(t)$. The neuron state update can thus be efficiently computed by updating these exponential functions as simple (memory-less) dynamical systems.

As noted, the signal approximation $\hat{S}(t)$ is computed as a sum of variable height kernels: it is this signal that is communicated through a sequence of spikes to the next, postsynaptic, neuron. At the postsynaptic neuron, a filter $\phi(t)$ smooths the (weighted) η kernels, which suppresses high frequency noise and reconstructs the signal as in the APSDM receiver [26]. In the network, for each arriving spike the corresponding η kernel is multiplied by the weight of the connection and added to the current $I(t)$ in the post-synaptic neuron. Since the height of the η kernel is adaptive, in this treatment each spike t_i effectively has a *height* $\vartheta(t_i)$. Thus, the ASN communicates spikes with an analog 'height' rather than binary valued spikes.

Adaptive signal encoding and decoding

The (unsmoothed) signal approximation $\hat{S}(t)$ computed by the spiking mechanism in the adaptive spiking neuron computes a ReLU function: plotted in Figure 9a is both the firing rate (dashed) and the mean and standard deviation of the signal approximation $\hat{S}(t)$ (solid) for increasing signal values S, for two different ratios of ϑ_0 and m_f. While the firing rate saturates, the approximation $\hat{S}(t)$ remains linearly growing with increasing S, albeit with increasing variance as the number of spikes used to encode the signal remains the same.

Since the ratio of the baseline threshold ϑ_0 and the multiplicative factor m_f determines the saturating firing rate, this ratio also determines the precision of the en-

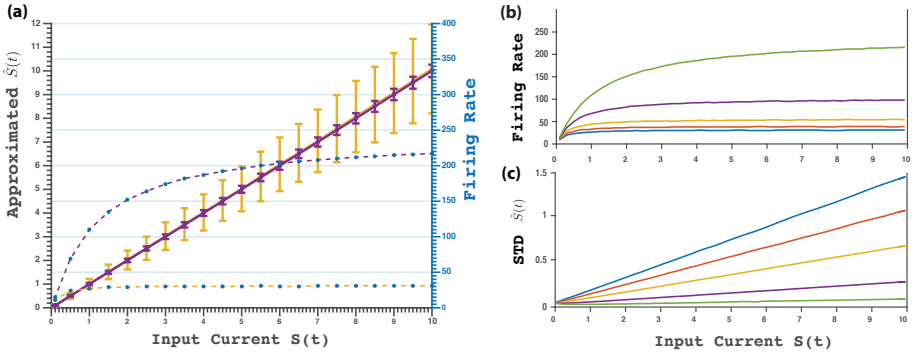

Figure 9 (a) Firing rates (dashed lines, right axis, computed over a 1 s time window), output signal $\hat{S}(t)$ with standard deviation (solid lines, left axis) of an ASN ReLU neuron for two firing rate regimes ($\vartheta_0 = 0.1$, $m_f = \vartheta_0$ (yellow), $m_f = 0.1\vartheta_0$ (purple)). Colors are the same for firing rate and corresponding signal $\hat{S}$. (b) Firing rate for 5 different values of $m_f = 0.01, 0.025, 0.05, 0.075, 0.1$ and $\vartheta_0 = 0.1$. (c) Standard deviation (std) for 5 different values of m_f. Colors correspond between (b) and (c).

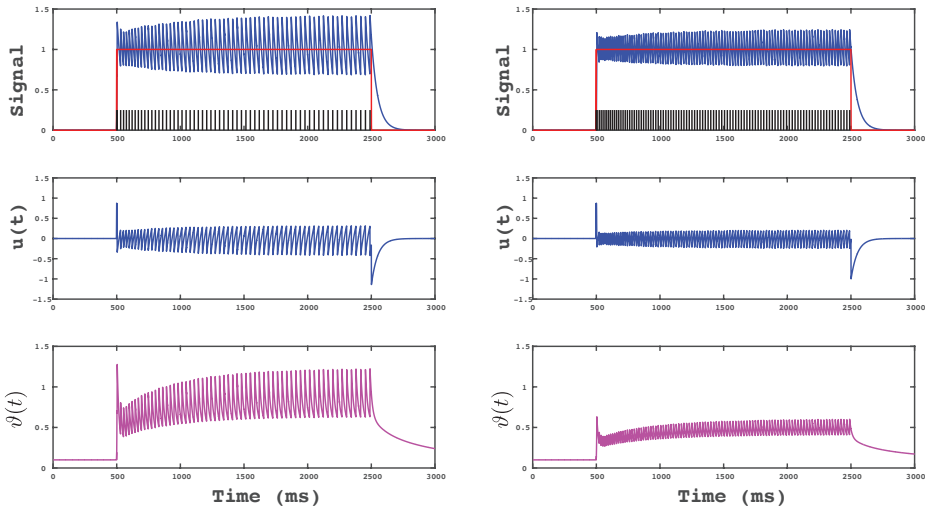

Figure 10 Encoding of two fixed size step functions for $S(t)$, illustrating the decreasing variance of the signal approximation $\hat{S}(t)$ for increasing firing rates. Parameters: $m_f = 1\vartheta_0$ (left) and $0.1\vartheta_0$ (right) for $\vartheta_0 = 0.1$.

coding. The inverse relationship between saturating firing rate and coding precision is plotted in Figure 9b,c for five different values of m_f/ϑ_0. We observe that the standard deviation linearly increases with signal magnitude, and inversely relates to the saturating firing rate.

In Figure 10, we illustrate signal encoding with the ASN with more or less spikes. In the top row we plot the encoding of a step function $S(t)$ (red) with a sum of adaptive kernels, $\hat{S}(t)$ (blue). The black dashes denote the spikes: the variance of $\hat{S}(t)$ decreases when more spikes are used. In the middle row, the membrane potential $u(t)$ is plotted for both cases, and in the bottom row the dynamical threshold $\vartheta(t)$. As can be seen, a lower firing rate is achieved by a higher average threshold and correspondingly larger refractory resets $\eta(t)$.

The time constant of the refractory response $\eta(t)$ is determined by τ_x: the value of this constant determines how much 'future' signal each spike transmits. To encode step functions as in Figure 10, a decay constant that better matches the temporal correlation in the approximated signal will yield a better approximation. For a step function, this effect is plotted in Figure 11. Shown is the sum squared error (SSE) approximating a 1 second segment of a step function with a fixed firing rate (35 Hz) for various values of τ_x. Increasing τ_x strongly reduces the SSE (blue line, left axis). The lower SSE however comes at the expense of responsiveness: when the step function steps back to 0, it takes longer before the approximation correctly matches the new, lower value. Plotted also (orange line, right axis) is the time it takes before the signal approximation is below 0.05 after stepping down.

Implementation

In the examples and in our network implementations, we use time constants that are roughly of the order of the corresponding values in biological spiking neurons, such as time constants of PSCs, membrane time constant and refractory response kernels, to obtain firing rates for active neurons in the range of $1{-}100$ Hz, compatible with what is observed in biology. We use a time constant of $\tau_x = 50$ ms for the exponential decay of the x kernel. The γ kernel was approximated as either a single decaying exponential or the sum of two exponentially decaying functions,

$$\gamma^1(t) = e(-t/\tau_{\gamma_1}),$$

or

$$\gamma^2(t) = \frac{1}{\gamma_1+\gamma_2}\left[\gamma_1 e(-t/\tau_{\gamma_1}) + \gamma_2 e(-t/\tau_{\gamma_2})\right],$$

with time constants $\tau_{\gamma_1} = 15$ ms and $\tau_{\gamma_2} = 100$ ms and respective weights $\gamma_1 = 0.1$ and $\gamma_2 = 0.01$. Adding additional components increases the long-memory adaptation behaviour of the ASN, but two components suffice here as we are not considering time-varying signals. We use a time constant of $\tau_{\text{smooth}} = 2.5$ ms for the signal reconstructing exponential smoothing filter $\phi(t)$ in all ASN units except for the output neurons. In the output units activity was filtered with an exponential filter with a longer time constant of $\tau_{\text{rout}} = 50$ ms, to compare activations between outputs for classification purposes. The simulations are computed with time steps of size 1 ms.

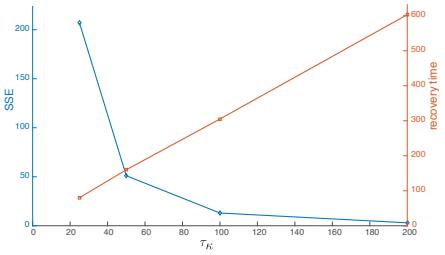

Figure 11 Error and responsiveness when encoding a step function with different $\eta(t)$ (or EPSP) time constants τ_x. Left axis: sum-squared error. Right axis: responsiveness when switching back.

Adaptive spiking neural networks (ASNN)

We implement adaptive spiking neural networks where the units are comprised of the ASNs described above. Inherently, the ASNNs compute over time-continuous input signals; most straightforward and standard applications of deep neural networks are concerned with classification tasks, such as determining the digit in an image (Figure 12a). To compare classification performance between a standard ANN and an SNN, an image is presented for 500 ms to the network, and we record from the output neurons to determine the classification. The image is thus taken as input to the network for every time step in the SNN, which may be as small as 1 ms (1000 Hz) (illustrated in the inset in Figure 12b).

Since our ASNs communicate analog valued spikes rather than binary spikes, the question is how the classification problem thus phrased compares to a standard ANN which also communicates with analog values. For an image, an ANN can obviously compute the classification in one go, essentially using just one 'analog spike'. We argue that the correct comparison between SNNs, ASNNs and ANNs is to treat the classification problem as a time-continuous problem. While the stimulus is present the network has to compute classifications. For both SNNs and ASNNs this is inherent to the operation of the network, while an ANN would need to sample the input at a certain frame rate. This is illustrated in Figure 12b: the ANN computes the classification for each frame for the entire network, and the computational complexity scales linearly with the frame rate (illustrated in the right part of Figure 12b). In contrast, the SNN and ASNN implement an asynchronous model of ongoing neural computa-

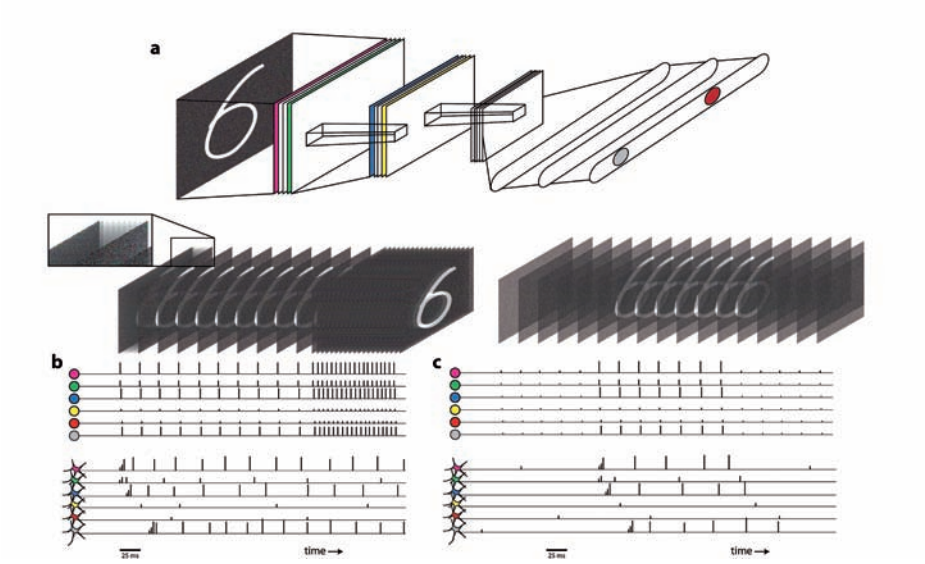

Figure 12 (a) deep convolutional neural network. (b) ANN versus ASNN classification. The ANN is computed for every frame, for the ASNN the neuron are updated at a fine resolution (inset), but network activity is asynchronous and sparse. Right part of the sequence: increasing the frame-rate increases ANN computations and not ASNN. (c) Flanked noise classification. The ANN computes at a fixed frame rate, also for noise input that activates feature neurons only slightly. For the ASNN, the input neurons rarely cross threshold and the network firing rate is very low for noise; spikes are only emitted when frames with features are presented.

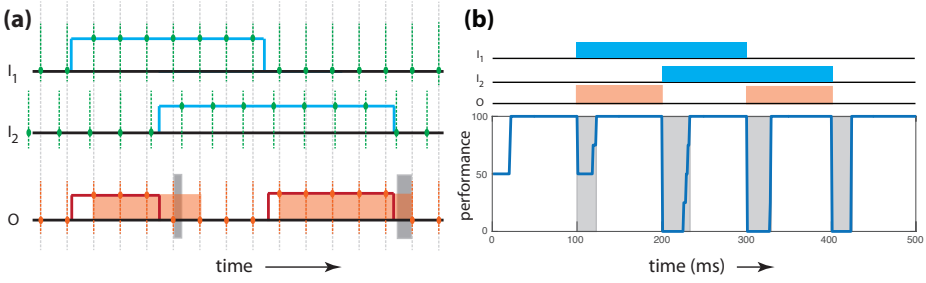

Figure 13 Asynchronously computing XOR: (a) illustration with inputs arriving asynchronously (dotted green lines), and XOR computed synchronously with the top (fastest) input rate. Due to the synchronous nature of computing, additional errors are made, like the shaded areas in the bottom figure. Processing the input asynchronously at their respective sample rates, the right shaded area would be avoided. (b) Asynchronous processing of XOR in a 2-5-1 ASNN network capable of computing XOR with about 15 Hz average firing rate and neurons using $\tau_x = 25$ ms. Novel input is processed at the update rate of the neurons (1 ms); the delay in classification when patterns switch is now determined by τ_x (shaded areas).

tion where the neurons are updated each small time step (1 ms), and communication between neurons is both localised (to active neurons), and a function of desired neural coding precision rather than frame-rate. Another benefit of the ASNN implementation is illustrated in Figure 12c: when no features are present in the frame, the spiking neural network does not generate spikes, or only very sparingly, whereas the ANN still computes the entire network every frame. The downside of asynchronous neural computation is that there is an inherent latency between input presentation and output: in each layer, the ASN applies an averaging filter to the spike-triggered input currents it receives.

Asynchronous neural computation offers benefits both for computing and for processing sensory motor data: with neural updating and network updating decoupled, sensory inputs (and actuator outputs) can be sampled at the high neural update frequency. This avoids the well known problem of synchronized processing [18]; the ASNN however cannot respond much faster to changing inputs than the τ_x time constant. This is illustrated in Figure 13 for the simple problem of streaming XOR: the network, using about a 15 Hz average firing rate, computes XOR from the two inputs. The bottom panel shows performance, and demonstrates that the network is still capable of responding faster to changes in input (≈ 25 ms) than a correspondingly synchronous sample rate.

Computational complexity

An examination of the computational cost and bandwidth requirements demonstrates the mixed ANN and SNN properties of the ASNN. In Table 1, these costs are specified. The ASNN shares the firing rate dependent network bandwidth cost with the SNN, but at an ANN-like cost per spike, and network delay is determined by the spike-decay time constant τ_x, (presumably) the same as in the SNN (not demonstrated in the literature). Since spike impact is computed as the product of spike height and connection weight, the ASNN shares the ANN's cost in terms of multiplications per spike/update, and the neuron update cost of the ASNN scales as an SNN.

This analysis ignores the fact that spikes in the ASNN (and SNN) are heavily localized to a subset of neurons: many neurons are silent while a few are active. Sparse and localised communication potentially offers a benefit to deep neural networks, as densely connected neural networks tend to be limited by the bandwidth required to read and write the appropriate weights from memory [24]. Thus reasoned, for an ASNN that incurs a 100 ms delay to compete in terms of bandwidth used with an ANN, it can use at most a firing rate of 10 Hz on average per neuron, since an ANN sampled with 10 Hz would achieve the same worst case delay. This ignores the benefit of the ASNN being able to process in principle a 1000 Hz frame rate. The exact benefit of sparse activity depends on the degree of sparseness and the degree to which parallel hardware can exploit sparseness.

	ANN	**SNN**	**ASNN**
Network bandwidth	$C \cdot [P+O] \cdot H_a$	$C \cdot O \cdot F_s$	$C \cdot [P+O] \cdot F_p$
Network delay	$1/H_a$	$\propto \tau_x + c \cdot L$	$\propto \tau_x + c \cdot L$
Network multiplications	$C \cdot P \cdot H_a$	—	$C \cdot P \cdot F_p$
Neuron multiplications	$H_a \cdot f(\mathrm{Re}LU)$	$U_s \cdot f(\mathrm{Re}LU)$	$U_p[3 + f(\text{threshold})]$

Table 1 Computational Cost. C: number of connections, P: pulse precision, H_a: ANN update frequency, O: addressing overhead, F_s: SNN firing rate, F_p: ASNN average firing rate, L: network depth (layers), U_s: update frequency of SNN, U_p: update frequency of ASNN, c: a constant.

Experimental networks

We demonstrate the ASNNs described above in fully connected feed-forward neural networks (FFNNs) and in a convolutional neural network (CNN) [14]. These architectures were first trained on standard datasets — IRIS, SONAR, and MNIST — with standard ANNs comprised of rectified linear (ReLu) neurons. The corresponding spiking neural networks were created by using the same weights and network connectivity as the trained architectures, and replacing the ReLU neurons with ASN units — this approach allows us to focus on spike-based coding and for now side steps the question of spike-based learning.

We selected well-known benchmark datasets of increasing complexity to demonstrate the robustness of the presented approach. The IRIS dataset is a classical non-linearly separable 'toy' dataset containing three classes — three types of plants — with fifty instances each, to be classified from four input attributes. Similarly, the SONAR dataset [12] contains 208 entries of sonar signals divided in 60 energy measurements in a particular frequency band, to be classified in metal cylinder or simple rocks classes. Lastly, we use the MNIST dataset [14], which has been a standard testbed for novel image classification methods. It is composed of $60{,}000$ entries of handwritten digits for the training set and $10{,}000$ entries for the validation set.

To carry out classification, for each instance the input neurons receive input current $I(t)$ corresponding to the respective feature values, for a simulation duration of 500 ms. During this period, input neurons generate spikes that are instantaneously transmitted to the next layer. There, the corresponding weighted PSCs are added to the membrane potential $u(t)$ through the smoothing filter $\phi(t)$; note that the smoothing filter effectively causes a delay in signal transmission of order τ_{smooth} per layer. This process is repeated for each successive layer in the network. Output values as used for classification are computed as internal current $I(t)$ in the output neurons, smoothed with longer time constant τ_{rout} for stable performance. At every 1 ms time step t of the simulation, classification performance is computed over all instances of the respective dataset from the outputs $I(t)$ at that time step t. Details for the architecture, training and parameters used are given in a box at the end of this article.

Computing with spikes

For all three datasets and the corresponding four network architectures, we computed the ANN performance and compared that to the ASNN performance. Figure 14 shows classification performance obtained for IRIS, SONAR and MNIST by the various ASNNs as a function of average firing rate in the network (and hence neural coding precision) during classification, obtained by varying the ratio of m_f and ϑ_0. We find that for all benchmarks we achieve performance with the ASNN identical to that of the corresponding ANN once a certain minimum firing rate is used, corresponding to the minimal required neural coding precision in the network. The networks that classify the IRIS and SONAR benchmarks require fairly high firing rates compared to the two MNIST architectures. Since the former architectures are comprised of far fewer neurons as compared to the MNIST networks, this suggests that in such smaller networks the coding precision needs to be quite high.

The different firing rate regimes were obtained by varying the multiplicative factor m_f as a function of ϑ_0, between $0.1\vartheta_0$ and $3\vartheta_0$, with $\vartheta_0 = 0.0128$ for the IRIS dataset, in 30 different simulations. The threshold $\vartheta_0 = 0.0128$ was selected such that the smallest positive input values in the training set were still encoded. For SONAR, we carried out simulations with m_f ranging between $0.1\vartheta_0$ and $3\vartheta_0$, using $\vartheta_0 = 1e^{-4}$. For the MNIST dataset we simulated both an FF-ASNN and C-ASNN architecture. For the FF-ASNN we carried out 35 simulations with m_f ranging between $0.1\vartheta_0$ and $3.5\vartheta_0$, using $\vartheta_0 = 3.9e-3$. For the MNIST networks, compared to the IRIS and SONAR networks, we find that performance is stable over a much greater range of firing rates. For each simulation we computed the time to which 101% of the minimum classification error is reached (Matching Time, MT), e.g., for MNIST-cnn this is when the performance exceeds 99.13%. Given parameters ϑ_0 and m_f, we considered the ASNN network as having performance identical to the corresponding ANN if, in the time window from MT to the end of the simulation (500 ms), the performance stays, on average, above the 101% error threshold. The variance is computed over the same time window, while the firing rate is computed in a time window of 100 ms at the end of the simulation. At low firing rates, the ANN performance is exceeded for some ranges by chance; the high neural coding precision for higher firing rates results in more stable performance, as can be seen in the low variance of the performance on the right part of Figure 14.

For all four ASNNs, we noted both the required minimum firing rate (as set through the ratio of m_f and ϑ_0) to reach the 101% error threshold, and the corresponding simulation time when this performance is first reached. We refer to these values as the Matching Firing Rate (FR) and the Matching Time (MT), and the results are shown in Table 2 in the column 'Lowest FR'. For MNIST, we find that the response time for the FF-ASNN is substantially faster as compared to the C-ASNN. This is likely caused by the fact that the C-ASNN is a deeper network. Additionally, we determined the lowest Matching Time and corresponding Firing Rate (Table 2 in the column 'Lowest MT'). We see that for the large MNIST networks, Matching Time

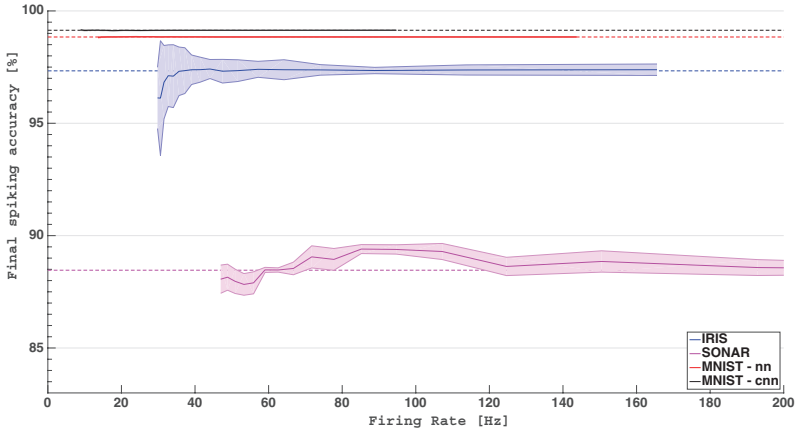

Figure 14 Classification performance on IRIS, SONAR, MNIST (MNIST-nn for FF-ASNN and MNIST-cnn for C-ASNN) for various average firing rates. Dashed: performance of original ANN.

DataSet	ANN	ASNN	Lowest FR		Lowest MT	
		P(%)	FR	MT	FR	MT
IRIS	97.33	97.33	36	107	41.4	46
SONAR	88.46	88.46	59.7	80	77.1	71
MNIST-nn	98.84	98.84	14.6	15	17.3	12
MNIST-cnn	99.14	99.14	8.6	87	10	8.9

Table 2 Performance (%), Matching Firing Rate (FR) (Hz) and Matching Time (MT) (ms).

improves substantially at limited cost in terms of FR. In general, we find that the Matching Time increases with lower firing rates (not shown).

Switching

We also computed the Matching Time to determine that time that input needs to be presented to the network before the output classification reaches ANN performance (101% of the minimum classification error). A more general streaming setting however is one where one stimulus is presented, followed by another stimulus. We illustrate this case in Figure 15: first, white noise is presented to the network for 100 ms, followed by the presentation of a digit, which after 100 ms is then switched to another digit. Shown is the average activation in each layer of the MNIST-cnn ($\vartheta_0 = 3.9e-3$, $m_f = 3\vartheta_0$) for 1000 random stimulus switches, as well as the average activation $S(t)$ in the output neurons and the classification performance. White noise has been reproduced by presenting a (different) Gaussian-noise sampled image with $\mu = 0$ and $\sigma = 0.5\vartheta_0$, at each ms frame. We see that noise only stimulates the first layer, and fails to substantially activate subsequent layers. Once the first actual digit is presented, the network rapidly and correctly recognizes this digit. After 200 ms the permuted images are presented: the classification performance for the new dataset reaches the 101% error threshold after a switching time of $\mathrm{ST} = 186$ ms. This switch from one digit to another is determined by — substantially longer — recovery time due to τ_x. Switching time can be improved by decreasing τ_x, but at the expense of an increase in firing rate.

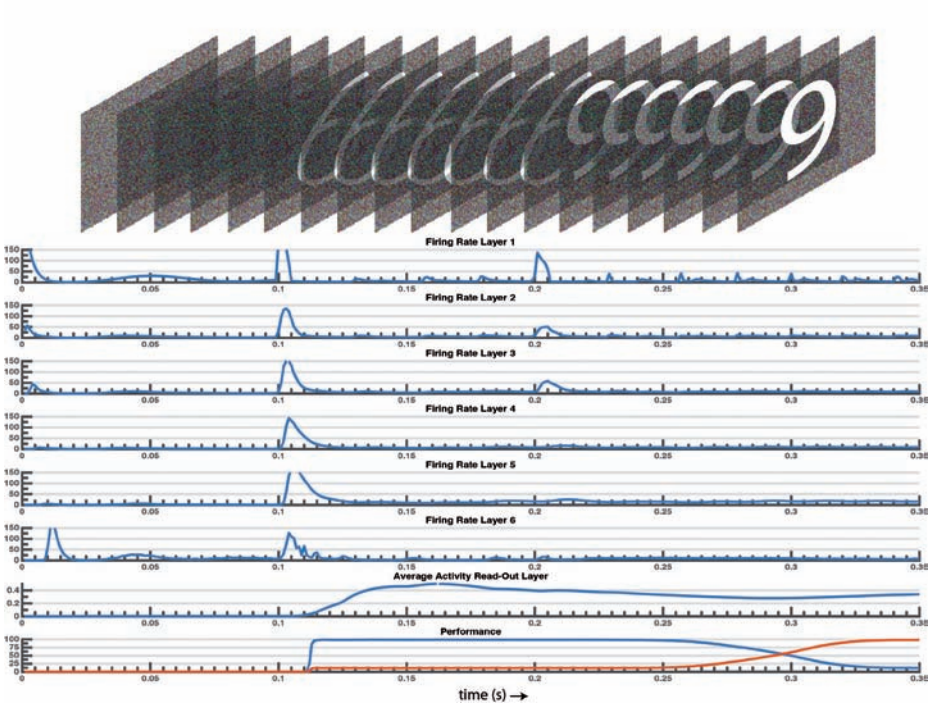

Figure 15 Switching example with C-ASNN. Top: an example of the switching images provided to the network. Middle, rows 1-6: the firing rate of the network's 5 layers plus the read-out layer. Middle, row 7: the average activity of the read-out layer, computed by filtering the internal state of the neurons. Note that, during the noise presentation, although a firing activity in the read-out layer is present, the internal state is silent — a rapid increase in the average activity signals that a classification is made. Bottom: the classification performance through time showing the switch between two test sets of a 1000 digits each.

Discussion and conclusion

We introduced deep neural networks and explained how they are presumed to relate to biology. Given some of the deficiencies of present deep neural networks, we focused on the question of efficient and asynchronous neural coding with spiking neurons.

Spiking neuron models like the ASN presented here capture many important adaptation phenomena in real neurons, and by coupling the synaptic plasticity model, we ensure that downstream neurons appropriately account for adaptation in presynaptic neurons. Thus, it is a prediction of this work that a tight coupling exists between neural adaptation and synaptic plasticity.

At the same time, we demonstrated that the resulting neural network model can replace a standard ANN in a one-to-one manner, without loss of performance, while using an asynchronous and sparse model of spike-based neural computation. As such, the presented ASNN can be considered as a novel paradigm for neural coding with spiking neurons, with an almost direct correspondence to biological spiking neurons.

In particular, we show that the proposed ASNNs can carry out neural computation with performance identical to the corresponding ANN for a number of classical benchmark datasets of increasing network size and complexity. Compared to an otherwise identical SNN that uses Poisson spiking neurons the presented approach has better or identical performance while using a much lower firing rate in the network. Additionally, due to the large dynamic range of the ASNs, no reweighting or normalization of the network was necessary: the ASNs function as drop-in spiking neuron replacements for the ReLU neurons in the standard ANNs. Effectively, the ASN computes using *adaptive* asynchronous sigma-delta pulse modulation, which is necessary because — unlike electrical circuit signals — the signals inside a neural network with ReLU neurons are not bounded to some fixed dynamic range. Note that though we focus here on standard neural networks without recurrence or memory, we recently showed that a similar approach can be applied to networks with memory [20], to learn cognitive tasks, like

tasks that require remembering a value for a number of steps and then being able to act on this value.

Compared to classical ANNs, the computations of the ASNNs are asynchronous, event driven and sparse. To truly exploit the efficiency of sparsely active asynchronous spiking neural networks, efficient GPU or ASIC implementations need to be created. Current CNN implementations are heavily optimized for carrying out convolutions on GPUs, an operation which closely fits the GPUs parallel architecture. For sparsely active neural networks, where most neurons are not active at any given time step, novel approaches need to be developed: since typically for any stimulus only a subset of neurons is active, fast caching methods are likely to hold promise. As most networks of spiking neurons, the reduction in communication between the neurons is traded against more complex dynamics in the neuron; since there are typically orders of magnitude fewer neurons than connections, this trade-off can be worthwhile provided that the neuron model requires limited memory and computation. The ASN model presented here can be computed with only a few variables (principally the components of the γ and η kernels), which when formulated as simple dynamical systems can be computed in a memory-less fashion, without tracking previous spike-times. Emerging hardware architectures like Intel's Loihi chip are eminently suitable for exploiting the efficiency of spike based computation.

Our adapting neurons effectively use analog spikes: each spike is associated with a refractory kernel of different height. In principle, the analog value of a spike can be reconstructed at the postsynaptic neuron from just the time since the previous spike, but at considerable computational expense. Compared to standard (analog) ANNs, the ASNNs compute in an asynchronous and localized manner: input information can be presented to the network at the precision with which neurons are updated, while the rate of information exchange in the network is determined by the neural coding precision required for classification. The network can thus process for instance 1000 Hz input frames when neural updates are carried out with 1 ms time steps: in this manner, new input can be processed almost immediately — albeit with the delay incurred in the consecutive layers. The neural activity is also localized, in that only a subset of neurons is really activated, emitting many spikes, and most neurons are silent or only very sparsely active. Since bandwidth, as used for reading weights from memory, is typically the limiting factor when computing an ANN, the sparse and localized neural computation offers a potentially more efficient way of time-continuous neural computing.

We showed how we can relate spike-based coding to the analog signals that standard artificial neural networks compute with. Such a translation allows us to design sparsely active neural networks while using the existing frameworks (like Tensorflow and PyTorch) to train the analog counterparts of these spiking networks. This is of course sufficient when the aim is to deploy a trained network on low-power hardware. To include learning, we have to develop spike-based learning algorithms. A straightforward solution there is to note that the error that is backpropagated in the error backpropagation learning rule could potentially be carried by a separated network of spiking neurons. Peter O'Connor recently showed some work in that direction [17], but in general the problem with this approach is that the approximation error in the 'spiking' AD/DA conversion becomes too large when the neural networks become very deep. To overcome this, different approaches to learning in deep networks may need to be found, where biology will again be a source of inspiration as most are convinced that the brain does not use error-backpropagation but rather relies on smart and data-efficient forms of learning that learn the natural structure of the world without being given explicit examples, as is needed for error-backpropagation.

Feed-forward neural networks

We trained fully connected FFNNs using dropout [25] to approximately match performance with state-of-the-art. We trained a four layer FFNN of size $[4-30-30-3]$ on the classical IRIS dataset with a dropout rate of 0.5, learning rate of 0.1, for 800 epochs. We used half of the dataset for training, and we obtained 97.33% on the validation set. For the SONAR dataset, we trained a four layer FFNN of size $[60-50-50-2]$, using the training set division reported in [12] for the angle-dependent experiment. We used a dropout rate of 0.5, learning rate of 0.2, and we trained for 1000 epochs to obtain 88.46% accuracy on the validation set. For the MNIST dataset, we used the trained network reported in [7] to directly compare with the method there. In [7], the authors trained a $[784-1200-1200-10]$ network, with a dropout rate of 0.5, learning rate of 1 and momentum of 0.5. With this network, we obtained 98.84% accuracy on the MNIST validation set (code and trained network were available online [27] using a modified version of the DeepLearnToolbox [19, 28]. As in [7], for all datasets the input values were scaled to the range $[0,1]$. We refer to the FFNNs that use ASN ReLU units as feed-forward adaptive spiking neural networks (FF-ASNN).

Convolutional neural networks

CNNs have become a standard tool for image classification tasks [14], and they generally outperform classical FFNNs. In [7] a competitive ReLU CNN implementation for MNIST was presented: we apply the ASN network to this architecture and compare our results to those obtained in [7]. The pre-trained network consists of a $[28\times28-12c5-2s-64c5-2s-10o]$ CNN, where 28×28 corresponds to the input image size, $N,\ c,\ K$ are the N-convolutional kernels of size K, $M,\ s,\ J$ are the M-averaging pooling filters of size J, and o is the size of the output layer; note that this network is available online [27]. Neurons in each of these layers use the ReLu activation function, and we can again map the ANN directly to our ASNN by substituting each ReLU unit with the adaptive spiking neuron. We refer to the CNNs equipped spiking neurons as convolutional adaptive spiking neural networks (C-ASNN).

References

1 Ch. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press, 1995.

2 M. Boerlin and S. Denève, Spike-based population coding and working memory, *PLoS Computational Biology* 7(2), February 2011, e1001080.

3 S. M. Bohte, Efficient spike-coding with multiplicative adaptation in a spike response model, in *NIPS* 25, MIT Press, 2012, pp. 1844–1852.

4 S. M. Bohte and J. O. Rombouts, Fractionally predictive spiking neurons, in *NIPS* 23, MIT Press, 2010, pp. 253–261.

5 R. Brette, Spiking models for level-invariant encoding, *Front. in Comp. Neurosc.* 5 (2011).

6 D. B. Chklovskii and D. Soudry, Neuronal spike generation mechanism as an oversampling, noise-shaping a-to-d converter, in *NIPS* 25, MIT Press, 2012, pp. 503–511.

7 P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-C. Liu and M. Pfeiffer, Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing, *IEEE IJCNN,* July 2015, pp. 1–8.

8 Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko and Trevor Darrell, Long-term recurrent convolutional networks for visual recognition and description, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2625–2634.

9 A. L. Fairhall, G. D. Lewen, W. Bialek and R. R. de Ruyter van Steveninck, Efficiency and ambiguity in an adaptive neural code, *Nature*, 412(6849) (2001), 787–792.

10 W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity,* Cambridge University Press, 2002.

11 Wulfram Gerstner, A framework for spiking neuron models: The spike response model, *Handbook of Biological Physics*, Volume 4, Elsevier, 2001, pp. 469–516.

12 R. P. Gorman and T. J. Sejnowski, Analysis of hidden units in a layered network trained to classify sonar targets, *Neural Networks* 1(1) (1988), 75–89.

13 Alan L. Hodgkin and Andrew F. Huxley, A quantitative description of membrane current and its application to conduction and excitation in nerve, *The Journal of physiology* 117(4) (1952), 500–544.

14 Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE* 86(11) (1998), 2278–2324.

15 Zachary F. Mainen and Terrence J. Sejnowski, Reliability of spike timing in neocortical neurons, *Science* 268(5216) (1995), 1503–1506.

16 Ilya Nemenman, Geoffrey D Lewen, William Bialek and Rob R de Ruyter van Steveninck, Neural coding of natural stimuli: information at submillisecond resolution, *PLoS Computational Biology* 4(3) (2008), e1000025.

17 Peter O'Connor, Efstratios Gavves and Max Welling, Temporally efficient deep learning with spikes, arXiv:1706.04159, 2017.

18 E. Olson, A passive solution to the sensor synchronization problem, in *Int. Conference on Intelligent Robots and Systems (IROS), IEEE/RSJ*, IEEE, 2010, pp. 1059–1064.

19 R. B. Palm, Prediction as a candidate for learning deep hierarchical models of data, 2012.

20 Isabella Pozzi, Roeland Nusselder, Davide Zambrano and Sander Bohté, Gating sensory noise in a spiking subtractive LSTM, submitted, 2018.

21 Ch. Pozzorini, R. Naud, S. Mensi and W. Gerstner. Temporal whitening by power-law adaptation in neocortical neurons, *Nature Neuroscience* 16(7) (2013), 942–948.

22 B. D. Ripley, *Pattern Recognition and Neural Networks*, Clarendon Press, 1996.

23 D. E. Rumelhart, G. E. Hinton and R. J. Williams, Learning representations by back-propagating errors, *Nature* 325 (1986), 533–536.

24 L. Ślażyński and S. Bohte, Streaming parallel gpu acceleration of largescale filter-based spiking neural networks, *Network: Computation in Neural Systems* 23(4) (2012), 183–211.

25 N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.* 15(1) (2014), 1929–1958.

26 Y. C. Yoon. LIF and simplified SRM neurons encode signals into spikes via a form of asynchronous pulse sigma-delta modulation, in *IEEE TNNLS*, 2016, pp. 1–14.

27 http://github.com/dannyneispikingJelu_conversion.

28 https://github.com/rasmusbergpalm/DeepLearnToolbox.