Spike-based Long Short-Term Memory networks

MSc Thesis

Roeland Nusselder
Mathematical Institute
Utrecht University
Machine Learning
CWI Amsterdam

Project supervisor:
Prof. dr. R.H. Bisseling
Daily supervisors:
dr. S.M. Bohte
dr. D. Zambrano
Second reader:
dr. C. Spitoni

July 10, 2017

Abstract

Spiking neural networks are investigated both as biologically plausible models of neural computation and also as a potentially more efficient type of neural network. While convolutional spiking neural networks have been demonstrated to achieve near state-of-the-art performance, no such networks have been presented for gated recurrent neural networks. Recurrent neural networks in the form of networks of gating memory cells have been central in state-of-the-art solutions in problem domains that involve sequence recognition or generation. Here, we design an analog gated Long Short-Term Memory (LSTM) cell where its constituent neurons can be substituted for efficient spiking neurons. These adaptive spiking neurons implement an adaptive form of sigma-delta coding to convert internally computed analog activation values to spike-trains. For such neurons, we derive the effective activation function, which resembles a half sigmoid. We show how analog neurons with such activation functions can be used to create an analog LSTM cell, networks of these cells can then be trained with standard backpropagation. We train these LSTM networks on a noisy and noiseless version of the original sequence prediction task from Hochreiter & Schmidhuber (1997), and also on a noisy and noiseless version of a classical working memory reinforcement learning task, the T-Maze. Substituting the analog neurons for corresponding adaptive spiking neurons, we then show that almost all resulting spiking neural network equivalents correctly compute the original tasks.

Contents

Chapter 1

Introduction

The goal of this work is to construct a brain-like model which is able to find patterns in time-series. A possible model for this is an artificial neural network, which is a graph where the neurons (the vertices) are connected to each other by weights (the edges). The output of a neuron is the result of a non-linear function applied on the weighted sum of the inputs of the neuron. The neurons are often arranged in such a way that multiple layers of neurons are created. The first layer is called the input layer, the last is the output layer and the layers in between are the hidden layers. During an inference pass the network computes the mapping from the input (e.g. an image) to the output (e.g. a classification), while during the training phase the weights of the network are adjusted to improve this mapping. To enable a neural network to process and predict time series Long Short-Term Memory (LSTM) units can be implemented in the neural network. Deep learning [1], which is the use of artificial neural networks, is inspired by the human brain, which consists of approximately 10^{11} neurons connected with each other by 10^{15} synapses. However, the type of neural network that has been shown to be very successful on a wide variety of tasks, like image recognition [2], speech recognition [3] and reinforcement learning problems [4], is not very biologically plausible, meaning that they are a great oversimplified model of the brain. In this work spiking neurons will be used to construct a brain-like model to extract patterns on time-series.

Spiking neural networks are a variant of artificial neural networks and consist of spiking neurons that communicate by using brief electric pulses – spikes – making them closer to biological neurons, which only fire 1-5 times per second on average [5]. They are being investigated as potential models for computational and energy efficiency and are also attractive since unlike artificial neural networks they have a natural notion of time. Spiking neural networks can be more efficient than analog neural networks for continuous time input, like video or audio. This is especially the case in a network that has already been trained. An analog neural network needs to do a full inference pass for every new input. So if a video has 60 frames per second,

the neural network needs to do 60 inference passes per second. For much continuous time data, the new input is very similar to the previous input. The states of the neurons in the new inference phase will be very similar to the states during the previous inference phase. It seems to be a waste to completely recompute the full network every time a new input is presented. Spiking neural networks are an attractive alternative, since a spiking neuron is more active when its input changes and less active when its input stays stable. This is unlike a traditional neural network, where the computational activity is the same for changing input as it is to stable input. A spiking neural network has a similar advantage over an analog neural network when it comes to handling asynchronous sensor input. A neural network that controls a self-driving car might have input coming from both a video camera with 60 frames per second and a radar sensor with 1000 outputs per second. The states of the neural network can be updated 60 times per second, which means that it would miss possibly essential input from the radar, or the states can be updated 1000 times per second, which will require an unnecessarily large amount of computational power. A spiking neural network could be a solution for this problem, by allowing the network to only update the states of the neurons that should change, instead of updating all the neurons in the network, without missing any input information.

More efficient neural networks might be achieved in the future by using spiking neural networks on standard hardware like Graphics Processing Units (GPUs), application-specific integrated circuits (ASICs) like Tensor Processing Units (TPUs) [6] or by mimicking spiking neurons with analog hardware [7]. It is still unclear which of these methods is best for spiking neural networks and even if spiking neural networks are the best way to implement energy efficient neural networks. But there is a great demand for energy efficient neural networks, since they can be used on a wide variety of problems, from assistants on mobile phones that need speech recognition, to drones and self-driving cars, see Figure 1.1.

The two main types of neural networks are feedforward neural networks [8] and recurrent neural networks [9] (RNNs). Convolutional neural networks [10] are the most common type of feedforward neural networks. The neurons in a convolutional neural network are only connected to a small subset of the neurons in the previous layer, creating a receptive field and this makes these networks work well on most visual tasks. A number of successful convolutional neural networks based on spiking neurons have been reported [11, 12, 13, 14], with varying degrees of biological plausibility and efficiency.

Figure 1.1: Elon Musk, the CEO of Tesla, Inc., on the need for more efficient solutions for machine learning, e.g. neural networks.

Still, while spiking neural networks have been applied successfully to solve image-recognition tasks, many deep learning algorithms use recurrent neural networks. Recurrent neural networks have units that send there output back to their own input, while the units in a feedforward neural network like a convolutional neural network send their output only to the next layers. This looping behaviour of the network gives it the ability to remember information. Although most types of recurrent neural networks are hard to train due to the vanishing gradient problem [15], Long Short-Term Memory (LSTM) [16] networks have been shown to work very well. Compared to convolutional neural networks, LSTMs use memory cells to store selected information and various gates to direct the flow of information in and out of the memory cells. To the best of our knowledge, no spiking neural network implementation of LSTM has been published.

Most neural networks are trained by using backpropagation to find the derivatives of the error with respect to the weights and then use gradient descent to update these weights. Since this method is very successful for analog neural networks, it is a promising way to train spiking neural networks in the same way; however, spiking neurons send brief spikes, which are not differentiable. One way to overcome this is to use analog neurons during training with activation functions that approximate the behaviour of the spiking neurons [14, 13, 17]. These analog neurons are then replaced by spiking neurons after training, such that the (more efficient) spiking neurons can be used for inference.

Related work

Research in recent years has drawn parallels between the Leaky Integrate-and-Fire (LIF) neuron [18] and the Spike Response Model (SRM) neuron [19] and the Sigma Delta Modulation (SDM) scheme [20]. In the SDM scheme the encoder, which is the sending neuron, encodes an analog signal (its membrane potential) to a digital signal (spikes).The decoder, which is the receiving neuron, can be fed with the spikes to reconstruct an estimate of the analog signal. The density of the spikes indicates the amplitude of the encoded signal. In [21] a Sigma-Delta Quantized Network is introduced to process video data. Each layer of the convolutional neural network sends a discretized form of its change in activation to the next layer. This means that the amount of computation will scale with the amount of change in the activations, rather than the size of the network. A similar method is used in [22] where a Delta Network is used to optimize the computation of a recurrent neural network. Each neuron transmits its value only when the change in its activation exceeds a certain threshold. Both of these methods lower the computational cost, but no performance improvements have been shown on hardware like GPUs. This is because the main bottleneck for the performance of neural networks is the limited (access to) memory on the current hardware [23], so new hardware is needed to unlock the potential of these new techniques. The neurons used in these methods are time-agnostic, in contrast to traditional spiking neurons which have a sense of time.

Neuromorphic hardware implements a neural network directly on a chip in a very energy-efficient way. A spiking convolutional network on IBM's TrueNorth chip [24, 11] has been shown to achieve near state-of-the-art results on several datasets, while being extremely energy-efficient. The performance of the network was good, despite several network architecture limitations and the fact that the neuron activations were limited to the states 0 and 1. Neuromorphic hardware is well adapted to handle data from event-based sensors, such as the Dynamic Vision Sensor [25]. Event-based sensors asynchronously send output when there is a change in the input, instead of regular sensors that send out output at a fixed rate.

A successful spiking convolutional neural network is presented in [17]. It uses the Adaptive Spiking Neuron model [26], which is biologically plausible and quickly adapts to the dynamic range of the input signal. Although the spiking neural network gets near deep learning state-of-the-art performance and is relatively biologically plausible, it uses spikes with analog values. This

is computationally a potential disadvantage compared to models that use binary spikes, since an extra float multiplication is needed. To our knowledge, there is no spiking LSTM model publicly available. [11] mentions results from an LSTM that is implemented on the neuromorphic TrueNorth chip, but no spiking LSTM model is presented.

LSTM is not the only successful model to incorporate memory in a neural network. The architecture and performance of Gated Recurrent Units (GRUs) and Recurrent Additive Networks (RANs) is similar to LSTMs, but they have less parameters. Attention-Gated MEmory Tagging (AuGMEnT) [27] is a more biologically plausible neural reinforcement learning model with working memory units, but is limited by the type of tasks that it can perform. This is due to the fact that it only looks at changes in input and cannot control the flow of information by using gates. AuGMEnT can for example not successfully work on tasks with noisy input, which makes it near impossible to learn complicated tasks like speech recognition.

Our work

In this work we demonstrate a gated recurrent spiking neural network that corresponds to an LSTM unit with a memory cell and an input gate. Analogous to recent work on spiking neural networks [14, 13, 17], we first train a network with modified LSTM units that computes with analog values, and show how this LSTM-network can be converted to a spiking neural network using adaptive spiking neurons that encode and decode information in spikes using a form of sigma-delta coding [20, 17, 21]. In particular, we develop a binary version of the adaptive sigma-delta coding proposed in [17]: we derive the shape of the transfer function that this model of fast-adapting spiking neurons exhibits, and we assemble the analog LSTM units using just this transfer function. The fact that the adjusted ASN sends binary spikes (spikes with a fixed height) instead of analog spikes is a big advantage since it makes it more biologically plausible and it is computationally less expensive. Since input-gating is essential for maintaining memorized information without interference from unrelated sensory inputs [16], and to reduce complexity, we model a limited LSTM neuron consisting of an input cell, input gating cell, a Constant Error Carousel (CEC) and output cell. The resultant analog LSTM network is then trained on a number of classical sequential tasks, such as the noiseless and noisy Sequence Prediction and the noiseless and noisy T-Maze

task [16, 28]. We demonstrate how nearly all the corresponding spiking LSTM neural networks correctly compute the same function as the analog version.

Note that the conversion of gated RNNs to spike-based computation implies a conversion of the neural network from a time-step based behavior to the continuous-time domain. For RNNs this means considering the continuous signal integration in the memory cell. We solve the time conversion problem by approximating the spiking memory cell behavior through time.

The focus of this work is not to show a more computational efficient LSTM, but this work is a first step towards using spiking neural networks in such diverse and challenging tasks like speech recognition and working memory cognitive tasks.

Chapter 2

Model

First a brief explanation about artificial neural networks, recurrent neural networks, spiking neural networks and Long Short-Term Memory is needed to be able to understand how to construct an Adapting Spiking LSTM network. We describe the Adaptive Spiking Neurons and analytically derive the corresponding activation function. We then show how an LSTM network can be constructed comprised of a spiking memory cell and a spike-driven input-gate, and we discuss how analog versions of this LSTM network are trained and converted to spiking versions.

Artificial neural networks

A traditional artificial neural network is a simplification of the brain, see Figure 2.1 for an example. The synapses are represented by a single number, a weight, although in reality the synapse has a complex behaviour that changes over time. The membrane potential of a neuron is replaced by a weighted sum of inputs from other neurons. The output of the neuron is generated by applying a non-linear function f, called the activation function, on the input value. This output value is a real-valued number and can be seen as the firing rate of the neuron. The firing rate is the number of spikes per time interval that reach the receiving neuron. A higher firing rate will raise the membrane potential of the receiving neuron more than a low firing rate. The relation between the membrane potential of a neuron and the firing rate of the neuron is described by the activation function.

The goal of the training phase is to minimize the error of the output. First, the network does a feed-forward pass with input x, in which all the neurons in the network are evaluated. In the case of supervised learning, the output y of the network is compared to the desired output $\bar{y}$ which results in an error E, for example by using a mean squared error or cross entropy error function. In the case of reinforcement learning [29] the neural network acts as the function approximator for the action-value function. An action is selected based on the output of the network and an error can be calculated.

The second training step is to calculate the derivative of the error E with respect to weight w_{ij} which connects neuron i to neuron j. We do this for all the weights in the network. Since the activation functions in the network are differentiable (except for maybe a finite number of points), a neural network can be seen as stacked differentiable functions. So the chain rule can be applied layer by layer to easily find the derivatives with respect to all the weights. This is called backpropagation. The last training step is to update the weights by using gradient descent. Each weight is updated in the direction that will minimize the error: $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$, where η is the learning rate.

Unlike traditional statistical techniques, where the goal is often to be able to map the training data correctly with as few free parameters as possible to prevent overfitting, neural networks use multiple orders of magnitude more parameters. It is remarkable that the huge number of free parameters in a neural network do not result in many overfitting problems. It is still unclear why this is the case and it is a very active area of research [30]. Also note that the brain has approximately 10^{15} synapses and that an average human lives for approximately 10^9 seconds. So it could be argued that the human brain has much more parameters than data [31].

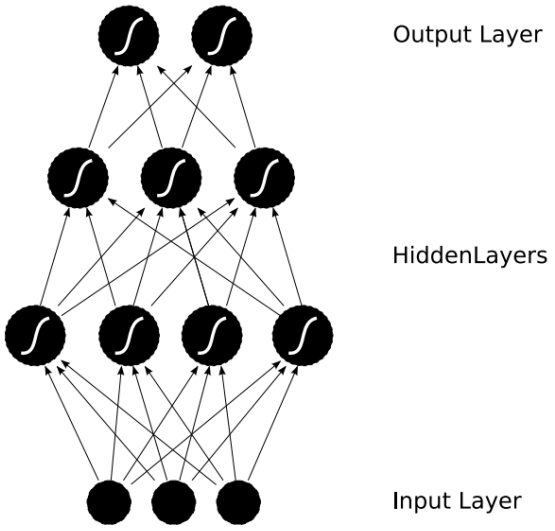

Figure 2.1: An artificial neural network with two hidden layers.

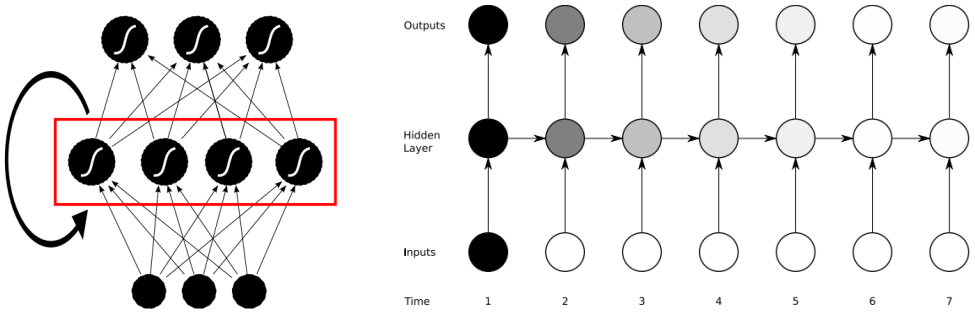

Figure 2.2: A recurrent neural network (left). The output of the hidden layer (red) is not only sent to the output layer but also to itself. A recurrent neural network can be visualized by unrolling it (right). In this case there is only one neuron in each layer. All the weights directed to the right are identical. The darker the shade of the node, the greater the sensitivity of the node to the input at time one. The sensitivity decays exponentially over time, resulting in the vanishing gradient problem.

Recurrent neural networks

Feedforward neural networks, which consist of only neurons that send their output to the next layers, have no time dimension. Recurrent neural networks, which consist of neurons that can send their output to the next layers, previous layers or the same layer, have a time dimension. This time dimension is discretized in fixed steps. A recurrent neural network has the ability to learn to remember information and use it at a later point in time. After enough training iterations and with the correct hyper-parameter settings, a recurrent neural network is able to link input at an earlier point in time with the desired output at a later point in time.

A recurrent neural network is the most general of all the traditional neural networks and feedforward neural networks are a subset of recurrent neural networks. By unrolling a recurrent neural network we can see it as a very deep feedforward neural network with layers with reoccurring weights, see Figure 2.2. This property makes a recurrent neural network very hard to train. The error at a later point in time needs to be backpropagated to an earlier point in time, so the derivative of the activation function of the same neuron needs to be applied for multiple time steps. This quickly leads to

a vanishing or exploding gradient of the error and makes training difficult, especially for tasks where something needs to be remembered for a long time.

Long Short-Term Memory (LSTM)

The vanishing gradient problem, see Figure 2.2, was a big obstacle for training recurrent neural networks. This problem was solved in 1997 by the introduction of the Long Short-Term Memory (LSTM) [16], which is well-suited to learn relations between events that are separated by time lags of unknown size. The central component of an LSTM is the Constant Error Carousel (CEC), this is the recurrent part of the LSTM. The CEC is a neuron with the identity function as its activation function, so its derivative will always be 1. This is why the gradient will not explode or vanish during backpropagation through time. To control the flow in and out of the CEC, an input and output gate are added. These gates are multiplied by the outputs of the input and output cell respectively. Often a forget gate is added to give the LSTM the ability to reset its CEC [32]. In newer versions peepholes are added to give the gates access to the state of the CEC [33]. The activation functions of the gates are usually sigmoid functions and the activation functions of the input and output cell are hyperbolic tangents.

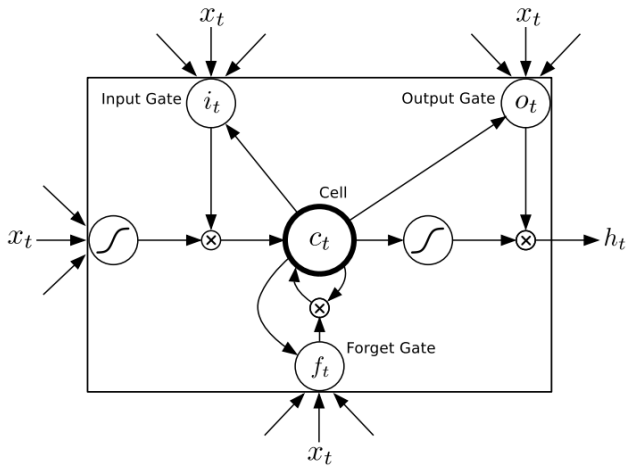

Figure 2.3: An LSTM cell, including the input, output and forget gates, the input and output cells, the Constant Error Carousel and the peephole connections.

Note that the operations in the LSTM are differentiable, so a network of LSTM cells can be trained by using backpropagation with gradient descent. An overview of an LSTM cell is shown in Figure 2.3.

Spiking neural networks

Spiking neurons are different from traditional artificial neurons, because they communicate by sending spikes. These spikes can have a variable analog value, as in [17], or a fixed binary value, as in this work. But the most important property of spiking neurons is the fact that they model the dynamics of the membrane potential, giving them the possibility to use time to encode information.

The computations in neural networks are mostly matrix-vector multiplications. During the inference phase, the weight matrix W_{lk}, containing the weights from layer k to layer l, is multiplied with the activation vector $\mathbf{y_k}$, which contains the output of the neurons in layer k. The result of $W_{lk}\mathbf{y_k}$ is the input vector to layer l, $\mathbf{x_l}$. In analog neural networks W_{lk} and $\mathbf{y_k}$ are both dense, but in spiking neural networks the majority of the neurons does not spike, especially when the the input to the neurons does not change, resulting in a sparse activation vector $\mathbf{y_k}$. The non-zero values in $\mathbf{y_k}$ are floats in [17], but are 1 in this work. So the matrix-vector multiplication $W_{lk}\mathbf{y_k}$ can be simplified to a summation: $x_{l_j} = \sum_{i \in \{i | y_{k_i}=1\}} W_{lk_{j,i}}$. A spiking neuron often needs to send multiple spikes to be able to communicate the same amount of information as an analog neural network, but it can still be more efficient if the input of the neuron does not change quickly [17].

Adaptive Spiking Neuron

Most standard spiking neuron models are described by a differential equation, like the Hodgkin-Huxley model [34] and the Izhikievic model [35]. A completely different class of spiking neuron models is based on filters, the most well-known of these formulations is the Spike Response Model [36]. The spiking neurons that are used in this thesis are also based on filters and are called Adaptive Spiking Neurons (ASNs), as described in [26]. This is a variant of an adapting Leaky Integrate & Fire (LIF) neuron model that includes fast adaptation to the dynamic range of input signals. The ASNs used here communicate with spikes of a fixed height h, in contrast to [17]

where ASNs use spikes with a variable height. The behavior of the ASN is determined by the following equations:

$$\text{incoming postsyn. current:} \quad I(t) = \sum_i \sum_{t_s^i} w_i \exp\left(\frac{t_s^i - t}{\tau_\beta}\right), \tag{2.1}$$

$$\text{input signal:} \quad S(t) = (\phi * I)(t), \tag{2.2}$$

$$\text{threshold:} \quad \vartheta(t) = \vartheta_0 + \sum_{t_s} m_f \vartheta(t_s) \exp\left(\frac{t_s - t}{\tau_\gamma}\right), \tag{2.3}$$

$$\text{internal state:} \quad \hat{S}(t) = \sum_{t_s} \vartheta(t_s) \exp\left(\frac{t_s - t}{\tau_\eta}\right), \tag{2.4}$$

where w_i is the weight (synaptic strength) of the neuron's incoming connection; $t_s^i < t$ denote the spike times of neuron i, and $t_s < t$ denote the spike times of the neuron itself; $\phi(t)$ is an exponential smoothing filter with a short time constant τ_ϕ; ϑ_0 is the resting threshold; m_f is a variable controlling the speed of spike-rate adaptation; $\tau_\beta, \tau_\gamma, \tau_\eta$ are the time constants that determine the rate of decay of $I(t), \vartheta(t)$ and $\hat{S}(t)$ respectively. The neuron emits a spike when $S(t) - \hat{S}(t) > \frac{\vartheta(t)}{2}$. See [26] and [17] for more details.

Activation function of the Adaptive Analog Neuron

In order to create a network of ASNs that performs correctly on typical LSTM tasks, our approach is to train a network of Adaptive Analog Neurons (AAN) and then convert the resulting analog network into a spiking one, similar to [14, 13, 17]. We define the activation function of the AANs as the function that maps the input signal S to the average PSC I that is perceived by the *next* (receiving) ASN. By setting $\tau_\eta = \tau_\beta = \tau_\gamma$ we can obtain the exact activation function as:

$$\text{AAN}(S) = \frac{-h}{\ln\left(\frac{2 \cdot S - \vartheta_0}{2 \cdot S(m_f + 1) + \vartheta_0}\right)} \text{ for } S > \frac{\vartheta_0}{2}, \text{ 0 otherwise.} \tag{2.5}$$

Using this mapping from the AAN to the ASN (see Figure 2.4), the activation function can be used during training: thereafter, the ASNs are used as "drop in" replacements for the AANs in the trained network; this activation function is derived in Chapter 3. Unless otherwise stated, the

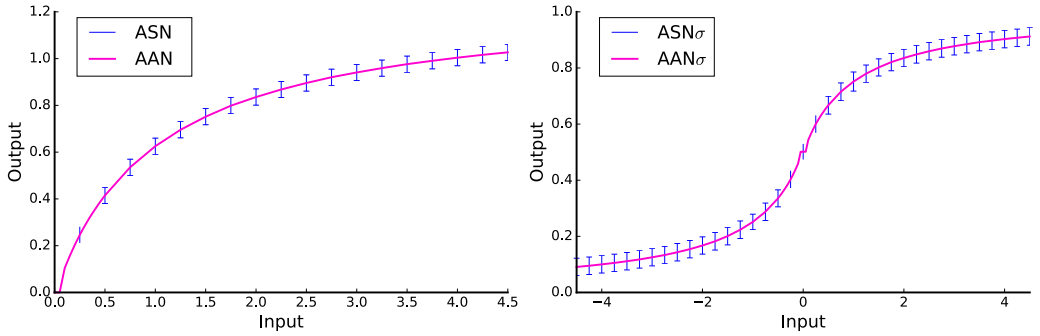

Figure 2.4: The average output signal of the ASN (left) as a function of its incoming PSC I, where the error bars indicate the standard deviation of the spiking simulation, and the corresponding AAN curve. The shape of the ASN curve is well described by the AAN activation function, Equation 2.5, so the two curves overlap. The ASN_σ (right) can be constructed by combining three ASNs and its shape is described by AAN_σ.

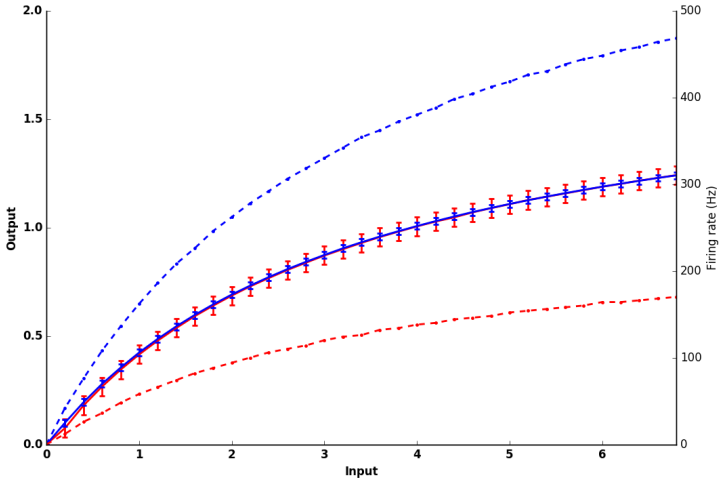

Figure 2.5: The solid lines correspond to the mean and standard deviation of the activation function of an ASN with $m_f = 0.1$, $\vartheta_0 = 0.1$ (blue) and an ASN with $m_f = 0.3$, $\vartheta_0 = 0.3$ (red), both with $\tau_\eta = 50$ and $\tau_\gamma = 20$. The values correspond with Equation 2.6.The dashed lines show their respective firing rates. A lower firing rate corresponds to a larger uncertainty in the signal.

AAN and ASN use $\tau_\eta = \tau_\beta = \tau_\gamma = 10$ms and a spike height, h, such that AAN(4) = ASN(4) = 1. ϑ_0 and m_f are set to 0.1 for all neurons. Note that using this formulation of the activation function allows us to communicate binary spikes only.

For some applications we might need a more general ASN with $\tau_\eta = \tau_\beta \neq \tau_\gamma$. The AAN of this ASN is approximated in Chapter 3, see Equation 3.6, by using a Taylor expansion and is:

$$f(S) = \frac{h}{\exp\left(\frac{2m_f\tau_\gamma^2 S + 2\vartheta_0\tau_\eta\tau_\gamma}{\tau_\gamma(m_f\tau_\gamma + 2(m_f+1)\tau_\eta)S + \vartheta_0\tau_\gamma\tau_\eta + \vartheta_0\tau_\eta^2}\right) - 1} - c, \tag{2.6}$$

for $S > \frac{\vartheta_0}{2}$ and $f(S) = 0$ for $S <= \frac{\vartheta_0}{2}$, with c a constant.

By slightly increasing m_f and ϑ_0 we can lower the firing rate of the neuron, but it also increases the variance (see figure 2.5). A low firing rate decreases the computational costs of the network, but increases the noise.

Adaptive Spiking LSTM

An LSTM cell usually consists of an input and output gate, an input and output cell and a CEC [16]. Deviating from the original formulation, and more recent versions where forget gates and peepholes were added [37], the Adaptive Spiking LSTM as we present it here only consists of an input gate, input and output cell, and a CEC. As noted, to obtain a working Adaptive Spiking LSTM, we first train its analog equivalent, the Adaptive Analog LSTM. Figure 2.6 shows the schematic of the Adaptive Analog LSTM and its spiking analogue. It is important to note that we aim for a one-on-one mapping from the Adaptive Analog LSTM to the Adaptive Spiking LSTM. This means that the Adaptive Analog LSTM is trained with the same time representation (time steps used for each input) that is used during the inference phase of the Adaptive Spiking LSTM.

Sigmoidal ASN

The original formulation of LSTM uses sigmoidal activation functions in the input gate and input cell that cannot easily be replaced by the activation function of the AAN, as the absence of a gradient for negative input in

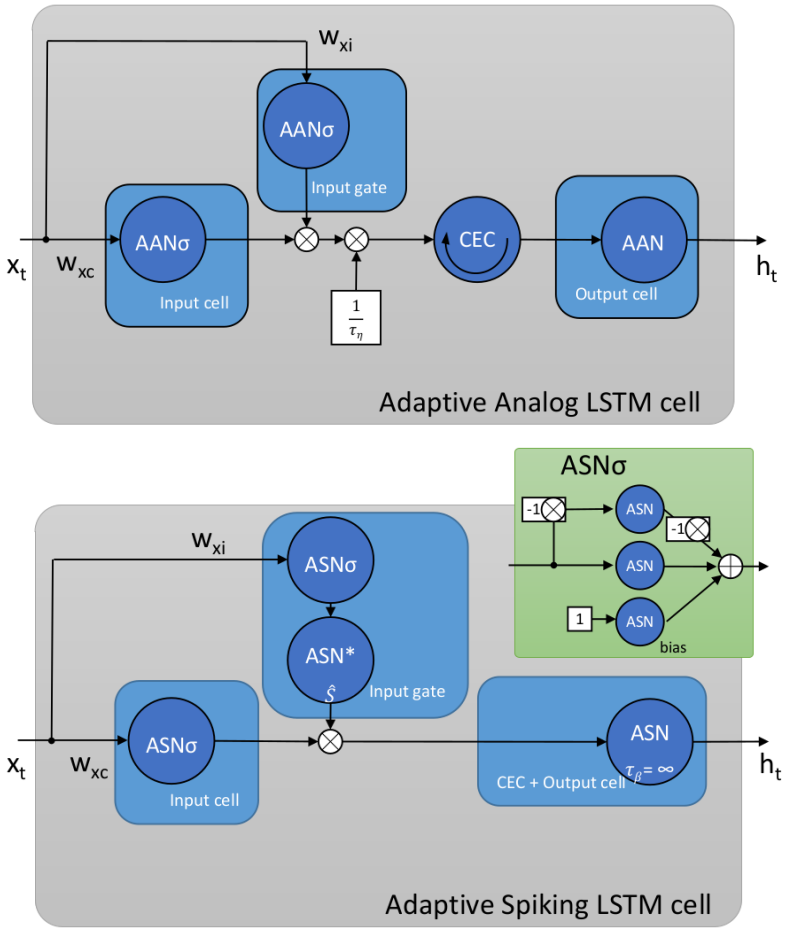

Figure 2.6: An overview of the construction of an Adaptive Analog LSTM (top) and an Adaptive Spiking LSTM cell (bottom). This compares to an LSTM with only an input gate. The green inset describes the construction of ASN_σ.

the AAN is problematic during training. We constructed a sigmoid-like activation function AAN_σ by combining three AANs (see the inset in Figure 2.6). First, we create an AAN_{tanh} unit, which is effectively a hyperbolic tangent-like activation function, by using two identical AANs, AAN_1 and AAN_2 in parallel; the AAN_1 receives input with weight w, while AAN_2 receives the same input but with input weight $-w$. Then the output signal of AAN_2 is multiplied by a fixed weight of -1 and added to the output signal of AAN_1.

The spike height h of AAN_1 and AAN_2 is set to $\ln\left(\frac{1}{m_f+1.}\right)$ to ensure that the positive and negative limits of $\text{AAN}_{\tanh}(S)$ are 1 and -1. Finally, to create AAN_σ the spike height h of AAN_1 and AAN_2 is halved and a bias neuron AAN_b is added. The AAN_b unit receives a constant input of 1 and its spike height h is set such that $\text{AAN}_b(1) = \frac{1}{2}$. A spiking sigmoidal neuron ASN_σ is then created simply by replacing all the AANs in the AAN_σ by ASNs.

A complication to the construction of the AAN_σ function is that, since the AAN function is zero for inputs smaller than $\frac{1}{2}\vartheta_0$, the AAN_σ has a constant output 0.5 for inputs between $-\frac{1}{2}\vartheta_0$ and $\frac{1}{2}\vartheta_0$ and thus the derivative is zero. This is problematic during training; to counter this, we used an artificial derivative of value $\text{AAN}_\sigma\left(\frac{1}{2}\vartheta_0 + \epsilon\right)$ for inputs between $-\frac{1}{2}\vartheta_0 - \epsilon$ and $\frac{1}{2}\vartheta_0 + \epsilon$, with $\epsilon = 0.05$. The AAN and resultant AAN_σ functions are shown in Figure 2.4.

Spiking input gate and spiking input cell

The AAN_σ functions are used in the Adaptive Analog LSTM cell for the input gate and input cell. The activation value of the input cell is multiplied by the activation value of the input gate, before it enters the CEC, see Figure 2.6. In the spiking version of the input gate, the three outgoing signals from the ASN_σ are accumulated in an intermediate neuron (ASN^* in Figure 2.6). The internal state $\hat{S}$ of this neuron is then multiplied with the spikes that move from the ASN_σ of the input cell to the ASN of the output cell. This leads to a direct mapping from the Adaptive Analog LSTM to the Adaptive Spiking LSTM.

Spiking Constant Error Carousel (CEC) and spiking output cell

The Constant Error Carousel (CEC) is the central part of the LSTM cell and avoids the vanishing gradient problem [16]. In the Adaptive Spiking LSTM, we merge the CEC and the output cell to one ASN with an internal state that does not decay. The value of the CEC in the Adaptive Analog LSTM corresponds with state I of the ASN output cell in the Adaptive Spiking LSTM.

In the Adaptive Spiking LSTM, we set τ_β in Equation 2.1 to a very large

value for the CEC cell to obtain the integrating behavior of a CEC. Since no forget gate is implemented this results in a spiking CEC neuron that fully integrates its input. When τ_β is set to ∞, every incoming spike is added to a non-decaying PSC I. So if the state of the sending neuron (ASN_{in} in Figure 2.7) has a stable inter-spike interval (ISI), then I of the receiving neuron (ASN_{out}) is increased with incoming spike height h every ISI, so $\frac{h}{\text{ISI}}$ per time step. In Chapter 3, see Equation 3.5, we derive that for $\tau_\eta = \tau_\gamma$ we have $\frac{h}{\text{ISI}} = \frac{S}{\tau_\eta}$ with S the output value of ASN_{in}.

The same integrating behavior needs to be translated to the analog CEC. Since the CEC cell of the Adaptive Spiking LSTM integrates its input S every time step by $\frac{S}{\tau_\eta}$, we can map this to the CEC of the Adaptive Analog LSTM. The CEC of a traditional LSTM without a forget gate is updated every time step by $\text{CEC}(t) = \text{CEC}(t-1) + S$, with S its input value. The CEC of the Adaptive Analog LSTM is updated every time step by $\text{CEC}(t) = \text{CEC}(t-1) + \frac{S}{\tau_\eta}$. This is depicted in Figure 2.6 via a weight after the input gate with value $\frac{1}{\tau_\eta}$.

The left plot in Figure 2.7 shows that setting τ_β to ∞ for ASN_{out} in a spiking network results in the same behavior as using an analog CEC that integrates with $\text{CEC}(t) = \text{CEC}(t-1) + \frac{S}{\tau_\eta}$, since the slope of the analog CEC is indeed the same as the slope of the spiking CEC. However, the spiking CEC still has a higher value in the end than the analog CEC. This is caused by the adaptive behavior of the ASN and results in large errors especially when the input of the network changes quickly. The plot in the right of Figure 2.7 shows the output signal of AAN_{out} and ASN_{out}. The spiking output is larger than the analog output during the time that the CECs are still increasing, because of the adapting behavior of the ASN. The output of ASN_{out} stabilizes to the same value as AAN_{out} shortly after the CECs stop increasing.

The spike height h of the output cell is set to 0.3, instead of a standard value of 0.12. This was done to increase the influence of the LSTM cell in the network.

Learning rule used for training the spiking LSTM

As noted before, the spiking network is solely used for inference. However, since the spiking neurons need time to adapt to the magnitude of their input signal, we have to train the analog network with multiple time steps per input x. For all the experiments $\Delta T = 40$ time steps are used per input.

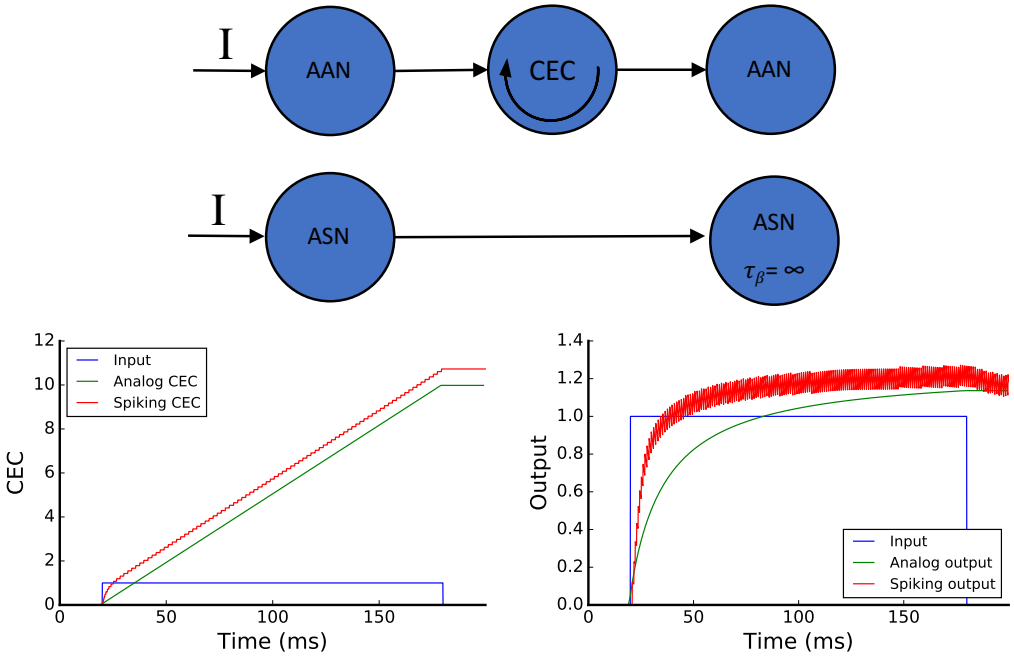

Figure 2.7: A simulation to illustrate how the analog CEC integrates its input signal with the same speed as an ASN with $\tau_\beta = \infty$ provided that the input signal does not change (left). As can be seen in the right figure, the spiking output signal deviates from the analog output signal during integration, but stabilizes to the same value as the analog output when the integration stops.

The equations that are used to update the Adaptive Analog LSTM in the inference phase are:

$$
\begin{aligned}
i_t &= \mathrm{AAN}_\sigma\left(W_{xi}\mathbf{x_t}\right), \\
c_t &= c_{t-1} + \frac{1}{\tau_\eta} i_t \mathrm{AAN}_\sigma\left(W_{xc}\mathbf{x_t}\right), \\
h_t &= \mathrm{AAN}\left(c_t\right),
\end{aligned}
$$

where i, c are respectively the input gate and CEC activations and h is the output of the Adaptive Analog LSTM.

To train the analog LSTMs on the supervised tasks, a customized truncated version of real-time recurrent learning (RTRL) was used. This is the same algorithm used in [37], where the partial derivatives w.r.t. the weights W_{xc} and W_{xi} (see Figure 2.6) are truncated. The derivatives are given by:

$$\frac{\partial c_t}{\partial w_{xc}} = \frac{\partial c_{t-1}}{\partial w_{xc}} + \mathrm{AAN}'_\sigma\left(W_{xc}\mathbf{x_t}\right) i_t x_t,$$
$$\frac{\partial c_t}{\partial w_{xi}} = \frac{\partial c_{t-1}}{\partial w_{xi}} + \mathrm{AAN}_\sigma\left(W_{xc}\mathbf{x_t}\right) \mathrm{AAN}'_\sigma\left(W_{xi}\mathbf{x_t}\right) x_t,$$

and the weights are updated as:

$$\Delta w_{xc}(t) = \alpha e_s(t) \frac{\partial c_t}{\partial w_{xc}},$$
$$\Delta w_{xi}(t) = \alpha e_s(t) \frac{\partial c_t}{\partial w_{xi}},$$

with:

$$e_s(t) = \frac{1}{\tau_\eta} \mathrm{AAN}'\left(c_t\right) e_c(t),$$

where $e_c(t)$ is the backpropagated error up to the LSTM output.

For the reinforcement learning (RL) tasks we used RL-LSTM [28], which uses the same customized, truncated version of RTRL that was used for the supervised tasks. RL-LSTM also incorporates eligibility traces to improve training and Advantage Learning [38]. All regular neurons in the network are trained with traditional backpropagation. Note that the extra $\frac{1}{\tau_\eta}$ weight after the input gate, in Figure 2.6, results in an extra $\frac{1}{\tau_\eta}$ term in the derivatives w.r.t. the weights W_{xc} and W_{xi}.

Let p be the number of inputs in a sequence that is presented to the network in each episode. All the weights are updated at every time step – so $\Delta T \times p$ times per episode – but the errors are only presented to the network at the end of each time step. Calculating the derivatives of all the weights in the network $\Delta T \times p$ times per episode is expensive, but we can heavily speed up the training since the presented architectures do not have a CEC that influences other CECs or itself. This means that we can multiply the partial derivatives w.r.t the weights W_{xc} and W_{xi} by ΔT and use only p forward passes, backward passes and weight updates per episode. This gives exactly the same results as using $\Delta T \times p$ updates per episode, but is much less expensive.

Chapter 3

Transfer functions

It is essential to have the AAN activation function that matches the behavior of the ASN if we want to transform a trained AAN network to a ASN network. We derive the AAN activation function; this is presented both for the case $\tau_\eta = \tau_\gamma$, which results in an exact derivation, and for the case $\tau_\eta \neq \tau_\gamma$, which results in an approximation for the activation function.

Exact derivation of the AAN activation function

We consider a spiking neuron with input S. This input S is constant over time and $\hat{S}(t)$ aims to approximate S using a threshold $\vartheta(t)$. Whenever $S - \hat{S}(t) > 0.5 \cdot \vartheta(t)$, the neuron emits a spike of fixed height h to the synapses connecting to target neurons, and a value of $\vartheta(t_f)$ is added to $\hat{S}$, with t_f the time of the spike. At the same time, the threshold is also increased with a value $m_f \vartheta(t_f)$. We assume that the synapse has a weight of 1. The post-synaptic current (PSC) in a target neuron is then given by $I(t)$, which is the signal that the receiving neuron will perceive. We are interested in deriving an activation function that maps the input signal S to the PSC I of the target neuron.

Since the variables of the Adaptive Spiking Neuron decay exponentially, they converge asymptotically. For a given fixed size current injection, we consider a neuron that has stabilized around an equilibrium, meaning that $\hat{S}(t)$ and $\vartheta(t)$ at the time of a spike always reach the same values. Let these values be denoted as $\hat{S}_l$ and ϑ_l respectively. We see $\vartheta(t_f) = \vartheta_l$ and $\hat{S}(t_f) = \hat{S}_l$ for all t_f. The PSC $I(t)$ also always declines to the same value, I_l, before it receives a new spike. So if we set $t = 0$ for the last time that there was a spike, we can rewrite our ASN equations, Equations 2.1, 2.2, 2.3 and 2.4, for $\tau_\beta = \tau_\eta$ and $0 < t < t_f$ to:

$$\hat{S}(t) = \hat{S}_l e^{-\frac{t}{\tau_\eta}} + \vartheta_l e^{-\frac{t}{\tau_\eta}},$$
$$\vartheta(t) = \vartheta_0 + (\vartheta_l - \vartheta_0)e^{-\frac{t}{\tau_\gamma}} + m_f \vartheta_l e^{-\frac{t}{\tau_\gamma}},$$
$$I(t) = I_l e^{-\frac{t}{\tau_\eta}} + h e^{-\frac{t}{\tau_\eta}}.$$

Here, we derive the activation function f of our spiking neuron model.

This activation function f is a function of S; f will be a bit larger than I_l since that is the lowest value of $I(t)$, and we are interested in the average value of $I(t)$ between two spikes: $f(S) = I_{average}$.

Since we are in a stable situation, the time between each spike is fixed; we define this time as t_e. So if the last spike occurred at $t = 0$, the next spike should happen at $t = t_e$. This implies that $\hat{S}(t), \vartheta(t)$ and $I(t)$ at $t = t_e$ must have reached their minimal values $\hat{S}_l, \vartheta_l$ and I_l respectively.

To obtain the activation function f, we solve the following set of equations:

$$\hat{S}(t_e) = \hat{S}_l,$$
$$\vartheta(t_e) = \vartheta_l,$$
$$I(t_e) = I_l,$$

and by noting that the neuron only spikes when $S - \hat{S}(t) > 0.5 \cdot \vartheta$, we also have:

$$S - \hat{S}_l = 0.5 \cdot \vartheta_l.$$

We first notice:

$$\frac{h e^{-\frac{t_e}{\tau_\eta}}}{1 - e^{-\frac{t_e}{\tau_\eta}}} = I_l. \tag{3.1}$$

We now want an expression for ϑ_l:

$$\vartheta_0 + (\vartheta_l - \vartheta_0)e^{-\frac{t_e}{\tau_\gamma}} + m_f \vartheta_l e^{-\frac{t_e}{\tau_\gamma}} = \vartheta_l,$$
$$\vartheta_0 - \vartheta_0 e^{-\frac{t_e}{\tau_\gamma}} = \vartheta_l - m_f \vartheta_l e^{-\frac{t_e}{\tau_\gamma}} - \vartheta_l e^{-\frac{t_e}{\tau_\gamma}}.$$

We can rewrite this to:

$$\vartheta_0 \frac{1 - e^{-\frac{t_e}{\tau_\gamma}}}{1 - (m_f + 1)e^{-\frac{t_e}{\tau_\gamma}}} = \vartheta_l. \tag{3.2}$$

Using equations $S - \hat{S}_l = 0.5 \cdot \vartheta_l$ and $\hat{S}(t_e) = \hat{S}_l$, we get:

$$\begin{aligned}
(S - 0.5 \cdot \vartheta_l)e^{-\frac{t_e}{\tau_\eta}} + \vartheta_l e^{-\frac{t_e}{\tau_\eta}} &= S - 0.5 \cdot \vartheta_l, \\
e^{-\frac{t_e}{\tau_\eta}}(2 \cdot S + \vartheta_l) &= 2 \cdot S - \vartheta_l.
\end{aligned}$$

Inserting Equation 3.2 gives:

$$e^{-\frac{t_e}{\tau_\eta}} \left(2 \cdot S + \vartheta_0 \frac{1 - e^{-\frac{t_e}{\tau_\gamma}}}{1 - (m_f + 1)e^{-\frac{t_e}{\tau_\gamma}}} \right) = 2 \cdot S - \vartheta_0 \frac{1 - e^{-\frac{t_e}{\tau_\gamma}}}{1 - (m_f + 1)e^{-\frac{t_e}{\tau_\gamma}}}.$$

This can be rewritten to:

$$\begin{aligned}
e^{-\frac{t_e}{\tau_\eta}} \left(2 \cdot S(1 - (m_f + 1)e^{-\frac{t_e}{\tau_\gamma}}) + \vartheta_0(1 - e^{-\frac{t_e}{\tau_\gamma}}) \right) &= 2 \cdot S(1 - (m_f + 1)e^{-\frac{t_e}{\tau_\gamma}}) \\
&\quad - \vartheta_0(1 - e^{-\frac{t_e}{\tau_\gamma}}), \\
(2 \cdot S + \vartheta_0)e^{-\frac{t_e}{\tau_\eta}} - (2 \cdot S(m_f + 1) + \vartheta_0)e^{-\frac{t_e}{\tau_\gamma}} e^{-\frac{t_e}{\tau_\eta}} &= 2 \cdot S - \vartheta_0 \\
&\quad - 2 \cdot S(m_f + 1)e^{-\frac{t_e}{\tau_\gamma}} \\
&\quad + \vartheta_0 e^{-\frac{t_e}{\tau_\gamma}}, \\
(2 \cdot S + \vartheta_0)e^{-\frac{t_e}{\tau_\eta}} - (2 \cdot S(m_f + 1) + \vartheta_0)e^{-t_e(\frac{1}{\tau_\gamma} + \frac{1}{\tau_\eta})} &+ (2 \cdot S(m_f + 1) - \vartheta_0)e^{-\frac{t_e}{\tau_\gamma}} \\
&= 2 \cdot S - \vartheta_0.
\end{aligned} \tag{3.3}$$

We are interested in an analytic solution for t_e: by setting $\tau_\eta = \tau_\gamma$ this last equation reduces to a quadratic polynomial with variable $x = e^{-\frac{t_e}{\tau_\eta}}$. Note that this last equation also has an exact solution when we set $\tau_\eta = 2\tau_\gamma$ or $\tau_\eta = 3\tau_\gamma$ or vice versa. This would result in a polynomial of degree three or four, which can also be solved exactly. Other combinations of τ_η and τ_γ can be computed by using a Taylor series expansion as approximation, this is done in the next section. Here, we set $\tau_\eta = \tau_\gamma$ and the quadratic polynomial is then solved by using the quadratic formula for $x = e^{-\frac{t_e}{\tau_\eta}}$. This results in either $x = 1$ so $t_e = 0$ or:

$$e^{-\frac{t_e}{\tau_\eta}} = \frac{2 \cdot S - \vartheta_0}{2 \cdot S(m_f + 1) + \vartheta_0}. \tag{3.4}$$

The desired activation function $f(S)$ should be equal to the average value of $I(t)$ between two spikes. Let the last spike with spike height h come in at $t = 0$ and the next one will come in at t_e. $I(t)$ decays exponentially with time constant τ_η. Then we can say that $I(t)$ is described by $(I_l + h) \cdot e^{-\frac{t}{\tau_\eta}}$ between two spikes at $t = 0$ and $t = t_e$. So this gives us the desired activation function:

$$\begin{aligned} f(S) &= \int_0^{t_e} (I_l + h) \cdot e^{-\frac{t}{\tau_\eta}} \mathrm{dt} \cdot \frac{1}{t_e}, \\ &= -\tau_\eta \, (I_l + h) \cdot (e^{-\frac{t_e}{\tau_\eta}} - 1) \cdot \frac{1}{t_e}, \\ &= \frac{h \cdot \tau_\eta}{t_e}, \end{aligned} \tag{3.5}$$

where the last equality comes from inserting Equation 3.1. By using Equation 3.4 we get our final activation function:

$$f(S) = \frac{-h}{\ln\left(\frac{2 \cdot S - \vartheta_0}{2 \cdot S(m_f+1)+\vartheta_0}\right)}.$$

Approximation of the AAN activation function

Instead of choosing $\tau_\eta = \tau_\gamma$ in Equation 3.3 to get an analytic solution for t_e, we can also use a (second order) Taylor series expansion to approximate the exponential function:

$$e^x \approx 1 + x + \frac{x^2}{2},$$

for x close to 0. We can use this in our previous equation:

$$\begin{aligned} (2 \cdot S + \vartheta_0)(1 - \frac{1}{\tau_\eta} t_e + \frac{1}{2\tau_\eta^2} t_e^2) \\ - (2 \cdot S(m_f + 1) + \vartheta_0)(1 - (\frac{1}{\tau_\gamma} + \frac{1}{\tau_\eta}) t_e + \frac{1}{2}(\frac{1}{\tau_\gamma} + \frac{1}{\tau_\eta})^2 t_e^2) \\ + (2 \cdot S(m_f + 1) - \vartheta_0)(1 - \frac{1}{\tau_\gamma} t_e + \frac{1}{2\tau_\gamma^2} t_e^2) \\ = 2 \cdot S - \vartheta_0. \end{aligned}$$

We need a few steps to isolate t_e:

$$
\begin{aligned}
&(2\cdot S+\vartheta_0)(-\frac{1}{\tau_\eta}t_e+\frac{1}{2\tau_\eta^2}t_e^2)\\
&\qquad-(2\cdot S(m_f+1)+\vartheta_0)(-(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta})t_e+\frac{1}{2}(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta})^2t_e^2)\\
&\qquad+(2\cdot S(m_f+1)-\vartheta_0)(-\frac{1}{\tau_\gamma}t_e+\frac{1}{2\tau_\gamma^2}t_e^2)=0,\\
&(2\cdot S+\vartheta_0)(-\frac{1}{\tau_\eta}+\frac{1}{2\tau_\eta^2}t_e)\\
&\qquad-(2\cdot S(m_f+1)+\vartheta_0)(-(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta})+\frac{1}{2}(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta})^2t_e)\\
&\qquad+(2\cdot S(m_f+1)-\vartheta_0)(-\frac{1}{\tau_\gamma}+\frac{1}{2\tau_\gamma^2}t_e)=0,
\end{aligned}
$$

$$
\begin{aligned}
&((2\cdot S+\vartheta_0)\frac{1}{2\tau_\eta^2}-(2\cdot S(m_f+1)+\vartheta_0)\frac{1}{2}(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta})^2\\
&\qquad+(2\cdot S(m_f+1)-\vartheta_0)\frac{1}{2\tau_\gamma^2})t_e\\
&\qquad=(2\cdot S+\vartheta_0)\frac{1}{\tau_\eta}-(2\cdot S(m_f+1)+\vartheta_0)(\frac{1}{\tau_\gamma}+\frac{1}{\tau_\eta})\\
&\qquad+(2\cdot S(m_f+1)-\vartheta_0)\frac{1}{\tau_\gamma},
\end{aligned}
$$

$$
\begin{aligned}
(-S\cdot m_f\frac{1}{\tau_\eta^2}-(2\cdot S(m_f+1)+\vartheta_0)\frac{1}{\tau_\gamma\tau_\eta}-\vartheta_0\frac{1}{\tau_\gamma^2})t_e\\
=-2\cdot S\cdot m_f\frac{1}{\tau_\eta}-2\vartheta_0\frac{1}{\tau_\gamma},\\
(-S\cdot m_f\tau_\gamma^2-(2\cdot S(m_f+1)+\vartheta_0)\tau_\gamma\tau_\eta-\vartheta_0\tau_\eta^2)t_e\\
=-2\cdot S\cdot m_f\tau_\gamma^2\tau_\eta-2\vartheta_0\tau_\gamma\tau_\eta^2.
\end{aligned}
$$

This leads to our expression for t_e :

$$
t_e=\frac{2\tau_\gamma\tau_\eta(S\cdot m_f\tau_\gamma+\vartheta_0\tau_\eta)}{S\cdot\tau_\gamma(m_f\tau_\gamma+2(m_f+1)\tau_\eta)+\vartheta_0\tau_\gamma\tau_\eta+\vartheta_0\tau_\eta^2}.
$$

We now insert this expression in Equation 3.1 and get:

$$I_l(S) = \frac{h}{e^{\frac{t_e}{\tau_\eta}} - 1} = \frac{h}{\exp\left(\frac{2\tau_\gamma(S \cdot m_f\tau_\gamma + \vartheta_0\tau_\eta)}{S\cdot\tau_\gamma(m_f\tau_\gamma + 2(m_f+1)\tau_\eta) + \vartheta_0\tau_\gamma\tau_\eta + \vartheta_0\tau_\eta^2}\right) - 1}.$$

To make sure that our activation function $f(S)$ is 0 at $S = \frac{\vartheta_0}{2}$ we choose our activation function to be:

$$f(S) = I_l(S) - I_l(\frac{\vartheta_0}{2}) = \frac{h}{\exp\left(\frac{2m_f\tau_\gamma^2 S + 2\vartheta_0\tau_\eta\tau_\gamma}{\tau_\gamma(m_f\tau_\gamma + 2(m_f+1)\tau_\eta)S + \vartheta_0\tau_\gamma\tau_\eta + \vartheta_0\tau_\eta^2}\right) - 1} - c, \tag{3.6}$$

for $S > \frac{\vartheta_0}{2}$ and $f(S) = 0$ for $S <= \frac{\vartheta_0}{2}$ with $c = I_l(\frac{\vartheta_0}{2})$.

Chapter 4

Experiments and results

Since the presented Adaptive Analog LSTM only has an input gate and no output or forget gate, we present four classical tasks from the LSTM literature that do not rely on these additional gates. The next step is to train the AAN networks and transform these to ASN networks. We present the results for the noiseless and noisy sequence prediction tasks and the noiseless and noisy T-maze tasks. Finally, we present examples of networks that failed to transform successfully from analog to spiking.

Sequence Prediction with Long Time Lags

The main concept of LSTM, the ability of a CEC to maintain information over long stretches of time, was demonstrated in [16] in a Sequence Prediction task: the network has to predict the next input of a sequence of $p+1$ possible input symbols denoted as $a_1, ..., a_{p-1}, a_p = x, a_{p+1} = y$. In the *noise free* version of this task, every symbol is represented by the $p+1$ input units with the $i-th$ unit set to 1 and all the others to 0. At every time step a new input of the sequence is presented. As in the original formulation, we train the network with two possible sequences, $(x, a_1, a_2, ..., a_{p-1}, x)$ and $(y, a_1, a_2, ..., a_{p-1}, y)$, chosen with equal probability. For both sequences the network has to store a representation of the first element in the memory cell for the entire length of the sequence (p). We train 20 networks on this task for a total of $100k$ trials, with $p = 100$, on an architecture with $p+1$ input units and $p+1$ output units. The input units are fully connected to the output units without a hidden layer.

The same sequential network construction method from the original paper was used to prevent the 'abuse problem': the Adaptive Analog LSTM cell is only included in the network after the error stops decreasing [16]. This is needed to speed up the training phase. If the LSTM cell is inserted into the network from the start, the network will try to use the LSTM as a bias unit to reduce the error for parts of the tasks that do not require memory. This will push the weights of the LSTM far away from 0 at the beginning of the

training phase. After the network learned to complete the non-memory part of the task, it will take a long time for the weights of the LSTM to get close to 0 again.

The error function used for this task was the cross-entropy error:

$$O = -\sum_{(x,z)\in S}\sum_{i=1}^{p+1} z_i \ln y_i + (1 - z_i) \ln (1 - y_i) \,,$$

with x the input of the network, y_i the resulting prediction of output neuron i and z_i the target of output neuron i.

In the *noisy* version of the sequence prediction task, the network still has to predict the next input of the sequence, but the symbols from a_1 to a_{p-1} are presented in random order and the same symbol can occur multiple times. Therefore, only the final symbols a_p and a_{p+1} can be correctly predicted. This version of the sequence prediction task avoids the possibility that the network learns local regularities in the input stream. We train 20 networks with the same architecture and parameters of the previous task, but for $200k$ trials. For both the noise-free and the noisy task we considered the network converged when the average error over the last 100 trials was less than 0.38, this usually indicated that the maximum difference between each output neurons value and its target was less than 0.25.

T-Maze task

In order to demonstrate the generality of our approach, we trained a network with Adaptive Analog LSTM cells on a Reinforcement Learning task, originally introduced in [28]. In the T-Maze task, an agent has to move inside a maze to reach a target position in order to be rewarded while maintaining information during the trial. The maze is composed of a long corridor with a T-junction at the end, where the agent has to make a choice based on information presented at the start of the task. The agent receives a reward of 4 if it reaches the target position and -0.4 if it moves against the wall. If it moves to the wrong direction at the T-junction it also receives a reward of -0.4 and the system is reset. The larger negative reward value, w.r.t. the one used in [28], is chosen to encourage Q-values to differentiate more during the trial. This is needed to increase the differences in the Q-values to make sure that the noisier spiking neural network will still select the correct action. The agent has 3 inputs

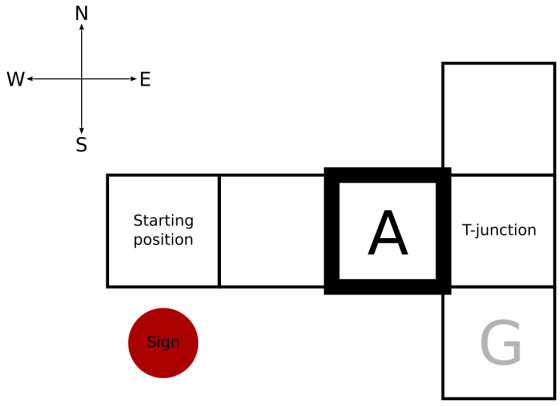

Figure 4.1: Overview of the T-maze task with corridor length 3. In this case the sign at the start of the trial is located on the south side, so the goal is also placed at the south side of the T-junction.

and 4 outputs corresponding to the 4 possible directions it can move to. At the beginning of the task the input can be either 011 or 110 (which indicates on which side of the T-junction the reward is placed). Here, we chose the corridor length $N = 20$.

A noiseless and a noisy version of the task were defined: in the noiseless version the corridor is represented as 101, and at the T-junction 010; in a noisy version the input in the corridor is represented as $a0b$ where a and b are two uniformly distributed random variables in a range of $[0, 1]$. While the noiseless version can be learned by LSTM-like networks without input gating [27], the noisy version requires the use of such gates. The network consists of a fully connected hidden layer with 12 AAN_{tanh} units and 3 Adaptive Analog LSTMs. The same training parameters are used as in [28], except for the exploration mechanism. While [28] uses a directed exploration mechanism where the agent is more likely to explore in the ambiguous parts of the task, we use the much simpler Max-Boltzmann controller [39]. It selects the greedy action (the action with the highest Q-value) with probability $1 - \epsilon$ and a random action k sampled from the Boltzmann distribution P_B with probability $\epsilon = 0.05$:

$$P_B(k) = \frac{\exp(Q_k)}{\sum_{k'} \exp(Q_{k'})}.$$

We train 10 networks for each task and all networks have the same architecture.

As a convergence criteria we checked whenever the network reached on average a total reward greater than 3.5 in the last 100 trials.

Experimental setup

The analog neural networks, the spiking neural networks and the tasks were built in Python from the ground up. No deep learning libraries like TensorFlow were used, because full control over the LSTM elements and the translation to spiking neurons was needed. The Numpy library was used for linear algebra operations. Pandas and matplotlib were used for analyzing and plotting the results. The neural networks were trained on Cartesius, the Dutch national supercomputer, to be able to train multiple networks in parallel. The performance of the spiking neural networks was checked on a regular desktop.

Experimental results

As shown in Table 4.1, almost all of the networks that were successfully trained for the noise-free and noisy Sequence Prediction tasks could be converted into spiking networks. Figure 4.2 shows the last 7 inputs of a noise-free Sequence Prediction task before (left) and after (right) the conversion, demonstrating the correct predictions made in both cases. For the noise-free version, only

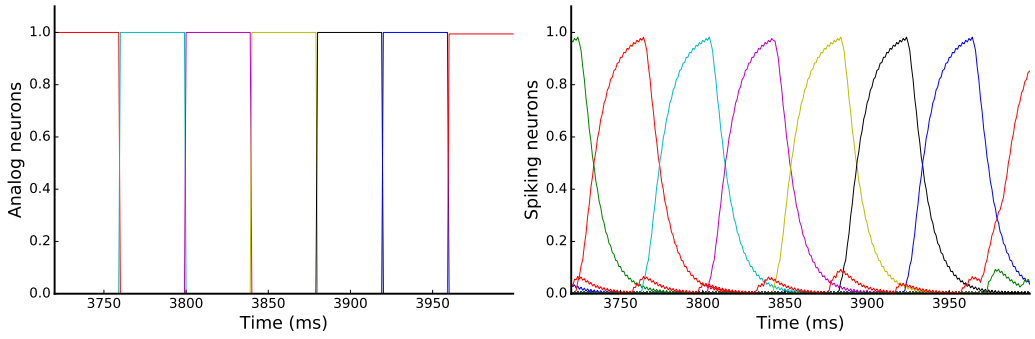

Figure 4.2: The output values of the analog (left) and spiking (right) network for the noise-free Sequence Prediction task. Only the last 280 time steps of the episode are shown. In the last 40ms it correctly predicts symbol y (red), instead of symbol x (green).

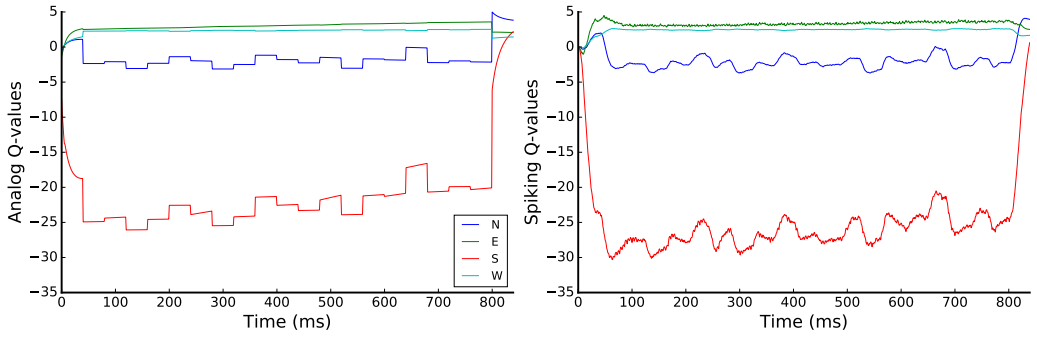

Figure 4.3: The Q-values of the analog (left) and spiking (right) network for the noisy T-Maze task. In the last 40ms both the analog and spiking networks select the correct action (north).

one network out of 20 did not perform correctly after the conversion. Indeed, for the 19 successful networks, after presenting either x or y as the first symbol of the sequence, the average difference in the value of the CECs was 3.47. It means that the CEC behaves differently when the x condition is presented w.r.t. the y condition: in one case it stores a high value and in the other a very low one. When the conversion failed, the CEC did not store values that were different enough. While such a small difference can be reliably expressed with a high-precision Adaptive Analog LSTM, the Adaptive Spiking LSTM will inevitably accumulate more noise, thus making the winner output difficult to recognize. Note that, in the noisy task, all the successfully trained networks were still working after the conversion: in this case, due to the input noise, the CEC values are always well separated. Finally, we found that the number of trials needed to reach the convergence criterion were, on average, higher than the one reported in [16].

Similar results were obtained for the T-Maze task, see Table 4.2. Only one network out of 10 failed the task after the conversion in the noise-free conditions, while all the networks were still working after the conversion in case of the noisy condition. Figure 4.3 shows the Q-values of a noisy T-Maze task, demonstrating the correspondence between the analog and spiking representation even in presence of noisy inputs.

In general, we see that the spiking CEC value is close to the analog CEC value, while always exhibiting some deviations. Most of this is the result from the Adaptive Analog LSTM slowly filling up during the course of the

	Seq. Prediction	noisy Seq. Prediction
[16] % Successful Trials	100	100
[16] Success After	5040	5680
AAN % Successful Trials	100	95
AAN Success After	8630	22358
ASN % Successful Trials	95	100
ASN Firing Rate	374.1 Hz	363.8 Hz

Table 4.1: Summary of the results for the Sequence Prediction tasks with $p = 100$. The 'Success After' values correspond to the number of episodes that were needed to converge during training. 'ASN % Successful Trials' is the percentage of the working analog neural networks that were successfully transformed to spiking neural networks. The firing rate is the average firing rate per neuron in the network.

	T-maze	noisy T-Maze
[28] % Successful Trials	100	100
[28] Success After	$1M$	$1.75M$
AAN % Successful Trials	100	100
AAN Success After	$1.29M$	$3.19M$
ASN % Successful Trials	90	100
ASN Firing Rate	349.2 Hz	359.2 Hz

Table 4.2: Summary of the results for the T-maze tasks with corridor length 20. The 'Success After' values correspond to the total number of steps that were needed to converge during training. 'ASN % Successful Trials' is the percentage of the working analog neural networks that were successfully transformed to spiking neural networks. The firing rate is the average firing rate per neuron in the network.

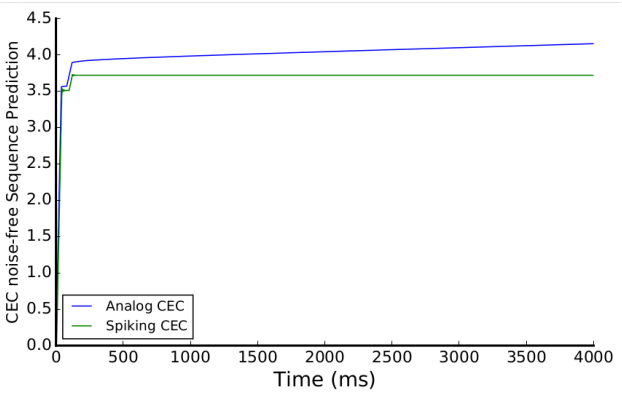

Figure 4.4: The values of the analog CECs and spiking CECs for the noise-free Sequence Prediction task. Only one CEC cell was used. The spiking CEC is the internal state $\hat{S}$ of the output cell of the Adaptive Spiking LSTM.

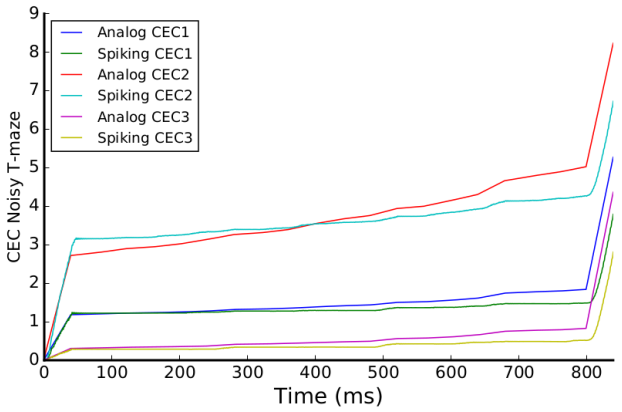

Figure 4.5: The values of the analog CECs and spiking CECs for the noisy T-maze task. Three CEC cells were used. The spiking CEC is the internal state $\hat{S}$ of the output cell of the Adaptive Spiking LSTM.

episode, which is caused by the AAN_σ. Although $\lim_{x\to\infty} \text{AAN}_\sigma(x) = 1$ and $\lim_{x\to-\infty} \text{AAN}_\sigma(x) = 0$ just like a sigmoid function, AAN_σ needs much higher input values to get close to 1 and much lower input values to get close to 0. The result is that the input gate and the input cell of the Adaptive Analog LSTM are essentially never fully closed, causing to significantly increase the CEC value if a high number of time steps are used per episode. This would

not be a problem if the Adaptive Spiking LSTM was able to exactly copy the behavior of its analog counterpart; however, Figure 4.4 shows that the spiking CEC does not increase at all when the analog CEC slowly increases. This is caused by the adapting behavior of the ASN, and thus the adapting behavior of ASN_σ. When a large negative value is inserted in ASN_σ the output will first be significantly smaller than the desired output. Some time is needed for ASN_σ to return to its intended state. The inputs in a task like the noise-free Sequence Prediction change so quickly (every 40ms a new input is presented), that the input gate and input cell of the Adaptive Spiking LSTM do not have time to adapt. This results in a completely closed input gate, since the value of the input gate is determined by internal state $\hat{S}$ which is 0 if the input is smaller than $\frac{\theta_0}{2}$. This is less of a problem for the T-maze task, since the same input is presented during the majority of the episode.

Failures

It is insightful to analyze the behavior of the neural networks that failed the tasks. Figure 4.6 shows the Q-values of the only neural network that was not successfully translated to a spiking neural network on the noiseless T-maze task with corridor length 20. Figure 4.7 shows the accompanying CEC values. The target is on the south side of the T-junction, but only the analog network takes the correct action in the end. This is caused by a seemingly small error in the second spiking CEC (green curve in Figure 4.7). This second CEC has a large negative weight connected to the north action and a large positive weight connected to the south action. A small error in this spiking CEC has a dramatic effect on the performance of the spiking neural network.

The Adaptive Analog and Adaptive Spiking LSTM networks performed well on a noiseless and noisy T-maze with a corridor of length 20. The maximum corridor length for which the LSTM network from [28] still had a maximum performance was 50 and 40, for respectively the noiseless and noisy T-maze task. The Adaptive Analog LSTM networks got 100% successful trials on the noiseless T-maze task with corridor length 50 and 90% successful trials on the noisy T-maze task with corridor length 40. But their spiking counterparts had 10% and 22.2% successful trials respectively. In Figure 4.8 you see the Q-values of a network on the noiseless T-maze task with corridor length 50 that works correctly in its analog version, but fails after it has been transformed to a spiking neural network. The spiking neural network takes

the correct actions until it is halfway the corridor, then it moves back to its starting position. Figure 4.9 shows the values of the CECs over time for these networks. The second and third CECs (green and red) increase much faster in the Adaptive Analog LSTMs than in the Adaptive Spiking LSTMs. The second CEC has a large positive weight connected to the 'east' Q-value, so the error in the spiking CEC lowers that Q-value. This results in failure of this task.

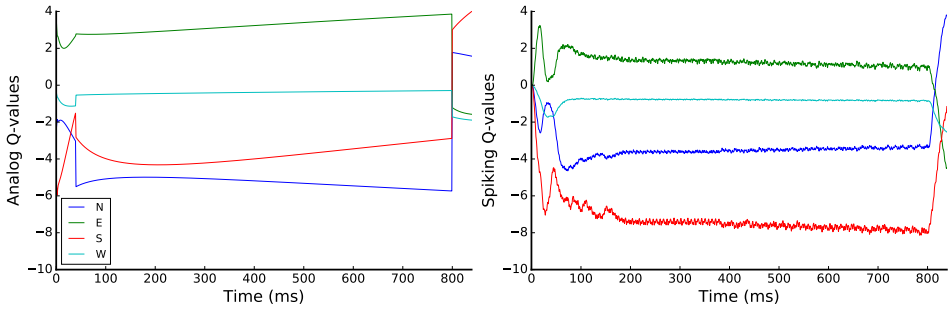

Figure 4.6: The Q-values of the analog (left) and spiking (right) network for the noiseless T-Maze task with corridor length 20. In the last 40ms the analog network selects the correct action (south), while the spiking network selects the wrong action (north).

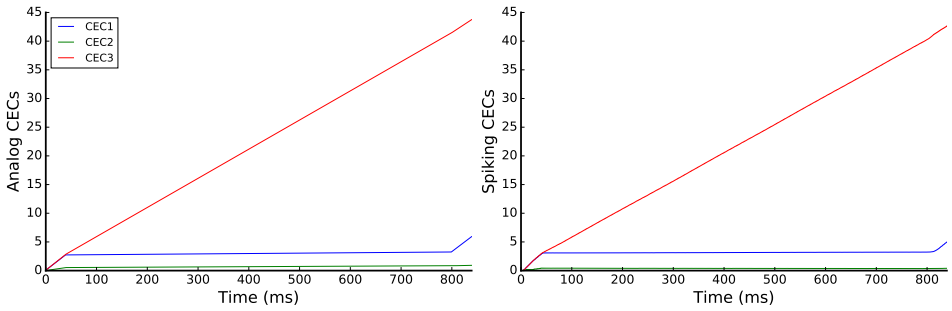

Figure 4.7: The three CEC values belonging to the analog and spiking networks with Q-values in Figure 4.6. The error in the second spiking CEC (green line) causes the network to fail the task.

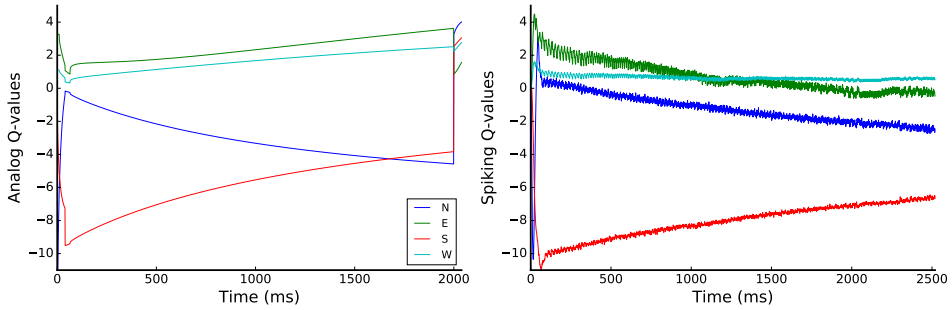

Figure 4.8: The Q-values of the analog (left) and spiking (right) network for the noiseless T-Maze task with corridor length 50. Approximately midway through the corridor, the spiking network starts selecting the wrong action (west instead of east). The spiking network never reaches the T-junction, while the analog network reaches the T-junction and finds the goal.

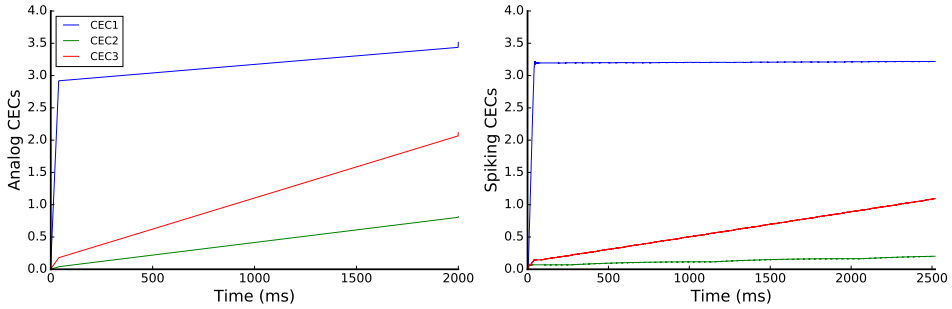

Figure 4.9: The three CEC values belonging to the analog and spiking networks with Q-values in Figure 4.8. The second spiking CEC (green line) does not increase fast enough during the course of the trial, causing the network to fail the task.

Chapter 5

Discussion

Gating is a crucial ingredient in recurrent neural networks that are able to learn long-range dependencies [16, 40]. Input gates in particular allow memory cells to maintain information over long stretches of time regardless of the presented - irrelevant - sensory input [16]. The ability to recognize and maintain information for later use is also that which makes gated RNNs like LSTM so successful in the great many sequence related problems, ranging from natural language processing to learning cognitive tasks [28].

To transfer deep neural networks to networks of spiking neurons, a highly effective method has been to map the transfer function of spiking neurons to analog counterparts and then, once the network has been trained, substitute the analog neurons with spiking neurons [14, 13, 17]. Here, we showed how this approach can be extended to gated memory units, and we demonstrated this for an LSTM network comprised of an input gate and a CEC.

The main contribution of this work is that we analytically derive the activation function of the ASN. This enabled us to transform the ASN to a spiking neuron with binary spikes, which, as many real neurons, approximates a half-sigmoid (Fig. 2.4). We showed how several spiking neurons together can approximate a full sigmoid function. This sigmoid function was needed since a differentiable gating function turned out to be a requirement for the gating mechanism of an LSTM, for which analog networks use sigmoidal units. The resultant LSTM network was then shown to be suitable for learning sequence prediction tasks, both in a noise-free and noisy setting, and a standard working memory reinforcement learning task, also both in a noise-free and noisy setting. The learned network could then successfully be mapped to its spiking neural network equivalent for at least 90% of the trained analog networks. So we successfully constructed a brain-like model – the Adaptive Spiking LSTM – capable of finding patterns in time-series.

Limitations and future improvements

The results in Chapter 3 and 4 are exciting, but it is worth mentioning that our work has some limitations. These limitations might become obstacles when we try our method on more difficult tasks, so we suggest some possible improvements and give a direction for future work.

For spiking neuron simulations with low time resolutions, or ASNs with small time constants, the behavior of the ASN is not well modelled by the AAN function. This is simply because the low time resolution of the simulation of the ASN will not result in the theoretically correct stable state of the ASN, which is given by the AAN function. These inaccuracies in the simulations were negligible if time steps of 0.5ms were used instead of time steps of 1ms. However, changing the time resolution of the spiking neural network simulations to 0.5ms did not change the results in Tables 4.1 and 4.2.

We also showed that some difficulties arise in the conversion of analog to spiking LSTMs. The three main obstacles are the adaptation of an ASN, the behavior of ASN_σ and the shape of AAN_σ.

Principally, the ASN activation function is derived for steady-state adapted spiking neurons, and the difference between the ASN and the AAN in the adaptation phase causes an error that may be large for fast changing signals. This error is subsequently integrated by the CEC. Analog-valued spikes as explored in [17] could likely resolve this issue since they need much less time to adapt to their input signal, at the expense of some loss of computational efficiency. One more method was explored to remove the adaptation error in the CEC. In Figure 2.7 (left) we can see that the spiking CEC increases too much when ASN_{in} is adapting, because ASN_{in} sends more spikes in its adaptation phase than in its stable phase. So some of these spikes need to be ignored or discounted in value. Also notice that in an ASN that has not reached its stable state yet input signal S differs significantly from the incoming PSC I, since S needs time to catch up to I. The difference between S and I can be used as a discounting factor for the incoming spikes to the CEC during the adaptation phase. $|I - S|$ is large during the adaptation phase and small during the stable phase. Multiplying all the incoming spikes of ASN_{out} in Figure 2.7 (top) by $1 - \frac{|I-S|}{h}$ removed the difference between the value of the analog CEC and the spiking CEC almost completely. However, this idea has not been tested sufficiently.

The problem with the behavior of ASN_σ is linked to the ASN adaptation problem. For example in the sequence prediction tasks the input gate and

the input cell of the Adaptive Analog LSTM are almost, but not completely, closed during most of the trial. But in the Adaptive Spiking LSTM ASN_σ will be 0, instead of close to 0. This is because every 40ms a new input neuron starts firing, so ASN_σ is trying to adapt to its input signal during the whole trial, without actually reaching its stable state. This results in a negative output for the ASN_σ of the input gate, which is rounded of to 0 by ASN^* in Figure 2.6. This small difference between the analog and spiking states of the input gate and input cell can add up over time to an error in the CEC value that will make a significant difference. There is no clear solution to solve this, since the slow adaptation is inherent to the ASN model.

The last problem, concerning the shape of AAN_σ, makes it hard for the AAN networks to learn the tasks and is also one of the causes for the problem with the behavior of ASN_σ. As can be seen in Figure 2.4 the positive and negative limits of ASN_σ are respectively 1 and 0, but ASN_σ approaches its limit much slower than a usual sigmoid function. Consequently, the Adaptive Analog LSTM will have a hard time to close its input gate and input cell and this leaking signal is then amplified because it is multiplied by $\frac{\Delta T}{\tau_\eta}$, which is equal to 4 in our experiments. This results in a CEC that slowly fills up during the course of the trial and makes it harder to train compared to the traditional LSTM. But the leaking CEC of the Adaptive Analog LSTM is also hard for the spiking CEC to imitate. The spiking LSTM generally handles sudden changes and stable signals in the CEC well, but has trouble when the CEC should slowly increase or decrease.

The translation from analog to spiking was successful for the noiseless and noisy T-maze tasks with corridor lengths 20, but failed for the noiseless and noisy T-maze tasks with corridor lengths 50 and 40 respectively. The reason for this failure seems to be twofold. First of all, the formulation of Advantage Learning demands the Q-values to increase when the agent gets closer to the T-junction, because future rewards are lowered by a discount factor γ per time step. Thus the agent needs to know where it is located in the corridor and the only way to do this is by using its LSTM unit. The agent slowly increases its CEC during the course of the trial to adjust its Q-values. But exactly this slow build-up in the CEC is hard to imitate for the spiking CEC, as was mentioned in the previous paragraph. By changing the discount factor γ from 0.98 to for example 0.995 the CECs will need to increase less during the trial. The second reason why the translation from analog to spiking for a long corridor failed was that the Q-values were too close to each other. This

made the action selection very sensitive to noise in the CECs and the noise in the ASNs in general. One way to enlarge the differences between the Q-values is to increase the difference between the positive and the negative rewards. Training the network with a positive reward of 4.0 and a negative reward of -0.5 instead of -0.2 will need more training iterations.

Although the adaptive spiking LSTM implemented in this thesis does not have output gates, they can be included by following the same approach used for the input gates: a modulation of the synaptic strength. The reasons for omitting the output gate are multiple: first of all, most of the tasks do not really require output gates and often the output of the LSTM can be suppressed by regular units; moreover, modulating each output synapse independently is less intuitive and biologically plausible than for the input gates. A similar argument can be made for the forget gates, which were not included in the original LSTM formulation: here, the solution consists in modulating the decaying factor of the CEC.

The previously described problems with the leaking CEC and the differences in behaviour between AAN_σ and ASN_σ need to be solved in order to get better results for more complicated tasks with the Adaptive Spiking LSTM. One could try to replace ASN_σ in the input gate and input cell by multiple ASNs, which can then be fully closed. But at least a small mismatch between the analog and spiking CEC seems to be inevitable, so it is also worth trying to design networks and training methods that are more robust to noisy CEC values.

In later versions an output and forget gate might be added to enable a network with Adaptive Spiking LSTMs to perform more difficult tasks, like speech recognition using the TIDIGIT dataset [41] or Wall Street Journal speech corpus [42]. Finally we might also try to tune the parameters in the ASN to decrease the firing rate without losing performance.

Bibliography

[1] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[3] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[5] D. Attwell and S.B. Laughlin. An energy budget for signaling in the grey matter of the brain. *J. Cerebral Blood Flow & Metabolism*, 21(10):1133–1145, 2001.

[6] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760*, 2017.

[7] Ning Qiao, Hesham Mostafa, Federico Corradi, Marc Osswald, Fabio Stefanini, Dora Sumislawska, and Giacomo Indiveri. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses. *Frontiers in neuroscience*, 9:141, 2015.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[9] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] SK Esser et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *PNAS*, page 201604850, September 2016.

[12] D. Neil, M. Pfeiffer, and S-C. Liu. Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks. 2016.

[13] 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. In *IEEE IJCNN*, pages 1–8, July 2015.

[14] P. O'Connor, D. Neil, S-C. Liu, T. Delbruck, and M. Pfeiffer. Real-time classification and sensor fusion with a spiking deep belief network. *Frontiers in neuroscience*, 7, 2013.

[15] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

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

[17] D. Zambrano and S.M. Bohte. Fast and efficient asynchronous neural computation with adapting spiking neural networks. *arXiv preprint arXiv:1609.02053*, 2016.

[18] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[19] Wulfram Gerstner. A framework for spiking neuron models: The spike response model. *Handbook of Biological Physics*, 4:469–516, 2001.

[20] Y.C. Yoon. LIF and Simplified SRM Neurons Encode Signals Into Spikes via a Form of Asynchronous Pulse Sigma-Delta Modulation. *IEEE TNNLS*, pages 1–14, 2016.

[21] P. O'Connor and M. Welling. Sigma delta quantized networks. *arXiv preprint arXiv:1611.02024*, 2016.

[22] Daniel Neil, Jun Haeng Lee, Tobi Delbruck, and Shih-Chii Liu. Delta networks for optimized recurrent network computation. *arXiv preprint arXiv:1612.05571*, 2016.

[23] Greg Diamos, Shubho Sengupta, Bryan Catanzaro, Mike Chrzanowski, Adam Coates, Erich Elsen, Jesse Engel, Awni Hannun, and Sanjeev Satheesh. Persistent rnns: Stashing recurrent weights on-chip. In *International Conference on Machine Learning*, pages 2024–2033, 2016.

[24] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[25] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128 $\times$ 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2):566–576, 2008.

[26] S.M. Bohte. Efficient Spike-Coding with Multiplicative Adaptation in a Spike Response Model. In *NIPS 25*, pages 1844–1852, 2012.

[27] J.O. Rombouts, S.M. Bohte, and P.R. Roelfsema. Neurally plausible reinforcement learning of working memory tasks. In *NIPS 25*, pages 1871–1879, 2012.

[28] B Bakker. Reinforcement Learning with Long Short-Term Memory. In *NIPS 14*, pages 1475–1482, 2002.

[29] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[30] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[31] Geoffrey Hinton. Can the brain do back-propagation. In *Invited talk at Stanford University Colloquium on Computer Systems*, 2016.

[32] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[33] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 189–194. IEEE, 2000.

[34] 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):500, 1952.

[35] Eugene M Izhikevich. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[36] Wulfram Gerstner and Werner M Kistler. *Spiking neuron models: Single neurons, populations, plasticity*. Cambridge university press, 2002.

[37] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[38] M.E Harmon and L.C. Baird III. Multi-player residual advantage learning with general function approximation. *Wright Laboratory*, pages 45433–7308, 1996.

[39] Marco Wiering and Jürgen Schmidhuber. Hq-learning. *Adaptive Behavior*, 6(2):219–246, 1997.

[40] K. Cho et al. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[41] R Gary Leonard and George Doddington. Tidigits speech corpus. *Texas Instruments, Inc*, 1993.

[42] Douglas B Paul and Janet M Baker. The design for the wall street journal-based csr corpus. In *Proceedings of the workshop on Speech and Natural Language*, pages 357–362. Association for Computational Linguistics, 1992.