Learning Feedforward and Recurrent Deterministic Spiking Neuron Network Feedback Controllers

Tae Seung Kang\textsuperscript{1} and Arunava Banerjee\textsuperscript{1}

\textsuperscript{1}Computer and Information Science and Engineering Department, University of Florida.

\textbf{Keywords:} Spiking neuron, feedback control

\textbf{Abstract}

We consider the problem of feedback control when the controller is constructed solely of deterministic spiking neurons. Although spiking neurons and networks have been the subject of several previous studies, analysis has primarily been restricted to a firing rate model. In contrast, we construct a spike timing based deterministic spiking neuron controller whose control output is one or multiple sparse spike trains. We model the problem formally as a hybrid dynamical system comprised of a closed loop between a plant and a spiking neuron network controller. The construction differs from classical controllers owing to the fact that the control feedback to the plant is generated...
by convolving the spike trains with fixed kernels, resulting in a highly constrained and stereotyped control signal. We derive a novel synaptic weight update rule via which the spiking neuron network controller learns to hold process variables at desired set points. We demonstrate the efficacy of the rule by applying it to the classical control problem of the cart-pole (inverted pendulum). Experiments demonstrate that the proposed controller has a larger region of stability as compared to the traditional PID controller, and its trajectories differ qualitatively from those of the PID controller. In addition, the proposed controller with a recurrent network generates sparse spike trains with rates as low as 1.99Hz.

1 Introduction

While there is considerable debate in the scientific community regarding the cognitive capacity of various animal species, there is general agreement that animals are exquisite control systems. Whether it be the flight of a dragonfly or the walking of a biped (such as a human), engineered systems pale in comparison to the versatility and robustness displayed by their animal counterparts. Even more intriguing is the fact that in many instances the particular skill, locomotion for instance, is learned. Our goal in this work is to address this question of learning to control in the context of biologically motivated constraints—specifically, the fact that the constituent neurons of animal brains communicate with one another using action potentials (also known as spikes).

In the vast majority of biological systems, the control signal received by the muscles are in the form of spike trains generated by motor neurons. The controller itself is a net-
work of spiking neurons that resides upstream from the motor neurons. The controller receives inputs, which in the case of a feedback controller are process variables that are to be maintained at fixed or dynamically varying set points. The process variable input into the controller is in turn computed elsewhere and incorporates the combined output of one or more sensory systems.

To bring the problem of learning a spiking neuron network controller into sharp relief, we abstract away all aspects of the system that are of secondary concern and replace them with simple, fixed, and predefined alternatives. In particular, we model the entire process beginning at the spike train output of the controller and culminating at the control signal generated (such as the force exerted by the muscle) using fixed convolution kernels. The impact of the control signal on the organism in its environment, we model using a fixed plant. Finally, we model the input of the process variables as postsynaptic potential inputs into specifically identified neurons of the controller. Our objective is to devise a formal synaptic weight update rule that when applied to the neurons of the controller, causes the controller to learn to perform the control task.

That the above problem differs from those previously studied in feedback control, can be discerned from the following observation. Traditional feedback controllers such as the proportional-integral-derivative (PID) controller or its variants are designed to solve a control problem in the continuous domain with few restrictions. The process variable is a bounded continuous function of time, and so is the control signal generated by the controller; there is little else that constrains these functions. In contrast, the control output generated by the spiking neuron network controller is an ensemble of spike trains. The spike trains when convolved with the fixed convolution kernel referred
to above, leads to a highly restricted and stereotyped signal. In particular, it is easy to observe that given a kernel, there exists a bound $C$ such that any non-zero control signal $f$ satisfies $\|f\|_\infty > C$—informally, the controller has the choice between generating no output or an output larger than a fixed strength. This has immediate implications for the stability of the fixed point (determined by the set point) of the combined (the controller and the plant in closed loop) dynamical system; the process variable can at best be made to oscillate around the set point.

The overall goal of the paper is to demonstrate that deterministic spiking neuron based controllers can be learned, and not to characterize a pre-given spike based controller. To our knowledge, this is the first deterministic spike based controller that has been demonstrated to learn a classical task. The controller does operate very much like a bang bang controller, and it has learned to operate that way. This work is a first step toward control of more complex dynamical systems such as winged flight or bipedal walking.

Our objective is described schematically in Figure 1. We consider a hybrid dynamical system constructed out of a closed loop between a plant (we consider the classical problem of the inverted pendulum in this work as described in Section 4, but this could be replaced with any well defined plant), and a network of spiking neurons that models the controller. In the figure, the black vertical bars indicate the weights on spikes. That is, we virtually assign weights to spikes (denoted by the height of the bars) instead of the corresponding synapse. The conceptual underpinnings of this are described in Section 5. The vertical double-headed arrows next to the black bars denote the perturbation of the corresponding weights. The four dashed arrows beginning at the second vertical
Figure 1: The hybrid dynamical system that models the control problem. (a) Plant. The plant has a state that can be changed by external forces. The state is described by a set of process variables. For the cart-pole plant, these are the vertical angle $\theta$ and the angular velocity $\dot{\theta}$. (b) Controller. The proposed spiking neuron controller is a feedforward or recurrent neuron network that takes the plant state as input and produces output forces to control the plant. The synaptic update rules are set such that the weights on the synapses receiving the continuous process variable inputs change only when there are spikes generated by the "output neurons" of the controller. This is indicated by the vertical dotted lines. Perturbing the synaptic weights perturb the output spikes of the network in time (dotted arrows), which when convolved with the kernel creates a perturbation in the control signal. The control signal is optimized based on an error function that embodies the deviation of the process signal from its set point. (c) Models. We present three models: Model 1, Model 2, and Model 3, based on the network structure of the output neurons, the number of output neurons of the controller, and their corresponding force magnitude assignment. A circle indicates an output neuron of the proposed controller shown in (b). The vertical arrow stemming from a circle denotes a force produced by the neuron with its length representing the force magnitude. The longer the length, the larger the force magnitude.
lines from $t = \Upsilon$ (Past) denote the impact of the weight perturbations on the subsequent future spike times.

In this paper, we present three models of successively increasing complexity, Model 1, Model 2, and Model 3, based on the network structure of the neurons, number of output neurons of the controller and their corresponding force magnitude. Model 1, the simplest model, is defined as a network with the simplest possible architecture, that is no hidden layers. In the case of the inverted pendulum there are two output neurons (for left and right direction) with the same force magnitude. Model 2, the model of intermediate complexity, is defined as a network with four or more output neurons (two or more left and two or more right) with distinct force magnitudes, with a feedforward architecture comprised of one or more hidden layers. Model 3, the most general model, is defined as a recurrent network where the output neurons are fully connected with each other as well as other neurons in the network. This model can be combined with Model 1 or Model 2 to make hybrid architectures. The goal in each case is to incrementally update the synaptic weights on the neurons of the network such that the network’s output spike trains when convolved with the force kernel causes the process variables of the plant to deviate as little as possible from a predefined set point. The process variables are in turn input into the spiking neuron network controller as postsynaptic potentials.

Our approach is based on the observation that if (a) synaptic weights are only updated at the times that the network generates spikes, and (b) one analyzes the past times at which the network generated spikes, one can perform a perturbation analysis that would suggest a superior set of weights in the past. Since we can not reach into the past to change synaptic weights, the current synaptic weights of the network are updated to
reflect these improvements. The synaptic update rule is used to train the network from randomly generated initial conditions of the plant in an online manner until failure. At failure, the plant is reinitialized to a different initial condition and the learning process continues.

The remainder of the paper is structured as follows. Section 2 reviews the literature and Section 3 describes the neuron model. Section 4 briefly describes the plant used in this work, as well as the process variables and their corresponding set points. Section 5 comprises the core of our contribution where the synaptic weight update rule is derived. Section 6 discusses the experimental results from several variations of the controller, and Section 7 concludes the paper.

## 2 Background

Neural networks have been popular in solving various problems since Rumelhart et al. reintroduced the gradient descent based error backpropagation algorithm [1] to the machine learning community. The error backpropagation algorithm has also been used for spiking neural networks [2, 3]. [4] introduced SpikeProp to compute the XOR problem by applying the backpropagation algorithm to spiking neuron networks based on temporal coding (or spike timing based coding) [5]. Their supervised learning rule generates a desired pattern of spikes with the constraint that each output neuron is allowed to fire only once in the prescribed time window. [6] extended this rule to multiple output spikes; however, the first output spike was used for the error function. [7] presented a spiking neural network based controller to regulate a robot’s arms with 4-degrees of
freedom. The neuron model they used is the Izhikevich’s model [8] and the learning algorithm is Spike Timing-Dependent Plasticity (STDP) [9]. Their controller displays high firing rates due to the assumption of rate-based coding. There are also other neuron models commonly used: the Hodgkin-Huxley with compartmental model [10] is the most realistic, the Leaky-integrate and fire (LIF) [11] is the simplest model, and the Spike Response model (SRM) [12] is intermediate. In this work, the SRM is chosen because it is biologically realistic and is yet simple to simulate [13].

A typical control loop feedback controller such as PID controller [14] produces a continuous controller output. Recently, Gerstner et al. [15, 16] studied control problems for motor systems using reinforcement learning. They used the actor-critic model [17, 18] to train the networks. Biped control simulation using spiking neural networks was discussed in [19, 20]. The authors used genetic algorithms with evolution strategies to train the biped to walk. Our controller’s outputs are in the form of spikes which are discrete signals in real time (fired or not). These signals when convolved with a fixed kernel and thus highly constrained. In this sense, our proposed controller has its unique properties.

Most of the models mentioned above are based on a rate-based coding paradigm. Our proposed controllers are based on a spike timing based coding, called temporal coding [5]. [21] studied a perturbation analysis [22] to reveal how perturbations in the weights and times of the input spikes of a neuron translate to perturbations in the timing of its output spikes. [23] have made some initial progress in understanding what transformations feedforward networks can implement given a depth/size and a spiking neuron model. [24] presented the Tempotron addressing the decision problem using
bounded input spike trains to generate an output spike for one case and not for the other case. Its gradient descent was implemented based on the error between the maximum postsynaptic potential (PSP) and the firing threshold. [25] proposed a probabilistic learning rule called ReSuMe that implements the Widrow-Hoff learning rule for linear-Poisson units. The instantaneous input firing rates were linearly combined with the synaptic weights to produce the instantaneous output rate. The output spike train was then sampled from a non-homogeneous Poisson process with the variable output rate. [26] extended this rule to multilayered networks, however the linearity of the neuron model does not generalize the capacity of the multilayer network. The measure of disparity between two vectors of spikes times, the output spike train and the desired spike train, has been studied in the literature [27, 28, 29]. Although our proposed controller can be described to be most similar to [21] and [4], unlike these, ours does not require the prescription of the desired spike train and the input to the network is continuous and/or spike trains. This paper also presents an extension to recurrent networks of [30] which discusses feedforward spiking neuron feedback controllers.

3 Neuron Model

We use a minor variation of the Spike Response Model (SRM) [12] for the neurons in our controller. The neurons receive continuous time process variable inputs at its synapses. In a feedforward network, the neurons do not receive any input from other neurons whereas they do in a recurrent network. All neurons that receive input signals from other neurons receive them as spikes which are then turned into postsynaptic
potentials (PSPs) generated from these afferent (incoming) spikes at synapses. The membrane potential at the soma of the neuron is the synaptically weighted sum of postsynaptic potentials generated by the current values of the process variables (or generated by afferent spikes) and afterhyperpolarizing potentials (AHPs) generated by the efferent (outgoing) spikes that have departed the soma of the neuron. The neuron generates a spike when the membrane potential crosses the threshold $\Theta$ from below. Formally, the membrane potential, $P$, of the neuron at the current time is given by

$$P = \sum_{i \in \Gamma} w_i x_i(0) + \sum_{k \in \mathcal{F}} \eta(t_k^O).$$ \hspace{1cm} (1)

where $\Gamma$ is the set of synapses, $w_i$ is the weight of synapse $i$, $x_i(t)$ is the continuous process variable input signal at synapse $i$, and $t = 0$ is the current time (with positive $t$ indicating past). Similarly, $\eta$ is the prototypical after-hyperpolarizing potential (AHP) elicited by an efferent spike of the neuron and $\mathcal{F}$ is the set of past efferent spikes of the neuron. We assume in addition that all efferent spikes that were generated earlier than $t = \Upsilon$ in the past have no effect on the present membrane potential of the neuron (See Figure 1). The functional form of the AHP of a spike that we have used (and this can be modified without affecting the analysis) is

$$\eta(t) = Re^{-t/\gamma} \text{ for } 0 < t \leq \Upsilon \text{ and } 0 \text{ otherwise} \hspace{1cm} (2)$$

where $R$ denotes the instantaneous fall in potential after a spike and $\gamma$ controls its rate of recovery.

Likewise, the membrane potential of intermediate neurons (which can potentially receive the process variable input signal) is given by

$$\sum_{i \in \Gamma} w_{i,l} x_i(0) + \sum_{i \in \Gamma} \sum_{j \in \mathcal{F}_i} w_{i,j} \xi(t_{i,j}^I) + \sum_{k \in \mathcal{F}} \eta(t_k^O).$$ \hspace{1cm} (3)
where $\Gamma$ is the set of synapses with continuous input signals, $\Gamma_s$ is the set of synapses with afferent (incoming) spikes, $w_{i,l}$ is the weight of synapse $i$ immediately prior to output spike $l$, $w_{i,j}$ is the weight of afferent spike $j$ coming from synapse $i$, and $\xi$ is the PSP elicited by an afferent spike. The functional form of $\xi_i$ that we have used (and this can be modified without affecting the analysis) is

$$\xi(t) = \frac{e^{-\beta d^2/t}}{d\sqrt{t}} e^{-t/\tau}$$  \hspace{1cm} (4)

where $\beta$ and $\tau$ control the rate of rise and fall of the PSP, and $d$ denotes the distance of the synapse from the soma.

## 4 Overview of Plant

The plant we consider in this work is the classical control problem of the cart-pole (also known as the inverted pendulum) as shown in Figure 2. The cart-pole comprises of an inverted rigid pendulum, with the mass at the top. The pendulum is fulcrumed at its base to the cart which rests on a frictionless surface. Force can be applied to the cart to move it along the horizontal axis. The control problem is to apply forces to the cart to maintain the upright position of the pendulum. The process variables that we have considered for this plant are: $\theta$, the angular deviation of the pendulum from the upright position, and $\dot{\theta}$, the angular velocity of the pendulum. The set points for the process variables are $\theta = 0, \dot{\theta} = 0$. The details of the system dynamics can be found in [18].

All quantities of interest as presented in Section 5 we have derived through numerical computations.
Figure 2: Inverted pendulum plant.

5 Feedback Control using Spiking Neurons

As described in the previous chapter, the desired state of the cart-pole plant is to maintain a zero vertical angle and a zero angular velocity of the pendulum. These process variables are input into different synapses of the controller neurons in the following form: the angle $\theta$ and the angular velocity $\dot{\theta}$. As discussed in Section 6, we have experimented both with the case where $\dot{\theta}$ is a process variable to be controlled at the set point 0, and the case where it is not. The control signal output of a neuron is generated by convolving the output spike trains of the neurons with a fixed force kernel $\kappa(t)$ as defined in the next section. Even number of neurons are used to generate forces, the first set generating forces to the right and the second set to the left. We analyze the general case of multiple neurons with different $\kappa(t)$ force kernels coming together to constitute the final control signal.
5.1 The error function

The proposed spiking neuron based controller depicted in Figure 1 can be formally modeled as follows. Consider a plant with process variables represented by vector \( \langle x_1(0), x_2(0), ..., x_D(0) \rangle \), where \( D \) is the number of process variables to be controlled. The desired state of the plant (i.e., the set point of the process variables) is represented by \( \langle x_{\ast 1}(0), x_{\ast 2}(0), ..., x_{\ast D}(0) \rangle \). The error \( E \) can then be defined by \( \frac{1}{2} \sum_{i=1}^{D} (x_i(0) - x_{\ast i}(0))^2 \).

The synaptic update rule that we derive next is based on minimizing this objective using gradient descent, which in the case of the full set of process variables.

A traditional controller receives continuous time process signals from the plant and generates a continuous time control output. The proposed controller, however, generates spike trains, one for each neuron, instead of a continuous output. The spike train output of each neuron \( j \) is then convolved with the kernel

\[
\kappa(t) = te^{-t/\tau_f}
\]  

(5)

to generate a force:

\[
F_j = \mu_j \sum_{i \in F_j} \kappa(j i t^O_i)
\]  

(6)

where \( \tau_f \) is the time constant, \( \mu_j \) is the magnitude assigned to neuron \( j \), \( j i t^O_i \) is the time elapsed since the generation of the \( i^{th} \) most recent efferent spike of the output neuron \( j \), and \( F_j \) is the set of past spikes of output neuron \( j \). The final force \( F \) applied to the plant is

\[
F = \sum_j \pm F_j
\]  

(7)

where \( \pm \) represents the sign of \( F_j \), + for neurons with the positive forces and \( - \) for those with the negative forces (opposite direction of the positive forces).
5.2 Gradients of the error function

Our overall objective is to compute the gradient of the error with respect to the synaptic weights on the controller neurons. We do this in stages. We first compute the gradient with respect to the output spike times of the controller neurons. Applying chain rule, we then have

$$\frac{\partial E}{\partial (j_{t_l}^O)} = \frac{\partial E}{\partial F} \frac{\partial F}{\partial (j_{t_l}^O)}$$  \hspace{1cm} (8)

where $j_{t_l}^O$ is the time elapsed since the departure of the $l^{th}$ most recent efferent (outgoing) spike of neuron $j$, and

$$\frac{\partial E}{\partial F} = \frac{1}{2} \sum_i \left( \frac{\partial (x_i(0) - x_i^*(0))^2}{\partial x_i(0)} \right) \frac{\partial x_i(0)}{\partial F} = \sum_i (x_i(0) - x_i(0)^*) \frac{\partial x_i(0)}{\partial F}$$  \hspace{1cm} (9)

$$\frac{\partial F}{\partial (j_{t_l}^O)} = \frac{\partial F}{\partial F_j} \frac{\partial F_j}{\partial (j_{t_l}^O)} \frac{\partial (j_{t_l}^O)}{\partial (j_{t_l}^O)} = \pm \mu_j \frac{\partial \kappa}{\partial t} \bigg|_{j_{t_l}^O}$$  \hspace{1cm} (10)

In Eq (9), $\frac{\partial x_i(0)}{\partial F}$ is drawn as a numerical derivative from the plant: $\frac{\partial x_i(0)}{\partial F_j} \approx \frac{\Delta x_i(0)}{\Delta F_j}$.

5.3 Perturbation analysis

Our goal now is to determine how perturbations in synaptic weights of the controller neurons translate to perturbations in the times of their output spikes. We achieve this by first assuming that synaptic weights are only perturbed at the times of the output spikes (see Figure 1).
5.1 Feedforward network

Consider the state of a neuron at the time of the generation of output spike $t^O_l$. The membrane potential of the neuron before perturbations of the weights on the input signals is given by

$$\tilde{\Theta} = \sum_{i \in \Gamma} w_{i,l}(x_i(t^O_l) - x^*_i(t^O_l)) + \sum_{k \in \mathcal{F}} \eta(t^O_k - t^O_l).$$

(11)

where $x_i$ is the process variable at synapse $i$, $x^*_i$ is the desired state of the plant at synapse $i$, $\Gamma$ is the set of synapses of the neuron, and $w_{i,l}$ is the weight of synapse $i$ immediately prior to output spike $l$. Note that we have replaced $\Theta$ with $\tilde{\Theta}$ to account for those output spikes that at the time of the generation of $t^O_l$ were less than $\Upsilon$ old, but are now past that bound. Since $(x_i(t^O_l) - x^*_i(t^O_l))$ is a vector, without loss of generality, the above equation can be restated as

$$\tilde{\Theta} = \sum_{i \in \Gamma} w_{i,l}x_i(t^O_l) + \sum_{k \in \mathcal{F}} \eta(t^O_k - t^O_l).$$

(12)

Had the various quantities in Eq (12) been perturbed, we would have

$$\tilde{\Theta} = \sum_{i \in \Gamma} (w_{i,l} + \Delta w_{i,l})x_i(t^O_l + \Delta t^O_l) + \sum_{k \in \mathcal{F}} \eta(t^O_k - t^O_l + \Delta t^O_k - \Delta t^O_l).$$

(13)

Using a first order Taylor approximation, we get

$$\tilde{\Theta} = \sum_{i \in \Gamma} \left( w_{i,l} + \Delta w_{i,l} \right) \left( x_i(t^O_l) + \frac{\partial x_i}{\partial t} \bigg|_{t^O_l} \Delta t^O_l \right)$$

$$+ \sum_{k \in \mathcal{F}} \left( \eta(t^O_k - t^O_l) + \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_l)} (\Delta t^O_k - \Delta t^O_l) \right).$$

(14)

Combining Eq (12) and (14), dropping higher order terms and rearranging, we get

$$\Delta t^O_l = \sum_{i \in \Gamma} \frac{\Delta w_{i,l} x_i(t^O_l) + \sum_{k \in \mathcal{F}} \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_l)} \Delta t^O_k}{\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_i}{\partial t} \bigg|_{t^O_l} + \sum_{k \in \mathcal{F}} \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_l)}}.$$
We can now derive the final set of quantities of interest from Eq (15). If we perturb
the weight $w_{i,l}$, there will only be a direct effect of the perturbation since $w_{i,l}$ does not
impact spikes prior to $t^O_l$. Therefore, we have

$$\frac{\partial t^O_l}{\partial w_{i,l}} = \frac{x_i(t^O_l)}{\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_i}{\partial t} \bigg|_{t^O_l} + \sum_{k \in \mathcal{F}} \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_l)}}. \quad (16)$$

If instead, we perturb $w_{i,p}$ where $p > l$ (so that $t^O_p > t^O_l$), there will only be an
indirect effect of the perturbation through previously generated spikes. Therefore, for
$p > l$ we have the recursion

$$\frac{\partial t^O_l}{\partial w_{i,p}} = \frac{\sum_{k \in \mathcal{F}} \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_p)} \frac{\partial t^O_k}{\partial w_{i,p}}}{\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_i}{\partial t} \bigg|_{t^O_l} + \sum_{k \in \mathcal{F}} \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_l)}}. \quad (17)$$

### 5.2 Recurrent network

In this section, we discuss the perturbation analysis in a recurrent network where
the output neurons are connected fully. If a synaptic weight is perturbed in a recurrent
network, its subsequent spike times are perturbed. The aim is to compute the effect
of the weight perturbation on the subsequent spike times. In the case of feedforward
networks, this was done with direct effects where a perturbation has an effect upon the
spike trains of the adjacent neurons. In a recurrent network, however, it is possible
that the effect of a spike travels across multiple neurons. If a spike crosses a neuron,
its direct effect on a spike of the next adjacent neuron will be vanished as they are
physically apart. Instead, its indirect effects will appear. Therefore, we need to know
the intermediate effects in between a pair of spikes. This way, the total effects between
a pair of spikes can be obtained by summing up the direct effect and indirect effects.
There are three cases of direct effect between a pair of spikes: input to input, input to output, and output to output. For input to input spike, there is no direct effect as there is no change to the membrane potential. For input to output spike, there is a direct effect due to the change to PSP. Similarly, there is a direct effect for output to output spike due to the change to AHP. This tells us that a spike of a neuron affects only on the spikes generated from neurons directly connected.

Computing the total effects of a perturbation can be solved by recursion where the total effect of the perturbation of a weight on the spike time is the sum of the direct effect and the sum of the multiplications of indirect effect and the total effect of the previous spikes. Then, the total derivative of $t^O_l$ with respect to $w_{i,j}$ at synapse $i$ can be written as a recursion

$$\frac{Dt^O_l}{\partial w_{i,j}} = \frac{\partial t^O_l}{\partial w_{i,j}} + \sum_{k \in \Gamma_{i,j}} \frac{\partial t^O_l}{\partial t^O_k} \frac{Dt^O_k}{\partial w_{i,j}} \quad (18)$$

where $\Gamma_{i,j}$ is the set of all spikes generated by all the output neurons since $w_{i,j}$ is perturbed. The first term indicates the direct effect of the weight perturbation and the second term indicates the indirect effect through other spikes. Similarly, the total derivative of $t^O_l$ with respect to $w_{i,p}$ can be written as a recursion

$$\frac{Dt^O_l}{\partial w_{i,p}} = \frac{\partial t^O_l}{\partial w_{i,p}} + \sum_{k \in \Gamma_{i,p}} \frac{\partial t^O_l}{\partial t^O_k} \frac{Dt^O_k}{\partial w_{i,p}} \quad (19)$$

where $\Gamma_{i,p}$ is the set of all spikes generated by all the output neurons since $w_{i,p}$ is perturbed.

Consider the state of a neuron at the time of the generation of output spike $t^O_l$ in a recurrent network. The membrane potential of the neuron before perturbations of the
weights on the input signals is given by

\[ \tilde{\Theta} = \sum_{i \in \Gamma} w_{i,l} x_i(t^O_l) + \sum_{i \in \Gamma_x} \sum_{j \in F_i} w_{i,j} \xi(t^I_{i,j} - t^O_l) + \sum_{k \in F} \eta(t^O_k - t^O_l). \] (20)

where \( \Gamma \) is the set of synapses with continuous input signals, \( \Gamma_x \) is the set of synapses with afferent (incoming) spikes, \( w_{i,l} \) is the weight of synapse \( i \) immediately prior to output spike \( l \), \( w_{i,j} \) is the weight of afferent spike \( j \) coming from synapse \( i \), and \( \xi \) is the PSP elicited by an afferent spike. Note that we have replaced \( \Theta \) with \( \tilde{\Theta} \) to account for those output spikes that at the time of the generation of \( t^O_l \) were less than \( \Upsilon \) old, but are now past that bound. Had the various quantities in Eq (20) been perturbed, we would have

\[ \tilde{\Theta} = \sum_{i \in \Gamma} (w_{i,l} + \Delta w_{i,l}) x_i(t^O_l + \Delta t^O_l) + \sum_{i \in \Gamma_x} \sum_{j \in F_i} (w_{i,j} + \Delta w_{i,j}) \xi(t^I_{i,j} + \Delta t^I_{i,j} - t^O_l - \Delta t^O_l) \]

\[ + \sum_{k \in F} \eta(t^O_k + \Delta t^O_k - t^O_l - \Delta t^O_l). \] (21)

Using a first order Taylor approximation, we get

\[ \tilde{\Theta} = \sum_{i \in \Gamma} (w_{i,l} + \Delta w_{i,l}) \left( x_i(t^O_l + \Delta t^O_l) + \frac{\partial x_i(t^O_l)}{\partial t^O_l} \Delta t^O_l \right) \]

\[ + \sum_{i \in \Gamma_x} \sum_{j \in F_i} (w_{i,j} + \Delta w_{i,j}) \left( \xi(t^I_{i,j} + \Delta t^I_{i,j} - t^O_l - \Delta t^O_l) + \frac{\partial \xi(t^I_{i,j} - t^O_l)}{\partial (t^I_{i,j} - t^O_l)} (\Delta t^I_{i,j} - \Delta t^O_l) \right) \]

\[ + \sum_{k \in F} \left( \eta(t^O_k - \Delta t^O_k - \Delta t^O_l) + \frac{\partial \eta(t^O_k - t^O_l)}{\partial (t^O_k - t^O_l)} (\Delta t^O_k - \Delta t^O_l) \right) \] (22)

Combining Eq (12) and (14), dropping higher order terms and rearranging, we get

\[ \Delta t^O_l = \frac{\sum_{i \in \Gamma} \Delta w_{i,l} x_i(t^O_l) + \sum_{i \in \Gamma_x} \sum_{j \in F_i} \Delta w_{i,j} \xi(t^I_{i,j} - t^O_l)}{\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_i}{\partial t} \bigg|_{t^O_l} + \sum_{i \in \Gamma_x} \sum_{j \in F_i} w_{i,j} \frac{\partial \xi}{\partial t} \bigg|_{(t^I_{i,j} - t^O_l)} + \sum_{k \in F} \frac{\partial \eta}{\partial t} \bigg|_{(t^O_k - t^O_l)}}. \] (23)
We can now derive the final set of quantities of interest from Eq (15). If we perturb the weight $w_{i,l}$, there will only be a direct effect of the perturbation since $w_{i,l}$ does not impact spikes prior to $t_{l}^{O}$. Therefore, we have

$$\frac{\partial t_{l}^{O}}{\partial w_{i,l}} = \frac{x_{i}(t_{l}^{O})}{-\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_{i}}{\partial t} |_{t_{l}^{O}} + \sum_{i \in \Gamma} \sum_{j \in F_{i}} w_{i,j} \frac{\partial \xi}{\partial t} |_{(t_{l,j}^{i} - t_{l}^{O})} + \sum_{k \in F} \frac{\partial \eta}{\partial t} |_{(t_{k}^{O} - t_{l}^{O})}},$$

(24)

and

$$\frac{\partial t_{l}^{O}}{\partial w_{i,p}} = 0 \quad (p > l),$$

(25)

$$\frac{\partial t_{l}^{O}}{\partial t_{i,p}} = \frac{w_{i,p} \frac{\partial \xi}{\partial t} |_{(t_{i,p}^{i} - t_{l}^{O})}}{-\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_{i}}{\partial t} |_{t_{l}^{O}} + \sum_{i \in \Gamma} \sum_{j \in F_{i}} w_{i,j} \frac{\partial \xi}{\partial t} |_{(t_{l,j}^{i} - t_{l}^{O})} + \sum_{k \in F} \frac{\partial \eta}{\partial t} |_{(t_{k}^{O} - t_{l}^{O})}},$$

(26)

and

$$\frac{\partial t_{l}^{O}}{\partial t_{p}^{O}} = \frac{\frac{\partial \eta}{\partial t} |_{(t_{p}^{O} - t_{l}^{O})}}{-\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_{i}}{\partial t} |_{t_{l}^{O}} + \sum_{i \in \Gamma} \sum_{j \in F_{i}} w_{i,j} \frac{\partial \xi}{\partial t} |_{(t_{l,j}^{i} - t_{l}^{O})} + \sum_{k \in F} \frac{\partial \eta}{\partial t} |_{(t_{k}^{O} - t_{l}^{O})}}.$$

(27)

If we perturb the weight $w_{i,j}$ for $j^{th}$ input spike at synapse $i$, we get

$$\frac{\partial t_{l}^{O}}{\partial w_{i,j}} = \frac{\xi(t_{l,j}^{i} - t_{l}^{O})}{-\sum_{i \in \Gamma} w_{i,l} \frac{\partial x_{i}}{\partial t} |_{t_{l}^{O}} + \sum_{i \in \Gamma} \sum_{j \in F_{i}} w_{i,j} \frac{\partial \xi}{\partial t} |_{(t_{l,j}^{i} - t_{l}^{O})} + \sum_{k \in F} \frac{\partial \eta}{\partial t} |_{(t_{k}^{O} - t_{l}^{O})}}.$$

(28)

### 5.4 Learning rules

Learning is accomplished via gradient descent. The learning rule is a type of Spike Timing-Dependent Plasticity (STDP) \[9\]; the weight updates depend on spike times.

The reason that the weights should be updated only when there are spikes is as follows.

If there are no spikes generated, the plant is in a safe kinematic range and thus no control
signal is necessary. This, in turn, indicates no need to update the weights. The weight updates are not independent. They get related to each other due to the common error functional on which gradient descent is performed. Applying chain rule, we get

\[
\frac{\partial E}{\partial w_{i,p}} = \sum_{l \in \mathcal{F}} \frac{\partial E}{\partial t_l^O} \frac{\partial t_l^O}{\partial w_{i,p}}. \tag{29}
\]

For a recurrent connection, we get

\[
\frac{\partial E}{\partial w_{i,p}} = \sum_{l \in \mathcal{F}} \frac{\partial E}{\partial t_l^O} \frac{Dt_l^O}{\partial w_{i,p}} \tag{30}
\]

where \(\frac{Dt_l^O}{\partial w_{i,p}}\) is the total derivative of \(t_l^O\) with respect to \(w_{i,p}\). These are computed using Eq 8, 16, and 17. The weight modification rule for synapse \(i\) at \(t_p^O\) is defined as

\[
w_{i,p} \leftarrow w_{i,p} - \alpha \frac{\partial E}{\partial w_{i,p}} \tag{31}
\]

where \(\alpha\) is the learning rate. Clearly we can not reach into the past to make these changes. We therefore institute a summed delayed update to the synapse at the current time.

\[
w_{i} \leftarrow w_{i} - \sum_{p \in \mathcal{F}} \alpha \frac{\partial E}{\partial w_{i,p}} \tag{32}
\]

The weight update is performed immediately after the generation of a spike by any of the output neurons. This way we are guaranteed that the weights on the synapses remain constant between any two successive spikes (on any neurons).

### 6 Experiments - Inverted Pendulum

To demonstrate the efficacy of the proposed controller, we performed multiple simulations on the inverted pendulum plant developed by Charles Anderson [31].
6.1 Setup

For these simulations, we defined a successful learning event of the controller as having balanced the pole without failure for one hour of simulation time. A failure was defined as an event where the process variables of the pole was not within a certain predefined range. For example, given the 1ms time step, the number of steps for a successful run was 3600000 (1 hour). The predefined range was $[-0.2094, 0.2094]$ for $\theta$ and $[-2.01, 2.01]$ for $\dot{\theta}$, respectively. Training of the controller was continued until success. During the training phase, the controller started with random network weights, random position ($\theta$), and random angular velocity ($\dot{\theta}$) of the pole. It then learned by updating the synaptic weights. Once it had successfully learned the task, we fixed the weights and tested the controller with random initial plant states to evaluate its robustness. If the pole fell before training succeeded, we restarted the training with random initial weights since the same weights would yield the same results of the failed run.

The plant was configured as: half-pole length $l = 0.5$ (m), pole mass $m = 0.1$ (kg), cart mass $M = 1.0$ (kg), and gravity $g = 9.8$ (m/s$^2$). The configuration of the controller was: time step $= 1$ms, threshold $= 0.001$, $\beta = 1.0$, $d = 1.5$, $\tau = 20$, $\tau_f = 20$ (ms), $R = -1000$, $\gamma = 1.2$, and $\alpha = 0.01$. The unit of $R$ is the same as that of the membrane potential. The magnitude of $R$ was set large to prevent a spike from being generated within 4-5 msec after a spike was generated by bringing down the membrane potential dramatically. We measured the firing rates of the output neurons in Hz (number of spikes per second). Specifically, the firing rate of a neuron is the total number of spikes generated by the neuron divided by the total running time of the simulation. In all the experiments, the same PID controller was used and its parameters
were set as $K_P = 20$, $K_I = 0.01$, and $K_D = 1$ obtained by the Ziegler-Nichols method.

### 6.2 Results

We start by describing a model that successfully learned the control response. Not surprisingly, when $\dot{\theta}$ as a state variable to control was removed, the controller failed to learn regardless of whether the controller was based on Model 1 or Model 2. To elaborate, although the controller was able to hold the pole upright for a short period in the training stage, it failed to hold the pole upright for 1 hour. Surprisingly, the controller with either model worked once we added $\dot{\theta}$. In what follows, we discuss the experimental results for each model.

#### 6.1 Model 1

Fig. 3 shows snapshots of the plant state (top) and the spike trains of the output neurons (bottom) of the proposed controller with 2 output neurons before and after training. The controller received two continuous inputs $(\theta, \dot{\theta})$ from the plant and produced output spike trains at the two output neurons. The output spike trains correspond to the left direction force and right direction force, respectively. In the top panels, the blue dotted line and the green dash-dot represent the time evolution of the vertical angle $\theta$ and angular velocity $\dot{\theta}$ of the pole in radian (°) and rad/sec, respectively. The red solid line is the force applied to the cart and the cyan dashed line is the error $E$. The force magnitude assigned to each output neuron was 100. The average firing rates of the two output neurons were 33.94Hz and 34Hz, respectively. Fig. 3a shows a snapshot
Figure 3: Snapshots of the plant state (top) and spike trains of the controller (bottom) for Model 1 with two output neurons before and after training. In all top figures, the blue dotted line is the vertical angle $\theta$, the green dash-dot is the angular velocity $\dot{\theta}$, the cyan dashed line is the error function $E$, and the red solid line is the force applied to the plant. (a) Before training, the synaptic weights are randomly chosen and fixed (no updates). As expected, the pole falls down quickly as the controller did not learn how to stabilize the plant. The spike train exhibits no particular pattern. The final force applied to the plant is scaled down by 10 fold ($\times 0.1$) of the actual values for improved visualization. (b) After training, the controller stabilizes the plant within a short amount of time (1000 ms). The spike trains are a bit dense but exhibit some patterns. The final force applied to the plant is scaled down by 100 fold ($\times 0.01$) of the actual values for improved visualization. (c) In the stable state, the pole oscillates around the set point (0, 0) and the error function $E$ is also at around 0. The trained controller behaves like a bang-bang controller. This results from the patterns of the output spike trains. After one neuron fires three times, the other neuron also fires three times, then the first neuron fires again, and so on.
Figure 4: Trajectories and coverages of Model 1 with two output neurons. (a) Trajectories of the plant state ($\theta$, $\dot{\theta}$) for Model 1 after training and the PID controller over time with different initial settings. The trajectories start at two points (-0.15, 1.4) and (-0.15, 1.5). (b) Trajectories in (a) zoomed in around the set point (0, 0). The plant for Model 1 oscillates between (0, -0.02) and (0, 0.02) in the stable condition. (c) Coverage of initial states ($\theta$, $\dot{\theta}$) with Model 1 controller. A green circle indicates a success while a red 'x' mark indicates a failure for the corresponding initial state. (d) Coverage of PID controller. The number of dots covered (green circle) is 36 for Model 1 controller and 32 for the PID controller. The proposed controller covers a larger area than the PID controller.
before training where the synaptic weights were randomly chosen and not updated. As shown in the top panel, the pole fell down shortly after the start of the simulation as the controller was untuned. This is further obvious from the bottom panel where the spike trains were randomly generated regardless of the state of the plant. After training, the pole successfully stood upright for 1 hour. The trained controller was tested on random initial conditions. Fig. 3b shows the starting condition in the testing phase to demonstrate how the proposed controller behaved in controlling the plant initially. Fig. 3c shows the stable state after a while. The final force applied to the plant is scaled down by 100 fold (×0.01) of the actual values for improved visualization. From the figures, it can be seen that the patterns of the spike trains are different in the start condition versus the stable state. Compared to the starting condition, the spike trains are sparse. In the starting condition, one output neuron generates spikes continuously while the other does intermittently. This is because the controller attempts to repeatedly push the plant in one direction. In the stable state, however, both neurons produce spikes alternately. After one neuron fires three times, the other neuron also fires three times, then the first neuron fires again, and so on.

Fig. 4 shows the trajectories of the plant state (θ, ˙θ) over time with two different initial settings. Further testing was performed over several initial states to determine the robustness and the coverage over initial states for the proposed controller with Model 1 as compared to the PID controller. In Fig. 4a, the trajectories start at two points (-0.15, 1.4) and (-0.15, 1.5). They are chosen to compare the performance of Model 1 against the PID controller. The blue solid line shows the trajectory of the plant state for Model 1 with the initial plant state (-0.15, 1.4). The green dotted line shows the trajectory of
the plant state for the PID controller with the same initial state. From the figure, we can observe that the trajectory of the proposed controller is different from that of the PID controller. While the plant for Model 1 eventually settles down, its trajectory is different from that of the PID controller. This demonstrates the fact that the proposed controller behaves differently suggesting a novel control mechanism. The red dashed line represents the trajectory of the plant state for Model 1 with the initial state (-0.15, 1.5). The cyan dash-dot is the trajectory for the PID controller with the same initial state. While Model 1 succeeds, the PID controller fails. Fig. 4b shows the trajectories in Fig. 4a zoomed in around the set point (0, 0) for better visualization. Fig. 4c and Fig. 4d show the coverages of stability for Model 1 and the PID controller, respectively. Although they do not exclusively include each other, Model 1’s coverage is larger than the PID. For example, the initial state (-0.2, 1.5) covered by Model 1 is not covered by the PID controller.

6.2 Model 2

We performed the same learning experiments above for 4, 6, and 8 output neurons with successively larger force kernels. In this case, the force magnitude assigned to each output neuron was symmetric: pairs of equal magnitude for the left and right force. For 4 output neurons, the force magnitude assigned to each output neuron was 1000, 500, 500, and 1000. For 6 output neurons, it was 300, 200, 100, 100, 200, and 300. For 8 output neurons, it was 250, 200, 150, 100, 100, 150, 200, and 250. Fig. 5 shows snapshots of the plant state (top) and the output spike trains of the controller (bottom) in the start and stable condition for Model 2 with 4, 6, and 8 output neurons after training.
Figure 5: Snapshots of the plant state (top) and the output spike trains of the controller (bottom) in the start and stable condition for Model 2 with 4, 6, and 8 output neurons after training. In the top figures, the blue dotted line and the green dash-dot represent the vertical angle $\theta$ and angular velocity $\dot{\theta}$ of the pole in radian ($^\circ$), respectively. The red solid line is the force applied to the cart and the cyan dashed line is the error function.
In the top figures, the blue dotted line and the green dash-dot represent the vertical angle $\theta$ and angular velocity $\dot{\theta}$ of the pole in radian ($^\circ$), respectively. The red solid line is the force applied to the cart and the cyan dashed line is the error function $E$. Note that the force applied to the plant is scaled down by 100 fold ($\times 0.01$) of the actual values for improved visualization. For each spike trains snapshot, the force magnitude assigned to each output neuron is symmetric: big to small and small to big. Fig. 5(a) and (b) show the snapshots for 4 output neurons. For the spike trains snapshot, the force magnitude assigned to each output neuron is 1000, 500, 500, and 1000 from top to bottom. Fig. 5(c) and (d) show the snapshots for 6 output neurons. The force magnitude assigned to each output neuron is 300, 200, 100, 100, 200, and 300 from top to bottom. Fig. 5(e) and (f) show the snapshots for 8 output neurons. The force magnitude assigned to each output neuron is 250, 200, 150, 100, 100, 150, 200, and 250 from top to bottom.

As shown in the figure, Model 2 controllers achieved the objective of stabilizing the plant. As in the case of Model 1, the spike trains exhibit regular patterns; neurons fired alternately periodically. In general, the spike train in the stable state was sparser than that in the start condition. It can also be observed that there are unnecessary spikes in the stable state. To elaborate, since we have neurons generating large forces as well as small ones, we need only the small force neurons to fire in the stable state. This can be mitigated by adding communication between output neurons so that they are aware of each other’s spike trains as in the case of Model 3 which will be discussed in the next section.

Fig. 6 shows the coverages of initial conditions that are controlled. As shown in the figures, the stability coverage increases with the number of neurons in the controller.
Figure 6: Stability coverages of the initial states \((\theta, \dot{\theta})\) of Model 2 controllers. The PID coverage is reproduced from Fig. 4d for convenient visual comparison. A green circle indicates a success while a red ‘x’ indicates a failure for the corresponding initial state. The 6 neuron controller covers a larger region than the 4 neuron controller, and the 8 covers a larger region than the 6. This implies that controllers with more neurons are more robust. When compared to the PID controller, the proposed controller covers more in general. Especially 6 and 8 neurons cover the whole region of that of the PID. This indicates the proposed controller has better control capability than the traditional PID controller.
4 has a larger region than 2, 6 has a larger region than 4, etc. Fig. 6d is reproduced from Fig. 4d to ease visual comparison with Model 2’s coverage. It is clear that Model 2 covers a much wider area than the PID. It should be noted that the coverage of 6 or 8 neurons subsumes the entire PID area. This suggests that the proposed controller is more robust than the traditional PID controller.

6.3 Model 3

We performed the same learning experiments for Model 3 in which the output neurons are recurrently connected. Fig. 7 shows the snapshots of the plant state and spike trains of the controller for Model 3 with two output neurons after training. The threshold for a neuron to fire was 0.001 and the force magnitude was 500. As shown in the figure, the controller stabilizes the plant within a short amount of time (1 second). Once the plant state has been stabilized, the plant settles down around the set point (0, 0) and the error function $E$ stays also at around 0. When compared to Model 1, the spike trains are much sparse: the average firing rates of the two output neurons are 14.10Hz and 11.59Hz, respectively. The firing rates for Model 1 in Fig. 3 were 33.94Hz and 34Hz. These are 2.4-2.93 times more dense than Model 3. This results from the fact the output neurons are connected such that they can communicate with each other. Unlike feedforward network in Fig. 3b if one neuron fires, the other one stops firing as shown in Fig. 7a.

We also conducted experiments for 4 output neurons. Fig. 8 shows the snapshots of the starting and stable conditions. The force magnitudes assigned to the output neurons are 300, 200, 200, and 300, respectively. The final force applied to the plant is scaled
Figure 7: Snapshots of the plant state (top figures) and spike trains of the controller (bottom figures) for Model 3 with two output neurons after training. In all top figures, the blue dotted line is the vertical angle $\theta$, the green dash-dot is the angular velocity $\dot{\theta}$, the cyan dashed line is the error function $E$, and the red solid line is the force applied to the plant. The final force applied to the plant is scaled down by 100 fold ($\times 0.01$) of the actual values for improved visualization. The average firing rates of the two output neurons are 14.10Hz and 11.59Hz, respectively. (a) After training, the controller stabilizes the plant within a short amount of time (1000 ms). The spike trains are sparse and exhibit some patterns. (b) In the stable state, the pole oscillates around the set point $(0, 0)$ and the error function $E$ is also at around 0.
Figure 8: Snapshots of the plant state (top figures) and spike trains of the controller (bottom figures) for Model 3 with recurrently connected 4, 6, and 8 output neurons after training. In all top figures, the blue dotted line is the vertical angle $\theta$, the green dash-dot is the angular velocity $\dot{\theta}$, the cyan dashed line is the error function $E$, and the red solid line is the force applied to the plant.
down by 100 fold ($\times 0.01$) of the actual values for improved visualization. The average firing rates of the 4 output neurons are 2.75Hz. Fig. 8a shows that after training, the controller stabilizes the plant within a short amount of time (1000 ms). The spike trains are sparse and exhibit some patterns. Fig. 8b shows that in the stable state, the pole oscillates around the set point (0, 0) and the error function $E$ stays also at around 0. The average firing rates of the 8 output neurons are 1.99Hz. Fig. 9 shows the coverages of the initial states ($\theta$, $\dot{\theta}$) for Model 3 with 2, 4, 6, and 8 output neurons after training. Fig. 9a shows the coverage of the controller with two output neurons in which 45 dots are covered. This is larger than 36 dots of Model 1 case with the same number of neurons. Fig. 9b shows the coverage for Model 3 with 4 neurons which covers 51 dots. This is larger than the 45 dots of Model 2 for the same number of neurons.

7 Conclusion

We have proposed spiking neuron network controllers and have applied them to the classical cart-pole control problem to demonstrate their efficacies. The derivation presented is general and can be applied to any feedforward or recurrent network. The primary advantage of our controllers is that it has a larger region of stability as compared to the traditional PID controller. Furthermore, our controllers behave in a manner different from the traditional PID controller. As demonstrated in our experiments, the proposed controllers succeed in several initial conditions where the PID controller fails. The proposed controllers produced different trajectories than that of the PID controller. We presented three controller models with different output neuron settings: two output
Figure 9: Stability coverages of the initial states ($\theta, \dot{\theta}$) for Model 3 with 2, 4, 6, and 8 output neurons after training. A green circle indicates a success while a red 'x' indicates a failure for the corresponding initial state. (a) 2 output neurons (b) 4 output neurons (c) 6 output neurons (d) 8 output neurons
neurons with the same force magnitude (Model 1), 4 or more neurons with different force magnitude kernels (Model 2), and recurrently connected output neurons (Model 3). From the experiments, we observed that more neurons with diverse force magnitudes can learn larger ranges and are thus more flexible and robust. In particular, the 6 or 8 output neuron controller performs substantially better than the PID controller. Furthermore, we demonstrated that the recurrent network produces very sparse spike trains with firing rates as low as 1.99Hz which is more natural in biological systems as they signify higher energy efficiency. In future work, we plan to add a kernel for filtering the inputs and shall consider other control costs. The former can readily be added to the current controller keeping the derivation the same. The latter requires using a more general error function that includes the number of control spikes and other output statistics.

Acknowledgments

The authors thank the Air Force Office of Scientific Research (Grant FA9550-16-1-0135) for their generous support of this research.

References

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” Cognitive modeling, vol. 5, p. 3, 1988.

[2] F. Rieke, Spikes: exploring the neural code. MIT press, 1999.
[3] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. The MIT Press, 2005.

[4] S. M. Bohte, J. N. Kok, and H. La Poutre, “Error-backpropagation in temporally encoded networks of spiking neurons,” *Neurocomputing*, vol. 48, no. 1, pp. 17–37, 2002.

[5] R. V. Florian, “The chronotron: a neuron that learns to fire temporally precise spike patterns,” *PloS one*, vol. 7, no. 8, p. e40233, 2012.

[6] O. Booij and H. tat Nguyen, “A gradient descent rule for spiking neurons emitting multiple spikes,” *INFORMATION PROCESSING LETTERS*, vol. 95, no. 6, pp. 552–558, 2005.

[7] A. Bouganis and M. Shanahan, “Training a spiking neural network to control a 4-dof robotic arm based on spike timing-dependent plasticity,” in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. IEEE, 2010, pp. 1–8.

[8] E. M. Izhikevich et al., “Simple model of spiking neurons,” *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569–1572, 2003.

[9] S. Song, K. D. Miller, and L. F. Abbott, “Competitive hebbian learning through spike-timing-dependent synaptic plasticity,” *Nature neuroscience*, vol. 3, no. 9, pp. 919–926, 2000.

[10] A. L. Hodgkin and A. F. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of physiology*, vol. 117, no. 4, pp. 500–544, 1952.
[11] W. Gerstner and W. Kistler, *Spiking Neuron Models: An Introduction*. New York, NY, USA: Cambridge University Press, 2002.

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

[13] R. Jolivet, T. J. Lewis, and W. Gerstner, “Generalized integrate-and-fire models of neuronal activity approximate spike trains of a detailed model to a high degree of accuracy,” *Journal of neurophysiology*, vol. 92, no. 2, pp. 959–976, 2004.

[14] N. Minorsky, “Directional stability of automatically steered bodies,” *Journal of the American Society for Naval Engineers*, vol. 34, no. 2, pp. 280–309, 1922.

[15] N. Frémaux, H. Sprekeler, and W. Gerstner, “Reinforcement learning using a continuous time actor-critic framework with spiking neurons,” *PLoS computational biology*, vol. 9, no. 4, p. e1003024, 2013.

[16] G. Hennequin, T. P. Vogels, and W. Gerstner, “Optimal control of transient dynamics in balanced networks supports generation of complex movements,” *Neuron*, vol. 82, no. 6, pp. 1394–1406, 2014.

[17] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” *Systems, Man and Cybernetics, IEEE Transactions on*, vol. Sep, no. 5, pp. 834–846, 1983.

[18] C. W. Anderson, “Strategy learning with multilayer connectionist representations,” in *Proceedings of the Fourth International Workshop on Machine Learning*, 1987, pp. 103–114.
[19] L. Wiklendt, “Spiking neural networks for robot locomotion control,” PhD thesis, University of Newcastle. Faculty of Engineering & Built Environment, School of Electrical Engineering and Computer Science, 2014. [Online]. Available: http://hdl.handle.net/1959.13/1049196

[20] N. Takase, J. Botzheim, and N. Kubota, “Evolving spiking neural network for robot locomotion generation,” in Evolutionary Computation (CEC), 2015 IEEE Congress on, May 2015, pp. 558–565.

[21] A. Banerjee, “Learning precise spike train-to-spike train transformations in multilayer feedforward neuronal networks,” Neural Comput., vol. 28, no. 5, pp. 826–848, May 2016. [Online]. Available: http://dx.doi.org/10.1162/NECO_a_00829

[22] ——, “On the phase-space dynamics of systems of spiking neurons. i: Model and experiments,” Neural Computation, vol. 13, no. 1, pp. 161–193, 2001.

[23] V. Ramaswamy and A. Banerjee, “Connectomic constraints on computation in feedforward networks of spiking neurons,” J. Comput. Neurosci., vol. 37, no. 2, pp. 209–228, Oct. 2014. [Online]. Available: http://dx.doi.org/10.1007/s10827-014-0497-5

[24] R. Güttig and H. Sompolinsky, “The tempotron: a neuron that learns spike timing-based decisions.” Nature Neuroscience, vol. 9, no. 5, pp. 420–8, 2006.

[25] F. Ponulak and A. Kasinski, “Supervised learning in spiking neural networks with
resume: sequence learning, classification, and spike shifting,” *Neural Computation*, vol. 22, no. 2, pp. 467–510, 2010.

[26] I. Sporea and A. Grüning, “Supervised learning in multilayer spiking neural networks,” *Neural computation*, vol. 25, no. 2, pp. 473–509, 2013.

[27] J. D. Victor and K. P. Purpura, “Nature and precision of temporal coding in visual cortex: a metric-space analysis,” *Journal of neurophysiology*, vol. 76, no. 2, pp. 1310–1326, 1996.

[28] M. C. van Rossum, “A novel spike distance,” *Neural Computation*, vol. 13, no. 4, pp. 751–763, 2001.

[29] S. Schreiber, J. Fellous, D. Whitmer, P. Tiesinga, and T. J. Sejnowski, “A new correlation-based measure of spike timing reliability,” *Neurocomputing*, vol. 52, pp. 925–931, 2003.

[30] T. S. Kang and A. Banerjee, “Learning deterministic spiking neuron feedback controllers,” in *Neural Networks (IJCNN), 2017 International Joint Conference on*. IEEE, 2017, pp. 2443–2450.

[31] C. W. Anderson, “Code for learning to balance a pole,” [http://www.cs.colostate.edu/~anderson/code/poledemo.tar.gz](http://www.cs.colostate.edu/~anderson/code/poledemo.tar.gz), 1988, accessed: 2017-07-13.

[32] J. ZIEGLER and N. NICHOLS, “Optimum settings for automatic controllers,” *Transactions*, vol. 760, 1942.