International Journal of Neural Systems, Vol. 24, No. 1 (2014) 1450002 (16 pages)
© World Scientific Publishing Company
DOI: 10.1142/S0129065714500026

CODE-SPECIFIC LEARNING RULES IMPROVE ACTION SELECTION BY POPULATIONS OF SPIKING NEURONS

JOHANNES FRIEDRICH*
Institute of Physiology, University of Bern
Bühlplatz 5, 3012 Bern, Switzerland

Computational and Biological Learning Lab
Department of Engineering, University of Cambridge
Trumpington Street, Cambridge CB2 1PZ, UK
jf517@cam.ac.uk

ROBERT URBANCZIK[†] and WALTER SENN[‡]
Institute of Physiology, University of Bern
Bühlplatz 5, 3012 Bern, Switzerland

Center for Cognition, Learning and Memory
University of Bern, Factory Street 8
CH-3012 Bern, Switzerland
[†]*urbanczik@pyl.unibe.ch*
[‡]*senn@pyl.unibe.ch*

Accepted 26 September 2013
Published Online

Population coding is widely regarded as a key mechanism for achieving reliable behavioral decisions. We previously introduced reinforcement learning for population-based decision making by spiking neurons. Here we generalize population reinforcement learning to spike-based plasticity rules that take account of the postsynaptic neural code. We consider spike/no-spike, spike count and spike latency codes. The multi-valued and continuous-valued features in the postsynaptic code allow for a generalization of binary decision making to multi-valued decision making and continuous-valued action selection. We show that code-specific learning rules speed up learning both for the discrete classification and the continuous regression tasks. The suggested learning rules also speed up with increasing population size as opposed to standard reinforcement learning rules. Continuous action selection is further shown to explain realistic learning speeds in the Morris water maze. Finally, we introduce the concept of action perturbation as opposed to the classical weight- or node-perturbation as an exploration mechanism underlying reinforcement learning. Exploration in the action space greatly increases the speed of learning as compared to exploration in the neuron or weight space.

Keywords: Spiking neural network; population coding; reinforcement learning; decision making; action perturbation; spike-timing-dependent synaptic plasticity; code specificity; policy gradient.

1. Introduction

Experimental data show that the sign of synaptic modifications can depend on the timing of pre- and postsynaptic spikes in the range of milliseconds.[1,2] But when linking synaptic plasticity to learning and behavior as in motor control, the relevant time scale may easily become tens or hundreds of milliseconds. In fact, behaviorally relevant information is often

*Corresponding author.

known to be encoded in neuronal firing rates rather than in precise spike times.[3,4] This raises the question whether spike-timing-dependent plasticity[5] is in any case the appropriate means to adequately change synaptic strengths, or whether plasticity should not be governed by different quantities tighter related to the behavioral information.

Here we address this question from a theoretical point of view by considering synaptic plasticity within a spiking neural network[6] (SNN) designed to incorporate behavioral decisions or actions. Behavioral responses are recognized to be encoded in a population of neurons coding for the same quantity.[7,8] The postsynaptic responses of the population neurons are aggregated by a decision-making circuit. In contrast to previous work,[9,10] we consider here a probabilistic read-out stage to facilitate exploration and denote this stochasticity as action perturbation. Yet, while population coding increases the reliability of the behavioral decisions, it has also been shown that reinforcement learning based on standard spike-timing-dependent synaptic plasticity slows down with increasing population size.[9] To overcome this problem it was suggested that the decision made by the population should be taken into account for modifying individual synapses targeting the population neurons.[9] This decision-dependent modulation of synaptic plasticity intrinsically incorporates information about the involved neuronal code. We show how to derive different learning rules for population coding which are specific to the code used for decision making. The concept is applied to a spike/no-spike, a spike count and a latency code. In contrast to existing population learning,[9–11] the rules apply not only to binary decisions, but also to decision making with multiple alternatives or to action selection in a continuous spectrum.

The learning rules are deduced by differently estimating the stochastic gradient of the expected reward. Though other optimization methods are possible,[12] the training of neural networks by gradient-based methods has been especially fruitful. Our introduction of action perturbation allows learning to proceed along the gradient of the expected reward instead of optimizing some other, though related, objective function.[9] While code-specificity in the overall learning rule naturally enters as a modulation term formed by the population decision, the base synaptic plasticity kernel which takes only pre- and postsynaptic quantities into account may itself be code-specific or not. It was previously suggested that for a single decision-making neuron the code-specificity of this base plasticity kernel should improve learning speed, but the claim could not yet be substantiated in simulations.[13] We discuss advantages and disadvantages of the code-specificity in terms of the modularity of base plasticity rules to be incorporated in population learning.

2. Models and Methods

2.1. *Population coding and policy gradient*

The basic setup in population coding is the following: each neuron ν in the population of size N receives a presynaptic stimulus X^ν of duration T and generates a stochastic output spike train Y^ν. For a summary of the notation used throughout the manuscript we would like to refer the reader to Table 1. The neurons need not have exactly the same input, but we assume that the X^ν are highly correlated. The lack of connectivity between the population neurons avoids issues relating to population spike correlations.[8] The decision readout circuitry aggregates the output spike trains into a decision D based on how

Table 1. Notation used throughout the manuscript.

ν	Neuron index
N	Population size
T	Stimulus duration
$\mathbf{X}$	Input spike pattern
X^ν	Input spike train of neuron ν
$\mathbf{Y}$	Output spike trains of all neurons
Y^ν/Y	Output spike train of neuron ν
$\mathbf{W}$	Weight matrix
$\mathbf{w}^\nu/\mathbf{w}$	Weight vector of neuron ν
$\mathbf{f}$	Feature vector of all neurons
f^ν/f	Feature of neuron ν
$\mathbf{f}^\setminus$	Feature vector of all other neurons
A_i	Activity of population i
$\bar{f}_i$	Feature average of population i
$D/\mathbf{D}$	Scalar/vectorial decision
R	Reward
η	Learning rate
$\mathbf{psp}$	Postsynaptic potentials for unity weights
u	Membrane potential
ϕ	Escape rate/instantaneous firing rate
$\mu_\mathbf{w}$	Expected number of output spikes
ϑ	Scalar target value for feature expansion

neurons encode information. Our central assumption is that the decision does not depend on the full output spike trains (list of spike times), but only on their coding features, e.g. the number of spikes (length of the list) or the latency of the first spike (first entry in the sorted list). We present throughout the manuscript how this knowledge about the neural code can be exploited to yield fast learning rules. We adopt a general approach and characterize each spike train by its coding feature f^ν, which assigns a numerical value to any spike train Y^ν. To facilitate exploration the decision making is stochastic, which we denote as action perturbation. Depending on the input and the decision the system receives reward. During learning the synaptic weights $\mathbf{W}$ of the population neurons are adjusted in order to maximize the expected reward.

Let $P_{\mathbf{W}}(\mathbf{Y}\,|\,\mathbf{X})$ be the probability density that a population with weight vectors $\mathbf{W} = (\mathbf{w}^1, \ldots, \mathbf{w}^N)$ generates the output spike trains $\mathbf{Y} = (Y^1, \ldots, Y^N)$ in response to the stimulus $\mathbf{X}$. Conditioned on the stimulus, the neurons are independent, so we have

$$P_{\mathbf{W}}(\mathbf{Y}\,|\,\mathbf{X}) = \prod_{\nu=1}^{N} P_{\mathbf{w}^\nu}(Y^\nu\,|\,\mathbf{X}). \tag{1}$$

Let $P(D\,|\,\mathbf{f}(\mathbf{Y}))$ be the probability of having the population decision D for a given $\mathbf{f} = (f^1, \ldots, f^N)$, where f^ν is the output feature of neuron ν, which is just a function of Y^ν. For sake of concise notation we skip the functional dependence and write f^ν instead of $f(Y^\nu)$. We use in the following the notation for discrete D, but actions can be continuous valued too and sums over D must merely be replaced by integrals. The population decision yields the reward $R(D, \mathbf{X})$. We want to maximize the expected reward,

$$\begin{aligned}\langle R\rangle &= \sum_D \int \mathrm{d}\mathbf{Y}\,\mathrm{d}\mathbf{X}\,R(D,\mathbf{X})P(D,\mathbf{Y},\mathbf{X})\\ &= \sum_D \int \mathrm{d}\mathbf{Y}\,\mathrm{d}\mathbf{X}\,R(D,\mathbf{X})P(D\,|\,\mathbf{f})\\ &\quad\times P_{\mathbf{W}}(\mathbf{Y}\,|\,\mathbf{X})P(\mathbf{X}),\end{aligned} \tag{2}$$

where we rewrote the joint distribution $P(D, \mathbf{Y}, \mathbf{X})$ using that D only depends on the features $\mathbf{f}$ and not the full spike trains $\mathbf{Y}$ and its conditional independence of $\mathbf{X}$ given $\mathbf{f}$. We calculate the gradient $\nabla_\nu \langle R\rangle$ with respect to the weight vector $\mathbf{w}^\nu$ of neuron ν, i.e. the vector of partial derivatives with respect to all afferent weights of neuron ν.

In policy gradient learning[14] one estimates the gradient by Monte Carlo sampling. Since one also samples over the stimuli, it suffices to focus on a single stimulus, suppressing the dependence on X. With this notation the straightforward approach is to sample the gradient, the components of which can be expressed using (1),

$$\begin{aligned}\nabla_\nu \langle R\rangle &= \sum_D R \int \mathrm{d}\mathbf{Y}\,P(D\,|\,\mathbf{f})P_{\mathbf{W}}(\mathbf{Y})\nabla_\nu \ln P_{\mathbf{w}^\nu}(Y^\nu)\\ &= \langle R\,\nabla_\nu \ln P_{\mathbf{w}^\nu}(Y^\nu)\rangle,\end{aligned} \tag{3}$$

where we used the identity $\nabla_\nu P_{\mathbf{w}^\nu}(Y^\nu) = P_{\mathbf{w}^\nu}(Y^\nu)\times \nabla_\nu \ln P_{\mathbf{w}^\nu}(Y^\nu)$, which is often referred to as the 'log-trick'. This yields the standard learning rule[15,16]

$$\Delta\mathbf{w}^\nu = \eta R \nabla_\nu \ln P_{\mathbf{w}^\nu}(Y^\nu) \tag{4}$$

with some positive learning rate η. The average weight change is in the direction of the gradient, the rule performs stochastic gradient ascent in the expected reward. It makes no assumptions about the code used in reading out the population. Learning speed deteriorates with increasing N for this rule,[9] thus we consider it just for the sake of comparison in this paper, but the expression $\nabla_\nu \ln P_{\mathbf{w}^\nu}(Y^\nu)$, known as characteristic eligibility in reinforcement learning,[14] constitutes a base synaptic plasticity kernel that will reappear in the derivation of the gradient rules for population learning. Note that the standard learning rule uses stochasticity in the output spike trains to estimate the reward gradient, thus implementing node perturbation.[17] For fluctuations that are positively correlated with reward, the weights are adapted to increase the probability to reproduce this spike train pattern.

Though it is not a remedy for the performance degradation with population size, it is still noteworthy that Eq. (3) can be slightly modified by subtracting a reinforcement baseline from the reward, which is conditionally independent of Y^ν given $\mathbf{w}^\nu$ and $\mathbf{X}$,[14] to reduce the variance of the updates while keeping their mean. A common choice is to subtract the stimulus-specific mean reward obtained as running average over past rewards.[18] This replacement of the reward by the reward prediction error can also be done in all rules we present. In the examples we choose reward values balanced around zero and expect this modification would increase performance only marginally.

2.2. *Weakly code-specific rules*

We denote rules where code-specificity enters only on the neural level through modulatory factors as "weakly code-specific", in contrast to "tightly code-specific" rules that are code-specific on the synaptic level.

2.2.1. *First-order expansion*

The whole point of population coding is to exploit redundancy. In other words the influence of any single neuron feature f^ν on $P(D\,|\,\mathbf{f})$ should be small in a large population. While details will depend on the specific read-out, in all of the architectures we consider below, the influence of any single neuron on $P(D\,|\,\mathbf{f})$ decays with population size as $1/\sqrt{N}$, to keep the variance of the population activity invariant under a change in population size, or even as $1/N$, such that the mean of the population activity is size invariant. Considerations for the square root decay hold for the linear decay too and we conservatively assume a decay with $1/\sqrt{N}$ in the following theoretical derivations. A common choice will be that the decision depends only on the normalized sum of features $A = \sum_\mu f^\mu/\sqrt{N}$, hence by the chain rule each successive derivation of $P(D\,|\,A(\mathbf{f}))$ with respect to f^ν contributes another factor $\frac{\partial A}{\partial f^\nu} = 1/\sqrt{N}$ and the nth derivative is of order $\mathcal{O}(N^{-\frac{n}{2}})$. In particular is the second derivative of order $\mathcal{O}(\frac{1}{N})$. For example a binary decision task ($D = \pm 1$) with $P(D\,|\,A(\mathbf{f})) = \frac{1}{2}(1 + \tanh(AD))$ has derivatives $\frac{\partial P(D\,|\,A(\mathbf{f}))}{\partial f} = \frac{D}{2\sqrt{N}}\cosh^{-2}(A)$ and $\frac{\partial^2 P(D\,|\,A(\mathbf{f}))}{\partial f^2} = -\frac{D}{N}\tanh(A)\cosh^{-2}(A)$. Due to this, the influence of any single neuron is well described by linearization. To lighten the notation in the following we focus on one neuron (the expressions for the other neurons being entirely analogous) and skip the neuron index ν. We denote the feature vector of all other neurons by $\mathbf{f}^\setminus$. Linearizing we have:

$$\begin{aligned} P(D\,|\,\mathbf{f}) &= P(D\,|\,\mathbf{f}^\setminus, f) \\ &= P(D\,|\,\mathbf{f}^\setminus, \vartheta) + (f - \vartheta)\frac{\partial}{\partial f}P(D\,|\,\mathbf{f}^\setminus, \vartheta) \\ &\quad + \mathcal{O}\left(\frac{1}{N}\right), \end{aligned} \tag{5}$$

where $P(D\,|\,\mathbf{f}^\setminus, \vartheta)$ would be the decision probability if the considered neuron had produced the output feature ϑ, i.e. $\frac{1}{2}(1 + D\tanh(A + (\vartheta - f)/\sqrt{N}))$ in our example, and $\frac{\partial}{\partial f}P(D\,|\,\mathbf{f}^\setminus, \vartheta)$ is the derivative of $P(D\,|\,\mathbf{f})$ with respect to f evaluated at the value ϑ, e.g. $\frac{D}{2\sqrt{N}}\cosh^{-2}(A + (\vartheta - f)/\sqrt{N})$. Here the exact value of the $\mathcal{O}(\frac{1}{N})$ term will depend on the choice of the parameter ϑ and one will want to choose ϑ such that $|f - \vartheta|$ tends to be small for the feature values encountered during learning.

When plugging this into Eq. (3) a term arises containing $P(D\,|\,\mathbf{f}^\setminus, \vartheta)$. But since it does not depend on the output of neuron ν, and since $\int \mathrm{d}Y P_\mathbf{w}(Y)\nabla \ln P_\mathbf{w}(Y) = \nabla \int \mathrm{d}Y P_\mathbf{w}(Y) = 0$, the term vanishes. Hence up to $\mathcal{O}(\frac{1}{N})$ corrections:

$$\begin{aligned} \nabla\langle R\rangle = \sum_D R \int \mathrm{d}\mathbf{Y}\, P_\mathbf{W}(\mathbf{Y}) \frac{\partial P(D\,|\,\mathbf{f}^\setminus, \vartheta)}{\partial f} \\ \times (f - \vartheta)\nabla \ln P_\mathbf{w}(Y). \end{aligned} \tag{6}$$

For the read-outs we use below, not only $P(D\,|\,\mathbf{f})$ but also its partial derivatives depend only weakly on any single neuron and, in particular, we have:

$$\frac{\partial}{\partial f}P(D\,|\,\mathbf{f}) = \frac{\partial}{\partial f}P(D\,|\,\mathbf{f}^\setminus, \vartheta) + \mathcal{O}\left(\frac{1}{N}\right). \tag{7}$$

Note that $\frac{\partial}{\partial f}P(D\,|\,\mathbf{f})$ is itself of order $\mathcal{O}(1/\sqrt{N})$. The term $P(D\,|\,\mathbf{f}^\setminus, \vartheta)$ of order $\mathcal{O}(1)$ vanished, we take the terms of order $\mathcal{O}(1/\sqrt{N})$ into account, and we can neglect higher orders. Hence up to $\mathcal{O}(\frac{1}{N})$ corrections:

$$\begin{aligned} \nabla\langle R\rangle &= \sum_D R \int \mathrm{d}\mathbf{Y}\, P_\mathbf{W}(\mathbf{Y}) \frac{\partial P(D\,|\,\mathbf{f})}{\partial f} \\ &\quad \times (f - \vartheta)\nabla \ln P_\mathbf{w}(Y) \\ &= \sum_D R \int \mathrm{d}\mathbf{Y}\, P_\mathbf{W}(\mathbf{Y}) P(D\,|\,\mathbf{f}) \frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f} \\ &\quad \times (f - \vartheta)\nabla \ln P_\mathbf{w}(Y). \end{aligned}$$

The ensuing update rule

$$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f}(f - \vartheta)\nabla \ln P_\mathbf{w}(Y) \tag{8}$$

has a structure similar to the standard learning rule given by Eq. (4), but now the reward signal is remodulated by feedback from the decision making and the feature actually produced by the neuron. The decision signal $\frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f}$ is of order $\mathcal{O}(1/\sqrt{N})$ for the population readouts considered below and the feature value f is of order $\mathcal{O}(1)$, hence the variance of the obtained gradient estimate is about a factor of N smaller than the standard estimate in Eq. (4).

2.2.2. *Exact linearization for binary features*

While above linearization is an approximation that increases in accuracy for large population size, an exact linearization is possible even for small population sizes, if the coding feature takes on only two values, denoted as f_0 and f_1, because one can always draw an exact line through two points. An example of such a binary feature is the spike/no-spike code considered later in Sec. 3.1.1. We describe the line through $P(D\,|\,\mathbf{f}^{\setminus}, f_0)$ and $P(D\,|\,\mathbf{f}^{\setminus}, f_1)$ by one point on the line and its slope. The point has coordinates $(\mu f_1 + (1-\mu)f_0, \mathrm{S}^{\mu}P(D\,|\,\mathbf{f}^{\setminus}))$, and the slope is $\frac{DP(D\,|\,\mathbf{f}^{\setminus})}{f_1-f_0}$, using the definitions

$$\mathrm{S}^{\mu}P(D\,|\,\mathbf{f}^{\setminus}) := \mu P(D\,|\,\mathbf{f}^{\setminus}, f_1) + (1-\mu)P(D\,|\,\mathbf{f}^{\setminus}, f_0),$$

$$DP(D\,|\,\mathbf{f}^{\setminus}) := P(D\,|\,\mathbf{f}^{\setminus}, f_1) - P(D\,|\,\mathbf{f}^{\setminus}, f_0),$$

where μ is a free parameter that does not depend on f, but is otherwise arbitrary. The linear function is only evaluated at f_0 and f_1, yielding

$$P(D\,|\,\mathbf{f}^{\setminus}, f) = \mathrm{S}^{\mu}P(D\,|\,\mathbf{f}^{\setminus}) + (\delta_{f_1 f} - \mu)DP(D\,|\,\mathbf{f}^{\setminus}),$$

where δ is the Kronecker delta, as can be verified by inserting the definitions for $\mathrm{S}^{\mu}P(D\,|\,\mathbf{f}^{\setminus})$ and $DP(D\,|\,\mathbf{f}^{\setminus})$ together with f_0 or f_1.

When plugging this into Eq. (3) the term containing $\mathrm{S}^{\mu}P(D\,|\,\mathbf{f}^{\setminus})$ vanishes, because it does not depend on the neuron for which the gradient is calculated.

$$\nabla\langle R\rangle = \sum_D R \int \mathrm{d}\mathbf{Y}\, P_{\mathbf{W}}(\mathbf{Y})P(D\,|\,\mathbf{f})\frac{DP(D\,|\,\mathbf{f}^{\setminus})}{P(D\,|\,\mathbf{f})} \times (\delta_{f_1 f} - \mu)\nabla \ln P_{\mathbf{w}}(Y). \tag{9}$$

In order to write it in a form that can be sampled, we had to introduce the denominator $P(D\,|\,\mathbf{f})$. This term is problematic, because rare decisions D can potentially lead to large values of the gradient estimator, as has been pointed out.[19] We follow the suggestion to choose

$$\mu = \frac{P(D\,|\,\mathbf{f}^{\setminus}, f_0)}{\mathrm{S}P(D\,|\,\mathbf{f}^{\setminus})}$$

with

$$\mathrm{S}P(D\,|\,\mathbf{f}^{\setminus}) = P(D\,|\,\mathbf{f}^{\setminus}, f_1) + P(D\,|\,\mathbf{f}^{\setminus}, f_0),$$

which circumvents a small denominator and yields the update rule

$$\Delta\mathbf{w} = \eta R \frac{DP(D\,|\,\mathbf{f}^{\setminus})}{\mathrm{S}P(D\,|\,\mathbf{f}^{\setminus})}(2\delta_{f_1 f} - 1)\nabla \ln P_{\mathbf{w}}(Y). \tag{10}$$

Compared to the standard learning rule given by Eq. (4) an additional factor $\frac{DP(D\,|\,\mathbf{f}^{\setminus})}{SP(D\,|\,\mathbf{f}^{\setminus})}$ arises. Its absolute value is smaller than 1, because the sum of two positive numbers is bigger than the difference, hence resulting in an estimate of lower variance than the standard one in Eq. (4).

2.3. *Tightly code-specific rules*

In above rules the feature itself entered the learning rule introducing code-specificity on the neural level, whereas on the synaptic level they share the base plasticity kernel $\nabla \ln P_{\mathbf{w}}(Y)$. We can further improve the gradient estimator when code-specificity enters not only on the neural but on the synaptic level. These rules are hence denoted as "tightly code-specific".

Pulling all terms in Eq. (6) that do not depend on the considered neuron outside of the integral over Y (note that $P_{\mathbf{W}}(\mathbf{Y})$ factorizes), we obtain

$$\nabla\langle R\rangle_{D,\mathbf{Y}} = \sum_D R \int \mathrm{d}\mathbf{Y}^{\setminus}\, P_{\mathbf{W}}(\mathbf{Y}^{\setminus})\frac{\partial P(D\,|\,\mathbf{f}^{\setminus}, \vartheta)}{\partial f} \times \int \mathrm{d}Y P_{\mathbf{w}}(Y)\underbrace{(f-\vartheta)\nabla \ln P_{\mathbf{w}}(Y)}_{g}. \tag{11}$$

For the last integral we have $\langle g\rangle_Y = \langle\langle g\rangle_{Y|f}\rangle_f$. The index of the angle brackets indicates the distribution over which the average is calculated and is suppressed throughout the manuscript if obvious to avoid notational clutter. Partial averaging with respect to $P(Y\,|\,f)$ yields

$$\begin{aligned}
&\langle (f-\vartheta)\nabla \ln P_{\mathbf{w}}(Y)\rangle_{Y|f} \\
&\quad = (f-\vartheta)\int \mathrm{d}Y P_{\mathbf{w}}(Y\,|\,f)\nabla \ln P_{\mathbf{w}}(Y) \\
&\quad = (f-\vartheta)\int \mathrm{d}Y \frac{P(f\,|\,Y)}{P_{\mathbf{w}}(f)}\nabla P_{\mathbf{w}}(Y) \\
&\quad = (f-\vartheta)\frac{1}{P_{\mathbf{w}}(f)}\nabla \int \mathrm{d}Y P(f\,|\,Y)P_{\mathbf{w}}(Y) \\
&\quad = (f-\vartheta)\nabla \ln P_{\mathbf{w}}(f),
\end{aligned}$$

where we used Bayes' rule in the second line. Hence, using Eq. (7), we obtain a first tightly code-specific plasticity rule

$$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f}(f-\vartheta)\nabla \ln P_{\mathbf{w}}(f), \tag{12}$$

which has a reduced variance compared to Eq. (8) due to analytical averaging. It amounts to replacing the log-likelihood of the output spike train by the log-likelihood of the feature, as has been done for single neurons.[13] This replacement of the base synaptic plasticity kernel $\nabla \ln P_{\mathbf{w}}(Y)$ by $\nabla \ln P_{\mathbf{w}}(f)$ introduces code-specificity also on the synaptic level.

Further averaging with respect to $P_{\mathbf{w}}(f)$ yields

$$\int \mathrm{d}f P_{\mathbf{w}}(f)(f-\vartheta)\nabla \ln P_{\mathbf{w}}(f) = \nabla\langle f\rangle,$$

where we assumed continuous features and used again that $\int \mathrm{d}f P_{\mathbf{w}}(f)\nabla \ln P_{\mathbf{w}}(f) = \nabla \int \mathrm{d}f P_{\mathbf{w}}(f) = 0$. For discrete features the integral merely has to be replaced by a sum and the probability density function by the probability mass function. For some codes we can calculate the gradient of the feature mean $\langle f\rangle$ analytically instead of leaving it to the sampling procedure. Using this result, Eq. (7) and the 'log-trick' to write Eq. (11) in a form suited for sampling leads to a second tightly code-specific update rule,

$$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f}\nabla\langle f\rangle, \tag{13}$$

which has even less variance.

$(f-\vartheta)\nabla \ln P_{\mathbf{w}}(f)$ is an unbiased estimate for the gradient $\nabla\langle f\rangle$ obtained by the well-known 'log-trick'. We propose another unbiased estimate of the gradient, obtained by partial integration. Denoting the cumulative probability distribution by $C_{\mathbf{w}}(f)$ we have

$$\begin{aligned}\nabla\langle f\rangle &= \nabla \int \mathrm{d}f P_{\mathbf{w}}(f) f = 0 - \nabla \int \mathrm{d}f C_{\mathbf{w}}(f)\\ &= -\int \mathrm{d}f P_{\mathbf{w}}(f)\frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)} = \left\langle -\frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}\right\rangle.\end{aligned}$$

Note that the derivative of the 'surface term' vanishes, because at the integration boundaries $C_{\mathbf{w}}$ is zero and one respectively independent of $\mathbf{w}$. This results in a third tightly code-specific update rule

$$\Delta\mathbf{w} = -\eta R \frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f}\frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}. \tag{14}$$

As shown in Appendix A this is even an exact policy gradient rule. Convex combinations of the derived estimators yield again valid estimators, in particular could one also add a term of the form $\vartheta\nabla \ln P_{\mathbf{w}}(f)$ to the estimate obtained by partial integration. An example for a tightly code-specific rule obtained by partial integration is the latency code presented in Sec. 3.1.3.

The update (13) has the smallest variance and is the preferred rule if an analytical expression for the gradient of the feature mean $\nabla\langle f\rangle$ exists, otherwise it is not obvious whether the log-trick estimator or the newly proposed partial integration estimator has lower variance. We find that for a broad class of probability functions the latter is the case (e.g. $P_{\mathbf{w}}(f) \propto f^a \exp(-f^b h(\mathbf{w}))$ with $f, a \geq 0$, $b > 0$ and h any different positive function), though not generally because rather exotic exceptions exist (e.g. $P_{\mathbf{w}}(f) \propto f^a \exp(-f^b h(\mathbf{w}))$ with $a < 0$ and large enough b).

We derived tightly code-specific rules by analytical averaging, which guarantees gradient estimates of lower variance compared to weakly code-specific rules. While the derivation by isolating a single neuron's contribution is intuitive, it relies on an approximation valid for large population sizes. In the appendix we show that instead of utilizing integration by parts in the last step to derive Eq. (14), we can directly start with partial integration and Eq. (14) turns out to be even exact. In Appendix A we also present the resulting update rule in the case of discrete features.

A summary of the derived rules is shown in Table 2 for the case of large population sizes, i.e. even for discrete features the difference $DP(D\,|\,\mathbf{f})$ is replaced by the differential, giving rise to the term $\frac{\partial \ln P(D\,|\,\mathbf{f})}{\partial f}$ which all rules have in common. These are the rules we used in the simulations presented in the results section.

2.4. *Single neuron model*

We use simple Poisson-type neurons where the postsynaptic firing rate is given by a nonlinear function $\phi(u)$ of the membrane potential u. The membrane potential at time t is:

$$\begin{aligned}u(t) &= u_0 + \sum_i w_i \sum_{s\in X_i}\epsilon(t-s)\\ &= u_0 + \mathbf{w}^T\mathbf{psp}(t).\end{aligned} \tag{15}$$

Here u_0 is the resting potential and $\epsilon(t)$ denotes the shape of the postsynaptic potential evoked by a single presynaptic spike at time 0. For future use, we have introduced $\mathbf{psp}(t)$ as the postsynaptic potentials that would be evoked if the synaptic weights were unity. As has been shown[15,16,20] the derivative of the log-likelihood of actually producing the output

Table 2. Summary of the derived learning rules for continuous features with arbitrary population size or for discrete features in the limit of large population size. The performance increases (typically) from top to bottom.

Code-specificity	Learning rule	Derivation
Weak	$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,\vert\,\mathbf{f})}{\partial f}(f - \vartheta)\nabla \ln P_{\mathbf{w}}(Y)$	Linear expansion
Tight	$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,\vert\,\mathbf{f})}{\partial f}(f - \vartheta)\nabla \ln P_{\mathbf{w}}(f)$	Averaging w.r.t. $P_{\mathbf{w}}(Y\,\vert\,f)$
	$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,\vert\,\mathbf{f})}{\partial f}\frac{-\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}$	Partial integration
	$\Delta\mathbf{w} = \eta R \frac{\partial \ln P(D\,\vert\,\mathbf{f})}{\partial f}\nabla\langle f\rangle$	Averaging w.r.t. $P_{\mathbf{w}}(Y)$

spike train Y, known as likelihood ratio in classical statistics or as characteristic eligibility,[14] is given by

$$\nabla \ln P_{\mathbf{w}}(Y) = \int_0^T \mathrm{d}t \left(\sum_{s\in Y} \delta(t-s) - \phi(u(t)) \right) \times \frac{\phi'(u(t))}{\phi(u(t))}\mathbf{psp}(t). \tag{16}$$

This expression constitutes the base synaptic plasticity kernel of the weakly code-specific rules. It considers the full output spike train and enables to learn postsynaptic firing at one or several desired firing times,[15] a task that has attracted much research attention and various other rules have been suggested.[21]

3. Results

We illustrate the framework by deriving policy-gradient rules for classification and regression tasks as well as different neural codes. We then study their performance at simple computational tasks. In all simulations throughout the paper we use an exponential stochastic intensity $\phi(u) \propto \exp(u)$, a resting potential of $u_0 = -1$ (arbitrary units) and, if not mentioned otherwise, stimulus length $T = 500\,\mathrm{ms}$. The postsynaptic kernel is given by $\epsilon(t) = \frac{1}{\tau_M - \tau_S}(\mathrm{e}^{-t/\tau_M} - \mathrm{e}^{-t/\tau_S})$, where $\tau_M = 10\,\mathrm{ms}$ is used for the membrane time constant and $\tau_S = 1.4\,\mathrm{ms}$ for the synaptic time constant. We assume that each population neuron synapses onto a site in the input layer with probability of 80%. Initial values for the synaptic strength were picked from a Gaussian distribution with mean 1 and standard deviation equal to 2.5, independently for each afferent and each neuron. Further the learning rate η was chosen optimally for each population size. For each scenario we low pass filtered the reward with a time constant of 50 trials. We averaged over 10 simulated runs and the figures show the mean (symbols) and its SEM (errorbars).

3.1. *Classification tasks*

For the decision making we assume binary decisions $D = \pm 1$. To start with unbiased initial conditions we consider the difference activity between two populations, cf. Fig. 1. Each of the two populations Pop$_i$, $i = 1, 2$ consists of N neurons with summed up and normalized population activity $A_i \propto \frac{1}{\sqrt{N}}\sum_\nu f_i^\nu$. The normalization is proportional to $\frac{1}{\sqrt{N}}$, reflecting the fact that, given the stimulus, neuronal responses are conditionally independent. The decision is chosen stochastically with $P(D\,|\,\mathbf{f}) = P(D\,|\,A_1, A_2) = \frac{1}{1+\exp(-2(A_1-A_2)D)}$. The modulating population factor is thus $\frac{\partial \ln P(D\,|\,A_1,A_2)}{\partial A_{1/2}} = \pm(D - \tanh(A_1 - A_2))$, where the sign is positive/negative for neurons in Pop$_1$ and Pop$_2$, respectively. To be concise, in the

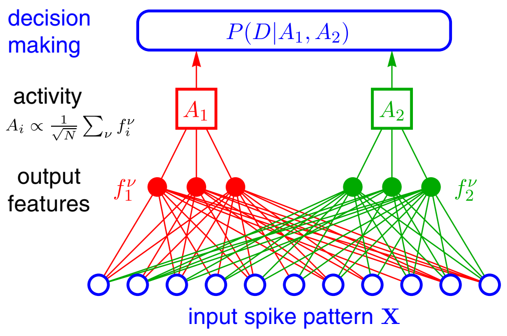


Fig. 1. Architecture of the spiking neural network for the stimulus classification task. The decision is stochastic with probability $P(D\,|\,A_1, A_2) = \frac{1}{1+\exp(-2(A_1-A_2)D)}$.

following we take the sign to be positive and hence present the plasticity rules for neurons in Pop_1. It is also possible to keep the weights of the second population fixed, so that it does not learn rather than learning with opposite sign.

On a side note, multi-class classifications are straight forwardly implementable in two ways. One approach is to use many binary classifiers, which gives rise to a multi-layer architecture as has already been presented elsewhere.[10] A second approach is to use one population for each of the k possible classes and use the generalization of the logistic function to multiple variables, known as softmax action selection, $P(D = j \,|\, A_1, A_2, \ldots, A_k) = \frac{\exp(A_j)}{\sum_{i=1}^{k} \exp(A_i)}$. This results in the simple expression $\frac{\partial \ln P(D=j \,|\, A_1, A_2, \ldots, A_k)}{\partial A_i} = \delta_{ij} - P(D = j \,|\, A_1, A_2, \ldots, A_k)$ for the population factor.

3.1.1. *Spike/no-spike code*

Let us first consider the case of a binary feature and assume the following spike/no-spike code: $f = f_0 = -1$ if the considered neuron does not spike within a given time window $[0, T]$, otherwise $f = f_1 = 1$. For the decision making we assume that the population readout sums up the features, $A_i = \frac{1}{\sqrt{N}} \sum_{\nu \in \mathrm{Pop}_i} f^\nu$. The reward is delivered at the end of this period. For this code an analytical expression for $\nabla \langle f \rangle$ exists. We derive it, using that the number of output spikes is Poisson distributed with mean $\mu_{\mathbf{w}}$, hence the probability of no spike is an exponential function of this mean:

$$P(f = f_0) = \mathrm{e}^{-\mu_{\mathbf{w}}} \quad \text{and} \quad P(f = f_1) = 1 - \mathrm{e}^{-\mu_{\mathbf{w}}},$$
$$\langle f \rangle = 1 - 2\mathrm{e}^{-\mu_{\mathbf{w}}},$$
$$\nabla \langle f \rangle = 2\mathrm{e}^{-\mu_{\mathbf{w}}} \nabla \mu_{\mathbf{w}}.$$

Plugging this into the learning rule Eq. (13), and absorbing the constant into the learning rate, yields the tightly code-specific rule:

$$\Delta \mathbf{w} = \eta R (D - \tanh(A_1 - A_2)) \frac{\nabla \mu_{\mathbf{w}}}{\exp(\mu_{\mathbf{w}})} \qquad (17)$$

with

$$\nabla \mu_{\mathbf{w}} = \int_0^T \mathrm{d}t\, \phi'(u(t)) \mathbf{psp}(t) \qquad (18)$$

and

$$\mu_{\mathbf{w}} = \int_0^T \mathrm{d}t\, \phi(u(t)), \qquad (19)$$

where $\mu_{\mathbf{w}}$ is the expected number of output spikes within time window $[0, T]$. Note that merely this expected value enters the learning rule, but not whether a postsynaptic spike was actually elicited or not. Plasticity depends only on the time course of the postsynaptic membrane potential but not on the spike timing.

We next turn to the weakly code-specific rule of Eq. (8) and set $\vartheta = 0$ to balance the case of spike ($f = 1$) and no spike ($f = -1$). This and Eq. (10) both yield for large population sizes

$$\Delta \mathbf{w} = \eta R (D - \tanh(A_1 - A_2)) f \nabla \ln P_{\mathbf{w}}(Y) \qquad (20)$$

with $\nabla \ln P_{\mathbf{w}}(Y)$ given in Eq. (16). We want to point out that the same update arises only due to our choice of the logistic function for $P(D \,|\, A_1, A_2)$. In the case of choosing e.g. the (shifted and scaled) error function as sigmoidal, the term DP/P in Eq. (9) can indeed become large. The same learning rule can also be derived by averaging over the two possible outcomes f_0 and f_1.[9]

We test the learning rules on the task of learning 10 stimulus-response associations. We use the same pattern statistics presented in Ref. 13, encoding the stimulus information in the rate pattern of 100 presynaptic neurons. For each stimulus, a different input pattern of duration $T = 500\,\mathrm{ms}$ is generated by drawing the firing rate of each input neuron independently from an exponential distribution with a mean of 10 Hz. In each trial, the input spike trains are generated anew from Poisson processes with these neuron- and stimulus-specific rates. Half the stimuli have a target label $+1$ and the other half -1. To learn the 10 prescribed stimulus–response associations, learning episodes are repeated with a randomly chosen stimulus–response pair used for each episode, immediately reinforced by $R = \pm 1$. Figures 2(a) and 2(d) illustrate that for the weakly code-specific rule (20) learning speeds up with increasing population size as opposed to standard reinforcement learning,[9] whereas the tightly code-specific rule (17) is rather insensitive to a change in population size. The tightly code-specific rule outperforms the weakly one, but for large population sizes the performances become similar and the benefit of using the tightly code-specific rule is marginal. Both learning rules are superior to the standard node perturbation rule equation (4), which is shown for the sake of comparison, too.

3.1.2. *Spike count code*

Next we assume that the coding feature f is the number of spikes within a given time window $[0, T]$. The probability distribution for the spike count is a Poisson distribution $P_{\mathbf{w}}(f) = \mu_{\mathbf{w}}^{f} \exp(-\mu_{\mathbf{w}})/f!$ with mean $\mu_{\mathbf{w}}$ as defined above in (18). For a Poisson distribution the term $-\frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}$ in (14) yields simply $\nabla\langle f\rangle = \nabla\mu_{\mathbf{w}}$. The correctly normalized activities are $A_i = \frac{1}{\sqrt{N\theta}} \sum_{\nu\in\text{Pop}_i} f^{\nu}$. Here θ is the mean number of output spikes per neuron during one stimulus. The ensuing tightly code-specific rule is

$$\Delta\mathbf{w} = \eta R(D - \tanh(A_1 - A_2))\nabla\mu_{\mathbf{w}} \qquad (21)$$

with $\nabla\mu_{\mathbf{w}}$ given in (19). Note that the plasticity is not spike-timing dependent. If the neuronal code is not spike-timing dependent, then neither is the tight plasticity rule.

Turning now toward the weakly code-specific rule (8) we expand around some target output activity θ. This results in

$$\Delta\mathbf{w} = \eta R(D - \tanh(A_1 - A_2))(f - \theta) \times \nabla \ln P_{\mathbf{w}}(Y) \qquad (22)$$

with $\nabla \ln P_{\mathbf{w}}(Y)$ given in (16). Here we obtain the standard update rule equation (4) with the two modulating factors $(D - \tanh(A))$ and $(f - \theta)$. The first one is again the attenuated decision signal. The second factor contains the behavior of the single neuron with respect to the decision boundary θ, thus its sign represents the decision of the single neuron. Taken together $R(D - \tanh(A))(f - \theta)$ constitutes an individual reward for the neuron. Note that the rule postulates a global modulatory signal of the form $R(D - \tanh(A))$ which is then combined with the local postsynaptic information $(f - \vartheta)$.

Figures 2(b) and 2(e) show simulation results for both newly derived update rules as well as for the standard node perturbation rule equation (4). For the weakly code-specific rule, we expanded around some target activity $\theta = 5$ corresponding to 10 Hz output rate. The tightly code-specific rule, outperforms the weakly code-specific one, and it quickly reaches perfect performance. Importantly, for both rules learning speeds up with population size (Fig. 2(e)), in contrast to node perturbation.

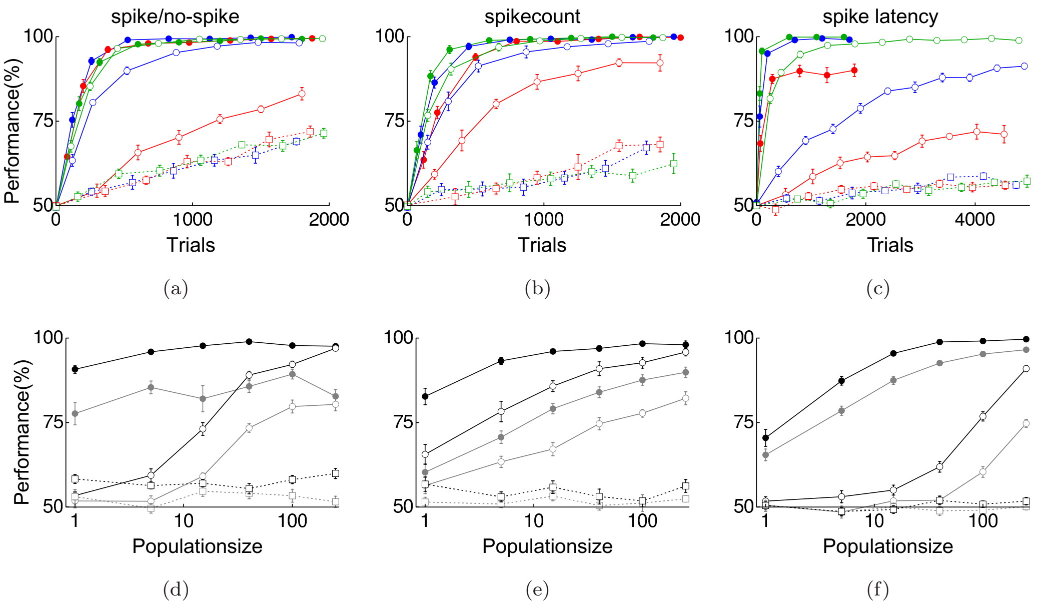


Fig. 2. (Color online) Simulation results for the classification task. Upper row: Learning curves for (a) spike/no-spike code, (b) spike count code and (c) spike latency code. Filled disks are used for the tightly code-specific rule, open circles for the weakly code-specific rule and open squares with dashed lines for the standard node perturbation rule (4). We used (17) and (20) for the spike/no-spike code, (21) and (22) for the spike count code, (23) and (24) for the spike latency code. The learning rules are summarized in Table 3, too. The population sizes used are: $N = 5$ (red), 40 (blue) and 250 (green). Lower row: Average reward after 200 (gray) and 500 (black) trials for (d) spike/no-spike code, (e) spike count code and (f) spike latency code.

Table 3. Summary of the derived classification learning rules for the specific codes. $\mathrm{GM} = \eta R(D - \tanh(A_1 - A_2))$ denotes the global modulatory signal. $\nabla \ln P_{\mathbf{w}}(Y) = \int_0^T \mathrm{d}t\, (\sum_{s\in Y} \delta(t-s) - \phi(u(t)))\frac{\phi'(u(t))}{\phi(u(t))}\mathbf{psp}(t)$, $\mu_{\mathbf{w}} = \int_0^T \mathrm{d}t\, \phi(u(t))$ and $\nabla\mu_{\mathbf{w}} = \int_0^T \mathrm{d}t\, \phi'(u(t))\mathbf{psp}(t)$ have already been introduced in (16), (18) and (19). The performance of the tightly is always superior to the weakly code-specific rule, cf. Fig. 2.

	Spike/no-spike	Spike count	Spike latency
Weak	$\Delta\mathbf{w} = \mathrm{GM} f \nabla \ln P_{\mathbf{w}}(Y)$	$\Delta\mathbf{w} = \mathrm{GM}(f-\theta)\nabla \ln P_{\mathbf{w}}(Y)$	$\Delta\mathbf{w} = \mathrm{GM}(\mathrm{e}^{-\frac{f}{\tau}} - \theta)\nabla \ln P_{\mathbf{w}}(Y)$
Tight	$\Delta\mathbf{w} = \mathrm{GM}\frac{\nabla\mu_{\mathbf{w}}}{\exp(\mu_{\mathbf{w}})}$	$\Delta\mathbf{w} = \mathrm{GM}\nabla\mu_{\mathbf{w}}$	$\Delta\mathbf{w} = \mathrm{GM}\mathrm{e}^{-\frac{f}{\tau}}\frac{\int_0^f \mathrm{d}t\,\phi'(u(t))\mathbf{psp}(t)}{\phi(u(f))}$

3.1.3. *Spike-latency code*

After presenting examples for binary and multivalued discrete features we turn now to continuous coding features by considering the latency f of the first spike time after stimulus onset as the relevant feature. We assume that a neuron's first spike triggers some process in the population readout. The strength of this effect decreases exponentially with the latency of the first spike, $c = \exp(-\frac{f}{\tau})$ if the neuron spiked and $c = 0$ if not, and for the population activity holds $A_i = \frac{1}{\sqrt{N}}\sum_{\nu\in\mathrm{Pop}_i} c^\nu$. The probability density of the spike latency is given by the product of the probability density to fire at time f and the probability that the neuron did not spike earlier, $P_{\mathbf{w}}(f) = \phi(u(f))\exp(-\int_0^f \mathrm{d}t\,\phi(u(t)))$. Whereas for the previously considered codes the gradient of the feature mean could be calculated analytically, that is not the case now. Hence we make use of Eq. (14) instead of Eq. (13). The cumulative distribution is given by one minus the probability that the neuron did not fire earlier, $C_{\mathbf{w}}(f) = 1 - \exp(-\int_0^f \mathrm{d}t\,\phi(u(t)))$. Hence the term $\frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}$ in (14) yields $\frac{\nabla\int_0^f \mathrm{d}t\,\phi(u(t))}{\phi(u(f))}$, leading to the tightly code-specific learning rule:

$$\Delta\mathbf{w} = \eta R(D - \tanh(A_1 - A_2))\mathrm{e}^{-\frac{f}{\tau}} \times \frac{\int_0^f \mathrm{d}t\,\phi'(u(t))\mathbf{psp}(t)}{\phi(u(f))}. \tag{23}$$

Let us assume the case that $A_1 > A_2$ is positively correlated with reward. Because for such activities the decision is on average $D = 1$, it further follows $\langle R(D - \tanh(A_1 - A_2))\rangle > 0$. In the considered case learning should increase the activity A_1 (and decrease A_2), hence the neurons in Pop_1 should emit their first spike earlier. In other words, the expected number of spikes between stimulus onset and the observed first spike latency f needs to increase. The gradient of this quantity is exactly the integral in the learning rule. Weight changes occur in the correct direction and are further modulated by some factors that depend on the observed first spike time. A slight similarity to the tightly code-specific rule for the spike count (21) code exists. There the expected number of spikes over the whole stimulus was modified, whereas here only the expected number up to the observed first spike changes. In the theoretical part we mentioned that in most cases the gradient estimate obtained by partial integration is better than the one obtained by the 'log-trick'. Indeed, simulations using update rule equation (12) did not yield good results (data not shown).

To obtain the weakly code-specific rule we expand c around θ:

$$\Delta\mathbf{w} = \eta R(D - \tanh(A_1 - A_2))(c - \theta)\nabla \ln P_{\mathbf{w}}(Y) \tag{24}$$

with $\nabla \ln P_{\mathbf{w}}(Y)$ given in (16). The intuition for this rule has already been described for (22) in the case of the spike count code.

We tested the learning rules again on the task of learning 10 stimulus–response associations. We use the same pattern statistics as above, i.e. Poisson spike trains with exponentially distributed rates with a mean of 10 Hz, but keep the input spike train patterns fixed. Instead of generating the input spike trains anew in each trial they are initially generated once and for all. Simulations were done using threshold $\theta = \frac{1}{2}$ and time constant $\tau = 250$ ms. Figures 2(c) and 2(f) show simulation results for both newly derived rules as well as for the standard node perturbation rule. Again, learning speeds up with population size only for the new action perturbation rules, and the tightly code-specific rule clearly outperforms the weakly code-specific one. The weakly code-specific rule considers the whole spike train instead of only the relevant part until the first spike occurs. Therefore the difference between tightly and weakly code-specific rule becomes even more dramatic for smaller values of τ and hence shorter first spike latencies.

3.2. *Continuous actions*

In order to demonstrate the generality of our approach we next turn to continuous actions. In the previous section we have already focused on different codes, here we consider just the spike count code. We start with a basic regression task and finally present results for a navigation task.

3.2.1. *Plain regression*

Instead of learning to produce a lower or higher activity than a contrast population, here the goal of a population is to learn a specific target output activity. We define the output activity as mean number of output spikes, $\bar{f} = \frac{1}{N}\sum_\nu f^\nu$, and the action as activity corrupted by some Gaussian noise ξ, $D = \bar{f} + \xi$. This leads to following tightly code-specific learning rule from Eq. (13):

$$\Delta \mathbf{w} = \eta R \xi \nabla \mu_{\mathbf{w}} \tag{25}$$

with $\nabla \mu_{\mathbf{w}}$ given in (19). Note that the population average enters via $\xi = D - \bar{f}$ into the rule. There is an intuition why the learning rule performs gradient ascent on the expected reward. The noise ξ fluctuates around zero on a trial to trial basis enabling exploration of the reward landscape. Note that the noise arises only in the action space, not in the weight[22] or neuron space.[17] The noise directly explores new actions, and the received feedback is exploited to estimate the reward gradient. We therefore refer to (25) as pure action perturbation rule, whereas general action perturbation might be combined with neuronal noise, as is the case for weakly code-specific rules. The action D is the sum over the features and the fluctuation, therefore for fluctuations ξ that are positively correlated with reward the features should change in the direction of these fluctuations in order to increase the probability to reproduce the rewarded action D, whereas for negative correlation the features should be modified in the opposite direction. Thus the mean of each neuron's feature should change proportional to $R\xi$, which is implemented by modifying the weights accordingly.

The weakly code-specific rule reads

$$\Delta \mathbf{w} = \eta R \xi (f - \theta) \nabla \ln P_{\mathbf{w}}(Y) \tag{26}$$

with $\nabla \ln P_{\mathbf{w}}(Y)$ given in (16). To gain some intuition let us assume the case $R > 0$ and $\xi > 0$ for which the features should increase. Here a comparison of the feature with the threshold θ is done for each neuron. If $f > \theta$, then the neuron successfully contributed to the learning task and the probability to reproduce its output spike train is increased, whereas if the feature is smaller than the reference value θ the probability is decreased.

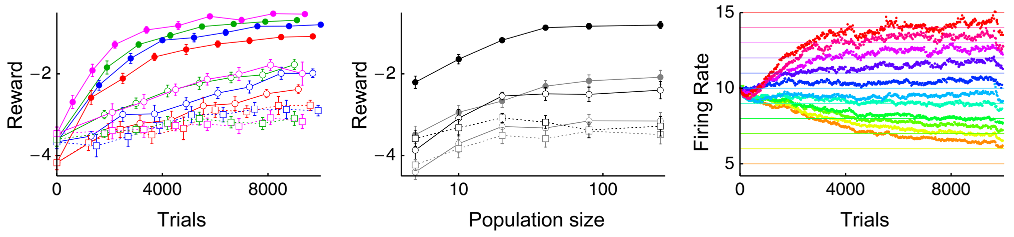
In the simulations (Fig. 3) we used a threshold $\theta = 5$ corresponding to 10Hz output rate and ξ was drawn from a centered normal distribution with standard deviation 0.4. A total of 11 input rate patterns were generated as described above and each has been associated with some target rate $z \in \{5, 6, 7, \dots, 15\}$ Hz. After each trial the action D is compared to the target value zT for the pattern, where $T = 500$ ms is the stimulus length, and the given reward depends gradually on how close they are, $R = -(D - zT)^2$. For each episode one out of 11 prescribed stimulus-action associations is randomly chosen. The results obtained for classification hold for regression, too: The tightly code-specific rule outperforms the weakly one, and for both rules learning improves with increasing population size, in contrast to standard node perturbation, cf. Fig. 3.

3.2.2. *Water maze*

To finally test the learning rules in a realistic paradigm, we simulated Morris' water maze learning task[23] with variable start condition, a task known to involve hippocampus.[24] Hippocampus is represented as a population of place cells,[25] with place cells centers organized on a rectangular grid, cf. Fig. 4. Hippocampal place cells are modeled as Poisson neurons with a firing rate ν that is a Gaussian function of the animal position in the environment,

$$\nu_i(x, y) = \nu_0 \exp\left(-\frac{(x - x_i)^2 + (y - y_i)^2}{2\sigma^2}\right),$$

where (x, y) is the current position of the animal, (x_i, y_i) is the position at which the ith place cell gives the strongest response, $\nu_0 = 100$ Hz is the maximum



(a) (b) (c)

Fig. 3. (Color online) Simulation results for the regression task. (a) Learning curves for the tightly (25, filled discs) and weakly (26, open circles) code-specific rules as well as the standard rule (4, open squares). The population sizes used are: $N = 10$ (red), 20 (blue), 40 (green) and 80 (magenta). (b) Average reward after 2000 (gray) and 4000 (black) trials. (c) Output firing rates for each pattern using the tightly code-specific rule with $N = 80$ neurons. The thin solid lines indicate the target activity. Coloring is used for different target values.

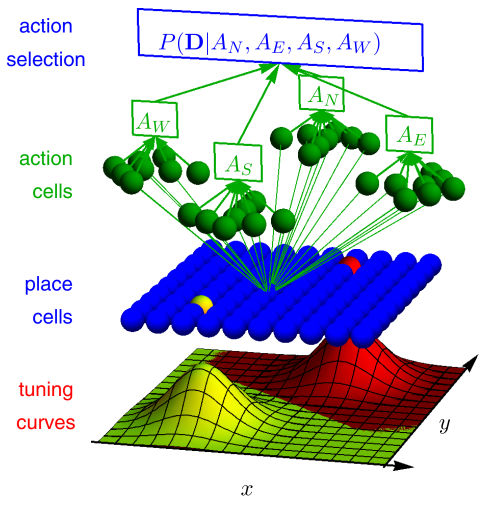


Fig. 4. (Color online) Architecture for water maze task. Place cells encode the position of the animal through place-dependent firing rates. Tuning curves are shown for the yellow and red highlighted place cells. Place cells are connected via feed-forward connections to the action cells (wiring probability 80%). The readout of each action cell population is fed to an action selection circuitry.

firing rate of the place cell. We consider in our simulations 100 such neurons placed on a grid of 10×10 cells, with a distance of 1 (arbitrary units) between two neighboring cells and with $\sigma = 1.2$ being the width of each place field. The ensemble activity of place cells encodes the position (x, y) of the animal. These place cells project onto action cells, modeled as described in Sec. 2.4. The population of action cells represents the next action to be chosen by the model rat and is organized in four populations of size N, one for each cardinal direction. The vector of movement $\mathbf{D}$ is determined by the linear combination of the populations' mean spike count per stimulus length $\bar{f}_i = \frac{1}{N}\sum_\nu f_i^\nu$, with $i = N, E, S, W$, plus some Gaussian noise,

$$\mathbf{D} = \begin{pmatrix}\Delta x\\ \Delta y\end{pmatrix} = \begin{pmatrix}\bar{f}_E - \bar{f}_W\\ \bar{f}_N - \bar{f}_S\end{pmatrix} + \begin{pmatrix}\xi_x\\ \xi_y\end{pmatrix}.$$

The activity of the action cells relates to the change of the animal's position according to the update $(x, y) \leftarrow (x, y) + (\Delta x, \Delta y)$. The average length of $\mathbf{D}$ does not depend on N but is proportional to the individual stimulus duration T, thus the animal's speed is independent of N and T. Here we choose $T = 250\,\text{ms}$, hence actions are taken with theta frequency.

Typically the animal performs a sequence of actions until it reaches the platform, hence reward is delayed. For tasks involving delayed reward, for each synapse an eligibility trace E, which keeps some memory of the neuronal spiking and behavioral actions, needs to be introduced.[26] Whenever an action is chosen the trace changes according to the following update.

$$\text{tight}: \mathbf{E} \leftarrow \mathrm{e}^{-\frac{T}{\tau}}\mathbf{E} + \xi\nabla\mu_{\mathbf{w}}, \tag{27}$$

$$\text{weak}: \mathbf{E} \leftarrow \mathrm{e}^{-\frac{T}{\tau}}\mathbf{E} + \xi(f - \theta)\nabla \ln P_{\mathbf{w}}(Y). \tag{28}$$

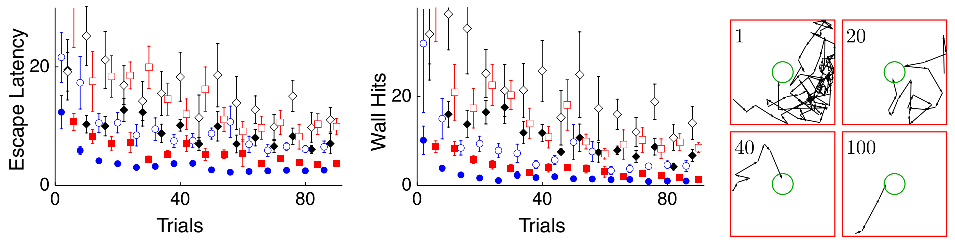
Here ξ depends on the population the considered neuron belongs to, $\xi = \pm\xi_{x/y}$, where ξ_x and ξ_y were drawn from a normal distribution with mean 0 and variance 1. Between the action times the eligibility traces decay exponentially with time constant τ, for which we used 2 s. This decay has already been included in the above updates. As soon as reward is delivered the synaptic weights are updated proportionally to the product of reward and eligibility trace,

$$\Delta \mathbf{w} = \eta R \mathbf{E}. \tag{29}$$

We simulated a model rat navigating in a square maze. The rat performs a number of trials, with each trial consisting of an attempt to find the goal within a time limit of 90 s. At the beginning of each trial, the rat is placed near one of the walls of the maze. Actions are chosen at theta frequency (every 250 ms). The rewarded region (target) is always at the center of the maze, whereas the initial position of the rat varies, as in the experimental paradigm.[24] Positive reward ($R = 1$) is only given if the rat reaches its target and negative reward ($R = -1$) if it hits the wall. Thus, synaptic modifications take place either at the time the rat reaches the platform or hits a wall. When a new set of trials starts, the positions of the rat is reinitialized and the synaptic eligibility traces reset. Thus each new set of trials corresponds to a different animal. The simulation results are summarized in Fig. 5. Panel a depicts the performance of the rat measured by the time it takes to reach the target, corresponding to the escape latency in the experimental literature. Similar to other algorithms, such as a variant with Hebbian bias[25] or using a continuous time actor-critic framework,[27] the escape latency reaches asymptotic performance within the first 20 trials, in line with experimental results.[24] As panel b depicts the number of times the rat hits the wall also decreases during learning. The speed of learning as well as the final performance increases with increasing population size for the tightly as well as the weakly code-specific rules. The common observation that the former outperforms the latter holds here too. For one animal the trajectories in trials 1, 20, 40 and 100 are shown in panel c. Whereas in the first trial the rat meanders around searching for the target, in the last trial it swims straight to the target.

4. Discussion

We have generalized population learning for different neuronal codes and multi-valued decision making. We presented two mathematical techniques to derive such rules, one based on linear approximation around a target feature, and one based on partial integration across feature values. The first technique yields modular learning rules which are weakly code-specific, with a spike-timing-dependent base synaptic plasticity kernel which is modulated by a reward signal and a code specific population signal. Further analytical averaging leads from weakly to tightly code-specific learning rules where details of the code-irrelevant spiking information are integrated away. For cases where an analytical expression does not exist and hence one has to resort to sampling, we propose a gradient estimator based on integration by parts to obtain a tightly code-specific rule. We found that in our population setting, the strongly code specific rule is superior in learning performance as compared to the weakly code-specific rule. This holds true for all population sizes, in particular also for the single neuron setting. The latter has been theoretically predicted before, but not yet been convincingly demonstrated in simulations.[13] However, as we find for population coding, a code-specific



(a) (b) (c)

Fig. 5. Simulation results for the water maze task. Escape latency in seconds (a) and number of wall hits (b) for population sizes $N = 1$ (black diamonds), 5 (red squares) and 40 (blue circles). Filled symbols are used for the tightly code-specific rule (27) and open symbols for the weakly code-specific rule (28). (c) Trajectories in trials 1, 20, 40 and 100 for one and the same run (animal) using the tightly code-specific rule with $N = 5$ neurons. Green and red coloring is used to indicate the target and walls respectively.

base plasticity rule does significantly improve learning performance as compared to the noncode-specific version. An improvement was observed for all three codes: spike/no-spike, spike count and spike latency code. Moreover, all our rules improve in learning speed with increasing population size.

The generality of our schemes for deducing synaptic learning rules in a decision-making circuitry is expressed in the variety of neuronal codes and tasks it applies to. Previous restrictions to binary decision making are relieved by extending the theory to multiple choice alternatives and continuous action selections. While the continuous action selection scenario requires a graded reward signal, this does not need to be provided by an external teacher, as assumed in our basic regression example. Instead the grading can emerge automatically due to the use of decaying eligibility traces based on a binary event, as is the case when a rat moving in Morris' water maze found its target. The reward attributed to a decision is remodulated by the eligibility trace, and decisions leading to earlier reward have higher eligibility and receive stronger reinforcement.

For mathematical clarity we presented the rules for an episodic learning scenario. But a biological plausible implementation of a fully online scheme is also possible[9,10]: to avoid an explicit separation of stimuli in time, eligibility traces can be further low-pass filtered across stimulus boundaries. Concentrations of different neurotransmitters can be used to encode feedback about the population decision and the global reward signal (e.g. acetylcholine or dopamine).

The set of population learning rules we derived can be seen as generalizations of previously studied single neuron learning rules derived in a policy gradient framework. The weakly code-specific population rule shows a base synaptic plasticity kernel which is spike-timing dependent.[15,16] This base synaptic plasticity is modulated by the noise component present in the decision-making circuitry at the population level, much like it was previously suggested by weight or node perturbation at the synaptic and neuronal level, respectively.[17,28] Interestingly, however, the rule which outperforms all other rules displays a remarkable feature at the single neuron level: its base plasticity kernel is not depending on whether a postsynaptic spike was actually elicited or not (Eqs. (17), (21) and (25)), but only on the probability of eliciting such a spike, and hence only on the time course of the postsynaptic membrane potential. As current interpretations of synaptic plasticity experiments focus on postsynaptic spikes rather than on the membrane potential dependence (for a review see Ref. 29), it would be interesting to reconsider these experiments and test how far they are compatible with our suggestion for an efficient and code-specific population learning rule.

Acknowledgments

This work was supported by a grant from the Neurochoice project of the Swiss SystemsX.ch initiative and a SNSF fellowship for prospective researchers (J.F.).

Appendix A

In the main text we derived tightly code-specific rules by analytical averaging, which guarantees gradient estimates of lower variance compared to weakly code-specific rules. While the derivation by isolating a single neuron's contribution is intuitive, it relies on an approximation valid for large population sizes. Instead of utilizing partial integration in the last step to derive Eq. (14), we here start directly with partial integration and Eq. (14) turns out to be even exact.

We choose to start with considering discrete features and handle the case of continuous features at the end by taking the limit. We apply partial summation to $\sum_{\mathbf{f}} P(D\,|\,\mathbf{f})P_{\mathbf{W}}(\mathbf{f})$ before calculating the gradient of $\langle R\rangle = \sum_D \sum_{\mathbf{f}} P(D\,|\,\mathbf{f})P_{\mathbf{W}}(\mathbf{f})R$.

We first assume that the features f take values in a fixed set (independent of $\mathbf{W}$) of $n+1$ scalars, $f_0 < f_1 < \cdots < f_n$. Being precise with regard to random variable and its realization for the considered neuron but remaining concise for the other variables, we write the gradient as

$$\begin{aligned}\nabla\langle R\rangle &= \sum_D R \sum_{\mathbf{f}^{\setminus}} P_{\mathbf{W}}(\mathbf{f}^{\setminus}) \\ &\quad\times \nabla \sum_i \underbrace{P(D\,|\,\mathbf{f}^{\setminus}, f=f_i)}_{d_i}\underbrace{P_{\mathbf{w}}(f=f_i)}_{p_i} \\ &= \sum_D R \sum_{\mathbf{f}^{\setminus}} P_{\mathbf{W}}(\mathbf{f}^{\setminus}) \\ &\quad\times \nabla\left(-\sum_i (d_{i+1}-d_i)\sum_{j=0}^{i} p_j\right),\end{aligned}$$

where the second line is obtained by partial summation, noting that the 'surface term' vanishes. Setting $C_{\mathbf{w}}(f_i) = \sum_{j=0}^{i} P_{\mathbf{w}}(f_j)$ to denote the cumulative probability as well as $\Delta P(D \,|\, \mathbf{f}) = P(D \,|\, \mathbf{f}^{\setminus}, f_{i+1}) - P(D \,|\, \mathbf{f}^{\setminus}, f_i)$ for the difference operator, we put the above result into a form useful in the context of a sampling procedure:

$$\nabla \langle R \rangle = \sum_D R \sum_{\mathbf{f}} P_{\mathbf{W}}(\mathbf{f}) P(D \,|\, \mathbf{f}) \frac{\Delta P(D \,|\, \mathbf{f})}{P(D \,|\, \mathbf{f})} \times \left(-\frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)} \right).$$

The above result is also valid when the features take values in a discrete but infinite set of scalars with the ensuing learning rule

$$\Delta \mathbf{w} = -\eta R \frac{\Delta P(D \,|\, \mathbf{f})}{P(D \,|\, \mathbf{f})} \frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}. \tag{A.1}$$

When n is finite, no update arises when the feature value is f_n since $\nabla C_{\mathbf{w}}(f_n) = \nabla 1 = 0$. Instead of using forward differences $d_{i+1} - d_i$ in the partial summation above, we could also have used the formulation with the backward differences $d_i - d_{i-1}$, yielding an expression for the gradient where no update arises when the feature value is f_0. A general update rule is obtained by taking convex combinations of the two expressions. This introduces a tunable parameter κ, with $0 \leq \kappa \leq 1$, and yields the update

$$\Delta \mathbf{w} = -\eta R \left(\kappa \frac{\Delta P(D \,|\, \mathbf{f})}{P(D \,|\, \mathbf{f})} \frac{\nabla_\nu C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)} + (1-\kappa) \frac{\Delta P_-(D \,|\, \mathbf{f})}{P(D \,|\, \mathbf{f})} \frac{\nabla_\nu C^-_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)} \right) \tag{A.2}$$

with the backward difference function $\Delta P_-(D \,|\, \mathbf{f}) = P(D \,|\, \mathbf{f}^{\setminus}, f_i) - P(D \,|\, \mathbf{f}^{\setminus}, f_{i-1})$ and $C^-_{\mathbf{w}}(f_i) = C_{\mathbf{w}}(f_{i-1})$.

For features which can take on a continuous range of values one can simply use partial integration or take the continuous limit $\Delta f_i \to \mathrm{d}f$ of (A.1) or (A.2), giving the update

$$\Delta \mathbf{w} = -\eta R \frac{\partial \ln P(D \,|\, \mathbf{f})}{\partial f} \frac{\nabla C_{\mathbf{w}}(f)}{P_{\mathbf{w}}(f)}. \tag{A.3}$$

Note that we recovered Eq. (14) without using any approximations.

References

1. H. Markram, J. Lübke, M. Frotscher and B. Sakmann, Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs, *Science* **275** (1997) 213–215.
2. G. Bi and M. Poo, Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength and postsynaptic cell type, *J. Neurosci.* **18**(24) (1998) 10464–10472.
3. M. N. Shadlen and W. T. Newsome, Noise, neural codes and cortical organization, *Curr. Opin. Neurobiol.* **4** (1994) 569–579.
4. M. J. Tovee and E. T. Rolls, Information encoding in short firing rate epochs by single neurons in the primate temporal visual cortex, *Vis. Cognit.* **2**(1) (1995) 35–58.
5. M. Badoual, Q. Zou, A. P. Davison, M. Rudolph, T. Bal, Y. Fregnac and A. Destexhe, Biophysical and phenomenological models of multiple spike interactions in spike-timing dependent plasticity, *Int. J. Neural Syst.* **16**(2) (2006) 79–97.
6. S. Ghosh-Dastidar and H. Adeli, Spiking neural networks, *Int. J. Neural Syst.* **19**(4) (2009) 295–308.
7. A. Pouget, R. S. Zemel and P. Dayan, Information processing with population codes, *Nat. Rev. Neurosci.* **1**(2) (2000) 125–132.
8. B. Averbeck, P. E. Latham and A. Pouget, Neural correlations, population coding and computation, *Nat. Rev. Neurosci.* **7** (2006) 358–366.
9. R. Urbanczik and W. Senn, Reinforcement learning in populations of spiking neurons, *Nat. Neurosci.* **12**(3) (2009) 250–252.
10. J. Friedrich, R. Urbanczik and W. Senn, Learning spike-based population codes by reward and population feedback, *Neural Comput.* **22**(7) (2010) 1698–1717.
11. J. Friedrich and W. Senn, Spike-based decision learning of Nash equilibria in two-player games, *PLoS. Comput. Biol.* **8**(9) (2012) e1002691.
12. S. Schliebs, N. Kasabov and M. Defoin-Platel, On the probabilistic optimization of spiking neural networks, *Int. J. Neural. Syst.* **20**(6) (2010) 481–500.
13. H. Sprekeler, G. Hennequin and W. Gerstner, Code-specific policy gradient rules for spiking neurons, in *Advances in Neural Information Processing Systems 22*, eds. Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams and A. Culotta (2009), pp. 1741–1749.
14. R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Mach. Learn.* **8** (1992) 229–256.
15. J. Pfister, T. Toyoizumi, D. Barber and W. Gerstner, Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning, *Neural Comput.* **18**(6) (2006) 1318–1348.
16. R. V. Florian, Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity, *Neural Comput.* **19**(6) (2007) 1468–1502.
17. I. R. Fiete and H. S. Seung, Gradient learning in spiking neural networks by dynamic perturbation

of conductances, *Phys. Rev. Lett.* **97**(4) (2006) 048104.

18. D. Di Castro, D. Volkinshtein and R. Meir, Temporal difference based actor critic learning — convergence and neural implementation, in *Advances in Neural Information Processing Systems 21*, eds. D. Koller, D. Schuurmans, Y. Bengio and L. Bottou (2008), pp. 385–392.
19. M. Schiess, R. Urbanczik and W. Senn, Gradient estimation in dendritic reinforcement learning, *J. Math. Neurosci.* **2**(1) (2012) 2.
20. X. Xie and H. S. Seung, Learning in neural networks by reinforcement of irregular spiking, *Phys. Rev. E* **69**(4) (2004) 041909.
21. A. Mohemmed, S. Schliebs, S. Matsuda and N. Kasabov, SPAN: Spike pattern association neuron for learning spatio-temporal spike patterns, *Int. J. Neural Syst.* **22**(4) (2012) 1250012.
22. H. S. Seung, Learning in spiking neural networks by reinforcement of stochastic synaptic transmission, *Neuron* **40**(6) (2003) 1063–1073.
23. D. Sridharan, P. S. Prashanth and V. S. Chakravarthy, The role of the basal ganglia in exploration in a neural model based on reinforcement learning, *Int. J. Neural Syst.* **16**(2) (2006) 111–124.
24. R. G. Morris, P. Garrud, J. N. Rawlins and J. O'Keefe, Place navigation impaired in rats with hippocampal lesions, *Nature* **297**(5868) (1982) 681–683.
25. E. Vasilaki, N. Frémaux, R. Urbanczik, W. Senn and W. Gerstner, Spike-based reinforcement learning in continuous state and action space: When policy gradient methods fail, *PLoS Comput. Biol.* **5**(12) (2009) e1000586.
26. J. Friedrich, R. Urbanczik and W. Senn, Spatio-temporal credit assignment in neuronal population learning, *PLoS Comput. Biol.* **7**(6) (2011) e1002092.
27. N. Frémaux, H. Sprekeler and W. Gerstner, Reinforcement learning using a continuous time actor-critic framework with spiking neurons, *PLoS Comput. Biol.* **9**(4) (2013) e1003024.
28. J. Werfel, X. Xie and H. S. Seung, Learning curves for stochastic gradient descent in linear feedforward networks, *Neural Comput.* **17**(12) (2005) 2699–2718.
29. C. Clopath, L. Büsing, E. Vasilaki and W. Gerstner, Connectivity reflects coding: A model of voltage-based STDP with homeostasis, *Nat. Neurosci.* **13**(3) (2010) 344–352.