Dynamic Neural Field Optimization using the Unscented Kalman Filter

Jeremy Fix, Matthieu Geist, Olivier Pietquin and Hervé Frezza-Buet
IMS, Supelec,
2, rue Edouard Belin, 57070 Metz
Email: Jeremy.Fix@Supelec.Fr

Abstract—**Dynamic neural fields have been proposed as a continuous model of a neural tissue. When dynamic neural fields are used in practical applications, the tuning of their parameters is a challenging issue that most of the time relies on expert knowledge on the influence of each parameter. The methods that have been proposed so far for automatically tuning these parameters rely either on genetic algorithms or on gradient descent. The second category of methods requires to explicitly compute the gradient of a cost function which is not always possible or at least difficult and costly. Here we propose to use unscented Kalman filters, a derivative-free algorithm for parameter estimation, which reveals to efficiently optimize the parameters of a dynamic neural field.**

I. Introduction

Dynamic neural fields have been proposed as a continuous model of a neural tissue. They are used in computational neuroscience to model biological functions of the primate brain but also as a computational paradigm to control artificial agents. In both contexts, tuning the parameters of a neural field is a challenging issue that most of the time relies on the knowledge of an expert and on trials and errors. The main difficulty behind tuning these parameters relies on the difficulty to obtain analytical results on the evolution of such a high-dimensional non-linear dynamical system. The first property observed on dynamic neural fields is the existence of stable localized patterns of activity [1]. Dynamical patterns such as traveling waves have also been obtained. These two properties are understood analytically in a restricted context, for a specific transfer function. Other dynamical properties are more tricky to handle analytically, in particular competition and working memory. These two behaviors are important sub-functions of decision making, for instance in cognitive robotics [2]–[6]. The former endows an artificial agent with the ability to settle a competition between, for example, alternative motor programs that cannot be executed simultaneously. It may also allow targets of one motor action to compete. The latter allows an agent keeping a trace of a transiently presented information relevant for guiding future behaviors, or temporarily store the spatial position of a set of stimuli when several have to be investigated (see for example [7]).

Using dynamic neural fields to design cognitive architectures is a challenging problem, especially because the functions one wants to introduce in the system result from distributed computations within the fields. These interactions are themselves ruled by a set of parameters that have a non-linear influence on a neural field's behavior. Such a hard problem motivates finding optimization algorithms to automatically tune neural field parameters.

So far, two optimization techniques have been applied for automatically tuning the parameters of a neural field: genetic algorithms [8] and genetic algorithms combined with a gradient descent [9]. In [8], the authors apply genetic algorithms to tune the parameters of a two-dimensional neural field performing selection and dynamic tracking of a moving input stimulus within noise or surrounded by distractors. In [9], the authors combine a genetic algorithm with a gradient-descent based method. The later requires to compute locally the gradient of a cost function which may be difficult or at least costly to perform.

Kalman filters are a popular collection of algorithms in the optimal control and machine learning community for estimating a hidden state from noisy observations, knowing a model of the underlying process and of the observation apparatus. There are several algorithms that have been developed since the seminal work of Kalman [10] on systems involving linear evolution and observation processes. For example the Extended Kalman Filter (EKF) and the Unscented Kalman filter (UKF) are efficient methods that can be used for three applications. They can be used for state estimation, i.e. estimating a hidden state observed through noisy measurements. They can also be used for parameter estimation, i.e. determining the parameters of a non-linear system given noisy observations. These two algorithms can finally be combined for performing so-called dual estimation, i.e. estimating both a hidden state and the parameters governing a non-linear system. The Extended Kalman Filter requires to explicitly compute the derivatives of the process' equations. In some applications (like the one we address in this article), analytically computing these derivatives can be difficult or even impossible in some cases. The Unscented Kalman Filter does not need to explicitly compute these derivatives. Neural networks are a particular case of non-linear processes and the techniques mentioned above can be applied to tune their parameters. The Kalman Filters reveal to be quite efficient compared to the more popular backpropagation algorithm [11]. The derivative-free aspect of UKF combined with its efficiency in optimizing the parameters of neural networks motivate the study presented in this article, where we seek to optimize the parameters of dynamic neural fields.

The Unscented Kalman Filter [12] belongs to the family of Sigma-Point Kalman filter (SPKF) which can be used to estimate hidden variables producing observations through a non-linear function (therefore extending the original Kalman filtering framework [10] applicable on linear functions). The challenge is to estimate the parameters of this non-linear model only from partial and noisy observations. The Unscented Kalman Filter has already been applied on several machine learning problems. For example, in [13], UKF is used for supervised learning and in [14], the authors use UKF in the context of reinforcement learning to approximate the value function given a reward scheme which provides the indirect noisy observations.

The paper is organized as follow: the unscented Kalman filter algorithm is provided in section II-A, the dynamic neural field equation and parameters are described in section II-B and two example scenarios are studied in section III. Extensions of the presented framework and the implications of such a method in the context of dynamic neural fields applied to cognitive robotics are then discussed.

II. Methods

A. Unscented Kalman Filter (UKF)

The Unscented Kalman filter (UKF) is an efficient derivative-free method for nonlinear filtering problems. In particular, it can be applied to estimate the parameters of a non-linear function. It relies on the unscented transform (UT) [12] which allows to compute a local linear approximation of the non-linear function. To do so, the unscented transform introduces a set of so-called sigma-points from which statistics of interest are computed. We briefly sketch below the different steps involved in the UKF for parameter estimation but a more detailed explanation of this algorithm can be found in [13].

Basically, the parameter estimation problem is stated as: given a non-linear function f_θ parametrized by the vector θ, given a set of input-output samples (x_i, y_i), find the parameter vector $\theta^\star$ that best accounts for the mapping $\forall i, y_i = f_\theta(x_i) + v_i$ in a least square sense, where v_i is a white observation noise (zero-mean and independent). Therefore, at time step i, UKF seeks at minimizing the following cost function:

$$\theta_i = \underset{\theta}{\operatorname{argmin}}(\sum_{j=1}^{i} \frac{1}{P_{vv_j}}(y_j - f_\theta(x_j))^2) \tag{1}$$

where P_{vv_j} is the variance of the noise v_j. This cost function is defined on the samples presented so far. The general purpose of UKF, like Kalman filtering, is to track the hidden state of some system given sequential noisy observations. Here, the state is the hidden parameter vector. The associated *state-space* formulation of the parameter estimation problem is given by:

$$\begin{cases} \theta_i & = \theta_{i-1} + n_i \\ y_i & = f_{\theta_i}(x_i) + v_i \end{cases}$$

Using the vocabulary of Kalman filtering, the first equation is the evolution equation: it specifies that the *true* parameter vector follows a random walk, the expectation of which corresponds to the optimal estimate of the parameters (in a least-square sense). The evolution noise n_i is white (independent and centered) and of variance P_{nn_i} (to be chosen by the practitioner). This allows handling non-stationary parameters (without harming the stationary case), but it also allows avoiding local minima of the cost-function (1) (its effects can roughly be understood as simulated annealing). The second equation is the observation equation: it links the parameters to the observations through the parametrized function f_θ. The observation noise v_i is also white and of variance P_{vv_i} (to be chosen by the practitioner, possibly on the basis of some physical considerations). UKF is an incremental algorithm that improves step by step the parameters. The idea is to obtain a Widrow-Hoff-like linear update rule of the form $\theta_i = \theta_{i-1} + K_i(y_i - \bar{y}_i)$ where $\bar{y}_i$ is a prediction of what should be the next observation y_i given the current estimate of the parameters and $\mathbf{K}_i$ is the so-called Kalman gain. The UKF algorithm aims at incrementally compute the best gain $\mathbf{K}_i$, that is the one that minimizes the cost function (1).

The algorithm we present below[1] and depicted schematically on figure 1 uses the Unscented Transform for clarity of presentation. However, it is the Scaled Unscented Transform [13] that is used in practice for efficiency reasons. Both rely on the same ideas, the Unscented Transform being shorter to introduce, and slight variations lead to the scaled transform which is provided in more details in [13]. In addition, the algorithm provided below is the vectorial algorithm while we used scalar notations previously for introducing it. The algorithm is initialized with random parameters θ_0 and a prior on the variance of these parameters $\mathbf{P}_0$ (in practice, a diagonal matrix). At step i, it first updates the covariance of the parameters :

$$\mathbf{P}_{i|i-1} = \mathbf{P}_{i-1} + \mathbf{P}_{nn_i}$$

Then, a set of $2p+1$ so-called sigma-points $\theta_i^j, j \in [0..2p]$ (p is the number of parameters to estimate) is introduced :

$$\begin{aligned} \boldsymbol{\theta}_i^0 &= \boldsymbol{\theta}_{i-1} \\ \boldsymbol{\theta}_i^j &= \boldsymbol{\theta}_{i-1} + \left(\sqrt{(p+\kappa)\mathbf{P}_{i|i-1}}\right)_j, 1 \leq j \leq p \\ \boldsymbol{\theta}_i^j &= \boldsymbol{\theta}_{i-1} - \left(\sqrt{(p+\kappa)\mathbf{P}_{i|i-1}}\right)_{j-p}, p+1 \leq j \leq 2p \end{aligned}$$

[1]In the description of the algorithm, a matrix is written with a capital bold letter (e.g. $\mathbf{P}_i$), a vector with a bold letter (e.g. $\boldsymbol{x}_i$) and a scalar with a normal letter (e.g. x_i).

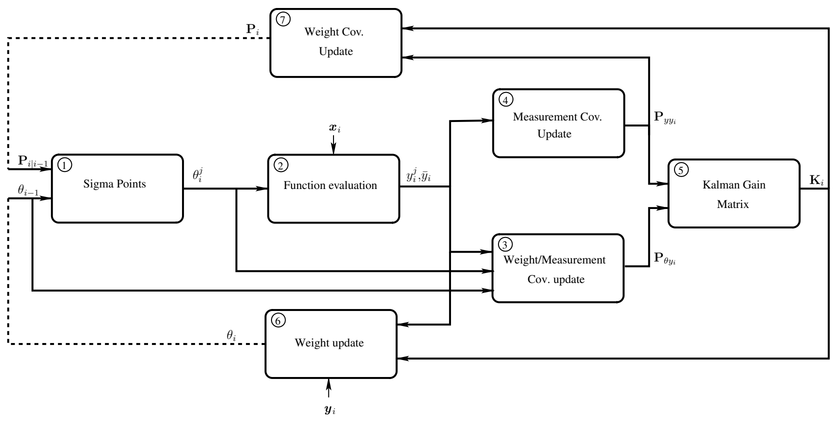


Fig. 1. Signal flow diagram for UKF parameter estimation. Based on a prior estimate of the parameter's covariance $\mathbf{P}_0$, of the parameters $\boldsymbol{\theta}$ and on a new sample (x_i, y_i), UKF for parameter estimation updates the covariance matrix as well as the parameters minimizing the cost function defined by equation (1).

where $\left(\sqrt{(p+\kappa)\mathbf{P}_{i|i-1}}\right)_j$ is the j-th column of the Cholesky decomposition of the matrix $(p+\kappa)\mathbf{P}_{i|i-1}$, and $\kappa \geq 0$ a parameter of the algorithm which controls the spread of the sigma-points sampling. The image of each of the sigma-points through the non-linear function f_θ is then computed, each sigma-point defining a specific parametrization:

$$\boldsymbol{y}_i^j = f_{\boldsymbol{\theta}_i^j}(\boldsymbol{x}_i), \forall j$$

A set of weights, used to compute different statistics of interest is then defined as:

$$w_0 = \frac{\kappa}{p+\kappa} \text{ and } w_j = \frac{1}{2(p+\kappa)}, \forall j > 0$$

From the image of the sigma-points, the set of weights previously defined and the current sigma-points, one can compute the following statistics:

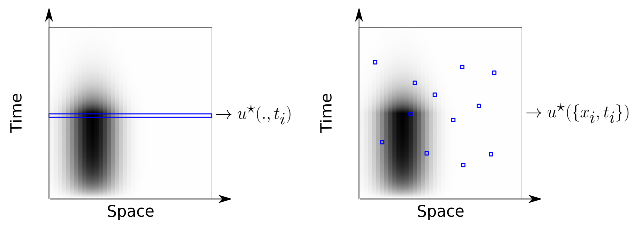
$$\begin{aligned}
\bar{\boldsymbol{y}}_i &= \sum_j w_j \boldsymbol{y}_i^j \\
\mathbf{P}_{\theta y_i} &= \sum_j w_j (\boldsymbol{\theta}_i^j - \boldsymbol{\theta}_{i-1})(\boldsymbol{y}_i^j - \bar{\boldsymbol{y}}_i)^T \\
\mathbf{P}_{yy_i} &= \sum_j w_j (\boldsymbol{y}_i^j - \bar{\boldsymbol{y}}_i)(\boldsymbol{y}_i^j - \bar{\boldsymbol{y}}_i)^T \qquad (2)
\end{aligned}$$

The last step is to update the Kalman gain $\mathbf{K}_i$, the parameters $\boldsymbol{\theta}_i$ as well as the covariance matrix of the parameters $\mathbf{P}_i$:

$$\begin{aligned}
\mathbf{K}_i &= \mathbf{P}_{\theta y_i} \cdot (\mathbf{P}_{vv_i} + \mathbf{P}_{yy_i})^{-1} \\
\boldsymbol{\theta}_i &= \boldsymbol{\theta}_{i-1} + \mathbf{K}_i(\boldsymbol{y}_i - \bar{\boldsymbol{y}}_i) \\
\mathbf{P}_i &= \mathbf{P}_{i|i-1} - \mathbf{K}_i(\mathbf{P}_{vv_i} + \mathbf{P}_{yy_i})\mathbf{K}_i^T
\end{aligned}$$

B. Dynamic neural fields

Dynamic neural fields have been introduced as a model of a continuous neural tissue [1], [15], [16]. The neural field equation (3) states that the evolution of the membrane potential at position x and time t, depends on an input at that position $i(x,t)$, on a baseline h and on the lateral influence within the field characterized by a weight function $w(x,y)$ and a transfer function f.



(a) (b)

Fig. 2. Two sampling methods are considered. In (a), a time is randomly chosen, and the activity of the whole field at that time is presented to the algorithm, i.e. the samples are $(t_i, \boldsymbol{u}^\star(.,t_i))$. In (b), a set of time and space positions are randomly chosen. Therefore, the samples at time i are $(\{x_i^j, t_i^j\}, \{u^\star(x_i^j, t_i^j)\})$

$$\tau \frac{\partial u}{dt}(x,t) = -u(x,t) + \int_y w(x,y) f(u(y,t)) dy + i(x,t) + h \qquad (3)$$

To be properly defined, the previous equation also requires an initial condition $\boldsymbol{u}_0 = \{u(x,0), \forall x\}$. In order to cast the optimization of the parameters of a neural field into a Kalman filtering problem, one needs to parametrize the neural field equation and specify how it is sampled. To parametrize the neural field, we first discretize equation (3) both in space and time using an Euler scheme. The following equations are therefore specific for the Euler scheme but could be easily adjusted to other schemes. The discrete neural field equation therefore reads:

$$\begin{aligned}
u(x,t+\Delta t) = (1 - \frac{\Delta t}{\tau})u(x,t) \\
+ \frac{\Delta t}{\tau} \sum_y w(x,y) f(u(y,t)) \\
+ \frac{\Delta t}{\tau}(i(x,t) + h) \qquad (4)
\end{aligned}$$

We will consider in the following a specific transfer function and a specific weight function. The transfer function is defined as a sigmoidal function parametrized by (a, b, x_0):

$$f(x) = \frac{a}{1 + \exp(b(x - x_0))} \tag{5}$$

The weight function is defined as a difference of gaussians parametrized by $(A_+, \sigma_+, A_-, \sigma_-)$:

$$w(x, y) = A_+ \exp\left(-\frac{(x-y)^2}{2\sigma_+^2}\right) + A_- \exp\left(-\frac{(x-y)^2}{2\sigma_-^2}\right) \tag{6}$$

This symmetric weight function is suitable for the neural field scenarios we consider in the following. However, we may have easily considered other weight functions such as the asymmetrical one used in [17] in order to obtain traveling patterns of activity. Given the previous parametrization, our problem is to optimize 9 parameters: $\theta = (\tau, h, a, b, x_0, A_+, \sigma_+, A_-, \sigma_-)$. Now, we need to specify the samples that are presented to the algorithm (inputs x_i) and the constraints that specify the optimization problem (outputs y_i).

The optimization problem is defined by providing both the input feeding the neural field and a rough estimate of the desired mean firing rate (called desired output in the following). We use the mean firing rate $f(u(x, t))$ instead of the membrane potential $u(x, t)$ since it is bounded and easier to specify a desired firing rate than a desired membrane potential. This constraint is just introduced for practical reasons but could be dropped if necessary. A given input and desired output define a neural field scenario. Two example scenarios are shown on figures 3a,b and 4a,b. These figures illustrate space/time representations of two scenarios: a competition scenario and a working memory scenario. Given these scenarios, a sample (the pairs (x_i, y_i) we introduced in section II-A) can be defined in several ways. We consider in this article two sampling methods, illustrated on figure 2. The first sampling method selects randomly a time and the whole neural field activity at that time is presented to the algorithm. The second sampling method selects randomly pairs of times and positions and presents the neural field activity at these times and positions. The performance of the algorithm with these two sampling methods will be compared in the Results section and others will be discussed in conclusion. The first sampling method is probably the most natural of the two for sampling the state of a dynamical system but the second makes the algorithm faster to converge. Given we used an Euler scheme for discretizing in time equation (3), we consider only temporal samples that are multiple of the time-step Δt.

To state the optimization of neural fields' parameters as a Kalman filtering problem, we must define the evolution and observation equations. The evolution equation states that the parameters follow a random walk, with a white independent noise n_i. The observation equation links the parameters with the observations. To define this equation, we need to introduce an additional notation. Let's denote $u(., t)$ the activity of the whole neural field at a given time t and denote g_θ the parametric function linking $u(., t + \Delta t)$ with $u(., t)$:

$$\begin{aligned} \boldsymbol{u}(., t + \Delta t) &= g_\theta(\boldsymbol{u}(., t)) \\ g_\theta(\boldsymbol{u}) &= (1 - \frac{\Delta t}{\tau})\boldsymbol{u} + \frac{\Delta t}{\tau}\mathbf{W} \cdot f(\boldsymbol{u}) \\ &+ \frac{\Delta t}{\tau}(\boldsymbol{i} + \boldsymbol{h}) \end{aligned} \tag{7}$$

where $\mathbf{W}$ denotes the weight matrix, $f(\boldsymbol{u})$ denotes the vector whose components correspond to the components of $\boldsymbol{u}$ on which f is applied. The state of the neural field at a certain time t, multiple of Δt, is defined as the repeated composition of g_θ applied on the initial condition $\boldsymbol{u}_0$:

$$\begin{aligned} \boldsymbol{u}(., j\Delta t) &= g_\theta(g_\theta(...g_\theta(\boldsymbol{u}_0)...)) \\ &= g_\theta^j(\boldsymbol{u}_0) \end{aligned}$$

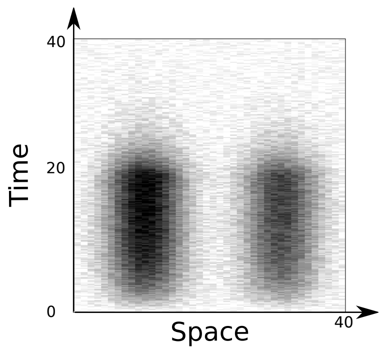
We can now define the so-called *state-space* description of the system, for a certain input $x_i = (k, j\Delta t), k \in [0, n[$:

$$\begin{cases} \theta_i &= \theta_{i-1} + n_i \\ y_i &= g_{\theta_i}^j(\boldsymbol{u}_0)(k) + v_i \end{cases}$$

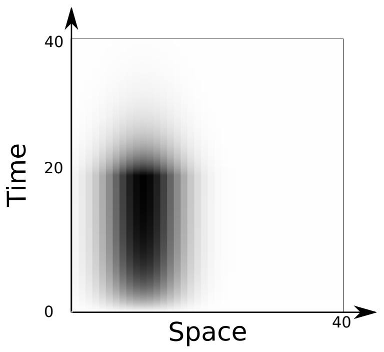
The previous *state-space* description is given for a pair of spatial/temporal position and should be slightly adjusted to correspond to the previous definition of a sample. We do not specify it further to avoid unreadable equations. One should note that the previous definition of the Kalman filtering problem is not dependent on the specific choice of the weight function and transfer function. Given that UKF is a derivative-free algorithm, we do not even need to use derivable functions. This is in contrast to gradient-descent-based methods such as in [9] which require to compute derivatives. The simulations of the neural field and the UKF algorithm were written in C++/GSL.

III. Results

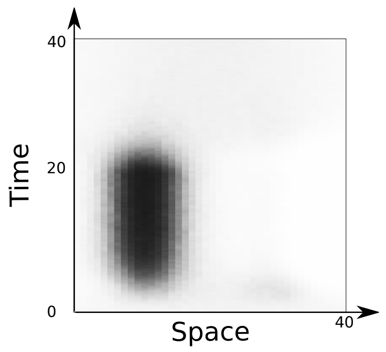
We applied the UKF algorithm on two classical scenarios of dynamic neural fields: competition and working memory. In the first scenario (fig. 3), several regions of a 1D neural field are locally excited and we want to find the parameters of the neural field settling a competition between these excited regions so that the field's activities converge to the representation of only one of the excited regions. Figures 3a,b illustrate the input feeding the neural field as well as the desired firing rate. The input consists in two locally excited regions with relatively close magnitudes (1.0 and 0.75), mixed with a random noise. As an output, we want the neural field to produce a competition between these two regions leading to the selection of the stimulus with the highest amplitude. Intuitively, this competition relies on lateral inhibition where both regions are inhibiting each other. A proper range and amplitude for the lateral inhibition term has to be found in order to get this competition. Figure 3c illustrates the firing rate obtained after 233 iterations of the algorithm showing that



(a)



(b)

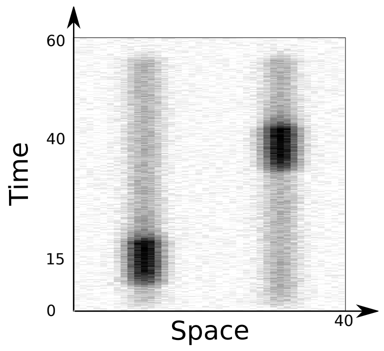


(c)

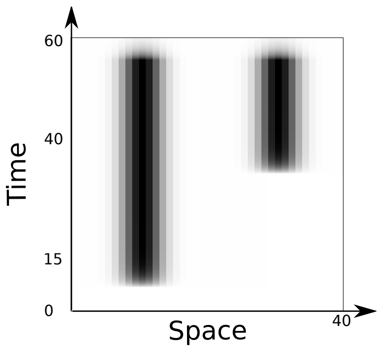
Fig. 3. Competition scenario. (a) Space/time representation of the input exciting the neural field. The two localized excitations are gaussians with an amplitude of 1.0 and 0.75, a standard deviation of 4.0, mixed with a uniform random noise of a maximal amplitude of 0.1 (b) The desired membrane potential seeks to produce competition between the two excitation regions, the one with the highest amplitude being the winner. (c) Obtained neural field after 233 iterations of the algorithm and a RMS error of 0.05. The algorithm converged to the following parameters: $a = 0.93, b = -4.99, x_0 = 0.59, A_+ = 1.53, \sigma_+ = 31.68, A_- = 1.50, \sigma_- = 47.74, \tau = 0.56, h = 0.02$. The neural field, involving 40 neurons, is simulated synchronously during 40 seconds with a time-step $\Delta t = 0.1s$.. The parameters of the scaled UKF (see [13] section 3.2.2 for the meaning of these parameters) were : $\alpha = 0.3, \beta = 2.0, \kappa = 0, \mathbf{P}_0 = 0.1.\mathbf{I}, \mathbf{P}_{nn_i} = 0.0, \mathbf{P}_{vv_i} = 0.1.\mathbf{I}$. The membrane potentials ranging from 0.0 to 1.0 are represented with a graded color from white to black. A video of the optimization algorithm is available at http://jeremy.fix.free.fr/demo.php?demo=CCMB2011 .

the algorithm has found a proper set of parameters allowing this behavior. To compare the two sampling methods and to get an idea of the time it takes for the algorithm to converge, we repeated 1000 times the optimization procedure. It took on average respectively 260 steps and 52 steps to converge (to reach a RMS error of 0.1) for the two sampling methods. The algorithm reached a local minima 13% of the trials for the sampling in time, and 3.5% of the trials for the sampling in time and space. The evolution of the mean RMS, through the iteration of the algorithm, is shown on figure 5a. In addition, some local minima reached by the algorithm with their respective RMS are shown. Sampling randomly in both time and space clearly leads to a faster convergence.

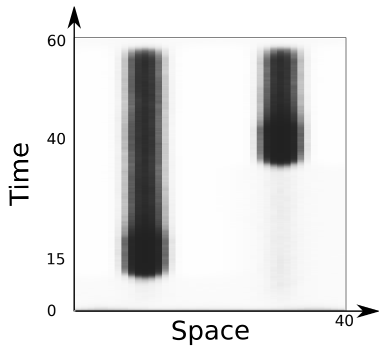
A more difficult scenario is proposed on figure 4. A working memory should be able to maintain the representation of an information despite the presence of distractors, and this as soon as some signals triggered the entrance of this information into the working memory. The input and desired output shown on figures 4a,b illustrate this behavior. There, the neural field is fed with two locally excited regions of initially weak amplitude. A transient increase of the amplitude of the input stimuli, reflecting some kind of selection or gating for the entrance in working memory, triggers the emergence of the stimulus in working memory. Removing the gating signal leads to decrease the amplitude of the selected stimulus back to its initial and weak value. Despite this decrease in amplitude, the stimulus should stay in the working memory. There are mainly three critical parameters allowing to get this



(a)

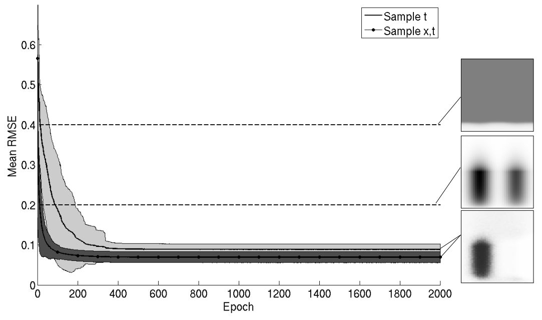


(b)

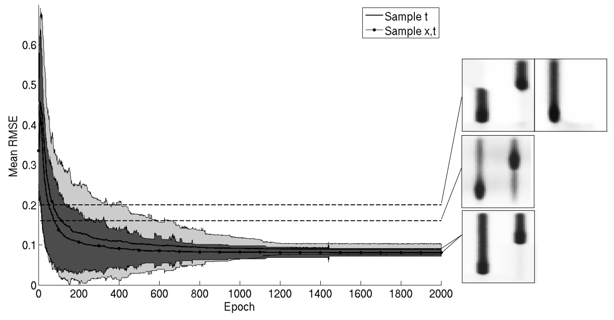


(c)

Fig. 4. Working memory scenario. (a) Space/time representation of the input feeding the neural field. The two localized excitations are gaussians with an initial amplitude of 0.3, transiently increased up to 1.0, and a standard deviation of 2.0. A uniform random noise of amplitude 0.1 is added to the input. The transient increase of the stimuli's amplitude reflects some kind of selection, allowing a target to enter the working memory. (b) The desired firing rate specifies a working memory behavior: when the amplitude of the input reaches a threshold, the stimulus enters the working memory and stays there until the input is completely suppressed. (c) Obtained neural field after 1000 iterations and a RMS error of 0.077. The algorithm converged to the following parameters: $a = 0.92, b = -1.93, x_0 = 1.12, A_+ = 1.21, \sigma_+ = 2.11, A_- = 0.34, \sigma_- = 5.28, \tau = 0.5, h = -0.5$. The neural field, involving 40 neurons, is simulated synchronously during 60 seconds with a time step $\Delta t = 0.1s$.. The parameters of the scaled UKF (see [13] section 3.2.2 for the meaning of these parameters) were : $\alpha = 0.3, \beta = 2.0, \kappa = 0, \mathbf{P}_0 = 0.1.\mathbf{I}, \mathbf{P}_{nn_i} = 0.0, \mathbf{P}_{vv_i} = 0.1.\mathbf{I}$. The firing rates ranging from 0.0 to 1.0 are represented with a graded color from white to black. A video of the optimization algorithm is available at http://jeremy.fix.free.fr/demo.php?demo=CCMB2011 .



(a)



(b)

Fig. 5. Mean RMS error curves for the scenarios shown on figure 3 and 4. For each figures, the curves show the evolution of the RMS through the iteration of UKF for the two sampling methods shown on figure 2 : sampling the whole state of the neural field at a random time or sampling randomly both time and space positions. The filled regions represent the standard deviation. The mean and standard deviation of the RMS are computed for the trials that did not converge to a local minima, defined by a threshold on the RMS. This threshold is set to 0.1 for the competition scenario and 0.12 for the working memory scenario. For both scenarios, we also show illustrations for some local minima with their respective RMS. a) The algorithm converged in a local minima 13% of the trials for the sampling in time, and 3.5% of the trials for the sampling in time and space. b) The algorithm converged to a local minima 42% of the trials for the sampling in time and 32% of the trials for the sampling in time and space.

behavior: the baseline, the lateral excitation and the lateral inhibition. The baseline allows to set up the threshold above which a stimulus will enter the working memory. The lateral excitation allows to maintain a stimulus in working memory when the input's amplitude is set back to its initial value. This lateral excitation compensates the decrease of the input drive. Usually, this lateral excitation is much stronger than in the competition scenario. Given the strength of this recurrent term, it may propagate over the whole field and saturate it completely. Lateral inhibition prevents this spread by somehow constraining the size of the excited regions within the field. Given the strength of the lateral excitation, the neural field may also become insensitive to the input somehow becoming saturated as soon as the input reaches a threshold. One may need that a stimulus gets removed from working memory when it disappears from the input, a behavior that cannot be guaranteed if lateral excitation is too strong. Therefore, a proper balance of lateral excitation/inhibition must be found. This additional constraint is added to the scenario by requiring the firing rate of the neural field to go back to the resting level as soon as the input stimuli disappear, which we see for the last time steps on figures 4a,b . Figure 4c shows the neural field obtained after 1000 iterations of the algorithm. Repeating the optimization procedure on 1000 trials, it took on average respectively 256 steps and 130 steps to converge (to reach a RMS error of 0.12) for the two sampling methods. The algorithm got trapped in a local minimum respectively 42% and 32% of the trials. The stopping criterion used in the simulations relied on the RMS error between the desired and obtained neural field activities. The bound at which the algorithm is stopped is defined qualitatively since it depends on the level of noise in the input, that remains in the neural field activities, but also on the level of noise introduced by defining a desired neural field activity. This desired activity profile is only a rough estimation of the state the neural field can reach. Therefore, a proper bound for the RMS error can only be tuned qualitatively. When setting this bound in the working memory scenario, we usually observed that a RMS error of 0.12 corresponds to a neural field activity that was relatively close to the desired profile (at least, all the trials that ended with such a RMS corresponded to a satisfactory solution from a visual inspection). When it was larger, it usually corresponds to a neural field which maintains only one of the two stimuli as shown on figure 5b. This illustration also shows the evolution of the RMS during the iteration of the algorithm. Again, we observe a tendency for sampling in space and time to converge more fastly and more reliably to the solution.

IV. Discussion

The Unscented Kalman Filter is an efficient statistical learning method that can be used for parameter estimation. We have shown here that it can be used efficiently for optimizing the parameters of a dynamic neural field, given a desired behavior (an input and a desired firing rate) has been specified and regardless the complexity of the neural field equation. In particular, non-linear or even non-derivable functions can be introduced in the equation. This derivative-free optimization algorithm is particularly efficient and makes it a good candidate to automatically tune the parameters of a neural field, which is a critical issue when one wants to apply dynamic neural fields to control artificial agents.

The major advantage of UKF compared to the previous works [8], [9] is its derivative-free aspect which does not restrict to the usage of derivable functions (e.g. a Heaviside function is also a classical transfer function). In terms of complexity, the genetic algorithms require to compute the fitness over the whole time horizon of all the individuals, i.e. an order of $O(N_I\, n_T\, n_x^2)$ computations, where N_I is the number of individuals, n_T the number of time steps and n_x the number of space positions. They also require to update the parameters but given that there are much less parameters than the number of space and time positions (i.e. $p \ll n_x$ and $p \ll n_T$), this step has a low impact on the overall complexity. The gradient-descent-based method used in [9] requires an order of $O(p\ n_T\ n_x^2)$ computations. UKF involves different operations such as matrix inversion, Cholesky decomposition, etc. . However, for the same reason as before (i.e. $p \ll n_x$ and $p \ll n_T$), the complexity of UKF is only dependent on $2p+1$ evaluations of a neural field (when computing the image of the sigma-points). Therefore, a single UKF update requires an order of $O(p\ n_T\ n_x^2)$ computations. The complexity of UKF and a gradient-descent are similar but one should expect that UKF converges faster since UKF belongs to the family of Sigma-Point Kalman Filters (SPKF) which are an online form of a modified Gauss-Newton method which is a variant of natural gradient descent (see [13], ch. 4.5). More than just estimating the complexity of a single step, one would also need to perform benchmarks of the three algorithms to determine the number of steps they require to converge in order to provide a fair comparison. In addition, we discuss below some improvements of the proposed method to make UKF even faster to converge.

Two improvements of the presented method are currently investigated. The first one is to propose a more efficient sampling method for speeding up the convergence of the algorithm. In particular, one can use the information on the covariance of the output (equation 2) to bias the sampling of the inputs. Instead of the random sampling used so far, we could use a probabilistic draw biased by the covariance of the output of the non-linear function. It would speed up the convergence by presenting the samples where the uncertainty on the predictions (of the observations) is higher. This uncertainty depends both on the uncertainty on the value of the parameters and also on the sensitivity of the non-linear function respect to the parameters. A second improvement is also based on the way the neural field is sampled. In particular, instead of evaluating the whole neural field until a certain time, one could use transitions from, say state $u(t)$ to $u(t+1)$ given that $u(t)$ is provided by the desired output. Given the desired output is only a rough estimation of the state that the dynamic neural field can reach, or said differently that the desired output is a noisy observation of a reachable output starting from a given initial condition, one can introduce additional equations in the *state-space* formulation to consider that the neural field state is also part of the evolution equation. These improvements would allow to scale up more easily the algorithm to deal with two or higher dimensional neural fields which are more common in practical applications.

Acknowledgment

This work has been supported by the Région Lorraine.

References

[1] S. Amari, “Dynamics of pattern formation in lateral-inhibition type neural fields.” *Biol Cybern*, vol. 27, no. 2, pp. 77–87, 1977.

[2] J. Fix, N. Rougier, and F. Alexandre, “A dynamic neural field approach to the covert and overt deployment of spatial attention,” *Cognitive Computation*, pp. 1–15, 2010, 10.1007/s12559-010-9083-y. [Online]. Available: http://dx.doi.org/10.1007/s12559-010-9083-y

[3] G. Schoner, “Dynamical systems approaches to cognition.” in *Cambridge handbook of computational cognitive modeling*. Cambridge University Press, 2007.

[4] W. Erlhagen and E. Bicho, “The dynamic neural field approach to cognitive robotics.” *J Neural Eng*, vol. 3, no. 3, pp. R36–54, 2006.

[5] J. Spencer and G. Schoner, “An embodied approach to cognitive systems: A dynamic neural field theory of spatial working memory.” in *Proceedings of the 28th Annual Conference of the Cognitive Science Society (CogSci 2006)*, 2006, pp. 2180–2185.

[6] J. Vitay, N. P. Rougier, and F. Alexandre, “A distributed model of spatial visual attention.” in *Biomimetic Neural Learning for Intelligent Robots*, ser. Lecture Notes in Computer Science, S. Wermter, G. Palm, and M. Elshaw, Eds., vol. 3575. Springer Berlin / Heidelberg, 2005, pp. 54–72.

[7] J. Fix, J. Vitay, and N. Rougier, “A distributed computational model of spatial memory anticipation during a visual search task.” in *Anticipatory Behavior in Adaptive Learning Systems*, ser. Lecture Notes in Computer Science, M. Butz, O. Sigaud, G. Pezzulo, and G. Baldassarre, Eds., vol. 4520. Springer Berlin / Heidelberg, 2007, pp. 170–188.

[8] J. Quinton, “Exploring and optimizing dynamic neural fields parameters using genetic algorithms,” in *Proceedings of IEEE World Congress on Computational Intelligence, IJCNN 2010*, 2010.

[9] C. Igel, W. Erlhagen, and D. Jancke, “Optimization of dynamic neural fields,” *Neurocomputing*, vol. 36, no. 1–4, 2001.

[10] R. Kalman, “A new approach to linear filtering and prediction problems,” *ASME Journal of Basic Engineering*, pp. 35–45, 1960.

[11] S. Haykin, *Kalman filtering and neural networks*. John Wiley and Sons, Inc., New York, 2001.

[12] S. Julier and J. Uhlmann, “Unscented filtering and nonlinear estimation.” in *Proceedings of IEEE,*, vol. 94, no. 3, 2004, pp. 401–422.

[13] R. van der Merwe, “Sigma-point kalman filters for probabilistic inference in dynamic state-space models.” Ph.D. dissertation, OGI School of Science & Engeneering at Oregon Health & Science University, Portland, OR, USA, 2004.

[14] M. Geist and O. Pietquin, “Kalman temporal differences.” *Journal of artificial intelligence research*, vol. 39, pp. 483–532, 2010.

[15] H. Wilson and J. Cowan, “A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue.” *Kybernetik*, vol. 13, no. 2, pp. 55–80, 1973.

[16] J. Taylor, “Neural bubble dynamics in two dimensions: foundations.” *Biol Cybern*, vol. 80, pp. 393–409, 1999.

[17] K. Zhang, “Representation of spatial orientation by the intrinsic dynamics of the head-direction cell ensemble: a theory.” *J Neurosci*, vol. 16, no. 6, pp. 2112–26, 1996.