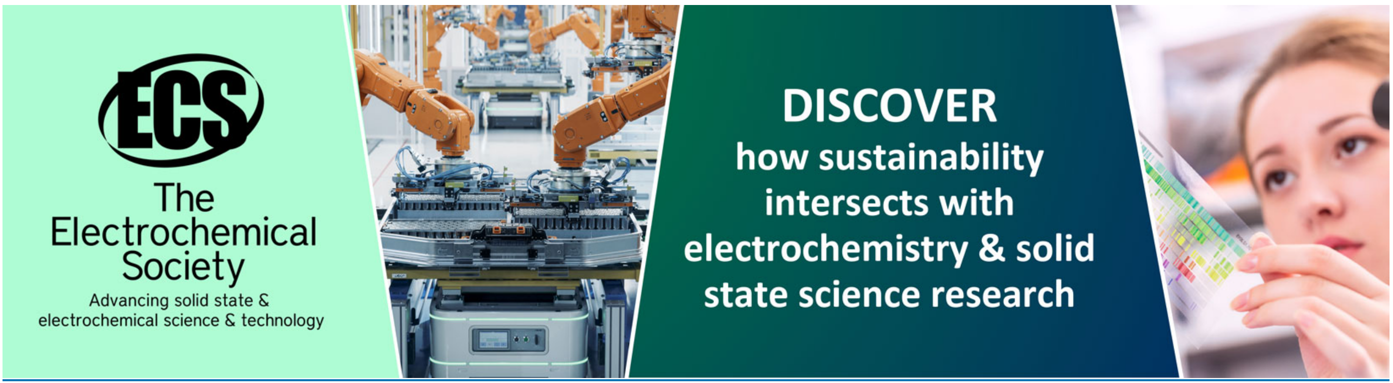
PAPER • OPEN ACCESS

Nonlinear Motion Tracking by Deep Learning Architecture

To cite this article: Arnav Verma *et al* 2018 *IOP Conf. Ser.: Mater. Sci. Eng.* **331** 012020


ECS
The Electrochemical Society
Advancing solid state & electrochemical science & technology
DISCOVER
how sustainability intersects with electrochemistry & solid state science research

Nonlinear Motion Tracking by Deep Learning Architecture

Arnav Verma[1]; Devesh Samaiya[2]; Karunesh K. Gupta[2]

University of Edinburgh, UK.
Birla Institute of Technology & Science, Pilani

kgupta@pilani.bits-pilani.ac.in

Abstract. In the world of Artificial Intelligence, object motion tracking is one of the major problems. The extensive research is being carried out to track people in crowd. This paper presents a unique technique for nonlinear motion tracking in the absence of prior knowledge of nature of nonlinear path that the object being tracked may follow. We achieve this by first obtaining the centroid of the object and then using the centroid as the current example for a recurrent neural network trained using real-time recurrent learning. We have tweaked the standard algorithm slightly and have accumulated the gradient for few previous iterations instead of using just the current iteration as is the norm. We show that for a single object, such a recurrent neural network is highly capable of approximating the nonlinearity of its path.

1. Introduction

The Object tracking is one of the most important problems in the world of artificial intelligence. It is used very commonly in public surveillance systems such as Face Watch in UK to discourage crimes and in the task of counting people indoors for biometric systems [1]. Crowd tracking basically involves segmenting moving people in an image and predicting their future positions based on previous positions-a task commonly called tracking. Major complexities in tracking occur due to lighting and illumination conditions, occlusion, and nonlinearity of motion, resulting in high misclassification rate and misplaced prediction in tracking [2].

Object tracking is performed in two steps; first it is detection then tracking. The detection generally involves, background model development followed by evaluation of deviations in each frame. Pixels that are transformed require further treatment such as background subtraction. Most common challenges faced during detection involve illumination changes which cause false classifications, camouflages resulting in objects similar to background, presence of irrelevant moving objects like leaves, water, flags et cetra, and shadows that may get classified as an object of interest. A more robust technique used for foreground detection is the mixture of Gaussians method [3]. In this algorithm, each pixel is generally modelled as a mixture of Gaussian densities. Based on the local behavior, some of these Gaussians will represent the foreground whereas, the rest will depict the background. Here, using a mixture of Gaussians is necessary because-

- If each pixel represents a single surface, a single Gaussian would have sufficed. However, in practice, multiple surfaces might appear in a single pixel.
- If lighting varied slowly with time, only a single adapting Gaussian would have sufficed. However, lighting conditions may change in many ways, and at varied rates.

Tracking, as the name suggests, refers to tracking the motion of the detected objects, also commonly known as trajectory estimation. In realistic environments, size, shape, and motion of the objects change over time, resulting in one object often getting recognized as a different object, causing the tracker to lose track. Occlusion i.e one object covering another is a very common problem faced in tracking. Occlusion usually occurs in two ways. Inter-object occlusion is when to objects of interest cloud each other, resulting in a single blob of different size than expected. If occlusion occurs in field-of-view, it is very easy to detect. However, if the objects were occluded when they entered the field-of view, as is often common in crowds, then it is very difficult to detect. Occlusion due to scene structure occurs when an object of interest moves behind an irrelevant object such as a car, tree, building et cetra. Additionally, imperfect detection also deteriorates the performance of tracking.

This paves the way for the Kalman filter. A Kalman filter is essentially a state space model for stochastic estimation, composed of predictor-corrector equations and is an optimal linear estimator. Kalman filters are widely used in the fields of aeronautics, orbit determination, speech enhancement, and various other tasks that involve path prediction, or autonomous or assisted navigation. Hence, while Kalman filter is the most widely used tracking technique in object tracking, it does have some shortcomings surrounding the assumptions it makes in arriving at an optimal solution- linearity, and white Gaussian noise. Most tracking systems assume the motion of the object occurs with constant acceleration, resulting in a roughly linear predicted path. When it comes to tracking a nonlinear time-series, which is what the task of tracking boils down to, Kalman filter and its nonlinear variants like Extended Kalman Filter, and Unscented Kalman Filter are inadequate as they require a prior knowledge of the nature of nonlinearity of the data which is to be tracked.

In this paper, the algorithm is proposed to track nonlinear time series using a Recurrent Neural Networks as they are highly efficient at dealing with nonlinearity, even in the absence of external inputs. Now, when we try to track an object, we would essentially be tracking the centroid of the object. This corresponds to online training and for RNNs, this is best carried out by an algorithm known as Real-time Recurrent Learning (RTRL)

2. Mixtures of Gaussians

2.1. Introduction and Initialization

For object detection, we have relied on foreground detection using Mixture of Gaussians due to reasons mentioned earlier [3].

$$\boldsymbol{X_1,\ldots,X_t = I(x_0,y_0,i): 1 \preccurlyeq i \preccurlyeq t}$$

$$\boldsymbol{P(X_t) = \sum_{i=1}^{K} w_{i,t} * n\left(X_t, u_{i,t}, \Sigma_{i,t}\right)}$$

$$\boldsymbol{n\left(X_t, u_{i,t}, \Sigma_{i,t}\right) = \frac{1}{(2\pi)^{n/2}(|\Sigma|)^{1/2}} e^{\frac{-1}{2}(X_t-u_t)^T\Sigma^{-1}(X_t-u_t)}}$$

We choose the number of Gaussians- K, which is usually kept between 3 and 5. The learning rate is generally taken between 0.1 and 0.01. These initializations are done at each pixel with crudely initialized mean vectors and covariance matrices, as the algorithm is expected to evolve and initial observations may be unreliable. At every frame t, we take the feature vector $x_t = [R_t\ G_t\ B_t]$. Now, using Expectation-Maximization algorithm at each pixel in every frame is computationally complex. The algorithm being used is an online K-means approximation.

2.2 The Algorithm

- For each vector x_t in current frame, find which Gaussian matches this observation. This is done by first assuming independence of the three color components and a shared variance and then choosing the Gaussian for which, the observation is within 2.5 of signal mean value. If more than one such Gaussian turns out to be a match, the best such match is considered.

- Now for the matched Gaussian, the weights are updated as shown below and renormalized.
- Additionally, the following updates are also carried out.
- If current pixel value matches none of the K Gaussian pdfs, the distribution with lowest probability is updated to a distribution with the current value as its mean value, an initially high variance, and low prior weight.

$$W_{K,t} = (1-\alpha)W_{k,t-1} + \alpha\left(M_{k,t}\right)$$

$$\mu_t = (1-\rho)\mu_{t-1} + \rho X_t$$

$$\sigma_t^2 = (1-\rho)\sigma_{t-1}^2 + \rho(X_t - \mu_t)^T(X_t - \mu_t)$$

where the second learning rate is-

$$\rho = \alpha\eta(X_t l \mu_k, \sigma_k)$$

The background will theoretically have low variance and high weight. Hence the Gaussians are first ordered based on ratio of weight and standard deviation. Hence, the first B models are chosen as part of background model based on the following constraint where T is a measure of the minimum amount of data that must be accounted for by the background.

$$B = argmin_b\left(\sum_{k=1}^{b} W_k > T\right)$$

Blurring as well as morphological operators such as erosion and dilation to fill holes and get regions corresponding to objects in the scene.

The same process is now repeated for the next frame.

3. Recurrent Neural Network

Tracking nonlinear motion is often carried out with algorithms such as the extended Kalman filter, unscented Kalman filter, particle filter etc. However, these algorithms require in an intrinsic understanding of the inherent nonlinearity of the system, which in turn can often be quite hard to quantify in terms of a simple mathematical function.

Thus, tracking, which is essentially the task of predicting a time-series, seems to demand a certain precognition of the expected path, which defeats the very purpose of tracking. Traditional discrete Kalman filters work very well for linear paths with the assumption of constant acceleration. This, however, does not extend to nonlinear paths as there are infinitely large number of possibilities that might act as an approximation function.

This is where neural networks fill in. We already know that feed-forward neural networks are capable of fitting curves of high amount of nonlinearity. However, predicting a time-series arising from position data of nonlinear motion requires information about history of the time-series to be provided to the network in a manner of online training. Such a system is called a recurrent neural network, a deep learning architecture that is known to do well for time-series data, and has a documented algorithm for online training called the real-time recurrent neural network [5], [6].

3.1 Architecture

- In proposed architecture, Type-I Forward Recurrent Neural Network is used.
- In this architecture, there would be M external inputs, N neurons in the hidden layer, and L output units. Additionally, the output units are a subset of the hidden layer neurons, as seen in Figure 1.

The outputs of the neurons from previous time instant or iteration are sent as inputs to the neurons of the next time instant or iteration. Hence, there are effectively, M+N inputs.

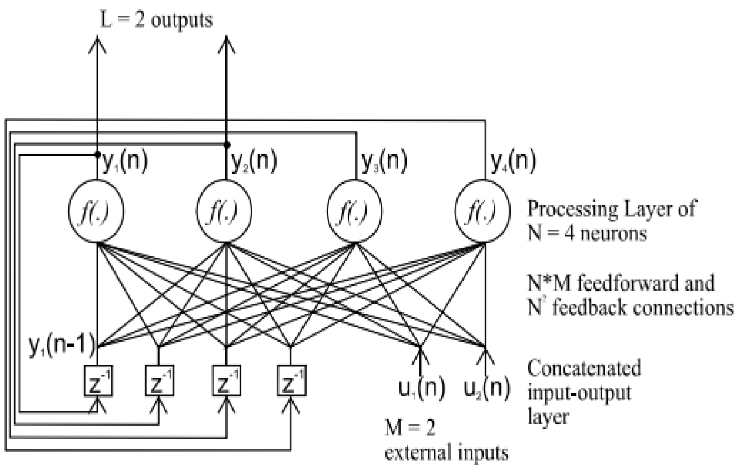

L = 2 outputs
y1(n)
y2(n)
y3(n)
y4(n)
f(.)
Processing Layer of N = 4 neurons
N*M feedforward and N² feedback connections
y1(n-1)
z-1
u1(n) u2(n)
Concatenated input-output layer
M = 2 external inputs

Figure 1. Proposed architecture of Type-I Forward Recurrent Neural Network

3.2 Real-time Recurrent Learning (RTRL)

RTRL is a gradient decent algorithm for training recurrent neural networks in which, an online learning is employed with the error gradient and update weights calculated for each and every time step [4], [6].

Let f be the transfer function of the neurons and z be the concatenated input vector that combines both external inputs as well as feedback of neurons from previous iterations. The algorithm proceeds as follows-

- Let I be the set of indices that correspond to inputs arising from external input. The problem of automatic tracking requires no external inputs and hence, we will not be employing external inputs in our final solution. Similarly, let U be the set of indices that correspond to inputs arising from feedback from neurons of previous iteration.

$$Z_k(t) = \begin{matrix} X_k(t) if k \in I \\ Y_k(t) if k \in U \end{matrix}$$

- If W is the weight matrix with m+n columns and n rows.

$$net_k(t) = \sum_{L \in U \cup I} W_{kL}\, Z_L(t)$$

Now, if the neural outputs are represented by the vector y,

$$y_k(t+1) = f_k\big(net_k(t)\big)$$

- Let the target values be stored in a matrix d, of vector targets at each time instant. Now, let's define the error for this time instant as:

$$E(\tau) = \frac{1}{2}\sum_{k \in U} e_k\,(\tau)^2$$

where

$$e_k(\tau) = \begin{matrix} d_k(t) - y_k(t) & if\ \ k \in T(t) \\ 0; & otherwise \end{matrix}$$

- If t_o is initial time instant, and t_l is the current time instant, the accumulated error over all time instants, which must be differentiated to obtain gradient for RTRL, and the corresponding gradient are:

$$E_{total}(t_0, t_1) = \sum_{\tau=t_0+1}^{t1} E\,(\tau)$$

$$\nabla_w E_{total}(t_0, t+1) = \nabla_w E_{total}(t_0, t) + \nabla_w E(t+1)$$

- Let the learning rate be μ. Hence, the weight update required, based on this single step would be (The net update usually employs accumulated weight updates over the previous time steps as well)

$$\Delta W_{i,j} = -\mu \frac{\partial E(t)}{\partial W_{i,j}}$$

$$\sum_{t=t_0+1}^{t1} \Delta W_{i,j}\,(t)$$

$$\frac{-\partial E(t)}{\partial W_{i,j}} = -\sum_{k\in U} \left(\frac{\partial E(t) \partial Y_k(t)}{\partial Y_{(k)}(t) \partial W_{(ij)}} \right)$$

- Now, we need to evaluate an expression for the partial derivative of the neural outputs with respect to each and every weight matrix element. Combining the expressions for net_k , gradient of error, and output of neurons,

$$\frac{\partial Y_k(t+1)}{\partial W_{i,j}} = f'_k\big(net_k(t)\big) \sum_{L\in U} W_{kL} \frac{\partial Y_L(t)}{\partial W_{i,j}} + \delta_{ik} z_j(t)$$

Here, δ is the Kronecker delta function

- The evaluated partial derivative of y involves another intricacy. As external inputs are independent of weights at any time step, their derivatives with respect to the elements of the weight matrix have been taken as 0, i.e-

$$\frac{\partial z_l(t)}{\partial W_{ij}} = 0 for l \in L$$

- Our starting state at time step to is again independent of the weights. Hence, for k spanning all neural outputs at this time instant, i spanning over all neurons, and j spanning over all inputs (internal, as well as external),

$$\frac{\partial y_k(t_0)}{\partial W_{ij}} = 0$$

- Let p be the matrix containing values of derivatives of neural outputs:

$$p_{ij}^k(t) = \frac{\partial y_k(t)}{\partial W_{ij}}$$

Hence,

$$\Delta W_{ij}(t) = \mu \sum_{k\in U} e_k\,(t) p_{ij}^k(t)$$

In our approach, we will slightly tweak the RTRL algorithm to accumulate error over few previous iterations, instead of just the current iteration, as is the norm.

4. Experimental Results

The method was tested using fabricated nonlinear centroid data, which was then normalized as the output of the tan sigmoid function that we used as the activation function ranges from 0 to 1 on MATLAB ver.12. We used one hidden layer with 10 hidden units, 300 frames, and normalized, symmetric, square, data to train the RNN. The optimum number of hidden units, and number of previous iterations over which we accumulate the error for a particular dataset maybe fixed by using Genetic Algorithm to optimize the total error over all iterations. Throughout all the tests, the learning rate was maintained at 0.1.

Table 1 shows a decreasing trend for the total error with increasing number of iterations for which, error gradient is accumulated until this measure reaches 7 for the case of symmetric, normalized, square centroids. However, even with accumulation over 6 iterations, the RNN tracks the initial data poorly.

Table 2 shows the variation of accumulated error with change in the number of units in the hidden layer. For the case of symmetric, normalized, square centroid, 8 hidden units provide optimal performance for prediction, for the RNN. While using 3, 6, and 7 hidden units provide lower total error, all these cases show better tracking for starting data and poorer tracking for later data, which is undesirable as opposed to using 8 hidden units, in which case, the tracking improves with arrival of new data.

Table 1. Iterations and accumulated error.

No: of iterations for which error is accumulated	Accumulated error
1	0.6188
2	0.2272
3	0.1443
4	0.988
5	0.746
6	0.718
7	0.1267

Table 2. Hidden Units and Accumulated Error

Number of hidden layers	Accumulated error
3	0.0868
4	0.1165
5	0.0925
6	0.0844
7	0.0640

8	0.0893
9	0.0943

Figure 2 (a) shows the recurrent neural network's attempt to track linear motion of Figure 2(b) without any prior knowledge of the path.

Figure 3 (a) & 3 (b) represent the coordinates of the centroid of the object along the X and Y axes respectively for 10 hidden units, and error gradient accumulation over 4 iterations. This network, when trained, predicts the X and Y coordinates as shown in Figure 4 (a) & (b) respectively, again without any prior indication of the nature of motion.

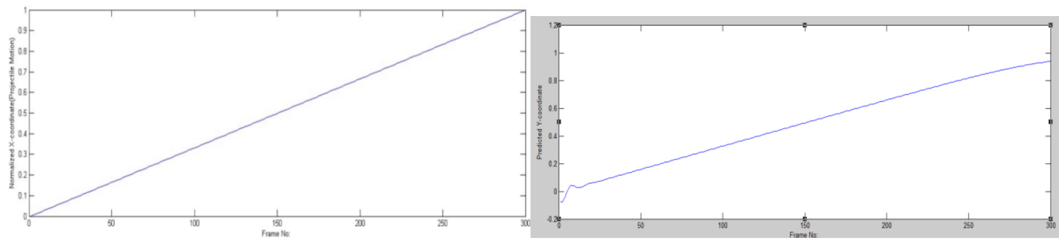


Figure 2. (a) line input (b) Line output

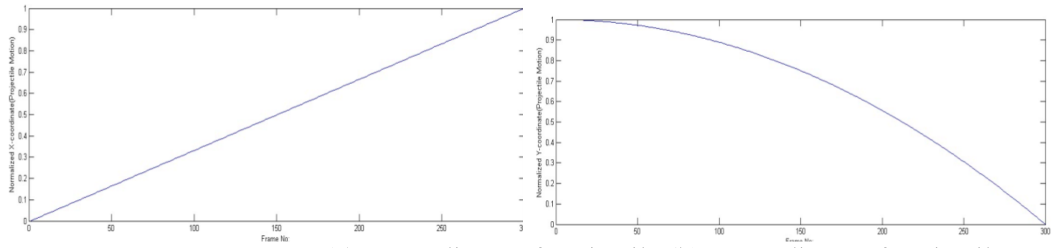


Figure 3. (a) x coordinate of projectile (b) y coordinate of projectile

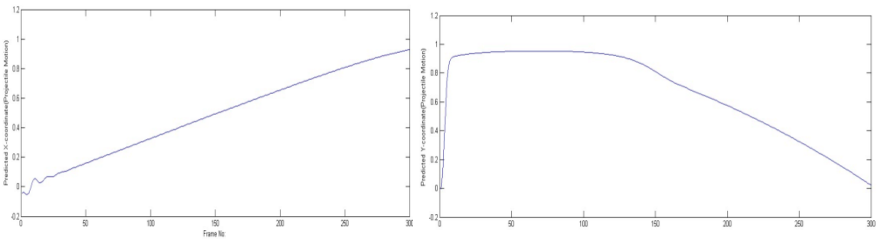


Figure 4. (a) projectile predicted X coordinate (b) Projectile Predicted Y coordinate

Finally, Figure 5(a) & (b) show the tracking performance for the case of sinusoidal data with low frequency for the first few frames with the tracked response and the target function plotted respectively. Highly chaotic data proved to be difficult to track in real-time using real-time recurrent learning.

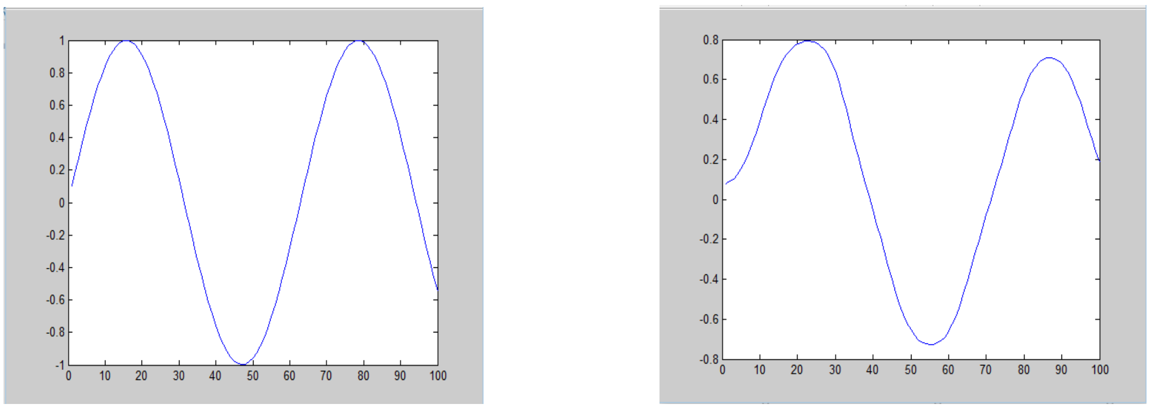


Fig 5. (a) target sinusoidal data (b) predicted sinusoidal data

5. Conclusions and Future work

The proposed algorithm based on deep learning architecture is highly capable of predicting nonlinear tracking. It has proven to be highly capable of predicting nonlinear paths of a single object without any previous knowledge or assumption about the path the object is going to take, albeit with some amount of deviation. This is an improvement over nonlinear tracking algorithms like Extended Kalman Filter and Unscented Kalman Filter. While the complexity of real-time recurrent learning is O(N4) for every time-step in the absence of external input, for fewer number of hidden units, the algorithms computes very quickly.

However, there are many aspects of this method to improve upon in the future, the first of which is to handle multi object tracking efficiently, even in the presence of occlusion. Occlusion has adverse effects on the tracking algorithm’s performance. There is also an observably larger error in prediction when the coordinates of centroids have different natures. This may be dealt with by using different RNNs for each coordinate. There is also work to be done in identifying the apt number of hidden layers and units within these layers that satisfy the trade-off between complexity and tracking performance.

References

[1] Milan Sonka, Vaclav Hlavac and Roger Boyle, “Image Processing, Analysis and Machine Vision” International Student Edition, Third Edition, 2008.
[2] Chris Stauffer, W.E.L.Grimson, Adaptive Background Mixture Model for real-time tracking, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. pp. 246-252, Santa Barbara, CA, June 1998.
[3] Stauffer, C., and Grimson, E. Adaptive background mixture models for real-time tracking. In Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition, 1999.
[4] Esko O. Djik, Analysis of Recurrent Neural Networks with Application to Speaker Independent Phoneme Recognition, Laboratory of Signals & Systems ñ Network theory, University of Twente Enschede, The Netherlands, 1999.
[5] Prof. C.Anderson, rtrl.m,rtrlinit.m
[6] R.J.Williams, D.Zisper, Gradient Based Learning Algorithms for Recurrent Networks and their Computational Complexity,1995.