CS229 Fall 2016, Stanford, CA

Seeing Beyond Seeing with Enhanced Deep Tracking

Zhiyang He
Stanford University, Stanford, CA
hzyjerry@stanford.edu

Abstract

Deep Tracking ([1]) is a recently proposed theoretical framework for end-to-end unsupervised visual tracking. Researchers have successfully applied the framework to directly predict full unoccluded space occupancy grid map from raw laser input data. Compared to traditional visual tracking methods on robotics and autonomous driving, which heavily relies on real world knowledge, the learning framework demonstrates very good potential without the need of feature engineering. In this work I further investigate the practical limitation of the current Deep Tracking architecture. I conduct multiple experiment on the framework, and provide detailed discussion of its viability and performance in different testing schemes. The source code and synthetic datasets used by this study can be found online.[1]

Introduction

Recent machine learning advances in deep neural network has led to advancement in computer vision. Deep Tracking is an end-to-end tracking framework recently proposed by researchers in University of Toronto. Using recurrent neural network, the model learns real world dynamics in unsupervised manner. It helps robotic systems understand occluded scenes --- see beyond seeing --- without the need of ground-truth. This is a very useful feature in situations where complete knowledge of the real world is unavailable, or costly to obtain: for example in robotic sensing problems, blind spot is a widely existing problem. Although the level of occlusions can be reduced by introducing more sensors, this option can be costly and usually unavailable. In these cases, a system that learns the real world without the need of labelled data is very crucial.

In the original literature, the researchers applied deep tracking on raw laser input, and achieved impressive outcome by training the system with raw data. From the raw footages, one cannot see the space and whereabouts of each object. Even for objects in the fore front, only the side facing the camera is revealed. As the results show, the model obtains information about the occluded scenes effectively, and is able to recover both seen and unseen objects.

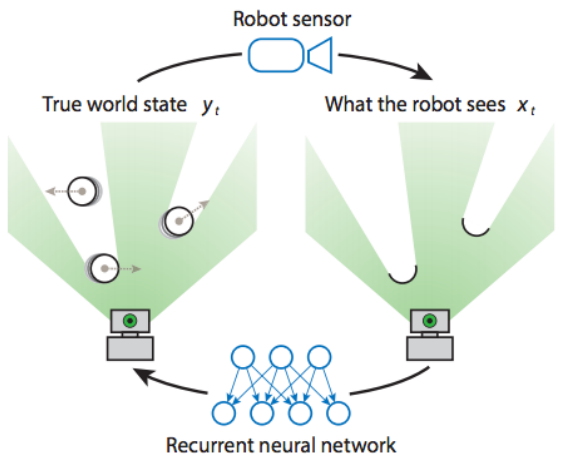

Figure 1: The real world state and the robotic sensor's perspective. Adapted from ([1])

This result has significant meaning for the following reasons: (1) the AI does not have any information about the system beforehand: the size and shape of the objects, etc. (2) the AI completely learns the information along while watching the video. (3) The recovery has very high accuracy. These make us wonder what is the learning capacity of the AI, and how sophisticated can the scenarios be in order for the AI to learn.

[1] Repository can be found at https://github.com/hzyjerry

In this project I extend the training result to objects of more complex shapes with different speeds and movements. I also experiment with different level of neural network complexities to test out the AI's learning capacities.

Related Work

The deep tracking framework proposes a novel way of modeling partially-observable real world processes. Traditional solutions to this problem heavily rely on pre-defined state representations. Bayesian filtering ([3]), a recent theoretical work which gives rise to a number of tracking methods, requires hand-designed representations and assumptions on model distribution. Other works either require additional assumption on the model, which limits its expressive power, or heavily relies on sampling. For instance, the widely applied Kalman Filter ([4]) assumes multivariate distribution, whereas Partical Filter ([5]) employs sampling based approach.

In contrast, the deep tracking model utilizes deep neural network's capability to learn and represent complex relationships. Its architecture is based on Feed-forward Recurrent Neural Networks ([6][7]), and computes inference and weight gradients using standard back-propagation learning procedure ([8]). This architecture has been proved in recent years to be very effective in processing time-series data. In this work I show that the framework also allows extension in intermediate layers using LSTM ([9]).

The training of deep tracking model using unsupervised manner is inspired by the theory of standard node dropout ([10]). The assumption is that in real world situations, future hidden states can be inferred from current observations. Because of this, if we successfully predict the future "visible area", we should have a relatively trustworthy prediction of the current "truth state".

Even though deep tracking framework is only recently proposed, studies has been widely conducted surrounding its features. Researchers have found interesting results in autonomous driving ([11]) and semantic segmentation ([2]). It is believed that this framework can lead to many potential usages. For example, future autonomous driving systems can use deep tracking to generalize to unseen objects. For space observation projects where the installation of camera lenses is the bottleneck, an end-to-end learning framework will help greatly reduce the cost.

Problem Formulation

Deep Tracking

In order to uncover the truth state y, the object movements can be formulated as Markov Process with our goal finding $P(y_t|x_{1:t})$ given $x_{t+1:t+n}$. Observed state x_i represents the location of all visible objects at time i; state y_i represent the true location of all objects at time i, both visible and invisible. Ground truth state h_i represents all the true information related to the system at time i, including object speed, acceleration, etc.

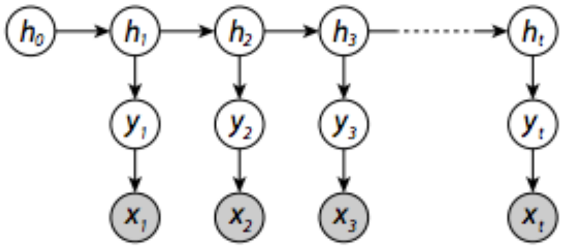

Figure 2: Hidden Markov Process. Adapted from ([1]).

The unsupervised learning challenge is we do not have knowledge of y_t as direct feedback to our prediction. Thus we rephrase the question to predict $P(y_{t+n}|x_{1:t})$. Standard node dropout theory ([10]) shows this is viable. We use Baysian Filtering for reconstruction of belief state. The belief weight matrix and iteration matrix can be modeled by neural network as learning parameters.

Synthetic Data Set

Our original literature uses synthetic laser input data. The imaginary situation is that there is a robot in the middle of a busy street surrounded by a crowd of moving people. Objects are modelled as circles independently moving with constant velocity in a random direction but never colliding with each other or with the robot. The number of objects present in the scene is varied over time between two and 12 as new objects randomly appear and later disappear at a distance from the robot.

There are two reasons for using synthetic dataset instead of real sensor input at this point. First it is hard to obtain real world object models that have the perfect round shape and can achieve uniform movements, using real world object will introduce additional complexities to the data set. Second, since Deep Tracking is a framework still at it's theoretical phase, using synthetic dataset is the most viable format that allows us to test out different hypothesis.

In this study I apply similar approach to the original deep tracking literature, by synthetizing new sensor data set. Since the original study does not provide their data generation technique, I implemented my own scripts to generate sensor recordings under different circumstances.

I use combinations of geometric shapes in space and model these shapes to move in different speed, at non-linear patterns such as cyclic and parabolic trajectories. The rationale being that given a powerful enough neural network, we should be able to interpret and the predict hidden states of the world.

Below is the format of our training sample:

```
#data
  {
     1 : 1000000, -- number of frames
     2 : 2,           -- input & output
     3 : 51,          -- width pixel count
     4 : 51,          -- length pixel count
  }
```

The dataset generated by my lua script are stored as native Torch 7 files, the size of individual set (with 1million frames) is around 7.3G, and 2.9G after compression. Due to storage concerns, they are yet to be published at this point. Latest updates can be found on my project page on Github.

Modeling: RNN structure

The deep neural network structure proposed in the original research is a four-layer recurrent neural network model. It has four layers using convolutional operations followed by sigmoid nonlinearity at every layer. Mapping consists of input-processing (the Encoder) followed by hidden state propagation (the Believe Tracker). The aim is that the Encoder will analyze sensor measurements, perform operations such as detecting directly-visible objects, and convert this information and feed it to the Belief tracker.

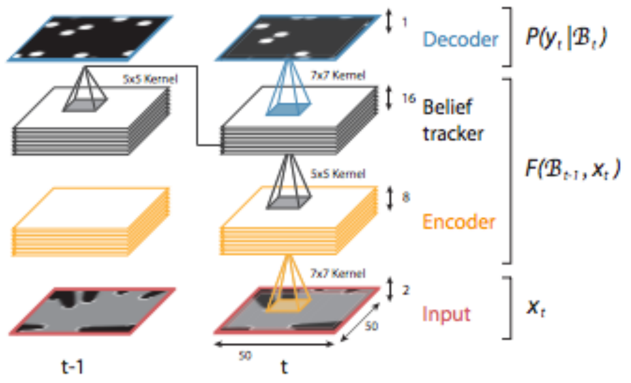

Figure 3: 4-layer recurrent neural network architecture in the original literature. Adapted from ([1]).

In our study we extend the learning capacity of our network by introducing additional 5th layer. The extra layer is set up for the purpose of tracking additional information such as different sizes, trajectories, rotations, etc. This also helps increase the model's robustness to noise.

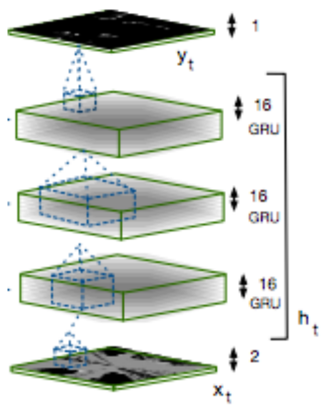

Figure 4: 5-layer architecture used for this study.

With training time being one big concern, I modified the original network by replacing sigmoid with ReLU function, which is an approach widely used in most recent deep learning project. Beside the relatively lighter computation workload, the benefit is two folds. First, using ReLU reduces the likelihood of gradient to vanish, which is an existing problem in sigmoid when the value of x becomes increasing large. Second, ReLU increases sparsity in our model, by zeroing out gradient of y with regard to x less than 0. This is more beneficial than dense representations, which is common when using sigmoid functions.

Training

Even with aforementioned modifications and improvements, the architecture takes non-trivial time to train. Using CPU of my macbook, the total training time of original 4-layer model scales up to 170hrs (~1 week). After migrating to the highest performance GPU on Amazon AWS, the training time comes down to 17 hours. In the original study, the Torch 7 based neural network takes 12 hours to train on NVIDIA Titan X GPU. For any real-time application like collision avoidance for self driving cars, this training time is apparently less than satisfactory. However, this overnight time is still feasible for post processing applications. One scenarios is that we receive a video footage of laser sensing data of object movements on Mars. It only takes overnight training time for us to model these objects from raw input.

In this study, the introduction of the extra layer increased training time from 17 hrs to 24 hrs. ReLU activation introduced slight improvement by around 1.4%, illustrated in the chart below.

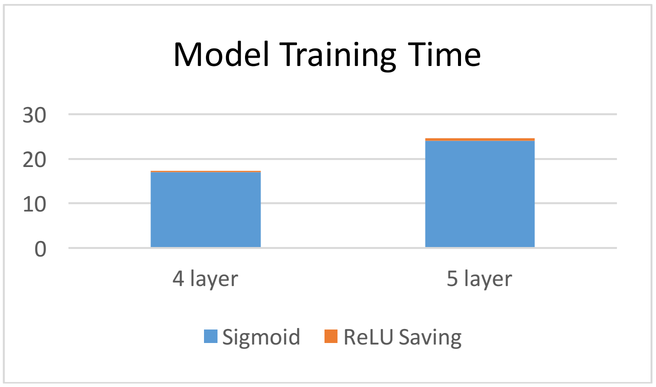

Chart 1: Comparison of multilayered model training time and the time saving from changing Sigmoid to ReLU.

Results and Discussions

In our testing, the factors subject to variations are: shape of the object, moving speed of the object (uniform or varied), trajectory of the object (straight line or round trajectory), rotation of the object (yes or no).

The comprehensive testing of these traits has very high requirement on computation time. Due to the time limit of the project, I only conducted a subset of all the available testing. The results are included in the following.

(1) Shape variation is one of the simplest factor that we can tune. Since we let all objects to have the same shape in our training data, we keep the complexity of the model at a moderate level. Compared to the original study which focuses on round object, the new shape we introduce (triangle, square) can impose non-trivial additional information because they are not rotation invariant as circles.

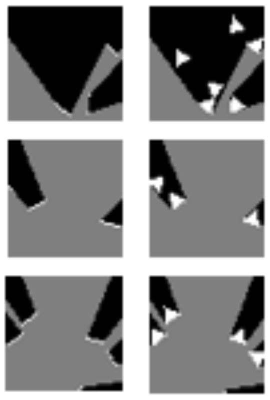

Figure 5: Recovery of triangle uniform speed straight-line object generated by Enhanced Deep Tracking

From the results shown above, we can see that the AI does surprisingly well in reconstructing the arbitrary shaped objects. Using Deep Tracking we are able to get the angles and directions of the triangle at any given time. This makes sense because for our model to "predict" future states, knowing the layout of triangles at current time is very necessary.

(2) Speed variation is another easy-to-tune variable in the data set configuration. It is also a crucial factor if we want to learn more complex models. The difficulty of predicting speed is that the model needs to very well incorporate past data with future data. For example, the speed when object is inside "blind spot" can only be inferred if we note its state before entering and after leaving the shadow area. The following is the result given by Deep Tracking.

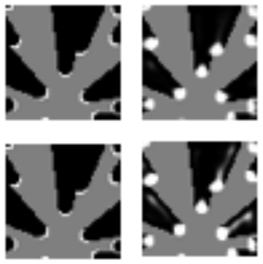

Figure 6: Recovery of round varied-speed straight-line object generated by Enhanced Deep Tracking

We can see that overall the depiction is accurate. However, for some of the objects, there is a blurred trace behind them. From the original footage, these trace are the past trajectories of the objects, and the blurring effect happens to the objects with relatively high speed. This shows us that the fast accelerated objects are harder to learn and predict.

(3) Trajectory variation. Varied trajectories allow us to introduce great amount of complexities to our model, as compared to the original straight line movements. This is a very risky factor because if blocking time is too long, the object may potentially conduct variations that will never be seen and possibly inferred by the AI. Thus in our data set we try to limit the amount of varied movements each objects can have. The following result is from data set of round objects with circular trajectories.

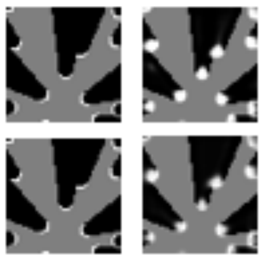

Figure 7: Recovery of round uniform speed circular-moving object generated by Enhanced Deep Tracking. (Captured at the same starting moment as Figure 6 for better comparison)

Deep Tracking framework gives us relatively satisfactory recovery of the varied movements. However, this level of complexity slightly influenced the original recovery quality. We can see from the picture above that the object outline becomes blurry.

(4) Further testing. Due to the limited space in this report, the rest of the experiments are included in the following chart for easier comparisons.

Shape	Speed	Move	Rotate	Result
round	varied	circle	no	Longer blurry trace for high speed circles
round	same	hyper-bole	yes	Some hyper-bole traces are not accur-ately recover-ed.
trian-gle	varied	line	no	Satisfactory with small increment of speed. In dra-stic changes, shapes beco-me less reco-gnizable.
train-gle	same	line	yes	Blurry recovery of triangle corners
squa-re	same	line	no	Satisfactory. Similar level as triangle.

squa-re	same	line	yes	Satisfactory under low rotational speed. When rotational sp-eed is high, becomes hard to distinguish from circle.

Summary

This study shows the practical limitation of the current Deep Tracking framework. At the same time, I demonstrate that some of the difficulties can be overcame by increasing complexity of the architecture (which leads to longer training time). Under the current state-of-the-art computation power, the framework is not suitable for real world application, but it generates results accurate enough for post-processing focused applications.

One of the biggest challenge for the study is to effectively generate data and conduct training experiments. Since each batch of training takes 24 hours on Amazon AWS, budgeting time turns out really crucial given the limited time of the quarter.

Another challenge for the study is due to the nature of unsupervised learning. For all the above reconstruction tasks, there is not a clear standard for error evaluation. This causes all the experiment results to be qualitative rather than quantitative. It also makes it difficult to compare two potential architectures that are about similar accuracy.

Future study can be conducted based on these existing difficulties. It will be of huge benefit to the subject if one could optimize the training speed of the framework, or develop a set of evaluation metrics for evaluation of recovery accuracy.

References

[1] P. Ondruska and I. Posner, "Deep tracking: Seeing beyond seeing using recurrent neural networks," in *The Thirtieth AAAI Conference on Artificial Intelligence (AAAI)*, Phoenix, Arizona USA, February 2016.

[2] P.Ondruska, J. Dequaire, D. Z. Wang, and I. Posner, "End-to-end tracking and semantic segmentation using recurrent neural networks," *arXiv preprint arXiv:1604.05091, 2016.*

[3] Yilmaz A., Javed O. and Shah M. "Object tracking: A survey, " *Acm computing surveys* (CSUR) 38(4):13.

[4] Kalman R. E., "A new approach to linear filtering and prediction problems," *Journal of Fluids Engineering* 82(1):35-45.

[5] Thrun S., Burgard W. and Fox D. "Probabilistic robotics, " *MIT press*. 2005

[6] Medsker L. and Jain L. "Recurrent neural networks," *Design and Applications*, 2001

[7] Graves A. "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*

[8] Rumelhart, D. E., Hinton G.E. and Williams R. J. "Learning internal representations by error propogation," *Technical report, DTIC Document*, 1985

[9] Hochreiter S. and Schmidhuber J. "Long short-term memory," *Neural computation* 9(8):1735-1780

[10] Srivastava N., Hinton G., Krizhevsky A., Sutskever I. and Salakhutdinov R. "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research* 15(1):1929-1958

[11] Dequaire J., Rao D., Ondurska P., Wang Z. D. and Posner I., "Deep tracking on the move: learning to track the world from a moving vehicle using recurrent neural networks," *arXiv*:1609.09365v1.