Visual Object Tracking with Deep Q Networks

by

Michael Noukhovitch

A report
presented to the University of Waterloo
in fulfillment of the
report requirement for SE 499

Waterloo, Ontario, Canada, 2016

Abstract

This is the abstract.

ABSTRACT ABSTRACT ABSTRACT ABSTRACT ABSTRACT

Acknowledgements

I would like to thank Pascal Poupart, my advisor, as well as Daniel Que, a student I worked alongside during SE 499. As well, I'd like to thank the Waterloo AI lab (MATLAB) for being nice and welcoming.

Dedication

This is dedicated to the one I love.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Artificial intelligence (or machine learning as it can be called) is a thriving field that applies intelligent algorithms to difficult problems and automate tasks. Early on, these problems were those that were difficult for humans but easy for machines. Now there has been a trend of having computers automate very human tasks that were previously deemed impossible for computers [1]. Computer vision is in line with this train of thought and deals with teaching computers to recognize and understand visuals. Visual object tracking, the question put forth here, is a computer vision task that aims to track a visual object over time (e.g. in a video). This difficult task has had great breakthroughs recently, due to the effectiveness of deep neural networks. This report explores using deep neural networks in intelligent ways to solve the task.

Chapter 2

Background Information

2.1 Deep Neural Networks

The basis of deep learning, the technology used, is neural networks. These are architectures that “learn to map a fixed-size input (for example, an image) to a fixed-size output (for example, a probability for each of several categories)” [4]. They do so by having “layers” of neurons where each neuron is a function on its input (usually the output of the previous layer). By feeding forward the inputs through many layers, complex features can be extracted by the network using interesting combinations of neurons.

By using the property that complex features tend to be built on top of simple features (e.g. using lines to form shapes), we find that having many layers of neurons allow us to learn to disinguish more and more complex features at the deeper layers [4]. This idea,

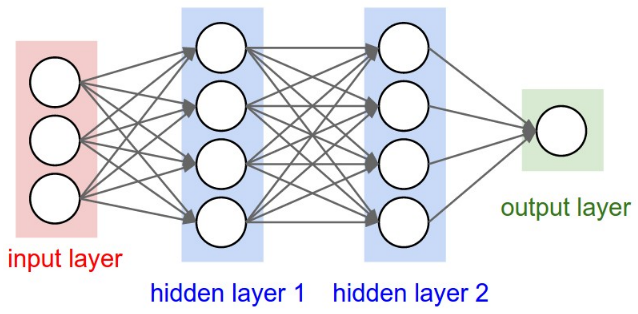

Figure 2.1: basic example of a neural network [3]

that complex features learned as a deeper layer of simple features, forms the basis for our approach to visual classification.

2.2 Convolutional Neural Networks

Just using normal neurons in combination is an effective but expensive strategy for images. If each neuron has a function that uses one weight variable per input (e.g. output is some weighted sum of inputs) then we can quickly get overwhelmed. For example, if we have a *fully connected* layer where each neuron is connected to each input, then for an image of a measly size of 32 pixels wide x 32 pixels tall x 3 colour channels (RGB), we already have 3072 weights in every input neuron! This is a very small image and more weights means more difficulty to train as well as overfitting. To this end, we use *convolutional* layers to decrease the number of parameters (weights) while still processing the input.

Based on the concept of a "sliding window" that slides over a whole image, we create a *filter* that will be our sliding window and assign it only as many weights as it needs (5x5 filter $\rightarrow$ 5x5x3 (RGB) = 75 weights). We then use this filter over the whole image and thus save on parameters [3]. We also get the benefit of locality since our window looks at groups of pixels at a time. This means that it doesn't matter where in the image our pixels are, as long as our sliding window passes over them it acts as the same input, meaning we have translational invariance.

These benefits, among others, mean that convolutional neural networks are excellent for processing images and indeed are the basis of the state of the art image object classification, video object classification, scene classification, and more [7] [2].

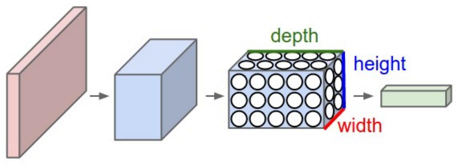

Figure 2.2: basic example of a convolutional network [3]

2.3 Reinforcement Learning

This research also makes use of another large field in machine learning, reinforcement learning. “Reinforcement learning is learning what to do–how to map situations to actions–so as to maximize a numerical reward signal.” [8] We start by modelling situations as Markov decision processes, where we specify each component of a task by modelling everything including uncertainty. We can say that a task is at a current state s that is one of the possible states in the state space S. As well, we can take an action a from this state (part of action space A) and recieve a reward for this action r [6].

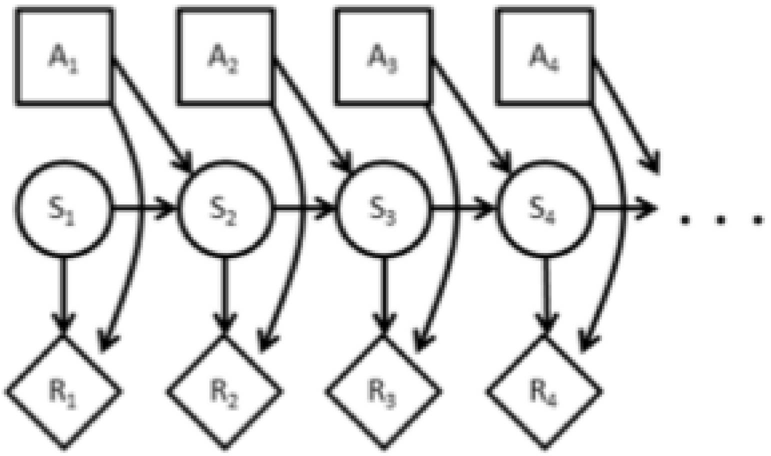

Figure 2.3: graphical model representation of an mdp[6]

We can also add the horizon h which specifies how far into the future out agent should plan, and the discount factor γ which determines how to weigh rewards at different steps in time, specifically, how the rewards decrease at later time steps.

Reinforcement learning deals with finding a policy π that will maximize our reward by choosing an action at every time step, which ends up being maximizing the “expected sum of discounted rewards earned while executing the policy” [6], for a value function $V^{\pi}(s)$

$$V^{\pi}(s) = \sum_{t=0}^{\infty} \gamma^t E_{\pi}[R(s_t, \pi(s_t))] \forall s \tag{2.1}$$

rewriting the value function to be recursive as the sum of the current reward and discounted future rewards

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} Pr(s'|s, \pi(s)) V^{\pi}(s') \forall s \tag{2.2}$$

an optimal policy is one that maximizes $V^{\pi}(s)$, so one where $V^{\pi *}(s) \geq V^{\pi}(s)\ \forall s, \pi$ [6], which means it will satisfy Bellaman's equation

$$V^{\pi *}(s) = \max_a R(s, a) + \gamma \sum_{s'} Pr(s'|s, \pi(s)) V^{\pi *}(s') \forall s \tag{2.3}$$

The question then becomes how do we find this policy π, especially if the probabilities or rewards are known only for some states? There are many approaches, but a simple one is value iteration. Value iteration which uses dynamic programming to optimize the values from the last time step and works backwards. Eventually, you arrive at the maximal value at each time step t and can find the optimal policy π. Another approach involves using monte-carlo methods instead of dynamic programming and yet other approach use temporal-difference learning. These learning methods learn from raw experience without a model (much like Monte Carlo methods) but also update their estimates of the solution as they go, bootstrapping in a sense [8].

One temporal learning algorithm that we will use is called “Q-learning”. This algorithm seeks to find an approximator to the optimal quality function $Q*$ which maps between state, action pairs and their value [8]. Using our approximation Q, we can then select the action which will lead to the highest value for this state and our policy is done! Formally, Q-learning can be written as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)][8] \tag{2.4}$$

Where α is the learning rate, used to stochastically update Q. This equation just means that Q is stochastically updated as it goes, with the improved policy at each state-action pair. The policy doesn't affect how the updates are made, Q is updated independent of the policy, but it does determine the order in which state-action pairs are visited and updated [8].

2.4 Deep Reinforcement Learning

The fields of deep learning and reinforcement learning were somewhat distinct and separate until certain breakthroughs showed that combining them can lead to powerful algorithms.

Deep reinforcement learning as done in [5] combines deep convolutional neural networks and q-learning to create a system that learns actions from raw image/video input. By saving the experiences of the agent in a dataset, called *replay memory*, we can train the neural network on batches of inputs and values to allow for learning both the Q-function and weights of the neural network at the same time. Formally the algorithm is called *deep Q-learning* and since using histories of different lengths can be difficult, we looks at fixed length representations of histories produced by a function ϕ [5].

Algorithm 2.1 Deep Q-Learning with Experience Replay [5]

```
Initialize replay memory $D$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode=1 to M do
    Initialize sequence $s_1 = x_1$ and preprocessed $\phi_1 = \phi(s_1)$
    for t=1 to $T$ do
        With probability $\eta$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \Theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \Theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \Theta))^2$ according the Q-learning
        algorithm
    end for
end for
```

This algorithm was used for playing Atari video game based on the last four frames of a current state, but we will be applying it to visual object tracking in a similar manner.

Chapter 3

The Problem

3.1 Visual Object Tracking

The challenge set forth is *visual object tracking*, which is a research area pursued in many different ways. The core of this task is to keep track of an object in a video over the length of the video. Many different datasets and approaches exist but a popular one that has emerged is the Visual Object Tracking challenge, most recently done in coordination with International Conference on Computer Vision (ICCV2015). This challenge standardizes the visual tracking task as one that "considers single-camera, single-target, model-free, causal trackers, applied to short-term tracking" [?]. Breaking that down, single camera implies that the input is a continuous video taken from a single camera (though the camera can move). Single target means that there is only one object to be tracked in the video. Model-free means that "the only supervised training example is provided by the bounding box in the first frame" [?], so the object can be anything and the input given is just a visual cue of where in the first frame the object is. Causal tracking means that the tracker can't use future frames or frames prior to re-initialization to detect the object. Finally, short-term implies that the tracker does not re-detect the object if the target is lost; drifting off of the target is considered failure.

With all these restrictions put into place, we end up with a fairly narrow and precise goal. Also useful, is that the VOT challenge provides labelled training data and this work will be using the most recent VOT 2016 training data which consists of 60 videos each containing somewhere between 50 and 1000 frames with the average being somewhere in the 300s. Each video is labelled at every frame with a rectangular bounding box known as the *ground truth*. Extra labels on the video are also provided as booleans for every frame: camera motion, illumination change, motion change, object size change.

3.2 Related Work

Since the VOT challenge has been running for three years now, there have already been many approaches to this problem from combined approaches that used traditional statistical machine learning such as kernelized correlation features (KCF) [?] to more recent approaches using neural networks [?]. VOT2015 saw many different approaches ultimately having neural networks be a part of many of the more successful entries [?].

One particularly interesting entry was the eventual contest-winner MDNet, which used a fairly standard convolutional network that reserved a final fully-connected layer for tracking specific to that video (referred to as the *domain* as MDNet stands for"Multi-Domain Network") [?]. The simple but effective convolutional layers already produced excellent results and combined with domain-specific final layers, made for a robust and accurate network. Since VOT2015 measured the networks by finding the expected average overlap between the ground truth bounding boxes and the output of the network, this robustness as well as accuracy managed to make up for he lack of speed in the network, crowning MDNet the winner [?]. These results are encouraging to our use of convolutional neural networks.

Chapter 4

The Approach

4.1 DQN for Visual Object Tracking

4.2 Implementation

4.3 Results

Chapter 5

Future Work

Chapter 6

Conclusion

APPENDICES

Appendix A

Matlab Code for Making a PDF Plot

A.1 Using the GUI

Properties of Matab plots can be adjusted from the plot window via a graphical interface. Under the Desktop menu in the Figure window, select the Property Editor. You may also want to check the Plot Browser and Figure Palette for more tools. To adjust properties of the axes, look under the Edit menu and select Axes Properties.

To set the figure size and to save as PDF or other file formats, click the Export Setup button in the figure Property Editor.

A.2 From the Command Line

All figure properties can also be manipulated from the command line. Here's an example:

```
x=[0:0.1:pi];
hold on % Plot multiple traces on one figure
plot(x,sin(x))
plot(x,cos(x),'--r')
plot(x,tan(x),'.-g')
title('Some Trig Functions Over 0 to \pi') % Note LaTeX markup!
legend('{\it sin}(x)','{\it cos}(x)','{\it tan}(x)')
hold off
```

```
set(gca,'Ylim',[-3 3]) % Adjust Y limits of "current axes"
set(gcf,'Units','inches') % Set figure size units of "current figure"
set(gcf,'Position',[0,0,6,4]) % Set figure width (6 in.) and height (4 in.)
cd n:\thesis\plots % Select where to save
print -dpdf plot.pdf % Save as PDF
```

References

[1] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[3] Andrej Karpathy. Cs231n: Convolutional neural networks for visual recognition. 2016.

[4] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, may 2015.

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[6] Pascal Poupart. An introduction to fully and partially observable markov decision processes. In Eduardo Morales Enrique Sucar and Jesse Hoey, editors, *Decision Theory Models for Applications in Artificial Intelligence: Concepts and Solutions*, chapter 3, pages 33–62. IGI Global, 2011.

[7] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[8] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998.