Deep Learning for Autonomous Driving

Shai Shalev-Shwartz
Mobileye

IMVC dimension,
March, 2016

S. Shalev-Shwartz is also affiliated with The Hebrew University
Autonomous Driving

Shai Shalev-Shwartz (MobilEye)
Autonomous Driving
Major Sub-Problems

Sensing:
- **Static objects**: Road edge, curbs, guard rails, ...
- **Moving objects**: Cars, pedestrians, ...
- **Semantic information**: Lanes, traffic signs, traffic lights, ...
Major Sub-Problems

Sensing:
- **Static objects**: Road edge, curbs, guard rails, ...
- **Moving objects**: Cars, pedestrians, ...
- **Semantic information**: Lanes, traffic signs, traffic lights, ...

Mapping:
- “Take me home”
- Foresight
- Robustness
Major Sub-Problems

Sensing:
- **Static objects**: Road edge, curbs, guard rails, ...
- **Moving objects**: Cars, pedestrians, ...
- **Semantic information**: Lanes, traffic signs, traffic lights, ...

Mapping:
- “Take me home”
- Foresight
- Robustness

Driving Policy:
- **Planning**: e.g.
  - Change lane now because you need to take a highway exit soon
  - Slow down because someone is likely to cut into your lane
- **Negotiation**: e.g.
  - Merge into traffic
  - Roundabouts, 4-way stops
Challenges

- Everything should run in real time
- Difficult driving conditions
- Robustness: No margin for severe errors
- Unpredictable behavior of other drivers/pedestrians
- Beyond “bounding box”: need to understand the entire image and must utilize contextual information
Example: Free Space

Where can I drive?
Example: Free Space

Where can I drive?  Need context!
Why Deep Learning?

Why Learning?
Manual engineering is not powerful enough to solve complex problems.
Why Deep Learning?

- **Why Learning?**
  Manual engineering is not powerful enough to solve complex problems

- **Why Deep Learning?**
  To solve hard problems, we must use powerful models
Why Deep Learning?

- Why Learning?
  Manual engineering is not powerful enough to solve complex problems
- Why Deep Learning?
  To solve hard problems, we must use powerful models
- Why Are Deep Networks Powerful?
Why Deep Learning?

- Why Learning?
  Manual engineering is not powerful enough to solve complex problems

- Why Deep Learning?
  To solve hard problems, we must use powerful models

- Why Are Deep Networks Powerful?
  **Theorem:**
  Any function that can be implemented by a Turing machine in $T$ steps can also be expressed by a $T$-depth network
Why Deep Learning?

Why Learning?
Manual engineering is not powerful enough to solve complex problems

Why Deep Learning?
To solve hard problems, we must use powerful models

Why Are Deep Networks Powerful?

- **Theorem:**
  Any function that can be implemented by a Turing machine in $T$ steps can also be expressed by a $T$-depth network

- **Generalization:**
  Deep networks are both expressive and generalizing (meaning that the learned model works well on unseen examples)
Additional Benefits of Deep Learning

- Hierarchical representations for every pixel ("pooling")
- Spatial sharing of computation ("convolutions")
- Accelerate computation by dedicated hardware ("lego")
- "Development language": by designing architectures and loss functions
- Modeling of complex spatial-temporal structures (using RNNs)
Is Deep Learning the Answer for Everything?

- Current algorithms fail for some trivial problems
  - Parity of more than 30 bits
  - Multiplication of large numbers
  - Modeling of piece-wise curves
  - ...

Main reason: Training a deep network is computationally hard, and understanding when and why it works is a great scientific mystery.

In practice: Deep learning is useful only when it is combined with smart modeling/engineering.

In practice: Domain knowledge is very helpful.

In practice: Architectural transfer only works for similar problems.

In practice: Standard training algorithms are not always satisfactory for automotive applications.
Is Deep Learning the Answer for Everything?

- Current algorithms fail for some trivial problems
  - Parity of more than 30 bits
  - Multiplication of large numbers
  - Modeling of piece-wise curves
  - ...

- Main reason: Training a deep network is *computationally hard*, and understanding when and why it works is a great scientific mystery
Is Deep Learning the Answer for Everything?

- Current algorithms fail for some trivial problems
  - Parity of more than 30 bits
  - Multiplication of large numbers
  - Modeling of piece-wise curves
  - ...
- Main reason: Training a deep network is **computationally hard**, and understanding when and why it works is a great scientific mystery

- **In practice**: Deep learning is useful only when it is combined with smart modeling/engineering
- **In practice**: Domain knowledge is very helpful
- **In practice**: Architectural transfer only works for similar problems
- **In practice**: Standard training algorithms are not always satisfactory for automotive applications

Shai Shalev-Shwartz (MobilEye)
DL for Autonomous Driving
IMVC'16
10 / 23
Example: Typical vs. Rare Cases
Typical vs. Rare Cases
Failures of Existing Methods for Rare Cases

- State-of-the-art training methods are variants of **Stochastic Gradient Descent (SGD)**
- SGD is an iterative procedure
- At each iteration, a random training example is picked
- The random sample is used to estimate an update direction
- The weights of the network are updated based on this direction
Failures of Existing Methods for Rare Cases

SGD finds an o.k. solution very fast, but significantly slows down at the end. Why?

- Rare mistakes: Suppose all but 1% of the examples are correctly classified. SGD will now waste 99% of its time on examples that are already correct by the model.
- High variance, even close to the optimum.
Requires Novel Algorithms

![Graph showing Iteration vs. % error with lines for SGD and FOL](image)

- **SGD**
- **FOL**

Shai Shalev-Shwartz (MobilEye) DL for Autonomous Driving IMVC'16
Deep Learning for Driving Policy

- Input: Detailed semantic environmental modeling
- Output: Where to drive and an what speed
Reinforcement Learning

**Goal:** Learn a policy, mapping from states to actions

**Learning Process:**
For $t = 1, 2, \ldots$
- Agent observes state $s_t$
- Agent decides on action $a_t$ based on the current policy
- Environment provides reward $r_t$
- Environment moves the agent to next state $s_{t+1}$
Reinforcement Learning vs. Supervised Learning

- In SL, actions do not effect the environment, therefore we can collect training examples in advance, and only then search for a policy.
- In SL, the effect of actions is local, while in RL, actions have long-term effect.
- In SL we are given the correct answer, while in RL we only observe a reward.
Most algorithms rely on Markovity — Next state only depends on current state and action

Yields a Markov Decision Process (MDP) — Can couple all the future into the so-called $Q$ function
Most algorithms rely on Markovity — Next state only depends on current state and action

Yields a Markov Decision Process (MDP) — Can couple all the future into the so-called $Q$ function

Inadequate for driving policy — Next state depends on other drivers
Decompose the problem into

1. Supervised Learning problems
   - Predict the near future
   - Predict the intermediate reward

2. and then explicitly optimize over the policy using Recurrent Neural Network
If we could express \( R(B, \pi \theta) \) as a differential function of \( \theta \), we could have utilized the Stochastic Gradient Descent (SGD) approach for maximizing (1). That is, starting with an initial \( \theta \), at each iteration of SGD we first sample \( B \), then we calculate the gradient of \( P_T = 1 \) \( R_t(B, \pi \theta) \) with respect to \( \theta \), and finally we update \( \theta \) based on this gradient.

Our key observation is that by solving two SL problems, described below, we can approximate \( R(B, \pi \theta) \) by a differential function of \( \theta \). Hence, we can implement SGD for learning \( \pi \theta \) directly.

The goal of the first SL problem is to learn a deep neural network (DNN), that represents the mapping from a \((state, action)\) pair into the immediate reward value. We denote this DNN by \( DNN_r \) and it is formally described as a function \( DNN_r : S \times A \to R \). We shall later explain how to learn \( DNN_r \) using SL, but for now lets just assume that we can do it and have the network \( DNN_r \) such that \( DNN_r(s_t, a_t) \approx r_t \).

The goal of the second SL problem is to learn a DNN that represents the mapping from \((state, action)\) into the next state. Formally, this DNN is the function \( DNN_N : S \times A \to S \), and for now lets assume we managed to learn \( DNN_N \) in a supervised manner such that \( DNN_N(s_t, a_t) \approx s_{t+1} \).

Equipped with \( DNN_r \) and \( DNN_N \) we can describe the process of generating a random \( B \) and calculating \( R(B, \pi \theta) \) as follows. Initially, the simulator picks a seed for its pseudo random number generator and then it determines the initial state \( s_1 \). At round \( t \), the agent receives \( s_t \) from the simulator and applies \( \pi \theta \) to generate the action \( a_t = \pi \theta(s_t) \). The simulator receives \( a_t \) and generates \( r_t \) and \( s_{t+1} \). At the same time, the agent applies \( DNN_r \) to generate \( \hat{r}_t = DNN_r(s_t) \) and applies \( DNN_N \) to generate \( \hat{s}_{t+1} = DNN_N(s_t) \). Let us denote \( \nu_{t+1} = s_{t+1} - \hat{s}_{t+1} \). Therefore, if the simulator receives \( \hat{s}_{t+1} \) it can generate \( \nu_{t+1} \) and send it to the agent.
The Deep Learning Revolution: Stunning empirical success in hard AI tasks
Existing deep Learning algorithms fail for some trivial problems
Prior knowledge is still here, it just shifted its shape
A deeper theoretical understanding of deep learning is the most important open problem in machine learning...