Autonomous Vision Group
Prof. Dr.-Ing. Andreas Geiger

Eberhard Karls
Universität
Tübingen

Self-Driving Cars

Ex. 2 – Pen and Paper (Lecture 02 + 03)

2.1 Self-driving basics

Self Driving

a) In the lecture, we have discussed 3 general approaches / paradigms to self-driving. State all 3 approaches to self-driving and specify one advantage and one disadvantage for each approach.

i) Modular Pipeline:
+ small components
+ easy to parallelize
+ interpetable
- piece-wise training
- heavily relies on HD maps
- require a lot of domain knowledge

ii) End-to-End Learning (IL/RL)
+ end-to-end -¿ trained for target objective
+ cheap annotations
+ simple
- interpretability
- RL hard to train
- generalization difficult

iii) Direct Perception:
+ compact representation
+ interpretable
- control not jointly trained
- representation manually chosen/ dificult to choose

Deep Learning

a) We pass an image of size $C = 3, W = 320, H = 320$ through the network below. Fill in values for the ?'s so that the architecture is valid. Then report the tensor size (C_{out}, H, W) after each layer when the image is passed through the network.

Layer
Conv(?, ?, 3, 1, 1)
ReLU
MaxPool(2, 2)
Conv(64, ?, 3, 1, 1)
ReLU
MaxPool(2, 2)
Conv(128, 256, 3, 1, 1)
ReLU
MaxPool(2, 2)
Conv(?, 512, 3, 1, 1)
ReLU
MaxPool(2, 2)

Layer	Output Shape
Conv(3, 64, 3, 1, 1)	(64, 320, 320)
ReLU	(64, 320, 320)
MaxPool(2, 2)	(64, 160, 160)
Conv(64, 128, 3, 1, 1)	(128, 160, 160)
ReLU	(128, 160, 160)
MaxPool(2, 2)	(128, 80, 80)
Conv(128, 256, 3, 1, 1)	(256, 80, 80)
ReLU	(256, 80, 80)
MaxPool(2, 2)	(256, 40, 40)
Conv(256, 512, 3, 1, 1)	(512, 40, 40)
ReLU	(512, 40, 40)
MaxPool(2, 2)	(512, 20, 20)

b) The network architecture above is the (simplified) convolutional part of a network called VGG16 by (Simonyan et al., 2014). This stack of convolutional and max-pooling layers is often followed by a few fully-connected layers. The first fully-connected layer of a VGG16 is a FC($C_{in} = 25088, C_{out} = 4096$). Calculate the number of trainable parameters for the first and the second convolutional layers and the first fully-connected layer.

The formulas to calculate the number of parameters:
Fully connected layer: $\#params = W * H * C_{out} * (W * H * C_{in} + 1)$
Convolutional layer: $\#params = C_{out} * (K * K * C_{in} + 1)$

i) Conv$(3, 64, 3, 1, 1)$: $\#params = 1,792$

ii) Conv$(64, 128, 3, 1, 1)$: $\#params = 73,856$

iii) FC$(25088, 4096)$: $\#params = 102,764,544$

The advantage of weight sharing becomes apparent.

2.2 Imitation Learning

a) Describe the key idea of IL. What is a policy and how can it be represented formally?

The goal of imitation learning (IL) is to train a policy that is able to mimic the behaviour of an expert. For this we need demonstrations provided by an expert that we can use as a supervision signal. The policy is a parametrized function that maps the state to actions. $\pi_\theta : \mathcal{S} \rightarrow \mathcal{A}$

b) Give two examples of possible environments and their states and actions - one where the state is fully known/ can be fully observed and one example where the state is only partly observable.

Fully observable:

- Chess (state: position of all pieces; action: next move): At every timestep we know the exact position of all pieces.
- most Atari games
- Privileged agents in a simulation where the whole state can be obtained through the simulator.

Partly observable:

- Self-driving vehicle (state: (noisy) GPS, Camera image/ Lidar of a part of the environment, ...) At no point in time it is possible to know all information about the environment.
- Card games where you don't see the cards of the opponents, e.g. poker.

c) What is the main assumption Behavior Cloning (BC) is based on? What does that mean for testing the trained policy? (See also FAQ Lecture 2 and [2])

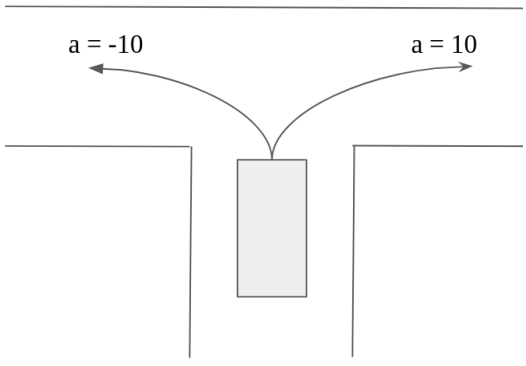
Assumption: IID (all data points are sampled independently and identically distributed). As BC is a supervised learning algorithm we need to make this assumption. But in reality this is not true. The next state that is observed depends on the current state and the executed action.

During testing (unrolling the trained policy), the policy predictions affect future states and violate the IID assumption. Intuition: The learned policy is not perfect and will deviate from the experts trajectory at some point. This leads to states that are outside of the training distribution (not observed by the expert).

d) What is the problem when using an expert driver for collecting data? How can this problem be handeled?

If we only have perfect trajectories the model can't learn to recover from states that were not observed during training. Solutions: Data augmentation, DAgger, Adding noise to the actions while collecting expert data ...

e) Consider a self-driving vehicle in front of a simple intersection (see Figure). We name this state $s_i = 0$. The goal is to train a policy that maps this state to an action a_i, which is the steering angle of the vehicle.



i) Consider this policy is trained with Behavior Cloning. What is the problem with BC in this case? If you train on data from this intersection, what does the car do during inference?

There are multiple possible actions for the exact same state. BC directly maps an state to an action. This leads to ambiguities in this case.

ii) We collect a dataset on this intersection: $D = \{(-10, 0), (10, 0), (10, 0), (-10, 0)\}$ with action-state pairs $(a_i^\star, s_i^\star)$. We train a deterministic policy that directly maps the state to the action. For this we use an L2-Loss: $\mathcal{L}\left(a_i^*, \pi_\theta(s_i^*)\right) = \left(a_i^* - \pi_\theta(s_i^*)\right)^2$.
Derive the optimal action fo the policy to take given the above dataset and loss function.

$$\hat{a} = ?$$

$$\mathcal{L}_D = \sum_{i=1}^{N} \left(a_i^* - \hat{a}\right)^2$$

$$\frac{\delta}{\delta \hat{a}} \mathcal{L}_D = 2 \sum_{i=1}^{N} \left(a_i^* - \hat{a}\right)(-1) \stackrel{!}{=} 0$$

$$\sum_{i=1}^{N} a_i^* - N\hat{a} = 0$$

$$\hat{a} = \frac{1}{N} \sum_{i=1}^{N} a_i^*$$

Here: $\hat{a} = 0$. This means the vehicle drives straight.

iii) Name an option to overcome this problem.

Conditional Imitation Learning (CIL): here the mapping to the actions not only depends on the state but we can also provide a navigation command (e.g. turn left or turn right) to the policy to resolve the ambiguity in the case of multiple plausible solutions.

f) What is the main difference between BC and CIL? Explain this based on the formulas for the objectives and assumptions.

The main difference is that BC maps only from the state to an action and CIL additionally takes a navigation command into account.

BC:

Objective:

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}\left(a_i^*, \pi_\theta(s_i^*)\right)$$

Assumption:

$$\exists f(\cdot) : a_i = f(s_i)$$

Often violated in practice!

CIL:

Objective:

$$\underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} \mathcal{L}\left(a_i^*, \pi_\theta(s_i^*, c_i^*)\right)$$

Assumption:

$$\exists f(\cdot, \cdot) : a_i = f(s_i, c_i)$$

Better assumption!

2.3 Direct Perception

Affordances

a) Give a high level description of what affordances are and name four examples in the context of self-driving.

> Affordances are attributes of the environment that can be used as an interpretable intermediate representation. Affordances limit the space of allowed actions [4].
> Examples: Distance to lead vehicle, Relative angle to road, Speed signs, Traffic lights,...

b) Which affordances are useful for i) highways ii) inner-city driving and iii) both.

> i) Indicator for on-ramp/ off-ramp lanes, Signs that indicate construction work or traffic jam
> ii) Traffic lights, Hazard stop (pedestrians)
> iii) Distance to centerline, Relative angle to road, Distance to lead vehicle

Visual Abstractions

a) What are relevant properties of visual abstractions?

> Invariant (hide irrelevant variations from policy)
> Universal (applicable to wide range of scenarios)
> Data efficient (in terms of memory/computation)
> Label efficient (require little manual effort)

b) Semantic segmentation can be used as an intermediate representation. Behl et.al [3] show that the performance can improve when using i) less classes, ii) coarsely anotated images. What can be a reason for this?

> i) When using only the relevant classes it is easier for the model to map the visual abstraction to an action since it doesn't need to learn to ignore the irrelevant classes.
> ii) In many cases the precise shape of an object is not relevant for the model. (E.g. it doesn't matter how a person in front of the car is shaped it just matter that there is a vehicle). Since objects are represented by bounding boxes, it is easier for the model to learn this representation. This leads to less noise compared to a standard segmentation model and therfore it is easier to map this to an action. Using only coarse annotation the labelling process is much faster.

c) In order to train a visual abstraction model we need two datasets: n_r (images annotated with semantic labels) and n_a (images annotated with expert actions). Why do we assume $n_r << n_a$? Do you have an intuition why it is possible to use less data to train the visual abstraction model?

> We assume $n_r << n_a$ because obtaining action labels is much cheaper than obtaining semantic labels. Intuition: Object detection or segmentation is an easier task then mapping images to actions. Think of a traffic light: you always have some red pixels in your image and the variation is limited. But if you map from an image to the action 'brake' there is much more variation and it is not only a red light that can lead to the action 'brake', therfore more data is necessary.

Evaluation

a) Describe the difference between online and offline evaluation. Name three metrics respectively.

> **Online evaluation** unrolls the policy on a real (or simulated) car. Using a real vehicle is expensive and dangerous. Metrics: Success rate, Average completion, km per infraction.
> **Offline evaluation** uses a pre-recorded dataset with states and corresponding expert actions. Metrics: Squared error, Absolut error, Quantized classification error ...

b) Why are offline metrics not always good proxies of online driving performance?

As shown in the case study [5] discussed in the lecture, large errors in the steering prediction can lead to a crash even if it is only for a a few timesteps. But in the offline metrics, e.g. average prediction error it is hard to detect this failure (one strong mistake is considered similar to continous noise).

2.4 References

[1] https://arxiv.org/pdf/1604.07316.pdf
[2] https://arxiv.org/pdf/1011.0686.pdf
[3] http://www.cvlibs.net/publications/Behl2020IROS.pdf
[4] http://proceedings.mlr.press/v87/sauer18a/sauer18a.pdf
[5] https://arxiv.org/pdf/1809.04843.pdf