Generative Adversarial Imitation Learning for Autonomous Driving

TEAM 14

Venkatarao Rebba
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ, USA
vrebba@asu.edu

Surya Kiran Cherupally
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ, USA
Scherup1@asu.edu

Elena Oikonomou
School of Computing and Augmented Intelligence
Arizona State University
Tempe, AZ, USA
e.oikonomou@asu.edu

***Abstract*—The aim of this project is to build an autonomous driving agent under the guidance of an expert using Generative Adversarial Imitation Learning (GAIL). To develop the Imitation Learning algorithm, we first need to obtain the expert policies. These policies can be extracted by either a human expert driving the car or by training a Reinforcement Learning (RL) agent. In this work, in order to produce the expert trajectories, we trained a RL agent. Therefore, this project consists of two separate components: Component 1 - Generation of expert trajectories via RL and Component 2 - Training of a race car via Imitation Learning. We begin by training the expert with two different RL methods (Deep Q-Learning and PPO) and then train an agent with Imitation Learning (GAIL) that tries to complete its goal of finishing a race by imitating the expert on the OpenAI Gym environment. This group project was developed as part of the course "EEE598 Reinforcement Learning in Robotics" at ASU.**

***Index Terms*—RL, autonomous driving, imitation learning, GAIL, OpenAI Gym**

I. Introduction

This project focuses on the development of an autonomous driving race car that operates on the OpenAI Gym environment [4]. The aim of this work is to make the race car learn an optimal policy in order for it to drive autonomously from demonstrations by an expert through imitation learning and more specifically, by using the Generative Adversarial Imitation Learning (GAIL) [1] method.

Imitation Learning is a machine learning technique where an agent learns to perform a task by imitating an expert (typically a human). This technique can be advantageous in cases where the rewards are sparse (e.g., whenever a reward is only received at the time a game is terminated) or a direct reward function does not exist and must be manually designed (e.g., when teaching a self-driving vehicle). However, designing a reward function manually that produces the desired behaviour can potentially be exceedingly complicated. Therefore, in Imitation Learning, instead of trying to learn from sparse rewards or manually specifying a reward function, an expert provides a set of demonstrations and the agent tries to learn the optimal policy by imitating the expert's decisions.

In general, Imitation Learning is useful when it is easier for an expert to demonstrate the desired behavior, rather than to specify a reward function which would create that behavior.

There are two main approaches in Imitation Learning:

- Behavioral Cloning
  - learns a policy as a supervised learning problem that maps state-action pairs to a policy
  - requires a large number of expert trajectories
  - copies actions exactly, even if they are not important to the task
- Inverse Reinforcement Learning (IRL)
  - learns the reward function from expert trajectories, then derives the optimal policy
  - is expensive to run

Generative Adversarial Imitation Learning (GAIL) was developed from the desire to have an algorithm that explicitly defines how an agent acts by directly learning a policy. It uses demonstration data by experts and learns both the unknown environment's policy and reward function. GAIL is not exactly IRL, due to the fact that it is learning the policy, not the reward function, directly from the data. GAIL performs better than behavioral cloning and sometimes even better than the experts, because it is performing Reinforcement Learning and is not constrained to be always close to the expert.

It is a model-free imitation learning framework that directly learns a policy from example expert behavior, without interacting with the expert or having access to a reinforcement signal. It is closely connected to Generative Adversarial Networks (GANs) and employs generative adversarial training to fit distributions of states and actions that define expert behavior.

The generative adversarial training corresponds to a two-player game where a Generator network and a Discriminative network are trained simultaneously. The Generator network is trying to fool the Discriminator by generating expert-looking trajectories and the Discriminator network is trying to distinguish whether the trajectories are generated by the expert or not.

To develop the Generative Adversarial Imitation Learning (GAIL) algorithm, we first need to obtain the expert demonstrations, which are in the form of expert policies. These expert policies can be extracted by either a human expert driving the car or by training a RL agent. In this project, in order to produce the expert trajectories, we will train a RL agent.

Therefore, this project consists of the following two separate components:

1) Generation of expert trajectories through RL.
2) Training of a race car via Imitation Learning with GAIL, by using the expert trajectories from Component 1.

The problem formulation for each component is defined in subsections A, B.

A. Component 1 - Generation of expert trajectories via RL

Problem statement: We are trying to make the car stay on the track while finishing the race as soon as possible, by controlling the steering angle, throttle and brake, based on observing the position of the car in the image.

- State space: Images (each state is a 96×96 pixel image)
- Action space: $A = \{s, t, b\}$, where:
  $s \in [-1, 1]$ is the steering angle,
  $t \in [0, 1]$ is the throttle and
  $b \in [0, 1]$ is the brake.
- Reward Function: -0.1 for every frame, +1000/N for every track tile visited (N: total # of tiles in track) and -100 & die if the car goes outside the border.
- Dynamics: OpenAI Gym simulator

B. Component 2 - Train of a race car via Imitation Learning

Problem statement: We are trying to minimize the divergence from the expert trajectories (so the car stays on the track while it finishes the race as soon as possible), by controlling the steering angle, throttle and brake, based on the generated expert policies produced in Component 1.

- State space: Images (each state is a 96×96 pixel image)
- Action space: $A = \{s, t, b\}$, where:
  $s \in [-1, 1]$ is the steering angle,
  $t \in [0, 1]$ is the throttle and
  $b \in [0, 1]$ is the brake.
- Reward Function: Will be learned from GAIL
- Dynamics: Model-free imitation learning

We observe that both Components 1 and 2 have:

- Discrete State Spaces-Continuous Action Spaces
- 96×96=9216-Dimensional State Spaces
- 3-Dimensional Action Spaces

II. METHODS

A. Deep Q-Learning (DQN)

Deep Q-Learning is a Reinforcement Learning method which uses a neural network to approximate the Q-value function for each state-action pair. In traditional Q-Learning, a Q-table is maintained which stores the q-values for each state-action pair. However, in many problems, the number of states and/or actions is so large, hence, is very inefficient to store such tables since the amount of memory needed becomes exceedingly large. In addition, it becomes very time-consuming to explore each state-action pair to update the table. Therefore, the main idea is to use a function approximator in order to estimate the action-value function. In deep Q-learning this is done with a deep neural network, called Deep Q-Network (DQN). DQNs allow for high-dimensional state spaces since there is no need to store the values in a Q-table like in Q-Learning.

B. Proximal Policy Optimization (PPO)

Proximal Policy Optimization is a Reinforcement Learning policy gradient method. It iterates over sampling data by interacting with the environment and optimizing a "surrogate" objective function using stochastic gradient ascent. This method has a number of the benefits of Trust Region Policy Optimization (TRPO) but is a lot simpler to implement and can be applied to more general cases. TRPO maximizes the following "surrogate" objective function:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta\left(a_t \mid s_t\right)}{\pi_{\theta_{\text{old}}}\left(a_t \mid s_t\right)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right]$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio, so $r(\theta_{\text{old}}) = 1$, π_θ is a stochastic policy, θ_{old} is the vector of policy parameters before the policy update and $\hat{A}_t$ is an estimator of the advantage function at timestep t. Maximizing L^{CPI} without any constraints would create a very large policy update, so the authors of PPO suggested the following objective that penalizes policy changes that move $r_t(\theta)$ away from 1 (i.e. if the new policy is far from the old one), by clipping the probability ratio in order to make it stay inside the interval $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyperparameter (e.g. $\epsilon = 0.2$):

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \operatorname{clip}\left(r_t(\theta), 1 - \epsilon, 1 + \epsilon\right) \hat{A}_t \right) \right]$$

C. Generative Adversarial Imitation Learning (GAIL)

In order to prevent overfitting of IRL algorithms, a cost function regularizer ψ is often introduced. While a constant regularizer is shown to lead to imitation learning algorithms that match occupancy measures exactly, it is intractable in large environments. On the contrary, indicator regularizers, lead to algorithms that do not exactly match occupancy measures without careful tuning, but are tractable in large environments. The authors of [1] proposed a regularizer ψ_{GA} that combines the advantages of the previous two. This regularizer places a low penalty on cost functions c that assign a large negative cost to expert state-action pairs and a big penalty if the function assigns small negative costs (close to zero, which is the upper bound) to the expert.

$$\psi_{\text{GA}}(c) \triangleq \{ \begin{array}{ll} \mathbb{E}_{\pi_E}[g(c(s,a))] & \text{if } c < 0 \\ +\infty & \text{otherwise} \end{array} \qquad (1)$$

$$\text{where } g(x) = \begin{cases} -x - \log\left(1 - e^x\right) & \text{if } x < 0 \\ +\infty & \text{otherwise} \end{cases}$$

A property of ψ_{GA} is that it constitutes an average over the expert data, and thus can adjust to any arbitrary expert dataset. According to the authors, the choice of this regularizer was motivated by the fact that:

$$\psi^*_{\mathrm{GA}}\left(\rho_\pi - \rho_{\pi_E}\right) = \max_{D\in(0,1)^{S\times A}} \mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1-D(s,a))] \quad (2)$$

where the maximum ranges over the discriminative classifiers $D : S \times A \to (0,1)$ and $\rho_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the policy's occupancy measure, where $\rho_\pi(s,a) = \pi(a \mid s) \sum_{t=0}^{\infty} \gamma^t P\left(s_t = s \mid \pi\right)$, which can be interpreted as the distribution of state-action pairs that an agent encounters when navigating the environment with policy π. Equation 2 is the optimal negative log loss of the binary classification problem of distinguishing between state-action pairs of a policy π and the expert policy π_E. If we let H be the causal entropy, that can be thought of as a policy regularizer which is controlled by $\lambda \geq 0$, then, the imitation learning algorithm becomes:

$$\min_\pi \psi^*_{\mathrm{GA}}\left(\rho_\pi - \rho_{\pi_E}\right) - \lambda H(\pi) = D_{\mathrm{JS}}\left(\rho_\pi, \rho_{\pi_E}\right) - \lambda H(\pi) \quad (3)$$

This algorithm obtains a policy whose occupancy measure minimizes the Jensen-Shannon divergence to the expert's policy. Eq. (3) shows the relation between Imitation Learning and Generative Adversarial Networks (GANs). GANs train a generative model G by trying to confuse a discriminative classifier D. The job of D is to distinguish between the distribution of data generated by G and the true data distribution. When D cannot distinguish the data generated by G from the true data, then G has successfully matched the true data. In the setting of Imitation Learning, the learner's occupancy measure ρ_π is analogous to the data distribution generated by G and the expert's occupancy measure ρ_{π_E} is analogous to the true data distribution. The Generative Adversarial Imitation Learning (GAIL) algorithm solves equation (3) by finding a saddle point (π, D) of the following expression:

$$\mathbb{E}_\pi[\log(D(s,a))] + \mathbb{E}_{\pi_E}[\log(1-D(s,a))] - \lambda H(\pi) \quad (4)$$

Initially, a function approximation for π and D is constructed by fitting a parameterized policy π_θ with weights θ and a discriminator network $D_w : S \times A \to (0,1)$ with weights w. Then, the algorithm alternates between an Adam gradient step on w to increase Eq.(4) with respect to D and a TRPO step on θ to decrease Eq.(4) with respect to π, which does not allow the policy to drastically change due to noise in the policy gradient. The algorith for GAIL can be seen in Algorithm 1.

Algorithm 1 Generative adversarial imitation learning

```
1: Input: Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters $\theta_0, w_0$
2: for $i = 0, 1, 2, \dots$ do
3:     Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4:     Update the discriminator parameters from $w_i$ to $w_{i+1}$ with the gradient
           $\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1 - D_w(s,a))]$
5:     Take a policy step from $\theta_i$ to $\theta_{i+1}$, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with
           $\hat{\mathbb{E}}_{\tau_i}[\nabla_\theta \log \pi_\theta(a \mid s) Q(s,a)] - \lambda \nabla_\theta H(\pi_\theta)$,
       where $Q(\bar{s}, \bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}]$
6: end for
```

III. Implementation and Simulation

A. Component 1 - Generate expert trajectories via RL

In order to produce the expert trajectories that are required in GAIL (Component 2), the race car was trained to drive autonomously with two different RL algorithms on the OpenAI Gym simulator: Deep Q-Learning (DQN) and Proximal Policy Optimization (PPO).

1) Deep Q-Learning (DQN): Deep Q-Learning cannot be applied to continuous domains since its objective is to find the specific action that maximizes the action-value function. Therefore, since our problem consists of a continuous action space, the actions were discretized so to be able to use a DQN. The discretized action space consists of 12 actions. These are: 3 actions for the steering wheel (left, straight, right), 2 actions for the throttle (full throttle, no throttle) and 2 actions for the break (20% break, no break).

2) Proximal Policy Optimization (PPO): In this approach, we are using 6 convolutional layers each followed by a ReLU activation function and two fully connected layers followed by ReLU and Softplus activation functions for each of the actor and critic. The first convolutional layer has a 3×3 kernel and the rest use a 4×4 kernel. Since the PPO method can be used for continuous domains, no action discretization was needed in this case. However, to reduce the complexity of the state space, the images were converted to gray scale and four consecutive images were stacked together. The CNN architecture for training the expert is shown in 'Fig. 1'.

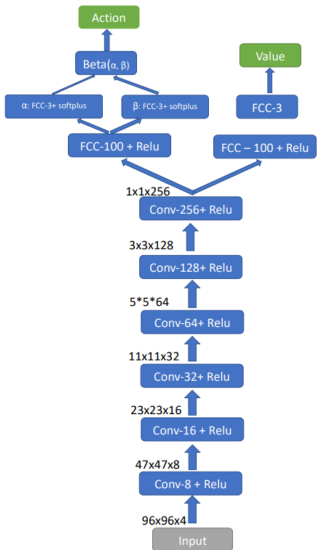

Fig. 1. CNN model architecture

B. Component 2 - Train a race car via imitation learning

For the Generative Adversarial Imitation Learning algorithm, two distinct neural networks are trained. One network corresponds to the Discriminator which utilizes an Adam optimizer and performs stochastic gradient ascent and the other network corresponds to the Generator and performs stochastic gradient descent by using the PPO method. The networks consist of two convolutional layers, followed by a ReLU activation function and two fully connected layers.

The main idea for GAIL is depicted in 'Fig. 2'.

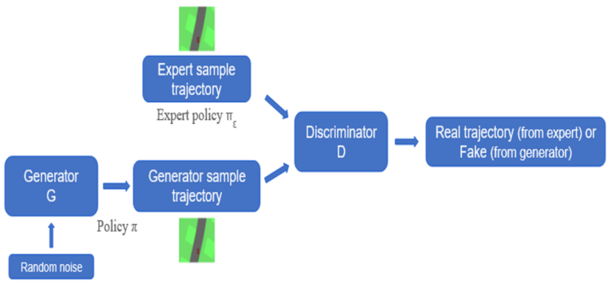

Fig. 2. GAIL diagram

IV. Results

1) Deep Q-Learning (DQN):

In the Deep Q-Learning (DQN) implementation, the expert was trained for 600 episodes. The moving average reward (with a window of 100 samples) over the last 100 episodes that the agent achieved was a little less than 600. This is shown in "Fig. 3". The agent, however, was able to achieve scores as high as 800.

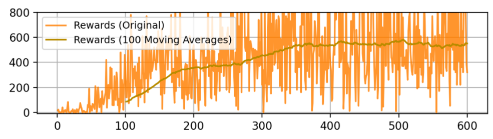

Fig. 3. Rewards for each training episode - DQN

2) Proximal Policy Optimization (PPO):

The rewards for each episode that our expert agent achieved during training are shown in "Fig. 4". The moving average reward (with a window of 100 samples) over the last 100 episodes that the agent achieved was 707. The expert was trained for about 4000 episodes and was able to achieve scores as high as 933.

Once, the expert was fully trained, we tested its performance in 10 different races (episodes). The results that our expert agent achieved are shown in "Fig. 5". The average score over these 10 episodes was 786 while the agent was able to achieve scores as high as 909. "Fig. 6" shows the race car successfully driving autonomously on the simulation environment as it is able to take the sharp turn and achieve a high score.

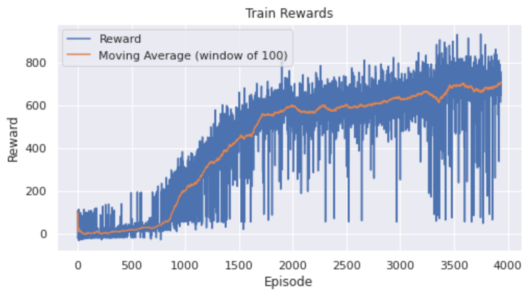

Fig. 4. Rewards for each training episode - PPO

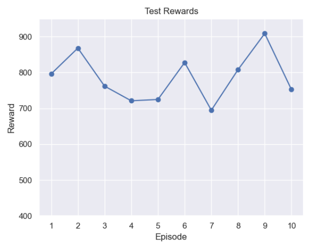

Fig. 5. Expert rewards for 10 test episodes - PPO

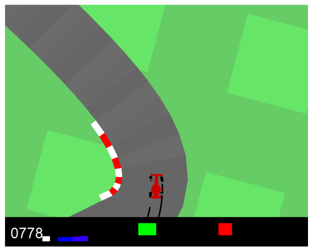

Fig. 6. The race car driving autonomously on OpenAI Gym

3) Generative Adversarial Imitation Learning (GAIL):

The imitation agent was trained for 7000 episodes and the moving average reward (with a window of 100 samples) over the last 100 episodes that the agent achieved was 610. The rewards the imitation agent achieved during training can be seen in "Fig. 7".

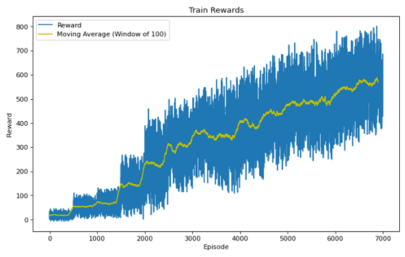

Fig. 7. Rewards for each training episode - GAIL

Again, once the agent was fully trained, we tested its performance in 10 different races (episodes). The results that our imitation agent achieved are shown in "Fig. 8". The average reward over these 10 episodes was 616 while the agent was able to achieve rewards as high as 795.

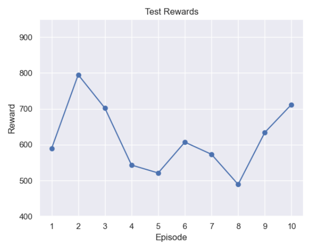

Fig. 8. Rewards for 10 test episodes - GAIL

V. Discussions & Conclusions

Table I summarizes the results for each of the three methods and "Fig. 9" provides a visual representation. Comparing the two methods used for training the expert via RL, the PPO method produced better results but the DQN needed far fewer training episodes to achieve good results. However, there are many parameters that play a significant role on the performance of each method. Moreover, for the DQN method we had to discretize our action space whereas with PPO the actions were continuous. Therefore, we cannot conclude that one method is better than the other, even for this specific problem.

We observe that there is great oscillation in the performance during training for both methods. We hypothesize that this is behavior is due to the race car trying different methods to increase its score which are not always correct (e.g. the car might decide to try and cut corners in order to save time or spin in place to avoid going out of the track).

We also see that the PPO method outperformed the imitation agent, but by adjusting the hyper-parameters of each network, we could possibly get much different results. Thus, again, we cannot draw any conclusions about which method is better.

TABLE I
Results for each method

	Average Reward	*Maximum Reward*
Expert - DQN	600	800
Expert - PPO	786	909
Imitation Agent - GAIL	616	795

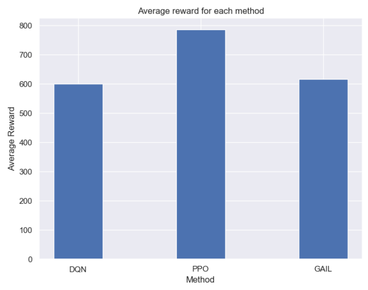

Fig. 9. Average rewards for each method

References

[1] J. Ho and S. Ermon, "Generative adversarial imitation learning," in Proceedings of the 30th International Conference on Neural Information Processing Systems, Red Hook, NY, USA, Dec. 2016, pp. 4572–4580.
[2] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, "Proximal Policy Optimization Algorithms," in
[3] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning," arXiv:1312.5602 [cs], Dec. 2013, [Online]. Available: http://arxiv.org/abs/1312.5602
[4] Brockman, G. et al., 2016. OpenAI Gym. arXiv preprint arXiv:1606.01540. (https://gym.openai.com/)
[5] https://github.com/andywu0913/OpenAI-GYM-CarRacing-DQN
[6] https://github.com/xtma/pytorch_car_caring

VI. Statement of Contributions

Venkatarao Rebba

- Read the main reference paper [1].
- Read about TRPO and searched for implementations.
- Worked on a TRPO implementation for training the expert but observed poor performance and so it was not included.
- Produced Fig. 1.
- Worked on GAIL implementation.
- Run the GAIL code and produced Fig. 7.

Surya Kiran Cherupally

- Read the main reference paper [1].
- Read reference paper [3] and searched for implementations.
- Worked on the DQN implementation using Tensorflow for training the expert.
- Run the DQN code and produced Fig. 3.
- Worked on GAIL implementation.

Elena Oikonomou

- Read the main reference paper [1].
- Read reference paper [2] and searched for implementations.
- Worked on the PPO implementation using PyTorch for training the expert.
- Run the PPO code and produced Fig. 4.
- Tested the PPO final model and produced Fig. 5, 6.
- Worked on GAIL implementation.
- Produced Fig. 8, 9.
- Created this report in its entirety.

This table is to be included in the final report as required in the final project assignment
(this table does not contribute towards the page count)
Please don't change the format of the table, i.e., only fill in the blanks
end of semester reflection - lessons learned from working on the final project
Team #14 Venkatarao Rebba, Surya Kiran Cherupally, Elena Oikonomou

Project title

	literature (not well written or self-contained, not specific on implementation, no data source indicated, no source code indicated…)	setting up the environment and obtaining data	to have the first successful test run (issues during debugging, compatibility problems…)	obtaining results (algorithm/method is dificult to implement, hyper parameters difficult to tune...)	obtaining results (cannot duplicate what was reported in paper, if so, why?)	reporting (Intro, method, result, discussions, …)
specific & detailed evidence is required to support claims (e.g., links, repository sites, equation #, figure #, paragraphs, sections, etc…)	Even though the chosen literature was well written, we found ourselves reading many other resources as well in order to understand how to implement the methods.	We initially had to figure out how to use the simulation environment. It turned out not to be hard but the documentation was not very informative and so we had to use instructional videos online.	We tested many environments for running our code. Since in our laptops the training process was slow or not possible at all (e.g. when using tensorflow), we also tried to use Agave and Google Colab. In Agave, we had to wait for many hours in order to get an active session and since Colab does not render gym, it was very difficult to make the code run there. However, after a lot of effort, we managed to solve this compatibility issue.	Hyperparameters play a very important role in the performance of the system. We observed that by making poor choices the performance of our agent droped a lot and this trial and error procedure was very time consuming. We also faced some problems obtaining results when Colab runtime kept disconnecting and we had to train from scratch.	We had difficulties training the imitation agent in the beggining. We then increased the expert trajectories samples and number of epochs, tuned some parameters better and eventually we were able to duplicate the method and obtain some good results.	We did not face any particular problems when creating our report.

footnote (e.g., reference citation…)