
Teachable machine learning
Mateusz Mierzejek | imatt.space

SUMMARY

Teachable machine learning is a rapidly growing field that seeks to make the process of training machine learning models more accessible to non-experts. Through the use of user-friendly interfaces and intuitive algorithms, teachable machine learning allows individuals with little or no programming experience to train their own machine learning models to perform specific tasks. This ability to democratize the process of training machine learning models has the potential to greatly expand the number of people who can use and benefit from this powerful technology. In this paper, we will explore the principles of teachable machine learning and discuss its potential applications and limitations.

CONTENTS

INTRODUCTION

Teachable machine learning is a subfield of artificial intelligence that focuses on making the process of training machine learning models more accessible to non-experts. Using user-friendly interfaces and intuitive algorithms, teachable machine learning allows individuals with little or no programming experience to train their own machine learning models to perform specific tasks. This ability to democratize the process of training machine learning models has the potential to greatly expand the number of people who can use and benefit from this powerful technology.

INNOVATION

One of the key innovations of teachable machine learning is the use of interactive interfaces that allow users to provide feedback and guidance to the machine learning model during the training process. This helps to ensure that the model is learning the desired behavior and can make the training process more efficient. Additionally, teachable machine learning algorithms often incorporate techniques that help to improve the interpretability of the trained model, making it easier for users to understand how the model is making its predictions.

APPLICATION

Potential applications of teachable machine learning include tasks such as image and speech recognition, natural language processing, and predictive modeling. This technology has the potential to greatly expand the range of individuals and organizations who can use machine learning, from small businesses and hobbyists to researchers and educators.

LIMITATIONS

However, there are also limitations to teachable machine learning. One of the main challenges is that, while these systems may be easy to use, they may not be as powerful or accurate as models trained by expert users. Additionally, the quality of the trained model will depend on the quality and quantity of the data provided by the user, which may be limited or biased.

ETHICS

Ethical concerns surrounding teachable machine learning include issues such as bias in the data used to train the model, the potential for misuse of the technology, and the need for transparency in the decision-making process of the trained model. Additionally, there is the potential for individuals with little or no understanding of machine learning to create models that could have harmful consequences if used incorrectly. It is important for the development of teachable machine learning to consider and address these ethical concerns in order to ensure that this technology is used responsibly and for the benefit of society.

REINFORCED LEARNING

Reinforcement learning is a type of machine learning that involves training a model to make decisions in an environment. The goal is to train the model to take actions that maximize a reward. The model learns through trial and error, receiving a positive reward for actions that lead to the desired outcome, and a negative reward for actions that don't. This type of learning is often used to train intelligent agents to play games or simulate real-world situations.

BENEFITS

One interesting thing about reinforcement learning is that it can be used to train models to solve complex problems that are difficult to solve using other machine learning techniques. For example, reinforcement learning has been used to train models to play games like Go and chess at a superhuman level.

Another interesting aspect of reinforcement learning is that it can be used to train models to take actions in the real world. This has led to the development of robots and other intelligent systems that can interact with their environment in complex ways.

Additionally, reinforcement learning is a type of unsupervised learning, which means that it doesn't require a large amount of labeled data to train a model. This is different from supervised learning, where the model is trained on a labeled dataset and then makes predictions on new data. Unsupervised learning can be more efficient and can allow the model to learn from a wider range of data.

CHALLENGES

One downside of reinforcement learning is that it can be difficult to design the right reward function for a given problem. The reward function defines what the model should optimize for, and if it is not designed properly, the model may not learn the desired behavior. This can be a challenge for complex real-world problems, where it can be difficult to define a clear reward function.

Another downside of reinforcement learning is that it can be computationally intensive. Training a model using reinforcement learning often requires a lot of data and a lot of trial and error, which can be time-consuming and require a lot of computational resources. This can make it difficult to apply reinforcement learning to problems that require real-time decision making**.**

Additionally, reinforcement learning algorithms can sometimes produce unpredictable results. Since the model is making decisions based on trial and error, it may make decisions that are hard to explain or that don't align with human expectations. This can make it difficult to use reinforcement learning in safety-critical applications where the decisions made by the model need to be reliable.

TECHNICAL ASPECTS

The learning process is typically modeled as a Markov Decision Process (MDP), in which the agent interacts with the environment by taking actions and receiving rewards. The goal of the learning process is to find the optimal policy, which is a mapping from states to actions that maximizes the expected long-term reward. The value function is used to evaluate the quality of different states and actions, and to guide the learning process towards the optimal policy.

TERMINOLOGY

- **Agent**: The agent is the entity that is being trained to make decisions. It can be a robot, a computer program, or any other system that can take actions in an environment.
- **Environment**: The environment is the world in which the agent operates. It could be a physical environment, like a robot's workspace, or a virtual environment, like a computer game.
- **State**: The state is the current situation that the agent is in. It includes all the information that the agent needs to decide.
- **Action**: An action is a decision that the agent makes in response to a given state. It could be a physical movement, like moving a robot arm, or a virtual action, like making a move in a game.
- **Reward**: The reward is a scalar value that the agent receives in response to taking an action. It is used to evaluate the quality of the action and to guide the learning process.
- **Policy**: The policy is a function that maps states to actions. It defines the behavior of the agent and determines what actions the agent will take in response to different states.
- **Value function**: The value function is a measure of how good a particular state or action is. It is used to evaluate the quality of different states and actions, and to guide the learning process.

RESULT DRIVEN

The optimal policy in reinforcement learning is achieved by using a value function to evaluate the quality of different states and actions. The value function is a measure of how good a particular state or action is, and it is used to guide the learning process towards the optimal policy.

There are different ways to define the value function, depending on the specific problem and the goals of the learning process. For example, the value function could be defined as the expected long-term reward that the agent will receive for taking a particular action in each given state. This would be used to train the agent to maximize its long-term reward.

To find the optimal policy, the reinforcement learning algorithm will iteratively update the value function and the policy. This typically involves estimating the value function using a sample of the agent's experiences, and then using the value function to update the policy. This process continues until the value function converges to the optimal value function, at which point the corresponding policy is the optimal policy.

Finding the optimal policy in reinforcement learning can be a challenging problem, and it may require a large amount of data and computational resources. However, once the optimal policy is found, the agent can use it to make decisions that maximize its reward in the given environment.

ALGORITHMS

There are many algorithms that can be used for reinforcement learning, and the choice of algorithm will depend on the specific problem and the goals of the learning process. Some of the most common algorithms used in reinforcement learning include the following:

- **Dynamic programming**: Dynamic programming is a method for solving optimization problems by breaking them down into smaller subproblems and solving them recursively. It can be used in reinforcement learning to find the optimal policy by solving the Bellman equations, which describe the relationship between the value function and the policy.
- **Policy gradient methods**: Policy gradient methods are a class of reinforcement learning algorithms that directly optimize the policy by using gradient descent to update the policy parameters. This allows the algorithm to learn complex policies that cannot be represented using a simple value function.
- **Monte Carlo methods**: Monte Carlo methods are a class of algorithms that use random sampling to solve problems. In reinforcement learning, Monte Carlo methods can be used to estimate the value function by sampling the agent's experiences and computing the expected return for each state or action.
- **Temporal difference learning**: Temporal difference learning is a method for updating the value function based on the difference between the expected return and the actual return. It can be used in reinforcement learning to quickly update the value function as the agent interacts with the environment, allowing the learning process to be more efficient.
- **Actor-critic methods**: Actor-critic methods are a class of reinforcement learning algorithms that use two separate models to represent the policy and the value function. The actor model is used to represent the policy, and it is trained to take actions that maximize the expected reward. The critic model is used to evaluate the quality of the actions taken by the actor, and it is trained to predict the value of each state or action.
- **Q-learning**: Q-learning is a model-free reinforcement learning algorithm that uses the Q-function to estimate the value of taking a particular action in each given state. The Q-function is updated using the Bellman equation, and the optimal policy is found by selecting the action with the highest Q-value for each state.
- **Deep Q-network**: A deep Q-network (DQN) is a reinforcement learning algorithm that uses a deep neural network to approximate the Q-function. It allows the algorithm to handle high-dimensional state spaces and complex environments, and it has been used to train agents to play games like Atari and Go.

ENVIRONMENTS

There are many environments where you can experiment with reinforcement learning, and the choice of environment will depend on your specific interests and goals. Some of the most common environments for reinforcement learning include the following:

- **Simulated environments**: Many reinforcement learning algorithms are tested and developed using simulated environments, which are computer-generated environments that mimic real-world scenarios. These environments can be created using a variety of simulation software, such as OpenAI Gym, Unity, and MuJoCo.
- **Video games**: Reinforcement learning has been used to train agents to play a variety of video games, including Atari games, first-person shooter games, and board games like Go and chess. This can be a fun and engaging way to experiment with reinforcement learning algorithms.
- **Robotics**: Reinforcement learning has been applied to the field of robotics, allowing robots to learn to perform a variety of tasks, such as manipulation and navigation. This can be a challenging and rewarding way to experiment with reinforcement learning, and it has the potential to lead to the development of intelligent robots with a wide range of capabilities.

INITIATION

With the technical details documented. The next step is defining how the system will be constructed. Most importantly defining methods for teaching. As the environment are very customizable it would take a large amount of time to define constraints. And for the duration of this project, it is not feasible.

To make the teaching environment more interactive and engaging, additional features can be incorporated, such as visualizations, explanations, and quizzes.

Some possible steps to teach reinforcement learning include:

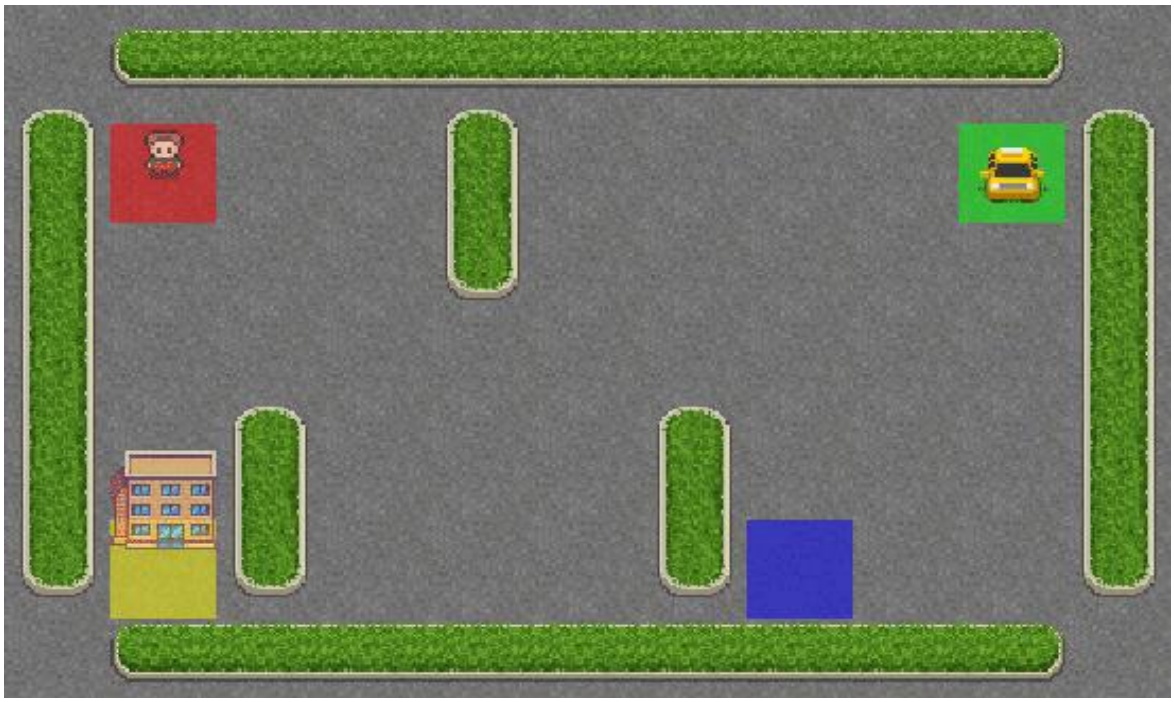
- Understand the fundamentals of reinforcement learning, including the components of a reinforcement learning system and the concepts of rewards, actions, and states.
- Choose a reinforcement learning algorithm, such as Q-learning or policy gradient methods, and implement it in a suitable programming language.
- Design a learning environment for your model to operate in. This could be a virtual environment, a simulation, or a real-world system.
- Create a set of rules and rewards for the model to learn from. This will involve defining the goals of the model and the rewards it will receive for achieving those goals.
- Train the model by providing it with input data and allowing it to take actions and receive rewards based on its decisions.
- Monitor the model's performance and adjust the training procedure as needed to improve its learning.

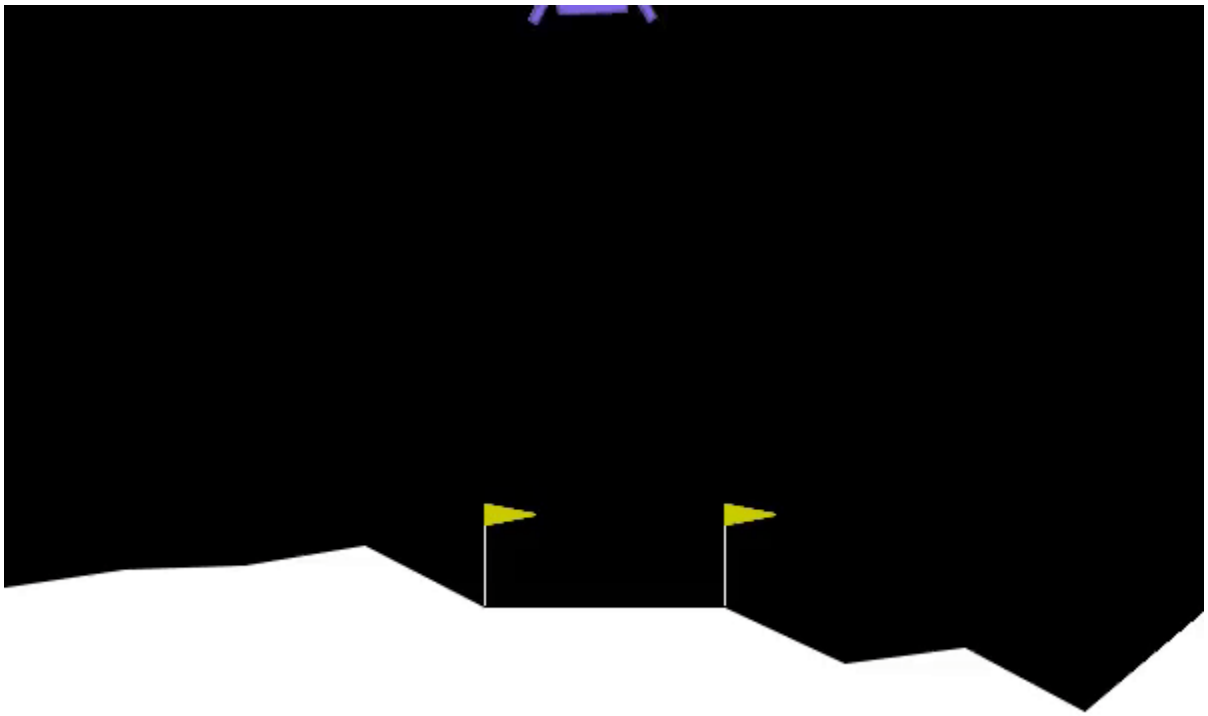
PRELIMINARY RESULTS

A variety of model has been trained in the OpenAI Gym environment. the OpenAI Gym is a toolkit for developing and comparing reinforcement learning algorithms. It provides a variety of environments, such as simulated games and other challenges, that can be used to test and evaluate different reinforcement learning algorithms and techniques. The environments in the OpenAI Gym have different observation and action spaces, which allows for a wide range of scenarios and challenges to be tested.

By using the OpenAI Gym, we can quickly and easily evaluate the algorithms and methods on a variety of different environments, without having to spend time and effort on building and maintaining their own testing environments. This makes the OpenAI Gym a very efficient and effective testing ground for reinforcement learning algorithms and methods. Some of the models trained for testing are shown below.

The difficulty of this project is the training time that is required and interfacing of methods. The models themselves require millions of timesteps of training to perform somewhat accurately. Tweaking these settings and understanding how things work becomes a large task. Only a small number of algorithms has been applied, because this is the most technically intensive subject. On the other hand, a larger number of environments has been trained for the environment to feel interesting. Algorithms might not look that different, but environments surely do!





INTERFACING

Initially a Wireframe app was planned. But based on the scale of the project and the required testing needed, choice was made for a CLI app. The app has two main endpoints that are of focus. One for Training, the other for playing or 'enjoying' an environment.

Currently only a single algorithm, a2c is implemented. But there are many environments available to play.

```
usage: train.py [-h]
                [--algo {a2c,ddpg,dqn,ppo,sac,td3,ars,qrdqn,tqc,trpo,ppo_lstm}]
                [--env ENV] [-tb TENSORBOARD_LOG] [-i TRAINED_AGENT]
                [--truncate-last-trajectory TRUNCATE_LAST_TRAJECTORY]
                [-n N_TIMESTEPS] [--num-threads NUM_THREADS]
                [--log-interval LOG_INTERVAL] [--eval-freq EVAL_FREQ]
                [--optimization-log-path OPTIMIZATION_LOG_PATH]
                [--eval-episodes EVAL_EPISODES] [--n-eval-envs N_EVAL_ENVS]
                [--save-freq SAVE_FREQ] [--save-replay-buffer] [-f LOG_FOLDER]
                [--seed SEED] [--vec-env {dummy,subproc}] [--device DEVICE]
                [--n-trials N_TRIALS] [--max-total-trials MAX_TOTAL_TRIALS]
                [-optimize] [--no-optim-plots] [--n-jobs N_JOBS]
                [--sampler {random,tpe,skopt}] [--pruner {halving,median,none}]
                [--n-startup-trials N_STARTUP_TRIALS]
                [--n-evaluations N_EVALUATIONS] [--storage STORAGE]
                [--study-name STUDY_NAME] [--verbose VERBOSE]
                [--gym-packages GYM_PACKAGES [GYM_PACKAGES ...]]
                [--env-kwargs ENV_KWARGS [ENV_KWARGS ...]]
                [-params HYPERPARAMS [HYPERPARAMS ...]] [-conf CONF_FILE]
                [-yaml YAML_FILE] [-uuid] [--track]
                [--wandb-project-name WANDB_PROJECT_NAME]
                [--wandb-entity WANDB_ENTITY] [-P]
                [-tags WANDB_TAGS [WANDB_TAGS ...]]
```

```
usage: enjoy.py [-h] [--env ENV] [-f FOLDER]
                [--algo {a2c,ddpg,dqn,ppo,sac,td3,ars,qrdqn,tqc,trpo,ppo_lstm}]
                [-n N_TIMESTEPS] [--num-threads NUM_THREADS] [--n-envs N_ENVS]
                [--exp-id EXP_ID] [--verbose VERBOSE] [--no-render]
                [--deterministic] [--device DEVICE] [--load-best]
                [--load-checkpoint LOAD_CHECKPOINT] [--load-last-checkpoint]
                [--stochastic] [--norm-reward] [--seed SEED]
                [--reward-log REWARD_LOG]
                [--gym-packages GYM_PACKAGES [GYM_PACKAGES ...]]
                [--env-kwargs ENV_KWARGS [ENV_KWARGS ...]] [--custom-objects]
                [-P]
```

CONCLUSION

The final demo is not ‘teachable’ ready. This is because during the project there were many ups and downs with dependency issues and technical abstractions. But this is also a good thing. Many things were learned along the way, and the technical basis of the project has been established. All that is left to complete is the UI.

In hindsight this project was very difficult to complete by myself. It would have been better to implement an existing technique inside a small application instead of attempting to create something completely new by myself.