Volume XXIV 2021
ISSUE no.2
MBNA Publishing House Constanta 2021

Scientific Bulletin of Naval Academy

SBNA PAPER • OPEN ACCESS

Artificial intelligence driven speed controller for DC motor in series

To cite this article: C. Bucur, P. Burlacu, Scientific Bulletin of Naval Academy, Vol. XXIV 2021, pg.83-88.

Submitted: 05.01.2022
Revised: 24.01.2022
Accepted: 14.02.2022

Available online at www.anmb.ro

ISSN: 2392-8956; ISSN-L: 1454-864X

doi: 10.21279/1454-864X-21-I2-007

Artificial intelligence driven speed controller for DC motor in series

C Bucur[1], P Burlacu[2]

[1]Lecturer PhD eng., "Ovidius" University, Constanța, RO
[2]Associate Professor PhD eng., Naval Academy "Mircea cel Bătrân", Constanța, RO

E-mail: burlacupaul@gmail.com. bcmircea@univ-ovidius.ro

Abstract. Recently a lot of work have been done to implement artificial intelligence controllers in the field of electrical motors. This paper presents a novel speed controller, developed through Reinforcement learning techniques, applied to series dc motors. We emphasize the ease of developed controller in available off the shelf hardware for industrial use. We used the open-source Python package gym-electric-motor [1] for environment setup, pytorch framework for developing the controller and .NET for performance evaluation.

1. Introduction

Electric motor control is an important topic in research and industry and lately the usage of artificial intelligence for development of new controllers, beside the classical PI and MPC (model predictive control), has grown. Lately, due to breakthrough developments in this field, the usage of machine learning (ML) and deep neural networks (DNN) has gained a lot of traction. Applying these techniques in the field of motor control is new and still in the development stages.

Different strategies and frameworks have been developed aimed directly to this field and we focused on an OpenAI Gym environment gym-electric-motor (GEM) for setting up the environment, training and validation, and reinforcement learning (RL) as the algorithm for developing such a controller.

Basically, the controller is and agent interacting with the environment and upon any action it's receiving a reward based on control objective. The main definitions used in reinforcement learning are:

a) **Environment** The world that an agent interacts with and learns from.
b) **Action** a: How the Agent responds to the Environment. The set of all possible Actions is called *action-space*.
c) **State** s: The current characteristic of the Environment. The set of all possible States the Environment can be in is called *state-space*.
d) **Reward** r: Reward is the key feedback from Environment to Agent. It is what drives the Agent to learn and to change its future action. An aggregation of rewards over multiple time steps is called **Return**.
e) **Optimal Action-**Value function $Q^*(s,a)$: Gives the expected return if you start in state s, take an arbitrary action a, and then for each future time step take the action that maximizes returns. *Q* can be said to stand for the "quality" of the action in a state. We try to approximate this function.

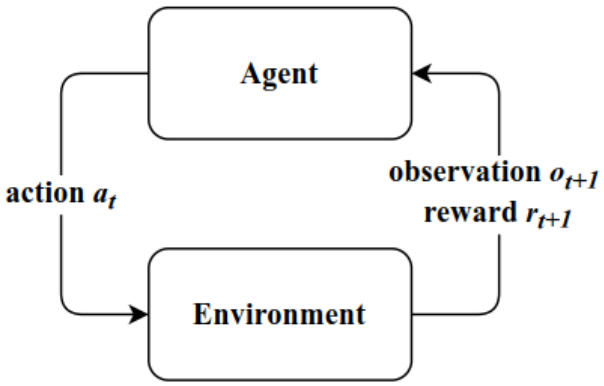

Figure 1. Basic reinforcement learning setting.

Until present application of reinforcement learning applied to electrical motors has been done mainly on Tensorflow with the same GEM environment [4] but with different algorithms like deep-deterministic-policy-gradient DDPG but on continuous action space. Our approach is to evaluate a controller on a different algorithm (DQN) on discrete action space (which should be more difficult in the chosen environment, series connection dc motors).

2. Environment setup

For developing the new controller, we choose a setup with a series DC motor supplied by a two-quadrant converter.

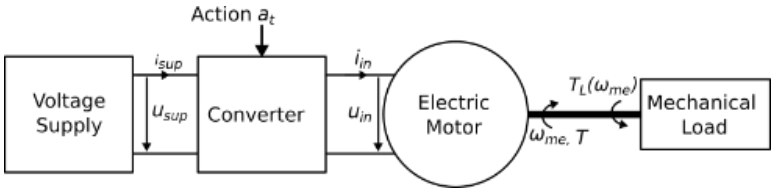

Figure 2. Supply converter motor load system.

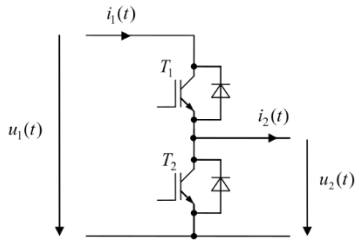

Figure 3. Two quadrant converter

Table 1. The parameters for dc motor are:

R_am	Ohm	2.93	Armature circuit resistance
R_em	Ohm	0.98	Exciting circuit resistance
L_am	H	6.1e-3	Armature circuit inductance
L_em	H	1.7e-3	Exciting circuit inductance
L_e_prime	H	0.04	Effective excitation inductance
J_rotor	kg/m^2	0.016	Moment of inertia of the rotor

In an OpenAI Gym environment the action- and observation space define the set of possible values for the actions and observations. In terms of electric motor control, the action space could be modelled in a discrete or continuous way [1]. In the discrete case, the actions are the direct switching commands for the transistors. Potential controllers are a hysteresis on-off controller or a DQN-agent.

The action space of the converter is Discrete(3): 0- Both Transistors off, 1- Upper Transistor on, 2- Lower Transistor on.

The generation of reference trajectories (e.g. the control set points) is a fundamental part of the environment and necessary for diverse training. The references should cover all use cases such that the RL-agent generalizes well and to avoid biased training data.

The objective is speed control and the refence used for learning process is a generator that generates a reference for speed state by a Wiener Process with the changing parameter sigma and mean = 0.

The Wiener process is a stochastic process W(t) for t>=0 with W(0)=0 such that the increment W(t)−W(s) is Gaussian with mean 0 and variance σ for any 0<=s<t, while increments for successive time steps are statistically independent.

The observations of the environments are a concatenation of the environment state, and the reference values the controller should track. All values are normalized by the limits of the state variables to a range of [−1, +1] or [0, +1] in case negative values are implausible for a state.

In case of series excitation dc motor the environment state is:

$$state = [\omega, T, i, u, u_{sup}] \tag{1}$$

3. Learning process

We implemented in pytorch a DQN (double Q learning algorithm) [2] for training the controller. The DQN algorithm is a known reinforcement learning technique that can be applied on a variety of problems but was mainly used on games. Our approach is to evaluate if the same algorithm can be applied on engineering related problems.

The core concept of DQN is the *replay memory* in which we store past observed measurements. We interact with the environment on episode bases and after the episode is finished, we are sampling the replay memory and try to minimize the loss function.

The error is the standard temporal difference error:

$$\delta = Q(s,a) - (r + \gamma \max Q(s',a)) \tag{2}$$

Where *s* is the current state, *a* is the action, *r* is the reward, γ is the discount factor and a' is the next state.The loss is calculated over a batch of transitions sampled for replay memory.

The network is implemented in a class named *Brain:*

```
class Brain(nn.Module):
  def __init__(self,dim_in,nr_actiuni):
    super(Brain,self).__init__()
    self.dim_in=dim_in
    self.nr_actiuni=nb_actiuni
    self.fc1=nn.Linear(input_size,128)
    self.fc2=nn.Linear(128,128)
    self.fc3=nn.Linear(128,128)
    self.fc4=nn.Linear(128,nb_actions)
  def forward(self,state):
    x=torch.relu(self.fc1(state))
    x=torch.relu(self.fc2(x))
    x=torch.relu(self.fc3(x))
    q_values=self.fc4(x)
    return q_values
```

The neural network consists of four layers with 128 neurons. The activation function for each neuron is Relu.

As per the DQN algorithm we have 2 networks, and the class is described below:

class NeuralNet(nn.Module):

```
def __init__(self, dim_in, iesire_dim):
    super(NeuralNet,self).__init__()
    self. dim_in = dim_in
    self.nb_actiuni= iesire_dim
    self.online = Brain(dim_in, iesire_dim)
    self.target = copy.deepcopy(self.online)
    for p in self.target.parameters():
        p.requires_grad = False

def forward(self, input, model):
    if model == "online":
        return self.online(input)
    elif model == "target":
        return self.target(input)
```

The model inputs are [speed, reference] and the outputs are the Q values for actions 0, 1 and 2. For choosing the control output we used ε *greedy* with a minimum exploration rate 0.15.

For optimization purposes we used Adam algorithm with a learning rate 0.001 and the loss function SmoothL1Loss. Gamma used is 0.99 and 3 the number of experiences between updates of Q_online. We train the model in 8e6 iterations with a reward function that gives 1 if the error between setpoint and actual speed value is decreasing and -1 in rest [3].
Below are presented snapshots in the process of learning:

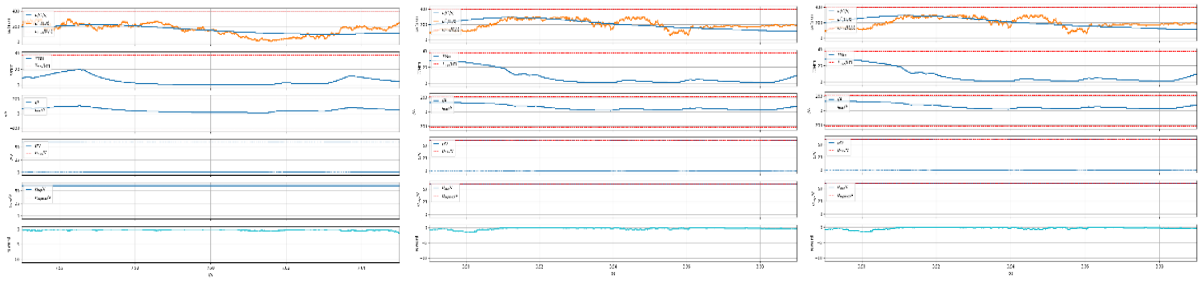

Figure 4. Process of learning.

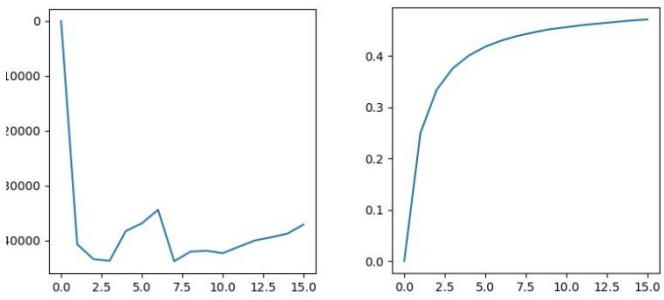

Fig.5 Reward and Loss plots

4. Results

For evaluation of controller performance we've created a dynamic link library throught Simulink embbeded coder of a model consisting of series dc motor feed by a 2 quadrant converter. The motor parameters are not the same as the parameters used for neural network training.

The controller was exported in ONNX format and we integrated both modules in a .net application through ML.NET framework. We followed this path since Matlab does not have a

working onnx model importer. Also, the onnx runtime provided by ML.NET is suitable for running on linux based embedded targets.

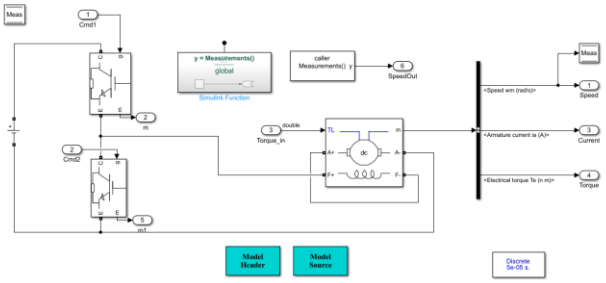

Fig. 6 Simulink model

Fig. 7 Onnx model view

The WPF application windows is presented below:

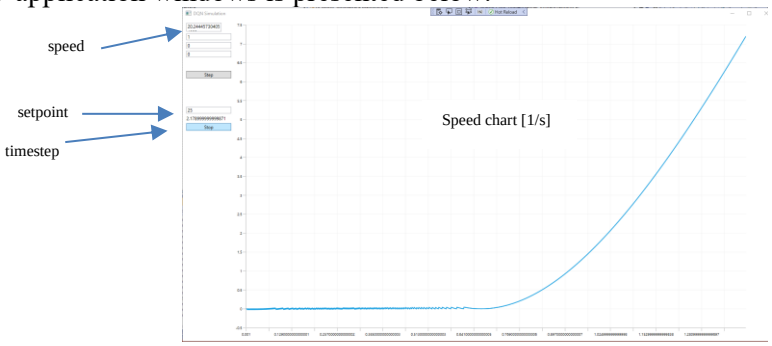

Fig.8 Application for controller evaluation

The step response for the system is presented below:

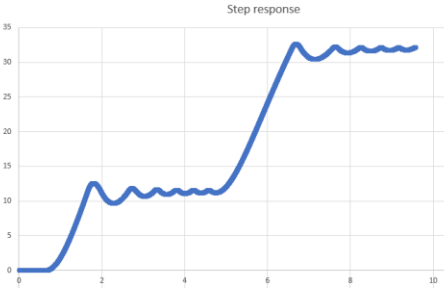

Fig.8 Series DC motor step response

As seen in the plot above, the step response is very good considering the nonlinear characteristics of series excitation dc motor and the discrete control is impacting the excitation field also.

On the same time the toolchain used for evaluation is novel and provide a foundation for further development. It's a known fact that Pythorch lacks a good support for production use, fact that makes TensorFlow the first choice for production environments.

The toolchain that we tested leverages the ML.NET, ONNX runtime in conjunction with Matlab for offline evaluation or .net on embedded devices for production (future implementation). We do not have knowledge of existing applications that use this approach.

5. Conclusions

In this paper we presented a novel DQN discrete speed controller for series DC motors fed by a two-quadrant converter, trained with pytorch framework.

The results are encouraging, and the next step is hardware implementation through ONNX format and real time hardware. The real time hardware can be any Linux or Windows 10 IOT that has support for .net 3.0 or above.

References

[1] A. Traue, G. Book, W. Kirchgässner, O. Wallscheid - Toward a Reinforcement Learning Environment Toolbox for Intelligent Electric Motor Control, *IEEE Transactions on Neural Networks and Learning Systems*, 2020;

[2] Hado van Hasseltl, Arthur Guez and David Silver - *Deep Reinforcement Learning with Double Q-learning*, arXiv:1509.06461v3, 2015;

[3] Diederik P. Kingma, Jimmy Lei Ba - ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, Published as a conference paper at ICLR 2015

[4] *gym-electric-motor/examples at master · upb-lea/gym-electric-motor · GitHub*