ISY – PROJECT LEARNING TO ACT – REINFORCEMENT LEARNING WITH DNN IN SIMULATED ENVIRONMENTS

PRESENTATION ON

CURIOSITY—DRIVEN EXPLORATION BY SELF-SUPERVISED PREDICTION

Authors:
Deepak Pathak
Pulkit Agrawal
Alexei A. Efros
Trevor Darrell

By:
Sunkara, Mohith Bhargav

UNDERSTANDING ABSTRACT

- What are author’s saying?

In many real-world scenarios, rewards extrinsic to agent are extremely sparse, or absent all together. In such cases, curiosity can serve as an intrinsic reward Signal to enable the agent to explore its environment and learn skills that might Be useful later in life.

- The proposed approach is evaluated in two environments:

1. Super Mario Bros
2. VIZDoom

- Where three broad settings are investigated:

1. Sparse extrinsic reward, where curiosity allows for far fewer interactions with The environment to reach the goal.
2. exploration with no extrinsic reward, where curiosity pushes the agent to explore More efficiently
3. generalization to unseen scenarios (e.g. new levels of the same game) where the knowledge gained from earlier experience helps the agent explore new places much faster than starting from scratch.

INTRODUCTION

- Reinforcement learning algorithms aim at learning policies for achieving target Tasks by maximizing rewards provided by the environment. In some scenarios, These rewards are supplied to the agent continuously. E.g. the running score in An Atari game, or distance between a robot arm and a object in a reaching task.

- But, in many real world scenarios rewards extrinsic to the agent are extremely sparse or missing altogether, and it is not possible to construct a shaped reward Function.

- As human agents, we are accustomed to operating with rewards that are so Sparse that we only experience them once or twice in a lifetime.
Example: let us consider a three year old enjoying a sunny Sunday afternoon on a Playground, most trappings of modern life – college, good job, a house, a family are So far into the future, and they don’t provide any useful reinforcement signal. Yet, The three year old has no trouble entertaining herself in that playground using what Psychologists call intrinsic motivation or curiosity.

- Motivation / curiosity have been used to explain the need to explore the environment And discover novel states. More generally, curiosity is a way of learning new skills which Might come handy in pursuing reward in the future.

- Authors are also gives a small explanation that, in reinforcement learning, Intrinsic motivation become critical whenever extrinsic rewards are sparse.

- Most formulations of intrinsic reward can be grouped into two broad classes:

1. Encourage the agent to explore “novel” states
2. Encourage the agent to perform actions that reduce the error/uncertainty in the Agent’s ability to predict the consequence of its own actions (i.e. its knowledge about the environment).

“ One proposed solution to all these problems is to only reward the agent when it encounters states that are hard to predict but are “learnable”. However, estimating learnability is a non-trivial problem ”

WORKING

- This work belongs to the board category of methods that generate an intrinsic Signal based on how hard it is for the agent to predict the consequences of its own Actions i.e. predict the next state given the current state and the executed action.
- However, author's managed to escape most of pitfalls of previous prediction Approaches with the following key insight:

1. They only predict those changes in the environment that could possibly be due to the actions of our agent or affect the agent, and ignore the rest.
2. Which means, instead of making predictions in the raw sensory space (e.g. pixels), we transform the sensory input into a feature space where only the information relevant to the action performed by the agent is represented.
3. They made learn this feature space using self-supervision – training a neural network on a proxy inverse dynamics task of predicting the agent's action given its current and next states.
4. Since the neural network is only required to predict the action, it has no incentive to Represent within its feature embedding space the factors of variation in the environment that do not affect the agent itself.
5. At the end they used this feature space to train a forward dynamics model that Predicts the feature representation of the next state, given the current feature Representation of current state and the action. Then they provided the prediction error Of the forward dynamics model to the agent as an intrinsic reward to encourage its curiosity

CURIOSITY – DRIVEN EXPLORATION

- The agent is composed of two subsystems:

1. A reward generator that outputs a curiosity – driven intrinsic reward signal
2. A policy that outputs a sequence of actions to maximize that reward signal

Note: in addition to intrinsic rewards, the agent optionally may also receive some Extrinsic rewards from environments.

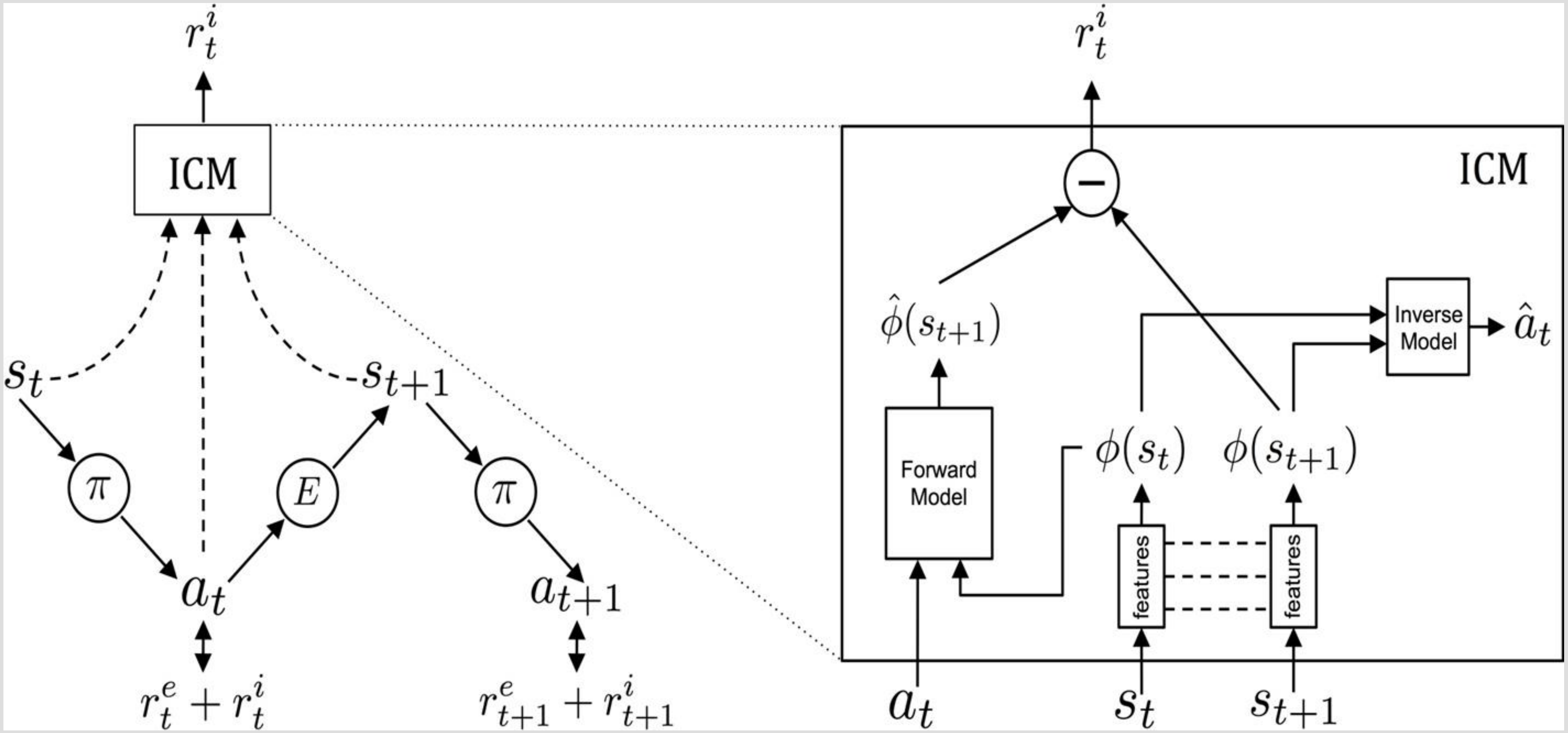
Let the intrinsic curiosity reward generated by the agent at time t be r(i)t and the extrinsic reward be r(e)t . The policy sub-system is trained to maximize the sum of these two rewards

r(t) = r(i)t + r(e)t , with mostly or always r(e)t == 0

The policy pi(s(t);theta(p)) is represented by a deep neural network with parameters Theta(p). Given the agent state s(t) it executes the action a(t) ~ pi(s(t);theta(p)) sampled From the policy. Theta(p) is optimized to maximize the expected sum of rewards,

$$\max_{\theta_P} \mathbb{E}_{\pi(s_t;\theta_P)}[\Sigma_t r_t] \qquad \rightarrow \text{eq(1)}$$

- The proposed curiosity reward model can potentially be used with a range of policy Learning methods; but in this experiment author's used A3C for policy learning.
- Because the main contribution is to design an intrinsic reward signal based on Prediction error of the agent's knowledge about its environment that scales to high— Dimensional continuous state spaces like images, bypasses the hard problem of predicting pixels and is unaffected by the unpredictable aspects of the environment that do not affect the agent.



Representations:

s(t) → present state

Pi → policy

a(t) → action

r(i)t → intrinsic rewards

r(e)t → extrinsic rewards

E → environment

s(t+1) → next state

a(t+1) → action of next state

ICM → intrinsic curiosity module

PREDICTION ERROR AS CURIOSITY REWARD

- Making predictions in raw sensory space (images) is undesirable not only because It is hard to predict pixels directly, but also because it is unclear if predicting pixels is Even the right objective to optimize.

Example : scenario where the agent is observing the movement of tree leaves in a Breeze.

So, now one may have question?
If not the raw observation space, then what is the right feature space for making Predictions?

- The author's say that at first let's divide all sources that can modify the agent's Observations into three cases:

1. Things that can be controlled y the agent
2. Things that the agent cannot control but that can affect the agent
3. Things out of the agent's control and not affecting the agent.

- They say that a good feature space for curiosity should model (1) and (2) and be Unaffected by (3).

SELF-SUPERVISED PREDICTION FOR EXPLORATION

- Instead of hand designing a feature representation for every environment, our aim is to come up with a general mechanism for learning feature representations such that the prediction error in the learned feature space provides a good intrinsic reward signal.

- They propose that such learning can be achieved by training a deep neural Network (Inverse Model) with two sub systems:

$$\hat{a}_t = g\left(s_t, s_{t+1}; \theta_I\right)$$ →eq(2)

$$\min_{\theta_I} L_I(\hat{a}_t, a_t)$$ →eq(3)

- In addition to inverse dynamics model, they also train another neural network Which takes :

$$\hat{\phi}(s_{t+1}) = f\left(\phi(s_t), a_t; \theta_F\right)$$ →eq(4)

$$L_F\left(\phi(s_t), \hat{\phi}(s_{t+1})\right) = \frac{1}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$$ →eq(5)

$$r_t^i = \frac{\eta}{2}\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2$$ →eq(6)

- The overall optimization problem that is solved for learning the agent is a Composition of equations 1, 3 and 5 and can be written as,

$$\min_{\theta_P,\theta_I,\theta_F}\left[-\lambda\mathbb{E}_{\pi(s_t;\theta_P)}[\Sigma_t r_t] + (1-\beta)L_I + \beta L_F\right]$$ →eq(7)

- Where :

→ $0 \leq \beta \leq 1$ is a scalar that weighs the inverse model loss against the forward model loss

→ $\lambda > 0$ is a scalar that weighs the importance of the policy gradient loss against the importance of learning the intrinsic reward signal.

EXPERIMENTAL SETUP

- Environments used

1. Super Mario Bros
2. VizDoom

- In the Super Mario Bros setup, they reparametrize the action space of the agent Into 14 unique actions. This game is played using a joystick allowing for multiple Simultaneous button presses, where the duration of the press affects what action Is being taken.
- This property makes the game particularly hard, e.g. to make a long jump over Tall pipes or wide gaps, the agent needs to predict the same action up to 12 times In a row, introducing long range dependencies.

- Training details
- All agents or agent in this work are trained using visual inputs that are pre-Processed.
- The input RGB images are converted into grey-scale and re-sized to 42 * 42.
- In order to model temporal dependencies, the state representation (s(t)) of the Environment is constructed by concatenating the current frame with three previous Frames.
- An action repeat of six is used in Mario.
- However, they sampled the policy without any action repeat during inference of the Finial sample model.

ARCHITECTURE

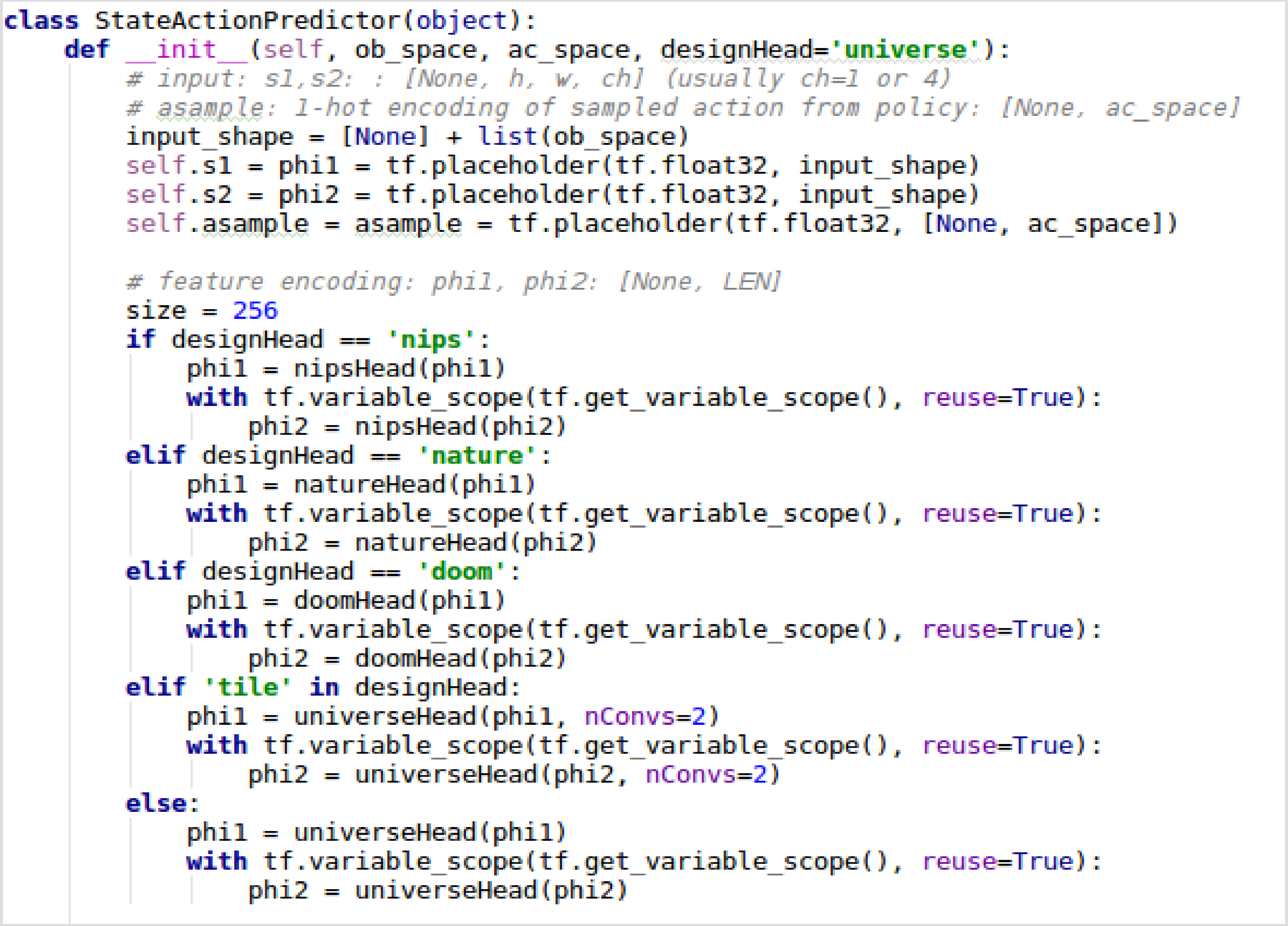
- A3C architecture

1. The input state (s(t)) is passed through a sequence of four convolution layers
With 32 filters each, kernel size of 3*3, stride of 2 and padding of 1.
2. An exponential linear unit (ELU) is used after each convolution layer.
3. The output of last convolution layer is fed into a LSTM with 256 units.
4. Two separate fully connected layers are used to predict the value function and
The action from the LSTM feature representation.

- Intrinsic Curiosity Module (ICM) architecture

1. As mentioned previously it consists of the forward model and the inverse model.
2. The IM first maps (s(t)) into a feature vector phi(s(t)) using a series of four
Convolution layers, each with 32 filters, kernel size 3*3, stride of 2 and padding of 1.
3. ELU non-linearity is used after each convolution layer.
4. The output of last convolution layer has dimensionality of 288.
5. The phi(s(t)) and phi(s(t+1)) are combined into a single feature vector and passed
Into a fully connected layer of 256 units and followed by an output fully connected
Layer with 4 units to predict one of the four possible actions.
6. The FM is constructed by combining phi(s(t)) with a(t) and passing it into a
Sequence of two fully connected layers with 256 and 288 units.
7. The beta = 0.2 and delta = 0.1. and the finial eq is minimized with learning rate
1*exp(-3).

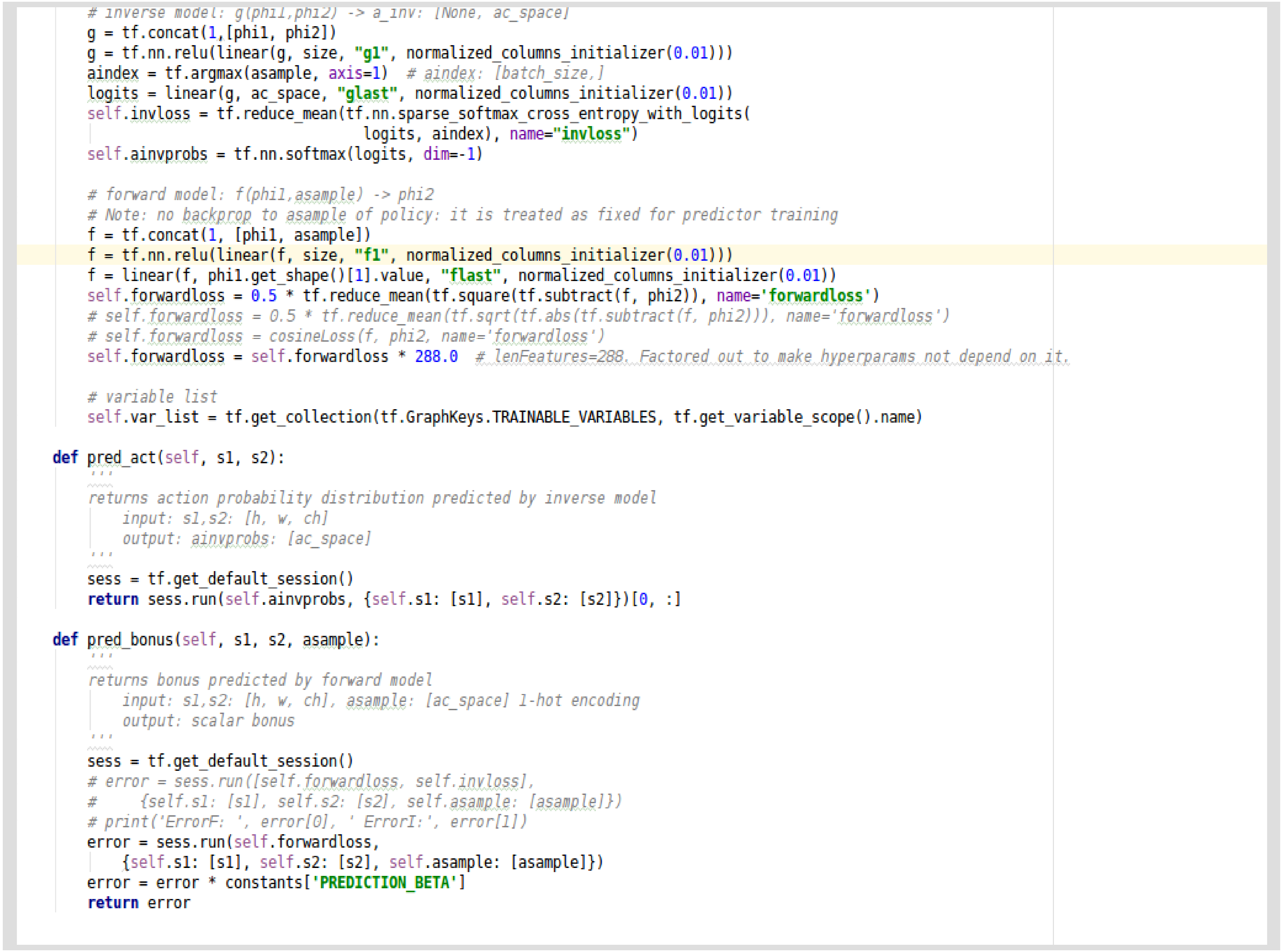
CODE SNIPPETS OF MAIN IDEA



```python
class StateActionPredictor(object):
    def __init__(self, ob_space, ac_space, designHead='universe'):
        # input: s1,s2: : [None, h, w, ch] (usually ch=1 or 4)
        # asample: 1-hot encoding of sampled action from policy: [None, ac_space]
        input_shape = [None] + list(ob_space)
        self.s1 = phi1 = tf.placeholder(tf.float32, input_shape)
        self.s2 = phi2 = tf.placeholder(tf.float32, input_shape)
        self.asample = asample = tf.placeholder(tf.float32, [None, ac_space])

        # feature encoding: phi1, phi2: [None, LEN]
        size = 256
        if designHead == 'nips':
            phi1 = nipsHead(phi1)
            with tf.variable_scope(tf.get_variable_scope(), reuse=True):
                phi2 = nipsHead(phi2)
        elif designHead == 'nature':
            phi1 = natureHead(phi1)
            with tf.variable_scope(tf.get_variable_scope(), reuse=True):
                phi2 = natureHead(phi2)
        elif designHead == 'doom':
            phi1 = doomHead(phi1)
            with tf.variable_scope(tf.get_variable_scope(), reuse=True):
                phi2 = doomHead(phi2)
        elif 'tile' in designHead:
            phi1 = universeHead(phi1, nConvs=2)
            with tf.variable_scope(tf.get_variable_scope(), reuse=True):
                phi2 = universeHead(phi2, nConvs=2)
        else:
            phi1 = universeHead(phi1)
            with tf.variable_scope(tf.get_variable_scope(), reuse=True):
                phi2 = universeHead(phi2)
```

Cont..



```
        # inverse model: g(phi1,phi2) -> a_inv: [None, ac_space]
        g = tf.concat(1,[phi1, phi2])
        g = tf.nn.relu(linear(g, size, "g1", normalized_columns_initializer(0.01)))
        aindex = tf.argmax(asample, axis=1)  # aindex: [batch_size,]
        logits = linear(g, ac_space, "glast", normalized_columns_initializer(0.01))
        self.invloss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_logits(
                                        logits, aindex), name="invloss")
        self.ainvprobs = tf.nn.softmax(logits, dim=-1)

        # forward model: f(phi1,asample) -> phi2
        # Note: no backprop to asample of policy: it is treated as fixed for predictor training
        f = tf.concat(1, [phi1, asample])
        f = tf.nn.relu(linear(f, size, "f1", normalized_columns_initializer(0.01)))
        f = linear(f, phi1.get_shape()[1].value, "flast", normalized_columns_initializer(0.01))
        self.forwardloss = 0.5 * tf.reduce_mean(tf.square(tf.subtract(f, phi2)), name='forwardloss')
        # self.forwardloss = 0.5 * tf.reduce_mean(tf.sqrt(tf.abs(tf.subtract(f, phi2))), name='forwardloss')
        # self.forwardloss = cosineLoss(f, phi2, name='forwardloss')
        self.forwardloss = self.forwardloss * 288.0  # lenFeatures=288. Factored out to make hyperparams not depend on it.

        # variable list
        self.var_list = tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES, tf.get_variable_scope().name)

    def pred_act(self, s1, s2):
        '''
        returns action probability distribution predicted by inverse model
            input: s1,s2: [h, w, ch]
            output: ainvprobs: [ac_space]
        '''
        sess = tf.get_default_session()
        return sess.run(self.ainvprobs, {self.s1: [s1], self.s2: [s2]})[0, :]

    def pred_bonus(self, s1, s2, asample):
        '''
        returns bonus predicted by forward model
            input: s1,s2: [h, w, ch], asample: [ac_space] 1-hot encoding
            output: scalar bonus
        '''
        sess = tf.get_default_session()
        # error = sess.run([self.forwardloss, self.invloss],
        #     {self.s1: [s1], self.s2: [s2], self.asample: [asample]})
        # print('ErrorF: ', error[0], ' ErrorI:', error[1])
        error = sess.run(self.forwardloss,
            {self.s1: [s1], self.s2: [s2], self.asample: [asample]})
        error = error * constants['PREDICTION_BETA']
        return error
```

EXPERIMENTS

- No reward setting
- A good exploration policy is one which allows the agent to visit as many states as possible even without any goals. In the case of 3-D navigation, we expect a good exploration policy to cover as much of the map as possible; in the case of playing a game, we expect it to visit as many game states as possible.

- Mario: Learning to play with no rewards
- Only reward given to agent is intrinsic curiosity rewards.
- All extrinsic rewards from environment are made false.
- The killing or dodging of enemies is automatically discovered with a possible Reason because getting killed by enemy will result in only seeing a small part of Game space, making it curiosity saturate.
- Therefore, in order to remain curious, it is in the agent's interest to learn how to Kill and dodge enemies so that it can reach new parts of the game space.
- This suggests that curiosity provides indirect supervision for learning interesting Behaviours.

- Generalization to novel scenarios
- In the above slide as I mentioned how the agent learns to explore large part
Of the space where its curiosity driven exploration policy was trained.
- However, it remains unclear whether the agent has done this by learning

1. "generalized skills" for efficiently exploring its environment.
2. If it simply memorized the training set.

- To understand the this learning is not memorized but general skills. Authors
has investigated this question by

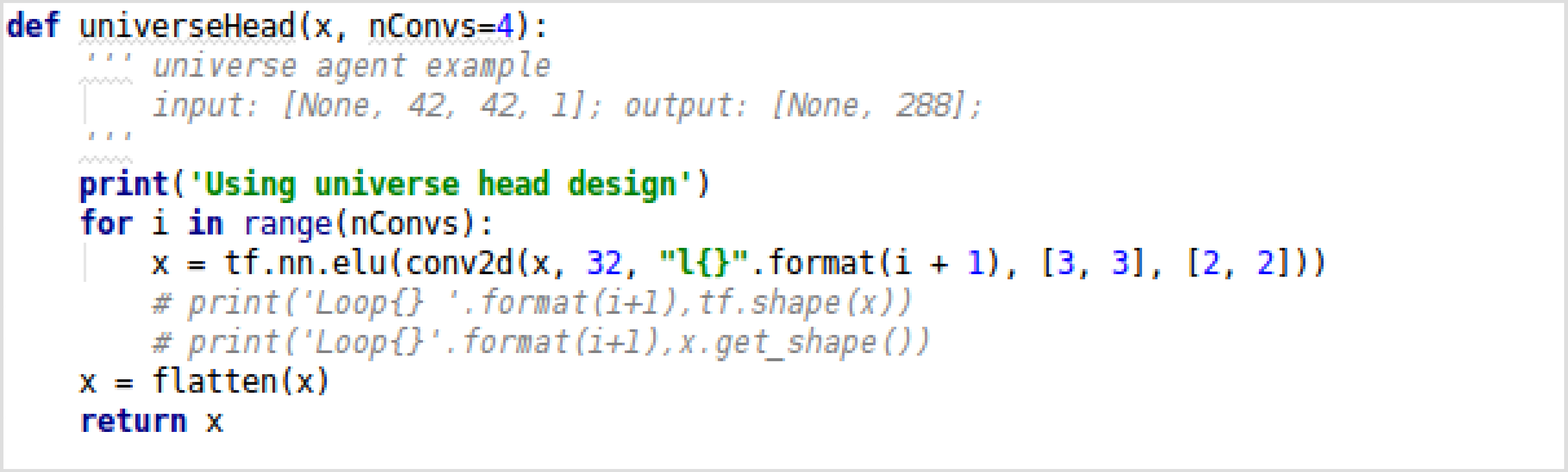
a) Apply the learned policy "as is" two new level – (used)
b) Adapt the policy by fine tuning with curiosity rewards only – (used)
c) Adapt the policy to maximize some extrinsic rewards – (proposed)

Video of trained agent
And tensor flow graphs

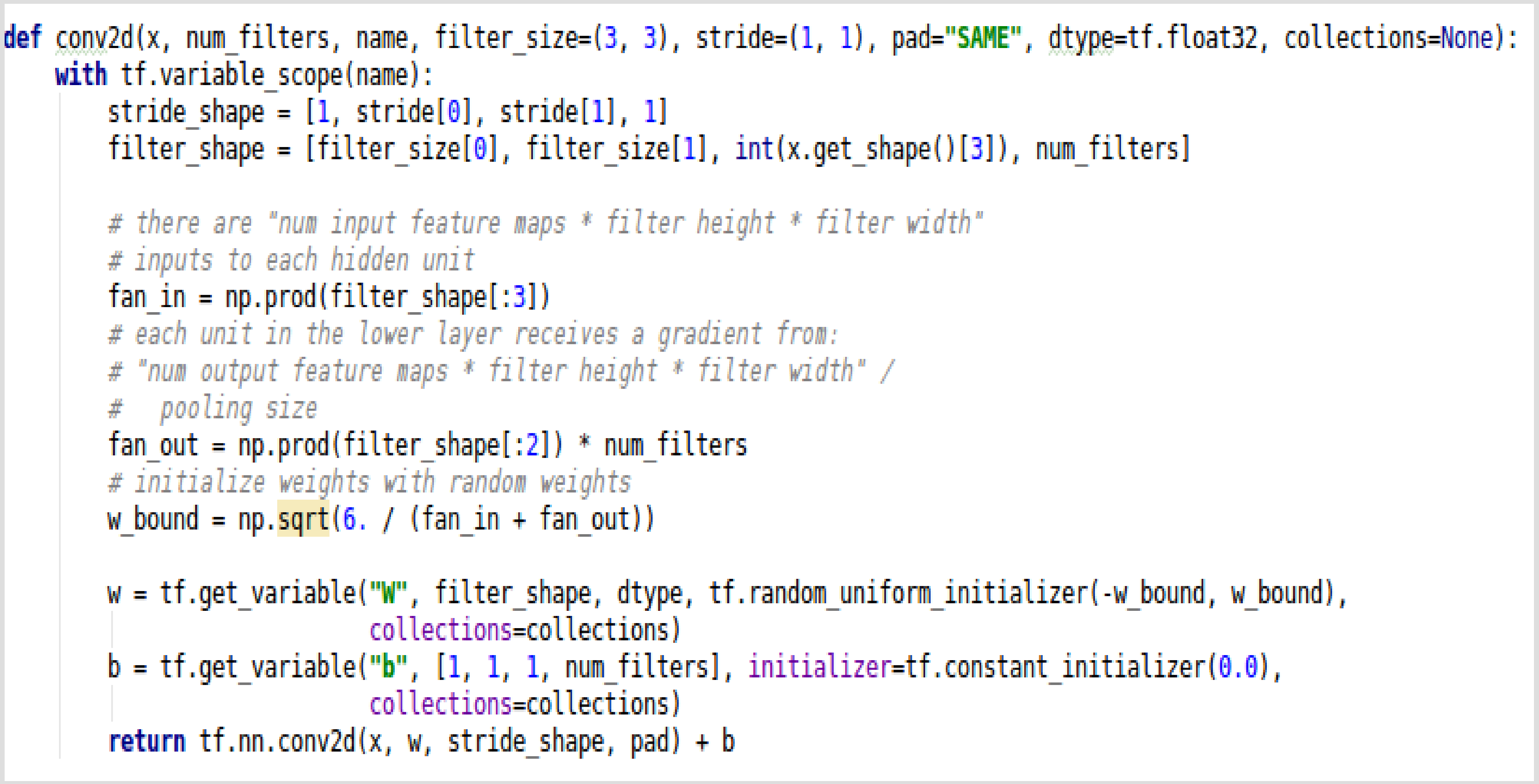
QUESTIONS FROM PRESENTATION

1. The agent translates a pixel-based state observation [42, 42, 4](?) into a feature representation [288](?). What is the exact learning procedure of this translation?

Answer: In the code implementation as we can see the code snippet below. The authors has mentioned that the agent used a “Universe head design” which is taken form “universe agent example” → where input to the agent is [42,42,1] and output is [288]. But in the implementation case as the agent take 4 frames → input is [42,42,4] and output is [288]. To achieve this they trained a convolutional neural network which can also be seen in next slide.



```python
def universeHead(x, nConvs=4):
    ''' universe agent example
        input: [None, 42, 42, 1]; output: [None, 288];
    '''
    print('Using universe head design')
    for i in range(nConvs):
        x = tf.nn.elu(conv2d(x, 32, "l{}".format(i + 1), [3, 3], [2, 2]))
        # print('Loop{} '.format(i+1),tf.shape(x))
        # print('Loop{}'.format(i+1),x.get_shape())
    x = flatten(x)
    return x
```



```
def conv2d(x, num_filters, name, filter_size=(3, 3), stride=(1, 1), pad="SAME", dtype=tf.float32, collections=None):
    with tf.variable_scope(name):
        stride_shape = [1, stride[0], stride[1], 1]
        filter_shape = [filter_size[0], filter_size[1], int(x.get_shape()[3]), num_filters]

        # there are "num input feature maps * filter height * filter width"
        # inputs to each hidden unit
        fan_in = np.prod(filter_shape[:3])
        # each unit in the lower layer receives a gradient from:
        # "num output feature maps * filter height * filter width" /
        #   pooling size
        fan_out = np.prod(filter_shape[:2]) * num_filters
        # initialize weights with random weights
        w_bound = np.sqrt(6. / (fan_in + fan_out))

        w = tf.get_variable("W", filter_shape, dtype, tf.random_uniform_initializer(-w_bound, w_bound),
                            collections=collections)
        b = tf.get_variable("b", [1, 1, 1, num_filters], initializer=tf.constant_initializer(0.0),
                            collections=collections)
        return tf.nn.conv2d(x, w, stride_shape, pad) + b
```

2. Is this translation being learned in parallel to learning the "state >> action" policy? Or is the "state >> features" translation being learned first and, after it is frozen, the agent starts to learn the "state >> action" policy? This is important to figure out, because the "state >> features" translation influences "r-intrinsic", which influences "state >> action" policy.
Answer: This translation is learned in parallel where the first the agent learn “state >> action” and then the ICM (Intrinsic curiosity model) takes this into and make “state >> features” which is then fed into inverse model and forward model to make their predictions and produce relevant outputs. I would like to support this above explanation using this lines written by authors in the paper → section 2.2 – Self – supervised prediction for exploration

> “the learned function (equation 2 above in the slide 9) g is also known as the inverse dynamics model and the tuple (s(t), a(t), s(t+1)) required to learn g is obtained while the agent interacts with the environment using its current policy pi(s)”

3. Tensor Board graphs: what represents the "ep_0/value" plot?

Answer: It is the value function of the episode zero which is obtained when the finial trained model is inference. As in this implementation inference.py does not use action repeat. So, we can say it as a function which represent how good trained model has made agent to attain good state using trained policy which gives an action.

THANK YOU