GOAL SOLVER

- Shareef Uddin Mohammed

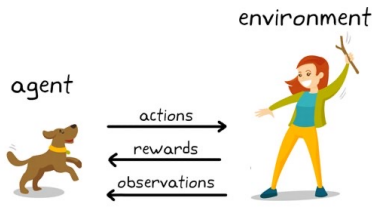
1

AGENDA

- What is Goal Solver(GS)?
- How we develop GS using Reinforcement Learning
- Action Masking in RL
- Tools and Framework used
- Lessons learned
- How you can take advantage of Goal Solver

2

GOAL SOLVER



- **Use-Case**: Goal Solver provides a decision-making framework for clients to achieve financial success on financial goals (Retirement, Education and Custom/Generic) by suggesting changes to retirement date, goal amounts , savings and spending towards their financial goals by maximizing their chance of success in reaching all the goals.
- **Ai/ML**: We have used sophisticated Ai/ML techniques called "Reinforcement Learning" to solve this optimization use case, which involves an agent taking actions and observes the reward while interacting with the environment.
- **How we did it**: Given a financial situation of the client during training of agent we call it exploration phase, it tries different actions (the actions could be increasing taxable savings, modifying retirement age, spending) and see which actions leads to success criteria and the actions which meets the criteria we give positive reward to the agent and if the actions doesn't meet the criteria, agent gets negative reward. Once we repeat this process over and over eventually the agent learns to take those sequence of actions which maximizes total reward in lifetime and gives client better chance of success.

3

WHAT IS GOAL SOLVER (GS)?

Goal Solver provides a **decision-making framework** for our clients by **optimizing** the chance of **Success** in meeting their financial goals.

Retirement goal "levers":

- Retirement Age
- Savings (contributions)
- Spending in retirement
- Custom goals amount

Custom Goal levers:

- Savings
- Custom Goal Target Amount

4

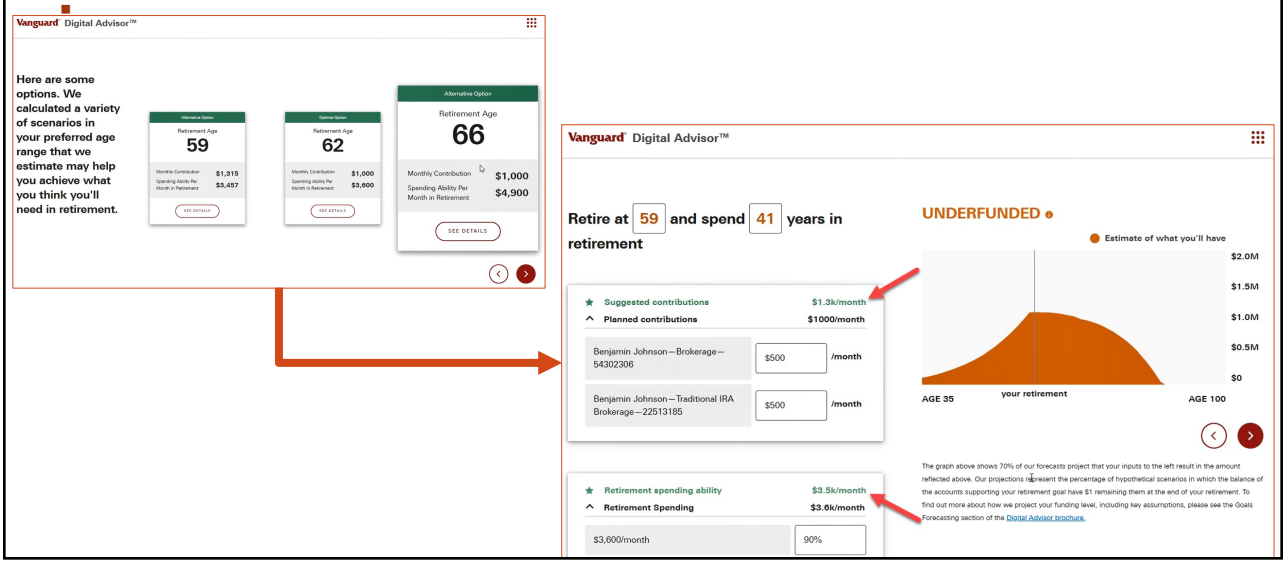
STRATEGIES

Since there are many levers to pull we came up with different strategies by keeping some of the levers constant and changing few at a time.

- Early Retirement (ER) Strategy: Change Spending and contribution by keeping retirement age constant at early retirement age.
- ER Flex: Change all the levers by taking few ages from early retirement range.
- LR Flex, O Flex
- R, S, C

5

DA – CONSUMER USAGE



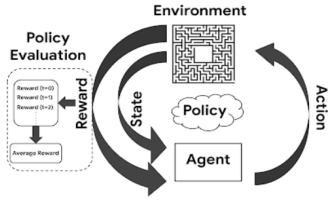
6

SUITE OF PRODUCTS UNDER GS

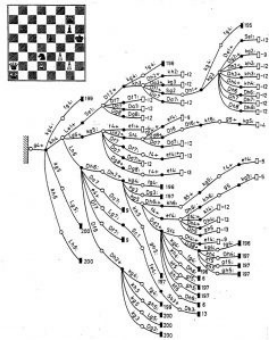
Financial Plan Setup

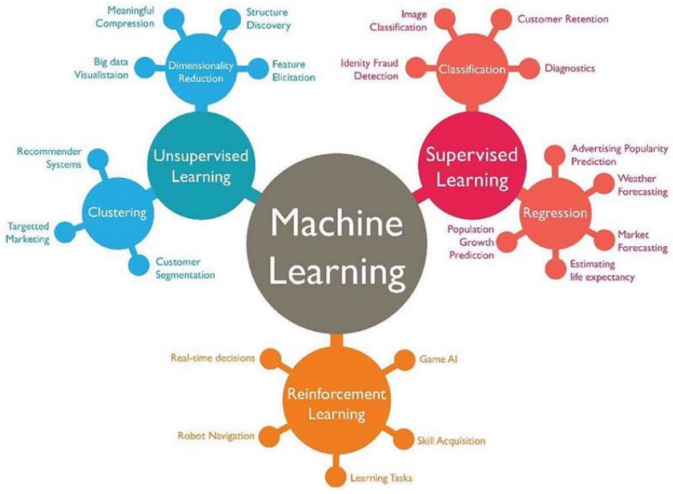
MG-NDA

7



Agent-environment interaction in reinforcement learning. At each step, an agent takes an action based on a policy, receives a reward and makes a transition to a new state.





8

TRAINING THE RL AGENT!

RL Training
The AI team have generated synthetic user profile data (many personas) and training a Reinforcement Learning (RL) agent with the intent of finding multiple viable client scenarios that result in success over a target success rate.

At the end of training, an RL agent must have seen enough data and interactions with FPW to recommend the actions a user should take to maximize the chance of success to meet all the goals in his lifetime.

The future RL training process expects to take into account both a predetermined set of training data as well as live traffic so that the model will get "smarter" over time. Methodology would either weigh in on, or help create, training data.

9

STATE, ACTION AND REWARD FUNCTION!

- State: It describes the current situation. The current tax deferred, taxable, tax-free, inherited-ira balances, the goals info., current savings, potential contribution amount.
- Action: It includes the percent of potential contribution a client can make from 0 to 100% in increments of 2.5% so in all 40 actions.
- Reward function: It defines the reward the agent gets by taking one of the 40 actions given a state. It could be +ve reward when the action reaches the target success rate or a –ve reward when the action fall short of the target!

```
self.state = (USstate_dict[USstate_taxes_residence], \
              taxable_bal, \
              tax_def_bal, \
              tax_free_bal, \
              inherited_ira_bal, \
              variable_annuities_bal, \
              after_tax_bal, \
              goal_count, \
              goal_target_amt[0], \
              goal_starts_in_yrs[0], \
              goal_target_amt[1], \
              goal_starts_in_yrs[1], \
              goal_target_amt[2], \
              goal_starts_in_yrs[2], \
              retirement_starts_in_yrs, \
              potential_contribution_amount, \
              taxable_savings, \
              trad_IRA_savings, \
              retirement_annual_spend_amt, \
              self.totalBalance)
```

10

REWARD FUNCTION

```
peaksr = target_success_rate + (tolerance / 2)
if success_rate <= peaksr:
    fac2 = np.log(4.5) / peaksr
    reward = 2 - np.exp((peaksr - success_rate) * fac2)
    # Max value = 2 - exp(0) * 1.504 / 73 ->
elif success_rate > peaksr:
    fac2 = np.log(2) / (100 - peaksr)
    reward = 2 - np.exp((success_rate - peaksr) * fac2)

reward = reward / self.nfactor
self.totalBalance = np.percentile(self.totalBalance, 30)
logging.info("Reward: {}, Success Rate: {}".format(reward, success_rate))
```

11

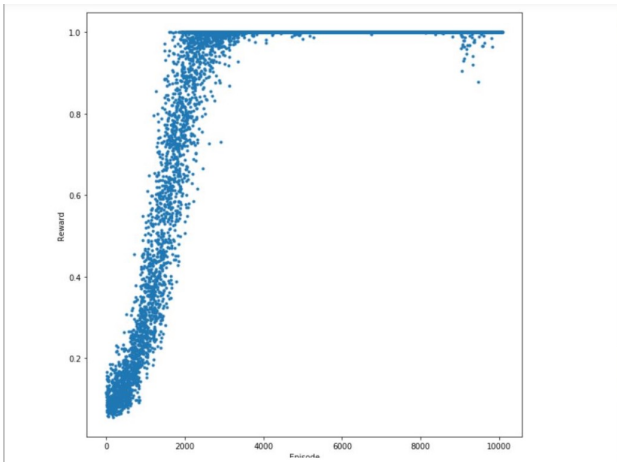
HPO

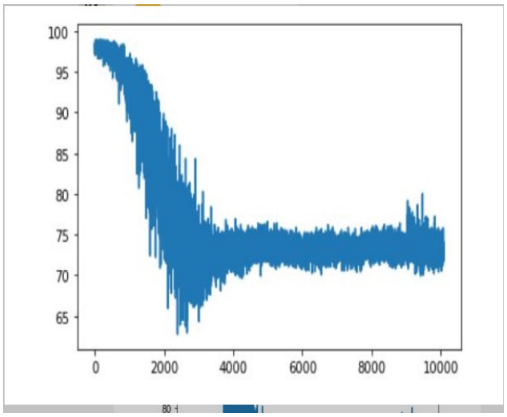
- Clip-param: 0.2
- KL-coeff: 0.01

```
ppo-cpu-train:
  run: PPO
  env: MGSEnvMaskER
  stop:
    episodes_total: 2000000
  checkpoint_freq: 10
  checkpoint_at_end: true
  restore: "/opt/ml/code/tmp/chkpts/checkpoint_230/checkpoint-230"
  config:
    gamma: 0.99
    lambda: 0.95
    lr: 0.0005
    lr_schedule: [[0, 0.00075], [1000,0.0005], [10000, 0.00025], [100000, 0.0001], [1000000, 0.00005]]
    num_sgd_iter: 30
    kl_coeff: 0.01
    kl_target: 0.003
    vf_loss_coeff: 0.5
    entropy_coeff: 0.01
    clip_param: 0.2
    vf_clip_param: 0.2
    log_sys_usage: True
    observation_filter: "NoFilter"
    vf_share_layers: True
    soft_horizon: True
    no_done_at_end: False
    normalize_actions: False
    clip_actions: True
    ignore_worker_failures: True
    use_pytorch: False
    sgd_minibatch_size: 128
    train_batch_size: 4096
    model:
      vf_share_layers: True
      custom_model: custom_attn_mgs_mask
      fcnet_hiddens: [512, 512, 512]
    rollout_fragment_length: 200
    batch_mode: complete_episodes
    num_workers: 136
    evaluation_num_workers: 4
    num_envs_per_worker: 1
    num_cpus_per_worker: 1
    num_cpus_per_worker: 1
    num_cpus_for_driver: 1
    shuffle_sequences: True
    explore: True
    eager: False
    log_level: "INFO"
    exploration_config:
      type: "StochasticSampling"
    evaluation_interval: 5000
    evaluation_num_episodes: 10
    evaluation_config:
      explore: False
```

12

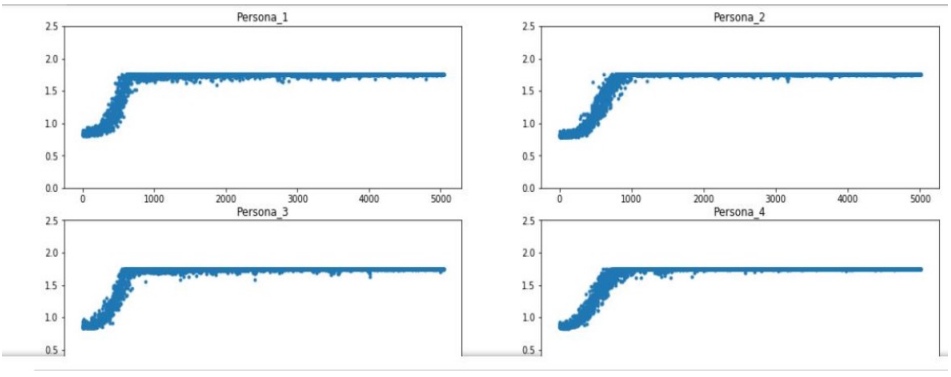
TESTING AND VALIDATION!





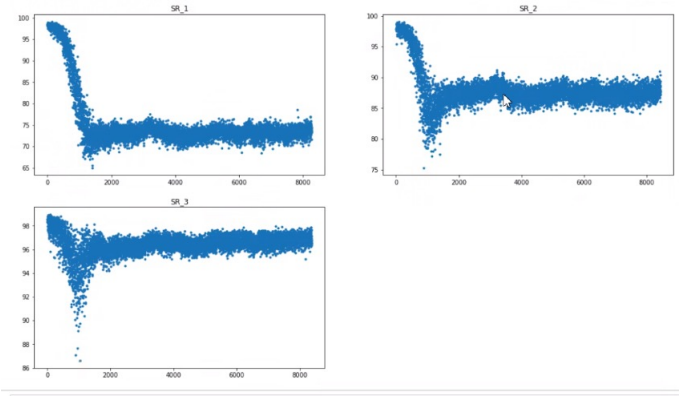
13

CONVERGENCE FOR DIFFERENT PERSONAS!



14

DIFFERENT SUCCESS RATE RANGES!



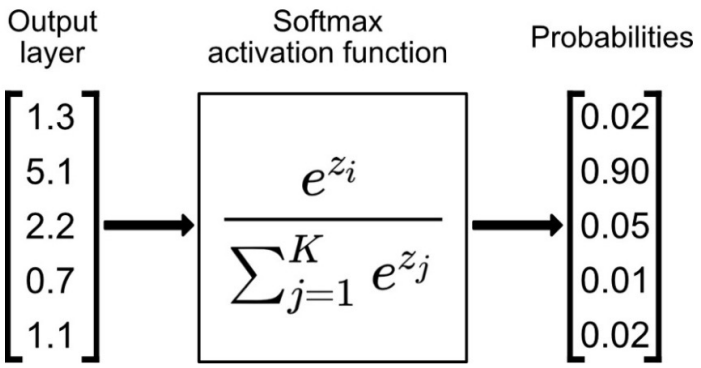
15

ACTION MASKING!

- RL algorithms learn by exploring the state space and taking random actions
- However sampling the full discrete action space is inefficient as many actions may be non-permissible .
- For e.g., taking a sequence of actions can violate some constraint
- In this case, avoiding such actions can accelerate the learning process

16

SOFTMAX RECAP!



17

ACTION MASKING!

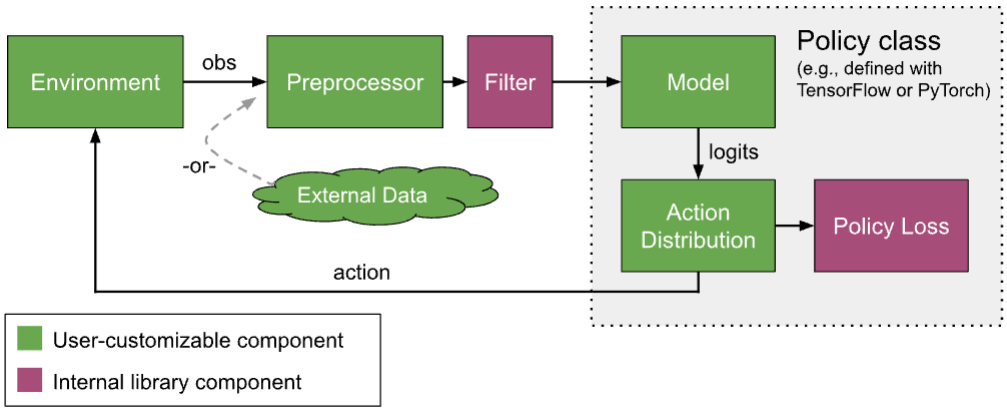
- Rllib, the reinforcement learning framework from Ray, is capable of implementing action masking for discrete action spaces
- Adding a large negative number to action logits before passing them to the softmax makes the probability of choosing these actions nearly 0

```
        # Expand the model output to [BATCH, 1, EMBED_SIZE]. Note that the
        # avail actions tensor is of shape [BATCH, MAX_ACTIONS, EMBED_SIZE].
        intent_vector = tf.expand_dims(action_embed, 1)

        # Batch dot product => shape of logits is [BATCH, MAX_ACTIONS].
        action_logits = tf.reduce_sum(avail_actions * intent_vector, axis=2)

        # Mask out invalid actions (use tf.float32.min for stability)
        inf_mask = tf.maximum(tf.math.log(action_mask), tf.float32.min)
        return action_logits + inf_mask, state

    def value_function(self):
        return self.action_embed_model.value_function()
```

- Reference: https://github.com/ray-project/ray/blob/master/rllib/examples/models/parametric_actions_model.py#L58

18

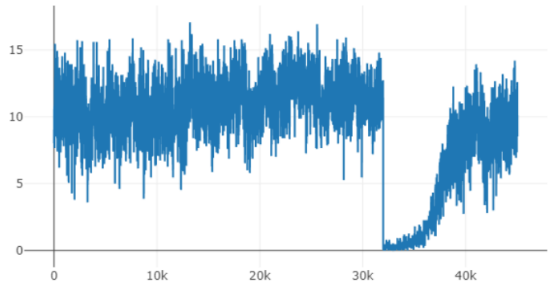
ACTION MASKING!



19

LESSON LEARNED!

- Proper Reward function
- Algorithm selection
- Good personas/data



PPO collapsing

20

TOOLS AND FRAMEWORK

- **Tools**: Sagemaker Studio (Notebooks, python libs, codecommit…)
- **Framework**: Ray
- **RL Algorithms**: DQN, A2C,DDPG and finally **PPO** worked

21

PPO

Algorithm 5 PPO with Clipped Objective

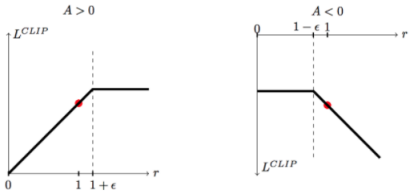
Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for



Source

22

REFERNCES!

- PPO Paper: https://arxiv.org/pdf/1707.06347.pdf, John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
- https://jonathan-hui.medium.com/rl-proximal-policy-optimization-ppo-explained-77f014ec3f12

23

QUESTIONS?

24