Bipedal Walker Using Augmented Random Search

Gautham Santhosh
IIM2014008

Problem

We are going to make a robot walk using reinforcement learning method in OpenAI Gym environment Bipedalwalker.

Using Augmented Random Search.

Why Augmented Random Search?

Deep learning algorithms work with gradient descent and back propagation this is very effective but also incredibly expensive in terms of computation

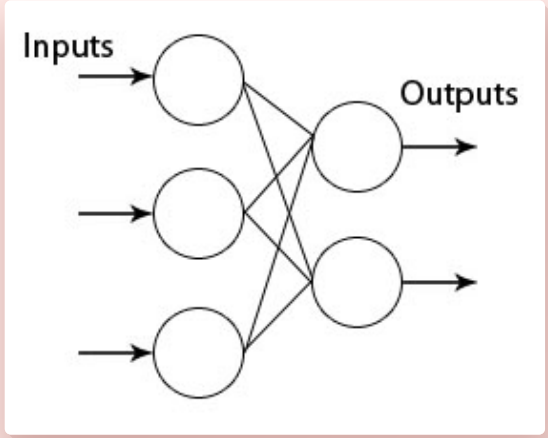
Training up to 15 times faster augmented random search is a shallow learning algorithm which uses random noise and genetic evolution to get cutting-edge performance on locomotion tasks

Perceptron

The basic building of this algorithm is a perceptron (the basic building block of a neural network / fully connected layer) .

Input Layer: The robot has sensors which output numbers numbers we normalize all the input data between 0 and 1.

Output Layer: Which outputs numbers which control the motors and actuators and the robots legs. In case of continuous control problem we no longer have binary 0 & 1 actions but continuous values which can the force of the motors it could be -1 is full reverse, 0 is totally off and 1 is full forward



How it works

> Add a random noise to weights
> Run a test
> If the reward improves, keep the new weights
> Otherwise discard

Method of Finite Differences

```
> Generate random noise(delta) the same shape as the
  weights (theta)
> Clone two versions of the current weights
> Add the noise to weights theta[+], subtract from
  theta[-]
> Test out both versions one episode each, collect
r[+],
  r[-]
> theta +=alpha(r[+] - r[-]).delta
> Test and repeat till needed
```

Normalize the Inputs

```
> normalized = (input/observation_mean) /
  observation_sigma
> mean = last_mean +
         (observation - last_mean) / num_observations
```

Training Loop

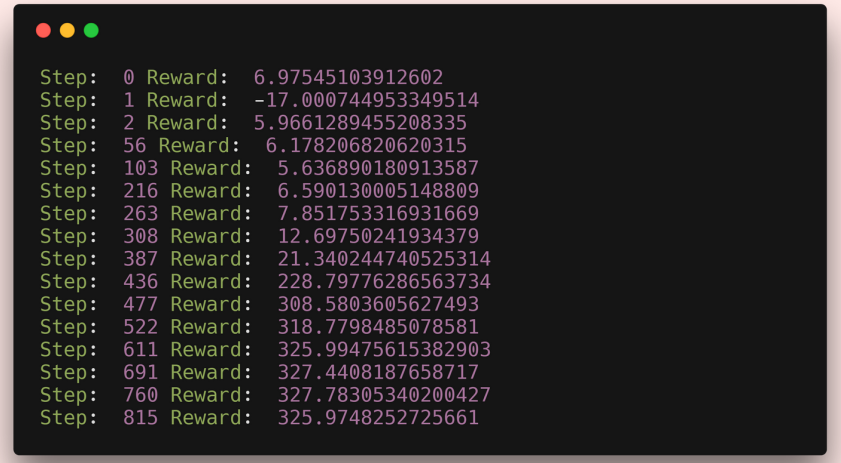
```
> Generate num_deltas,deltas & eval positive and
negative
> Run num_deltas episode with positive and negative
  variations
> collect roll outs as (r[+], r[-], delta) tuples
> calculate the standard deviation of all
  rewards(sigma_rewards)
> sort the roll outs by maximum reward and select the
  best num_best_deltas roll outs
> step = sum((r[+]-r[-])*delta), for each roll out
> theta += learning_rate / (num_best_deltas *
  sigma_rewards)*step
> Evaluate: Play an episode with the new weights to
  measure improvement
> Continue till the desired performance is reached
```

Dependencies

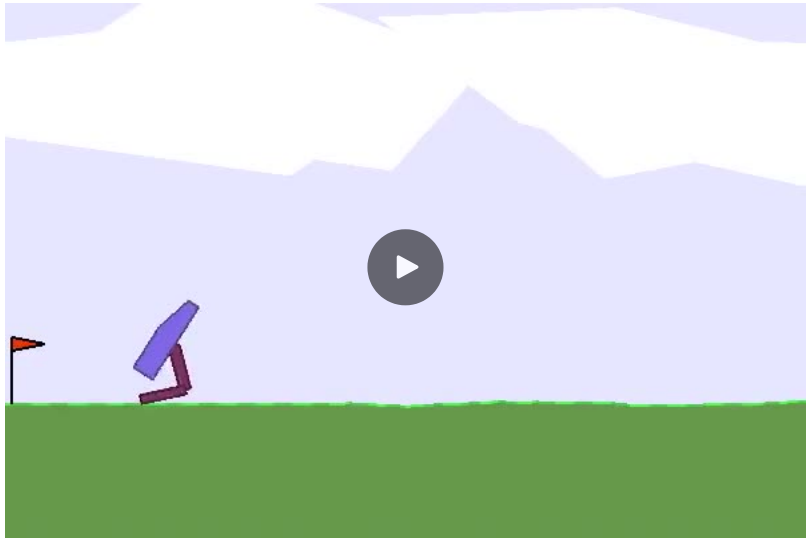
- OpenAI Gym
- Box2d
- PyBullet Enviornments
- Google Collab to train

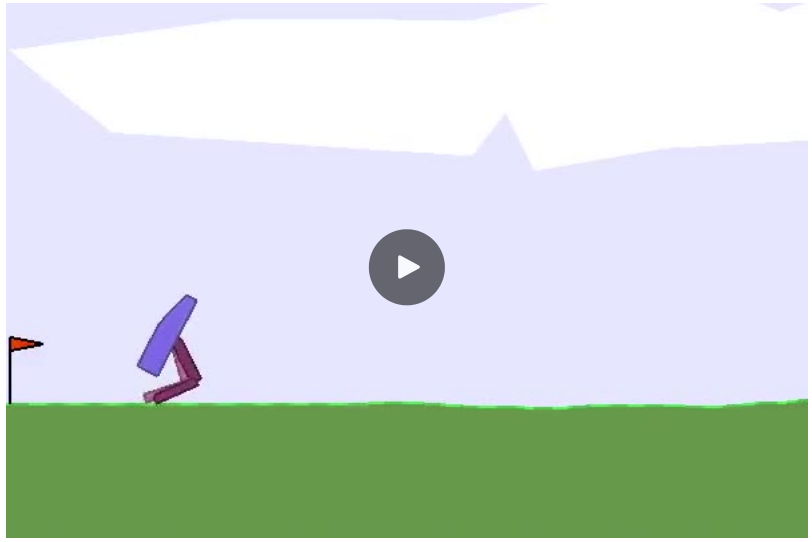
Results

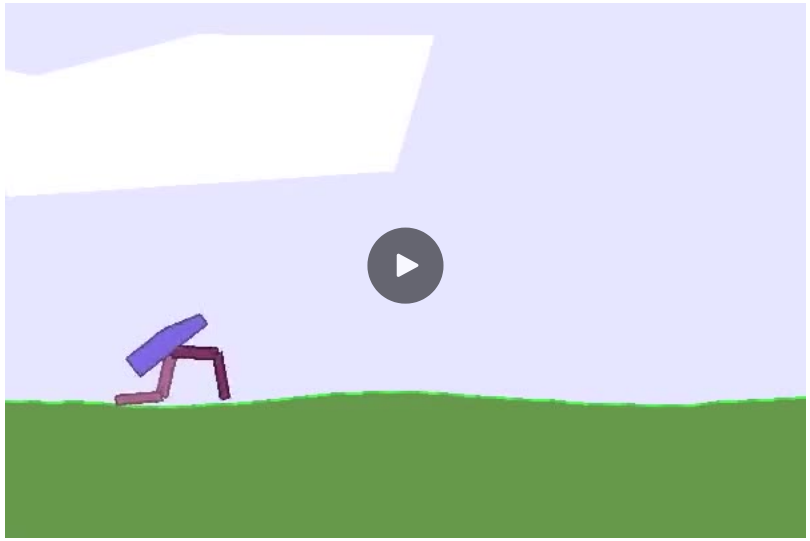
> Around 450 iterations it reached the needed 300+ reward score to reach till the end of track
> After which it stayed almost same
> Training took 1 hour in Google Collab less than most deep reinforcement learning models
> When robot starts learning to walk 360 iteration score ~ 20 , it quickly learns (100 steps) to goes score ~ 300



Videos







Thank You

Gautham Santhosh
IIM2014008