**Unraveling the Layers of Intelligence: Neural Networks, Reinforcement Learning and Proximal Policy Optimization**

Artificial Intelligence (AI), an emergent technology, is steadily gaining significance across all spheres of our lives. From accelerating responses to complex queries to enhancing the delivery of services, AI has become a game-changer, transforming how we interact with the world around us. From the daily interactions with virtual personal assistants like Apple’s Siri, Amazon’s Alexa, or Google Assistant, it’s evident that AI’s autonomy is redefining our relationship with technology. This ground-breaking technology has revolutionized a myriad of sectors, including healthcare and entertainment, and has positioned itself as an indispensable tool in contemporary scientific research.

Central to this AI-driven paradigm shift are neural networks, the powerhouse behind many of the advanced machine learning applications we see today. They mimic the human brain’s functionality at a granular level, starting with the perceptron, which serves as a basic building block for more advanced types of artificial neurons used in modern deep neural networks. We’ll delve into how these rudimentary units can assemble into a vast, dynamic network and how they’re trained via an ingenious method known as backpropagation. By iteratively learning from its mistakes and adjusting its internal parameters, a neural network refines its predictive prowess over time.

Next, we introduce reinforcement learning (RL), a distinctive branch of machine learning wherein an “agent” learns to make informed decisions by continually interacting with its environment. RL’s versatility has led to its implementation in an array of applications, from mastering video games to powering advanced robotics.

The highlight of our tutorial is Proximal Policy Optimization (PPO), a cutting-edge algorithm in the RL space. Rooted in policy gradient methods, PPO revolutionizes the way we approach policy learning. It ensures stability and robustness in the learning process while also maintaining remarkable efficiency. PPO’s adaptability makes it invaluable across a wide range of fields, from guiding autonomous vehicles over challenging terrains and powering sophisticated game-playing AIs. It has even found utility in the realms of resource management and algorithmic trading.

PPO’s impact is instrumental in thrusting the evolution of artificial intelligence into a new epoch. In this era, AI systems can learn to execute tasks with minimal supervision, adapt to novel environments, and make intelligent decisions. By the end of this tutorial, you’ll comprehend not only the mechanisms behind neural networks and PPO but also appreciate the transformative potential they hold for the future of AI.

Please note, many portions of this tutorial were written with the help of Chat GPT as a test to understand the system’s functions and how it can incorporate their own information with material written by a human.

# **Prerequisites**

Neural network

An artificial neural network is a type of computing system that processes data in the same manner as a human brain. It uses interconnected processing units, called neurons or nodes, formed into structured layers to function. This is a process of supervised learning where the network is given the inputs and corresponding outputs during training. It will then adjust the parameters in order to create a system with minimal difference between the prediction estimates and actual values. If there are more neurons and layers in a network, the level of complexity a pattern that it can learn increases.

A simple neural network is constructed of input layers, hidden layers, and an output layer (Figure 1). The input layer is where the network collects input features. Each dataset’s feature coincides with a neuron in this layer. Next, the neurons in the hidden layer(s) transfer information from the input to the output layers and perform computations. Finally, the output layer is where the results are found. Throughout each layer’s neurons, weighted sums are calculated to help determine the strength of each input. After a result is given in the output layer, an error term is found from the difference in predicted and actual results. Finally, local gradients in the hidden and output layers are backpropagated to adjust the weights.

A screenshot of a computer

Description automatically generated

*Figure 1: Structure of Simple Neural Network*

The neural network will perform training cycles to learn new data. In the training cycle, the training data will be divided into minibatches, which will help the process to be faster. Minibatches are subsets of input data that are used for training the policy. They will bring in stochasticity to prevent overfitting. After each minibatch, the weights of the network are updates during an epoch, which is one full pass through the training data. More specifically, each neuron will receive an input [] from a set of more neurons. The inputs will be multiplied by corresponding weights [] and added together with a bias term. Then, a weighted sum and activation will transform the input into an output. The formula for the weighted sum in the xth layer is as follows:

Where:

* is the weight, parameter, θ
* denotes the actual output
* is the bias term

Then, an activation, , is applied to get the output in the yth layer:

In the next layer, the hidden nodes will also calculate a weighted sum, , using the inputs from the input layer. Then, a similar process is as follows where the weighted inputs are summed together for a weighted sum that has an activation function, , that is applied to the weighted sum to help calculate approximations:

Furthermore, the network produces an output through a similar process with weighted sums being used to find the predicted values. Since the actual value is already known, the difference between the estimation and actual value is known as the error. Its formula is:

The goal is to minimize the error with the help of the least mean square function:

Where:

* is the actual output
* is 1 time step

The error will be backpropagated through the cycle to update the weights and biases and continue through the cycle. There are local gradients, , used in the hidden and output layers to help minimize the loss and adjust the weights. The local gradient for the hidden nodes is the derivative of the activation function, multiplied by the weighted sum of the following layer:

The local gradient for the output node is the negative derivative of the activation function, multiplied by the following layer’s weighted sum and the error from the output:

Through the process of adjusting the weights, the errors are propagated “backwards” through the cycle from output to input layer. The local gradients are used for each neuron. Then, the derivative of the loss with respect to the weighted sum is found by the chain rule. Mathematically, this is shown as:

Delta is needed to understand how to update the weights for the output. By using the gradients, we can use the highlighted formula to adjust our weights. Here, the loss gradient is also known as the activation gradient. The formulas can be found here:

Weight gradient:

Loss/Activation gradient:

Local gradient:

Weight update:

There are optimization algorithms that are used to adjust the network’s weights. For example, the Stochastic Gradient Descent (SGD) (Figure 2) will minimize the loss function in the network. This algorithm will use the gradient of the previous weights’ loss function to update the weights. The gradient will be computed using a minibatch in order to make the process more efficient. Another optimization algorithm is the Adaptive Moment Estimation (ADAM). It works to update each weight’s learning rate using the estimations from the first and second moment of the gradients.

A screenshot of a computer

Description automatically generated

*Figure 2: Gradient Descent (CITE)*

Reinforcement Learning

Neural networks can be used to approximate value functions and policies in complex reinforcement learning processes as they intake a state and outputs estimations. Reinforcement learning is a method of machine learning that uses an agent that analyses an environment to understand whether certain actions are beneficial or not. Actions’ beneficial nature is decided by a reward function that determines numerical reward values for various outcomes. There are two types of reinforcement learning approaches: model-based and model-free. In a model-based structure, the agent will have access to a model of an environment that it can study to predict state transitions and rewards. In a model-free structure, there is not model available for the agent. Instead, it must understand state transitions and rewards through action attempts in the environment. There are many algorithms that fall under reinforcement learning (Figure 3). Each different algorithm serves as a different way to apply this method.

A screenshot of a computer

Description automatically generated

*Figure 3: Various Reinforcement Learning Algorithms*

Policy (Actor) Method

The goal of this actor-only method is to optimize the policy directly. The policy will set the learning behaviour of an agent at a given time. It will take no regard for the value function when learning the optimal policy.

Value (Critic) Method

Critic-only methods will aim to find the value function in order to predict future rewards. The value of a state is the total amount of the reward an agent could expect to accumulate over the future, starting at that state. Value methods will learn an approximate value function to base their policy on.

Actor-Critic Method

This method of reinforcement learning is a combination of the policy and value methods. It contains the role of an actor who performs actions and the role of a critic that evaluates the actions using a value function.

Learn the Model Method

These model-based methods aim to have the agent create a model of an environment to study and predict the next state and reward based on the current state and action.

Given Model Method

In model-free methods, the agent must learn to make decisions without access to an environment or knowledge of its dynamics. Alternatively, the value function or policy are understood directly from interacting with the environment.

Proximal Policy Optimization (PPO)

The PPO is a reinforcement learning algorithm that extracts the positive outcomes from previous policy optimization methods and adjusts the parameters to create an efficient system. The PPO aims to control the size of each adjustment to ensure that the algorithm does not become unstable.

# **Core Concepts**

In the following sections, we will cover some concepts that are pertinent to the PPO algorithm. These are the key components that shape the function of this method.

Temporal Difference Learning (TD Learning)

TD Learning is a method where an agent studies an episode even when the final outcome is unknown. It is a blend of Monte Carlo and Dynamic Processing techniques. The main objective is to update the expected future reward of a state, which is based on the value of the subsequent state.

Temporal Difference Error (TD Error)

The TD error represents the difference between the estimated and observed values of a state. This error can be used to assess the accuracy of the estimates. A positive TD error indicates a higher-than-expected reward, whereas a negative TD error signifies a reward that is less than expected. The TD error plays a critical role in the value function and determines the magnitude of each update.

Advantage Function

The advantage function () updates the value function and policy. It measures the relative benefit of taking action in state compared to the average value of the state . It can be considered equivalent to the TD-error, , in TD Learning and the fundamental actor-critic. For a single timestep, it can be expressed as:

The generalized advantage estimation , that is applied for td-learning for all timesteps, can be represented as:

In this equation, λ (0 ≤ λ ≤ 1) controls the degree of bootstrapping, allowing a balance between bias and variance in the estimation of the advantage function. λ = 0 corresponds to the GAE being the TD-error ( ) for one timestep, and λ = 1 corresponds to no effect and the normal GAE is used. Furthermore, k is the step index, ranging from 0 to the end of the episode.

When the advantage function is truncated, it is as follows:

Value Function Objective

The value function tracks the expected reward an agent might receive at a certain state. Then, the value loss function minimizes the least square errors between real and estimated rewards. The formula is as follows:

The parameters, , are adjusted to reduce the loss. Through the Actor-Critic method, this value function acts as the critic that estimates the advantage function to alter the policy, which is referred to as the actor.

For all timesteps, we use:

For one timestep, we have:

PPO Policy Objective

A policy objective, derived from the Trust Region Policy Optimization (TRPO). Is described by the formula:

where is the policy function determining the probability of in state

Here, , is a probability ratio that helps with convergence. If the probability were to be higher for a new policy where a better action is taken, there would be an increase in advantage and a ratio greater than 1. On the other hand, if the probability of the new policy’s action was lower than the old policy’s probability, the ratio would be less than 1 with a reduction in advantage.

In each iteration, the policy parameters, , are adjusted in hopes of maximizing the policy’s improvement. However, if too many or big of changes are made in one iteration, it can render the algorithm inefficient or ineffective. Thus, we use the formula, , to assure that there is not a large deviation from the previous policy iteration. The formula is as follows:

Overall PPO Objective

This function is optimized. The main goal is to adjust the parameter, , to maximize A, while avoiding large updates that could render the algorithm inefficient. The formula is:

Where:

* denotes the parameters for policy network and value network
* is the current timestep
* is the objective for Policy – maximize the advantage
* is the objective for Value Function – minimize the LSE, is an objective function so we need to put a negative sign
* are the relative weights on the sub-objectives
* is the state at timestep t
* is an entropy bonus – a random factor for policy exploration
* is empirical expectation at timestep

This objective aims to find better actions (policy objective) while estimating the value of the action across various states (value function), all while maintaining a level of randomness (entropy bonus) to ensure exploration of all possibilities.

# **The Algorithm**

The algorithm initiates with random selected ­0 values for the policy and value network parameters. The process continues through several iterations until the policy converges. In each cycle, actions are chosen from the policy at random, rewards are received, and states are updated. The advantage for all states in all rollouts is computed to determine the best action for each state. The parameters for the next cycle are adjusted to enhance the policy. The objective function is minimized using Stochastic Gradient Descent (SGD) with ADAM, and the network is trained for X epochs, taking Y steps per epoch with size Z minibatches. Step-by-step, the process is as follows:

Initialize ­0­­

# Initialize the values for the policy and value network parameters.

For k = 1, 2, … until convergence

# Run the algorithm loop until the policy reaches convergence.

Initialize

# Initialize the starting state of the environment with the function provided.

:

# Choose an action at random from the policy. provides probabilities.

:

# Once the action is taken, reward is received and converted to state

Start a new episode (a complete game) if is the end of an episode.

Compute advantage for all the states in all rollouts.

# This will compute whether a certain action in a particular state is the best choice over other actions in that state.

Use the states to trainfor the next cycle

# Continue the process and alter the parameters for the next cycle to improve the policy.

Using Stochastic Gradient Descent (SGD) with ADAM, minimize the objective function

Train X epochs with Y steps of minibatch size Z

# Train network for X epochs, taking Y steps per epoch with size Z minibatches.

## **Implementing the PPO Algorithm – CartPole Problem**

The CartPole problem is a rudimentary reinforcement learning problem that uses the PPO algorithm. In this example, we are the agent that is controlling a cart that is balancing a pole atop it while moving on a frictionless track. The objective is to balance the pole while applying either +1 or -1 forces (the actions) to the cart. A +1 reward is given each timestep that the pole remains upright. If the pole is greater than 15 degrees from vertical, or the cart moves greater than 2.4 units from the center, the episode ends. When the average total reward for the episode over 100 consecutive trials is greater than or equal to 195, the problem is considered solved.

## Applying the PPO Algorithm in the Command Prompt

To start, the PPO source code must be downloaded to your system. The programs came from the ICLR blog post, *The 37 Implementation Details of Proximal Policy Optimization* (Huang et al., 2022). The repertoire of code can be found here: <https://github.com/vwxyzjn/ppo-implementation-details> . I refactored the PPO code to improve its readability. The command prompt I used was Windows Subsystem for Linux (WSL). It is important to ensure that Python is installed into the command prompt before starting. There will be commands that use the program Anaconda, so it must also be downloaded to WSL. In order to create the program, a new environment must first be created. Then, various packages must be installed. Namely, Pytorch is a learning framework that allows automatic differentiation and gradient descent. It is where the policy and value network is defined, and used to calculate the loss function and adjust parameters when training the PPO algorithm. As the model is defined in Pytorch, we will train the model using “gym” in our created environment. After all the code has run, a video will pop up showing the cart balancing the pole. It will look like this:

## CartPole Installation Steps

To install Python onto WSL:

# The package list is updated, and Python 3 and its packages are installed.

sudo apt update

sudo apt install python3

sudo apt install python3-pip

To install Anaconda onto WSL:

# The Anaconda installer for Linux is downloaded. Note that the URL may change over time with new versions of Anaconda. The device may need to be restarted after the bash command where the installer script is run.

sudo apt install wget

wget https://repo.anaconda.com/archive/Anaconda3-2021.05-Linux-x86\_64.sh

bash Anaconda3-2021.05-Linux-x86\_64.sh

To run the Cartpole Problem:

# The environment is created and activated.

conda create -n ppo python==3.8.17 -c conda-forge

conda activate ppo

# The PyOpenGL and ffmpeg libraries are installed from the conda-forge channel. The ffmpeg library is a software program with libraries that can be used for videos, audio, and multimedia files.

conda install -c conda-forge pyopengl

conda install -c conda-forge ffmpeg

# Try to run ffmpeg now to see if that gives an error. Resolve any problems before you continue.

# The OpenGL Extension Wrangler (GLEW), Mesa 3-D graphics and GLFW3 libraries are installed.

conda install -c conda-forge glew

conda install -c conda-forge mesalib

conda install -c menpo glfw3

# Run one of the following 2 lines depending on whether you have a GPU

# Run this one if you have a GPU. The packages necessary for this problem are installed. Specifically, pytorch, torchvision and torchaudio from the pytorch channel.

conda install pytorch torchvision torchaudio pytorch-cuda=11.8 -c pytorch -c nvidia

# Run this one if you do not have a GPU.

conda install pytorch torchvision torchaudio cpuonly -c pytorch

# Various libraries are installed through pip, a Python package installer. These libraries are used to facilitate running the Cartpole problem. Some notable functions developing and comparing reinforcement learning algorithms and providing a set of environments to test the algorithms from the gym toolkit. Another library, OpenCV contains programming functions that are used to process videos and images.

pip install PyOpenGL\_accelerate

# The library PyOpenGL\_accelerate is installed. It is an open module of OpenGL that accelerates some PyOpenGL features through more efficient operations.

pip install tensorboard==2.12

# pip is the Python package installer. The 2.12 version of tensorboard is installed. It is a tool that provides measurements and visualizations that are required for the machine learning workflow.

pip install setuptools==59.5

# Setuptools is an actively maintained and stable library that facilitates packaging Python projects. It is an indirect dependency of various Python packages.

pip install imageio-ffmpeg==0.3

# The 0.3 version of imageio-ffmpeg is installed to read and write multimedia files.

pip install gym==0.21

# Gym is a toolkit used to develop and compare reinforcement learning algorithms. It also provides a set of environments to test algorithms.

pip install pyglet==1.5.21

# Pyglet is a Python library used to create games and multimedia applications. The Gym toolkit uses it to create the GUI and render environments.

pip install opencv-python==4.5.5.62

# OpenCV is a library of programming functions that is often used to process videos and images. This installs version 4.5.5.62 of the OpenCV Python library.

pip install cython==0.29.26

# Cython is a programming language that focuses on being a superset of Python. It gives C-like performances written in majorly Python code. This installs version 0.29.26 of Cython. It is often used to optimize Python code and interface C libraries.

pip install lockfile

# Lockfile is installed. It is used to create an easy way to handle file locking. It proves useful when the program needs to prevent concurrent access to shared resources.

pip install -U 'mujoco-py<2.2,>=2.1‘

# This installs Mujoco, a physics engine for detailed, efficient rigid body simulations with contacts.

# Using the Python script with the PPO algorithm from the repertoire, use the tool VirtualGL to ensure the graphics rendering uses GPU hardware acceleration. This code will run the Cartpole problem. ‘-capture-video’ will run the video for this problem.

vglrun python ppo\_refactor.py –capture-video

## **Humanoid Problem**

In this problem, we are attempting to program the humanoid robot to perform human-like movements. Like the Cartpole problem, it uses the PPO algorithm to adjust the actions of the robot to improve its performance.

## Running the Humanoid Problem in WSL

Before starting, make sure that WSL and Ubuntu are installed. All code will be run in WSL. Next, the MuJoCo directory must be set up. MuJoCo is a physics engine that is used for detailed and efficient body simulations. Then, all relevant libraries will be installed. After running each command of code, the source code for the humanoid problem will run. The code may run for upwards of 4.5 hours. After it is complete, in the specific humanoid folders, there will be a collection of videos for each episode of this problem. These videos are the humanoid training videos from the environments. They will look something like this:

## Humanoid Installation Steps

# Create the MuJuCo directory in the home directory and download the 2.10 version for Linux.

mkdir ~/.mujoco

cd ~/.mujoco

wget <https://mujoco.org/download/mujoco210-linux-x86_64.tar.gz>

tar -xzvf mujoco210-linux-x86\_64.tar.gz

# The environment variable ‘LD\_LIBRARY\_PATH’ is extended and linked to the required libraries from MuJoCo. The specific path will vary for each user.

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/… /.mujoco/mujoco210/bin

# Install any missing libraries. These libraries will help to facilitate running the Humanoid problem. They are installed to encode video streams, render graphics, etc.

sudo apt-get install libx11-dev

# This downloads the libraries for development files for the windowing system (X11).

sudo apt-get install libglew-dev

# This installs the OpenGL Extension Wrangler Library. It is used for modern graphical functions.

sudo apt-get install libx264-dev

# This installs the development files for the x264 library. It is used to encode video streams.

sudo apt-get install libosmesa6-dev

# This installs the library, Mesa. It is a 3D graphics library that does off-screen rendering.

sudo apt-get install libgl1-mesa-glx libglfw3 patchelf

# This will download different graphics and utility libraries.

# Install and update the old libffi7 library to support applications.

wget http://es.archive.ubuntu.com/ubuntu/pool/main/libf/libffi/libffi7\_3.3-4\_amd64.deb

# dpkg -i installs the Debian package.

sudo dpkg -i libffi7\_3.3-4\_amd64.deb

# The system’s package databases and installed packages are updated.

sudo apt-get update

sudo apt-get upgrade

# Go to the folder containing the humanoid code using your specified path.

cd/.../humanoid.py

# Run the humanoid code.

python humanoid.py

Work Cited

Huang, S., Doussa, R. F. J., Raffin, A., Kanervisto, A., & Wang, W. (2022, March 25). The 37 Implementation Details of Proximal Policy Optimization. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/