UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

ACTIVE ROBOTIC PERCEPTION BY MEANS OF OBJECT MANIPULATION

Diplomová práca

2023 Bc. Patrik Modrovský

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

ACTIVE ROBOTIC PERCEPTION BY MEANS OF OBJECT MANIPULATION

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Vedúci práce:	prof. Ing. Igor Farkaš, Dr.

Bratislava, 2023 Bc. Patrik Modrovský

43738645

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Patrik Modrovský
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	informatika
Typ záverečnej práce:	diplomová
Jazyk záverečnej práce:	anglický
Sekundárny jazyk:	slovenský

Názov: Active robotic perception by means of object manipulation
Aktívne videnie robota prostredníctvom manipulácie s objektom

Anotácia: Proces percepcie v robotickom kontexte je charakteristický aktívnou interakciou s objektmi počas učenia. Tieto procesy súčasne umožňujú "lacné" generovanie rozmanitých trénovacích dát, ktoré sa dajú využiť napríklad na budovanie 3D reprezentácie objektu alebo predikovanie perceptuálnych dôsledkov akcií robota.

Cieľ: 1. Preštudujte literatúru v oblasti aktívnej percepcie pomocou manipulácie s objektom.
2. Navrhnite model neurónovej siete učiacej sa posilňovaním, ktorá umožní robotovi skúmať objekt držaný v ruke v rôznych perspektívach.
3. Natrénujte dopredný model na predikciu perceptuálnych efektov vykonávaných akcií.
4. Vyhodnoťte presnosť oboch modelov.

Literatúra: Kostrikov I., Erhan D., Levine S. (2016). End-to-end active perception. NIPS, http://www.dumitru.ca/files/DL_symposium_active_perception.pdf
Zaky Y., Paruthi G., Tripp B., Bergstra J. (2020). Active Perception and Representation for Robotic Manipulation, https://arxiv.org/abs/2003.06734

Vedúci:	prof. Ing. Igor Farkaš, Dr.
Katedra:	FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry:	doc. RNDr. Tatiana Jajcayová, PhD.
Dátum zadania:	16.10.2022
Dátum schválenia:	06.11.2022

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

....

študent vedúci práce

I hereby declare that I have written this thesis by myself, only with help of referenced literature, under the careful supervision of my thesis advisor.

. . . .

Bratislava, 2023 Bc. Patrik Modrovský

Acknowledgement

do later

Abstract

This thesis addresses the problem of robot-object interaction and the prediction of perceptual consequences of robot actions. In the first part, it introduce reader into basic of neural networks and reinforcement learning, along with brief explanation of used PPO algorithm. In next chapters, design and implementation of used NICO robot is explained. Model of nco was created by Mima Mattová and modified for training purposes by Iveta Bečková. Training was done virtualy in Unity environment with The Unity Machine Learning Agents Toolkit. For traning we The Proximal Policy Optimization (PPO) algorithm based upon neural networks and Policy gradient.

Keywords: robot, ppo, reinforcement learning, Unity, neural networks

Abstrakt

Táto práca sa venuje problematike interakcie robota s objektamy a predikovanie perceptuálnych dôsledkov akcií robota. Súčaťou tejto práce je aj úvod do neurálnych sietí a učenia posilňovaním, spolu s stručním vysvetlením fungovania použitého algoritmu PPO. Dalej je tu popísaný návrh a implementácia učenia robota NICA vo virtuálnom prostredí Unity. Model NICA bol vytvorený Mimou Mattovou a ďalej upravný pre potreby simulácie Ivetou Bečkovou. Na jeho trénovanie bol použitý Unity Machine Learning Agents Toolkit. Na trénovanie bol použitý algoritmus PPO (Proximal Policy Optimization), využívajúci neurálne siete a Policy gradient.

Kľúčové slová: robot, ppo, učenie posilňovaním, Unity, neurálne siete

Contents

Motivation

...

Chapter 1

Introduction

1.1 Neural networks

An Artificial Neural Network (ANN) is a processing system made from large number of interconnected simple processing elements - neurons. Architecture ANNs was inspired by structure of biological brain - both brain and ANN process information similarly and learns from examples.

This creates different approach to problem solving compared to conventional computing. Instead of predictable algorithmic approach, ANNs allows solving problems without understanding of problem, but with lower accuracy. [7]

1.1.1 History

In 1943 neurophysiologist Warren McCulloch and mathematician, Walter Pitts developed first model of ANN. Neurons in their models had fixed threshold and were able to simulate simple logic functions. [7]

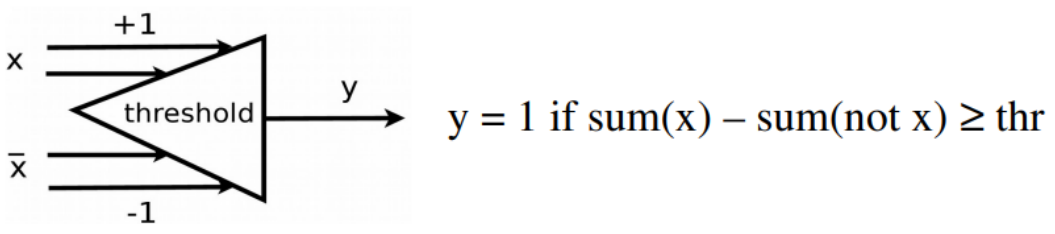

Figure 1.1: Model of an artificial neuron according to McCulloch and Pitts [4]

After initial period of enthusiasm and few other contributions to the field, period of frustration followed. During that time there were only few researchers continued work on solving problems with neural networks. During this time Klopf developed basis for learning and Werbos developed back-propagation algorithm.

This period lasted up to the 1980s, when multiple new contributions renewed wider interest.Hopfield described recurrent neural networks and presented paper *Neural Networks and Physical Systems with Emergent Collective Computational Abilities*.[7]

Neural networks become recently much more publicly know, caused by advances in computing power allowing much bigger number of neurons. It allowed for much more complex tasks as image generation from natural language prompts or chatbots capable of deeper conversations.

1.1.2 Different models of neural networks

There are hundreds of different models and architectures of neural networks with different advantages and disadvantages for different purposes. Some of the most used architectures are:

- Single perceptrons
- Multi-layer perceptrons
- Convolutional neural network
- Recurrent neural network
- Self-Organizing Maps
- Radial Basis Function Network
- Hopfield autoassociative memory

In practise, multiple architectures are often combined to achieve desired outcome.[4]

Simple perceptron

It was proposed in 1958 by American psychologist, F. Rosenblatt. It is more general model, with free parameters, stochastic connectivity and threshold elements.[4]

Activation of this perceptron consists of weighted sum of inputs, from which we subtract threshold θ (1.1). Result is then run through activation function, either uni/bipolar for discrete perceptron or sigmoid for continuous perceptron. In practise, threshold is expressed as another weighted input x_{n+1}, with static value of -1, commonly called bias(1.2).

$$o_j = f(\sum_{i=1}^{n} w_{ij}x_i - \theta_j) \tag{1.1}$$

$$o_j = f(\sum_{i=1}^{n+1} w_{ij}x_i) \tag{1.2}$$

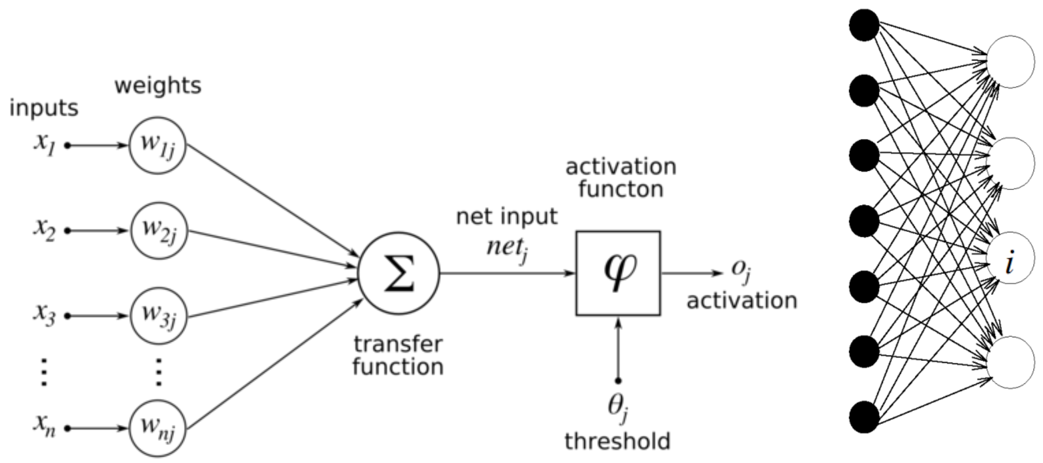

Figure 1.2: Model of simple perceptron and layer of simple perceptrons [4]

Learning rule for adjust weights in discrete perceptron:

$$w_j(t+1) = w_j(t) + \alpha(d - y)x_j \tag{1.3}$$

And for continuous perceptrons:

$$w_j(t+1) = w_j(t) + \alpha(d^{(p)} - y^{(p)})f'x_j \tag{1.4}$$

Where $\alpha(Alpha)$ represents learning rate.

Simple perceptron can solve only lineary separable classes. This caused loss of interest in neural networks in many researchers 1970s as many real problems are lineary non-separable, thus simple perceptron cannot be used to solve them.

Multi-layer perceptron

Multi-layer perceptron(MLP) is probably one of most used architectures today. It is generalisation of simple perceptrons and it contains additional hidden layers. MLP originates from *Rumelhart & McClelland: Parallel distributed processing* but was earlier described by Werbos in 1974. It is response to critique on perceptrons.[4]

In MLP Activations are similar to activations in simple perceptron. Their distinction comes from addition of hidden layers and additional weights. Due this change, output layer is calculated from hidden layer, instead of inputs.

$$h_k = f(\sum_{j=1}^{n+1} v_{kj}x_j) \tag{1.5}$$

$$y_i = f(\sum_{j=1}^{q+1} w_{ik}h_k) \tag{1.6}$$

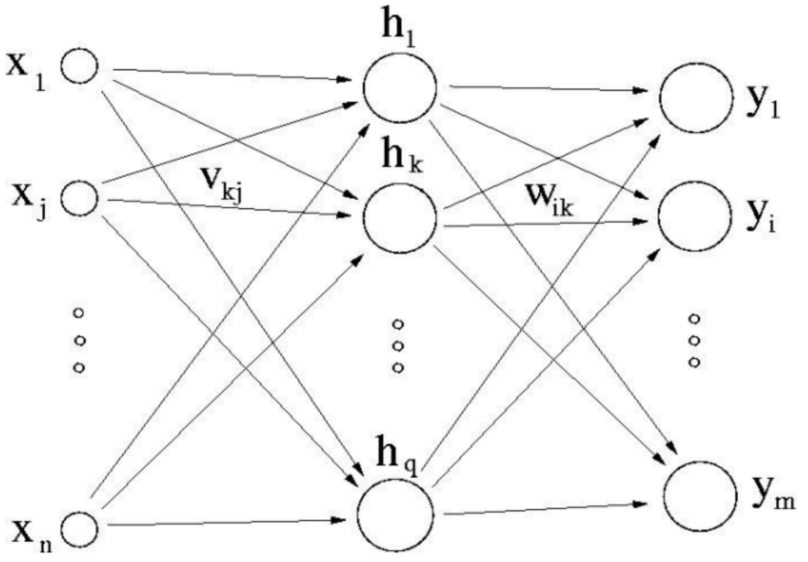

Figure 1.3: Model of multi-layer neural network [4]

Activation functions on hidden layers doesn't have similar constrains as output function, but linear function should not be used, as it doesn't add any depth to network. Common activation functions are sigmoid or hyperbolic tangent function.

Learning of MLP uses back-propagation algorithm with different equations for hidden and output layers. Weight adjustments are calculated backwards from output layers to the first hidden layer.[4]

Equation for calculating Hidden-output weights:

$$w_{ik}(t+1) = w_{ik}(t) + \alpha\delta_i h_k \text{ where } \delta_i = (d_i - y_i)f'_i \tag{1.7}$$

And for Input-hidden weights:

$$v_{kj}(t+1) = v_{kj}(t) + \alpha\delta_k x_j \text{ where } \delta_k = (\sum_i w_{ik}\delta_i)f'_k \tag{1.8}$$

1.1.3 Creating neural networks

To successfully create a neural network for accurate classification or prediction tasks, three key components play major role: high-quality dataset, the selection of an appropriate neural network model, and the identification of optimal hyperparameters for the training process itself.

Training process

During training, model iterates through dataset, predict outcome and calculates weight changes based on error. One such iteration is called **epoch**.

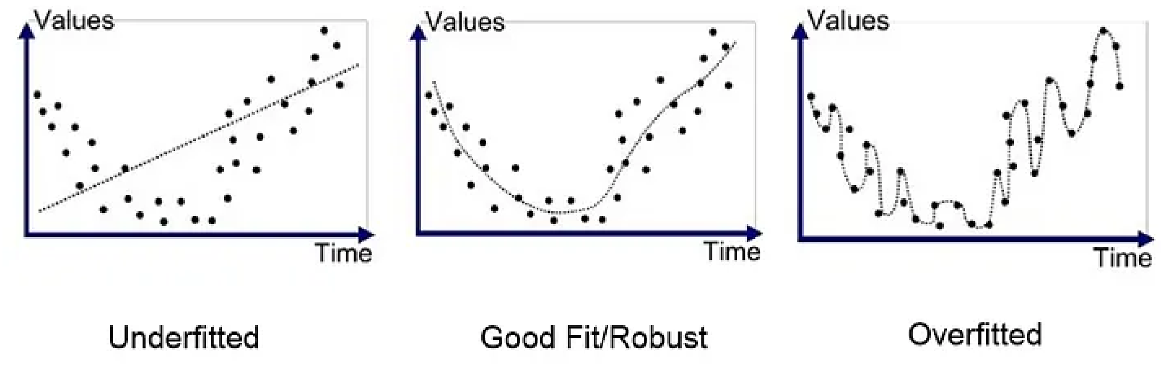

Figure 1.4: Underfit, Good fit and Overfit [2]

One training iteration usually consist of these steps:

- for each input x and desired output d (in random order)

  1. compute output: $y = ?(Wx)$
  2. compute error: $e = d - y$
  3. compute adjustment: $\Delta W = ?(W, x, y, e)$
  4. adjust weights:$\Delta W(t+1) = W(t) + \alpha \Delta W$

This iteration continues until certain stopping criteria is meet, usually number of passed epochs or accuracy of model. Properly setting stopping criteria is important in order to prevent overfitting or underfitting of model for certain data. **Underfitting** is when model didn't properly capture desired problem. **Overfitting** is when model is trained very well for training data, but reacts too much on random outliers or noise, making it very inaccurate in generalisation (Fig. 1.5). [2]

In relation of correct fitting and training length, despite training error decreasing, at certain point validation error stops decreasing and start increasing, as model is becoming more overfit, reacting to noise in the dataset.

This is also called *sequential training.* Other used type of training is called *batch training.* In batch, instead of iterating through input-output pairs is everything calculated in parallel. This is done through matrix multiplication and can significantly speed up calculations, at cost of higher memory requirement and harder implementation.[4]

Datasets

Dataset is a collection of structured data used for various purposes such as analysis, research, or machine learning.

Selection of proper dataset is crucial for training process - dataset should properly represent data from problem. In practise, getting ideal datasets is complicated and

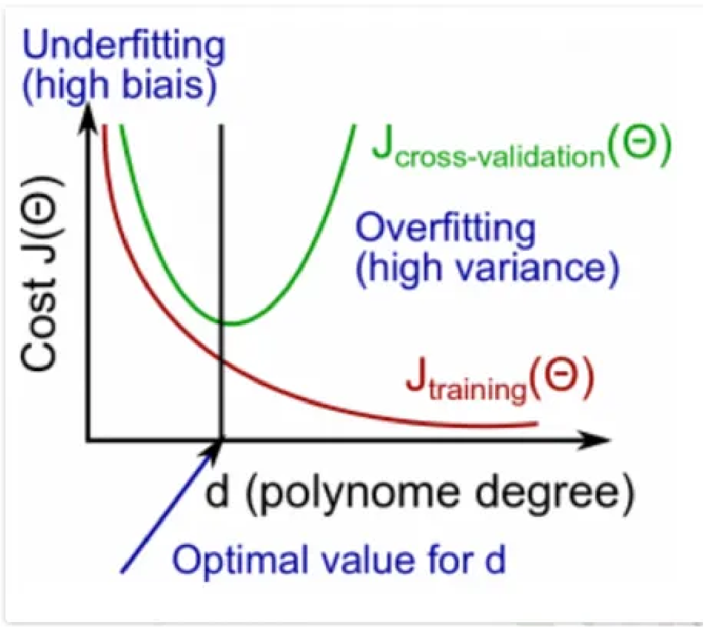

Figure 1.5: Error in relation to training length [2]

often impossible and contain too small sample of data. This can be partially corrected by using data augmentation.

Data augmentations refers to changing available dataset to better suit task needs or generating new data based on present one. As example, in article *Improving handgun detection through a combination of visual features and body pose-based data* [12], authors edited original dataset of high quality images by editing lighting and "zooming out" to make humans on them appear smaller (Fig. 1.6). This changes simulated of CCTVs, that often cover poorly lit big areas. They also used generation of new data by flattening 3D pose data to 2D from multiple points of view (Fig. 1.7).

Although, we could use dataset for training of model as is, and training error could be sufficiently minimize . However, model may not be good enough at generalisation as we would be unable to detect overfitting. This is why datasets are commonly split into two or three parts:

- Training set
- Validation set
- Testing set

Training set is used for training itself - it is used every epoch and it directly affects model weights. Training sets are usually biggest from all three. **Validation set** is used for evaluation of model and fine tune its hyperparameters. It can be used after every epoch, or every few epochs, based on performance. **Testing set** is used for measurement of model performance after training was finished. It is usually used only once (Fig. 1.8).

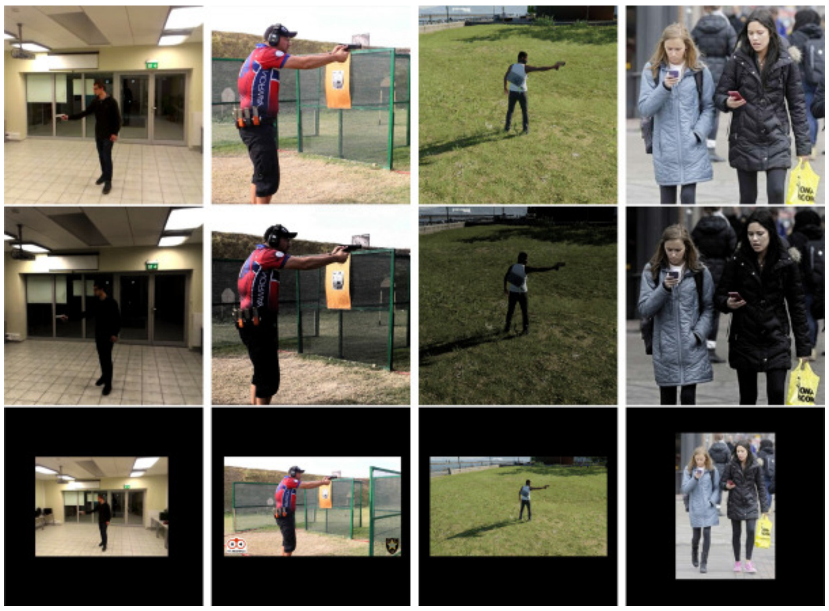

Figure 1.6: Unedited and edited images [12]

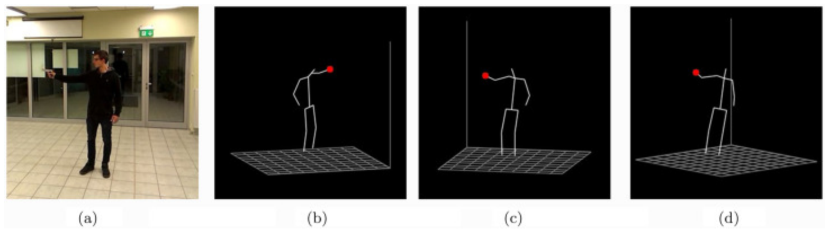

Figure 1.7: Different points of view [12]

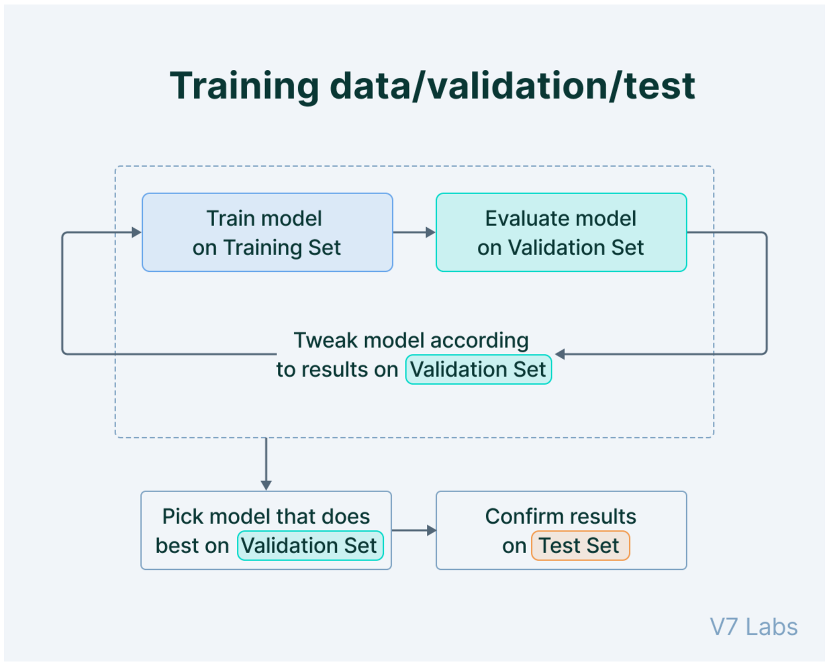
Training data/validation/test
Train model on Training Set
Evaluate model on Validation Set
Tweak model according to results on Validation Set
Pick model that does best on Validation Set
Confirm results on Test Set
V7 Labs

Figure 1.8: Difference between training, validation and testing sets [1]

There is no hard rule on how dataset should be split as it depends on model and task, but in general, $80\% - 10\% - 10\%$ for training, validation and testing is good split to start from. Lowering amount of data for training increase training variance and with less data for validation/testing, model evaluation and performance will have greater variance.[1]

Simplest way to distribute data between splits is **Random sampling**. In random sampling data are picked at random from full dataset. This work best for balanced datasets, but in unbalanced ones it can cause bias towards one category. For example in dataset with 800 images of dogs and 200 of cats, training set could get most of dog images, making cat detection unreliable. For these tasks **Stratified sampling** is used. In stratified sampling, data is picked from each category separately in same ratio as split between training, validation and testing sets.[1]

Different commonly used way to split dataset is **k-Fold Cross-Validation**. It is especially useful for smaller datasets where result of training can be strongly affected by randomness. In k-Fold Cross-Validation, dataset is split into k non-overlapping subsets of same size. During training, one subset is used for validation and the remaining $k-1$ are used for training. This process is repeated k times for every subset. Results from k trainings are then averaged into one performance estimation - cross-validation coefficient, that is used to pick best model (Fig. 1.9). [8]

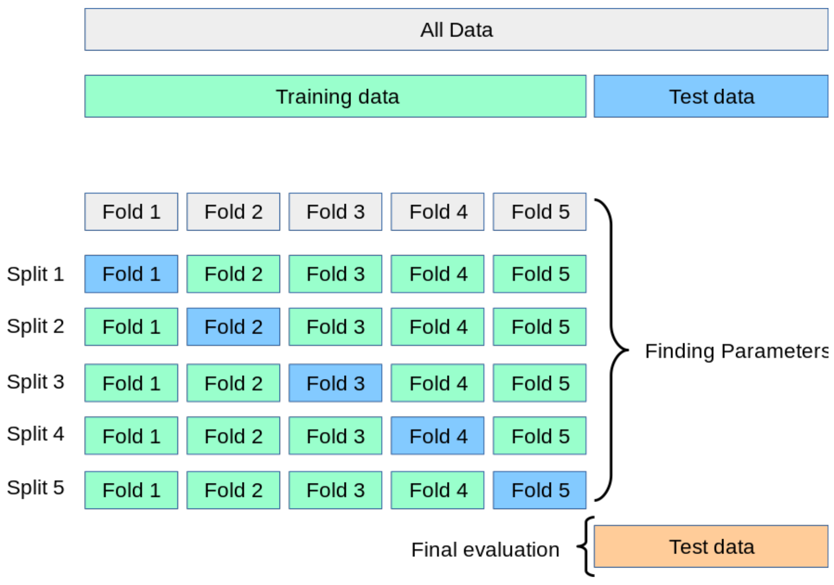

Figure 1.9: Example of 5-fold cross validation. [10]

Basic k-Fold Cross-Validation can suffer from same class imbalance as random splitting. It is solved by using stratified k-Fold Cross-Validation, which preserves class ratios.

1.2 Reinforcement learning

In the realm of artificial intelligence, learning methods can be categorized into three types: supervised learning, unsupervised learning, and reinforcement learning, each having its own uses and advantages.

Supervised learning learns from a labelled dataset, where each input has a corresponding desired output. The goal is to minimize the difference between values predicted by the model and true labels from dataset and generalize for new, unseen data . This method is commonly used for regression and classification tasks (function approximation, pose detection). Prediction accuracy is heavily upon quality and size of the dataset.

Unsupervised learning learns on unlabelled datasets. It discovers inherent patterns or structures within the data without any explicit guidance or instruction. . Common tasks include clustering similar data points or reducing dataset dimensionality. Unsupervised learning is used for exploring large datasets, uncovering hidden relationships, and helping in data visualization.

Reinforcement learning learns from interaction with environment through actions and rewards. Its goal is mapping environment states to actions order to maximize long term reward. It is used in robotics, self-driving cars or playing games. Two main characteristics of RL are trial and error and delayed reward.

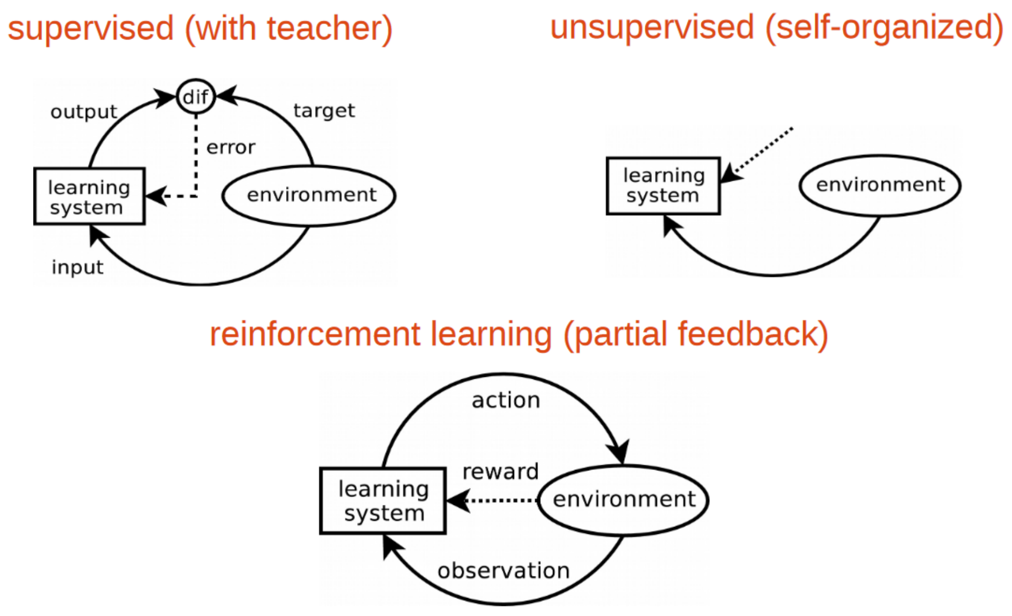

Figure 1.10: Different categories of AI learning methods [4]

1.2.1 Introduction to RL

It is mapping actions for every state in order to maximize rewards. Rewards are not know to the learner and have to be discovered by trying them. In more complex cases, actions affects all subsequent states and rewards.

Reinforcement learning simultaneously refers to problem, class of solution methods and field that studies this problem.

In RL problem, a learning agent must be able to sense state of the environment, take actions that can affect said state and have goal or goals relating to the state of environment.

Reinforcement learning is different from supervised learning. Supervised learning learns from already mapped set of data, where every label has a correct action and model tries to generalise for previously unseen data, for example correctly categorize animal in picture or calculate y from input x. Despite its usefulness, it is inadequate for learning from interactions, as it is often impractical to obtain representative examples of desired action for all situations in which the agent has to act. Agent must be able to learn from its own experience in order to find best action for every state.[14]

A good way to understand RL is through examples from real life:

- Newborn gazelle calf learning how to run in under half hour
- Baby learning how to stack cubes into high tower
- Kid learning how to ride a bicycle
- Chess player learning how to play chess

In all this actions, decision-making agent learns through interaction with environment, despite uncertainty about it. Falling on the ground or cube tower collapsing provides negative reward and reaching partial goals give positive reward. Actions may affect the future state of the environment (e.g., the next chess position, future position of cube), thereby affecting actions and opportunities available to the agent. Correct choice of action requires taking into account both immediate future and indirect, delayed consequence thus requiring foresight or planning.[14]

1.2.2 Elements of Reinforcement Learning

There are four basic subelements of reinforcement learning beyond agent and environment:

- Policy - defines behaviour
- Reward signal - defines goal and rewards

- Value function - defines what is good in long term
- Model of environment - optional element that mimics environment

Policy defines agent behaviour at given time - it is mapping of actions from perceived states of environment. It can be as simple as lookup table or it may involve complex calculations. Instead of one action, policies often return probability for each action. Policy corresponds to stimulus–response rules or association from psychology.[14]

Reward signal defines goal of a reinforcement learning problem. After each action, the environment sends to agent evaluation of current state - **reward**. Reward defines good and bad events for the agent, as agent tries to maximize total reward over long run.The reward signal is primary basis for alteration of the policy. If action had low reward, probability of it being taken is lowered and similarly, if action brought high reward, its probability in policy is heightened. We can think of rewards as pleasure or pain.[14]

Value function indicates what is good in long - it is total amount of expected reward agent can accumulate from this state. Values indicates long term desirability of states, based on possible future states and their rewards. States can have high reward, but are followed by more states with lower ones. On the other hand different state have lower one, but allows for better future states.

Example is building tower from blocks. Always placing block on top will lead to high tower and big immediate reward, but will quickly lead to collapse. On the other hand, creating stronger foundations brings little reward, but allows for much higher tower and reward in the end.

Without rewards, we could not get values, and values only purpose is to maximize reward. But it is values we use during evaluation of decisions - we seek actions that will bring states with highest vale and not reward. Unfortunately, it is harder to determine values, as they must be estimated from observations over entire agent lifetime.[14]

Model of environment mimics the behaviour of the environment. It allows inferences to be made about how the environment will be affected by agent actions. For example, given a state and action, the model might predict the next state and reward. They are used for planning - consideration of future situations before they are experienced. Methods using model are called *method-based.* Opposite of that are *model-free* methods, based on trial and error and almost viewed as opposite of planning.[14]

1.2.3 Tabular Solution Methods

Tabular methods are used for tasks, where sate and action spaces can be represented as arrays or tables, thus methods can often find optimal value function and optimal policy. For this method, **k -armed Bandit Problem** is often used as example, it is analogy to slot machine, often called "One armed bandit". In this problem, agent is choices of actions - "levers", each with its own reward distribution (Fig. ??). Goal of agent is to maximize total reward over multiple steps by focusing on best lever. [14]

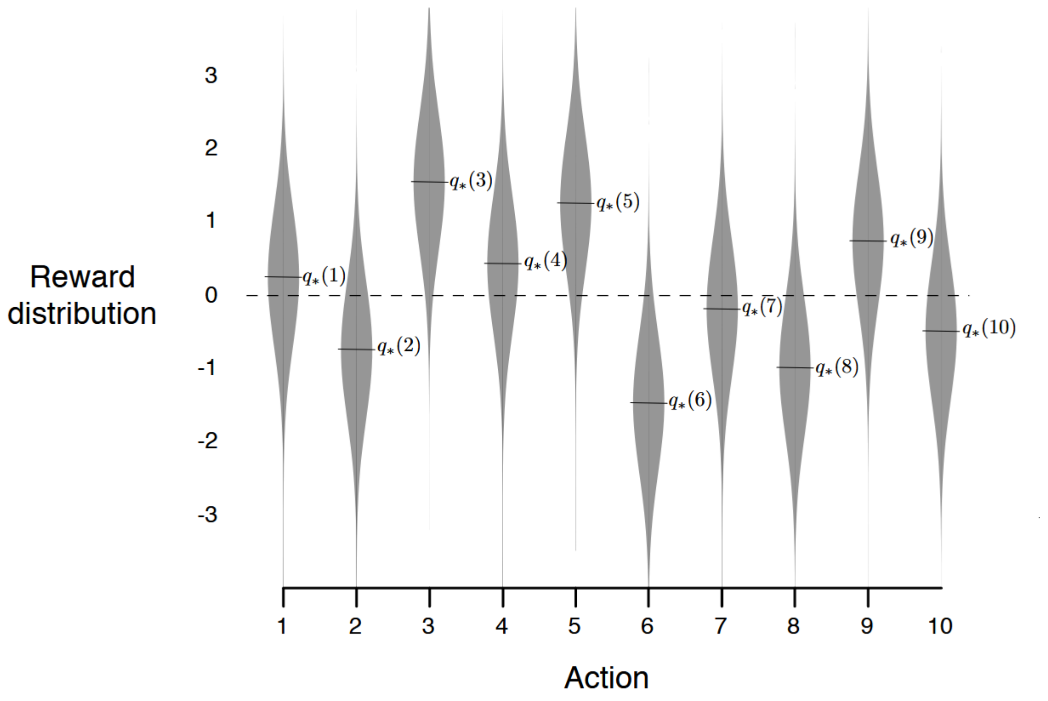

Figure 1.11: Distribution of rewards of ten levers [14]

In k -armed bandit problem, each action has expected - mean reward given, after said action is selected. It is called the value of action. Action selected at time t is denoted as A_t and reward is called R_t. Value of arbitrary action a (denoted as $q_*(a)$) is expected reward:

$$q_*(a) \doteq \mathbb{E}[R_t|A_t = a] \tag{1.9}$$

As we in practise lack true values of each action, we use estimates. Estimated value of action a at time t is denoted as $Q_t(a)$. Our goal is improve this estimate through multiple actions, to be s close s possible to real values $q_*(a)$.[14]

If we have at least two possible actions, then there always bust be at least one action with highest estimated value. These are **greedy** actions and selecting one of them is called **exploitation** of current knowledge. Picking any other action is called **exploration** instead.[14]

Exploitation is best choice for short term reward, as it always picks "best" action by current knowledge. Although it helps improve estimation of said best actions, it ignores others that can have higher true value, but lower estimation value. For that there is exploration, that improves estimation of all actions, but it brings smaller total reward. Because it is not possible to take both action during one step, we must find way to balance them, which is often dependant on multiple factors as precise values of the estimates, uncertainties, and the number of remaining steps. [14]

1.2.4 Finite Markov Decision Processes

Finite Markov Decision Processes, or finite MDPs, are a classical formalization of sequential decision making, where action influence not just immediate rewards, but also future possible states and their rewards. This means, in MDPs we need to deal with trade off between immediate reward and delayed reward. In tabular methods, we estimate value $q_*(a)$ for each action. In MDPs we estimate value $q_*(s,a)$ of each action a in state s.[14]

Agent and environment interacts at sequence of discrete time steps $t - 0, 1, 2, ...$ At each step, agent receives representation of environment's state $S_t \in \mathcal{S}$ and selection of actions $A_t \in \mathcal{A}$. At next time step, it receives reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$ and new state S_{t+1}. This gives a rise to sequence:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, ... \tag{1.10}$$

In MDP all sets of states, actions, and rewards have a finite number of elements. Random variable R_t and S_t have defined discrete probability distribution and are dependent only on preceding state and action S_{t-1} and A_{t-1}. So, for particular values of $s' \in \mathcal{S}$ and $r' \in \mathcal{R}$ at time t, their probability given values of action and state at $t - 1$ is:

$$p(s', r|s, a) \doteq PrS_t = s', R_t = r|S_{t-1}, A_{t-1} \tag{1.11}$$

for all $s', s \in \mathcal{S}, r \in \mathcal{R}, and a \in \mathcal{A}(s)$.

This is best viewed as restriction on the state, as state must include information about all past agent-environment interaction that is relevant for future. If state fulfill this requirement, we say it has **Markov property**.[14]

1.2.5 Policy gradient

Most of methods in reinforcement learning are not applicable to problems such as robotics or motor control, as they have problems with uncertain state information. Continuous states and actions in high dimensional spaces is also common source of problems. Most traditional reinforcement learning methods have no convergence guarantees.

Policy gradient methods differ significantly. Although, uncertainty in the state still might degrade the performance of the policy, continuous states and actions can be easily dealt with in same way s discrete ones. Policy gradient methods also have guaranteed convergence, at least to local optimum. [11]

They are based upon gradient descent, similar to supervised learning. In an example pong task, agent has two possible action: UP and DOWN. After feeding state into neural network, we gain log probabilities (-1.2, -0.36) for both actions. In supervised learning we have label of correct action, in this example it is UP, so we enter gradient of 1.0 on log probability of UP and run backprob to calculate gradient vector (Fig **??**).[6]

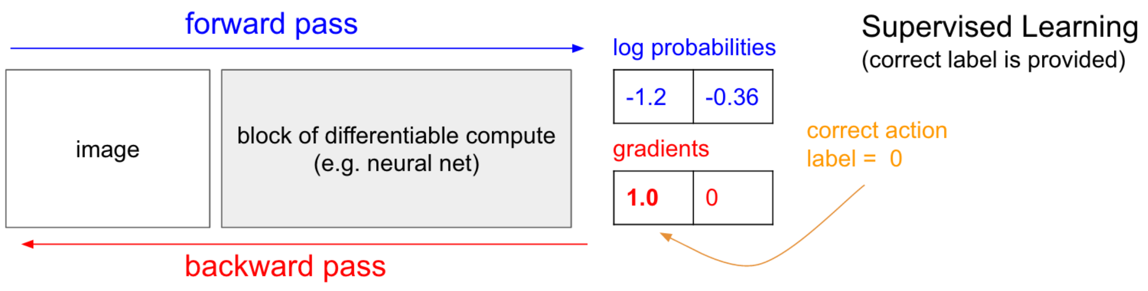

Figure 1.12: Training with Supervised Learning [6]

In reinforcement learning we lack correct labels, so we cant immediately calculate correct gradient. After our network calculated probabilities, we sample action, for our example action DOWN and execute it simulation. We could immediately fill in gradient of 1.0 for DOWN and run backprob, but we don't know yet, if this action is good or not. For this reason we wait, usually until end of episode. Now we can take reward (+1 for winning, -1 for losing) and enter it as gradient for all action tken during that episode.

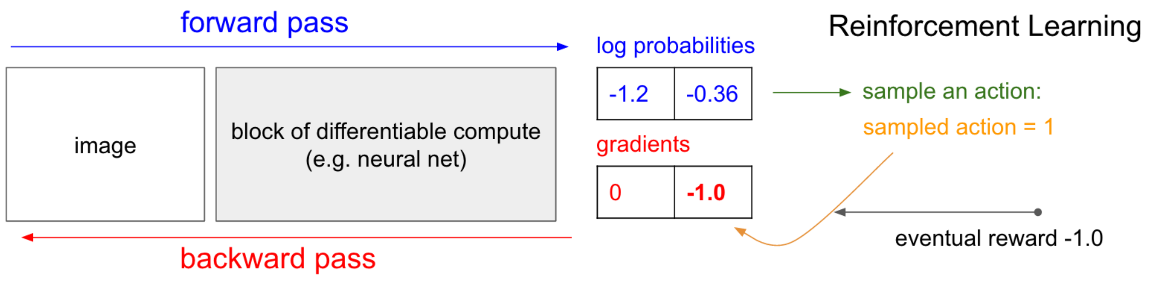

Figure 1.13: Training with Policy gradient [6]

1.2.6 Proximal Policy Optimization Algorithms

Proximal Policy Optimization or PPO in short is one of many approaches to the Policy gradient as TRPO or ACER. PPO was successful improvement over TRPO developed by openAI. It improves training stability by avoiding too large policy updates, as too-big step can result in "falling of cliff" - new policy is significantly worse then old one. This can significantly slow down training.[13]

In simple policy gradient, we use this policy objective function:

$$L^{PG}(\theta) = E_t[log\pi_\theta(a_t|s_t) * A_t] \tag{1.12}$$

However, with small step size, training was slow and with large, variability in training was too big.

PPO uses modified function:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{1.13}$$

Where $r_t(\theta)$ denotes probability ratio between old and new policy:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} sor_t(\theta_{old}) = 1 \tag{1.14}$$

Unclipped part of function then can be expressed as (CPI refers to conservative policy iterations):

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}] = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t] \tag{1.15}$$

Although, this could lead to successful learning it will have excessively large policy update. This is solved by second part of function, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$. This term clips probability ratio and removes incentive for moving r_t outside of interval $[1-\epsilon, 1+\epsilon]$. From this two, we pick minimum - lower or pessimistic bound.

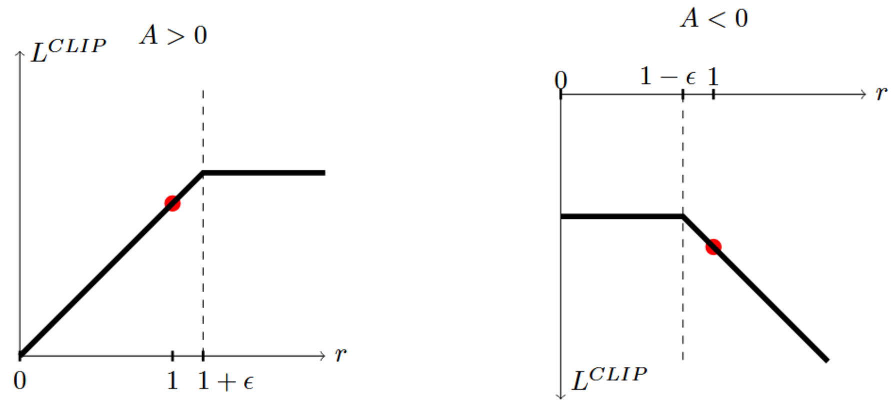

Figure 1.14: Plots showing surrogate function L^{clip} [13]

If we are using neural network architecture, that shares parameters between policy and value function, we must also add loss function combining policy surrogate and a value function error term. This is further augmented by entropy, resulting n following function [13]:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L^{CPI}(\theta)_t - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)] \tag{1.16}$$

Chapter 2

Proposed methods

Chapter 3

Software design

Chapter 4

Implementations

Chapter 5

Research

Goal of this thesis is to create model of feed-forward neural network for robot NICO. This model will allow prediction of perceptual effects of his actions. Effectivity of this model will be then evaluated. Training of model itself will be done through RL model, encouraging exploration of target object from multiple sides.

Chapter 6

Results

Chapter 7

Conclsuion

Bibliography

[1] Pragati Baheti. Train test validation split: How to & best practices [2023]. 2023.

[2] Anup Bhande. What is underfitting and overfitting in machine learning and how to deal with it. 2018.

[3] Arden Dertat. Applied deep learning - part 4: Convolutional neural networks. 2017.

[4] Igor Farkaš. Neural networks. page 192. Knižničné a edičné centrum FMFI UK, 2011.

[5] Dominique A. Heger. An introduction to artificial neural networks (ann)-methods , abstraction , and usage. 2015.

[6] Andrej Karpathy. Deep reinforcement learning: Pong from pixels. 2016.

[7] Bohdan Macukow. Neural networks – state of art, brief history, basic models and architecture. In Khalid Saeed and Władysław Homenda, editors, *Computer Information Systems and Industrial Management*, pages 3–14, Cham, 2016. Springer International Publishing.

[8] Jose García Moreno-Torres, José A. Saez, and Francisco Herrera. Study on the impact of partition-induced dataset shift on k-fold cross-validation. *IEEE Transactions on Neural Networks and Learning Systems*, 23(8):1304–1312, 2012.

[9] Hai Nguyen and Hung La. Review of deep reinforcement learning for robot manipulation. In *2019 Third IEEE International Conference on Robotic Computing (IRC)*, pages 590–595, 2019.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] J. Peters. Policy gradient methods. *Scholarpedia*, 5(11):3698, 2010. revision #137199.

[12] Jesus Ruiz-Santaquiteria, Alberto Velasco-Mata, Noelia Vallez, Oscar Deniz, and Gloria Bueno. Improving handgun detection through a combination of visual features and body pose-based data. *Pattern Recognition*, 136:109252, 2023.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[15] Youssef Zaky, Gaurav Paruthi, Bryan Tripp, and James Bergstra. Active perception and representation for robotic manipulation. *CoRR*, abs/2003.06734, 2020.