University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

Visual Language Navigation in Continuous Environments

Supervisor
Prof. Lamberto Ballan
University of Padova

Co-supervisor
Dott. Tommaso Campari
University of Padova

Master Candidate
Giacomo Barzon

Student ID
1236597

Academic Year
2021-2022

"I suppose therefore that all things I see are illusions; I believe that nothing has ever existed of everything my lying memory tells me. I think I have no senses. I believe that body, shape, extension, motion, location are functions. What is there then that can be taken as true? Perhaps only this one thing, that nothing at all is certain."
— Rene Descartes

Acknowledgments

First of all I want to deeply thank my family for supporting me for the entirety of my academic studies that took many years.

I want to thank then all friends that surrounded me during these years and gave me the possibility to experience brief moments of relief from the stress and the responsibilities of my studies.

I want also to give a thanks to all my fellow students that accompanied during the whole academic journey and with whom i shared every single exam, project and so on.

Finally a special thanks goes to the Professor Lamberto Ballan and my Co-Supervisor and Doctor Lamberto Ballan for guiding me through the entire thesis project and always beeing available whenever i had problems.

Padova, Settembre 2022 *Barzon Giacomo*

Abstract

From Philip K. Dick to Isaac Asimov novel writers from all ages have constantly dreamt about robots capable of assisting humans in their daily chores or even accomplishing tasks that would be completely unfeasible by a mere man. Nowadays the recent advancements in artificial intelligence are bringing day by day the dreams of those mans closer to reality.
In the recent years research on artificial intelligence has quickly shifted from Internet AI tasks, which completely revolve around datasets of images text or videos extracted from the internet, to tasks which fall under the Embodied AI field.
With the term Embodied AI we refer to all those tasks that involve a physical agent capable of interacting with the real world through concrete and tangible hardware.
More specifically Visual Language Navigation (VLN) is a sub-field of Embodied AI that tasks an agent to navigate through an environment, which potentially he could have no knowledge about, by following instructions given through natural language. Tasks belonging to the VLN field were originally modeled through navigation graphs which highly abstracted the environment by using nodes to represent locations and edges to indicate navigability between such locations.
This approach has the problem of abstracting too much the task by making it much more similar to teleportation than actual navigation.
The next step over VLN tasks was placing the agent inside continuous environment where he can freely navigate by executing low level actions like moving forward of x centimeters or turning left of y degrees. This kind of tasks take the name of Visual Language Navigation in Continuous Environments (VLN-CE) and are very challenging due to the high amount of input modalities that the agent needs to understand to achieve their goals. The aim of this thesis project is improving performances of the baseline VLN-CE model over RxR-Habitat dataset by proposing new solutions that exploit a better instruction encoding or the implementation of auxiliary tasks.

Contents

Listing of figures

Listing of tables

Listing of acronyms

AI ..... Artificial Intelligence
ADP ..... Action Distribution Prediction
ANS ..... Active Neural SLAM
BERT ..... Bidirectional Encoder Representations from Transformers
Bi-RNN ..... Bidirectional Recurrent Neural Network
BPE ..... Byte Pair Encoding
CMA ..... Cross Modal Attention
CNN ..... Convolutional Neural Network
CPC ..... Contrastive Predictive Coding
CPC|A ..... Contrastive Predictive Coding Action|Action
CSE ..... Cross Entropy
DTW ..... Dynamic Time Warping
GID ..... Generalized Inverse Dynamics
GloVe ..... Global Vector
GRU ..... Gated Recurrent Unit
LM ..... Language Modeling
LSTM ..... Long Short Term Memory
MLP ..... Multy Layer Perceptron
MSE ..... Mean Squared Error
nDTW ..... Normalized Dynamic Time Warping
NLP ..... Natural Language Processing
NN ..... Neural Network
NSP ..... Next Sequence Prediction
POMDP ..... Partially Observable Markov Decision Process
PBL ..... Prediction of Bootstrap Latents
PL ..... Path Length
R2R ..... Room to Room
ResNet ..... Residual Neural Network
RNN ..... Recurrent Neural Network
RoBERTa ..... Robust Bidirectional Encoder Representations from Transformers Approach
RxR ..... Room Across Room

Seq2Seq ..... Sequence to Sequence
SLAM ..... Simultaneous Localization And Mapping
SPL ..... Success Weighted by Path Length
SR ..... Success Rate
ViLBERT ..... Visiolinguistic BERT
VLN ..... Visual Language Navigation
VLN-CE ..... Visual Language in Continuous Environments

1 Introduction

Robots capable of understanding human language and reacting accordingly have been fantasized by many sci-fi novel writers in the past. Nowadays, thanks to the efforts of many researchers and the many technology advancements that have been made in the previous decades, these fantasies are becoming closer to reality day by day.

Embodied AI is a field of artificial intelligence in which agents are no longer limited to virtual worlds and internet datasets (like collections of labeled images or videos) but are physically present in realistic environments and learn by exploring and interacting with such environments, similarly to human beings.
One of the key advancements in this field has been the development of simulators that allow agents to train inside realistic virtual environments reconstructed through photogrammetry that make up for the limitations of the physical world.
Embodied AI tasks can vary from tasks in which the agent is simply asked to explore the environment while constructing a virtual map of such, to more complex tasks in which the agent has to simultaneously explore the environment while having to reach a goal location in the minimum amount of steps possible, or even tasks in which he has to explore the environment and acquire enough information to be able to answer questions asked about the environment.

More specifically the thesis work falls under the Visual Language Navigation (VLN) task. VLN tasks ask the agent to navigate inside the environment to a goal location by following natural language instructions.
VLN tasks where usually defined over navigation graphs, in which each node represents a reachable area in the environment and the edges between nodes indicate navigability. This representation method ab-

stracted too much the problem by making it more similar to a visually guided graph search than actual navigation. The authors of [18] introduce Visual Language Navigation in Continuos Environments (VLN-CE). VLN-CE tasks lift all the assumptions made by previous VLN models by putting the agent inside a continuous environment navigable by executing low level actions like: move forward, turn left, turn right and stop.

RxR-Habitat [15] is a competition that occurs yearly, in which the participants are tasked to train a VLN-CE agent with the best possible performances. A public leaderboard is provided to offer participants the possibility to track their progress. The RxR-Habitat provides also a starting code [16] that was used as a baseline for this project.
The objective of this thesis project consists in making modifications to the baseline model that improve performances as much as possible. More specifically we propose two main modifications to the baseline model:

Instructions Encoding In Visual Language Navigation tasks Natural Language Instructions act as a bridge between Visual and Textual perceptions. It becomes of key importance then to provide in input to the agent's model a representation of the instructions which encapsulates as much information as possible inside it. Our first contribution consists in integrating into the models encodings produced by RoBERTa, a state of the art model which has been proven to reach performances exceeding the ones of the BERT model in many situations.

Auxiliary Tasks The continuous environments in which our agent operates are complex and rich. Good Performances can be obtained by maximizing cumulative reward. However integrating into the model auxiliary losses which force the model to maximize other secondary pseudo-rewards can bringe notable improvements in many situations. Our second contribution consists in integrating some of the auxiliary losses proposed by [27] into the baseline VLN-CE model.

1.1 Text Organization

Chapter 1 - Introduction presents a brief introduction of the thesis project

Chapter 2 - Background Introduces all the background knowledge that acts as a foundation for all the concepts presented in the following chapters of the thesis.

Chapter 3 - Related Works presents a bird-eye view of the Embodied AI field by introducing the mains tasks and concepts, including Visual Language Navigation (VLN), the area of work where falls the thesis work.

Chapter 4 - Datasets Gives a brief introduction to the various datasets used during the thesis work along with the Habitat environment used to simulate the various experiments.

Chapter 5 - Methods after giving an initial introduction of the work made by Krantz et al. [18], which was used as a baseline work, the chapter will introduce in detail the proposed contributions.

Chapter 6 - Results Introduces the standard metrics used for the evaluation of the experiments and exposes the obtained results.

Chapter 7 - Conclusions Draws conclusions over the project and proposes potential future work that can be done on it.

2
Background

Inside the background chapter I'm going to present all the basic knowledge needed to understand the work done in the following chapters. These concepts are going going to be picked up many times during the entire document so, to avoid having to explain them later, they are all explained in this section.
All the information provided regarding neural networks was extracted from [1], on the other hand [2] and [9] were used as sources regarding reinforcement learning.

2.1 Neural Networks

Neural Networks are one of the most used computational models in the field of machine learning due to their incredible representational power.
Every neural network usually follows the structure that can be seen in Figure 2.1. The network is composed by an input layer, an output layer and a number of hidden layers. Each layer is fully connected with the following one by a set of weights.
The output of the n^{th} layer of a neural network can be represented mathematically through the following formula:

$$f(\mathbf{x}; \mathbf{w}_n, b_n) = f(\mathbf{x}; \mathbf{w}_{n-1}, b_{n-1})^T \mathbf{w}_n + b_n \tag{2.1}$$

Where:

- $\mathbf{x}$: Is the vector given in input to the network
- $\mathbf{w}_k$: Is the vector of weights of the k^{th} layer of the network
- $\mathbf{b}_k$: Is a constant scalar value also called as bias

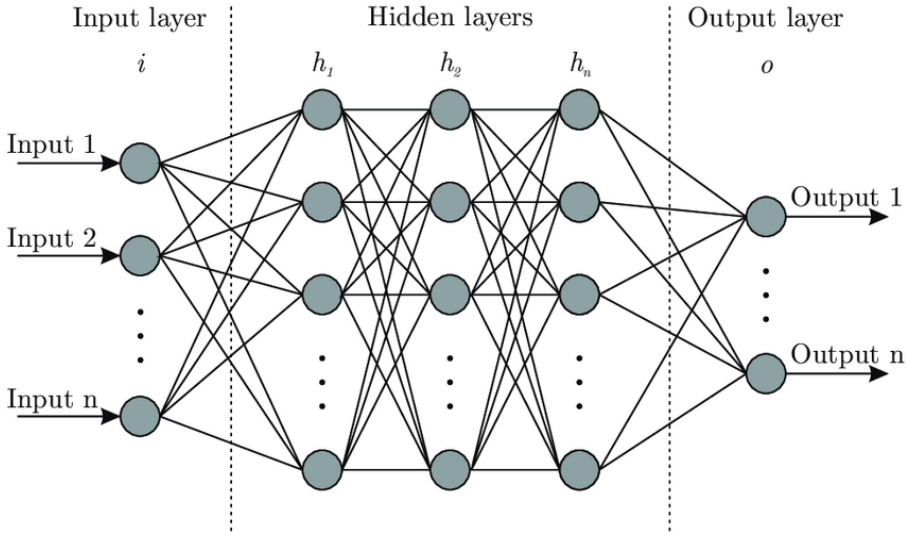


Figure 2.1: Graphical representation of the structure of a Neural Network

The problem of the model above is that it has a really low representational power, in fact it can represent only linear functions. Most of the functions that we want to learn are really complex and can't be closely approximated by using linear functions.
To do that non-linearity has to be introduced into the model. This is usually done by applying a non-linear function to the vector produced in output by the layer.
So the output of the n^{th} layer then becomes:

$$f(\mathbf{x}; \mathbf{w}_n, b_n) = g_i(f(\mathbf{x}; \mathbf{w}_{n-1}, b_{n-1})^T \mathbf{w}_n + b_n) \tag{2.2}$$

Where g^k is the chosen non-linear function.

2.1.1 Training

The training of a neural network has the objective of learning the weight matrix $\mathbf{W} = [\mathbf{w}_1^T ... \mathbf{w}_N^T]$ that approximates as closely as possible some function f^*.
So given a ground truth of values $\mathbf{x} \in X$ for which it's target value $f^*(\mathbf{x})$ is known, the objective of our training is minimizing a function that estimates the difference between f^* and the output of the neural network f.
Such functions are usually called as loss functions. One of the most important loss functions is the Mean

Squared Error(MSE)

$$MSE = \frac{1}{N}\sum_{i=1}^{N} ||f(\mathbf{x}_i) - f^*(\mathbf{x}_i)||^2 \tag{2.3}$$

Mean Squared Error is one of the most intuitive definitions of a loss that can be thought of since it tries to minimize the actual difference between the target values of f^* and the actual values produced by our network. However this isn't always the best choice, other losses can work better in specific contexts like for example Cross Entropy when the output values of our network are probabilities.

$$CSE = \sum_{i=1}^{N} log\, p(f^*(\mathbf{x}_i)|\mathbf{x}_i, \mathbf{W}) \tag{2.4}$$

Calculating the best possible set of weights $\mathbf{W}$ that minimizes the chosen loss function is usually a problem computationally to hard even for relatively small networks. So we have to find a solution that is good enough.

The training of a neural network is usually composed by mainly two phases:

- The **forward propagation** phase were the examples of our ground truth are given in input to our network. The input propagates trough the network, the values of each layer are calculated from the input layer to the output one.
- In the **back propagation** phase the information propagates backward, from the last layer of the network to the first one by calculating the gradient of our loss function with respect to the weights of the current layer.

What is done during back propagation is called gradient descent, since computing the best set of weights that minimizes our loss function isn't possible due to it's high computational complexity a valid solution that approximates the best possible solution is found by iteratively moving through our function by making small steps towards the minimum which is pointed by the negation of the gradient of the function.

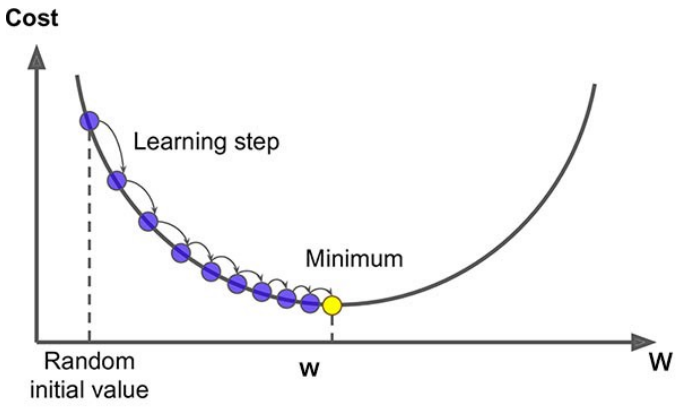


Figure 2.2: Graphical representation of the gradient descent procedure in a two dimensional space. The minimum is found by iteratively making small steps towards by calculating the gradient of the function.

2.1.2 Autoencoders

Autoencoders are a special kind of neural network trained to reconstruct as closely as possible the same input they received. Autoencoders are often called also as Encoder-Decoder networks due to the structure of their architecture consisting of two parts:

- An encoder $\mathbf{h} = f(\mathbf{x})$ which produces a feature representation of it's input $\mathbf{x}$
- And a decoder $\mathbf{r} = g(\mathbf{h})$ which attempts to reconstruct the input $\mathbf{x}$ given it's feature representation $\mathbf{h}$

Autoencoders are trained with the same techniques used in regular neural networks.

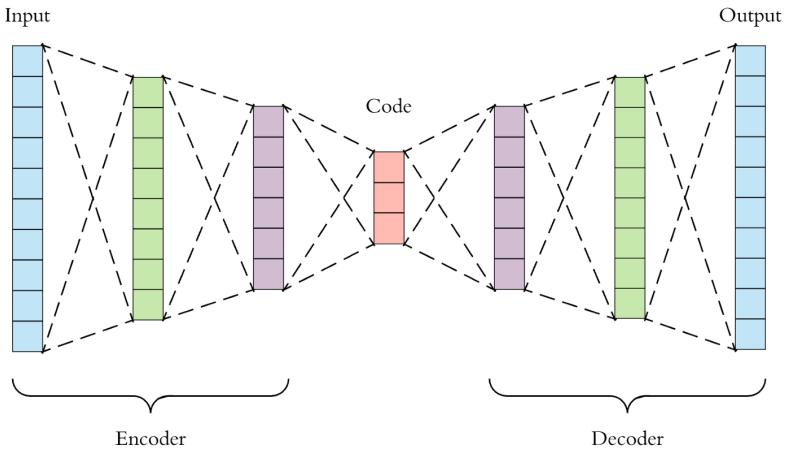


Figure 2.3: Graphical representation of an Encoder-Decoder Neural Network. The image shows the inherently symmetrical structure of such kind of networks

2.2 Convolutional Neural Networks

Convolutional neural networks(CNNs) are a specific category of neural networks that have been proven very effective in tasks that involve images. Generally speaking a convolution is an operation which involves two functions, an input function and a kernel function.

$$(x * w)(t) = \int x(a)w(t-a)da \tag{2.5}$$

However since images are discrete and not continuos we can use the discrete version of the convolution operation:

$$(x * w)(t) = \sum_{a=-\inf}^{\inf} x(a)w(t-a) \tag{2.6}$$

Which if applied over more than one axis becomes:

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.7}$$

Where I is our two-dimensional image and K our two-dimensional kernel.

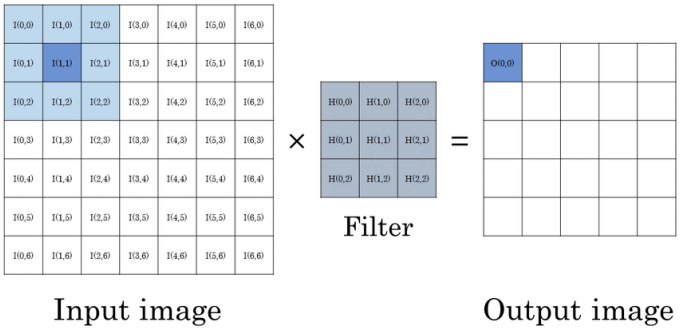


Figure 2.4: Convolution of a Kernel on an Image

In a convolutional layer of a neural network the Kernel is a set of weights that are learned through back-propagation.
The output of the convolutional layer is usually put through a non-linear activation function, like in a normal fully connected layer, and a pooling function. *A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs*[1]. One of the most commonly used pooling operations is max pooling which produces in output the maximum value within a rectangular neighborhood. Other common pooling functions are the average or the L^2 norm of a rectangular neighborhood.
The real strength of the Pooling operations is that they make the feature representation learned by our model invariant to small translations. Being invariant to small translations is a vital property in tasks that involve images since the objects represented inside them can bee seen from many different positions and many different angles.

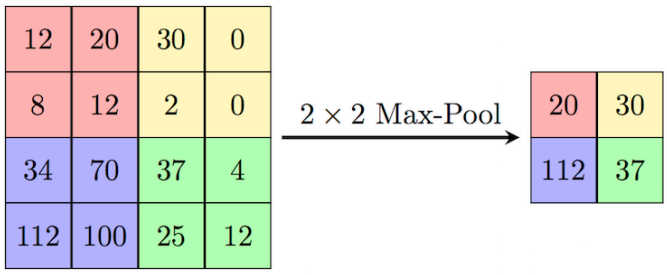


Figure 2.5: Max Pooling

A convolutional layer followed by an activation function and a pooling operation is often called as a convolutional block.
Figure 2.6 shows the typical structure of a convolutional neural network which is made of a sequence of several convolutional blocks followed by a classical fully-connected network that takes in input the flattened output of the last convolutional block and produces the output needed for the task.

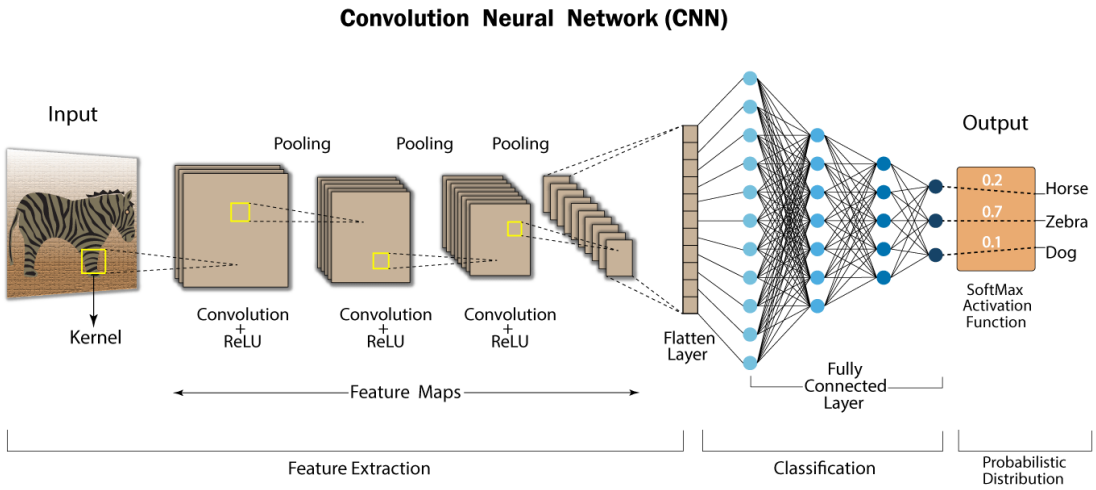


Figure 2.6: Graphical representation of a Convolutional Neural Network. Each convolutional layer learns a different filter which becomes finer and finer the more deep it is in the architecture.

Convolutional Neural Networks have mainly three advantages over regular Neural Networks: sparse interactions, parameter sharing and equivariant representations.

Sparse interactions the kernel function is usually much smaller than the input hence we can store fewer parameters by not having to fully connect each layer with the following one.

Parameter sharing The same filter can be applied over every portion of the input. In classical neural networks each weight is used exactly once when computing the output of a layer. In a CNN it's used at almost every position of the input.

Equivariance to Translation A function f is equivariant to some other function g if $f(g(x)) = g(f(x))$ which in the context of convolutional neural networks means that applying a convolution to a translated image produces the same result of translating a convoluted image.

2.3 Recurrent Neural Networks

Recurrent Neural Networks (often abbreviated as RNNs) are a kind of neural networks developed to handle sequences of vectors $\mathbf{x}^{(1)}....\mathbf{x}^{(t)}$.
In a RNN the output of an hidden layer is now a function that not only depends from the output of the previous layer but also by the output of the same hidden state at the previous time step in the sequence.

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; "\theta) \tag{2.8}$$

Figure 2.7 shows the structure of a basic Recurrent Neural Network with an input layer, an output layer and a single hidden layer which can be expressed mathematically like this:

$$\begin{aligned} \mathbf{a}^{(t)} &= \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} + b, \\ \mathbf{h}^{(t)} &= g(\mathbf{a}^{(t)}), \\ \mathbf{h}^{(t)} &= \mathbf{V}\mathbf{h}^{(t)} + c \end{aligned} \tag{2.9}$$

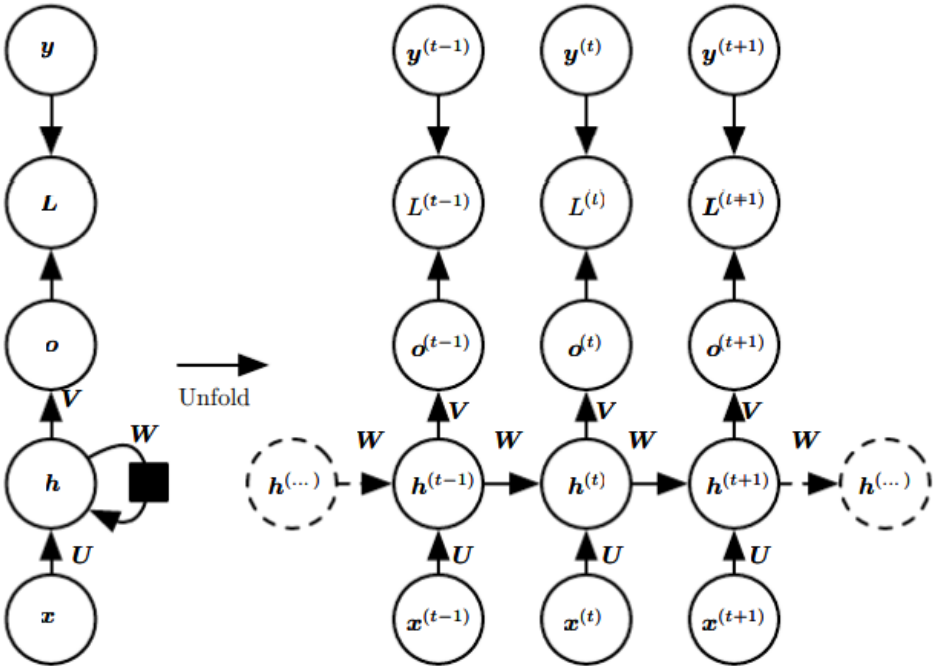


Figure 2.7: *(Left)* A Recurrent Neural Network with a single recurrent connection. The black square indicates a delay of a single time step. *(Right)* The same Recurrent Neural Network but unfolded trough time where each node refers to a specific time instance

A Recurrent Neural Network not only has the weight matrices **U** and **V** that respectively connect the input layer to the hidden layer and the hidden layer to the output layer but also a weight matrix **W** which connects the hidden layer to the hidden layer in the previous time step
The rightmost part of Figure 2.7 shows how a neural network unfolds trough time. One of the key concepts around which RNNs are based is parameter sharing. Storing and training a different weight matrix for each time step is computationally unfeasible so the same parameters are re-used for each time step.

2.3.1 Bidirectional RNN

In a regular RNN each state at timestep t captures only information from the past items $\mathbf{x}^{(1)}....\mathbf{x}^{(t-1)}$ in the sequence up to timestep $t-1$. A Bidirectional Recurrent Neural Network (Bi-RNN) combines an RNN that moves forward through time beginning from the start of the sequence with another RNN that moves backwards through time beginning from the end of the sequence. This allows the output units to produce representations that depend on both past and future inputs.

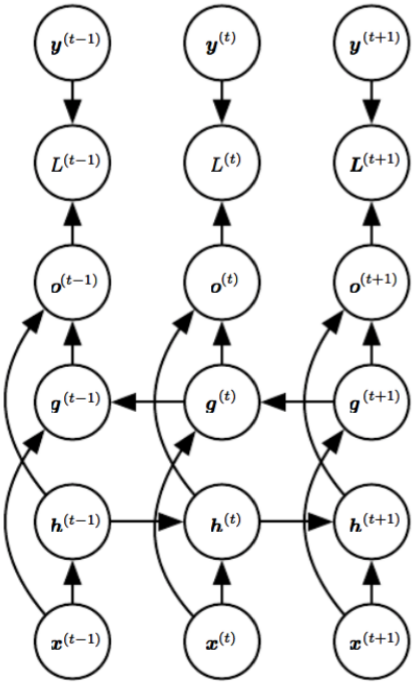


Figure 2.8: Computation of a typical Bidirectional Recurrent Neural Network. The **h** recurrence propagates information forward in time while the **g** recurrence propagates information backwards in time.

2.4 Long term dependencies

One of the biggest problems of Recurrent Neural Networks is that gradients propagated over really long sequences tend to either vanish or explode due to the repeated application of the same parameters. The values of the gradient become so small (or big) that learning becomes unstable.
The easiest way to cope with this problem is gradient clipping, which consists in setting a maximum norm value that the gradient can assume. If the gradient has a norm bigger than that value we divide the gradient by it's norm and multiply by the maximum norm value. The disadvantage is losing the alignment with the true gradient despite still pointing to a descent direction.

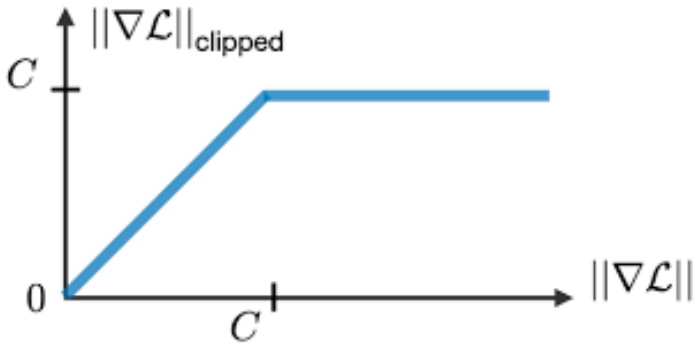


Figure 2.9: Gradient Clipping

2.4.1 Long Short-Term Memory

Another very effective technique is substituting the classical neural network units that consist of only affine transformations with special kinds of units designed to solve this exact problem.
LSTM units (Long Short Term Memory) are one of the most important ones. Each LSTM unit has the same inputs and outputs of a classical neural network's unit but introduces a system of internal gates that allows to control the flow of information.
The most important component of an LSTM is the state unit which acts as a memory by keeping track of all the information encountered during the sequence. The state unit is controlled by a forget gate that decides how much of past information has to be forgotten in favour of new information. A forget gate works similarly to a leaky unit which can be described with the following formula:

$$\mu^{(t)} \leftarrow \alpha\mu^{(t-1)} + (1-\alpha)v^{(t)} \tag{2.10}$$

Where:

- $\mu^{(t)}$ is the value of our state at timestep t

- $v^{(t)}$ is the new input that the unit receives at timestep t
- the parameter α controls how much past information is forgotten in favour of new information

An LSTM also has an input and an output gate which works similarly to the forget unit described above by respectively controlling how much information enters and exits the unit.

2.4.2 Gated Recurrent Units

One of the biggest drawbacks of LSTMs is their incredible complexity. Every gate in fact increases the number of parameters that must be learned through backpropagation. Are all of the components of an LSTM really neaded or can some of them be discarded to semplify the structure?
GRUs (Gated Recurrent Units) are another kind of special units that work on this concept by keeping only the most essential components of an LSTM.
More specifically in a GRU a single gating unit simultaneously controls both the input gate and forget gate.

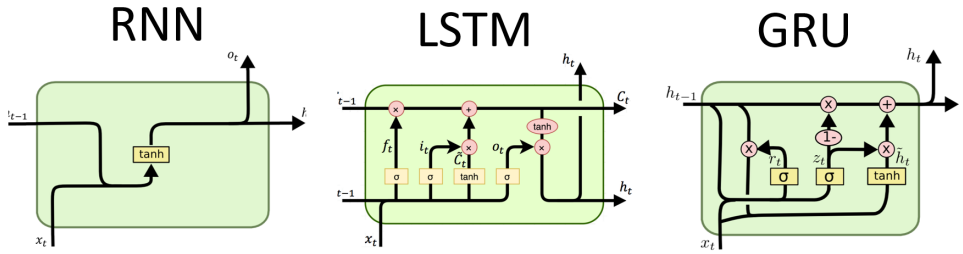


Figure 2.10: (Left) Graphical representation of a Classical RNN's node. (Center) Graphical representation of a LSTM node. (Right) Graphical representation of a GRU node.

2.5 Reinforcement Learning

Reinforcement Learning is a class of machine learning algorithms that are especially useful in situations where we lack of labeled examples. Reinforcement Learning is based around the key concept of giving the agent positive feedback (or rewards) when he obtains a positive outcome and instead giving neutral or even negative rewards when he doesn't. For example in a navigation task a positive reward is given if the agent manages to reach the it's goal and a negative one otherwise.
Usually every reinforcement learning algorithm follows the following steps:

- Observing the environment the agent is in
- Deciding the best possible action we can take in the current state to make progress toward our goal based on a policy (Like for example a Neural Network)

- Executing the chosen action
- Updating our policy based on the reward obtained.
- Iterate until we have found an optimal strategy

The objective of reinforcement learning tasks is finding an optimal policy (or nearly optimal), a policy which maximizes the expected total reward.

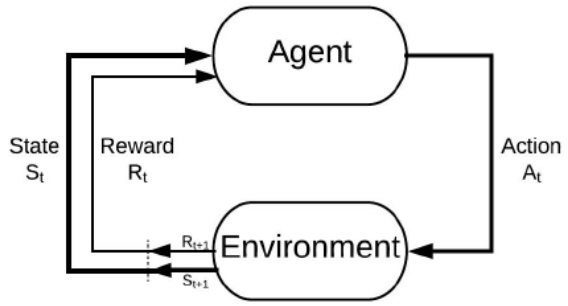


Figure 2.11: Graphical representation of the reinforcement learning process

3
Related Works

Embodied AI is a term used to encapsulate all tasks related to the field of artificial intelligence that involve a physical agent which can interact with the real world through concrete and tangible hardware.
For many years researchers from all around the world have been focused on developing and studying *"Internet AI"*[4] tasks. Tasks that completely revolve around datasets of images, text or videos extracted from the internet. Now thanks to the recent advances in computer graphics, robotics and artificial intelligence in general the focus has shifted to Embodied AI.
It's believed that true intelligence can't arise by solely passive experiences in Internet AI tasks, but it must emerge trough active perception, movement and interaction. One of the key advancements in the field of Embodied AI tasks has been the development of simulators that allow agents to train and evaluate inside a virtual environment before beeing deployed in the real world. Well talk about these simulators more in depth in Chapter 2 where the simulator used for the work of this thesis is going to be presented along with the dataset it's based on.
We can group most of research work done in the Embodied AI field in three main tasks: Visual Exploration, Visual Navigation and Embodied Question and Answering. These tasks have increasing complexity, and each category acts as a foundation for the next one.

The first weeks of the project where spent studying the Embodied AI field and it's related tasks. Although this thesis work falls into the category of Visual Navigation this chapter is going to give a brief introduction to all three class categories in order to provide a complete overview of the Embodied AI field. Special importance is going to be given to the VLN tasks since it's the field in which the project the operates.
Two surveys were used as main source for building this chapter one for Embodied AI [4] in general and another for Visual Language Navigation [33] specifically.

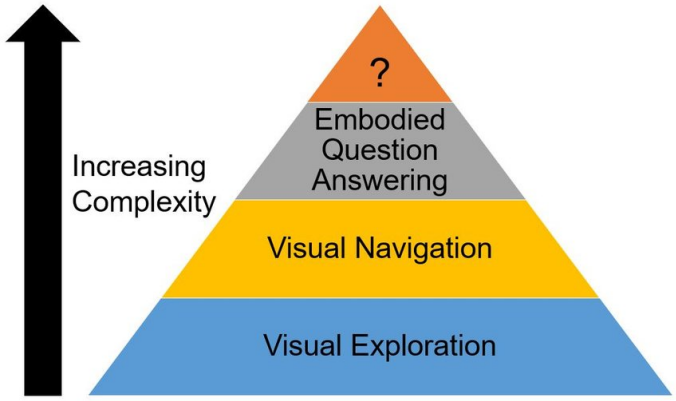


Figure 3.1: Hierarchical structure of Embodied AI tasks. The higher in the pyramid the higher the complexity.

3.1 Visual Exploration

The objective of visual exploration tasks is gathering as much information as possible about the environment by executing the least amount of movements. The information gathered is used to construct a model (or map) of the environment that can be used by the agent for more complex tasks like Visual Navigation.

Visual Exploration can be done either before or simultaneously while navigating the environment. In the first case the agent is allowed to execute a limited amount of movements to build up a map of the environment before executing the navigation task, in the second case the agent has to navigate an unseen environment and construct the map while executing the navigation task.

3.1.1 Simultaneous Localization And Mapping

The SLAM problem consists in placing an agent in a completely unknown environment at an unknown position and simultaneously constructing a map of the environment while at same time determining the current position of the agent in such environment.

To formulate the SLAM problem we define the following vectors:

- $\mathbf{x}_k$: The state vector. Describes the location and orientation of the agent inside the environment at time step k.
- $\mathbf{u}_k$: The control vector. Describes the movement that the agent has to make at time step $k-1$ to drive him from state $\mathbf{x}_{k-1}$ to state $\mathbf{x}_k$.

- $\mathbf{m}_i$: A vector which describes the location in the environment of the i^{th} landmark. Landmarks are assumed to be invariant with respect to the current time step.
- $\mathbf{z}_{ik}$: An observation of the i^{th} landmark taken from the the agent's current location at time step k. Often written as $\mathbf{z}_k$ by omitting the the landmark's index if it is not necessary to specify it.

Also we define the following sets:

- $\mathbf{X}_{0:k} = \{\mathbf{x}_0, ..., \mathbf{x}_k\} = \{\mathbf{X}_{0:k-1}, \mathbf{x}_k\}$: The set of all vehicle locations up to time step k
- $\mathbf{U}_{0:k} = \{\mathbf{u}_0, ..., \mathbf{u}_k\} = \{\mathbf{U}_{0:k-1}, \mathbf{U}_k\}$: The set of all vehicle movements up to time step k
- $\mathbf{m} = \{\mathbf{m}_1, ..., \mathbf{m}_n\}$: The set of all landmarks
- $\mathbf{Z}_{0:k} = \{\mathbf{z}_0, ..., \mathbf{z}_k\} = \{\mathbf{Z}_{0:k-1}, \mathbf{z}_k\}$: The set of all landmark observations up to time step k

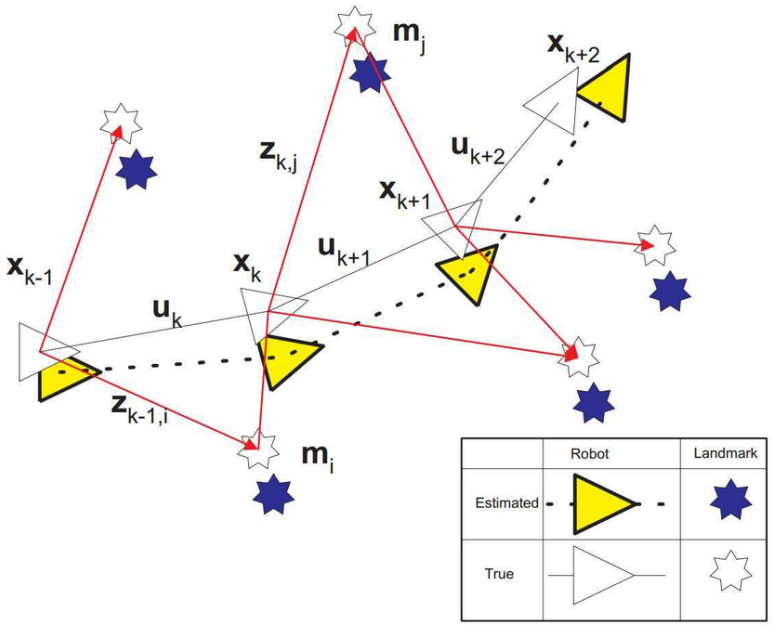


Figure 3.2: Graphical representation of the SLAM Problem

Classical probabilistic SLAM requires the following probability distribution to be computed for every time step k.

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{3.1}$$

This probability distribution describes the joint probability of the current vehicle state and all landmark locations given the set of all recorded observations, vehicle movements and the starting location of the agent.
The above probability distribution is usually calculated iteratively by starting with an estimate of the distribution at the previous time step $P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0)$. Then a control vector $\mathbf{u}_k$ is applied to the current state $\mathbf{x}_{k-1}$ of the agent and the new observations $\mathbf{z}_k$ at the new state $\mathbf{x}_k$ are calculated by

using the Bayes Theorem.
To estimate the state vector $\mathbf{x}_k$ and the landmark observations $\mathbf{z}_k$ we need to define an observation model and a motion model.
An observation model calculates the probability of observing a landmark at timestep k given the current agent state $\mathbf{x}_k$ and the set of landmark locations. It can be mathematically defined as below:

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \tag{3.2}$$

On the other hand the motion model calculates the probability of ending in a new state $\mathbf{x}_k$ given the previous state $\mathbf{x}_{k-1}$ and the control vector $\mathbf{u}_k$. It can be mathematically defined as:

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{3.3}$$

3.1.2 Active Neural SLAM

SLAM is a purely geometrical approach proposed by classical robotics. It is really well studied and an incredible amount of research work has been done on it, however it has plenty of room for improvements. A purely geometrical approach is highly susceptible to sensor noise which can be handled much more consistently by learning based approaches. Like in classical SLAM the agent's location in the environment is represented by a vector (x, y, o) where x and y are the coordinates of the agent in the map measured in meters and o is the orientation of the agent measured in radians counter-clockwise from the x axis. At each time-step t the model mantains in memory the agent's location $\mathbf{x}_t$ and a spatial map $\mathbf{m}_t$. The map is a $2 \times M \times M$ matrix where M is a constant defining the size of the map. Each cell in such matrix identifies an area of 0.25 meters in the map. For each cell in the map we keep track of two variables:

- the probability that the cell contains an obstacle
- the probability that the cell was explored. A cell is considered as explored when it's known if it contains an obstacle or not.

The map is initialized with all values set to zero and with the agent starting in the center facing east $\mathbf{x}_0 = (M/2, M/2, 0)$. The Active Neural SLAM model is composed of three main components a Neural SLAM module, a Global policy and a Local policy.

Neural SLAM Module The Neural SLAM Module predicts the map the environment and the agent's location from current observations and previous predictions. More precisely the Neural SLAM module takes in input the following parameters:

- the current RGB observation $\mathbf{s}_t$
- the agent pose $\mathbf{x}'_{t-1}$ obtained through the sensor readings at the previous timestep
- the agent pose $\mathbf{x}'_t$ obtained through the sensor readings at the current timestep

- the estimate of the map at the previous timestep $\mathbf{m}_{t-1}$
- the estimate of the agent position at the previous timestep $\mathbf{x}_{t-1}$

Then the Neural SLAM Module can be defined as a function that takes in input the parameters defined above and returns the estimates $\mathbf{m}_t, \mathbf{x}_t = f(\mathbf{s}_t, \mathbf{x}'_{t-1}, \mathbf{x}'_t, \mathbf{m}_{t-1}, \mathbf{x}_{t-1}|\theta)$ where θ are the learnable parameters of the Neural SLAM module.

Global Policy The global policy uses the estimated map and agent location to produce a long term goal g_t^l for the agent. A matrix $\mathbf{h}_t \in \{0,1\}^{4\times M\times M}$ is constructed starting from the map $\mathbf{m}_t$ and the agent location $\mathbf{x}_t$. The first two channels of $\mathbf{h}_t$ are identical to the two channels of the map $\mathbf{m}_t$. The third channel is set to 1 for the cell that contains the agent's current position and to 0 for all the other cells. The fourth channel is set to 1 if the cell has been visited by the agent and 0 otherwise. A matrix of size $4 \times G \times G$ is subsampled from $\mathbf{h}_t$ around the agent's location. Then a max pooling of such matrix is performed. The subsampled matrix is stacked along with the pooled matrix forming a new matrix of size $8 \times G \times G$. The latter matrix is given in input to a Convolutional Neural Network to predict the long term goal g_t^l in the $G \times G$ space. We can define the Global Policy as a function $\pi_G(\mathbf{h}_t|\theta_G)$ where θ_G are the learnable parameters of the Global Policy

Local Policy The shortest path from the estimated agent current position to the long term goal $\mathbf{g}_t^l$ is calculated by using the fast marching method on the estimated map $\mathbf{m}_t$. Then a short term goal $\mathbf{g}_t$ is computed by identifing the furthest point along the identified path within 0.25 meters from the agent. The calculated short term goal $\mathbf{g}_t$ is given in input to a Recurrent Neural Network along with current RGB observation to produce a navigational action a_t We can then define the Local Policy as a function $\pi_L(\mathbf{s}_t, \mathbf{g}_t|\theta_L)$ Where θ_L are the learnable parameters of the Local Policy

3.2 Visual Navigation

The objective of visual navigation tasks is to navigate the agent from a starting position to a goal as efficiently as possible.
Classical navigation approaches the problem by using a mixture of many different hand-engineered subcomponents like localization, mapping, path-planning and locomotion. Visual Navigation on the other hand aims to learn all of these paradigms from data. While classical navigation still outperforms visual navigation the latter has been proven much more robust to sensor noise then classical navigation, moreover the AI approach avoids having to hand engineer case specific problems.

3.2.1 Goal Types

Many different kinds of goals can be used for developing visual navigation taks, among those some of the most important ones are: Point Navigation, Object Navigation, Image Navigation and Area Navigation.

3.2.1.1 Point Navigation

In point navigation an agent is asked to navigate from a starting position $\mathbf{x}_0 = \{0, 0, 0\}$ as closely as possible to a specific point in the environment $\mathbf{x}_* = \{x, y, z\}$.
The task is evaluated as successfully completed if the agent managed to reach a location in the environment which isn't further away from the goal location then a specific fixed distance.
The agent usually is equipped with a GPS and a compass that allows him to access to his current location and the location of the goal. The compass can be configured to be either static or dynamic. A static compass gives to the agent the location of the goal only at the start of the episode, a dynamic one on the other hand does it at every time step.
Recent developments of Point Navigation tasks have moved to more complex tasks where the agent posses no GPS or compass.

3.2.1.2 Object Navigation

In object navigation tasks the agent is initialized at a random position in the environment and asked to navigate to the first instance of a labeled object inside such environment.
Object Navigation is usually much more complex than Point Navigation since it demands to the agent a much wider set of skills than the latter.
Among those we can mention: long-term episodic memory to keep track of explored and unexplored areas, object detection to correctly recognize and classify objects's shapes, semantic understanding to allow the agent to understand where specific categories of objects are more likely to be found in the environment.

3.3 Visual Language Navigation

In Visual Language Navigation (VLN) tasks, the agent is asked to navigate in the environment to a specific goal location by following natural language instructions.
"A natural language or ordinary language is any language that has evolved naturally in humans through use and repetition without conscious planning or premeditation"[7].
Natural languages lack of any kind of structure and hence are inherently difficult to comprehend by computers. Natural Language Processing (NLP) is a completely different field of Artificial Intelligence that aims at enabling computers to understand natural language the same way humans do.

So a VLN agent not only needs to possess all the skills described for general Visual Navigation before but also needs to posses skills related to the NLP field.

Current VLN tasks can be mainly categorized based on two axis: Comunication Complexity and Task Objective.

Comunication Complexity Defines the level at which the agent comunicates with his oracle. Can be classified on three levels:

- At the first level the agent is asked to only understand an initial instruction before the beginning of the navigation.
- At the second level the agent is capable of asking guidance to the oracle by sending him a signal of help whenever it's necessary.
- At the third and last level the agent is capable of asking questions to the oracle in the form of natural language while navigating the environment

Task Objective How the agent manages to reach it's goal given it's initial instructions.

- **Fine Grained Navigation**: The agent receives a detailed step by step description of the path he needs to follow to reach the goal.
- **Coarse-grained Navigation**: The agent needs to reach a much more distant goal by following a coarse path description. In this task the agent is asked to reason more about the environment and possibly ask help to the oracle.
- **Object Interaction**: The agent is tasked to not only navigate through the environment but also to interact with it, for example by opening doors to reach otherwise unreachable locations.

Visual Language Navigation tasks are very complex and possess many different challenging aspects that need to be tackled. Such problems and the related methods used to solve them can be mainly categorized into four different areas: Representation Learning, Action Strategy Learning, Data Scarcity, Seen to Unseen environments generalization.

Representation Learning Agents needs to understand and align informations coming from many different sources and modalities. This is usually done through Representation Learning techniques.
Pre-trained encoder models like BERT can help the agent to better understand information coming from single modalities. This models can also be trained further more over the task data to improve the encoding capabilities. Models like ViLBERT [34] construct joint representations that combine into a single representation both the visual and text features.
Graph structures that explicitly model connections between the environment and the received instructions provide semantic knowledge that can aid the agent during navigation.
Navigating inside the environment accumulates a lot information which can be problematic to handle

especially over paths made of many steps. LSTM layers are frequently used to solve this problem and allow the agent to better leverage features encountered at the beginning of the path.
Auxiliary Tasks can help the agent to better adapt to the environment through additional loss functions.

Action Strategy Learning Action Learning methods focus on providing the agent the ability to pick at each step the best possible action. VLN tasks are a sequential decision making problem and can thus be modeled like a Markov Process. Reinforcement Learning methods are often proposed to learn the action policy. The main problem with RL methods is that the agent receives it's reward only at the end of the episode, and since the paths are usually very long is hard to identify which actions to penalize. More advanced models are capable to ask for help to the oracle whenever in need by sending him a signal or even by asking questions through natural language.

Data Centric Learning Collecting data for VLN tasks is really expensive and time consuming. All existing datasets are relatively small if compared to the complexity of the task.
Data Centric methods work around this problem by Augmenting the dataset with synthetic data produced starting from the existing data available. New trajectories can be created by generating random paths inside the environment and then using a trained model capable of generating natural language instructions given the path to generate the instructions. On the contrary new environments can be generated by randomly masking visual features or by subdividing the environments into house scenes and randomly mixing them. The two tecniques can also be combined togheter to generate even more data.
Another way to tackle the problem is by training the model into similar tasks. The knowledge gained from those tasks can then be transferred into VLN tasks and benefit agent's performances.

Prior Exploration Models which can reach a good performance on seen environments not necessarily are able to reach the same levels of performance on unseen environments. Prior exploration methods focus on allowing the agent to generalize well the skills learned inside seen environments over unseen environments.

3.4 Embodied Question and Answering

Embodied Question and Answering can be easily considered as the hardest task that can be currently encountered in embodied AI research.
In Embodied Question and Answering the agent not only has to be able to freely explore and navigate the environment but also has to correctly answer questions about the environment he is in.
The problem is often tackled by subdividing the agent into two sub-components: a navigation component and question and answering component. The first component is mandatory and can't be omitted for the correct success of the task. The agent must be able to explore the environment before answering

questions about it. These two sub-components are often evaluated and tested separately.
More complex research works in Embodied AI Q&A tasks have been proposed like for example multi target embodied Q&A where the agent is asked to answer to question that involve multiple objects inside the environment or Interactive Q&A where the agent needs to interact with the environment to answer the questions like for example by opening or closing some doors.

4 Dataset

This chapter will cover all the datasets used and encountered during the work of the thesis. Starting from Matterport3D [8] essential to virtually reconstruct realistic environments where our agent can easily navigate in for training and evaluation, arriving to Room to Room containing [13] and it's subsequent improvement Room Across Room [14] which both contain a set of paths and relative instructions used to train our agent.
I'll also talk about the Habitat environment [12] a very important and popular tool needed for executing simulations on a virtual environment.

4.1 Matterport3D

Matterport3D is a collection of 10,800 panoramic views constructed from 194,400 RGB-D images taken inside 90 different indoor home environments. This dataset was born with the objective of overcoming the limitations of previously existing datasets like for example: limited number of images contained inside the dataset, limited view points or motion-blurred pictures.
The dataset was captured using a tripod mounted camera rig with three color cameras and three depth cameras all of which rotate around the vertical axis to capture images from six different directions.
The panoramic images are then used to reconstruct a textured three-dimensional mesh containing all the visible surfaces of the environment.
The three-dimensional meshes are then manually labeled using specific tools developed by the authors in order to include segmentation data for both the regions of the reconstructed building and the objects contained in it.

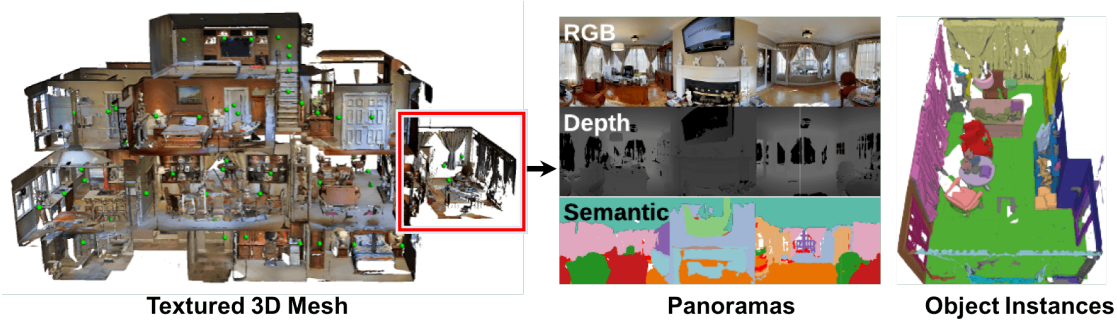


Figure 4.1: (*Left*) The textured three-dimensional mesh of the environment reconstructed starting from the RGB-D images. (*Center*) Example of an RGB image of the dataset along with it's Depth Image and object semantic segmentation*. (*Right*) The semantic annotation of the objects of a room.

The Matterport3D dataset possesses the following properties that differentiate it from previous panorama datasets:

- Matterport 3D provides full RGB-D panorama images. Previous datasets provided either no depth images at all or approximations of depth images derived from the mesh data.
- All 90 environments come with precise global alignment of camera poses. Previous RGB-D datasets provided limited data about the camera poses alignment often covering only few rooms or even small part of the rooms.
- Images are taken with a stationary camera to avoid motion blur and other artifacts commonly present in many other RGB-D datasets taken through hand-held cameras.
- Most RGB-D datasets provide data for single rooms or small sets of adjacent rooms. Matterport on the contrary provides data for 90 entire buildings.
- Images taken inside private homes. Most other RGB-D datasets are often limited to academic spaces.
- Probably the largest RGB-D dataset available.

The 90 scenes are split in three sets for experiment purposes, 61 scenes are reserved for training, 11 for validation and the remaining 18 for testing.

4.2 Habitat

As mentioned in it's website *"Habitat is a simulation platform for research in Embodied AI"*[12]. Training and evaluating an agent inside the real world poses many problems that normally don't need to be confronted in a virtual environment like:

*It is the process of dividing an image into different regions based on the characteristics of pixels to identify objects or boundaries.[17]

- Slowness of training due to physical world limitations and lack of parallelization.
- Danger of hurting both the agent's robotical system and the people in environment near him.
- High costs of physical hardware.
- Difficulty or even impossibility of replicating the same environment conditions.

Thanks to Habitat Embodied Agents can be trained with much lower costs. When training is finished the skills learned inside the virtual environment can then be transfered to the real world.
Prior to Habitat many different simulators where developed and each one of them differed with respect to the 3D data they used. The existence of multiple simulators can cause problems like fragmentation, reproduction of effort and difficulty in reproduction of the experiments.

- Tight coupling of datasets simulators and tasks.
- Hard-Coded agent configuration.
- Poor rendering and agent performances.
- Restricted control over the simulation environment. Position of 3D objects cannot be easily modified.

Habitat AI is mainly made by three components:

Habitat Sim The real 3D simulator extensively described above which includes support for physics simulations, rigid body mechanics and navigation inside 3D scans of indoor/outdoor spaces (like the Matterport3D datasets) or CAD models.

Habitat Lab A library built on top of Habitat Sim that allows to easily define Embodied AI tasks, like interaction or navigation, and configure the simulated agent's physical properties including agent shape and dimensions, number and kind of sensors and general capabilities.

Habitat Challenge An annual challenge that aims to benchmark and accelerate progress in the field of Embodied AI. The participants are asked to upload the agent's code which is then evaluated over unseen environments to test it's generalization. Habitat was designed to be an unifying platform that aims to overcome all of the previously cited shortcomings.

4.3 Room To Room (R2R)

Room To Room, often abbreviated as R2R, is a dataset containing over 21.567 natural language instructions with an average lenght of 29 words that describe agent trajectories over the 3D reconstructed environments comprised in the Matterport3D dataset.

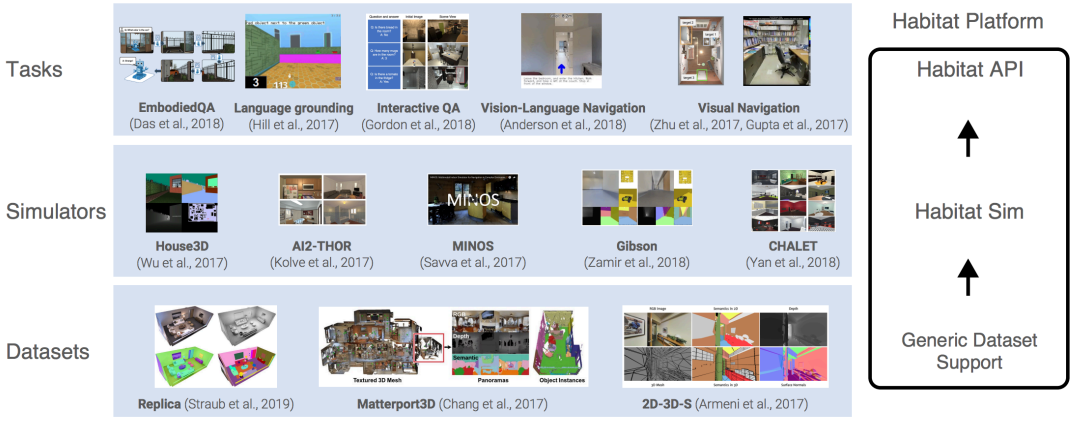


Figure 4.2: The standardized software stack proposed by Habitat AI. At the lower layer we find the datasets containing the 3D assets and it's semantic annotations. Then simulators (like Habitat) use such assets to render realistic environments in which the agent can navigate. Finally on top of the simulators tasks are defined to evaluate scientific progress.

The R2R dataset is built around a Visual Language Navigation task in which the environment is modeled as a graph $G = (V, E)$ where:

- V is the set of all 3D points associated with a panoramic viewpoint in the scene.
- E is the set of edges of the graph. The presence of an edge indicates robot navigability between the two linked viewpoints.

The agent starts from an initial state $\mathbf{s}_0 = \{\mathbf{v}_0, \psi_0, \theta_0\}$ where:

- $\mathbf{v}_t \in V$ represents the position of the agent at timestep t.
- $\psi_t \in [0, 2\pi]$ represents the agent's heading at timestep t.
- $\theta_t \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ represents the camera elevation at timestep t.

At each timestep t the agent receives in input:

- A natural language instruction $\bar{\mathbf{x}}_t = \mathbf{x}_1, ..., \mathbf{x}_L$ where $\mathbf{x}_i$ is the feature representation of the i^{th} word in the instruction and L is the lengt of the instruction.
- An RGB image $\mathbf{o}_t$ obtained from the Matterport3D environment given the current state $\mathbf{s}_t$.

Based on the current state $\mathbf{s}_t$, on the instruction $\bar{\mathbf{x}}_t$ and on the observation $\mathbf{o}_t$ the agent chooses an action $\mathbf{a}_t$ to execute that leads him to a new state $\mathbf{s}_{t+1} = \{\mathbf{v}_{t+1}, \psi_{t+1}, \theta_{t+1}\}$. Each $\mathbf{v}_t \in \mathbf{s}_t$ must respect the following condition:

$$\mathbf{v}_t \in W_t \tag{4.1}$$

Where $W_t \subset V$ is defined as the set of all reachable viewpoints from the current viewpoint v_t that are framed by the camera, or more formally as:

$$W_t = \{\mathbf{v}_{t-1}\} \cup \{\mathbf{v}_i \in V | (\mathbf{v}_{t-1}, \mathbf{v}_i) \in E \wedge \mathbf{v}_i \in P_{t-1}\} \tag{4.2}$$

Where P_t is the region of space enclosed by the camera at timestep t.
The objective of the agent is finding a sequence of actions $\mathbf{a}_0, \ldots, \mathbf{a}_T$ that brings him as close as possible to the goal $\mathbf{v}_*$, where $\mathbf{a}_T$ is a special action stop.
The trajectories in the dataset were generated by procedurally sampling a start pose $\mathbf{s}_0$ and a goal location $\mathbf{v}_*$ and then computing the shortest path between them. The sampled trajectories were then showed to human workers who had to provide a valid description via a specific tool developed by the authors of the paper.

Below is an example of a dataset's instruction:

> *Go past the ovens and the counter and wait just before you go outside.*
> *Walk through the kitchen towards the living room. Walk around the island and step onto the patio near the two chairs and stop in the patio doorway.*
> *Exit the kitchen by walking past the ovens and then head right, stopping just at the doorway leading to the patio outside.*

The dataset follows the same splits defined in the Matterport3D dataset. A total of 4.173 instructions over 18 different scenes are reserved for testing and 2.349 instruction are reserved for validation over 11 environments unseen during training. The remaining 61 scenes are all encountered during training, the set of instructions over those 61 scenes are split in 14.025 scenes for training over seen environments and 1.020 for validation over environments seen during training.

4.4 Room Across Room (RxR)

Room Across Room (RxR) is a dataset that improves on R2R by trying to address some of his shortcomings.

Multilinguality Besides English, RxR includes instructions for two more typologically different languages, Hindi and Telugu. All the instructions are manually translated by native speakers.

Scale Many Visual Language navigation Tasks suffer from scarsity of training data. RxR addresses this shortage by extensively augmenting the data at disposal. For each language RxR contains 14.000 paths with 3 instructions per path, for a total of 126.000 instructions.

Fine-Grained Grounding Like in R2R human annotators are asked to describe predefined paths with natural language instruction. In RxR however the annotation tool keeps track of the position of the annotator in the environment while he is describing the path and later time-aligns with the words in the instruction.

Follower Demonstrations Annotators are also asked to act as followers, meaning to listen to another annotator's instructions and follow the path indicated by him. This is a good way to measure instructions performance and gives us an indication of how a real human interpreted the instructions. These human interpreted trajectories are usefull for agent training.

Path desiderata RxR paths are usually much longer than R2R ones and are generated by following rules that make them much more natural and easier to describe by human annotators. Path variance is also increased to avoid agents to learn priors that are not generalizable in other environments.

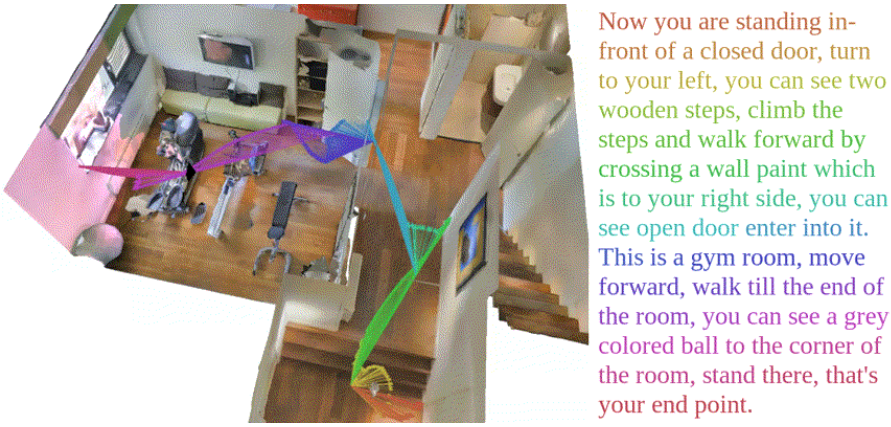


Figure 4.3: Example of an RxR instruction along with the path taken by the agent in the environment. The image also shows how both the position of the agent in the trajectiory and the words in the instruction are temporally aligned. The position in time is indicated by the color gradient.

In total 16522 paths are sampled which are subdivided by following the same splits of Matterport3D and R2R. 11.089 trajectories for training, 1.232 for validation over environments seen during training, 1.517 over unseen environments and 2.684 for testing.

4.5 RxR Challenge

RxR also includes a separated test set used for holding annual challenges. This set is divided in two different splits: test-standard and test-challenge. A public leadearboard is also provided to allow the community to track progress and fairly evaluate the performance of their model. More specifically RxR hosts two challenges:

- **RxR Competition**: Where the agent navigates trough a sparse graph of panoramas.
- **RxR-Habitat Competition**: Where the agent navigates in a continuos environment using the Habitat Simulator.

All the thesis work was made upon the starting code of the RxR-Habitat Competition [16].

5 Methods

Now that all the necessary background information has been provided this chapter will dive deep into the thesis work.
Firstly Section 5.1 will give a brief introduction to the attention mechanisms [19] used by many models encountered during the thesis work. Then in Section 5.2 the original project [18] that acted as starting bases for all the thesis work is going to be introduced.
Finally Section 5.3 and Section 5.4 will describe the proposed modifications made to the original model in order to increase it's performance.

5.1 Attention Mechanisms

"An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and the output are all vectors."[19].

LSTMs and GRUs have long established as state of the art approaches for modeling long sequences. Despite their proven effectiveness their intrinsic sequential nature makes parallelization during training and evaluation really hard to implement. This becomes critical over long sequences due to the memory constraints. Attention mechanisms allow to model dependencies between sequences without any regard about their distance.

The authors of [19] firstly propose a simple attention mechanism called Scaled Dot-Product Attention which takes in input a set of queries and keys of size d_k and a set of values of size d_v. In practice all the

queries, keys and values are packed togheter forming the matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$. The attention function is computed as follows:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V} \tag{5.1}$$

Also a second attention mechanism called Multi-Head Attention is proposed. In this model queries, keys and values are linearly projected h times, each time using a different learned linear projection and then the previously defined Scaled Dot-Product Attention is applied. The h different outputs are finally concatenated togheter.

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, .., \text{head}_h)\mathbf{W}^{\mathbf{O}} \tag{5.2}$$

Where

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^{\mathbf{Q}}, \mathbf{K}\mathbf{W}_i^{\mathbf{K}}, \mathbf{V}\mathbf{W}_i^{\mathbf{V}}) \tag{5.3}$$

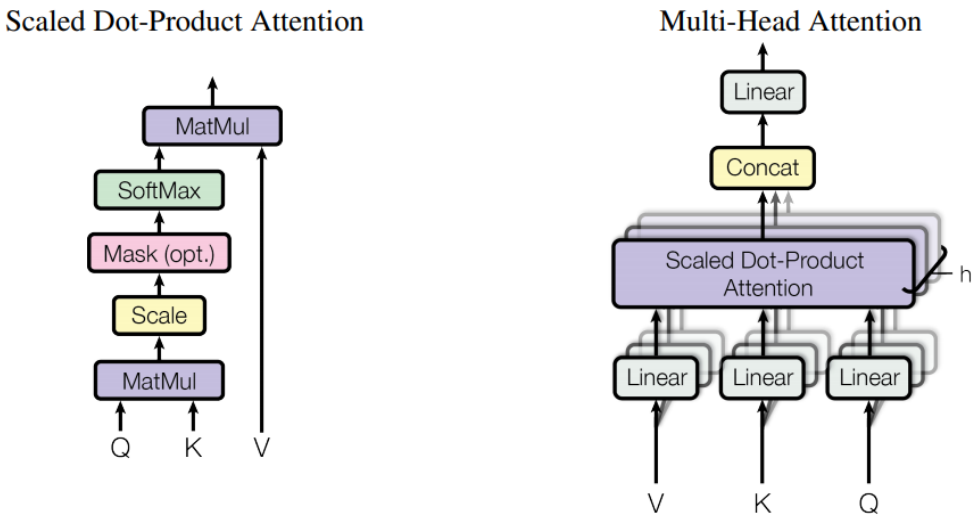


Figure 5.1: (Left) Scaled Dot-Product Attention (Right) Multi-Head Attention

Finally the authors of [19] propose the transformer architecture, a new model which doesn't make use of recurrence but instead completely relies on attention mechanisms. The transformer architecture is composed by two components, the encoder and the decoder.

Encoder The encoder is composed by a stack of N identical layers. Each one of them is made out of two sub-layers, a multi-head attention mechanism followed by a fully connected feed forward network.

Decoder Similarly to the encoder the decoder is composed by N identical layers. Here however each layer posses a third sub-layer which performs multi head attention over the output of the encoder stack. Moreover the first attention layer is modified in order to mask at each position the subsequent positions. This ensures that the output for the position i depends only by the positions lower than i.

Both in the encoder and in the decoder layer normalization and residual connections are applied at each sub-layer. All sub-layers and layers as well produce outputs of the same dimension d_{model}.

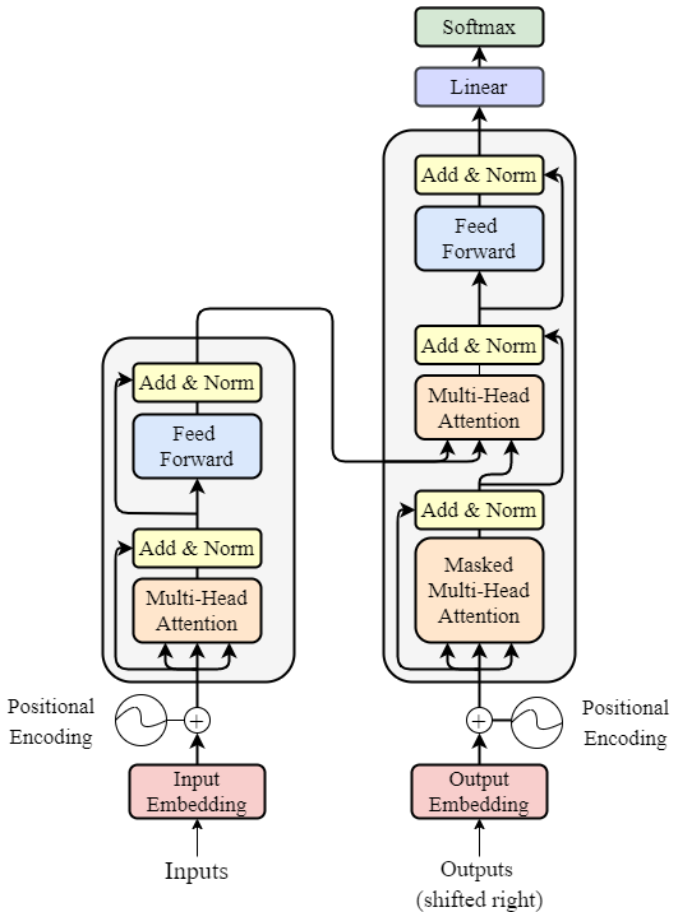


Figure 5.2: Graphical representation of the transfomer model

5.2 Visual Language Navigation in Continuos Environments

The authors of [18] introduce a new modeling technique for Visual Language Navigation tasks. Previous tasks were usually defined over navigation graphs. In a navigation graph each node corresponds to a 360° panoramic image taken at a fixed location and each edge that connects two nodes indicates that the agent can navigate between the two locations in the environment. Navigation Graph make too many assumptions and poorly abstract the problem.

Known topology Rather than operating on a continuos environment the agent operates on graphs of fixed topology. Even in unseen environment the agents still possess prior information about the structure of the environment. Furthermore it is still unknown how an agent should acquire and update such topology in new environments where no data is possessed

Oracle navigation The model implies the presence of an oracle capable of navigating the agent between the nodes connected in the graph while avoiding possible obstacles present between the two locations. This abstracts too much the problem of visual navigation making it more similar to teleportation.

Perfect localization The agent knows at all times it's precise location and orientation inside the environment.

VLN-CE lifts this assumptions by placing the agent in continuous three dimensional environment where he can move freely trough a set of predefined actions. More specifically the agent can take the following actions:

- move forward of 0.25 meters
- turn left of 15 degrees
- turn right of 15 degrees
- stop, when he thinks he has reached the goal

5.2.1 Trajectory Conversion

Instead of creating a new dataset the authors of [18] transform the original R2R dataset from a navigation-graph setting to a continuous environment. Since the trajectories are defined over navigation graphs and each node of the navigation-graph possesses a correspondent (x, y, z) coordinate it seems seemingly simple to convert the coordinates into continuous environments. However such locations not always correspond to actual reachable locations in the environment. Figure 5.4 shows some of the problems that can

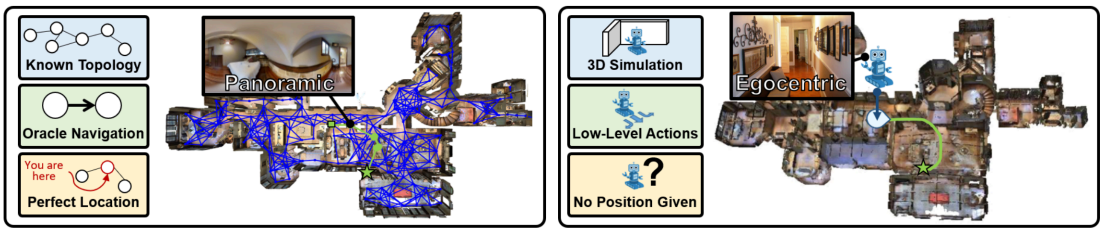


Figure 5.3: Comparison between classical Vision and Language Navigation and Vision and Language Navigation in Continuos Environments. On the left the fixed topology where the agent operates on classical visual and language navigation. On the right the path taken by the agent in a continuous environment.

be encountered during the conversion.
For each node the nearest navigable point within 0.5m is identified by casting a ray long two meters down-

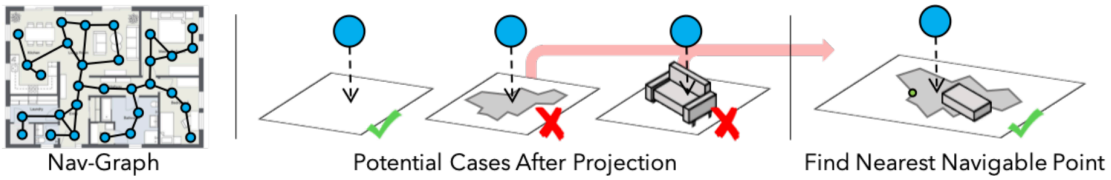


Figure 5.4: (Left) Reachable Location. (Centre) Hole in the map where the mesh reconstruction Failed. (Right) The coordinate refers to a place occupied by furniture, commonly tables where the camera was placed during data collection.

ward from the coordinate. At small fixed intervals along the ray the position is projected to the nearest mesh point. If multiple navigable points are found the one with less horizontal displacement from the coordinate is chosen. If no navigable point with 0.5m from the coordinate is found the coordinate is considerate as unreachable inside the reconstructed Matterport3D mesh and thus invalid. Some of the invalid nodes where manually fixed by for example shifting them on the side of a furniture. After the manual fix 98.3% of the coordinates are successfully transferred.
Given a trajectory of converted coordinates $\tau = [\mathbf{w}_1, ..., \mathbf{w}_T]$ to verify that an agent can actually navigate through the trajectory an A*-Based heuristic is employed to compute an approximated shortest path between each pair of coordinates $\mathbf{w}_i$ and $\mathbf{w}_{i+1}$. A trajectory is considered navigable if for each pair of consecutive coordinates it's possible to reach a location within 0.5m from the next coordinate by following the computed shortest path. In total 77% of the coordinates are navigable.

5.2.2 Data Representation

Natural language instructions are firstly tokenized and then transformed into GloVe embeddings. GloVe is an algorithm used to produce vector representation of words. Most word vector methods measure word similarity by calculating the euclidean distance of the two vector representations. GloVe instead measures the various dimensions of difference between the words. For example the difference between two words like *king* and *queen* should be really similar to the difference between two other words like *man* and *woman*.
We denote the GloVe embedded tokens as $\mathbf{w}_1, \ldots, \mathbf{w}_T$ where T is the length of the instruction. Theese embeddings are then going to be encoded by the models with a recurrent encoder network.

Similarly the RGB and Depth obsevations are encoded separately using different models. The RGB observations are encoded with a ResNet50 pretrained on ImageNet, while the depth observations are encoded with a modified ResNet50 trained on point-goal navigation tasks. We define the set of the encoded RGB observations as $\mathcal{V} = \{\mathbf{v}_i\}$ and the set of the encoded Depth observations as $\mathcal{D} = \{\mathbf{d}_i\}$.

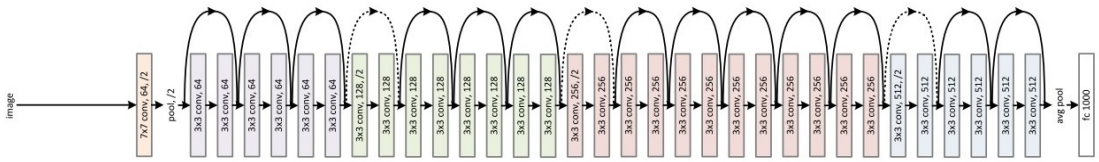


Figure 5.5: Graphical representation of a Residual Network (ResNet). More Specifically the picture depicts a ResNet34 network. The network is composed by 34 convolutional blocks connected at the end with a fully convolutional layer.
The core idea behind residual networks is the introduction of shortcut connections. Connections that link a layer with another further down in the architecture by skipping one or more layers.

5.2.3 Sequence-to-Sequence Model

The first model proposed by the authors of [18] is a simple sequence-to-sequence (Seq2Seq) baseline model.
The model is made of a recurrent neural network that takes in input the encoding of the visual observations (Depth and RGB) and the instruction's embeddings.

Firstly an LSTM encoder network is used to encode the instruction's embeddings.

$$\mathbf{s} = \text{LSTM}(\mathbf{w}_1, \ldots, \mathbf{w}_T) \tag{5.4}$$

The visual features are then concatenated together and a mean pooling operation is applied to the RGB features.

$$\begin{aligned} \bar{\mathbf{v}}_t &= \text{mean-pool}(\mathcal{V}_t) \\ \bar{\mathbf{d}}_t &= [\mathbf{d}_1, ..., \mathbf{d}_{wh}] \end{aligned} \tag{5.5}$$

Finally we give in input the previously defined vectors to a GRU network.

$$\begin{aligned} \mathbf{h}_t^{(a)} &= \text{GRU}([\bar{\mathbf{v}}_t, \bar{\mathbf{d}}_t, \mathbf{s}], \mathbf{h}_{t-1}^{(a)}) \\ a_t &= \underset{a}{\text{argmax}} \quad \text{softmax}(W_a \mathbf{h}_t^{(a)} + \mathbf{b}_a) \end{aligned} \tag{5.6}$$

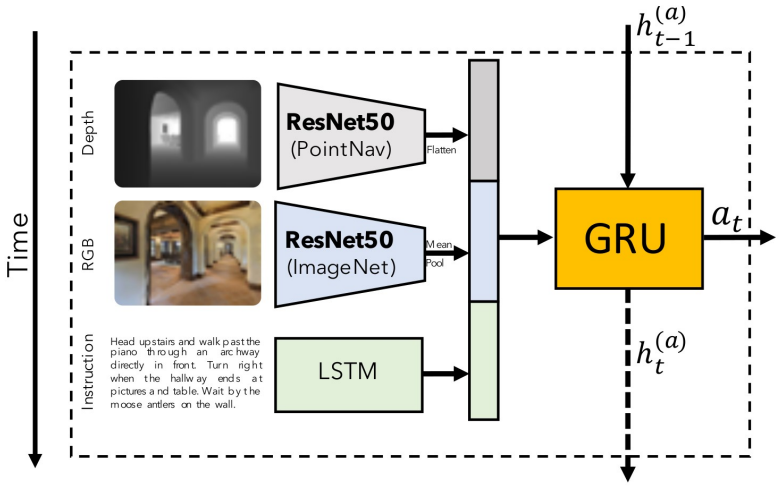


Figure 5.6: Graphical representation of the Sequence to Sequence Model

5.2.4 Cross-Modal Attention Model

The previous model is a great start but better performances can be achieved by implementing into the architecture powerful state of the art modeling techniques like attention mechanisms and spatial visual reasoning.
The second model proposed by the architecture is composed of two recurrent networks, one in charge of tracking visual observations and the other one in charge of making decisions based upon the instructions and the visual observations.

The first recurrent network is defined similarly to the one seen in the Seq2Seq model. This network

however takes in input only the visual features along with a learned linear embedding of the previous action.

$$\begin{aligned}\bar{\mathbf{v}}_t &= \text{mean-pool}(\mathcal{V}_t)\\ \bar{\mathbf{d}}_t &= [\mathbf{d}_1, ..., \mathbf{d}_{wh}]\\ \mathbf{h}_t^{(attn)} &= \text{GRU}([\bar{\mathbf{v}}_t, \bar{\mathbf{d}}_t, \mathbf{a}_{t-1}], \mathbf{h}_{t-1}^{(attn)})\end{aligned} \tag{5.7}$$

The instructions are encoded using a bi-directional LSTM. All of the intermediate states are kept in memory to use them in later computations.

$$\mathcal{S} = \{\mathbf{s}_1, ..., \mathbf{s}_T\} = \text{BiLSTM}(\mathbf{w}_1, ..., \mathbf{w}_T) \tag{5.8}$$

The following scaled dot product attention is defined.

$$\text{Attn}(\{\mathbf{x}_i\}, \mathbf{q}) = \sum_{\forall i} \frac{\text{softmax}((\mathbf{W}_K \mathbf{x}_i)^T \mathbf{q})}{\sqrt{d_q}} \mathbf{x}_i \tag{5.9}$$

An attended instruction feature $\hat{\mathbf{s}}_t$ is computed which is in turn used to compute the attended visual features $\hat{\mathbf{v}}_t$ and $\hat{\mathbf{d}}_t$

$$\begin{aligned}\hat{\mathbf{s}}_t &= \text{Attn}(\mathcal{S}, \mathbf{h}_t^{attn})\\ \hat{\mathbf{v}}_t &= \text{Attn}(\mathcal{V}_t, \hat{\mathbf{s}}_t)\\ \hat{\mathbf{d}}_t &= \text{Attn}(\mathcal{D}_t, \hat{\mathbf{s}}_t)\end{aligned} \tag{5.10}$$

The second recurrent network takes in input a concatenation of all the features listed above along with the current state of the first GRU and an encoding of the previous chosen action.

$$\begin{aligned}\mathbf{h}_t^{(a)} &= \text{GRU}([\bar{\mathbf{s}}_t, \hat{\mathbf{v}}_t, \hat{\mathbf{d}}_t, \mathbf{a}_{t-1}, \mathbf{h}_{t-1}^{(a)})\\ a_t &= \underset{a}{\text{argmax}} \quad \text{softmax}(W_a \mathbf{h}_t^{(a)} + \mathbf{b}_a)\end{aligned} \tag{5.11}$$

5.3 Improving instruction's encoding

The natural language instructions given in input to the agent describe the path that the agent must take from it's starting location to reach the goal. It's of key importance then to provide in input to the agent's model a representation of the instructions which encapsulates as much information as possible inside it. The RxR Habitat Starter Code [16] uses BERT [22] to encode before training time the natural language instructions.

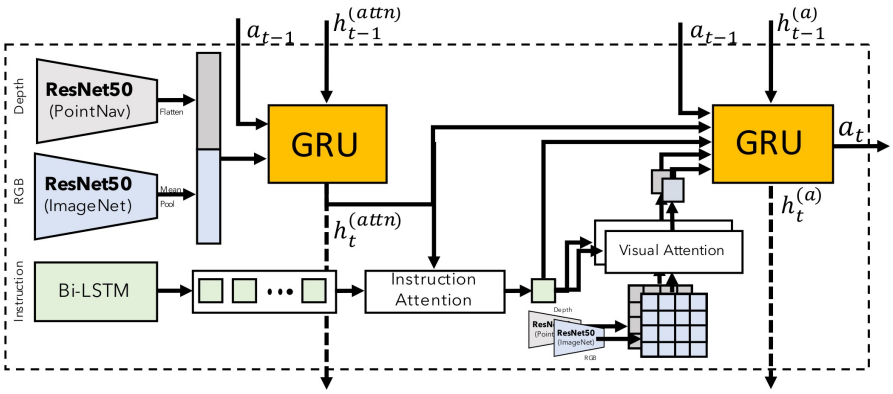


Figure 5.7: Graphical representation of the Cross-Modal Attention Model

RoBERTa is a state of the art model based upon the original BERT model which improves it's performances over many situations. Our first contribution consists in implementing the RoBERTa encoder into our model architecture in place of the original BERT model.

5.3.1 BERT

BERT's architecture consists of a multilayer bi-directional transformer encoder. More specifically [22] proposes two models:

- **BERT$_{\text{BASE}}$**$(N = 12, d_{\text{model}} = 768, h = 12)$
- **BERT$_{\text{LARGE}}$**$(N = 24, d_{\text{model}} = 1024, h = 16)$

Where N is the number of the encoder's layers, d_{model} the size of the embedding layers and h the number of projections applied in the multi-head attention mechanisms.

BERT is capable of representing both a single sentence and a pair of sentences (For example question and answer) in one token sequence. To differentiate the first sentence from the second one the two sentences are separeted by a special token [SEP]. Every sentence always starts with a special classification token [CLS], the final representation for this token produced by the model is used as an aggregate sequence representation for classification tasks.
BERT training is composed of two steps: pre-training and fine-tuning.

PRETRAINING During pre-training the model is trained on unlabeled data collected over two different pre-training tasks. The first task is called masked LM and consists in randomly masking a percentage of the input tokens and then asking the model to predict the masked tokens. When a token is chosen for masking it can be replaced with:

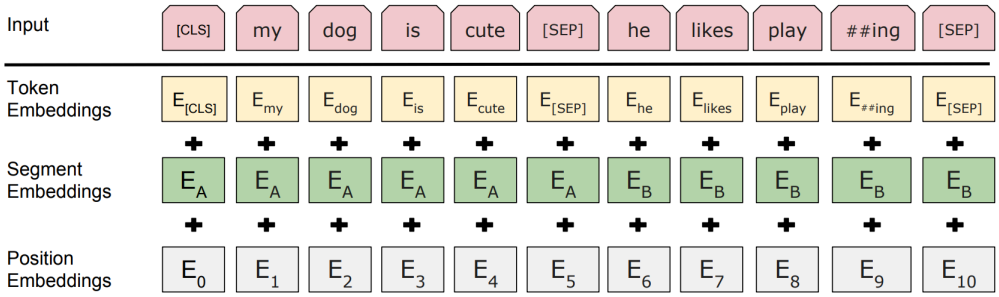


Figure 5.8: BERT input representation is the sum of the token embeddings obtained via wordpiece embeddings [24], the learned segment embeddings which differentiate between the first and second sentence, and the positional embeddings.

- a special [MASK] token (80% of the times)
- a random token (10% of the times)
- with the same unchanged token (10% of the times)

This is done to avoid a mismatching between pre-training and fine-tuning where the [MASK] token aren't present.
To allow BERT to capture relationships between two sentences the model is trained over a next sentence prediction task. Each example in the task is composed by two sentences and a binary label which indicates if the second sentence naturally follows the first one or not. Such dataset can be easily created from a monolingual unlabeled corpus. The feature representation C is used for the next sentence prediction task.

Fine-Tuning On the other hand fine-tuning is very straight-forward. The model is trained over the task's specific inputs and outputs starting from the pre-trained model's weights. The token representations are then fed to an output layer for token level tasks like question and answering. Similarly the [CLS] representation is fed to an output layer for classification tasks.

5.3.2 RoBERTa

Robustly Optimized BERT Approach (RoBERTa) is an improved version of the original BERT model that can match or exceed the performance of all the post-BERT methods. RoBERTa proposes the following modifications to the original BERT model.

Dynamic Masking As stated in the previous section BERT pre-training is done by randomly masking a subset of the input tokens and subsequently asking the model to predict them.
In the original implementation of BERT the masking was performed once during data pre-processing

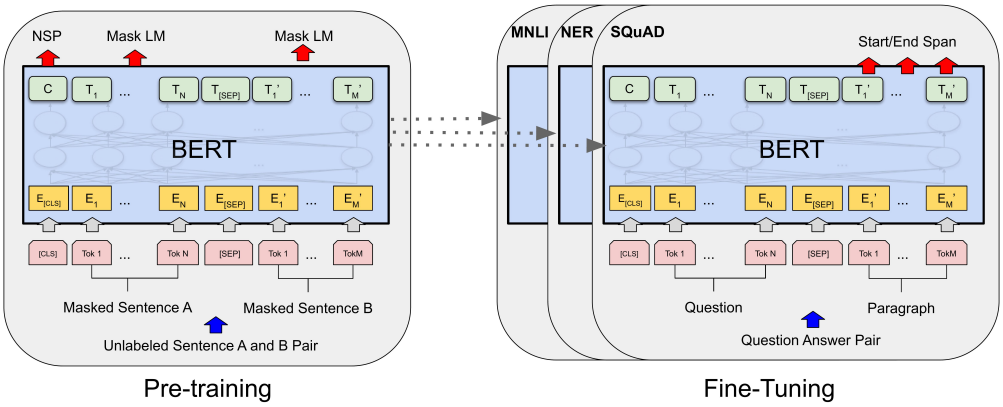


Figure 5.9: BERT architecture in both pre-training and fine-tuning. The input embeddings are denoted as E while the final representation produced by the BERT model of the special token [CLS] as $C \in \mathbb{R}^{d_{\text{model}}}$ and representation of the i^{th} input token as $C \in T_i^{d_{\text{model}}}$.

causing a single static mask. Since the model was trained over 40 epochs and each sentence was repeated 10 times with 10 different masks each mask was encountered 4 times during training.
RoBERTa implements dynamic masking by generating the masking pattern at the start of each epoch. This is very important for training over larger datasets or with more epochs.

Next Sentence Prediction Input Format Research works have questioned the utility of the NSP loss in the original BERT model. Several alternative training formats are compared:

- **Segment-Pair+NSP**: The input format used in the original implementation of BERT. The model receives in input two sequences of words which can contain multiple natural sentences. The NSP loss is retained.
- **Sentence-Pair+NSP**: The input contains a pair of natural sentences that can be sampled from a contiguous portion of the same document or from different documents. The NSP loss is retained.
- **Full-Sentences**: The model receives in input a full sentence sampled contiguosly from one or more documents. If the end of the document is reached a special token is added as separator to indicate the beginning of a new document. The NSP loss is removed.
- **Doc-Sentences**: The same as Full-Sentences but the input can belong to only a single document. The NSP loss is removed.

The authors of [23] found out that Doc-Sentences outperforms the original $\text{BERT}_{\text{BASE}}$ model and that removing the NSP loss matches or slightly improves the performances of the downstream tasks.

Training with Longer Batches Past works in Neural Machine Translations have shown how training with larger batches can improve performances if the learning rate is increased appropriately. Also larger batches are easier to parallelize.

Text Encoding Byte Pair Encoding (BPE) is an hybrid between character and word-level encodings. BPE uses subword units calculated trough statistical analysis. The original BERT implementation uses a character level BPE vocabulary of size 30K. RoBERTa instead uses a byte-level vocabulary of 50k subword units.

5.4 Auxiliary Tasks

Auxiliary tasks are used to provide to the agent complementary objectives that can improve the efficiency of the primary task. Auxiliary Tasks are often implemented via self-supervised tasks which derive the supervision through the agent's own experience and hence do not need labeled data. To improve the agent performances we tried to implement the auxiliary tasks introduced by [26] and [27] into the VLN-CE architecture described in Section 5.2.

The architecture proposed by [26] is separated into three different components, an encoder network which encodes the input observations, a belief module which produces a summary representation given multiple observations and a policy module which chooses the next action given a belief module output.

In practice each Auxiliary Task operates on observations, chosen actions and outputs of the belief modules. Moreover each Auxiliary Task possesses it's own belief module. The outputs of the belief modules of all the auxiliary tasks are then fused together creating the summary representation that is then given input to the policy module.
During training the parameters are optimized to minimize the following loss function

$$L(\theta_m; \theta_a^1 ... \theta_a^{n_{\text{Aux}}}) = L_{\text{RL}}(\theta_m) + \sum_{i=1}^{n_{\text{Aux}}} \beta_i^{\text{Aux}} L_{\text{Aux}}(\theta_m; \theta_a^i) \tag{5.12}$$

Where:

- $L_{\text{RL}}(\theta_m)$ is the primary objective loss
- θ_m are the parameters of the encoder and policy modules
- θ_a^i are the parameters of the i^{th} auxiliary loss
- n_{Aux} is the number of auxiliary losses
- β_i^{Aux} is an hyper-parameter used for balancing the losses

Four different Auxiliary Tasks where implemented into the VLN-CE model architecture:

- Contrastive Predictive Coding Action (CPC|A) [28, 29]
- Prediction of Bootstrap Latents (PBL) [30]
- Generalized Inverse Dynamics (GID) [27]
- Action Distribution Prediction (ADP) [27]

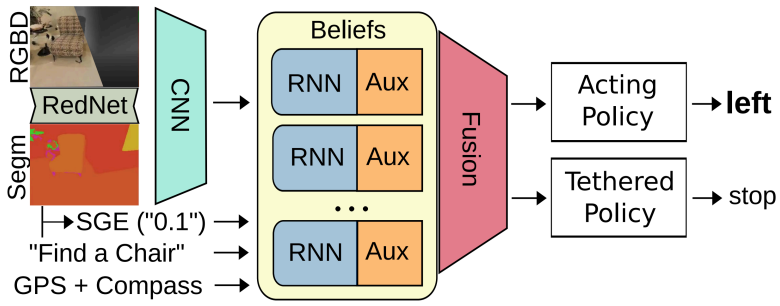


Figure 5.10: Auxiliary Task Architecture

5.4.1 Contrastive Predicting Coding

Contrastive Predictive Coding(CPC) is an unsupervised representation learning method.
Firstly a non-linear encoder g_{enc} maps the input sequence of observations $\mathbf{x}_1, \ldots, \mathbf{x}_t$ to a sequence of latent representations $\mathbf{z}_1, \ldots, \mathbf{z}_t$ where $\mathbf{z}_i = g_{\text{enc}}(\mathbf{x}_i)$. Finally an auto-regressive model g_{ar} produces a context latent representation $\mathbf{c}_i = g_{\text{ar}}(\mathbf{z}_i)$ for all $i < t$. Future observations $\mathbf{x}_{t+k}$ are then predicted through a simple bilinear-model:

$$f_k(\mathbf{x}_{t+k}, \mathbf{c}_t) = exp(\mathbf{z}_{t+k}^T \mathbf{W}_k \mathbf{c}_t) \tag{5.13}$$

The authors of [28] use a linear transformation $\mathbf{W}_k \mathbf{c}_t$ with a different $\mathbf{W}_k$ at each step k. Alternatively a non linear feed-forward network or recurrent neural network can be used.

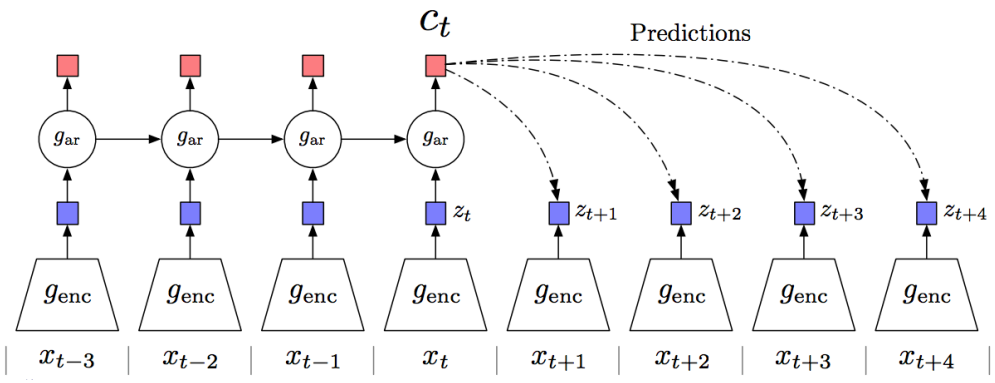


Figure 5.11: CPC architecture.

5.4.2 CPC|Action

CPC|A is representation learning method based on the CPC architecture described in Section 5.4.1.
A Partially Observable Markov Decision Process (POMDP) is used as a general framework to define the partially observable sthocastic environment where the agent operates. Formally POMDP is defined as $M = (\mathcal{X}, \mathcal{A}, \mathcal{O}, \mathcal{P}, O)$ where:

- $\mathcal{X}$ is the state space.
- $\mathcal{A}$ is the action space.
- $\mathcal{O}$ is the observation space.
- $\mathcal{P}$ maps each state-action pair $(\mathbf{x}, \mathbf{y})$ to a probability over the next state y $P(\mathbf{y}|\mathbf{x}, \mathbf{a})$.
- O maps each state x to a probability $O(\cdot|\mathbf{x})$ over possible observations.

In a POMDP the agent, at any given timestep t, has access only to some observation $\mathbf{o}_t \in \mathcal{O}$ that gives incomplete information about the real state $\mathbf{x}_t \in \mathcal{X}$.
It becomes crucial then to compute a belief state $\mathbf{b}_t$ that models a probability distribution $P_b(\cdot|\mathbf{h}_t)$ over the possible states given the current history of past actions and observations $\mathbf{h}_t = \{\mathbf{o}_0, \mathbf{a}_0, \ldots, \mathbf{o}_t, \mathbf{a}_t\}$.

Figure 5.12 shows the CPC|A architecture. In addition to the standard CPC architecture, composed by the two Convolutional and GRU blocks (yellow and blue), CPC|A uses the belief $\mathbf{b}_t$ to initialize a GRU (red) which is then fed by the future actions $\{\mathbf{a}_{t+k}\}_{k=0}^{T-1}$. Moreover at each time step a feed-forward network (grey MLP block) receives in both both the output of this last GRU(red) and the embedded observation z_{t+k}

5.4.3 PBL

Similarly to CPC|A the environment is modeled as a POMDP $(\mathcal{X}, \mathcal{A}, \mathcal{O}, \mathcal{P}, O, r, \gamma)$ where $\mathcal{X}, \mathcal{A}, \mathcal{O}, \mathcal{P}, O$ are defined identically to Section 5.4.2, $r \in \mathbb{R}^{\mathcal{X}\times\mathcal{A}}$ represents the reward function and γ is the discount factor.
The agent chooses the next action via a policy $\pi : \{\mathcal{O} \times (\mathcal{A} \times \mathcal{O})^n : n \in \mathbb{N}\} \to \Delta(\mathcal{A})$ where $\Delta(\mathcal{A})$ denotes the space of action distributions.
Given a fixed policy π, the starting state x_0^π, observation o_0^π, history h_0^π and action a_0^π are initialized as follows:

$$\begin{aligned} \mathbf{x}_0^\pi &\sim \rho \\ \mathbf{o}_0^\pi &\sim O(\mathbf{x}_0^\pi) \\ \mathbf{h}_0^\pi &= \mathbf{o}_0^\pi \\ \mathbf{a}_0^\pi &\sim \pi(\mathbf{h}_0^\pi) \end{aligned} \tag{5.14}$$

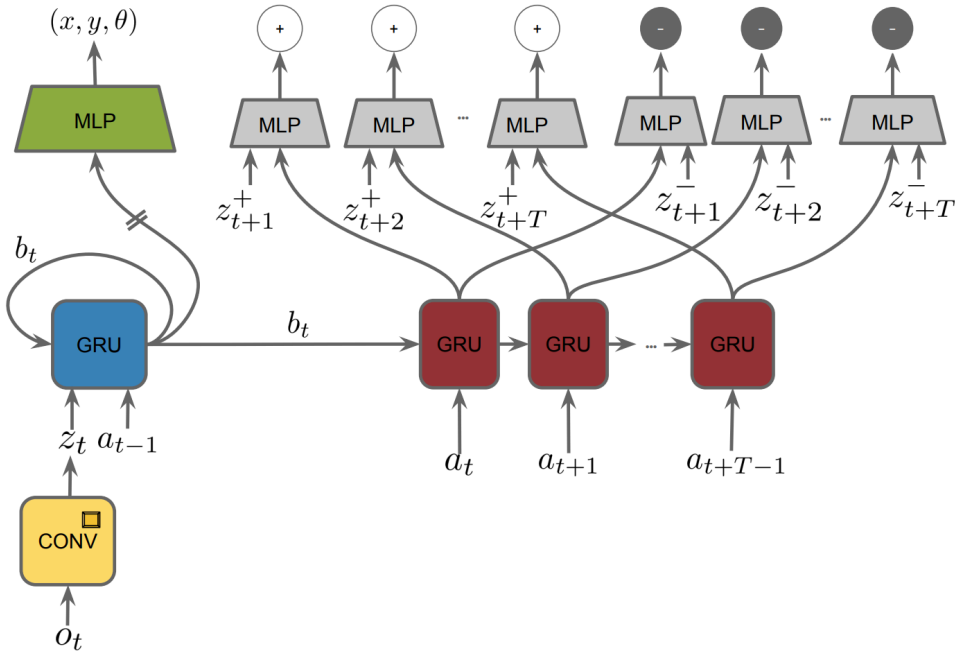


Figure 5.12: CPC|A architecture

Where ρ is some initial distribution.

Then for all timesteps $t > 0$ we recursively define $\mathbf{x}_t^\pi, \mathbf{o}_t^\pi, \mathbf{h}_t^\pi, \mathbf{a}_t^\pi$ as follows:

$$\begin{aligned}
\mathbf{x}_t^\pi &\sim P(\mathbf{x}_{t-1}^\pi, \mathbf{a}_{t-1}^\pi) \\
\mathbf{o}_t^\pi &\sim O(\mathbf{x}_t^\pi) \\
\mathbf{h}_t^\pi &= (\mathbf{h}_{t-1}^\pi, \mathbf{a}_{t-1}^\pi, \mathbf{o}_{t-1}^\pi) \\
\mathbf{a}_t^\pi &\sim \pi(\mathbf{h}_t^\pi)
\end{aligned} \tag{5.15}$$

Moreover the partial history $\mathbf{h}_{t,k}^\pi$ is defined as the history $\mathbf{h}_t^\pi$ and the k subsequent actions:

$$\mathbf{h}_{t,k}^\pi = (\mathbf{h}_t^\pi, \mathbf{a}_t^\pi, ..., \mathbf{a}_{t+k-1}^\pi) \tag{5.16}$$

The model is trained in the POMDP setting by finding the policy that maximizes the expected and discounted sum of rewards:

$$\max_\pi E(\sum_{t=0}^{\inf} \gamma^t r(\mathbf{x}_t^\pi, \mathbf{a}_t^\pi)) \tag{5.17}$$

The problem can also be tackled by compressing the full history $\mathbf{h}_t^\pi$ as the agent state $\mathbf{b}_t^\pi$ through a neural network. So the problem that needs to be solved is the following:

$$\max_\pi E(\sum_{t=0}^{\inf} \gamma^t r(\mathbf{b}_t^\pi)) \tag{5.18}$$

Where:

$$r(\mathbf{b}_t^\pi) = E(r(\mathbf{x}_t^\pi, \mathbf{a}_t^\pi)|\mathbf{b}_t^\pi) \tag{5.19}$$

From now on the π notation will be dropped from the variables to make formulas less confusing.
Figure 5.13 describes the architecture used to compress the histories. The horizontal direction outlines the RNN $\mathbf{h}_f$ compressing the full history $\mathbf{h}_t$ into the agent state $\mathbf{b}_t$. So the agent state $\mathbf{b}_t$ is defined as follows:

$$\begin{aligned} \mathbf{b}_0 &= 0 \\ \mathbf{b}_t &= \mathbf{h}_f(\mathbf{b}_{t-1}, \mathbf{o}_t, \mathbf{a}_{t-1}) \end{aligned} \tag{5.20}$$

On the other hand the vertical direction shows the RNN h_p compressing the partial histories $b_{t,k+1}$ as follows:

$$\begin{aligned} \mathbf{b}_{t,1} &= \mathbf{h}_p(\mathbf{b}_t, \mathbf{a}_t) \\ \mathbf{b}_{t,k+1} &= \mathbf{h}_p(\mathbf{b}_{t,k}, \mathbf{a}_{t+k}) \end{aligned} \tag{5.21}$$

The PBL architecture can be seen in Figure 5.14 and consists of two auxiliary prediction tasks:

Forward prediction task predicts latent embedded observations z_{t+k} from compressed partial histories $b_{t,k}$ by solving the following minimization problem:

$$\min_{h \in \mathbb{H}, g \in \mathbb{G}} \sum_{t,k} ||g(\mathbf{b}_{t,k}) - \mathbf{z}_{t,k}||_2^2 \tag{5.22}$$

Where:

- $\mathbb{H}$ and $\mathbb{G}$ are two hypothesis spaces induced by neural networks.
- g is a feed-forward neural network
- $h = (\mathbf{h}_f, \mathbf{h}_p)$ are the two RNNs that compute $\mathbf{B}_t$ and $\mathbf{B}_{t,k}$

Reverse prediction task predicts the compressed histories $b_{t,k}$ from the latent embedded observations z_{t+k}.

$$\min_{f \in \mathbb{F}, g' \in \mathbb{G}'} \sum_t ||g'(f(\mathbf{o}_t)) - \mathbf{b}_t||_2^2 \tag{5.23}$$

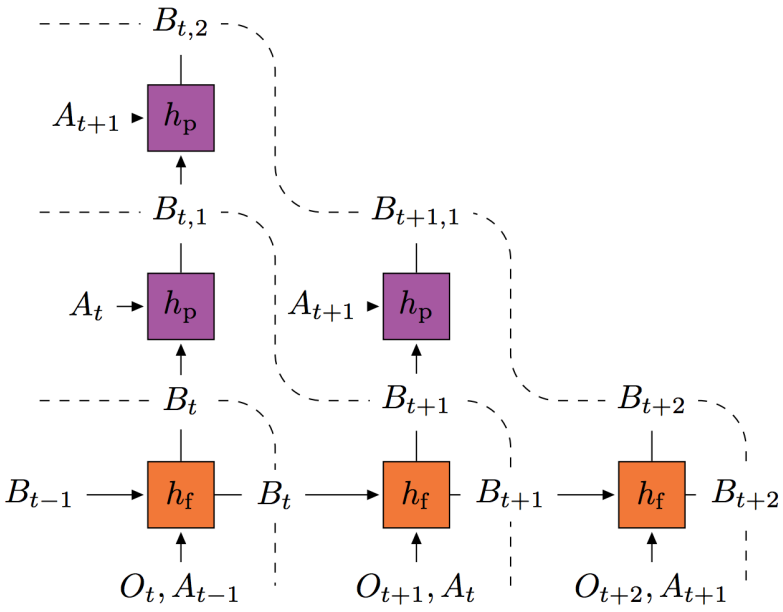


Figure 5.13: PBL History Encoder

Where:

- $f(\mathbf{o}_t) = \mathbf{z}_t$
- $\mathbb{F}$ and $\mathbb{G}'$ are two hypothesis spaces induced by neural networks.
- f and g' are two other feed-forward neural networks.

Together the forward and reverse cycle form a bootstrap effect that allows bootstrap useful information from far into the future.

5.4.4 GID and ADP

GID and ADP where introduced by the authors of [27]. Both auxiliary tasks predict actions taken between two observations k frames apart. Besides the k actions $\mathbf{a}_{[t:t+k]}$ the auxiliary tasks take in input the belief of the first frame $\mathbf{h}_t$ and the visual encoding φ_{t+k} .
ADP uses a 2-layer feed forward neural network to model an action distribution. It's loss consists in evaluating the KL-divergence between the network prediction and k actions.

$$L_{\text{ADP}} = \text{KL}(\text{MLP}(\mathbf{h}_t, \varphi_{t+k}), \mathbf{a}_{[t:t+k]}) \tag{5.24}$$

GID generates the first prediction by using a linear layer f.

$$g_t^{\text{GID}} = f(\mathbf{h}_t, \varphi_{t+k}) \tag{5.25}$$

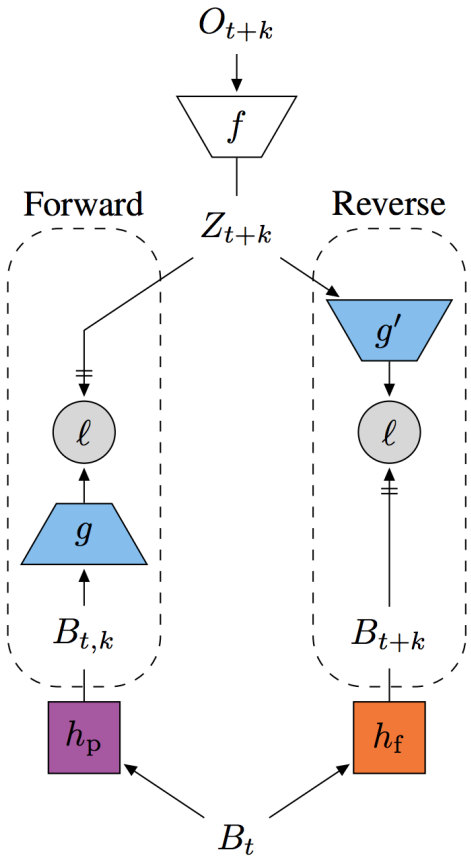


Figure 5.14: PBL architecture

Then the remaining actions are individually predicted by using a GRU that takes in input the previous action and predicted action.

$$g_{t+i}^{\text{GID}} = GRU(\mathbf{a}_{t+i-1}, g_{t+i-1}^{\text{GID}}) \tag{5.26}$$

The GID loss is calculated by measuring the Cross Entropy between all the actions and predicted actions.

$$L_{\text{GID}} = \sum_{i=1}^{k-1} \text{CrossEnt}(f'(\mathbf{g}_{t+i}^{\text{GID}}), a_{t+i}) \tag{5.27}$$

Where f' is another linear layer.

6 Results

The following section will firstly give a brief introduction of the standard metrics used for the evaluation of the experiments, then all the experiments are going to be described in detail. Finally the obtained results are given.

6.1 Standard Metrics

In total four standard metrics are used during evaluation: Success Rate (SR), Path Length (PL), Success weighted by Path Lenght (SPL) [31] and Normalized Dynamic Time Warping (nDTW) [32].

Success Rate Success Rate is the most obvious way of evaluating agents inside goal oriented tasks. SR is obtained by calculating the percentage of episodes in which the agent manages to reach the goal. This metric alone however isn't enough to model the performances of the agent inside path oriented tasks.

$$\text{SR} = \frac{1}{N}\sum_{i=1}^{N} S_i \tag{6.1}$$

Where $S_i \in 0, 1$ is a binary variable which is equal to 1 if the agent successfully reached the goal during epoch i and 0 otherwise.

Path Length Path Length measures the total length of the predicted path, the optimal value should be equal to the length of the reference path.

Success weighted by Path Length Distance from the goal alone isn't a good indicator to measure for proximity since it doesn't take into account the structure of the environment. For example the agent could be close to the goal in terms of distance while beeing separated by a wall.
Success weighted by Path Length is a metric which tries to summarize into a single value the ability of the agent to reach it's goal while pursuing the shortest possible path. SPL is defined as:

$$\text{SPL} = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{max(p_i, l_i)} \tag{6.2}$$

Where:

- N is the number of test episodes
- l_i is the shortest path distance from the agent position to the goal at episode i.
- p_i the length of the path actually taken by the agent at episode i
- $S_i \in 0, 1$ is a binary variable which is equal to 1 if the agent successfully reached the goal during epoch i and 0 otherwise.

SPL is a very stringent measure, most of the times an agent capable of reaching an SPL value of 0.5 is considered as a good result. To give some context an SPL of 0.5 can be reached by either reaching the goal in only 50% of the episodes while taking the optimal path in all of them, or by reaching the goal in all of the episodes while taking a sub-optimal path which is exactly twice as long as the optimal path.

Normalized Dynamic Time Warping Dynamic Time Warping (DTW) is a similarity function between time-series which has been used very frequently in many fields like speech processing, robotics or data-mining.
Given two time series R (reference) and Q (query) who belong to some feature space $\mathcal{F}$, Dynamic Time Warping finds the optimal ordered alignment between the two series by minimizing the following function:

$$\text{DTW}(R, Q) = \min_{w \in W} \sum_{(i_k, j_k) \in W} \partial(r_{i_k}, q_{j_k}) \tag{6.3}$$

Where:

- $\partial : \mathcal{F} \times \mathcal{F} \to \mathbb{R}^+$ is a distance function mapping pairs of elements from the two series to a real non-negative number.
- $W = w_1, ..., w_{|W|}$ is a warping where $w_k = (i_k, j_k) \in [1 : |R|] \times [1 : |Q|]$

Moreover $W = w_1, ..., w_{|W|}$ to be considered valid must respect the following conditions:

- $w_{k+1} - w_k \in \{(1,1), (1,0), (0,1)\}$
- $w_1 = (1,1)$
- $w_|W| = (|R|, |Q|)$

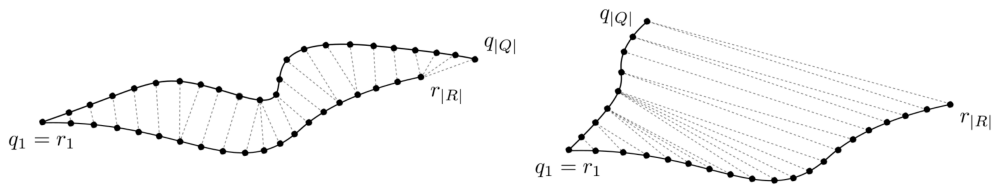


Figure 6.1: Two time series R and Q and the optimal warping between them computed trough DTW

In the context of navigation usually $\mathcal{F}$ consists in the set of navigable points and $\partial(r, q)$ returns the length of the shortest path between the two points r, q. In some contexts the euclidian distance can be used as ∂.
The main problem of DTW in the context of navigation is that it isn't invariant to the scale and density of nodes. Normalized Dynamic Time Warping (nDTW) is an adjusted version of DTW which takes into consideration the previously mentioned problem.

$$\text{nDTW}(R, Q) = exp(-\frac{\text{DTW}(R, Q)}{|R|d_{th}}) \tag{6.4}$$

The calculated DTW is normalized by $\frac{1}{|R|d_{th}}$ where d_{th} is a sampling rate invariant threshold distance defined for measuring success. Moreover to make the results in an interval between 0 and 1 the negative exponential is taken. The closer the score is to 1 the better is the result.
Overall the nDTW metric possess following positive properties:

- Softly penalizes deviations.
- It naturally models the importance of the goal by forcing the alignment of the two sequences at their beginning and end.
- Insensitive in changes of scale and density of nodes.
- Sensitive to the order of nodes in the sequences.
- The exact score can be computed in a quadratic time and approximation can be computed in linear time.

6.2 Hardware Limitations

The VLN-CE tasks in which this project falls is characterized by an high computational complexity and especially requires a GPU with many gigabytes of memory capable of containing the elevated amount of 3D meshes that compose the Matterport3D environments.

My personal computer equipped with an NVIDIA 980 with 4GB of memory couldn't handle the training at all. Just starting the process made it instantly crash.
Even the PC gave at my disposal by my Co-Supervisor, despite being equipped with a powerful NVIDIA 2080 GPU with 8GB of memory, was barely enough to handle it and several days were required to complete even few epochs of training.
The 8GB of memory of the GPU card are barely enough to train the agent with a batch size of one. Higher batches weren't feasible but could have heavily influenced the training performances. For the same reasons instantiating simultaneously multiple environments to speed up training wasn't possible and the long training times made it hard to train for high amounts of epochs.

Unfortunately hardware specifications are not mentioned explicitly anywhere in [18], so comparisons between the two setups used to run the experiments cannot be made. Krantz et al only mention that they trained their models until convergence for up to 30 epochs and with a batch size of 5 full trajectories. So it's a safe assumption that they had at disposal a much more powerful setup than the one at our disposal.

6.3 Experiments

Table 6.1 displays the results obtained through the different experiments. We evaluate the experiments by using the standard metrics described in Section 6.1. We consider nDTW as the main metric followed by SPL.

RxR-Habitat-Team The first row in Table 6.1 contains the results of the baseline model [16] provided by Jacob Krantz in the leaderboard of the RxR-Habitat Challenge [15].
We had to use those values as a baseline reference since no other valid baseline could be found in other papers. The results found in [18] are obtained by using a different dataset for the trajectories, since [18] uses a version of the R2R dataset converted into a continuos environment while the baseline model uses a version of the RxR dataset that followed the same conversion process.
On the contrary the experiments carried out in [14] are done inside a nav-graph environment instead of a continuos one.

Starter-Code Checkpoint shows the performances of the base model with it's pre-trained weights and without any additional training or fine-tuning.

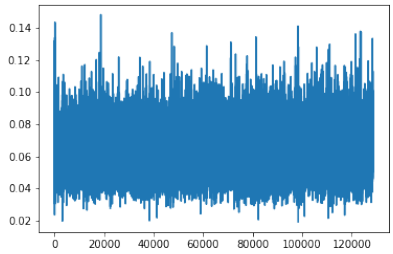
roBERTa no fine tuning The VLN-CE model is trained over the instructions encodings produced by a RoBERTa model which used the default pre-trained weights and received no additional fine-tuning.

roBERTa with fine tuning On the contrary in this experiment we firstly fine-tune the RoBERTa model on the entire corpus of instructions of the RxR dataset for a couple of days. Then the fine-tuned model is used to produce the instruction's encodings. Finally our VLN-CE model is trained over the produced encodings.

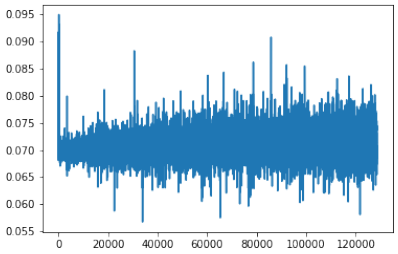
Training with Auxiliary Tasks In this experiment the we trained the augmented with the auxiliary tasks describe in [27]. In Table 6.1 we can see how this allowed us to achieve the highest nDTW out of all the experiments, however both the SPL and the success rate suffered a great decrease. The introduction of additional losses made the agent less optimized at reaching the goal location.

AuxTasks + RoBERTa In the last experiment we tried to combine both the auxiliary tasks and the RoBERTa's encodings into the same model to check how they perform toghether. With this model we obtained the highest nDTW out of all the experiments carried out during the thesis work. Moreover we can also see how the RoBERTa encoding compensated for the great decrease of the SPL and Success Rate values by bringing them back to the values obtained through the other experiments.
We tracked the values of each single auxiliary loss during training. In Figure 6.2 we can see that at the end of the training most losses managed to decrease significantly. The only loss who didn't undergo any significant decrease was ADP. Most likely this auxiliary loss is not beneficial toward our task. An additional experiment could be carried out by trying to remove or substitute this auxiliary loss and see if the performances increase.

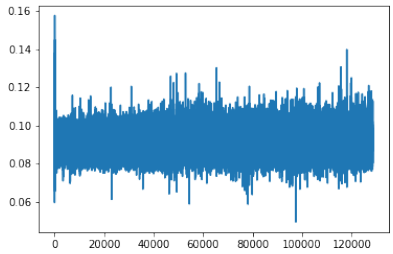
The results of experiments are heavily conditioned by the limited resources at disposal. Each model was trained for only five epochs and with a batch size of 1 which heavily conditioned the training performances.



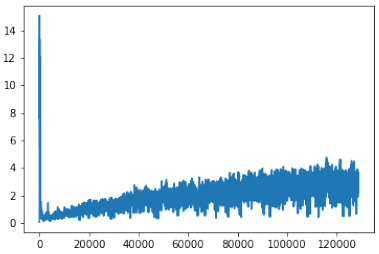
(a) ADP



(b) CPCA



(c) GID



(d) PBL

Figure 6.2: Plots of the losses of the four different Auxiliary Tasks

	nDTW	SPL	Path Length	Success Rate
RxR-Habitat-Team	30.86	**11.96**	7.33	**13.93**
Starter-Code Checkpoint	25.46	6.84	0.64	6.84
roBERTa no fine tuning	35.39	6.15	7.34	7.67
roBERTa with fine tuning	35.56	7.83	7.16	9.00
Training with AuxTasks	36.84	4.95	5.36	5.72
AuxTasks + Roberta	**37.48**	7.24	6.64	8.61

Table 6.1: Experiments results

7

Future Works and Conclusions

7.1 Future Works

Due the high amount of time required to train and evaluate a model, the amount of experiments that have been carried out is limited. Therefore there is still a lot of room for experimentation and the improvements that can be made over the original model.

Like we already sayd in Chapter 6 we found out, by examing the values of the auxiliary losses at the end of the training, that the ADP was responding very poorly to our VLN-CE task. For starting another additional experiment could consist in removing such loss and see if it results in improvements in the agent performances.
Furthermore up to now we only limited ourselves in utilizing the auxiliary losses described by Joel Ye et al. in [27]. Another possible experiment could consist in developing a custom auxiliary task that could be implemented in substitution to the previously removed pbl task.
For example a simple auxiliary task which could be very effective for this problem could take in input current coordinates of the agent inside the environment and the path taken up to the current step and predict the length of the remaining path that has to be followed to reach the goal.

Moreover the instructions representation could be further down improved by using ViLBERT the model proposed by Jiasen Lu, et al in [34] which combines both the visual observations and the natural language instructions into a single representation providing the agent with much more context.

7.2 Conclusions

VLN-CE is a very complex and challenging task that involves many concepts coming from many different fields of Artificial Intelligence like Embodied AI, Computer Vision or Natural Language Processing.
The elevated computational complexity that characterizes the VLN-CE tasks and the limited resources available however made experiments really long and very complex to handle. The limited resources available made experiments time consuming to carry out. Training a model for a reasonable amount of epochs required at least a full week and another couple of days just to evaluate it.
The results obtained are heavily conditioned by these premises, nonetheless this project gave me the opportunity to broaden my knowledge about Computational Intelligence by exploring a field I have never worked on and learn about the many state of the art methods, models and algorithms used inside it.
Moreover working along with professional researchers and learning about the process followed to study and find new state of the art methods was very educational.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] P. N. Stuart Russel, *Artificial Intelligence a Modern Approach, Third Edition*. Pearson, 2016.

[3] R. Szeliski, *Computer Vision, Algorithms and Applications*. Springer, 2011.

[4] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, "A survey of embodied ai: From simulators to research tasks," in *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2022. [Online]. Available: https://arxiv.org/abs/2103.04918

[5] H. Durrant-Whyte, I. Fellow, and T. Bailey, "Simultaneous localisation and mapping (slam): Part i the essential algorithms," 2009. [Online]. Available: https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf

[6] D. S. Chaplot, D. Gandhi, S. Gupta, A. Gupta, and R. Salakhutdinov, "Learning to explore using active neural slam," in *ICLR-2020*, 2020. [Online]. Available: https://arxiv.org/abs/2004.05155

[7] "Natural language definition." [Online]. Available: https://en.wikipedia.org/wiki/Natural_language

[8] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Nießner, M. Savva, S. Song, A. Zeng, and Y. Zhang, "Matterport3d: Learning from rgb-d data in indoor environments," 2017. [Online]. Available: https://arxiv.org/abs/1709.06158

[9] C. Shyalika, "A beginners guide to q-learning," 2019. [Online]. Available: https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c

[10] Manolis Savva*, Abhishek Kadian*, Oleksandr Maksymets*, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra, "Habitat: A Platform for Embodied AI Research," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019. [Online]. Available: https://arxiv.org/abs/1904.01201

[11] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. Chaplot, O. Maksymets, A. Gokaslan, V. Vondrus, S. Dharur, F. Meier, W. Galuba, A. Chang, Z. Kira, V. Koltun, J. Malik, M. Savva, and D. Batra, "Habitat 2.0: Training home assistants to rearrange their habitat," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021. [Online]. Available: https://arxiv.org/abs/2106.14405

[12] "aihabitat.org." [Online]. Available: https://aihabitat.org/

[13] P. Anderson, Q. Wu, D. Teney, J. Bruce, M. Johnson, N. Sünderhauf, I. Reid, S. Gould, and A. van den Hengel, "Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. [Online]. Available: https://arxiv.org/abs/1711.07280

[14] A. Ku, P. Anderson, R. Patel, E. Ie, and J. Baldridge, "Room-Across-Room: Multilingual vision-and-language navigation with dense spatiotemporal grounding," in *Conference on Empirical Methods for Natural Language Processing (EMNLP)*, 2020. [Online]. Available: https://arxiv.org/abs/2010.07954

[15] "Rxr challenge." [Online]. Available: https://ai.google.com/research/rxr/

[16] "Rxr-habitat competition starter code." [Online]. Available: https://github.com/jacobkrantz/VLN-CE/tree/rxr-habitat-challenge

[17] "Semantic segmentation." [Online]. Available: https://ai.stanford.edu/~syyeung/cvweb/tutorial3.html

[18] J. Krantz, E. Wijmans, A. Majumdar, D. Batra, and S. Lee, "Beyond the nav-graph: Vision-and-language navigation in continuous environments," 2020. [Online]. Available: https://arxiv.org/abs/2004.02857

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[20] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," 2014. [Online]. Available: https://nlp.stanford.edu/pubs/glove.pdf

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Tech report*, 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[22] J. Devlin, M.-W. Chang, Kenton, and L. K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[23] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[24] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016. [Online]. Available: https://arxiv.org/abs/1609.08144

[25] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *ACL 2016*, 2016. [Online]. Available: https://arxiv.org/abs/1508.07909

[26] J. Ye, D. Batra, E. Wijmans, and A. Das, "Auxiliary tasks speed up learning pointgoal navigation," 2020. [Online]. Available: https://arxiv.org/abs/2007.04561

[27] J. Ye, D. Batra, A. Das, and E. Wijmans, "Auxiliary tasks and exploration enable objectnav," 2021. [Online]. Available: https://arxiv.org/abs/2104.04112

[28] O. V. Aaron van den Oord, Yazhe Li, "Representation learning with contrastive predictive coding," 2019. [Online]. Available: https://arxiv.org/abs/1807.03748

[29] Z. D. Guo, M. G. Azar, B. Piot, B. A. Pires, and R. Munos, "Neural predictive belief representations," 2019. [Online]. Available: https://arxiv.org/abs/1811.06407

[30] D. Guo, B. A. Pires, B. Piot, J. bastien Grill, F. Altché, R. Munos, and M. G. Azar, "Bootstrap latent-predictive representations for multitask reinforcement learning," 2020. [Online]. Available: https://arxiv.org/abs/2004.14646

[31] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, and A. R. Zamir, "On evaluation of embodied navigation agents," in *Report of a working group on empirical methodology in navigation research*, 2018. [Online]. Available: https://arxiv.org/abs/1807.06757

[32] G. Ilharco, V. Jain, A. Ku, E. Ie, and J. Baldridge, "General evaluation for instruction conditioned navigation using dynamic time warping," in *Thirty-third Conference on Neural Information Processing Systems (NeurIPS 2019)*, 2019. [Online]. Available: https://arxiv.org/abs/1907.05446

[33] J. Gu, E. Stefani, Q. Wu, J. Thomason, and X. E. Wang, "Vision-and-language navigation: A survey of tasks, methods, and future directions," 2022. [Online]. Available: https://arxiv.org/abs/2203.12667

[34] J. Lu, D. Batra, D. Parikh, and S. Lee, "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks," 2019. [Online]. Available: https://arxiv.org/abs/1908.02265

A

Code

Listing A.1: Insert code directly in your document

```
import json
import os
import torch
import numpy as np

roberta = torch.hub.load('pytorch/fairseq', 'roberta.large')
splits=os.listdir("json")

for split in splits:
    print("LOADING SPLIT "+split)
    file_name="./json/"+split
    with open(file_name) as f:
        data = json.load(f)
    i=0
    for episode in data["episodes"]:
        print("Loading split "+str(i)+" of "+str(len(data["
            episodes"])))
        i=i+1
```

```
18          instruction_text=episode["instruction"]["
                instruction_text"]
19          language=episode["instruction"]["language"]
20          instruction_id=episode["instruction"]["instruction_id"]
21
22          embedding_file_name="text_features.npz"
23          language_name_component=""
24          if "en" in language:
25              language_name_component="en"
26          if "hi" in language:
27              language_name_component="hi"
28          if "te" in language:
29              language_name_component="te"
30          embedding_file_name=language_name_component+"_"+
                embedding_file_name
31          id_name_component=instruction_id
32          while len(id_name_component)<6:
33              id_name_component="0"+id_name_component
34          embedding_file_name=id_name_component+"_"+
                embedding_file_name
35          embedding_file_name="text_features/rxr_"+split.replace
                ("_guide.json","")+"/"+embedding_file_name
36          if not os.path.isfile(embedding_file_name):
37              tokens = roberta.encode(instruction_text)
38              if len(tokens)>512:
39                  tokens=tokens[:512]
40              last_layer_features = roberta.extract_features(
                    tokens)
41              instruction_encoding=last_layer_features.detach().
                    numpy()[0,:,:]
42              tokens_string = list(map(str, tokens.numpy()))
43              np.savez(embedding_file_name, tokens=tokens_string,
                    features=instruction_encoding)
```

B

Hyperparameters

B.1 Common Task Parameters

Name	Description	Value
lr	Learning Rate, scalar multiplied to gradient which regulates the gradient step	2.5^{-4}
batch_size	Number of examples over which the gradient is calculated	1
epochs	Maximum number of epochs that can be reached during training	15

Table B.1: Common task parameters

B.2 CPCA Auxiliary Task Parameters

Name	Description	Value
loss_factor	Determines the impact of aux loss over the original loss	0.5
num_steps	Number of predicted steps	4
subsample_rate	Percentage of examples picked by the aux loss at each step	0.2
dropout		0.0

Table B.2: CPC|A auxiliary task parameters

B.3 PBL Auxiliary Task Parameters

Name	Description	Value
loss_factor	Determines the impact of aux loss over the original loss	0.2
num_steps	Number of predicted steps	4
subsample_rate	Percentage of examples picked by the aux loss	0.15

Table B.3: PBL auxiliary task parameters

B.4 GID Auxiliary Task Parameters

Name	Description	Value
loss_factor	Determines the impact of aux loss over the original loss	0.2
num_steps	Number of predicted steps	4
subsample_rate	Percentage of examples picked by the aux loss	0.2

Table B.4: GID auxiliary task parameters

B.5 ADP Auxiliary Task Parameters

Name	Description	Value
loss_factor	Determines the impact of aux loss over the original loss	0.2
num_steps	Number of predicted steps	4
subsample_rate	Percentage of examples picked by the aux loss	0.2

Table B.5: ADP auxiliary task parameters