Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Trabajo práctico I

Introducción a la Inteligencia Artificial

Autores:
Arroyo, Joaquín
Bolzan, Francisco
Montoro, Emiliano

23 de marzo de 2023

Índice

1. Introducción

El proyecto seleccionado para la investigación es *LM-Nav: Robotic Navigation with Large Pre-trained Models of Language, Vision, and Action* [1], el cuál es desarrollado por investigadores que pertenecen a alguno/s de los siguientes grupos:

- Universidad de Berkeley
- Universidad de Varsovia
- Google Research

1.1. Planteo del problema

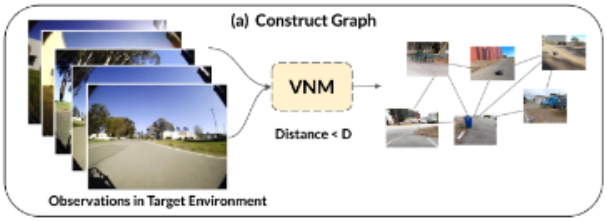
Dada una instrucción en lenguaje natural, para navegar un ambiente del mundo real, ¿cómo puede un robot seguirlas solamente a partir de observaciones visuales egocéntricas?

1.2. Idea principal

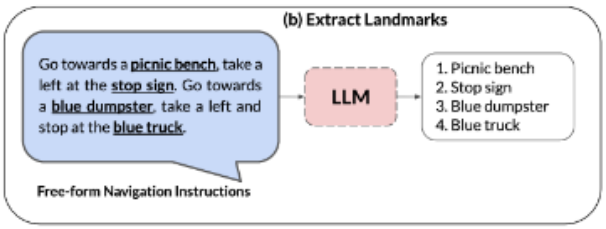
La idea principal del proyecto, es utilizar modelos pre entrenados de procesamiento de imágenes y lenguajes, para proveer una interfaz textual a un modelo de navegación.
Dentro de los modelos, utiliza los siguientes tres

de la siguiente manera:

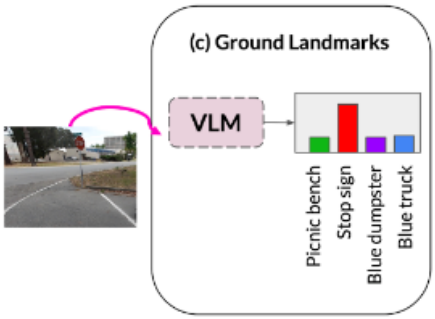
- Dado un conjunto de observaciones del ambiente, la función de distancia condicionada por la meta (esta forma parte de **VNM**) es usada para inferir conectividad entre dichas observaciones, para así crear el grafo topológico de conectividad del ambiente.



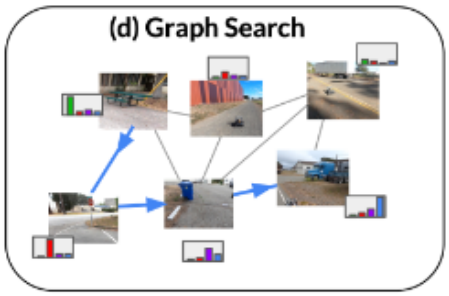
- **LLM** es usado para parsear instrucciones en lenguaje natural, a una secuencia de puntos de referencia que pueden servir como sub-metas intermediarias para la navegación.



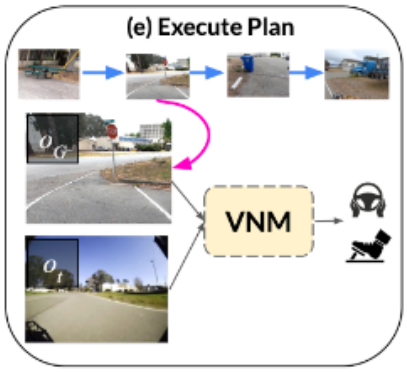
- **VLM** es usado para transformar las observaciones visuales del robot en puntos de referencia. Dicho modelo infiere una distribución de probabilidad conjunta sobre las descripciones de puntos de referencia y las imágenes (que forman nodos en el grafo anterior).



- Usando la distribución de probabilidades generada por **VLM** y el grafo de conectividad de **VNM**, un algoritmo de búsqueda es utilizado para generar un plan óptimo en el ambiente tal que **(i)** satisface la instrucción recibida, y **(ii)** es el camino más corto del grafo que realiza **(i)**.



- El plan anterior es ejecutado junto a la política (conjunto de reglas) condicionada por la meta, que forma parte de **VNM**.



1.3. Motivación

La motivación principal de este proyecto es lograr resultados que superen a la suma de sus partes, donde los modelos antes mencionados ya fueron utilizados (ya sea por su cuenta o en combinación con otros) con objetivos de navegación robótica autónoma pero presentando diversos problemas.

Acercamientos anteriores a la navegación mediante lenguaje natural (**VLN**) dependían de conjuntos de datos pre-procesados a modo de realizar una clasificación previa de los posibles puntos de interés de forma manual, lo cual impide la escalabilidad de dicha solución. A su vez, la falta de un modelo dedicado al lenguaje hace que las instrucciones deban ser de bajo nivel y ajustarse a las capacidades del robot, quitando así versatilidad.

Mediante la combinación de los modelos previamente vistos obtenemos una solución mas robusta, versatil y expandible que no depende de datos manipulados ni comandos refinados.

2. Tecnologías

2.1. LLM

Los modelos de lenguaje (**LLM**) se utilizan para el procesamiento del lenguaje natural. Estos son modelos generativos basados en la arquitectura Transformer, entrenados en corpus de textos de Internet masivos. LM-Nav utiliza GPT-3 [2] para parsear una instrucción en lenguaje natural en una secuencia de puntos de referencia, la cual es luego utilizada por **VLM**.

2.1.1. Transformers vs otras soluciones

En comparación con otras soluciones utilizadas para el procesamiento de lenguaje natural, como las redes neuronales recurrentes (RNN) y las redes neuronales convolucionales (CNN), los Transformers ofrecen varias ventajas clave.

Son capaces de procesar la información de entrada en paralelo.

Pueden capturar dependencias de largo alcance entre los elementos de la secuencia, lo que les permite modelar relaciones más complejas.

Son más fáciles de entrenar debido a su estructura modular

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

En el cuadro se presenta la comparación entre modelos que utilizan CNN o RNN contra dos modelos que utilizan Transformer. La comparación se da sobre el problema de trauducción automática, y se puede notar como ambos modelos con arquitectura Transformer, tienen menos costo de entrenamiento y obtienen mejores resultados.

2.1.2. Modelo y arquitectura

La idea detrás de la arquitectura Transformer es que, en lugar de procesar secuencias de palabras de forma secuencial como lo hacen las redes neuronales recurrentes, esta arquitectura utiliza la atención múltiple para enfocarse en diferentes partes de la secuencia de entrada en paralelo. Esto permite que el modelo procese de una forma mas eficiente secuencias mas largas de texto.

En este caso particular, GPT-3 utiliza una arquitectura Transformer muy similar a la que utiliza GPT-2 [3], incluyendo la **(i) inicialización modificada**, la **(ii) pre-normalización** y la **(iii) tokenización reversible**, con la excepción de que utiliza **(iv) patrones de atención densos y localmente dispersos** alternados en las capas del Transformer, similares a los que utiliza **Sparse Transformer** [2].

(i) La idea detrás de la **inicialización modificada** es que los pesos de la red se inicializan con valores aleatorios pero con una varianza específica para que los datos fluyan de manera más eficiente durante el entrenamiento. Esto ayuda a mejorar la convergencia del modelo durante el entrenamiento y, en última instancia, ayuda a mejorar su rendimiento.

(ii) La **pre-normalización** es una técnica utilizada para normalizar los vectores de entrada antes de que se procesen en la red neuronal. En el contexto de GPT-2 y GPT-3, esto significa que se normalizan los vectores de embedding de cada token de entrada antes de que se procesen en la red. Esta técnica ayuda a estabilizar el proceso de entrenamiento y reduce el riesgo de problemas de gradiente explosivo o desvaneciente, que pueden ocurrir durante el entrenamiento de modelos profundos.

(iii) La **tokenización reversible** es una técnica utilizada para descomponer el texto en tokens que puedan ser procesados por el modelo de lenguaje. En el contexto de GPT-2 y GPT-3, esto significa que se convierte el texto en una secuencia de tokens que representan las palabras, símbolos y caracteres en el texto. La tokenización reversible es importante porque permite que el texto se reconstruya a partir de los tokens generados por el modelo, lo que es útil para tareas como la generación de texto y la traducción de idiomas.

(iv) La **atención múltiple** es una parte fundamental de la arquitectura Transformer. Esta es una técnica que permite a la red neuronal enfocarse en diferentes partes de la entrada en función de su relevancia para la tarea en cuestión. En este caso, GPT-3, como Sparse Transformer, utilizan, además de **patrones de atención densos**, **patrones de atención localmente dispersos** para mejorar la eficiencia y la escalabilidad del modelo. Estos dos patrones, son combinados en cada capa. Para hacer esto, se divide la entrada en bloques y aplica una atención densa solo dentro de cada bloque y una atención dispersa entre los bloques. Esto último se hace mediante el uso de una matriz de adyacencia dispersa que representa las conexiones entre los bloques. Al utilizar patrones de atención dispersos, se reduce la cantidad de cálculos necesarios para procesar la entrada, lo que lo hace más eficiente en términos de tiempo y memoria. También permite que el modelo pueda procesar secuencias más largas de texto.

2.1.3. Dataset de entrenamiento

El dataset **Common Crawl** se ha vuelto particularmente grande, con casi un trillón de palabras, pero se ha descubierto que las versiones sin filtrar o ligeramente filtradas de este dataset tienden a tener una calidad inferior que los conjuntos de datos más cuidadosamente seleccionados, por lo que GPT-3 utilizó, como uno de sus datasets, a un Common Crawl modificado [2], a partir de las siguientes tres medidas:

(i) Se aplicó un filtro a una versión del dataset basada en la similitud con una variedad de conjuntos de datos de referencia de alta calidad. Esto se hace para asegurar que el conjunto de datos utilizado contenga textos de alta calidad y relevancia, lo que puede ayudar a mejorar la precisión y el desempeño del modelo.

(ii) Se eliminaron duplicados a nivel de documento, tanto dentro como entre conjuntos de datos, para evitar redundancia y mantener la precisión del conjunto de validación como una medida efectiva de sobreajuste.

(iii) Se agregaron otros conjuntos de datos de referencia de alta calidad a la mezcla de entrenamiento para mejorar Common Crawl y aumentar su diversidad. Esto ayuda a los modelos a aprender a generalizar mejor en lugar de memorizar patrones específicos en un solo conjunto de datos.

Otros de los datasets que utilizó GPT-3 son: WebText, BooksCorpus y Wikipedia, además, también se entrenó en tareas específicas de lenguaje natural utilizando datasets de entrenamiento específicos para cada tarea, como el conjunto de datos de preguntas y respuestas de TriviaQA o el conjunto de datos de resúmenes de CNN/Daily Mail.

2.1.4. Proceso de entrenamiento

Para entrenar todas las versiones de GPT-3 [2], se utilizó el optimizador Adam. Se limitó la norma global del gradiente a 1.0, para evitar que el gradiente sea demasiado grande y cause problemas de entrenamiento, y se utilizó un descenso de coseno para la tasa de aprendizaje hasta el 10 porciento de su valor, durante 260 mil millones de tokens (después de 260 mil millones de tokens, el entrenamiento continúa al 10 porciento de la tasa de aprendizaje original), esto significa que a medida que el modelo aprende más, la tasa de aprendizaje disminuye para que los ajustes de los parámetros sean más precisos. Todas las versiones fueron entrenadas en GPUs V100 en una parte de un clúster de ancho de banda alto proporcionado por Microsoft.

Las versiones se entrenaron en **secuencias de ventana de contexto** de 2048 tokens completos, tomando múltiples documentos en una sola secuencia cuando los documentos tienen menos de 2048 tokens, con el fin de aumentar la eficiencia computacional.

Para entrenar las versiones más grandes de GPT-3, se utilizó una técnica llamada **model parallelism**, que divide el modelo en diferentes partes para ser entrenadas en diferentes dispositivos. En este caso, se utilizó una combinación de paralelismo de modelo dentro de cada multiplicación de matriz

y paralelismo de modelo a través de las capas de la red para evitar quedarse sin memoria durante el entrenamiento. Esto permitió que el modelo pudiera ser entrenado de manera eficiente utilizando múltiples dispositivos de procesamiento de forma simultánea.

2.2. VLM

Los Modelos Visión-Lenguaje (**VLM**) son modelos diseñados para analizar la relación entre imágenes y lenguaje natural. Estos modelos abarcan desde el etiquetado o clasificación de imágenes hasta la generación de nuevas imágenes que se ajusten a una descripción textual.

LM-Nav utiliza CLIP [4] (Contrastive Language-Image Pre-training), una red neuronal que presenta la ventaja de ser un modelo de clasificación *zero-shot*, es decir, que no requiere entrenamiento previo para clasificar nuevos datos, y que además es más robusto que muchos de sus homólogos debido a su pre-entrenamiento con datasets de gran tamaño de datos no etiquetados.

2.2.1. Modelo y arquitectura

CLIP incorpora una etapa de pre-entrenamiento que consiste en entrenar simultáneamente dos modelos de codificación: uno visual y otro de lenguaje. El propósito de esta etapa es permitir que ambos modelos codifiquen sus entradas en un campo común, lo que facilita el establecimiento de relaciones entre ambos medios. Para lograr este objetivo, se utilizan datasets de pares "(image, text)", que se codifican por ambos modelos. Luego, se cruzan las codificaciones buscando maximizar las similitudes y minimizar las diferencias (contrastive loss function). El modelo CLIP intenta entonces predecir aquellos pares que realmente ocurrieron.

Esta estrategia de pre-entrenamiento permite entrenar a ambos modelos con una gran cantidad de datos sin necesidad de manipularlos previamente. Cabe destacar que en esta etapa no se asocian las imágenes con palabras individuales de su texto correspondiente, sino con el texto completo. Este enfoque ofrece una mejora considerable en términos de eficiencia temporal sin comprometer la calidad del entrenamiento en sí. Es importante destacar que se utilizan modelos de codificación específicos, como ResNet-50, Vision Transformer y un transformador de lenguaje con modificaciones en su arquitectura, para lograr los mejores resultados en la codificación de los datos.

CLIP aduiere entonces la capacidad de predecir si una imagen y su correspondiente extracto de texto están relacionados. Esta funcionalidad puede ser aplicada a imágenes que no pertenecen a sus categorías conocidas (conocido como zero-shot) o, alternativamente, puede ser entrenado para clasificar nuevas categorías (conocido como few-shot learning).

2.2.2. CLIP en LM-Nav

En el caso de LM-Nav, CLIP es utilizado para asociar los sitios de interes generados por **LLM** con los nodos del grafo generado por **VNM**, estimando la probabilidad de que cada nodo en el grafo corresponda con los sitios de interés.

Entonces, dada una lista de los posibles sitios de interes y una imágen que representa un nodo del grafo, se aplica CLIP sobre dicha imágen y un texto de la forma *"This is a photo of a X"*, CLIP calcula entonces las probabilidades de similaridad; seguido de una operación softmax se obtiene entonces la probabilidad definitiva de que dicha imágen se asocie a alguno de los sitios de interés.

2.3. VNM

Los modelos de navegación visual (**VNM**) aprenden el comportamiento de navegación y las posibilidades de navegación directamente a partir de observaciones visuales, asociando imágenes y acciones a través del tiempo. Utilizamos el modelo **ViNG VNM**, un modelo condicionado a objetivos que predice las distancias temporales entre pares de imágenes y las acciones correspondientes a ejecutar. El VNM tiene dos propósitos:

(i) Dado un conjunto de observaciones en el entorno objetivo, las predicciones de distancia del VNM se pueden utilizar para construir un grafo topológico G(V, E) que representa un "mapa mental"del entorno.

(ii) Dado un "paseo", que consta de una secuencia de subobjetivos conectados a un nodo objetivo, el VNM puede guiar al robot a lo largo de este plan.

El grafo topológico G es una abstracción importante que permite una interfaz simple para planificar sobre la experiencia pasada en el entorno y ha sido utilizado con éxito en trabajos previos para realizar navegación a largo plazo. Para deducir la conectividad en G, utilizamos una combinación de estimaciones de distancia aprendidas, proximidad temporal (durante la recolección de datos) y proximidad espacial (utilizando mediciones de GPS). Para cada par conectado de vértices vi, vj, asignamos esta estimación de distancia al peso de borde correspondiente D(vi, vj).

2.3.1. Modelo y arquitectura

Se utiliza una combinación de estimaciones de distancia aprendidas (de VNM), proximidad espacial (de GPS) y proximidad temporal (durante la recolección de datos) para deducir la conectividad de las aristas. Si los registros de tiempo correspondientes de dos nodos están cercanos (< 2s), lo que sugiere que se capturaron en rápida sucesión, entonces los nodos correspondientes están conectados, agregando aristas que fueron físicamente atravesadas. Si las estimaciones VNM de las imágenes en dos nodos están cercanas, lo que sugiere que son alcanzables, entonces también se conectan los nodos correspondientes, agregando aristas entre nodos distantes a lo largo de la misma ruta y dando un mecanismo para conectar nodos que fueron recolectados en diferentes trayectorias o en diferentes momentos del día pero corresponden a ubicaciones cercanas. Para evitar casos de distancias subestimadas por el modelo debido a observaciones aisladas, por ejemplo, campos verdes abiertos o una pared blanca, filtramos las aristas prospectivas que están significativamente más lejos según sus estimaciones de GPS, por lo tanto, si dos nodos están cercanos según su GPS, por ejemplo, nodos en diferentes lados de una pared, pueden no estar desconectados si el VNM no estima una distancia pequeña; pero dos nodos de aspecto similar a 100 metros de distancia, que pueden estar frente a una pared blanca, pueden tener una pequeña estimación de VNM pero no se agregan al grafo para evitar agujeros de gusano.

2.4. Algoritmos para creación y búsqueda del grafo topológico

Para crear el grafo mencionado en **VNM** (Sec. 2.3) se utiliza el siguiente algoritmo:

Algorithm 2: Graph Building

Input: Nodes $n_i, n_j \in \mathcal{G}$ containing robot observations; **VNM** distance function f_d; hyperparameters $\{\tau, \epsilon, \eta\}$
Output: Boolean e_{ij} corresponding to the existence of edge in $\mathcal{G}$, and its weight
learned distance $D_{ij} = f_d(n_i[\text{'image'}], n_j[\text{'image'}])$
timestamp distance $T_{ij} = |n_i[\text{'timestamp'}] - n_j[\text{'timestamp'}]|$
spatial distance $X_{ij} = \|n_i[\text{'GPS'}] - n_j[\text{'GPS'}]\|$
if ($T_{ij} < \epsilon$) **then** return $\{\textit{True}, D_{ij}\}$
else if ($D_{ij} < \tau$) AND ($X_{ij} < \eta$) **then** return $\{\textit{True}, D_{ij}\}$
else return *False*

y para la generación del plan óptimo, a partir del grafo anterior, y de la distribución de probabilidades generada por **VLM** (Sec. 2.2.2) se utiliza el siguiente algoritmo:

Algorithm 1: Graph Search

Input: Landmarks $(\ell_1, \ell_2, \ldots, \ell_n)$.
Input: Graph $\mathcal{G}(V, E)$.
Input: Starting node S.
$\forall_{\substack{i=0,\ldots,n \\ v \in V}} \; Q[i, v] = -\infty$
$Q[0, S] = 0$
Dijkstra_algorithm($\mathcal{G}, Q[0, *]$)
for i in $1, 2, \ldots, n$ **do**
  $\forall v \in V Q[i, v] = Q[i-1, v] + \text{CLIP}(v, \ell_i)$
  Dijkstra_algorithm($\mathcal{G}, Q[i, *]$)
end for
destination $= \arg\max(Q[n, *])$
return backtrack(destination, $Q[n, *]$)

3. Conclusión

Pensamos que en su forma actual, LM-Nav proporciona un prototipo simple y atractivo de cómo los modelos pre-entrenados pueden combinarse para resolver tareas complejas, e ilustra que estos modelos pueden servir como una interfaz para controladores robóticos que se entrenan sin anotaciones de lenguaje. Aunque creemos que aún no está preparado para interactuar en un entorno con varios actores, reconocemos que enfatiza la importancia de reutilizar modelos para abordar problemas que de otra manera serían difíciles de resolver. La combinación de modelos puede ser una solución efectiva en situaciones en las que un único modelo no es suficiente.

Referencias

[1] B. Itcher D. Shah, B. Osinski and S. Levine. Lm-nav: Robotic navigation with large pre- trained models of language, vision, and action. 2022.

[2] N. Ryder M. Subbiah J. D. Kaplan P. Dhariwal A. Amodei T. Brown, B. Mann. Language models are few-shot learners. in advances in neural information processing systems. 2020.

[3] Rewon Child David Luan Dario Amodei Alec Radford, Jeffrey Wu and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

[4] C. Hallacy A. Ramesh G. Goh S. Agarwal G. Sastry A. Askell P. Mishkin J. Clark et al. A. Radford, J. W. Kim. Learning transferable visual models from natural language supervision. In International Conference on Machine Learning, 2021.