Facultad de Matemática, Astronomía, Física y Computación

Universidad Nacional de Córdoba

Usar la historia del diálogo para mejorar modelos de Visual Question Answering

Tésis

para obtener el título de

Licenciado en Ciencias de la Computación

Autor

Thomas Santiago Vadora

Director: Mauricio Diego Mazuecos Perez

Córdoba, Argentina 2022

DEDICATORIA

Dedicado a Samuel Vadora y Victoria Sasia por su apoyo durante todos estos años.

Agradecimientos

A mi director Mauricio Mazuecos, a mi co-directora Luciana Benotti, a Franco Luque y Jorge Sanchez. Todos ellos hicieron que esto sea posible con consejos, ideas, paciencia y trabajo duro. También a mis grandes amigos Federico Gonzalez Kriegel y Mariano Piatti por acompañarme durante toda la carrera.

Clasificación (ACM CCS 2012):

- Computing methodologies Natural Language Processing
- Computing methodologies Machine Learning
- Computing methodologies Artificial Intelligence

Palabras Clave:

- Deep Learning
- Artificial Intelligence
- Natural Language Processing

Resumen

En este trabajo presentamos algunas técnicas para poder crear modelos que utilicen el historial de una conversación para responder preguntas sobre una imagen. En particular desarrollamos sobre un modelo llamado Oráculo que debe responder preguntas sobre un objeto particular en una imagen en un contexto de un diálogo con otro jugador, el Preguntador. Dicho juego se llama GuessWhat!?. Es una tarea muy interesante ya que mezcla la visión y el lenguaje.

Overview

In this work we present some techniques to create models that use the history of a conversation to answer questions about an image. In particular we do all the research on a model called Oracle that must answer questions about a particular object in an image in the context of a dialogue with another player, the Questioner. This game is called GuessWhat!?. It is a very interesting task since it mixes vision and language.

Índice general

Capítulo 1

Introducción y Motivación

En esta tesis abordamos el problema de entrenar modelos de aprendizaje automático utilizando redes neuronales que sean capaces de responder preguntas sobre un objeto en una imagen. Pero en este trabajo nos concentramos en hacer que dichos modelos puedan responder cada turno en un dialogo teniendo en cuenta como se desenvolvió anteriormente el mismo ya que los trabajos de investigación para estos problemas mayormente no analizan la historia del diálogo. Para entrenar los modelos usamos un conjunto de datos llamado GuessWhat!? [32] basado en un juego multijugador en donde un Oráculo (o Oracle en inglés), al que se le asigna un objeto en una imagen, debe responder las preguntas (con Si, No, o No Aplica) que realiza otro jugador, el Preguntador (o Questioner en inglés), que debe realizar preguntas intentando adivinar cuál fue el objeto asignado (llamado referente, o *objecto target* en inglés). Por último un Adivinador (o Guesser en inglés), cuando el diálogo finaliza con las preguntas realizadas por el Preguntador y las respuestas dadas por el Oráculo debe adivinar el referente. Los diálogos en los que dicho jugador logra adivinar son considerados juegos exitosos. En este trabajo nos enfocamos en hacer que el modelo SOTA para el Oráculo [30] pueda responder mejor las preguntas que si dependen de las respuestas dadas en turnos anteriores del diálogo, ya que el trabajo realizado por la comunidad en este problema no estaba teniendo en cuenta la historia de los diálogos.

Modelar la historia para problemas de visión y lenguaje no es una tarea trivial

ya que los modelos deben aprender que turnos del diálogo ayudan a responder la pregunta actual y que otros solamente confunden. En esta tesis realizamos distintos experimentos en donde usamos redes neuronales recurrentes tomando como entrada de cada celda el vector de modalidad cruzada que genera el modelo de LXMERT aplicado para el Oráculo [30] para cada turno del diálogo, es decir cada par imagen y pregunta. LXMERT es un modelo basado en una arquitectura reciente llamada Transforemers y basicamente aprende a cread vectores en donde se tiene en cuenta no solo el texto de la pregunta sino tambien las relaciones de cada palabra con los objetos de la imagen y viceversa. En este trabajo demostramos que este enfoque no es suficiente para mejorar los modelos y que la mejora en las preguntas dependientes de la historia es muy pequeña.

La idea original que se persigue en este trabajo es guiar al modelo en el transcurso del diálogo a prestar atención a las partes de la imagen que el mismo diálogo hace foco. Como mencionamos anteriormente intentamos darle esa responsabilidad a redes recurrentes, estos experimentos nos hicieron ver que debiamos, quizás, probar algo mas explícito sobre la imagen para guiar al modelo. Por esto mismo con el grupo de Visión y Lenguaje de la Facultad de Matemática, Astronomía, Física, y Computación (FAMAF) continuamos una investigación que resulto en una publicación [19] para *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* que titulamos *Region under Discussion* (Región en Disucsión en castellano). De el mismo los autores son Mauricio Mazuecos, Jorge Sanchez, Franco Luque, Hernán Maina, y Luciana Benotti. En dicho trabajo le damos al modelo en cada turno la ubicación del referente respecto a una región que contiene los objetos candidatos, es decir que pueden ser el referente, y dichos objetos los obtenemos con una heuristica de emparejamiento que discutimos en detalle en el capítulo 6. Además concluimos que solo el 13 % del conjunto de datos son preguntas que dependen de la historia y damos ese conjunto que contiene solo dichas preguntas y llamamos GWhist. Por último creamos una herramienta para anotar este conjunto de datos.

La motivación de este trabajo final de licenciatura es poder hacer que modelos puedan aprender a interpretar el mundo en su totalidad, empezando de a poco como visión y lenguaje como hacemos en esta tesis. Si logramos hacer modelos robustos que entiendan de ambas modalidades podemos indirectamente ayudar a personas

que tienen dificultad ya sea para la visión o para el lenguaje. Si bien el Oráculo y el juego de GuessWhat!? no son aplicaciones directas para ayudar a personas con visión o movilidad reducida, crear avances en modelos y problemas similares puede permitir que luego otras investigaciones si culminen en ayudas directas. Por otro lado, nos resultaba interesante intentar resolver un problema que no muchas personas en la comunidad estaban intentando, hacer que los modelos que se usan en diálogo conozcan y usen a su favor la historia del mismo.

Capítulo 2

Descripción del problema y trabajo previo

En este capítulo introduciremos y explicaremos el problema con el que experimentamos en este trabajo, analizaremos el problema de forma conceptual y también estudiaremos el conjunto de datos con el que se trabajó en esta tesis. En la primera sección vamos a describir el paper que presenta oficialmente el problema y el conjunto de datos [32], luego hablaremos del conjunto de datos que utilizamos en todos los experimentos de este trabajo. En la tercera explicaremos brevemente los modelos que el mismo paper propone como modelos bases y por último revisaremos un poco de trabajo previo.

2.1. GuessWhat?! Descubrimiento visual de objetos a través de diálogo multi modal

Las personas usan el lenguaje natural como la forma más efectiva de comunicarse, incluso cuando se trata de describir el mundo visual que los rodea. A menudo sólo necesitan unas pocas palabras para referirse a un objeto específico en una escena compleja y con muchos elementos. Siempre que las expresiones apuntan inequívocamente a un objeto, hablamos de una expresión referente. Sin embargo, la identificación única del objeto en una imagen no siempre es posible, ya que depende del estado mental del oyente y del contexto de la escena. Muchas situaciones de la vida real, por lo tanto, requieren múltiples intercambios antes de que quede claro a qué objeto se hace referencia, un ejemplo que ilustra el problema podría ser:

- ¿Viste a ese perro?
* ¿Te refieres al que está en la esquina?
- No, el que está corriendo.
* Ah! Pero hay dos que corren, el de la izquierda o el de la derecha?
- Derecha
* Si, que pasa?

Un sistema de visión por computadora capaz de mantener conversaciones sobre lo que se vé sería un paso importante hacia la comprensión de escenas visuales. Tales sistemas serían más transparentes e interpretables porque los humanos pueden naturalmente interactuar con ellos, por ejemplo, haciendo preguntas aclaratorias sobre lo que perciben. Aún así, un desafío fundamental permanece: cómo crear modelos que entiendan las descripciones del lenguaje natural y las basen en el mundo visual. Gracias a los gran avances en el entrenamiento de redes neuronales [9] y la disponibilidad de conjuntos de datos de clasificación a gran escala [16, 21, 34], el reconocimiento automático de objetos ha alcanzado desempeño a nivel humano [LeCun et al, 2015]. Como resultado de esto, la atención en la investigación se ha desplazado hacia tareas que implican una comprensión de imágenes de alto nivel. Algunos de los ejemplos más comunes podrían ser Image Captioning [16] en donde dada una imagen se genera un texto descriptivo de la misma, VQA (Visual Question Answering) [3] en donde una red neuronal responde preguntas abiertas sobre una imagen, más cerca

a este trabajo es ReferIt [13] en donde se intenta generar descripciones de un objeto particular en una imágen. Por otro lado también hubo interesantes avances en sistemas de diálogos [22] en el área del procesamiento del lenguaje natural [5]. Uno de los problemas que tienen los sistemas de diálogo tradicionales es que no tienen una evaluación automatizada clara, es decir, no hay métricas de evaluación que se simule una evaluación humana [17] dado lo complejo que es el lenguaje natural. Por este motivo, una de las alternativas prometedoras son tareas de diálogo orientadas a un objetivo [15, 25, 33] donde hay agentes que conversan para cumplir un objetivo. Luego la tasa de éxito de los agentes se puede usar como una métrica automática. En este trabajo [32] unen las áreas de procesamiento del lenguaje natural y visión por computadoras en una de estas tareas de diálogo orientadas a un objetivo con un juego que ellos llaman "GuessWhat?!" ("Adivina Qué", en español). GuessWhat?! Es un juego cooperativo para dos jugadores en el que ambos jugadores ven una imagen con varios objetos. A un jugador, el oráculo, se le asigna un objeto en la imagen al principio del diálogo. Este objeto no es conocido por el otro jugador, el interrogador, cuyo objetivo es identificar el objeto oculto. Para hacerlo, el interrogador puede hacer una serie de preguntas que van a ser respondidas por el oráculo quien solo puede responder Sí, No, o No aplica (N/A). El interrogador no conoce la lista de objetos, lo único que puede ver es la imagen. Una vez que el interrogador ha reunido suficiente evidencia para localizar el objeto, notifica al oráculo de que están listos para adivinar el objeto. Luego revelamos la lista de objetos, y si el interrogador elige el objeto correcto, consideramos el juego exitoso. De lo contrario, el juego termina sin éxito. Algo interesante de este problema es justamente que está basado en diálogo y no solo en preguntas independientes, esto mismo lo hace más interesante y más complejo también. Para más claridad podemos ver el ejemplo de la Figura 2.1 y 2.2.

El problema a resolver tanto como para el oráculo como para el interrogador es complejo ya que cuando referenciamos objetos en una imagen tenemos que hacer uso de múltiples recursos como agrupar, localizar, entender colores, formas y relaciones entre objetos, además muchas de las cosas son relativas ya que por ejemplo un auto solemos pensarlo con un objeto grande, pero si lo compramos con un edificio, pasa a ser pequeño.

Interrogador	Oráculo
¿Es una persona?	No
¿Es un objeto que está siendo usado o sostenido?	Si
¿Es un snowboard?	Si
¿Es rojo?	No
¿Es el que lo está llevando la persona de azul ?	Si

Figura 2.1: Ejemplo del juego 168019. En esta imagen el objeto segmentado en verde es el referente.

Interrogador	Oráculo
¿Es una vaca?	Si
¿Es la vaca grande del medio?	No
¿Está la vaca a la izquierda?	No
¿A la derecha?	Si
¿Es la primera cerca de nosotros?	Si

Figura 2.2: Ejemplo del juego 203974. En esta imagen el objeto segmentado en verde es el referente.

2.2. El conjunto de datos

Para recolectar los datos utilizaron Amazon Mechanical Turk que es una herramienta de colaboración abierta distribuida en donde uno puede agregar tareas que no son directamente realizables por computadoras conocidas como HITs (Human Intelligence Tasks) y personas de todo el mundo pueden realizarlas y obtener dinero por ello. Crearon dos de estas tareas, una para el interrogador y otra para el Oráculo, y para asegurarse obtener datos de calidad agregaron distintas penalidades. Primero, los trabajadores tuvieron que pasar por una ronda de clasificación que consistió en completar con éxito 10 juegos y producir menos de 4 errores o desconexiones. Después de la calificación, los HITs continuaron consistiendo en un lote de 10 juegos exitosos. Incentivaron a los trabajadores a producir tantos diálogos exitosos seguidos proporcionando bonificaciones por hacer menos errores. En segundo lugar, los jugadores podían reportarse entre sí por lo que banearon a aquellos reportados múltiples veces por otros, de esta forma se incentivó a los jugadores a cooperar. Al final, solo se mantuvieron los diálogos de personas calificadas y los diálogos exitosos de la ronda de clasificación.

El conjunto de datos GuessWhat?! está compuesto por 155,280 diálogos que contienen 821,889 pares de (preguntas,respuestas) en 66,537 imágenes únicas y 134,073 objetos únicos. Las respuestas son respectivamente 52.2 % no, 45.6 % sí y 2.2 % N / A. En promedio, hay 5.2 preguntas por diálogo y 2.3 diálogos por imagen. Los diálogos contienen 3.986.192 palabras en total, que componen 11,465 palabras diferentes con al menos una ocurrencia y 5444 palabras con al menos 3 ocurrencias. Además, el 84,6 % de los diálogos son exitosos, el 8.4 % no y 7,0 % no se completan. Algunas de estas estadísticas se resumen en la Figura 2.3.

En la Figura 2.4 podemos ver el radio de diálogos según la cantidad de preguntas de cada uno.Se puede observar que la cantidad de preguntas disminuye exponencialmente ya que los humanos intentan terminar un juego con la menor cantidad de preguntas, pero crece desde cero hasta 3 ya que es difícil encontrar un objeto con una o dos preguntas únicamente. En la Figura 2.5 podemos encontrar un fenómeno interesante, observamos que la cantidad promedio de preguntas según la cantidad de objetos en la imagen crece como una función entre lineal y logarítmica, Una estrategia de preguntar es simplemente listar objetos (“¿Es la silla?”, “¿El control

	Todos	Finalizados	Exitosos
# Diálogos	155280	144434	131394
# Preguntas	821889	732081	648493
# Palabras	3986192	3540497	3125219
Tamaño del vocabulario	11465	10985	10469
Imágenes	66537	65112	62954
Objetos	134073	125349	114271

Figura 2.3: Tabla que describe el conjunto de datos.

remoto?", etc) que implicaría crecimiento lineal en la cantidad de preguntas mientras que una búsqueda binaria óptima implicaría crecimiento logarítmico. Aparentemente los humanos aplican una técnica que se encuentra en el medio de ellas. Por último en la Figura 2.6 podemos ver un gráfico de frecuencias de palabras en las preguntas del conjunto de datos. De la misma podemos concluir que muchas de las preguntas hacen referencias a locaciones (left/right), a colores (red/black/white) y a objetos (car/chair). En el apéndice de este documento se encuentran algunas estadísticas más sobre este conjunto.

2.3. Modelo base para el Oráculo que propusieron los creadores del conjunto de datos

En el trabajo de [32] proponen un modelo báscio tanto sea para el oráculo como para el preguntador. Como en este trabajo presentamos experimentos para el primero de ellos, nos enfocaremos en dicho modelo. Sería útil definir el problema un poco mas formal: tenemos una imagen $I \in R^{M*N}$ que contiene un conjunto de K objectos segmentados $O = \{O_1, ..., O_K\}$. Cada objecto O_i tiene asignado una categoria $c_i \in \{1, ..., C\}$ y una máscara de pixeles $s_i \in \{0, 1\}^{M*N}$ que especifica la ubicacion del objeto O_i en la imagen I y su forma. Por lo tanto, el juego consiste en una serie de pares pregunta respuesta $D = \{q_1, a_1, q_2, a_2, ..., q_j, a_j\}$ producidas por el interrogador y el oráculo respectivamente con $a_i \in \{Si, No, N/A\}$ para $1 \leq i \leq j$.

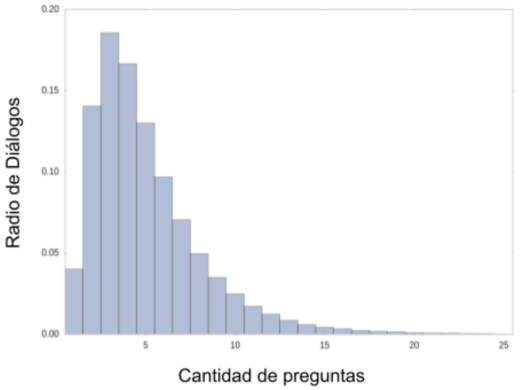


Figura 2.4: Porcentaje de diálogos en el dataset según la cantidad de preguntas.

Figura 2.5: Gráfico de frecuencias de palabras. Tamaños más grandes indican mayor frecuencia y viceversa.

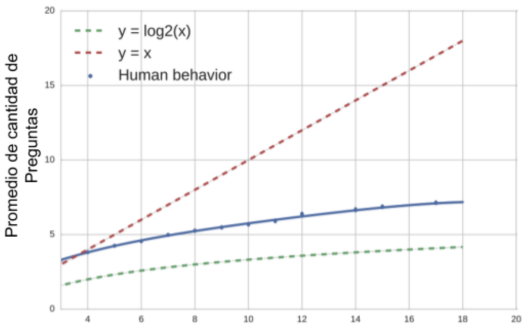


Figura 2.6: Se puede ver como la técnica de los humanos para realizar preguntas está entre medio de una función lineal y una logarítmica si comparamos cantidad de preguntas y cantidad de objetos en una imágen.

Luego el interrogador elgirá algun O_f y si es el objecto elegido por el oráculo al principio del juego (O_{target}) el mismo se considera exitoso. Para el oráculo tomaron una estrategia muy simple y común en modelos de redes neuronales, encodear cada entrada con algún modelo que obtiene una representación en forma de vector del input, luego concatenar todos ellos y agrega capas lineales sobre ese vector para obtener la predicción. Concretamente a la imagen I y el referente O_{target} se pasan por una red VGG16 [24] que es capaz de extraer features de las imágenes, es decir un vector que representa la imágen. La categoría se va a representar como un vector usando One Hot encoding, luego la información espacial de O_{target} la representan con un vector $X_{spatial} = [X_{min}, Y_{min}, X_{max}, Y_{max}, X_{centre}, Y_{centre}, W_{box}, H_{box}]$ donde X_{min}, Y_{min} representan el punto inferior izquierdo de un rectángulo sobre I, X_{max}, Y_{max} representan el punto inferior derecho, y X_{centre}, Y_{centre} representan el centro del mismo. W_{box}, H_{box} representan el ancho y el alto del rectángulo respectivamente (ambos normalizados entre -1 y 1). Por último, para representar la pregunta, se utiliza una red LSTM como indicamos en la Figura 2.7. Todas estas representaciones vectoriales se concatenan y son procesadas por un Perceptrón Multicapa para realizar las predicciones. Todos estos modelos y conceptos los explicaremos con detenimiento en el Capítulo 3 de esta tésis que es el marco teórico.

En lo que continua de este trabajo final intentaremos hacer un modelo para el oráculo que mejore los resultados del modelo base de [32] y que utilce no solo la pregunta actual para responder sino que todo el contexto del dialogo previo también.

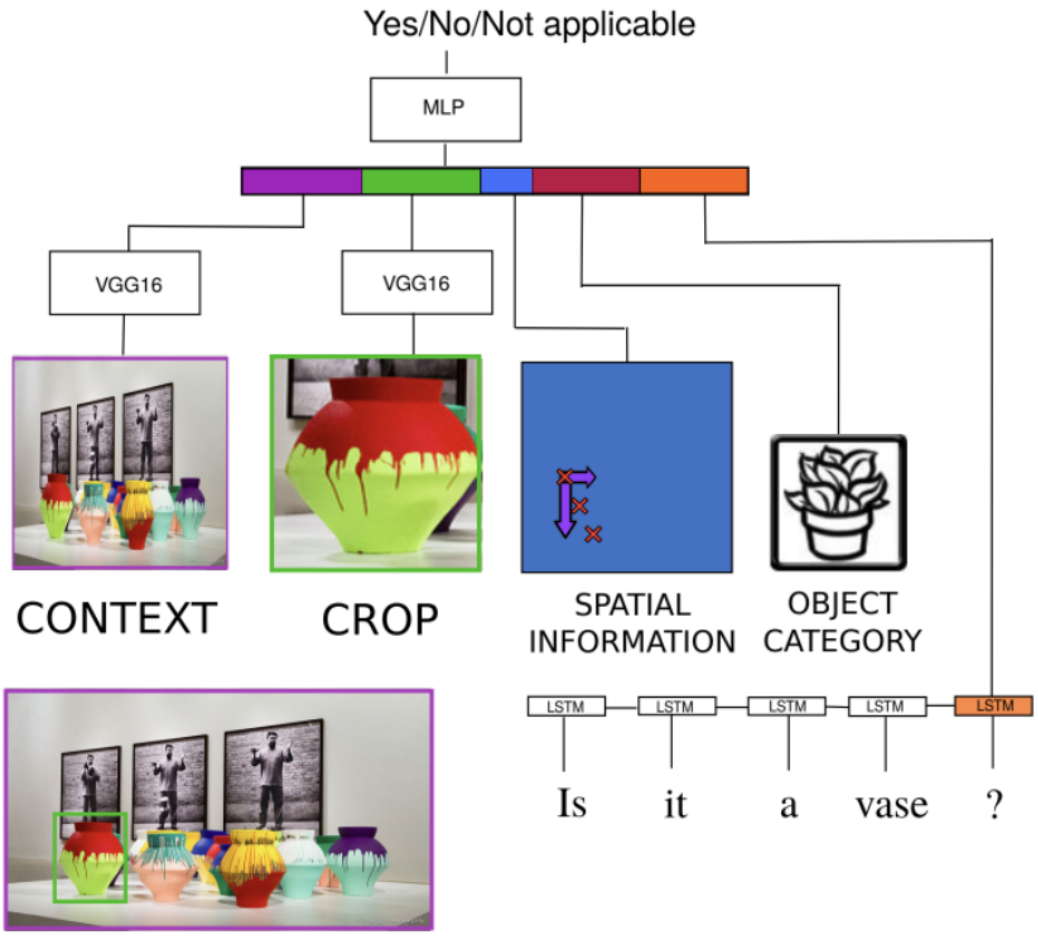


Figura 2.7: Modelo base que propone el paper de GuessWhat?! para el Oráculo.

2.4. Otros trabajos previos relacionados

Ahora vamos a comentar brevemente algunos trabajos previos que de alguna forma estan relacionados o dan pie a la realización de este trabajo final.

End-to-end optimization of goal-driven and visually grounded dialogue systems: En [26] proponen utilizar reinforcement learning para mejorar al guesser. Los autores proponen que el aprendizaje supervisado en un sistema de diálogo generalmente trae malos resultados porque el agente solo aprende a decir exactamente las mismas oraciones que están en el conjunto de entrenamiento. El aprendizaje por refuerzo parece ser una mejor opción, ya que no trata de hacer coincidir exactamente las oraciones, sino que permite una mayor flexibilidad siempre que obtenga una recompensa positiva al final. El problema es: en un contexto de diálogo, ¿cómo puede saber si el diálogo fue "bueno"(recompensa positiva) o "malo"(recompensa negativa)? En el contexto del juego de GuessWhat?!, la recompensa es fácil. Si el adivinador puede encontrar el objeto que se le asignó al oráculo, obtiene una recompensa positiva; de lo contrario, obtiene una recompensa negativa.

En este caso el procedimiento de entrenamiento comprende en primero entrenar con aprendizaje supervisado los modelos base propuestos en [32], es decir entrenar el oráculo, el questioner y el guesser. Una vez que se completa el entrenamiento, se obtiene un sistema de diálogo que es lo suficientemente bueno para jugar solo, pero el modelo de preguntas (el questioner) sigue siendo bastante malo. Para mejorarlo, lo entrenan usando el algoritmo de REINFORCE (refuerzo), siendo la recompensa positiva si el modelo de pregunta adivinó el objeto bueno, negativa en caso contrario, como se comentó anteriormente.

En la figura 2.8 se pueden ver que los resultados obtenidos son bastante prometedores, teniendo en cuenta que se muestran como performance humana, no absoluta (recordar que la precisión humana para esta dataset es de 84.4 %).

Beyond Task Success: A Closer Look at Jointly Learning to See, Ask, and GuessWhat: En [23] se enfocan también en el Questioner como los trabajos previos anteriormente comentados, pero hacen dos contribuciones grandes a la comunidad.

En el artículo se comparan cuatro modelos. El primero fue el modelo baseline pro-

	New objects	New images
Baseline (Supervised)	53.4%	53%
Reinforce	63.2%	62%

Figura 2.8: Resultados del modelo como porcentaje de la performance humana. A la izquierda son los resultados para imágenes que ya habían sido vistas por el modelo, y a la derecha imágenes que el modelo nunca había visto.

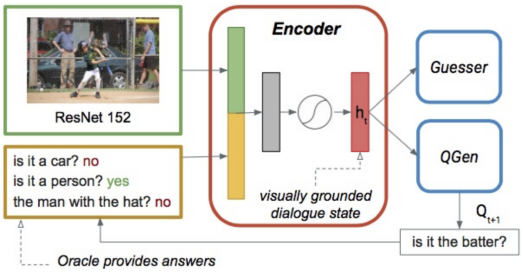


Figura 2.9: Imágen que representa el modelo GDSE de [23]

puesto en el paper original de GuessWhat?! [32] (BL), el segundo y tercero son modelos novedosos que ellos proponen llamados "grounded dialogue state encoder"(GDSE) que no solo entrena el questioner sino que entrena un questioner y guesser en el mismo modelo utilizando un encoder en común como se puede ver en la Figura 2.9. Primeramente lo entrenan con supervised learning (SL) y luego con un método de aprendizaje que ellos llaman Çooperative Learning"(CL), por último también utilizan los resultados de [26] para luego comparar.

Cooperative Learning comprende en entrenar el encoder común con aprendizaje supervisado utilizando los datos humanos, es decir el conjunto de datos de GuessWhat!?, luego dejar que el questioner y el guesser jueguen seleccionando un referente al azar, de esta forma el questioner genera preguntas y el guesser intenta adivinar, así generan nuevos datos. Luego entrenan el encoder común del modelo GDSE propagando hacia atrás el error a través del guesser y luego cada N epochs entrenar con los datos nuevos generados, y así entrenar el questioner.

Model	5Q	8Q
Baseline	41.2	40.7
GDSE-SL	47.8	49.7
GDSE-CL	53.7 (±.83)	58.4 (±.12)
RL	56.2 (±.24)	56.3 (±.05)

Figura 2.10: Resultados para el Questioner. 5Q significa que se le pide al questioner generar 5 preguntas para cada imagen y 8Q igual pero se le pide generar 8 preguntas

Los resultados obtenidos con Cooperative Learning son parecidos a los obtenidos en [26] pero lo bueno de este trabajo es que entrenar con este método es mucho más simple y computacionalmente menos exigente que entrenar el modelo de RL.

Otra contribución muy importante para este trabajo de tesis es la clasificación de las preguntas del conjunto de datos de GuessWhat?! en distintos tipos según la información por la que pregunta. Esta herramienta sirve para comparar cómo se comporta su modelo de Questioner para las distintas clases de preguntas en el conjunto de testeo. Dichas categorías están explicadas con detalle en el Capítulo 4 de este trabajo y lo usamos también para comparar los resultados que obtenemos en nuestros experimentos de modelos para el Oráculo.

Guessing State Tracking for Visual Dialogue: En [14] los autores cuentan que hacer únicamente un solo paso de predicción para el guesser cuando diálogo termina es contraintuitivo, eso mismo es lo que se venía haciendo hasta el momento, es decir, el guesser tomaba como input directamente todos los pares de preguntas respuestas que el questioner y el oracle producían cuando se terminaba el diálogo. Por eso mismo ellos consideran la tarea de adivinar o "guessçomo un proceso continuo y que es necesario realizar un seguimiento del estado en cada ronda del diálogo. Así es que ellos proponen un modelo que llaman *Guessing State Traking* (GST) que en cada ronda del diálogo hace una actualización sobre la distribución de probabilidad que tiene cada objeto de la imagen.

Se relaciona con este trabajo final de licenciatura en que se utiliza el mismo conjunto de datos y propone una mejora para uno de los tres modelos en el juego de

GuessWhat?!. A diferencia de [14] aquí estamos trabajando para el Oráculo, el que se encarga de responder las preguntas para que sea posible armar los pares de preguntas respuestas, es decir el modelo GST usaría los outputs de nuestro Oracle para obtener las respuestas en cada ronda. Similarmente, en este trabajo de tésis se toma la tarea del oráculo también como un proceso continuo, es decir respondemos y propagamos el error ronda a ronda intentando, en cada turno, mejorar las predicciones de nuestro modelo.

Visual Dialog: En [6] introdujeron una nueva tarea para el area de la inteligencia artificial, un nuevo extenso conjunto de datos, los primeros modelos o baselines, un framework para evaluar los mismos, muestran algunos estudios comparando la performance de sus modelos y la de los humanos, y por último presentan lo que, según los autores, sería el primer chat bot visual.

Visual Dialog requiere que un agente mantenga un diálogo significativo con humanos en un lenguaje conversacional natural sobre algún contenido visual. Específicamente, dada una imagen, un historial de diálogo y una pregunta sobre la imagen, el agente debe basar la pregunta en la imagen, inferir el contexto a partir de la historia y responder la pregunta con precisión. Este agente es bastante similar a nuestro oráculo pero al mismo tiempo diferente ya que nuestro espacio de salida de nuestros modelos es mucho más chico ya que los outputs de esta tarea es una oración en lenguaje natural mientras que en nuestro trabajo solo se espera una elección entre tres posibles respuestas (Si, No, o No Aplica), y además nuestro oráculo debe responder preguntas teniendo en cuenta un referente en particular. El conjunto de datos que propusieron contiene 1 diálogo de 10 pares de preguntas con respuestas para cada una de las aproximadamente 140 mil imágenes, es decir, en total colaboraron con aproximadamente 1.4 millones de pares de preguntas y respuestas.

Los autores experimentaron con varios modelos pero sobre todo propusieron una familia de modelos encoder-decoders que muestran una mejora comparados con los modelos anteriormente propuestos en VQA [2] adaptados a la tarea de Visual Dialog. Dicha familia corresponde a la combinación entre 3 tipos de encoders y 2 decoders (6 modelos en total). Los encoders van a intentar representar la información de entrada en un vector y luego los decoders, utilizando el vector de output de los encoders como input, van a generar la respuesta a la pregunta actual.

Encoders:

1. Late Fusion: en este encoder la historia es tratada como un string largo ya que concatena todas las preguntas y respuestas en dicha string. Luego la pregunta actual y la historia se corren por dos LSTMs diferentes y luego se crea un embedding que contiene la imagen y estas dos representaciones que generan ambas LSTMs.

2. Hierarchical Recurrent Encoder: en este modelo se propone una red recurrente a nivel de diálogo arriba de lo que ellos llaman bloques recurrentes. Los bloques recurrentes incrustan la imagen y la pregunta actual a través de una LSTM y también computa una corrida de LSTM para la historia hasta el momento, luego unifica todos esos outputs para pasar ese vector como entrada de la red recurrente superior. Dicha RNN produce una codificación para esta ronda actual y un contexto de diálogo para pasar a la siguiente ronda. También agregamos un mecanismo de atención sobre la historia permitiendo que el bloque recurrente elija y asista a la ronda de la historia relevante para la pregunta actual. Se puede observar una ilustración de este modelo en la Figura 2.11.

3. Memory Network: que trata cada par anterior como un "hecho.en su banco de memoria y aprende a "sondear"los hechos almacenados y la imagen para desarrollar un vector de contexto.

En cuanto a los decoders propusieron dos:

1. uno generativo que utiliza una LSTM para generar el string de salida, la respuesta, durante entrenamiento se maximiza la log-likelihood entre la salida y la respuesta correcta encodeada.

2. y otro discriminativo que hace una softmax entre 100 respuestas candidatas, que dicho conjunto de respuestas se calcula con una heurística descrita detenidamente por los autores en el trabajo.

Las ideas de Late Fusion y de Hierarchical Recurrent Encoder son, de cierta forma, usadas como inspiración de la mayoría de los experimentos de este trabajo final como veremos en los capítulos siguientes.

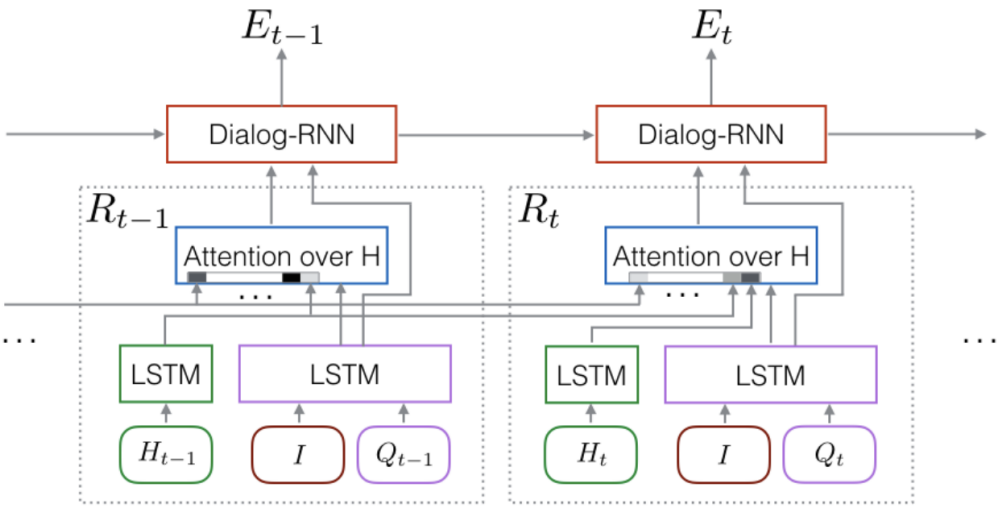


Figura 2.11: Ilustracion del modelo Hierarchical Recurrent Encoder propuesto en [6]

History for Visual Dialog: Do we really need it?: En [1] propusieron varios modelos encoder-decoder que utilizan co-attention que lograron los mejores resultados para ese entonces para el dataset de Visual Dialog ([6]) y muestran que los modelos que utilizan como entrada el encoding de la historia del diálogo son más performantes que los que no. Por ese lado estos resultados son interesantes para este trabajo ya que nos hace pensar que podemos lograr resultados superadores para la tarea de GuessWhat?!.

Por último realizaron un estudio en donde colocaban una imagen y una pregunta, extraídas del conjunto de Visual Dialog, y un anotador humano debía responder una de entre las siguientes respuestas:

1. Puedo responder la pregunta correctamente con confianza solamente con ver la imagen
2. Para responder quiero saber qué preguntas se realizaron anteriormente a la actual
3. Puedo responder la pregunta usando el sentido común
4. Solamente puedo adivinar la respuesta

5. No puedo responder la pregunta

6. La pregunta no es relevante

De esta forma encontraron que solo el 11 % de todas las preguntas del conjunto de datos dependía de la historia y contribuyeron con un conjunto de datos que es subconjunto de Visual Dialog en donde solo se encuentran los diálogos en donde hay preguntas que si dependen de la historia según los humanos que realizaron el estudio que se habla en el artículo. Nos inspiramos en esta idea y también anotamos un subconjunto del GuessWhat en donde solo se encuentran las preguntas que encontramos que necesitan la historia del diálogo para ser respondidas .

Capítulo 3

Marco teórico

En este capítulo explicamos modelos y arquitecturas que serán necesarias para entender los próximos capítulos. Hacemos énfasis en que es un Multi Layer Perceptron, cómo funciona el mecanismo de Backpropagation que es una de las ideas fundamentales que construyen al entrenamiento de las redes neuronales. Luego explicamos redes neuronales recurrentes, técnicas más avanzadas y nuevas como mecanismos de atención y arquitecturas "Transformer". Por último el modelo LXMERT [27] que es un modelo basado en transformers que utilizaremos en este trabajo. La estructura del capítulo será la siguiente:

1. Multilayer Perceptron
2. RNN y LSTM
3. Transformers
4. LXMERT

3.1. Perceptron multicapa

Una neurona es la unidad más pequeña y básica de cómputo presente en los modelos de redes neuronales, a veces se la llama "unit" o nodo. Se puede ver como una función que recibe una cantidad fija de variables de entrada, cada una de ellas tiene asociada un peso w que se asigna sobre la base de su importancia relativa a las demás entradas.

La neurona aplica una función f a la suma pesada de las entradas como se muestra en la Figura 3.1.

La neurona representada en la Figura 3.1 toma dos entradas numéricas $X1$ y $X2$ con pesos $w1$ y $w2$ asignados a cada una respectivamente. Adicionalmente, hay un valor de sesgo (b en la Figura 3.1) cuya función es proveer a cada nodo con una variable entrenable constante. La función f generalmente es no lineal y es llamada "Activation Function" o, en español, función de activación. Si no aplicamos esta no linealidad a nuestra suma pesada de las entradas las redes neuronales no serían más que una regresión lineal. Además los datos de problemas del mundo real suelen ser complejos y no lineales por lo que queremos que nuestros modelos sean capaces de aprender esas representaciones. Más adelante en el capítulo veremos algunas de las funciones de activación más comunes en el aprendizaje profundo por computadoras.

El perceptron multicapa (MLP de ahora en adelante) es el primer y más simple modelo de los que hoy se conocen como "Artificial Neural Networks" (ANNs). Básicamente se compone de múltiples neuronas organizadas en capas o "layers". Neuronas de capas adyacentes se conectan entre sí y estas conexiones tienen pesos asociados. En esencia, simplemente estamos tomando un conjunto de neuronas (capas) y apilándolas una encima de la otra, capa por capa, y conectando estas capas a través de vectores de pesos; las salidas de la capa 1 se alimentan como entradas a la capa 2, las salidas de la capa 2 se alimentan a la capa 3, y así sucesivamente hasta que llegamos a la capa de salida final.

Generalmente se dividen los nodos en 3 tipos como muestra la Figura 3.2:

1. Nodos de entrada o "input nodes": los nodos de entrada proporcionan información del mundo exterior a la red y, en conjunto, se denominan capa de entrada o "input layer". No se realiza ningún cálculo en ninguno de los nodos de entrada, simplemente pasan la información a los nodos ocultos.

2. Nodos ocultos o "hidden nodes": los nodos ocultos no tienen conexión directa con el mundo exterior (de ahí el nombre 'oculto'). Realizan cálculos y transfieren información desde los nodos de entrada a los nodos de salida. Una colección de nodos ocultos forma una capa oculta o "hidden layer".

3. Nodos de salida o "output nodes": los nodos de salida se denominan colecti-

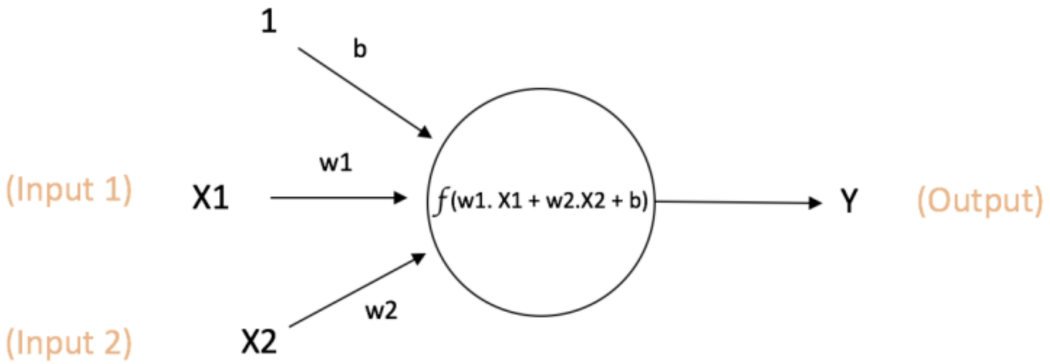


Figura 3.1: Representación gráfica de una neurona computacional

vamente capa de salida o "output layer" y son responsables de los cálculos y la transferencia de información desde la red al mundo exterior. Estos son los nodos de predicción.

Elegir la función de activación correcta para un modelo no es una tarea trivial, y hay mucha investigación dedicada a comprender cuándo y dónde ciertas activaciones funcionan mejor. Las funciones más usadas en esta rama de la computación son las mencionada a continuación graficadas en la Figura 3.3:

- Sigmoide: toma una entrada de valor real y la aplasta para que oscile entre 0 y 1 (la aplicación de la función de activación sigmoidea a una sola neurona simplemente equivale a una regresión logística)

  $\sigma(x) = \frac{1}{1+e^x}$

- Tanh: toma una entrada de valor real y la aplasta al rango [-1, 1] (esta es esencialmente una versión desplazada de la función sigmoide)

  $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

- ReLU: ReLU son las siglas de Rectified Linear Unit. Toma una entrada de valor real y la establece como umbral en cero (reemplaza los valores negativos con cero)

  $relu(x) = max(x, 0)$

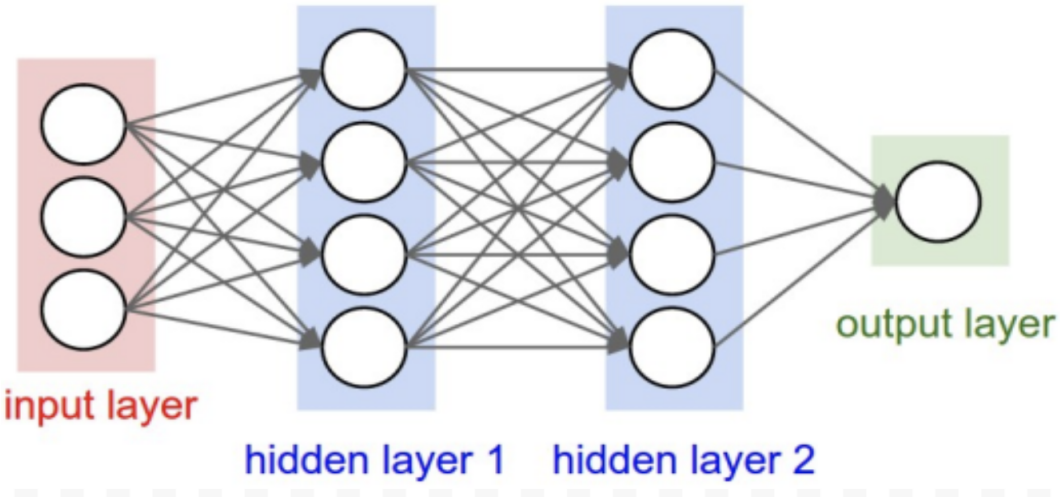


Figura 3.2: Representación gráfica de un multilayer perceptron

- GeLU: añaden la no linealidad al multiplicar de manera estocástica por 0 o 1 dependiendo de un muestreo a una distribución normal estándar.

  $gelu(x) = x\phi(x)$

Pero cómo las redes neuronales aprenden? Lo hacen a través de un algoritmo conocido como *Backpropagation* que en castellano se podria traducir como "propagación hacia atrás de errores.º retropropagación". Es un algoritmo para el aprendizaje supervisado de redes neuronales mediante el descenso de gradientes. Dada una red neuronal y una función de error, el método calcula el gradiente de la función de error con respecto a los pesos de cada capa de la red neuronal y luego sobre escribe esos pesos intentando minimizar dicha función, la hipótesis es que si hacemos eso muchas veces podemos encontrar un minimo interesante que nos permita resolver el problema. Por lo que el proceso de aprender es primero hacer un paso hacia delante, es decir, dado un input computar el output que genera la red y luego calcular el gradiente con respecto a cada peso de cada capa para que con ese gradiente poder actualizar los pesos y mejorar la predicción. Lo que se intenta con este algoritmo es encontrar el conjunto de pesos que minimice la funcion de costo. Más formalmente , entrenar una red requiere el cálculo del gradiente de la función de error $E(X, \theta)$ con respecto a los pesos w_{ij}^k (los pesos entre el nodo j de la capa k y el nodo i de la capa $k-1$) y los biases b_i^k (bias del nodo i en la capa k). Luego, de acuerdo a un

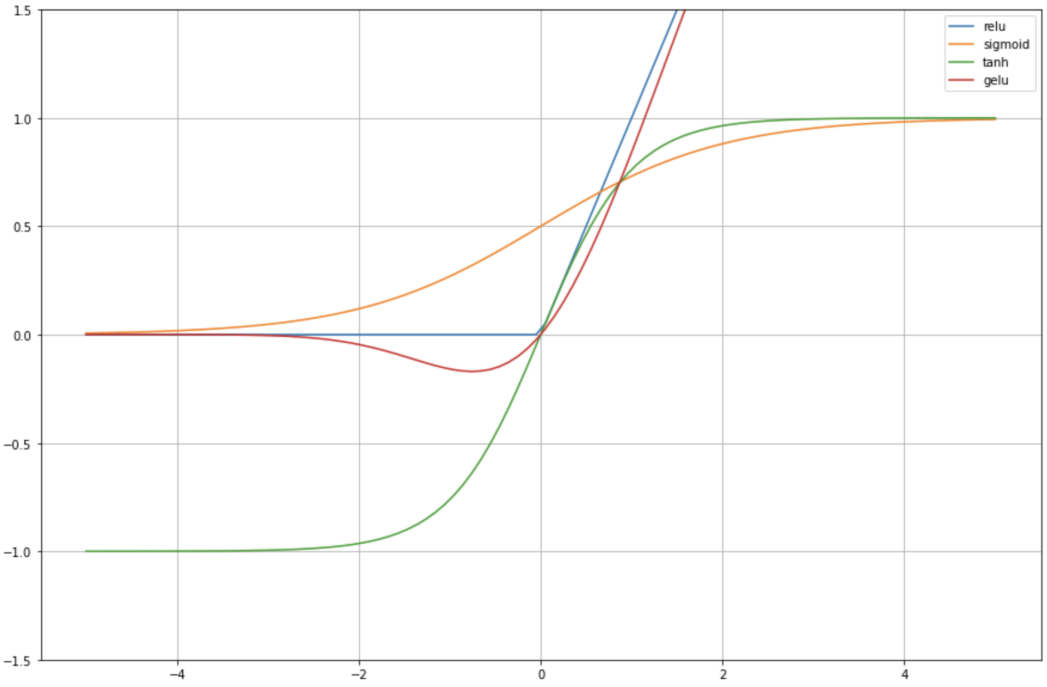


Figura 3.3: Funciones de activación

hiper parámetro llamado *learning rate* α, en cada iteración se actualizan los pesos y los biases (conjuntos llamémoslos θ) de la siguiente forma:

$$\theta^{t+1} = \theta^t - \alpha \frac{\partial E(X, \theta^t)}{\partial \theta}$$

donde θ^t denota los parámetros de la red en la iteración t del algoritmo.

3.2. RNN y LSTM

3.2.1. Redes Neuronales Recurrentes

Hasta ahora nos hemos centrado en las redes neuronales feedforward (o MLP), donde las activaciones fluyen solo en una dirección, desde la capa de entrada a la capa de salida. Una red neuronal recurrente se parece mucho a un MLP, excepto que también tiene conexiones que apuntan hacia atrás. Veamos el RNN más simple posible, compuesto por una neurona que recibe entradas, produce una salida y envía esa salida a sí misma, como se muestra en la parte izquierda de Figura 3.4. En cada paso de tiempo t, esta neurona recurrente recibe las entradas $x_{(t)}$ así como su propia salida del paso de tiempo anterior, $y_{(t-1)}$. Dado que no hay una salida previa en el primer paso de tiempo, generalmente se establece en 0. Podemos representar esta pequeña red contra el eje del tiempo, como se muestra en la parte derecha de Figura 3.4. Esto se llama *desenrollar la red a través del tiempo*. Es la misma neurona recurrente representada una vez por paso de tiempo.

Se puede crear fácilmente una capa de neuronas recurrentes. En cada paso de tiempo t, cada neurona recibe tanto el vector de entrada $x_{(t)}$ como el vector de salida del paso de tiempo anterior $y_{(t-1)}$, como se muestra en la Figura 3.5.

Cada neurona recurrente tiene dos conjuntos de pesos: uno para las entradas $x_{(t)}$ y otro para las salidas del paso de tiempo anterior, $y_{(t-1)}$. Llamemos a estos vectores de peso w_x y w_y. Si consideramos toda la capa recurrente en lugar de solo una neurona recurrente, podemos colocar todos los vectores de peso en dos matrices de peso, W_x y W_y. El vector de salida de toda la capa recurrente se puede calcular de la siguiente manera:

$$y_{(t)} = \phi(W_x x_{(t)} + W_y y_{(t-1)} + b)$$

en donde b es el vector de sesgo y $\phi(\Delta)$ es alguna función de activación (ReLU, por

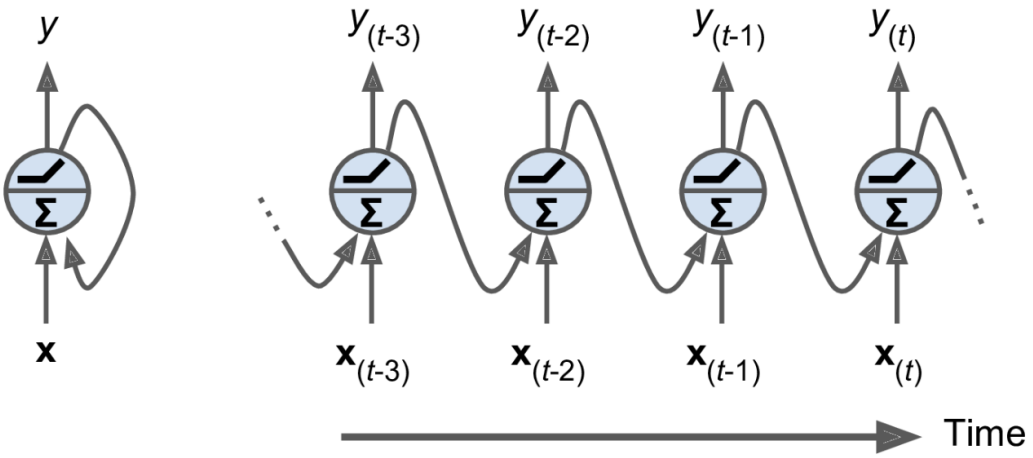


Figura 3.4: A la izquiera una RNN simple, a su derecha la misma pero desenrollada a travéz del tiempo. Imagen de [8].

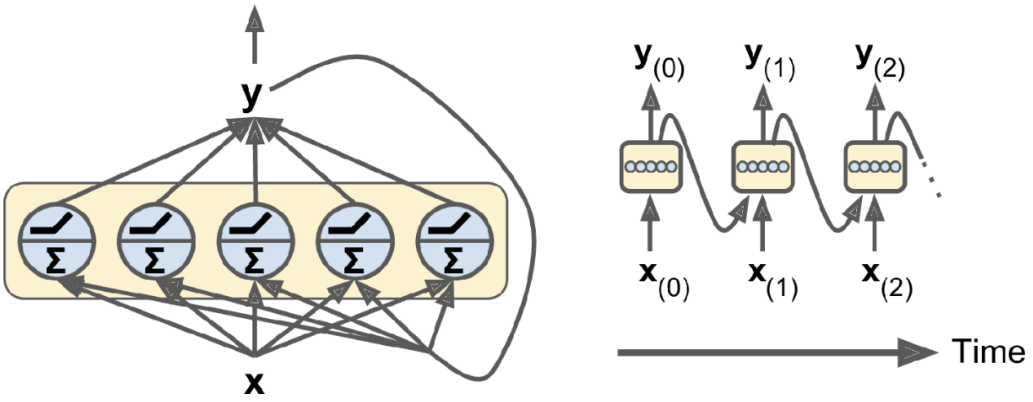


Figura 3.5: A la izquierda una capa de neuronas recurrentes, a la izquierda la capa desenrollada en el tiempo. Imagen de [8]

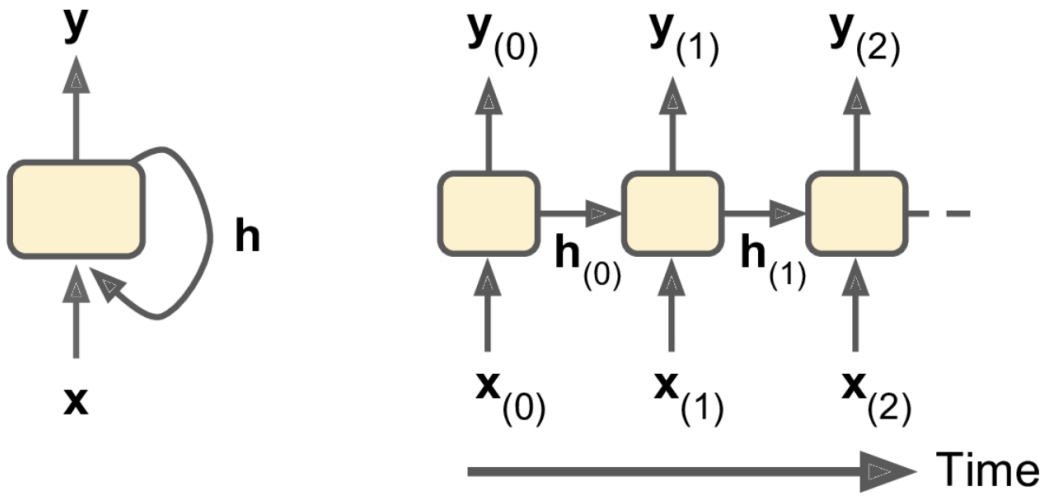


Figura 3.6: Los estados ocultos de una celda (o hidden states en inglés) pueden ser distintos a la salida de la misma. Imagen de [8]

ejemplo). Observar que $y_{(t)}$ es una función de $x_{(t)}$ y $y_{(t–1)}$, que es una función de $x_{(t–1)}$ y $y_{(t–2)}$, que es una función de $x_{(t–2)}$ y $y_{(t–3)}$, y así sucesivamente. Esto hace que $y_{(t)}$ sea una función de todas las entradas desde el tiempo $t = 0$ (es decir, $x_{(0)}$, $x_{(1)}$, ..., $x_{(t)}$). En el primer paso de tiempo, $t = 0$, no hay salidas anteriores, por lo que normalmente se supone que son todos ceros. Dado que la salida de una neurona recurrente en el paso de tiempo t es una función de todas las entradas de los pasos de tiempo anteriores, se podría decir que tiene una forma de memoria. Una parte de una red neuronal que conserva algún estado a lo largo de los pasos de tiempo se denomina celda de memoria (o simplemente celda). En general, el estado de una celda en el tiempo t, denotado $h_{(t)}$ (la h significa oculto, proviene de *hidden* en inglés), es una función de algunas entradas en ese paso de tiempo y su estado en el paso de tiempo anterior: $h_{(t)} = f(h_{(t–1)}, x_{(t)})$. Su salida en el paso de tiempo t, denotado $y_{(t)}$, también es una función del estado anterior y las entradas actuales. En el caso de las celdas básicas que hemos analizado hasta ahora, la salida es simplemente igual al estado, pero en celdas más complejas no siempre es así, como se muestra en la Figura 3.6.

Una RNN puede tomar simultáneamente una secuencia de entradas y producir una

secuencia de salidas. Alternativamente, podría alimentar a la red con una secuencia de entradas e ignorar todas las salidas excepto la última. En otras palabras, esta es una red de secuencia a vector. Por el contrario, podría alimentar a la red con el mismo vector de entrada una y otra vez en cada paso de tiempo y dejar que genere una secuencia. Esta es una red de vector a secuencia. Por último, podría tener una red de secuencia a vector, generalmente llamada *encoder* (de codificador en inglés), seguida de una red de vector a secuencia, llamada *decoder* (de decodificador en inglés). Por ejemplo, esto lo podemos usar para traducir una oración de un idioma a otro. Alimentamos la red con una oración en un idioma, el encoder convertiría esta oración en una representación de un solo vector y luego el decoder decodificaría este vector en una oración en otro idioma. Este modelo de dos pasos, usualmente llamado encoder-decoder, funciona mucho mejor que tratar de traducir sobre la marcha con un único RNN de secuencia a secuencia: las últimas palabras de una oración pueden afectar las primeras palabras de la traducción, por lo que debe esperar hasta que haya visto la oración completa antes de traducirla. En la Figura 3.7 podemos ver una representación visual de estas opciones.

Para entrenar una RNN, el truco consiste en desenrollarla a lo largo del tiempo y luego simplemente usar backpropagation. Esta estrategia se llama BackPropagation Through Time (BPTT).

El principal problema que hace que el entrenamiento de RNN sea muy lento e ineficiente para su uso es el problema del gradiente de *desvanecimiento de gradientes* (conocido en inglés como *Vanishing Gradient*). El proceso para una red neuronal tradicional como lo es un MLP es el siguiente: a) el paso hacia adelante genera una predicción b) ese output lo utilizamos para calcular la función de costo c) el valor de pérdida se usa para realizar el backpropagation para calcular los gradientes con respecto al pesos d) estos gradientes con respecto a los pesos ajustan los pesos con el fin de mejorar el rendimiento de la red neuronal. A medida que la manipulación de pesos ocurre según la capa anterior, los pequeños gradientes tienden a disminuir en grandes márgenes después de cada capa y alcanzan un punto en el que están muy cerca de cero, por lo que el aprendizaje cae para las capas iniciales. Por lo tanto, el *desvanecimiento de gradientes* hace que las RNN no aprendan bien las dependencias de largo alcance a lo largo de los pasos en el tiempo. Esto significa que los tokens anteriores de una secuencia no tendrán gran importancia incluso si son cruciales para

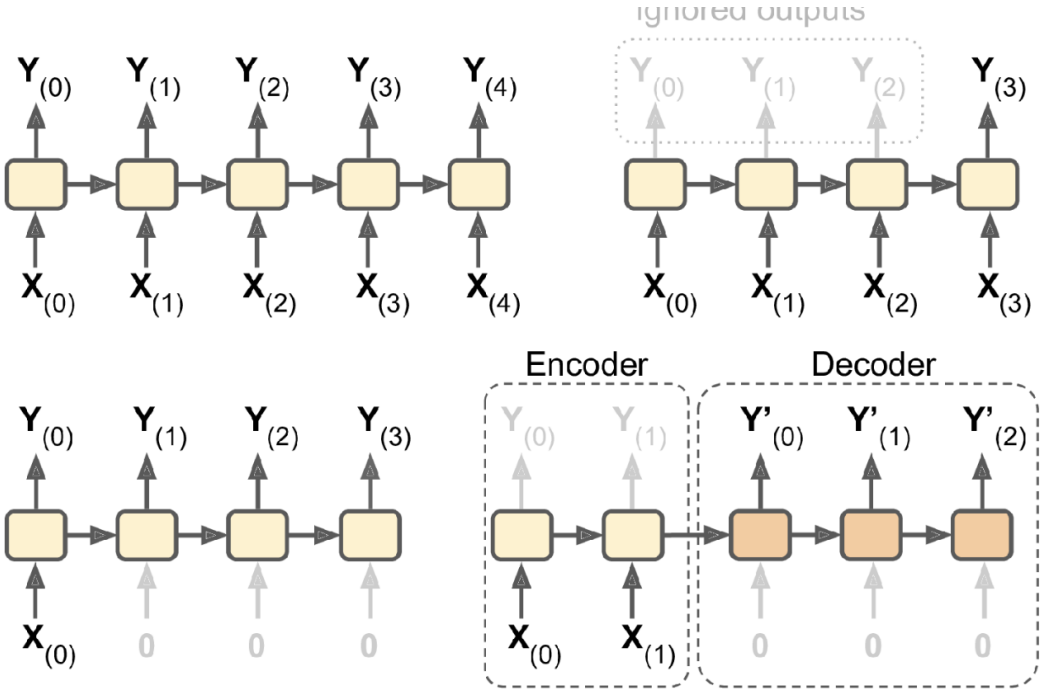


Figura 3.7: En la esquina superior izquiera vemos una red que toma una secuencia de entradas y devuelve una secuencia de salidas, en la esquina superior derecha una que toma una secuancia de entradas y devuelve un vector. En la esquina inferior izquierda una red que toma un vector y devuelve una secuencia, y por último en la esquina inferior derecha vemos una red $Encoder - Decoder$. Imagen de [8]

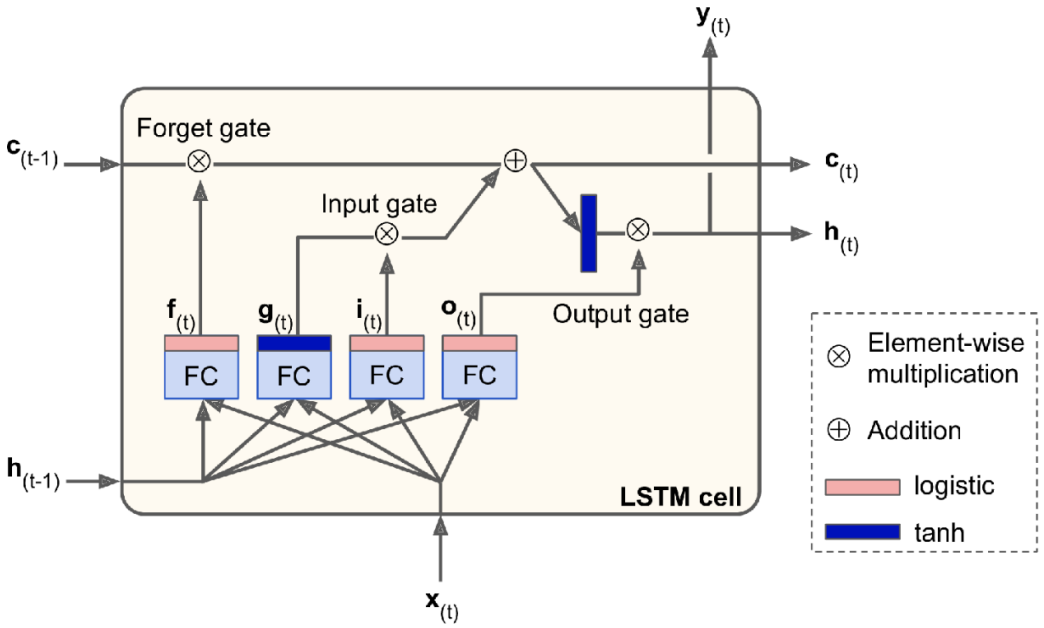


Figura 3.8: Ilustración de la arquitectura de una LSTM. FC es sinonimo de MLP en este caso ya que provience de $Fully - Connected$ en inglés. Imagen de [8]

todo el contexto. Por lo tanto, esta incapacidad para aprender en secuencias largas da como resultado una memoria a corto plazo (*short term memory*, en inglés).

Para abordar este problema, se han introducido varios tipos de células con memoria a largo plazo. Han tenido tanto éxito que las células básicas ya no se usan mucho. Por eso a continuación introducimos la arquitectura LSTM.

3.2.2. LSTM: Long Short-Term Memory

La celda Long Short-Term Memory (LSTM) [12], si la miramos como una caja negra, es similar a una celda RNN tradicional, excepto que su estado se divide en dos vectores: $h_{(t)}$ y $c_{(t)}$ (c viene de celda). Se puede pensar en $h_{(t)}$ como el estado a corto plazo y $c_{(t)}$ como el estado a largo plazo. En la Figura 3.8 se puede ver una ilustración de la arquitectura.

La idea principal es que la red pueda aprender que le conviene almacenar en el estado a largo plazo y que desechar. A medida que el estado a largo plazo $c_{(t-1)}$

atraviesa la red de izquierda a derecha, se puede ver en la Figura 3.8 que primero pasa por una puerta de olvido, eliminando algunos recuerdos y luego agrega algunos recuerdos nuevos a través de la operación de suma. El resultado $c_{(t)}$ se envía directamente, sin más transformación para que la utilice la celda en el siguiente paso de tiempo. Entonces, en cada paso de tiempo, se eliminan algunos recuerdos y se agregan otros.

Primero, el vector de entrada actual $x_{(t)}$ y el estado anterior de corto plazo (el hidden state anterior) $h_{(t-1)}$ alimentan cuatro MLPs distintos, todas con un propósito diferente.

La capa principal es la que genera $g_{(t)}$. Tiene la función de analizar $x_{(t)}$ y $h_{(t-1)}$. En una celda RNN tradicional, no hay nada más que esta capa, y su salida va directamente a $y_{(t)}$ y $h_{(t)}$. Por el contrario, en una celda LSTM, la salida de esta capa no sale directamente, sino que sus partes más importantes se almacenan en el estado a largo plazo y el resto se descarta.

Las otras tres capas son como compuertas contraladoras. Dado que utilizan la función de activación *sigmoide*, es decir que sus salidas van de 0 a 1. Sus salidas se alimentan de operaciones de multiplicación, por lo que si emiten 0, cierran la puerta, y si emiten 1, la abren. Más específicamente: la compuerta puerta de olvido controlada por $f_{(t)}$ controla qué partes del estado a largo plazo deben borrarse, la compuerta controlada por $i_{(t)}$ controla qué partes de $g_{(t)}$ deben agregarse al estado a largo plazo, y la compuerta de salida controlada por $o_{(t)}$ controla qué partes del estado a largo plazo deben leerse y enviarse en este paso de tiempo, tanto a $h_{(t)}$ como a $y_{(t)}$. En resumen, una celda LSTM puede aprender a reconocer una entrada importante, almacenarla en el estado a largo plazo, conservarla durante el tiempo que sea necesario y extraerlo cuando sea necesario.

Matematicamente podemos expresar una LSTM como:

$$
\begin{aligned}
i_{(t)} &= \sigma(W_{xi}x_{(t)} + W_{hi}h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma(W_{xf}x_{(t)} + W_{hf}h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma(W_{xo}x_{(t)} + W_{ho}h_{(t-1)} + b_o) \\
g_{(t)} &= tanh(W_{xg}x_{(t)} + W_{hg}h_{(t-1)} + b_g) \\
c_{(t)} &= f(t) \bigotimes c_{(t-1)+i_{(t)}} \bigotimes g_{(t)} \\
y_{(t)} &= h_{(t)} = o_{(t)} \bigotimes tanh(c_{(t)})
\end{aligned}
$$

Donde $W_{xi}, W_{xf}, W_{xo}, W_{xg}$ son las matrices de peso de cada una de las cuatro capas para su conexión con el vector de entrada $x_{(t)}$, $W_{hi}, W_{hf}, W_{ho}, W_{hg}$ son las matrices de peso de cada una de las cuatro capas para su conexión con $h_{(t-1)}$, y $b_i, b_f, b_o y b_g$ son los términos de bias para cada una de las cuatro capas.

3.3. Transformers

3.3.1. Mecanismos de Atención

Para comprender de manera fácil los mecanismos de atención veamos un modelo simple de traducción automática que traducirá oraciones del inglés al francés. Como se puede ver en la Figura 3.9, las oraciones en inglés se envían al encoder y el decoder genera las traducciones al francés. Notar que las traducciones al francés también se utilizan como entradas para el decoder, pero se desplazan hacia atrás un paso, el decoder recibe como entrada la palabra que debería haber emitido en el paso anterior. Para la primera palabra, se le da el token de inicio de secuencia (en este caso particular de la Figura 3.9 es $< sos >$ que proviene del ingles start of secuence). Se espera que el decoder finalice la oración con un token de fin de secuencia (en este caso $< eos >$ de end of secuence en inglés). Las oraciones generalmente se invierten antes de que se envíen al encoder. Por ejemplo, "I drink milk" ("yo tomo leche" en castellano) se invierte en "drink milk I". Esto asegura que el comienzo de la oración en inglés se enviará en último lugar al encoder, lo cual es útil porque generalmente es lo primero que el decoder necesita traducir. Cada palabra se representa inicialmente por su ID (por ejemplo, 288 para la palabra "milk"). A continuación, una capa devuelve el *embedding*, es decir la representación vectorial de esa palabra. Este *embedding* es lo que realmente se envía al encoder y al decoder. En cada paso, el decoder genera una puntuación para cada palabra en el vocabulario de salida (en este caso francés), y luego la capa *softmax* convierte estas puntuaciones en probabilidades.

Este ejemplo lo introducimos porque nos ayuda a ver un problema que podriamos mejorar. Consideremos el camino desde la palabra "milk" hasta su traducción "lait" en la Figura 3.9, es bastante largo e imaginemos que la frase es aun más larga el camino de una palabra desde su encoder hasta su traducción en un decoder puede ser muy largo. Esto significa que una representación de esta palabra (junto con

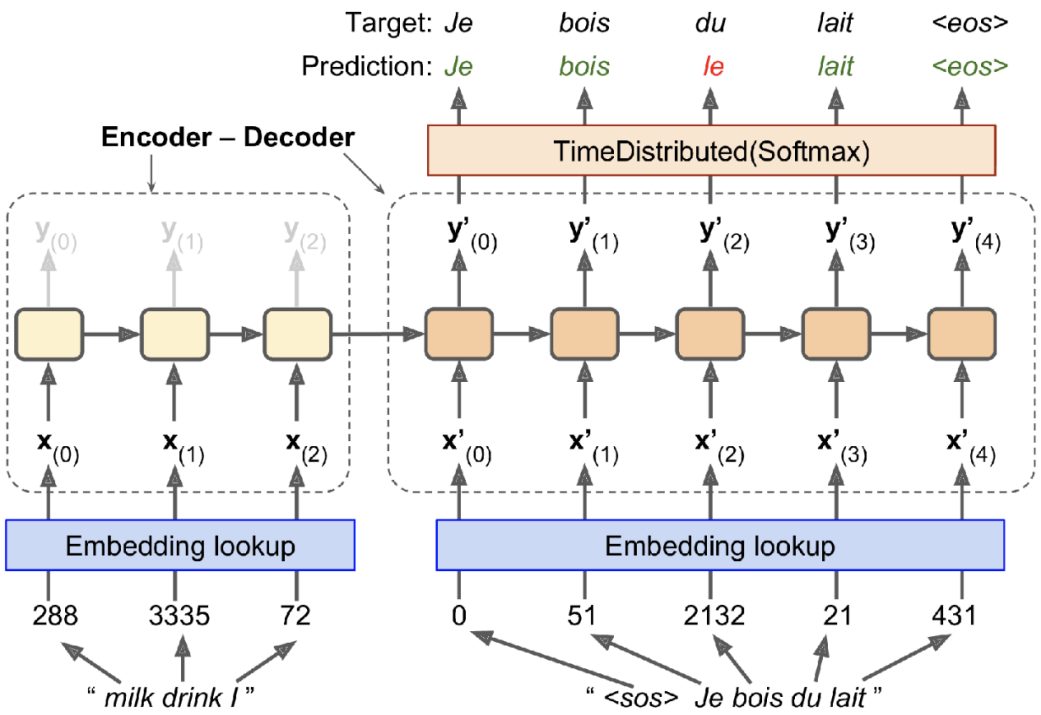


Figura 3.9: Modelo de ejemplo para resolver el problema de traducción automática con una red *Encoder-Decoder*. Imagen de [8]

todas las demás palabras) debe llevarse a cabo durante muchos pasos antes de que se use realmente. Esto nos hace pensar si pordríamos mejorar o acortar ese pasaje de información.

En [4] introdujeron una técnica que permitía al decoder enfocarse en las palabras apropiadas. En la Figura 3.10 mostramos la arquitectura de este modelo. A la izquierda, tienes el encoder y el decoder. En lugar de simplemente enviar el estado oculto final del encoder al decoder, ahora enviamos todas sus salidas al decoder. En cada paso de tiempo, la celda de memoria del decoder calcula una suma ponderada de todas estas salidas del encoder: esto determina en qué palabras se enfocará en cada paso. El peso $\alpha_{(t,i)}$ es el peso de la salida i del encoder en el paso t de tiempo del decoder. Por ejemplo, si el peso $\alpha_{(3,2)}$ es mucho mayor que los pesos $\alpha_{(3,0)}$ y $\alpha_{(3,1)}$, entonces el decoder prestará mucha más atención a la palabra número 2 ("milk", que significa leche en castellano) que a las otras dos palabras, al menos en este paso de tiempo. Cada $\alpha_{(t,i)}$ es generado por un tipo de pequeña red neuronal llamada modelo de alineación (o capa de atención), que se entrena conjuntamente con el resto del modelo Encoder-Decoder. Este modelo de alineación se ilustra en el lado derecho de la Figura 3.10. Comienza con una capa densa distribuida en el tiempo con una sola neurona, que recibe como entrada todas las salidas del encoder, concatenadas con el estado oculto anterior del decoder. Esta capa genera una puntuación para cada salida del encoder (por ejemplo, en la imagen, $e_{(3,2)}$): esta puntuación mide qué tan bien se alinea cada salida con el estado oculto anterior del decoder. Finalmente, todos los puntajes pasan por una capa softmax para obtener un peso final para cada salida del encoder. Este método se lo suele llamar atención concatenativa.

Otro mecanismo de atención fue propuesto poco después [18]. Debido a que el objetivo del mecanismo de atención es medir la similitud entre una de las salidas del encoder y el estado oculto anterior del decoder, los autores propusieron simplemente calcular el producto punto de estos dos vectores, ya que esto suele ser bastante buena medida de similitud podemos calcularlo mucho más rápido. Para que esto sea posible, ambos vectores deben tener la misma dimensionalidad. El producto punto da una puntuación, y todas las puntuaciones pasan por una capa de softmax para dar los pesos finales. Otra simplificación que propusieron fue usar el estado oculto del decoder en el paso de tiempo actual en lugar del paso de tiempo anterior

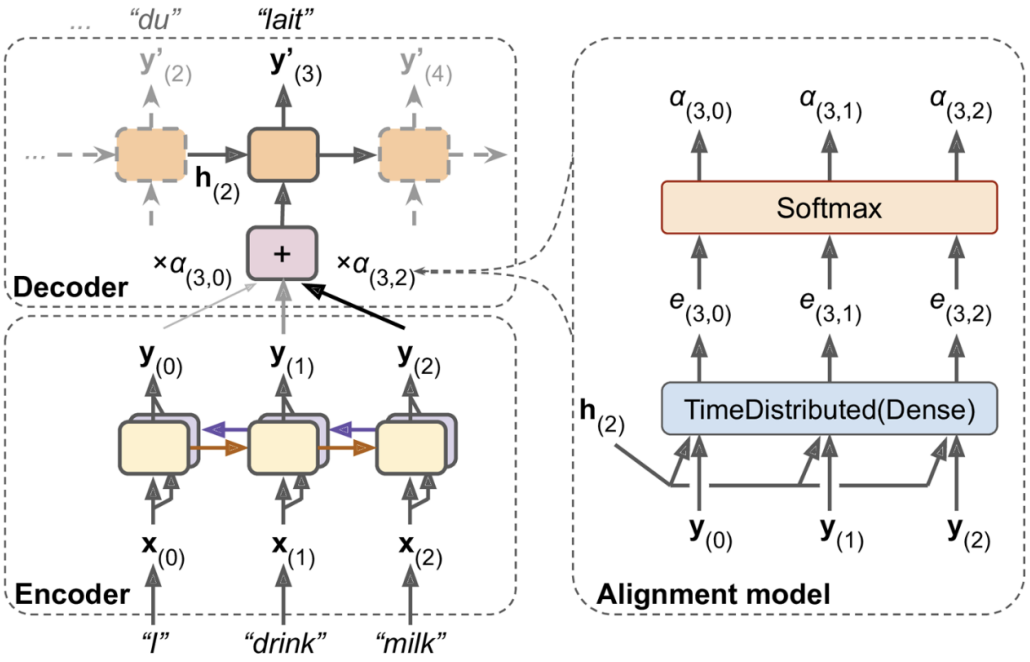


Figura 3.10: Ilustración de la técnica de atención para modelos Encoder-Decoder. Imagen de [8]

(es decir, $h_{(t)}$ en lugar de $h_{(t-1)}$), luego usar la salida del mecanismo de atención (llamémoslo $\tilde{h}_{(t)}$) directamente para calcular las predicciones del decoder en lugar de usarlo para calcular el estado oculto actual. También propusieron una variante del mecanismo del producto punto en el que las salidas del encoder pasan primero por una transformación lineal, es decir, una capa densa antes de que se calculen los producto. Compararon ambos enfoques de productos puntos con el mecanismo de atención concatenativa (agregando un vector de parámetros de cambio de escala v) y observaron que las variantes de productos escalares funcionaron mejor que la atención concatenativa.

Matemáticamente:

$$\tilde{h}_{(t)} = \sum_i \alpha_{(t,i)} y_i \text{ donde } \alpha_{(t,i)} = \frac{\exp e_{(t,i)}}{\sum_j \exp e_{(t,j)}} \text{ donde } e_{(t,i)} = \begin{cases} h_{(t)} y_{(i)} \\ h_{(t)} W y_{(i)} \\ v^\top tanh(W[h_{(t)}; y_{(t)}]) \end{cases}$$

3.3.2. Atención es todo lo que necesitas

En [31] sugiren que "Todo lo que se necesita es atención"(conocido en inglés el trabajo por el nombre ."Attention is all you need"). Crearon una arquitectura que se llama Transformer que utiliza solo mecanismos de atención (más algunas capas de embeddings, capas densas y capas de normalización). Esta arquitectura también es mucho más rápida de entrenar y más fácil de paralelizar que modelos recurrentes. La Figura 3.11 ilustra este modelo.

El modelo Transformer utiliza una arquitectura *Encoder-Decoder*. El encoder consta de capas que procesan la entrada de forma iterativa una capa tras otra, mientras que el decoder consta de capas que hacen lo mismo con la salida del encoder. La función de cada capa del encoder es generar vectores que contengan información sobre qué partes de las entradas son relevantes entre sí y pasar dichas representaciones a la siguiente capa como entradas. Cada capa del decoder hace lo contrario, toma todas los vectores y usa su información contextual incorporada para generar una secuencia de salida. Para lograr esto, cada capa hace uso de un mecanismo de atención. Para cada entrada, la atención mide la relevancia de todas las demás entradas y se basa en ellas para producir la salida. Cada capa del decoder tiene un mecanismo de atención adicional que extrae información de las salidas del decoder anterior, antes de que extraiga información de las salidas del encoder. Ambas capas tienen un MLP (o

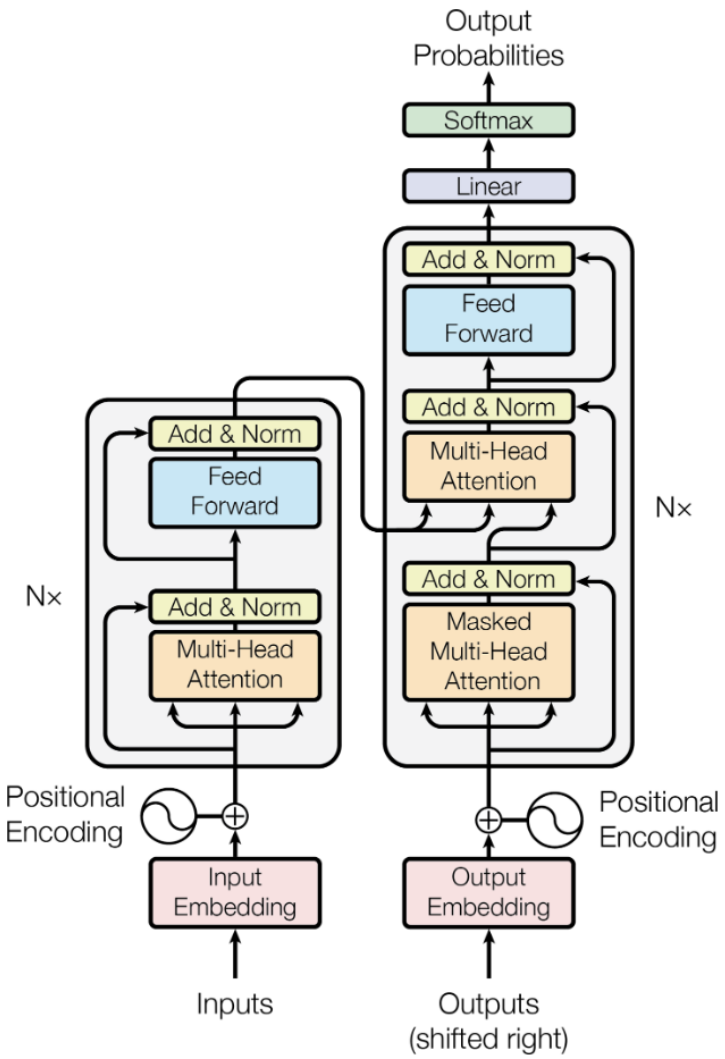


Figura 3.11: Ilustración de la arquitectura de un modelo Transformer del paper [31].

red *feedfoward* en la Figura 3.11) para el procesamiento adicional de las salidas y contienen conexiones residuales y capas de normalización.

Los bloques de atencion en la arquitectura de transformers son unidades de atención de producto punto a escala (conocido en inglés como *scaled dot-product attention*). Cuando se pasa una oración los pesos de atención se calculan entre cada token simultáneamente. La unidad de atención produce, para cada token, embeddings para todos los otros tokens y también una ponderadación de todos los tokens relevantes. Para cada unidad de atención, el modelo aprende tres matrices de pesos; las ponderaciones de consulta W_Q, las ponderaciones de clave W_K y las ponderaciones de valor W_V.

Para cada token i, el embedding de entrada x_i se multiplica con cada una de las tres matrices de peso para producir un vector de consulta $q_i = x_i W_Q$, un vector clave $k_i = x_i W_K$, y un vector de valor $v_i = x_i W_V$. Los pesos de atención se calculan usando los vectores clave y de consulta: el peso de atención a_{ij} del token i al token j es el producto punto entre q_i y k_j. Los pesos de atención se dividen por la raíz cuadrada de la dimensión de los vectores clave, $\sqrt{d_k}$, que estabiliza los gradientes durante entrenamiento, y luego pasan por una capa *softmax* que normaliza los pesos. La salida de la unidad de atención para el token i es la suma ponderada de los vectores de valor de todos los tokens ponderados por a_{ij}.

El cálculo de atención para todos los tokens se puede expresar como un cálculo de matriz, donde Q, K y V se definen como matrices donde las *i-ésimas* filas son vectores q_i , k_i y v_i respectivamente. Matemáticamente:

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V$$

Cuando la atención se realiza en consultas, claves y valores generados a partir del mismo embedding, se denomina *self-attention*. Cuando la atención se realiza en consultas generadas a partir de un embedding y claves y valores generados a partir de otro embedding, se denomina *cross-attention*. Un conjunto de (W_Q, W_K, W_V) matrices se denomina *attention head*, y cada capa en un modelo Transformer tiene varias *attention heads*. Con múltiples de ellas el modelo puede hacer calcular atenciones para diferentes definiciones de relevancia". Por ejemplo, los algunas pueden prestar mayor atención a la siguiente palabra, mientras que otros atienden principalmente de los verbos a sus complementos directos, etc.

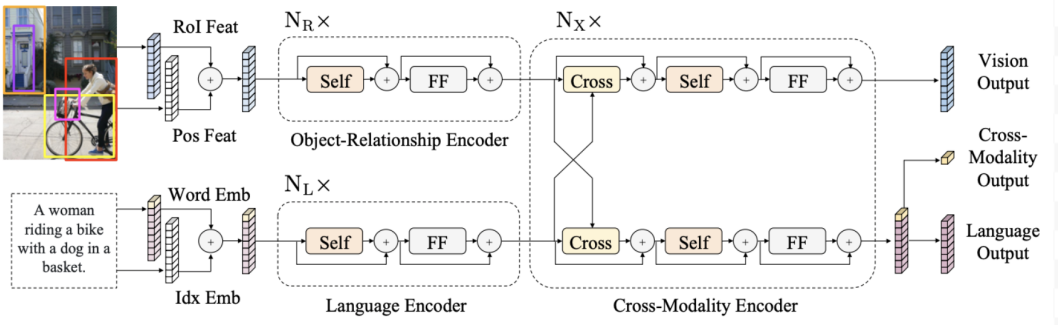


Figura 3.12: Arquitectura del modelo LXMERT. Sus inputs, sus encoders y su output.

3.4. LXMERT: Aprendiendo a codificar representaciones de modalidades cruzadas con Transformers

El razonamiento en tareas de Visión-Lenguaje requiere el entendimiento de modalidades visuales, lingüísticas y las relaciones entre ambas. En los últimos años ha habido mucho trabajo en el desarrollo de modelos *backbone* para realizar representaciones de lenguaje y de visión. Este modelo publicado en [27] viene a proponer poder aprender representaciones tanto como de cada modalidad por separado como de ambas juntas. Está modelado basándose en innovaciones basadas en BERT, adaptándolo para escenarios de modalidad cruzada, enfocandose en aprender interacciones multimodales. A grandes rasgos, consiste en tres encoders: uno que se encarga de modelar las relaciones entre objetos en la imagen, otro encargado de modelar relaciones entre las palabras del texto, y por último otro encoder que modela las relaciones entre lenguaje y la visión. LXMERT fue entrenado en 5 tareas diversas y representativas que profundizaremos más adelante en esta sección ya que es importante saber como es el pre-entrenamiento porque nos dice que cosas la red neuronal sabe hacer y qué cosas puede aprender. La arquitectura de este modelo se puede ver gráficamente en la Figura 3.12

Como se puede ver en la Figura 3.12, LXMERT toma dos entradas: la imagen y su oración asociada (descripción o pregunta). Cada imagen es representada como una secuencia de objetos, y cada oración es representada como una secuencia de palabras. La idea es que luego de una combinación de varias capas de cada encoder,

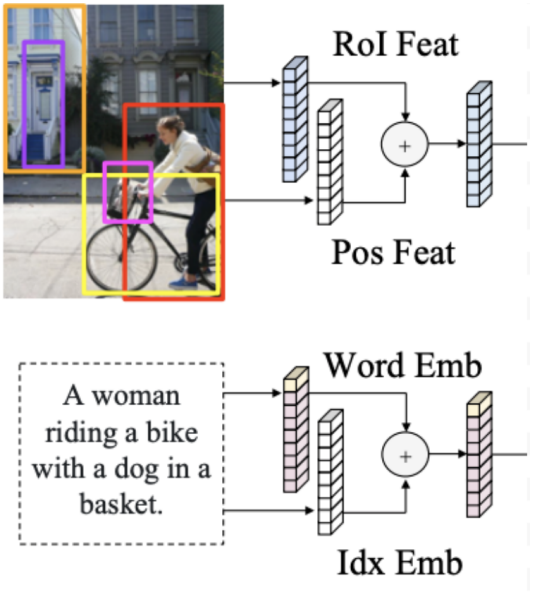


Figura 3.13: Las entradas de LXMERT y sus respectivos embeddings

el modelo sea capaz de generar representaciones visuales, de lenguaje, y de ambas modalidades mezcladas, que son las salidas que se pueden ver en la figura como "Vision Output", "Language Output" y "Cross-Modality Output" respectivamente. Ahora continuemos describiendo cada componente con un poco más de detalle.

3.4.1. Las entradas y los embeddings en LXMERT

Primero una oración la partimos en una secuencia de palabras, cada palabra se tokeniza usando WordPiece Tokenizer, por lo que tenemos una representación $\{w_1, ..., w_n\}$ de tamaño n. Como se ve en la Figura 3.13, cada palabra wi también tiene su posición absoluta en la oración ($IdxEmbed$). Esto se suele usar en modelos basados en Transformers porque, a diferencia de modelos recurrentes, todas las palabras se procesan al mismo tiempo (en un formato de bolsa de palabras), y necesitamos agregarle noción del orden de los mismos. Pues no es lo mismo "Un perro mordió a una persona" que "Una persona mordió a un perro", están compuestas por las mismas palabras, es decir los mismos sus conjuntos w son iguales, pero su significado es completamente distinto. Para construir el embedding final h_i se realiza, entonces:

$\hat{w_i} = WordEmbed(w_i)$
$\hat{u_i} = IdxEmbed(i)$
$\hat{h_i} = LayerNorm(\hat{u_i} + \hat{w_i})$

Con respecto a la imagen, se corre un detector de objetos, en nuestro caso FasterRCNN, y se obtienen m objetos $\{o_1, ..., o_m\}$ de la imagen. Cada objeto ojestá representado por su posición p_j en la imagen, es decir, las coordenadas de su respectiva "bounding box", y su vector de dimensión 2048 f_j que representa las feautures de ese pedazo de imagen. Para obtener el embedding final v_j se realiza algo similar al canal de lenguaje porque las posiciones en donde cada objeto está son importantes ya que nos va a permitir hacer relaciones entre objetos.

3.4.2. Los Encoders de LXMERT

En LXMERT, luego de las capas de embedding, se aplican los respectivos encoders, el "Language Encoder" y el "Object-Relationship Encoder", cada uno de ellos se enfoca en su modalidad (lenguaje y visión respectivamente). Cada capa contiene un módulo de mecanismo de auto atención ("Self" que proviene de "Self Attention"), un módulo de capas densas (es otra forma de nombrar a un Multi Layer Perceptron), cada uno de ellos tiene una conexión residual y entre módulo y módulo se hace una normalización de capa. Estos encoders se pueden apilar, es decir hay varias capas de encoders, en la Figura 3.14 se puede ver cuantas capas de cada encoder hay, esos valores son N_R y N_L respectivamente, que son hiperparametros del modelo. Lo importante de estos encoders es que podemos obtener matrices de atención que relacionan cada objeto con todos los demás en el caso del "Object-Relationship Encoder" y matrices de atención que relacionan cada palabra con las otras en el "Language Encoder".

Luego, la salida del último encoder de lenguaje y la salida del último encoder visual se conecta a un "Cross-Modality Encoder" y luego estos mismos se apilan Nx veces. Este último tipo de encoder es muy interesante ya que es el que relaciona las dos modalidades. Cada uno está compuesto por dos "Cross-Attention", dos "Self-Attention" y dos "Fully Connected Layers". Las "Cross-Attention" funcionan igual que las "Self-Attention" o como cualquier otro mecanismo de atención, pero aquí tendremos uno que relaciona el lenguaje contra la visión, y otro que relaciona la

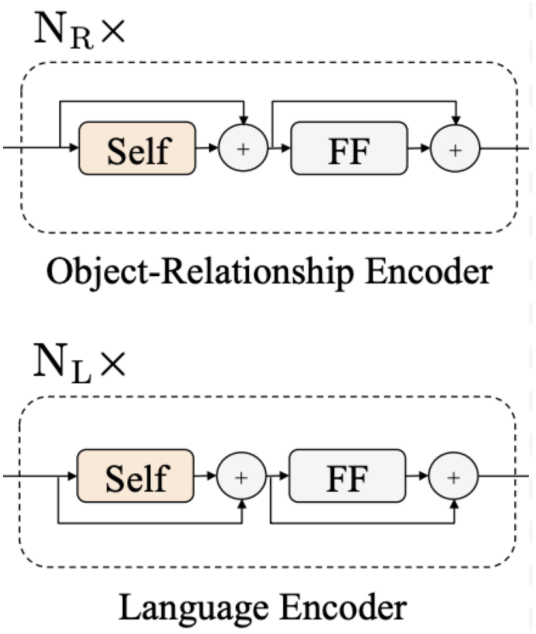


Figura 3.14: Encoders de modalidad singular. “Object Relationship Encoder” encargado de modelar relaciones entre objetos de la imagen. “Language Encoder” encargado de relacionar las palabras del texto de entrada. La salida de ambas luego se usa como entrada del siguiente encoder que mezcla ambas modalidades para aprender tanto del lenguaje como de la visión.

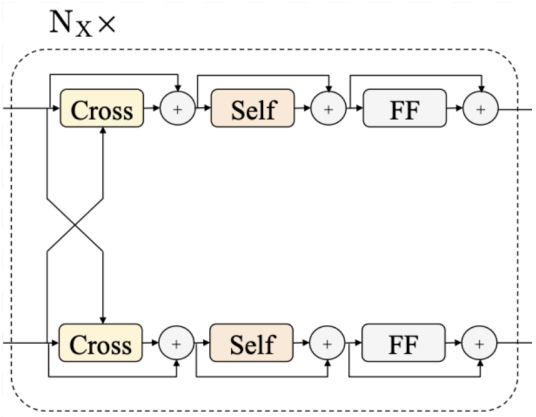


Figura 3.15: Encoder multimodal. Capaz de relacionar objetos con palabras y viceversa.

visión con el lenguaje. Lo podemos ver más graficamente en la Figura 3.15

Sean $\{h_i^{k-1}\}$ y $\{v_i^{k-1}\}$ la salida de las últimas capas del "Language Encoder" y del "Object-Relationship Encoder" respectivamente. Luego computo ambas "Cross-Attention":

$$\hat{h}_i^k = CrossAtt_{L \longrightarrow R}(h_i^{k-1}, \{v_i^{k-1}, ..., v_m^{k-1}\})$$

$$\hat{v}_i^k = CrossAtt_{R \longrightarrow L}(v_i^{k-1}, \{h_i^{k-1}, ..., h_n^{k-1}\})$$

Con esto podemos obtener para cada palabra un puntaje de atención sobre todos los objetos, y viceversa. Luego aplico "Self-Attention":

$$\tilde{h}_i^k = SelfAtt_{L \longrightarrow L}(\hat{h}_i^k, \{\hat{h}_i^k, ..., \hat{h}_m^k\})$$

$$\tilde{v}_i^k = SelfAtt_{R \longrightarrow R}(\hat{v}_i^k, \{\hat{v}_i^k, ..., \hat{v}_n^k\})$$

Luego se le aplica una red Feed Forward distinta a $\tilde{h}_i^k$ y a $\tilde{v}_i^k$ para finalmeente obtener los outputs del modelo. Para obtener el vector "Cross Modality Output" siguieron la misma idea utilizada en BERT [7]. Es decir se agrega al principio de cada entrada de texto un token [CLS], el correspondiente vector de salida proveniente de este token es usado como la salida multimodal, para esta tesis es un vector muy importante porque se usa para realizar todas las predicciones para el Oráculo.

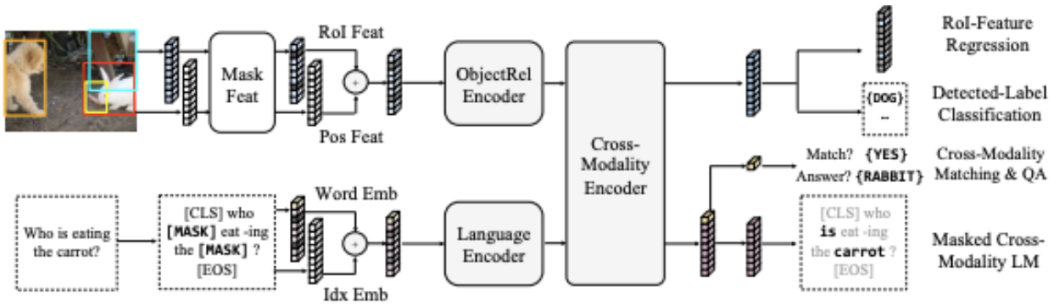


Figura 3.16: Pre-entrenamiento en LXMERT.

3.4.3. Estrategias de Pre-entrenamiento en LXMERT

Como ya mencionamos anteriormente, este modelo fue diseñado para usarse en muchos problemas que combinen visión y lenguaje. La estrategia es pre-entrenar en tareas elementales y luego cuando se quiera pasar a una tarea particular realizar un fine-tuning al modelo. Todas las siguientes tareas se usaron como entrenamiento del modelo al mismo tiempo como se puede ver en la Figura 3.16 que luego explicaremos en detalle.

La primera tarea asociada únicamente al lenguaje se llama "Masked Cross-Modality Language Model". La idea es muy simple y es similar a la utilizada en el conocido modelo BERT, cada palabra es enmascarada con un token especial [**MASK**] al azar con una probabilidad de 0.15, y el modelo tiene que poder predecir qué palabras son las que están escondidas detrás de la máscara. Veamos un ejemplo:

Supongamos que el texto de entrada original es "Hay un perro que está mordiendo el zapato del payaso vestido de rosa", luego la entrada del modelo la convertiremos en:

"[**CLS**] Hay un perro [**MASK**] está mordiendo [**MASK**] zapato del payaso vestido de [**MASK**]"

Por lo que LXMERT va a entrenarse para poder obtener el texto original prediciendo las palabras enmascaradas. De esta forma estamos forzando al modelo de que sea capaz de entender el funcionamiento del lenguaje natural, además como este modelo también ve los objetos de la imagen asociada a la sentencia está aprendiendo el lenguaje de una forma multimodal.

Figura 3.17: Imagen ilustrativa para ejemplificar Masked Object Prediction

La segunda y tercera tarea de pre-entrenamiento es "Masked Object Prediction", es muy similar a la anterior tarea pero está enfocada en los objetos de la imagen. Al azar se van a enmascarar algunos objetos con probabilidad 0.15 (aquí enmascarar es convertir el vector de una región en el vector cero) y la red neuronal va a tener que ser capaz no solo de hacer una regresión de los vectores enmascarados sino también de predecir la clase del objeto enmascarado. Un ejemplo podría ser:

Supongamos que fijamos las regiones a 2, entonces nuestro detector de objetos va a proponer dos regiones como indica la Figura 3.17

Supongamos que se enmascara el auto, entonces el vector que representa el auto dentro de las "RoI features" la vamos a poner en cero, visualmente sería como que la imagen a la que LXMERT tiene acceso se veria como la Figura 3.18, en donde esta el auto tengo pixeles en 0.

En este caso LXMERT debería poder reconstruir las features del auto, y además clasificar el objeto enmascarado con la categoría "Auto". Las tareas mencionadas

Figura 3.18: Imagen ilustrativa para ejemplificar Masked Object Prediction

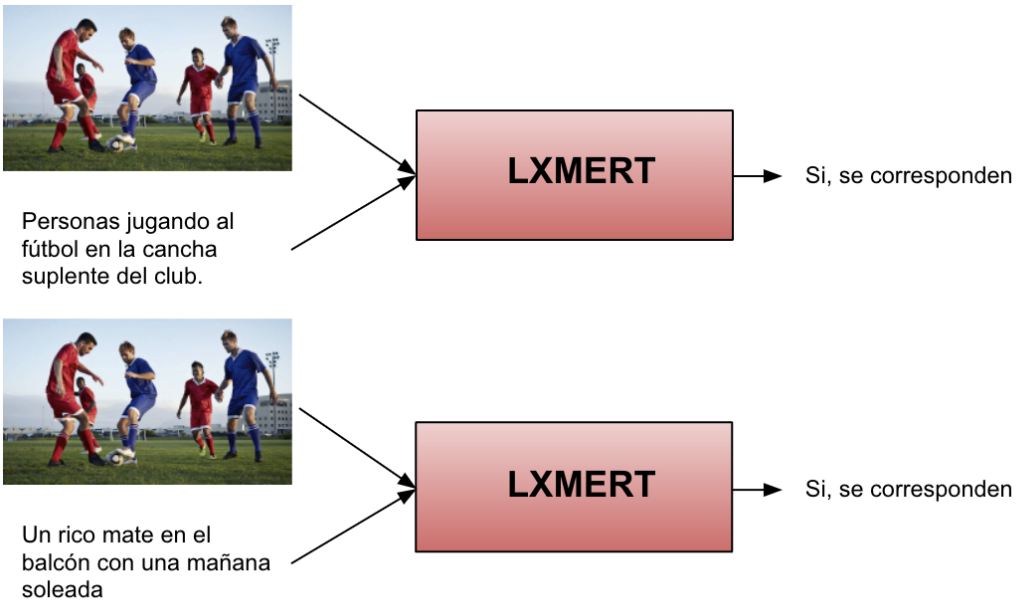


Figura 3.19: Imagen ilustrativa para ejemplificar Cross Modality Matching

se enfocan en una sola modalidad, las tareas multimodalidad son las siguientes y ambas usan el vector "Cross Modality Output" proveniente del token **[CLS]** como comentamos anteriormente para realizar las predicciones.

Una de estas tareas es llamada "Cross Modality Matching" y se basa en reemplazar la entrada de texto con probabilidad 0.5 por otra oración del conjunto de datos y lo que el modelo tiene que ser capaz de predecir es si ese texto es realmente el que corresponde a la imagen o no. La Figura 3.19 intenta ejemplificar esta task.

Por último, entrenaron el modelo en "Image Question Answering". Para agrandar el conjunto de datos 1/3 de las oraciones son preguntas sobre las imágenes. Se le pidió al modelo predecir la respuesta de las mismas solo cuando no se realizaba el cambio de oraciones en la tarea anterior, es decir si se correlacionó la imagen con el texto de entrada.

Con todas estas tareas de pre-entrenamiento obtuvieron un modelo muy robusto que luego evaluaron en varios conjuntos de datos muy conocidos del área del lenguaje y visión realizando un fine-tuning para cada tarea y obtuvieron resultados estado del

arte para todos ellos.

Capítulo 4

Arquitecturas Propuestas y Experimentos

En este capítulo narraremos sobre los distintos experimentos y arquitecturas de redes neuronales utilizadas en esta tesis. Implementamos y analisamos varios modelos y serán expuestos en orden creciente de complejidad. En este trabajo pusimos el esfuerzo en obtener resultados sobre modelos que puedan entender no solo una pregunta particular en el diálogo y la imagen, sino también el historial del diálogo. Los trabajos previos sobre el dataset GuessWhat [32] solo utilizan como entrada de sus modelos la pregunta actual, y no su contexto para el modelo del oráculo. Por esto mismo, al no haber trabajo en modelar historia de los diálogos para esta tarea, realizamos experimentos simples para proponer un punto de partida para futura investigación.

En el Capítulo 2 de este trabajo introdujimos brevemente el modelo base que proponen en [32], en los experimentos de esta tesis vamos a utilizar como modelo base el propuesto en [30]. Este mismo propone usar LXMERT, que es una arquitectura neuronal basada en transformers similares a las descritas en el Capítulo 3, modificado para la tarea de el Oráculo. Los resultados ofrecieron el estado del arte y el primer experimento de esta tesis fue poder reentrenar el modelo y reproducir los resultados. Lo que parece algo muy sencillo pero la verdad es que no lo es, es un arduo trabajo ya que el modelo es muy complejo y necesita de un gran entendimiento de muchas

tecnicas en el area de la inteligencia artificial. La Sección 4.1 introduce este capítulo y la metodología usada en el diseño de las arquitecturas y los experimentos, discute también las métricas y el esquema de clasificación de las preguntas. La sección 4.2 describe cómo LXMERT se adaptó en [30] para la tarea del Oráculo sin considerar la historia del diálogo. En la sección 4.3 describiremos un enfoque para agregar historia modificando la pregunta de entrada al modelo. En la sección 4.4 introduciremos algunos enfoques para agregar la historia combinando las salidas del modelo sobre los distintos turnos del diálogo basado en redes LSTM.

4.1. Introducción y Metodología

Es importante aclarar que esta red neuronal que proponen en [30] no mira la historia del diálogo, sigue viendo solamente la pregunta actual como independiente de las anteriores. Pero lo usaremos como punto de partida para construir modelos simples que sí utilicen el contexto del diálogo. En esta sección pondremos el enfoque en los distintos experimentos realizados y sus respectivas arquitecturas. Luego, en el Capítulo 5, presentaremos y discutiremos algunas hipótesis sobre los resultados.

Cada experimento tendrá su motivación, su explicación y sus resultados en tablas que comparan contra los demás experimentos. Siempre cuando se hacen experimentos es importante aclarar y explicar cómo se evalúan los modelos y qué es lo que se muestra precisamente en los resultados. En este trabajo usaremos las siguientes estrategias de evaluación que se describen a continuación.

En primer lugar reportaremos precisión total. Su formulación matemática cantidad de predicciones correctas realizadas por el modelo divida por la cantidad total de predicciones intentadas. En el caso del Oráculo es:

$$\frac{\textit{cantidad de preguntas respondidas correctamente}}{\textit{cantidad total de preguntas en el conjunto de datos}}$$

En el caso del Oráculo los textos que procesamos son preguntas que hacen referencia no solo a objetos en una imagen, sino también a secciones de la misma, y no solo a la posición de un objeto sino a características del mismo como color, textura, etc. Por lo que en este trabajo tomamos la decisión de clasificar las preguntas en distintos

tipos con un clasificador determinístico basado en palabras clave propuesto en [23] que nos va a permitir hacer métricas y análisis.

Según esta clasificación una pregunta pertenece a una o más de las siguientes clases:

1. Objeto: "¿Es un jarrón?"
2. Super Categoría: cuando la pregunta hace referencia a una categoría grande que representa múltiples objetos de la misma familia, por ejemplo, "frutas" es una super categoría de "banana", "manzana", etc. "¿Es una fruta?"
3. Color: "¿Tiene una gorra roja?"
4. Forma: "¿Es rectangular?"
5. Tamaño: "¿Es una vasija pequeña?"
6. Acción: "¿las personas se sientan en él?"
7. Textura: "¿Es de acero?"
8. Espacial: "¿Está en el cielo?" o "¿Está a la izquierda?"
9. No Clasificada: Este conjunto contiene todas las preguntas que no tienen una clasificación, no entra en ninguna de las categorías anteriores.

Los ejemplos son solo para ilustrar y ayudar al lector, las preguntas suelen ser más complejas y multicategoría, por ejemplo: "¿Es la chica de la izquierda que está jugando baseball vestida de rojo?", es una pregunta de tipo objeto, color, espacial y acción al mismo tiempo. Entonces en la sección de resultados no solo vamos a tener "Accuracy Total", sino que vamos a poder ver y analizar la eficacia del modelo para cada tipo de pregunta. Como el objetivo principal de este trabajo es poder representar de manera simple la historia del diálogo, armamos un conjunto de datos para evaluar los modelos que contiene solamente preguntas en las cuales sólo se pueden responder si tenemos información de los turnos anteriores del diálogo, es decir, son preguntas que o no podríamos responder por sí solas o, que, conociendo la historia del diálogo, nuestra respuesta cambiaría. Las llamamos Preguntas dependientes de la historia. Para ilustrar cómo fueron elegidas estas preguntas veamos algunos ejemplos representativos obtenidos del conjunto de datos con el que se trabajó:

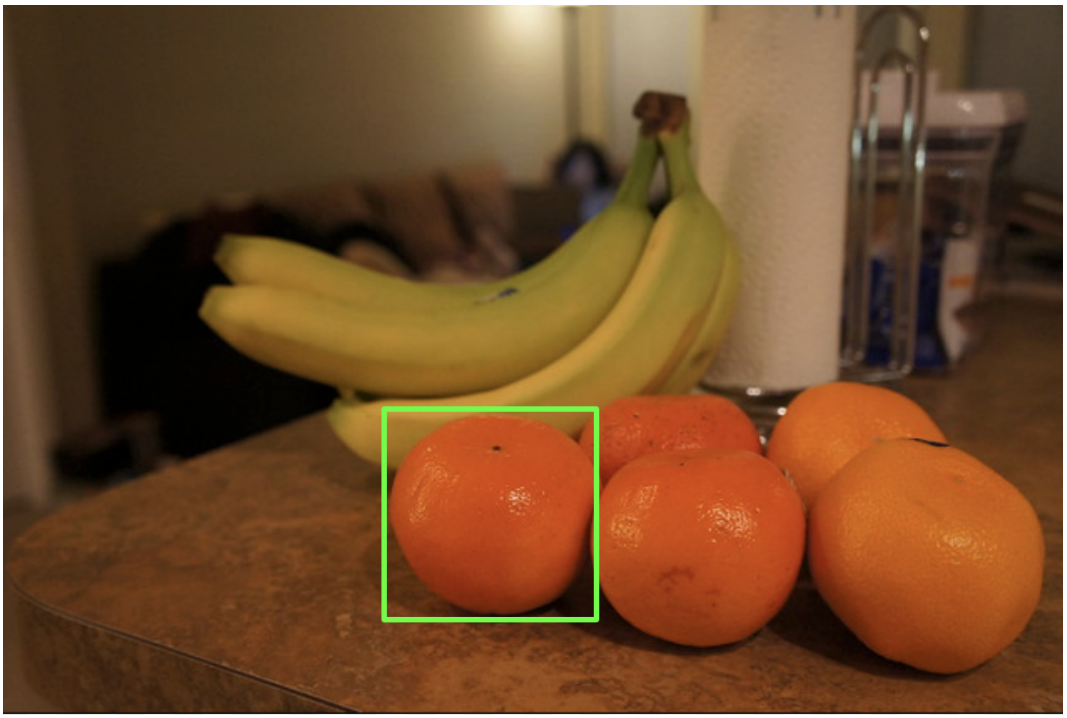


Figura 4.1: Imagen del conjunto de datos con la respectiva bounding box del referente

Supongamos que se está estableciendo un diálogo sobre la siguiente imagen de la Figura 4.1 y el referente es el semáforo recuadrado en verde, y el turno que estamos analizando es la pregunta "¿El objecto está en el medio?".

Pregunta para el lector, ¿Qué respondería? Al menos lo que parece tener más sentido es responder Sí, porque al tener la pregunta suelta solemos referenciarla con el el centro de la imágen, pero no podemos saber si está haciendo referencia de "al medio" de un conjunto de otros objetos o lo que sea que la expresión al medio signifique en el contexto del diálogo, pero como no tenemos acceso a él, podemos no responder correctamente esa pregunta. La respuesta es No, pues veamos el conjunto de preguntas y respuestas anteriores en la Figura 4.2 nos damos cuenta que la reeferncia .ªl medio"no es es el medio de la imagen si no un medio enmarcado por las preguntas anteriores del dialogo.

Eso no sólo ejemplifica cuál fue el criterio para la selección de preguntas dependientes de historia, sino que también ilustra otra vez más la necesidad de hacer que

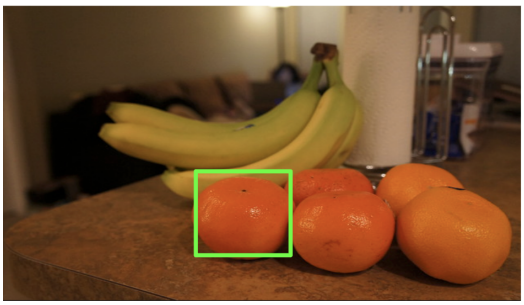


Figura 4.2: Imagen del conjunto de datos con la respectiva bounding box del referente y su dialogo asociado

nuestros modelos no solo vean la pregunta como algo independiente, sino como algo fuertemente ligado a las preguntas anteriores.

4.2. Primera idea de LXMERT aplicado al oráculo

En el Capítulo 3 explicamos cómo funciona el modelo LXMERT, vimos su arquitectura y cómo fue su pre-entrenamiento, en esta sección respondemos la pregunta: ¿Cómo lo adaptamos para nuestro problema? Esta pregunta fue abordada y presentada en [30]. El primer experimento de esta tesis fue poder recrear los resultados obtenidos en dicho estudio ya que son el estado del arte en la tarea del Oráculo para la fecha en la que se escribe este trabajo. Lo que realizamos fue lo que se describe a continuación.

Siguiendo con la arquitectura propuesta en la implementación básica de LXMERT [28] se instanció un modelo LXMERT con los siguientes hiper parámetros:

1. Cantidad de objetos o regiones detectados en cada imagen por FasterRCNN [20] : 36
2. Modelo para la extracción de features: ResNet [11]
3. $N_X = 5, N_R = 5, N_L = 9$

Luego reemplazamos los features de la última región encontrada por el detector por los features del referente del Oráculo, es decir, por el objeto que se está intentando descubrir. De esta forma fue como le agregaron a LXMERT el conocimiento del

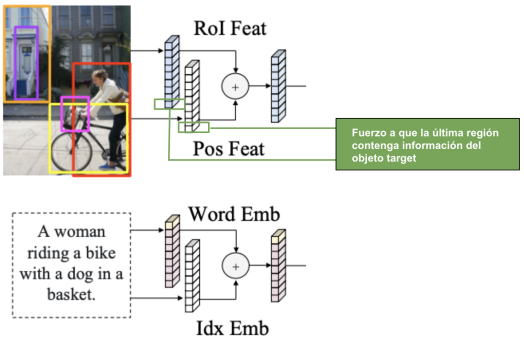


Figura 4.3: Imagen ilustrativa para ejemplificar como se reemplaza las features del último objecto por las del referente

referente (Figura 4.3). Decidimos reemplazar la última sección porque FasterRCNN identifica las regiones más importantes primero y las ordena de más saliente a menos saliente. Se considera que la detección de saliencia es un mecanismo de atención clave que facilita el aprendizaje y la supervivencia al permitir que los organismos concentren sus limitados recursos perceptivos y cognitivos en el subconjunto más importante de los datos sensoriales disponibles [REF]. Por lo tanto la última región identificada, la región 36 es la menos importante de la imágen y reemplazarla causa poca pérdida de información. Es una idea cognitivamente motivada y efectiva consiguiendo una mejor precisión total del 83 % [30]. Por último utilizaron el vector de salida multimodal, es decir, el "Cross-Modality Output" y sobre el mismo construyeron un Multi Layer Perceptron de 3 capas, la primera con 1024 neuronas, las segunda con 128, y la última con 3 neuronas como se intenta ejemplificar visualmente en la Figura 4.4. En todas ellas usaron ReLU como función de activación excepto por última que se usó una función "SoftMax" que permite obtener como salida de la red un vector de probabilidades que nos va a servir para realizar la clasificación en Sí, No, y N/A. Para entrenar todo este modelo hemos usado como optimizador Adam con un *learning rate* de 10^{-5} , y como función de costo "Cross entropy". Para recrear los resultados usamos batch size = 32 y entrenamos el modelo en 2 GPU de la computadora Nabucodonosor del Centro de Computación de Alto Desempeño de la Universidad Nacional de Córdoba.

Los resultados de todos los experimentos se ven en el Capítulo 5.

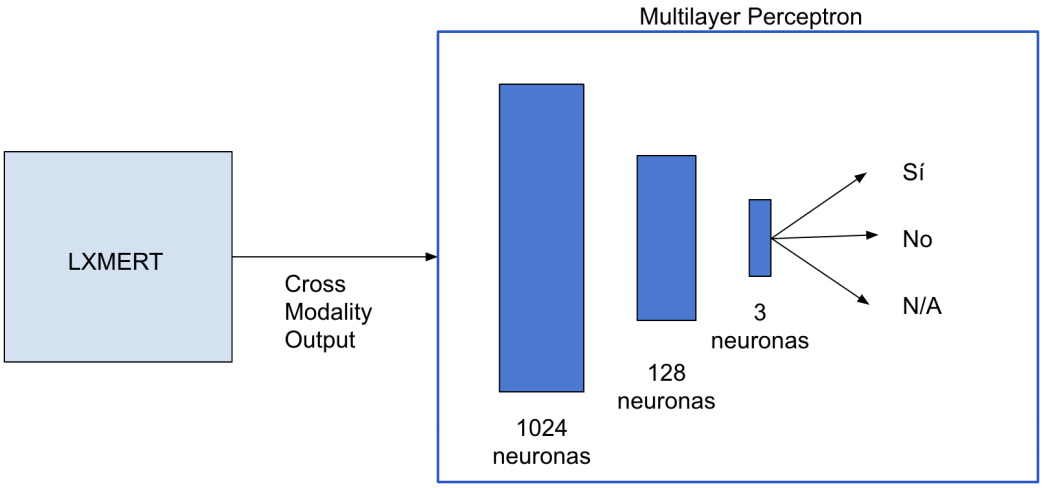


Figura 4.4: Imagen ilustrativa para ejemplificar la MLP que se colca por encima de una de las salidas de LXMERT en [30] para eel Oráculo

4.3. Representando historia usando lenguaje

En el experimento anterior, que es nuestro experimento base que reproduce trabajo previo, todavía no se está haciendo uso de la historia del diálogo. ¿Cómo podríamos usar ese modelo que es muy robusto para que también sea consciente de la historia? Vamos a intentar responder esa pregunta en los siguientes experimentos. A uno se le podría ocurrir algo muy simple, ya que LXMERT está pre entrenado en tareas complejas y que no solo aceptan una oración o una pregunta sino que varias oraciones, podemos concatenar la historia al texto de entrada. No solo eso, sino que vamos a hacer uso de la idea que se concluye en [10] que defiende que las últimas preguntas suelen ser más largas, positivas (se responden con Sí) y contienen información y detalles más adecuados para adivinar el referente . Los primeros experimentos fueron aplicando esta idea: concatenar la historia positiva.

Por ejemplo, si tomamos como entrada la imagen y diálogo de la Figura 4.2 el computo del modelo base seria como mostramos en la Figura 4.5 mientras que nuestro nuevo experimento seria como ilustra la Figura 4.6. Como es obvio, el primer modelo no tiene nada de informacion de lo que ocurrió antes de la pregunta actual, mientras que nuestro nuevo experimento le entrega al modelo al menos informacion

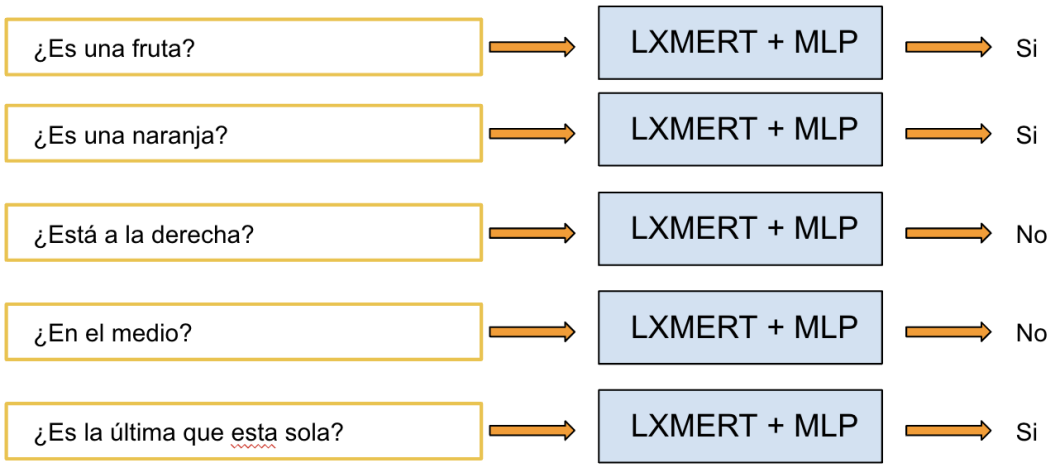


Figura 4.5: Análisis paso a paso del modelo base para el dialogo de la Figura 4.2

positiva de lo ocurrido anteriormente en forma de texto.

Es necesario aclarar que los signos de pregunta de las preguntas que son de turnos anteriores se quitan y se cambian por '.' (puntos). La intuición es que se van armando textos que le agregan más descripción e información que de cierta forma referencian al objetivo. Pues en el ejemplo anterior, en el último turno estamos cambiando "¿Es el que está al lado del de remera naranja?" por "Es humano. Está a la derecha. Es hombre. Tiene remera azul oscuro. ¿Es el que está al lado del de remera naranja?". El modelo a la hora de contestar la pregunta, tiene información importante del objeto que le puede servir para contestar ya que, como vimos anteriormente, hay preguntas que son imposibles de contestar correctamente si no sabemos y no somos conscientes de los turnos anteriores con sus respuestas. En este trabajo primero probamos solamente evaluar la red neuronal en el conjunto de testeo realizando esta técnica y también por otro lado realizamos un fine tuning, es decir, se re-rentreno el modelo concatenando la historia, ya que es intuitivo pensar que podría ser mejor para la red que "acostumbrarse" a cómo va a ser la entrada (el formato de concatenar y cambiar '?' por '.').

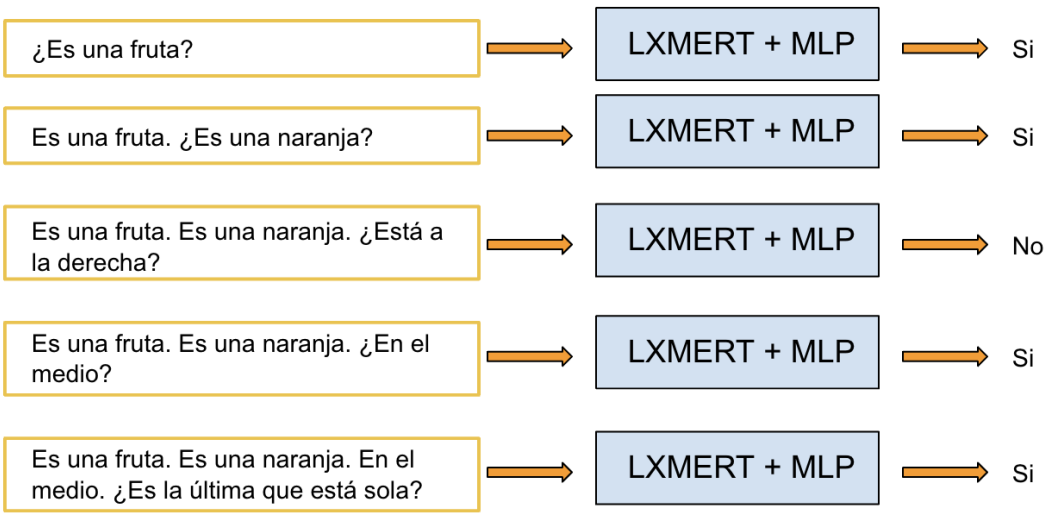


Figura 4.6: Análisis paso a paso del modelo que concatena la historia positiva para el dialogo de la Figura 4.2

4.4. Representando historia multimodal con late fusion

Queremos un modelo que pueda usar la historia del diálogo para responder las preguntas. Los humanos hacemos esto con la memoria, es decir, tenemos almacenada la información de lo que ocurrió antes y luego con eso podemos deducir una respuesta usando nuestra memoria. Además somos capaces de cambiar el contexto sobre el que interpretamos una pregunta a medida que un diálogo progresa. Por esto, el orden en que procesamos una historia de un dialogo es importante, y podemos ejemplificarlo usando la idea de foco sobre una imagen. Supongamos que existe algún dialogo $D = \{\textit{¿Está a la derecha?}, \textit{si}, \textit{¿Está a la izquierda?}, \textit{si}, \textit{¿Está a la izquierda?}, \textit{si}\}$, si yo estoy procesando la última pregunta de D no es lo mismo tomar como historia "¿Está a la derecha?, si, ¿Está a la izquierda?, si"que "¿Está a la izquierda?, si, ¿Está a la derecha?, si"; pues si bien uno podria decir que es la misma historia porque son las mismas preguntas con las mismas respuestas, pero el foco que causan en la imagen por el que analiza el diálogo es distinto, te llevan a enfoncarte en distintas partes de una imagen, por eso nuestra hipotesis es que es muy importante tener en cuenta el orden de las preguntas en nuestros modelos. La Figura 4.7 intenta ilustrar este efecto. donde dado el diálogo D la interpretacion de la historia correctamente

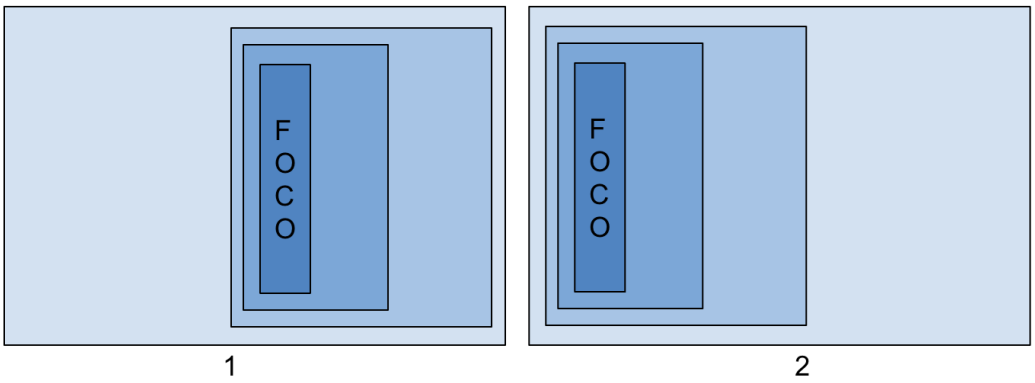


Figura 4.7: Ilustrativameente podemos ver los dos focos que se generan sobre alguna imagen usando el diálogo $D = \{¿Está\ a\ la\ derecha?, si, ¿Está\ a\ la\ izquierda?, si, ¿Está\ a\ la\ izquierda?, si\}$ pero la Imagen 1 usa la historia en el orden correcto mientras la imagen 2 usa la historia sin tener en cuenta el orden, en este caso, cambia de orden las primeras dos preguntas. El foco correcto es el de la Imagen 1, por esta razon creemos que hay que ser cuidadoso con nuestros modelos para representar la historia consciente del orden

deberia guiarnos hacia el foco que propone la

Cuando vemos y analizamos todo lo que necesitamos, automáticamente se nos viene a la mente las redes neuronales recurrentes LSTM ("Long Short Term Memory", Figura X), que explicamos detalladamente en el Capítulo 3 de este trabajo. En esta sección proponemos algunas arquitecturas que exploran el uso de LXMERT con LSTMs.

La intención de los siguientes experimentos es que nuestro modelo sea capaz de guiar su atención de las regiones de la imagen por el foco que va generando el diálogo pero al mismo tiempo, como mencionamos anteriormente, intentar hacerlo de una manera simple. Por ejemplo, si miramos el ejemplo de la Figura 4.8, humanamente podemos graficar los focos en donde debería estar puesta la atención. Una pregunta dentro de ese diálogo en particular que para responderse correctamente se necesita conocer de la historia es "En el medio?". En la Figura 4.8 se muestra como cambia el foco y como va cambiando el significado de estar en el medio. Lo deseado es que los

modelos puedan aprender esas relaciones, en este trabajo experimentamos en darle esa responsabilidad a los "hidden state" de las celdas LSTM.

4.4.1. LSTM muchos a muchos

Comúnmente las redes recurrentes se usan para codificar oraciones, es decir secuencias de palabras, en estas mismas la entrada de cada celda es el embedding asociado a esa palabra. En este experimento ya no analizamos cada pregunta individualmente sino que la entrada de nuestro modelo va a ser el diálogo, es decir una secuencia de preguntas, donde cada pregunta es una secuencia de palabras y por supuesto la imagen en cuestión como se muestra en la Figura 4.9. Lo que hicimos fue, para cada pregunta, computar los vectores que la representan utilizando el "Cross Modality Output" de LXMERT, por lo que obtenemos una secuencia de vectores que representan al diálogo. Sobre estos mismos vamos a aplicar una LSTM muchos a muchos, en donde con cada output de cada celda vamos a computar la respuesta para cada pregunta construyendo una pequeña capa lineal para poder obtener la clasificación.

Este modelo es similar al estado del arte en que usa la misma salida de LXMERT para hacer las predicciones, con la diferencia de que aquí los pesos de las capas de LXMERT no se usan en el backpropagation del modelo ya que los vectores "Cross Modality Output" se pre-computaron, y sobre estos mismos vectores se cambia el MLP por una celda LSTM. Este cambio se realiza con la idea de que cada vez que el modelo va a contestar una pregunta, tiene a su disposición no solo el vector de entrada si no también el vector "hidden state" de la celda anterior que, en principio, representa los turnos anteriores con la idea de que en estos vectores las LSTMs puedan ver las atenciones que realiza LXMERT y entender las relaciones de las mismas a través del tiempo enfocándose en las regiones que el foco del diálogo va enmarcando (más detalle en Capítulo 5). Luego de analizar el modelo sobre algunos ejemplos, podemos detectar dos errores fundamentales. Si bien le estamos dando al modelo el historial del diálogo, no le estamos dando información de si cada pregunta fue respondida positiva o negativamente, es decir le estamos dando muchísima información pero no le estamos diciendo cómo usarla. Además este modelo no es aplicable en el mundo real porque espera quee ya le des el dialogo completo y en una aplicación rel solo tenes la pregunta actual y las anteriores a la misma.

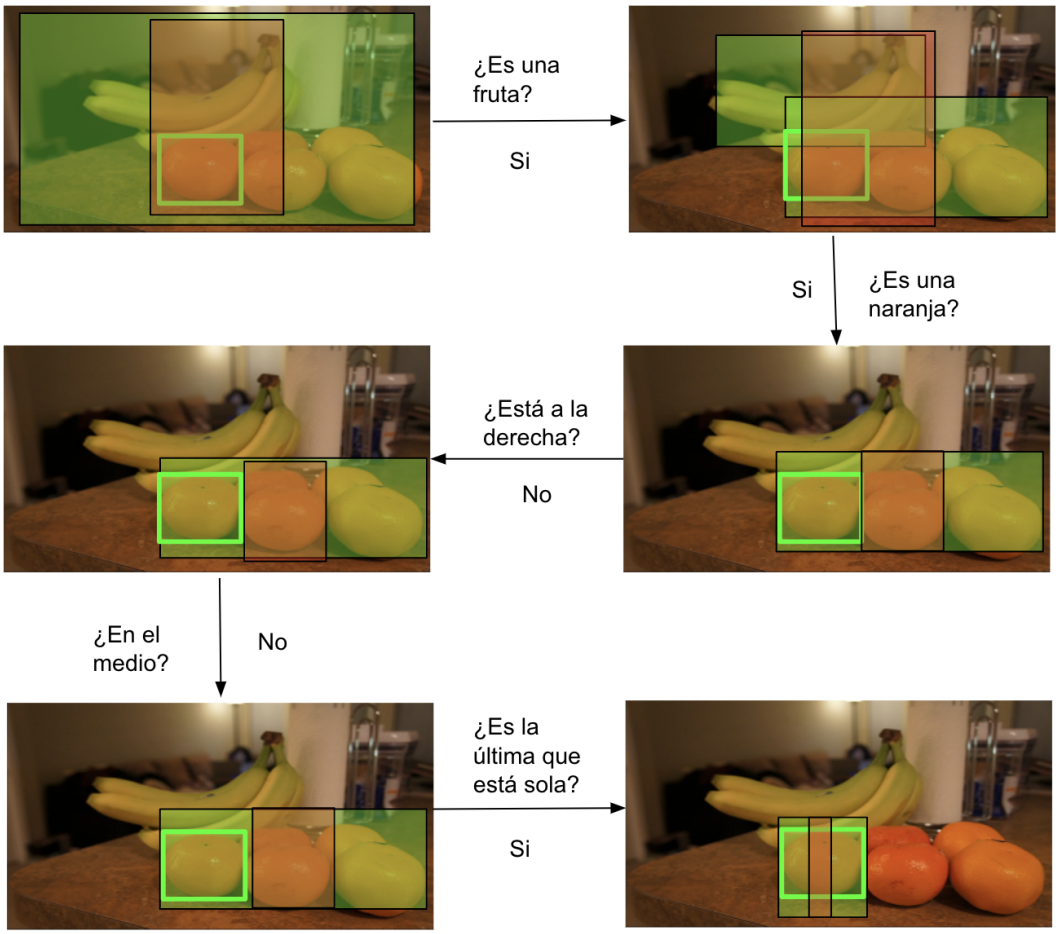


Figura 4.8: En este gráfico se puede ver como humanamente se cambia el foco (recuadro verde con transparencia) a medida que transcurre el diálogo y cómo cambia (recuadro rojo) el significado de "medio", pues el mismo depende del foco, de lo que enmarca el historial del diálogo.

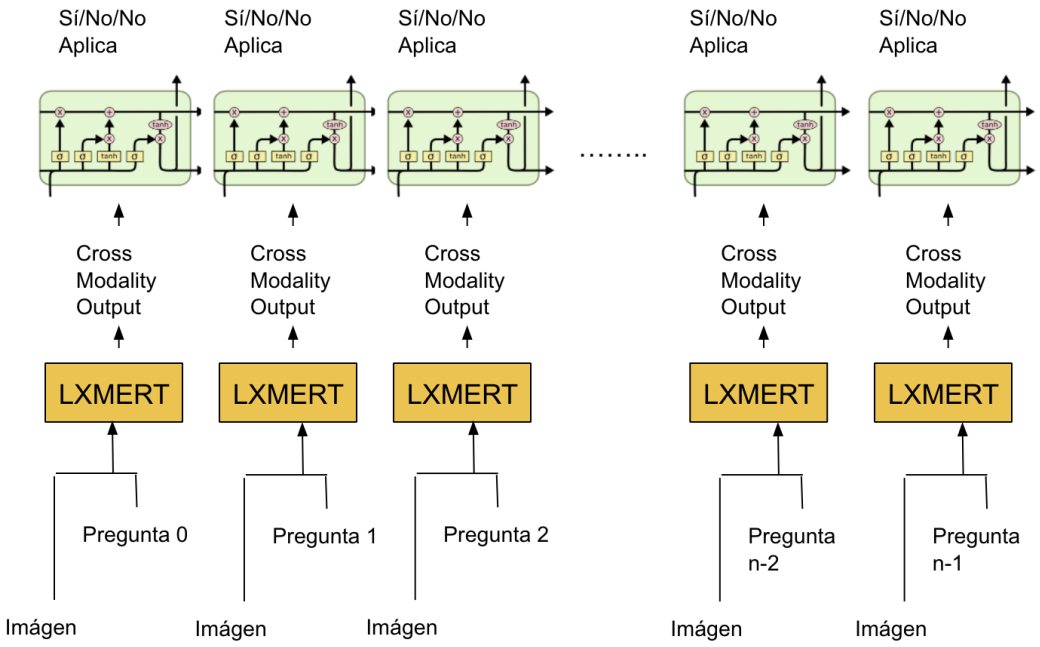


Figura 4.9: Arquitectura del modelo que usa LXMERT con LSTMs muchos a muchos. En un mismo modelo se computa todo el diálogo. Para cada par (imagen, pregunta) obtengo el output de modalidad cruzada de LXMERT por lo que obtengo una secuencia de vectores que va a ser la entrada de las celdas LSTM en donde con cada una de ellas voy a obtener la respuesta a su pregunta correspondiente.

4.4.2. LSTM muchos a uno

Ahora vamos a mejorar algunas cosas del modelo anterior. Primero, la entrada no será más el diálogo completo sino la pregunta actual y las preguntas anteriores, y la salida va a ser una sola: la respuesta del turno en cuestión. Y además solo vamos a acarrear las preguntas que su verdadera respuesta es "Sí", parecido a lo que haciamos en el modelo simple que concatenaba historia positiva.

Ahora graficamente el computo del diálogo seria como lo muestran las Figuras 4.10, 4.11, 4.12, 4.13, 4.14.

La idea principal es que nuestros modelos puedan hacer uso de la historia sin perder la precisión que se tiene viendo cada pregunta independientemente, como se puede ver en el siguiente capítulo en los modelos basados en LSTMs explicados anteriormente la precisión total es menor a nuestro modelo base por lo que se realizó un experimento en donde solamente hacemos uso de las celdas de LSTM para los turnos anteriores al actual y luego concatenamos el último "hidden state" al vector de salida de LXMERT para el turno actual. De esta forma obtenemos un vector donde por una parte tenemos representada la historia y por otra, el vector de modalidad cruzada de la pregunta que está siendo procesada. Arriba del mismo colocamos un MLP para realizar las predicciones. La ejecución del modelo se puede ver en las Figuras 4.15, 4.16, 4.17, 4.18, 4.19 donde todos los turnos anteriores que son respondidos con "Sí"se pasan por LSTMs como hacíamos en experimentos anteriores y para el caso de la pregunta actual hacemos lo mismo que nuestro modelo base. Luego concatenamos y predecimos con un Multi Layer Perceptron. De nuevo, todos los LXMERT los utilizamos con sus pesos congelados.

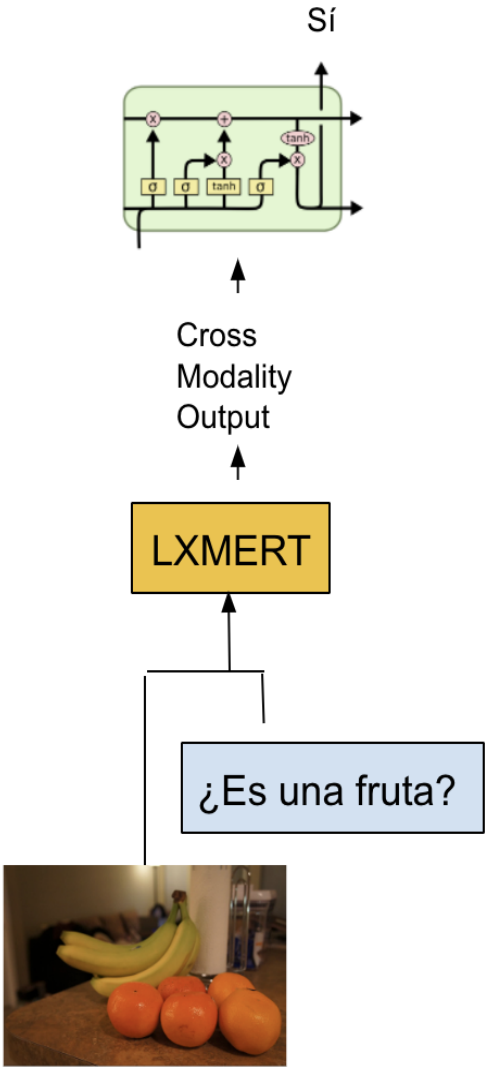


Figura 4.10: Procesamiento del primer turno para el modelo de late fusion con LSTM muchos a uno.

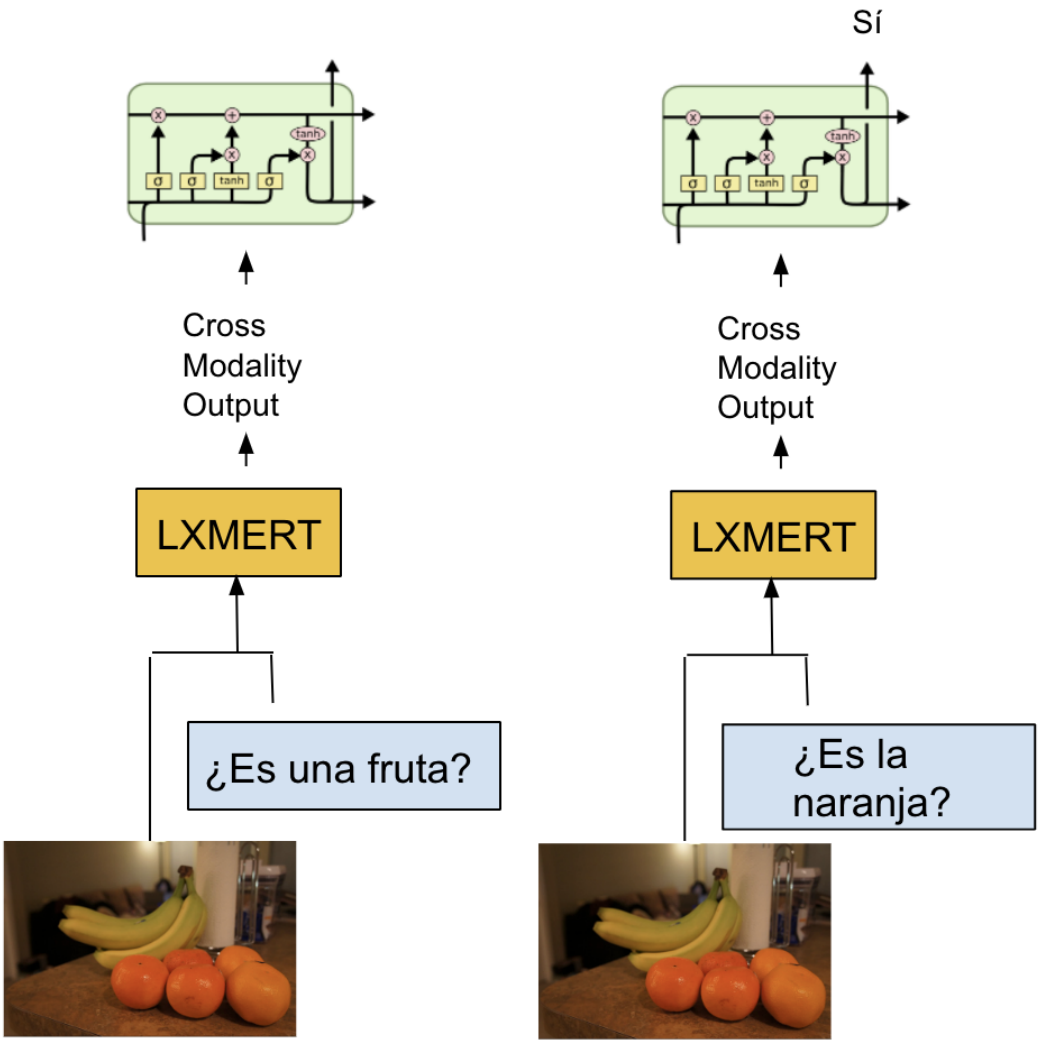


Figura 4.11: Procesamiento del segundo turno para el modelo de late fusion con LSTM muchos a uno

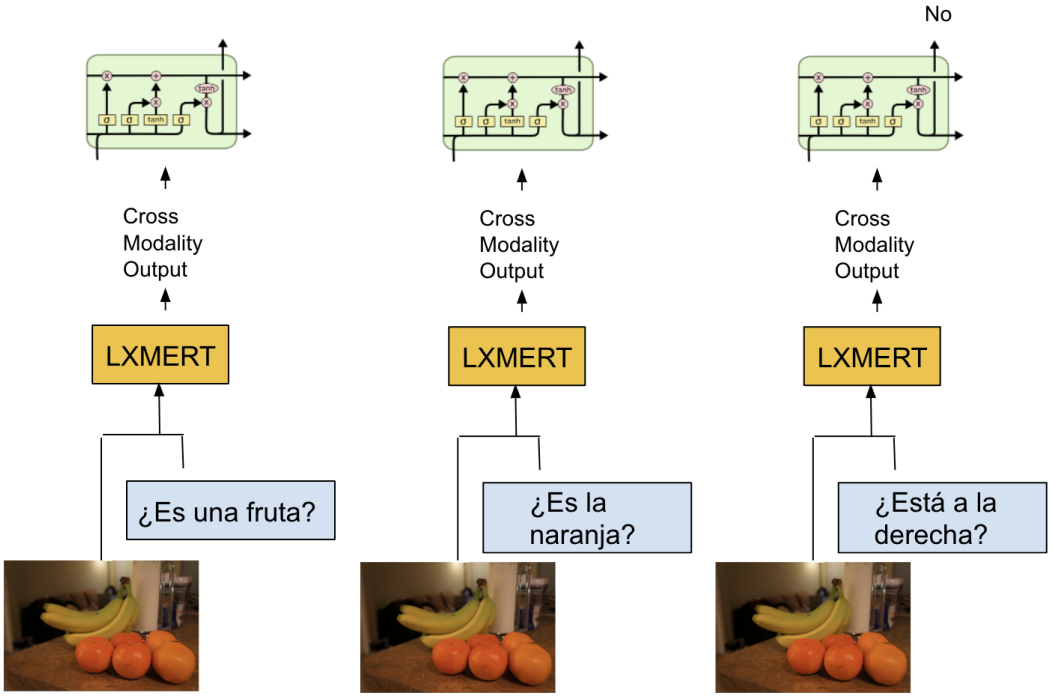


Figura 4.12: Procesamiento del tercer turno para el modelo de late fusion con LSTM muchos a uno

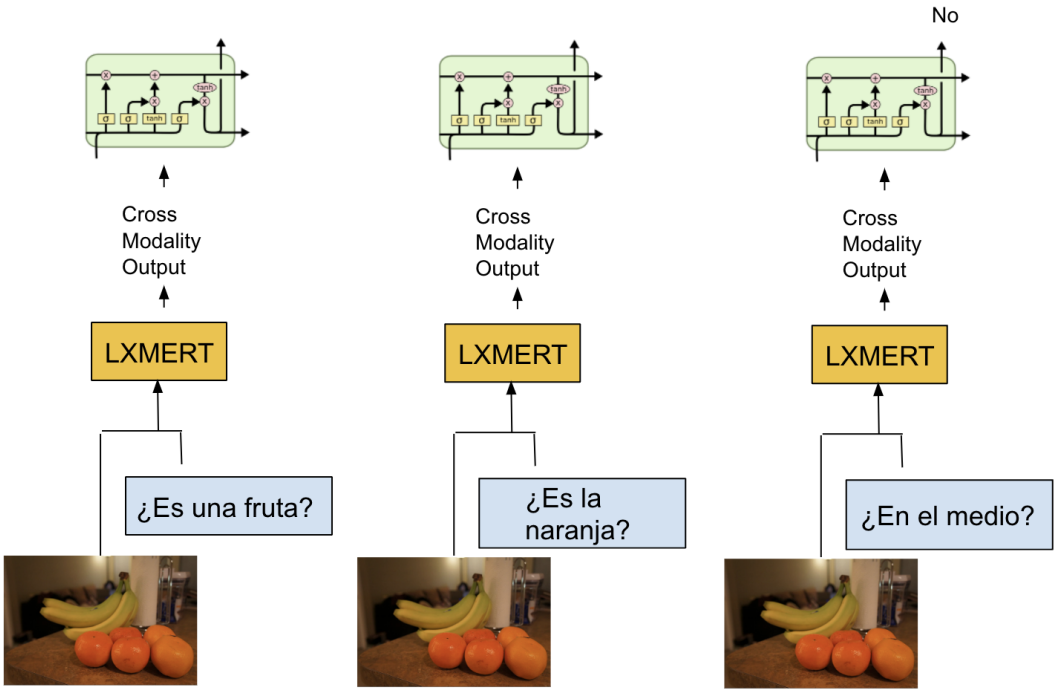


Figura 4.13: Procesamiento del cuarto turno para el modelo de late fusion con LSTM muchos a uno

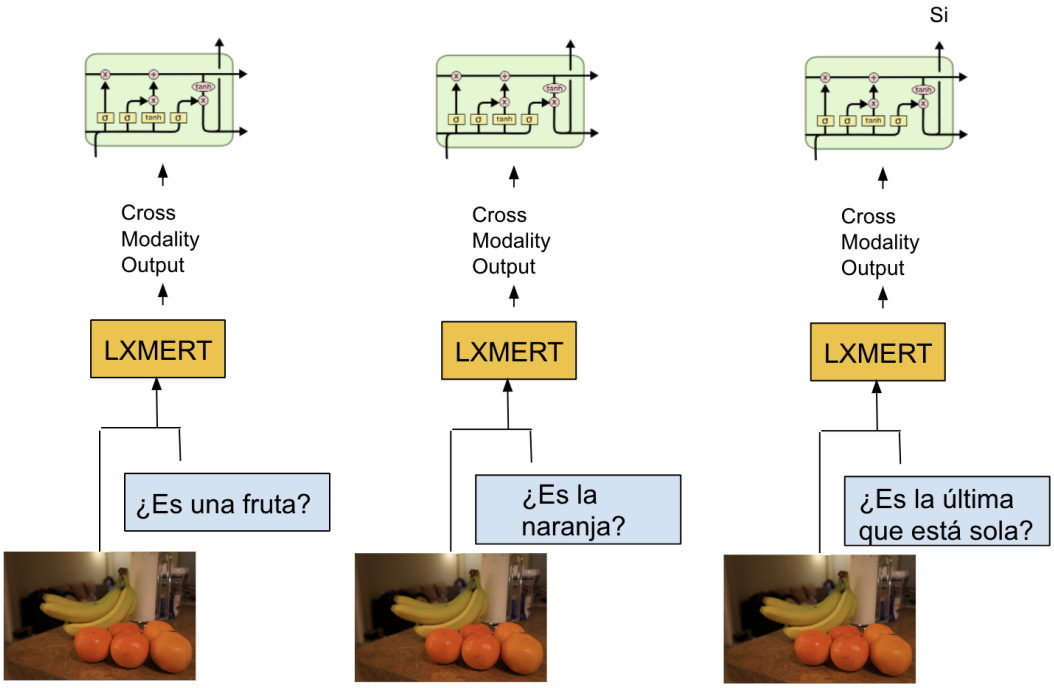


Figura 4.14: Procesamiento del quinto turno para el modelo de late fusion con LSTM muchos a uno

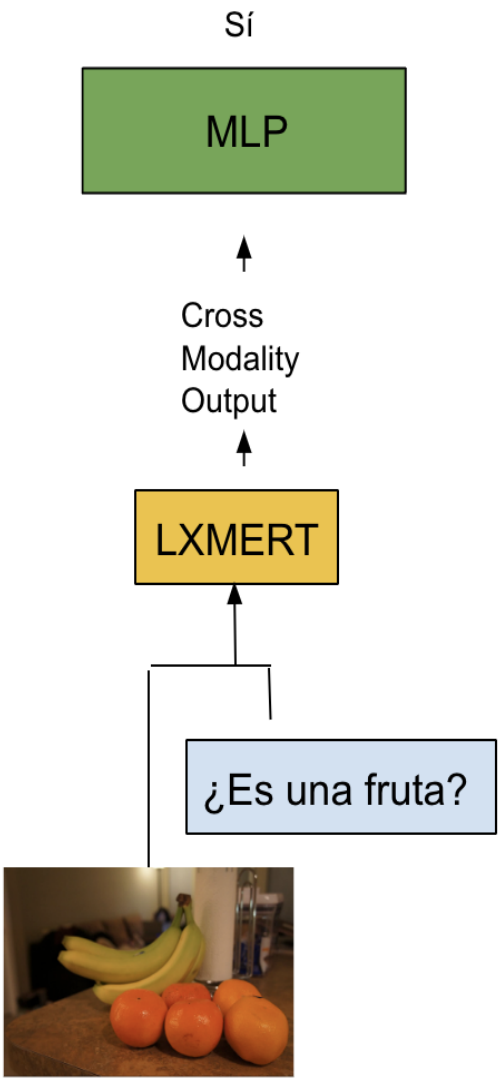


Figura 4.15: Procesamiento del primer turno para el modelo de late fusion de historia con LSTM y pregunta actual con MLP.

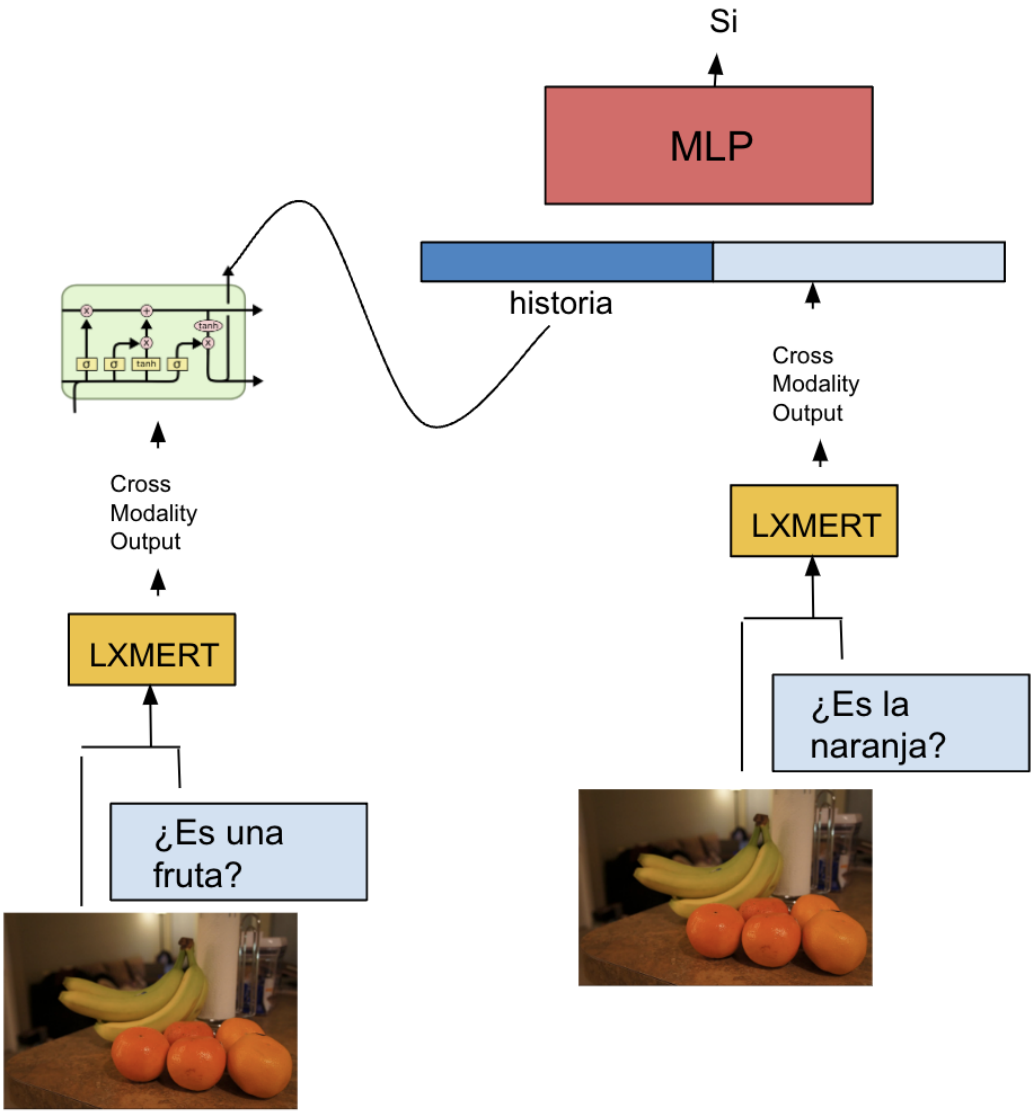


Figura 4.16: Procesamiento del segundo turno para el modelo de late fusion de historia con LSTM y pregunta actual con MLP.

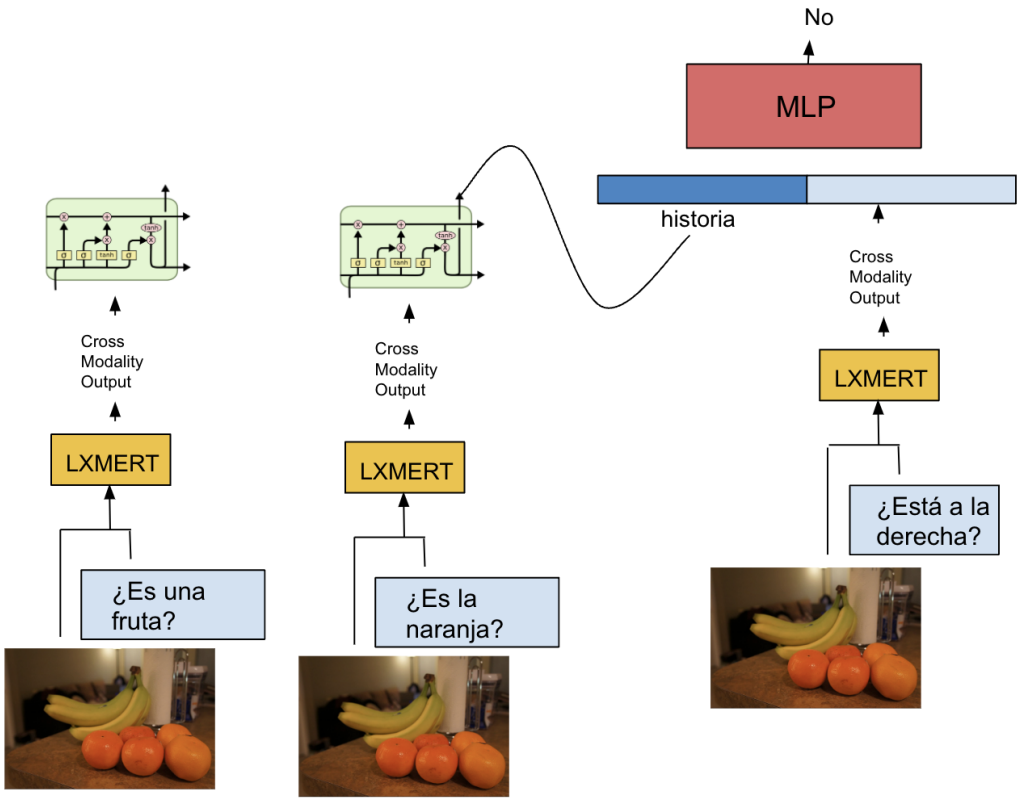


Figura 4.17: Procesamiento del tercero turno para el modelo de late fusion de historia con LSTM y pregunta actual con MLP.

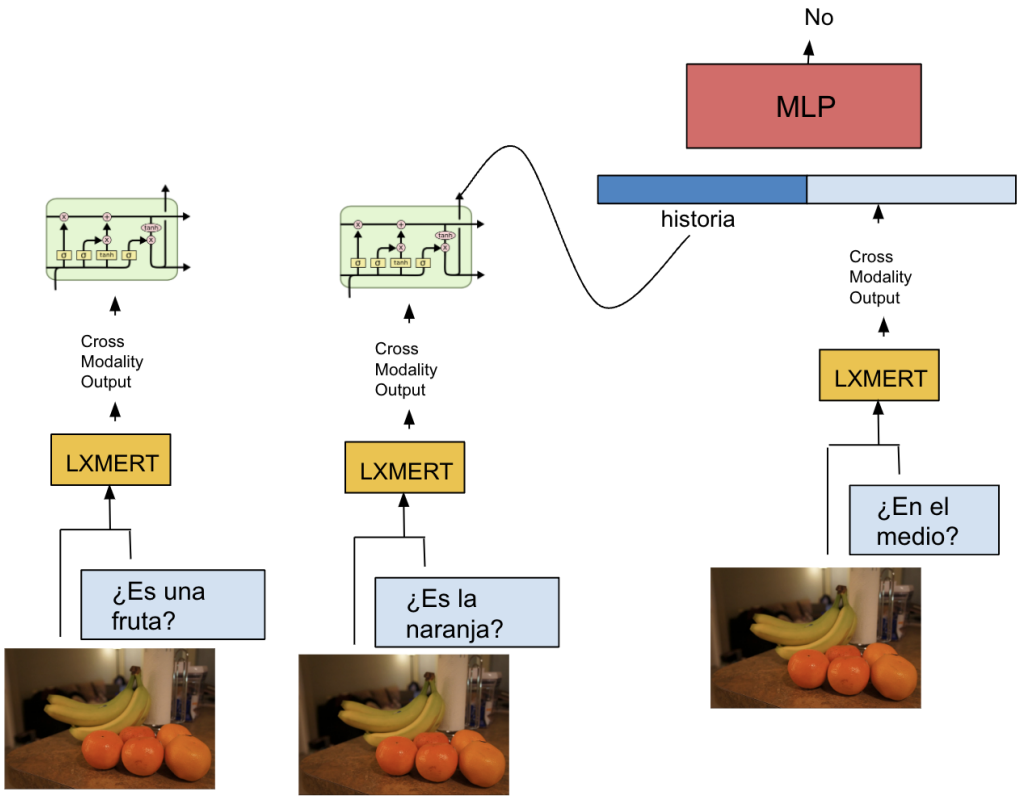


Figura 4.18: Procesamiento del cuarto turno para el modelo de late fusion de historia con LSTM y pregunta actual con MLP.

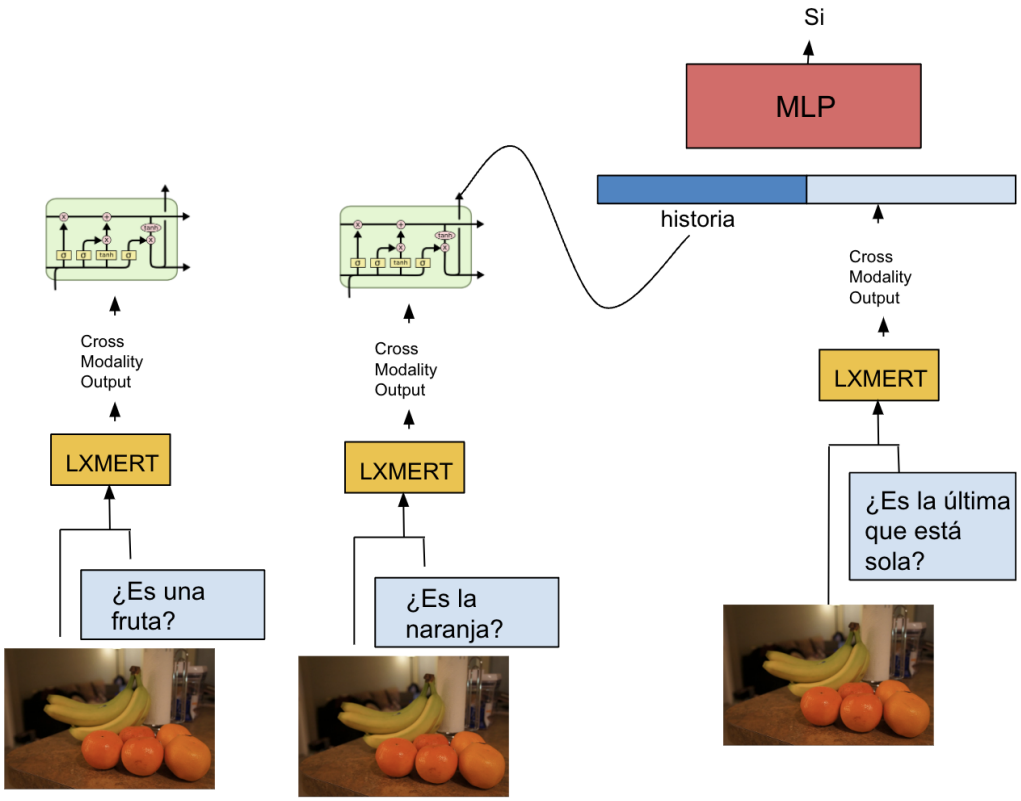


Figura 4.19: Procesamiento del quinto turno para el modelo de late fusion de historia con LSTM y pregunta actual con MLP.

Capítulo 5

Resultados y discusión

En este capítulo mostraremos los resultados obtenidos en los experimentos anteriormente explicados. No vamos a poder hacer afirmaciones sobre las razones de porqué, por ejemplo, un experimento no obtuvo mejores resultados, ya que un problema grande que tienen los modelos basados en redes neuronales es su interpretabilidad, es decir, no es tarea trivial explicar porqué y cómo un modelo particular toma una decisión, por eso mismo se realizarán hipótesis. Primero veremos las métricas anteriormente discutidas en tablas, luego veremos matrices de confusión para poder ver en más detalle cómo se comportan los modelos, vamos a analizar las atenciones que está haciendo LXMERT con algunos ejemplos y por último se dejará al lector algunos diálogos para que pueda ver y comprobar como se comportaron los modelos de este trabajo.

Para poder presentar los resultados vamos a nombrar cada experimento con siglas.

1. **LX-Oracle**: modelo que adapta LXMERT para la tarea del Oráculo, en este caso solo se replicaron los resultados de un trabajo. Este es nuestro modelo base.

2. **LX-Eval-Pos**: experimento donde se evaluó en el conjunto de testeo a LX-Oracle dando como entrada el texto concatenado de los turnos anteriores positivos.

3. **LX-Train-Pos**: modelo que no solo evalúa cómo LX-Eval-Pos sino que se re entrena la red neuronal.

4. **LX-LSTM**: modelo que usa LXMERT para computar los vectores de modalidad cruzada para todos los turnos de un diálogo y que luego son pasados a una LSTM muchos a muchos.

5. **LX-LSTM-Pos**: LSTM muchos a uno utilizando como historia solamente los turnos que se responden con "Sí".

6. **LX-LSTM-PosObj**: experimento que a LX-LSTM-Pos además de que un turno sea positivo para ser considerado parte de la historia que se modela le agrega la condición de que la pregunta sea de tipo "objeto".

7. **LX-H+E**: modelo que concatena la historia y el vector de modalidad cruzada del turno actual y utiliza un MLP sobre el mismo para realizar la predicción.

5.1. Tablas de resultados

Los resultados del conjunto de testeo se pueden ver en la Figura 5.1 y los resultados sobre el subconjunto de preguntas dependientes de la historia en la Figura 5.2,

5.2. Matrices de Confusión

Una de las métricas más comunes para algoritmos de aprendizaje automático de clasificación son las matrices de confusión. Las filas y columnas de dichas matrices representan cada categoría, cada celda se computa como la sumatoria de datos que son de la categoría que representa la fila en cuestión y que el modelo predijo la categoría que representa la columna. La matriz de confusión de un modelo 100 % preciso es tal que la suma de la diagonal es igual a la cantidad de datos, es decir, todas las celdas que no son parte de la diagonal son 0. Generalmente se suele dividir cada celda por el total de cada clase pero se decidió no hacerlo para que sea visible la cantidad de preguntas que hay en cada clase. A continuación se muestran dichas matrices para algunos de los modelos, todas las demás se podrán ver en el Apéndice de este trabajo.

Tipos de preguntas	Modelos						
	LX-Oracle	LX-Eval-Pos	LX-Train-Pos	LX-LSTM	LX-LSTM-Pos	LX-LSTM-PosObj	LX-H+E
Todas (99784)	**82.93**	82.93 (+0.00)	82.44 (-0.49)	81.73 (-1.20)	81.92 (-1.01)	81.80 (-1.13)	81.82 (-1.11)
Objeto (43303)	90.07	**90.08** (+0.01)	89.73 (-0.34)	88.43 (-1.64)	88.71 (-1.36)	88.66 (-1.21)	88.63 (-1.44)
Super Categoría (2372)	**92.33**	92.33 (+0.00)	92.66 (+0.33)	90.60 (-1.73)	90.77 (-1.56)	90.81 (-1.52)	90.60 (-1.73)
Color (15403)	**76.34**	76.34 (+0.00)	75.43 (-0.91)	74.70 (-1.64)	75.08 (-1.26)	74.82 (-1.52)	74.54 (-1.80)
Forma (301)	69.44	69.44 (+0.00)	70.76 (+1.32)	73.09 (+3.65)	73.42 (+3.98)	**74.09** (+4.65)	72.43 (+2.99)
Tamaño (1364)	74.05	74.05 (+0.00)	**74.85** (+0.80)	73.83 (-0.22)	73.68 (-0.37)	73.49 (-0.46)	73.61 (-0.44)
Acción (7645)	**76.76**	76.76 (+0.00)	76.01 (-0.75)	75.67 (-1.09)	75.49 (-1.27)	75.19 (-1.59)	75.58 (-1.18)
Textura (901)	77.36	77.36 (+0.00)	**79.69** (+2.33)	77.14 (-0.22)	77.80 (+0.44)	78.02 (+0.66)	77.25 (-0.11)
Espacial (39250)	**76.59**	76.59 (+0.00)	75.89 (-0.70)	75.99 (-0.60)	76.04 (-0.55)	75.89 (-0.70)	76.00 (-0.59)
N/A (1519)	75.64	75.64 (+0.00)	**76.10** (+0.46)	74.46 (-1.18)	74.98 (-0.66)	75.25 (-0.39)	75.44 (-0.20)

Figura 5.1: Resultados del conjunto de testeo.

Tipos de Preguntas	Modelos						
	LX-Oracle	LX-Eval-Pos	LX-Train-Pos	LX-LSTM	LX-LSTM-Pos	LX-LSTM-PosObj	LX-H+E
Total (51)	13.72	13.72	13.72	9.80	7.84	11.76	9.80
History (4)	0.00	0.0	0.0	25.00	0.25	0.25	0.25
Middle (11)	18.18	18.18	18.18	0.00	0.00	9.09	0.00
X-Axis (35)	14.28	14.28	14.28	11.42	8.57	11.42	11.42
Y-Axis (1)	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figura 5.2: Resultados del pequeño conjunto dependiente de la historia.

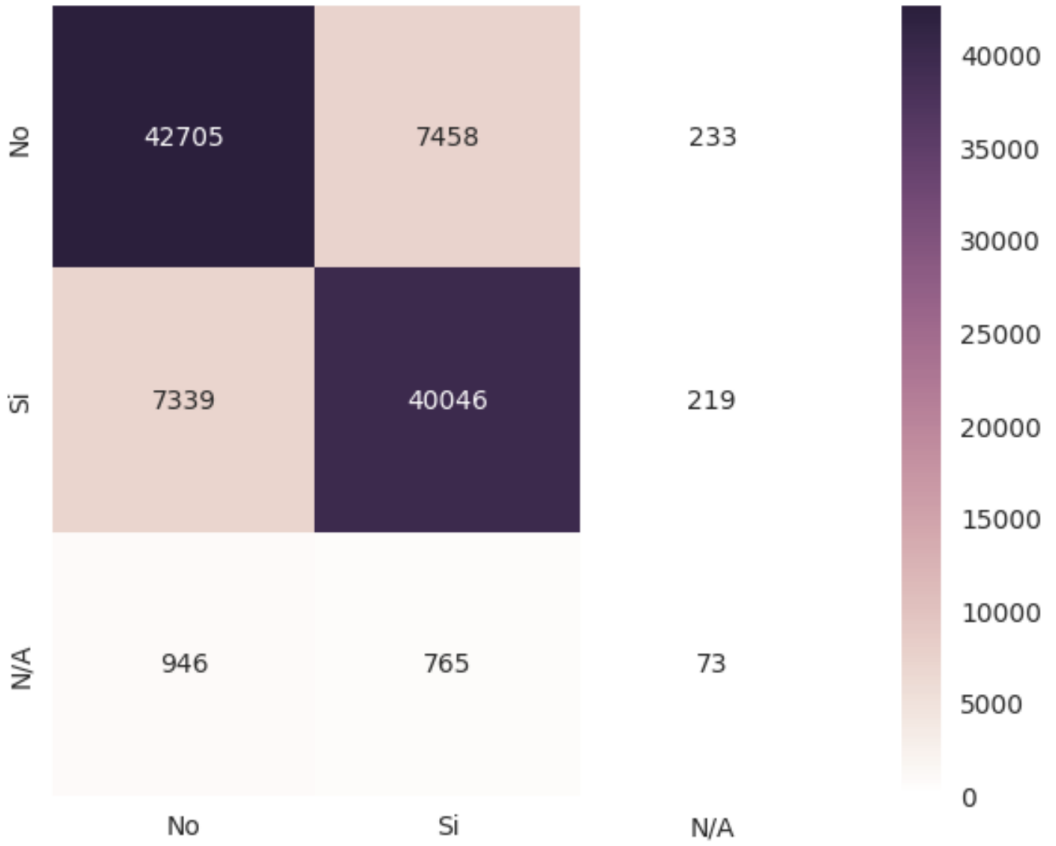


Figura 5.3: Matriz de confusión para el modelo **LX-Oracle**

En la Figura 5.3 se puede ver la matriz del experimento **LX-Oracle**, en la Figura 5.4 del **LX-Train-Pos**, y en la Figura 5.5 la del **LX-LSTM-Pos**.

5.3. Visualizando atenciones

Si bien la interpretabilidad de redes neuronales es un problema actual, LXMERT al usar varios módulos de atención nos permite visualizar esas matrices de atención. Para hacer análisis de error vamos a extraer dichas matrices de las capas de "Cross Attention", de estas teníamos dos tipos bien definidos, lenguaje contra visión, y visión contra lenguaje.

Es importante aclarar que como en todos los experimentos los pesos de las capas de

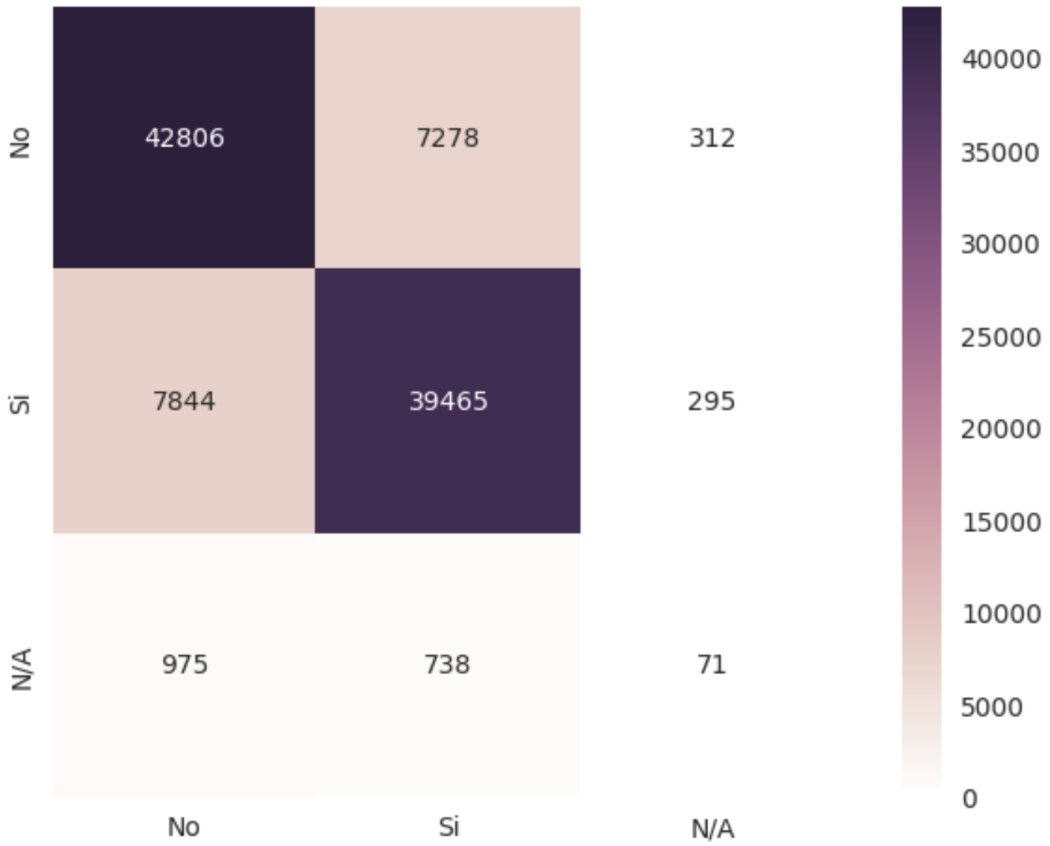


Figura 5.4: Matriz de confusión para el modelo **LX-Train-Pos**

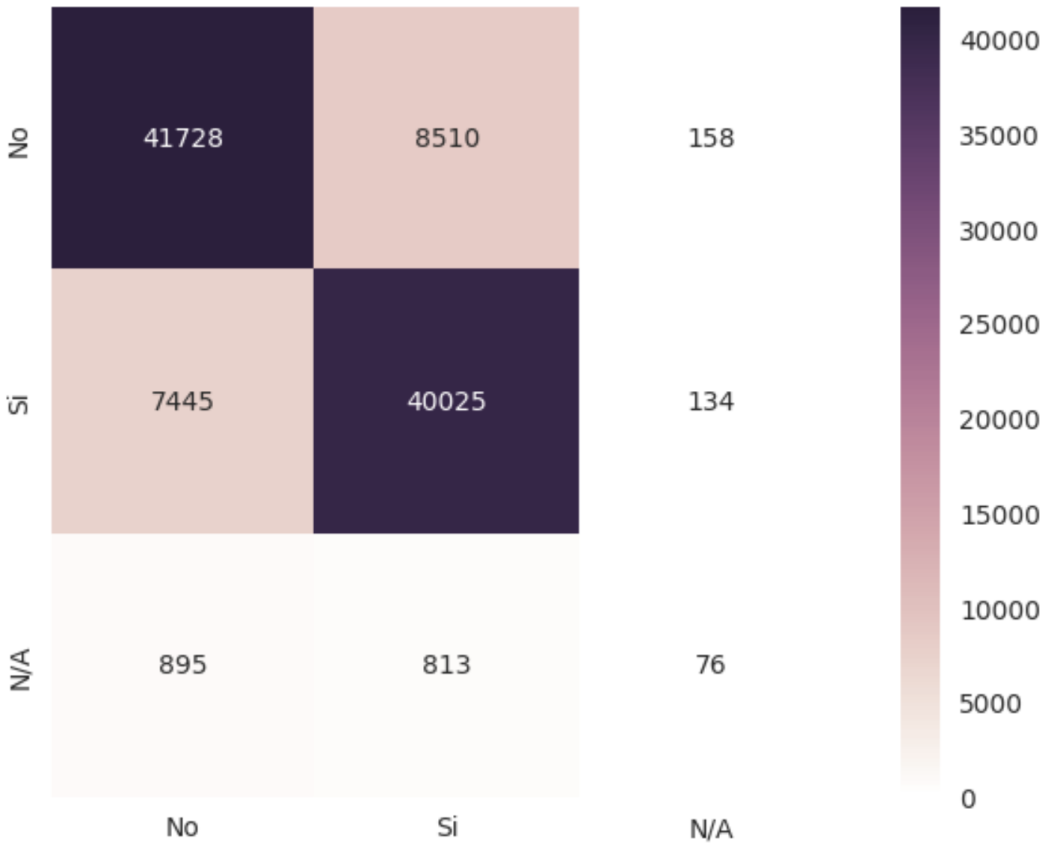


Figura 5.5: Matriz de confusión para el modelo **LX-LSTM-Pos**

Figura 5.6: Ejemplo del conjunto de datos

LXMERT estaban congelados, las atenciones son las mismas para todos los modelos. A continuación veremos un ejemplo de un juego en particular (Figura 5.6), como tenemos 5 encoders de modalidad cruzada apilados vamos a tener 10 mecanismos de atención cruzada de los cuales 5 son visión a lenguaje que son en los que nos vamos a concentrar. Por supuesto que todos ellos son importantes pero éste nos permite mostrar las atenciones más visualmente porque podemos graficar las bounding boxes de los objectos con probabilidades mas altas. En la Figura 5.7 vemos la primer capa, en la Figura 5.8 la segunda, en la Figura 5.9 la tercera, en la Figura 5.10 la cuarta y en la Fiugura 5.11 la quinta.

5.4. Algunos ejemplos

En las Figuras 5.12 y 5.13 se pueden ver ejeemplos de dialogos y como respondieron los distintos modelos.

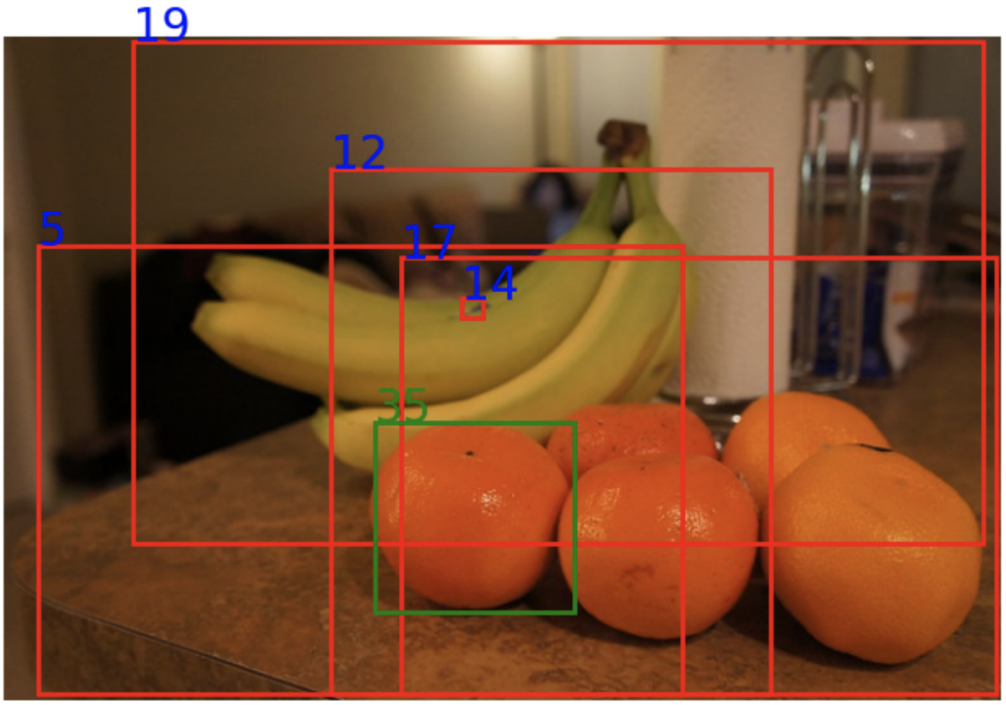


Figura 5.7: 5 regiones con mayor probabilidad de la capa 0 del mecanismo de atención de modalidad cruzada de LXMERT, y el referente en verde

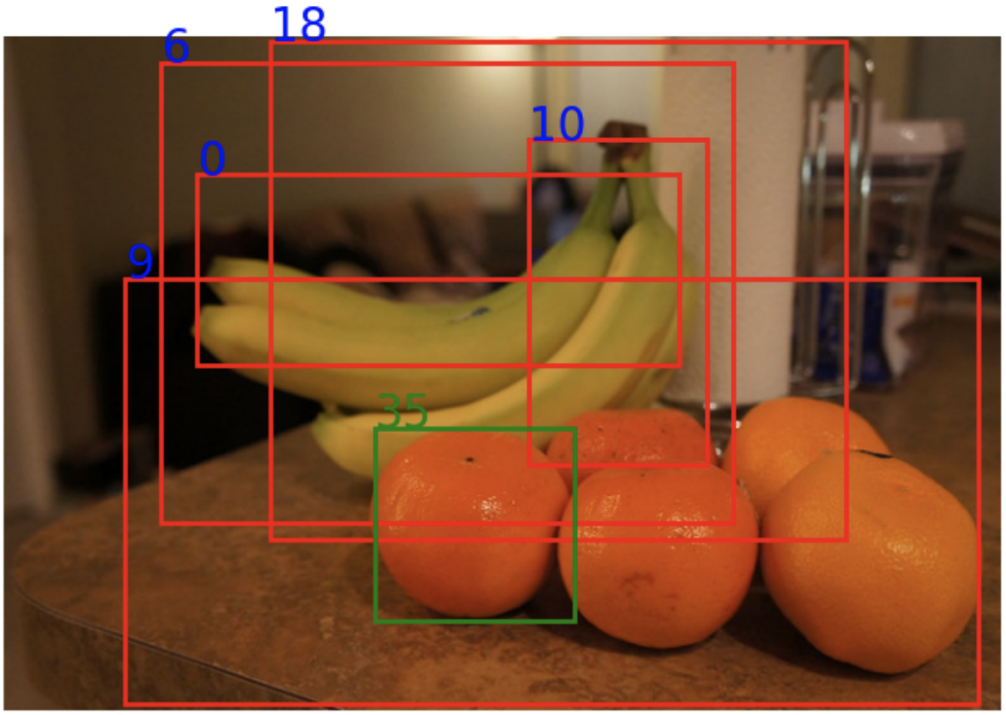


Figura 5.8: 5 regiones con mayor probabilidad de la capa 1 del mecanismo de atención de modalidad cruzada de LXMERT, y el referente en verde

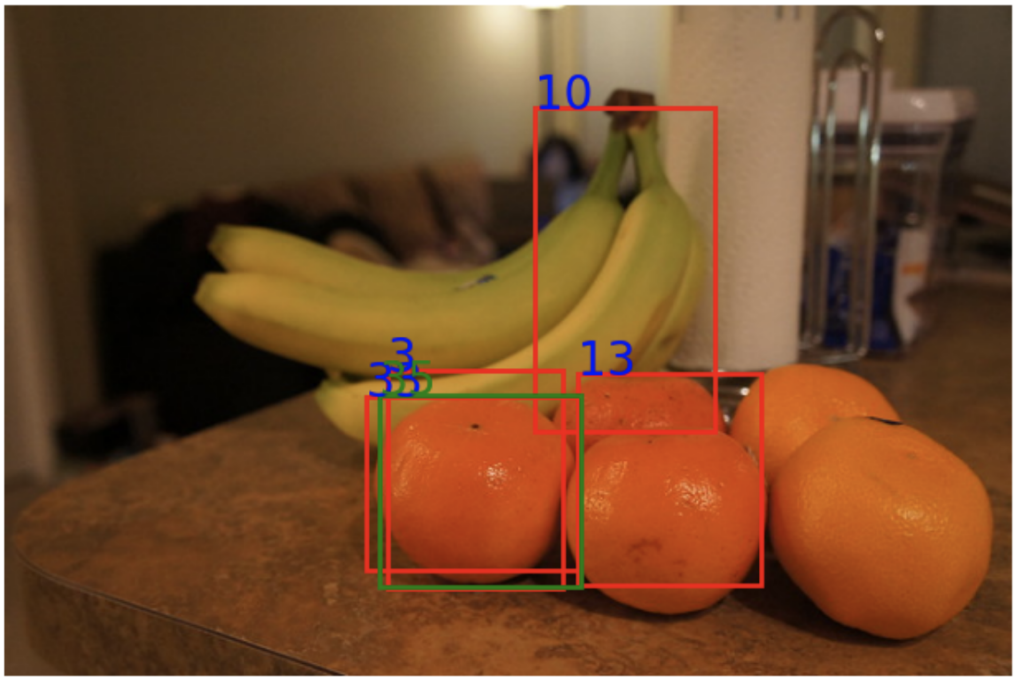


Figura 5.9: 5 regiones con mayor probabilidad de la capa 2 del mecanismo de atención de modalidad cruzada de LXMERT, y el referente en verde

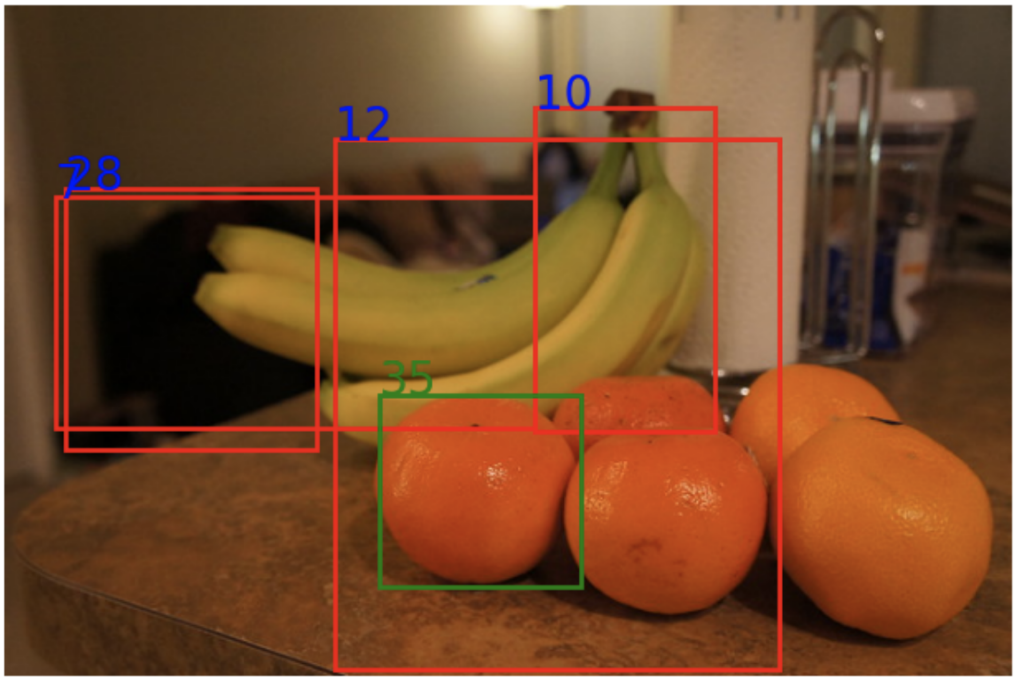


Figura 5.10: 5 regiones con mayor probabilidad de la capa 3 del mecanismo de atención de modalidad cruzada de LXMERT, y el referente en verde

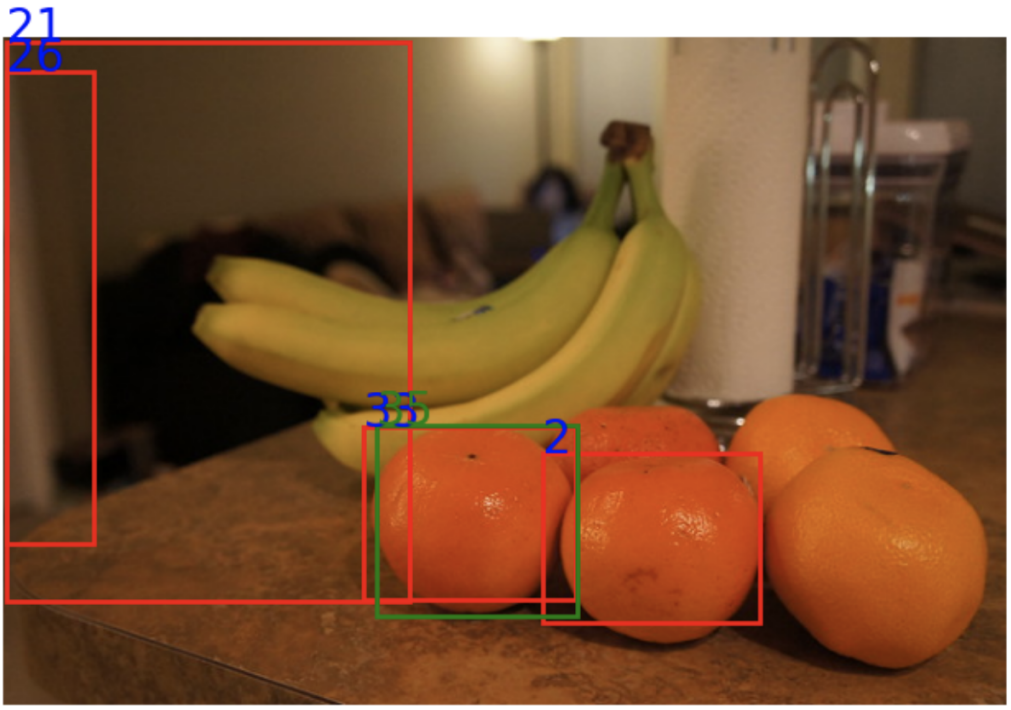


Figura 5.11: 5 regiones con mayor probabilidad de la capa 4 del mecanismo de atención de modalidad cruzada de LXMERT, y el referente en verde

Diálogo --- **Modelos**	¿Es algo donde te podés sentar?	¿Está sobre la mesa de café?	¿Está al medio?	¿Es una pila de papeles?	¿Es la más cercana a nosotros?
Respuesta Humana Real	No	Si	No	Si	Si
LX-Oracle	No	Si	Si	No	Si
LX-Eval-Pos	No	Si	Si	No	Si
LX-Train-Pos	No	Si	Si	No	Si
LX-LSTM	No	Si	Si	Si	No
LX-LSTM-Pos	No	Si	Si	Si	No
LX-LSTM-Pos Obj	No	Si	Si	Si	Si
LX-H+E	No	Si	Si	Si	Si

Figura 5.12: Ejemplo extraido del conjunto de datos con las respectivas respuestas que calculan los distintos modelos.

Diálogo ---------- Modelos	¿Es una fruta?	¿Es la naranja?	¿Está a la derecha?	¿En el Medio?	¿Es la última que está sola?
Respuesta Humana Real	Si	Si	No	No	Si
LX-Oracle	Si	Si	No	Si	No
LX-Eval-Pos	Si	Si	No	Si	No
LX-Train-Pos	Si	Si	No	Si	No
LX-LSTM	Si	Si	No	Si	No
LX-LSTM-Pos	Si	Si	No	Si	Si
LX-LSTM-PosObj	Si	Si	No	Si	No
LX-H+E	Si	Si	No	Si	No

Figura 5.13: Ejemplo extraido del conjunto de datos con las respectivas respuestas que calculan los distintos modelos.

Capítulo 6

Región en discusión para diálogo visual

En los capítulos anteriores de este trabajo de tésis experimentamos formas de representar la historia de forma lingüística uniendo preguntas anteriores a las actuales y también mecanismos más sofisticados de modalidad cruzada haciendo algunas pruebas con late fusion sobre LXMERT. En el capítulo cinco se puede ver que no se logra superar al modelo base propuesto en [30]. La idea original, como se menciona en el capitulo 4, es guiar al modelo a que, a medida que ocurren los turnos del diálogo, preste más atención a ciertas zonas que a otras. Este fenómeno ya lo habíamos analizado en la Figura 4.8 en donde a medida que el diálogo avanzaba debíamos hacer foco en distintas zonas de la imagen y si, por ejemplo, el orden del diálogo cambiaba, el foco también lo hacia. Es por esto que, inspirandonos en esta idea, decidimos probar con más fuerza del lado visual y así es como con Mauricio Mazuecos, Franco Luque, Jorge Sánchez, Hernán Maina y Luciana Benotti escribimos el paper Region under Discussion for visual dialog"[19] publicado en "Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing"(EMNLP2021).

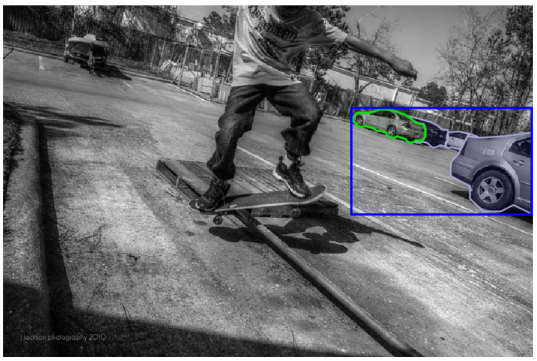
En este capítulo definimos lo que significa que una pregunta visual requiera un historial de diálogo considerando las propiedades visuales intrínsecas y relativas, diseñamos una metodología para anotar un subconjunto de preguntas de Guesswhat?! para las que se requiere su historial de diálogo, proponemos una representación

interpretable de la historia a la que llamamos Región bajo Discusión (RuD), y extendemos el modelo Oracle de [32] y el modelo basado en LXMERT de [30] con nuestro RuD.

6.1. RuD

Definimos una Región bajo discusión (RuD, por sus siglas en inglés) para el diálogo visual como una representación de las restricciones que establece el historial del diálogo. La interpretación de una pregunta depende de su RuD. La Figura 6.1 muestra un de diálogo del conjunto de datos, el referente está resaltado en verde. El modelo base de Oracle propuesto por [32] que, como visto en el capítulo 2, es basicamente un multilayer perceptron sobre un vector que concatena las coordenadas y la imagen del referente, y la salida de una LSTM de la pregunta actual, responde correctamente las cuatro primeras preguntas, fallando únicamente en la pregunta número 5 con una respuesta negativa. Esta pregunta no parece particularmente difícil. Entonces, ¿por qué creó un problema? Porque *la pregunta 5 es la única pregunta para la cual el historial de diálogo modifica la respuesta.* Todas las demás preguntas se pueden responder correctamente simplemente mirando la imagen e ignorando lo que se dijo antes, es decir, las preguntas 1 a 4 son turnos de VQA (así se les suele decir a las preguntas que no dependen del historial del diálogo). Si respondemos la pregunta 5, ¿está a la izquierda? ignorando el historial de diálogo, la respuesta correcta es no, porque el referente está claramente a la derecha de la imagen, no a la izquierda. La RuD para esta pregunta, representada en azul en la figura, modifica la respuesta.

En este trabajo, modelamos en el RuD las restricciones que están relacionadas con las propiedades intrínsecas del objeto que han sido previamente acordadas entre los participantes del diálogo. Una propiedad intrínseca es aquella que es inherente e inseparable del objeto y no depende del contexto visual en el que se coloca el objeto. En este ejemplo, dicha propiedad intrínseca es el hecho de que el objeto es un auto. Otra propiedad intrínseca puede ser que el objeto sea un vehículo, pero no el hecho de que el objeto esté junto a otro auto. Decimos que tal propiedad no es intrínseca del objeto sino relativa a la posición del auto. Decidimos representar en RuD solo la historia intrínseca, motivados por la literatura del diálogo de robots, donde las



Pregunta	**Respuesta Humana**
1. It is a person?	no
2. It is a car?	yes
3. Is it in the back?	yes
4. Are there two together?	yes
5. *Is it on the left?*	yes

Figura 6.1: Este ejemplo ilustra nuestra definición de pregunta dependiente de la historia. La pregunta 5 podría ser respondida con "no" si fuera preguntada al principio del diálogo, cuando la historia es vacía, ya que el referente (marcado con verde) no está a la izquierda de la imagen. Sin embargo, cuando el RuD (en azul) es enmarcado por las preguntas anteriores sabemos que la respuesta es "yes" (Sí) porque el referente está a la izquierda dentro del RuD.

propiedades intrínsecas son abundantes y estables [29]. Creemos que restringir el RuD a las propiedades intrínsecas nos permite centrarnos en los fenómenos que nos interesan manteniendo el modelo simple y fácilmente interpretable.

En resumen, la mayoría de las preguntas de este diálogo se pueden responder correctamente independientemente del contexto de diálogo: no necesitan el historial. En efecto, a excepción del último turno, el diálogo de la Figura 6.1 es solo VQA. En las siguientes secciones de este capítulo mostramos cómo modelamos la historia del diálogo como restricciones que representan la parte de la imagen que los interlocutores acuerdan que es el RuD y sobre la cual se interpretarán el resto de las preguntas. Para nuestro ejemplo, con respecto al recuadro azul, la respuesta correcta de ¿Está a la izquierda? es sí ya que el coche está a la izquierda de la RuD acordada.

6.2. Anotación al conjunto de datos

Para detectar preguntas dependientes de la historia, primero se hizo un muestreo de un conjunto de preguntas relativas que siguen a una pregunta de objeto respondida positivamente en un diálogo. Luego, dos anotadores identifican preguntas de tal manera que la polaridad de la respuesta cambia cuando se hace la pregunta considerando su historial. El procedimiento de anotación es el siguiente: Primero pedimos que miren la imagen y la pregunta candidata sin mirar el historial de diálogo, luego que respondan a la pregunta con "sí", "tal vez sí", "tal vez no", "no" o "no sé", luego comparamos con la respuesta en el corpus que la persona dio a esa pregunta considerando el historial de diálogo. Si las respuestas no coinciden, marcamos la pregunta como dependiente de la historia. En este entorno, los desacuerdos entre los anotadores surgen principalmente de diferentes puntos de vista sobre propiedades vagas de los objetos. Sorprendentemente, y en contraste con lo que suele suponerse en trabajos anteriores [1], las preguntas visuales dependientes de la historia del diálogo no contienen más pronombres y elipsis que las preguntas visuales independientes de la historia. De las 1658 preguntas analizadas, dos anotadores acordaron que 204 preguntas dependen de la historia. Llamamos a estas 204 preguntas nuestro conjunto de pruebas GWHist. Mediante este procedimiento, marcamos el 12,3 % de las preguntas de la muestra como dependientes de la historia. En la Figura 6.2 se puede ver la herramienta desarrollada para que usen los anotadores para construir el conjunto de datos.

6.3. Armando las RuDs

En resumen, para construir los RuD, analizamos y combinamos las preguntas en cada historial de diálogo para construir un historial semántico, es decir, una representación de las propiedades intrínsecas conocidas del objeto de destino. Luego, usamos esta información para filtrar los objetos en la imagen y obtener un conjunto de objetos candidatos que formarán parte del RuD.

Analizamos preguntas que establecen relaciones de tipo "is it" y "is the" entre un sintagma nominal (NP por "noun phrase.[en] inglés) y el referente. Un sintagma es una palabra o grupo de palabras que desempeñan una función sintáctica dentro de una oración. Estas palabras se articulan alrededor de un término denominado núcleo

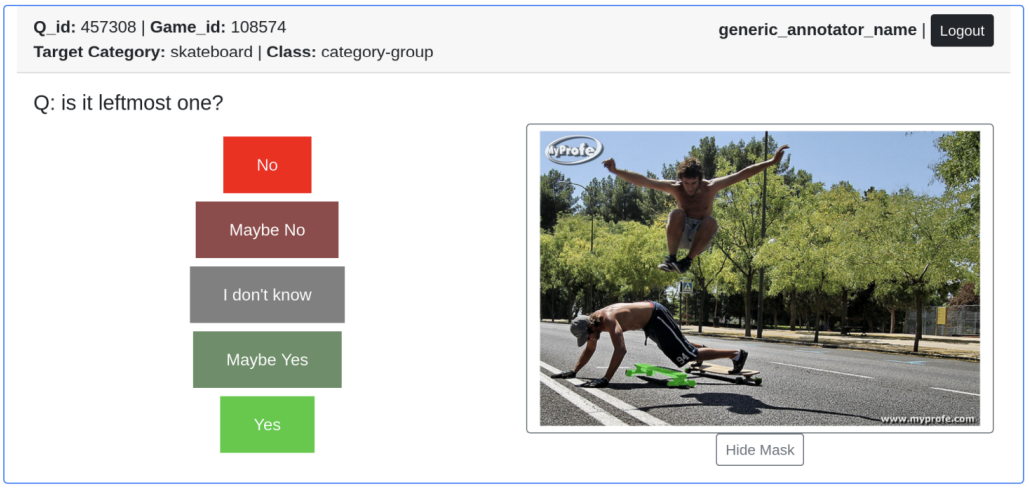


Figura 6.2: Herramienta desarrollada para hacer las anotaciones para construir GW-Hist

Patron	**Ejemplo**
NP?	*1. person ?*
is it a NP ?	*2. is it a red car ?*
is the NOUN a NP ?	*3. is the object a plate ?*
is it one of the NP ?	*4. is it one of the boats ?*
NP = NOUN | NOUN NOUN | ADJ NOUN	

Cuadro 6.1: Patrones sintácticos comunes.

del sintagma, el cual determina el tipo de sintagma. Por ejemplo, si el núcleo de un sintagma es un sustantivo, dicho sintagma será un sintagma nominal (NP) mientras que si el núcleo es un verbo, se tratará de un sintagma verbal. Las respuestas a estas preguntas suelen transmitir información sobre la categoría del objeto, como en Ïs it a person?". Una respuesta positiva a una pregunta de categoría implica que los objetos candidatos incluyen solo objetos de esa categoría, mientras que una respuesta negativa implica que estos objetos no son candidatos. Definimos expresiones regulares para los patrones sintácticos más comunes. Tokenizamos y etiquetamos las preguntas usando NLTK y Stanza. La Tabla 6.1 muestra algunos de los principales patrones que utilizamos.

Después del análisis, los NP obtenidos se lematizan mediante NLTK y se comparan con las 80 categorías del conjunto de datos COCO. En el caso de algunas preguntas de categoría que presenten dos tokens NP, generalmente solo el segundo token se refiere a la categoría, mientras que el primero se refiere a otra propiedad intrínseca (mayormente un adjetivo). En este caso, emparejamos solo el segundo token con una categoría, si la respuesta es positiva. Algunos SN se refieren a categorías que no están presentes en COCO sino a supercategorías, es decir, sustantivos que cubren varias categorías de COCO (por ejemplo, çomida", que abarca "manzana", "banana", "brócoli", etc.). Hacemos coincidir estos sustantivos usando una lista precalculada de supercategorías conocidas. Las supercategorías y su mapeo a categorías se obtienen de WordNet (Fellbaum, 1998) extrayendo relaciones de hiperónimos.

Los procesos de análisis y emparejamiento dan como resultado un historial semántico que está disponible para cada pregunta en un juego. La historia semántica es la lista ordenada de relaciones positivas y negativas con categorías o supercategorías encontradas en los turnos anteriores (por ejemplo, [(pos, "vehicle"), (neg, çar")] significa que el referente es un vehículo pero no es un auto). Los objetos de la imagen se filtran utilizando el historial para obtener un conjunto de objetos candidatos. A continuación, describimos nuestros enfoques para los elementos positivos y negativos de la historia por separado. Para la historia positiva usamos solo el último elemento, asumiendo que es el más específico. Seleccionamos los objetos que son consistentes con la categoría o supercategoría de este elemento. Para el historial negativo, nuestra política es eliminar de los candidatos todos los objetos en las categorías o super categorías negadas. Después de procesar el historial semántico, verificamos que los objetos candidatos estén bien formados. Decimos que el conjunto está mal formado si no incluye el objeto de destino. En este caso, forzamos la inclusión del objeto de destino como una política ad-hoc.

6.4. Extendiendo los modelos del Oráculo con RuD

Ampliamos dos modelos populares para el Oráculo en el diálogo visual. El baseline Pregunta+Categoría+Espacial (llamado QCS) propuesto por [32] y el más reciente Oracle (llamado CMO) intermodal basado en LXMERT propuesto por [30] que es el modelo tomado como base en este trabajo de tésis. Nos basamos en estos modelos y

proponemos dos extensiones simples para codificar el RuD. Nombramos a nuestros modelos como QCS+RuD y CMO+RuD, respectivamente. Para ambos modelos, definimos el RuD como el cuadro delimitador más pequeño que encierra todos los objetos en el conjunto de candidatos. Los objetos candidatos se calculan a partir del historial de diálogo como se describió en la sección anterior.

Para el caso de QCS extendemos las entradas del modelo para que ademas de tomar las coordenadas del referente con respecto al total de la imagen, también tome un vector con las coordenadas escaladas con respecto al RuD, se puede ver de forma ilustratda en la figura 6.3. Concretamente, sean (x_1, y_1, x_2, y_2) las coordenadas superior izquierda y derecha inferior del referente y (X_1, Y_1, X_2, Y_2) las del RuD. Definamos $x_0 = \frac{(x_1+x_2)}{2}$, $y_0 = \frac{(y_1+y_2)}{2}$ y dejemos que (w, h) y (W, H) denoten el ancho y la altura del cuadro objetivo y RuD, respectivamente. Agregamos las siguientes características a la incrustación de entrada QCS: $2\frac{x_1-X_1}{W} - 1$, $2\frac{y_1-Y_1}{H} - 1$, $2\frac{x_2-X_1}{W} - 1$, $2\frac{y_2-Y_1}{H} - 1$, $\frac{x_0}{W}$, $\frac{y_0}{H}$, $\frac{w}{W}$, $\frac{h}{H}$. Para CMO, el modelo espera como entradas no solo los vectores de palabras y regiones, sino también su ubicación con respecto a la pregunta y la imagen de referencia, respectivamente. Para la modalidad visual, esta información se codifica en forma de coordenadas de cuadro delimitador después del módulo de detección de objetos. En nuestro caso, esto corresponde a las coordenadas de las esquinas superior izquierda e inferior derecha de cada cuadro delimitador de objeto. Es decir, codificamos las coordenadas espaciales de cada caja como $\frac{x_1-X_1}{W}$, $\frac{y_1-Y_1}{H}$, $\frac{x_2-X_2}{W}$, $\frac{y_2-Y_2}{H}$, en donde x_1, y_1, x_2, y_2 son las coordeenadas del referente con respecto al total de la imagen y X_1, Y_1, X_2, Y_2 son las coordenadas de nuestro RuD. En la Figura 6.4 mostramos cómo implementamos RuD para CMO. Tenga en cuenta que, en este caso, las coordenadas que se encuentran fuera del RuD serán negativas o con un valor mayor que uno. Esto no sucede para el modelo QCS+RuD porque solo se modifican las coordenadas del referente y estas siempre caen dentro del RuD. De todas formas nuestra intuición es que el modelo utilice esas coordenadas fuera de rango para ignorarla y enfocarse aún más en las que sí entran dentro del RuD.

6.5. Resultados

Como se muestra en la Tabla 6.2 la precisión en el conjunto de datos de testeo es ligeramente superior para los modelos que agregan RuD, pero la diferencia es pe-

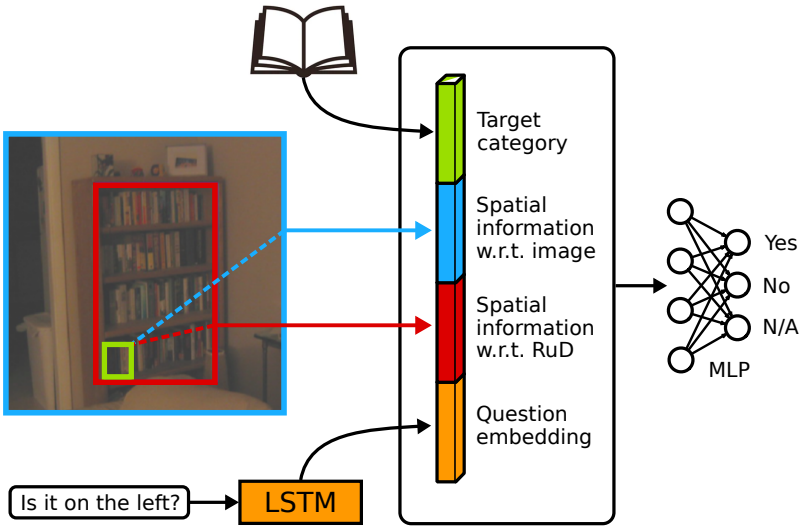


Figura 6.3: QCS Model

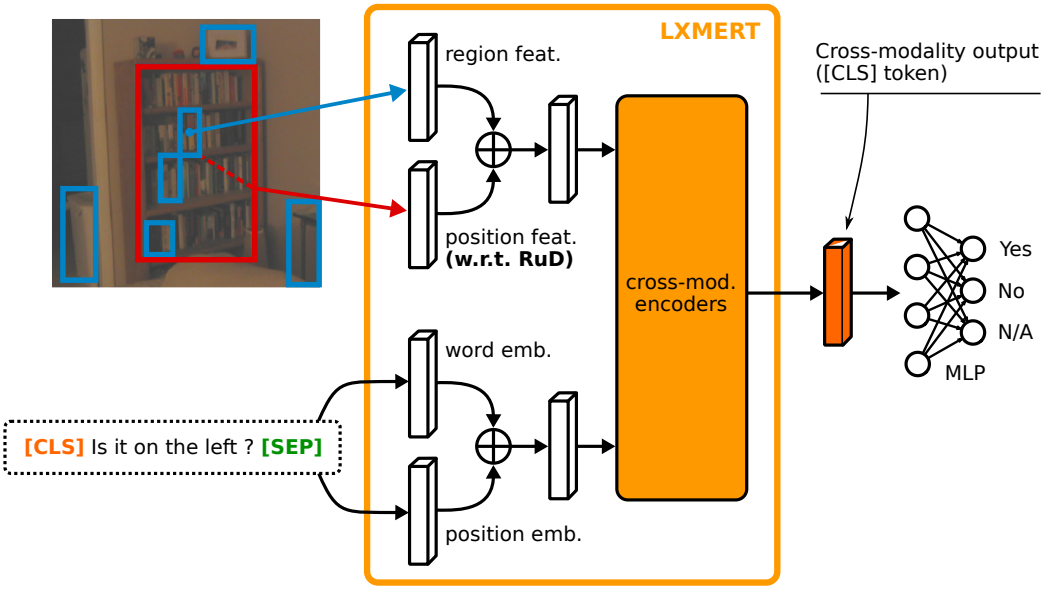


Figura 6.4: CMO Model

Conjunto	**QCS**	**QCS+RuD**	**CMO**	**CMO+RuD**
GW	0.733	0.744	0.809	0.813
GWHist	0.285	0.402	0.285	0.416

Cuadro 6.2: Precisión de los distintos modelos

queña. Esto se debe al hecho de que la mayoría de las preguntas en GW, incluidas las preguntas espaciales, no dependen de la historia, como argumentamos anteriormente. Sin embargo, el efecto de agregar el RuD mejora en el GWHist dependiente de la historia, donde QCS + RuD y CMO +RuD muestran un incremento de 41 % y 46 %, respectivamente. El hecho de que ambos modelos mejoren constantemente muestra que el RuD está capturando la región de la imagen en la que se interpreta la pregunta dependiente del historial. Con una precisión máxima de 0.416 para preguntas dependientes de la historia.

Capítulo 7

Conclusión y trabajo futuro

En este trabajo probamos muchas estrategias simples de modelar la historia usando el lenguaje y usando como modelo base el modelo de LXMERT modificado para el Oráculo en el juego de GuessWhat!?. Demostramos que no es tarea fácil hacer que los sistemas tengan en cuenta el contexto de los diálogos para responder mejor preguntas referidas a un objeto en una imagen. Aparentemente los modelos que intentan hacer uso de distintas heurísticas sobre el lenguaje no son suficientes para resolver el problema. Por eso mismo tuvimos que crear modelos que modifiquen la visión que tienen los mismos sobre la imagen, por eso mismo creamos la region en discusión (RuD) y un dataset que es subconjunto del dataset de GuessWhat!? en donde solo aparecen las preguntas que son dependientes de la historia del diálogo, también descubrimos que el porcentaje de preguntas que necesitan el dialogo no es muy grande. Además disponibilizamos una herramienta para poder seguir anotando ese conjunto y otros conjuntos que sean diálogos sobre imágenes.

Lo que creemos que podría ser muy interesante para el futuro es volver a realizar todos los experimentos del Capitulo 4 pero con el modelo LXMERT sin congelar sus pesos, ya que en los experimentos de esta tésis los vectores que representaban al par de la imagen y la pregunta como salida de LXMERT estaban pre-computados, es decir, en tiempo de entrenamiento los pesos de LXMERT no se actualizaban. Creemos que si pudiésemos hacer que esos pesos se actualicen en entrenamiento, no solo las redes LSTM tendrían la responsabilidad de saber de la historia si no que tam-

bién el modelo de LXMERT aprendería a generar los vectores también conociendo la historia, y esto podría mejorar la *performance* de los modelos.

Por ultimo un trabajo futuro que consideramos que es necesario no solo para este trabajo es crear un modelo que pueda recrear los resultados pero que sea mas pequeño, ligero y de entrenamiento más rápido. En nuestro caso podríamos construir un *tiny-LXMERT* para crear un *tiny-Oracle* para poder desplegar estos modelos en la realidad y que puedan ser usados para inferencia de forma rápida.

Referencias

[1] Shubham Agarwal y col. "History for Visual Dialog: Do we really need it?" En: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, jul. de 2020, págs. 8182-8197. DOI: 10.18653/v1/2020.acl-main.728. URL: https://aclanthology.org/2020.acl-main.728.

[2] Stanislaw Antol y col. "VQA: Visual Question Answering". En: *International Conference on Computer Vision (ICCV)*. 2015.

[3] Stanislaw Antol y col. "VQA: Visual Question Answering". En: *CoRR* abs/1505.00468 (2015). arXiv: 1505.00468. URL: http://arxiv.org/abs/1505.00468.

[4] Dzmitry Bahdanau, Kyunghyun Cho y Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". En: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. por Yoshua Bengio y Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.0473.

[5] Kyunghyun Cho y col. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". En: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[6] Abhishek Das y col. "Visual Dialog". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[7] Jacob Devlin y col. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". En: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[8] Aurélien Géron. Sebastopol, CA.

[9] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[10] Claudio Greco, Alberto Testoni y Raffaella Bernardi. "Which Turn do Neural Models Exploit the Most to Solve GuessWhat? Diving into the Dialogue History Encoding in Transformers and LSTMs". En: *NL4AI@AI*IA*. 2020.

[11] Kaiming He y col. "Deep Residual Learning for Image Recognition". En: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[12] Sepp Hochreiter y Jürgen Schmidhuber. "Long Short-term Memory". En: *Neural computation* 9 (dic. de 1997), págs. 1735-80. DOI: 10.1162/neco.1997.9.8.1735.

[13] Sahar Kazemzadeh y col. "ReferItGame: Referring to Objects in Photographs of Natural Scenes". En: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, oct. de 2014, págs. 787-798. DOI: 10.3115/v1/D14-1086. URL: https://aclanthology.org/D14-1086.

[14] Sang-Woo Lee, Yu-Jung Heo y Byoung-Tak Zhang. "Answerer in Questioner's Mind for Goal-Oriented Visual Dialogue". En: *CoRR* abs/1802.03881 (2018). arXiv: 1802.03881. URL: http://arxiv.org/abs/1802.03881.

[15] Oliver Lemon y Olivier Pietquin. *Data-Driven Methods for Adaptive Spoken Dialogue Systems: Computational Learning for Conversational Interfaces*. Oct. de 2012. ISBN: 978-1-4614-4803-7. DOI: 10.1007/978-1-4614-4803-7.

[16] Tsung-Yi Lin y col. "Microsoft COCO: Common Objects in Context". En: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[17] Chia-Wei Liu y col. "How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation". En: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, nov. de 2016, págs. 2122-2132. DOI: 10.18653/v1/D16-1230. URL: https://aclanthology.org/D16-1230.

[18] Minh-Thang Luong, Hieu Pham y Christopher D. Manning. "Effective Approaches to Attention-based Neural Machine Translation". En: *CoRR* abs/1508.04025 (2015). arXiv: 1508.04025. URL: http://arxiv.org/abs/1508.04025.

[19] Mauricio Mazuecos y col. "Region under Discussion for visual dialog". En: *Proceedings of the 2021 Conference on Empirical Methods in Natural Lan-*

guage Processing. Online y Punta Cana, Dominican Republic: Association for Computational Linguistics, nov. de 2021, págs. 4745-4759. DOI: 10.18653/v1/2021.emnlp-main.390. URL: https://aclanthology.org/2021.emnlp-main.390.

[20] Shaoqing Ren y col. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". En: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[21] Olga Russakovsky y col. "ImageNet Large Scale Visual Recognition Challenge". En: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[22] Iulian Vlad Serban y col. "A Survey of Available Corpora for Building Data-Driven Dialogue Systems". En: *CoRR* abs/1512.05742 (2015). arXiv: 1512.05742. URL: http://arxiv.org/abs/1512.05742.

[23] Ravi Shekhar y col. "Beyond task success: A closer look at jointly learning to see, ask, and GuessWhat". En: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, jun. de 2019, págs. 2578-2587. DOI: 10.18653/v1/N19-1265. URL: https://aclanthology.org/N19-1265.

[24] Karen Simonyan y Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". En: *arXiv 1409.1556* (sep. de 2014).

[25] Satinder Singh y col. "Reinforcement Learning for Spoken Dialogue Systems". En: (oct. de 1999).

[26] Florian Strub y col. "End-to-end optimization of goal-driven and visually grounded dialogue systems". En: *CoRR* abs/1703.05423 (2017). arXiv: 1703.05423. URL: http://arxiv.org/abs/1703.05423.

[27] Hao Tan y Mohit Bansal. "LXMERT: Learning Cross-Modality Encoder Representations from Transformers". En: *CoRR* abs/1908.07490 (2019). arXiv: 1908.07490. URL: http://arxiv.org/abs/1908.07490.

[28] Hao Tan y Mohit Bansal. "LXMERT: Learning Cross-Modality Encoder Representations from Transformers". En: *CoRR* abs/1908.07490 (2019). arXiv: 1908.07490. URL: http://arxiv.org/abs/1908.07490.

[29] Xiang Zhi Tan y col. "Now, Over Here: Leveraging Extended Attentional Capabilities in Human-Robot Interaction." En: *ACM/IEEE International Conference on Human-Robot Interaction (HRI '20 Late-breaking Reports)*. ACM, mar. de 2020, págs. 468-470. URL: https://www.microsoft.com/en-us/research/publication/now-over-here-leveraging-extended-attentional-capabilities-in-human-robot-interaction/.

[30] Alberto Testoni y col. "They Are Not All Alike: Answering Different Spatial Questions Requires Different Grounding Strategies". En: *Proceedings of the Third International Workshop on Spatial Language Understanding*. Online: Association for Computational Linguistics, nov. de 2020, págs. 29-38. DOI: 10.18653/v1/2020.splu-1.4. URL: https://aclanthology.org/2020.splu-1.4.

[31] Ashish Vaswani y col. "Attention is All you Need". En: *Advances in Neural Information Processing Systems*. Ed. por I. Guyon y col. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[32] Harm de Vries y col. "GuessWhat?! Visual object discovery through multi-modal dialogue". En: *CoRR* abs/1611.08481 (2016). arXiv: 1611.08481. URL: http://arxiv.org/abs/1611.08481.

[33] Tsung-Hsien Wen y col. "A Network-based End-to-End Trainable Task-oriented Dialogue System". En: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*. Valencia, Spain: Association for Computational Linguistics, abr. de 2017, págs. 438-449. URL: https://aclanthology.org/E17-1042.

[34] Bolei Zhou y col. "Learning Deep Features for Scene Recognition using Places Database". En: *Advances in Neural Information Processing Systems*. Ed. por Z. Ghahramani y col. Vol. 27. Curran Associates, Inc., 2014. URL: https://proceedings.neurips.cc/paper/2014/file/3fe94a002317b5f9259f82690aeea4cd-Paper.pdf.