Anexos

Anexos I

Descripción del modelo de red neuronal clave-valor

La siguiente descripción ha sido obtenida traduciendo y adaptando el articulo original [15]. Se trata de un modelo de red neuronal entrenado *end-to-end*, está compuesto principalmente por dos memorias distintas, denominadas clave y valor, estas pretenden facilitar la búsqueda de respuestas en función de una pregunta.

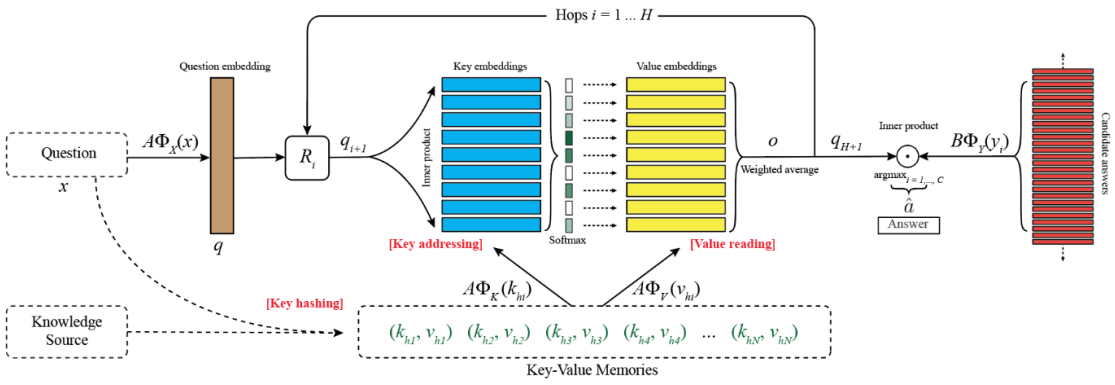

Figura I.1: Modelo Clave-Valor

El funcionamiento de este modelo se centra en dos fases; una primera fase de direccionamiento y lectura de la memoria, y una segunda de acceso a la misma. La fase de direccionamiento se basa en la memoria clave, en la cuál se mantiene guardada la información relevante respecto a la pregunta realizada, mientras que la fase de lectura, en la que se devuelve el resultado a la pregunta, utiliza la memoria valor. A continuación se introduce un resumen detallado y una Figura I.1 del funcionamiento del modelo.

Sea x una pregunta cualquiera y $(k_1, v_1), \ldots, (k_N, v_N)$ la información disponible representada como vectores. El direccionamiento y lectura de la memoria consta de tres pasos:

1. El primer paso es la preselección de un subconjunto de M $(k_{h_1}, v_{h_1}), \ldots, (k_{h_M}, v_{h_M})$ memorias con $M < N$ en las cuáles las claves tienen en común al menos una palabra con la pregunta con una frecuencia menor de 1000, de esta manera se eliminan palabras vacías[1] (*stop words*)

2. Los datos de entrada y la pregunta se codifican como k_{h_i}, v_{h_i} y x respectivamente, las claves y los valores se guardan en la memoria correspondiente, comparando las memorias clave con la pregunta se asignan una serie de probabilidades a las memoria valor, cuanto más alta, más relevante es la memoria valor para la pregunta:

$$p_{h_i} = \text{Softmax}(A\Phi_X(x) \cdot A\Phi_K(k_{h_i}))$$

   con $\Phi_{\_}$ funciones características de dimensión D, A una matriz $d \times D$.

3. Por último, se devuelve el vector de la suma ponderada de los valores v_{h_i} usando las probabilidades asociadas a cada uno de los valores, quedando de la siguiente manera:

$$o = \sum_i p_{h_i} A\Phi_V(v_{h_i}).$$

Con este resultado se actualiza la consulta usando tanto la salida obtenida o como la consulta previa $q = A\Phi_X(x)$, quedando la nueva consulta como $q_2 = R_1(q + o)$ donde R_1 es una matriz $d \times d$. Los pasos 2 y 3 de la lista anterior se realizan $H - 1$ veces más para llegar al total de saltos definidos para el modelo

$$p_{h_i} = \text{Softmax}(q_{j+1}^T A\Phi_K(k_{h_i})), \qquad q_{j+1} = R_j(q_j + o).$$

Tras las H saltos se calcula la predicción final $\hat{a}$ sobre todas las posibles salidas y_i:

$$\hat{a} = \underset{i=1,\ldots,C}{\text{argmax}}\ \text{Softmax}(q_{H+1}^T B\Phi_Y(y_i)),$$

donde y_i son los posibles candidatos a respuesta (todas las entidades almacenadas).

El modelo aprende a mejorar los accesos a la memoria para minimizar la pérdida de entropia cruzada entre la respuesta correcta a y $\hat{a}$ y así devolver la respuesta objetivo a. Para aprender las matrices $A, B, R_1, \ldots, R_H$ se utilizan los métodos de retropropagación y gradiente descendente estocástico.

[1] artículos, pronombres, preposiciones, conectores, palabras que no aporten significado semántico

Anexos II

Extracción de entidades

Se ha utilizado un *NER* (*Named Entity Recognition*) desarrollado por el departamento de Inteligencia Artificial y Sistema Cognitivos del Instituto Tecnológico, la extracción de entidades es una subtarea de la extracción de información que busca localizar y clasificar entidades dentro del texto en categorías predefinidas como nombres propios, porcentajes, cantidades, organizaciones, localizaciones, fechas. Este NER ha sido utilizado para proyectos de diversa índole debido a su versatilidad.A continuación se listan los proyectos que utilizan esta implementación:

1. BOA → datos provenientes de un crawling, texto plano.
2. Aragón OpenData → poblar ontología, texto extraído de páginas Web.
3. DGA Convenios → similar al BOA, proceso de crawling, texto plano.
4. Social Media → datos de Twitter.

En este anexo se expone de que manera se ha utilizado, así como los cambios y adaptación realizada sobre el NER original para cumplir con los objetivos del proyecto.

En este caso y con el conjunto de datos del corpus, el NER ha sido utilizado para extraer tanto nombres propios como organizaciones, fechas y roles, dentro de los datos se tienen situaciones anómalas que producen una mala clasificación, a continuación se desglosan los problemas encontrados en nombres propios y roles

II.1. Nombres propios

La extracción de nombres propios es la parte más importante ya que todos los datos tienen referencia a una persona, ya sea para especificar organización, rol o fechas de inicio y fin del período de trabajo, además las preguntas a contestar sobre el conjunto de datos son sobre personas también.

Dentro de los nombres propios encontramos varios casos particulares no detectados por el NER, a continuación se van a listar y detallar los diferentes procesos que se realizan para conseguir que esta detección sea posible, en ocasiones no se ha conseguido dar con una solución para dicha detección debido a la respuesta obtenida por el NER. Más adelante se comenta qué se hace con estos casos especiales sin solución.

Casos particulares junto a sus soluciones:

En los datos se encuentran apellidos conjugados unidos con una "y", el NER clasifica esto como dos personas independientes, como en los datos se sabe que en cada frase sólo se tiene un nombre de persona, esto se soluciona concatenando los resultados devueltos por el NER con la correspondiente "y". Un ejemplo sería: Jaime Gaspar y Auría, lo cuál se detecta como Jaime Gaspar por un lado y Auria por otro, concatenando ambas personas con una "y" se obtiene el nombre completo.

Existen nombres que contienen localizaciones reales en su nombre: León, Guadalupe y Montserrat, la utilización del NER en este caso no contempla obtener localizaciones ya que no tienen interés para los datos a tratar por lo que para resolver esta confusión entre lugar y persona, se hace una verificación previa sobre el resultado de la respuesta y si se encuentra que ha detectado una localización, se puede determinar con exactitud que se ha confundido con el nombre de una persona y utilizar dicha localización como el nombre propio buscado.

En los datos, no todos los puestos tienen a una persona en cargo de dicho puesto, esto significa que hay puestos vacantes que quedan por cubrir, la única manera de comprobar si se trata de este caso es buscar a mano en la frase a procesar si se encuentra alguna de las siguientes subcadenas: `Vacante Temporal, Sin Particular, Vacante Provisional`.

Otro caso ocurre cuando se tiene un apellido compuesto con más de dos palabras previas, es decir, apellidos como: de la Rosa, de la Fuente, de la Torre... En estos casos, no se puede dar por supuesto que después de un apellido si encontramos dos palabras la siguiente sea un apellido aún comenzando esta con letra mayúscula, por ejemplo: Álvaro Monteagudo de la Isla de Pascua, en este caso Isla de Pascua es un lugar geográfico. Para resolver estos casos se

ha repetido el mismo proceso que el caso anterior, buscar en el texto alguno de estos nombres y si se encuentra no seguir buscando.

Existe un último caso de un nombre no detectado por el NER, es el caso de `Vitelio Manuel Tena Piazuelo`, para este caso se utiliza la misma solución que en los dos casos anteriores, buscar el nombre en el texto.

Además de estos casos, encontramos otros en los que el NER modifica el nombre, es el caso de los nombres cuyos apellidos tienen un guión, `Luis Fernández-Caballero Lamana` por ejemplo, para estas situaciones el NER devuelve el nombre igual pero eliminando el guión: `Luis FernándezCaballero Lamana`. Se ha utilizado una expresión regular que detecta una letra mayúscula justo a continuación de una minúscula para detectar estos casos y recuperar el nombre original incluyendo un guión entre las letras detectadas.

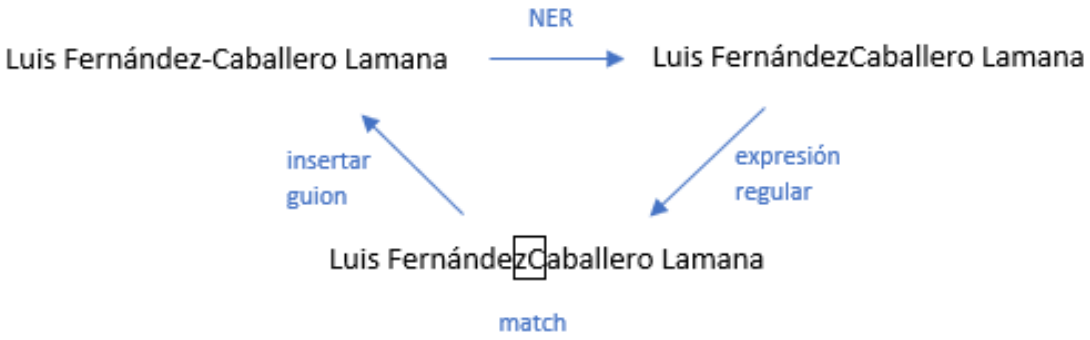

Figura II.1: Proceso recuperación guión

El nombre de María también produce un pequeño contratiempo y es que el NER transforma Mª a María, para solucionar esto lo único que se hace es, en caso de haber extraído María, comprobar si en la frase original se encuentra Mª, si es así se hace una simple sustitución para trabajar con el nombre original.

A lo largo de esta presentación de problemas encontrados, se ha hecho hincapié en un concepto: **nombre original**, esto es debido a que para tokenizar la frase original es necesario hacer un emparejamiento exacto, exceptuando mayúsculas y minúsculas, ya que es necesario que los nombres sean un único token y que la frase quede segmentada por entidades del NER y por palabras una vez estas entidades han sido sustituidas.

La tabla con los nombres que se ha mencionado que no ha sido posible detectar de otra manera más que haciendo emparejamiento directo sobre el texto son los siguientes:

```
nombres = ["Pilar de la Vega Cebrián", "Hipólito Gómez de las Roces",
           "María Soledad de la Puente Sánchez",
           "Gaspar Castellano y de Gastón", "Marta de la Rosa Lamata",
           "Vacante temporal", "José María Hernández de la Torre y García",
           "María de las Mercedes Navarro Gajón",
           "César de la Parte Serna",
           "Angel García Viana y Caro",
           "Vitelio Manuel Tena Piazuelo"]
```

II.2. Roles

Para la detección de roles se observo que los roles los devolvía en el mismo resultado que las personas, es decir, en una lista de elementos separados por coma, había que generar una solución que fuera capaz de diferenciar que elemento era el nombre de una persona y cual era un cargo o rol. Para ello se partió de una tabla de prefijos comunes en los cargos, con esto y una expresión regular que comprueba si comienza por alguno de esos prefijos es suficiente para detectar un rol en primera instancia.

Posteriormente se comprobó que esta primera aproximación no era suficiente para cubrir todos los roles, hay situaciones en las que el NER no extrae la entidad completa. El primer problema encontrado detectar `subdirector/a` como rol ya que el NER detecta `director/a`, suprimiendo el "sub", para resolver estas situaciones, cada vez que se extrae un rol que contiene `director/a` se comprueba si en la frase a procesar se encuentra `subdirector/a`, en caso de que suceda, se sustituye `director/a` por `subdirector/a` para conseguir el rol correcto.

Otra situación especial sucede cuando extrae parte del rol, es decir, no extrae todo el rol ya que alguna palabra corta el reconocimiento de la entidad. Ejemplo: `Jefe/a del Servicio de Relaciones Institucionales y con las Cortes de Aragón`, el NER sólo es capaz de reconocer `Jefe/a del Servicio de Relaciones Institucionales` suprimiendo el resto del cargo, para la resolución de estos casos se hace una búsqueda del rol extraído en la frase original y le concatenamos lo que haya desde el final del emparejamiento hasta el final de la frase, esto es factible debido a que el formato de los datos que se ha definido, posiciona el rol al final de las frases en las que aparece uno.

Al igual que con los nombres propios, existen algunos casos en los que el NER no es capaz de extraer la entidad correctamente, esto puede ser debido a diversos motivos.

- No detecta ninguna entidad como rol ya que no contiene ninguno de los prefijos de la tabla. Ejemplo: `Titular de la Secretaria Particular`
- Detecta dos roles separados en uno que realmente es combinado. Ejemplo: `Jefa de Servicio de Régimen Jurídico y Coordinación Administrativa`, el NER devuelve dos cargos por separado, `Jefa de Servicio de Régimen Jurídico` por un lado y `Jefa de Servicio de Coordinación Administrativa`
- El cargo original está mal definido en la base de datos. Ejemplo: `GERENTE DE ARAGONESA DE SERVICIOS TELEMÁTICOS`

Todos estos casos al igual que con los nombres propios se preprocesan buscando si aparece alguno de ellos en la frase y extrayéndolo directamente.

Anexos III

Framework conversacional

Para la creación del *chatbot* se ha utilizado la librería *open-source* Rasa. Es un *framework* de código abierto, altamente escalable en *Machine Learning* útil para la construcción y creación de software conversacional, por ejemplo, un asistente virtual. A lo largo de este apéndice se va introducir las características que componen Rasa así como una breve introducción a su funcionamiento.

Rasa se compone de un par de librerías (Rasa NLU & Rasa Core), las cuales permiten a los desarrolladores a expandir los *chatbots* mucho más allá de responder preguntas básicas, los *chatbots* pueden mantener conversaciones contextuales con los usuarios.

Dentro de los agentes virtuales existen dos pilares fundamentales, la comprensión del texto: extracción de entidades, eliminación de *stopwords*...Y un segundo pilar que se encarga del control del diálogo, es decir, comprender el orden de la conversación, mantener las palabras más relevantes de la misma y construir una conversación que tenga sentido para el usuario. Es por esto que existen dos módulos independientes: Rasa NLU y Rasa Core, el primero toma texto en formato libre y lo convierte en datos estructurados y el segundo actúa como gestor del diálogo, mantiene el flujo de la conversación y decide como proceder con la misma.

A continuación se desglosa cada una de las librerías, explicando brevemente y a un nivel básico el funcionamiento de cada una de ellas, para un funcionamiento en detalle con varios ejemplos prácticos, visitar la página Web oficial de Rasa. La razón principal por lo que se ha optado por utilizar Rasa es que es una librería de código abierto y por lo tanto no supone ningún coste, además es una de las tecnologías nuevas de gestión de diálogo más completa y con desarrollo continuo, tienen un foro para responder cuestiones sobre el uso de Rasa, además, Rasa no utiliza servidores de Microsoft o Google, por lo que se puede utilizar en entornos internos que requieren mantener la confidencialidad de los datos.

III.1. Rasa NLU

Herramienta de procesamiento del lenguaje natural de código abierto para clasificación de intenciones y extracción de entidades. Un ejemplo gráfico:

```
"I am looking for a Mexican restaurant in the center of town"
```

Rasa procesaría la frase y devolvería una estructura como la siguiente:

```
{
  "intent": "search_restaurant",
  "entities": {
    "cuisine" : "Mexican",
    "location" : "center"
  }
}
```

Figura III.1: Ejemplo estructura información extraída

Como se puede observar Rasa se encarga de detectar entidades e intenciones a partir de texto plano. Las ventajas más importantes de utilizar comprensión del lenguaje natural (*NLU*) de código abierto es que no dependes de las grandes compañías como Google, Microsoft o Facebook para entrenar datos, Rasa también permite personalizar tus modelos de Machine Learning para que sean lo más ajustado y eficiente para tus datos y por último pero no menos importante, Rasa NLU se ejecuta en cualquier sistema, por lo que no se necesitan conexiones extras remotas para procesar tus datos. Para más información y código de ejemplo del funcionamiento interno de Rasa, visitar este blog o su GitHub [**?**].

III.2. Rasa Core

En este apartado se va a explicar una a una todas las piezas que forman el control del diálogo de rasa. Para empezar hay que definir que Rasa Core funciona aprendiendo de conversaciones de ejemplo y generalizando a partir de ellas. Una historia se define como una serie de intenciones seguidas de acciones, estas acciones pueden ser desde enviar un mensaje hasta realizar una llamada a una API externa. Ejemplo:

El formato de las historias se define usando formato *Markdown*, las frases que comienzan con '`##`' son la cabecera de la historia, útil para etiquetar y diferenciar historias, las líneas que comienzan por '$*$' son las intenciones detectadas por el bot en función de una entrada del usuario y las que comienzan por '$-$' son las acciones que lleva a cabo el bot una vez

```
## historia
* saludar
   - utter_saludo
```

Figura III.2: Ejemplo historia

reconocida una intención e identificada una historia. A continuación se van a aclarar todos estos conceptos de intenciones y acciones y como el bot es capaz de identificarlos y actuar en consonancia.

En segundo lugar es necesario establecer un dominio, este representa el "mundo" en el que el bot va a "vivir". Un ejemplo de dominio sería:

```
intents:
  - greet
  - goodbye
  - mood_affirm
  - mood_deny
  - mood_great
  - mood_unhappy

actions:
- utter_greet
- utter_cheer_up
- utter_did_that_help
- utter_happy
- utter_goodbye

templates:
  utter_greet:
  - text: "Hey! How are you?"

  utter_cheer_up:
  - text: "Here is something to cheer you up:"
    image: "https://i.imgur.com/nGF1K8f.jpg"

  utter_did_that_help:
  - text: "Did that help you?"

  utter_happy:
  - text: "Great carry on!"

  utter_goodbye:
  - text: "Bye"
```

Figura III.3: Dominio simple bot

¿Qué significan cada uno de estos elementos definidos en el dominio?

actions	cosas que el bot puede hacer y decir
templates	cadenas de texto para las cosas que el bot puede decir
intents	cosas que se espera que haga el usuario
entities	piezas de información que se quieren extraer de los mensajes
slots	dónde se guarda la información relevante de la conversación

Tabla III.1: Términos dominio

El ejemplo de dominio proporcionado es muy simple pero útil para la comprensión, para tener más detalle de como funcionan los *slots*, las *entities* o las acciones personalizadas, visitar la documentación en Rasa Core.

El siguiente paso es hacer que el *chatbot* comprenda el lenguaje natural para establecer una comunicación con el usuario y que esta tenga sentido. Un interprete es responsable de procesar los mensajes. Realiza la comprensión del lenguaje natural (*NLU*) y transforma estos mensajes en datos estructurados. Para esto se puede utilizar tanto Rasa NLU como cualquier otro software de comprensión de texto disponible. En este caso vamos a introducir como sería usando Rasa NLU.

Con Rasa NLU, necesitamos definir los mensajes del usuario que el bot debe ser capaz de gestionar y reconocer, se definen estos datos en un fichero, se puede utilizar tanto formato *Markdown* como formato *JSON*, en este caso usaremos el formato *Markdown*. Ejemplo en III.4

Como se puede observar para cada una de las intenciones definidas en el dominio, se han creado diferentes expresiones o frases que el usuario podría utilizar y se asocian con dicha intención. Además se debe definir un fichero de configuración para determinar que componentes se van a utilizar para la extracción de entidades, el lenguaje de los datos, componentes externos para realizar otro tipo de tareas como extracción de fechas, tokenizador. Ejemplo de fichero de configuración en III.5

Todos estos componentes son los necesarios para poner en marcha un bot usando Rasa, para obtener más información de como se tienen que entrenar los datos mediante scripts de Python o comandos, de los diferentes componentes que pueden definir un fichero de configuración y ver ejemplos de diferentes bots en funcionamiento, visitar la documentación completa en Rasa, todos los bots de ejemplo de la página tienen su código en GitHub [**?**].

```
## intent:greet
- hey
- hello
- hi
- good morning
- good evening
- hey there

## intent:goodbye
- bye
- goodbye
- see you around
- see you later

## intent:mood_affirm
- yes
- indeed
- of course
- that sounds good
- correct

## intent:mood_deny
- no
- never
- I don't think so
- don't like that
- no way
- not really

## intent:mood_great
- perfect
- very good
- great
- amazing
- I am feeling very good
- I am great
- I'm good

## intent:mood_unhappy
- sad
- very sad
- unhappy
- bad
- very bad
- awful
- terrible
- not very good
- extremly sad
```

Figura III.4: Datos de entrenamiento

```
language: "es"

pipeline:
- name: "nlp_spacy"
  model: "es"
  case_sensitive: true
- name: "tokenizer_spacy"
- name: "intent_featurizer_spacy"
- name: "intent_entity_featurizer_regex"
- name: "intent_classifier_sklearn"
- name: "ner_crf"
- name: "ner_synonyms"
- name: "ner_duckling"
  dimensions: [ "time", "duration", "amount-of-money"]
  language: "es"
```

Figura III.5: Configuración bajo la que funciona el bot

Anexos IV

Herramientas y tecnologías utilizadas

En el presente Anexo se listan todas las herramientas utilizadas para la realización del proyecto.

- **Python:** Lenguaje de programación utilizado para la implementación del sistema, en concreto usando las librerías de `Tensorflow`, `Rasa` y `Flask`
- **PyCharm:** Entorno de desarrollo integrado utilizado en la programación de computadora, específicamente para el lenguaje Python, desarrollado por la compañía JetBrains
- **Jupyter Notebook:** Entorno de desarrollo de Python que permite crear y compartir documentos que contienen código, ecuaciones y texto.
- **Conda:** Gestor de paquetes de Python, permite crear entornos personales por lo que cada usuario tiene instalado sólo los paquetes que necesite para su proyecto.
- **Moriarty Studio:** Plataforma desarrollada por el departamento de Big Data y Sistemas Cognitivos del ITA, se compone de un conjunto de herramientas para el desarrollo de servicios relacionados con el científico de datos de manera ágil mediante el uso de cuadernos de *Jupyter Notebook* y el gestor Conda.
- **Tex:** Sistema de tipografía desarrollado por Donald E. Knuth especialmente usado para el desarrollo de artículos científicos por su eficiente manera de componer fórmulas matemáticas complejas pero, especialmente en la forma de LaTex y otros paquetes de macros.
- **LaTex:** Sistema de composición orientado para la creación de documentos escritos con una alta calidad tipográfica, se basa en el uso del lenguaje de composición tipográfica

Tex. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados en LaTeX, se la considera adecuada a las necesidades de una editorial científica de primera línea, muchas de las cuales ya lo emplean.

- **Overleaf:** Herramienta colaborativa de escritos en LaTex que facilitan la escritura de documentos científicos más fácil y rápidamente.
- **Shell Script:** Programa diseñado para ser ejecutado por un interprete de linea de comandos para la realización de tareas.
- **HTML, CSS y JS:** Tecnología para el desarrollo Web.

Anexos V

Pseudocódigo

En este anexo se incluye el pseudocódigo del algoritmo seguido para determinar si utilizar o no el modelo de red neuronal clave-valor, en el capítulo 3 se enumeraron una serie de requisitos que debía cumplir la entrada del usuario para utilizar éste modelo. Se ha utilizado una función con dos parámetros de entrada, el mensaje del usuario y la sesión de `TensorFlow` en la cual se tiene el modelo cargado y preparado para predecir nuevas salidas a partir de entradas validas. El modelo ya se encuentra poblado con los pesos guardados y con la historia sobre la que responder. Existen variables globales inicializadas fuera del algoritmo como el diccionario de palabras que aparecen en los datos. Un listado con todas las variables globales que se utilizan durante la ejecución junto a su funcionalidad es la siguiente:

1. **Diccionario**: contiene todas las palabras del modelo de red neuronal, necesario para acceder al valor de la cadena en función del resultado numérico de la red.
2. **Modelo**: restauración del mejor modelo para predecir respuestas durante la conversación.
3. **Interprete Rasa**: interprete del lenguaje natural, se encarga de extraer entidades, detectar intenciones, entidades...etc.
4. **Sesión de tensorflow**: sesión para utilizar el modelo de red neuronal.
5. **Agente**: controlador del diálogo, encargado de detectar que respuesta corresponde para las diferentes entradas del usuario.
6. **Nombre de usuario**: variable global que se actualiza cada vez que el usuario informa de su nombre, se utiliza visualmente en la Web para dar dinamismo a la conversación.

Algoritmos de predicción de la respuesta, un primer algoritmo 1 para la predicción del modelo de red neuronal, y un segundo 2 con la integración del primero con Rasa.

Algorithm 1 Predicción OpenData

procedure RespuestaOpenData(*mensaje*, *sesion*)
    mensaje ← *eliminar_tildes*(*mensaje*)
    tokens, *es_pregunta* ← *procesar_entrada*(*mensaje*)
    if not *es_pregunta* **then**
        return *lista_vacia*, *lista_vacia*
    else if *len*(*tokens*) > 5 **then**
        return *lista_vacia*, *lista_vacia*
    existe, *nombre_original* ← *get_nombre_original*(*mensaje*)
    if not *existe* **then**
        return *lista_vacia*, *lista_vacia*
    tokens[*tokens.length*] = *nombre_original*
    num_tokens ← palabras en *tokens* que estan en el diccionario
    if *num_tokens* < 3 **then**
        return *lista_vacia*, *lista_vacia*
    tokens_vectorizados ← *vectorizar_tokens*(*tokens*)
    id_respuesta, *vector_probabilidades* ← *predecir*(*sesion*, *tokens_vectorizados*)
    respuesta, *confianza* ← *buscar_en_vocabulario*(*id_respuesta*)
    return *respuesta*, *confianza*

Algorithm 2 Respuesta chatbot

procedure RespuestaChatbot(*entrada_usuario*)
    respuesta, *confianza* ← RespuestaOpenData(*entrada_usuario*, *sesion*)
    if *respuesta* ≠ *lista_vacia* **and** *confianza*[0] > 0,75 **then**
        return *respuesta*
    respuesta_rasa, *entrada_procesada* ← PrediccionRasa(*entrada_usuario*)
    if *entrada_procesada.entidades.ultima* = *informar/cambiar nombre* **then**
        actualizar nombre de usuario en el chat
    if *entrada_procesada.entidades.nombre* = *entidad_meteorologia* **then**
        return *respuesta*, *iconos_tiempo*
    else
        return *respuesta*

Anexos VI

Integración *chatbot* con avatar virtual

En este anexo se exponen diferentes imágenes del chatbot usado junto al avatar virtual. El avatar procesa el texto del usuario ya sea escrito o hablado, con este texto se realiza una llamada al *backend* del sistema, es este el encargado de predecir una respuesta a la entrada del usuario. Este *backend* utiliza la tecnología desarrollada en este proyecto para controlar la conversación y responder de manera adecuada.

Figura VI.1: Avatar 1 - Posición inicial

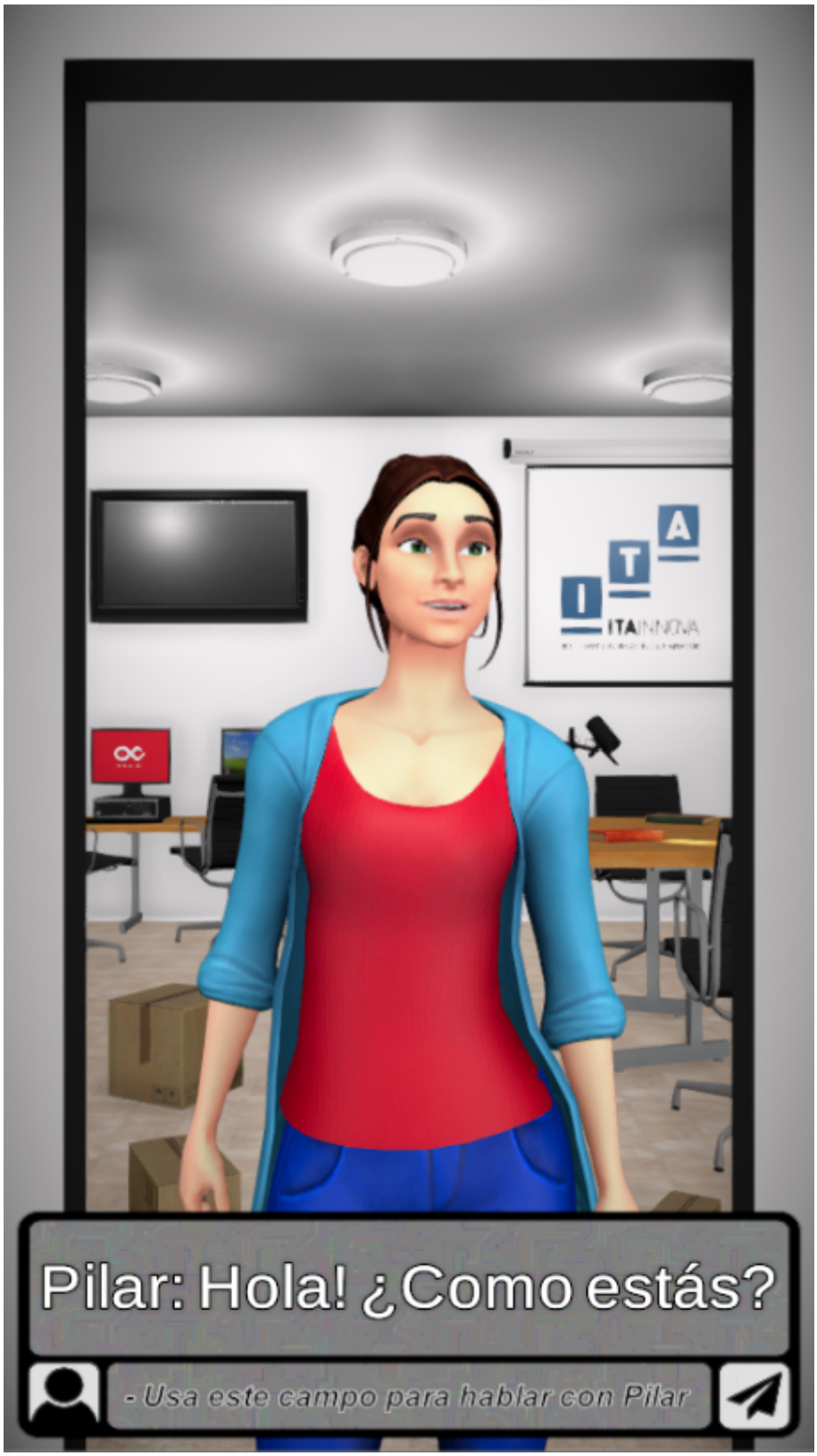

Figura VI.2: Avatar 2 - Respuesta a “Hola”

Figura VI.3: Avatar 3 - Respuesta a “Número de trabajadores en el ITA”

Figura VI.4: Avatar 4 - Respuesta a “Tiempo mañana”

Anexos VII

Gestión del proyecto

En este anexo se describe la metodología de trabajo seguida para la realización del proyecto y la estimación del tiempo empleado para cada una de las diferentes fases que han integrado el proyecto.

VII.1. Metodología de trabajo

Para el desarrollo del proyecto se han utilizado técnicas características de diferentes metodologías ágiles como Scrum, Extreme Programming y Kanban. Como el proyecto no se puede enmarcar dentro de ninguna de las metodologías de una manera estricta, se han usado diferentes prácticas como:

1. **Scrum**
    - Solapamiento de las diferentes fases del proyecto
    - Desarrollo del producto de manera incremental a través de iteraciones (Sprints)
    - Revisiones periódicas al final de cada iteración
2. ***Extreme Programming***
    - Planificación flexible y abierta
    - Desarrollo de software funcional frente a conseguir una buena documentación
3. **Kanban**
    - Diario del proyecto
    - Etiquetado de tareas en progreso, completadas y pendientes.

El proyecto se ha dividido en seis fases principales:

1. FASE 1: Estudio y análisis de la documentación sobre las tecnologías utilizadas en la elaboración del proyecto.
2. FASE 2: Diseño y especificación del sistema, sus componentes y arquitectura.
3. FASE 3: Implementación y desarrollo del sistema.
4. FASE 4: Validación y verificación de todos los módulos que integran el sistema.
5. FASE 5: Despliegue del sistema.
6. FASE 6: Documentación del proyecto, redacción de la memoria.

VII.2. Dedicación de esfuerzos

Este proyecto ha sido desarrollado durante seis meses, desde abril de 2018 hasta octubre del mismo año. Se han invertido un total de 900 horas, un despliegue de los esfuerzos dedicados para cada una de las fases del proyecto quedan reflejados en la Figura VII.1.

Gestión del proyecto

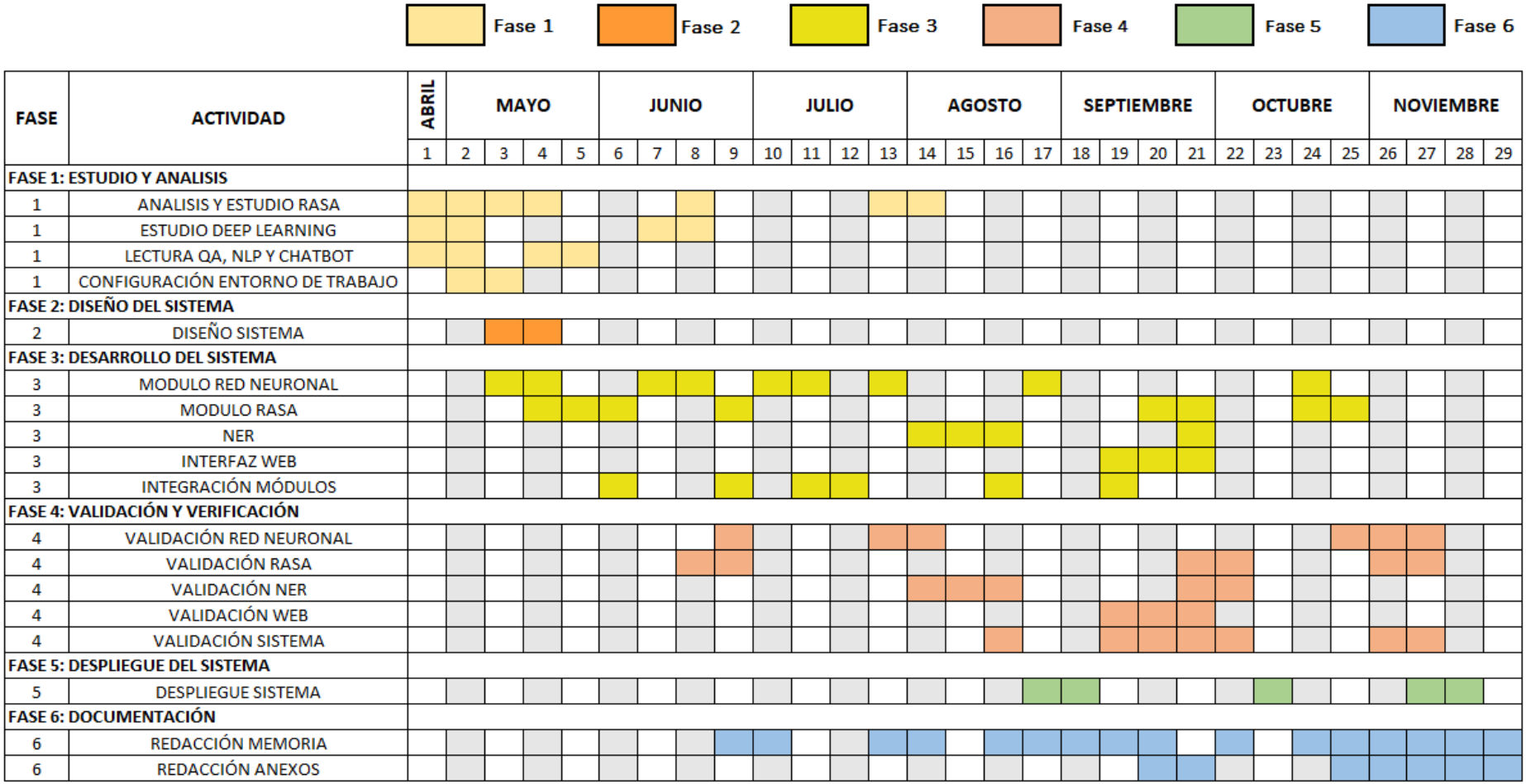

Figura VII.1: Diagrama de Gantt