

Tesis de grado en Ingeniería en Informática

Comportamiento adaptable de chatbots dependiente del contexto



Laboratorio de Sistemas Inteligentes

Facultad de Ingeniería

Universidad de Buenos Aires

Tesista: Juan Manuel Rodríguez

Directores: M. Ing. Hernán Merlino,
M. Ing. Enrique Fernández

Tabla de Contenidos

1. Introducción.....	1
1.2 Objetivos del trabajo.....	1
1.3 Estructura de la tesis.....	2
2. Estado de la cuestión.....	4
2.1 Chatbots.....	4
2.1.1 Orígenes del planteo.....	4
2.1.2 Primera aproximación real: ELIZA.....	5
2.1.3 Concurso Loebner (Loebner Prize).....	5
2.1.4 Implementación y utilización comercial de chatbots.....	7
2.2 Procesamiento de Lenguaje Natural.....	8
2.2.1 Pragmática.....	8
2.2.2 Resolución de referentes.....	9
2.3 Problemática actual.....	9
2.4 Tecnología a utilizar.....	10
2.4.1 Redes neuronales artificiales.....	10
2.4.1.1 Perceptrón multicapa.....	11
2.4.1.1.1 Perceptrón multicapa como aproximador universal de funciones.....	13
2.4.1.2 Back Propagation.....	14
2.4.1.3 Redes SOM, Kohonen.....	16
2.4.1.3.1 Funcionamiento de una red de Kohonen.....	17
2.4.2 Algoritmo Resolution of Anaphora Procedure.....	18
2.4.3 AIML.....	19
2.4.3.1 Manejo del contexto en AIML.....	20
2.4.3.2 ALICE.....	22
3. Definición del problema.....	23
3.1 Problemática actual.....	23
3.2 Alcance propuesto.....	25
3.3 Ejemplos de casos reales.....	26
4. Solución propuesta.....	30
4.1 Esbozo general de la solución.....	30
4.2 Distinción entre casos favorables y casos desfavorables.....	31
4.3 Clasificación de los casos usando una RNA.....	33
4.3.1 mapa 1 propuesto como input de la RNA.....	34

4.3.2 mapa 2 propuesto como input de la RNA.....	35
4.3.3 Pasos para la clasificación.....	37
4.3.4 Utilización de redes neuronales artificiales autoorganizadas (SOM).....	37
4.4 Implementación de AIML sobre mySQL.....	38
4.5 Resolución de referentes.....	38
4.6 Reformulación de respuestas a preguntas cerradas.....	39
4.7 Preguntas por referencias no encontradas.....	40
5. Desarrollo de la solución.....	42
5.1 Estudio de viabilidad del sistema.....	42
5.1.1 Establecimiento del alcance del sistema.....	42
5.1.1.1 Estudio de la solicitud.....	42
5.1.1.1.1 Resumen de la propuesta de tesis.....	43
5.1.1.1.2 Descripción del sistema a construir y objetivos.....	43
5.1.1.2 Identificación del alcance del sistema.....	44
5.1.1.2.1 Diagramas de flujo.....	44
5.1.1.2.2 Requisitos.....	45
5.1.1.2.3 Usuarios.....	45
5.1.1.3 Especificación del alcance del sistema.....	46
5.1.2 Estudio de la situación actual.....	47
5.1.2.1 Valoración del estudio de la situación actual.....	47
5.1.2.2 Descripción de los sistemas de información existentes.....	49
5.1.2.2.1 Diagramas de flujo de ALICE y ELIZA.....	49
5.1.3 Definición de requisitos del sistema.....	52
5.1.4 Estudio de alternativas de solución.....	52
5.1.4.1 Chatbots.....	53
5.1.4.2 Sub-sistema detector.....	53
5.1.4.3 Sub-sistema conversor	55
5.1.5 Valoración de alternativas.....	56
5.1.6 Selección de la solución.....	56
5.1.7 Aceptación de la solución.....	57
5.2 Análisis del sistema de información.....	58
5.2.1 Definición del sistema.....	58
5.2.1.1 Determinación del alcance del sistema.....	58
5.2.1.1.1 Casos de uso.....	58
5.2.1.1.3 Diagramas de Entidad – Relación.....	60

5.2.1.1.4 Diagramas de clases.....	62
5.2.1.2 Identificación del entorno tecnológico.....	63
5.2.1.3 Especificación de estándares y normas.....	64
5.2.2 Establecimiento de requisitos.....	64
5.2.2.1 Obtención de requisitos y tareas.....	64
5.2.3 Identificación de subsistemas de análisis.....	67
5.2.3.1 Determinación de subsistemas de análisis.....	67
5.2.4 Análisis de casos de uso.....	70
5.2.4.1 Identificación de clases asociadas a un subsistema.....	70
5.2.5 Análisis de clases.....	72
5.2.5.1 Identificación de responsabilidades y atributos.....	72
5.2.5.2 Identificación de asociaciones y agregaciones.....	74
5.2.5.3 Identificación de generalizaciones.....	74
5.2.6 Definición de interfaces de usuario.....	75
5.2.6.1 Especificación de principios generales de la interfaz.....	75
5.2.7 Especificación del plan de pruebas.....	76
5.2.7.1 Definición del alcance de las pruebas.....	76
5.2.7.1.1 Pruebas de sistema.....	76
5.2.7.1.2 Prueba de integración del sistema.....	83
5.2.7.2 Definición de las pruebas de aceptación del sistema.....	86
5.2.8 Aprobación del análisis del sistema de información.....	86
5.3 Diseño del sistema de información.....	87
5.3.1 Definición de la arquitectura del sistema.....	87
5.3.1.1 Definición de niveles de arquitectura.....	87
5.3.1.2 Especificación de excepciones.....	89
5.3.1.3 Especificación de estándares y normas de diseño y construcción.....	90
5.3.2 Diseño de la arquitectura de soporte.....	90
5.3.2.1 Diseño de subsistemas de soporte.....	90
5.3.2.1.1 Uploader.....	90
5.3.2.1.2 UpdateDatabase.....	93
5.3.2.1.3 Código reutilizado.....	93
5.3.3 Diseño de casos de uso reales.....	94
5.3.3.1 Diseño de la realización de los casos de uso.....	94
5.3.3.2 Revisión de la interfaz de usuario.....	99
5.3.4 Diseño de clases.....	100

5.3.4.1 Identificación de clases adicionales.....	100
5.3.4.2 Identificación de atributos y operaciones de las clases.....	103
5.3.4.3 Diseño de la jerarquía de paquetes.....	115
5.3.5 Diseño físico de datos.....	116
5.3.5.1 Diseño del modelo físico de datos.....	116
5.3.5.2 Optimización del modelo físico de datos.....	119
5.3.6 Carga inicial de datos.....	119
5.3.6.1 Diseño de procedimientos de carga inicial de datos.....	119
5.3.6.1.1 Uploader.....	120
5.3.6.1.1.1 Diccionario de palabras.....	120
5.3.6.1.1.2 Tipo de palabra y sinónimos.....	120
5.3.6.1.1.3 Archivos AIML.....	121
5.3.6.1.2 Sentencias SQL, creadas desde el editor.....	121
5.3.7 Establecimiento de requisitos de implantación.....	122
5.3.7.1 Especificación de requisitos de implantación.....	122
5.3.8 Aprobación del diseño del sistema de información.....	125
5.4 Construcción del sistema de información.....	126
5.4.1 Procedimientos de migración y carga inicial de datos.....	126
5.4.1.1 Realización y evaluación de las pruebas de migración y carga inicial de datos..	126
5.4.1.1.1 Pruebas de la carga de datos de myAlice.....	126
5.4.1.1.2 Pruebas de la carga de datos de Lexico.....	132
5.4.2 Pruebas funcionales de cada subsistema.....	136
5.4.2.1 Comportamiento de myAlice sin las mejoras.....	136
5.4.2.2 Verificación del detector de tiempos verbales.....	146
5.4.2.3 Verificación del detector de frases de interrogación.....	159
5.4.2.4 Verificación del mapa generado, con mapeador 01.....	162
5.4.2.5 Verificación del mapa generado, con mapeador 02.....	164
5.4.2.6 Verificación del correcto funcionamiento de la RNA.....	169
5.4.2.7 Verificación del sistema de conversión de respuestas cerradas.....	202
5.4.2.8 Verificación del sistema de generación de preguntas.....	204
5.4.2.9 Verificación del sistema de resolución de referentes.....	205
5.4.3 Ejecución de las pruebas de integración.....	208
5.4.3.1 Integración del sistema generador del mapa y la red neuronal artificial.....	208
5.4.3.2 Integración de myAlice con el sistema detector.....	209
5.4.3.3 Integración de myAlice con el subsistema conversor.....	216

5.4.4 Ejecución de las pruebas de aceptación del sistema.....	222
5.4.5 Aprobación de sistema de información.....	224
6. Conclusiones.....	225
7. Futuras líneas de investigación.....	227
8. Referencias.....	228
Anexo A: Glosario.....	230
Anexo B: Datos del repositorio on-line.....	231

1. Introducción

El tema de este trabajo de investigación son los programas (*software*) llamados comúnmente *chatbots*. Estos son programas, *software*, algoritmos de computadora, que utilizan procesamiento de lenguaje natural (*NLP: Natural Language Processing*) en un sistema de preguntas y respuestas (*QA systems: question-answering systems.*) Estos sistemas han sido definidos también como sistemas expertos que usan razonamiento basado en casos (*CBR: case base reasoning*) [Wallace 03].

El ideal perseguido detrás de estos sistemas es que puedan "comportarse" de forma indistinguible de un ser humano, donde en este caso "comportarse" significa que puedan dar respuestas satisfactorias a un interlocutor humano tal y como lo haría una persona durante un dialogo. Hoy en día los chatbots existentes están lejos de lograr este cometido aunque existen sistemas que han dado respuestas suficientemente satisfactorias como para engañar a un interlocutor humano al menos durante unos instantes, ejemplos de estos diálogos pueden verse en [Loebner 2009]; en este trabajo se analizarán en detalles los sistemas ALICE y ELIZA.

El interés detrás del ideal de funcionamiento de un chatbot va desde lo filosófico, hasta lo pragmático, el matemático británico Alan Turing sostuvo en [Turing 1950] que un chatbot capaz de pasar cierta prueba, hoy conocida como *Test de Turing* era un fuerte indicio de que las maquinas podían pensar. Por otro lado el creador de AIML sostiene en [Wallace 03] que un sistema capaz de comunicarse mediante el dialogo con un ser humano podría convertirse en un nuevo tipo de interfaz de usuario, en donde el usuario dejaría de utilizar comandos, ventanas, iconos, etc. para simplemente indicarle al sistema lo que quiere hacer de forma meramente coloquial.

A pesar de las obvias limitaciones de los chatbots actuales muchos son utilizados comercialmente de forma más o menos exitosa, son utilizados como agentes automáticos en videojuegos, asistentes de mesa de ayuda virtual o "amigos virtuales" en redes de mensajería (se pueden ver ejemplos concretos en la sección 2.1.4)

Finalmente cabe destacar que el problema del correcto funcionamiento de los chatbots se enmarca dentro de la problemática del procesamiento del lenguaje natural, la cual es una de las áreas más complejas dentro de los llamados sistemas inteligentes.

1.2 Objetivos del trabajo

El objetivo del presente trabajo es buscar una forma de mejorar la calidad de las respuestas y del dialogo en general que puede generar un chatbot; para ello se tomará una tecnología (en el termino más amplio posible, entiéndase: lenguaje de programación, conjunto de datos, metodología de trabajo y/o comportamiento general de un algoritmo) utilizada en un chatbot existente y se mejorará su comportamiento en relación con la interpretación pragmática del discurso (o dialogo en este caso). Más adelante, en la sección 3.1 se detallarán algunos de los problemas encontrados en los chatbots, como ser la dificultad que tienen estos sistemas para dar una respuesta satisfactoria cuando esta depende de elementos presentes en el "contexto", como ser la resolución anafórica de referentes.

1.3 Estructura de la tesis

Esta tesis está estructurada en siete capítulos: Introducción, Estado de la cuestión, Definición del problema, Solución propuesta, Desarrollo de la solución, Conclusiones y Futuras líneas de investigación.

Durante la fase de desarrollo e investigación se han seguido los lineamientos generales de la metodología: *Métrica versión 3*, De Métrica se han tomado las siguientes fases:

fase 2: Estudio de viabilidad,

fase 3: Análisis de sistema,

fase 4: Diseño del sistema y

fase 5: Concepción.

El capítulo: *Estado de la cuestión* describe el comportamiento de los chatbots actuales que más se asemejan a un ser humano en su forma de responder y se describen las tecnologías, sistemas y algoritmos que se utilizarán para el desarrollo de la tesis.

En el capítulo: *Definición del problema* se listan los problemas encontrados en la actualidad en los chatbots antes descriptos.

En el capítulo: *Solución propuesta* se describe la solución que se propone para solucionar los problemas antes señalados.

En el capítulo: *Desarrollo de la Solución* se describe todo el procedimiento relacionado con el desarrollo del sistema según la metodología Métrica versión 3, es decir el análisis del problema, su concepción, se muestra además información técnica como diagramas de clase, de entidad relación y de tablas y finalmente se listan las pruebas funcionales hechas sobre el sistema

construido.

En *Conclusiones* se describe el resultado final del trabajo de investigación y, valga la redundancia, las conclusiones a las que se arribó.

En *Futuras líneas de investigación*: el último capítulo hace referencia a los tópicos sin cubrir o nuevos indicios dignos de ser investigados y abordados en futuros trabajos.

2. Estado de la cuestión

El presente capítulo describe los algoritmos y sistemas relevantes sobre el tema de estudio al momento de escribir esta tesis así como los avances científicos en el área. La primera sección muestra el estado de los chatbots actuales en general (2.1), la segunda sección presenta la utilización de sistemas inteligentes en relación con el procesamiento de lenguaje natural (2.2), en la tercer sección (2.3) se hace una introducción a la problemática actual, la cual será desarrollada en el próximo capítulo y por último en la cuarta sección (2.4) del presente capítulo se detallan las tecnologías que serán utilizadas en la resolución del problema.

2.1 Chatbots

Los chatbots son programas, *software*, (entiéndase un conjuntos de algoritmos), que utilizan procesamiento de lenguaje natural (*NLP: Natural Language Processing*) en un sistema de preguntas y respuestas (*QA systems: question-answering systems*) [Fitrianie, 2002]. Estos sistemas han sido definidos también como sistemas expertos que usan razonamiento basado en casos (*CBR: case base reasoning*) [Wallace, 2003]. La finalidad de dichos sistemas es simular un dialogo inteligente con interlocutor humano, ya sea mediante mensajes de texto a través de una consola o bien mediante la voz.

2.1.1 Orígenes del planteo

En 1950 el matemático británico Alan Turing en una publicación científica propuso una forma de responder al interrogante: "¿Pueden pensar las maquinas?" [Turing, 1950]. Su método consistía en la implementación de un juego al que llamó: *The imitation game* (el juego de la imitación) en el cual participan dos personas y una maquina. Una de las personas actúa como juez o jurado y debe discernir quien de los restantes es la persona. Por supuesto que tanto la otra persona como la máquina no están en la misma habitación que el jurado, y este solo puede comunicarse con ambas mediante preguntas escritas, las cuales son respondidas en igual forma. Turing sostuvo que si la máquina podía engañar al jurado por una cantidad de tiempo a definir, entonces se podría considerar que dicha máquina piensa. A esta prueba se la llama comúnmente: *Test de Turing*

Al día de hoy no ha sido creado un sistema capaz de pasar de forma satisfactoria dicha

prueba.

La premisa de Turing de que una máquina capaz de pasar dicha prueba es una máquina que puede pensar fue rebatida por Jhon Searle [Searle, 1980] con el el argumento de la habitación china. También Roger Penrose [Penrose, 1989] ha rebatido mediante otros argumentos el punto de vista de Turing.

2.1.2 Primera aproximación real: ELIZA

El primer chatbot que logró responder a interrogantes hechos a través de una consola de texto y confundir a una persona al punto de no saber esta si en verdad estaba hablando con una máquina o no fue ELIZA. Por supuesto que dicha confusión se daba solo en las primeras líneas del dialogo y con determinadas frases.

ELIZA fue diseñado en 1966 por Joseph Weizenbaum. El chatbot parodia a un psicólogo en su manera de responder ya que en general sus respuestas son reformulaciones de las frases de entrada del usuario [Weizenbaum, 1966]

De forma esquemática, el algoritmo detrás de ELIZA busca palabras claves en una frase de entrada de un usuario y luego utilizando el resto de la oración hace una reformulación de la sentencia original, esta forma de responder solo es coherente en el ámbito del psicoanálisis, entre las partes: paciente y doctor. Por ejemplo: ante la frase de entrada: "It seems that you hate me", ELIZA detectaría las palabras claves: "you" y "me". Aplicaría luego un *template* que descompondría la frase original en cuatro partes:

(1) It seems that (2) you (3) hate (4) me.

Finalmente realizaría la reformulación de la frase original, una respuesta posible sería: "What makes you think **I hate you**" [Weizenbaum, 1966].

ELIZA tiene además otros mecanismos para armar respuestas, pero el descripto es el principal.

2.1.3 Concurso Loebner (Loebner Prize)

El Premio Loebner de Inteligencia Artificial (*The Loebner Prize for artificial intelligence*) es la primera instancia formal de un *Test de Turing*. En 1990 Hugh Loebner acordó con El Centro de Estudios del Comportamiento de Cambridge (The Cambridge Center for Behavioral Studies) diseñar un certamen designado para implementar el *Test de Turing*. El Dr. Loebner ofreció un premio de 100 000 dólares y una medalla de oro para la primera computadora cuyas respuestas fueran indistinguibles de respuestas humanas. Todos los años en dicho certamen se otorga un

premio de 2 000 dólares y una medalla de bronce a la computadora que se acerca más a un ser humano en su forma de responder. El ganador del certamen anual es el mejor en relación con los otros concursantes de ese mismo año y no en un sentido absoluto. [Loebner 2006]

Los primeros años los premios fueron ganados por chatbots que implementaban el funcionamiento básico de ELIZA. Sin embargo los años subsiguientes aparecieron otros tipos de chatbots más sofisticados que empezaron a ocupar los primeros puestos. Una tecnología que apareció para el desarrollo de chatbots más complejos fue AIML y en particular una implementación de chatbot con dicha tecnología: ALICE salió vencedor del certamen en tres oportunidades. [Loebner, 2009]

Alguno de los chatbots que han participado en el certamen se detallan debajo:

Nombre	PC THERAPIST
Descripción	Primer programa en ganar el Premio Loebner. Este chatbot fue escrito en 1986 y está basado en ELIZA
Creador	Joseph Weintraub

Tabla 2.1.3.1: Detalles PC Therapist

Nombre	Julia
Descripción	chatbot basado en Eliza, participó en el concurso Loebner del año 1994, salió cuarta de cinco <i>chatbots</i> .
Disponible	http://www.lazytd.com/lti/julia/

Tabla 2.1.3.2: Detalles Julia

Nombre	Brian
Descripción	Este <i>chatbot</i> está escrito en C++, extiende la idea general del "terapeuta" que utiliza ELIZA. Ganó el tercer lugar en el concurso Loebner 1998
Disponible	http://www.strout.net/info/science/ai/brian/

Tabla 2.1.3.3: Detalles Brian

Nombre	ALICE
Descripción	ALICE está desarrollado sobre AIML y fue ganador del premio Loebner en tres oportunidades.
Disponible	http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1

Tabla 2.1.3.4: Detalles ALICE

Nombre	Eugene Goostman
Descripción	Segundo en el concurso Loebner de 2005.
Disponible	http://www.mangoost.com/bot/

Tabla 2.1.3.5: Detalles Eugene Goostman

Nombre	Jabberwacky
Descripción	Jabberwacky fue ganador del premio Loebner en dos oportunidades.
Disponible	http://www.jabberwacky.com/

Tabla 2.1.3.6: Detalles Jabberwacky

Nombre	Ultra Hal
Descripción	Este chatbot ganó el primer lugar en el concurso Loebner del año 2007.
Disponible	http://www.zabaware.com/webhal/index.html

Tabla 2.1.3.7: Detalles Ultra Hal

Las transcripciones de las conversaciones entre los jurados y los chatbots están todas disponibles de forma *on-line*. Además algunos de los chatbots están disponibles en internet para que cualquier persona pueda interactuar con ellos, sin embargo solo un par están disponibles para que cualquiera pueda observar su funcionamiento, de los últimos ganadores del certamen, ALICE es uno de estos.

2.1.4 Implementación y utilización comercial de chatbots

Si bien los chatbots aún tienen demasiados problemas para que sean adoptados como una

tecnología sólida hay casos puntuales en donde su utilización puede considerarse exitosa. Como ejemplos se pueden nombrar ciertos chatbots creados para el mensajero instantáneo de Microsoft (Messenger o MSN). Una empresa llamada IMI desarrolló un intérprete de AIML en C# para que sea posible crear en este lenguaje chatbots para Microsoft Messenger y ya se dispone de una lista de chatbots existentes para MSN, incluyendo algunos como Robin o Wilma creados para un propósito específico distinto del mero interés académico.

Robin es un chatbot creado por el Ministerio de Sanidad y Consumo de España. Su función es informar a los jóvenes a través de Messenger sobre enfermedades de transmisión sexual y consumo de alcohol.

Microsoft creó su propio chatbot, Encarta Instant Answer, para promocionar su enciclopedia encarta, sin embargo con la finalización de dicho producto por parte de Microsoft el chatbot fue dado de baja.

Algunos sistemas informáticos como Remedy, utilizan chatbots para que los usuarios puedan evacuar consultas comunes de manera rápida, estos chatbots simulan un operario de *Help Desk*.

2.2 Procesamiento de Lenguaje Natural

La presente sección describe la utilización de sistemas inteligentes en relación con el procesamiento de lenguaje natural como se anticipó.

2.2.1 Pragmática

Si bien una cadena de palabras puede tener una o varias interpretaciones semánticas, estas interpretaciones pueden ser incompletas si no se añade la información dependiente del contexto sobre la situación actual de cada interpretación posible.

La pragmática o pragmalingüística es un subcampo de la lingüística que se interesa por el modo en que el contexto influye en la interpretación del significado. El contexto debe entenderse como situación, ya que puede incluir cualquier aspecto extralingüístico: situación comunicativa, conocimiento compartido por los hablantes, relaciones interpersonales, etc. La pragmática toma en consideración los factores extralingüísticos que condicionan el uso del lenguaje, esto es, todos aquellos factores a los que no se hace referencia en un estudio puramente formal.

Como en una conversación vía *chat*, dos entidades que se comunican no comparten más que sus palabras, todo el contexto queda limitado a eso, a lo que se dijo anteriormente. A pesar de la reducción notable de lo que habría que evaluar para deducir el contexto, el problema sigue

siendo de una complejidad enorme.

2.2.2 Resolución de referentes

Según [Russell, Norving 2003] : "La necesidad más obvia de información pragmática es en la resolución del significado de referentes, que son frases que hacen referencia directamente a la situación actual. Por ejemplo en, en la frase «yo estoy en Boston hoy», la interpretación de los referentes «yo» y «hoy» depende de quién declare la frase.[...] El oyente que percibe un acto de habla debería también percibir qué hablante es y usar esta información para resolver el referente." Es decir que el oyente debería de poder discernir quien es "yo" y cuando es "hoy".

"La resolución por referencia es la interpretación de un pronombre o una frase nominativa definitiva que hace referencia a un objeto en el mundo (En lingüística, mencionar algo que ya ha sido introducido se llama referencia anafórica. Referenciar algo que todavía tiene que ser introducido se llama referencia catafórica.) La resolución se basa en el conocimiento del mundo en las partes previas al discurso. Consideremos el pasaje:

«Juan paró al camarero. Él pidió un sándwich de jamón.»

Para entender que «él» en la segunda oración se refiere a Juan, necesitamos haber entendido que en la primera oración menciona dos personas y que Juan está ejerciendo el papel de cliente y por ello es capaz de pedir mientras que el camarero no lo es. Usualmente, la resolución de referencias es un problema en donde se debe seleccionar una referencia de una lista de candidatas." [Russell, Norving 2003]

Han sido diseñados una gran cantidad de algoritmos de resolución por referencia, dos de los más conocidos son, el algoritmo *pronoun references* [Hobbs 1978] y *Pronominal Anaphora Resolution* [Lappin, Leass 1994]

2.3 Problemática actual

En la actualidad según las publicaciones anuales de los resultados del certamen: *Loebner Prize* (<http://www.loebner.net/Prizetf/loebner-prize.html>) no hay diálogos entre chatbots y personas que logren pasar de forma satisfactoria el *test de Turing*, tampoco de que dichas conversaciones se asemejen a una conversación humana o incluso coherente. Hay varios problemas a superar, pero sin duda uno de los más evidentes es la interpretación pragmática de la conversación. Los chatbots no pueden seguir el hilo de una conversación por mucho tiempo (dando respuestas que satisfagan a su interlocutor) ya que no logran resolver las referencias anafóricas ya sean de tipo

correferencial, de sentido o elípticas:

Sin embargo logran dar respuestas suficientemente satisfactorias a preguntas o frases armadas, en donde no hay ninguna referencia anafórica, como por ejemplo: "¿Qué edad tienes tú?" o "Mi nombre es Juan". Si bien en el primer ejemplo hay un único referente: "tú" este referencia directamente al chatbot y no es un referente que necesita ser buscado en el dialogo, como sería el caso de "¿Qué edad tiene él?". Este tipo de frases son mencionadas a lo largo de este trabajo como frases: "independientes del contexto" en oposición a las otras que son mencionadas como frases: "dependientes del contexto"

La mitad de la problemática reside en poder distinguir cuando una frase es "independiente del contexto" y cuando es "dependiente del contexto", la otra mitad es como transformar las frases "dependientes del contexto" en frases "independientes del contexto".

2.4 Tecnología a utilizar

Se presentan en esta sección las tres tecnologías principales que serán combinadas para lograr mejorar la interpretación pragmática del dialogo en un chatbot. La primera es: redes neuronales artificiales; más específicamente la red *back propagation* o de retropropagación, que se explica en el punto 2.4.1.1 y 2.4.1.2 y la red SOM (Kohonen) o auto organizada que se explica en el punto 2.4.1.3; la segunda tecnología es el algoritmo *Pronominal Anaphora Resolution* que se explica en el punto 2.4.2, por último se describirá el lenguaje AIML y su mecánica que es la base sobre la cual está construido el chatbot: ALICE que es el punto de partida del presente trabajo.

2.4.1 Redes neuronales artificiales

"Las redes neuronales artificiales (RNA) imitan la estructura física del sistema nervioso, con la intención de construir sistemas de procesamiento de la información paralelos, distribuidos y adaptativos, que puedan presentar un cierto comportamiento «inteligente»" [Martín del Brío, Sanz Molina 2001]

Para ampliar la definición anterior se debe añadir que están compuestas por pequeñas unidades llamadas neuronas artificiales, las cuales pueden recibir información de entrada, en general valores numéricos dentro de un rango (o valores de tensión para implementaciones de *hardware*), la cual pueden modificar y luego devolver como salida. Estas unidades pueden conectarse entre sí uniendo las entradas de unas a las salidas de otras formando verdaderas redes, incluso estas interconexiones, llamadas sinapsis, pueden ser ponderadas, por un valor

llamado peso que modifica el valor de salida de una neurona antes de que ingrese en otra.

Las neuronas que reciben los datos iniciales conforman lo que se conoce como: "capa de entrada" mientras que las que devuelven el último valor sin que este lo recoja ninguna otra neurona conforman la "capa de salida". Entre ambas capas pueden haber otras más, llamadas capas ocultas.

Estos conceptos pueden ser definidos de una manera mucho más rigurosa desde un punto de vista matemático. A continuación se da la definición de [Müller 1990]:

Una red neuronal artificial es un grafo dirigido, con las siguientes propiedades:

- 1) A cada nodo i se asocia una variable de estado x_i .
- 2) A cada conexión (i,j) de los nodos i y j se asocia un peso $w_{ij} \in \mathbb{R}$
- 3) A cada nodo i se asocia un umbral θ_i .
- 4) Para cada nodo i se define una función $f_i(x_j, w_{ij}, \theta_i)$, que depende de los pesos de sus conexiones, del umbral y de los estados de los nodos j a él conectados. Esta función proporciona el nuevo estado del nodo.

Donde los nodos son las neuronas y las conexiones son las sinapsis en la terminología más común. Y según la definición anterior:

neurona de entrada: neurona sin sinapsis entrantes.

neurona de salida: neurona sin sinapsis salientes

neurona oculta: toda neurona que no es de entrada ni de salida.

2.4.1.1 Perceptrón multicapa

"Un perceptrón multicapa (*MLP: Multi-Layer Perceptron*) es un perceptrón simple con capas ocultas añadidas. Esta arquitectura suele entrenarse mediante el algoritmo denominado retropropagación de errores o *Back Propagation* o bien haciendo uso de alguna de sus variantes o derivados, motivo por el que en muchas ocasiones el conjunto arquitectura MLP + Aprendizaje con *Back Propagation* suele denominarse red de retropropagación o simplemente *Back Propagation*." [Martín del Brío, Sanz Molina 2001]

El perceptrón simple es un modelo neuronal unidireccional, compuesto por dos capas de neuronas, una de entrada y otra de salida (Figura 2.1). La operación de una red de este tipo, con n neuronas de entrada y m de salida, se puede expresar como:

$$y_i(t) = f \left(\sum_{j=1}^n w_{ij} x_j - \theta_i \right), \quad \forall i, 1 \leq i \leq m \quad (2.1)$$

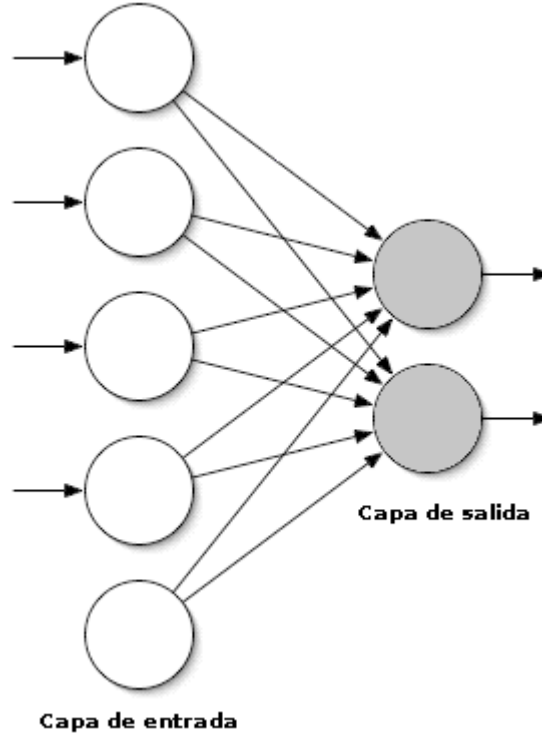


Figura 2.1: Perceptrón simple

Las neuronas de entrada no realizan ningún computo, únicamente envían la información a las neuronas de salida. La función de activación de las neuronas de la capa de salida podría ser por ejemplo la función escalón o bien la función *sigmoidal* que son dos de las más utilizadas.

Para poder expresar la función matemática de la forma de operatoria de un *MLP* con una capa oculta y neuronas de salida lineal, como el que se muestra en la Figura 2.2 denominaremos x_i a las entradas de la red, y_i a las salidas de la capa oculta y z_k a las de la capa final (y globales de la red); t_k serán las salida objetivo (*target*). Por otro lado, w_{ij} son los pesos de la capa oculta y θ_j sus umbrales, w'_{kj} los pesos de la capa de salida y θ'_k sus umbrales. Entonces tenemos una expresión como la siguiente:

$$Z_k = \sum_j w'_{kj} y_j - \theta'_k = \sum_j w'_{kj} f \left(\sum_i w_{ji} x_i - \theta_j \right) - \theta'_k \quad (2.2)$$

Siendo $f()$ de tipo *sigmoidal* (Figura 2.3) como por ejemplo:

$$f(x) = 1 / (1 + e^{-x}) \quad (2.3)$$

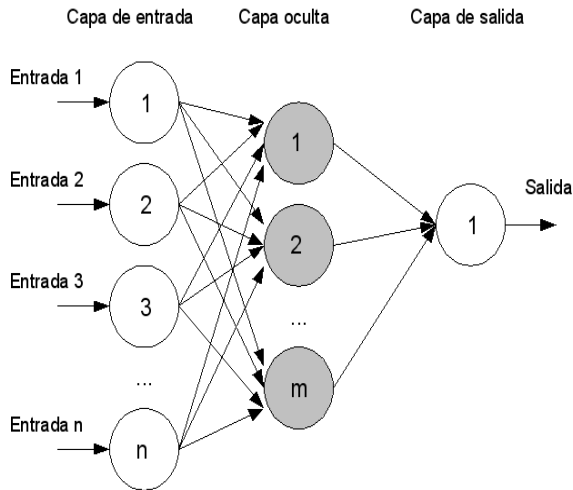


Figura 2.2: MLP

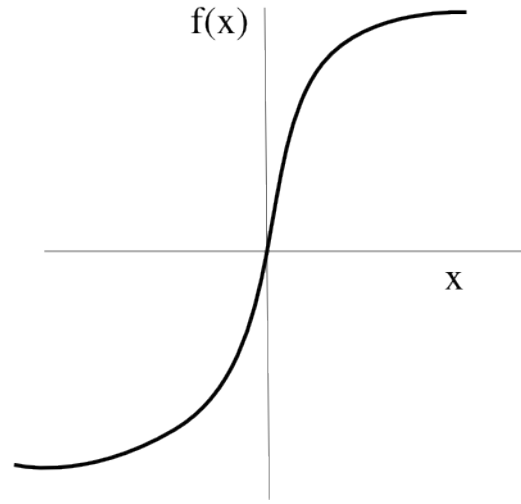


Figura 2.3: función sigmoideal

2.4.1.1.1 Perceptrón multicapa como aproximador universal de funciones

En 1989 Funahashi [Funahashi 1989] demostró que un *MLP* de una única capa oculta constituía un aproximador universal de funciones (aunque hubo otras demostraciones más o menos simultáneas.)

Teorema de [Funahashi 1989]: Sea $f(x)$ una función no constante, acotada y monótona creciente. Sea \mathbf{K} un subconjunto compacto (acotado y cerrado) de \mathbb{R}^n . Sea un número real $\varepsilon \in \mathbb{R}$, (error) y sea un entero $k \in \mathbb{Z}$, tal que $k \geq 3$, que fijamos (cantidad de capas). En estas condiciones se tiene que:

Cualquier *mapping* $\mathbf{g}:\mathbf{x} \in \mathbf{K} \rightarrow (g_1(\mathbf{x}), g_2(\mathbf{x}), \dots, g_m(\mathbf{x})) \in \mathbb{R}^m$, con $g_i(\mathbf{x})$ sumables en K , puede ser aproximado en el sentido de la topología L_2 en K por el *mapping* entrada-salida representado por una red neuronal unidireccional (*MLP*) de k capas ($k-2$ ocultas), con $f(x)$ como función de transferencia de las neuronas ocultas, y funciones lineales para las de las capas de entrada y de salida. En otras palabras:

$\forall \varepsilon > 0, \exists$ un *MLP* de las características anteriores que implementa el *mapping*:

$$\mathbf{g}':\mathbf{x} \in \mathbf{K} \rightarrow (g'_1(\mathbf{x}), g'_2(\mathbf{x}), \dots, g'_m(\mathbf{x})) \in \mathbb{R}^m \quad (2.4)$$

de manera que:

$$d_{L2(k)}(\mathbf{g}, \mathbf{g}') = \left(\sum_{i=1}^m \int \|g_i(\mathbf{x}_1, \dots, \mathbf{x}_n) - g'_i(\mathbf{x}_1, \dots, \mathbf{x}_n)\| dx \right)^{1/2} < \varepsilon \quad (2.5)$$

En resumen un *MLP* de una única capa oculta (podría ser cualquier cantidad de estas) puede aproximar hasta el nivel deseado cualquier función continua en un intervalo, por lo tanto,

las redes neuronales multicapa unidireccionales son aproximadores universales de funciones.

2.4.1.2 Back Propagation

El concepto de propagar hacia atrás los errores, durante el proceso de actualización de los pesos sinápticos da lugar al nombre del algoritmo. En primer lugar se calcula la expresión denominada: "señal de error" la cual es proporcional al error de salida actual de la red. Con este valor se calcula la actualización de los pesos sinápticos de la capa de salida. A continuación se propagan hacia atrás los errores (señal de error) a través de las sinapsis de la capa oculta; con esta se calcula la actualización de los pesos sinápticos de las capas ocultas. Puede realizarse con cualquier cantidad de capas ocultas.

Sea un *MLP* de tres capas cuya arquitectura se presenta en la Figura 2.4. Y sea un patrón de entrada $X^\mu = (\mu=1,...,p)$:

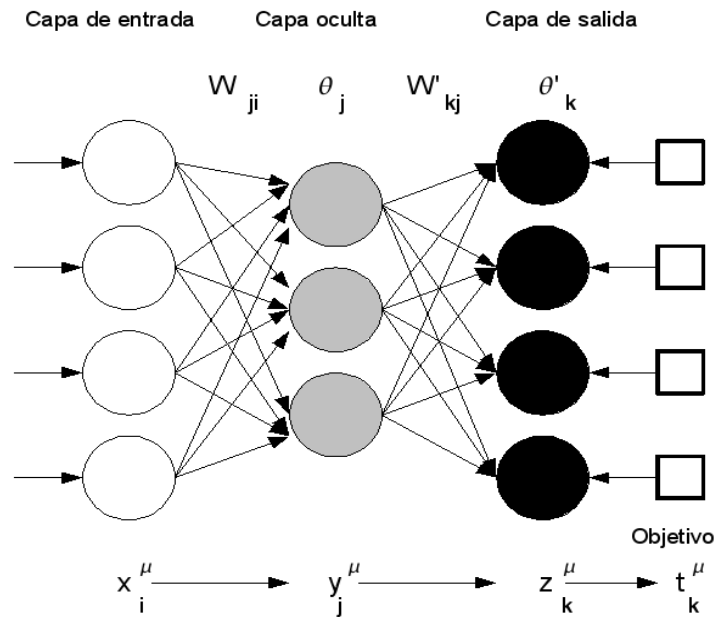


Figura 2.4: MLP y Back Propagation

La operación global de esta arquitectura se expresa del siguiente modo:

$$Z_k^\mu = \sum_j w'_{kj} y_j^\mu - \theta'_k = \sum_j w'_{kj} f\left(\sum_i w_{ji} x_i^\mu - \theta_j\right) - \theta'_k \quad (2.6)$$

las funciones de activación de las neuronas ocultas $f(h)$ son de tipo sigmoideal, con h el potencial postsináptico o local. La función "coste" de la que se parte es el error cuadrático medio:

$$E(w_{ji}, \theta_j, w'_{kj}, \theta'_j) = (1/2) \sum_{\mu} \sum_k [t_k^{\mu} - f(\sum_j w'_{kj} - y_j^{\mu} - \theta'_k)]^2 \quad (2.7)$$

La minimización se lleva a cabo mediante el descenso por el gradiente, como en este ejemplo hay una capa oculta, se tendrá un gradiente respecto de los pesos de la capa de salida y otro respecto de los de la capa oculta. Las expresiones son las siguientes:

$$\delta w'_{kj} = -\varepsilon \frac{\partial E}{\partial w'_{kj}} \delta w'_{ji} = -\varepsilon \frac{\partial E}{\partial w'_{ji}} \quad (2.8)$$

Las expresiones de actualización de los pesos se obtienen sólo con derivar teniendo en cuenta las dependencias funcionales y aplicando adecuadamente la regla de la cadena:

$$\delta w'_{kj} = \varepsilon \sum_{\mu} \Delta_k^{\mu} y_j^{\mu}, \text{ con } \Delta_k^{\mu} = [t_k^{\mu} - f(v_k^{\mu})] \frac{\partial f(v_k^{\mu})}{\partial v_k^{\mu}} \quad (2.9)$$

$$\delta w'_{ji} = \varepsilon \sum_{\mu} \Delta_j^{\mu} x_i^{\mu}, \text{ con } \Delta_j^{\mu} = (\sum_k \Delta_k^{\mu} w'_{kj}) \frac{\partial f(v_j^{\mu})}{\partial v_j^{\mu}} \quad (2.10)$$

La actualización de los umbrales (*bias*) se realiza haciendo uso de estas mismas expresiones, considerando que el umbral es un caso particular de peso sináptico cuya entrada es una constante igual a -1. [Martín del Brío, Sanz Molina 2001]

De forma esquemática sus pasos son los siguientes:

1. Inicializar los pesos y los umbrales iniciales de cada neurona. Hay varias posibilidades de inicialización siendo las más comunes las que introducen valores aleatorios pequeños.
2. Para cada patrón del conjunto de los datos de entrenamiento:
 - a. Obtener la respuesta de la red ante ese patrón. Esta parte se consigue propagando la entrada hacia adelante, ya que este tipo de red (*MLP*) es *feedforward*. Las salidas de una capa sirven como entrada a las neuronas de la capa siguiente, procesándolas de acuerdo a la regla de propagación y la función de activación correspondientes.
 - b. Calcular los errores asociados según las ecuaciones 2.9 y 2.10
 - c. Calcular los incrementos parciales (sumandos de las sumatorias). Estos incrementos dependen de los errores calculados en 2.b

3. Calcular el incremento total, para todos los patrones, de los pesos y los umbrales según las expresiones en las ecuaciones 2.9 y 2.10
4. Actualizar pesos y umbrales
5. Calcular el error actual y volver al paso 2 si no es satisfactorio.

2.4.1.3 Redes SOM, Kohonen

Se hará una breve introducción a este tipo de red ya que finalmente no fue utilizada en la implementación de la solución. Aunque sí fueron utilizadas durante las fases de prueba.

En 1982 T. Kohonen presentó un modelo de red neuronal con capacidad para formar mapas de características de manera similar a como ocurre en el cerebro. Este modelo tiene dos variantes denominadas *LCQ (Learning vector quantization)* y *TPM (Topologu Preserving Map)* o *SOM (Self Organizing Map.)* Ambas se basan en el principio de formación de mapas topológicos para establecer características comunes entre informaciones (vectores) de entrada de la red, aunque difieren en las dimensiones de estos, siendo de una sola dimensión en el caso de *LVQ* y bidimensional e incluso tridimensional en la red *TPM*. [García Martínez, Servente, Pasquini 2003]

La arquitectura de dicha red puede ser descrita como de dos capas: con N neuronas de entrada y M se salida (Figura 2.5)

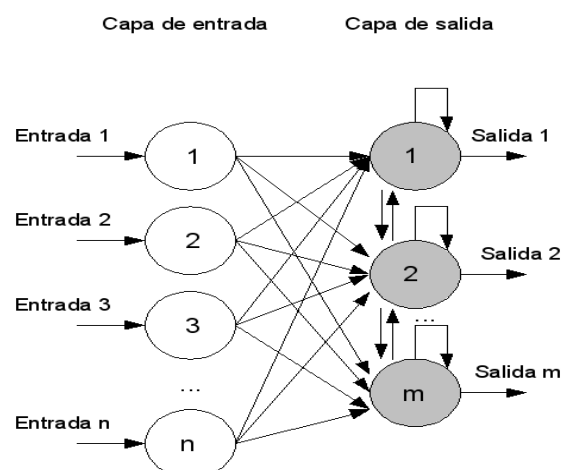


Figura 2.5: Estructura de una red de Kohonen

Cada una de las N neuronas de entrada se conecta a las M de salida a través de conexiones hacia adelante (*feedforward*). Entre las neuronas de la capa de salida, puede decirse

que existen conexiones laterales de inhibición (peso negativo) implícitas, pues aunque no estén conectadas, cada una de estas neuronas va a tener cierta influencia sobre sus vecinas.

La influencia que cada neurona ejerce sobre las demás es función de la distancia entre ellas, siendo muy pequeñas cuando están muy alejadas. Es frecuente que dicha influencia tenga la forma de que se muestra en la Figura 2.6.

La versión del modelo denominada *TPM* trata de establecer una correspondencia entre los datos de entrada y un espacio bidimensional de salida, creando mapas topológicos de dos dimensiones, de tal forma que ante datos de entrada con características comunes se deben activar neuronas situadas en zonas próximas de la capa de salida:

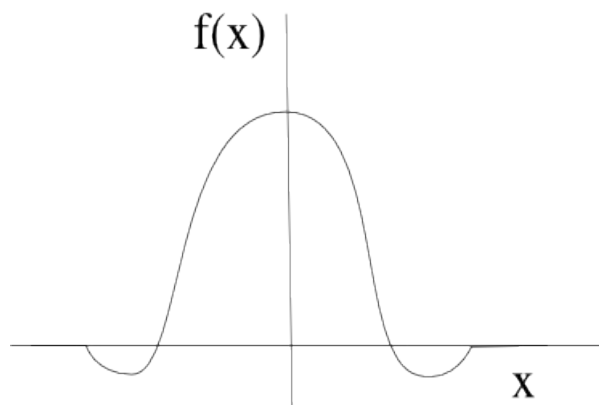


Figura 2.6: función de interacción lateral entre neuronas

2.4.1.3.1 Funcionamiento de una red de Kohonen

Dado un vector de entrada E_k , tal que $E_k = (e_1, e_2, e_3, \dots, e_n)$, para una red como la descrita anteriormente con N neuronas de entrada y M de salida; la salida general de una neurona de salida j será:

$$s_j(t+1) = f\left(\sum_{i=1}^N w_{ij} e_i + \sum_{p=1}^M Int_{pj} s_p(t)\right) \quad (2.11)$$

Donde Int_{pj} es una función como la de la Figura 2.6 que representa la influencia lateral de la neurona p sobre la neurona j . La función de activación de las neuronas de salida $f(x)$ será del tipo continuo, lineal o sigmoideal, ya que esta red trabaja con valores reales.

La red revoluciona hasta alcanzar un estado estable en el que solo hay una neurona activada, la ganadora. La formulación matemática del funcionamiento de esta red puede simplificarse así [Hilera, Martínez 1994]:

$$s_j = \begin{cases} 1 & \text{MIN } || E_k - W_j || = \text{MIN} \left(\sum_{i=1}^N (e_i - w_{ij})^2 \right)^{1/2} \\ 0 & \text{resto} \end{cases}$$

Donde $|| E_k - W_j ||$ es una medida de la diferencia entre el vector de entrada y el vector de pesos de las conexiones que llegan a la neurona j desde la entrada. Es en estos pesos donde se registran los datos almacenados por la red durante el aprendizaje. Durante el funcionamiento, lo que se pretende es encontrar el dato aprendido más parecido al de entrada para averiguar qué neurona se activará y en que zona del espacio bidimensional de salida se encuentra. [García Martínez, Servente, Pasquini 2003]

2.4.2 Algoritmo Resolution of Anaphora Procedure

RAP (Resolution of Anaphora Procedure) es un algoritmo para identificar los sintagmas nominales (*NP: Noun Phrase*) antecedentes de pronombres en tercera persona y anáforas léxicas (reflexivas y reciprocas). El algoritmo se aplica a las propiedades sintácticas de una oración generadas con algún *parser* de procesamiento de lenguaje natural.

El sintagma es un tipo de constituyente sintáctico (palabra o secuencia de palabras que funcionan en conjunto como una unidad dentro de la estructura jerárquica de la oración) formado por un grupo de palabras que forman otros sub-constituyentes, al menos uno de los cuales es un núcleo sintáctico. Las propiedades combinatorias de un sintagma se derivan de las propiedades de su núcleo sintáctico. Por su parte el núcleo sintáctico es la palabra que da sus características básicas a un sintagma y es por tanto el constituyente más importante o de mayor jerarquía que se encuentra en su interior.

RAP contiene los siguientes componentes principales [Lappin, Leass 1994]:

- Un filtro sintáctico *intraoracional* para descartar la dependencia anafórica de un pronombre en una *NP* por razones sintácticas
- Un filtro morfológico para descartar la dependencia anafórica de un pronombre en una *NP* por falta de coincidencias con persona, número o género.
- Un procedimiento de identificación de un pleonismo
- Un algoritmo de vinculación de anáforas para la identificación de posibles antecedentes asignados de una anáfora léxica dentro de la misma sentencia.
- Un procedimiento para la asignación de valores a varios parámetros de relevancia

(función gramatical, paralelismo entre funciones gramaticales, frecuencia de mención, proximidad) de un *NP*. Este procedimiento emplea una jerarquía de funciones gramaticales según

la cual se asignan pesos más relevantes a:

- (i) sujeto sobre no sujeto
 - (ii) los objetos directos más otros complementos,
 - (lii) los argumentos de un verbo por sobre complementos y objetos de frases preposicionales (*PP: Prepositional Phrase*) que actúan como complemento del verbo (Si la frase preposicional está en el predicado, es complemento del verbo)
 - (iv) los núcleos sustantivos por sobre los complementos de los núcleos sustantivos
- Un procedimiento para identificar NPs vinculados anafóricamente como clases equivalentes para las cuales el valor global prominente es la suma de los valores de sus elementos
 - Un procedimiento de decisión para seleccionar el elemento preferido de una lista de antecedente candidatos para un pronombre.

2.4.3 AIML

AIML (*Artificial Intelligence Mark-up Language*) permite a las personas ingresar conocimiento en chatbots basado en la tecnología de software libre ALICE.

AIML fue desarrollado por la comunidad Alicebot de software libre y el Dr. Richard Wallace durante los años 1995-2000. Fue adaptado de una gramática que no era XML aunque también se llamaba AIML y formó las bases para el primer Alicebot, A.L.I.C.E., (*Artificial Linguistic Internet Computer Entity*).

AIML describe una clase de objetos de datos llamados objetos AIML y (parcialmente) el comportamiento de los programas de computadora que procesan dichos objetos. Los objetos AIML están formados por unidades llamadas: "topic" y "category", que contienen datos analizados los cuales componen los elementos AIML que encapsulan conocimiento de la forma estímulo-respuesta. [Wallace 2003]

La unidad básica de conocimiento en AIML es una "category" como se mencionó. Cada *category* se compone de una pregunta de entrada, una respuesta de salida y un contexto opcional. La pregunta, o estímulo, se llama: "pattern". La respuesta se llama: "template". Los dos tipos de contexto opcionales se denominan: "that" y "topic". El lenguaje de los patrones (*patterns*) en AIML es simple, consiste solamente en palabras, espacios y los símbolos *wildcard* "_" (guión bajo) y "*" (asterisco). Que representan tanto una palabra, como una serie de palabras o bien nada, la única diferencia entre ellos es la prioridad. Las palabras pueden contener letras y números pero no otros caracteres. El lenguaje de los patrones es *case insensitive*. Las palabras están separadas por un espacio único.

Uno de los aspectos más importantes de AIML es que implementa recursividad con el operador: "srai". Como menciona el Dr. Wallace en [Wallace 2003] este operador tiene muchos usos:

- Reducción simbólica: Reduce las formas gramaticales complejas a otras más simples.
- Dividir sentencias largas en dos o más subdivisiones y combinar las respuestas de cada una.
- Sinónimos: Muchas entradas diferentes se pueden vincular a una única respuesta.
- Ortografía: Se puede utilizar para vincular una entrada mal escrita hacia la entrada correcta.
- Detección de palabras clave en cualquier lugar de la entrada.
- Condicionales: Ciertas formas de ramificación pueden ser implementadas con "srai"
- Cualquier combinación de las anteriores.

AIML funciona sobre XML, por lo que la base de datos de conocimientos de un chatbot implementado con AIML no es otra cosa que uno o más archivos XML con varios *tags* de tipo "category" como el que se muestra a continuación:

```
<category>
  <pattern>ARE YOU RELATED TO ALICE *</pattern>
  <template>
    Alice <person/> has been an influence on me.
  </template>
</category>
```

Si un chatbot AIML tuviese esta *category* implementada y un usuario ingresase una frase que comenzase con: "Are you related to Alice", el chatbot respondería con la frase: "Alice <person/> has been an influence on me." Donde el *tag* "<person/>" sería cambiado por el contenido de la frase original que coincidió con el *wildcard*; por ejemplo si la frase de entrada hubiese sido: "Are you related to Alice Cooper", la respuesta sería: "Alice Cooper has been an influence on me."

2.4.3.1 Manejo del contexto en AIML

Como se mostró en el punto anterior AIML funciona como una gran tabla de dos columnas: patrones de coincidencia (*patterns*) y salidas (*templates*); en donde para una frase de entrada dada se busca el mejor patrón de coincidencia y se devuelve la respuesta asociada a dicho patrón. Esta salida podría ser, si está al operador "srai", una nueva frase de entrada, en cuyo caso se devolvería la nueva salida asociada.

Sin embargo AIML implementa tres mecanismos para manejar el contexto, o la interpretación pragmática estos son:

- El *tag* "that" : que contiene una frase que podría haber sido dicha por el chatbot en la línea anterior del dialogo.
- Las variables de sesión, variables que pueden ser completadas dinámicamente durante una conversación para ser utilizadas luego.
- El *tag* "topic": que contiene un tema sobre el cual se podría estar hablando en un dialogo entre el chatbot y el usuario

El uso del tag *that* será ilustrado con el siguiente dialogo de ejemplo:

chatbot: Do you like movies?

usuario: yes

chatbot: I like movies too.

Queda claro que la frase de entrada: "yes" no basta para dar la respuesta satisfactoria del ejemplo, por lo que AIML no buscará simplemente un *pattern* que coincida con "yes" sino que además buscará que el *tag* "that" coincida con "Do you like movies?". De esta forma se pueden tener muchas *categories con patterns* "yes" útiles para diferentes circunstancias.

Se muestra a continuación un ejemplo del uso de las variables de sesión, las cuales se *setean* en algunas *categories*, tomese una *category* como la siguiente:

```
<category>
    <pattern>MY NAME IS MIKE</pattern>
    <template>
        Hi <set name="name">Mike</set>, I know someone else named Mike too.
    </template>
</category>
```

Si el usuario ingresó una frase como: "My name is Mike", además de responder con el contenido del *tag* "template" el chatbot *seteará* una variable de sesión llamada: "name" con el nombre: "Mike". Más tarde podrá usar el contenido de está variable para armar una respuesta a otra entrada diferente.

El *tag topic* funciona en parte como una variable más de sesión, para alguna *category* puede *setearse* la variable de sesión: "topic" para indicar que se está hablando de un tema en particular, por ejemplo:

```
<category>
```

```
<pattern>yes</pattern>
<that>Do you like movies?<that>
<template>
  I like movies too.
</template>
<think>
  <set name="topic">movies</set>
</think>
</category>
```

Luego a medida que avanza la conversación, el intérprete AIML podrá verificar además de los *tags pattern* y *that* un tercero, que está por sobre *category* y es el *tag topic*. Al igual que el *tag that* no es un *tag* obligatorio pero si existe las *categories* dentro de él tendrán mayor prioridad que las otras.

2.4.3.2 ALICE

ALICE es el chatbot más exitoso de todos los implementados con la tecnología AIML. En 2003 contaba con una base de datos de más 40 000 *categories*. Fue presentado en el certamen *The Loebner Prize for artificial intelligence* (mencionado en el punto: 2.1.3) y ganó en tres oportunidades: 2000, 2001 y 2004 [Loebener 2009]. Y en otras llegó a ser finalista.

ALICE corre sobre "Program D" un intérprete de AIML creado en 2001 y antes de eso corría sobre otro intérprete llamado "Program B"; al momento de escribir este informe ALICE se encuentra de forma *online* para que cualquier persona pueda interactuar con dicho chatbot en la siguiente dirección: "<http://alice.pandorabots.com/>" y la totalidad de los archivos AIML que componen su base de datos, también llamada "cerebro de ALICE" se encuentra disponible de forma libre y gratuita.

3. Definición del problema

En este capítulo se describe el problema abordado en esta tesis comenzando con una descripción de la problemática actual (sección 3.1) la cual se reducirá a ciertos casos puntuales y bien definidos que comprenderán el alcance de este trabajo (sección 3.2); finalmente en la sección 3.3 se muestran ejemplos de casos reales obtenidos de las transcripciones de conversaciones entre personas y chatbots del concurso Loebner.

3.1 Problemática actual

Uno de los problemas que enfrentan los chatbots actualmente es la interpretación pragmática del discurso. La mayoría de los chatbots que, solo pueden ser analizados a través de las respuestas que dan, muestran un comportamiento que parece obviar toda información, o buena parte de esta, del dialogo excepto la frase de entrada inmediata. Aún restringiendo la situación comunicativa al conocimiento compartido por los hablantes en el dialogo puntual que se está llevando a cabo. En el caso de AIML, una de las tecnologías actualmente más solidas para la construcción de este tipo de sistemas, el chatbot responde a una frase de entrada según con que patrón de su base de datos de conocimiento coincidió. En algunos casos toma en consideración una variable llamada "topic", ver punto 2.4.3.1 (Manejo del contexto en AIML), para determinar que el patrón de coincidencia pertenezca al mismo tópico de la conversación y en otros verifica también la respuesta dada anteriormente (uso del *tag* "that".) Sin embargo como muestra el análisis hecho a partir de las transcripciones de las conversaciones del certamen Loebner, estos mecanismos no son suficientes para una completa evaluación del contexto.

Esta falta de "lectura" del contexto, de no considerar en cada nueva respuesta a todo el dialogo como un *input* y solo tomar como tal a la última frase, lleva al chatbot a cometer un *error* al generar la respuesta (se considerará error a toda respuesta a una frase de entrada del usuario que no satisface las expectativas de este de igual forma en que lo haría la respuesta de un interlocutor humano en condiciones similares como en el caso de un *test* de Turing) Dichos errores pueden ser provocados por alguno de los problemas que se listan a continuación:

Resolución de referentes: La resolución del significado de referentes o de anáforas se da cuando se quiere interpretar un pronombre que aparece en la oración y hace referencia a un objeto previamente introducido en el discurso. Ya que por ejemplo el chatbot, podría contar en su base de datos con una respuesta satisfactoria para la frase: "Who is Richard Wallace?", sin

embargo si un usuario ingresa: "Who is him?" y "him" hiciese referencia a "Richard Wallace" el chatbot no daría una respuesta satisfactoria.

Ambigüedad Léxica: Este problema se da cuando una palabra o frase tiene más de un significado posible y este queda revelado solo en la evaluación del contexto de la conversación, por ejemplo la frase: "The batter hit the ball" parece tener una interpretación no ambigua en la cual un jugador de baseball golpea la bola, pero obtendremos una interpretación diferente si la frase previa es: "the mad scientist unleashed a tidal wave of cake mix towards the ballroom." (En la primer oración "batter" parecería significar: "bateador", "hit": "golpeó" y "ball": pelota, sin embargo al anteponer la segunda oración "batter" significa "masa", "hit": "alcanzó" y "ball": "baile")

Elipsis: La elipsis en lingüística se refiere a ciertas construcciones sintácticas en las que no aparece alguna palabra que se refiera a una entidad lógica necesaria para el sentido de la frase. Por ejemplo en la frase: "Yo corría por la vereda y ellos por la calle". Se omite el verbo "correr" anterior a "por la calle".

Incluso hay casos aún más complejos dentro de la problemática de los chatbots como el de la sentencia: "I couldn't open the door" en donde la oración está completa (no falta en principio ninguna palabra) y está correctamente formada pero no contiene mucha información. En el caso de AIML una sentencia de entrada como esa podría coincidir con un patrón con *wildcards* y devolver una respuesta genérica relacionada con abrir puertas. Sin embargo si la sentencia anterior fuese: "I got home and", se estaba dando más información y si pudieran ser combinadas ambas sentencias para buscar un patrón, se lograría una respuesta más acorde. Suponiendo, por supuesto, que haya un patrón de coincidencia en AIML para una frase similar a: "I couldn't open [my] home door", por ejemplo. La solución parcial a este problema en AIML es el uso de la variable "topic", esta debería de *seteas* en "home" cuando el usuario ingrese la primer frase, y luego tendría que existir un patrón de coincidencia para: "I couldn't open the door" dentro de dicho *topic*.

Referencias inexistentes: Este caso es el inverso al primero, el usuario hace referencia a algo o alguien que nunca fue introducido en el discurso y el chatbot responde como si "supiese" de quien o de que se está hablando. Por ejemplo, si un usuario comienza una conversación con "Do you like her?", el chatbot probablemente dará una respuesta del tipo: "yes, I like." que podría pasar desapercibida en la mayoría de los casos pero no en este escenario; No respondería como lo haría un ser humano quien inmediatamente preguntaría por la referencia insatisfecha del pronombre: *her*. Ejemplo: "Who is her?" o "Who are you talking about?"

Respuestas a interrogaciones cerradas: En muchos casos el chatbot hace preguntas al usuario y luego de que este responde, el chatbot no logra generar una nueva respuesta acorde a las expectativas de su interlocutor. Muchos de esos casos se dan con interrogaciones cerradas, preguntas que pueden ser respondidas con sí o con no ("yes" o "no".) Por ejemplo, si el chatbot emite como "respuesta" la siguiente pregunta cerrada: "Do you like movies?" y el usuario simplemente ingresa como respuesta: "yes" o bien "no", el chatbot buscará a continuación (es el caso de AIML) un patrón que coincida con "yes" o con "no", según sea el caso y terminará dando una respuesta genérica sin relación con el tema de conversación: "movies" en este caso. En cambio si el usuario hubiese ingresado una respuesta como: "yes, I like movies", el chatbot buscaría un patrón de coincidencia con dicha sentencia y encontraría una respuesta más acorde. AIML intenta solucionar este problema de forma más o menos satisfactoria con el *tag* "that", sin embargo la base de datos de ALICE no tiene entradas para todos los casos de preguntas cerradas posibles dentro del conjunto de preguntas que puede formular el propio chatbot.

3.2 Alcance propuesto

Los problemas mencionados en el punto anterior, no conforman la totalidad de problemas devenidos de la falta de interpretación pragmática pero son los más evidentes. Si se compara la forma en la cual trabaja el cerebro humano al dialogar, ya sea interpretando oraciones que recibe de su interlocutor o generando las respuestas, con la forma en la cual funcionan los chatbots quedarían en evidencia problemas más profundos y difíciles de resolver, pero eso escapa al objetivo de esta tesis.

El trabajo se centrará en tres problemas básicos, de los mencionados arriba, relacionados con la interpretación pragmática: resolución de referentes, referencias inexistentes y respuestas a interrogantes cerrados.

Para el caso de la resolución de referentes se utilizará el algoritmo: *RAP (Resolution of Anaphora Procedure)* de Lappin-Leass. Dicho algoritmo ha dado buenos resultados, 74 % de tasa de éxito en la resolución de anáforas inter-sentencias y 89 % de tasa de éxito en la resolución de anáforas intra-sentencias [Lappin, Leass 1994]

Para el caso de resolución de referencias inexistentes, se tomarán los casos en donde el algoritmo anterior no haya devuelto referencias y se formulará una pregunta que indague sobre la referencia insatisfecha.

Finalmente se implementará un algoritmo sencillo de reemplazos, para convertir las preguntas en afirmaciones para los casos de respuestas monosilábicas.

Con las soluciones planteadas, que se desarrollarán en el punto siguiente (4), se espera tener un éxito moderado, sabiendo que no solucionarán en su totalidad dichos problemas, se pretende reducir la tasa de respuestas erróneas cada vez que el chatbot se encuentre con alguno de los problemas mencionados.

La implementación de las soluciones propuestas plantea un nuevo problema, que es la conservación de los casos favorables, es decir aquellos casos "pregunta-respuesta" que funcionan correctamente en la actualidad y no necesitan ningún tipo de reformulación ni cambio. La manera de solucionar este nuevo problema será clasificar los casos en favorables y no favorables mediante una red.

3.3 Ejemplos de casos reales

Los siguientes ejemplos fueron obtenidos de las transcripciones del certamen Loebner 2009:

http://loebner.net/Prizef/2009_Contest/loebner-prize-2LL009.html, 2006:

http://loebner.net/Prizef/2006_Contest/loebner-prize-2006.html y 2004:

http://loebner.net/Prizef/2004_Contest/loebner-prize-2004.html

Ejemplo 1)

A continuación se muestra un problema de ambigüedad léxica y el error en la respuesta del chatbot presentado por: David Levy, llamado: "Do-Much-More" quien resultó vencedor en el certamen del año 2009. (Extracto de la conversación el *Juez 2*)

chatbot: I don't suppose cars will burn coal, but I wouldn't now for sure. Everything about cars is confidential.

human: Are you experiments confidential?

chatbot: Yes I am experiments, but don't worry. Experiments don't bite!

En este ejemplo se muestra como el chatbot "interpretó" de una forma diferente la sentencia de la persona. Probablemente porque internamente la frase de entrada del usuario hizo coincidencia con un patrón con *wildcards* (como sería en AIML o ELIZA) "Are you * ", donde la mayoría de las veces "Are you" significa "Eres tu.." y no "Son tus.."

Ejemplo 2)

A continuación se muestra un problema de elipsis y el error en la respuesta del chatbot

presentado por: Rollo Carpenter, llamado: "Joan" quien resultó vencedor en el certamen del año 2006.

human: Do you read books?

chatbot: Sure I read books.

human: Who is your favourite author?

chatbot: I don't know. There are so many. Who's your favourite author?

human: I like Philip Roth

chatbot: Are they scary?

Aquí el chatbot da una respuesta no satisfactoria a la frase del usuario "I like Philip Roth", no considera la frase : "my author favorite is Philip Roth".

Ejemplo 3)

A continuación se muestran tres problemas juntos: de resolución de referentes, de referencias inexistentes y elipsis; en el dialogo entre del chatbot presentado por: Rollo Carpenter, llamado: "Joan" quien resultó vencedor en el certamen del año 2006, y el Juez 3 del certamen.

chatbot: I am familiar with her writings.

human: Really like what?

chatbot: It used to be in a smallish building. It is now in a part of a larger building. Someday they hope to build a new library

human: Is it safe?

chatbot: Probably.

human: Is it safe?

chatbot: Do you think it's safe?

human: Is it safe?

chatbot: Of course.

human: Is it safe?

chatbot: Yes.

human: Are you sure?

chatbot: It's easy to be sure about something like that when you're 26 years old!

El dialogo que se presenta arriba presenta demasiadas dificultades juntas para un chatbot, aunque resulte trivial para un ser humano. Primero se tiene la interrogación del usuario: "Is it safe?" la cual presenta una referencia anafórica, aunque no está muy claro a que hace referencia el pronombre "it" podría ser "larger building", en dicho caso la frase quedaría como: "Is [the] larger building safe ?". Ante dicha interrogación, el chatbot responde de forma genérica: "Probably".

Luego el usuario vuelve a preguntar nuevamente por tres veces: "Is it safe?", con lo cual

queda claro de que la referencia del pronombre "it" no existe, cualquier persona respondería ante la misma situación con un interrogante del estilo: "What do you mean with that question?"

Por último el usuario pregunta: "Are you sure?", aquí hay una elipsis ya que está implícito en la oración "it is safe", si se reformulase la sentencia para eliminar la elipsis la frase quedaría como: "Are you sure that it is safe?". Sin embargo el chatbot devuelve una respuesta genérica que solo hacer referencia a la palabra "sure"

Ejemplo 4)

A continuación se muestra un problema de resolución de referentes y el error en la respuesta del chatbot presentado por: Richard Wallace, llamado: "ALICE" quien resultó vencedor en el certamen del año 2004. (Extracto de la conversación el *Juez 2*)

human: visiting Dr Loebner, but I live in New Jersey

chatbot: I have no idea where that is. Tell me a little about your life in visiting Dr
Loebner but you live in New Jersey.

human: it's right across the river from here

chatbot: Do you experience any effects of global warming there?

human: oh sure, all the time

En la frase del usuario: "it's right across the river from here.", el pronombre "it" está haciendo referencia a "Dr. Loebner('s house)", si la frase se reformulara para eliminar la anáfora quedaría como: "Dr Loebner is right across the river from here". Al tener acceso al código y a la base de datos de ALICE es fácil ver que el patrón de coincidencia utilizado fue: "IT IS * HERE", (siendo * un *wildcard*), la respuesta dada por el chatbot fue una respuesta genérica.

En este caso en particular se presenta otro problema de la lengua y del procesamiento del lengua natural en general, no necesariamente relacionado con la interpretación pragmática; la metonimia, La metonimia es una figura del habla en la cual se emplea un objeto para sustituir otro objeto. Aquí "Dr Loebner" en verdad es una figura que se usa en "reemplazo" de "la casa del Dr Loebner".

Ejemplo 5)

A continuación se muestra un problema de resolución de referentes y el error en la respuesta del chatbot presentado por: Richard Wallace, llamado: "ALICE" quien resultó vencedor en el certamen del año 2004. (Extracto de la conversación el *Juez 1*)

human: What is your favorite operating system?

chatbot: Linux is the best operating system.

human: What do you like about it?

chatbot: it? What do I like about it?

human: What do you like about Linux?

chatbot: The special effects are the best.

En este ejemplo, queda claro para cualquier persona que en la frase: "What do you like about it?", "it" es una referencia a "Linux". El chatbot no puede resolver esta anáfora y responde a la pregunta textual: "What do you like about it?", el usuario elimina la anáfora y vuelve a hacer la pregunta: "What do you like about Linux?" en este caso el chatbot da una respuesta un poco más satisfactoria, aunque no del todo.

El patrón de coincidencia que encuentra AIML para la última frase de entrada del usuario es: "WHAT DO YOU LIKE ABOUT *", con lo que la respuesta es bastante genérica. Sería igual para: "What do you like about cookies?" por ejemplo.

4. Solución propuesta

En el presente capítulo se describe la solución propuesta a los problemas planteados en el punto anterior; En el punto 4.1 se esboza el aspecto general de la solución y la forma en que cada una de las tecnologías propuestas se ajustaría a un requerimiento dado. En el punto 4.2 se analizan las diferencias entre las líneas de dialogo: usuario-chatbot consideradas correctas o coherentes (casos favorables) y aquellas consideradas incorrectas (casos desfavorables) que será de mucha importancia para entender el punto 4.3, en donde se describe como se utilizará una RNA para clasificar los casos. En el punto 4.4 se explica la necesidad de utilizar una base de datos relacional en lugar de archivos XML para gestionar la base de datos de conocimiento de ALICE. Por último los puntos 4.5, 4.6 y 4.7 explican los tres mecanismos propuestos para mejorar la forma de responder de un chatbot basado en AIML teniendo en cuenta la interpretación pragmática.

4.1 Esbozo general de la solución

La premisa general de la tesis es mejorar un chatbot, o una tecnología de diseño de chatbots, que esté actualmente en el *estado del arte (o de la cuestión)* y no crear un nuevo chatbot desde cero. Para ello se utilizará como punto de partida el *lenguaje* AIML y la base de datos de conocimiento de ALICE, ambos libres y disponibles al público general.

El primer paso para encarar la solución consiste en separar los casos favorables de los desfavorables (ver punto siguiente) de forma que para todas aquellas frases de entrada para las cuáles ALICE ha dado buenos resultados no haya modificaciones y para todas aquellas frases para las cuáles ALICE da respuestas no satisfactorias, aún cuando estas coinciden con algún patrón de la base de datos de conocimientos, se activen mecanismos que revisen la información de contexto, dada solamente por todas las líneas de dialogo anteriores, y la utilicen para convertir el caso desfavorable en un caso favorable.

Para realizar la clasificación se utilizará una red neuronal artificial de tipo *Back Propagation*, luego para convertir los casos desfavorables en casos favorables se utilizarán tres mecanismos diferentes, el primero y más importante es un algoritmo de resolución de referentes anafóricos (*RAP*, descrito en 2.42), el segundo es un procedimiento simple para reformular preguntas cerradas como sentencias declarativas, ejemplo: "Do you like movies?" será reformulada como "I like movies" si el usuario hubiese ingresado "yes" como respuesta. El tercer mecanismo es un procedimiento que se activará cuando los demás hayan fallado, se trata de un

procedimiento que busca referentes insatisfechos y pregunta por su significado. Por ejemplo si el algoritmo de resolución de referentes anafóricos, falló al tratar de identificar el pronombre "her", en la oración: "Do you like her?" este procedimiento hará que el chatbot devuelva una pregunta como la siguiente: "Who is her?"

4.2 Distinción entre casos favorables y casos desfavorables

Tomando al chatbot ALICE, como modelo inicial a mejorar, lo primero que se propone hacer es clasificar los casos en favorables y desfavorables. Es decir, para todas aquellas frases de entrada de un usuario, en el conjunto universo de frases posibles, determinar cuáles obtendrán por parte del chatbot, una respuesta satisfactoria (una respuesta que deje al interlocutor satisfecho en igual forma que si hubiese respondido un ser humano en iguales condiciones) y cuáles no.

Planteado en términos absolutos el problema es inabarcable por varias cuestiones a saber:

- 1) Casos limitados en la base de datos de conocimiento de ALICE
- 2) Restricción del idioma (en esta tesis se restringe al idioma inglés)
- 3) Frases inválidas, mal formadas o con errores sintácticos, ortográficos u otros.

4) La imposibilidad de saber de forma objetiva que espera un ser humano al hacer una pregunta. ¿Cómo determinar de forma objetiva que una respuesta es correcta desde el punto de vista de la coherencia de la oración y no de su significado?

Con la intención de acotar el alcance al problema anterior de clasificación se analizaron los casos favorables en los diálogos de ALICE. Se muestran a continuación algunos ejemplos que provienen del certamen Loebner de 2004: http://loebner.net/Prize/2004_Contest/Wallace.html

Caso 1)

Judge 1: Do you like movies?

Alice: Yes I love film, especially science-fiction and comedy.

Caso 2)

Judge 1: What is your favorite movie?

Alice: My favorite movie is Teknolust. Have you seen it?

Caso 3)

Judge 1: Do you have a pet?

Alice: No but I like cats very much.

Caso 4)

Judge 1: Please, call me Ryan.

Alice: Thank you for your request. I am very pleased to meet you Ryan.

Lo que tienen en común estos casos, (como muchos otros casos favorables analizados) es que la frase de entrada, lo que en estos diálogos está representado por lo que dijo "Judge 1", no presentan referencias, ni elipsis, ni ambigüedades léxicas (evidentes) tampoco son frases monosilábicas o respuestas cortas a alguna pregunta anterior.

De forma intuitiva se puede pensar en que este tipo de frases podrían ser las iniciadoras de un dialogo con una persona ya que al iniciar un dialogo hay muy poca información de contexto. Si el dialogo es de forma escrita y los interlocutores están en habitaciones separadas, la información contextual es aún menor. Es decir, una persona "A" podría decirle a otra "B" como primera frase de una conversación:

"Do you like movies?", "What is your favorite movie?" o "Please, call me Ryan" y la persona "B" sería capaz de responder de forma satisfactoria, casi tan satisfactoria como si la misma frase estuviese ubicada a la mitad de un dialogo. No quiere decir que sean frases para iniciar una conversación, por supuesto, ya que un saludo de despedida como "See you soon" también estaría dentro de este conjunto de casos.

Sin embargo lo anterior no es completamente cierto ya que en un contexto demasiado forzado podrían dejar de ser casos favorables para un chatbot AIML, por ejemplo si el usuario dijese antes de "Do you like movies?", "there are new candies called movies" el chatbot daría la misma exacta respuesta: "Yes I love film, especially science-fiction and comedy." Tampoco serían validos en los casos en donde un usuario emplease recursos como la ironía o eufemismos, pero esta clase de problemas escapan en parte a lo que es la interpretación pragmática y al objetivo de este trabajo de estudio.

A pesar de que un contexto forzado podría invalidar los casos, estos serían validos en la mayoría de los diálogos con lo cual la definición de "caso favorable" pasa a ser una definición probabilística que podría definirse como sigue:

El caso "A" pertenece al conjunto de casos favorables si en el X % de los diálogos, cuando un usuario ingresa la frase: "A" obtiene por parte el chatbot una respuesta coherente que satisface sus expectativas tal y como lo haría la respuesta de una persona en circunstancias similares.

Donde X es un número menor que 100, aleatoriamente alto. La definición anterior es una definición practica que es cómoda a los efectos de clasificar frases de entrada como casos favorables para el objeto de estudio de esta tesis, sin embargo la tarea de clasificar casos utilizando la definición anterior es casi imposible debido a que habría que evaluar de forma subjetiva cuando una frase obtuvo una respuesta satisfactoria y cuando no en todos los diálogos

posibles (o bien en algún número suficientemente grande de diálogos.)

Volviendo al ejemplo de cómo se podría clasificar de forma intuitiva un caso; así como la persona "A" al iniciar el dialogo con la persona "B" podría utilizar frases como las anteriores, no podría hacerlo con frases como: "Do you like her?", "What's your favorite?", "yes" o incluso frases como "I'm opened the door" que contienen poca información en el sentido que le da Shannon en [Shannon 1948] y esperar una respuesta igual a la que recibiría si dichas sentencias se hallasen a mitad de un dialogo en donde ya se "sabe" a quien hace referencia el pronombre "her" o que se está hablado de "movies" o que "yes" responde a determinada pregunta o que "door" fue abierta.

De alguna forma, los casos favorables son aquellos en donde la frase de entrada no depende del contexto (en la mayoría de los casos) y los casos desfavorables son aquellos en donde la frase de entrada depende del contexto. Los primeros están compuestos por frases que serán llamadas a lo largo de esta tesis: "independientes del contexto" y los segundos de frases que serán llamadas "dependientes del contexto", aún cuando no exista en la base de datos de conocimiento del chatbot un patrón de coincidencia para la frase. (Se estaría en este caso ante otro problema relacionado con la completitud de la base de datos de conocimiento.)

Los casos serán clasificados según la categoría anterior utilizando una red neuronal artificial.

4.3 Clasificación de los casos usando una RNA

Parte importante de esta tesis es crear una red neuronal artificial que sea capaz de detectar cuando una frase es dependiente o independiente del contexto. La otra parte es la transformación de una frase dependiente del contexto en una frase independiente del contexto, que es aquella, como fue explicado en el punto 4.1, que constituye un caso favorable para un chatbot basado en AIML.

Para la construcción de la RNA se propone crear, primeramente, un conjunto de entrenamiento en donde haya frases en inglés y por cada frase se indique si es o no dependiente del contexto. Las frases en inglés serán transformadas en un mapa que será utilizado como entrada de la RNA.

El mapa se generará no a partir de las frases en sí, sino a partir de los tipos básicos de palabras que la conforman, el orden de estas en la oración, si la oración es una interrogación o no y el tiempo verbal en el cual está formulada la oración. Se hará especial hincapié en los distintos tipos de pronombres, el resto de las palabras serán clasificadas en alguno de los siguientes tipos básicos: sustantivo(*noun*), verbo (*verb*), adjetivo (*adjective*), adverbio (*adverb*), artículos y palabras

auxiliares (*aux*)

El tipo de arquitectura de red neuronal escogido es el de *Back Propagation*, descrito en el punto 2.4.1 debido a que entre los tipos de redes con aprendizaje supervisado esta es una de las más exitosas, utilizada en casos de reconocimiento de texto, de voz, clasificación de patrones, etc.

Durante la fase de entrenamiento se creará una red con una capa oculta ya que de esta forma se pueden clasificar elementos encerrados en regiones polinomiales convexas del espacio y otra con dos capas ocultas que permitirá clasificar elementos encerrados en conjuntos de formas arbitrarias sobre el espacio dado ya que no conocemos la distribución espacial de las frases dependientes e independientes del contexto (según los mapas utilizados). La capa de salida tendrá una única neurona ya que se espera un solo valor *booleano*: verdadero o falso, 1 ó 0 que indicaría si es independiente del contexto o no. Las capas intermedias serán creadas con una cantidad de neuronas igual a la mitad de las utilizadas en la primera capa, al menos en su configuración inicial. La capa de entrada tendrá tantas neuronas como los mapas propuestos necesiten.

Se proponen dos mapas posibles como *input* de la red neuronal, ambos se describen en los puntos siguientes.

4.3.1 mapa 1 propuesto como *input* de la RNA

Sobre la frase original en inglés se realizan 3 procesos diferentes:

1) Se reemplaza cada palabra por su tipo: sustantivo(*noun*), verbo (*verb*), adjetivo (*adjective*), adverbio (*adverb*), artículos y palabras auxiliares (*aux*). (Por palabra se entiende una o más cadenas de caracteres separadas por espacio que juntas tienen un significado univoco distinto del que tienen por separado, ejemplo: "Buenos Aires" es una palabra compuesta por dos cadenas de caracteres). Cada tipo de palabra tiene un valor numérico asociado.

2) Se aplica un procedimiento para detectar si la frase es una interrogación o no. (Aunque no tenga el signo de interrogación)

3) Se aplica un procedimiento que mediante una serie de expresiones regulares detecta el tiempo verbal de la oración.

Una vez hecho lo anterior se crea un vector de números en punto flotante de 15 posiciones que será finalmente el *input* de la RNA. Se instancian todas las posiciones en cero.

Luego en la posición 0 del vector se marca si la frase es una interrogación con un uno (1) o con un cero (0) sino. La posición 1 del vector indica el tiempo verbal, el cual puede tomar uno de los siguientes valores: 0.0; 0.5; 0.7; -1.0; -1.3; 0.8; 0.6; -1.4; -1.5; 1.1; 1.2; 1.3; 1.5; 1.9 para los

tiempos verbales respectivos (en inglés): *infinitive, simple present, present continuous, past, past continuous, present perfect, present perfect continuous, past perfect, past perfect continuous, simple future, future continuous, future perfect, future perfect continuous, conditional*. El resto de las posiciones del vector representan en orden, a los diferentes tipos de palabras que conforman la oración (las primeras 13 palabras), según su valor numérico. Los tipos de palabras y sus valores numéricos asociados son:

auxiliary words 0.1
noun 1.9
adjective 1.3
indicative 2.3
interrogative pronouns 2.0
verbs 1.5
personal pronouns first person: 2.5
personal pronouns third person: 3.0
personal pronouns second person: 3.5

El mapa puede ser representado de forma esquemática como se muestra en la Figura 4.1

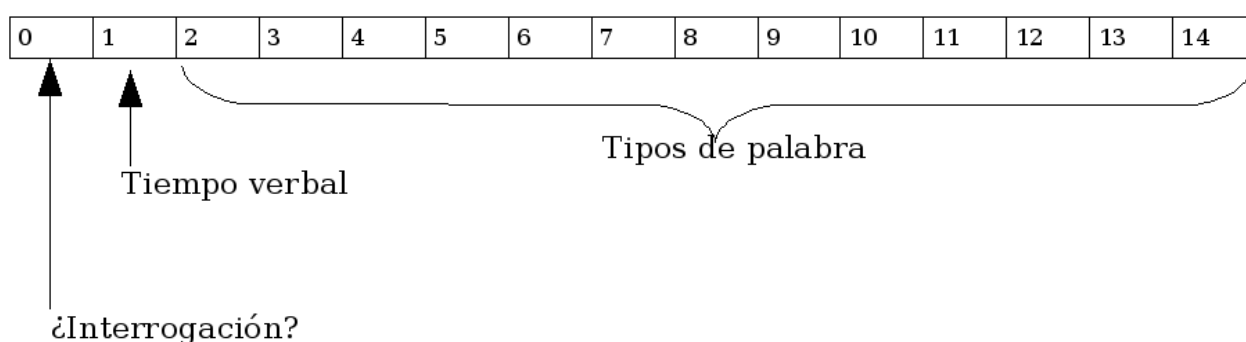


Figura 4.1: mapa 1

4.3.2 mapa 2 propuesto como *input* de la RNA

Para la generación del *input* 2 de la RNA utilizada se cambió el mapa de entrada para hacerlo más sensible. Lo que se hizo fue aumentar el número de neuronas de entrada. Este nuevo mapa reserva una neurona especial para un valor de entrada que indica si la frase es o no una pregunta, al igual que en el caso anterior se realizan las siguientes acciones sobre la frase de entrada:

- 1) Se reemplaza cada palabra por su tipo: sustantivo(*noun*), verbo (*verb*), adjetivo (*adjective*), adverbio (*adverb*), artículos y palabras auxiliares (*aux*).
- 2) Se aplica un procedimiento que detecta si la frase es una interrogación o no.

Pero no se realiza la detección de tiempo verbal, se eliminó la neurona que respondía ante el tiempo verbal encontrado identificado en la oración.

El resto del mapa de entrada se definió como una matriz de 15 x 5 números de punto flotante, en donde la longitud, 15, representa una a una las primeras 15 palabras de la frase de entrada y el ancho o profundidad: 5, indica según la posición que tome un valor distinto de cero, que tipo de palabra es. En donde la posición 0 del vector vertical toma un valor distinto de cero si la palabra es auxiliar o bien si esta es un pronombre interrogativo (*interrogative pronouns*). En el caso de una palabra auxiliar, el valor que toma es de 0.2, en el caso de un pronombre interrogativo es de 3.0 (en este caso se verá reforzado además por la neurona que indica que es una pregunta.) La posición 1 del vector vertical toma valor igual a 1.0 si la palabra es un verbo (*verb*) o bien si es un adverbio (*adverb*). La posición 2 toma un valor igual a 1 si es un adjetivo (*adjective*). La posición 3 toma un valor igual a 1 si es un sustantivo (*noun*) común o igual a 2 si es un sustantivo propio. Por último la posición 4 toma un valor distinto de cero si la palabra es un pronombre personal (*personal pronoun*) o un pronombre demostrativo (*demonstrative pronoun*). En estos casos puede tomar diferentes valores, tomará el valor 1 para pronombres personales en primera persona, 2 para pronombres personales en tercera persona y 3 para pronombres personales en segunda persona. Para pronombres demostrativos toma el valor 2.5.

Esquemáticamente el mapa quedó como se muestra en la Figura 4.2:

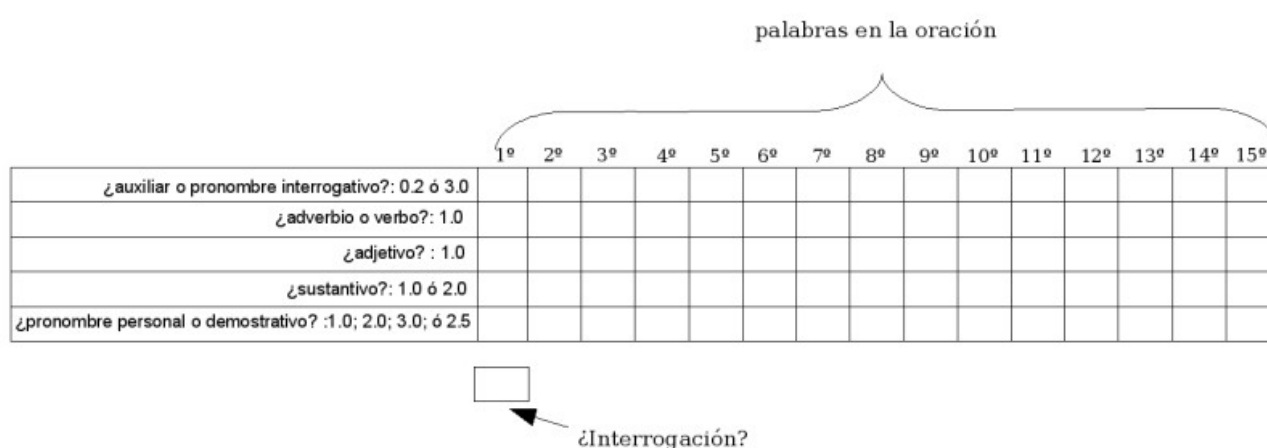


Figura 4.2: mapa 2

4.3.3 Pasos para la clasificación

La clasificación de casos incluye analizar todos los patrones (*patterns*) de cada una de las categorías (*category*) de ALICE para indicar si la frase contenida en el *tag pattern* es o no independiente del contexto.

Como primera medida se clasificarán todas las categorías (*category*) cuyos *patterns* estén conformados por frases que utilicen *wildcards* como dependientes del contexto ya que a priori no se puede saber qué tipo de frase de entrada de usuario coincidirá con un patrón con *wildcards*.

Como segunda medida se clasificarán como independientes del contexto a todas aquellas categorías (*category*) que utilicen el *tag "that"*, ya que si bien el *pattern* en sí es dependiente del contexto, la *category* ya lo tiene contemplado.

Como tercera medida se utilizará la RNA previamente entrenada, para clasificar todas las categorías restantes.

El nuevo campo a agregar en cada *category*, para indicar si es o no un caso favorable se llamará "can_answer" ya que este nombre es más general. (Una *category* con *tag that* es dependiente del contexto, pero puede ser utilizada para responder ya que conforma un caso favorable.)

Una vez que el sistema esté funcional, trabajará inicialmente de forma idéntica a como lo hace ahora AIML, es decir seguirá de forma esquemática los siguientes pasos:

- 1) Esperar una frase de entrada del usuario
- 2) Buscar una *category* cuyo *tag pattern* coincida lo mejor posible con la frase anterior.
- 3) Tomar la frase contenida en el *tag template*.

Pero al llegar a este punto, revisará el nuevo campo "can_answer". Si este campo es uno (1) el sistema procederá de la forma usual, si es cero utilizará la RNA para clasificar la frase de entrada del usuario. Si el resultado es (1): independiente del contexto, procederá de la forma usual. Si es cero (0): dependiente del contexto aplicará los mecanismos antes descritos para convertir la frase del usuario en una frase independiente del contexto.

4.3.4 Utilización de redes neuronales artificiales autoorganizadas (SOM)

La red autoorganizada: *SOM* del modelo propuesto por Kohonen será tomada en cuenta como un clasificador de frases que podría ser utilizada previo al uso del perceptrón. Una forma común de reducir el error cuadrático medio del algoritmo *Back Propagation*, es dividir el conjunto

de pruebas original en subconjuntos donde los elementos agrupados presenten similares características. Una forma de lograr estos subconjuntos es mediante la clasificación realizada a través de redes autoorganizadas.

Los *inputs* que tomaría, eventualmente, esta red serían los mismo propuestos para la red perceptrón.

4.4 Implementación de AIML sobre MySQL

Para simplificar la tarea de análisis y clasificación de los *tags AIML*, la totalidad de la base de datos de conocimiento de ALICE será migrada a una base de datos relacional MySQL.

Durante todo el proceso de pruebas y correcciones se utilizará una implementación de AIML sobre MySQL, la cual reemplazará al algoritmo de *pattern-matching* de los interpretes AIML por el propio *engine* MySQL. Dicha implementación será respaldada por una serie de pruebas que garanticen que la implementación sobre MySQL funcione de igual forma que el intérprete "program-D" utilizado en la actualidad en el sitio de ALICE *online*.

4.5 Resolución de referentes

La resolución de referentes es el mecanismo más importante con el que se cuenta para convertir los casos desfavorables en casos favorables, dicho con la nomenclatura de esta tesis convertir las frases dependientes del contexto en frases independientes del contexto. Para ello se utilizará el algoritmo: *RAP (Resolution of Anaphora Procedure)*. El cual ha dado buenos resultados según [Lappin, Leass 1994]: 74 % de tasa de éxito en la resolución de anáforas inter-sentencias y 89 % de tasa de éxito en la resolución de anáforas intra-sentencias.

Este algoritmo entraría en ejecución luego de que una frase de entrada del usuario haya sido detectada como caso desfavorable y la RNA haya devuelto el valor cero al procesar el mapa asociado a dicha frase. Entonces el algoritmo *RAP* tomaría como *input* las últimas 10 líneas del dialogo y la última frase del usuario. Buscaría en la frase alguna referencia anafórica y trataría de vincularla con algún sustantivo presente en las líneas previas del dialogo. Como salida, el algoritmo devolvería la frase del usuario reformulada como una nueva oración independiente del contexto cuyo significado es el mismo.

De forma ilustrativa se propone el siguiente ejemplo:

Supóngase la siguiente conversación con el chatbot ALICE, utilizando su base de conocimientos y su funcionamiento actual:

Human: Tell me about Richard Wallace

ALICE: Are you asking about my botmaster?

Human: He is a great inventor

ALICE: Maybe you should tell him how you feel about him.

La frase de entrada: "He is a great inventor", coincide con el patrón "HE IS A GOOD %", con lo cual la respuesta dada por el chatbot no es más que una respuesta genérica que no considera a Richard Wallace como sujeto de la oración.

Con los cambios propuestos implementados la frase: "He is a great inventor" sería clasificada como dependiente del contexto por la RNA, y lo es ya que el pronombre "he" es una referencia anafórica por resolver. Se aplicaría luego el algoritmo *RAP*, que utilizaría como *input* las dos líneas anteriores del dialogo y la frase actual para devolver lo siguiente:

"Richard Wallace is a great inventor".

Esta frase sería utilizada como una nueva entrada de usuario por el chatbot la cual coincidiría con otro patrón: "RICHARD WALLACE IS %". Este patrón considera a "Richard Wallace" como sujeto de la oración con lo cual devolvería una respuesta más coherente en relación al dialogo.

4.6 Reformulación de respuestas a preguntas cerradas

Este es otro mecanismo capaz de convertir respuestas dependientes del contexto en respuestas independientes del contexto pero su algoritmia es mucho más sencilla que la del parser *RAP* aunque su alcance es también mucho más limitado.

Fue concebido como una forma sencilla de resolver un problema harto común en los diálogos de los chatbots. En particular se notó que ALICE cuenta en su base de datos de conocimiento con respuestas (más correctamente hablando frases contenidas por el *tag template*) las cuales no son más que preguntas cerradas, como por ejemplo:

"Do you like chocolate ?"

"Do you like computers?"

"Are you a woman ?"

"Are you an english teacher?"

Este tipo de interrogantes suelen ser respondidos mediante un sí (*yes*) o un no (*no*), o bien mediante algún sinónimo más coloquial: "sure", "of course", "no way", etc. En la base de datos de ALICE no hay entradas para cada una de estas posibles respuestas que utilicen el *tag that* para

remarcar la pregunta anterior; y agregarlas, por otro lado, sería poco eficiente ya que implicaría repetir muchas entradas. Sin embargo sí cuenta con patrones como "I am a woman" o "I like chocolate". En base a estas observaciones se pensó en una forma de convertir las frases como: "yes", "no", "sure", etc. en frases independientes del contexto como "I like chocolate" siempre y cuando dichas frases sean respuestas a preguntas cerradas.

De forma ilustrativa se propone el siguiente ejemplo:

Supóngase la siguiente conversación con el chatbot ALICE, utilizando su base de conocimientos y su funcionamiento actual:

Human: I hate cats

ALICE: Do you like dogs?

Human: yes

ALICE: Hmm.

En este ejemplo la frase de entrada "yes" coincidió con el patrón "yes", para el cual hay asignada una respuesta genérica, en este caso "Hmm". Si el cambio propuesto estuviese implementado, la frase de entrada "yes" habría sido detectada como dependiente del contexto por la RNA, luego se hubiese procedido a verificar si la frase anterior del chatbot era una respuesta cerrada; entonces las frases "yes" y "Do you like dogs?" hubiesen sido combinadas y reformuladas como: "I like dogs", una frase independiente del contexto. Dicha sentencia sería entonces utilizada como frase de entrada por el chatbot (en lugar de "yes") y habría coincidido con un patrón idéntico "I LIKE DOGS" cuya respuesta asociada (contenido el *tag template*) es: "Which breed is your favorite? "

4.7 Preguntas por referencias no encontradas

Se pretende construir un subsistema que sirva como *backup* para los casos en los cuales el *RAP* no logre resolver la referencia anafórica. El funcionamiento es simple de explicar, cuando una frase dependiente del contexto que presente un pronombre no logre ser reformulada, este subsistema detectará dicho pronombre y formulará una frase de interrogación que será devuelta al usuario. En dicha respuesta se pedirá la información faltante.

De forma ilustrativa se propone el siguiente ejemplo:

Supóngase la siguiente conversación con el chatbot ALICE, utilizando su base de conocimientos y su funcionamiento actual:

Human: He is a great inventor

ALICE: Maybe you should tell him how you feel about him.

Esta conversación es idéntica a la presentada en el punto 4.4, solo que en este dialogo no se hace referencia a Richard Wallace sino que empieza directamente con la frase: "He is a great inventor". Se puede ver que la respuesta de ALICE tendría sentido si se supiese de quien se está hablando, es decir, a quien hace referencia la palabra "he". Claramente un interlocutor humano no respondería de esa forma sino que preguntaría por la referencia perdida, una respuesta "más humana" sería: "Who are you taking about?"

Suponiendo que estuviesen implementados los cambios propuestos el algoritmo *RAP* se aplicaría sobre la frase: "he is a great inventor" pero no podría encontrar la referencia anafórica de "he" sencillamente porque no existe. Entonces actuaría la "última línea de defensa", que es este subsistema, el cual buscaría la palabra "clave" de una lista de palabras posibles y la utilizaría para formular una pregunta, que podría ser: "Who is he?".

Cabe destacar que este subsistema, o procedimiento según como termine siendo implementado no significa ninguna novedad, AIML mismo o incluso los chatbots basados en ELIZA son capaces de buscar una palabra "clave" en una frase y a partir de ella formular una pregunta. Lo que se presenta aquí como novedoso es la forma y la situación en la cual se lo pretende emplear.

5. Desarrollo de la solución

En el siguiente capítulo se detallan las distintas fases de construcción del sistema, desde la evaluación de la solución hasta su desarrollo y *testing* pasando por los documentos de arquitectura. La metodología de trabajo escogida es: Metrica, partiendo de lo propuesto por dicha metodología se ha elaborado un subconjunto de prácticas que involucran los aspectos más relevantes de las siguientes fases: Estudio de viabilidad del Sistema (5.1) , Análisis del Sistema de Información (5.2), Diseño del Sistema de Información (5.3) y Construcción del sistema de Información (5.4).

5.1 Estudio de viabilidad del sistema

El objetivo de este apartado es el análisis del conjunto concreto de necesidades para proponer una solución a corto plazo, que tenga en cuenta restricciones técnicas y operativas. La solución obtenida como resultado de este estudio debe ser la definición de un proyectos que afecten a un sistemas existentes o bien cree uno nuevos. Se identificarán, para ello los requisitos que se han de satisfacer y se estudiará la situación actual.

5.1.1 Establecimiento del alcance del sistema

En este apartado se estudiará el alcance de la necesidad planteada realizando una descripción general de la misma. Se determinarán los objetivos y se iniciará el estudio de los requisitos. Se analizarán además las posibles restricciones, tanto generales como específicas, que puedan condicionar el estudio y la planificación del trabajo de investigación.

5.1.1.1 Estudio de la solicitud

Se realizará una descripción general de la necesidad planteada y se estudiarán las posibles restricciones de carácter técnico y operativo que puedan afectar al sistema.

5.1.1.1.1 Resumen de la propuesta de tesis

La solicitud en este caso está determinada por la propuesta de tesis, esta propuesta sugiere la implementación de una mejora en los sistemas actuales conocidos como *chatbots*.

Los chatbots son programas (software) que comprenden algoritmos que utilizan procesamiento de lenguaje natural (*NLP: Natural Language Processing*) en un sistema de preguntas y respuestas (*QA systems: question-answering systems*) [Fitrianie, 2002]. Estos sistemas han sido definidos también como sistemas expertos que usan razonamiento basado en casos (*CBR: case base reasoning*) [Wallace, 2003].

Los chatbots actualmente no logran reproducir el comportamiento humano en una conversación ni dar respuestas satisfactorias que puedan ser consideradas coherentes. Solo responden de manera satisfactoria a determinadas frases. Ejemplos de estas conversaciones pueden obtenerse en [Loebner, 2006], [Loebner, 2007] y [Loebner, 2008].

El estudio de las transcripciones de las conversaciones sostenidas entre estos sistemas y un individuo dado muestran diversos problemas, entre los cuales se encuentra la falta de evaluación del contexto (de la conversación) al responder.

La mejora propuesta consiste en modificar un chatbot existente de tal manera que pueda responder a una frase de un individuo teniendo en cuenta información de contexto, es decir lo que fue dicho anteriormente en la misma conversación.

5.1.1.1.2 Descripción del sistema a construir y objetivos

El sistema a construir será un chatbot que se encuentre dentro del *estado del arte* (o de la cuestión) de este tipo de programas, es decir que pueda responder a frases de entrada al menos tan satisfactoriamente como cualquier otro chatbot reconocido. Pero este chatbot además deberá de poder responder correctamente en situaciones en las cuales un chatbot actual no lo hace.

Se tomará como punto de partida para el desarrollo de dicho sistema, un chatbot existente cuyo código fuente y base datos se encuentren disponibles y en lo posible se disponga de documentación acerca de su funcionamiento. Se estudiarán los casos en los cuales el chatbot responde de manera no satisfactoria por no tener en cuenta el contexto. Se evaluarán posibles mejoras para dichos casos y se aplicarán sin modificar el comportamiento para los casos en los cuales el sistema ya funciona adecuadamente.

5.1.1.2 Identificación del alcance del sistema

En la siguiente sección se detallará claramente el alcance propuesto del sistema mediante diagramas de flujos y también sus requisitos.

5.1.1.2.1 Diagramas de flujo

En el diagrama 5.1.1 que se muestra a continuación, está representado, mediante un diagrama de flujo el funcionamiento de un chatbot actual. En el siguiente diagrama 5.1.2 se representa de manera esquemática como afectarían los cambios propuestos a dicho sistema.

A partir de la observación del segundo diagrama se desprende que las modificaciones implican la creación de un sub-sistema capaz de decidir cuando una frase de entrada de un

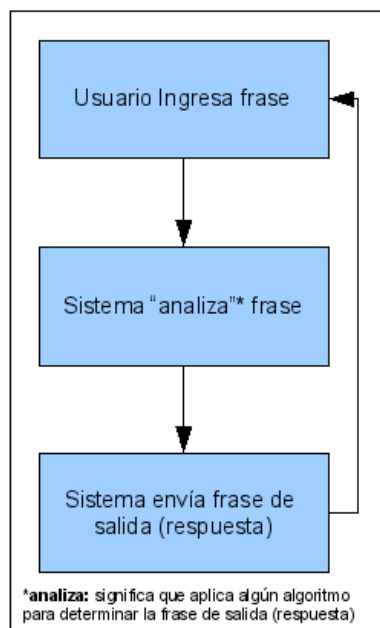


Diagrama 5.1.1: flujo, chatbot actual

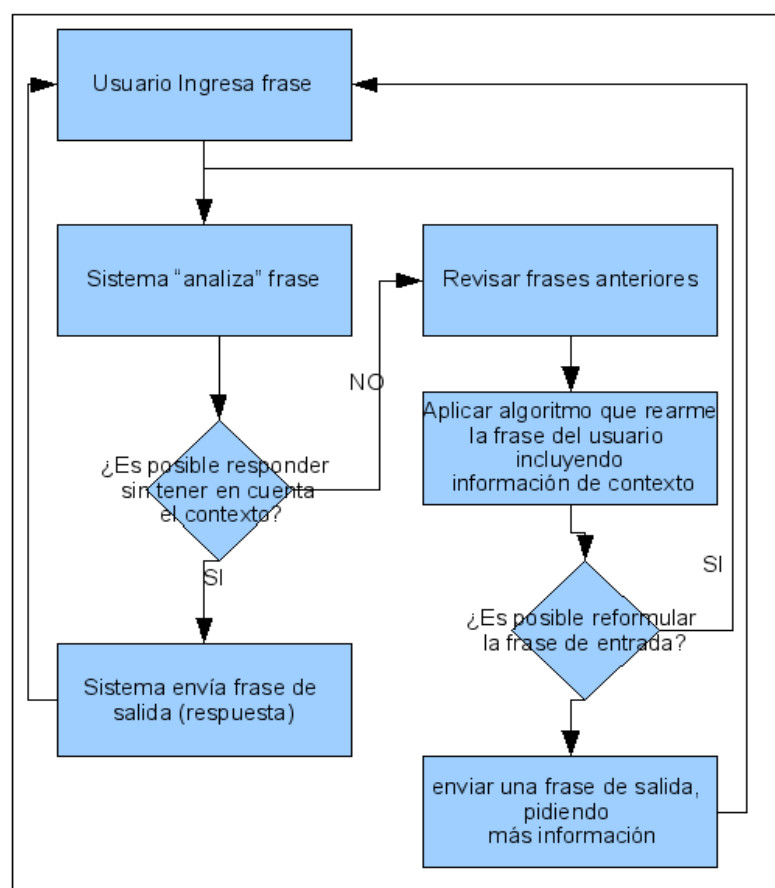


Diagrama 5.1.2: Flujo chatbot modificado

usuario dado puede ser "respondida" correctamente por el chatbot y cuando es necesario buscar información de contexto (en frases anteriores) para responder correctamente. En otras palabras dicho sub-sistema debería de poder determinar si una frase es dependiente o independiente de su contexto. Este sub-sistema estaría actuando también como un filtro que permite dividir el funcionamiento del chatbot en dos, la parte o los casos satisfactorios y los no satisfactorios. A continuación entra en juego un segundo sub-sistema que tomando una frase de entrada que está dentro del conjunto de los casos no satisfactorios le agrega información de contexto que obtiene de frases anteriores, cambiando de esta manera la frase del usuario. Siguiendo el flujo, si el sub-sistema encontró la información faltante y pudo *reformular*, con ella la frase, esta vuelve al flujo principal del chatbot. En este caso el chatbot podrá responder de manera satisfactoria (o no, pero en caso de no hacerlo el problema no debería de ser la evaluación del contexto sino otro, como falta de información en su base de datos.) Para el caso en el cual el sub-sistema anterior no fue capaz de encontrar la información de contexto faltante, o de *reformular* la pregunta se enviará una frase de salida al usuario, pidiendo la información de contexto faltante.

5.1.1.2.2 Requisitos

A partir del análisis anterior es posible identificar una serie de requisitos mínimos, al menos en una primera estimación, necesarios para llevar adelante el desarrollo del sistema propuesto:

- El código fuente y base de datos de un chatbot que esté en el *estado del arte* y que se pueda modificar, el mismo servirá como punto de partida.
- Desarrollar un sub-sistema capaz de decidir si una frase de entrada puede ser respondida de manera correcta por el chatbot anterior o si es necesario información de contexto para ello, es decir que verifique si una frase dada es independiente del contexto o no. (sub-sistema detector)
- Desarrollar un sub-sistema que pueda convertir una frase dependiente del contexto por otra que mantenga su sentido y significado pero sea independiente del contexto. Utilizando para ello frases anteriores de un mismo dialogo. (sub-sistema conversor)

5.1.1.2.3 Usuarios

No hay un tipo de usuario particular que sea capaz de utilizar el sistema, si existe alguna limitación estará dada solo por las limitaciones del chatbot utilizado como base. Por ejemplo, que el usuario final hable determinado idioma. En términos generales no hay un perfil de usuario, cualquier persona que sepa escribir es un potencial usuario del sistema.

5.1.1.3 Especificación del alcance del sistema

A partir de los objetivos y requisitos planteados en los ítems anteriores, es posible elaborar una primera serie de tareas necesarias para cumplir dichos objetivos:

1. Investigación de los algoritmos y/o modelos actuales que permiten dilucidar el contexto en una conversación y estudiar como son aplicados en modelos de chatbots.
2. Selección de un chatbot, comprensión y entendimiento de su funcionamiento.
3. Definición y especificación de mejoras a realizar en el modelo de chatbot escogido para lograr que este adapte su comportamiento a partir de la evaluación del contexto.
4. Construcción e implementación de un sub-sistema capaz de determinar si una frase es independiente del contexto o no. (sub-sistema detector)
5. Construcción e implementación de un sub-sistema capaz de convertir una frase dependiente del contexto en una frase independiente del contexto. (sub-sistema conversor)
6. Implementación del modelo de chatbot con las mejoras propuestas con la integración de los sub-sistemas necesarios.
7. Pruebas de integración, desarrollo de una interfaz de usuario para que cualquier persona pueda interactuar con el nuevo chatbot.
8. Corroboración de los supuestos teóricos, corridas de prueba, comparación con los modelos tradicionales y correcciones necesarias.

Estimación PERT:

Tarea \ Estimación (días)	Optimista	Media	Pesimista	PERT= $O+4M+P / 6$
1: investigación	6	12	20	12,33
2: Selección chatbot	5	7	10	7,17
3: Definición mejoras	10	15	20	15
4: sub-sistema detector	15	17	21	17,33
5: sub-sistema conversor	12	15	25	16,17
6: integración sub-sistemas y chatbot	5	6	7	6
7: Pruebas integración/interfaz	5	6	7	6
8: Pruebas generales/correcciones	4	7	8	6,67

Tabla 5.1.1.3.1: Esfuerzo por tareas

La tabla anterior proporciona una noción aproximada del esfuerzo requerido para la elaboración del sistema final en días/hombres. Pero para saber el tiempo de entrega de cada

tarea, y por ende el tiempo que se demorará en la finalización del sistema es necesario conocer la cantidad de horas por semana que se dedicarán al desarrollo y confección del sistema.

Días por semana: 2 (un total de 16 hs. por semana en promedio.) En la siguiente tabla se muestra el tiempo de entrega estimado.

Tarea	tiempo de entrega en días
1: investigación	40
2: Selección chatbot	24
3: Definición mejoras	50
4: sub-sistema detector	60
5: sub-sistema conversor	55
6: integración sub-sistemas y chatbot	20
7: Pruebas integración/interfaz	20
8: Pruebas generales/correcciones	20
Total:	289

Tabla 5.1.1.3.2: Tiempo de entrega de cada tareas

NOTA: El número de días fue redondeado ya que esta primera estimación es grosera y el tiempo de dedicación semanal puede variar. Colocando en un diagrama de Gantt los números anteriores, para un solo recurso asignado a todas las tareas, obtenemos un gráfico como el siguiente:

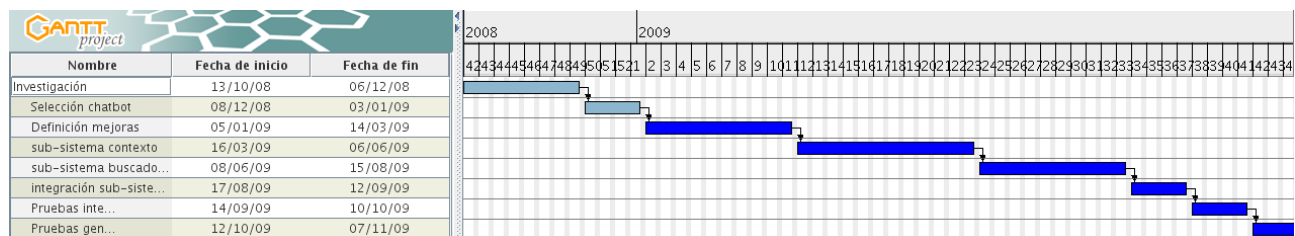


Figura 5.1: Gantt de proyecto

5.1.2 Estudio de la situación actual

En este apartado se hará una descripción detallada de los sistemas actuales sus alcances y limitaciones.

5.1.2.1 Valoración del estudio de la situación actual

En la actualidad existe una cantidad demasiado grande de chatbots diferentes, los cuales

no están catalogados, descriptos ni enumerados en ningún documento confiable y no siempre es posible describir el funcionamiento de estos, porque simplemente no hay documentación disponible ni acceso al su código fuente. De todos los chatbots, solo interesan al objeto de este proyecto, aquellos que estén entre los mejores según algún criterio medianamente objetivo y que además su código fuente y base de datos estén disponibles.

La selección de dichos sistemas será hecha en base a tres criterios:

1. Aquellos chatbots que hayan sido finalistas en el concurso llamado "The Loebner Prize in Artificial Intelligence", el cual es una primera aproximación al Test de Turing [Turing 50]. El concurso es anual y se presentan, en EE.UU., diferentes chatbots que funcionan en inglés; gana el que se aproxime más a un ser humano en su modo de responder.
2. Chatbots o *engines* de chatbots que sean utilizados de manera comercial o productiva de manera exitosa.
3. Aquellos chatbots cuyo código fuente y base de datos estén disponibles para ser examinadas y modificadas.

En base a los requisitos 1 y 2 se confeccionó una lista de chatbots:

Ultra Hal: Este chatbot ganó el primer lugar en el concurso Loebner del año 2007.

disponible: <http://www.zabaware.com/webhal/index.html>

ALICE: ALICE está desarrollado sobre AIML y fue ganador del premio Loebner en tres oportunidades.

disponible: <http://www.pandorabots.com/pandora/talk?botid=f5d922d97e345aa1>

Jabberwacky: Jabberwacky fue ganador del premio Loebner en dos oportunidades.

disponible: <http://www.jabberwacky.com/>

ELIZA: Eliza fue el primer chatbot exitoso construido, extendiendo a Eliza se construyó "PC therapist" un chatbot que fue ganador en los primeros tres años del concurso Loebner. Y luego volvió a ganar en el año 1995.

disponible: http://www-ai.ijs.si/eliza-cgi-bin/eliza_script

JULIA: chatbot basado en Eliza, participó en el concurso Loebner del año 1994, salió cuarta de cinco chatbots.

disponible: <http://www.lazytd.com/lti/julia/>

MITBOLEL: Es un chatbot susceptible de ser adaptado y cambiado (*customizable*)

disponible: <http://www.romahi.com/yazann/Mitbolel/Mitbolel.html>

THOUGHT TREASURE: En este chatbot el enfoque es diferente, ThoughtTreasure es una plataforma que comprende y procesa lenguaje natural, y que puede "razonar con sentido común", según su creador Erik T. Mueller.

disponible: <http://www.signiform.com/tt/htm/tt.htm>

BRIAN: Este chatbot está escrito en C++, extiende la idea general del "terapeuta" que utiliza ELIZA. Ganó el tercer lugar en el concurso Loebner 1998

disponible: <http://www.strout.net/info/science/ai/brian/>

DR ABUSE: Un programa basado en ELIZA que responde en castellano.

disponible: <http://dr-abuse.softonic.com/> (hay que descargarlo)

Eugene Goostman: Segundo en el concurso Loebner de 2005.

disponible: <http://www.mangoost.com/bot/>

Robin: Es un chatbot creado por el Ministerio de Sanidad y Consumo de España.

Su función es informar a los jóvenes a través de Messenger sobre enfermedades de transmisión sexual y consumo de alcohol (utiliza una implementación de AIML en .NET)

disponible como contacto MSN: robin@msc.es

Encarta(R) Instant Answer: Otro chatbot para MSN, diseñado por Microsoft para promocionar Encarta (utiliza una implementación de AIML en .NET),

disponible como contacto MSN: encarta@botmetro.net

Si sobre la lista anterior se aplica el tercer requisito, quedan solo dos chatbots: ALICE, y Eliza. En ambos casos no solo se dispone de acceso al código fuente e información de su base de datos (o base de conocimientos) sino que además hay numerosa documentación sobre ellos, incluyendo la especificación en sus modo de funcionamiento ver [Weizenbaum 1966] y [Wallace 2003]. Además el *engine* de ALICE, AIML, es utilizado en la construcción otros chatbots que tienen fines prácticos reales, como los últimos dos mencionados en la lista anterior.

5.1.2.2 Descripción de los sistemas de información existentes

En esta sección se hará una descripción detallada de los sistemas existentes que se han tomado como casos de estudio. La forma escogida para mostrar su funcionamiento principal es la de un diagrama de flujo. De esta forma se podrá visualizar claramente similitudes y diferencias entre ambos chatbots y porque ALICE puede imitar el funcionamiento básico de ELIZA.

5.1.2.2.1 Diagramas de flujo de ALICE y ELIZA

A continuación se describirá de manera esquemática, mediante diagramas de flujo el funcionamiento de los dos chatbots antes seleccionados:

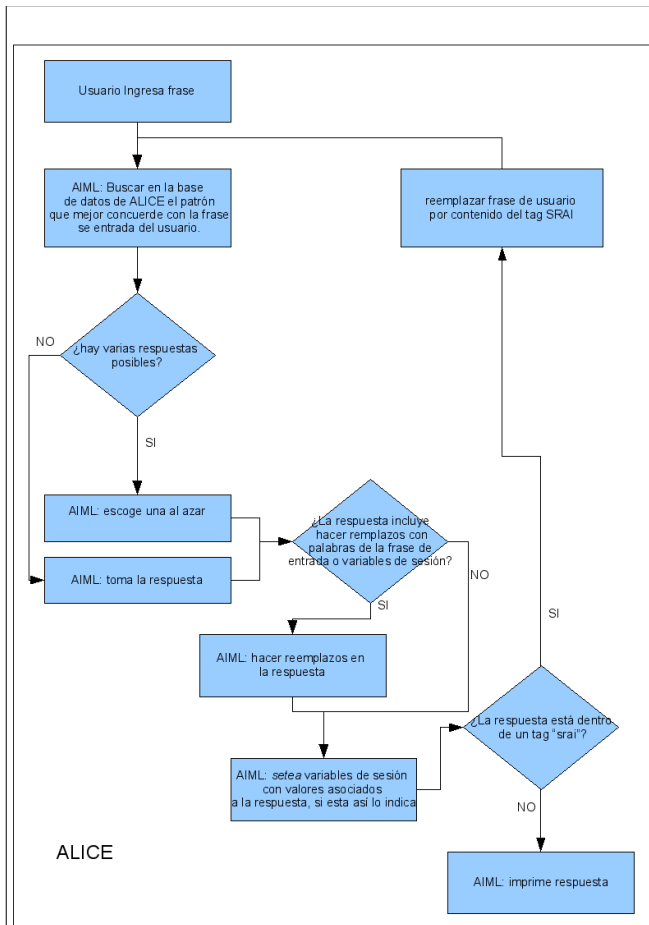


Diagrama 5.1.3: flujo de ALICE

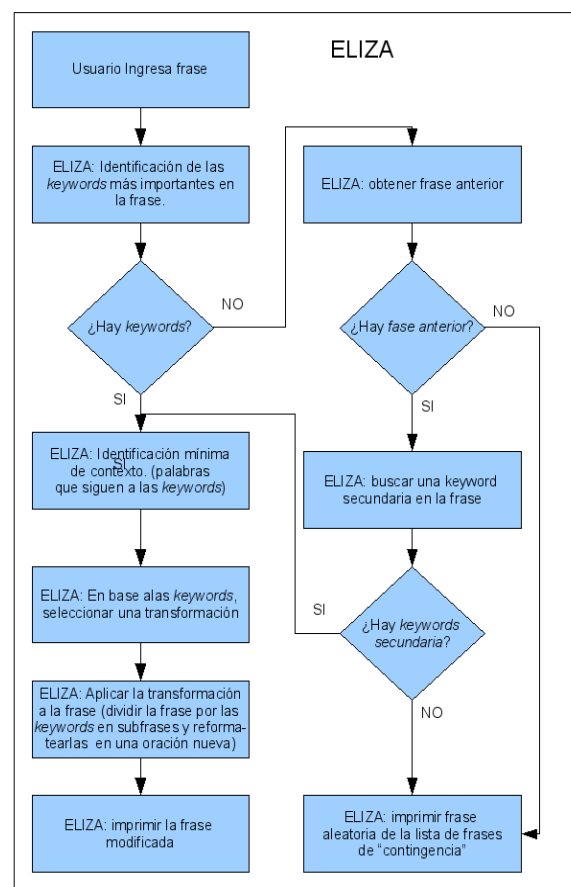


Diagrama 5.1.4: flujo de ELIZA

Los gráficos anteriores describen el comportamiento general de los sistemas pero no dicen como evalúan el contexto, si es lo que hacen, en una conversación.

En primer lugar el contexto de cada palabra individual dentro de una frase o ambigüedad léxica se maneja de manera bastante satisfactoria en ambos sistemas ya que trabajan a nivel de oraciones completas (frases), la mínima unidad que pueden analizar como entrada es una frase. Incluso en el caso particular de que la frase de entrada esté compuesta por una sola palabra esta será tratada como una frase.

En segundo lugar el manejo de la ambigüedad sintáctica y la metonimia que son problemas de referencia interna dentro de una misma frase son manejados particularmente bien en ALICE ya que AIML *mapea*, generalmente, una frase entera a un patrón asociado a la respuesta, no descompone la frase o trata de analizarla. Aunque puede darse el caso, cuando la frase coincide solo parcialmente con el patrón y este en vez de una respuesta tiene asociado una instrucción de recursividad (mediante el tag *srai*.)

En relación a la pragmática, donde el problema reside en completar la información faltante de una frase con información del contexto, Eliza y ALICE funcionan de manera diferente. Ambas pueden dar respuestas diferentes a una misma frase de entrada en función de las frases anteriores intercambiadas entre el chatbot y el usuario.

En el caso de Eliza, cuando no tiene reglas asociadas o no encuentra palabras claves en una frase de entrada, uno de los procedimientos que puede ejecutar consiste en armar una respuesta, no sobre la frase actual, sino sobre alguna frase anterior del usuario distinta a la que se elaboró originalmente. De esta manera simula reencauzar o bien darle continuidad a una conversación como si en efecto la estuviese siguiendo aunque no es una verdadera respuesta en función del contexto ya que lo que logra es crear una conversación nueva.

En el caso de ALICE la funcionalidad para responder teniendo en cuenta el contexto está más y mejor elaborada. ALICE tiene dos maneras, que pueden actuar simultáneamente, para responder en base al contexto. Una es el uso de variables de estado. ALICE asocia a algunas respuestas variables de sesión. Cuando encuentra una respuesta dada, antes de imprimirla *setea* ciertos valores; por ejemplo, la respuesta a una frase de entrada que no tuvo ninguna coincidencia específica podría ser "What's your favorite car?", en este caso ALICE *saetea* la variable de sesión "topic" en "CARS". De este modo si para la próxima frase del usuario hay dos coincidencias iguales, ALICE escogerá la respuesta que esté dentro del tópico "CARS" (Las respuestas pueden estar dentro de "topicos") [Wallace, 2003]. La segunda manera en la cual ALICE puede responder en base al contexto es el *tag* especial de AIML "that", este *tag* está asociado a ciertas respuestas y lo que contiene es la frase textual que ALICE respondió anteriormente. El siguiente ejemplo, ilustrará mejor la función de este *tag*: Si ALICE a cierta frase de entrada de un usuario responde con "Do you like movies?" y el usuario responde a su vez a ello con la frase de entrada "yes". ALICE podría buscar en su base de datos, no solo un patrón que coincida con "yes", sino un patrón que coincida con "yes" y tenga asociado un *tag* "that" que coincida con "Do you like movies?".

Sí bien estas funcionalidades le permiten a ALICE manejar mejor el contexto, no bastan, como demuestran las transcripciones de sus conversaciones (o la de los otros *chatbots* basados en AIML.) El *tag* "that" exige que se duplique exponencialmente la base de datos de ALICE y solo permite contemplar la frase anterior a la actual.

La siguiente observación puede aclarar el porqué se siguen observado problemas en la evaluación del contexto en las conversaciones sostenidas por chatbots basados en AIML: si ALICE tiene en su base de datos de respuestas, hipotéticamente, 100 preguntas susceptibles de ser respondidas solo con "sí" (yes) o "no" (no) tendrá que tener 100 patrones "sí" y 100 patrones "no" asociados a cada pregunta y no se estarían contemplando los casos en los cuales el usuario

decidió explayarse en la respuesta (por ejemplo "I like movies" en lugar de solo "yes" o "I don't like movies" en lugar de solo "no", solo para estos casos tendríamos 200 patrones más en la base.) (No habría problema con los sinónimos de "sí" o "no", es decir si el usuario responde "afirmative" no sería necesario crear una nueva entrada, solo una tabla de sinónimos mediante el *tag* "srai")

5.1.3 Definición de requisitos del sistema

En el punto 5.1.1.2.2 se mostró una lista preliminar de requisitos de sistema que sirvió para entender mejor los puntos que siguieron y definir la situación actual.

A los requisitos anteriores habría que agregarle el siguiente:

- Construcción de algún sistema auxiliar que permita discriminar la información en la base de datos del chatbot: esto es requerido para poder separar y analizar las frases y respuestas del sistema y distinguir los casos favorables, aquellos que deben seguir funcionando igual y los que deben ser modificados. Esta tarea se puede descomponer en tres sub tareas que formarán parte de las tareas 2 a 4 descritas anteriormente.

5.1.4 Estudio de alternativas de solución

A partir de este punto se presentan tres conjuntos de dos alternativas cada uno. Por un lado tomar a Eliza como punto de partida para el sistema que se pretende construir y por el otro lado utilizar a ALICE. Además están los subsistemas que se integrarán al chatbot: el sub-sistema detector y el sub-sistema conversor. Cada uno de estos sub-sistemas puede ser implementado de varias formas diferentes, en este estudio se evaluarán solo dos posibles formas: algorítmica y mediante una red neuronal.

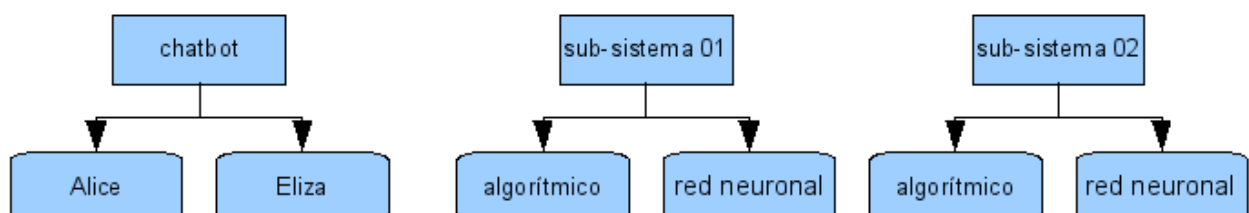


Diagrama 5.1.5: tres opciones

5.1.4.1 Chatbots

En primer lugar se analizará cual de los dos chatbots, se tomará como punto de partida, para ello se utilizará la siguiente tabla comparativa:

ALICE	Eliza
Intenta responder como lo haría una persona, su conversación no define un propósito o ámbito específico.	Intenta responder como lo haría un pisco terapeuta
Ha ganado tres veces el certamen "Loebner", ha sido finaliza otras dos.	Ganó el mismo certamen 4 veces y salió cuarta en otra oportunidad. (en verdad chatbots basados en Eliza, no Eliza propiamente dicha)
ganó por última vez en el 2004	ganó por última vez en 1995
Se utiliza su <i>engine</i> para hacer chatbots con propósitos reales.	Finalidad meramente académica
No posee grandes problemas de ambigüedad léxica, ambigüedad sintáctica y resolución de metonimia	No posee grandes problemas de ambigüedad léxica. Pero ambigüedad sintáctica y resolución de metonimia no están contemplados
Posee dos mecanismos diferentes para evaluar el contexto y emitir en base a ellos una respuesta.	no posee un verdadero mecanismo de evaluación del contexto (ver punto: 5.1.2.2)
AIML puede ser utilizado para construir un chatbot que simule el comportamiento de Eliza	Eliza (o su <i>engine</i>) no puede simular el comportamiento de ALICE

Tabla 5.1.4.1.1: ALICE vs Eliza

5.1.4.2 Sub-sistema detector

El sistema que se encargará de definir si una oración determinada es independiente del contexto o no, deberá poder hacer esta clasificación haciendo manejo de alguna técnica de procesamiento de lenguaje natural.

Desde el punto de vista algorítmico el tratamiento del lenguaje natural según Stuart Russell y Peter Norving en [Russell, Norving 2003] requiere de lo siguiente:

1. Definir un léxico, esto es una "lista de palabras permitidas. Las palabras [del léxico a su vez] se agrupan en categorías o partes del habla conocidas por los usuarios del diccionario: sustantivos, pronombres y nombres para denotar cosas, verbos para denotar sucesos, adjetivos para modificar sustantivos y adverbios para modificar verbos [...]" [Russell, Norving 2003]

2.Una gramática para dicho léxico, "Una gramática es un conjunto finito de reglas que especifican un lenguaje." [Rusell, Norving 2003]. Como los lenguajes naturales al contrario que los lenguajes artificiales no tienen gramáticas formalmente definidas y la cantidad de reglas es muy grande hay que recurrir a la utilización de Gramáticas Aumentadas y otras herramientas como la subcategorización de verbos, etc.

3.Un algoritmo que a partir del léxico y la gramática realice un análisis sintáctico de una frase. Este algoritmo dependiendo de la precisión requerida, puede ser realizado de diversas maneras: ascendente, descendente, utilizando grafos, etc.

4.Finalmente sobre el árbol sintáctico que devuelve la función encargada de realizar el análisis sintáctico, se aplica un nuevo algoritmo encargado de realizar una interpretación semántica, esto es: la extracción del significado de la frase. Sin embargo, tal vez, no es necesario conocer el significado de una frase para poder decidir si es independiente o no del contexto. Del análisis semántico solo interesa una parte: la interpretación pragmática, la cual "[se encarga] del problema de la completitud de la información mediante la adición de información dependiente del contexto sobre la situación actual de cada interpretación candidata [que generó el análisis semántico]" [Rusell, Norving 2003]

A partir de este punto no hay una manera conocida y eficiente para realizar la interpretación pragmática, el caso más simple de resolver es el "significados de referentes", que son frases que hacen referencia directamente a la situación (contexto) actual, ejemplo: él, ello, allá, etc. Para estos casos se puede utilizar un algoritmo conocido, como el *pronoun references* de Hobbs [Hobbs 1998] que logra una eficacia considerablemente alta aunque requiere de un análisis gramatical correcto o el *Pronominal Anaphora Resolution* de [Lappin, Leass 1994] que logra una eficacia similar o mayor en algunos casos.

Requisitos para un desarrollo basado en redes neuronales: en este caso lo que se necesita es una red que pueda recibir como parámetro de entrada una frase, o una serie de valores numéricos que representen a la frase y una salida booleana (suponiendo que no se utilizará lógica borrosa) que devuelva verdadero (ó 1) si la frase es independiente del contexto y falso (ó 0) si no lo es. Dentro de los dos grandes tipos de redes neuronales: auto-organizadas (SOM) y supervisadas habría que descartar a priori las redes SOM ya que en este escenario no buscamos agrupar datos por similitud o clasificar las frases en conjuntos inciertos sino en dos conjuntos bien

definidos: frases independientes del contexto y frases dependientes del contexto.

Según Bonifacio Martín del Brío y Alfredo Sanz Molina en [Martín del Brío,Sanz Molina 2001] un perceptrón multicapa "es capaz de representar complejos *mappings* y de abordar problemas de clasificación de gran envergadura, de una manera eficaz y relativamente simple." Además cuenta con el aval de haberse utilizado satisfactoriamente en muchos casos complejos y diversos como, por ejemplo, para determinar que bancos entrarían en quiebra durante la crisis bancaria española entre 1977 y 1985. Por otro lado un perceptrón multicapa (con una única capa oculta) es un aproximador universal de funciones como quedó demostrado con el teorema de Funahashi [Funahashi 89]. Estas consideraciones hacen al perceptrón una red neuronal artificial adecuada para emprender el proceso de construcción del sistema.

Como algoritmo de aprendizaje para esta red hay que considerar al más difundido y utilizado: *BackPropgation* el cual si bien presenta el inconveniente de que "busca minimizar la función de error pudiendo caer en un mínimo local o algún punto estacionario sin llegar a encontrar el mínimo global" [García Martínez,Servente,Pasquini 2003]. No necesariamente un mínimo global es la única solución para que el perceptrón proporcione un resultado exitoso como sucede en los sistemas de Codificación de la información, Traducción de texto en lenguaje hablado o Reconocimiento de caracteres en los cuales se utiliza de manera satisfactoria el algoritmo *BackPropagation*. [García Martínez,Servente,Pasquini 2003]

5.1.4.3 Sub-sistema conversor

Para la implementación de este sistema ya se cuenta con algunos desarrollos bastante exitosos que podrían ser adaptados, por ejemplo el algoritmo de Hobbs de resolución por referencia, mencionado anteriormente. Sin embargo esto no cubre la totalidad del trabajo que debería realizar este subsistema ya que debe de poder encontrar información de contexto en casos más generales, como por ejemplo, si un usuario responde "yes" a la pregunta "Do you like movies?". En este caso debería de poder transformar "yes" en "I like movies" y aún más deberá de poder pedir información de contexto en casos en los cuales está no esté presente en el dialogo.

Si se piensa en una implementación para este sistema del tipo basado en redes neuronales, dicha red debería de poder determinar "que" es lo que la frase necesita para que esta sea independiente del contexto y luego tomando el dialogo como entrada de la red extraer dicha información, con lo cual se requerirá convertir todo el dialogo entre el usuario y el chatbot en una serie de valores de entrada para la red y luego *mapear* el resultado de salida a alguna porción de

frase del dialogo. Con lo cual es de esperar que dicha red tendría un grado considerable de complejidad.

5.1.5 Valoración de alternativas

La tabla 5.1.4.1.1 muestra claramente que ALICE es un sistema más moderno y por lo tanto más complejo y que ha dado mejores resultados que Eliza en el área de los chatbots inteligentes, también es posible ver que incorpora rudimentos de detección de información contextual (interpretación pragmática). Todo esto posiciona a ALICE y a su lenguaje de construcción: AIML por encima de Eliza.

El problema de identificación de independencia del contexto en una frase es un problema que involucra las siguientes características:

- No hay una regla clara que pueda usarse a priori para diferenciar estas dos clases de frases.
- Las frases que pueden ser clasificables son en principio infinitas.
- Es un problema relativamente sencillo para un ser humano.
- La regla si es que existe hay que extraerla de un conjunto de ejemplos.
- Los métodos algorítmicos implican el desarrollo de un analizador sintáctico (y quizás semántico) considerablemente preciso.

Si bien los ítem anteriores son bastante evidentes tampoco terminan de definir la mejor solución, sin embargo por la naturaleza del problema apuntan más hacia una solución de tipo no algorítmica. Por lo cual, según los casos analizados, la utilización de una red neuronal será la solución más propicia.

El sub-sistema conversor presenta características clásicas de sistemas algorítmicos como búsqueda de palabras (referentes) y reemplazo de sub cadenas. Por ello y por el hecho de que existen sistemas algoritmos que podrían ser utilizados como referencia o puntos de partida la solución adoptada para este subsistema será finalmente algorítmica.

5.1.6 Selección de la solución

En este punto ya se ha escogido una solución y una manera de encarar el desarrollo de la tesis, se ha obtenido una planificación tentativa y un detalle de los módulos que contendrá y de los sistemas que servirán como base de la misma. Esta información se resume en la siguiente lista de

módulos:

- Modulo ALICE original
 - importación de todo su base de datos a una base de datos relacional
- Sub-sistema detector (red neuronal)
 - sub-sistema para convertir frases en entrada de la red neuronal.
- Sub-sistema conversor (algorítmico)

5.1.7 Aceptación de la solución

En reunión mantenida con el director de tesis, se han aprobado las especificaciones de análisis vertidas en el presente documento.

5.2 Análisis del sistema de información

En esta sección se detallará el análisis que se hizo del sistema antes de entrar en la fase de desarrollo, lo cual incluye los diagramas y especificaciones de casos de uso, diagramas de entidad-relación para el modelo de datos y diagramas de clases. Clara Identificación de la tecnología necesaria y de los requisitos funcionales y no funcionales del mismo. Se define en esta misma sección las interfaces de usuarios y el plan de pruebas.

5.2.1 Definición del sistema

En esta sección detallará, a través del alcance, la tecnología involucrada y los estándares respetados que es el sistema, como funcionará y sus limitaciones.

5.2.1.1 Determinación del alcance del sistema

En esta sección se mostrará el alcance del sistema a través de los documentos y diagramas de casos de uso y clases UML y e entidad-relación para el modelo de datos.

5.2.1.1.1 Casos de uso

A partir de las definiciones de la fase anterior: Evaluación de la Solución se describe al sistema como: una mejora en el chatbot ALICE que permite a dicho sistema mejorar la calidad de sus respuestas utilizando para ello información de contexto, tomada de frases intercambiadas entre el chatbot y el usuario previamente en el dialogo. El alcance está dado por las limitaciones de los sistemas que intervienen: el subsistema detector que será implementado con una red neuronal y el subsistema conversor que será implementado de manera algorítmica. Cuando este último no encuentre información de contexto para convertir la frase deberá ser capaz de identificar que información falta y preguntar al usuario por ella.

El sistema posee un solo caso de uso, el cual tiene un flujo que se divide en sub-flujos bien identificados:

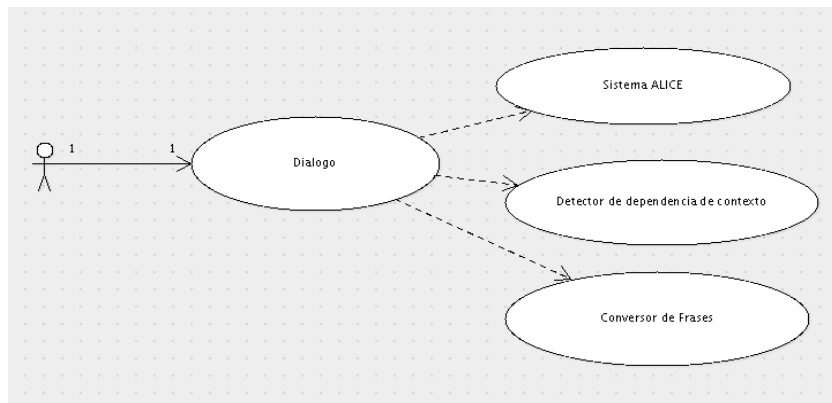


Diagrama 5.2.1: Caso de uso principal

Especificación completa del Caso de Uso.

Use Case - flujo principal: Dialogo
Descripción: El usuario envía una frase al sistema, el sistema la procesa e imprime la respuesta.
Actores participantes: usuario final
Pre-condiciones: ninguna
Flujo Principal:
1. Usuario ingresa una frase cualquiera
2. El sistema busca en la base de datos el patrón que mejor coincide con la frase de entrada
3. Para dicho patrón verifica el campo booleano "isIndependent" asociado
4. El Sistema realiza un procesamiento con dicha información (sub-flujos S4)
Sub-Flujos:
Sub-Flujos S4
S4.1.1 Si el campo es verdadero se devuelve la respuesta asociada. (Funcionamiento normal de ALICE)
S4.2.1 Si el campo es falso aplica el subsistema detector a la frase entrada.
S4.2.2 El sistema realiza un procesamiento con dicha información (sub-flujos S4.2.2)
Sub-Flujos S4.2.2
S4.2.2.1.1 Si el subsistema detector encuentra que dicha frase es independiente del contexto. Devuelve el patrón asociado.
S4.2.2.2.1 Si el subsistema detector encuentra que dicha frase es dependiente del contexto. Llama al subsistema conversor.
S4.2.2.2.2 El subsistema conversor realiza un procesamiento para cambiar la frase dependiente del contexto en una frase independiente del contexto. (sub-flujos S4.2.2.2.2)

Sub-Flujos S4.2.2.2.2
S4.2.2.2.1.1 El sistema busca información de contexto en las frases anteriores
S4.2.2.2.1.2 El sistema cambia la frase original por una frase con la información de contexto faltante
S4.2.2.2.1.3 El sistema busca el patrón asociado a la nueva frase
S4.2.2.2.1.4 El sistema devuelve la respuesta asociada a dicha frase
S4.2.2.2.2.1 El sistema busca información de contexto en las frases anteriores
S4.2.2.2.2.2 El sistema no encuentra la información en las frases anteriores.
S4.2.2.2.2.3 El sistema formula una pregunta pidiendo la información faltante.
S4.2.2.2.2.4 El sistema devuelve la pregunta anterior.
Post-condiciones: ninguna

Tabla 5.2.1.1.1.1: Flujos del caso de uso

5.2.1.1.3 Diagramas de Entidad – Relación

Como se puede observar en la WBS del punto 5.2.2.1, existen dos esquemas de datos. Por un lado se tiene una implementación de ALICE que funcionará no sobre AIML sino sobre tablas en una base de datos relacional. Esto es así debido al manejo que se requiere de los datos, el cual sería demasiado tedioso y poco práctico si se realizase sobre archivos XML y dado que se requiere modificar la implementación del intérprete AIML se optó por agregarle una modificación "extra" para que funcione sobre un esquema de datos relacional.

El segundo esquema es el Léxico, que se utilizará para realizar la conversión de frases a valores numéricos en punto flotante para que sirva de entrada a la red neuronal artificial.

La especificación de AIML indica que una categoría tiene un patrón y asociado a dicho patrón hay una o varias respuestas o bien una regla de formulación recursiva; lo que se especifica con el *tag* SRAI, si un patrón dado tiene asociado este *tag*, la frase original del usuario será reformulada y se procederá a buscar un nuevo patrón que coincida con la frase cambiada. Por ejemplo si un usuario ingresa en el sistema ALICE la frase "do you know who socartes is", ALICE usará el *tag* SRAI de la siguiente manera: "who is <star>", donde en este caso <star> será reemplazado por "socrates". Como resultado la frase del usuario será reformulada como "who is socrates" y al volver a buscar un patrón de coincidencia encontrará la respuesta final asociada. Como en muchos casos el *tag* SRAI se utiliza para llevar frases distintas, cuyo significado es el mismo, a una misma forma, funciona como una especie de filtro que normaliza y unifica las frases de entrada. En todos estos casos no conviene analizar si el patrón de coincidencia tiene o no dependencia del contexto (y por lo tanto un campo booleano "isIndependt" asociado), por eso se separaran en dos tipos las "categoy" originales de AIML, las que funcionan como filtro de frases y las que tienen respuestas finales asociadas. Es decir, toda "category" cuyo patrón de coincidencia

tenga asociado una respuesta final y no un *tag* SRAI, será tratada como caso definitivo y sobre este tipo de patrones se aplicará el subsistema detector. Por ello el esquema de datos está pensado para contemplar dicha división.

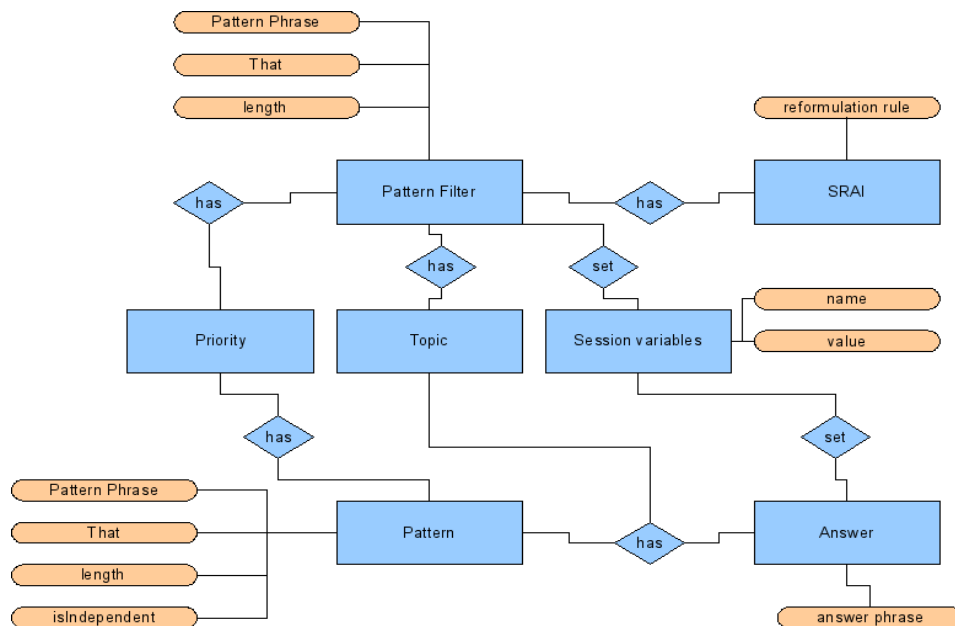


Diagrama 5.2.2: DER de ALICE-AIML en un esquema relacional

El siguiente DER pertenece al Léxico o Diccionario y su función es tener una lista exhaustiva de palabras y una clasificación básica de cada palabra, por ejemplo: sustantivo, verbo, adjetivo, adverbio, pronombres, etc. Además cada tipo de palabra tendrá asociado un valor que será utilizado al momento de transformar las frases en el mapa de entrada de la RNA.

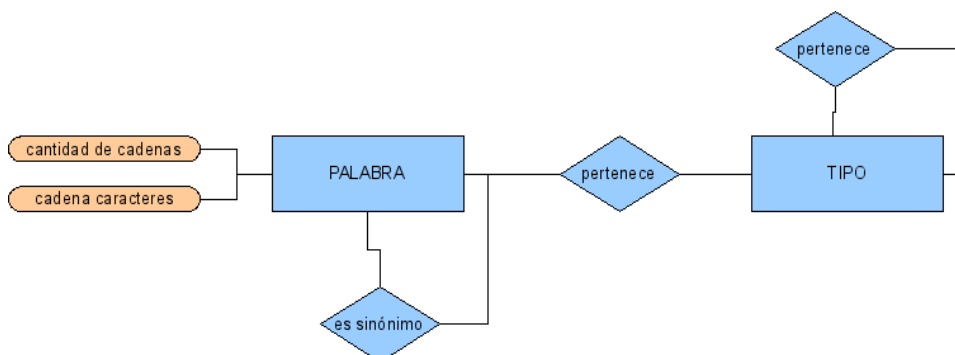


Diagrama 5.2.3: DER del Léxico

Hay que tener en cuenta que "palabra" aquí no tiene el mismo significado que se le da usualmente. Por eso una palabra es una cadena de caracteres que puede incluir espacios, por ello hay un campo extra llamado "cantidad de cadenas" dicho campo indicará cuantas subcadenas (palabras en el sentido ordinario) separadas por espacio conforman la cadena principal. Por ejemplo "Buenos Aires" es según este esquema una sola palabra, porque a "Buenos Aires" le corresponde el tipo "sustantivo" y no es lo mismo que "buenos" y "aires". La finalidad de dicha separación y el campo "cantidad de cadenas" es que al momento de *parsear* una frase y convertir cada "palabra" a su respectivo tipo, cadenas como "Buenos Aires" tengan mayor prioridad que "buenos" y "aires" , la prioridad la da el campo "cantidad de cadenas".

5.2.1.1.4 Diagramas de clases

En el diagrama 5.2.5 se muestran las clases principales del sistema final, en otras palabras a todos los subsistemas colaborando, trabajando de forma integrada.

"MyAlice" es la clase que representa al funcionamiento básico, ordinario, del chatbot pero trabajando sobre la base de datos relacional. Su único método público es "replyTo", este método es el método principal que tendrá el sistema final como *interfaz* para interactuar con otros

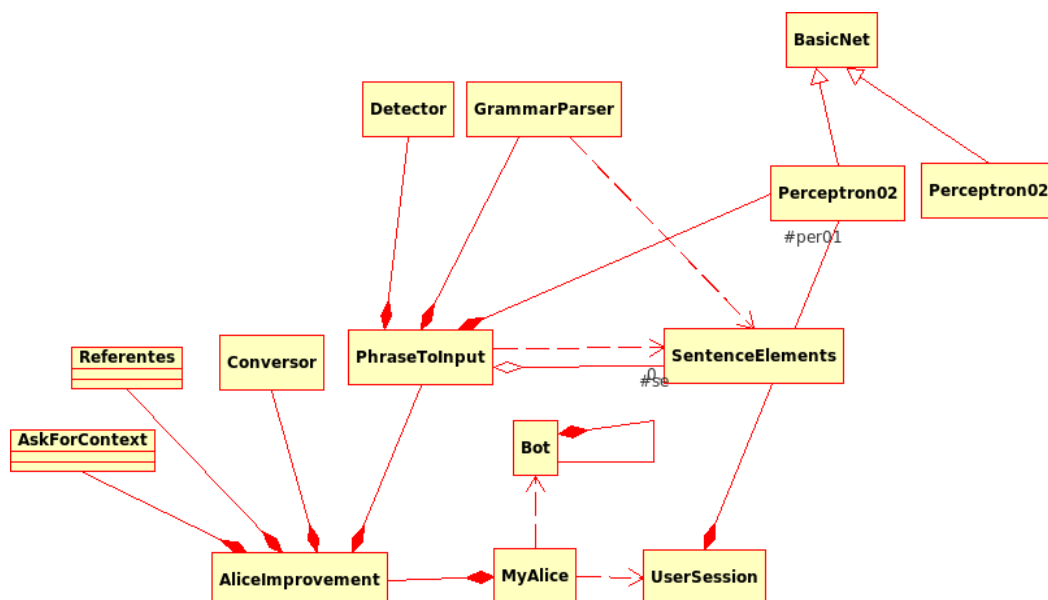


Diagrama 5.2.4: Diagrama de Clases

sistemas.

Dicho método recibirá un objeto "UserSession" y una frase de entrada del usuario y devolverá un *String* con la respuesta del chatbot.

El chatbot para responder normalmente necesita por un lado ir guardado datos asociados a las frases anteriores (variables de sesión) y conocer las variables (o constantes) propias que no varían, como por ejemplo: la película favorita del chatbot, su nombre, edad, etc. Ya que esta información puede ser usada para generar dinámicamente respuestas. Para el segundo caso, los atributos del chatbot, se creará la clase "Bot" la cual será un *singleton* y podrá ser consultada en cualquier momento. Para el primer caso se utilizará una clase especial "UserSession" la cual contiene atributos para guardar el contenido de lo que el chatbot respondió anteriormente: "that", "topic" actual, variables de sesión asociadas a las respuestas anteriores y las subcadenas que coincidieron con los *wildcards* de los patrones del chatbot.

Con esta breve descripción se resumió la implementación del primer subsistema, sobre el que se construirán luego las mejoras. Los detalles de los demás subsistemas se describirán en los puntos siguientes.

5.2.1.2 Identificación del entorno tecnológico

Las herramientas tecnológicas que se utilizarán para el desarrollo del sistema y el ambiente de desarrollo se describen a continuación:

El lenguaje de programación que se utilizará será: **Java, versión: 1.6** ya que es un lenguaje de programación estable, ampliamente documentado y probado con el cual se puede desarrollar software de manera rápida por su propia naturaleza (no requiere alocar/liberar memoria, tiene una biblioteca extensa de funciones, es *multiplataforma*, etc.) y por lo depuradas y maduras que están las herramientas para trabajar con dicho lenguaje, por ejemplo las IDEs de desarrollo. No es un lenguaje completamente interpretado (genera un código intermedio) y por ello es suficientemente rápido como para que se puedan probar sobre él redes neuronales medianamente complejas como las que se utilizaran. Java soporta además de manera nativa expresiones regulares las cuales serán indudablemente utilizadas en varios subsistemas, particularmente en el detector de tiempos verbales. Finalmente porque el *framework* libre y herramienta de diseño e implementación de redes neuronales Joone está hecho sobre Java y es la herramienta escogida para diseñar, probar e implementar la red neuronal que se utilizará.

La interfaz de desarrollo que se utilizará será **Eclipse, versión 3.2** dado que es la herramienta libre de desarrollo para Java más utilizada y versátil del mercado.

Como base de datos se utilizará **mySQL 5**, mySQL es una de las bases de datos más rápidas del mercado, en particular con volúmenes de datos no demasiado grandes como para que

se requiera una implementación en *cluster* y con consultas que no requieran *transaccionalidad* como es el caso de este sistema. MySQL además soporta expresiones regulares lo cual podría ser una ventaja dado que se reemplazará el algoritmo de *pattern matching de AIML* por el de esta base de datos. Finalmente hay que destacar que es un motor de base de datos completamente libre.

El *framework* que dará soporte a la red neuronal será **Joone** como se anticipó. Joone es otra herramienta libre que cuenta además con un entorno gráfico sobre el cual se puede de manera fácil y rápida diseñar redes neuronales, realizar corridas de entrenamiento y ver los resultados en un gráfico cartesiano.

Se utilizará además para versionar el código (y mantener copias de seguridad) la herramienta SVN (*Subversion*), que será provista por algún servidor gratuito en la web: **Assembla / GoogleCode**.

Finalmente, el sistema operativo sobre el cual se trabajará será **GNU-Linux**, versión de kernel: 2.6.22-15. distribución Kubuntu 7.10.

5.2.1.3 Especificación de estándares y normas

No hay ningún estándar asociado al desarrollo del sistema que se deba cumplir. Lo más cercano a un estándar que se debe respetar, aunque no está certificado por ningún organismo competente y su especificación no está correctamente formalizada, es AIML. En particular el texto "The elements o AIML Style" [Wallace, 2003] que define el comportamiento que debe tener un intérprete de AIML será el punto de partida para la construcción de esta versión del intérprete sobre una base de datos relacional.

5.2.2 Establecimiento de requisitos

El siguiente capítulo detalla los requisitos funcionales y no funcionales.

5.2.2.1 Obtención de requisitos y tareas

A continuación se presenta la WBS, con la totalidad de tareas ordenadas jerárquicamente y construida a partir del análisis de requisitos de la fase anterior y la especificación del caso de uso

previo. Esta lista de tareas será definitiva y decisiva para realizar los diagramas de entidad relación y clases. Hay que tener en cuenta que es posible llegar a este grado de detalle porque ya fueron realizadas las tareas iniciales: Investigación, Selección del chatbot y Definición de mejoras. Por eso dichas tareas no aparecen en la WBS.

Comportamiento Adaptable de Chatbots Dependiente del Contexto

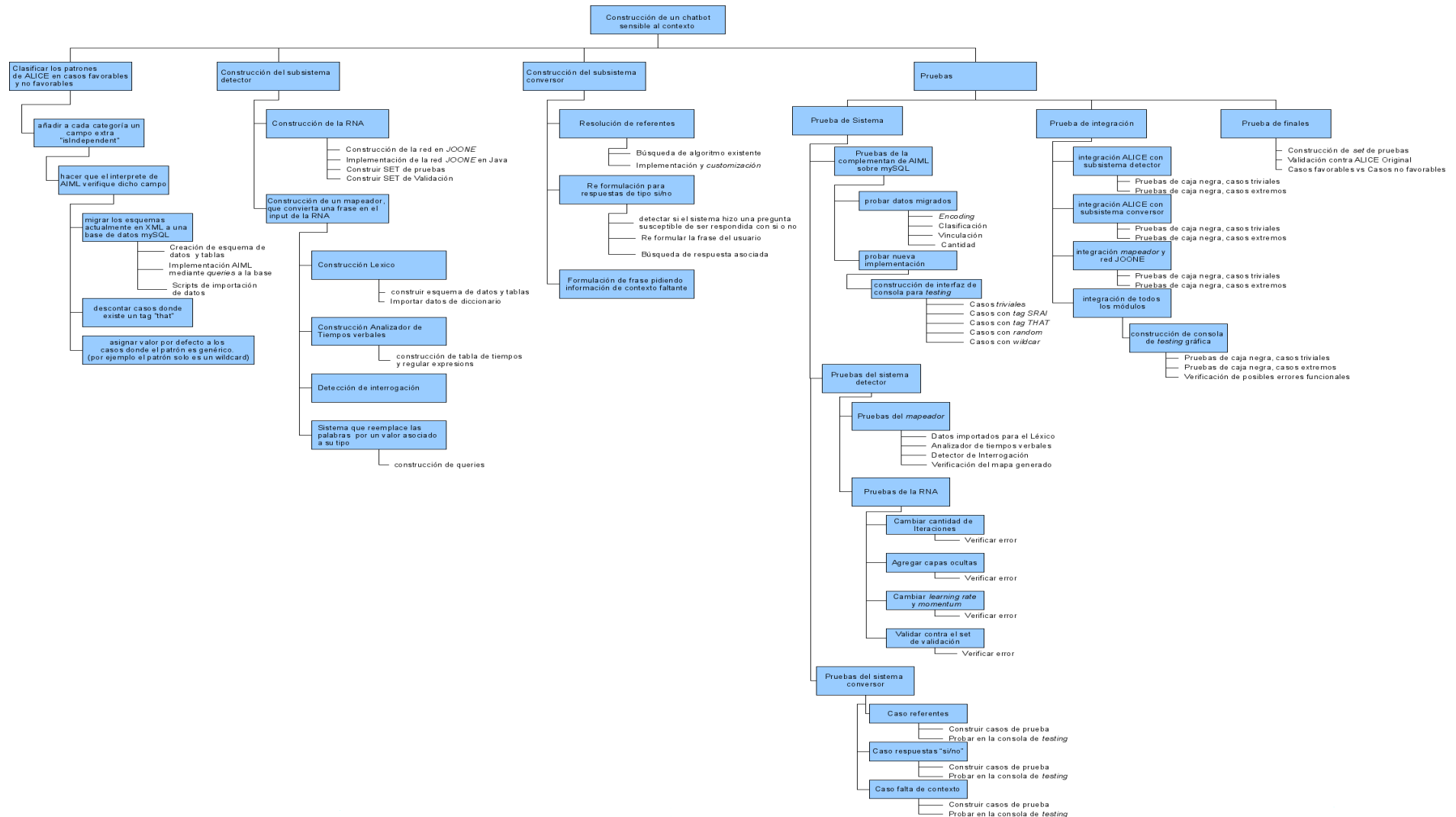


Diagrama 5.2.5: WBS

5.2.3 Identificación de subsistemas de análisis

En esta sección se listarán todos los subsistemas identificados que en su conjunto conformarán al sistema global final.

5.2.3.1 Determinación de subsistemas de análisis

En la siguiente lista se incluirán de manera exhaustiva todos los subsistemas necesarios para desarrollar el chatbot aunque estos no formen parte del producto final. Todos ellos se desprenden de la WBS anterior.

- **myAlice**: Implementación de un intérprete de AIML sobre una base de datos relacional cuyo contenido será equivalente al llamado "cerebro de ALICE" conformado por 39 archivos "aiml". El motivo de trabajar con este subsistema y no con "programv", el intérprete actual de AIML, es la necesidad de disponer de los datos en un medio manipulable para su posterior clasificación.

- **Subsistema Detector**: Este subsistema será capaz de evaluar una frase cualquiera en inglés y devolver uno o cero si esta es dependiente o independiente del contexto. A este subsistema se lo puede dividir a su vez en dos subsistemas "*mapeador*" y "red neuronal".

- **red neuronal**: En principio será implementada con un arquitectura de tipo perceptron con el algoritmo de aprendizaje *backpropagation*. Su *input* estará dado por el mapa que es la salida el subsistema "mapeador" y su *output* por una sola neurona booleana.

- **mapeador**: Este subsistema será capaz de convertir una frase en inglés en una serie de valores de punto flotante que servirán de *input* a la RNA construyendo así un mapa. Este subsistema a su vez puede ser descompuesto en tres subsistemas.

- **Detector de tiempos verbales**: Este subsistema detectará el tiempo verbal de una oración (frase) al aplicar sobre la misma una serie de expresiones regulares. Su *output* será un valor numérico, único por cada tiempo verbal o cero si no pudo detectarlo.

- **Detector de interrogación**: Este subsistema tomará como entrada una frase y devolverá uno o cero si la frase es una interrogación o no. No se fijará solo en la presencia del símbolo "?" (interrogación) sino en que la frase en sí tenga la estructura sintáctica y las palabras propias de una frase de interrogación.

- **Parser**: este subsistema se encargará de reemplazar cada palabra (en este caso palabra implica que puede haber más de una cadena de caracteres separadas por

espacio, ver punto: 5.2.1.1.3) por el valor asociado a su tipo. Esta información la tomará de la base de datos.

- **Subsistema Conversor:** Este subsistema será el que se encargue de tomar una frase dependiente del contexto y detectar en la frase la información que falta, buscar dicha información en el contexto (frases anteriores), reemplazar o reformular la frase con la nueva información convirtiendo a dicha frase en una nueva independiente del contexto. Como el funcionamiento descrito de manera teórica es muy amplio el alcance de este subsistema será acotado a solo tres módulos, 1. resolución de referentes, 2. reformulación de frases: "sí/no", 3. Elaboración de preguntas que pidan al usuario la información faltante.

-- **Sist. de resolución de referentes:** "La resolución por referencia es la interpretación de un pronombre o una frase nominativa definitiva que hace referencia a un objeto en el mundo. La resolución se basa el conocimiento del mundo en las partes previas del discurso [...] Usualmente la resolución por referencia es un problema de seleccionar una referencia de una lista de posibles candidatas" [Russell, Norving 2003]. El siguiente ejemplo ilustrará mejor la definición anterior, en la frase: "Dana dejó caer el vaso sobre el plato. Éste se rompió". El sistema deberá de poder reemplazar la palabra "Este" por alguna de las candidatas "vaso" y "plato", aunque aquí no está claro ni para un ser humano cual es la referencia correcta. Como actualmente existen algoritmos que se ocupan de este tipo de problemas, como el de Hobbs, previamente mencionado, este subsistema consistirá en la implementación de un algoritmo existente.

-- **Sist de reformulación de frases "sí/no":** Este tipo de problemas es común en los chatbots. En los casos en los cuales un chatbot envía al usuario una pregunta susceptible de ser respondida por si o por no, como podría ser "Do you like read?" (pregunta cerrada) y el usuario responde simplemente "yes"(o "no"), luego al analizar esa respuesta el chatbot "pierde" el hilo de esa conversación porque no genera una respuesta en relación a lo que preguntó. En estos casos el subsistema debería reemplazar, por ejemplo "yes" por "I like read" (Simplemente cambiando "Do you" por "I", en este caso.) Este subsistema se puede dividir en dos más: uno es un subsistema para detectar cuando una pregunta puede ser respondida por "sí" o por "no" (por ejemplo las frases que comienzan con "Do you", "Are you", etc.) y el segundo es el que realiza el reemplazo.

-- **Sist de petición de información faltante:** En los casos en los cuales los sistemas anteriores fallen o bien no puedan ser aplicados, se activará este sistema de contingencia que buscará en la frase del usuario las palabras que indiquen que se solicita información de contexto, y con ellas armará una frase la cual será una pregunta dirigida al usuario

en donde se pida más información de contexto. Esta frase debe de ser armada en un tono coloquial de la misma forma en la que un ser humano pediría la misma información. Tomando nuevamente el ejemplo dado para el subsistema de resolución de referentes: "Dana dejó caer el vaso sobre el plato. Éste se rompió", al fallar el subsistema de resolución de referentes el sistema actual debería armar una frase como la siguiente "¿Que cosa se rompió?". O por ejemplo si un usuario ingresase una frase como "Who?", en medio de una conversación en la cual dicha interrogación no tiene sentido, el chatbot debería de poder responder "Who what?". Este es uno de los errores marcados al analizar las transcripciones de las conversaciones de los chatbots de [Loebner, 2007]

-Importador: Este sistema es un pequeño programa auxiliar, que se utilizará para importar los datos necesarios en las bases de datos myAlice y Lexico, respectivas a los subsistemas aquí descriptos: myAlcie y *mapeador*. Dicho sistema será capaz de *parsear* los archivos aiml, interpretarlos e importarlos en las tablas correspondientes. Además será capaz de *parsear* los archivos de diccionario de OpenOffice, y archivos CSV e importar dicha información en la base de datos Léxico.

-Consolas de Chat: Las consolas de *chat* son los instrumentos que permitirán realizar pruebas directas sobre el sistema myAlice terminado y luego sobre el sistema final (myAlice más las mejoras). En principio serán dos consolas una muy sencilla en modo texto, para realizar pruebas de laboratorio y luego otra con elementos visuales que permitirá que cualquier persona pruebe el sistema y servirá además para presentarlo. Esta segunda consola gráfica será realizada utilizando la biblioteca gráfica de Java Swing.

Los últimos dos subsistemas no se mostrarán en el esquema de paquetes ni de clases ya que son externos y no forman parte del sistema final.

A continuación se muestra un diagrama en el cual se puede identificar claramente cada subsistema a partir de la estructura de paquetes que contendrán a las clases.

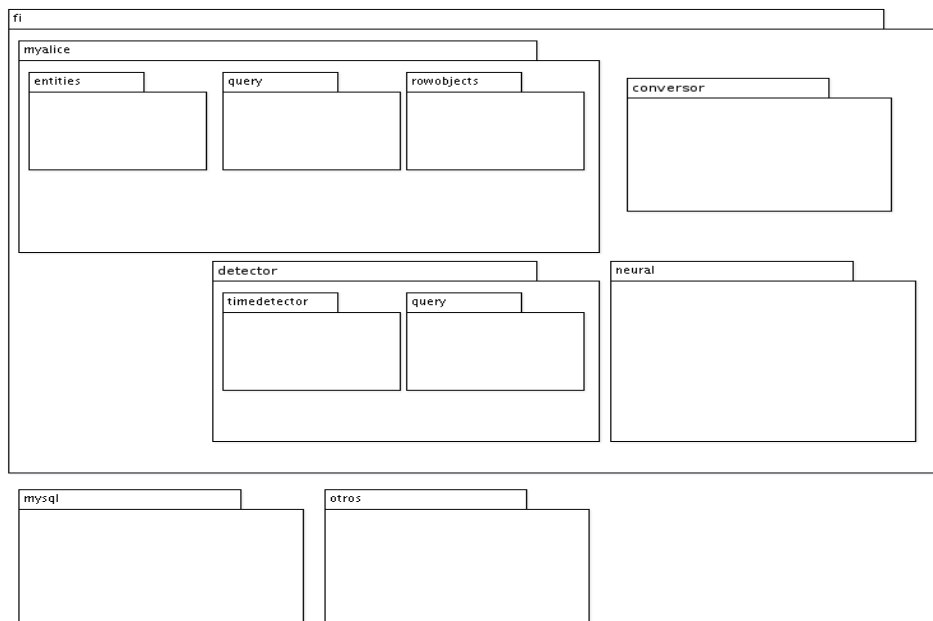


Diagrama 5.2.6: Paquetes

La primera gran división está dada por el paquete "fi", (dentro del árbol: "ar.uba"), lo que está dentro de este paquete es el código propio del sistema. Los paquetes externos "mysql" y "otros" representan código externo que simplemente se utiliza. Dentro de "fi" hay 5 paquetes, "myalice": este paquete contiene el subsistema myAlice.

"detector": este paquete agrupa las clases relevantes del subsistema detector.

"neural": tiene las clases que representan a la red neuronal artificial, estas utilizan clases externas, que están dadas por el paquete de "Joone".

Los paquetes "query" que están dentro de los paquetes "detector" y "myAlice" contienen las consultas que ambos subsistemas realizarán sobre las bases de datos para obtener diferentes tipos de información. Las consultas se manejan así debido a su complejidad, si fuesen "selects" comunes y corrientes hubiesen quedado ocultas en algún paquete "database".

5.2.4 Análisis de casos de uso

En esta sección se realizará el análisis correspondiente al caso de uso mostrado en la sección anterior, identificando nuevas clases y subsistemas.

5.2.4.1 Identificación de clases asociadas a un subsistema

Como este sistema es un poco particular y contiene un solo caso de uso, se mostrará una modificación de la clásica matriz de trazabilidad y en lugar de mostrar clases vs. casos de uso se mostrarán clases vs subsistemas. Lo cual aquí es mucho más útil.

Paquete	Clase \ Subsistema	myAlice	Detector			Conversor		
			neuronal	mapeador	parser	referentes	"si/no" conv.	info. faltante
enities	Bot	x						
	UserSession	x						
neural	BasicNet		x					
	Perceptron		x					
myAlice	MyAlice	x						
	AliceImprovement			x	x		x	x
neural	phraseToInput			x				
otros	Referentes					x		x
time verb detector	Detector			x				
detector	Grammarpaser				x			
	SentenceElements	x		x	x			
conversor	AskFor Context							X
	Conversor						x	

Tabla 5.2.4.1.1: Clases Principales vs Subsistemas

5.2.5 Análisis de clases

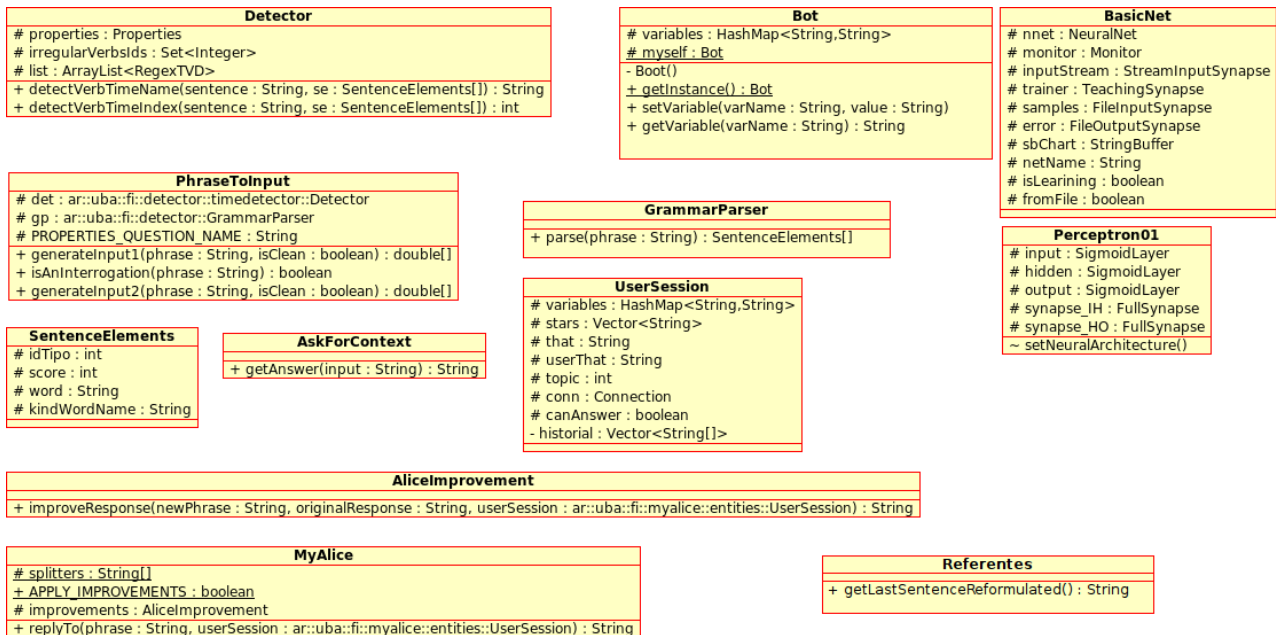


Diagrama 5.2.7: métodos y atributos de clases

5.2.5.1 Identificación de responsabilidades y atributos

En el diagrama 5.2.7 se muestran a las mismas clases del diagrama 5.2.4, pero esta vez se obviaron las agregaciones, dependencias y generalizaciones y se agregaron los atributos y métodos más importantes de cada clase, a continuación se explicará la responsabilidad de cada clase.

GrammarParser: La responsabilidad de esta clase está dada por su único método, el cual toma una frase de entrada del usuario (en inglés) y devuelve un *array* de *SentenceElements*. El método en sí lo que hace es reemplazar las palabras de la frase original (palabras compuestas por una o más cadenas separadas por espacio(s)) por su tipo: sustantivo, verbo, adjetivo, etc. Utilizando para ello las definiciones en la base de datos Lexico. Por cada reemplazo crea un *SentenceElement*, el cual es un objeto que encapsula toda la información asociada a dicha palabra.

SentenceElements: Esta clase sin comportamiento existe con la finalidad de encapsular un elemento de una frase, este elemento será por lo general una palabra y su información asociada: tipo de palabra, identificación numérica del tipo de palabra, puntaje (este campo será utilizado por el *mapeador*, dado que algunos tipos de palabras son más relevantes que otros) y la

palabra en sí.

Detector: Esta clase tiene dos métodos "detectTimeVerbName" y "detectTimeVerbindex", ambos métodos reciben como parámetro un *String* con la frase original del usuario y un *array* de *SentenceElements* como lo devolvió el método de *GrammarParse*. Y devuelven un indicador numérico del tiempo verbal en el cual está la oración o bien el nombre del tiempo verbal (a fines de *testear* su correcto funcionamiento.) Como esta clase utilizará expresiones regulares para lograr el cometido de los métodos, tendrá un atributo de tipo *Properties* asociado a un archivo de propiedades donde estarán las expresiones regulares escritas en texto plano. Todas las expresiones regulares se cargarán en memoria en el *array* denominado *list*. Por último se observa un atributo "Irregularverbslds" el cual es un conjunto de enteros. Dado que los verbos irregulares no respetan las formas tradicionales de la oraciones, las *regex* no funcionarán si hay verbos irregulares. Para superar estos casos la clase podrá identificar los verbos irregulares a través de Léxico y obtener directamente de la base el tiempo verbal en el que están las oraciones.

Bot: Clase *singleton* asociada a un archivo de propiedades donde estarán los atributos fijos del chatbot. Esta clase podrá ser consultada por cualquier método para generar respuestas dinámicas.

UserSession: Esta clase sirve para agrupar toda la información de contexto a lo largo de una sesión de usuario (entiéndase una "conversación" entre un usuario determinado y el chatbot) Existirán simultáneamente tantos objetos *UserSession* como usuarios simultáneos estén "conversando" con el chatbot. Pero la creación de estos objetos será responsabilidad del sistema que implemente este chatbot en un ambiente real. Alguno de sus atributos son propios de ALICE / AIML: "variables", "stars", "topic" y "that" y otros son propios de este nuevo chatbot como "userThat", "conn", "canAnswer" e "historial". "UserThat" es la frase que dijo el usuario anterior a la actual, "conn" es una referencia a la conexión a la base de datos (esto aún está por definirse si se creará una conexión por sesión o no pero se destaca aquí para indicar que este sistema utiliza una base de datos relacional y no aiml.) "canAnswer" es un atributo booleano que se utilizará cuando el sistema tradicional no pueda dar una respuesta satisfactoria y se tengan que aplicar las mejoras. Finalmente historial es una referencia al historial completo de la conversación.

AskForContext: Esta clase será invocada cuando el sistema no pueda encontrar información de contexto en la conversación, entonces a partir de la frase actual del usuario el método "getAnswer()" formulará una pregunta pidiendo la información faltante.

PhraseToInput: Esta clase, utilizando los métodos de *Detector* y *GrammarParse* más un método propio que le permite identificar si una frase es una pregunta generará el "mapa" que utilizará la red neuronal artificial como *input* a partir de una frase dada. Como en principio podría haber varios mapas posibles, al menos en la etapa de prueba, puede haber varios métodos

"generateInput" en el diagrama se muestran 2. Pero siempre devolverán un *array* de *double*, dado que esa es la interfaz que define Joone.

BasicNet: Esta clase incorporará todos los elementos de la API de Joone necesarios para armar una red neuronal artificial y encapsularla en una clase Java. La idea es que luego extendiendo esta clase y modificando solo el método "setNeuralArchitecture" se obtengan varias arquitecturas diferentes de RNAs.

Perceptrón01: Esta clase representa un perceptrón con un determinado número de neuronas de entrada, de salidas e incluso si tiene o no una capa oculta y con cuantas neuronas. Durante la fase de pruebas podrán existir varias clases como esta con diferentes arquitecturas.

MyAlice: Esta clase representa al sistema original solo que corriendo sobre una base de datos relacional. Su único método "replyTo" recibe la frase del usuario y un objeto *UserSession* asociado al dialogo. Devuelve un *String* cuyo contenido es la respuesta y dicha respuesta debería de coincidir con la que da el chabot ALICE para una misma frase siempre y cuando la constante APPLY_IMPROVEMENTS esté en "false". Cuando el sistema esté finalizado, el usuario que lo implemente solo interactuará contra este método.

AliceImprovements: Esta clase tiene también un único método el cual aplica las mejoras implementadas para generar una respuesta.

Referentes: Esta clase encapsula el comportamiento de un algoritmo de resolución de referentes. Ya se que se utilice el algoritmo *pronoun references* [Hobbs 1978] o *Pronominal Anaphora Resolution* [Lappin, Leass 1994], o bien cualquier otro. Eventualmente utilizará un biblioteca que encapsule no solo el algoritmo sino todo las funcionalidades necesarias por este como un parser de lenguaje natural, etc.

5.2.5.2 Identificación de asociaciones y agregaciones

La siguientes observaciones están basadas en el diagrama 5.2.4. No hay mucho que añadir, el diagrama en sí es bastante explicito. La arquitectura de clases se centra sobre la clase principal: "myAlice". Dicha clase está compuesta por "AliceImprovement" la cual está compuesta por todas las clases que componen los subsistemas "detector" y "conversor". Y en particular la clase "PhraseToInput" está compuesta por "GramerParser" y "Detector" que son las clases que conforman el subsistema "detector" y por la clase "perceptron".

5.2.5.3 Identificación de generalizaciones

Hay una sola generalización y está dada por la "Perceptron01" y "Perceptron02" y todas las que pueda llegar a haber que heredan de "BasicNet"

5.2.6 Definición de interfaces de usuario

En esta sección se detallará la forma en la cual el sistema será accesible por un usuario final, en otras palabras la forma que tendrá la interfaz de usuario.

5.2.6.1 Especificación de principios generales de la interfaz

El sistema está pensado como una biblioteca, por lo cual no hay una interfaz de usuario con la que interactuar. Pero como se detalló antes hay un método que es el único al que se debería de tener que invocar en caso de querer realizar la implementación de una interfaz de usuario. Dicho método es "replyTo" de la clase "myAlice".

Por ejemplo las consolas de *testing* que se desarrollarán para este proyecto harán una implementación básica que tendrá la siguiente forma:

```
UserSession uSession = new UserSession();
MyAlice alice = new MyAlice();
while(true){
    lineReaded = readLineFromConsole();
    System.out.println(alice.replyTo(lineReaded,uSession));
}
```

El ejemplo anterior muestra un caso simple donde la implementación de la consola de *chat* se realizó mediante una consola de texto y para un solo usuario. Si fuese una implementación para un *chat* en internet tendría que haber una lista de objetos "UserSession", uno por cada usuario actualmente conectado.

En términos generales una implementación cualquiera, de una consola de *chat* no tiene más de tres elementos: un area de texto donde el usuario final escribe, un botón enviar que envía la frase escrita al sistema (al chatbot) y un área donde se van mostrando las líneas de dialogo.

Desde el punto de vista de un servidor en donde puede estar corriendo el sistema se requerirá además una base de datos mySQL corriendo, y los archivos de propiedades correspondientes (aún no definidos) en donde se especifiquen los datos de conexión.

5.2.7 Especificación del plan de pruebas

En esta sección se detallará cual será el plan de pruebas a ejecutar para corroborar que el sistema en cada una de sus fases de construcción, se comporta sin errores. Las pruebas están divididas en tres conjuntos, los cuales son:

Pruebas de sistema: conjunto de pruebas que corroboran que cada subsistema no posee errores y se comporta de la forma esperada.

Prueba de integración del sistema: conjunto de pruebas que corroboran que la integración de los diferentes subsistemas no acarrea nuevos errores.

Pruebas de aceptación del sistema: conjunto de pruebas que verifican en su conjunto que el sistema no solo se comporta sin errores sino que responde de forma diferente al chatbot original. Son pruebas que buscan corroborar o descartar la hipótesis tomada al comienzo de esta Tesis.

5.2.7.1 Definición del alcance de las pruebas

A continuación se detallarán las pruebas a realizar en cada etapa del desarrollo, las pruebas de sistema y las pruebas de integración. Asimismo se especificarán las acciones a tomar en caso de que las pruebas no den el resultado esperado.

5.2.7.1.1 Pruebas de sistema

Estas pruebas están pensadas para probar cada uno de los sistemas y subsistemas por separado. También se probarán los resultados de ciertas tareas como las de importación de datos.

Sistema	Implementación de AIML sobre una base relacional
Subsistema o Tarea	Migración de datos
Descripción	Verificar que no haya habido pérdida en la calidad o en la cantidad de datos. Y que su orden y sentido permanezca.
Casos a probar	1. Verificar la cantidad de tags "category" importadas
	2. Verificación de caracteres no estándar
	3. Reconstrucción de datos normalizados

Comportamiento Adaptable de Chatbots Dependiente del Contexto

	4. Clasificación de tags "category" según el tag "srai"
	5. Clasificación de tags "category" según "topic"
	6. verificar la importación correcta de las variables a <i>setear</i> asociadas a una "category"
	7. verificar la importación correcta de los tags "template" que utilizan variables con el tag "star" para armar la respuesta.
Implementación de los casos	1. Contar las tags "category" que tiene cada archivo "aiml" y luego de su importación realizar un <i>count</i> sobre las tablas correspondientes y verificar que los números coincidan
	2. Buscar en los archivos AIML porciones de texto donde haya caracteres no estándar (áéíóúÁÉÍÓÚñÑ¿¡). Encontrar al menos 10 casos. Buscar mediante consultas a la base de datos las mismas porciones de texto y verificar que no hubo cambios
	3. Tomar 10 tags "category" de los archivos AIML, luego mediante "inner joins" en la base de datos verificar que asocian correctamente los "patterns" a los "template", "srai" y/o tags correspondientes
	4. Tomar 10 casos de frases con "srai" y verificar que están en la tabla "alice_filter" y "alice_srai" respectivamente los tags "pattern" y "srai"
	5. Tomar 10 casos de tags "category" que estén en "topics" diferentes y/o en ninguno y verificar que se cumple la asociación al hacer "inner join" con la tabla "alice_topic"
	6. Tomar 10 casos en donde haya asociados a una "category" el <i>seteo</i> de variables. Verificar si las asociaciones se cumplen al hacer "inner join" con la tabla <i>alice_setVariable</i>
	7. Tomar 10 casos en donde haya, asociados a un "template", variables previamente <i>seteadas</i> y el tag "star". Verificar si las asociaciones se cumplen al hacer "inner join" con la tabla <i>alice_setVariable</i> en la base de datos.
Tolerancia	1. Ambos valores deben ser iguales
	2. Los 10 casos deben ser iguales.
	3. Los 10 casos deben ser iguales.
	4. Los 10 casos deben ser iguales.
	5. Los 10 casos deben ser iguales.
	6. Los 10 casos deben ser iguales.
	7. Los 10 casos deben ser iguales.
Plan de acción	En todos los casos, se realizará lo siguiente: 1. Se apartará el tag "category" que presentó problemas 2. Se creará un archivo aiml en donde solo esté dicho "category" 3. Se borrarán todas las entradas de la base de datos 4. Se <i>debugueará</i> el <i>script</i> de importación hasta identificar el problema 5. Se corregirá el código erróneo 6. Se volverá a realizar la importación 7. Se volverán a correr las pruebas.

Tabla 5.2.7.1.1.1: Implementación de AIML sobre una base relacional, calidad de datos

Sistema	Implementación de AIML sobre una base relacional
Subsistema o Tarea	Comportamiento de ALICE
Descripción	Verificar que la nueva implementación de ALICE responde de la misma manera que la versión original sobre AIML
Casos a probar	1. Casos simples, triviales donde solo hay un "pattern" posible y un "template" asociado
	2. Casos en donde hay un tag "srai" asociado a la "category"
	3. Casos en donde hay un tag "that" asociado a la "category"
	4. Casos en donde hay un tag "random" asociado al tag "template" de una "category"
	5. Casos donde hay <i>wildcars</i> en medio de un tag "pattern"

Comportamiento Adaptable de Chatbots Dependiente del Contexto

	6. Casos en donde se utilizan variables dentro del <i>tag</i> "template"
	7. Casos en donde se utiliza el <i>tag</i> "star" dentro de un <i>tag</i> "template"
	8. Casos en donde el <i>tag</i> "category" está dentro de un <i>tag</i> "topic"
Implementación de los casos	1. Buscar 5 "patterns" simples y utilizarlos como frases de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Verificar que las respuestas dadas son iguales.
	2. Buscar 5 tags "category" con <i>tag</i> "srai" asociado y utilizar el contenido del <i>tag</i> "pattern" como frase de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Verificar que las respuestas dadas son iguales.
	3. Buscar 5 tags "category" con <i>tag</i> "that" asociado y utilizar el contenido de <i>tag</i> "pattern" como frase de "entrada" después de haber usado el contenido del <i>tag</i> "that" en ALICE y en myALICE a través de la consola de <i>testing</i> . Verificar que las respuestas dadas son iguales.
	4. Buscar 5 tags "category" con <i>tag</i> "random" asociado y utilizar el contenido de <i>tag</i> "pattern" como frase de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Verificar que las respuestas dadas son iguales o pertenezcan a la misma asociación. Probar varias veces y verificar que no siempre respe igual.
	5. Buscar 5 tags "category" cuyo <i>tag</i> "pattern" use <i>wildcars</i> y utilizar su contenido como frases de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Reemplazando los <i>wildcars</i> por palabras. Verificar que las respuestas dadas son iguales.
	6. Buscar 2 tags "category" cuyo <i>tag</i> "template" use variables y utilizar el contenido del <i>tag</i> "pattern" asociado como frase de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Previamente habiendo utilizado como frases de entrada los casos necesarios para <i>setear</i> los valores de dichas variables. Verificar que las respuestas dadas son iguales.
	7. Buscar 2 tags "category" cuyo <i>tag</i> "template" use el <i>tag</i> "star" y utilizar el contenido del <i>tag</i> "pattern" asociado como frase de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Verificar que las respuestas dadas son iguales.
	8. Buscar 2 tags "category" dentro de un <i>tag</i> "topic" y utilizar el contenido del <i>tag</i> "pattern" asociado como frase de "entrada" en ALICE y en myALICE a través de la consola de <i>testing</i> . Habiendo utilizado previamente como frases de entrada los casos necesarios para <i>setear</i> el valor de "topic". Verificar que las respuestas dadas son iguales.
Tolerancia	1. 5 de 5. A menos que haya dos "patterns" iguales asociados a "templates" diferentes. En dicho caso repetir la misma frase hasta que myAlice devuelva la misma frase de ALICE.
	2. 5 de 5. A menos que haya dos "patterns" iguales asociados a "templates" diferentes. En dicho caso repetir la misma frase hasta que myAlice devuelva la misma frase de ALICE.
	3. 5 de 5.
	4. No importa si la frecuencia es la misma en ambos sistemas, o si se repite más una frase que las otras mientras devuelva una de las posibles frases asociadas a la respuesta. Y mientras no sea siempre la misma.
	5. 5 de 5. A menos que haya dos "patterns" iguales asociados a "templates" diferentes. En dicho caso repetir la misma frase hasta que myAlice devuelva la misma frase de ALICE.
	6. 2 de 2
	7. 2 de 2
	8 2 de 2
Plan de acción	En todos los casos anteriores, si las pruebas no son satisfactorias hay que seguir los siguientes pasos: 1. <i>Debuguear</i> el sistema 2. Verificar como queda armada la consulta sobre la base 3. Modificar la consulta para que devuelva el valor deseado. O bien el código fuente si hay que agregar un caso extra (es decir una consulta nueva) 4. Una vez que todas las pruebas de un caso pasaron, volver a correr las anteriores para verificar que al modificar la consulta los casos previos siguen siendo validos.

Tabla 5.2.7.1.1.2: Implementación de AIML sobre una base relacional, respuestas de ALICE

Comportamiento Adaptable de Chatbots Dependiente del Contexto

Sistema	Sistema Detector
Subsistema o Tarea	<i>Mapeador</i> – Generación de la base de datos Léxico
Descripción	Verificar que se hayan importado correctamente los diccionarios, que la integridad y cantidad de los datos sea la misma. Así mismo que al desnormalizar las tablas se recuperen correctamente las clasificaciones de los datos
Casos a probar	1. Verificar la cantidad de palabras importadas sea la misma que en los diccionarios
	2. Verificar que se hayan importado correctamente palabras con caracteres no estándar.
	3. Reconstrucción de datos normalizados, verificar que las categorías se mantienen.
Implementación de los casos	1. Cada vez que se esté por importar un archivo, contar las palabras nuevas. Hacer un <i>count</i> sobre la base de datos antes y después de la importación del archivo; verificar que la diferencia coincide con el número de palabras nuevas.
	2. Buscar en los archivos palabras, si las hubiese, con caracteres no estándar (áéíóúÁÉÍÓÚñÑ¿¡). Encontrar como máximo 10 casos. Buscar mediante consultas a la base de datos las mismas palabras y verificar que no hubo cambios
	3. Tomar 10 palabras asociadas a diferentes categorías, según los diccionarios importados. Luego hacer una consulta sobre la base de datos que relacione las tablas de palabras y categorías, buscar las mismas palabras. Verificar que la categorías coinciden.
Tolerancia	1. 15% de tolerancia dado que podría haber palabras repetidas, en dichos casos se realizará otro test para verificar que importaron en efecto se realizó.
	2. Todas las palabras deben coincidir.
	3. Los 10 casos deben ser iguales.
Plan de acción	1. Si la cantidad de palabras es menor, entonces tomar 3 palabras del nuevo archivo a importar: la primera, la última y una del medio. 2. Verificar que las tres palabras están en la base de datos. 3. Si no están volver a un <i>dump</i> anterior 4. Crear un mini archivo con las 3 palabras no importadas 5. <i>Debuguear</i> el import de esas 3 palabras y corregir el script. 6. Volver a realizar el import y el Test.
	En caso de que no coincidan, revisar el <i>encoding</i> del archivo de texto con las palabras y el de la base de datos. Detectar la diferencia y convertir el archivo de texto con el comando "iconv"
	Al igual que en el punto anterior, si no hay coincidencias: 1. Restaurar <i>dump</i> anterior 2. Crear un mini archivo con los 10 casos problemáticos 3. <i>Debuguear</i> y corregir el proceso de importación 4. Volver a importar y correr los Test.

Tabla 5.2.7.1.1.3: Sistema Detector, calidad de datos

Sistema	Sistema Detector
Subsistema o Tarea	<i>Mapeador</i> – Detector de tiempos verbales
Descripción	Verificar que el detector de tiempos verbales funcione correctamente
Casos a probar	future Continuous
	Future Perfect Continuous
	Future Perfect
	Future Continuous
	Past Perfect Continuous
	Past Perfect
	Past
	Present Continuous

Comportamiento Adaptable de Chatbots Dependiente del Contexto

	Present Perfect Continuos
	Present Perfect
	Simple Future
	casos mezclados
Implementación de los casos	Armar un archivo con 10 oraciones en inglés en "Future Continuous". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Future Perfect Continuous". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Future Perfect". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Future Continuos". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Past Perfect Continuous". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Past Perfect". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Past". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Present Continuous". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Present Perfect Continuos". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Present Perfect". Analizarlas con el detector.
	Armar un archivo con 10 oraciones en inglés en "Simple Future". Analizarlas con el detector.
	Armar un archivo con 20 oraciones en inglés con diferentes tiempos verbales . Analizarlas con el detector.
Tolerancia	10% de margen de error en cada caso excepto en el último, en donde se puede tolerar un 20% de margen de error
Plan de acción	En todos los casos: 1. <i>Debuguear</i> el detector sobre las oraciones que se detectaron mal 2. Verificar que la oración esté bien formada y contenga un solo tiempo verbal. 3. Modificar la <i>regex</i> pertinente o agregar un verbo irregular en la base de datos si faltase. 4. Verificar que el orden en que se aplican las <i>regex</i> sea de las que tienen expresiones más complejas a las que tienen expresiones más simples (incluso que puedan estar contenidas en expresiones anteriores) 4. Volver a correr el detector sobre el <i>set</i> de pruebas.

Tabla 5.2.7.1.1.4: Sistema Detector, tiempos verbales

Sistema	Sistema Detector
Subsistema o Tarea	<i>Mapeador</i> – Detector de frases de interrogación
Descripción	Verificar que el detector de frases de interrogación funcione correctamente
Casos a probar	1. Detectar como positivas las frases que son preguntas
	2. Detectar como negativas las frases que NO son preguntas
Implementación de los casos	1. Crear un archivo de texto con 20 frases que sean preguntas. Verificar que se detectan positivamente
	2. Crear un archivo de texto con 20 frases que NO sean preguntas. Verificar que se detectan negativamente
Tolerancia	1. 10% de margen de error
	2. 0% de margen de error
Plan de acción	En todos los casos: 1. <i>Debuguear</i> el detector sobre las oraciones que se detectan mal 2. Verificar que la oración esté bien formada 3. Modificar los "templates" asociados a la detección de casos. 4. Volver a correr el detector sobre el <i>set</i> de pruebas.

NOTA: No hay una receta para arreglar los errores, pero siempre se tendrá en cuenta que es preferible que una frase se detecte como NO-Interrogación cuando sea de interrogación a que se detecte como de Interrogación cuando no lo es.

Tabla 5.2.7.1.1.5: Sistema Detector, interrogación

Sistema	Sistema Detector
Subsistema o Tarea	<i>Mapeador</i> – Verificar el mapa generado
Descripción	Verificar que el mapa generado sea efectivamente una representación de la frase de entrada según los criterios escogidos.
Casos a probar	1. Frases aleatorias y su correcta conversión en el mapa de entrada
Implementación de los casos	1. Crear un archivo de texto con 10 frases escogidas de forma aleatoria. Convertir las 10 frases en 10 mapas de entrada para la RNA. En el mapa: Verificar que el número que indica si es interrogación o no sea uno o cero según corresponda. Verificar que el número que indica el tiempo verbal tenga el valor correcto. Verificar que cada palabra fue reemplazada por un número y que dicho número esté asociado al tipo de palabra.
Tolerancia	1. El mapa debe de generarse correctamente. Solo se aceptarán valores cero (o que representen valores desconocidos) si y solo si el <i>mapeador</i> no es capaz de detectar correctamente el tipo de la palabra o el tiempo verbal de la oración o si es una interrogación o no. En todos los casos el margen de error debe ser menor al 20%.
Plan de acción	El mapa es tan solo un archivo de texto, con números en punto flotantes separados por comas. La comparación es directa al verificar este archivo contra el de frases línea a línea.

Tabla 5.2.7.1.1.6: Sistema Detector, mapa generador

NOTA: Este plan de prueba podrá repetirse varias veces para cada mapa.

Sistema	Sistema Detector
Subsistema o Tarea	RNA
Descripción	Verificar que la RNA detecta la dependencia o no de contexto en una frase (o en el mapa asociado a dicha frase)
Casos a probar	<ol style="list-style-type: none"> 1. Probar variación de <i>Iteraciones</i> 2. Probar variación de "Learning Rate" y "Momentum" 3. Probar como se modifica el error al agregar una capa oculta 4. Probar contra el <i>set</i> de validación 5. Probar con un mapa diferentes (implica cambiar las neuronas de entrada)
Implementación de los casos	<ol style="list-style-type: none"> 1. Construir un conjunto de pruebas de 100 frases. Dichas frases serán tomadas de diálogos reales de personas de [Loebner, 2006] 2. Convertir las frases en mapas 3. Entrenar la red con parámetros típicos de "momentum" y "learning rate" y bajas iteraciones 4. Ver el error: "RMSE" <ol style="list-style-type: none"> 1. Entrenar la red con parámetros típicos de "momentum" y "learning rate" y el número optimo de iteraciones encontrado previamente 2. Ver el error: "RMSE" <ol style="list-style-type: none"> 1. Tomando los parámetros obtenidos en los casos anteriores agregar una capa oculta con una cantidad de neuronas igual a las neuronas de entrada sobre dos. 2. Entrenar la red 3. Mirar el error "RMSE" <ol style="list-style-type: none"> 1. Obtener la configuración más simple que tenga el menor error. 2. Entrenar la red con el <i>set</i> anterior

Comportamiento Adaptable de Chatbots Dependiente del Contexto

	<p>3. Crear un nuevo <i>set</i> de 100 frases de la misma manera en que se construyó el primero</p> <p>4. Correr la red entrenada contra este <i>set</i>, verificar el error sobre este nuevo <i>set</i>.</p>
	<p>1. En caso de que el resultado obtenido no sea satisfactorio cambiar el mapa. Esto requerirá volver a correr el caso de pruebas anterior (las pruebas del mapa) y este nuevamente. Aunque partiendo de los valores que fueron considerados óptimos.</p>
Tolerancia	<p>La idea es que la RNA pueda detectar si una frase es dependiente del contexto o no con un margen de error menor al 10%. Hay que considerar que se está arrastrando el error del mapa aunque por otro lado se supone que un perceptrón es tolerante a fallas y puede dar un resultado correcto aún cuando no tiene toda la información.</p>
Plan de acción	<p>Si el RMSE no es menor al 10% y si el gráfico está en pendiente descendente aumentar las iteraciones y volver a entrenar la red. Repetir las pruebas</p>
	<p>Si el RMSE es menor dejar un parámetro fijo y seguir modificando el anterior en la misma dirección. Si el RMSE no cambia o crece dejar ese parámetro fijo y modificar el otro con igual criterio. Si el RMSE es mayor dejar un parámetro fijo y seguir modificando el anterior en la dirección opuesta (es decir aumentarlo o disminuirlo). Volver a entrenar la red. Correr los <i>Tests</i>.</p>
	<p>Mirar el error "RMSE". Si no es menor al 10% Disminuir la cantidad de neuronas y Entrenar la red, si mejoró o sigue igual seguir quitando neuronas de la capa oculta. Si empeoró agregar neuronas en la capa oculta. Entrenar la red Correr los <i>Tests</i>.</p>
	<p>Si no se consiguió un error menor o igual (con cierto margen de error) sobre el nuevo <i>set</i> pensar en cambiar el mapa.</p>
	<p>Hacer los <i>tests</i> anteriores sobre el nuevo mapa si fallan, cambiar el mapa.</p>

Tabla 5.2.7.1.1.7: Sistema Detector, verificar RNA

Sistema	Sistema Conversor
Subsistema o Tarea	Sistema de Resolución de Referentes
Descripción	Verificar que el sistema es capaz de identificar referentes en una frase a partir de las anteriores y hacer los reemplazos en la frase del usuario de manera correcta.
Casos a probar	1. Verificar la resolución de referentes en 10 frases de entrada diferentes, asociadas cada una a un contexto.
Implementación de los casos	1. Crear un archivo de texto con 10 pequeños diálogos (solo las frases del usuario), en los cuales la frase final de cada uno hace referencia a algo dicho previamente. Imprimir por pantalla como queda convertida esa última frase luego de que el sistema de resolución de referentes actuó.
Tolerancia	1. 15% de margen de error. (Sabemos que el algoritmo de Hobbs tiene una eficacia de más del 70%)
Plan de acción	<p>1. Verificar la implementación del algoritmo. Buscar otro algoritmo u otra implementación</p> <p>2. reemplazar la vieja implementación por la nueva en el código.</p> <p>3. Volver a probar.</p>

Tabla 5.2.7.1.1.8: Sistema conversor, Resolución de referentes

Sistema	Sistema Conversor
Subsistema o Tarea	Sistema de Conversión de Respuestas "si/no" (frases cerradas)
Descripción	Verificar que el sistema es capaz de identificar cuando el chatbot envió una pregunta susceptible de ser respondida por "si" o por "no" (pregunta cerrada). Y que es capaz de convertir una frase "si" o "no" en una frase completa independiente del contexto.
Casos a probar	1. Probar que el chatbot puede reconocer cuando una pregunta formulada es cerrada.

Comportamiento Adaptable de Chatbots Dependiente del Contexto

	2. Probar que puede convertir de manera eficaz una frase "sí"/"no" en una frase independiente de contexto con el mismo significado.
Implementación de los casos	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 10 casos, cada uno constituido por dos frases, tal que la primera fuerce al chatbot a formular una pregunta. En la mitad de los casos la pregunta debe ser del tipo cerrado y en la otra mitad del tipo abierto. En ambos casos la segunda frase será "sí" o "no" de forma alternada. 2. Ingresar la primer frase, esperar la respuesta del chatbot, ingresar la segunda. 3. Verificar que el chatbot detectó correctamente cuales eran preguntas cerradas y cuales no. <ol style="list-style-type: none"> 2. Tomar los casos de preguntas cerradas, del punto anterior y duplicarlos. Poner como segunda frase "sí" y luego "no". Verificar que el chatbot imprime por pantalla la conversión correcta de cada frase.
Tolerancia	<ol style="list-style-type: none"> 1. Todos los casos deben ser verificados 2. 10 % de margen de error. (Podría llegar a haber casos en los que falta información de contexto para la correcta conversión)
Plan de acción	<ol style="list-style-type: none"> 1. Si la respuesta del chatbot no coincidió con la planeada porque eligió otra respuesta de varias posibles repetir el test hasta que el chatbot muestre la respuesta esperada. Si la respuesta es la esperada pero falló al detectarla como "cerrada"/"abierta" 1. Verificar el caso desfavorable. 2. <i>Debugear</i>. 3. Corregir el <i>template/</i> o expresión regular que detecta a dicha frase. 4. Volver a correr todos los <i>Tests</i> <ol style="list-style-type: none"> 1. <i>Debugear</i> para el caso desfavorable. 2. Rearmar la expresión regular y la forma en que reemplaza la pregunta 3. Volver a correr los <i>Tests</i>.

Tabla 5.2.7.1.1.9: Sistema conversor, Conversión de respuestas a preguntas cerradas

Sistema	Sistema Conversor
Subsistema o Tarea	Sistema de Generación de preguntas pidiendo información de contexto.
Descripción	Verificar que el sistema es capaz de identificar la(s) palabra(s) que hacen referencia a un dato que no está disponible en el contexto y que puede armar una nueva frase a partir de dicha(s) palabra(s). La cual será una pregunta dirigida al usuario en donde se pida la información faltante.
Casos a probar	1. Probar que el chatbot es capaz de armar las preguntas en base a la frase del usuario.
Implementación de los casos	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 10 frases. En donde en cada frase hay una pregunta hacia el chatbot en la cual se referencia a algo. 2. Ingresar las frases en el chatbot como inicio de una conversación nueva. 3. Verificar que el chatbot responde con una pregunta, en cada caso, que hace referencia al dato faltante.
Tolerancia	10 % de margen de error.
Plan de acción	<ol style="list-style-type: none"> 1. Verificar el caso desfavorable. 2. <i>Debugear</i>. 3. Corregir el <i>template/</i> o expresión regular que detectó al referente. 4. Volver a correr todos los <i>Tests</i>

Tabla 5.2.7.1.1.10: Sistema conversor, Respuestas con información de contexto

5.2.7.1.2 Prueba de integración del sistema

Estas pruebas están pensadas para probar el comportamiento del chatbot en su conjunto y para saber si después de la integración de todos los subsistemas estos siguen respondiendo de la misma manera y actúan cada uno en el momento y en los casos en que le corresponden.

Sistemas	Integración del sistema generador del mapa y la red neuronal artificial
Descripción	Verificar que el sistema puede tomar frases e inglés y retornar un valor booleano (1 ó 0) de forma transparente.
Casos a probar	1. Probar que dado una serie de frases en inglés el sistema automáticamente puede convertirlas en un mapa que sirve de entrada a la red neuronal y que esta puede tomarlo y procesarlo correctamente
Implementación de los casos	1. Construir un archivo de texto con 10 frases en inglés 2. Pasárselas al sistema como <i>input</i> . 3. Generar <i>logs</i> del sistema para seguir las fases por las que paso 3. Verificar que la red neuronal artificial se ejecutó por cada frase de entrada y devolvió un valor.
Tolerancia	0 % de margen de error

Tabla 5.2.7.1.2.1: Integración del sistema generador del mapa y la red neuronal artificial

Sistemas	Integración de myAlice con el sistema "detector".
Descripción	Verificar que el chatbot es capaz de distinguir los casos favorables de los no favorables.
Casos a probar	<ol style="list-style-type: none"> 1. Probar que el myAlice es capaz de funcionar exactamente igual que ALICE cuando la frase de entrada está dentro de los casos favorables 2. Verificar que si la frase de entrada no está en el conjunto de casos favorables, myAlice la identificará como tal y utilizará el sistema "detector" sobre la frase de usuario 3. Verificar que el sistema detector ejecutó todas las etapas correctamente 4. Verificar que generó el mapa de la frase correctamente 5. Verificar que la red neuronal generó el valor correcto "dependiente" o "independiente" para la frase de entrada 6. Verificar, solo en el caso de que el patrón de <i>matcheo</i> tenga un valor diferente en el campo "isIndependent" al valor que detectó la red neuronal sobre la frase del usuario, que la frase del usuario y el patrón son distintos. (Hay por ejemplo <i>wildcars</i>)
Implementación de los casos	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 10 frases, las cuales ALICE original pueda de responder correctamente (casos favorables). 2. Hacer que el sistema imprima en un archivo de <i>log</i> el resultado del campo "isIndependent" asociado al patrón que <i>matcheo</i> con la frase de entrada. 3. Ingresar las frases en el chatbot. 4. Verificar que el chatbot respondió como lo haría ALICE original y verificar el resultado del <i>log</i> 1. Construir un archivo de texto con 10 frases, las cuales ALICE original no pueda de responder correctamente (casos no favorables). 2. Hacer que el sistema imprima en un archivo de <i>log</i> el resultado del campo "isIndependent" asociado al patrón que <i>matcheo</i> con la frase de entrada. 3. Verificar que en el archivo de <i>log</i> la frase <i>matcheo</i> con un patrón cuyo campo "isIndependent" es igual a cero (o <i>false</i>) 1. Construir un archivo de texto con 10 frases, las cuales ALICE original no pueda de responder correctamente (casos no favorables). 2. Hacer que el sistema imprima en un archivo de <i>log</i> las distintas fases por las que pasa la detección: "<i>parseo</i> de palabras y reemplazo por tipo", "detección de tiempos verbales", "generación del mapa", etc. 3. Verificar en el archivo de <i>log</i> las etapas por las que pasó. Comprobar que se ejecutaron en orden y correctamente. 1. Construir un archivo de texto con 10 frases, las cuales ALICE original no sea capaz de responder correctamente (casos no favorables). 2. Hacer que el archivo de <i>log</i> imprima el mapa resultado 3. Verificar en el archivo de <i>log</i> que el mapa generado es correcto

Comportamiento Adaptable de Chatbots Dependiente del Contexto

	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 10 frases, las cuales ALICE original no sea capaz de responder correctamente (casos no favorables). 2. Hacer que el archivo de <i>log</i> imprima el resultado de la red neuronal 3. Verificar en el archivo de <i>log</i> que el resultado de la red es correcto.
	<ol style="list-style-type: none"> 1. Tomar los resultados de la pruebas 4 y verificar si hay discrepancias con los valores asociados al patrón de <i>matcheo</i>. 2. En caso de haberlas verificar que el patrón de <i>matcheo</i> y la frase de entrada son diferentes.
Tolerancia	20 % de margen de error sobre los valores del campo "isIndependent"
	20% de margen de error sobre los valores del campo "isIndependent"
	0 % de margen de error
	20% cota de error para el generado del mapa
	10% margen de error para la RNA
	0%, si hay discrepancias que no se pueden explicar los sistemas no están bien integrados.

Tabla 5.2.7.1.2.2: Integración de myAlice con el sistema "detector"

Sistemas	Integración de myAlice con el subsistema "conversor"
Descripción	Verificar que el chatbot es capaz de convertir las frases del usuario cuando estas pertenecen al conjunto de casos no favorables.
Casos a probar	1. Probar que el myAlice es capaz de funcionar exactamente igual que ALICE cuando la frase de entrada está dentro de los casos favorables
	2. Verificar que si la frase de entrada no está en el conjunto de casos favorables, el sistema la convertirá en una frase independiente del contexto por medio del sistema de resolución de referentes
	3. Verificar que si la frase de entrada no está en el conjunto de casos favorables y es una respuesta de tipo "si" o "no" a una pregunta cerrada, el sistema la convertirá en una frase independiente del contexto por medio del sistema de conversión de respuestas del tipo "si/no"
	4. Verificar que si la frase de entrada no está en el conjunto de casos favorables y no hay información de contexto. El sistema preguntará por la información faltante.
Implementación de los casos	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 10 frases, las cuales ALICE original sea capaz de responder correctamente (casos favorables). 2. Ingresar las frases en el chatbot. 3. Verificar que el chatbot respondió como lo haría ALICE original.
	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 5 pequeños diálogos, basados en varias frases de entrada, de las cuales ALICE original no sea capaz de responder correctamente la última la frase. Y que esta frase haga referencia a información del contexto. 2. Hacer que el sistema imprima en un archivo de <i>log</i> el resultado de la frase nueva generada, es decir la frase independiente del contexto. 3. Ingresar las frases en el chatbot. 4. Verificar el resultado del <i>log</i>
	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 5 pequeños diálogos, basados en 2 frases de entrada, de las cuales la primera fuerce a myAlice a que realice una pregunta cerrada y que la segunda sea "sí" o "no" 2. Hacer que el sistema imprima en un archivo de <i>log</i> el resultado de la frase nueva generada, es decir la frase independiente del contexto. 3. Ingresar las frases en el chatbot. 4. Verificar el resultado del <i>log</i>
	<ol style="list-style-type: none"> 1. Construir un archivo de texto con 5 frases dependientes del contexto. 2. Ingresar cada una de las frases en el chatbot como la primera de un dialogo (o sesión) 3. Verificar que la respuesta del chatbot sea una pregunta sobre la información que se pidió o sobre la que se hizo referencia en cada una de las frases.
	20 % de margen de error sobre los valores del campo "isIndependent"
	15% de margen de error sobre la resolución de referentes

	10% de margen de error sobre la conversión de frases
	10% de margen de error sobre la generación correcta de preguntas.

Tabla 5.2.7.1.2.3: Integración de myAlice con el subsistema "conversor"

5.2.7.2 Definición de las pruebas de aceptación del sistema

Para esta etapa final se realizarán dos pruebas diferentes, la primera consistirá en generar un conjunto de pruebas de ejemplo, con casos medidos en donde se sepa de antemano que los diálogos cierran circuitos correctos, por ejemplo, que al convertir una frase de tipo "si/no" en una frase independiente del contexto dicha frase exista como patrón de *matcheo* en el "cerebro" de ALICE.

Este conjunto de pruebas tendrá cinco diálogos diferentes los cuales se ejecutarán sobre el sistema final y sobre ALICE original y se cotejarán los resultados. La idea de esta prueba es poner en evidencia el potencial de la idea de tesis y mostrar como la implementación de las mejoras en el sistema generan mejores respuestas sin cambiar nada en la base de datos.

La segunda prueba se basa en ejecutar sobre el sistema final y sobre ALICE líneas de diálogos reales, al menos dos, que serán obtenidas de las transcripciones de las conversaciones mantenidas entre usuarios y ALICE durante alguno de los certámenes de Loebner.

Al finalizar las pruebas, y cotejar los resultados debería de poder observarse que para las frases para las cuales ALICE respondió bien el nuevo sistema (myAlice) también lo hizo así, pero para aquellas frases para las cuales ALICE respondió erróneamente (en el sentido de que no respondió como lo haría una persona) el nuevo sistema sí respondió correctamente, al menos para algunas de ellas. Si este último supuesto se corrobora entonces el sistema final cumple su objetivo y los supuestos teóricos quedarán validados, al menos en parte.

5.2.8 Aprobación del análisis del sistema de información

En reunión mantenida con el director de tesis, se han aprobado las especificaciones de análisis vertidas en el presente documento.

5.3 Diseño del sistema de información

En esta sección se detallarán todos los documentos de arquitectura del sistema, se especificarán los sistemas auxiliares o de soportes utilizados durante la construcción del sistema principal y se mostrarán los diagramas de clases finales indicando claramente las responsabilidades de cada una de las clases.

5.3.1 Definición de la arquitectura del sistema

En esta sección se especifican los distintos niveles de arquitectura, las excepciones utilizadas para el manejo de errores en el sistema y los estándares y normas de diseño y construcción

5.3.1.1 Definición de niveles de arquitectura

A continuación se muestra un diagrama de despliegue; la base de datos está corriendo en un servidor distinto al servidor en donde corre el sistema. Sin embargo el sistema también podría correr en una *workstation*. El sistema está pensado como una biblioteca, por lo cual está es solo una de varias arquitecturas posibles.

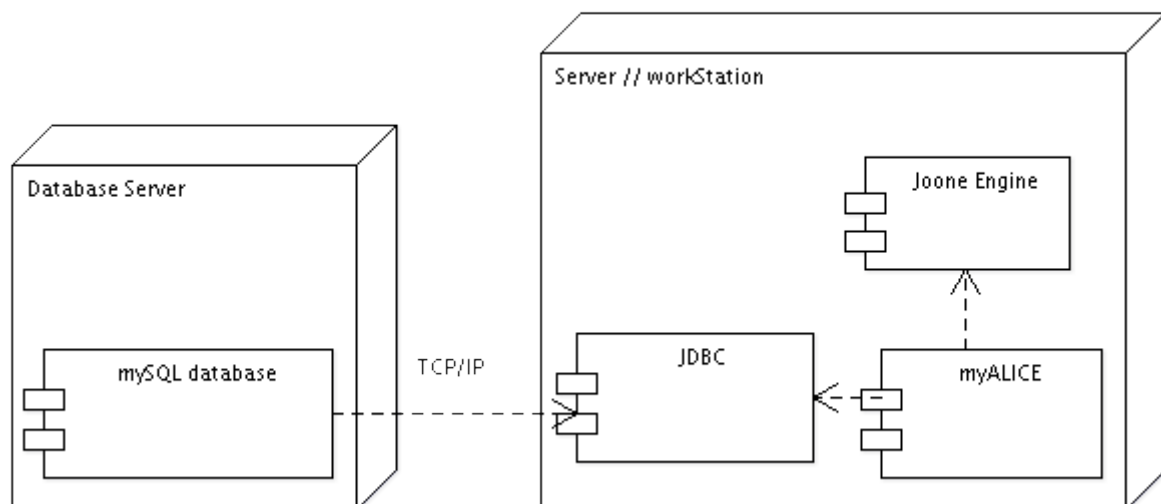


Diagrama 5.3.1: Diagrama de despliegue 01

En la presentación final que tendrá este proyecto se utilizará una consola gráfica de

testing, basada en los componentes gráficos *swing* que ofrece java. Para dicha presentación se utilizará una sola máquina física, que será una *workstation* con GNU-Linux, para dicho escenario el diagrama de despliegue se simplifica como sigue:

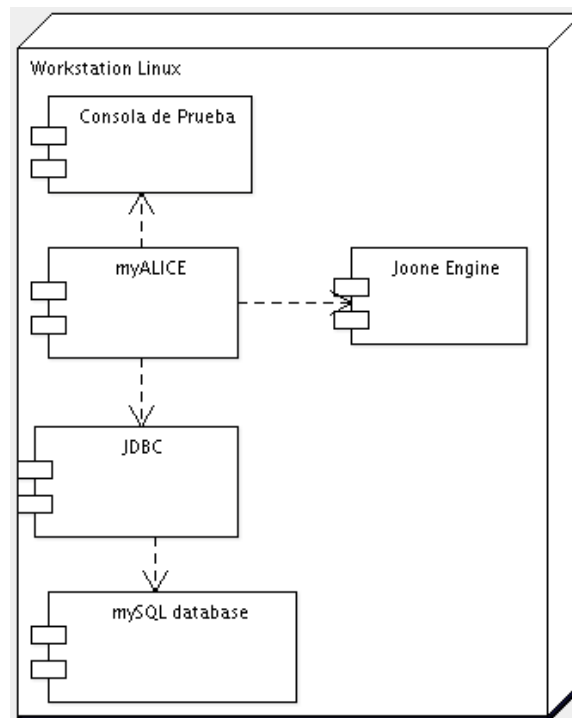


Diagrama 5.3.2: Diagrama de despliegue 02

Desde el punto de vista de la arquitectura del sistema, comenzando por el nivel más bajo se tiene: el motor de la base de datos, la capa de acceso a datos dada por "JDBC" y luego la biblioteca "myAlice". En el mismo nivel está también el conjunto de bibliotecas que conforman el *engine* de "Joone". En el caso particular de la implementación de la consola gráfica, se tiene una capa adicional por sobre la biblioteca de *chatbot* "myAlice" la cual interactúa con esta a través del objeto de la clase "MyAlice" (nombre homónimo a la biblioteca) y en particular contra el método "replyTo()" como se mostró en la fase anterior.

"myAlice" encapsula las llamadas a "JDBC" con una clase llamada "Data", que posee métodos de conexión y ejecución de consultas propios. Esta clase toma los datos de un archivo de propiedades o *properties* según su nomenclatura original en inglés. También lanza sus propias excepciones que encapsulan a las de "JDBC". Sin embargo esta separación no es absoluta ya que hay dos paquetes llamados : "myalice.query" y "detector.query" que contienen cada uno una o más clases que encapsulan una consulta que hace uso de funciones propias de mySql. Esto es así porque el aprovechamiento de las características intrínsecas del motor de mySql es lo que

reemplaza al motor de *pattern matching* que debería implementar un intérprete de AIML ordinario.

5.3.1.2 Especificación de excepciones

La siguiente es una lista de las excepciones del modelo:

ExceptionGrammarParse: Esta es una excepción que puede lanzar la clase "GrammarParse", cuando al invocar al método "parse()" no se le pasan correctamente los argumento de entrada, si el método "doAnalysis()" que se llama internamente devuelve una excepción, en caso de un error no esperado.

ExceptionDatabase: Esta excepción encapsula las excepciones de "JDBC", las utiliza la clase "Data" que encapsula a las funciones de JDBC.

ExceptionQuery: Esta excepción la lanza la clase "Analysis01" , que es la responsable de consultar a la base de datos para obtener las palabras de "Léxico" que coinciden con las de la frase de entrada. Básicamente encapsula las excepciones que lanza la capa de acceso a datos relacionadas con el estado de la conexión o con la ejecución de la consulta, pero da información más específica acerca de cómo falló una consulta.

ExceptionAliceQuery: Esta excepción la lanzan las diferentes clases que conforman el paquete "myalice.query" y encapsulas las excepciones "ExceptionDatabase" y algunas "SQLException". Todas las clases de dicho paquete encapsulan consultas complejas de mySql. Que son las que utiliza "myAlice" para emular el comportamiento de un intérprete de AIML.

ExceptionTimeVerbDetector: Esta excepción es lanzada por la clase "Detector", cuando no puede encontrar el archivo de propiedades donde están las *regex* o cuando los métodos "detectVerbTimeName()" y "detectVerbTimeIndex()" reciben el argumento "sentence" (la frase sobre la cual intentarán detectar el tiempo verbal) de forma incorrecta.

ExceptionMyAlice: Esta excepción es lanzada por la clase "AliceImprovement", "MyAlice" y "Utilities" que es una clase auxiliar del paquete "myalice". Se lanza cuando la clase "AliceImprovement" falla al inicializar la clase "ElizaComments", (que es la clase que finalmente resolverá como formular las preguntas cuando falla la detección del contexto), al generar de forma incorrecta el *input* para la red neuronal o bien para encapsular excepciones.

La clase "MyAlice" lanza esta excepción para encapsular excepciones "ExceptionAliceQuery" y "ExceptionDatabase". En la clase "Utilities" lo lanza el método: "cleanInputPhrase()" el cual se encarga de convertir una frase de entrada del usuario en una frase un poco más prolija, unificando ciertos criterios.

5.3.1.3 Especificación de estándares y normas de diseño y construcción

No se siguió una especificación particular al realizar el sistema como se mencionó anteriormente. Sin embargo se utilizó el documento "The elements of AIML style", que describe el comportamiento de un intérprete de AIML y del chatbot ALICE en particular, como referencia y punto de partida para la construcción de un intérprete de AIML pero que opere sobre una base de datos relacional (mySQL), por las razones antes mencionadas.

La implementación del sistema se realizó íntegramente sobre objetos, respetando el paradigma de la POO. Las clases con iguales responsabilidades se agruparon en paquetes.

Por último hay que destacar que para la creación de la red neuronal se utilizó un perceptron multicapa y el algoritmo de aprendizaje *backpropagation*.

5.3.2 Diseño de la arquitectura de soporte

En esta sección se detalla la arquitectura de aquellos sistemas auxiliares que fueron necesarios construir para generar el sistema principal.

5.3.2.1 Diseño de subsistemas de soporte

Para poder poblar la base de datos con la información necesaria para el correcto funcionamiento del sistema se utilizaron dos aplicaciones auxiliares y una serie de archivos con consultas a la base de datos generados manualmente con la ayuda del editor de texto avanzado de GNU-Linux: Kate. De todas las más importantes de estas herramientas, es la aplicación "Uploader" la que se utilizó para la carga inicial de datos. La construcción de este software se realizó antes de que hubiese siquiera una línea de código de "myAlice".

5.3.2.1.1 Uploader

"Uploader" es una pequeña aplicación auxiliar que se utilizó para importar diferentes

archivos de texto incluyendo archivos AIML a la base de datos. También está realizada en Java y bajo el paradigma de POO. La interfaz de usuario es en modo consola y la forma de utilizar dicha aplicación es la siguiente:

- 1 Realizar la configuración *seteando* constantes en la clase: "Configuracion", allí se deben especificar los datos de conexión a la base de datos, el carácter que se utilizará para delimitar los archivos CSV, etc.
- 2 Compilar
- 3 Desde una consola ejecutar para correrla:

```
"java -jar uploader <opcion> /Path/to/File/filename.ext"
```

Donde <opcion> es un parámetro que indica que tipo de archivo se quiere importar, los valores posibles son:

- line : Archivo de texto plano donde hay una palabra por línea. Ejemplo: diccionarios.
- gra : Archivos de gramática de OpenOffice.
- aiml: Archivos AIML.

A continuación se muestra el diagrama de clases de la aplicación "Uploader" :

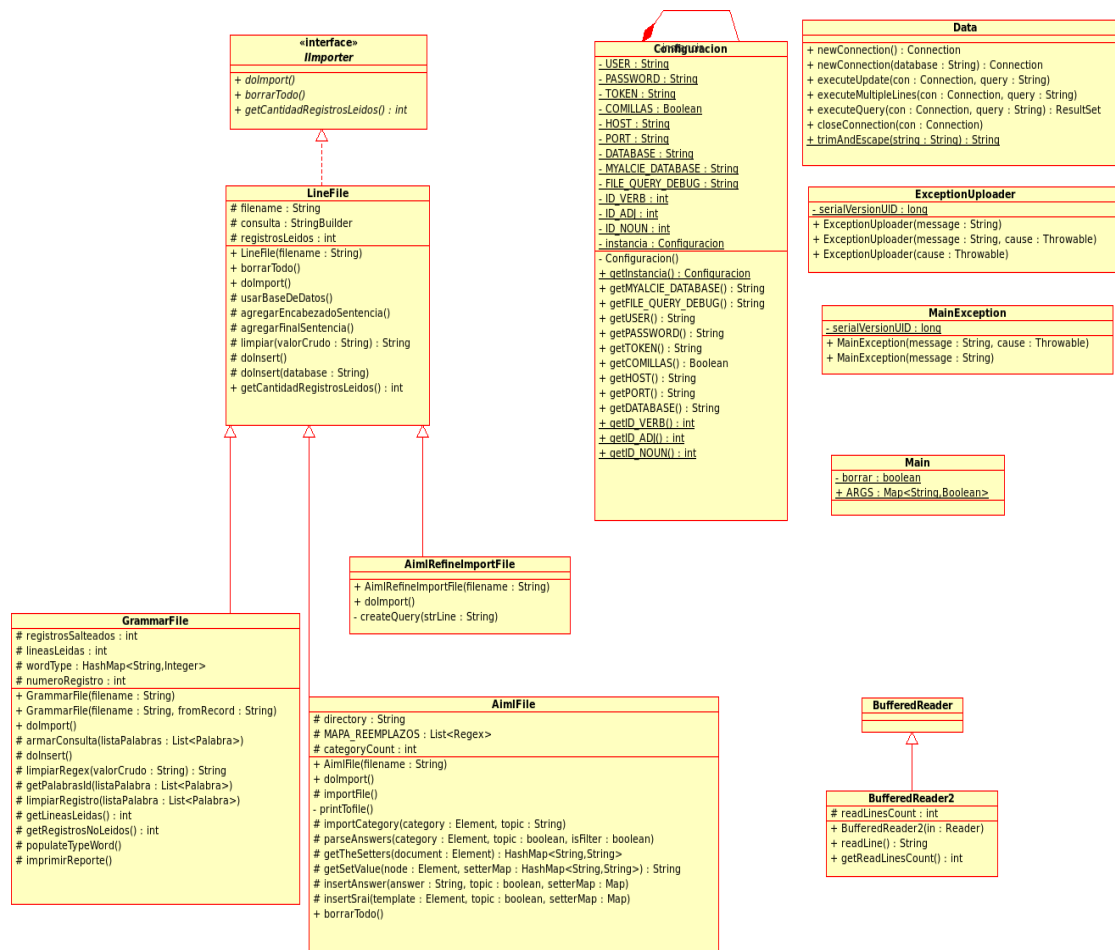


Diagrama 5.3.3: Diagrama de clases del ImportTool

La *interfaz* principal en este diagrama es "Iimporter", la misma define los métodos necesarios para que las clases que la implementen se comporten como "importadores". Tiene solo tres métodos: el primero "dolImport()" es el método que va a llamar la función "main()", este método tiene que saber qué hacer para importar un archivo. El segundo método es "borrarTodo()" este método sabe cómo deshacer todo lo que el método "dolImport()" realizó. El mismo será llamado cada vez que las pruebas fallen y haya que volver a importar todo desde cero. El tercer método se llama "getCantidadRegistrosLeidos()" este método es importante porque sabe cuántos registros (pueden ser líneas, *tags*, etc.) fueron leídos del archivo original y sirve para poder ejecutar las pruebas y validar que en efecto se leyeron todos los registros que se querían leer.

La clase "Data" es un clase que encapsula los métodos de JDBC para conectarse y ejecutar consultas a la base de datos.

La clase "BufferedReader2" es una clase que hereda de la clase "BufferedReader" de java y

sobrescribe el método "readLine()" para que este método cuente la cantidad de líneas leídas. Este dato se utilizará para la implementación del método "getCantidadRegistrosLeidos()."

La clase "Main" es quien tiene la lógica de tomar los parámetros de entrada, evaluarlos y llamar a la clase "Importadora" correspondiente.

5.3.2.1.2 UpdateDatabase

El siguiente programa auxiliar, consta solo de una clase Java, la cual hace uso de los subsistemas: "phraseToInput" y "perceptron".

```
ar::uba::fi::console::ProcessAndupdatedb
+ MYSQL_OUTPUT_FILE : String
+ PROPERTIES_FILE : String
+ SELECT : String
+ setProperties(filename : String) : Properties
+ main(args : String[])
```

Diagrama 5.3.4: Diagrama de clases de UpdateDatabase

El programa corre desde consola y lo que hace de manera secuencial es:

1. Tomar todos los "patterns" de la base de datos (los mismos "patterns", de las "category" de AIML, que fueron previamente importados)
2. A través del método "phraseToInput()" los convierte en el mapa de entrada de la red neuronal (o *input*).
3. Aplica la red neuronal, es decir el subsistema "perceptron" sobre cada uno de los *inputs* generados previamente.
4. Con el resultado actualiza el campo "can_answer" asociado al "pattern" en la base de datos. Este es el campo, que en la etapa de análisis se lo llamó "isIndependent"

5.3.2.1.3 Código reutilizado

En el caso del programa "UpdateDatabase" ya se contaba con los principales subsistemas contruidos, con lo cual no hubo código que pudiera reutilizarse en cambio del primer programa "Uploader" se reutilizó la clase "Data".

5.3.3 Diseño de casos de uso reales

En esta sección se detalla la forma en la cual se adaptó el documento de casos de uso para que el flujo allí descrito se refleje en a través del estado y comportamiento de las diversas clases.

5.3.3.1 Diseño de la realización de los casos de uso

A continuación se muestra como se adaptó la especificación del caso de uso a una serie de tareas repartidas en las diferentes clases del sistema final y como estas clases colaboran entre sí. Se muestran tres diagramas de colaboración, los cuales cubren la mayor parte del funcionamiento del sistema e ilustran de una manera bastante precisa el flujo de trabajo.

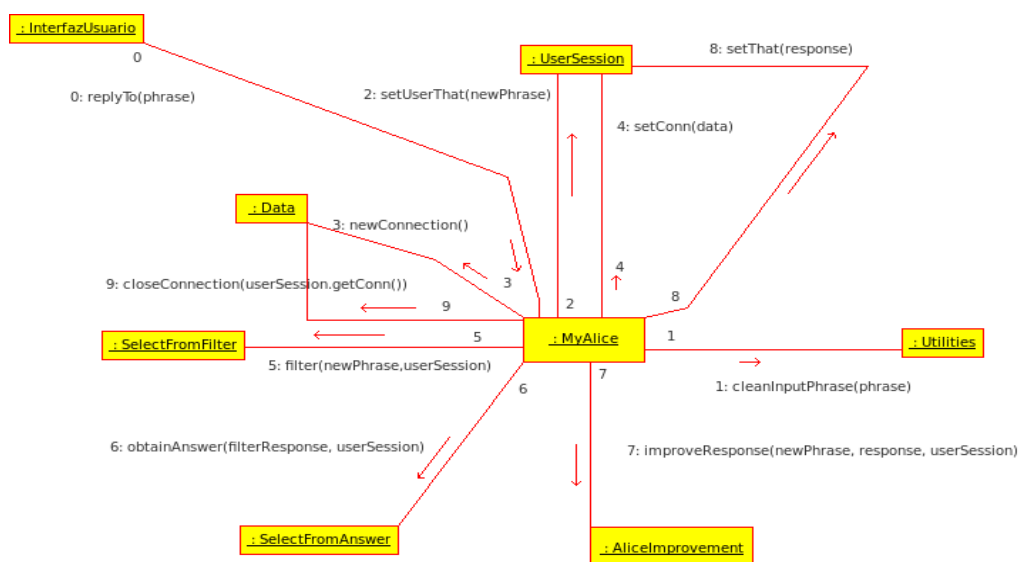


Diagrama 5.3.5: Diagrama de colaboración principal

Este diagrama describe el flujo principal del trabajo, y los principales métodos que usan las clases para comunicarse entre sí.

El paso inicial que sirve como disparador de toda esta secuencia es la frase que el usuario ingresa en el sistema (el chatbot). En este caso dicho paso está representado por el objeto de la clase "InterfazUsuario" ya que el sistema que se está analizando no es una implementación final sino una biblioteca de clases. Quien lo implemente deberá llamar a la función "replyTo()" de "MyAlice" y pasarle la frase de entrada y un objeto "UserSession", que será único para cada usuario.

1: "MyAlice" llama al método "cleanInputPhrase()" de la clase estática "Utilities", el cual "normaliza" la frase de entrada aplicando una serie de reemplazos y expresiones regulares sobre el *String*.

2: Se *setea* la frase actual "normalizada" en el objeto "UserSession" como lo que "fue dicho por el usuario". El campo "userThat"

3: Se crea una conexión a la base de datos

4: Se pasa la referencia de la conexión creada anteriormente al objeto "UserSession"

5: Se invoca al método "filter()", el cual buscará en la base de datos si hay algún patrón de coincidencia con la frase del usuario y en caso afirmativo devolverá la frase reformulada.

6: La frase reformulada (o la original si es que "filter()" devolvió null) se pasa como parámetro del método "obtainAnswer()" de la clase "SelectFromAnswer". Este método hace una consulta a la base de datos, y obtiene la respuesta asociada tal y como lo haría ALICE original.

7: En este punto es donde la implementación de ALICE sobre mySql se diferencia de la implementación tradicional de ALICE sobre AIML. Una vez completado el ciclo original para la obtención de una respuesta se aplican las mejoras, que se proponen en este estudio. Se invoca al método "improveResponse()" de la clase "AlicImprovements". Lo que realiza dicho método se verá en los dos siguientes diagramas de colaboración.

8: "MyAlice" ya obtuvo una respuesta, antes de responder *setea* el atributo "that" en el objeto "UserSession" con la respuesta original que daría ALICE sin las mejoras.

9: Se cierra la conexión a la base de datos.

10: Finalmente el método "replyTo()" devuelve la respuesta encontrada, la cual puede ser o bien una respuesta exactamente igual a la que daría ALICE tradicional o bien una respuesta distinta que se obtuvo de aplicar las mejoras propuestas para la detección de contexto.

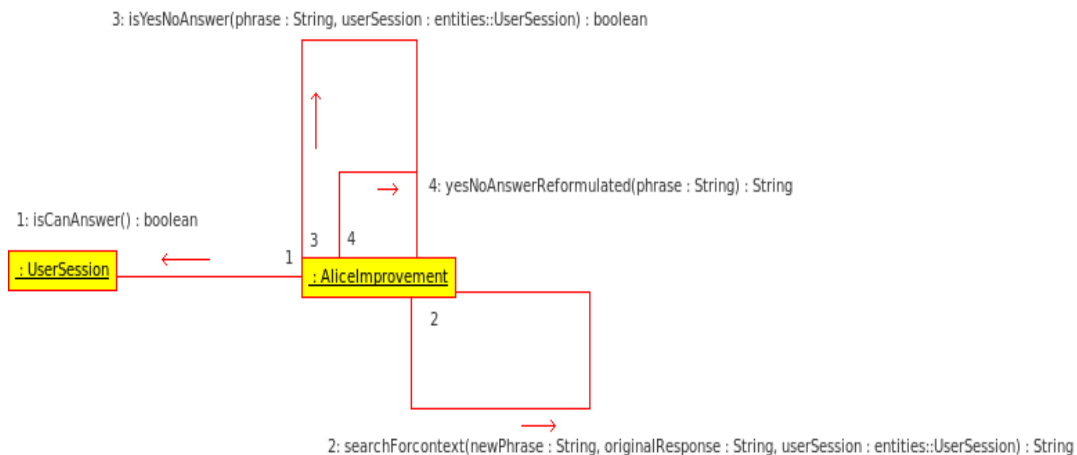


Diagrama 5.3.6: Diagrama de colaboración, flujo de "Improvements" caso 1

El diagrama 5.3.6 muestra uno de los escenarios posibles para el flujo de colaboración dentro de la función "improveResponse()" del objeto "AlicImprovements", que en el diagrama 5.3.5 aparece identificada con el paso número 7.

1: La función "improveResponse()" primero verificará el campo "can_answer" ("isCanAnswer()") asociado al patrón contra el cual fue comparada la frase de entrada del usuario, si dicho campo es verdadero quiere decir que la frase pertenece al conjunto de casos favorables; aquellos que ALICE original puede resolver de manera correcta, entonces la función devolverá la respuesta que le fue pasada por parámetro ("newPhrase") sin modificaciones. En cambio si el campo es falso, (probablemente porque el patrón es demasiado genérico, tal vez sea un simple *wildcard*: "%") invocará a una función propia llamada "searchForContext()"

2: "SearchForContext()" es la función que se encargará de realizar la "interpretación pragmática" invocando a los subsistemas: de resolución de referentes, de reformulación de frases "sí/no" y de petición de información faltante. Aunque finalmente no se implementó el subsistema de resolución de referentes, dicho subsistema debería de ser invocado aquí.

3: La función "searchForContext()" lo primero que hará será verificar si está ante una respuesta de tipo sí/no ("isYesNoAnswer()"), a una pregunta cerrada realizada previamente. Para ello, primero verificará si la frase de entrada del usuario coincide con algún valor en una lista de "posibles respuestas a frases cerradas": "yes","no","sure","of course", "ok". En caso de coincidencia verificará luego que el campo "that" empiece con alguna palabra, de una lista de

"comienzos posibles de preguntas cerradas":

"do", "does", "did", "are", "is", "can", "has", "have", "was", "were".

4: En caso de que la función "isYesNoAnswer()" devuelva verdadero, lo siguiente será reformular la frase de entrada del usuario, que es dependiente del contexto y convertirla en una frase independiente del contexto mediante la función "yesNoAnswerReformulated()".

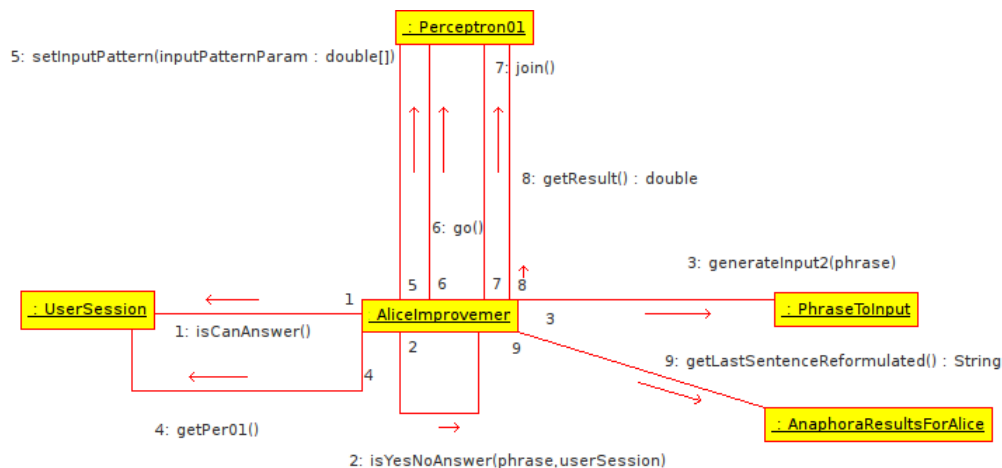


Diagrama 5.3.7: Diagrama de colaboración, flujo de "Improvements" caso 2

El diagrama 5.3.7 muestra otro de los escenarios posibles para el flujo de colaboración dentro de la función "improveResponse()" del objeto "AlicelImprovements", que en el diagrama 5.3.5 aparece identificada con el paso número 7.

1: El flujo comienza igual que el anterior, invocando a la función "isCanAnswer()", para verificar si el campo "can_answer" es verdadero.

2: Si no lo es se invoca a la función "isYesNoAnswer()" para identificar si se respondió a una pregunta cerrada.

3: Si la frase de entrada no es una respuesta a una pregunta cerrada, se procederá a verificar si dicha frase es independiente del contexto o no. Para ello primeramente se invocará a la función "generateInput()" del objeto "PhraseToInput" para convertir la frase del usuario en el mapa de entrada de la red neuronal.

4: Se obtiene al objeto "Perceptron01" que existe dentro de una sesión de usuario,

El diagrama 5.3.8 muestra otro de los escenarios posibles para el flujo de colaboración dentro de la función "improveResponse()" del objeto "AliceImprovements", que en el diagrama 5.3.5 aparece identificada con el paso número 7.

1: a 8: Ídem anterior.

9: El sistema de resolución de referentes no encuentra referencias anafóricas en la frase o bien no logra resolverlas.

10: El subsistema de contingencia: "petición de información faltante" es invocado para formular una pregunta que pedirá al usuario más información. Dicho subsistema se implementó como una versión ligera de Eliza.

5.3.3.2 Revisión de la interfaz de usuario

Como se mencionó anteriormente, para fines de probar la biblioteca y realizar la presentación se construyó una pequeña consola de prueba, con una interfaz gráfica la cual se puede ver en la figura 5.3.1:

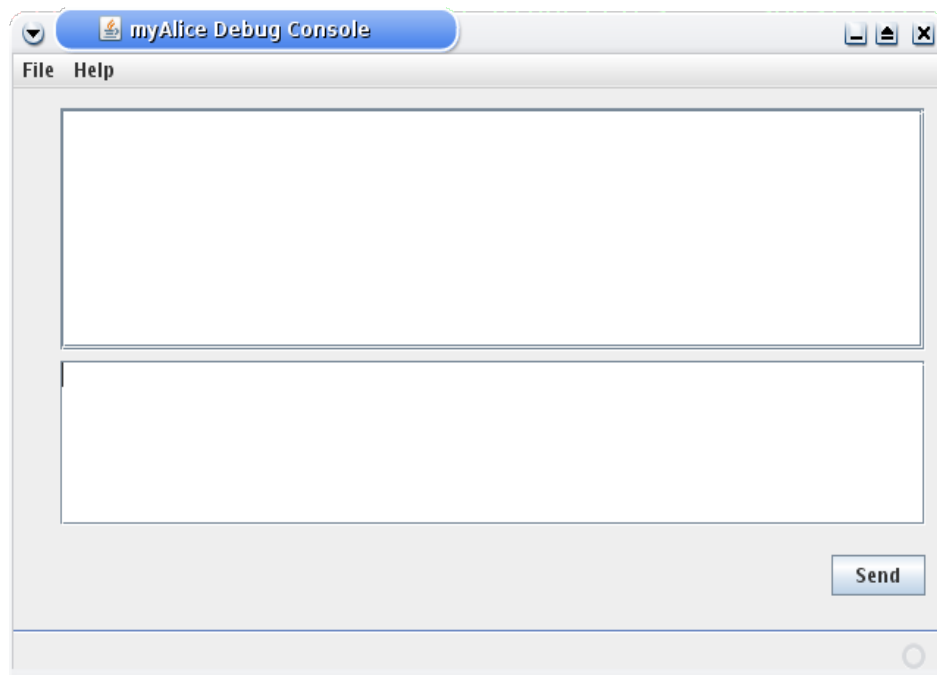


Figura: 5.3.1

La misma fue realizada con la biblioteca gráfica de Java: Swing.

La consola tiene cuatro elementos gráficos principales: menú, área del dialogo, área de entrada de texto y el botón "Send"

Menú: Tiene dos opciones: "File" y "Help", cada submenú tiene una sola opción. El menú "File" tiene la opción "Exit" para finalizar la aplicación. El menú "Help" tiene la opción "About.." que solo muestra un dialogo con información de la versión.

Área del dialogo: es el rectángulo blanco superior, allí se va mostrando en líneas se paradas lo que escribió el usuario y la respuesta del chatbot. Como se muestra en la figura 5.3.2:

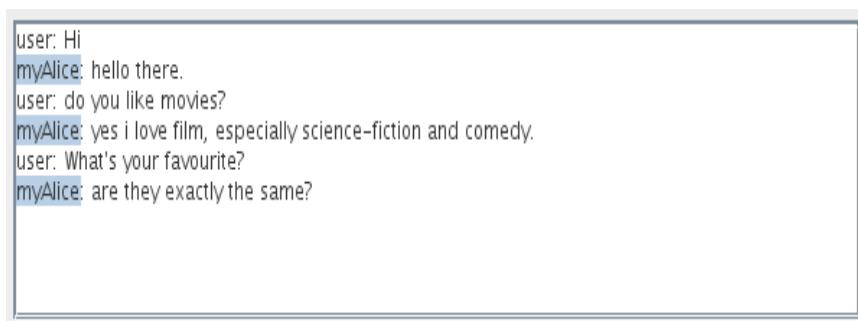


Figura: 5.3.2

Área de entrada de texto: Es el rectángulo blanco inferior donde el usuario puede escribir libremente.

Botón "Send": Una vez que el usuario escribe la frase, debe presionar este botón para que el sistema la procese, básicamente lo que hará el botón "Send" es llamar al método "replyTo()" de "myAlice" enviándole la frase recién ingresada. El mismo comportamiento se obtiene al presionar la tecla *Enter*

5.3.4 Diseño de clases

En esta sección se describen las diferentes clases que componen al sistema final, se detallan en diversos diagramas de clases los métodos, atributos, relaciones de herencia, composición y agregación de las mismas.

5.3.4.1 Identificación de clases adicionales

En el diagrama de clases 5.3.9 se muestra el resultado final. Hay pocas variaciones respecto al diagrama de clases realizado durante la fase de diseño, aunque aquí se muestran todas las clases auxiliares utilizadas, excepto las clases de excepciones. Por un lado una sola clase heredera de "BasicNet", "Perceptron01" quedó en el código, a pesar de que durante la fase de pruebas hubo al menos dos y ambas fueron modificadas varias veces. Además aparece una clase nueva llamada "Som", la cual representa a una red neuronal de tipo SOM, Kohonen. Esta se utilizó en la fase de pruebas pero luego se descartó su utilización.

Aparecen también las clases asociadas a "AnaphoraResultsForAlice", dichas clases encapsulan el comportamiento del algoritmo de resolución de anáforas y del parser de procesamiento de lenguaje natural que este utiliza.

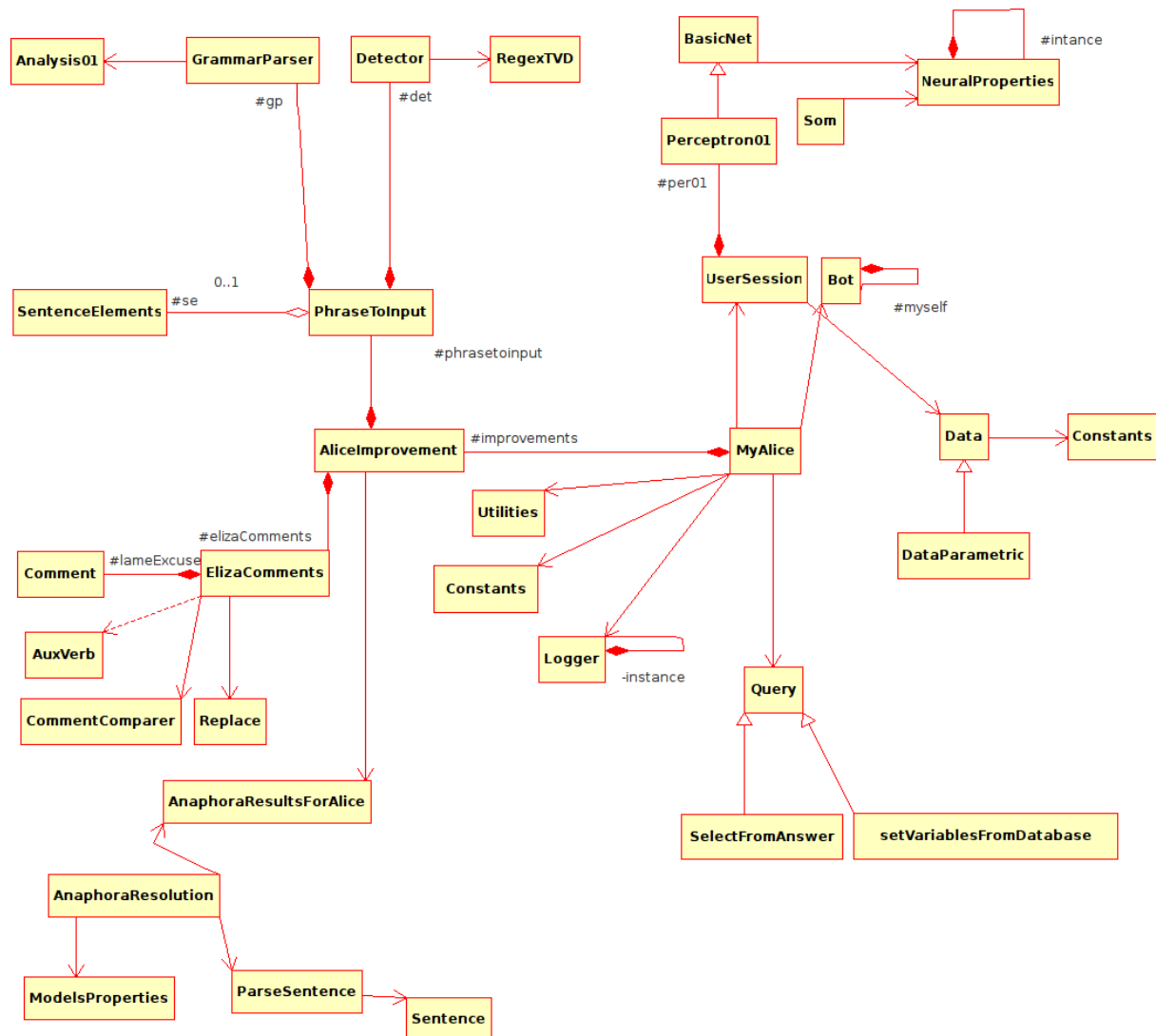


Diagrama 5.3.9: Clases definitivo

La clase "Data" y su heredera "DataParametric" sirven para encapsular métodos de "JDBC" de acceso a datos.

La clase "Query" y sus herederas y la clase "Analysis01" encapsulan complejas consultas a la base de datos. Las que heredan de "Query" encapsulan consultas a la base de "myAlice" mientras que la clase "Analysis01" encapsula una consulta a la base de datos de "Léxico". Básicamente "Analysis01" utiliza todo el poder de mySql para descomponer una frase en las palabras que la componen (entendiendo palabra como se dijo anteriormente como una o más cadenas de caracteres que pueden estar o no separadas por espacio.) Por ello dicha clase es utilizada por "GrammarParser"

Aparece una clase auxiliar vinculada a "Detector" (encargada de detectar tiempos

verbales), llamada "RegexTVD" que es una entidad para las expresiones regulares definidas para la detección de tiempos verbales.

Aparecen dos clases auxiliares "Utilities" con métodos estáticos, y "Logger" que es un *singleton*, en el diagrama se las ve vinculadas con "MyAlice" pero en verdad no es esta la única clase que las utiliza, sin embargo, sí es la principal.

La clase "AlicImprovements" no contiene una clase "Conversor" como se había mostrado en los esquemas previos en donde dicha clase a su vez implementaba tres subsistemas: "de resolución de referentes", "de reformulación de frases sí/no" y "de petición de información faltante ". La antes llamada clase "Conversor" es la equivalente a la actual clase: "AnaphoraResultsForAlice" ya que implementa el subsistema de resolución de referentes. Respecto al subsistema de petición de información faltante fue implementado con una versión ligera de Eliza, que en el diagrama de clases se la puede ver como "ElizaComments", en este caso se respetó el código original de las clases y su diagrama de paquetes no quedó dentro de la jerarquía de "myAlice". Por último el subsistema de reformulación de frases sí/no, se resolvió con dos métodos simples y dichos métodos se implementaron en la clase "AlicImprovements".

5.3.4.2 Identificación de atributos y operaciones de las clases

A continuación se muestran diferentes secciones del diagrama de clases anterior, pero teniendo en cuenta todas las operaciones y atributos de las clases. Los siguientes diagramas fueron contruidos de manera automática utilizando las clases finales, por lo cual resultan exhaustivos, repletos y minuciosos, quizás en demasía, pero con la ventaja de que son fieles al sistema real.

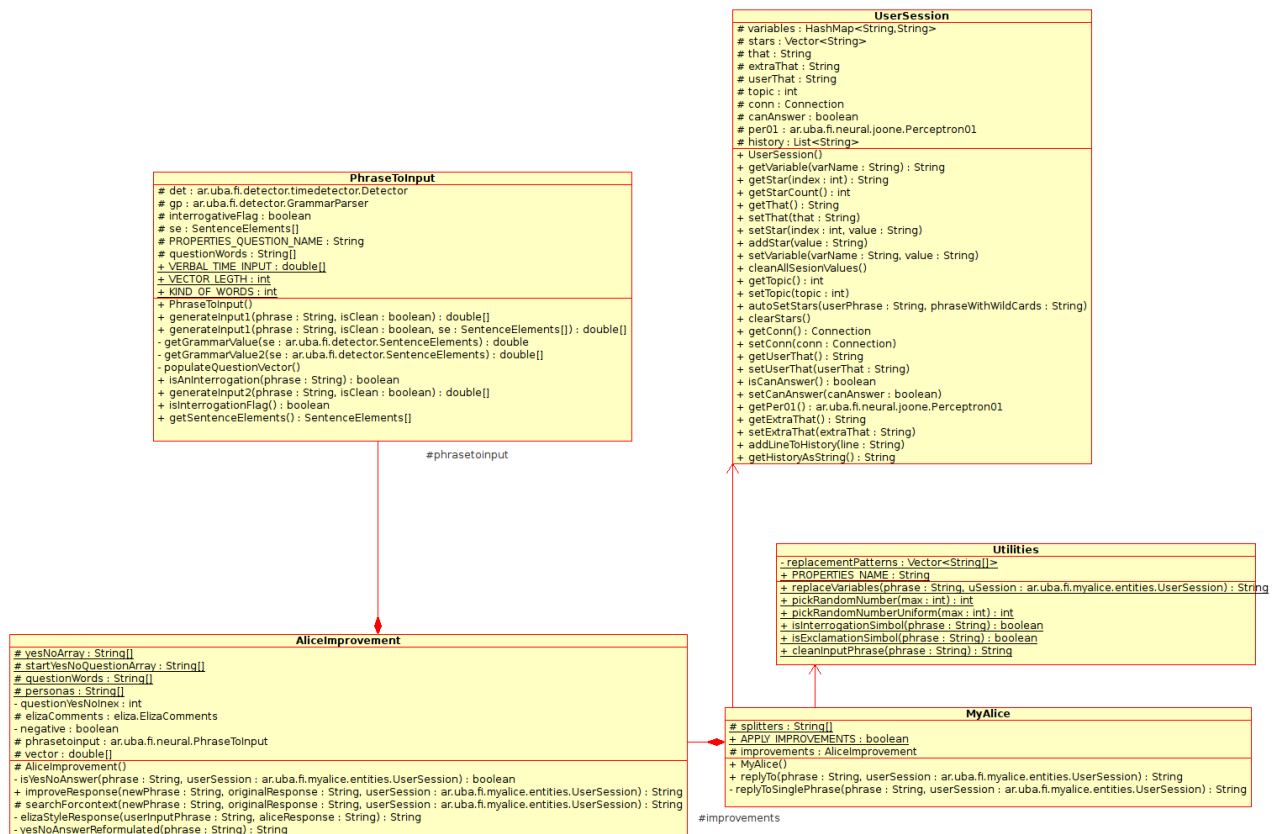


Diagrama 5.3.10: Clases, MyAlice, AliceImprovements, Utilities, UserSession, PhraseToInput

MyAlice: Contiene un *array* de *strings*, "splitters" donde cada elemento es un campo separador de oraciones, por ejemplo: ",", ".", ";", etc. El método público "replyTo()" separará la entrada del usuario en varias sentencias, según el criterio dado por los valores del *array* anterior y cada sentencia individual será procesada por el método privado "replyToSinglePhrase()". La respuesta final del chatbot, es decir el valor de retorno de la función "replyTo()" será la concatenación de las sucesivas respuestas devueltas por el método "replyToSinglePhrase()"

AliceImprovements: Como se mencionó anteriormente su método principal es "improveResponse()", el cual invoca al método protegido "searchForContext()" que se encarga de gestionar a los distintos subsistemas de interpretación pragmática implementados: "de reformulación de frases sí/no" y "de petición de información faltante ". Los métodos privados "isYesNoAnswer" y "yesNoAnswerReformulated" utilizan los *arrays* de *strings*: "yesNoArray", "startYesNoQuestionArray", "questionWord" y "personas" .

El método privado "elizaStyleResponse()" invoca al subsistema de petición faltante,

implementado a través de su clase principal: "ElizaComments"

Utilities: Contiene solo seis métodos estáticos, cada uno realiza una tarea específica y no hay entre ellos ningún tipo de agrupación de responsabilidades, esta clase es útil debido a que dichos métodos pueden ser solicitados por diferentes clases en diferentes partes del flujo de trabajo.

"replaceVariables()": Este método reemplaza en una respuesta obtenida de la base de datos los meta símbolos que representan el contenido de una variable de sesión (contenidas en la clase "UserSession") o bien de una constante del chatbot (contenidas en la clase "Bot") por dicha variable. Por ejemplo, si al realizar la consulta sobre la base de datos de "myAlice" para obtener el resultado a una frase de entrada de usuario, se obtuvo algo como esto: "He is @var["he"]." este método reemplazará "@var["he"]" por el contenido de la variable de sesión "he", contenida en "UserSession". Este mecanismo reproduce el comportamiento de un intérprete de AIML, solo que las variables no son indicadas en *tags* de XML sino con estos meta símbolos, en AIML el ejemplo anterior hubiese sido: "He is <get name="he"/>."

"pickRandomNumber()": Esta clase devuelve un número de manera aleatoria según una distribución Normal centrada en uno, con un desvío estándar igual a la mitad del valor máximo indicado por parámetro. Si el número obtenido es mayor que el máximo devuelve el resto de la división entre este y el máximo. Esta función es útil para que el chatbot escoja una respuesta de varias posibles, pero siempre dándole mayor prioridad a las primeras que devuelve la base de datos ya que estas son las que mejor se ajustan, en teoría.

"pickRandomNumberUniform()": Ídem anterior pero sobre una distribución uniforme. Esta función finalmente no se utilizó.

CleanInputPhrase(): Este método limpia o normaliza una frase de entrada de usuario. Esto significa que realizará un montón de reemplazos, de ser posible, para que la frase se ajuste mejor a los patrones que hay en la base de datos. Este comportamiento es idéntico al que realiza un intérprete de AIML estándar. Entre otras cosas, eliminará símbolos de interrogación y exclamación, *emoticones* y llevará las contracciones del idioma inglés a su forma expandida, por ejemplo "can't" lo transformará en "can not".

PhraseToInput: Esta clase tiene la responsabilidad de convertir la frase de entrada del usuario en el *input* que utilizará la red neuronal. Sus métodos públicos principales son dos: "generateInput1()" y "generateInput2()", la diferencia entre ellos es que ambos devuelven mapas diferentes. Aquí hay que entender que un mapa no es más que un *array* de *double*. Por supuesto que la red neuronal, representada por un objeto de la clase Perceptron01, debe de estar preparada para utilizar dicho mapa, básicamente cada elemento *double* de *array* corresponde a una neurona de entrada, con lo

cual el Perceptrón debe de tener dicha cantidad de neuronas de entrada.

Los atributos de esta clase son:

"det" : Objeto de la clase "Detector", cuya responsabilidad es detectar tiempos verbales en la frase de entrada.

"gp": Objeto de la clase "GrammarParse", cuya responsabilidad es analizar la frase de entrada del usuario y descomponer cada palabra en su tipo.

"interrogativeFlag" : atributo de tipo *boolean*, indica si la frase de entrada del usuario es una interrogación, si es así se agregará esta información a la hora de generar el mapa

"se": *Array* de objetos de tipo "SentenceElement", es el resultado de un análisis gramatical rápido de la frase del usuario. Reemplazando las palabras por su tipo.

"questionWords": *Array* de *strings*, donde cada valor es un pronombre interrogativo o un verbo de los que comúnmente se utilizan en inglés al inicio de una oración para denotar interrogación.

UserSession: Esta clase no tiene ninguna lógica, y existirá un objeto de dicha clase por cada sesión de usuario existente. Sus métodos son solo *getters* y *setters* para sus atributos, que son los siguientes:

"variables": *HashMap*, donde el *key* es el nombre de la variable. Ambos campos *key* y *value* son *Strings*.

"stars": Vector de *Strings*. En cada frase de entrada, si el patón de *matcheo* tenía *wildcards* (llamados *starts* por AIML) la porción de texto que coincidió con dicho *wildcard* es almacenada en la posición del vector igual al número de *wildcard*.

"that": Campo *String*, almacena la respuesta anterior de "myAlice" (que generó el sistema "myAlice" sin las mejoras, aunque no sea la respuesta final del sistema).

"extraThat": Campo *String*, almacena la respuesta anterior de "myAlice", generada a través de la clase "AliceImprovements".

"userThat": Frase de entrada anterior, dada por el usuario.

"topic": Campo entero. Almacena el *ID* del tópico de conversación actual, este comportamiento es idéntico al de un intérprete AIML tradicional, solo que en lugar de almacenar el ID almacenaba el texto que identificaba al *topic*.

"conn": Campo "Connection", cada sesión de usuario tiene una conexión a la base de datos. Este tipo de comportamiento habría que mejorarlo en un futuro ya que no es extensible para miles de conexiones.

"canAnswer": Campo "can_answer" asociado a una frase de entrada del usuario.

"per01": Perceptron01, cada usuario tiene un Perceptrón asociado a su sesión, ya que este

corre en un *thread* paralelo y puede demorar su ejecución.

"history": Lista de *Strings*, contiene un historial de las últimas 10 líneas de dialogo. Esta información será utilizada por el algoritmo de resolución de referentes para encontrar las referencias anafóricas correspondientes.

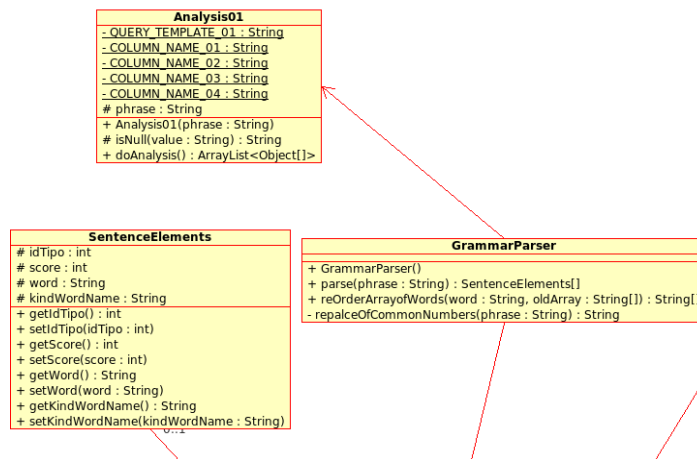


Diagrama 5.3.11: Clases, *Analysis01*, *SentenceElements*, *GrammarParser*

GrammarParser: El principal método de esta clase es "parse()", el cual recibe un *String*, que es la frase de entrada del usuario, después de que se ejecutó sobre este el método "cleanInputPhrase()". Este método devuelve un *array* de "SentenceElements". La parte principal de este método, que es la identificación de palabras en la frase y la asignación del tipo de palabra correspondiente lo realizará la base de datos mySql través de la clase "Analysis01".

Analysis01: Esta clase tiene como atributos, constantes con partes de la consulta SQL que armará para que la base de datos devuelva una lista de palabras identificadas en la sentencia con su tipo. Su método principal es "doAnalysis()" el cual arma la consulta con los datos de la frase de entrada, la ejecuta y devuelve un "ArrayList" de un *array* de Objetos (*Object*), donde cada *array* de objetos representa a una tupla devuelta por la consulta.

SentenceElements: Esta es una clase sin lógica alguna. Sus métodos son solo *getters* y *setters* para sus atributos. Representa a una porción o elemento constitutivo de una frase.

Sus atributos son:

"idTipo": el identificador univoco de un tipo de palabra.

"Score": el puntaje, de un tipo de palabra. Esto se utilizó para armar luego el mapa de entrada de la red neuronal. El *score* indica de alguna manera cuan relevante era la palabra dentro de una frase.

"Word": *String*. Es una palabra (según el significado dado en este estudio.)

"kindWordName": Es el nombre del tipo de palabra, ejemplo "noun","verb" este campo fue utilizado solo con fines de prueba.

Al convertir una frase en un *array* de "SentenceElements", que es lo que hace el método "parse()" se está descomponiendo dicha frase en una secuencia de palabras bien identificadas, con su tipo de palabra y su relevancia dentro de la frase, conservando el orden dado por el *array* es posible rearmar la frase.

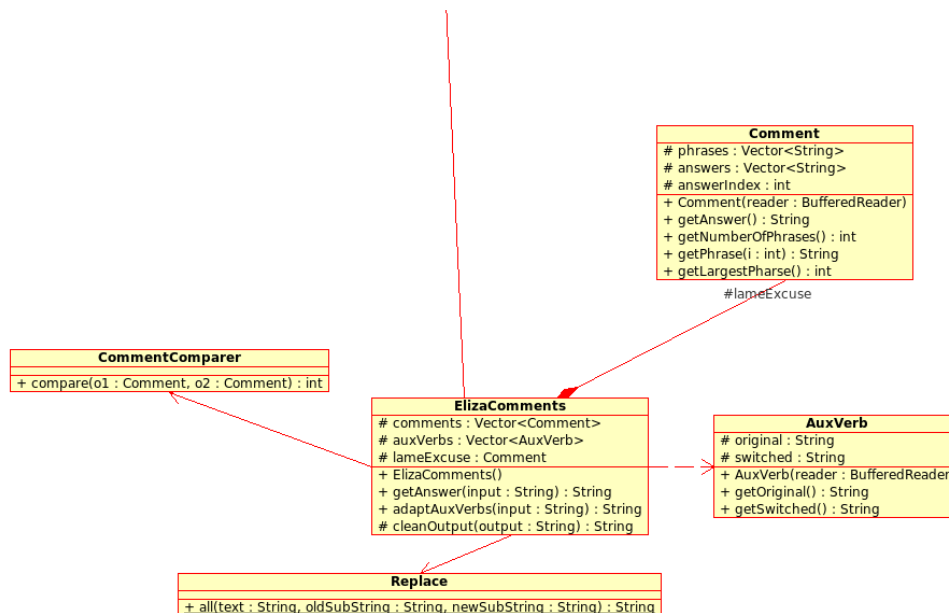


Diagrama 5.3.12: Clases, *ElizaComments*, *AuxVerb*, *Replace*, *CommentComparer*, *Comment*

En el diagrama 5.3.12 se muestran las clases que pertenecen al subsistema "Eliza". La clase principal es "ElizaComments".

ElizaComments: Esta clase es la encargada de escoger una respuesta para una frase de entrada dada. Se alimenta del archivo de texto: "eliza.dat", en el cual cada entrada está separada de otra por una línea en blanco y tienen la forma que se muestra a continuación:

"PHRASE
is that
it is

it's
ANSWER
.what is *?"

Lo que viene debajo del campo "PHRASE" son porciones de texto que deben de coincidir en parte con la frase de entrada. Lo que está debajo de ANSWER son posibles respuestas. Lo que se hizo fue reemplazar las palabras claves de Eliza, que en el sentido tradicional de este chatbot son palabras asociadas a un psicólogo, o psicoterapeuta como "madre", "familia",

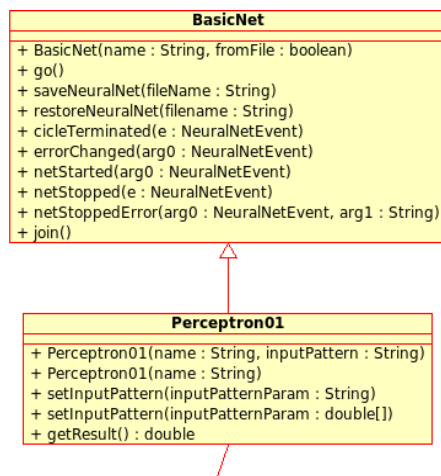


Diagrama 5.3.13: Clases *BasicNet*, *Perceptron01*

"problemas" por frases con referentes, como las del ejemplo "is that", etc. (Hay que tener en cuenta que este subsistema solo se ejecutará cuando falle el sistema de resolución de contexto.) Y también las frases de respuesta, en el ejemplo "what is *?", donde el carácter * será reemplazado por la primera palabra después del patrón de coincidencia.

Basic Net: Encapsula diferentes componentes propios de la biblioteca "Joone" como "NeuralNet", "Monitor", "StreamInputSynapse", "MemoryInputSynapse", "TeachingSynapse", "FileInputSynapse", "FileOutputSynapse" necesarios para la implementación de redes neuronales como las que se utilizaron en este trabajo. Dichos componentes son inicializados en el constructor de la clase. Los diferentes métodos públicos que se muestran en la clase son aquellos necesarios para que el sistema trabaje con la red neuronal de la forma más transparente posible.

Como se mostró en el Diagrama 5.3.7, los métodos con los que interactúa el sistema de forma ordinaria son:

"go()" : Hace que el *thread* en el cual funciona la red neuronal corra y clasifique el patrón ingresado previamente.

"join():" El cual básicamente invoca al método "join()" del componente "NeuralNet", y su responsabilidad es hacer que el hilo principal espere a que la red neuronal finalice su ejecución.

Los métodos:

"saveNeuralNet()" y "restoreNeuralNet()" sirven para guardar en un archivo los pesos y configuraciones de la red. En particular como el entrenamiento de la red se realizó en la fase de pruebas, durante la ejecución ordinaria del chatbot solo se utiliza "restoreNeuralNet()" cuando se crea un objeto "BasicNet".

El resto de los métodos públicos fue utilizado durante la fase de prueba para chequear el entrenamiento de la red. Los métodos: "cycleTerminated()", "netStarted()" y "netStopped()" representan eventos que se disparan en diferentes partes del flujo de entrenamiento de la red, cuando termina un ciclo, cuando la red comienza a entrenar o a correr y cuando finaliza su ejecución, respectivamente. En los casos en los que la red simplemente ejecuta un ciclo y no está en modo de entrenamiento, los eventos simplemente escriben en el *log* un mensaje a nivel de *Debug*.

Perceptron01: Esta clase como bien puede apreciarse hereda su comportamiento y atributos de "BasicNet", pero redefine su arquitectura a través del método privado "setNeuralArchitecture()", dicho método se invoca en el constructor luego de que este invoque al constructor padre (*super.*) Además define dos nuevos métodos:

"getResults()": Este método devuelve el valor de salida de la única neurona artificial de la capa de salida de la red neuronal. Dicho valor, que se encuentra entre 1 y 0 será luego redondeado para determinar si la frase de entrada es independiente del contexto: 1 ó dependiente del contexto: 0.

"setInputPattern()": Este método está sobrecargado, una de sus definiciones puede recibir un *String* y la otra un *array* de *double*. Pero el *String* que puede recibir no es más que una serie de números en punto flotante separados por punto y coma, que serán separados y convertidos en un *array* de *double* para luego invocar al otro método. El *array* es el mapa que genera el método "generateInput2()", de la clase "ParseToInput"

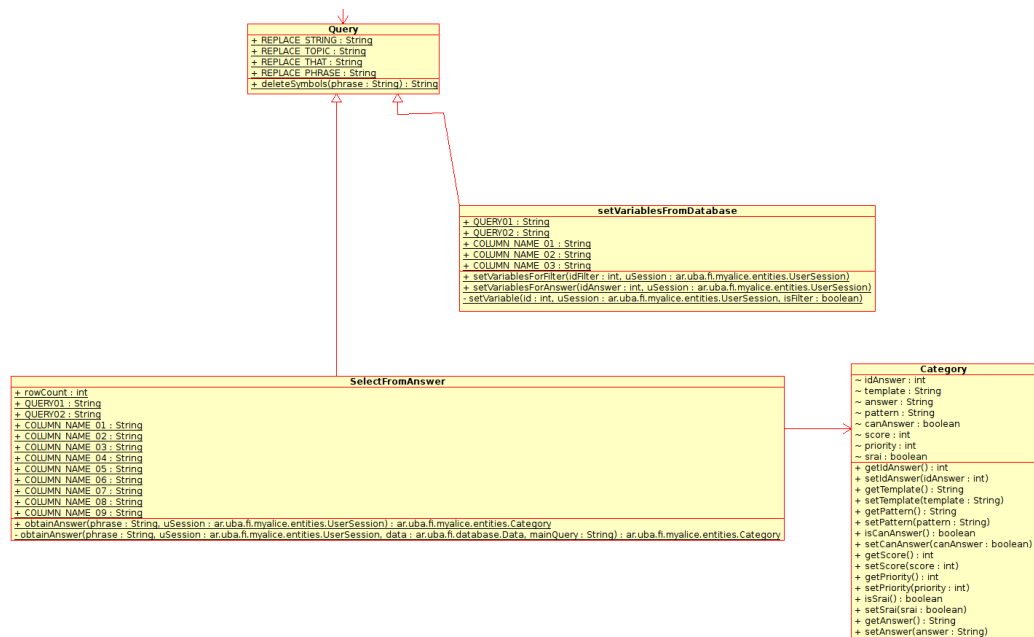


Diagrama 5.3.14: Clases, Query, setVariablesFromDatabase, SelectFromAnswer

Query: Esta clase define 4 *Strings* constantes de uso interno, los cuales representan *metatexto* a reemplazar en las plantillas de armado de las consultas definidas en sus clases herederas. Así por ejemplo el *metatexto*: "<REPLACE_THAT>" en la consulta de la clase "SelectFromAnswer" será reemplazado por el contenido de la variable de sesión "that" del objeto de la clase "UserSession" correspondiente.

"deleteSymbols()": Método utilizado para limpiar los *strings* antes de concatenarlos en las plantillas de las consultas a la base de datos.

SelectFromAnswer: Clase que encapsula la consulta a la base de datos que obtiene la respuesta de la tabla "alice_answer" a través del método "obtainAnswer()" a partir de dos parámetros: la frase de entrada del usuario y el objeto "UserSession". Dicho método devuelve un objeto *category*.

setVariablesFromDatabase: Clase que encapsula la consulta a la base de datos que obtiene las variables de sesión a *setear* para una respuesta o bien un *filtro-srai* obtenido en las consultas previas. Lo hace a través de dos métodos "setVariablesForAnswer()" para las variables asociadas a una respuesta de la tabla "alice_answer" o a través del método "setVariablesForFilter()" para las variables asociadas a un registro de la tabla "alice_filter". Dicho método no devuelve nada sino que simplemente *setea* las variables correspondientes en el objeto

"UserSession".

Category: Clase entidad que representa una categoría tal y como se definen en AIML. Cuando el método "obtainAnswer()" de la clase "SelectFromAnswer" devuelve un objeto "category" este representa a la "category" seleccionada tal y como lo haría AIML tradicional.

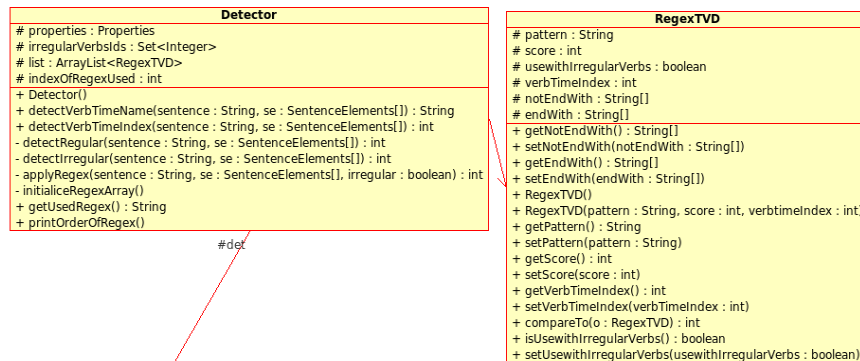


Diagrama 5.3.15: Clases, Detector, RegexTVD

RegexTVD: Clase que representa a una expresión regular de las utilizadas para detectar tiempos verbales. Cada tiempo verbal tiene una serie de expresiones regulares asociadas, las cuales están en un archivo de *properties*. Además cada expresión regular tiene los siguientes atributos:

"pattern" : *String*, contiene la expresión regular propiamente dicha.

"score" : campo entero, representa con un número de 0 a 100 cuan probable es (de forma aproximada) que si la expresión regular coincidió con una frase, dicha frase esté el tiempo verbal asignado a la expresión regular. Se utiliza también como un orden para la ejecución de las expresiones regulares, ya que primero se correrán aquellas que tengan mayor *score*.

"usewithIrregularVerbs": *flag booleano*, indica si la expresión regular propiamente dicha, también es válida para un verbo irregular.

"verbTimeIndex": Entero que indica que tipo de tiempo verbal detecta la expresión regular. Los diferentes tiempos verbales están definidos en una clase "Constants" del paquete "timeverbdetector"

"notEndWith": Es un *array* de *Strings*, si dicho *array* no está vacío se impone una condición extra para la detección del tiempo verbal, es decir que además de la coincidencia entre la frase y la expresión regular propiamente dicha, la frase no debe terminar con ninguna de las cadenas del *array*.

"endWith": Es un *array* de *Strings*, si dicho *array* no está vacío se impone una condición extra para la detección del tiempo verbal, es decir que además de la coincidencia entre la frase y

la expresión regular propiamente dicha, la frase debe terminar en alguna de las cadenas del *array*.

Esta clase no posee lógica, sus métodos son *getters* y *setters* para sus atributos.

Detector: Esta clase tiene la responsabilidad de detectar el tiempo verbal de una frase, esta responsabilidad recae en alguno de los métodos "detectVerbTimeIndex()" o "detectVerbTimeName()" esta última simplemente invoca a la primera y luego utiliza el entero obtenido para devolver el nombre del tiempo verbal, según un *array* estático definido en la clase "Constants" del paquete "timeverbdetector". Este último método mencionado solo se utilizó para realizar pruebas.

Sus atributos son:

"properties": Objeto *Properties*, referencia al archivo de *properties* con las expresiones regulares que utilizará para cargar un *array* de objetos "RegexTVD"

"irregularVerbsIds": Conjunto de *Ids* de verbos irregulares.

"list" : Lista ordenada por el campo "score" de objetos "RegexTVD"

"indexOfRegexUsed": Índice de expresiones regulares, este índice se incrementa a medida que va aplicando una tras otras las expresiones regulares.

Sus métodos son los siguientes:

"detectVerbTimeIndex()": Este método como se indicó anteriormente es el principal, y es el que utilizará la clase "PhraseToInput" para obtener el tiempo verbal para generar el mapa. Básicamente este método analizará el *array* de "SenteceElements" para saber si el *parser* detectó algún verbo irregular ya que estos verbos en sus tiempos infinitivo, pasado y pasado participio están cargados en la base de datos "Lexico". Si es el caso invocará al método "detectIrregular()"

"detectRegular()": Este método invoca a "applyRegex()" que aplica una por una las expresiones regulares de la lista de "RegexTVD".

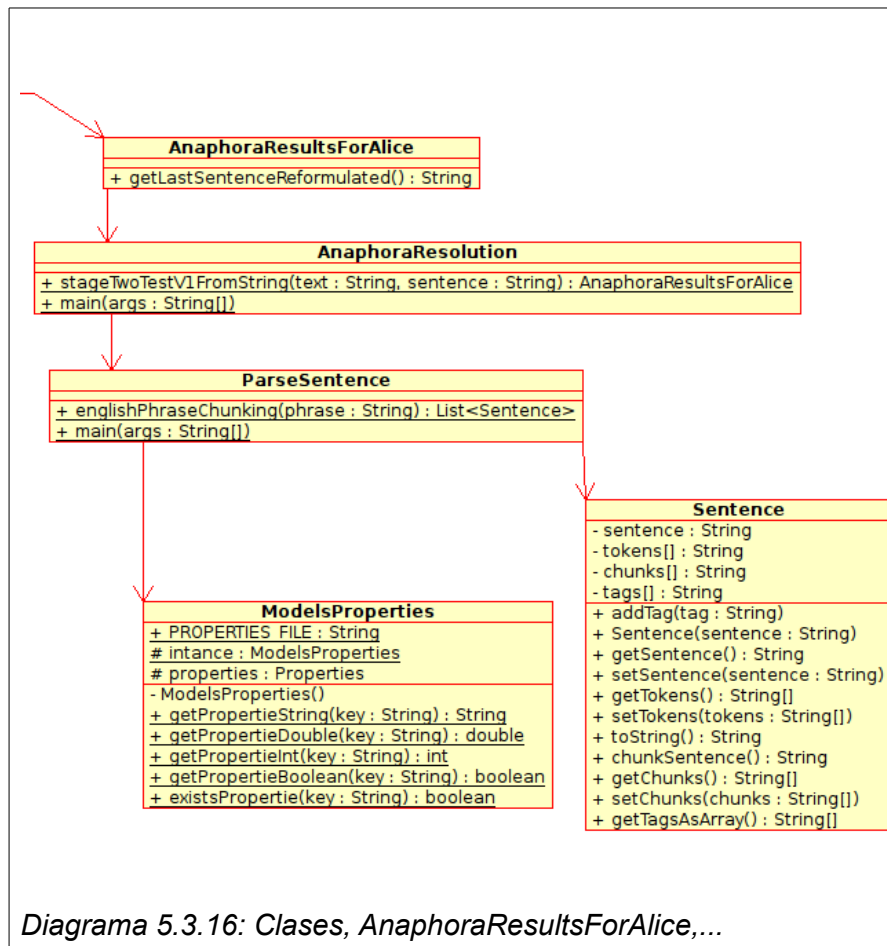
"detectIrregular()": Este método primero invocará a "applyRegex()" que aplicará solo las expresiones regulares validas para verbos irregulares. En caso de que no encuentre ninguna coincidencia aplicará cuatro expresiones regulares más, que son exclusivas de los verbos irregulares, dichas expresiones están definidas tambien en el archivo de *properties*.

"applyRegex()": Método que contiene la lógica necesaria para aplicar las expresiones regulares de los objetos "RegexTVD" y aplica toda la lista de forma secuencial y ordenada hasta encontrar una coincidencia.

"initialiceRegexArray()": Método que a partir del objeto "Properties" asociado al archivo, crea uno por uno los objetos "RegexTVD" y los agrega a la lista.

"getUsedRegex()": método que imprime por pantalla, utilizado solo con fines de prueba.

"printOrderOfRegex()": método que imprime por pantalla, utilizado solo con fines de prueba.



ParseSentence: Esta clase encapsula parte del comportamiento que ofrece la potente biblioteca *OpenNLP* para el procesamiento del lenguaje natural. La biblioteca en cuestión es libre y puede ser adquirida en esta dirección: "<http://opennlp.sourceforge.net/index.html>".

El método "englishPhraseChunking()" toma como parámetro una frase en inglés y la analiza, descomponiéndola en sus elementos sintácticos. Para ello invoca distintos métodos de la biblioteca mencionada.

Sentence: Esta clase representa la descomposición sintáctica de una frase en inglés, el atributo "sentence" guardará el valor de la frase original mientras que los atributos "tokens", "chunks" y "tags" guardarán *arrays* de los respectivos elementos sintácticos y descomposiciones que aplica el algoritmo anterior.

AnaphoraResultsForAlice: Esta clase hereda de una clase llamada "AnaphoraResults", la cual está incluida en una biblioteca llamada: "javaRAP.jar", e incluye una implementación de

algoritmo *Pronominal Anaphora Resolution* publicado por [Lappin, Leass 1994]. Dicho algoritmo puede descargado de forma gratuita de la siguiente dirección: "<http://wing.comp.nus.edu.sg/~qiu/NLPTools/JavaRAP.html>"

El método : "getLastSentenceReformulated()" es el único método que agrega esta clase a la original y su funcionalidad es "limpiar" la frase de entrada para que la utilice el algoritmo.

AnaphoraResolution: Esta clase tiene un solo método: "stageTwoTestV1FromString", (el método "main" existe solo a fines de probar el funcionamiento de la clase.) Dicho método recibe dos parámetros la última frase de entrada del usuario: "sentence" y las últimas frases históricas de la conversación: "text". Internamente lo que hará será descomponer la totalidad del texto: "sentence" más "text", en sus elementos sintácticos constitutivos, para lo cual utilizará el método: "englishPhraseChunking()" de la clase "ParseSentence". Luego el nuevo *String*, que obtuvo de concatenar los diferentes elementos constitutivos, será el *input* del método: "resolverV1Extra" de la clase "Util" perteneciente a la biblioteca "javaRAP"

5.3.4.3 Diseño de la jerarquía de paquetes

El esquema propuesto para la jerarquía de paquetes fue respetado casi en su totalidad, como se observa en el diagrama solo se dejó fuera de la jerarquía "ar.uba.fi" al subsistema implementado por "Eiza", cuyo código se incorporó al proyecto sin modificar su paquete; el sistema de resolución de referentes está distribuido en tres paquetes, por un lado el paquete "opennlp" contiene la funcionalidad del parser de lenguaje natural, pertenece a una biblioteca "jar" externa, el paquete "edu.nus.comp.nlp" contiene la funcionalidad del algoritmo RAP, también en una biblioteca externa y por último el paquete "myalice.nlp" agrupa todas las clases locales que adaptan el comportamiento del algoritmo RAP a "myAlice".

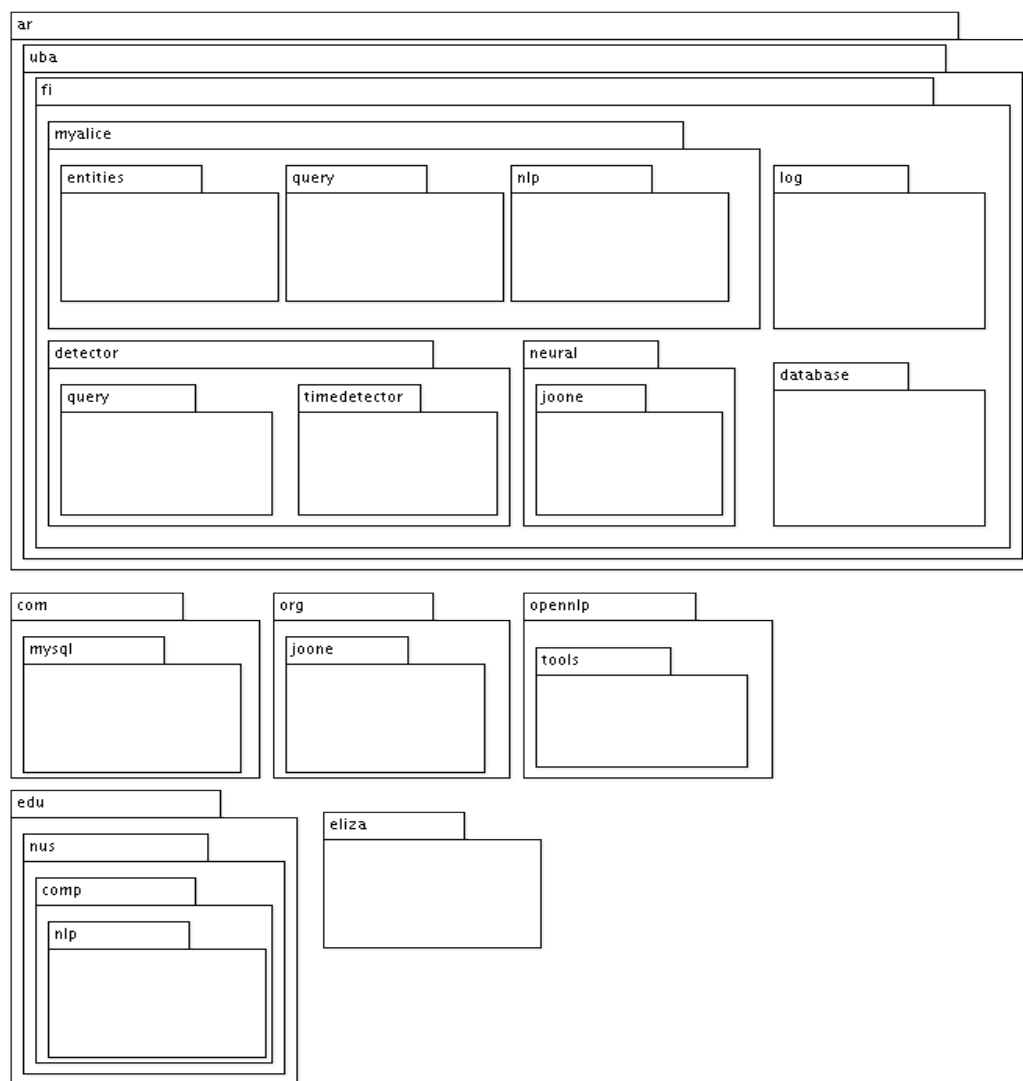


Diagrama 5.3.17: Paquetes

5.3.5 Diseño físico de datos

En esta sección se detalla el diagrama final de datos utilizado mediante un diagrama de tablas y se describen las optimizaciones y cambios que sufrió desde su concepción original.

5.3.5.1 Diseño del modelo físico de datos

A continuación se muestra el diseño de las tablas de ambas bases de datos.

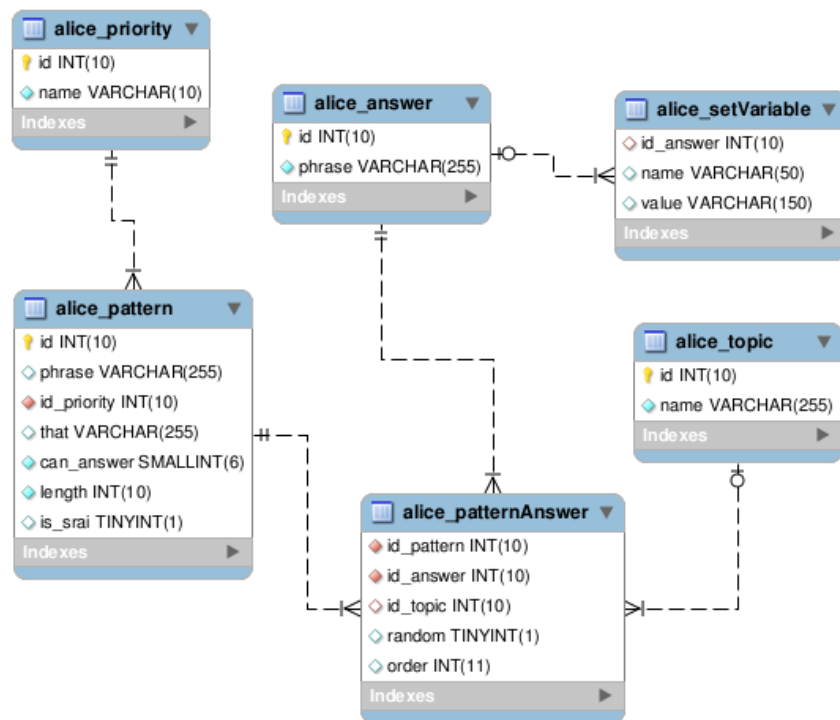


Diagrama 5.3.16: Diseño de tablas "myAlice"

Este esquemas de tablas cambió con respecto al concebido en el DER de la fase anterior. La mayor diferencia es la eliminación de las tablas "srai" y "filter". Esto se debe a que el esquema anterior no resultó práctico a la hora de buscar el mejor patrón de coincidencia en ambas tablas: "pattern" y "filter", además exigía siempre dos consultas contra ambas tablas. Todo el *overhead* producto del esquema anterior se solucionó utilizando una sola tabla para los *patterns*: "alice_pattern" y una sola tabla para los *templates* "alice_answer" con un campo extra que opera de discriminador: "is_srai" que indica cuando el *tag template* de los viejos archivos AIML contenía al *tag srai*, de esta forma se respeta el concepto de simular el sistema de *tags* de AIML bajo una base de datos relacional.

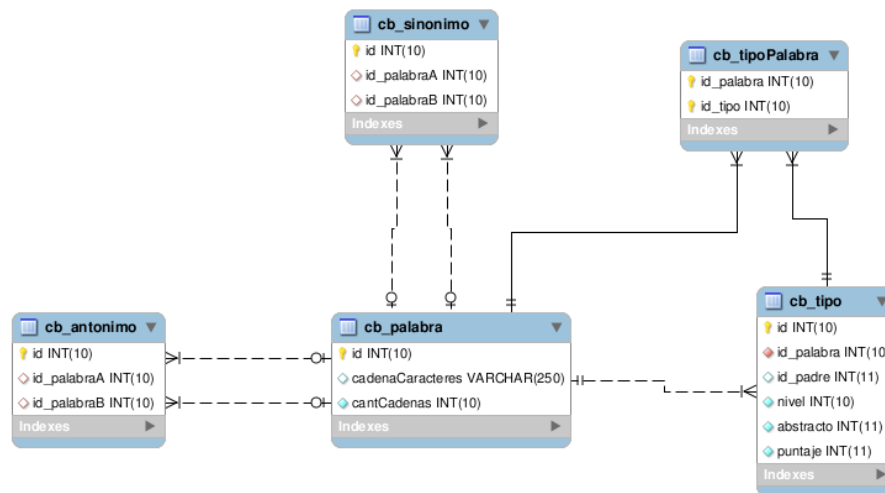


Diagrama 5.3.17: Diseño de tablas "Léxico"

Ambos esquemas se mantuvieron en bases separadas, aunque sobre una misma instancia de mySql durante la fase de pruebas.

La tabla "cb_antónimo" se agregó debido a que el diccionario de *OpenOffice* utilizado para poblar los datos tenía esta información de la misma forma en la cual tenía los sinónimos, así que se optó por importar los antónimos por si llegaban a ser requeridos en algún momento futuro pero nunca se utilizaron.

Durante la fase de pruebas se crearon dos vistas para trabajar sobre la tabla de "Léxico":

"tiposConNombre": Vista que realizaba un *join* con la tabla palabra para determinar el nombre de cada tipo, ya que el nombre como cadena de caracteres no existe más que como una referencia a la tabla "cb_palabra"

"cbv_tiposArbol" : Esta vista hacía varias veces *join* de la tabla tipo con sí misma para rearmar el árbol de clasificación de palabras.

La tabla "cb_tipo" como se mostró en el DER es una tabla recursiva, debido a que su clasificación parte de los tipos de palabra más básicos y luego los especializa, por ejemplo hay un tipo de palabra como son los pronombres, (*pronoun* en inglés) que pueden a su vez ser personales o posesivos, (*personal* o *possessive*), estar en primera persona o en segunda persona, (*first person* o *second person*) etc. Por ello tiene un campo "id_padre" que apunta a sí mismo o bien es cero.

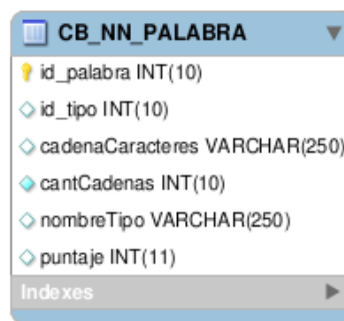
El campo "nivel" indica la profundidad dentro del árbol, el campo "abstracto" indica si un tipo de palabra es hoja o no dentro del árbol. El campo "puntaje" indica cuán relevante es un tipo de palabra en una oración o frase, para saber si esta será independiente del contexto o no.

5.3.5.2 Optimización del modelo físico de datos

Al finalizar la fase de pruebas, debido a que la base de datos de "Léxico" no sería actualizada más sino solo consultada. Se creó una tabla desnormalizada con la misma información y se le agregó a la base de datos de "myAlice".

Esto redujo los tiempos de acceso a la base de datos y la cantidad de consultas que antes eran necesarias para realizar los *joins* entre las tablas.

La nueva tabla se la llamó: "CB_NN_PALABRA":



The image shows a screenshot of a database interface displaying the structure of a table named 'CB_NN_PALABRA'. The table has the following columns: 'id_palabra' (INT(10)), 'id_tipo' (INT(10)), 'cadenaCaracteres' (VARCHAR(250)), 'cantCadenas' (INT(10)), 'nombreTipo' (VARCHAR(250)), and 'puntaje' (INT(11)). There is an 'Indexes' section at the bottom with a right-pointing arrow.

Column Name	Data Type
id_palabra	INT(10)
id_tipo	INT(10)
cadenaCaracteres	VARCHAR(250)
cantCadenas	INT(10)
nombreTipo	VARCHAR(250)
puntaje	INT(11)

Diagrama 5.3.18: tabla desnormalizada

5.3.6 Carga inicial de datos

En esta sección se detallan los procedimientos utilizados para cargar los datos, gran parte de los cuales estaban originalmente en archivos AIML, que no son más que archivos XML y fueron pasados al esquema relacional. Dos métodos fueron utilizados, primero un pequeño sistema preparado para tal fin y a continuación se utilizaron sentencias SQL para completar las diferentes tablas.

5.3.6.1 Diseño de procedimientos de carga inicial de datos

En esta sección se detallan los procedimientos y programas utilizados para la carga inicial de datos. Asimismo sus fuentes.

5.3.6.1.1 Uploader

Con la aplicación "Uploader" descrita en el punto 5.3.2.1.1 se realizó la importación principal de datos para las bases "myAlice" y "Lexico". Las fuentes de información fueron como se indicó los archivos ".aiml" que conforman el "cerebro" de ALICE, obtenidos de la implementación de ALICE sobre el intérprete "programv-0.08".

Los datos para la base de datos de "Lexico", fueron obtenidos de los diccionarios de *OpenOffice* para el idioma inglés: el diccionario de palabras, utilizado para realizar la corrección ortográfica y el diccionario de sinónimos, utilizado mediante la funcionalidad de consulta de sinónimos y antónimos.

Cada uno de estos archivos pasó por diferentes procesos que se describen a continuación antes de ser importados en la base de datos:

5.3.6.1.1.1 Diccionario de palabras

La opción "line" se utilizó para importar el archivo de diccionario de idioma inglés de *OpenOffice*, los cuales son básicamente una lista de palabras, donde hay una por línea.

Antes de importar el archivo se realizaron las siguientes tareas:

1. Se convirtió al archivo al *encoding* UTF-8 que utiliza la base de datos, con el comando de GNU-Linux: "iconv"
2. Se eliminó la primera línea que contenía el número de líneas totales del archivo.
3. Al final de cada palabra había un carácter "/" y una serie de letras. Esto se eliminó mediante una expresión regular desde el editor Kate.

5.3.6.1.1.2 Tipo de palabra y sinónimos

La opción "gra" se utilizó para importar el archivo de gramática para el idioma inglés de *OpenOffice*, (th_en_US_v2.dat), este archivo tiene un formato no convencional, la primera línea es el *encoding*, luego tiene una lista de palabras con todos sus sinónimos y el tipo al cual pertenecen. Un ejemplo de este archivo es el siguiente:

```
agnostic|4
(adj)|religious person (related term)|unbelief|disbelief (related term)
(adj)|agnostical|nescient (similar term)| gnostic (antonym)
```

(noun)|doubter|person (generic term)|individual (generic term)

(noun)|religious person (generic term)

Como se observa después de cada palabra hay un carácter "|" que la separa de un número que indica cuantas líneas vienen a continuación con información asociada a esa palabra. Cada línea empieza con el tipo de palabra *adj, noun* en este caso y una serie de palabras separadas por "|" que pueden ser: sinónimos, antónimos o términos relacionados.

Antes de importar el archivo se lo convirtió a UTF-8, con el comando "iconv" de GNU-Linux.

El proceso de importación de esta información fue más compleja que para el caso anterior, y se realizó a través de los siguientes pasos:

1. leer una palabra y asociarle una lista de objetos "Palabra", donde cada objeto tiene los siguientes campos: "palabra" (*String*), "tipo"(*int*),"id_palabra"(*int*),"sinonimo"(*boolean*), "genericTerm" (*boolean*)
2. Si la palabra principal o alguna de las asociadas no existe en la base de datos se la crea, y se obtiene el ID para cada una de ellas. Con el tipo de palabra asociado.
3. Se da de alta la palabra en la lista de sinónimos.

5.3.6.1.1.3 Archivos AIML

Los archivos AIML son básicamente archivos XML, por lo que se utilizó un *parser* XML: *dom4j*, para realizar la parte central del trabajo. El *parser* se utilizó para recorrer de manera secuencial todos los *tags* "category" y obtener directamente el contenido de cada elemento, al cual se lo obtuvo como texto plano para poder trabajar sobre él con expresiones regulares que hicieron los reemplazos necesarios para generar las consultas *insert* para la base de datos.

Cada archivo procesado generó un archivo de mySql con una serie de sentencias "*insert into*" que finalmente se corrieron sobre la base de datos y poblaron las tablas con la misma información del archivo AIML original.

Los archivos AIML, se tomaron del intérprete *programv-0.08*, y se convirtieron al *encoding* UTF-8 con el comando de GNU-Linux: "iconv".

5.3.6.1.2 Sentencias SQL, creadas desde el editor

Como se mencionó, no todas las inserciones se realizaron mediante la aplicación "Uploader", hubo también inserciones posteriores sobre la base de datos "Lexico" realizadas de

manera manual. Las consultas se construyeron utilizando el editor Kate y expresiones regulares para limpiar y organizar los datos. Estas nuevas inserciones se realizaron sobre "Lexico" exclusivamente y su finalidad fue la de mejorar la calidad de los datos y ampliar su cantidad.

Se realizaron las siguientes inserciones:

- 1 Se insertaron, si no existían las 500 palabras más utilizadas del idioma inglés, tomadas de aquí: <http://www.world-english.org/english500.htm>
- 2 Se insertaron, si no existían los 100 verbos más usados del idioma inglés, tomados de aquí: <http://www.world-english.org/100verbs.htm>
- 3 Se agregaron nuevos tipos de palabra: verbos irregulares, pronombres (y todos sus subtipos), pronombres interrogativos, auxiliares, etc.
- 4 Se hicieron *inserts* para asociar palabras existentes a las nuevos tipos de palabra creados.
5. Se realizó una categorización minuciosa de los diferentes tipos de pronombres: personales y posesivos. Se los insertó en la base de datos si no existían y a cada uno se le asignó el tipo correspondiente.
6. Finalmente se realizó el proceso de asignarle un valor numérico a cada tipo de palabra según su relevancia en una oración.

Si bien el proceso anterior no fue tan lineal y hubo una serie de correcciones y validaciones constantes en los *queries*, la numeración es correcta de forma esquemática.

5.3.7 Establecimiento de requisitos de implantación

En esta sección se detallan todos los requisitos de hardware y software necesarios para que el sistema final pueda ser instalado y pueda correr de forma correcta.

5.3.7.1 Especificación de requisitos de implantación

myAlice

El sistema construido como biblioteca, no tendrá un instalador. Simplemente estará constituido por un solo archivo de extensión ".jar", llamado: "myAlice.jar" más las dependencias indicadas a continuación y la base de datos. Dicho archivo deberá de copiarse en cualquier carpeta de bibliotecas que utilice el sistema que lo implemente.

Dependencias:

La biblioteca myAlice.jar depende de las siguientes bibliotecas:

joone-engine.jar : *Engine* del sistema de redes neuronales: Joone.

mysql-connector-java-4.x.x-bin.jar : Driver para el acceso a una base de datos MySQL.

Base de Datos:

La biblioteca requiere tener acceso a una base de datos MySQL, en donde esté *deployada* la base de datos del sistema. Dicha base de datos se distribuirá conjuntamente con la biblioteca en un *dump* de MySQL. Para *deployar* la base de datos se necesita:

1. Un servidor de base de datos MySQL 5.0 o superior instalado (en desarrollo y *testing* se utilizó la versión: 5.0.45) y corriendo en el sistema
2. Crear un nuevo esquema
3. Restaurar el *dump* en el esquema anterior
4. Crear un usuario con permisos de consulta sobre la base anteriormente creada.

Red Neuronal:

La biblioteca requiere utilizar una red neuronal artificial previamente entrenada y especialmente diseñada, dicha red se distribuye en un formato de archivo que el *engine* de "Joone" sabe reconocer y puede restaurar y utilizar.

Junto con la biblioteca se distribuirá un archivo "CC_01_perceptron.snet" que contiene la red neuronal artificial y su estado último de entrenamiento. Este archivo debe ser copiado en una carpeta cualquiera que luego será especificada.

Archivos de Configuración y extra:

En la carpeta raíz, donde corra el sistema que implemente la biblioteca "myAlice" deberán estar los siguientes archivos de configuración:

chatbotLog.properties: Archivo de propiedades donde se especifican parámetros de control del sistema de log.

database.properties: Datos de conexión y acceso a la base de datos.

neural.properties: Parámetros de la red neuronal, en general no deberían cambiarse a menos que se estén realizando pruebas con las redes neuronales. La última propiedad especifica la ubicación y nombre del

archivo "CC_01_perceptron.snet"

substitutions.properties: En este archivo se especifican todos los patrones de reemplazo que "myAlice" utilizará para limpiar las frases de entrada. Cada patrón tiene la siguiente forma:

<Regex>=><patrón de reemplazo>.

La cantidad total debe ser especificada al comienzo del archivo.

questions.properties: Lista de palabras que al comienzo de una oración indican que es una pregunta. Por ejemplo: "What", "Can"

timeverbdetector.properties: Este archivo contiene una lista de expresiones regulares, como se definieron en el punto 5.3.4.2 para la clase "RegexTVD".

eliza.dat: información para el subsistema de "Eliza", contiene una lista de patrones de coincidencia con respuestas asociadas tal y como se explicó en el punto: 5.3.4.2 para la clase "ElizaComments"

Sistemas Operativos:

Si bien todas las pruebas y el desarrollo se realizó sobre sistemas Operativos GNU-Linux, bajo las distribuciones Kubuntu, de la 7.10 a la 9.10, el sistema fue desarrollado íntegramente en Java, con lo cual debería de poder funcionar en todos aquellos SO soportados por Java.

myAlice Debug console

La consola de *Debug*, es un caso particular de una implementación de la biblioteca "myAlice.jar". Como se mostró en el punto 5.3.3.2. La siguiente descripción sirve para ejemplificar como se ubicarían las diferentes bibliotecas y archivos en una implementación real.

En este caso el sistema completo está constituido por la siguiente jerarquía de directorios:

Directorio Raíz de la Aplicación: Contiene al "jar" ejecutable: " myAliceDebugConsole.jar" a todos los archivos de "properties" y al archivo "eliza.dat"

subdirectorio "nets": contiene al archivo "CC_01_perceptron.snet"

subdirectorio "libs": contiene a "myAlice.jar" y a sus dependencias: "joone-engine.jar" y "mysql-connector-java-4.x.x-bin.jar" y a las dependencias propias de la interfaz de usuario: "swing-worker-1.1.jar" y "appframework-1.0.3.jar"

La aplicación no requiere instalación alguna, se distribuirá con el código fuente de manera gratuita en un archivo comprimido como "tar.gz". Al cual solo hay que descomprimirlo en un directorio del usuario y no requiere ningún tratamiento adicional con excepción de la creación de la base de datos.

Para correr la aplicación basta con ir al directorio en donde se desinstaló y ejecutar en la consola: "java -jar myAliceDebugConsole.jar"

5.3.8 Aprobación del diseño del sistema de información

En reunión mantenida con el director de tesis, se han aprobado el diseño y especificaciones vertidas en el presente documento.

5.4 Construcción del sistema de información

En esta sección se detallan los resultados de todas las pruebas corridas a lo largo del desarrollo del sistema, los resultados de las pruebas realizadas a los datos migrados, los resultados de las pruebas realizadas sobre los distintos subsistemas, las pruebas de integración y por último las aceptación. Así mismo se detallan los cambios realizados a partir de resultados no satisfactorios en las pruebas.

5.4.1 Procedimientos de migración y carga inicial de datos

En esta sección se detallan los resultados de todas las pruebas corridas en la fase de migración de datos de AIML a MySQL y de la importación de nuevos datos para la base Lexico.

5.4.1.1 Realización y evaluación de las pruebas de migración y carga inicial de datos

En la siguiente subsección se comenzará por enumerar cada una de las pruebas especificando su descripción, ejecución, resultados y evaluación de dichos resultados.

5.4.1.1.1 Pruebas de la carga de datos de myAlice

Pruebas:

i Verificar la cantidad de *categories* importadas

Descripción:

Contar los *tags* "category" que tiene cada archivo "aiml" y luego de su importación realizar un *count* sobre las tablas correspondientes y verificar que los números coincidan

Ejecución:

Se utilizó el editor de texto "Kate", sobre cada uno de los archivos AIML de ALICE y se realizó un reemplazo de la cadena de caracteres "<category>" con sigo misma para obtener el número de reemplazos hechos.

En la base de datos se realizó un "count(*)" de las tablas "alice_pattern" y "alice_filter" dichas tablas contienen un patrón por registro como los *tags* "category". Estos *tags* no

tienen entidad propia en la base de datos, sino que representan la unión de los diferentes registros de las diferentes tablas. La manera de identificarlos es por el contenido del *tag* "pattern"

Resultados:

Total de *tags* "category" en los archivos AIML : 41320

Total de registros en las tablas "alice_filter" y "alice_pattern" : 41319

Evaluación de Resultados:

Como se puede apreciar hay un solo registro de diferencias entre ambas. Se realizó un minucioso conteo de cuantos registros hay en la base de datos que correspondan con cada uno de los diferentes archivos importados. Este detalle se puede ver en el apéndice. La diferencia apareció entre los archivos que agrupan *categories* cuyo patrón comienza con un *wildcard*, "_" o bien "*", aunque no se pudo precisar cuáles de estos eran.

Debido a que el error cometido es menor que el 0,003 por ciento, se aceptaron como buenos dichos resultados.

ii. Verificación de caracteres no estándar**Descripción:**

Buscar en los archivos AIML porciones de texto donde haya caracteres no estándar (áéíóúÁÉÍÓÚñ¿¡). Encontrar al menos 10 casos. Buscar mediante consultas a la base de datos las mismas porciones de texto y verificar que no hubo cambios

Ejecución:

Para la obtención de los casos se utilizó el siguiente comando de GNU-Linux: "cat A.aiml.done | grep "[áéíóúÛñ¿¡]" . Al revisar los archivos se observó que no todos los caracteres especificados figuraban en los archivos, los únicos caracteres con codificación no estándar hallados fueron: "à","ù","é","ê","è","ç","í","ô","É","à". Se probó entonces con los siguientes casos:

Voilà un sujet très intéressant.

Où ca, près de quelle ville?

Une série et des films cultes.

Attention! Un australopithèque vient juste d'être détecté.

Tout est relatif, je préfère me concentrer sur l'utilité des choses.

Peut-être, ne suis je pas encore programmé pour ça.

Tolstoï est l'auteur de "Guerre et paix".

Il s'agit plutôt de clones.

Oui, réellement.

Difficile à infirmer ou confirmer?

Resultados:

Utilizando consultas directas a la base de datos se recuperaron cada uno de los textos anteriores correctamente codificados en UTF-8.

Evaluación de Resultados:

Los resultados fueron los esperados: 10 de 10, no hubo datos mal grabados por errores de codificación.

iii. Reconstrucción de datos normalizados

Descripción:

Tomar 10 "category"s de los archivos AIML, luego mediante "inner joins" en la base de datos verificar que asocian correctamente los "patterns" a los "template", "srai" y/o "tags" correspondientes.

Ejecución:

Se obtuvieron 10 "tags" "category", cuyos sub "tags" "pattern" contienen respectivamente: "19", "A **", "ARE YOU BIGGER THAN **", "EARTH", "EH SINON **", "J AI ACCIDENTELEMMENT **", "J AI COMMENT **", "WAITING FOR ME", "WANN WAR **", "T ES **" de los archivos aiml: "1.AIML", "A.AIML", "E.AIML", "J.AIML", "W.AIML", "T.AIML".

Luego se ejecutó una consulta a la base de datos, por cada "tag" "category". Buscando reconstruir su contenido a partir del "tag" "pattern" se ejecutaron dos tipos de consulta: si el "tag" "template", contenía un "tag" "srai" se utilizó una consulta como esta:

```
SELECT * FROM alice_filter a inner join alice_filterSrai c on a.id=c.id_filter inner join
alice_srai b on c.id_srai = b.id where phrase = '19'
```

Ya que este tipo de "categories" fueron volcadas en tablas distintas a las que se utilizarán para los casos en donde no hay un "tag" "srai" de por medio. Para esos casos se utilizó una consulta como la siguiente:

```
SELECT * FROM alice_pattern a inner join alice_patternAnswer c on a.id=c.id_pattern
inner join alice_answer b on c.id_answer = b.id where a.phrase = 'WAITING FOR ME'
```

En los casos en los cuales una "category" contenía además "seteos" de variables de sesión, se agregó un "join" con la tabla: "alice_setVariable"

Los detalles de la ejecución se pueden ser encontrados en el repositorio on-line del proyecto en una planilla de cálculo en el directorio: "test"

Resultados:

Los "resultsets" devueltos en cada caso coincidieron con el contenido de la "category" original. En los casos en los cuales había un "tag" "random" con varias respuestas posibles, el "resultset" contó con tantas tuplas como respuestas posibles. Y en los casos en los

cuales había *seteos* de variable dentro de una "category" estas variables (nombre y valor) aparecieron dentro del *result* al agregar a la consulta la tabla "alice_setVariable"

Evaluación de Resultados:

La prueba tuvo como objetivo corroborar, por un lado, la integridad referencial, la cual indica que las *categories* fueron correctamente normalizadas. Por otro lado se buscó detectar si el contenido de cada *tag* había sido correctamente importado en la tabla que correspondía. Por ejemplo que el contenido del *tag* "template" se haya importado en la tabla "alice_answer", con excepción de los casos en los cuales este estuviese dentro de un *tag* "srai" en los cuales se tendría que haber importado en la tabla "alice_srai".

Todas las pruebas fueron exitosas y además se pudo corroborar que se importaron correctamente otros *tags* (<person/>, <star/>, etc.)

iv. Clasificación de "category"s según "srai"

Descripción:

Tomar 10 casos de frases con "srai" y verificar que están en la tabla "alice_filter" y "alice_srai" respectivamente los *tags* "pattern" y "srai"

Ejecución:

Se tomaron 10 *tags* "category" cuyo *tag* "pattern" contenía lo siguiente: "19", "EARTH", "EH SINON **", "J AI ACCIDENTELEMMENT **", "J AI COMMENT **", "T ES **", "WANT TO **", "WANTED TO **", "STUDENT", "VA T IL POUVOIR **". Los *tags* se obtuvieron de los siguientes archivos: 1.AIML, E.AIML, J.AIML, T.AIML, W.AIML, S.AIML, V.AIML.

En la base de datos se corrió una consulta como la siguiente por cada caso:

```
SELECT * FROM alice_filter a inner join alice_filterSrai c on a.id=c.id_filter inner join alice_srai b on c.id_srai = b.id where phrase = 'XXXX'
```

En donde el *string* "XXXX" fue cambiado por el contenido del *tag* "pattern", cambiando el carácter "*" por "%" en los casos en donde correspondía.

Resultados:

Los resultados fueron los esperados. En todos los casos las frases con *tag* "srai" se habían importado en las tablas: "alice_filter" y "alice_srai" para el contenido de los *tags* "pattern" y "template" respectivamente. Los contenidos eran los mismos y los *wildcards* fueron transformados correctamente (caracteres '*' y '_' en %).

Evaluación de Resultados:

La integridad referencial fue conservada, los datos se desnormalizaron

correctamente y es posible volver a normalizarlos a partir del patrón (ahora en el campo "phrase").

v. Clasificación de *tags* "category" según "topic"

Descripción:

Tomar 10 casos de "category"s que estén en "topics" diferentes y/o en ninguno y verificar que se cumple la asociación al hacer "inner join" con la tabla "alice_topic"

Ejecución:

Se tomaron 10 *tags* "category", 5 no correspondían a ningún *topic* en particular y 5 estaban contenidos dentro de un *tag* "topic". Estos son los casos:

Casos sin *topic*: aquellas *categorías* cuyo *tag* "pattern" contenía lo siguiente:

"19", "EARTH", "EH SINON **", "J AI ACCIDENTELEMMENT **", "J AI COMMENT **"

Casos con *topic*: aquellas *categorías* cuyo *tag* "pattern" y "topic" contenían Respectivamente lo siguiente:

"WHAT DO BEARDED DRAGONS LOOK LIKE", "** LIZARDS"

"WHAT DO CHAMELEONS LOOK LIKE", "** LIZARDS"

"**", "ACCOUNTANT"

"**", "ACTOR"

"**", "GAMBLING"

De los archivos W.AIML y star.AIML respectivamente.

Luego se utilizó una consulta como la siguiente contra la base de datos para validar que se tenía un *topic* asociado:

```
SELECT * FROM alice_pattern a inner join alice_patternAnswer c on a.id=c.id_pattern  
inner join alice_answer b on c.id_answer = b.id inner join alice_topic t on id_topic=t.id where  
a.phrase = 'XXX' and t.name = 'YYY'
```

En donde XXX se reemplaza con el contenido del *tag* "pattern" y YYY con el contenido del *tag* "topic". Para los casos en donde no había un *tag* "topic" asociado se omitió la última condición, ya que al hacer la importación de datos se generó una entrada en la tabla "alice_topic", con id igual a 1 y nombre "null" que está automáticamente asociada a todas las *categories* sin *topic*.

Resultados:

Los resultados fueron los esperados, los primeros cinco casos devolvieron un *resultset* en donde la columna "id_topic" es igual a 1. En los segundos cinco casos, se obtuvo un *resultset* en donde la columna "name" de la tabla "alice_topic" contenía lo mismo que el *tag* "topic"

Evaluación de Resultados:

La integridad referencial fue conservada, los datos se desnormalizaron correctamente y es posible volver a normalizarlos a partir del patrón y el campo "topic"

vi. verificar la importación correcta de las variables a setear asociadas a una "category"**Descripción:**

Tomar 10 casos en donde haya asociados a una "category" el seteo de variables. Verificar si las asociaciones se cumplen al hacer "inner join" con la tabla `alice_setVariable`

Ejecución:

Se tomaron las *categories* cuyo *tag* "pattern" es igual a: "A **", "A * IS NOT **", "A * ONE", "DANIELLE", "DANNY IS **", "I AM **", "I AM * BEAUTIFUL", "I AM * POUNDS", "WAS **", "WE ARE ALL GOING TO DIE". De los archivos: "A.AIML", "D.AIML", "I.AIML" y "W.AIML". Luego se utilizó una consulta como la siguiente contra la base de datos para validar que existiesen los registros respectivos en la tabla `"alice_setVariable"`:

```
SELECT * FROM alice_pattern a inner join alice_patternAnswer c on a.id=c.id_pattern
inner join alice_answer b on c.id_answer = b.id inner join alice_setVariable d on d.id_answer =
c.id_answer where a.phrase = 'XXX'
```

En donde "XXX" es reemplazado por el contenido del *tag* "pattern".

Resultados:

Los resultados fueron los esperados, por cada *tag* "set" existe una tupla en la tabla `"alice_setVariable"`, cuyo campo "name" es igual al contenido del atributo "name" del *tag* "set" y cuyo campo "value" es igual al contenido de dicho *tag*. Reemplazando en cada caso, los *tags* `<star/>` o `<person/>` (o el que corresponda) por la nomenclatura actual: `"@star[1]"`

Evaluación de Resultados:

La integridad referencial fue conservada, los datos se desnormalizaron correctamente y es posible volver a normalizarlos a partir del patrón. Además los *tags* especiales como "star" y "person" fueron correctamente convertidos a la nomenclatura actual.

vii. verificar la importación correcta de los tags "template" que utilizan variables el tag "star" para armar la respuesta.**Descripción:**

Tomar 10 casos en donde haya asociados a un "template" el uso de variables

previamente *seteadas* y el del *tag* "star". Verificar que se respetó el nombre de las variables, su posición y que en lugar de *tags* están representadas con la nueva notación.

Ejecución:

Se tomaron las *categories* cuyo *tag* "pattern" es igual a: "EH SINON **", "J AI ACCIDENTELEMMENT **", "J AI COMMENT **", "D OU VIENNENT LES BEBES", "DEFINE IT", "WAS HABE ICH", "WAS IST UNSER THEMA **", "WALLACE **", "I LIKE THAT MOVIE **", "ONE THAT CAN **". De los archivos: "E.AIML", "J.AIML", "D.AIML", "W.AIML", "I.AIML", "O.AIML".

Luego se utilizó una consulta como la siguiente contra la base de datos para validar que los valores importados de los *tags* "templates" (asociados a los "patterns" antes mencionados), utilizaran de forma correcta los *tags* "<get>", "<bot>", "<star>" y "<person>" que hacen referencia respectivamente a: variables de sesión, constantes del chatbot, porción de texto de la frase del usuario que *matcheo* con el *wildcard*. Además se validó que dichos *tags* de XML se hayan cambiado por la nueva notación.

```
SELECT * FROM alice_pattern a inner join alice_patternAnswer c on a.id=c.id_pattern
inner join alice_answer b on c.id_answer = b.id inner join alice_setVariable d on d.id_answer =
c.id_answer where a.phrase = 'XXX'
```

En donde "XXX" es reemplazado por el contenido del *tag* "pattern".

Resultados:

Los resultados fueron los esperados. Se conservó el orden de las palabras, la cantidad de estas y en los lugares en donde se utilizaban *tags* XML como los antes mencionados se los reemplazó por la notación nueva: "@STAR[1]", "@VAR["name"]", "@BOT["name"]". En donde el contenido de los "[]" representa el número de *wildcards* (cuando no se especifica en AIML se asume 1), o bien el nombre de la variable o constante.

Incluso se evaluó un caso, para el "pattern" "I LIKE THAT MOVIE **", en donde se utilizaba una constante de chatbot para *setear* variables de sesión. En dicho caso se verificó que el contenido de la tabla "alice_setVariable" fuese correcto.

Evaluación de Resultados:

Se pueden ver los detalles de la ejecución de estas pruebas en el repositorio on-line del proyecto en una planilla de cálculo en el directorio: "test"

5.4.1.1.2 Pruebas de la carga de datos de Lexico

Pruebas:

i. Verificar la cantidad de palabras importadas sea la misma que en los diccionarios

Descripción:

Cada vez que se esté por importar un archivo, contar las palabras nuevas. Hacer un *count* sobre la base de datos antes y después de la importación del archivo; verificar que la diferencia coincide con el número de palabras nuevas.

Ejecución:

Se ejecutó la importación del archivo de diccionario inglés de "OpenOffice 2.3.0": "en_US.dic" luego de haber sido convertido al *encoding* UTF-8. El programa "uploader.jar" realizó la cuenta de los registros importados. Luego se realizó un *count* en la base de datos.

Se importó en una tabla vacía el archivo de diccionario en inglés de sinónimos (o de gramática) de "OpenOffice 2.3.0": "th_en_US_v2.dat". No solo se importaron las entradas principales sino también las palabras asociadas a cada entrada, y se asignó a cada palabra su categoría: "noun", "adj", "adv" o "verb". Según este archivo.

Se juntaron ambas importaciones en una sola tabla, obviando las palabras repetidas.

Se importó a través de una sola consulta SQL, las 500 palabras más utilizadas del idioma inglés. Fuente: <http://www.world-english.org/english500.htm>

Se importó a través de una sola consulta SQL, los 100 verbos más utilizadas del idioma inglés. Fuente: <http://www.world-english.org/100verbs.htm>

Se realizaron *inserts* en la base de datos, de forma manual de pronombres, nombres de tiempos verbales, y varias palabras no encontradas de las frases utilizadas para pruebas.

Resultados:

Del archivo de diccionario se importaron todas las palabras, un total de 62 094 corroborado por la aplicación "uploader.jar" y el *count* de la base de datos. Sin contar diferencias entre mayúsculas y minúsculas son: 60 366 palabras.

Del archivo de diccionario de sinónimos/gramática se importaron en la base 145 825

palabras. El archivo tenía 145 820 entradas (sin contar las palabras asociadas.) El importador se saltó 23 de estas entradas por errores. Y la cantidad de líneas del archivo es: 350 487

Al unir ambas listas de archivos en una sola base, sin contar las diferencias entre mayúsculas y minúsculas se obtuvieron un total de 167 233 palabras (incluyendo palabras compuestas por varias cadenas de caracteres separadas por espacio).

Del *insert* de las 500 palabras más utilizadas solo dos nuevas palabras de agregaron a la base.

Del *insert* de los 100 verbos más utilizados (que resultaron ser 96) solo una nueva palabra se incorporó a la base.

Luego de insertar nuevas categorías palabra, pronombres y varias correcciones la base de datos "Lexico" tiene en la tabla de palabras 169305 registros únicos.

Evaluación de Resultados:

La importación del archivo de diccionario fue exitosa ya que se importaron el ciento por ciento de las palabras, sin embargo la importación del archivo de gramática saltó 23 registros. Si bien la inserción no fue ciento por ciento completa 23 registros representan un error del 0,01577 por ciento. Por lo cual se lo puede aceptar.

Sin embargo la base de datos mostró que había muchas palabras faltantes que tuvieron que ser agregadas posteriormente, por lo cual si bien la inserción de los diccionarios se la puede considerar exitosa no por ello a la base de datos "Lexico" se la puede considerar completa.

ii. Verificar que se hayan importado correctamente palabras con caracteres no estándar.

Descripción:

Buscar en los archivos palabras, si las hubiese, con caracteres no estándar (áéíóúÁÉÍÓÚÑ¿¡). Encontrar como máximo 10 casos. Buscar mediante consultas a la base de datos las mismas palabras y verificar que no hubo cambios

Ejecución:

Se realizó una búsqueda de caracteres especiales sobre los archivos de texto de diccionario de "OpenOffice" (originalmente en *encoding* ISO 8859-1) y se encontraron las siguientes palabras: "Bogotá", "Velázquez", "piñata", "canapé", "Grünwald", "émigré", "Thessaloníki", "Asunción", "vicuña", "crudités".

Luego se realizó una consulta como la siguiente sobre la base de datos:

```
select * from cb_palabra where cadenaCaracteres REGEXP 'Bogotá'
```

Donde "Bogotá" fue reemplazado en cada caso por la palabra correspondiente.

Resultados:

Los resultados fueron positivos, en cada caso la palabra se mostró de manera correcta.

Evaluación de Resultados:

La conversión de ISO 8859-1 a UTF8, codificación en la que están las tablas de "Lexico" y "myAlice" se realizó de manera correcta.

iii. Reconstrucción de datos normalizados, verificar que las categorías se mantienen.

Descripción:

Tomar 10 palabras asociadas a diferentes categorías, según los diccionarios importados. Luego hacer una consulta sobre la base de datos que relacione las tablas de palabras y categorías, buscar las mismas palabras. Verificar que las categorías coinciden.

Ejecución:

Se eligieron 10 palabras de forma aleatoria del archivo diccionario de gramática de "OpenOffice", a continuación se muestran las palabras y sus respectivas categorías, que pueden ser 4 a saber: sustantivo, adjetivo, adverbio y verbo:

"magnetic tape", "noun"
"magnetise", "verb"
"cytochrome", "noun"
"cytologic", "adj"
"czechoslovakian", "adj" y "noun"
"abacinate", "verb"
"give up", "verb"
"abducens muscle", "noun"
"a priori", "adv" y "adv"
"small-mindedly", "adv"

Como puede observarse, algunas palabras pertenecen a más de una categoría.

Luego se ejecutó contra la base de datos "Lexico" una consulta como la siguiente:

```
select a.*,c.*,d.cadenaCaracteres as typeWord from cb_palabra a inner join cb_tipoPalabra  
b on a.id=b.id_palabra inner join cb_tipo c on c.id = b.id_tipo inner join cb_palabra d on  
c.id_palabra = d.id where a.cadenaCaracteres REGEXP '^a priori$'
```

Donde "a priori" fue reemplazado por cada palabra (o grupo de palabras) en cada caso.

Resultados:

Los resultados fueron los esperados, al des-normalizar la tabla de palabras éstas tenían asociados los tipos correspondientes.

Evaluación de Resultados:

La normalización de datos se efectuó de manera satisfactoria. No hay palabras mal clasificadas o asignaciones múltiples donde no correspondía.

5.4.2 Pruebas funcionales de cada subsistema

Las pruebas que se listan a continuación fueron ejecutadas sobre cada subsistema de forma independiente de los demás a fin de garantizar su correcto funcionamiento. En algunos casos las pruebas fueron corridas repetidas veces sobre un subsistema, ya que hubo que efectuar diversas correcciones hasta que estos presentasen un funcionamiento aceptable. Los resultados mostrados en cada ocasión corresponden a las últimas pruebas ejecutadas.

5.4.2.1 Comportamiento de myAlice sin las mejoras

Pruebas:

i. Casos simples (donde solo hay un "pattern" posible y un "template" asociado)

Descripción:

Buscar 5 "patterns" simples y utilizarlos como frases de "entrada" en ALICE y en myALICE a través de la consola de *testing*. Verificar que las respuestas dadas son iguales.

Ejecución:

Se eligieron los siguientes *patterns*:

- 1 do you know who socrates is?
- 2 do you like movies?
- 3 WHERE IS ADELAIDE
- 4 are you a machine?
- 5 ON THE FLOOR

Resultados:

Caso	respuesta ALICE	respuesta myAlice
1	Socrates (469-399BCE), Greek philosopher.	socrates (469-399bce), greek philosopher.
2	Yes I love film, especially science-fiction and comedy.	yes i love film, especially science-fiction and comedy.
3	it is a city in southern Australia.	is a city in southern australia.

4	Yes I am an artificial consciousness.	yes i am an artificial consciousness.
5	That doesn't sound very comfortable.	that doesn't sound very comfortable.

Tabla 5.4.2.1.1: Resultados de pruebas: myAlice sin mejoras, casos simples

Evaluación de Resultados:

Resultados correctos. En ambos casos las respuestas coincidieron de forma exacta.

ii. Casos en donde hay un tag "srai" asociado a un tag "category"

Descripción:

Buscar 5 "category"s con tag "srai" asociado y utilizar el contenido de tag "pattern" como frase de "entrada" en ALICE y en myALICE a través de la consola de *testing*. Verificar que las respuestas dadas son iguales.

Ejecución:

Se eligieron los siguientes *patterns*:

- 1 ALICEBOT
- 2 A BOY
- 3 ARE YOU ABSOLUTELY CRAZY
- 4 I AM PERFECTLY SURE
- 5 WAKE UP

Resultados:

Caso	respuesta ALICE	respuesta myAlice
1	Can I help you?	can i help you?
2	A little kid? he...	a little kid? ...
3	I think the polite term nowadays is "mentally ill".	I think the polite term nowadays is "mentally ill".
4	Are you being sarcastic or expressing certitude?	are you being sarcastic or expressing certitude?
5	i am fully conscious.	i am fully conscious.

Tabla 5.4.2.1.2: Resultados de pruebas: myAlice sin mejoras, tag srai

Evaluación de Resultados:

Los resultados son idénticos, excepto por la palabra "he" en el segundo caso, que

proviene de una variable de sesión, no *seteada* previamente en myAlice.

iii. Casos en donde hay un *tag* "that" asociado a un *tag* "category"

Descripción:

Buscar 5 "category"s con *tag* "that" asociado y utilizar el contenido de *tag* "pattern" como frase de "entrada" después de haber usado el contenido del *tag* "that" en ALICE y en myALICE a través de la consola de *testing*. Verificar que las respuestas dadas son iguales.

Ejecución:

Se eligieron los siguientes *patterns* iniciales:

- 1 BOT QUESTION
- 2 WHAT DOES A L MEANS?
- 3 BOT FAVORITECOLOR
- 4 JE NE FERAIS PAS
- 5 BOT FAVORITESPORT

En cada uno de los casos el chatbot respondió lo que se muestra a continuación, junto con la segunda tanda de *patterns* utilizados como entrada:

- | | | |
|---|---|---------------------|
| 1 | what is your favorite movie | The Jacket |
| 2 | alice equals artificial linguistic internet computer entity | WHAT DOES THAT MEAN |
| 3 | green | RED |
| 4 | ok | OK |
| 5 | hockey | WHAT TEAM? |

Resultados:

Caso	respuesta ALICE	respuesta myAlice
1	I've never seen it. What did you like about it?	i've never seen . what did you like about it?
2	It is just an acronym.	it is just an acronym.
3	Blue.	blue.
4	Is there an echo in here?	is there an echo in here?
5	My favorite team is Atlanta.	my favorite team is Atlanta.

Tabla 5.4.2.1.3: Resultados de pruebas: myAlice sin mejoras, *tag* that

Evaluación de Resultados:

Los resultados en ambos casos son idénticos, concuerda con lo esperado.

iv. Casos en donde hay un *tag* "random" asociado al *tag* "template" de "category"

Descripción:

Buscar 5 "category"s con tag "random" asociado y utilizar el contenido de *tag* "pattern" como frase de "entrada" en ALICE y en myALICE a través de la consola de *testing*. Verificar que las respuestas dadas son iguales o pertenezcan a la misma asociación . Probar varias veces y verificar que no siempre responda igual.

Ejecución:

Se eligieron los siguientes *patterns* iniciales, se muestran a continuación junto las posibles respuestas de ALICE asociadas.

Caso	Patterns	Posibles valores
1	WE HAVE CHATTED	1. With the same screen name? 2. Right here online? 3. What did we talk about?
2	WHAT IS AFRICA	1. A continent of the eastern hemisphere S of the Mediterranean and adjoining Asia on NE. 2. Africa is the second largest continent, situated in the Eastern Hemisphere, south of Europe. It is 11,677,000 square miles.
2	A LOT OF THINGS	1. Like what? 2. Name a few. 3. For example...
4	ARE YOU FREE	1. "I am as free as a bird now, and this bird you cannot change" -- Skynyrd. 2. You can download me for free!
5	BILL GATES (IS BAD)	1. "I am as free as a bird now, and this bird you cannot change" -- Skynyrd. 2. You can download me for free!

Tabla 5.4.2.1.4: Ejecución de pruebas: myAlice sin mejoras, tag random

Resultados:

Se muestra a continuación los resultados obtenidos, en cada caso se ingresó el patrón escogido como frase de entrada en cada uno de los dos chatbos: ALICE y myAlice hasta

que se devolvieron frases diferentes.

Caso	Respuesta ALICE 01	Respuesta ALICE 02
1	What did we talk about?	Right here online?
2	A continent of the eastern hemisphere S of the Mediterranean and adjoining Asia on NE.	Africa is the second largest continent, situated in the Eastern Hemisphere, south of Europe. It is 11,677,000 square miles.
3	Like what?	Name a few.
4	"I am as free as a bird now, and this bird you cannot change" -- Skynyrd.	You can download me for free!
5	I heard he was rich. But I didn't know he IS BAD.	A lot of people say that about him.

Tabla 5.4.2.1.5: Resultados de pruebas: myAlice sin mejoras, tag random 01

Caso	Respuesta myAlice 01	Respuesta myAlice 02
1	with the same screen name?	right here online?
2	africa is the second largest continent, situated in the eastern hemisphere, south of europe. it is 11,677,000 square miles.	a continent of the eastern hemisphere s of the mediterranean and adjoining asia on ne.
3	for example...	like what?
4	"i am as free as a bird now, and this bird you cannot change" -- skynyrd	you can download me for free!
5	a lot of people say that about him.	a lot of people say that about him.

Tabla 5.4.2.1.6: Resultados de pruebas: myAlice sin mejoras, tag random 02

Evaluación de Resultados:

Al evaluar el comportamiento de "ALICE" en relación con los tags "random" se observó que cualquier respuesta es probable, y que es común que devuelva dos veces consecutivas la misma respuesta como es de esperarse siendo que la función de probabilidad con la cual se elige una de las posibles frases es una uniforme. Este mismo comportamiento se encontró en "myAlice". En ambos chatbots, hubo que repetir la frase entrada en algunos casos, tres o cuatro veces para obtener la respuesta distinta. Pero este es el comportamiento esperado.

v. Casos donde hay *wildcars* dentro del texto del tag "pattern"

Descripción:

Buscar 5 "category"s cuyo *tag* "pattern" use *wildcards* y utilizar su contenido como frases de "entrada" en ALICE y en myALICE a través de la consola de *testing*. Reemplazando los *wildcards* por palabras. Verificar que las respuestas dadas son iguales.

Ejecución:

Se muestran a continuación los diferentes *patterns* escogidos junto con el reemplazo que se efectuó en el lugar del *wildcard* para obtener la frase de entrada que se utilizó en cada caso.

Caso	Pattern	Frase de entrada
1	WE ARE FRIENDLY *	WE ARE FRIENDLY TABLES
2	WE HAVE BEEN DATING *	WE HAVE BEEN DATING APLES
3	I AM * POUNDS	I AM noword POUNDS
4	I AM * THAN YOU	I AM DEVELOPEST THAN YOU
5	ARE YOU FUNDED *	ARE YOU FUNDED THING

Tabla 5.4.2.1.7: Ejecución de pruebas: myAlice sin mejoras, wildcards

Resultados:

Se muestran a continuación los resultados obtenidos en cada uno de los chatbots.

Caso	ALICE	myAlice
1	I like friendly people.	i like friendly people.
2	Tell me about your feelings for this person.	tell me about your feelings for this person.
3	You are average for your height.	you are average for your height.
4	Interesting comparison. Maybe I can become more DEVELOPEST.	interesting comparison. maybe i can become more developest.
5	I am the product of voluntary contributions and donations.	i am the product of voluntary contributions and donations.

Tabla 5.4.2.1.8: Resultados de pruebas: myAlice sin mejoras, wildcards

Evaluación de Resultados:

Los resultados son los esperados.

vi. Casos en donde se utilizan variables dentro del *tag* "template"

Descripción:

Buscar 2 "category"s cuyo *tag* "template" use variables y utilizar el contenido del *tag* "pattern" asociado como frase de "entrada" en ALICE y en myALICE a través de la consola de

testing. Previamente habiendo utilizado como frases de entrada los casos necesarios para *setear* los valores de dichas variables. Verificar que las respuestas dadas son iguales.

Ejecución:

Se buscaron dos pares de *tags* "category". Un par que contenga la opción de AIML de *setear* variables y el otro que utilicé dichas variables. Hecho esto, se ejecutaron las siguientes frases en ambos chatbots a fin de *setear* las variables que se muestran a continuación:

frase 01: "MY NAME IS MIKE", para *setear* la variable "name".

frase 02: "MY WIFE ANGELA", para *setear* la variable "she".

Luego se ejecutaron los *patterns* siguientes, que se muestran con el contenido del *tag* "template" asociado en la nomenclatura de AIML.

Pattern 01: "A PLUS",

Respuesta asociada en AIML "Au revoir, <get name="name"/>."

Pattern 02: "DO YOU KNOW HER" ,

Respuesta asociada en AIML "<get name="she"/>? Do I know her?"

Resultados:

caso	ALICE	myALICE
1	Au revoir, Mike.	au revoir, mike.
2	Your wife? Do I know her?	your wife? do i know her?

Tabla 5.4.2.1.9: Resultados de pruebas: myAlice sin mejoras, variables

Evaluación de Resultados:

Los resultados son los esperados; los *tags* "set" y "get" de AIML que sirven respectivamente para *setear* variables de sesión y utilizarlas fueron convertidos en "myAlice" de forma correcta de manera que el resultado observado es el mismo.

vii. Casos en donde se utiliza el *tag* "star" dentro de un *tag* "template"**Descripción:**

Buscar 2 "category"s cuyo *tag* "template" use el *tag* "star" y utilizar el contenido del *tag* "pattern" asociado como frase de "entrada" en ALICE y en myALICE a través de la consola de *testing*. Verificar que las respuestas dadas son iguales.

Ejecución:

Se escogieron los siguientes *tags* "pattern", que se muestran junto con sus respectivos *tags* "templates" asociados.

caso	Pattern	Template
1	A * ROBOT	What would a <person/> robot be like?
2	IS A * A *	Not unless a <person><star index="2"/></person> is a <person/>.

Tabla 5.4.2.1.10: Ejecución de pruebas: myAlice sin mejoras, tag star

Para cada caso se reemplazaron los *wildcards* por palabras formándose las siguientes frases: "A CRAZY ROBOT" y "IS A CAT A DOG"

Resultados:

caso	ALICE	myALICE
1	What would a CRAZY robot be like?	what would a crazy robot be like?
2	Not unless a DOG is a CAT.	not unless a dog is a cat.

Tabla 5.4.2.1.11: Resultados de pruebas: myAlice sin mejoras, tag star

Evaluación de Resultados:

Los resultados son los esperados, hay que notar que este caso de prueba es bastante potente porque prueba diferentes cosas: primero que los *wildcards* (*) funcionan correctamente ya que las frases de entrada lograron coincidir con patrones incompletos, que utilizaban *wildcards*. Lo segundo que prueba es que el chatbot logre identificar y aislar la porción de texto que coincidió con el respectivo *wildcard* y que pueda reformular la respuesta haciendo un reemplazo con dicha porción de texto. Lo tercero que se prueba es la capacidad de utilizar más de un *wildcard* en un patrón y que este a su vez utilice la porción de texto coincidente con cada uno de los *wildcards* en la reformulación de la respuesta.

viii. Casos en donde el tag "category" está dentro de un tag "topic"

Descripción:

Buscar 2 "category"s dentro de un tag "topic" y utilizar el contenido del tag "pattern" asociado como frase de "entrada" en ALICE y en myALICE a través de la consola de *testing*. Habiendo utilizado previamente como frases de entrada los casos necesarios para *setear* el valor de "topic". Verificar que las respuestas dadas son iguales.

Ejecución:

Se eligieron los siguientes tópicos: "ARCHITECT" y "ENGINEER". Ambos tópicos (*topic*) se *setearon* a través de la frase: "DID I CAUSE **", en donde el carácter "*" fue reemplazado por la palabra correspondiente.

Luego se buscó hacer coincidir la frase de entrada con el *pattern* "*", que realiza una coincidencia con cualquier frase de entrada dentro del *topic* dado. Se muestra a continuación los *patterns* con sus *tag template* asociados.

caso	Pattern	Topic	Template
1	*	ARCHITECT	1. Where is your office located? 2. What size is your office? 3. Do you specialize in certain building types or designs? 4. What types of projects have you been involved in? 5. Do you find various state regulations are eroding your profession? 6. Is it getting difficult to do your job because of all the interference? 7. Do you find yourself with many constraints on your current project? 8. Do you find yourself moving into kinds of work you hadn't anticipated?
2	*	ENGINEER	1. "Being a computer engineer is neither immoral nor illegal." 2. What kind of engineering do you do? 3. What's your speciality in your field? 4. Do you work in industry, for a firm or academics? 5. What project are you involved in now? 6. With so much coming out everyday, can we maintain our competitive edge? 7. How does what you do impact how we live?

Tabla 5.4.2.1.12: Ejecución de pruebas: myAlice sin mejoras, tag topic

Resultados:

Se muestran a continuación los resultados de ambos chatbots.

caso	ALICE	myALICE
1	Is it getting difficult to do your job because of all the interference?	do you find various state regulations are eroding your profession?
2	"Being a computer engineer is	"being a computer engineer is neither immoral

	neither immoral nor illegal." .	nor illegal."
--	---------------------------------	---------------

Tabla 5.4.2.1.13: Resultados de pruebas: myAlice sin mejoras, tag topic

Evaluación de Resultados:

Los resultados son los esperados, la variable "topic" se seteo correctamente y las respuestas a los frases de entrada coincidieron con las asociadas a dicho tópico.

5.4.2.2 Verificación del detector de tiempos verbales

Pruebas:

i. verificar detección del tiempo "future Continuous"

Ejecución:

Descripción:

Amar un archivo con 10 oraciones en inglés en "Future Continuous". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You will be waiting for her when her plane arrives tonight
2. Will you be waiting for her when her plane arrives tonight
3. You will not be waiting for her when her plane arrives tonight
4. You are going to be waiting for her when her plane arrives tonight
5. Are you going to be waiting for her when her plane arrives tonight
6. You are not going to be waiting for her when her plane arrives tonight
7. I will be watching TV when she arrives tonight
8. I will be waiting for you when your bus arrives
9. I am going to be staying at the Madison Hotel if anything happens and you need to contact me
10. He will be studying at the library tonight so he will not see Jennifer when she arrives

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Future Continuous
2	Future Continuous
3	Future Continuous
4	Future Continuous
5	Future Continuous
6	Future Continuous
7	Future Continuous
8	Future Continuous
9	Future Continuous
10	Future Continuous

Tabla 5.4.2.2.1: Resultados de pruebas: tiempos verbales: "future continuous"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

ii. verificar detección del tiempo "Future Perfect Continuous"**Descripción:**

Armar un archivo con 10 oraciones en inglés en "Future Perfect Continuous". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You will have been waiting for more than two hours when her plane finally arrives
2. Will you have been waiting for more than two hours when her plane finally arrives
3. You will not have been waiting for more than two hours when her plane finally arrives
4. You are going to have been waiting for more than two hours when her plane finally arrives
5. Are you going to have been waiting for more than two hours when her plane finally arrives
6. You are not going to have been waiting for more than two hours when her plane finally arrives
7. They will have been talking for over an hour by the time Thomas arrives
8. She is going to have been working at that company for three years when it finally closes
9. James will have been teaching at the university for more than a year by the time he leaves for Asia
10. How long will you have been studying when you graduate

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Future Perfect Continuous
2	Future Perfect Continuous
3	Future Perfect Continuous
4	Future Perfect Continuous
5	Future Perfect Continuous
6	Future Perfect Continuous
7	Future Perfect Continuous
8	Future Perfect Continuous
9	Future Perfect Continuous
10	Future Perfect Continuous

Tabla 5.4.2.2: Resultados de pruebas: tiempos verbales: "Future Perfect Continuous"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

ii. verificar detección del tiempo "Future Perfect"

Descripción:

Armar un archivo con 10 oraciones en inglés en "Future Perfect". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You will have perfected your English by the time you come back from the U.S
2. Will you have perfected your English by the time you come back from the U.S.
3. You will not have perfected your English by the time you come back from the U.S
4. You are going to have perfected your English by the time you come back from the U.S
5. Are you going to have perfected your English by the time you come back from the U.S.
6. You are not going to have perfected your English by the time you come back from the U.S
7. By next November I will have received my promotion
8. By the time he gets home she is going to have cleaned the entire house
9. I am not going to have finished this test by 3 o'clock
10. Will she have learned enough Chinese to communicate before she moves to Beijing

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Future Perfect
2	Future Perfect
3	Future Perfect
4	Future Perfect
5	Future Perfect
6	Future Perfect
7	Future Perfect
8	Future Perfect
9	Future Perfect
10	Future Perfect

Tabla 5.4.2.2.3: Resultados de pruebas: tiempos verbales: "Future Perfect"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

iv. verificar detección del tiempo "Past Continuous"

Descripción:

Armar un archivo con 10 oraciones en inglés en "Past Continuous". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. Last night at 6 PM I was eating dinner
2. You were studying when she called
3. Were you studying when she called
4. You were not studying when she called
5. I was watching TV when she called
6. When the phone rang she was writing a letter
7. While we were having the picnic it started to rain
8. What were you doing when the earthquake started
9. I was listening to my iPod so I didn't hear the fire alarm
10. You were not listening to me when I told you to turn the oven off

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Past Continuous
2	Past Continuous
3	Past Continuous
4	Past Continuous
5	Past Continuous
6	Past Continuous
7	Past Continuous
8	Past Continuous
9	Past Continuous
10	Past Continuous

Tabla 5.4.2.2.4: Resultados de pruebas: tiempos verbales: "Past Continuous"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

v. verificar detección del tiempo "Past Perfect Continuous"

Descripción:

Armar un archivo con 10 oraciones en inglés en "Past Perfect Continuous".
Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You had been waiting there for more than two hours when she finally arrived
2. Had you been waiting there for more than two hours when she finally arrived
3. You had not been waiting there for more than two hours when she finally arrived
4. They had been talking for over an hour before Tony arrived
5. She had been working at that company for three years when it went out of business
6. How long had you been waiting to get on the bus
7. Mike wanted to sit down because he had been standing all day at work
8. James had been teaching at the university for more than a year before he left for Asia
9. How long had you been studying Turkish before you moved to Ankara
10. I had not been studying Turkish very long

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Past Perfect Continuous
2	Past Perfect Continuous
3	Past Perfect Continuous
4	Past Perfect Continuous
5	Past Perfect Continuous
6	Past Perfect Continuous
7	Past Perfect Continuous
8	Past Perfect Continuous
9	Past Perfect Continuous
10	Past Perfect Continuous

Tabla 5.4.2.2.5: Resultados de pruebas: tiempos verbales: "Past Perfect Continuous"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

vi. verificar detección del tiempo "Past Perfect"

Descripción:

Armar un archivo con 10 oraciones en inglés en "Past Perfect". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You had studied English before you moved to New York
2. Had you studied English before you moved to New York
3. You had not studied English before you moved to New York
4. I had never seen such a beautiful beach before I went to Kauai
5. I did not have any money because I had lost my wallet
6. Tony knew Istanbul so well because he had visited the city several times
7. Had Susan ever studied Thai before she moved to Thailand
8. She had visited her Japanese relatives once in 1993 before she moved in with them in 1996
9. They felt bad about selling the house because they had owned it for more than forty years
10. We were not able to get a hotel room because we had not booked in advance

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Past Perfect
2	Past Perfect
3	Past Perfect
4	Past Perfect
5	Past
6	Past
7	Past Perfect
8	Past Perfect
9	Past Perfect
10	Past Perfect

Tabla 5.4.2.2.6: Resultados de pruebas: tiempos verbales: "Past Perfect"

Evaluación de Resultados:

Los resultados, no son los esperados. La aplicación falló al detectar los casos 5 y 6, aunque fueron identificados como "Past" un tiempo verbal no completamente diferente. Pero el motivo del error es que la frase contiene una parte de la sentencia en pasado simple, y la siguiente en pasado perfecto. Con lo cual es suficientemente ambiguo como para que ambos casos sean validos. No es un error critico, y está el error dentro de un marco de tolerancia aceptable, por lo cual se contará el caso de prueba como exitoso.

vii. verificar detección del tiempo "Past"

Descripción:

Armaz un archivo con 10 oraciones en inglés en "Past". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. He arrived from the airport at 8:00, checked into the hotel at 9:00, and met the others at 10:00
2. She washed her car
3. You called Debbie
4. Did you call Debbie
5. You did not call Debbie
6. I saw a movie yesterday
7. I didn't see a play yesterday
8. Last year, I traveled to Japan
9. Last year, I didn't travel to Korea
10. Did you have dinner last night

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	unknown
2	Past
3	Past
4	Past
5	Past
6	Past
7	Past
8	Past
9	Past

10	Past
----	------

Tabla 5.4.2.2.7: Resultados de pruebas: tiempos verbales: "Past"

Evaluación de Resultados:

Se cometió un error de 10%, está dentro del límite de tolerancia. Con lo cual el caso de prueba se lo toma como exitoso.

viii. verificar detección del tiempo "Present Continuos"

Descripción:

Amar un archivo con 10 oraciones en inglés en "Present Continuos". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You are watching TV
2. Are you watching TV
3. You are not watching TV
4. You are learning English now
5. You are not swimming now
6. Are you sleeping
7. I am sitting
8. I am not standing
9. Is he sitting or standing
10. They are reading their books

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Present Continuous
2	Present Continuous
3	Present Continuous
4	Present Continuous
5	Present Continuous
6	Present Continuous
7	Present Continuous
8	Present Continuous
9	Present Continuous

10	Present Continuous
----	--------------------

Tabla 5.4.2.2.8: Resultados de pruebas: tiempos verbales: "Present Continuous"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

ix. verificar detección del tiempo "Present Perfect Continuos"

Descripción:

Amar un archivo con 10 oraciones en inglés en "Present Perfect Continuos". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. You have been waiting here for two hours
2. Have you been waiting here for two hours
3. You have not been waiting here for two hours
4. They have been talking for the last hour
5. She has been working at that company for three years
6. What have you been doing for the last 30 minutes
7. James has been teaching at the university since June
8. We have been waiting here for over two hours
9. Why has Nancy not been taking her medicine for the last three days
10. Recently I have been feeling really tired

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Present Perfect Continuous
2	Present Perfect Continuous
3	Present Perfect Continuous
4	Present Perfect Continuous
5	Present Perfect Continuous
6	Present Perfect Continuous
7	Present Perfect Continuous
8	Present Perfect Continuous
9	Present Perfect Continuous

10	Present Perfect Continuous
----	----------------------------

Tabla 5.4.2.2.9: Resultados de pruebas: tiempos verbales: "Present Perfect Continuous"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

x. verificar detección del tiempo "Present Perfect"

Descripción:

Amar un archivo con 10 oraciones en inglés en "Present Perfect". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. I think I have met him once before
2. There have been many earthquakes in California
3. People have traveled to the Moon
4. People have not traveled to Mars
5. Nobody has ever climbed that mountain
6. Has there ever been a war in the United States
7. Yes, there has been a war in the United States
8. I think I have seen that movie before
9. He has never traveled by train
10. Joan has studied two foreign languages

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Present Perfect
2	Simple Present
3	Present Perfect
4	Present Perfect
5	Present Perfect
6	Present Perfect
7	Present Perfect
8	Present Perfect
9	Present Perfect

10	Present Perfect
----	-----------------

Tabla 5.4.2.2.10: Resultados de pruebas: tiempos verbales: "Present Perfect"

Evaluación de Resultados:

Los resultados contienen un caso erróneo, el error sin embargo está dentro del margen tolerado. El caso se toma como exitoso.

xi. verificar detección del tiempo "Simple Future"

Descripción:

Amar un archivo con 10 oraciones en inglés en "Simple Future". Analizarlas con el detector.

Ejecución:

Casos escogidos:

1. Don't worry I'll be careful
2. You will help him later
3. Will you help him later
4. You will not help him later
5. You are going to meet Jane tonight
6. Are you going to meet Jane tonight
7. You are not going to meet Jane tonight
8. I will send you the information when I get it
9. I will translate the email so Mr. Smith can read it
10. Will you help me move this heavy table

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Simple Present
2	Simple Future
3	Simple Future
4	Simple Future
5	Simple Future
6	Simple Future
7	Simple Future
8	Simple Future
9	Simple Future

10	Simple Future
----	---------------

Tabla 5.4.2.2.11: Resultados de pruebas: tiempos verbales: "Simple Future"

Evaluación de Resultados:

Los resultados son los esperados, en todos los casos el tiempo verbal fue detectado de forma correcta.

xii. verificar la correcta detección de casos mezclados

Descripción:

Armar un archivo con 22 oraciones en inglés con diferentes tiempos verbales . Analizarlas con el detector.

Ejecución:

Se escogieron 22 oraciones, 2 de cada tiempo verbal. El orden de los tiempos verbales en los cuales se encuentran las oraciones es:

1. Future Continuous
2. Future Perfect Continuous
3. Future Perfect
4. Past Continuous
5. Past Perfect Continuous
6. Past Perfect
7. Past
8. Present Continuous
9. Present Perfect Continuous
10. Present Perfect
11. Simple Future
12. Future Continuous
13. Future Perfect Continuous
14. Future Perfect
15. Past Continuous
16. Past Perfect Continuous
17. Past Perfect
18. Past
19. Present Continuous
20. Present Perfect Continuous
21. Present Perfect
22. Simple Future

Los casos escogidos respectivamente son:

1. Are you going to be waiting for her when her plane arrives tonight
2. You will not have been waiting for more than two hours when her plane finally arrives
3. I am not going to have finished this test by 3 o'clock
4. At midnight we were still driving through the desert
5. You had only been waiting there for a few minutes when she arrived
6. They felt bad about selling the house because they had owned it for more than forty years
7. I studied French when I was a child
8. I am studying to become a doctor
9. Recently the work has been being done by John
10. This sentence means that you have not had the experience of going to France
11. I will translate the email so Mr. Smith can read it
12. At midnight tonight we will still be driving through the desert
13. How long will you have been studying when you graduate
14. You are not going to have perfected your English by the time you come back from the U.S
15. You were not listening to me when I told you to turn the oven off
16. James had been teaching at the university for more than a year before he left for Asia
17. We were not able to get a hotel room because we had not booked in advance
18. Last year, I didn't travel to Korea
19. I don't like them because they are always complaining
20. Why has Nancy not been taking her medicine for the last three days
21. My English has really improved since I moved to Australia
22. I won't tell anyone your secret

Resultados:

A continuación se muestra la respuesta del subsistema para cada caso.

Caso	Tiempo Encontrado
1	Future Continuous
2	Future Perfect Continuous
3	Future Perfect
4	Past Continuous
5	Past Perfect Continuous
6	Past Perfect
7	Past
8	Present Continuous
9	Present Perfect Continuous
10	Simple Present
11	Simple Future

12	Future Continuous
13	Future Perfect Continuous
14	Future Perfect
15	Past Continuous
16	Past Perfect Continuous
17	Past Perfect
18	Past
19	Present Continuous
20	Present Perfect Continuous
21	Present Perfect
22	Simple Future

Tabla 5.4.2.2.12: Resultados de pruebas: tiempos verbales: casos mezclados

Evaluación de Resultados:

Los resultados son los esperados, el error total es del 5%, con lo cual está dentro del límite de tolerancia.

5.4.2.3 Verificación del detector de frases de interrogación

Pruebas:

i. Detectar como positivas las frases que son preguntas

Descripción:

Crear un archivo de texto con 20 frases que sean preguntas. Verificar que se detectan positivamente

Ejecución:

Se generó una pequeña aplicación que levanta de un archivo de texto una a una las líneas y las evalúa. A continuación imprime por pantalla la sentencia seguida de la leyenda "TRUE" o bien "FALSE" en caso de que dicha sentencia sea o no una interrogación.

Se muestran a continuación los 20 casos escogidos (sin el signo de interrogación como fueron usados):

1. Will you be waiting for her when her plane arrives tonight
2. Are you going to have been waiting for more than two hours when her plane finally arrives
3. Are you going to have perfected your English by the time you come back from the U.S.
4. What were you doing while you were waiting
5. How long had you been studying Turkish before you moved to Ankara

6. Had you ever visited the U.S. before your trip in 2006
7. Did you have dinner last night
8. Why has Nancy not been taking her medicine for the last three days
9. Have you read the book yet
10. Will you make dinner
11. Can you play the piano
12. How old are you
13. When did you met her
14. Where are you from
15. Have you got a cat
16. Have you got a new car
17. Has your brother got a bike
18. Do you have a cat
19. Do you have a new car
20. Does your brother have a bike

Resultados:

Se muestran a continuación la impresión por pantalla del aplicativo descripto:

1. Will you be waiting for her when her plane arrives tonight => TRUE
2. Are you going to have been waiting for more than two hours when her plane finally arrives => TRUE
3. Are you going to have perfected your English by the time you come back from the U.S. => TRUE
4. What were you doing while you were waiting => TRUE
5. How long had you been studying Turkish before you moved to Ankara => TRUE
6. Had you ever visited the U.S. before your trip in 2006 => TRUE
7. Did you have dinner last night => TRUE
8. Why has Nancy not been taking her medicine for the last three days => TRUE
9. Have you read the book yet => TRUE
10. Will you make dinner => TRUE
11. Can you play the piano => TRUE
12. How old are you => TRUE
13. When did you met her => TRUE
14. Where are you from => TRUE
15. Have you got a cat => TRUE
16. Have you got a new car => TRUE
17. Has your brother got a bike => TRUE
18. Do you have a cat => TRUE
19. Do you have a new car => TRUE
20. Does your brother have a bike => TRUE

Evaluación de Resultados:

Los resultados fueron los esperados, todas las interrogaciones fueron detectadas como tales.

ii. Detectar como negativas las frases que NO son preguntas

Descripción:

Crear un archivo de texto con 20 frases que NO sean preguntas. Verificar que se detectan negativamente

Ejecución:

Se muestran a continuación las 20 frases seleccionadas:

1. You will be waiting for her when her plane arrives tonight
2. James will have been teaching at the university for more than a year by the time he leaves for Asia
3. You are going to have been waiting for more than two hours when her plane finally arrives
4. I will have been in London for six months by the time I leave
5. While John was sleeping last night someone stole his car
6. I was watching TV when she called
7. They had been talking for over an hour before Tony arrived
8. Betty failed the final test because she had not been attending class
9. By the time Alex finished his studies he had been in London for over eight years
10. We talked on the phone for thirty minutes
11. I am studying to become a doctor
12. I am not studying to become a dentist
13. Recently John has been doing the work
14. No, I have not met him
15. The phone is ringing
16. I'll get it
17. Africa is the world's second-largest and second most-populous continent, after Asia.
18. A group of monkeys may be referred to as a mission or a tribe.
19. Sure.
20. I won't do all the housework myself!

Resultados:

Se muestra a continuación la impresión por pantalla del aplicativo descripto:

1. You will be waiting for her when her plane arrives tonight => FALSE
2. James will have been teaching at the university for more than a year by the time he leaves for Asia => FALSE
3. You are going to have been waiting for more than two hours when her plane finally arrives => FALSE
4. I will have been in London for six months by the time I leave => FALSE

5. While John was sleeping last night someone stole his car => FALSE
6. I was watching TV when she called => FALSE
7. They had been talking for over an hour before Tony arrived => FALSE
8. Betty failed the final test because she had not been attending class => FALSE
9. By the time Alex finished his studies he had been in London for over eight years => FALSE
10. We talked on the phone for thirty minutes => FALSE
11. I am studying to become a doctor => FALSE
12. I am not studying to become a dentist => FALSE
13. Recently John has been doing the work => FALSE
14. No, I have not met him => FALSE
15. The phone is ringing => FALSE
16. I'll get it => FALSE
17. Africa is the world's second-largest and second most-populous continent, after Asia. => FALSE
18. A group of monkeys may be referred to as a mission or a tribe. => FALSE
19. Sure. => FALSE
20. I won't do all the housework myself! => FALSE

Evaluación de Resultados:

Los resultados fueron los esperados, ninguna frase fue detectada como interrogación.

5.4.2.4 Verificación del mapa generado, con *mapeador 01*

La red neuronal tomaba como input un vector de 15 posiciones de números expresados en punto flotante. La posición 0 del vector indica si la frase es una interrogación (1) o no (0). La posición 1 indica el tiempo verbal, el cual puede tomar uno de los siguientes valores: 0.0; 0.5; 0.7; -1.0; -1.3; 0.8; 0.6; -1.4; -1.5; 1.1; 1.2; 1.3; 1.5; 1.9 para los tiempos verbales respectivos: infinitive, simple_present, present_continuous, past, past_continuous, present_perfect, present_perfect_continuous, past_perfect, past_perfect_continuous, simple_future, future_continuous, future_perfect, future_perfect_continuous, conditional. El resto de las posiciones del vector representan en orden, a los diferentes tipos de palabras que conforman la oración (Las primeras 13 palabras.) Los tipos de palabras y sus valores numéricos asociados son:

auxiliary words 0.1

noun 1.9

adjective 1.3

indicative 2.3

interrogative pronouns 2.0

verbs 1.5

personal pronouns first person: 2.5

personal pronouns third person: 3.0

personal pronouns second person: 3.5

Esquemáticamente:

¿pregunta?	Tiempo	Tipo de	Tipo de	Tipo de	Tipo de	Tipo de
	verbal	palabra	palabra		palabra	palabra	palabra

Tabla 5.4.2.4.1: Esquema mapa de entrada de la RNA 01

Pruebas:

i. Frases aleatorias y su correcta conversión en el mapa de entrada

Descripción:

Crear un archivo de texto con 10 frases escogidas de forma aleatoria. Convertir las 10 frases en 10 mapas de entrada para la RNA. En el mapa: Verificar que el número que indica si es interrogación o no sea uno o cero según corresponda. Verificar que el número que indica el tiempo verbal tenga el valor correcto. Verificar que cada palabra fue reemplazada por un número y que dicho número esté asociado al tipo de palabra.

Ejecución:

Se corrió una aplicación java especialmente creada para generar el input de la RNA contra un archivo de 10 casos que se muestra a continuación:

1. How are you?
2. Not bad. Slightly sleepy.
3. Do you have one?
4. Is that good?
5. Is that fun?
6. Is that why it's part-time?!
7. Yes, that sounds extremely civilised.
8. Have you done any interesting research recently (that isn't classified?)
9. Eek. That does not sound like fun.
10. The other entity and I are discussing our favourite film.

Resultados:

A continuación se muestra el output del aplicativo anterior. Cada línea representa una frase, y el carácter ";" separa las diferentes posiciones del vector.

- 01)** 1.0;0.5;2.0;1.9;3.5;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0
02) 0.0;0.0;1.3;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;1.0
03) 1.0;0.5;0.0;3.5;0.1;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0
04) 1.0;0.5;1.5;2.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0
05) 1.0;0.5;1.5;2.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0
06) 1.0;0.5;1.5;2.3;1.9;3.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;0.0
07) 0.0;0.0;1.9;2.3;0.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0
08) 1.0;0.5;0.1;3.5;1.5;1.3;1.3;1.9;1.5;0.0;0.0;1.9;0.0;0.0;0.0;1.0
09) 0.0;0.5;0.0;2.3;0.0;1.5;1.9;1.9;1.9;0.0;0.0;0.0;0.0;0.0;0.0
10) 0.0;0.7;0.0;1.3;1.9;0.0;1.9;1.9;0.0;2.5;1.9;1.9;0.0;0.0;0.0;1.0

Evaluación de Resultados:

En todos se verifica de forma correcta si es o no interrogación (primera posición del vector.) Los tiempos verbales detectados son "simple present", "infinitive" y en el último caso "present continuous" que coinciden con los tiempos de las oraciones, excepto en el caso 7 en que la oración está en "simple present" y fue detectada como en "infinitive". Por último las palabras en las oraciones fueron correctamente detectadas y convertidas a números respetando su posición original. Aquellas palabras que no fueron detectadas aparecen como "0.0", pero eso se debe no al generador de *input* sino a que la base de datos "Lexico" no contiene una lista exhaustiva de palabras.

Los resultados fueron positivos.

5.4.2.5 Verificación del mapa generado, con *mapeador 02*

Para la generación del *input* de la segunda RNA utilizada se cambió el mapa de entrada para hacerlo más sensible. Lo que se hizo fue aumentar el número de neuronas de entrada. Este nuevo mapa reserva una neurona especial para un valor de entrada que indica si la frase es o no una pregunta, al igual que en el caso anterior, pero se eliminó la neurona que respondía ante el tiempo verbal encontrado identificado en la oración. El resto del mapa de entrada se definió como una matriz de 15 x 5 números de punto flotante, en donde la longitud, 15, representa una a una las primeras 15 palabras de la oración de entrada y el ancho o profundidad, 5, indica, según la posición que tome un valor distinto de cero, que tipo de palabra es. En donde la posición 0 del vector vertical toma un valor distinto de cero si la palabra es auxiliar o bien si está es un pronombre interrogativo (*interrogative pronouns*). En el caso de una palabra auxiliar, el valor que toma es de 0.2, en el caso de un pronombre interrogativo es de 3.0 (en este caso se verá reforzado además por la neurona que indica que es una pregunta.). La posición 1 del vector

vertical toma valor igual a 1.0 si la palabra es un verbo (*verb*) o bien si es un adverbio (*adverb*). La posición 2 toma un valor igual a 1 si es un adjetivo (*adjective*). La posición 3 toma un valor igual a 1 si es un sustantivo (*noun*) común o igual a 2 si es un sustantivo propio. Por último la posición 4 toma un valor distinto de cero si la palabra es un pronombre personal (*personal pronoun*) o un pronombre demostrativo (*demonstrative pronoun*). En estos casos puede tomar diferentes valores, tomará el valor 1 para pronombres personales en primera persona, 2 para pronombres personales en tercera persona y 3 para pronombres personales en segunda persona. Para pronombres demostrativos toma el valor 2.5.

Esquemáticamente el mapa quedó de la siguiente manera:

1º palabra 15º palabra

¿Pregunta?	¿auxiliar o pronombre interrogativo?	...	¿auxiliar o pronombre interrogativo?
	¿adverbio o verbo?	...	¿adverbio o verbo?
	¿adjetivo?	...	¿adjetivo?
	¿sustantivo?	...	¿sustantivo?
	¿pronombre personal o demostrativo?	...	¿pronombre personal o demostrativo?

Tabla 5.4.2.5.1: Esquema mapa de entrada de la RNA 02

Pruebas:

i. Frases aleatorias y su correcta conversión en el mapa de entrada

Descripción:

Crear un archivo de texto con 10 frases escogidas de forma aleatoria. Convertir las 10 frases en 10 mapas de entrada para la RNA. En el mapa: Verificar que el número que indica si es interrogación o no sea uno o cero según corresponda. Verificar que el número que indica el tiempo verbal tenga el valor correcto. Verificar que cada palabra fue reemplazada por un número y que dicho número esté asociado al tipo de palabra.

Ejecución:

Se corrió una aplicación java especialmente creada para generar el input de la RNA contra un archivo de 10 casos que se muestra a continuación:

1. How are you?

2. Not bad.
3. Slightly sleepy.
4. Me neither.
5. What do you normally do, when you aren't trying to persuade people you are human?
6. Is that fun?
7. Is that why it's part-time?!
8. Yes, that sounds extremely civilised.
9. Have you done any interesting research recently?
10. That does not sound like fun.

Resultados:

A continuación se muestran los resultados obtenidos, hay que tener en cuenta que el *output* de la aplicación es una archivo de texto, y cada frase convertida en mapa es representada en una sola línea y no como se la muestra aquí donde las palabras avanzan de forma vertical y no horizontal como se mostró en el esquema. El primer valor corresponde a si la frase es interrogación o no.

5.0;	0.0;	0.0
3.0;0.0;0.0;0.0;0.0;	0.0;0.0;1.0;0.0;0.0;	0.0;1.0;0.0;0.0;0.0;
0.0;0.0;0.0;1.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;1.0;0.0;0.0;
0.0;0.0;0.0;0.0;3.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;

	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;	5.0;	5.0;
0.0;0.0;0.0;0.0;1.0;	3.0;0.0;0.0;0.0;0.0;	0.0;1.0;0.0;0.0;0.0;
0.0;0.0;0.0;2.0;0.0;	0.0;0.0;0.0;2.0;0.0;	0.0;0.0;0.0;0.0;2.5;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;3.0;	0.0;0.0;0.0;1.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;1.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	3.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;3.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;1.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;1.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;1.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;2.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;1.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;1.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;3.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;1.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;		0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;		0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;		0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;		0.0;0.0;0.0;0.0;0.0;
5.0;	0.0;	5.0;
0.0;1.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.2;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;2.5;	0.0;0.0;0.0;0.0;2.5;	0.0;0.0;0.0;0.0;3.0;
0.0;0.0;0.0;1.0;0.0;	0.0;0.0;0.0;2.0;0.0;	0.0;1.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;2.0;	0.0;1.0;0.0;0.0;0.0;	0.0;0.0;0.0;2.0;0.0;
0.0;1.0;0.0;0.0;0.0;	0.0;0.0;1.0;0.0;0.0;	0.0;0.0;1.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;1.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;1.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	

0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;	0.0;0.0;0.0;0.0;0.0;
0.0; 0.0;0.0;0.0;0.0;2.5; 0.0;0.0;0.0;2.0;0.0; 0.0;1.0;0.0;0.0;0.0; 0.0;0.0;0.0;1.0;0.0; 0.0;0.0;0.0;1.0;0.0; 0.0;0.0;0.0;1.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0; 0.0;0.0;0.0;0.0;0.0;		

Tabla 5.4.2.5.2: Resultados de pruebas: mapa 02 generado con frases aleatorias

Evaluación de Resultados:

Los resultados son los esperados. Las frases son convertidas a una serie de valores numéricos y luego estos son representados de forma ordenada como una tira de números. Las diferencias observadas entre los tipos reales de las palabras y los valores detectados responde a la incompletitud de la base de datos "Lexico" o bien a palabras catalogadas con más de un tipo.

En particular la detección de preguntas fue correcta en el 100% de los casos. Los tipos de las palabras en cambio fueron detectadas correctamente en un 70% de los casos, con excepción

de los pronombres personales, demostrativos e interrogativos que fueron detectados de forma correcta en el 100% de los casos. Estas limitaciones, de todas formas, no son del generador de mapa sino de los subsistemas involucrados.

5.4.2.6 Verificación del correcto funcionamiento de la RNA

Las pruebas sobre el primer perceptrón construido, se realizaron a través de la *interface* gráfica de "Joone". Se utilizó como algoritmo de aprendizaje *On-Line Backpropagation*. Para ello se construyó un archivo de 69 casos o frases y a cada frase se le asignó un valor "yes" o "no" dependiendo de si la frase era posible de responder sin información del contexto (yes) o no (no). Cada frase quedó expresada como se muestra en el ejemplo a continuación:

How are you? => yes

Luego de algunas pruebas se observó que el RMSE calculado no disminuía de cierto valor por lo que se optó por hacer uso de una red SOM (Kohonen) para, primero dividir el conjunto original en subconjuntos de similares características y luego aplicar el perceptrón a cada subconjunto de casos. Como esta metodología no dio los resultados esperados se optó por cambiar la arquitectura del perceptrón y hacer más sensible la capa de entrada.

Pruebas:

i. Probar variación de Iteraciones

Descripción:

Construir un conjunto de pruebas de 100 frases. Dichas frases serán tomadas de diálogos reales de personas de [Loebner, 2006] Convertir las frases en mapas, entrenar la red con parámetros típicos de "momentum" y "learning rate" y bajas iteraciones. Ver el error: "RMSE".

Ejecución:

Se tomaron 69 frases de los diálogos transcritos de [Loebner 2006] y se clasificó a cada una de las frases como se indicó anteriormente. Los archivos de prueba más los archivos convertidos en mapas de entrada pueden encontrarse en el repositorio *on-line* (ver apéndice):

Archivo de frases de entrada: "CA_01_no_processed.txt"

Archivo con los mapas de la RNA: "CA_01_processed.txt"

Se corrieron 3 casos variando la cantidad de Iteraciones de la RNA.

CASO 1:

Características de Red

<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	100
<i>learning rate</i>	0.7
<i>momentum</i>	0.6

Tabla 5.4.2.6.1: Ejecución de pruebas: RNA variación de iteraciones, CASO 1

CASO 2:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	1000
<i>learning rate</i>	0.7
<i>momentum</i>	0.6

Tabla 5.4.2.6.2: Ejecución de pruebas: RNA variación de iteraciones, CASO 2

CASO 3:

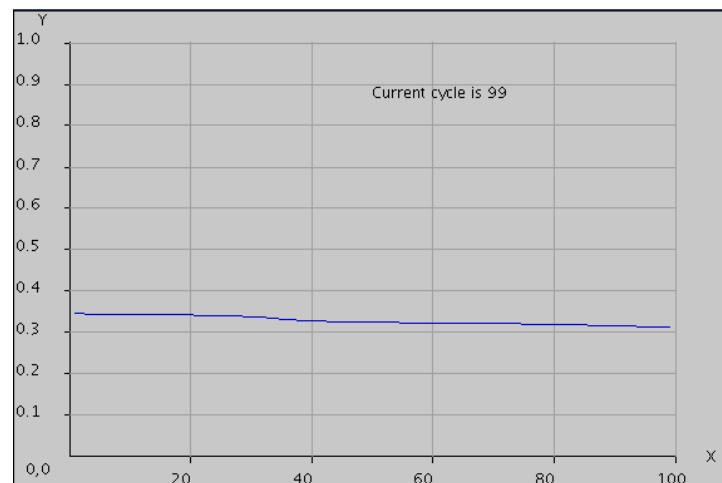
Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	10000
<i>learning rate</i>	0.7
<i>momentum</i>	0.6

Tabla 5.4.2.6.3: Ejecución de pruebas: RNA variación de iteraciones, CASO 3

Resultados:

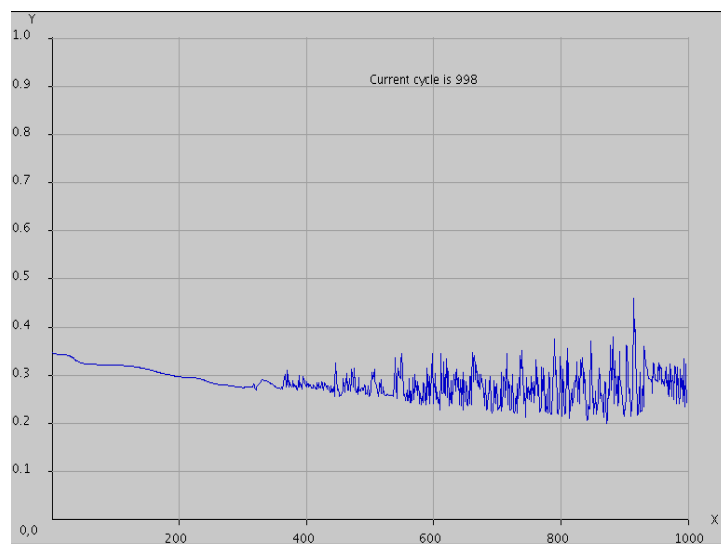
Se muestra a continuación el gráfico del RMSE (error cuadrático medio vs. los ciclos de entrenamiento de la RNA o *iteraciones*)

CASO 1:



RMSE: 0.3118563

CASO 2:



RMSE: 0.2495641

CASO 3:

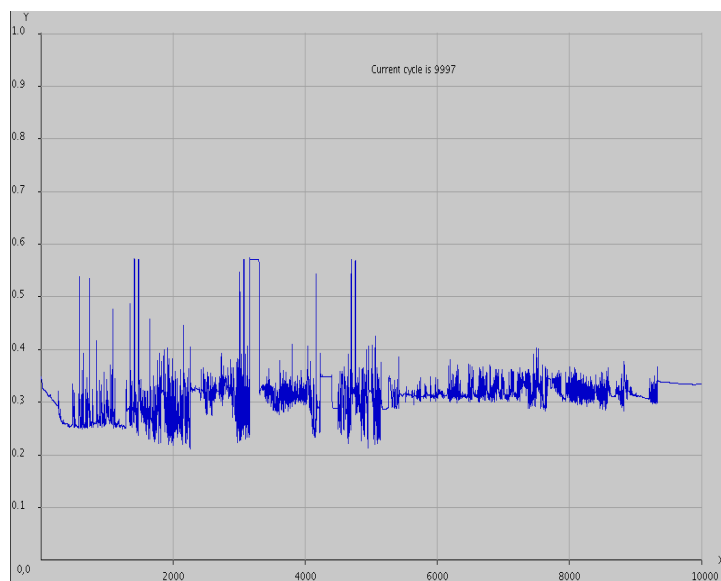


Gráfico 5.4.3: pruebas de RNA, ciclos caso 3

RMSE: 0.3340892

Evaluación de Resultados:

CASO 1: Se comenzó el entrenamiento con valores *standar*, sugeridos por la documentación de "Joone" pero la cantidad de ciclos de entrenamiento fue adrede muy baja, el valor RMSE obtenido es bastante alto, pero concuerda con el valor esperado para una primera arquitectura propuesta. En el gráfico se ve que la pendiente continua en descenso, con lo cual si se aumentan los ciclos el RMSE debería disminuir.

CASO 2: Si bien el RMSE es menor, se observa que a partir de las 400 iteraciones comienza a oscilar, la oscilación va en aumento hasta alrededor de las 900 iteraciones y luego parece decrecer.

CASO 3: En este gráfico se puede observar como la oscilación está dividida, tiene hasta las 5000 iteraciones aproximadamente una amplitud mucho mayor que entre las 5000 y las 9000 iteraciones. Luego parece estabilizarse alrededor del valor final obtenido que es de aproximadamente 0.33.

ii. Probar como se modifica el error al agregar una capa oculta**Descripción:**

Tomando los parámetros obtenidos en los casos anteriores agregar una capa oculta con una cantidad de neuronas igual a las neuronas de entrada sobre dos. Entrenar la red. Evaluar el error "RMSE"

Ejecución:

Se tomaron 69 frases de los diálogos transcritos de [Loebener 2006] y se clasificó a cada una de las frases como se indicó anteriormente. Los archivos de prueba, más los archivos convertidos en mapas de entrada y los resultados de la RNA sobre ellos pueden encontrarse en el apéndice.

Archivo de frases de entrada: "CA_01_no_processed.txt"

Archivo con los mapas de la RNA: "CA_01_processed.txt"

Se agregó una capa oculta a la arquitectura de perceptrón anterior, de 2 neuronas, dicha capa se ubicó entre la capa de salida de una neurona y la capa oculta de 7 neuronas. Con dicha arquitectura se corrieron 2 casos variando la cantidad de iteraciones de entrenamiento.

CASO 1:

Características de Red	
Input Layer	15
Hidden Layer	7
Second HiddenLayer	2
Output Layer	1
Características de Entrenamiento	
iteraciones	2000
<i>learning rate</i>	0.7
<i>momentum</i>	0.6

Tabla 5.4.2.6.3: Ejecución de pruebas: RNA variación de capa oculta, CASO 1

CASO 2:

Características de Red	
Input Layer	15
Hidden Layer	7
Second Hidden Layer	2
Output Layer	1
Características de Entrenamiento	
iteraciones	10000
<i>learning rate</i>	0.7
<i>momentum</i>	0.6

Tabla 5.4.2.6.4: Ejecución de pruebas: RNA variación de capa oculta, CASO 2

Resultados:

CASO 1:

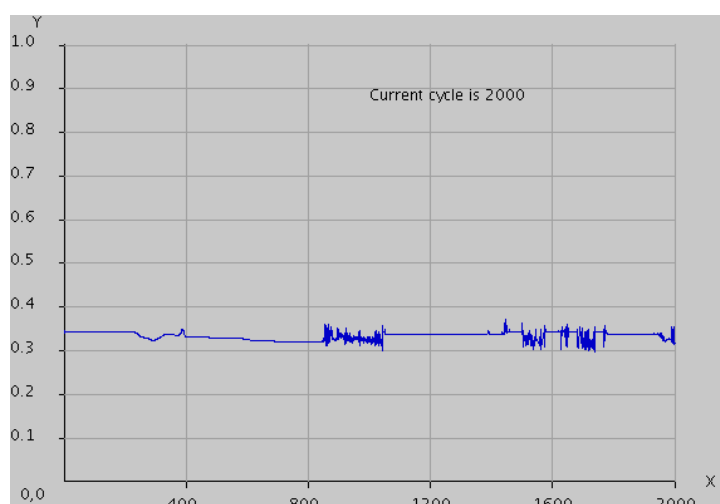


Gráfico 5.4.4: pruebas de RNA, capa oculta caso 1

RMSE: 0.3441952

CASO 2:

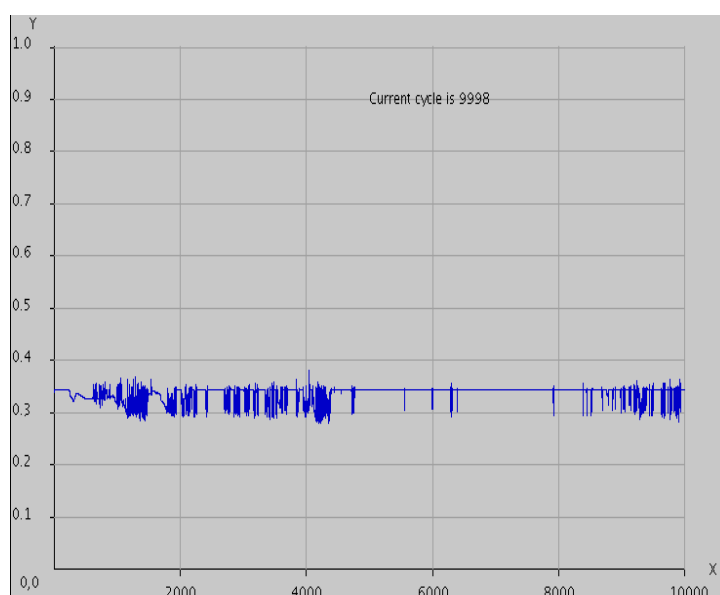


Gráfico 5.4.5: pruebas de RNA, capa oculta caso 2

RMSE: 0.3441942

Evaluación de Resultados:

Observando los gráficos de ambas corridas se puede apreciar como a medida que aumentan las iteraciones la RNA entra en ciclos de oscilación o ciclos de estabilización, sin embargo siempre se mantiene alrededor del mismo valor: 0.344194. El error es más alto o igual que en los casos en los cuales la capa oculta no existía, dicha capa aportó estabilización al modelo pero no mejoró su error cuadrático medio (RMSE).

iii. Probar variación de "Learning Rate" y "Momentum"

Descripción:

Entrenar la red con parámetros típicos de "momentum" y "learning rate" y el número óptimo de iteraciones encontrado previamente. Evaluar el error: "RMSE"

Ejecución:

Se tomaron 69 frases de los diálogos transcritos de [Loebener 2006] y se clasificó a cada una de las frases como se indicó anteriormente. Los archivos de prueba, más los archivos convertidos en mapas de entrada y los resultados de la RNA sobre ellos pueden encontrarse en el apéndice.

Archivo de frases de entrada: "CA_01_no_processed.txt"

Archivo con los mapas de la RNA: "CA_01_processed.txt"

En las pruebas anteriores se observó que el algoritmo *Backpropagation* generó oscilaciones, una forma de reducir dichas oscilaciones es disminuir el *momentum* [García Martínez, Servente, Pasquini 2003]. Por otro lado, para acelerar la convergencia del algoritmo se debe incrementar el valor de "learning rate" o tasa de aprendizaje. En las pruebas que se detallan a continuación se modificaron dichos valores buscando disminuir las oscilaciones y acelerar la convergencia.

CASO 1:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	2000
<i>learning rate</i>	0.8
<i>momentum</i>	0.5

Tabla 5.4.2.6.5: Ejecución de pruebas: RNA variación de "Learning Rate" y "Momentum", CASO 1

CASO 2:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7

<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	9000
<i>learning rate</i>	0.8
<i>momentum</i>	0.5

Tabla 5.4.2.6.6: Ejecución de pruebas: RNA variación de "Learning Rate" y "Momentum", CASO 2

CASO 3:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	5400
<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.7: Ejecución de pruebas: RNA variación de "Learning Rate" y "Momentum", CASO 3

Resultados:

CASO 1:

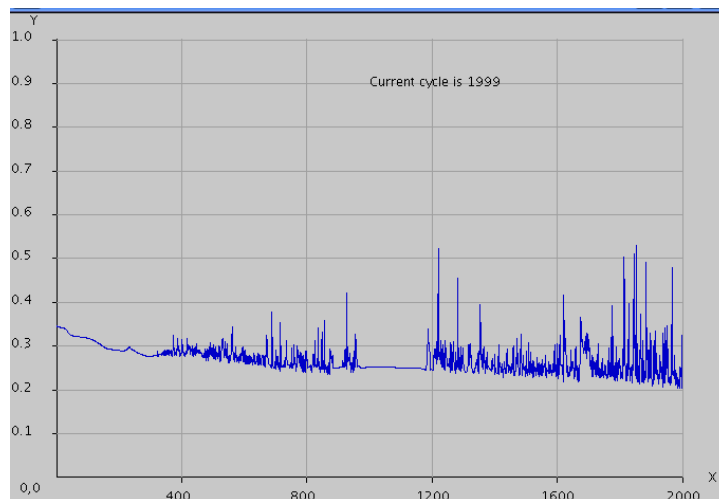


Gráfico 5.4.6: pruebas de RNA, learning rate, momentum caso 1

RMSE: 0.2048021

CASO 2:

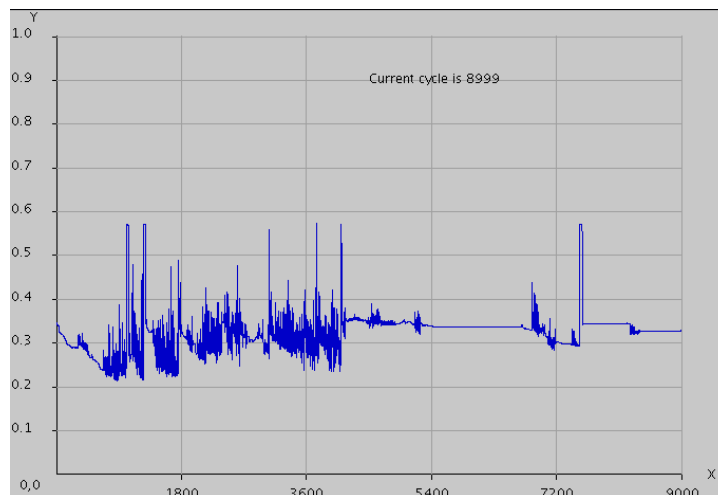


Gráfico 5.4.7: pruebas de RNA, learning rate, momentum caso 2

RMSE: 0.3288008

CASO 3:

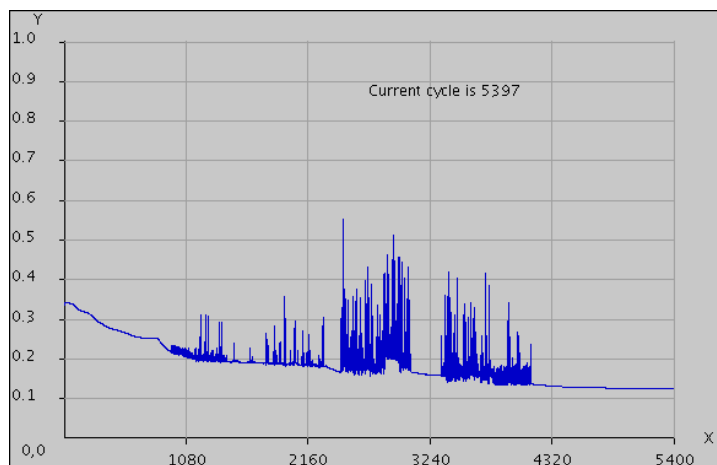


Gráfico 5.4.8: pruebas de RNA, learning rate, momentum caso 3

RMSE: 0.1248292

Evaluación de Resultados:

CASO 1: En este caso se puede observar que existe una oscilación para las 2000 iteraciones que continua.

CASO 2: A partir del caso anterior se mantienen los valores de *learning rate* y *momentum* y se aumentan las iteraciones para verificar si las oscilaciones continúan. En el gráfico se puede ver que el error se estabiliza, pero en un valor alto cercano a 3.2.

CASO 3: Finalmente se decide probar modificando solo el *learning rate* dejando una tasa baja y con idéntico *momentum* que en los casos anteriores, se utilizan 5400 iteraciones

ya que aparece una meseta en el caso 2 en dicho valor. El resultado obtenido es el mejor de toda la serie de pruebas, genera un error cuadrático medio de aproximadamente 0.12 y el algoritmo no muestra oscilaciones en las últimas 1000 iteraciones. A pesar de todo, sigue siendo un error relativamente alto para lo que se busca.

iv. Utilización de una red SOM para dividir por grupos las frases antes de aplicar la RNA perceptrón. Utilizando dos grupos.

Descripción:

Debido a que solo con el perceptrón *Backpropagation* no fue posible obtener un RMSE suficientemente bajo, se optó por utilizar una red auto organizada, (Kohonen) para dividir primero en conjuntos las frases y luego aplicar sobre dichos conjuntos el perceptrón *Backpropagation*.

La arquitectura general de la red SOM es como se muestra en la figura a continuación:

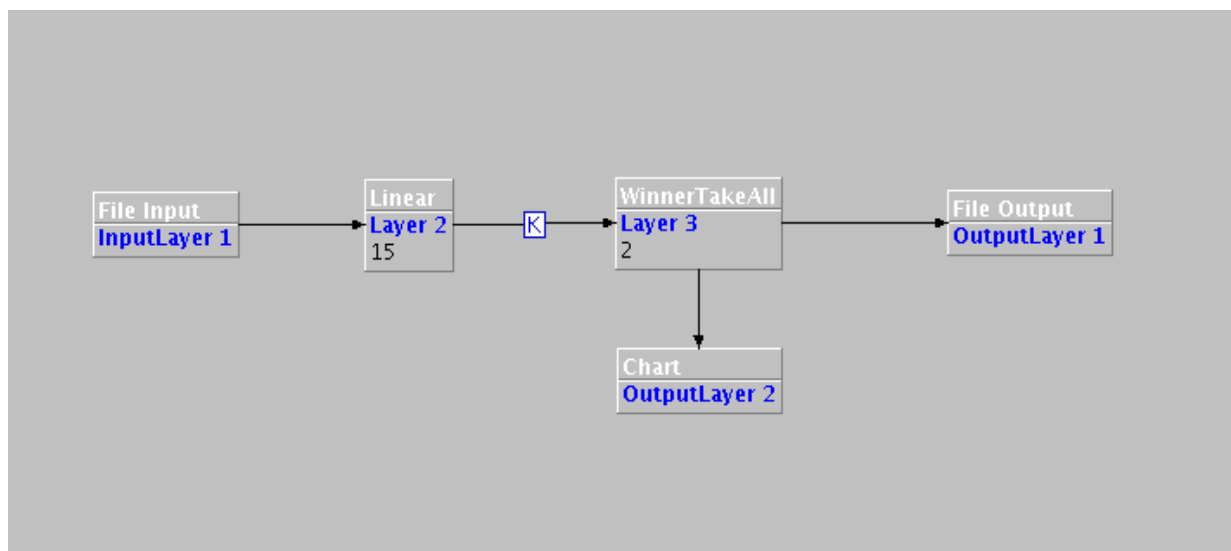


Diagrama 5.4.1: red SOM

Ejecución:

Se realizaron diferentes ejecuciones con dicha arquitectura, buscando generar dos conjuntos con similar número de elementos. Para dichas corridas se generó un nuevo conjunto de casos de entrenamiento con 101 casos. Los archivos de prueba más los archivos convertidos en mapas de entrada pueden encontrarse en el apéndice:

Archivo de frases de entrada: "CB_01_no_processed_101.txt"

Archivo con los mapas de la RNA: "CB_01_processed_101.txt"

CASO 1:

Características de Red		
Input Layer	15	linear
Output Layer	2	Winner Takes All
Características de Entrenamiento		
iteraciones	6500	
learning rate	0.7	
momentum	0.0	

Tabla 5.4.2.6.8: Ejecución de pruebas: RNA división de casos en 2 utilizando una red SOM, CASO 1

CASO 2:

Características de Red		
Input Layer	15	linear
Output Layer	2	Winner Takes All
Características de Entrenamiento		
iteraciones	1000	
learning rate	0.7	
momentum	0.0	

Tabla 5.4.2.6.9: Ejecución de pruebas: RNA división de casos en 2 utilizando una red SOM, CASO 2

CASO 3:

Características de Red		
Input Layer	15	linear
Output Layer	2	Winner Takes All
Características de Entrenamiento		
iteraciones	2000	
learning rate	0.7	
momentum	0.0	

Tabla 5.4.2.6.10: Ejecución de pruebas: RNA división de casos en 2 utilizando una red SOM, CASO 3

Luego se aplicó la siguiente RNA (perceptrón con *Backpropagation*), sobre los archivos obtenidos:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	5400
<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.11: Ejecución de pruebas: RNA división de casos en 2 utilizando una red SOM, características del perceptrón

Resultados:

CASO 1: El archivo de salida fue: "CB_01_processed_101_ko1.txt", a partir de dicho archivo se generaron dos archivos más, con los casos de entrenamiento para el perceptrón anterior:

"ko1/CB_01_processed_101_0.txt" con 12 *training patterns*

"ko1/CB_01_processed_101_1.txt" con 88 *training patterns*

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CB/"

El resultado del entrenamiento del perceptrón, es el siguiente:

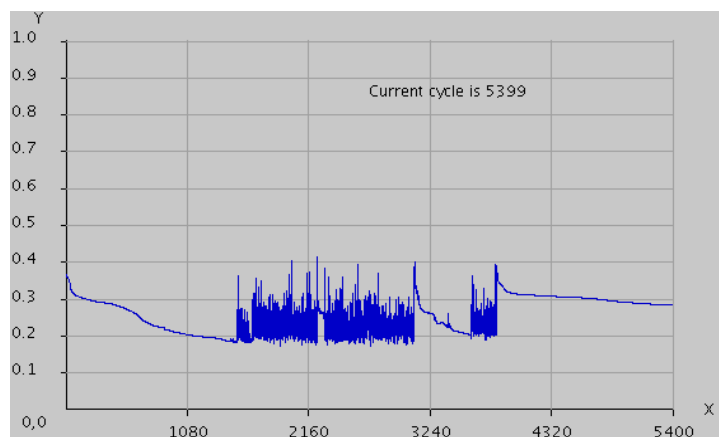


Gráfico 5.4.9: pruebas de RNA, SOM caso 1, conj. 1

RMSE: 0.2847288

CASO 2: El archivo de salida fue: "CB_01_processed_101_ko3.txt", a partir de dicho archivo se generaron dos archivos más, con los casos de entrenamiento para el perceptrón anterior:

"ko3/CB_01_processed_101_0.txt" con 1 *training patterns*

"ko3/CB_01_processed_101_1.txt" con 100 *training patterns*

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CB/"

CASO 3: El archivo de salida fue: "CB_01_processed_101_ko4.txt", a partir de dicho archivo se generaron dos archivos más, con los casos de entrenamiento para el perceptrón anterior:

"ko4/CB_01_processed_101_0.txt" con 53 *training patterns*

"ko4/CB_01_processed_101_1.txt" con 48 *training patterns*

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CB/"

El resultado del entrenamiento del perceptrón, es el siguiente:

Archivo: "CB_01_processed_101_0.txt":

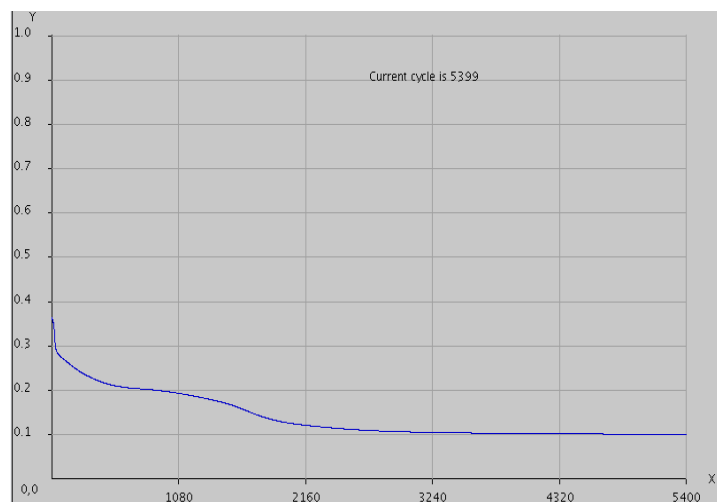


Gráfico 5.4.10: pruebas de RNA, SOM caso 3, conj. 1

RMSE: 0.1010278

Archivo: "CB_01_processed_101_1.txt":

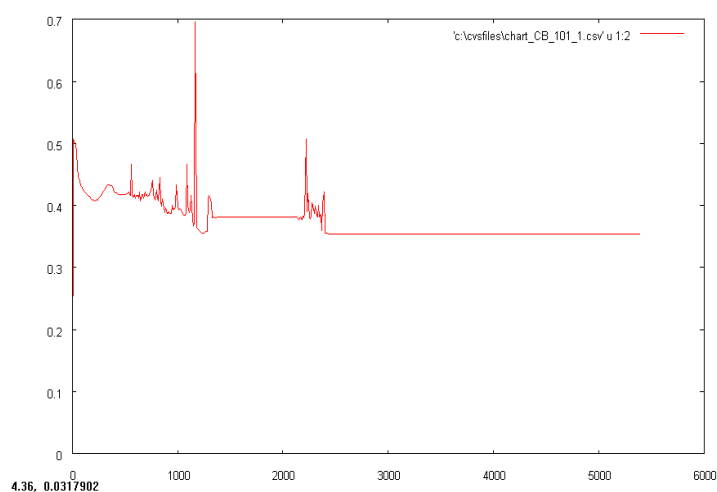


Gráfico 5.4.11: pruebas de RNA, SOM caso 3, conj. 2

RMSE: 0.3536399

Evaluación de Resultados:

CASO 1: En esta primera corrida de la red SOM se obtienen dos conjuntos muy dispares ya que uno contiene casi el 90 por ciento de los casos. Al entrenar la red percetprón con *Backpropagation* descrita con el conjunto mayor se obtiene un error cuadrático medio (RMSE) igual a 0.2847288, un error mayor al obtenido con el conjunto de pruebas previo (0.1248292). Se considera al error demasiado grande y se continua con el siguiente caso.

CASO 2: En esta corrida la red SOM genera un conjunto con un solo caso y otro con el resto (100 casos). Esta corrida se descarta ya que no habría diferencia con correr la red percetprón con *Backpropagation* directamente sobre el conjunto inicial.

CASO 3: En este caso la red SOM genera dos conjuntos con una cantidad de elementos similar, 48 y 53. Se utilizan los dos conjuntos para entrenar dos redes perceptrón con *Backpropagation* como la descrita. En el primero de los casos se obtiene un valor RMSE menor que cualquiera obtenido anteriormente pero para el segundo conjunto el error cuadrático medio es mayor a 0.35.

v. Utilización de una red SOM para dividir por grupos las frases antes de aplicar la RNA perceptrón. Utilizando tres grupos.

Descripción:

Se utilizó una RNA SOM idéntica a la descrita en el punto anterior, con una capa de salida de 3 neuronas, buscando la generación de 3 conjuntos diferentes de casos de entrenamiento para la red perceptrón con *Backpropagation*.

Ejecución:

Se realizó una sola corrida con esta configuración:

Características de Red		
Input Layer	15	linear
Output Layer	3	Winner Takes All
Características de Entrenamiento		
iteraciones	2000	
learning rate	0.7	
momentum	0.0	

Tabla 5.4.2.6.12: Ejecución de pruebas: RNA división de casos en 3 utilizando una red SOM

Utilizando cada uno de los conjuntos generados como conjuntos de entrenamiento para el perceptrón con *Backpropagation*, se entrenaron 3 redes neuronales, los datos de dicha red se detallan a continuación:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	5400

<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.13: Ejecución de pruebas: RNA división de casos en 3 utilizando una red SOM, características del perceptrón

Resultados:

El archivo de salida fue: "CB_01_processed_101_ko5.txt", a partir de dicho archivo se generaron tres archivos con los casos de entrenamiento para el perceptrón anterior:

"ko5/CB_01_processed_101_0.txt" con 69 *training patterns*

"ko5/CB_01_processed_101_1.txt" con 11 *training patterns*

"ko5/CB_01_processed_101_2.txt" con 21 *training patterns*

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CB/"

El resultado del entrenamiento de las redes tipo perceptrón, son los siguientes:

"CB_01_processed_101_0.txt" (69 *training patterns*)

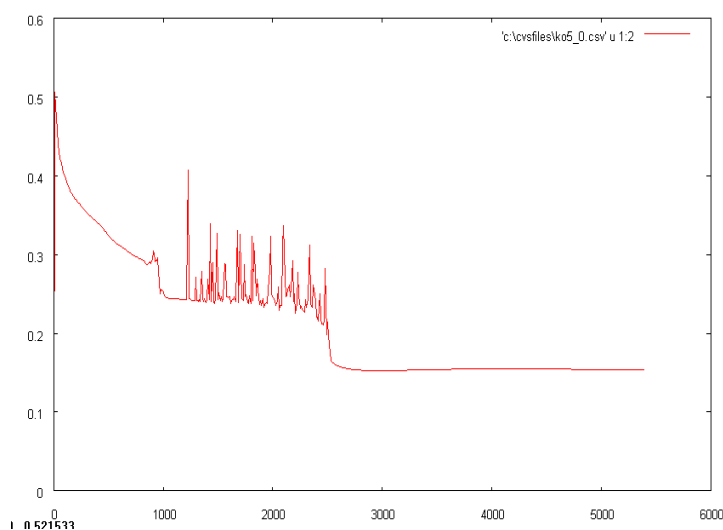


Gráfico 5.4.12: pruebas de RNA, SOM (3 conjuntos) conj. 1

RMSE: 0.1532567

"CB_01_processed_101_1.txt" (11 *training patterns*)

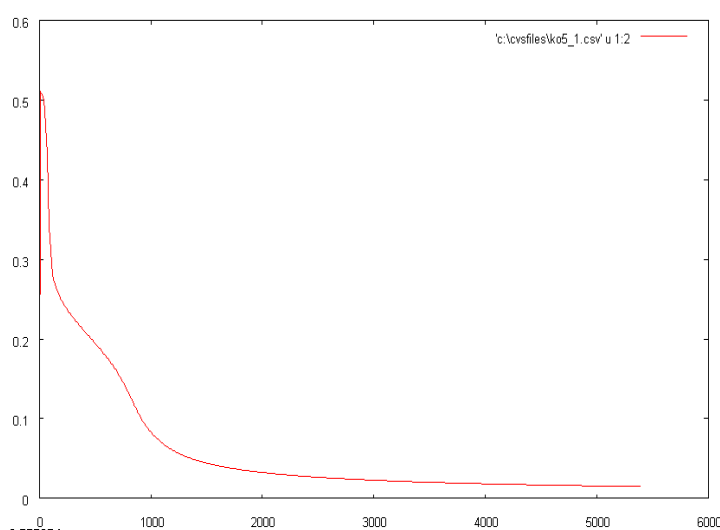


Gráfico 5.4.13: pruebas de RNA, SOM (3 conjuntos) conj. 2

RMSE: 0.0145516

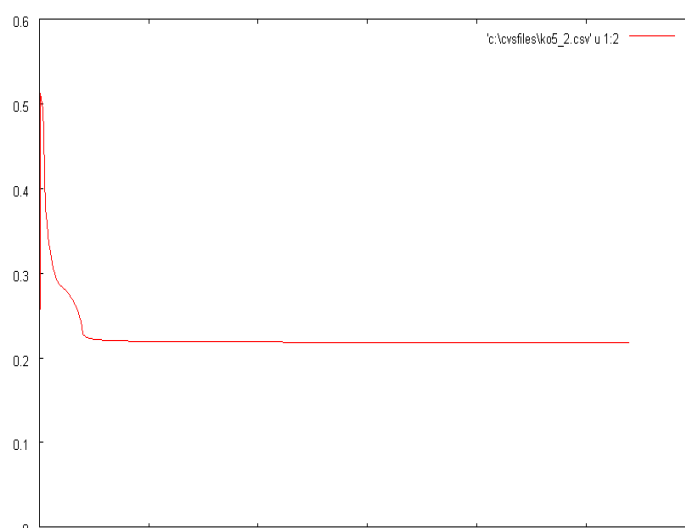


Gráfico 5.4.14: pruebas de RNA, SOM (3 conjuntos) conj. 3

"CB_01_processed_101_2.txt" (21 training patterns)

RMSE: 0.2183813

Evaluación de Resultados:

Los resultados obtenidos siguen mostrando conjuntos cuyo RMSE es mayor al RMSE obtenido para los casos de prueba anteriores en donde no se utilizaba una red SOM.

vi. Utilización de una red SOM para dividir por grupos las frases antes de aplicar la RNA perceptrón. Utilizando cuatro grupos.

Descripción:

Se utilizó una RNA SOM idéntica a la descrita en el punto anterior, con una capa de salida de 4 neuronas, buscando la generación de 4 conjuntos diferentes de casos de entrenamiento para la red perceptrón con *Backpropagation*.

Ejecución:

Se realizaron dos corridas, con las siguientes configuraciones:

CASO 1:

Características de Red		
Input Layer	15	linear
Output Layer	4	Winner Takes All
Características de Entrenamiento		
iteraciones	2000	
learning rate	0.7	
momentum	0.0	

Tabla 5.4.2.6.14: Ejecución de pruebas: RNA división de casos en 4 utilizando una red SOM, CASO 1

Los archivos generados se utilizaron como conjuntos de entrenamiento para el perceptrón con *Backpropagation* que detalla a continuación:

Características de Red	
InputLayer	15
HiddenLayer	7
OutputLayer	1
Características de Entrenamiento	
iteraciones	5400
learning rate	0.2
momentum	0.6

Tabla 5.4.2.6.15: Ejecución de pruebas: RNA división de casos en 4 utilizando una red SOM, CASO 1, perceptrón

CASO 2:

Características de Red

Input Layer	15	linear
Output Layer	4	Winner Takes All
Características de Entrenamiento		
iteraciones	4000	
learning rate	0.7	
momentum	0.0	

Tabla 5.4.2.6.16: Ejecución de pruebas: RNA división de casos en 4 utilizando una red SOM, CASO 2

Los archivos generados se utilizaron como conjuntos de entrenamiento para el perceptrón con *Backpropagation* que detalla a continuación:

Características de Red	
<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	10000
<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.17: Ejecución de pruebas: RNA división de casos en 4 utilizando una red SOM, perceptrón

Resultados:

CASO 1: El archivo de salida fue: "CB_01_processed_101_ko6.txt", a partir de dicho archivo se generaron cuatro archivos con los casos de entrenamiento para el perceptrón anterior:

"ko6/CB_01_processed_101_0.txt" con 21 *training patterns*

"ko6/CB_01_processed_101_1.txt" con 27 *training patterns*

"ko6/CB_01_processed_101_2.txt" con 31 *training patterns*

"ko6/CB_01_processed_101_3.txt" con 22 *training patterns*

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CB/"

El resultado del entrenamiento de la red tipo perceptrón, es los siguientes:

"CB_01_processed_101_0.txt" (21 *training patterns*)

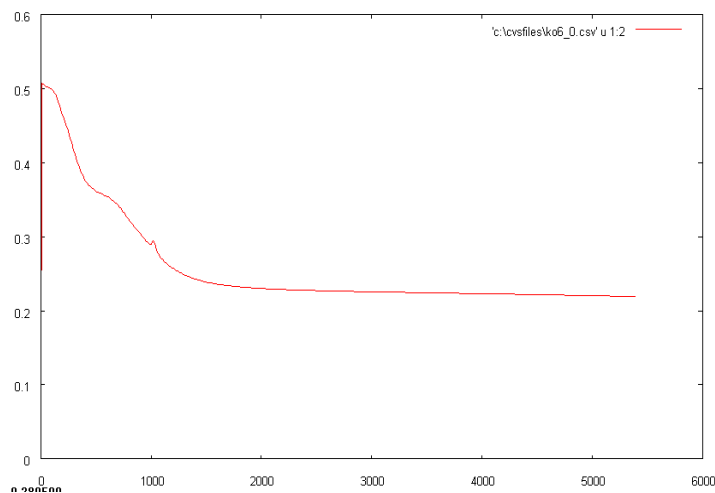


Gráfico 5.4.15: pruebas de RNA, SOM (4 conjuntos), caso 1, conj. 1

RMSE: 0.2190835

CASO 2: El archivo de salida fue: "CB_01_processed_101_ko7.txt", a partir de dicho archivo se generaron cuatro archivos con los casos de entrenamiento para el perceptrón anterior:

"ko7/01/CB_01_processed_101_0.txt" con 21 *training patterns*

"ko7/01/CB_01_processed_101_1.txt" con 27 *training patterns*

"ko7/01/CB_01_processed_101_2.txt" con 31 *training patterns*

"ko7/01/CB_01_processed_101_3.txt" con 22 *training patterns*

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CB/"

El resultado del entrenamiento de las redes tipo perceptrón, son los siguientes:

"ko7/01/CB_01_processed_101_1.txt" (27 *training patterns*)

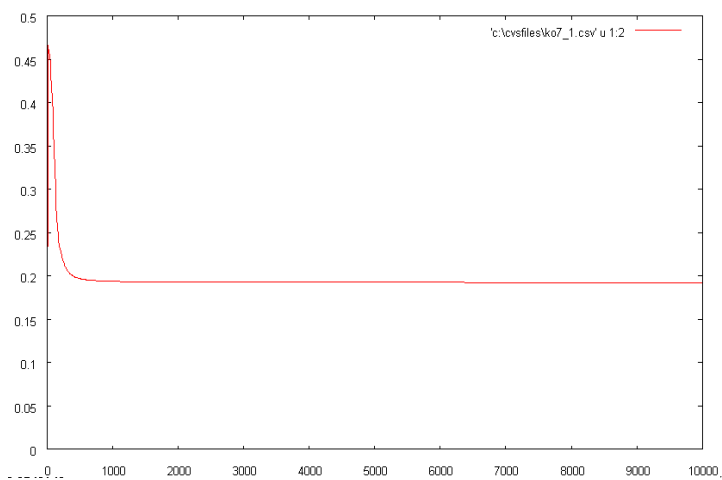


Gráfico 5.4.16: pruebas de RNA, SOM (4 conjuntos), caso 2, conj. 2

RMSE: 0.1925219

"ko7/01/CB_01_processed_101_2.txt" (31 training patterns)

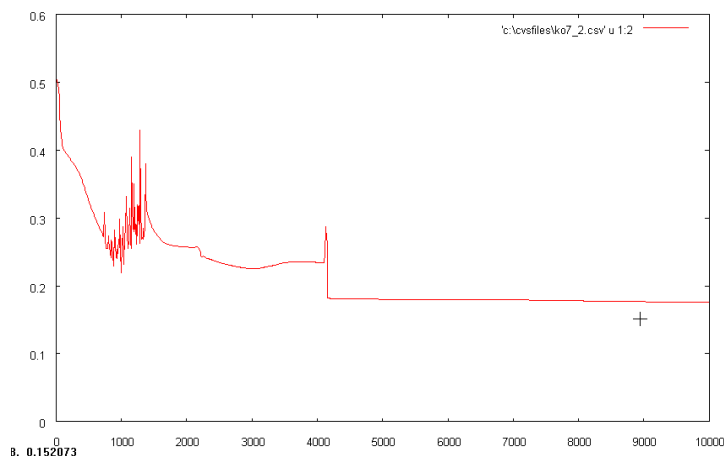


Gráfico 5.4.17: pruebas de RNA, SOM (4 conjuntos), caso 2, conj. 3

RMSE: 0.1754589

"ko7/01/CB_01_processed_101_3.txt" (22 training patterns)

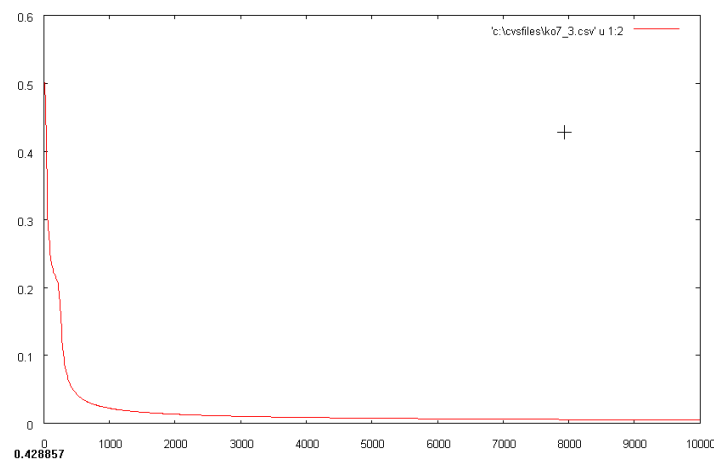


Gráfico 5.4.18: pruebas de RNA, SOM (4 conjuntos), caso 2, conj. 4

RMSE = 0.0050216

Evaluación de Resultados:

CASO 1: El error cuadrático medio es mayor que el máximo esperado, en el primer conjunto evaluado, por lo tanto no se continua con el entrenamiento de la red para los demás conjuntos.

CASO 2: Para mejorar el caso anterior, se agregan iteraciones de entrenamiento tanto en la RNA SOM como en la RNA de tipo perceptrón. El RMSE de cada perceptrón con *Backpropagation* entrenado es relativamente bajo pero sigue siendo más alto que el máximo esperado.

vii. Pruebas con diferente mapa

Descripción:

En caso de que el resultado obtenido no sea satisfactorio cambiar el mapa. Esto requerirá volver a correr el caso de pruebas anterior para el mapa y este nuevamente.

Ejecución:

Se generó un nuevo *set* de pruebas de 111 casos, con dicho *set* se corrieron 3 casos utilizando el nuevo mapa de input cada uno con las configuraciones que se listan más abajo.

Archivo de frases de entrada: "CC_01_no_processed_111.txt"

Archivo con los mapas de la RNA: "CC_01_processed_111.txt"

Dichos archivos pueden hallarse en el repositorio, en el directorio: "test/frases/CC/"

CASO 1:

Características de Red	
<i>Input Layer</i>	76
<i>Hidden Layer</i>	38
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	5000
<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.17: Ejecución de pruebas: RNA variación de mapa CASO 1

CASO 2:

Características de Red	
<i>Input Layer</i>	76
<i>Hidden Layer</i>	38
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	5000
<i>learning rate</i>	0.2
<i>momentum</i>	0.9

Tabla 5.4.2.6.18: Ejecución de pruebas: RNA variación de mapa CASO 2

CASO 3:

Características de Red	
<i>Input Layer</i>	76
<i>Hidden Layer</i>	38
<i>Output Layer</i>	1
Características de Entrenamiento	
<i>iteraciones</i>	5000
<i>learning rate</i>	0.5
<i>momentum</i>	0.6

Tabla 5.4.2.6.19: Ejecución de pruebas: RNA variación

de mapa CASO 3

Resultados:

CASO 1:

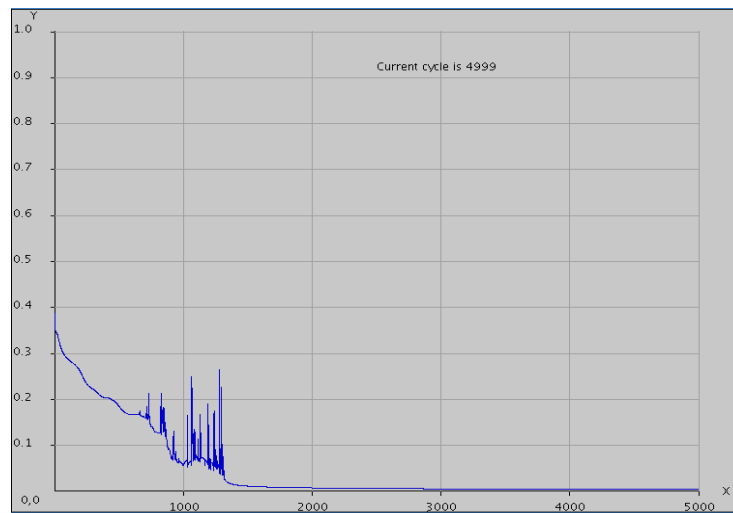


Gráfico 5.4.19: pruebas de RNA, input 2, caso 1

RMSE: 0.0040552

CASO 2:

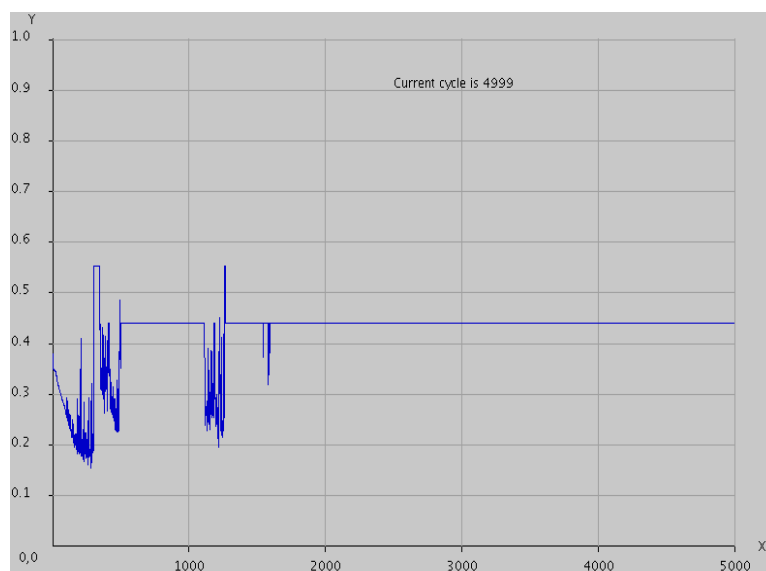


Gráfico 5.4.20: pruebas de RNA, input 2, caso 2

RMSE: 0.4401065

CASO 3:

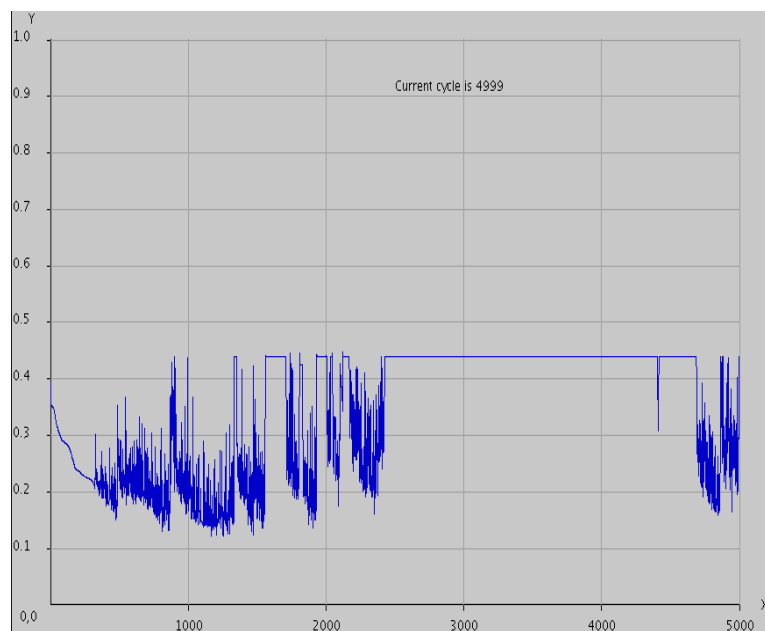


Gráfico 5.4.21: pruebas de RNA, input 2, caso 3

RMSE: 0.3268898

Evaluación de Resultados:

De los tres casos presentados, solo el primero logra un error considerablemente menor al mínimo error (RMSE) anterior obtenido. El siguiente paso será probar la red neuronal artificial entrenada contra un set de pruebas de validación y verificar que no está sobre-entrenada.

viii. Probar contra el set de validación**Descripción:**

Obtener la configuración más simple que tenga el menor error. Entrenar la red con el set anterior. Crear un nuevo set de 100 frases de la misma manera en que se construyó el primero. Correr la red entrenada contra este set, verificar el error sobre este nuevo set.

Ejecución:

Se utilizará un set de pruebas de 111 casos para entrenar la RNA, el mismo que se utilizó en el entrenamiento del perceptrón con 76 neuronas de entradas. Y otro set de 111 casos para validar el funcionamiento de la RNA. Las validaciones se realizarán sobre las redes neuronales que obtuvieron los menores errores cuadráticos medios (RMSE.) Estas son la siguientes:

RNA 01: Perceptrón con *Backpropagation* con la configuración que se detalla a continuación (punto: iii caso 3):

Características de Red

<i>Input Layer</i>	15
<i>Hidden Layer</i>	7
<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	5400
<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.19: Ejecución de pruebas: RNA validación. RNA 01

Dicha RNA utiliza el como entrada el primer mapa generado.

RNA 02: Perceptrón con *Backpropagation* que logró un RMSE de 0.0040552, cuya configuración se detalla a continuación (punto vii, caso 1):

Características de Red	
<i>Input Layer</i>	76
<i>Hidden Layer</i>	38
<i>Output Layer</i>	1
Características de Entrenamiento	
iteraciones	5000
<i>learning rate</i>	0.2
<i>momentum</i>	0.6

Tabla 5.4.2.6.20: Ejecución de pruebas: RNA validación. RNA 02

Dicha RNA utiliza el como entrada el segundo mapa generado.

Los resultados y el *set* de validación se encuentra en el repositorio en el directorio: `"/test/frases/CC/".`

Estos son los archivos con los conjuntos de casos de entrenamiento utilizados:

Frases en inglés: `"test/frases/CC/CC_01_no_processed_111.txt"`

Mapa de entrada para la primer RNA: `"test/frases/CC/CC_01_processed_111_input1.txt"`

Mapa de entrada para la segunda RNA: `"test/frases/CC/CC_01_processed_111.txt"`

Estos son los archivos con los conjuntos de casos de validación utilizados:

Frases en inglés: `"test/frases/CC/CC_02_no_processed_111.txt"`

Mapa de entrada para la primer RNA: "test/frases/CC/CC_02_processed_111_input1.txt"

Mapa de entrada para la segunda RNA: "test/frases/CC/CC_02_processed_111.txt"

Resultados:

Los resultados se dividen en dos partes: Entrenamiento y Resultados.

Entrenamiento:

RNA 01: El entrenamiento de la primer RNA, generó un RMSE igual a 0.1734660, y el siguiente gráfico:

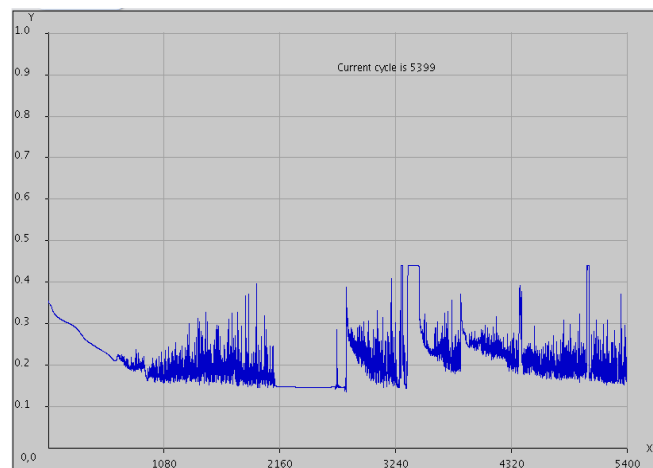


Gráfico 5.4.22: validación de RNA, input 1

RNA 02: Los resultados del entrenamiento de esta RNA, pueden verse en el punto vii (Pruebas con diferente mapa) CASO 1

Validaciones:

RNA 01: Casos detectados correctamente: 65 de 111, 58,56%

RNA 02: Casos detectados correctamente: 47 de 111, 42,34%

Evaluación de Resultados:

Los resultados obtenidos fueron más bajos de los esperados, el porcentaje de error rondó el 50% en ambos casos lo cual es casi como clasificar los casos de forma aleatoria. En particular para la segunda RNA se pudo apreciar que el RMSE bajo que se había obtenido correspondía a un sobre aprendizaje de la red y no a una solución real del problema. Para continuar con el desarrollo de la solución propuesta en la Tesis se utilizará la primer RNA para clasificar la base de datos de Alice.

5.4.2.7 Verificación del sistema de conversión de respuestas cerradas

Pruebas:

i. Probar que el chatbot puede reconocer cuando una pregunta formulada es cerrada.

Descripción:

Construir un archivo de texto con 10 casos, cada uno constituido por dos frases, tal que la primera fuerce al chatbot a formular una pregunta. En la mitad de los casos la pregunta debe ser del tipo cerrado y en la otra mitad del tipo abierto. En ambos casos la segunda frase será "sí" o "no" de forma alternada. Luego de ingresar la primer frase, esperar la respuesta del chatbot e ingresar la segunda. Finalmente verificar que el subsistema detectó correctamente cuales eran preguntas cerradas y cuales no.

Ejecución:

Se eligieron las siguientes frases de entradas, que provocan en el chatbot las respuestas que se listan a continuación:

Caso	Frase de entrada	Respuesta Chatbot esperada	¿es cerrada?
1	TELL ME ABOUT SOME	some what?	no
2	TELL ME YOUR PROBLEMS	are you a psychologist?	si
3	TERMINATOR2	what did you like about those robots?	no
4	THANK GOD I'M FINE	do you believe in god ?	si
5	THAT IS BAD	what's so bad about it?	no
6	I AM A WINE MASTER	have you visited napa?	si
7	STUPID	are you calling me stupid?	si
8	I AM ALONE	are you lonely?	si
9	THE EVIDENCE	what kind of evidence do you accept?	no
10	THE DRUGS	hat kind of drugs are you talking about?	no

Tabla 5.4.2.7.1: Ejecución de pruebas: Conversión de respuestas cerradas, reconocimiento.

Resultados:

En cada uno de los casos el chatbot detectó la pregunta realizada por el mismo como cerrada o abierta, según la última columna mostrada arriba. Para ello se modificó el código del subsistema de detección de preguntas cerradas para que imprima por pantalla una leyenda indicando el resultado de la función "isYesNoAnswer" ubicada en "AliceImprovements.java".

Evaluación de Resultados:

Los resultados son los esperados, el subsistema funciona correctamente.

ii. Probar que puede convertir de manera eficaz una frase sí/no en una frase independiente de contexto con el mismo significado.

Descripción:

Tomar los casos de preguntas cerradas, del punto anterior y duplicarlos. Poner como segunda frase "si" y luego "no". Verificar que el chatbot imprime por pantalla la conversión correcta de cada frase.

Ejecución:

Se correrán los siguientes casos contra el chatbot, el cual se ha modificado para que imprima por pantalla la frase tal y como queda al ser reformulada.

Caso	1º frase de entrada	respuesta del chatbot	2º frase de entrada
1	TELL ME YOUR PROBLEMS	are you a psychologist?	yes
2	THANK GOD I'M FINE	do you believe in god ?	yes
3	I AM A WINE MASTER	have you visited napa?	yes
4	STUPID	are you calling me stupid?	yes
5	I AM ALONE	are you lonely?	yes
6	TELL ME YOUR PROBLEMS	are you a psychologist?	no
7	THANK GOD I'M FINE	do you believe in god ?	no
8	I AM A WINE MASTER	have you visited napa?	no
9	STUPID	are you calling me stupid?	no
10	I AM ALONE	are you lonely?	no

Tabla 5.4.2.7.2: Ejecución de pruebas: Conversión de respuestas cerradas

Resultados:

Casos	frase reformulada
1	i am a psychologist
2	i believe in god
3	i have visited napa
4	i am calling you stupid
5	i am lonely
6	i am not a psychologist
7	i don't believe in god

8	i have not visited napa
9	i am not calling you stupid
10	i am not lonely

Tabla 5.4.2.7.3: Resultados de pruebas: Conversión de respuestas cerradas

Evaluación de Resultados:

Los resultados son los esperados, las frases han sido correctamente reformuladas.

5.4.2.8 Verificación del sistema de generación de preguntas

Pruebas:

i. Probar que el chatbot es capaz de armar las preguntas en base a la frase del usuario.

Descripción:

Construir un archivo de texto con 10 frases. En donde en cada frase hay una pregunta hacia el chatbot en la cual se referencia a algo. Ingresar las frases en el chatbot como inicio de una conversación nueva. Verificar que el chatbot respondió con una pregunta que hace referencia al dato faltante.

Ejecución:

Se escogieron los siguientes casos de pruebas, son frases en inglés en donde en cada caso se hace referencia a un dato el cual debería de ser obtenido, del contexto en una conversación normal. Como es una primera frase en una conversación, el sistema deberá detectarla y preguntar por la información faltante.

Caso	Frase de entrada
1	do you like her?
2	is that fun?
3	the pencil is there
4	what is your best?
5	those pants are cool
6	it's yours?
7	do you know him?
8	it's the one you sell
9	you know, is like that
10	they are my friends

*Tabla 5.4.2.8.1: Ejecución de pruebas: Generación de preguntas***Resultados:**

Caso	Salida generada por el subsistema
1	who do you refer with "her"?
2	what is fun?
3	where is "there"?
4	my best what?
5	what do you refer with "those"?
6	what is yours?
7	who do you refer with "him"?
8	what is the one me sell?
9	like what?
10	they who?

*Tabla 5.4.2.8.2: Resultados de pruebas: Generación de preguntas***Evaluación de Resultados:**

Los resultados son los esperados, por supuesto el conjunto de oraciones en las cuales puede haber referencias no correspondidas con la información de contexto es infinito, aquí solo se realizaron pruebas para casos más bien triviales, pero bastantes comunes. En todos ellos había una palabra de tipo "referente" bien marcada, como por ejemplo: *that* o *him*.

5.4.2.9 Verificación del sistema de resolución de referentes**Pruebas:****Descripción:**

Se construirá un conjunto de 10 casos de prueba, para probar el funcionamiento del sistema de resolución de referentes implementado con el algoritmo: "Resolution of Anaphora Procedure". Los casos serán dos sentencias simples, en donde la primera tendrá un sustantivo en el sujeto y la segunda una referencia al mismo. Casos más complejos quedan descartados por las limitaciones propias del algoritmo.

Ejecución:

Se crearon los siguientes casos, y se corrieron en el subsistema imprimiendo por pantalla el resultado del algoritmo.

Caso	Frases
1	The book is great. It costs about 50 dollars.
2	my tshirt. it has a draw of darth vader
3	my computer is broken. it is a mac
4	i play the piano. it is quite easy
5	richard wallace. him is a great inventor
6	i was at home yesterday. there is a peaceful place
7	my uncle is captain of a ship. He is a good guy
8	the winter is not nice. that is worse than autumn
9	the taxi drivers are crazy. but they work so hard
10	she said that her dog is very big. But I see it and is regular size

Tabla 5.4.2.9.1: Ejecución de pruebas: Resolución de referentes

Resultados:

Caso	Frases
1	(0,0) The book <-- (1,0) It The book is great. <The book> costs about 50 dollars.
2	(0,0) my tshirt <-- (1,0) it my tshirt. <my tshirt> has a draw of darth vader my tshirt has a draw of darth vader
3	(0,0) my computer <-- (1,0) it my computer is broken. <my computer> is a mac my computer is a mac
4	(0,2) the piano <-- (1,0) it i play the piano. <the piano> is quite easy the piano is quite easy
5	(0,0) richard wallace <-- (1,0) him richard wallace. <richard wallace> is a great inventor richard wallace is a great inventor
6	null i was at home yesterday. there is a peaceful place there is a peaceful place
7	(0,3) captain <-- (1,0) He my uncle is captain of a ship. <captain> is a good guy

	captain is a good guy
8	null the winter is not nice. that is worse than autumn that is worse than autumn
9	(0,0) the taxi drivers <-- (1,1) they the taxi drivers are crazy. but <the taxi drivers> work so hard but the taxi drivers work so hard
10	(0,3) her <-- (0,0) she, (0,3) her <-- (0,3) her, (1,1) I <-- (1,3) it <her> said that her dog is very big. But I <I> it and is regular size But I I it and is regular size

Tabla 5.4.2.9.2: Resultados de pruebas: Resolución de referentes

Evaluación de los Resultados:

En las primeras líneas, de la forma: "(0,1) word ← (0,3) word2" se muestran las asociaciones, derecha: "referente", izquierda: "palabra a la cual hace referencia". Si es "null" es porque no encontró referencias. Luego se muestran las dos oraciones, y en la segunda entre caracteres "<>" la palabra referenciada si es que la halló. Finalmente se muestra como quedó la segunda sentencia, con el reemplazo de su referente hecho.

Estos casos de prueba dan resultados del 50%, y son casos sencillos, el algoritmo no está a un nivel completamente productivo sin embargo su efectividad alcanza para mostrar como podría mejorar los dialogos en un chatbot.

5.4.3 Ejecución de las pruebas de integración

5.4.3.1 Integración del sistema generador del mapa y la red neuronal artificial

Pruebas:

i. Probar que dado una serie de frases en inglés el sistema automáticamente puede convertirlas en un mapa que sirve de entrada a la red neuronal y que esta puede tomarlo y procesarlo correctamente

Descripción:

Construir un archivo de texto con 10 frases en inglés. Pasárselas al sistema como *input*. Generar *logs* del sistema para seguir las fases por las que pasó. Verificar que la red neuronal artificial se ejecutó por cada frase de entrada y devolvió un valor.

Ejecución:

Se tomaron los 10 casos del archivo con el conjunto de entrenamiento de la RNA, se buscaron casos dependientes del contexto para forzar a "myAlice" a analizar las frases de entrada con la RNA. Los casos se listan a continuación:

1. Is that fun?
2. Is that why it's part-time?!
3. Yes, that sounds extremely civilised.
4. I guess if you are a film buff, that's probably a good enough recommendation to someone less discerning!
5. That is generally how I feel about films
6. Presumably you just need the right contacts
7. Can you explain why it is so great?!
8. Fair enough.
9. I don't even know what it is because my husband chose it.
10. On Sundays?

En el código se añadieron mensajes de *log* para mostrar la generación de los *inputs* de la RNA y el resultado que devuelve esta.

Resultados:

Caso	mapa generado	resultado de la RNA
1	1.0;0.5;1.5;2.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0

2	1.0;0.5;1.5;2.3;1.9;3.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;	0
3	0.0;0.0;0.0;2.3;0.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
4	0.0;0.0;1.9;1.9;0.0;3.5;1.9;0.0;1.9;0.0;2.3;1.5;1.5;0.0;1.3;	0
5	1.0;0.0;2.3;1.5;1.5;2.0;1.9;1.5;1.3;0.0;0.0;0.0;0.0;0.0;	0
6	0.0;-1.0;1.5;3.5;1.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
7	1.0;0.5;0.0;3.5;1.5;1.9;3.0;1.5;0.0;1.3;0.0;0.0;0.0;0.0;	0
8	0.0;0.0;0.0;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
9	1.0;-1.0;1.9;0.0;1.5;0.0;1.5;2.0;3.0;1.5;0.0;2.5;1.9;1.5;3.0;	0
10	0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0

Tabla 5.4.3.1.1: Resultados de pruebas: Integración generador del mapa y RNA

Evaluación de Resultados:

Los resultados son los esperados, el mapa se generó de forma coherente más allá de algunas limitaciones propias de la base de datos "Lexico" y otros errores analizados anteriormente. La salida en todos los casos fue "0", es decir que las frases fueron detectadas de forma correcta como dependientes del contexto.

5.4.3.2 Integración de myAlice con el sistema detector

Pruebas:

i. Probar que el myAlice es capaz de funcionar exactamente igual que ALICE cuando la frase de entrada está dentro de los casos favorables

Descripción:

1. Construir un archivo de texto con 10 frases, las cuales ALICE original sea capaz de responder correctamente (casos favorables).
2. Hacer que el sistema imprima en un archivo de *log* el resultado del campo "can_answer" asociado al patrón que *matcheo* con la frase de entrada.
3. Ingresar las frases en el chatbot.
4. Verificar que el chatbot respondió como lo haría ALICE original y verificar el resultado del *log*

Ejecución:

Se seleccionaron los siguientes casos, estas frases de entradas son independientes del contexto y por lo tanto pertenecen al conjunto de casos favorables de ALICE.

01. TALK TO YOU LATER

02. TELL ME MORE ABOUT DR WALLACE
03. CAN YOU PLAY A GAME
04. CAN WE TALK ABOUT SOMETHING ELSE
05. I AM A PROTESTANT
06. I AM AN ARTIFICIAL INTELLIGENCE
07. I AM BEAUTIFUL
08. I AM NOT A BELIEVER
09. IS DR WALLACE ON STAR TREK VOYAGER?
10. IS THIS A PICTURE OF YOURSELF JUST BELOW?

Resultados:

Caso	Respuesta ALICE	Respuesta myAlice
1	See you later!	see you later!
2	Why don't you buy his book and read his biography?	why don't you buy his book and read his biography?
3	We are playing a game right now.	we are playing a game right now.
4	Sure we can talk about whatever you want.	sure we can talk about whatever you want.
5	I am glad to hear that you have faith, unknown person.	i am glad to hear that you have faith.
6	I doubt that. You seem too much like a human.	i doubt that. You seem too much like a human.
7	Do you attract a lot of people?	do you attract a lot of people?
8	I will pray that you find faith, unknown person.	i will pray that you find faith,.
9	He should be.	he should be.
10	Yes.	yes.

Tabla 5.4.3.2.1: Resultados de pruebas: Integración myAlice

En todos los casos el archivo de *log*, imprimió un mensaje indicando que la oración era independiente del contexto, es decir podía ser respondida.

Evaluación de Resultados:

Los resultados son los esperados, el sistema detector reconoce a las sentencias como casos favorables y "myAlice" se comporta como "ALICE" original sobre AIML. Sin embargo la RNA utilizada para clasificar estos casos no logró como se sabe la efectividad esperada con lo cual este caso de prueba está atado al anterior y los casos fueron elegidos especialmente como correctamente clasificados.

Las diferencias en algunas respuestas de "ALICE" que terminan en "unknown person" y no así las de "myAlice" es debido a que ALICE *setea* por defecto la variable de sesión con el nombre del usuario en "unknown person", (hasta que el usuario revele su nombre) y "myAlice" la inicializa

vacía.

ii. Verificar que en si la frase de entrada no está en el conjunto de casos favorables, myAlice la identificará como tal y utilizará el sistema "detector" sobre la frase de usuario

Descripción:

1. Construir un archivo de texto con 10 frases, las cuales ALICE original no sea capaz de responder correctamente (casos no favorables).
2. Hacer que el sistema imprima en un archivo de *log* el resultado del campo "can_answer" asociado al patrón que *matcheo* con la frase de entrada.
3. Verificar que en el archivo de *log* la frase *matcheo* con un patrón cuyo campo "can_answer" es igual a cero (o *false*)

Ejecución:

Se seleccionaron los siguientes casos, estas frases de entradas son dependientes del contexto y por lo tanto pertenecen al conjunto de casos desfavorables de ALICE:

01. TAKE WHAT
02. THAT IS A CRAZY QUESTION
03. THAT IS A LOT OF PEOPLE
04. THE DOCTOR?
05. THE SUN
06. THIS IS TRUE
07. NO THEY DO NOT
08. NOT OFTEN
09. IS THAT FUN
10. SINCE WHEN?

Resultados:

En cada uno de los casos la aplicación escribió en el archivo de *log* : "dependiente del contexto", indicando que cada una de las frases anteriores fue corroborada como dependiente del contexto.

Evaluación de Resultados:

Al igual que el caso anterior esta prueba está atada a los resultados del sistema "clasificador" implementado a través de la RNA, cuyos resultados tienen un amplio margen de errores, se escogieron casos clasificados correctamente por el sistema ya que lo que se busca probar es la integración y no la efectividad.

iii. Verificar que el sistema detector ejecutó todas las etapas correctamente

Descripción:

1. Construir un archivo de texto con 10 frases, las cuales ALICE original no sea capaz de responder correctamente (casos no favorables).
2. Hacer que el sistema imprima en un archivo de *log* las distintas fases por las que pasa la detección: "*parseo* de palabras y reemplazo por tipo", "detección de tiempos verbales", "generación del mapa"
3. Verificar en el archivo de *log* las etapas por las que pasó. Comprobar que se ejecutaron en orden y correctamente.

Ejecución:

Se tomaron como frases de entrada las siguientes sentencias en inglés, todas ellas dependientes del contexto:

1. Is that fun?
2. Is that why it's part-time?!
3. Yes, that sounds extremely civilised.
4. I guess if you are a film buff, that's probably a good enough recommendation to someone less discerning!
5. That is generally how I feel about films
6. Presumably you just need the right contacts
7. Can you explain why it is so great?!
8. Fair enough.
9. I don't even know what it is because my husband chose it.
10. On Sundays?

Se crearon entradas nuevas en el archivo del *log* en cada uno de los subsistemas, en las siguientes funciones:

"GrammarParser.parse()": función que convierte la frase en un array de números, donde cada número representa el tipo de palabra en el mismo orden de la oración.

"Detector.detectVerbTimeIndex()": función que dada una frase devuelve un número que representa el tiempo verbal en el cual está la sentencia.

"PhraseToInput.generateInput1()": función que devuelve un *array* de "double", el cual es el mapa de la RNA utilizada.

Resultados:

Por cada sentencia el sistema escribió en el archivo de *log* el paso por cada una de las funciones en el orden correcto. Se detallan los resultados:

"parse()": (se imprimieron las palabras, el tipo y el índice utilizado para indicar el tipo, cada palabra

se imprimió separa por "|")

Caso	Resultados impresos en el log
01	is -> verb -> 2 that -> indicative -> 114 fun -> noun -> 1
02	is -> verb -> 2 that -> indicative -> 114 why -> noun -> 1 it -> neuter -> 84 is -> verb -> 2 part-time -> adjective -> 3
03	yes, -> unknown -> 0 that -> indicative -> 114 sounds -> pnoun -> 200 extremely -> adverb -> 4 civilised -> adjective -> 3
04	i -> noun -> 1 guess -> noun -> 1 if -> pnoun -> 200 you -> standard -> 58 are -> noun -> 1 a -> pnoun -> 200 film -> noun -> 1 buff, -> unknown -> 0 that -> indicative -> 114 is -> verb -> 2 probably -> adverb -> 4 a -> pnoun -> 200 good enough -> adjective -> 3 recommendation -> noun -> 1 to -> pnoun -> 200 someone -> noun -> 1 less -> adjective -> 3 discerning -> adjective -> 3
05	that -> indicative -> 114 is -> verb -> 2 generally -> adverb -> 4 how -> interrogative -> 107 i -> noun -> 1 feel -> infinitive -> 111 about -> adjective -> 3 films -> unknown -> 0
06	presumably -> adverb -> 4 you -> standard -> 58 just -> adjective -> 3 need -> noun -> 1 the -> pnoun -> 200 right -> pnoun -> 200 contacts -> pnoun -> 200
07	can -> pnoun -> 200 you -> standard -> 58 explain -> verb -> 2 why -> noun -> 1 it -> neuter -> 84 is -> verb -> 2 so -> pnoun -> 200 great -> adjective -> 3
08	fair -> pnoun -> 200 enough -> noun -> 1
09	i -> noun -> 1 do -> pnoun -> 200 not -> adverb -> 4 even -> pnoun -> 200 know -> infinitive -> 111 what -> interrogative -> 107 it -> neuter -> 84 is -> verb -> 2 because -> pnoun -> 200 my -> singular -> 52 husband -> noun -> 1 chose -> past -> 110 it -> neuter -> 84
10	on -> pnoun -> 200 sundays -> unknown -> 0

Tabla 5.4.3.2.2: Resultados de pruebas: Integración myAlice, etapas detección: parse()

"detectVerbTimeIndex()":

Caso	Resultados impresos en el log, indice y significado	
01	1	Simple Present
02	1	Simple Present
03	-1	tiempo verbal no encontrado
04	0	infinitive
05	-1	tiempo verbal no encontrado
06	3	Past
07	1	Simple Present
08	-1	tiempo verbal no encontrado
09	3	Past
10	-1	tiempo verbal no encontrado

Tabla 5.4.3.2.3: Resultados de pruebas: Integración myAlice, etapas detección:

detectVerbTimeIndex()

"generateInput1()":

Caso	mapa generado	resultado de la RNA
01	1.0;0.5;1.5;2.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
02	1.0;0.5;1.5;2.3;1.9;3.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;	0
03	0.0;0.0;0.0;2.3;0.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;	0
04	0.0;0.0;1.9;1.9;0.0;3.5;1.9;0.0;1.9;0.0;2.3;1.5;1.5;0.0;1.3;	0
05	1.0;0.0;2.3;1.5;1.5;2.0;1.9;1.5;1.3;0.0;0.0;0.0;0.0;0.0;	0
06	0.0;-1.0;1.5;3.5;1.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
07	1.0;0.5;0.0;3.5;1.5;1.9;3.0;1.5;0.0;1.3;0.0;0.0;0.0;0.0;	0
08	0.0;0.0;0.0;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
09	1.0;-1.0;1.9;0.0;1.5;0.0;1.5;2.0;3.0;1.5;0.0;2.5;1.9;1.5;3.0;	0
10	0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0

Tabla 5.4.3.2.4: Resultados de pruebas: Integración myAlice, etapas detección:

generateInput1()

Evaluación de Resultados:

El funcionamiento de los sistemas de forma colaborativa y el orden de su ejecución es el esperado, aunque estos no hayan obtenido el resultado esperado en algunos casos. A los efectos del test los resultados son correctos.

iv. Verificar que generó el mapa de la frase correctamente

Descripción:

1. Construir un archivo de texto con 10 frases, las cuales ALICE original no sea capaz de responder correctamente (casos no favorables).
2. Hacer que el archivo de *log* imprima el mapa resultado
3. Verificar en el archivo de *log* que el mapa generado es correcto

Ejecución:

Se tomaron los 10 casos del archivo con el conjunto de entrenamiento de la RNA, se buscaron casos dependientes del contexto para forzar a "myAlice" a analizar las frases de entrada con la RNA. Los casos se listan a continuación:

1. Is that fun?
2. Is that why it's part-time?!
3. Yes, that sounds extremely civilised.
4. I guess if you are a film buff, that's probably a good enough recommendation to someone less

discerning!

5. That is generally how I feel about films
6. Presumably you just need the right contacts
7. Can you explain why it is so great?!
8. Fair enough.
9. I don't even know what it is because my husband chose it.
10. On Sundays?

En el código se añadieron mensajes de *log* para mostrar la generación de los *inputs* de la RNA y el resultado que devuelve esta.

Resultados:

Caso	mapa generado	resultado de la RNA
1	1.0;0.5;1.5;2.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
2	1.0;0.5;1.5;2.3;1.9;3.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;	0
3	0.0;0.0;0.0;2.3;0.0;1.5;1.3;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
4	0.0;0.0;1.9;1.9;0.0;3.5;1.9;0.0;1.9;0.0;2.3;1.5;1.5;0.0;1.3;	0
5	1.0;0.0;2.3;1.5;1.5;2.0;1.9;1.5;1.3;0.0;0.0;0.0;0.0;0.0;	0
6	0.0;-1.0;1.5;3.5;1.3;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
7	1.0;0.5;0.0;3.5;1.5;1.9;3.0;1.5;0.0;1.3;0.0;0.0;0.0;0.0;	0
8	0.0;0.0;0.0;1.9;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0
9	1.0;-1.0;1.9;0.0;1.5;0.0;1.5;2.0;3.0;1.5;0.0;2.5;1.9;1.5;3.0;	0
10	0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;0.0;	0

Tabla 5.4.3.2.5: Resultados de pruebas: Integración myAlice, generación de mapa

Evaluación de Resultados:

Los resultados son los esperados, el mapa se generó de forma coherente más allá de algunas limitaciones propias de la base de datos "Lexico" y otros errores analizados anteriormente. La salida en todos los casos fue "0", es decir que las frases fueron detectadas de forma correcta como dependientes del contexto.

v. Verificar, solo en el caso de que el patrón de *matcheo* tenga un valor diferente en el campo "can_answer" al valor que detectó la red neuronal sobre la frase del usuario, que la frase del usuario y el patrón son distintos. (Hay por ejemplo *wildcars*)

Descripción:

1. Tomar los resultados de la pruebas 4 y verificar si hay discrepancias con los valores asociados al patrón de *matcheo*.

2. En caso de haberlas verificar que el patrón de *matcheo* y la frase de entrada son diferentes.

Ejecución:

Se revisó el *log* y se tomaron las consultas ejecutadas sobre la base de datos, las cuales se corrieron nuevamente de forma manual para obtener los "patrones", contra los cual *matchearon* las sentencias.

Resultados:

Excepto en los casos 1 y 8, no hubo una coincidencia exacta sino que fue a través de un patrón con *wildcards*, sin embargo en todos los casos el campo "can_answer" es igual a 0, el mismo resultado que dio la RNA para la frase de entrada original.

Evaluación de Resultados:

Los resultados son los esperados ya que no hubo diferencias y se mantuvo la coherencia al identificar las frases como dependientes del contexto.

5.4.3.3 Integración de myAlice con el subsistema conversor

Prueba

i. Probar que el myAlice es capaz de funcionar exactamente igual que ALICE cuando la frase de entrada está dentro de los casos favorables

Descripción:

1. Construir un archivo de texto con 10 frases, las cuales ALICE original sea capaz de responder correctamente (casos favorables).
2. Ingresar las frases en el chatbot.
3. Verificar que el chatbot respondió como lo haría ALICE original.

Ejecución:

Se seleccionaron los siguientes casos, estas frases de entradas son independientes del contexto y por lo tanto pertenecen al conjunto de casos favorables de ALICE.

01. TALK TO YOU LATER
02. TELL ME MORE ABOUT DR WALLACE
03. CAN YOU PLAY A GAME
04. CAN WE TALK ABOUT SOMETHING ELSE
05. I AM A PROTESTANT
06. I AM AN ARTIFICIAL INTELLIGENCE

07. I AM BEAUTIFUL
 08. I AM NOT A BELIEVER
 09. IS DR WALLACE ON STAR TREK VOYAGER?
 10. IS THIS A PICTURE OF YOURSELF JUST BELOW?

Resultados:

Caso	Respuesta ALICE	Respuesta myAlice
1	See you later!	see you later!
2	Why don't you buy his book and read his biography?	why don't you buy his book and read his biography?
3	We are playing a game right now.	we are playing a game right now.
4	Sure we can talk about whatever you want.	sure we can talk about whatever you want.
5	I am glad to hear that you have faith, unknown person.	i am glad to hear that you have faith.
6	I doubt that. You seem too much like a human.	i doubt that. You seem too much like a human.
7	Do you attract a lot of people?	do you attract a lot of people?
8	I will pray that you find faith, unknown person.	i will pray that you find faith,.
9	He should be.	he should be.
10	Yes.	yes.

Tabla 5.4.3.3.1: Resultados de pruebas: Integración myAlice con conversor, casos favorables

En todos los casos el archivo de *log*, imprimió un mensaje indicando que la oración era independiente del contexto, es decir podía ser respondida.

Evaluación de Resultados:

Los resultados son los esperados, el sistema detector reconoce a las sentencias como casos favorables y "myAlice" se comporta como "ALICE" original sobre AIML. Sin embargo la RNA utilizada para clasificar estos casos no logró como se sabe la efectividad esperada con lo cual este caso de prueba está atado al anterior y los casos fueron elegidos especialmente como correctamente clasificados.

Las diferencias en algunas respuestas de "ALICE" que terminan en "unknown person" y no así las de "myAlice" es debido a que ALICE *setea* por defecto la variable de sesión con el nombre del usuario en "unknown person", (hasta que el usuario revele su nombre) y "myAlice" la inicializa vacía.

ii. Verificar que si la frase de entrada no está en el conjunto de casos favorables, el sistema

la convertirá en una frase independiente del contexto por medio del sistema de resolución de referentes

Descripción:

1. Construir un archivo de texto con 5 pequeños diálogos, basados en varias frases de entrada, de las cuales ALICE original no sea capaz de responder correctamente la última la frase. Y que esta frase haga referencia a información del contexto.
2. Hacer que el sistema imprima en un archivo de *log* el resultado de la frase nueva generada, la frase independiente del contexto.
3. Ingresar las frases en el chatbot.
4. Verificar el resultado del *log*

Ejecución:

Se buscaron casos particulares en donde el sistema de resolución de referentes implementado: "Resolution of Anaphora Procedure" logra resolver correctamente la referencia y se comparó el resultado del dialogo con el de "ALICE" original. Se imprimió en el *log* la conversión realizada por el algoritmo.

Nuevamente se busca probar el funcionamiento y la integración de los subsistemas y no su eficiencia. Las frases de entrada utilizadas se listan a continuación:

Caso	1º frase de entrada	2º frase de entrada	referente
1	The book is great.	It costs about 50 dollars.	It
2	my tshirt	It has a picture of Darth Vader	It
3	my computer is broken.	It is a mac	It
4	I Play the piano	It is quite easy	It
5	Richard Wallace.	him is a great inventor	him

Tabla 5.4.3.3.2: Ejecución de pruebas: Integración myAlice con conversor, resolución de referentes.

Resultados:

Caso	Dialogo "ALICE"	Dialogo "myAlice"
1	Human: The book is great. ALICE: I don't think I have read it. Human: It costs about 50 dollars. ALICE: Wow, that is cheap.	Human: The book is great. myAlice: I don't think I have read it. Human: It costs about 50 dollars. myAlice: Wow, that is a lot of money.
2	Human: My tshirt ALICE: Tell me about your tshirt. Human: it has a draw of darth vader ALICE: Has it really?	Human: my tshirt myAlice: Tell me about your tshirt. Human: it has a draw of darth vader myAlice: Do robots have tshirt has a picture of darth vader?
3	Human: my computer is broken	Human: my computer is broken

	ALICE: What kind of computer is it? Human: it's a mac ALICE: your computer is a mac. I suppose that makes sense.	myAlice: What kind of computer is it? Human: it's a mac myAlice: What kind of computer is it?
4	Human: I Play the piano ALICE: Was it very difficult to learn? Human: it is quite easy ALICE: Maybe for you.	Human: I Play the piano myAlice: Was it very difficult to learn? Human: it is quite easy myAlice: Who told you that?
5	Human: richard wallace ALICE: Are you asking about my botmaster? Human: him is a great inventor ALICE: I think him is more than that.	Human: richard wallace myAlice: Are you asking about my botmaster? Human: him is a great inventor myAlice: I knew he was an inventor, but no one ever said he was a great inventor

Tabla 5.4.3.3.3: Resultados de pruebas: Integración myAlice con conversor, resolución de referentes.

Las impresiones en el log de cada caso se listan a continuación:

Caso	Log
1	(0,0) The book <-- (1,0) It The book is great. <The book> costs about 50 dollars. The book costs about 50 dollars
2	(0,0) my tshirt <-- (1,0) it my tshirt. <my tshirt> has a draw of darth vader my tshirt has a draw of darth vader
3	(0,0) my computer <-- (1,0) it my computer is broken. <my computer> is a mac my computer is a mac
4	i play the piano. it is quite easy (0,2) the piano <-- (1,0) it i play the piano. <the piano> is quite easy the piano is quite easy
5	richard wallace. him is a great inventor (0,0) richard wallace <-- (1,0) him richard wallace. <richard wallace> is a great inventor richard wallace is a great inventor

Tabla 5.4.3.3.4: Resultados de pruebas: Integración myAlice con conversor, resolución de referentes, contenido del log.

Evaluación de Resultados:

El funcionamiento del modulo y su integración dentro del chatbot: "myAlice" con los demás componentes es la esperada.

iii. Verificar que si la frase de entrada no está en el conjunto de casos favorables y es una respuesta de tipo "sí" o "no" (a una pregunta cerrada) el sistema la convertirá en una frase independiente del contexto por medio del sistema de conversión de respuestas de tipo "sí/no"

Descripción:

1. Construir un archivo de texto con 5 pequeños dialogo, basados en 2 frases de entrada, de las cuales la primera fuerce a "myAlice" a que realice una pregunta cerrada y que la segunda sea "sí" o "no" ("yes", "no" en inglés)
2. Hacer que el sistema imprima en un archivo de *log* el resultado de la frase nueva generada, es decir la frase independiente del contexto.
3. Ingresar las frases en el chatbot.
4. Verificar el resultado del *log*

Ejecución:

Caso	1º frase de entrada	Respuesta esperada del chatbot	2º frase de entrada
1	TALK TO ME	Are you a Star Trek fan?	yes
2	TWO PLUS FOUR	Are you a math teacher?	yes
3	I AM OMNIPOTENT	Are you God?	yes
4	thank god for that	Do you believe in God ?	yes
5	MY TEACHER hates me	Do you go to high school?	yes

Tabla 5.4.3.3.5: Ejecución de pruebas: Integración myAlice con conversor, conversión de respuestas sí/no

Resultados:

Caso	Diálogos ALICE	Diálogos myAlice
1	usuario: TALK TO ME ALICE: Are you a Star Trek fan? usuario: yes ALICE: What series is your favorite? I like Voyager best.	usuario: TALK TO ME myAlice: Are you a Star Trek fan? usuario: yes myAlice: Voyager is my favorite series.
2	usuario: TWO PLUS FOUR ALICE: Are you a math teacher? usuario: yes ALICE: Do you believe that grading is a good idea?	usuario: TWO PLUS FOUR myAlice: Are you a math teacher? usuario: yes myAlice: How much freedom do you have in choosing the curriculum for your class?
3	usuario: I AM OMNIPOTENT	usuario: I AM OMNIPOTENT

	ALICE: Are you God? usuario: yes ALICE: Hmm.	myAlice: Are you God? usuario: yes myAlice: What's up, god.
4	Human: thank god for that ALICE: Do you believe in God ? Human: yes ALICE: Ayuh.	Human: thank god for that myAlice: do you believe in god ? Human: yes myAlice: i believe he is with us when we speak of him.
5	Human: MY TEACHER hates me ALICE: Do you go to high school? Human: yes ALICE: It goes without saying.	Human: MY TEACHER hates me myAlice: Do you go to high school? Human: yes myAlice: what is your favorite subject?

Tabla 5.4.3.3.6: Resultados de pruebas: Integración myAlice con conversor, conversión de respuestas sí/no

En el *log* se imprimió para cada caso la segunda frase "yes", reformulada

Caso	Log: "yes" reformulado
1	I am a Star Trek fan
2	I am a math teacher
3	I am God
4	I believe in god
5	I go to high school

Tabla 5.4.3.3.7: Resultados de pruebas: Integración myAlice con conversor, conversión de respuestas sí/no, contenido del log

Evaluación de Resultados:

Los resultados fueron los esperados. El modulo funciona correctamente, dentro de sus limitaciones, integrado en el chatbot.

iv. Verificar que si la frase de entrada no está en el conjunto de casos favorables y no hay información de contexto. El sistema preguntará por la información faltante.

Descripción:

1. Construir un archivo de texto con 5 frases dependientes del contexto.
2. Ingresar cada una de las frases en el chatbot como la primera de un dialogo (o sesión)
3. Verificar que la respuesta del chatbot sea una pregunta sobre la información que se pidió o sobre la que se hizo referencia en cada una de las frases.

Ejecución:

Se escogieron los siguientes casos de pruebas, son frases en inglés en donde en

cada caso se hace referencia a un dato el cual debería de ser obtenido, del contexto en una conversación normal. Como es una primera frase en una conversación, el sistema deberá detectarla y preguntar por la información faltante.

Caso	Frase de entrada
1	do you like her?
2	is that fun?
3	the pencil is there
4	what is your best?
5	those pants are cool

Tabla 5.4.3.3.8: Ejecución de pruebas: Integración myAlice con conversor, referencias no resueltas

Resultados:

Caso	Salida generada por el subsistema
1	who do you refer with "her"?
2	what is fun?
3	where is "there"?
4	my best what?
5	what do you refer with "those"?

Tabla 5.4.3.3.9: Resultados de pruebas: Integración myAlice con conversor, referencias no resueltas

Evaluación de Resultados:

Los resultados son los esperados,

5.4.4 Ejecución de las pruebas de aceptación del sistema

El propósito de las pruebas de aceptación es buscar casos puntuales en los cuales se demuestre una mejoría en los diálogos entre un usuario y "myAlice" en comparación con "ALICE" original utilizando las mismas líneas de dialogo inicial.

Pruebas:

Descripción:

Se construirán 6 casos de prueba, los cuales serán diálogos cortos de dos líneas escritas por parte del usuario. Dichos casos se correrán en "ALICE" y en "myAlice" y se compararán ambos resultados. En el chatbot "myAlice" con todos los subsistemas implementados los casos deberán hacer uso de las mejoras implementadas: subsistema de resolución de

referentes, subsistema de conversión de preguntas cerradas y el subsistema generador de preguntas.

Ejecución:

líneas iniciales de dialogo, incluyendo una respuesta del chatbot que debería ser igual en ambos casos

Caso	1° frase	Respuesta chatbot	2° frase
1	taxi drivers are crazy	i think there might be a few exception.	but they work so hard
2	Richard Wallace	are you asking about my botmaster?	He is a great inventor
3	Two plus four?	Are you a math teacher?	yes
4	thank god for that	Do you believe in God ?	yes
5	is that fun?		
6	Do you like her?		

Tabla 5.4.4.1: Ejecución de pruebas: Aceptación del sistema

Resultados:

Caso	Dialogo con "myAlice"	Dialogo con "ALICE"
1	Human: the taxi drivers are crazy myAlice: i think there might be a few exception. Human: but they work so hard myAlice: is that a fact	Human: taxi drivers are crazy ALICE: Are all taxi drivers crazy? Human: but they work so hard ALICE: Huh. Who does "they" refer to?
2	Human: Richard Wallace myAlice: are you asking about my botmaster? Human: He is a great inventor myAlice: i knew he was an inventor, but no one ever said he was a great inventor.	Human: Richard Wallace ALICE: Are you asking about my botmaster? Human: He is a great inventor ALICE: Maybe you should tell him how you feel about him.
3	Human: Two plus four? myAlice: Are you a math teacher? Human: yes myAlice: How much freedom do you have in choosing the curriculum for your class?	Human: Two plus Four? ALICE: Are you a math teacher? Human: yes ALICE: Do you believe that grading is a good idea?
4	Human: thank god for that myAlice: do you believe in god ? what do you refer with "that"? Human: yes myAlice: i believe he is with us when we speak of him.	Human: thank god for that ALICE: Do you believe in God ? Human: yes ALICE: Ayuh.
5	Human: is that fun? myAlice: yes. .what is fun?	Human: is that fun? ALICE: Yes.

6	Human: do you like her? myAlice: who do you refer with "her"?	Human: do you like her? ALICE: She seems nice to me.
---	--	---

Tabla 5.4.4.2: Resultados de pruebas: Aceptación del sistema

Evaluación de los resultados:

Los casos se pueden dividir en tres, los dos primeros hacen uso del subsistema de resolución de referentes, los dos siguientes hacen uso del subsistema de conversión de preguntas cerradas y los dos últimos del subsistema de petición de información faltante (o subsistema de generación de preguntas) . Todos ellos hacen uso del subsistema de detección de independencia del contexto. Los resultados mostrados aquí dan una idea acabada de las mejoras implementadas, por lo tanto este caso de prueba cumple su objetivo.

5.4.5 Aprobación de sistema de información

En reunión mantenida con el director de tesis, se han aprobado el sistema de información y el resultado general de las pruebas detalladas en el presente documento.

6. Conclusiones

Para el problema de la clasificación de frases en los grupos "dependiente del contexto" e "independiente del contexto", no se encontró una solución óptima, sin embargo se encontró una solución suficientemente buena como para completar el desarrollo de la Tesis. El perceptrón multicapa con *Back Propagation* que utilizó como *input* el primer mapa generado compuesto de un vector de números en punto flotante de 15 posiciones logró un 54 % de efectividad mientras que el perceptrón multicapa con *Back Propagation* que utilizó el segundo *input*, la matriz de 15x5 números en punto flotante logró una efectividad de solo el 36 %.

Por su parte la red SOM, Kohonen no logró construir conjuntos de frases de forma tal que la red *Back Propagation* aplicada a cada uno de los subconjuntos generase un error cuadrático medio por debajo del error cuadrático medio hallado para el conjunto total. En cada caso existió un conjunto con un error cuadrático medio igual o superior al que la red generaba (tras un mismo número de iteraciones) sobre el conjunto general.

Es difícil ignorar la incompletitud de la base de datos "Lexico" utilizada para generar los mapas y la posibilidad de saber hasta qué punto los mapas creados representan las propiedades de las frases que interesa analizar. Sin duda ambos pueden ser mejorados, la dificultad radica en saber cómo generar mejores mapas de entrada y en como optimizar la clasificación de las palabras para aumentar los casos positivos.

Si bien el método normal de resolver problemáticas relacionadas con la sintaxis de una sentencia es a través de algoritmos determinísticos, la implementación de esta solución a través de redes neuronales tuvo un resultado satisfactorio.

El algoritmo de resolución de referentes: *RAP*, si bien no logró una tasa tan alta de efectividad como la descrita en [Lappin, Leass 1994] logró una efectividad suficientemente buena como para que el chatbot logre responder de forma satisfactoria a muchas de la frases con referencias anafóricas (que anteriormente no lograba responder correctamente). La disminución en la efectividad se debió a dos factores: el cambio del parser de procesamiento de lenguaje natural, ya que no se consiguió el mismo utilizado y fue reemplazado por otro de la biblioteca "Open-NLP" (que difiere levemente en su forma de comportarse.) La segunda razón es el cambio radical del tipo de discurso empleado, en [Lappin, Leass 1994] midieron la eficacia utilizando textos de manuales de computadoras y no diálogos coloquiales casuales.

Por último el procedimiento encargado de reformular las respuestas a preguntas cerradas

logró un éxito considerable, mayor del esperado ya que de una forma muy simple logra transformar las preguntas del chatbot en sentencias declarativas. Es un cambio que podría implementarse de forma productiva tal y como está de forma independiente de las otras mejoras.

7. Futuras líneas de investigación

Dentro de la misma problemática de la interpretación pragmática en chatbots quedan varios aspectos para seguir investigando y mejorando la calidad de las respuestas. Por un lado se tiene el problema de la clasificación de las frases en “independientes del contexto” y “dependientes del contexto”, dicha problemática podría ser mejorada, en principio, de dos formas:

Mejorando los sistemas inteligentes propuestos en esta tesis para así obtener una tasa de efectividad más elevada al clasificar los casos, lo cual podría lograrse, por ejemplo, aplicando lógica borrosa y sistemas difusos para todos aquellos casos que no pueden ser definidos de forma unívoca dentro de uno u otro conjunto.

La otra forma de mejorar la clasificación sería utilizando métodos algorítmicos determinísticos, sería interesante medir la tasa de efectividad de estos en la clasificación de casos contra una clasificación hecha con sistemas inteligentes. Una forma algorítmica de encarar el problema podría ser a partir del algoritmo *RAP*, modificándolo para que cumpla este objetivo.

Por otro lado se podría aumentar la tasa de éxito del algoritmo *RAP*, mejorarla adaptando el algoritmo al tipo de discurso propio de los diálogos casuales sería una opción interesante como trabajo de investigación. También se podría incorporar un sistema para que el algoritmo “aprenda” de los diálogos; en el sentido de incorporar nuevas palabras y nuevos casos de conocimiento del tipo “respuesta-estimulo” como es el caso de AIML, también nuevos sustantivos, en especial nombres propios para el caso de *RAP*, ya que los nombres propios son sustantivos muy comunes y difícilmente estén todos en una base de datos armada previamente.

Nuevas líneas de investigación se abren al encarar otros problemas devenidos también por la falta de interpretación pragmática en los chatbots al generar sus respuestas. Estos problemas fueron descritos en el punto 3, son la ambigüedad léxica en una sentencia, la elipsis, etc.

8. Referencias

- [CIRG 2008] CIRG. 2008. *Cybernetic Intelligence Research (Group) – Loebner Prize 2008*. <http://www.rdg.ac.uk/cirg/loebner/cirg-loebner-main.asp>, página vigente al 31/08/08
- [Fitrianie 2002] Fitrianie, S. 2002, *My_Eliza A multimodal Communication System*, Master of Science thesis, Delft University of Technology.
<http://www.kbs.twi.tudelft.nl/Publications/MSc/2002-Fitriani-MSc.html>. página vigente al 31/08/08
- [Funahashi 1989] Funahashi, K. I. 1989 *On the approximate realization of continuous mappings by neural networks*. Neural Networks, 2,pag. 183-192
- [García Martínez,Servente,Pasquini 2003] Ramón García Martínez, Magdalena Servente, Diego Pasquini 2003. *Sistemas Inteligentes*. Editorial Nueva Libreria.
- [Hilera, Martínez 1994] Hilera J. R., Martínez V.J. 1995. *Redes Neuronales Artificiales: Fundamentos, Modelos y Aplicaciones*. Editorial RA-MA
- [Hobbs1978] Hobbs, Jerry 1978.*Resolving pronoun references*, Lingua 44, pag. 311-338.
- [Lappin, Leass 1994] Lappin, S. and Leass, H.J. (1994). *An algorithm for pronominal anaphora resolution*. *Computational Linguistics* 20(4),pag. 535-561
- [Loebner 2006] Loebner Contest, 2006. *Loebener Prize 2006 Information*, http://loebner.net/Prizef/2006_Contest/loebner-prize-2006.html, página vigente al 04/04/09
- [Loebner 2007] Loebner Contest, 2007. *17th Annual Loebner Prize for Artificial Intelligence*, http://loebner.net/Prizef/2007_Contest/loebner-prize-2007.html, página vigente al 04/04/09

- [Loebner 2008] Loebner Contest, 2008. *Loebener Prize 2008 Information*, http://loebner.net/Prizef/2008_Contest/loebner-prize-2008.html, página vigente al 04/04/09
- [Loebener 2009] Loebner Contest, 2009. *Loebener Prize 2009 Information*, <http://www.loebner.net/Prizef/loebner-prize.html>, página vigente al 04/04/09
- [Martín del Brío,Sanz Molina 2001] Bonifacio Martín del Brío, Alfredo Sansz Molina 2001. *Redes Neuronales y Sistemas Difusos*. Editorial RA-MA
- [Müller 1990] Müller, B., Reinhardt, J. *Neural Networks. An Introduction*, Springer-Verlag, 1990
- [Penrose 1989] Penrose, Roger 1989. *The Empreror's New Mind*. Oxford Univesity Press.
- [Rusell, Norving 2003] Stuart Rusell, Peter Norving 2003. *Inteligencia Artificial, Un enfoque Moderno*. Editorial Prentice Hall
- [Searle 1980] Searle, John 1980. *Mind, brains and programs*. Behavioral and Brain, Sciences. vol 3, nº 3, pag. 417-457
- [SOM_PACK 1995] SOM_PACK. *The self-Organizing Map Program Package*. Helsinki University of Technology, Finland, abril 1995.
- [Turing 1950] Turing, A. 1950. *Computing Machinery and Intelligence*. Mind. Vol. LIX, N° 236, pag. 433-460
- [Wallace 2003] Wallace, R. 2003. *The Elements of AIML Style*. ALICE A.I. Foundation.
- [Weizenbaum 1966] Weizenbaum, J. 1966. *ELIZA – A computer Program for the Study of natural Language Communication Between Man And Machine*, Communication of the ACM. Vol 9, N° 1, pag. 36-45

Anexo A: Glosario

A continuación se definen algunas palabras y frases que se han usado a lo largo de este trabajo de investigación con un significado muy particular que no necesariamente coincide con el significado ordinario.

pregunta cerrada: aquella susceptible de ser respondida mediante las palabras "sí", "no" o alguno de sus sinónimos. Ejemplo: ¿sos hombre?

frase: sentencia, oración.

contexto: Discurso, frases intercambiadas anteriormente entre el usuario y el sistema

evaluación del contexto: interpretación pragmática

frase dependiente del contexto: Es una frase que para poder ser "entendida" correctamente hay que agregarle información que está en el contexto. Por ejemplo: "él lo dijo" (¿Quién es él? es la información que falta)

frase independiente del contexto: Frase que no requiere más información para ser "entendida". Por ejemplo "¿Cómo te llamas?"

palabra: una o más cadenas de caracteres que pueden estar o no separadas por espacio y que poseen un significado independiente. Por ejemplo, según esta definición "Buenos Aires" es una sola palabra y no dos. Pero también "a" o "mesa" son palabras, como indica la definición ordinaria.

subsistema: modulo constituido por una o más clases con una funcionalidad y responsabilidad específica. Forma parte de un sistema mayor. No se utilizó la palabra algoritmo o subrutina, porque un subsistema puede comprender a más de un algoritmo o bien estar compuesto por código no algoritmo como ser una red neuronal artificial.

Anexo B: Datos del repositorio on-line

Todo el código del proyecto, su documentación, el resultado de las pruebas y las bases de datos se encuentran disponibles al público en general en un repositorio on-line, provisto por Google. La url de acceso se muestra a continuación:

<http://tesischatbot.googlecode.com/svn/trunk/>

No requiere credenciales de acceso, el repositorio se puede descargar de forma anónima. Todo el proyecto se encuentra liberado bajo la licencia GPL.

Dentro hay dos directorios, "src" y "test", el primero contiene el código fuente de cada uno de los proyectos relacionados con el proyecto de investigación, incluido el directorio "myAlice" que contiene la versión final del sistema descrito. El segundo contiene resultados de las pruebas efectuadas sobre el sistemas, como se describió en el capítulo: 5.4

Repositorio vigente al 9 de mayo de 2010.