Sistema N-dimensional semantico-en-paralelo escalable y dinámico de entrenamiento-consulta neuro-linguistica alimentado através de contribuciones voluntarias de usuarios del mundo real de las redes sociales.

Hassan Uriostegui . 2012 . hassan.uriostegui@gmail.com

1. Introduccion

La influencia de los motores de indexado y búsquedas de internet que alcanzaron su punto defintiivo de caracterización tecnológica y como modelos de negocio entre 1994 y 2001, es de suma relevancia en la manera que los usuarios modernos se han acercado a internet, y más aún han creado un lenguage y un paradigma de comunicación digital en los últimos 15 años.

La existencia de este paradigma de comunicación ha influido aún en los más reicentes productos de comunicación digital global, tales como los servicios ofrecidos por twitter o facebook.

En ete paradigma de comunicación digital, los primeros robots encargados del indexado de los contenidos en internet, tenían que enfrentarse al problema de categorizar y etiquetar miles de páginas ya creadas, para cumplir con el objetivo de que su contenido pudiera ser encontrado al buscar la coincidencia de alguno de sus contenidos constitutivos con alguna o todas las palabras de búsqueda.

Durante el desarrollo de estos motores de búsqueda , se experimentó con una importante inversión de capital de riesgo en las diferentes compañías que habrían de luchar por ubicarse como las líderes del mercado.

De entre ellos google ofreció una mejora en su sistema de indexamiento categorizando además el número de links que apuntaban hacia una misma dirección web, y empleando después este criterio de prioridad al “hyper-linking” para identificar a las páginas con redes más amplias de conexión como aquellas que ofrecían mejores resultados para el término de búsqueda.

Desde una perspectiva actual, y a futuro, los sistemas de indexamiento primigenios ofrecen una acercamiento muy primitivo en el contexto del lenguage humano, ya que su sistema de etiquetamiento, desnaturaliza el contexto y el sentido de lo que se dice, ofreciendo una capacidad pobre de comunicación humanamente amigable con los usuarios, exigiendoles un aprendizje y el desarrollo de estrategias linguisticas, con el fin de mejorar los resultados obtenidos por el motor de búsqueda.

Esto va de la mano con el cambio de la naturaleza de los contenidos creados mayoritariamente en el internet.

Mientras que originalmente internet era empleado como un medio de divulgación enteramente científico y posteriormente evolucionó en diversos sentidos, no es sino recientemente que su sentido principal como herramienta de consulta para información descriptiva y estática relativa a elementos de la historia reciente o antigua, de los más diversos tópicos de la vida humana en general, se ha visto ampliamente superado.

Se puede rastrear una primer ruptura con la aparición de los blogs alredor de 1999-2004, que se convertirían por primera vez en una herramienta masiva y de fácil acceso que haría crecer notablemente la cantidad de autores de contenido “personal” , donde por primera vez millones de personas podría crear sus propios sitios web sin necesidad de tener conocimiento alguno en el funcionamiento de internet o las páginas web.

Ese sería apenas un primer esbozo donde comenzarían a crearse fuentes auto-sustentadas de información describiendo a detalle diversos aspectos de la forma de vida única y personal de los autores.

Cuando lo vemos en retrospectiva, mientras que en un primer momento de el internet , dada la mayor riqueza de contenidos estáticos y de tópicos genéricos, la búsqueda por coincidencia de palabras resulta suficiente, en el sentido de uso de quien va a una biblioteca a consultar algún dato puntual, o quien busca en un quiosco de revistas, alguna fuente de entretenimiento que coincida con sus intereses generales.

Sin embargo cuando analizamos el punto de ruptura

ocacionado por el fenómeno de “blogging” podemos observar que la metáfora de comunicación resulta un tanto dificil de igualar.

Máxime si seguimos su evolución hasta llegar a las pubilicaciones en “tiempo real” que se pueden realizar através de facebook y twitter, sobre los sentires más inmediatos y francos de los autores respecto a si mismos, sus círculos sociales, y los más diversos tópicos, siempre caracterizados por ser profundamente personales.

En este sentido surge una nueva intención de búsqueda que va más alla de la simple igualación por coincidencia de palabras.

Se trata de una intención de búsqueda más humana y personal que debe atender a necesidades complejas en terminos de computo, por ejemplo encontrar personas que se sienten igual que yo, descubrir momentos de mi vida donde me he sentido como en la actualidad, o más aún encontrar otros humanos que complementen mi manera de sentir.

Esta parte de una necesidad de como la comunicación humana espera que funcione al interactuar con otro interlocutor humano, ya que cada vez más para las personas, la conciencia de que la interacción con internet se es dada através de máquinas y software, se extingue notablemente, y constantemente se refuerza la idea, de que exactamente del otro lado de la pantalla se encuentran otros seres humanos interactuando conmigo, constantemente.

En este sentido debemos hacer una pausa para reflexionar sobre la manera en como estos blogs sociales, diseñados para vincularnos con nuestras familias, amigos, o para crear seguidores de nuestros pensamientos, se han convertido en una manera evolucionada de los canales de comunicación convencionales.

El uso más arraigado notablemente en las nuevas generaciones de las redes sociales digitales, suple la existencia de canales de comunicación más directa, y empuja a la creación de sistemas de indexado y búsqueda que se encuentren a la par de la complejidad del nivel de comunicación que se encuentran transmitiendo.

Para puntualizar claramente las dos posturas, podríamos decir que es mucho más “simple” indexar en respecto a un tema humano inpersonal y superficial bien delineado que representa una realidad genérica que habra de ser encontrado a través de criterios de búsqueda mas o menos uniformes, que indexar pensamientos y frases breves pero que entrañan una asombrosa requiza personal enlazada profundamente con la historia del autor, que aun cuando resultan breves en términos de cantidad de palabras ofrecen una compleja y valiosa riqueza de información no directamente en lo que se escribe, sino en el análisis de como “se siente” y que “significado contextual” tiene lo que se dice.

En este punto queda claro que los sitemas actuales de indexamiento son apropiados para contenidos impersonales, mientras que para el indexado y búsqueda de contenidos personales es necesaria la creación de técnicas novedosas que permitan indexar, el “sentir” y el “contexto” de lo que se dice a fin de crear una comunicación de más alto nivel entre el usuario y la máquina.

1b. Sistemas semánticos actuales.

En los últimos 5 años la implementación masiva de sistemas semánticos en productos de consumo vino a ofrecer un esbozo de lo que serían los más novedosos sistemas de indexamiento y búsqueda.

Estos fueron empujados principalmente con la finalidad de crear sistemas inteligentes para la identificación de “puntos de venta virtuales” inmediatos generados a partir de estudiar el contexto semántico de lo que se escribe, y poder así mostrar publicidad muy dirigida y oportuna en el momento exacto de hablar de la necesidad.

Se pueden ver ejemplos de esto en el actual servicio de Gmail, o en la publicidad mostrada dentro de Facebook. De tal manera que si en nuestra cadena de mensajes nos encontramos hablando de peces, es factible que el sistema nos muestre publicidad relacionada con la pesca y los acuaristas.

Aparentemente y sin poder tener acceso a información exacta estos sistemas realizan una valoración semántica de cada palabra, o bien como optmización únicamente de las palabras que aparecen con mayor frecuencia a lo largo de una

cadena de intercambio de correos o mensajes, para luego dar mayor relevancia a aquellas más repetidas o que comparten valores semánticos entre si, a fin poder definir de manera general el tema o temas de lo que se habla y entonces poder usar esa información del tema de interés para cualquier fin convenido.

Las técnicas actuales de búsqueda semántica, conocidas en esta investigación y desarrollo, se basan en el uso de diccionarios que relacionan familias de palabras con grupos semánticos compartidos.

Debido a la complejidad de mantener diccionarios que abarquen todas las palabras existentes en un determinado lenguage, es bien aceptado el uso de diccionarios que contienen unicamente lo “lemmas” o raices linguisticas de las palabras, omitiendo asi conjucaciones y variaciones verbales, limitando notablemente el número de palabras y relaciones.

De manera genérica se puede decir que el proceso de relación semántica requiere de un proceso de desconjugación , llamada también desambiguación morfológica linguistica, en el cual se pretende poder extrar el lemma que corresponde con la palabra a analizarse, para posteriormente poder buscar su coincidencia en el diccionario semántico.

Sin embargo este proceso no es siempre sencillo ya que según las diferentes funciones de la lengua y sentidos que puede cobrar una misma palabra, el desambiguador morfológico se enfrente comunmente al caso de que una palabra podría estarse usando en múltiples sentidos.

A fin de determinar el sentido y el uso de la lengua apropiado para la palabra que esta siendo analizando en la expresión de alguna oración, el sistema de analisis semántico se puede apoyar de los usos y sentidos de la lengua de palabras vecinas, para ayudar a realizar una valoración contextual del sentido en el que se quiere emplear la palabra.

Sin embargo también comunmente, las palabras vecinas suelen tener tambien más de un sólo uso de la lengua o sentido , haciendo aún más dificil la tarea de desambiguación morfológica.

Finalmente los sistemas más precisos terminan usando métodos de fuerza bruta que primeramente interpretan todos los usos posibles del lenguage para la desambiguación morfológica y terminan empleando algún sistema de valoración de patrones y formas linguisticas más comunes apoyados tambien por reglas gramaticales, para dar un puntaje diferente a cada uso del lenguage y luego pretender encontrar el uso de la lengua y sentido más apropiado para cada palabra.

Una vez que el transformador morfológico realiza su mejor tarea, el lemma , o lemmas obtenidos son buscados en los diccionarios semánticos, y devuelven así la valoración semántica de cada palabra.

Como se verá más adelante, este proceso de transformación morfológica del lenguage, así como sus subyacentes procesos de desambiguación de uso y sentido del lenguage, son suficientemente complejos para considerarse un tema aparte de lo aquí discutido en relación a sistemas de indexado novedosos, sin embargo son parte constitutiva de la investigación por lo que se muestran las soluciones más adecuadas y las potimizaciones que se realizarón para la creación del sistema neuro-linguistico semantico-en-paralelo aquí discutido.

2. Propuesta y límites del sistema.

Dado el estado de cosas discutido en nuestra introducción, se propone la creación de un sistema de indexado y búsqueda textual que ofrezca un nivel de comunicación mucho más elevado y complejo entre la máquina y el usuario, capaz de recrear la sensación de interactuar con un locutor humano, ofreciendo la misma flexibilidad y sensibilidad de entidimiento que se esperaría de una conversasión humana convencional.

En este sentido entra en juego un concepto de interacción humana que comunmente se le denomina “sentido común”, que se puede entender como la capacidad que poseen la mayoría de las personas, para emitir en juicio que se considere socialmente razonable sobre una determinada situación o cosa.

Segun este razonamiento y a fin de delinear los limites del concepto, el “sentido comun” se encuentra definido primeramente por contextos socio-culturales , y es transmitido de manera

generacional, en la genealogía de las sociedades.

Sin embargo según la generalidad de los tópicos sujetos de la evaluación y del estado de similutd que ciertos temas pueden guardar de manera general determinadas necesidades fundamentales inherenetes al sentido humano más profundo, podemos ir escalando através de diversas interpretaciones de "sentido comun" llendo desde las más singulares en círculos más cerrados y pequeños como pueden ser el familiar o de amistades, hasta ir escalando por las agrupaciones sociales y llegar al "sentido comun" de la humanidad.

En esta relfexión resulta válido hablar entonces de que internet através de las redes sociales agrupa y genera contenido sujeto a la calificación del "sentido comun" grupal correspondiente a la red donde el autor del contenido siente que se encuentra tansmitiendo su mensaje.

(Aquí se hace una pausa, para reflexionar sobre como internet y las redes sociales le exigiría al hablante ampliar su capacidad cognitiva en el sentido de virtualizar a sus oyentes y dimensionarlos según diversos grupos sociales, que operan simultaneamente en diversos niveles de agrupación, en un sentido que en el mundo físico resulta inexitente. Esta exigencia supera por mucho a la capacidad de comunicación del hablante actual, quien al hablar apenas puede visualizar a un pequeño grupo de personas de reciente contacto, generando el fenómeno de escribir actualizaciones e información personal en facebook que realmente impacta en un reducido número de oyentes realmente considerados en la dirección del mensaje, mostrando también una necesidad que se abordará más tarde cuando hablemos de los criterios de búsqueda de nuestro sistema).

Regresando al tema del "sentido común" , entonces el hablante o autor del contenido, transmite un mensaje calificado y de alguna manera "sensurado" en la espera de impactar de una manera determinada en el grupo que el piensa que lo escucha.

En este hablar cobra vital relevancia, la estructura y la selección exacta de palabras sinónimas, ya que en la discriminancia de ellas, se pueden vislumbrar complejas relaciones sémáticas que nos ayudan a entender como el "sentido comun" opera en la formación de los conceptos y su traducción a palabra escrita.

Así pues con estos antecedentes se plantea como meta la creación de un sistema experto capaz de funcionar como lo haría el "sentido común" en el proceso de calificación y selección de resultados similares a fin de poder ofrecer temas de conversación en comun y crear la sensación de tener un diálogo humano y natural con el hablante, permitiendole así tener una comunicación del más alto nivel con la interfase máquina y permitiendole alcanzar experiencias de interacción hasta ahora desconocidas en el estado de la técnica general.

2b. Superación de límites y metas máximas.

Asimismo se plantea que la creación de un sistema de "sentido común" general llevado a su máxima expresión podría operar a modo de un servicio de consulta global y general para que otros sistemas más especializados se interconecten y pudieran así resolver de manera rápida, y de aprendizaje acumulativo, complejas desiciones disyuntivas, pero adicionalmente, contribuir a la creación de sistemas de inteligencia artificial colectiva, cuyo entrenamiento generalizado y no destructivo, podría permitir la creación de una "conciencia" virtual que funcionaría a modo de "sentido comun" humano y especializado para ser consumido por las más diversas interfases fuesen máquinas robóticas o software puro. Se cita como ejemplo quie mientras un sistema experto podría entrenar a un robot a caminar, podría consultar el "sentido comun" global para determinar como debe caminar en base a la relación semántica disyuntiva de su entorno, podría así determinar que en un hospital debería caminar lentamente y extremando cuidados, mientras que en un parque o terreno abierto, lo podría hacer de manera más rápida y desenfadada, apelando a como lo hace el sentido común en los humanos.

3a. Planteamiento metafórico y observación.

Para hacer más acequible al lector se plantea en términos generales que el objeto del presente sistema es la recreación del funcionamiento de ciertas estructuras del cerebro humano, específicamente de las neuronas y su proceso de sinapsis.

De manera también general se puede plantear que está comprobado que si se somete a un espectador

humano a una serie de pruebas , mientras es monitoreado através de un encefalograma, o algún otro aparato capaz de medir la capacidad electrica del cerebro en sus diferentes regiones, cuando el sujeto de analisis es sometido a estimulos suficientemente fuertes y singulares, es posible identificar con los aparatos, patrones electricos que se repiten de manera similar através de la corteza cerebral.

Así pues se plantea que através de estos sensores es posible caracterizar el comportamiento electrico del cerebro y sus respuestas a estimulos singulares.

Adicionalmente y según los estudios en el área de neurología, se ha podido comprobar que esta actividad eléctrica es el producto de un procesos conocido como sinapsis , donde una red específica de neuronas se ve exitada electricamente , por respuesta a un estimulo del sistema nervioso, producido por una reacción química, transmitiendo impulsos de diversa intensidad y en grupos específicos para generar una respuesta igualmente eléctrica que se transmite a cualquier otro tipo de estructura del sistema nervioso y finalmente muscular o glandular.

La intensidad y forma en que este proceso sináptico se realiza a lo largo de la cadena de transmición neural , se puede caracterizar y medir como ya se ha descrito antes, y constituye una explicación metafórica y una guía de observación en la forma en que nuestro sistema está constituido.

Así pues se plantea la creación de un sistema de objetos en la arquitectura de software, que reproduzca el comportamiento de las neuronas en su proceso sináptico descrito en grupos de neuronas que se exitan de manera específica ante los estimulos externos, (que en este caso son palabras), y produza complejas respuestas basadas en la interacción paralela del conjunto neuronal virtual.

3b. Planteamiento tecnológico y arquitectura general.

Debido al alto nivel de complejidad y volumén de procesaimiento de datos, para poder plaicar masivamente en un sistema de servicio en linea, el siguiente sistema, se planteo el uso del lenguage C, a fin de obtener la mayor velocidad posible en los procesamientos.

La variante de C seleccionada fue la creada por la compañía NExT y soportada por actualmente por Apple “C-Objetivo”, ya que ofrecía la posiblidad de desarrollar un laboratorio de bajo costo en un entorno Unix de fácil administración, (como lo es el sistema opertavio Mac OS X Snow Leopard) , que permitiera además desarrollar facilmente de una manera humanamente legible, pero además extremandamente estructurada, la rápida creación de compleas estructuras de objetos, manteniendo un código legible y de fácil mantenimiento, que además pudiera ser portado a otras arquitecturas.

En este sentido se confirmó que el código C-Objetivo escrito sin dependencia de los frameworks de Apple o de Cocoa, podia ser fácilmente recompilado posteriormente en otras paltaformas usando el compilador GCC correspondiente.

Asimismo el entorno de desarrollo xCode ofrecería una rápida y cinfiable plataforma que permitiría simplificar las labores de creación de builds y deliveries.

Adicionalmente se optó por las tecnologías gratuitas PHP y MySQL para ejecutar el frond-end del sistema. Asimismo se habilito en el servidor los permisos de ejecucion shell para PHP, a fin de ejecutar de esta manera las verdaderas rutinas de procesamiento corriendo los ejecutables de C-Objetivo , y posteriormente haciendo una conexión directa entre C-Objetivo y MySQL. Aun cuando se estimó el uso de una solución más robusta en el sistema de producción para el manejo de la base de datos, se optó por MySQL al ser una solución gratuita que permtía llegar a los alcances del prototipo de laboratorio.

La arquitectura del sistema se divide en dos grandes estructuras.

La primera es la estrucutra de aprendizaje y entrenamiento semantico linguistico, mientras la segunda es la de analisis y entrenamiento neuronal o de experiencia.

Estas estructuras no se encuentran completamente separadas, sino que por el contrario comparten las rutinas de análisis linguistico, y acceso a datos.

Asimismo existen dos grandes casos de uso.

El primero es el del entrenamiento continuo através de un robot de indexado, en este sentido, el sistema es alimentado desde una fuente continua de informacion como lo es la API de streaming de twitter. En este caso es importante decir que se pretende no almacenar en si la información de los tweets y sus usuarios, sino solamente el resultado de los analisis.

El segundo es el caso de uso de usuario real, en este caso, el usuario realiza una consulta enviando su id y una cadena de palabras para analizar al servidor, siendo estas igualmente analizadas y almacenadas igual que en el caso anterior, pero adicionalmente enviando una respuesta al cliente.

Para el sistema de consulta se orientó a la creación de un servicio web, que permitiera su uso desde internet atraves de una interface web basada en HTML5 y CSS3, soportada a través de una serie de servicios REST proveidos por PHP que ejecuta en el servidor todas las rutinas de analisis escritas en C-Objectivo conectandose directamente a MySQL, luego devolviendo los ersultados através de la salida estandar “stdout” a php, para formatearlos y devolver al frontend HTML5 para ser rendereados de la manera más conveniente.

4. Wordnet , diccionario lexico.

Tras una serie de primeros planteamientos en los que se consideró la creación de un diccionario semántico linguistico a partir del entrenamiento con categorizadores humanos, se descubrió de la existencia de una serie de diccionarios orientados al análisis linguistico por computadora y liberado como código abierto para fines de investigación por la universidad de Princeton.

Asimismo se decidió por el uso del idioma ingles en concideración sus relativamente uniformes reglas lexico-morfológicas, y tambien por considerar que el universo de usuarios del idioma inglés se ajustaría mejor posteriormente, al momento de definir el “sentido comun” de la cultura occidental.

Este diccionario llamado Wordnet, consta de una base de datos lexica agrupada en sets cognitivos sinonimos o “synsets” que definen las propiedades conceptuales de la noción cognitva y que se enlazan a una serie de “lemmas” o palabras sinónimas que corresponden con el mismo apelativo cognitivo.

Estas bases de datos son proveidas en un formato propio, por lo que adicionalmente se comparten una serie de programas escritos en C que funcionan como desambiguadores morfologicos y a la vez como drivers de acceso a la base de datos y su formato singular de una manera muy rapida.

El primer paso fue convertir el formato de la base de datos a un formato moderno más estandarizado como XML y posteriormente indexarlo en una serie de tablas MySQL para mantener el acceso rápido a los datos.

Para este fin se crearon rutinas que convertirian cada uno de los archivos de la base de datos de Wordnet a XML.

Principalmente Wordnet separa su base datos en varios tipos de archivos:

a) Archivos de definicion de los set sinonimo cognitivos “synsets” . Dividiendolo en 4 grandes grupos segun el uso de la lengua: adjetivos , adverbios, sustantivos y verbos. Ejemplo:

00029769 00 s 01 accumulative 0 003 & 00029343 a 0000 + 02304982 v 0103 + 00158804 v 0101 | marked by acquiring or amassing; “we live in an accumulative society”

b) Archivos de “indice” conteniendo los “lemmas” que se relacionan conlos synsets. Divididos igualmente en adjetivos , adverbios, sustantivos y verbos. Ejemplo:

extensional a 1 3 & + ; 1 0 00722707
extensive a 3 3 ! & + 3 2 01386234 00526062 01514598
extenuating a 1 1 & 1 0 00923671
exterior a 1 4 ! & ^ = 1 1 00952395
exterminable a 1 1 & 1 0 00898013
exterminated a 1 1 & 1 0 00734798

c) Archivos de excepcion, conteniendo mapeos de transformacion morfologica de palabras irregulares que no pueden obtener su raiz morfologica por las reglas gramaticales convencionales. Divididos

igualmente en adjetivos , adverbios, sustantivos y verbos.
d) Una tabla de categorias lexicas que ofrece una primera opcion para la categorizacion semantica de las palabras. Ejemplo :

```
00      adj.all   3
01      adj.pert3
02      adv.all   4
03      noun.Tops       1
04      noun.act        1
05      noun.animal     1
...
26      noun.state      1
27      noun.substance          1
28      noun.time       1
29      verb.body       2
30      verb.change     2
...
35      verb.contact    2
36      verb.creation   2
...
43      verb.weather    2
44      adj.ppl 3
```

A fin de convertir las bases de datos, indices, exlusion y categorias a formato XML, se escribieron una serie de rutinas, donde la clase NSScanner fue de vital ayuda en el proceso de escaneo y reconocimiento de la estructura de los datos.

Por ejemplo este es el método para convertir una fila en la base de datos de synsets de wordnet a un Objeto NSDictionary abstracto que luego pueder ser traducido a un plist XML.

Extracto de WNImporter.m

```
-(NSDictionary *)dictionaryFromWNDataString:(NSString *)data{


   // if([data length ]<4) return nil;

   NSMutableDictionary *tmp = [[NSMutableDictionary alloc] init];
        NSScanner * scanner =[[NSScanner alloc] initWithString:data];

   int location=0;
   [scanner setScanLocation:location];

   NSString *synset_offset =[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator intoString:&synset_offset];

   NSString *lex_filenum =[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator intoString:&lex_filenum];

   NSString *ss_type =[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator intoString:&ss_type];


   NSString *w_cnt =[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator intoString:&w_cnt];

   w_cnt = [self getHexStringValueFromString: w_cnt];

    NSMutableArray *words = [[NSMutableArray alloc] init];
   for (int i=0; i<[w_cnt intValue]; i++) {

      NSString *word=[[NSString alloc] init];
          [scanner scanUpToString:kFieldSeparator intoString:&word];

       NSScanner * subscanner1 =[[NSScanner alloc] initWithString:word];
     [subscanner1 scanUpToString:@"(" intoString:nil];

              NSString *advPos=[[NSString alloc] initWithString:@""];

                [subscanner1 scanUpToString:@"" intoString:&advPos];
      if([advPos length] >1){
         NSLog(@"%@",advPos);
      }
      [word stringByReplacingOccurrencesOfString:advPos withString:@""];
      word=[[NSString alloc] initWithString:word];

      NSString *lex_id=[[NSString alloc] init];
          [scanner scanUpToString:kFieldSeparator intoString:&lex_id];

              NSDictionary *tmp = [NSDictionary dictionaryWithObjectsAndKeys:
                  word,kWord,
                  lex_id,kWord_lexID,
                  advPos,kWordAdvPosition,
```

```
                    nil];
    [words addObject:tmp];

  }

  NSString *p_cnt =[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&p_cnt];

   //  p_cnt = [self getHexStringValueFromString: p_
cnt];



 NSMutableArray *pointers = [[NSMutableArray alloc]
init];
  for (int i=0; i<[p_cnt intValue]; i++) {

    NSString *pointer_symbol=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&pointer_symbol];



    NSString *synset_offset=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&synset_offset];

    NSString *pos=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&pos];

    NSString *sourcetarget=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&sourcetarget];

    NSString *source;
    NSString *target;
    if([sourcetarget length] == 4){
         source = [[NSString alloc] initWithString:
[sourcetarget substringWithRange:NSMakeRan
ge(0,2)]];
       source = [self getHexStringValueFromString:s
ource];

          target = [[NSString alloc] initWithString:
[sourcetarget substringWithRange:NSMakeRan
ge(2,2)]];
       target = [self getHexStringValueFromString:ta
rget];

    } else{
       source=@"-1";
       target=@"-1";
    }
          NSDictionary *tmp = [NSDictionary
dictionaryWithObjectsAndKeys:
              pointer_symbol,kPointer_symbol,
            synset_offset,kPointer_synset_offset,
              pos,kPointer_pos,
            //sourcetarget,kPointer_sourceTarget,
              source,kPointer_source,
              target,kPointer_target,
              nil];

    [pointers addObject:tmp];

  }



  NSString *f_cnt =[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&f_cnt];

  NSMutableArray *frames = [[NSMutableArray alloc]
init];

       if( [f_cnt compare:kGlossSeparator] !=
NSOrderedSame){

    for (int i=0; i<[f_cnt intValue]; i++) {

      NSString *symbol=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&symbol];

      NSString *f_num=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&f_num];

      NSString *w_num=[[NSString alloc] init];
        [scanner scanUpToString:kFieldSeparator
intoString:&w_num];

          NSDictionary *tmp = [NSDictionary
dictionaryWithObjectsAndKeys:
              symbol,kFrames_countsymbol,
              f_num,kFrames_fnum,
              w_num,kFrames_wnum,
              nil];

    [frames addObject:tmp];

    }
```

```
    }

    NSString *gloss  =[[NSString alloc] init];
    [scanner scanUpToString:@"" intoString:&gloss];


            NSDictionary *result = [NSDictionary
dictionaryWithObjectsAndKeys:
                    synset_offset,kSynset_offset,
                    lex_filenum,kLex_filenum,
                    ss_type,kSs_type,
                    w_cnt,kWord_cnt,
                    words,kWords,
                    p_cnt,kPointer_cnt,
                    pointers,kPointers,
                    f_cnt,kFrames_count,
                    frames,kFrames,
                    gloss,kGloss,
                    nil];

    return  result;
}
```

Posteriormente se escribieron rutinas para guardar en tablas MySQL el contenido de Wordnet.

Una vez completado el proceso de portación resulto de gran ayuda el poder contar con los archivos XML para la realización de pruebas de laboratorio durante el analisis exhaustivo que requqería millones de consultas a la base de datos. En este sentido el peso total de las diversas fuentes de datos XML ronda los 177MB, que una vez cargados en memoria RAM resultan en un muy rápido acceso de las rutinas de consulta desde C-Objetivo.

5. Árboles lexicos

El siguiente planteamiento fue encontrar la manera de poder entrenar a una neurona para que pudiera valorar el significado de una palabra en un contexto determinado.

En estido se planteo la disyuntiva de que un categorizador humano podria distinguir las diferencias entre conceptos polares (como el amor y el odio), usando el sentido comun y que el sistema debería ser capaz después de su entrenamiento de determinar si un concepto se relacionaba más con un término o con su opuesto.

La primera hipótesis fue que si preguntaramos al categorizador humano sobre como determinó a cual de los dos conceptos polares se parecía más el concepto en cuestión, este habría de ir explicando a manera de un árbol lexico-semántico los diversos criterios y conocimientos que le permitían deducir el juicio disyuntivo.

En este sentido se planteo realizar un proceso muy similar tomando cada uno de los conceptos polares y calculando su árbol léxico en un proceso iterativo que de describe como sigue:

1. Seleccionar la palabra raiz.
2. Ver si la palbra no se ha analizado ya, sino buscar en el diccionario la definición de su glosario sus conceptos hypernimos (superiores) e hipónimos (inferiores) aumentar el contador de jerarquia en la que se encontraron los conceptos y poner las definiciones anteriores en la entrada del ciclo. Si ya esta analizada ignorarla.
3. Iterar nuevamente por el paso 1 pero para cada una de las palabras del paso 2.

Este es la funcion principal de iteraccion descrita en el paso 2:

Extracto de DJSalomon.m

```
-(void) familyForWords:(NSString *)word {

    [lexTree addObject:word];

    NSArray *wordFound = [self searchForWordsSepar
atedBySpace:word];

   NSMutableArray *alllexUp = [[NSMutableArray alloc]
init];
     NSMutableArray *alllexDown = [[NSMutableArray
alloc] init];
  NSMutableArray *allwordRelated = [[NSMutableArray
alloc] init];
      NSMutableString *glosses = [[NSMutableString
alloc] init];


   NSMutableArray *alllsyns = [[NSMutableArray alloc]
init];
        NSMutableArray *allDomainDownLemmas =
[[NSMutableArray alloc] init];
          NSMutableArray *allDomainUpLemmas =
[[NSMutableArray alloc] init];
```

```
    for (int i=0; i<[wordFound count]; i++) {

                NSDictionary *worddict = [wordFound
objectAtIndex:i];

     NSString *offset = [worddict objectForKey:kSynset_
offset];
        NSString *pos = [worddict objectForKey:kPos];
        NSString *lexFile = [worddict objectForKey:kLex_
filenum];
         offset=[NSString stringWithFormat:@"%@%@%@"
,offset,pos];

          [self countSynsOnFamilyDict:worddict];

        if( ! [self countKeyOnFamilyDict:offset]){

           if(isMainLoop){
               if([invalidContexts objectForKey:lexFile] ==
nil){
                     [validLexContexts setObject:lexFile
forKey:lexFile];
                   NSLog(@"valid context :%d",[lexFile
intValue]);
               } else{

                  [validLexContexts setObject:[NSNumber
numberWithInt:-1] forKey:@"-01"];
               }
           }


      NSString *gloss = [worddict objectForKey:kGloss];
          gloss =[self cleanGloss:gloss];

                        NSArray *lexUp = [worddict
objectForKey:kWordsLexUp];
                      NSArray *lexDown = [worddict
objectForKey:kWordsLexDown];
                        NSArray *syns = [worddict
objectForKey:kWordsSinonyms];
                     NSArray *domainUp = [worddict
objectForKey:kWordsDomainsUp];
          NSArray *domainDown = [worddict objectFor
Key:kWordsDomainsDown];
                    NSArray *wordRelated = [worddict
objectForKey:kWordsRelated];

                       NSArray *synsLemmas= [self
wordLemmasInSynsArray:syns];
                    NSArray *lexUpLemmas = [self
wordLemmasInArray:lexUp];
                     NSArray *lexDownLemmas = [self
wordLemmasInArray:lexDown];
                  NSArray *wordRelatedLemmas = [self
wordLemmasInArray:wordRelated];
                 NSArray *domainDownLemmas = [self
wordLemmasInArray:domainDown];
                   NSArray *domainUpLemmas = [self
wordLemmasInArray:domainUp];

         [alllexUp addObjectsFromArray:lexUpLemmas];
           [alllexDown addObjectsFromArray:lexDownL
emmas];
          [alllsyns addObjectsFromArray:synsLemmas];
           [allwordRelated addObjectsFromArray:wordR
elatedLemmas];
          [allDomainUpLemmas addObjectsFromArray:
domainUpLemmas];
          [allDomainDownLemmas addObjectsFromArr
ay:domainDownLemmas];

          [glosses appendString:gloss];
       }
       //}
    }
    // NSLog(@"result %@",result);

    //NSLog(@"Tree %@",tmpFamilyMemory);

    NSString *result;


    //--

    if(currentRoot%100==0){
                  NSLog(@"%d connections and so
on...",[tmpFamilyMemory count]);
    }

    currentRoot = currentRoot+1;
    result= [NSString stringWithFormat:@" %@ %@ %@
",
          [alllexDown componentsJoinedByString:@" "],
           [alllsyns componentsJoinedByString:@" "],
         [allwordRelated componentsJoinedByString:@"
"]
           ];
     isMainLoop=NO;

    if([result isAlphanumeric] ){

        [self familyForWords:result];
```

}

}

De este modo se genera un arbol semántico donde todas las palbras del diccionario se acomodaban mediante un ranking valorativo donde las palabras más cercanas a la definición original del concepto se encontraban en niveles jerarquicos inferiores más cercanos a la raíz y posteriormente los conceptos semánticos más distantes

Finalmente se agregaron mejoras al sistema , restringiendo la traza del árbol únicamente a aquellas palabras que comparten la misma categoría léxica que la palabra original.

Posteriormente obteniendo los dos árboles lexico-semánticos de los conceptos polares, resultaba posible valorar en términos de distancia jerarquica, la relación más cercana de cualquier palabra del diccionario entre los dos conceptos polares.

De este modo dados los criterios polares “amor y odio” el programa era capaz de categorizar el concepto “esperanza” más cercano al amor , y el concepto “guerra” más cercano al odio.

Aunque existían ciertos límites de precisón el sistema mostraba una precisión impresionante en el uso del criterio disyuntivo polar.

Este proceso se automatizo mediante el uso del programa Conceptualizer, que permitia entrenar al sistema para crear una valoración única apoyado en la clase DJSalomon haciendo una llamada del tipo

Extracto de main.m en Conceptualizer

```
DJSalomon *salomon = [[DJSalomon alloc] init];
NSDictionary *judge =
[salomon
createJudgementBetweenGood:@”happiness”
andBad:@”sadness”];
```

Este primer paso de la investigación demostró que resultaba posible entrenar un criterio neuronal apartir del entrenamiento lexico y léxico-semántico de dos conceptos polares, sería la fundación de procesos posteriores como se verá posteriormente.

6. Experiencia del mundo real.

Con la finalidad de allegar una cantidad masiva de datos de analisis que permitieran funcionar como materia prima para el estudio y creación del sistema, se delineó que el contenido generado por los usuarios de twitter, se acomodaba perfectamente a las exigencias requeridas por el sistema.

Esta decisión en el sentido de que los mensajes son breves, emiten juicios de valor personales, y adicionlamente que la plataforma de twitter permite el acceso puntualizado a los millones de mensajes que se crean por segundo, permitiendo leerlos atraves de su “streaming API” que funciona a modo de un servicio REST-JSON en el que consume de manera “infinita” un string al que se van concatenando constantemente los ultimos mensajes enviados a la red social.

Adicionalmente y con la visión de la aplicación comercial se determinó que los usuarios de tweeter serían muy suceptibles de asimilar el nuevo concepto de búsqueda y conexión interpersonal propuesto por el sistema.

Así que para este fin se procedio a la creación de un software llamado “TweetLeech”, encargado de conectarse, autenticarse, y leer en manera de loop continuo los mensajes obtenidos mediante el “streaming API” .

El contenido consumido se delimitó al generado en Estadous Unidos y preferentemente generado en idioma inglés para poder ser suceptible de su analisis lexico.

Durante las diferentes etapas de la investigación se crearon diferentes versiones del software “TweetLeech” variando principalmente en la manera y las tablas seleccionadas para almacenar la información la base de datos.

El sistema empleado para la captura de los tweets de entre el stream continuo no requerería de una precisión mandatoría para la captura de todos los mensajes por lo que se permitío la creación de un método que toleraba “perdidas” en la categorización de los tweets.

Es importante decir que en este proceso se

emplearons los frameworks de código abierto ASIHTTP y JSON-FRAMEWORK para C-Objetivo.

El método consiste en reunir bloques de 4096 * 4 bytes , convertir los bytes en strings UTF8 y enviarlos a un parser JSON y después observar si encuentran objetos completos en el diccionario.

Como se puede ver en el siguiente ejemplo:

Extracto de TweetLeech.m

```
-(void) request:(ASIHTTPRequest *)req
 didReceiveData:(NSData*)data {

NSString *received =
[[NSString alloc]
initWithData:data
encoding:NSUTF8StringEncoding];
   [streams appendString:received];
   NSLog(@"received %@",received);

   int size= [streams length];
   if(size >4096*4){
      NSArray *twits = [streams componentsSeparate
dByString:@"\n"];
    // NSLog(@"%d",[twits count]);
      [streams release];
      streams=nil;
      streams=[[NSMutableString alloc] init];
      for( int i=0; i<[twits count]; i ++){
         NSString *item = [twits objectAtIndex:i];
    NSDictionary *twit = [parser objectWithString:item];
         if(twit !=nil){
            //[cage addObject:twit];
            //NSLog(@" cage %d",[cage count]);
            if(delegate!=nil){
                     if([delegate respondsToSelector:@
selector(newTwit:)]){
                  [delegate newTwit:twit];
               }
            }
         }
      }
   }
   [received release];
}
```

Posteriormente los Tweets ya convertidos en objetos abstractos se enviaban a las tablas mySQL correspondientes.

En una primera etapa se realizó la captura de 500,000 mensajes únicos.

6a. Desambiguación del lenguaje preámbulo.
Una vez que se tuvieron disponibles suficientes datos de análisis se procedió a realizar una valoración linguistica de la información para observar la complejidad y metodologías necesarias para poder llevar acabo los procesos de desambiguación morfológica de uso de la lengua y de sentido.

Tras realizar algunos muestreos aleatorios se descubrió rapidamente que los datos ingresados en el sistema mostraban características notables de un pobre apego a las características lexicas , gramáticales y de estructura que se esperan en el lenguaje formal.

Por ejemplo cobró vital importancia la interpretación de emoticonos textuales que representan estados de ánimo, así como las abreviaturas más comunes para estos.

Igualmente resulto de importancia detectar y categorizar la morfología característica del lenguaje de tweetet donde se reconocen los “hashtags” através del símbolo “#” al incio de las palabras y también las citas a otros usuarios con el símbolo “@”.

Finalmente las características estructurales del lenguaje formal como las negaciones, que alteran notablemente el sentido de las palabras.

7. Desambiguación del lenguaje.

La estrategia planteada, se enriquece notablemente a partir de los estudios realizados por los expertos Lexicos detrás de Wordnet y deducible apartir de la visualización de la arquitectura de sus bases de datos y programas incluidos.

Sin embargo Wordnet se encuentra orientado a la desambiguación de palabras “solas” y no a oraciones, o párrafos.

En ese sentido fue necesario un planteamiento más robusto en razón de las estructuras linguisticas que debían ser desambiguadas.

A fin de limitar el alcance del problema se definió

que la estrucutra linguistica más adecuada para transmitir un concepto o idea concreta, es fundamentalmente la oración.

Una oración tiene una gran variedad de posibles variaciones linguisticas en su estructura, sin embargo
es posible delinear sus componentes principales hablando en térmios de “uso de la lengua” como:

a) Sustantivos
b) Verbos
c) Adjetivos
d) Adverbios
e) Enlazadores del lenguaje

Se planteo que si el software pudiera ser capaz de desambiguar el sentido de una oración identificando los siguientes elementos podría ser posible encontrar una interpretación general muy acertada del hecho narrado :

1) Sustantivo: Qué o quien ?
2) Verbo: Que hace o sucede ?
3) Adjetivo: Como lo hace o en que modo ?
4) Adverbio: En que situación o contexto

Sin embargo los primeros estudios realizados en textos extraídos de blogs personales y noticias, demostraron que la expresión común del lenguaje es sumamente compleja y pocas veces se apega a patrones sencillos en la construcción de las oraciones, pudiendo así integrar diversos sustantivos, verbos y adjetivos ubicados en las más diversas formas.

Estos primeros estudios consistieron en aplicar incialmente las mismas reglas lexicas que Wordnet realiza en su desambiguador morfológico.

Las reglas de desambiguación morfológica del idioma inglés determinan las formas más comunes de por ejemplo, “retirar” la conjuación o sufijos de tiempo verbal de un verbo y convertirlo a su raiz o “lemma”.

Sin embargo para complicar más aún la situación, las reglas morfológicas varían dependiendo del uso de la lengua que tiene la palabra, así las reglas morfológicas para los verbos son distintas de los sustantivos o de los adjetivos.

Estas son las reglas morfológicas planteadas por wordnet:

Extracto de DJMorphy.m

```
suffixesNoun = [NSArray arrayWithObjects:
                @"",@"s",
                @"s",@"ses",
                @"x",@"xes",
                @"z",@"zes",
                @"ch",@"ches",
                @"sh",@"shes",
                @"man",@"men",
                @"y",@"ies",
                nil];
  [suffixesNoun retain];
  suffixesVerb = [NSArray arrayWithObjects:
                @"",@"s",
                @"y",@"ies",
                @"e",@"es",
                @"",@"es",
                @"",@"ed",
                @"e",@"ed",
                //@"",@"ning",
                @"",@"ing",
                @"e",@"ing",
                nil];
  [suffixesVerb retain];
  suffixesAdj = [NSArray arrayWithObjects:
                @"",@"er",
                @"",@"est",
                @"e",@"er",
                @"e",@"est",
                nil];


  [suffixesAdj retain];
```

Ello plantea la necesidad idónea de conocer primeramente el uso de la lengua antes de determinar el camino de desambiguación morfológica a emplear.

Sin embargo se optó por crear un método de “fuerza bruta” , similar el empleado en wordnet, donde la palabra es sometida a sus posibles procesos de remoción de sufijos, y una vez obtenido el lemma transformado, este es buscado en los archivos de indices para comprobar si es un lemma válido y definido, o en caso contrario ignorarlo.

Esta etapa del proceso se puede definir brevemente

del modo siguiente.

1. Buscar la palbra en su forma directo en los archivos de indice para los distntos usos de la lengua.
2a. Si existe se almacena en cada uno de los usos de la lengua que es valida.
2b. Si no existe en un archivo de indice de uso de la lengua específico se intenta aplicandole su transformación morfológica correspondiente.
3b. Se vuelve a buscar el lemma obtenido en 2b. en los archivos de indice.
4a. Si se encuentra se almacena de modo igual que en el paso 2a.
4b. Si no se encuentra se intenta buscandolo en los archivos de excepciones de sufijos correspondientes para su uso de la lengua.
5b. Si finalmente no existe se envia a un catalogo de palabras desconocidas.

Con este primer esbozo fue posible realizar una categorizacón linguistica de las palabras pudiendo acceder a los lemmas y por tanto a sus definiciones lexicas en los synsets.

Sin embargo habriamos de enfrentarnos a muchas otras dificultades en el sentido de desambigacion correcta de las oraciones como:

1) El interespaciado de las problemas no es uniforme.
2) La existencia de “palabras grupo” en donde más de una palabra debe mantenerse unida para preservar su sentido como “jesus of nazareth”
3) El interespaciado y uso de puntuación como ?,!.
4) El uso de otros simbolos o caracteres no reconocidos en el lenguaje formal, tales como los emoticonos, que deben ser transformados en las palabras que expresan.
5) Los errores de escritura.
6) La desambiguacion final del uso de la lengua, intentando definir el uso real que tiene cada palabra en la oración.
7) Detectar las formas de negación en las oraciones.

Cada uno de estos problemas fue abordado y resuelto hasta alcanzar niveles de eficiencia en la clase DJMorphy.

La eficiencia de los algoritmos se evaluo primeramente buscando obtener la mayor “claridad” posible buscando obtener en único uso de la lengua para las oraciones analizadas y posteriormente a partir de los resultados de interpretación que arrojaba al integrar el sistema linguistico con el neurologico.

La meta del sistema de desambiguación linguistica consiste en:

1) Identificar unicamente aquellas palabras trascendentes por su uso de la lengua que ayudan a determinar :

a) Sustantivo: Qué o quien ?
b) Verbo: Que hace o sucede ?
c) Adjetivo: Como lo hace o en que modo ?
d) Adverbio: En que situación o contexto

2) Dar una categorizacion especial a los auxiliares de la lengua identificando:

a) Negaciones
b) Puntuación de pausa fuerte
c)Puntuación de pausa suave
d) Preposiciones
e) Negaciones
f) Delimitadores
g) Artículos
h) Pronombres
i) Verbos Auxiliares
j) Conectores
k) Posesivos

3) Como objetivo máximo convertir un texto, en una estructura etiqueta, de lemmas categorizados por usos de la lengua y auxiliares del lenguaje.

El proceso general de desambiguación del lenguaje se puede observar en la siguiente función:

Extracto de DJMorphy.m

```
-(NSMutableArray*) tagTextInMagicWordsArray:(NSString *)text{

    NSDate *date1 = [NSDate date];
   lang = [text languageOfString];
   if(lang==nil)lang=@”en”;

            NSString *worktext= [[NSString alloc] initWithFormat:@”| %@ |”,text];

      NSAutoreleasePool *pool =[[[NSAutoreleasePool alloc] init];
```

```
    worktext =[ worktext lowercaseString];

    // Standarize word separation  " "
            NSMutableArray *textBreaked1= [self
breakTextWithSpaces:worktext];

    // Group words as posible
            NSMutableArray *groupWords = [self
groupWords:textBreaked1];


      // take back to linear text and replace relevant
punctuation
                        worktext =[ groupWords
componentsJoinedByString:@" "];


    worktext = [self standarizeRelevantPunctuationInTe
xt:worktext];

    // break again
            NSMutableArray *textBreaked2= [self
breakTextWithSpaces:worktext];

    // clean symbols
    NSMutableArray *textCleaned =  [self cleanSymbol
sInTextArray:textBreaked2];

    // root / morph / tag
            NSMutableArray *morphWords = [self
rootWords:textCleaned];

    // deambiguation using grammar morphology
             NSMutableArray *clarifyPos = [self
clarifyText:morphWords];

    //nounds collecting

             cachedNouns =[self poses:kTypeNoun
fromText:clarifyPos                max:[NSNumber
numberWithInt:1]];
             cachedVerbs =[self poses:kTypeVerb
fromText:clarifyPos                max:[NSNumber
numberWithInt:1]];
  cachedAdjs =[self poses:kTypeAdj fromText:clarifyPos
max:[NSNumber numberWithInt:1]];

             cachedAllNouns=[self poses:kTypeNoun
fromText:clarifyPos                max:[NSNumber
numberWithInt:100]];
             cachedAllVerbs =[self poses:kTypeVerb
fromText:clarifyPos                max:[NSNumber
numberWithInt:100]];
             cachedAllAdjs =[self poses:kTypeAdj
fromText:clarifyPos                max:[NSNumber
numberWithInt:100]];

                        cachedUnknown = [self
unkwownWordsFrom:clarifyPos];

    //stat viewer
     NSString *result =[self analizedArrayToString:clarify
Pos];
    NSDate *date2 = [NSDate date];
             NSTimeInterval elapsed = [date2
timeIntervalSinceDate:date1];

        //NSLog(@"morphy done in %fs \nresult =
%@",elapsed,result);

    //clean
    [textBreaked1 removeAllObjects];
    [textBreaked1 release];
    textBreaked1=nil;

    [groupWords removeAllObjects];
    [groupWords release];
    groupWords=nil;

    [textBreaked2 removeAllObjects];
    [textBreaked2 release];
    textBreaked2=nil;

    [textCleaned removeAllObjects];
    [textCleaned release];
    textCleaned=nil;


    [morphWords removeAllObjects];
    [morphWords release];
    morphWords=nil;

    [result release];
    result=nil;

    [pool drain];
    return  clarifyPos;

}
```

En términos generales se puede esbozar el sistema en el modo siguiente:

1) Definir el lenguaje del texto

2) Estandarizar el inicio y fin del texto con un espacio.
3) Convertir todo a minusculas.
4) Estandarizar la separación de palabras " "
4) Convertir el texto en un array de palabras
5) Identificar palabras grupales, unirlas con "_"
6) Convertir nuevamente de array en texto plano
7) Usar metodos de busqueda y reemplazo de strings, para identificar todos los auxiliares de la lengua, puntuacion y emoticonos, usando lo siguientes mapeos:

Extracto de DJMorphy.h

```
#define kNegation  @" {¡} "
#define kEmoticon  @" {XD} "
#define kHardBreak @" {.} "
#define kSoftBreak @" {,} "
#define kPreposition @" {p} "
#define kAuxNegation  @" {!} "
#define kDelimiter @" {d} "
#define kArticle @" {t} "
#define kPronouns @" {pn} "
#define kAuxVerb @" {av} "
#define kConector @" {&} "
#define kPosesive @" {my} "

#define kSenseAnger @" {:@} "
#define kSenseDisgust @" {:(} "
#define kSenseFear @" {:S} "
#define kSenseHappiness @" {:)} "
#define kSenseSadness @" {:_(} "
#define kSenseSurprise @" {:o} "
#define kSenseBore @" {:|} "
#define kTypeAny @" {any} "
#define kTypeNothing @" {-} "
```

8) Separar nuevamente el texto por " " y convertirlo en un array
9) Iterar por el array y remover cualquier otro simbolo o caracter no reconocido anteriormente.
10) Buscar todos los lemmas posibles para cada palabra y categorizar su uso de la lengua en este punto el array se convierte en un array de "arrays" donde cada palabra puede contener 1 o multiples usos de la lengua.
11) Desambiguar los resultados con más de un uso de la lengua a partir de morfologia gramatical
12) Devolver el array conteniendo los resultados analizados.

En este sentido resulta interesante hacer una pausa para analizar más a fondo lo descrito en el paso 11).

A fin de mejorar la desamgiuación del sentido para una palabra con múltiples sentidos posibles, se optó por crear un sistema básico que en base al análsisi de las palabras vecinas y sus funciones gramaticales ayudara a hacer una valoración acumulativa, para determinar cual podría ser el uso más común para esa estructura de palabras.

En este sentido se creo un array que describe la estructura de las reglas gramticales más comunes, conciderando la palabra a desambiguar al centro y sus dos vecinos circundantes.

```
morphRules =[NSArray arrayWithObjects:

        kTypeAny,kTypeVerb,kTypeNoun,
        kTypeNoun,kTypeVerb,kTypeAny,
        kAuxVerb,kTypeVerb,kTypeAny,

        kAuxVerb,kTypeVerb,kTypeAny,
        kPronouns,kTypeVerb,kPreposition,
        kPreposition,kTypeVerb,kArticle,
        kTypeAdv,kTypeVerb,kArticle,

        kArticle,kTypeNoun,kTypeAny,
        kPosesive,kTypeNoun,kTypeAny,
        kArticle,kTypeNoun,kAuxVerb,
        kArticle,kTypeNoun,kHardBreak,
        kArticle,kTypeNoun,kSoftBreak,
        kPreposition,kTypeNoun,kHardBreak,
        kPreposition,kTypeNoun,kSoftBreak,
        kTypeVerb,kTypeNoun,kTypeAny,

        kPosesive,kTypeAdj,kTypeAny,
        //kPronouns,kTypeAdj,kTypeAny,
        kArticle,kTypeAdj,kTypeAny,
        kAuxVerb,kTypeAdj,kTypeAny,

        kTypeAdj,kTypeAdv,kTypeAny,

        nil];
  [morphRules retain];
```

Luego cada palabra es sometida a una valoracón de estas reglas y obtiene puntos si coincide con cierto patrón gramatical para un uso de la lengua específico.

Al final se selecciona el uso de la lengua con mayor puntaje, y aquellos que empaten con el.

Aquí se muestran los métodos principales para lograr este proceso:

Extracto de DHMorphy.m

```
-(NSMutableArray *)clarifyText:(NSMutableArray *)text{

    if(cachedAsserts !=nil){
        [cachedAsserts removeAllObjects];
        [cachedAsserts release];
        cachedAsserts=nil;
    }
    textPosClarified=text;
    while ([self clarifyTextLoop]) {
        // NSLog(@"clarifyTextLoop");
    }

    return textPosClarified;
}

-(BOOL ) clarifyTextLoop{
    // NSLog(@"analyzing: %@",textPosClarified);

       NSAutoreleasePool *pool =[[NSAutoreleasePool alloc] init];
     NSMutableArray *result = [[NSMutableArray alloc] init];
    NSMutableArray *globalAsserts =[[NSMutableArray alloc] init];

    for( int i=0; i<[textPosClarified count]; i++){

        NSMutableArray *posesLeft =nil;
        NSMutableArray *posesRight =nil;
        NSString *lemmaLeft =kTypeAny;
        NSString *lemmaRight =kTypeAny;

            NSMutableArray *poses = [textPosClarified objectAtIndex:i];

        if((i-1)>=0){
          posesLeft= [textPosClarified  objectAtIndex:i-1];
            if([posesLeft count]==1){
                lemmaLeft = [posesLeft objectAtIndex:0];
                if([lemmaLeft isJSONstring]){
            lemmaLeft=[[NSString alloc] initWithFormat:@"%@ ",lemmaLeft];
                } else{
                    lemmaLeft=[lemmaLeft pos];
                }
            }
        } else{
            lemmaLeft =kTypeNothing;
        }

        if(i+1<[textPosClarified count]){
        posesRight= [textPosClarified objectAtIndex:i+1];
            if([posesRight count]==1){
                lemmaRight = [posesRight objectAtIndex:0];
                 if([lemmaRight isJSONstring]){
              lemmaRight=[[NSString alloc] initWithFormat:@"%@ ",lemmaRight];
                } else{
                    lemmaRight=[lemmaRight pos];
                }

            }
        }else{
            lemmaRight =kTypeNothing;
        }



         NSMutableArray *allAsserts =[[NSMutableArray alloc]init];

        for( int j=0; j<[poses count]; j++){

            int asserts=0;

            NSString *lemma = [poses objectAtIndex:j];

            if([poses count] >1){
                if(![lemma isJSONstring]){

                    //NSLog(@"analyzing %@",lemma);

                    NSString *lemmaPos=[lemma pos];

                    for( int g=0; g<[morphRules count]; ){
                            NSString *leftRule=[morphRules objectAtIndex:g];
                            NSString *lemmaRule=[morphRules objectAtIndex:g+1];
                            NSString *rightRule=[morphRules objectAtIndex:g+2];
                        g=g+3;

                        // NSLog(@"- %@ -",lemma);
                           //NSLog(@"leftRule %@ lemmaRule %@ lemmaRight %@ ",leftRule,lemmaRule,rightRule);
                        //NSLog(@"lemmaLeft %@ lemmaPos %@ lemmaRight %@ ",lemmaLeft,lemmaPos,lemmaRight);
```

```
                    if ([leftRule compare:kTypeAny] ==NSOrderedSame) {
                        leftRule=lemmaLeft;
                    }
                     if ([lemmaRule compare:kTypeAny] ==NSOrderedSame) {
                        lemmaRule=lemmaPos;
                    }
                       if ([rightRule compare:kTypeAny] ==NSOrderedSame) {
                        rightRule=lemmaRight;
                    }

                    // NSLog(@"%d if %@ %@ %@ ",g,leftRule,lemmaRule,rightRule);
                    // NSLog(@"%d if %@ %@ %@ ",g,lemmaLeft,lemmaPos,lemmaRight);


                     if ([leftRule compare:lemmaLeft] == NSOrderedSame &&
                        [lemmaRule compare:lemmaPos] == NSOrderedSame &&
                        [rightRule compare:lemmaRight] == NSOrderedSame)
                    {

                        //NSLog(@"OK %@ %@ %@ ",leftRule,lemmaRule,lemmaRight);
                        // NSLog(@"OK %@ %@ %@ ",lemmaLeft,lemmaPos,lemmaRight);

                        asserts=asserts+1;
                    }
                }


            }
        }
                     [allAsserts addObject:[NSNumber numberWithInt:asserts]];

    }


    //NSLog(@"allAsserts %@",allAsserts);
    int max=-1;

    for(int k=0;k <[allAsserts count]; k++){
            NSNumber *currentAssert= [allAsserts objectAtIndex:k];
            int c =[currentAssert intValue];
            if(c>max)max=c;
        }

        NSMutableArray *currentSloc=[[NSMutableArray alloc] init];
        for( int j=0; j<[poses count]; j++){
                NSNumber *currentAssert= [allAsserts objectAtIndex:j];
            NSNumber *lemma= [poses objectAtIndex:j];

            if( [currentAssert intValue] == max ){
                [currentSloc addObject:lemma];
            }

        }

        [result addObject:currentSloc];
        [globalAsserts addObject:allAsserts];
    }

    textPosClarified =result;

    newAsserts=globalAsserts;

    if(cachedAsserts ==nil){
            cachedAsserts = [[NSMutableArray alloc] initWithArray:newAsserts copyItems:YES];
        [newAsserts release];
        newAsserts=nil;
        return YES;
    } else{

       //NSLog(@"newAsserts %d",[newAsserts count]);
        //NSLog(@"cachedAsserts %d",[cachedAsserts count]);
        int differences=0;

        for(int a=0; a<[newAsserts count]; a++){
                NSArray *oldAsserts = [newAsserts objectAtIndex:a];
            NSArray *newAsserts = [cachedAsserts objectAtIndex:a];

            for(int b=0; b<[oldAsserts count]; b++){

            NSNumber *old=[oldAsserts objectAtIndex:b];
          NSNumber *new=[newAsserts objectAtIndex:b];
            // NSLog(@"[old intValue]=%d",[old intValue]);
                // NSLog(@"[new intValue]=%d",[new intValue]);
```

```
            if([old intValue] != [new intValue] ){

                differences=differences+1;
            }
        }
    }
    // NSLog(@"differences=%d",differences);
    if(differences==0){
        return NO;
    } else{
            cachedAsserts = [[NSMutableArray alloc]
initWithArray:newAsserts copyItems:YES];
        [newAsserts release];
        newAsserts=nil;
        return YES;
    }
  }

  [pool drain];

}
```

Finalmente se muestra un ejemplo de como funciona el etiquetador de texto en una oración concreta:

@n - Noun- Sustantivo
@v - Verb - Verbo
@a - Adjective - Adjetivo
@r - Adverb - Adverbio

Entonces Para el texto

"it´s amazing to create a revolution on the internet. feeling really happy"

Se obtiene:

```
textMorphed=[
["{my}"],           // Posesivo
["amazing@a"], // @a adjetivo
["{p}"],            // Preposicion
["create@v"],    //@v verbo
["{t}"],            // Articulo
["revolution@n"],  //Sustantivo
["{p}"],            // Preposición
["{t}"],            // Articulo
["internet@n"],  // @n Sustantivo
["{.}"],            //  Puntuación pausa fuerte
["feeling@n","feel@v"],  // @n sustantivo o @v verbo
["really@r"],      //@r Adbervio
["happy@a"]]   // @a Adjetivo
```

Resumen :

Conceptos Claros:
lastNounsTag=["revolution@n","internet@n"]
lastAdjsTag =["amazing@a","happy@a"]
lastVerbsTag=["create@v"]

Todos los conceptos:
lastAllNounsTag=
["revolution@n","internet@n","feeling@n"]
lastAllAdjsTag=["amazing@a","happy@a"]
lastAllVerbsTag=["create@v","feel@v"]

Para el texto

"today is not going to rain XD, what a shame ! "

Se Obtiene:

```
textMorphed=[
["today@r","today@n"],
["{!}"],
["going@a","going@n","go@v"],
["{p}"],
["rain@v","rain@n"],
["happy@a"],
["{,}"],
["{d}"],
["{t}"],
["shame@n"],
["{.}"]]
```

Interpretación

Conceptos Claros:
lastNounsTag=["shame@n"]
lastAdjsTag=["happy@a"]
lastVerbsTag=[]

Todos los conceptos:
lastAllNounsTag=
["today@n","going@n","rain@n","shame@n"]
lastAllAdjsTag=["going@a","happy@a"]
lastAllVerbsTag=["go@v","rain@v"]

Aun cuando el presente sistema permitió continuar con la investigación, se anticipa la necesidad de mejorar el proceso de desambiguación de uso de la lengua y sentido, mediante la implementación de un sistema de valoración recursiva similar al descrito anteriormente en la clarificación con base gramatical, pero que considere en lugar de reglas,

un diccionario de patrones de palabras vecinas con las que más comunmenente aparece una palabra en un uso de la lengua específico.

8. Creación dinámica del corpus léxico y sus relaciones de frecuencia.

Una vez experimetado con los límites y alcances de las técnicas de etiquetado y reconocimiento linguistico, surgieron una serie de hipótesis respecto a como mejorar el sistema, y como definir mejor los límites completos del sistema neurolinguistiico.

A fin de obtener más información se decidió identificar mediante el analisis de la base de datos de tweets, que en su tabla correspondiente contenia alrededor de 500,000 mensajes, capturados entre agosto y octubre de 2011, aquellas palabras de mayor uso, a fin de identificar el "corpus" lexico objeto de nuestro analisis.

De esta manera, se procedió a crea una serie de rutinas directamente como "Stored Procedures" de MySQL a fin de obtener esta información.

A fin de hacer este proceso más rápido la tabla que originalmente se encontraba en formato INNODB , se clono al formato MyISAM, a fin de permitir el indexado y búsqueda usando operadores de Búsqueda de Texto que permiten indexar campos tipo Text para optimizar la velocidad de búsqueda.

Una vez identificado el corpus del lenguaje , el siguiente paso era observar como estas palabras se relacionan entre sí contabilizando el número de apariciones de cada uno de las palabras del corpus en relación a todas las demás y determinar de esta manera como una palaba del corpus, se usa con más frecuencia en relación a otras.

Para estos fines se delineo el siguiente proceso:

1) Crear un indice con las palabras que más aparecen en el sistema, para este primer fin, se consideraron todas aquellas palabras que aparecieran más de 100 veces de entre el medio millon de mensajes o .005% de la muestra, y crear este modo el corpus del lenguaje.

2) Crear una tabla donde se relacione cada palabra conformante del corpus con el resto, y contabilizar el número de apariciones que tiene una en relación a la otra.

Para acometer con el primer objetivo se escribieron 2 Procedures, uno para contabilzar y otro como bucle de iteracción.

MySQL procedure countWebInputLabs

```
PROCEDURE `countWebInputLabs`
(IN lemma VARCHAR(100), OUT count INT)
BEGIN
  SELECT COUNT(*)
  FROM `WEBInputsLab`
   WHERE
        MATCH(WEBInputsLab.inputText)   AGAINST
(lemma IN BOOLEAN MODE) INTO count;
END
```

MySQL procedure doTAGFrequencyIndex

```
PROCEDURE `doTAGFrequencyIndex`()
BEGIN
DECLARE _lemma CHAR(100);
DECLARE keysearch CHAR(100);

DECLARE preCount INT;

DECLARE no_more_rows BOOLEAN;

DECLARE cursor1 CURSOR FOR

SELECT DISTINCT BAIndexes.lemma FROM
BAIndexes;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET no_more_rows =FALSE;

OPEN cursor1;

LOOP1: loop

    fetch cursor1 into _lemma;

    if no_more_rows then

       close cursor1;

       leave LOOP1;

    end if;
```

```
        if not (_lemma REGEXP '[0-9]') then

          SET _lemma = REPLACE(_lemma,"_"," ");

          SET _lemma = REPLACE(_lemma,"-"," ");

          SET _lemma = REPLACE(_lemma,"'","");




          SET keysearch = CONCAT('+" ',_lemma);

          SET keysearch = CONCAT(keysearch,' "');




         call countWebInputLabs(keysearch,preCount);




         if preCount>100  THEN

                    INSERT INTO TAGFrequencyIndex
(lemma,count)
              VALUES

              (_lemma ,preCount)

                 ON DUPLICATE KEY UPDATE lemma =
VALUES(lemma);

          end if;

        end if;

    end loop LOOP1;

end
```

De este análisis se obtuvo primero la tabla TAGFrequencyIndex , que enumeraba un corpus de 2,262 palabras de uso más frecuente en la muestra.

Aqui un ejemplo de los primeros resultados ordenados por número de apariciones:

Extracto de TAGFrequencyIndex Tabla:

id	lemma	count
987	http	82195
1212	love	12341
853	good	10277
502	day	8916
137	back	8484
2044	time	8114
507	de	7545
2056	today	7544
763	follow	6793
1494	people	6402
1775	shit	5544
1239	make	5460
2107	twitter	5037
1407	night	5020
1728	school	4725
1172	life	4493
810	ga	4492
913	happy	4338
2221	work	4316
800	fuck	4230
1251	man	4180
45	aku	4153
18	ada	3977
1351	morning	3960

Posteriormente a partir de este indice se uso el Stored Procedure llamado doTAGWordFrequencyPro para calcular la relación que existia entre cada uno de los términos del conjunto.

MySQL Procedure doTAGWordFrequencyPro

```
PROCEDURE `doTAGWordFrequencyPro`()
BEGIN

DECLARE lemma CHAR(100);
DECLARE related CHAR(100);

DECLARE lemmaclean CHAR(100);
DECLARE relatedclean CHAR(100);

DECLARE lemmaCount CHAR(100);
DECLARE relatedCount CHAR(100);

DECLARE keysearch CHAR(200);

DECLARE magickey CHAR(200);

DECLARE preCount INT;

DECLARE weight FLOAT;

DECLARE _row INT;

DECLARE _col INT;
```

```
DECLARE no_more_rows INT;

DECLARE cursor1 CURSOR FOR
SELECT  TAGFrequencyIndex.id,TAGFrequencyIndex.
lemma,TAGFrequencyIndex.count FROM
TAGFrequencyIndex;

DECLARE cursor2 CURSOR FOR

SELECT  TAGFrequencyIndex.id,TAGFrequencyIndex.
lemma,TAGFrequencyIndex.count FROM
TAGFrequencyIndex;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET no_more_rows =1;

SET no_more_rows =0;

OPEN cursor1;

LOOP1: loop

    fetch cursor1 into _col,lemma,lemmaCount;

  if no_more_rows=1 then

    leave LOOP1;

  end if;

   OPEN cursor2;

    LOOP2: loop

        fetch cursor2 into _row,related,relatedCount;

      INSERT INTO TAGFrequencyStatus (id,col,row)
VALUES(1,_col,_row) ON DUPLICATE KEY UPDATE
col= VALUES(col), row=VALUES(row);

      if no_more_rows=1 then

        leave LOOP2;

      end if;


        SET lemmaclean = CONCAT(' +" ',lemma);

       SET lemmaclean = CONCAT(lemmaclean,' " ');

        SET relatedclean = CONCAT(' +" ',related);

       SET relatedclean = CONCAT(relatedclean,' " ');

         SET keysearch = CONCAT(lemmaclean,relat
edclean);

        SET magickey = CONCAT(lemma,'@');

        SET magickey = CONCAT(magickey,related);

      call countWebInputLabs(keysearch,preCount);

        if preCount>0 THEN

          SET weight = preCount/ ((lemmaCount +
relatedCount) -preCount);

         INSERT INTO TAGFrequencyPro (weight, ma
gickey,lemma,related,keysearch,count)
          VALUES

          (weight, magickey,lemma,related,keysearch,
preCount)

            ON DUPLICATE KEY UPDATE count =
VALUES(count),weight = VALUES(weight);

          end if;

      end loop LOOP2;

     close cursor2;

     set no_more_rows =0;

 end loop LOOP1;

 close cursor1;

end
```

De este análisis se obtuvo la tabla TAGFrequencyPro la cual muestra la relación entre cada una de las palabras del corpus y el resto del corpus.
Aquí un ejemplo de los valores en esta tabla organizados por mayor número de apariciones usando el query : *(ver tabla 1)*

Adicionalmente en este paso se calcula el valor del “peso” que tiene esa relación para el universo

*tabla 1. Select * from TAGFrequencyPro where lemma='money' order by count DESC LIMIT 20*

id	weight	magickey	lemma	related	count	keysearch
723228	1	money@money		money money	1937	+" money " +" money "
723026	0.00615903	money@http	money	http	515	+" money " +" http "
723167	0.0315158	money@make	money	make	226	+" money " +" make "
723449	0.0421348	money@save	money	save	105	+" money " +" save "
723283	0.0320672	money@online	money	online	103	+" money " +" online "
723151	0.00556377	money@love	money	love	79	+" money " +" love "
723645	0.00741706	money@time	money	time	74	+" money " +" time "
723173	0.0209677	money@making		money making	65	+" money " +" making "
722527	0.00569388	money@back	money	back	59	+" money " +" back "
722613	0.0194463	money@buy	money	buy	59	+" money " +" buy "
723542	0.0239787	money@spend	money	spend	54	+" money " +" spend "
723306	0.00591074	money@people		money people	49	+" money " +" people "
723762	0.00789813	money@work	money	work	49	+" money " +" work "
722937	0.00977756	money@give	money	give	40	+" money " +" give "
722948	0.00328569	money@good	money	good	40	+" money " +" good "
723301	0.0150713	money@pay	money	pay	37	+" money " +" pay "
723012	0.006157	money@home	money	home	36	+" money " +" home "
723395	0.00811725	money@real	money	real	36	+" money " +" real "
722915	0.00554378	money@fuck	money	fuck	34	+" money " +" fuck "
722742	0.00304991	money@day	money	day	33	+" money " +" day "

total de apariciones que independientemente tienen cada una de las palabras, esto es :

weight = preCount/
((lemmaCount + relatedCount) -preCount);

Donde

lemmaCount: es el total de apariciones de manera independiente del término en la muestra.
relatedCount: es el total de apariciones de manera independiente del término en la muestra.
preCount: es el total de apariciones de manera independiente de la relación de términos en la muestra.

Este valor llamado "weight" o peso será de gran importancia posteriormente al realizar los calculos de la respuesta neuronal, ya que representa el valor de la relación entre estos dos términos dentro de la muestra, en una escala flotante de 0 a 1.

Como se puede observar, esta es una aproximación de alto rendimiento ya que se ejecuta directamente en MySQL, para calcular de manera dinámica el corpus léxico del lenguaje .

Una gran ventaja derivada es que sería posible correr de manera automatizada los procesos de indexamiento de frecuencia y relaciones, descritos anteiormente, con periodicidad para mantener el corpus léxico actualizado en relación al más reciente uso del lenguaje.

Sin embargo representa una importante descentaja en relación a la actualización din´mica, en el sentido de que al cambiar las valoraciones de "weight" o peso para las relaciones de los lemmas, exigiría la actualización de cualquier otro calcula basado en este valor (como lo es el cálculo de la respuesta neuronal).

Adicionalmente se tiene la debilidad de que el indexamiento de estos indice de frecuencia esta limitado a aquellas palabras que existen en el índice de lemmas originalmente extraídos desde Wordnet, y aunque el sistema muestra un mecanismo para indentificar aquellas palabras "desconocidas" pero que se usan con mayor frecuencia, requeriría de un categorizador humano, que vinculará la relación de este nuevo lemma, con algún synset existente, o crear una definición completamente novedosa.

9. Paralelismo Semántico

Retomando el tema planteado en el punto "5. Árboles léxicos" , se resume la posibilidad de

programar y entrenar neuronas “disyuntivas” que apartir de valorar la distancia de un concepto de búsqueda en dos árboles léxicos polares, resultaba capaz de determinar la mayor similitud en términos léxicos de ese concepto de búsqueda determinando así la posible filiación más cercano a determinado concepto polar.

De este modo resulta posible determinar por ejemplo que la palabra “paz” tiene una relación léxica más cercana al concepto “amor” que al concepto “guerra”.

Sin embargo esta aproximación presenta problemas al intentar evaluar conceptos más complejos de naturaleza contradictoría como la frase “odio la paz” o la frase “amo la guerra”. En ambos casos la valoración de la neurona disyuntiva antes citada devolvería un valor cercano a 0, al anularse mutuamente los valores polares.

Como una solución a esta observación surge el “paralelismo semántico”. En este planteamiento la valoración del concepto se realiza de manera paralela y simultanea a lo largo de un arreglo de N árboles léxicos , representados como neuronas léxico conceptuales, y el valor único devuelto al estímulo único de cada neurona , forma parte integra de la respuesta del impulso, siendo así el resultado, un arreglo paralelo y simultaneo de valores que forman un Hash o combinación única en un espacio N-dimensional.

9a. Neuronas Léxico Conceptuales

Para aclarar más este concepto, entraremos en detalle en la definición de neurona léxico conceptual.

Una neurona léxico conceptual, es en términos computacionales un arreglo de lemmas creado apartir de la traza del árbol léxico del concepto raíz.

Así pues se puede crear una neurona léxico conceptual para definir el concepto “amor”. Como se describe en el punto “5. Árboles semánticos” el árbol léxico de esta palabra se trazaría apartir de las relaciones hipónimas y similares al concepto original, creando un nuevo nivel jerárquico o relación padre-hijo entre el concepto “amor” y sus hipónimos y similes. A continuación se llevaría a cabo un ciclo recursivo con estos hijos par crear el siguiente nivel jerarquico. El proceso se repetiría así siendo limitado por 2 criterios.

1) Un lemma ya estudiado no puede volverse a analizar.

2) Solo se admiten aquellos hijos que compartan la misma categoría léxica como la definida por wordnet para la palabra raiz.

En caso de que no existiera la segunda limitante, se comprobó que el proceso continúa hasta “conectar” en diversos niveles jerarquicos con todas las palabras que conforman el diccionario. Comportamiento interesante, ya que a partir de tomar como punto de origen distinto conceptos raíz se delinea un árbol léxico que abarca todas las palabras pero en diferente momento y jerarquía.

Sin embargo se comprobó que debido a las variaciones de sentido y uso de la lengua, que actualmente se encuentran apenas en los márgenes de eficacia descritos en el apartado de morfología linguistica, no era posible realizar un vinculación jerarquica de manera eficiente através de todas las palabras del diccionario.

De este modo con la restircción de categoría lexica, se observa la creación de árboles léxicos de tamaño variable y muy inferior al árbol máximo, limitados por mucho en el sentido de la calidad de definición y riqueza multi funcional de los términos hijos del concepto raíz.

Adicionalmente por motivos de intensidad de cálculo se determino que el número máximo de anidamientos jerarquicos fuera 7, ya que de otra manera el tiempo de calculo se elvaba de manera significativamente escapando de los alcances de la investigación.

Más adelante se proponen las técnicas que enriquecerían estas limitantes através de la experiencia adquirida del analisis de las muestras de lenguaje del mundo real.

Así pues una neurona léxico conceptual, es una tabla que contiene la distancia entre el término inicial y los demás lemmas trazados alamacenando el valor de su posición jerarquica o de anidamiento en la creación de su árbol léxico.

9b. Conceptos complejos

Así pues resulta posible hablar de las neuronas léxico

conceptuales, (en adelante solamente “neuronas”), como neuronas que se entrenan a partir de la definición léxica de un lemma determinado.

Sin embargo para avanzar en el proceso de la definición de conceptos más complejos se vuelve necesarió ampliar el concepto reintegrando el antes citado paralelismo semántico.

Planteemos una vez el supuesto de valorar en términos humanos si la palabra “paz” se vincula más con el concepto “felicidad” o a “tristeza” , con lo que se ha dicho antes, se podría suponer en entrenamiento de dos neuronas léxico conceptuales, para valorar los niveles de “exitación” que alcanza enc ada una de ellas el concepto “paz”.

Aquí haremos una pausa para definir el concepto de “exitación” o “respuesta neuronal” y sus similes, para referirnos a la valoración de la distancia del concepto raiz al concepto de búsqueda. En este sentido la escala de medición se normaliza a un rango de 0 a 1 tomando como máximo , el número mayor de niveles jerarquicos que puede alcanzar un árbol léxico semántico, que como ya se ha dicho antes en este sistema es 7.

Así pues somentiendo nuestro experimento hipotético al prototipo real obtenemos los siguientes valores:

peace@n = 0.477101 happy
peace@n = 0.340148 sad

Podemos ver como el valor de exitación es mayor en el árbol léxico “happy” que en el árbol léxico “sad” y sin embargo las relaciones no son demasiado distantes entre sí.

Así pues aun cuando “peace” es un concepto complejo y que no esta sujeto a una valoración polar sencilla nisquiera por un categorizador humano, podemos ver como este análisis semantico simultaneo desde la perspectiva de los conceptos “happy” o “peace” nos arroja una percepción valorativa muy similar a lo que un categorizador humano podría decir en el sentido de que la paz no es la alegría ni la tristeza nisquiera en un 50%, pero se inclina ligeramente a estar más relacionado ala felicidad.

Cabe señalar que el prototipo final con el que se realizó esta medición contiene una serie de optimizaciones que se discutirán más adelante , pero son suficientes para demostrar el concepto de “exitación” neuronal.

De este modo es como se plantea que la definición de conceptos complejos através de neuronas léxico conceptuales se vuelve posible apartir del uso de arreglos de disntitas neuronas aportando diferentes perspectivas o formas de valorar el concepto de búsqueda, y que la interpretación del concepto complejo detrás de este término, se vuelve descriptivo através de la relación simulatnea y paralela de la exitación individual de las neuronas conformantes del arreglo.

Así pues se puede plantear la creación de redes neuronales o “cerebros” conformados por conjuntos de neuronas léxico conceptuales, que al exitarse de manera paralela y simultánea generan un conjunto de “exitación” único y característico, que representa en el universo de ese conjunto de neuronas, un hash o representación única de la respuesta de ese “cerebro” respecto a un determinado criterio de búsqueda.

Haciendo remembranza a la comparación con la respuesta cerebral humana a estimulos reales y su medición eléctrica através de sensores.

Este tema se abordará más adelante al momento de hablar del concepto de “tag de pensamiento” o “thought tag”.

10. Definición del sentido común

Dado el planteamineto anterior y retomando el objetivo central del sistema, se vislumbra un camino técnico posible para la creación de un sistema experto capaz de evaluar juicios del modo que el “sentido común” lo hace.

Según lo dicho en el punto “9. Conceptos Complejos” para definir el “sentido comun” como un concepto y preparar nuestro sistema para realizar una valoración en esa forma, es necesario primeramente determinar que “conceptos llave” habran de conformar nuestra red neuronal simultanea y paralela (o en adelante “cerebro”) capaz de procesar en juicio del “sentido común”.

Como primer paso se procedió al estudio de los planteamientos conceptuales que definen al “sentido común”. Tal como se discutió incialmente el sentido común se repreenta por un conjunto de juicios valorativos socialmente aceptados en relación a las más amplias esferas que rodean la vida humana.

Así pués debia realizarse un planteamiento que fuera capaz de evaluar en relación a las distintas esferas de la vida humana tales como la escuela, trabajo, diversión, sufrimiento, vida, muerte .

Asimismo estas debían representarse por un lemma existente tanto en el catálogo de índices del diccionario como en las tablas del corpus de lenguake (o mayor frecuencia de uso).

Los lemmas propuestos para este cerebro fueron 44 y son los siguientes:

id	*brain*	*neuron*
1	*1*	*love@*
2	*1*	*hate@*
3	*1*	*happy@*
4	*1*	*sad@*
5	*1*	*honest@*
6	*1*	*lie@*
7	*1*	*success@*
8	*1*	*fail@*
9	*1*	*beautiful@*
10	*1*	*ugly@*
11	*1*	*health@*
12	*1*	*sick@*
13	*1*	*sex@*
14	*1*	*sleep@*
15	*1*	*fun@*
16	*1*	*work@*
17	*1*	*party@*
18	*1*	*school@*
19	*1*	*young@*
20	*1*	*tired@*
21	*1*	*dream@*
22	*1*	*real@*
23	*1*	*friend@*
24	*1*	*family@*
25	*1*	*girl@*
26	*1*	*boy@*
27	*1*	*hope@*
28	*1*	*lost@*
29	*1*	*gym@*
30	*1*	*hungry@*
31	*1*	*spirit@*
32	*1*	*money@*
33	*1*	*night@*
34	*1*	*day@*
35	*1*	*gay@*
36	*1*	*straight@*
37	*1*	*rich@*
38	*1*	*poor@*
39	*1*	*pleasure@*
40	*1*	*pain@*
41	*1*	*care@*
42	*1*	*hurt@*
43	*1*	*live@*
44	*1*	*dead@*

Se puede observar también que en la selección de estos términos se busco la inclusión de t´èrminos polares y que se organizarón de manera ordenada mostando con un indice impar el termino positivo con un indice par el término positivo.

Este ordenamiento cobrará relevancia cuando se discutan las técnicas de búsqueda e indexamiento.

El proceso de los lemmas fue sujeto al capricho de un categorizador humano, y su criterio previamente educado en el tema, sin embargo es motivo de estudio la mejor manera para definir los términos raices empleados para la constitución de un sistema neuronal paralelo y simultaneo.

11. Recapitulación y Metodología Para el Entrenamiento Neuronal

Vale la pena hacer una remembranza de lo planteado en términos de la arquitectura de nuestro sistema neuronal.

Como un preambulo se discutieron las técnicas de desambiguación linguistica que permiten a partir de un texto cualquiera que representa una idea abstracta, extraer una valoración léxica vinculada a las definiciones contenidas en un diciconario léxico

Luego se plantea y comprueba que es posible crear estructuras de código bajo el modelo de neuronas, que aprendan las relaciones léxicas del lenguaje a partir de la alimentación de datos, también desde un diccionario léxico semántico.

Liego se ha planteado que es posible crear una red

neuronal de N-dimeniones donde la medición de los impulsos de manera simulanea y paralela puede conllevar a la valoración de juicios sobre conceptos complejos, de modo tal que podrían considerarse cerebros, capaces de emitir juicios sobre conceptos linguisticos complejos.

Y ahora es momento de discutir la manera exacta en que este entrenamiento o aprendizaje toma lugar en nuestro sistema.

Retomando lo referente al punto “9. Conceptos Complejos” y al punto 5. “Árboles Léxicos” donde se discute la manera en que se puede almacenar en una neurona, la jerarquía representativa de un árbol léxico a partir de un lemma o concepto raíz, es importante retomar una limitante mencionada en el sentido de que los árboles léxico limitados mediante el filtrado de aquellos hijos que comparten la categoría o categorías léxicas del concepto raíz, derivaba en el cálculo de árboles léxicos truncos que no vinculan de manera completa todos los lemmas del lenguaje.

Para putualizar este debilidad, debemos plantear que tras la generación completa de un árbol léxico para un concepto X, es muy fáctible descubrir que un concepto Y cualquiera no se va a encontrar en la definición de su árbol léxico.

A fin de contrarestar estas y otras limitantes en la generación de los árboles léxicos , se realizó un planteamiento en el que através de la defincíión de conceptos similares que ampliaran la definición del concepto raiz, resultaría posible abarcar un maypr porcentaje del universo total de lemmas dene l lenguaje.

Así pues en un primer planteamiento se intento de manera discriminatoría por un categorizador humano, amplia la definición del concepto raíz por medio de agregar en la propia raíz otros conceptos sinónimos o relaciones que pudieran ser comptaibles.

Así por ejemplo al intentar definir el “amor” se paso más bien a intentar definir los conceptos rectores de un concept superior como el de las “emociones positivas” para crear así una familai de raices como “amor, compasión, amistad, alegría, etc. “

Sin embargo y como en todos los casos donde el criterio es discriminado por un categorizador humano único, se determinó que la selección de los términos se vería viciada por la experiencia personald el categorizador.

Fue así como se recurrió a los estudios realizados en el apartado “8. Creación dinámcia del corpus léxico y sus relaciones de frecuencia” para que mediante los resultados arrojados por las relaciones de frecuencia de aparición se seleccionaran aquellos términos linguisticamente más relacionados con el concepto raíz, planteando así un método automatizado y medible para ampliar los conceptos llave.

Así pues un primer sampleo por ejemplo para la palabra “love” nos devuelve los siguientes resultados que organizados por la relación “weight” descrita anteiormente nos devuelve estos resultados:

Select distinct magickey,count,weight from TAGFrequencyPro where lemma='love' order by weight DESC LIMIT 20

magickey	*count*	*weight*
love@love	*12341*	*1*
love@in love	*502*	*0.0406774*
love@life	*332*	*0.0201188*
love@song	*279*	*0.0200072*
love@people	*324*	*0.0175905*
love@happy	*254*	*0.0154642*
love@hate	*239*	*0.0152872*
love@day	*318*	*0.015187*
love@baby	*184*	*0.0129989*
love@girl	*198*	*0.0129972*
love@make	*224*	*0.0127439*
love@good	*284*	*0.012716*
love@show	*184*	*0.0126608*
love@god	*185*	*0.0124094*
love@heart	*167*	*0.0122147*
love@follow	*230*	*0.0121667*
love@time	*229*	*0.0113221*
love@back	*229*	*0.0111187*
love@birthday	*154*	*0.0106885*
love@http	*957*	*0.0102267*

De este modo se puede observar como la palabra “love” aparece con mayor frecuencia en el sentido “in love” , pero seguido también por life,song,peo pe,happy,hate,day,baby y otros, Estas relaciones muestran un coherente sentido en relación a la valoración de juicio comun de un categorizador humano y se convierten en una fuente de una gran

calidad para determinar los conceptos similes que habran de definir el concepto raíz.

Tambien es interesante notar la aparición del lemma “hate” que es un concepto polar a “love”.

En este sentido quedó demostrado que en el hablar común es muy frecuente expresarse en una misma idea empleando conceptos polares.

En esta conciencia como se verá más adelante se crearon métodos para excluir las relaciones polares durante la creación de los árboles léxicos enriquecidos por la información de frecuencia de uso, (en adelante “arboles léxico semánticos”).

11b. Entrenamiento Linguistico Total Basado en el Corpus Léxico

Durante la investigación se determino que a fin de acelerar el proceso de creación de árboles léxicos que en un primer momento y siguiendo lo descrito en el punto “5. Árboles Léxicos” se realizaba através de un programa cliente llamado Conceptualizer, y que tardaba entre 6-18 horas en calcular un árbol léxico completo, se planteo la posbilidad de migrar las rutinas directamente al servidor.

Esto vino a ser reforzado cuando el planeamiento realizado anteriormente el el punto 11. venía a requerir la rápida concatenación de varios árboles léxicos para su integración con los conceptos similes descritos através de las tablas de frecuencia del árbo léxico.

En este sentido el proceso final de creación de los árboles léxico semánticos se puede esbozar como :

1) Seleccionar un lemma que represente a un concepto raíz.

2) Extraer del corpus las palabras de uso frecuente ordenadas por la relación de peso, entre el concepto raíz y sus relacionados.

3) Iterar en la creación del árbol léxico para cada uno de los términos similes, y calcular nuevamente la distancia pero esta vez tomando en cuanto no solo la distancia jerarquica sino tambien la “distancia” del peso de la relación en términos de frecuencia de aparición.

2b y 3b) Excluir de este procedimiento los términos que podrían categorizarse como ántimos al concepto raíz original.

Como se puede ver en este planteamiento , la distancia entre el concepto raíz y sus relacionados, adquiere una nueva dimensión en el sentido de que ya no sólo es relevante que tan distantes son los términos relacionados en términos de anidamiento / posición jerarquica, sino que también se vuelven más distantes en términos de su frecuencia de uso, siendo más cercanos aquellos que se usan con más frecuencia, y siendo esta magnitud representada por la propiedad weight que representa el pedo de la relación mutua de lemmas para la muestra.

Ante este panorama se planteó la necesidad de crear un cache, o precalculo, de los árboles léxicos de las palabras de uso más común para que se encontrarán disponibles al momento de realizar el entrenamiento enriquecido por el sistema antes descrito de similes de frecuencia de aparición.

Para lograr este objetivo las rutinas empleadas en el software de C-Objetivo Conceptualizer fueron portadas para funcionar directamente en MySQL como Stored Procedures.

Sobrará decir que el proceso de traducción de las funciones fue por demás complejo y dificil de “debuguear” , pero al final se logró con éxito mediante el proceso que se describirá a continuación.

De manera general el proceso opera del modo siguiente:

1) se invoca al Procedure NETdoTAGFrequencyTrees
   a) Este genera una lista o cursor de recursión proviente de la tabla que describe el corpus del lenguaje
   b) Itera sobre el cursor del punto a) enviando cada uno de los lemmas correspondientes al procedure NETdoNewLexTree.

2) cada que se invoca NETdoNewLexTree suceden dos procesos principales:

El proceso de punteros similares :

   a) Se definen los “punteros” de Wordnet para relaciones hipónimas y similes en la tabla

TAGLextreePointers

b) Mediante una llamada a NETdoSetupLexTree se crea una selección de los lemmas que constituyen el nivel 0 o raiz los resultados se escriben en la tabla, indicando el punturo raíz que los vincula, después estos valores se escriben en 3 tablas:

I) TAGLextreeRoots que guarda la referencia de los conceptos raiz.

ii) TAGLextree que es el destinatario final de los árboles y

iii) TAGLextreeCurrents donde se indican los lemmas para el siguiente ciclo de recurción.

c) Después se hace una llamada al método recursivo NETdoLexTree. Este se ejecuta hasta que se alcanza el nivel máximo de jerarquía o que se terminan los elementos existentes en TAGLexTreeCurrents. Recursivan actualiza las tablas:

i) TAGLextree que es el destinatario final de los árboles y

ii) TAGLextreeCurrents donde se indican los lemmas para el siguiente ciclo de recurción.

El proceso de punteros antónimos :

a) Se define el puntero de antónimos el la tabla TAGLextreePointers truncadola primeramente.

b) Mediante una llamada a NETdoSetupLexTree se crea una selección de los lemmas que constituyen el nivel 0 o raiz los resultados se escriben en la tabla, indicando el punturo raíz que los vincula, después estos valores se escriben en 3 tablas:

I) TAGLextreeRoots que guarda la referencia de los conceptos raiz.

ii) TAGLextree que es el destinatario final de los árboles y

iii) TAGLextreeCurrents donde se indican los lemmas para el siguiente ciclo de recurción.

c) Se cambia la definicion de punteros , por los punteros sinónimos e hiponimos, de esta manera se vincularan todos los sinónimos e hiponimos de los terminos antonimos definidos en las raices, pero se preservara una referencia para indicar que propvienen de un árbol antónimo.

d) Después se hace una llamada al método recursivo NETdoLexTree. Este se ejecuta hasta que se alcanza el nivel máximo de jerarquía o que se terminan los elementos existentes en TAGLexTreeCurrents. Recursivan actualiza las tablas:

i) TAGLextree que es el destinatario final de los árboles y

ii) TAGLextreeCurrents donde se indican los lemmas para el siguiente ciclo de recurción.

El resultado final de este procedimiento es la creación de filas en la tabla TAGLexTree donde se define de manera completa el árbol Léxico Semántico Correspondiente al concepto raiz analizado.

La creación de llaves únicas que representen la relación jerarquica única de los elementos en TAGLextree se lgora mediante la concatenación durante el proceso de recursión dando por resultado muestras como la siguiente:

love@branch.1.1431723@v.138078@n.kiss@v

con el formato:

love@	*//raiz del arbol*
branch.	*//si se trata de un termino raiz o rama*
1.	*//nivel jerarquico o de anidamiento*

id	*superkey*	*name*	*branch*	*level*	*offset*	*related*	*lextag*	*magickey*	*type*
4332802	*love@root.-1.7543288@n.7546465@n.love@n*	*love@*	*-1*	*0*	*7543288@n*	*7546465@n*	*12*	*love@n !*	
4331840	*love@branch.1.1410223@v.1236164@v.strike@v*	*love@*	*1*	*6*	*1410223@v*	*1236164@v*	*35*	*strike@v*	*&*
4331841	*love@branch.1.1431723@v.138078@n.kiss@v*	*love@*	*1*	*6*	*1431723@v*	*138078@n*	*35*	*kiss@v*	*&*

1431723@v. //synset rama destino
138078@n. // synset rama origen
kiss@v //lemma de la rama origen

A continuación se muetra el codigo de los Procedures antes citados del más general al más atómico:

MySQL Procedure NETdoTAGFrequencyTrees

```
PROCEDURE `NETdoTAGFrequencyTrees`()
BEGIN

DECLARE lemma VARCHAR(100);
DECLARE related VARCHAR(100);


DECLARE lemmaclean VARCHAR(100);
DECLARE relatedclean VARCHAR(100);

DECLARE lemmaCount INT;

DECLARE _count INT;

DECLARE _current INT;

DECLARE _row INT;

DECLARE no_more_rows INT;

DECLARE cursor1 CURSOR FOR

SELECT  TAGFrequencyIndex.id,TAGFrequencyIndex.
lemma,TAGFrequencyIndex.count  FROM
TAGFrequencyIndex;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET no_more_rows =1;

SET _current=1;

TRUNCATE TABLE TAGLextreeSuperStatus;

SELECT  currentID  FROM  TAGLextreeSuperStatus
WHERE id=1 LIMIT 1 INTO _count ;

SET no_more_rows =0;

if _count is NULL THEN

 set _count=0;

END IF;

OPEN cursor1;

if _count>0 THEN

  LOOP2: loop

    fetch cursor1 into _row,lemma,lemmaCount;

    set _current =_current+1;

      SELECT _current;

    if _current>=_count then

      leave LOOP2;

    if no_more_rows=1 then

      leave LOOP2;

    end if;

    end if;

  end loop LOOP2;

END IF;

SET no_more_rows =0;

LOOP1: loop


    fetch cursor1 into _row,lemma,lemmaCount;

    if no_more_rows=1 then

      leave LOOP1;

    end if;

   SET lemmaclean =REPLACE(lemma,' ','_');

   SET lemmaclean = CONCAT(lemma,'@%');

  INSERT INTO

     TAGLextreeSuperStatus ( id, currentID, lemma,
lemmaclean) VALUES
```

```
   (1, _row, lemma,lemmaClean) ON DUPLICATE KEY
UPDATE  currentID=VALUES(currentID),lemma=VALU
ES(lemma),lemmaClean=VALUES(lemmaClean) ;

        call NETdoNewLextree(lemmaclean);

 end loop LOOP1;

 close cursor1;

end
---------------

MySQL Procedure NETdoNewLextree

-- ----------------------------------------------------------
---------------------
-- Routine DDL
-- ----------------------------------------------------------
---------------------
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`NETdoNewLextree`(IN lemma VARCHAR(100))
BEGIN



DECLARE lemmaClean VARCHAR(100);

SET lemmaClean =  REPLACE(lemma,'%',");


#elminar registro anterior

TRUNCATE TABLE TAGLextreeStatus;

DELETE FROM TAGLextree WHERE name LIKE
lemmaClean;

#primero los sinonimos

TRUNCATE TABLE TAGLextreeMemory;

INSERT INTO TAGLextreeStatus (id,branch)

VALUES (1,1) ON DUPLICATE KEY UPDATE
branch=VALUES(branch);

TRUNCATE TABLE TAGLextreePointers;

INSERT INTO TAGLextreePointers (pointer)
VALUES("<");

INSERT INTO TAGLextreePointers (pointer)
VALUES("&");

call deyavoo.NETdoSetupLextree( lemma);

call deyavoo.NETdoLextree;

#ahora los antonimos

TRUNCATE TABLE TAGLextreeMemory;

INSERT INTO TAGLextreeStatus (id,branch)

VALUES (1,-1) ON DUPLICATE KEY UPDATE
branch=VALUES(branch);

TRUNCATE TABLE TAGLextreePointers;

INSERT INTO TAGLextreePointers (pointer)
VALUES("!");

call deyavoo.NETdoSetupLextree( lemma);

TRUNCATE TABLE TAGLextreePointers;

INSERT INTO TAGLextreePointers (pointer)
VALUES("<");

INSERT INTO TAGLextreePointers (pointer)
VALUES("&");
call deyavoo.NETdoLextree;

END

---------------

MySQL Procedure NETdoNewLextree

-- ----------------------------------------------------------
---------------------
-- Routine DDL
-- ----------------------------------------------------------
---------------------
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE
`NETdoSetupLextree`(IN lemma VARCHAR(100))
BEGIN
```

```
DECLARE no_more_rows INT;

DECLARE _superkey VARCHAR(500);

DECLARE _name VARCHAR(100);
DECLARE _branch INT(11);
DECLARE _level INT(11);
DECLARE _offset VARCHAR(100);

DECLARE _related VARCHAR(100);
DECLARE _lextag INT(11);
DECLARE _magickey VARCHAR(100);
DECLARE _type VARCHAR(100);

DECLARE _newsuperkey VARCHAR(500);

DECLARE lemmaClean VARCHAR(100);

DECLARE cursor1 CURSOR FOR

SELECT superkey,name,branch,level,offset,related,lex
tag,magickey,type FROM TAGLextreeTMP;

DECLARE CONTINUE HANDLER FOR NOT FOUND
SET no_more_rows =1;

SET lemmaClean =  REPLACE(lemma,'%','');

TRUNCATE  TAGLextreeRoots;

TRUNCATE  TAGLextreeCurrents;

TRUNCATE  TAGLextreeMemory;

TRUNCATE  TAGLextreeTMP;

INSERT INTO TAGLextreeStatus (id, level, treename)

VALUES (1, 0, lemmaClean) ON DUPLICATE KEY
UPDATE treename=VALUES(treename),branch=bran
ch, level=0, lastcount=0;

INSERT IGNORE INTO TAGLextreeTMP
(superkey,name,branch,level, offset,lextag,type,magic
key,related)

SELECT
RAND(),

TAGLextreeStatus.treename as name,
TAGLextreeStatus.branch as branch,
TAGLextreeStatus.level as level,
SYNSET.lemmasynset as offset,
SYNSET.lemmatag as lextag,
BADatas_Relations.type as type,
#BADatas_Indexes.magickey

SYNSET.lemma as magickey,
#BADatas.magickey as pointer,
#BADatas.lextag,
#BADatas.gloss

BADatas_Relations.related
FROM
TAGLextreeStatus
INNER JOIN
BADatas
INNER JOIN BADatas_Indexes
ON BADatas_Indexes.offset = BADatas.magickey
INNER JOIN BADatas_Relations
ON BADatas_Relations.related = BADatas.magickey

INNER JOIN TAGLextreePointers

ON BADatas_Relations.type = TAGLextreePointers.
pointer

INNER JOIN (
SELECT DISTINCT
BADatas_Indexes.magickey as lemma,
BADatas.magickey as lemmasynset,
BADatas.lextag lemmatag,
BADatas.gloss
FROM
BADatas
INNER JOIN BADatas_Indexes
ON BADatas_Indexes.offset = BADatas.magickey
WHERE
BADatas_Indexes.magickey LIKE lemma
) SYNSET

WHERE BADatas_Relations.offset IN( SYNSET.
lemmasynset );

# CREAR LAS SUPERLLAVES

SET no_more_rows =0;

OPEN cursor1;

  LOOP1: loop

    fetch cursor1 into _superkey,_name,_branch,_
level,_offset,_related,_lextag,_magickey,_type;
```

```
    if no_more_rows=1 then
      leave LOOP1;
    end if;
    SET _newsuperkey = CONCAT(_name,'root.');
      SET _newsuperkey = CONCAT(_newsuperkey,_branch);
    SET _newsuperkey = CONCAT(_newsuperkey,'.');
      SET _newsuperkey = CONCAT(_newsuperkey,_offset);
    SET _newsuperkey = CONCAT(_newsuperkey,'.');
      SET _newsuperkey = CONCAT(_newsuperkey,_related);
    SET _newsuperkey = CONCAT(_newsuperkey,'.');
      SET _newsuperkey = CONCAT(_newsuperkey,_magickey);

    INSERT INTO TAGLextreeRoots
     ( superkey,name,branch,level,offset,related,lextag,magickey,type)  VALUES
        (_newsuperkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type) ON DUPLICATE KEY UPDATE superkey=superkey;
    INSERT INTO TAGLextree
     ( superkey,name,branch,level,offset,related,lextag,magickey,type)  VALUES
        (_newsuperkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type) ON DUPLICATE KEY UPDATE superkey=superkey;
    INSERT INTO TAGLextreeCurrents
     ( superkey,name,branch,level,offset,related,lextag,magickey,type)  VALUES
        (_newsuperkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type) ON DUPLICATE KEY UPDATE superkey=superkey;
  end loop LOOP1;

CLOSE cursor1;

TRUNCATE  TAGLextreeTMP;

END

MySQL Procedure NETdoLextree

-- ----------------------------------------------------------------------------
-- Routine DDL
-- ----------------------------------------------------------------------------
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `NETdoLextree`()
BEGIN

DECLARE _counter INT;

DECLARE _currentlevel INT;

DECLARE no_more_rows INT;
DECLARE _superkey VARCHAR(500);

DECLARE _name VARCHAR(100);
DECLARE _branch INT(11);
DECLARE _level INT(11);
DECLARE _offset VARCHAR(100);

DECLARE _related VARCHAR(100);
DECLARE _lextag INT(11);
DECLARE _magickey VARCHAR(100);
DECLARE _type VARCHAR(100);

DECLARE _newsuperkey VARCHAR(500);

DECLARE cursor1 CURSOR FOR

SELECT superkey,name,branch,level,offset,related,lextag,magickey,type FROM TAGLextreeTMP;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows =1;

SET no_more_rows=0;
```

```
#DROP TEMPORARY TABLE IF EXISTS _RELATIONS;

#aumentar la cuenta

INSERT INTO

TAGLextreeStatus (id) VALUES

(1) ON DUPLICATE KEY UPDATE level=level+1,branch=branch;

#incir los selects

TRUNCATE TABLE TAGLextreeTMP;

INSERT INTO TAGLextreeTMP (

superkey,

name,

branch,

level,

offset,

lextag,

magickey,

type,

related

)

SELECT DISTINCT
CURRENT_TIMESTAMP,
TAGLextreeStatus.treename as name,
TAGLextreeStatus.branch as branch,
TAGLextreeStatus.level as level,
BADatas.magickey as offset,
BADatas.lextag as lextag,
BADatas_Indexes.magickey as magickey,

_RELATIONS.type,

_RELATIONS.related

FROM
TAGLextreeStatus

INNER JOIN
(SELECT DISTINCT BADatas_Relations.offset, BADatas_Relations.related, BADatas_Relations.type from

TAGLextreeCurrents INNER JOIN
BADatas_Relations ON
BADatas_Relations.offset = TAGLextreeCurrents.related

INNER JOIN

TAGLextreePointers ON

BADatas_Relations.type = TAGLextreePointers.pointer

LEFT JOIN
TAGLextreeMemory ON
TAGLextreeMemory.offset=BADatas_Relations.offset
WHERE
TAGLextreeMemory.offset is null ) as _RELATIONS

INNER JOIN
BADatas ON
_RELATIONS.offset = BADatas.magickey

INNER JOIN
BADatas_Indexes ON
_RELATIONS.offset = BADatas_Indexes.offset

INNER JOIN
TAGLextreeRoots ON
TAGLextreeRoots.lextag = BADatas.lextag;

# CREAR LAS SUPERLLAVES

# CREAR LAS SUPERLLAVES

TRUNCATE TABLE TAGLextreeCurrents;

SET no_more_rows =0;

OPEN cursor1;
```

```
LOOP1: loop

    fetch cursor1 into _superkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type;

  if no_more_rows=1 then

    leave LOOP1;

  end if;

  SET _newsuperkey = CONCAT(_name,'branch.');

    SET _newsuperkey = CONCAT(_newsuperkey,_branch);

  SET _newsuperkey = CONCAT(_newsuperkey,'.');

    SET _newsuperkey = CONCAT(_newsuperkey,_offset);

  SET _newsuperkey = CONCAT(_newsuperkey,'.');

    SET _newsuperkey = CONCAT(_newsuperkey,_related);

  SET _newsuperkey = CONCAT(_newsuperkey,'.');

    SET _newsuperkey = CONCAT(_newsuperkey,_magickey);

 # INSERTAR RESULTADOS ACTUALES

  INSERT INTO TAGLextreeCurrents

   ( superkey,name,branch,level,offset,related,lextag,magickey,type)  VALUES

      (_newsuperkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type) ON DUPLICATE KEY UPDATE superkey=superkey;

  INSERT INTO TAGLextreeMemory

   ( superkey,name,branch,level,offset,related,lextag,magickey,type)  VALUES

      (_newsuperkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type) ON DUPLICATE KEY UPDATE superkey=superkey;

    INSERT INTO TAGLextree

   ( superkey,name,branch,level,offset,related,lextag,magickey,type)  VALUES

      (_newsuperkey,_name,_branch,_level,_offset,_related,_lextag,_magickey,_type) ON DUPLICATE KEY UPDATE superkey=superkey;

  end loop LOOP1;

CLOSE cursor1;

# DESTURIR TABLA TEMPORAL
TRUNCATE  TAGLextreeTMP;

# CONTAR LOS RESULTADOS PARA DETERMINAR LA RECURSION

SELECT COUNT(*) FROM TAGLextreeCurrents INTO _counter;

INSERT INTO

TAGLextreeStatus (id,lastcount) VALUES

(1,_counter)  ON  DUPLICATE  KEY  UPDATE branch=branch, lastcount=VALUES(lastcount);

SELECT level from TAGLextreeStatus  LIMIT 1 INTO _currentlevel;

if _counter >0  AND _currentlevel<7 THEN

    # PREPARAR SIGUIENTE RECURSION

    call deyavoo.NETdoLextree;

END IF;

END
```

11c. Entrenamiento Neuronal Completo

Como se ha discutido anteiormente, después de la creación de los árbelos léxicos para las palabras de uso más frecuente que aparecen el corpus dinámico del lenguaje, el siguiente paso es el entrenamiento final de las neuronas léxico-semanticas contextuales.

Para este último proceso de entrenamiento, se creo una familia de Stored Prcedures de alto desempeño

para ser calculados a gran velocidad directamente dentro de MySQL.

El procedimiento final de entrenamiento se basa, como ya se ha discutido anteriormente, en que :

1) Se definen los términos a “aprender” en la tabla NETNeuronsSimplesIndex, y se toma un conceptoraiz para iterar y enviarlo a NETdoTAGFrequencyNeuron en el paso 2

2) a partir de un concetp raíz se listan las palabras conformantes del corpus, organizadas por aquellas que tienen una relación de peso “mayor” entre ellas.

3) Posteriormente se realiza un proceso iterativo seleccionando todos aquellos árboles léxicos de las palabras relacionadas con el término raíz, haciendo previamente ajustes para excluir aquellos conceptos que podrían provenir de los conceptos antónimos de la raíz.

4) Esta información es almacenada en la tabla NETNeuronsSimples, en donde se almacena la información léxico semántica y se deine el importantísimo valor de la propiedad distance, que se define para reflejar tanto la distancia jerarquica como la distancia contextual o de frecuencia de uso en una variable única integrada.

((1-_weight) + (level/7)) as distance

El resultado final es la creación de llaves que representan el “conocimiento” neuronal léxico y semántico respecto al concepto raíz vinculando la referncia contextual semñantica y léxica en una unica variable llamada distance.

Aquí un ejemplo:

love@14038993@n 1.13866

Esto nos indica que la relación entre el concepto love , y el concepto relacionado en el synset 14038993@n es de 1.13866.

Para mayor ejemplificación abajo se muestra un sampleo de la tabla organizada por menor distancia, donde se puede notar que los synsets más cercanos corresponden con las definiciones directas de amor por ejemplo en el lugar 1 el synset:

5813229@n nos dice “any object of warm affection or devotion; ‘the theater was her first love’; ‘he has a passion for cock fighting’;”

y en el lugar 13 el synset :

1427278@v dice “ have sex without being married “

En otro caso opuesto el synset con mayor distancia para love es el :

13496017@ relacionado con “release@“ que dice “(chemistry) the absorption of a liquid by a solid or gel “ seguido por

1476046@a relacionado con “senior@” que dice “calling for the strength of a man; ‘a man-sized job’ “

De esta manera mediante la tabla NETNeuronsSimples resulta posible obtener el

id	magickey	treename	related	offset	weight	level	distance
1	love@5813229@n	love@	love@	5813229@n	1	0	0
2	love@7488340@n	love@	love@	7488340@n	1	0	0
3	love@9849598@n	love@	love@	9849598@n	1	0	0
4	love@846515@n	love@	love@	846515@n	1	0	0
5	love@7543288@n	love@	love@	7543288@n	1	0	0
6	love@1426397@v	love@	love@	1426397@v	1	0	0
7	love@1828736@v	love@	love@	1828736@v	1	0	0
8	love@1775535@v	love@	love@	1775535@v	1	0	0
9	love@1775164@v	love@	love@	1775164@v	1	0	0
10	love@7544213@n	love@	love@	7544213@n	1	1	0.142857
11	love@7544351@n	love@	love@	7544351@n	1	1	0.142857
12	love@7546125@n	love@	love@	7546125@n	1	1	0.142857
13	love@1427278@v	love@	love@	1427278@v	1	1	0.142857

valor de exitación o distancia que existe entre el concepto definido por la neurona y cualquier otro concepto o synset, representando caulquier lemma del diccionario.

Como se verá más adelante, esta constituye la base del sistema de indexado y de consulta.

Para finalizar se muestra el código de los Procedures para la creación de las neuronas:

MySQL Procedure NETdoTAGFrequencyNeurons

```
-- -----------------------------------------------------------------------
-- Routine DDL
-- -----------------------------------------------------------------------
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `NETdoTAGFrequencyNeurons`()
BEGIN

DECLARE _lemma VARCHAR(100);

DECLARE no_more_rows INT;

DECLARE cursor1 CURSOR FOR

SELECT lemma   FROM NETNeuronsSimplesIndex;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows =1;

SET no_more_rows =0;

open cursor1;

LOOP1: loop

    fetch cursor1 into _lemma;

     SELECT _lemma;

    if no_more_rows=1 then

      leave LOOP1;

    end if;

    call NETdoTAGFrequencyNeuron(_lemma);

 end loop LOOP1;

 close cursor1;

end
```

MySQL Procedure NETdoTAGFrequencyNeuron

```
-- -----------------------------------------------------------------------
-- Routine DDL
-- -----------------------------------------------------------------------
DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `NETdoTAGFrequencyNeuron`(IN currentLemma VARCHAR(100))
BEGIN

DECLARE lemmaAny VARCHAR(100);

DECLARE lemmaTreeName VARCHAR(100);

DECLARE _lemma VARCHAR(100);

DECLARE _related VARCHAR(100);

DECLARE _weight VARCHAR(100);

DECLARE _count int(100);

DECLARE no_more_rows INT;

DECLARE cursor1 CURSOR FOR

SELECT lemma, related, weight  FROM NETindexTMP ORDER BY weight DESC;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET no_more_rows =1;

SET lemmaTreeName = CONCAT(REPLACE(currentLemma," ","_"),"@") ;

SET lemmaAny = CONCAT(lemmaTreeName,"%");
```

```
SELECT lemmaAny;

drop table  if exists NETindexTMP;
drop table if exists NETantonymsTMP;

CREATE TABLE if not exists NETantonymsTMP (
`id` int(11) unsigned NOT NULL,
offsets VARCHAR(100) NOT NULL,
magickey varchar(100) NOT NULL,
lemma varchar(100) NOT NULL,
PRIMARY KEY (id)
);

CREATE TABLE NETindexTMP (
  lemma varchar(100) NOT NULL,
  related varchar(100) NOT NULL,
  weight float DEFAULT NULL
);

CREATE TABLE if not exists NETNeuronsSimples (

`id`bigint(33)unsignedNOTNULLAUTO_INCREMENT,

magickey VARCHAR(100) NOT NULL,
treename VARCHAR(100) NOT NULL,
related VARCHAR(100) NOT NULL,

offset VARCHAR(100) NOT NULL,
weight float,
level int(11),
distance float,
PRIMARY KEY (id),

UNIQUE KEY `magickey` (`magickey`),

KEY `treename` (`treename`),

KEY `offset` (`offset`)
);

CREATETABLEifnotexistsNETNeuronsSimplesStatus
(

`id`bigint(33)unsignedNOTNULLAUTO_INCREMENT,

treename VARCHAR(100) NOT NULL,
related VARCHAR(100) NOT NULL,
lastCount VARCHAR(100) NOT NULL,
PRIMARY KEY (id)
);

#agregar los antonimos
INSERT INTO NETantonymsTMP
SELECT DISTINCT id,offset,magickey,
REPLACE(REPLACE(REPLACE(REPLACE(magickey,
"@a",""),"@n",""),"@v",""),"@r","") FROM TAGLextree
WHERE NAME LIKE lemmaAny and branch=-1 and
level!=0;


#agregar los sinonimos excluyendo los anotnimos
INSERT INTO NETindexTMP
SELECT DISTINCT
CONCAT(REPLACE(lemma," ","_"),"@") as lemma,
CONCAT(REPLACE(related," ","_"),"@") as related ,
weight
FROM TAGFrequencyPro WHERE
lemma LIKE currentLemma AND
related NOT IN ( SELECT lemma FROM
NETantonymsTMP ) ;

#ahora hacer un loop por el cursor

SET no_more_rows=0;

DELETE FROM NETNeuronsSimples WHERE
treename LIKE lemmaTreeName;

OPEN cursor1;

  LOOP1: loop

  fetch cursor1 into _lemma,_related,_weight;

    if no_more_rows=1 then

        leave LOOP1;

     end if;


    INSERT  INTO NETNeuronsSimples

    (SELECT DISTINCT

     RAND(),

     CONCAT(lemmaTreeName,offset) as magickey,

    lemmaTreeName,

    _related,
```

```
    offset,

    _weight,

    level,

    ((1-_weight) + (level/7) ) as distance

      FROM TAGLextree where  branch >0 AND
name LIKE  _related)

            ON DUPLICATE KEY UPDATE
NETNeuronsSimples.id=NETNeuronsSimples.id;


  SELECT COUNT(*) FROM NETNeuronsSimples
into _count;

    #if(_count >30000) then

     # SET no_more_rows=1;

    #end if;


       INSERT INTO NETNeuronsSimplesStatus
(id,treename,related,lastCount)  VALUES(1,_
lemma,_related,_count)

            ON DUPLICATE KEY UPDATE
treename=VALUES(treename),
related=VALUES(related),
lastCount=VALUES(lastCount);

    #SET no_more_rows=1;

 end loop LOOP1;

END
```

12. Toughtag y el Sistema de Indexado-Consulta

Retomando lo dicho en el apartado “10. Definición
del Sentido Común” y en base a lo recientemente discuto en relación al entrenamiento neuronal en el apartado 11c. el siguiente paso es tomar la lista de 44 conceptos que definen el juicio de “sentido comun” y procesarla mediante el proceso de entrenamiento neuronal mediante una llamada a NETdoTAGFrequencyNeurons.

Una vez iniciado el proceso de “aprendizaje” la máquina se encontrará calculando en la tabla NETNeuronsSimples , la relación absoluta que guarda cada concepto definiendo el juicio de “sentido común” y todas las otras palabras que confroman su arbol léxico semántico contextual.

El resultado final de este cálculo es por ejemplo para el concepto “love” 49010 conexiones, para “hate” 46952, “happy” 47966.

Cada una de estas conexiones representa una conexión directa entre el sentido (synset) de un lemma determinado y el concepto raiz representado mediante cada neurona.

Así pues la forma en que opera el sistema de consulta de los niveles de exitación de los valores neuronales se puede describir de modo general como:

1) Definir una palabra , o palabra grupo
2) Extraer la raíz o lemma correspondiente
3) Buscar el uso de la lengua más claro posible
4) Definir los posibles sentidos para ese lemma
5) Consultar en base al lemma la relación que guarda con una determinada neurona.

De este modo si tomamos por ejemplo el concepto “war” y lo evaluamos contra las 44 neuronas obtenemos los siguientes valores ordenados por los más exitados primero:

“war” es :

Salida de DJNewstein.app

```
{"poor",:1},
{"sad",:0.95925182104110718},
"pleasure",:0.94591891765594482},
"ugly",:0.86389398574829102},
"success",:0.7807847261428833},
"fail",:0.7433050274848938},
"dream",:0.65192675590515137},
"health",:0.43198150396347046},
"lie",:0.42006304860115051},
"pain",:0.38635575771331787},
"spirit",:0.37265774607658386},
"hurt",:0.35008975863456726},
"school",:0.3178476095199585},
"party",:0.31474384665489197},
```

”dream”,:0.30672502517700195},
”work”,:0.30364453792572021},
”day”,:0.3014684319496154 8},
”hope”,:0.29664376378059387},
”hate”,:0.2954447865486145},
”night”,:0.29385361075401306},
”girl”,:0.29317250847816467},
”happy”,:0.29276993870735168},
”money”,:0.2924078404903 4119},
”friend”,:0.29196131229400635},
”tired”,:0.27904590964317322},
”hungry”,:0.27687731385231018},
”love”,:0.27615436911582947},
”pro-healt”,:0.27521342039108276},
”real”,:0.27257177233695984},
”sex”,:0.2682439386844635},
”care”,:0.26526129245758057},
”fun”,:0.16125936806201935},
”beautiful”,:0.14644394814968109},
”rich”,:0.13291752338409424},
”sick”,:0.12514747679233551},
”dead”,:0.11428459733724594},
”young”,:0.11125706881284714},
”honest”,:0.086021184921264648},
”boy”,:0.078431382775306702},
”family”,:0},
”lost”,:0},
”gay”,:0},
”straight”,:0},
”life”,:0}

”night”,:0.33943295478820801},
”success”,:0.33902224898338318},
”lie”,:0.32123133540153503},
”school”,:0.31060656905174255},
”day”,:0.30670419335365295},
”health”,:0.2979235053062439},
”pain”,:0.28493136167526245},
”boy”,:0.28266355395317078},
”sex”,:0.2643965482711792},
”pro-healt”,:0.26415583491325378},
”hate”,:0.22652527689933777},
”friend”,:0.2176443487405777},
”poor”,:0.21103687584400177},
”party”,:0.20895974338054657},
”honest”,:0.19031509757041931},
”ugly”,:0.1511128693819046},
”fun”,:0.13613471388816833},
”work”,:0.12488824129104614},
”tired”,:0.12390623986721039},
”hungry”,:0.12293114513158798},
”dream”,:0.12245945632457733},
”pleasure”,:0.11431646347045898},
”sick”,:0.0945315659046 1731},
”real”,:0.088868297636508942},
”beautiful”,:0.012205327861011028},
”family”,:0},
”lost”,:0},
”gay”,:0},
”straight”,:0},
”life”,:0}]

ahora probemos con la palabra “peace”, nos resulta bajo el mismo criterio:

“peace” es :

”happy”,:0.99999994039535522},
”love”,:0.88778287172317505},
”hurt”,:0.83868730068206787},
”sad”,:0.71294718980789185},
”young”,:0.67121016979217529},
”dream”,:0.62792062759399414},
”fail”,:0.52584367990493774},
”girl”,:0.48813602328300476},
”spirit”,:0.47025170922279358},
”money”,:0.45244982838630676},
”dead”,:0.44241189956665039},
”care”,:0.43902894854545593},
”hope”,:0.40471607446670532},
”rich”,:0.3975214958190918},

Los resultados del sistema podrían contraríar en algunos sentidos lo esperado por un categorizador humano, sin embargo hay que resaltar que reflejan el estado “real” contextual según la manera en que se escribía en twitter en Estados Unidos entre agosto-octubre 2011.

El siguiente paso, requería del planteamiento de un sistema de indexamiento que permitiera mantener la singularidad de los conjuntos de impulsos pero que a la vez permitiera agrupar aquellos altamente similares.

Así fue como se planteo el concepto de “tag de pensamiento” o “tought tag”, el objetivo de este concepto era econtrar una forma de crear un hash que representara de manera única un conjunto de impulsos neuronales manteniendo su simultaneidad, paralelismo y multidimensionalidad.

Para poder lograr este objetivo se procedió a determinar el nivel minimo de precisión requerido para disintguir entre conceptos diferentes.

Luego de realizar pruebas con rangos del 30% y del 10% sin obtener resultados satisfactorios, se dició seleccionar un valor mucho menor del 2% que entrego resultados satisfactorios.

Entonces se escribieron funciones que convertirían cualquier valor en rango de 0 a 1 a su correspondiente valor en una coleeción de caracteres de base 50 como el siguiente:

Extracto BabilonCore.m

```
base50= [[NSArray alloc] initWithObjects:
        @"9",
        @"8",
        @"7",
        @"6",
        @"5",
        @"4",
        @"3",
        @"2",
        @"1",

        @"A",
        @"B",
        @"C",
        @"D",
        @"E",
        @"F",
        @"G",
        @"H",
        @"I",
        @"J",
        @"K",
        @"L",
        @"M",
        @"N",
        @"O",
        @"P",
        @"Q",
        @"R",
        @"S",
        @"T",

        @"a",
        @"b",
        @"c",
        @"d",
        @"e",
        @"f",
        @"g",
        @"h",
        @"i",
        @"j",
        @"k",
        @"l",
        @"m",
        @"n",
        @"o",
        @"p",
        @"q",
        @"r",
        @"s",
        @"t",

        @"0",
        nil ];
```

Y el método para mapear los valores como :

Extracto BabilonCore.m

```
-(NSString *)level50ForMag:(NSNumber *)magnitude {
    NSString *result;

    if(base50 ==nil) [self initDB];

    float mag = [magnitude floatValue];
    float indexf = mag / .02;

    if( indexf == 0){
        NSString *level = [base50 objectAtIndex:49];
        return level;
    }

    indexf=50.0-indexf;
    int index = round(indexf);

    if( index == 0){
        NSString *level = [base50 objectAtIndex:0];
        return level;
    }


    NSString *level = [base50 objectAtIndex:index-1];

    return level;

   //return result;
}
```

De este modo regresando a los ejemplos anteriores resulta posible representar primeramente una palabra y sus valores neuronales correspondientes dentro de un determinado cerebro, mediante una llave única que es representativa de todas las relaciones en una manera simultanea y paralela, por ejemplo la palabra “peace” devuelve el “toughttag”:

“peace” es:
Kj9kAkjk9Ka0TK9ikR0ajKj0jjJ0ak0tHI00kAAa0K0a

Pero más aún usando una función iterativa y aditiva que calcula el valor de todas las palabras en una oración (implementando un código específico para invertir los valores al usar auxiliares negativos) es posible calcular el ToughTag para cualquier texto limitando la longitud de texto únicamente a la capacidad de procesamiento del harware.

Por ejemplo la frase

“al you need is love” es:
9eebeQieGendIffffabGFIO0HIJ0OTQOJe00KkecMf0b

cuando sus valores neuronales son:

Salida de DJNewstein.app

“love”:1,
“dream”:0.70545125007629395
“beautiful”:0.68762964010238647
“tired”:0.67411577701568604
“girl”:0.65911388397216797
“real”:0.64961963891983032
“boy”:0.63772571086883545
“sex”:0.63711684942245483
“hope”:0.62596476078033447
“night”:0.61758029460906982
“rich”:0.60547071695327759
“care”:0.55866318941116333
“money”:0.5270305871963501
“pro-healt”:0.52120530605316162
“friend”:0.5145910382270813
“lie”:0.48677974939346313
“spirit”:0.47568634152412415
“hungry”:0.42293873429298401
“school”:0.39287844300270081
“young”:0.37868073582649231
“sad”:0.37717649340629578
“dead”:0.37449178099632263
“pain”:0.3670269250869751
“sick”:0.34287133812904358
“hate”:0.32648363709449768
“ugly”:0.32395237684249878
“pleasure”:0.32353603839874268
“happy”:0.32073655724525452
“honest”:0.31707227230072021
“fail”:0.31351646780967712
“day”:0.31021663546562195
“dream”:0.30932179093360901
“hurt”:0.30882471799850464
“work”:0.30572021007537842
“fun”:0.30034774541854858
“party”:0.29864269495010376
“success”:0.24953562021255493
“poor”:0.20789757370948792
“health”:0.14701423048973083
“family”:0
“lost”:0
“gay”:0
“straight”:0
“life”:0

De este modo se vuelve posible crear un id o hash único para representar cualquier texto , manteniendo el contexto de significado que refleja el analisis neuronal simultaneo del sistema.

Por último se muestra el método recursivo para la valoración de una oración en una neurona específica:

Extracto de NETBrain2.m

```
-(NSNumber *) valueForText:(NSArray *)magicArray inNeurona:(NETABNeurona2 *)neurona {

    NSAutoreleasePool *pool =[[NSAutoreleasePool alloc] init];


  float result =0;
  BOOL negateNextChance=NO;
  BOOL negateNow=NO;

  for(int i=0;i < [magicArray count]; i++){

    NSArray *words = [magicArray objectAtIndex:i];

    float wordVal=0;
    //for(int j=0;j< [words count];j++){

    NSString *magicKey=[words objectAtIndex:0];
                    NSNumber *value = [self
```

```
valueForKeyIndex:[NSNumber numberWithInt:i] inText:magicArray inNeurona:neurona];

    // NSLog(@"magickey=%@",magicKey);

    if ([self isaNegation:magicKey]){
        negateNextChance=YES;
        value=nil;
    } else if ([self isaBreak:magicKey]){
        negateNextChance=NO;
        negateNow=NO;
        value=nil;
            } else if( [self isaVerb:magicKey] && negateNextChance){
        negateNow=YES;
            } else if( [self isaAdj:magicKey] && negateNextChance){
        negateNow=YES;
            } else if( [self isaNoun:magicKey] && negateNextChance){
        negateNow=YES;
    }

   NSLog(@"v %@ = %f",magicKey, [value floatValue]);

    if(value !=nil){
        float v = [value floatValue];
        wordVal = v;
    }

    NSLog(@"wordVal %@ = %f",magicKey,wordVal);
    //}
    if(negateNow) {
        //wordVal = wordVal*-1;
        if(wordVal!=0){
            wordVal = 1.0 -wordVal;
        }

        negateNow=NO;
        negateNextChance=NO;
        //  NSLog(@"negating ");
    }


    //NSLog(@"current=%f + %f",result,wordVal);
    result = result +wordVal;
  }



  [pool drain];
  return [[NSNumber alloc ] initWithFloat:result];
}
```

13. Revaloracion de experiencia real

Una vez definido el sistema de indexamiento, fue necesario volver a capturar nuevos mensajes desde tweeter como se describe en el apartado “6. Experiencia del mundo real” pero esta vez indexandolos usando el sistema de ToughTag antes descrito.

El proceso de indexamiento consiste en crear un indice general donde se almacena cada llave única que se crea en el sistema, y en ella se va realizando un promedio de los valores numéricos que representan el rango correspondiente para esa llave,

Extracto de SQLAlejandria.m

```
-(void)insertTagTough:(NSDictionary *)tough{
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

  [self connectDeyavoo];

  NSMutableString *update=[[NSMutableString alloc] init];
  for(int i=0; i<50;i++){
      NSString *line = [NSString stringWithFormat:@" n%d=( VALUES(n%d)*.5 +n%d*.5 ) ",i+1,i+1,i+1];
      [update appendString:line];
      if(i < 49){
          [update appendString:@","];
      }
  }

 NSString*updater=[NSStringstringWithFormat:@"ON DUPLICATE KEY UPDATE %@",update];
  //NSLog(@"updater %@",updater);

  MysqlInsert *insertCommand = [MysqlInsert insertWithConnection:connection];
  insertCommand.table =kTableTAGToughs2;
  insertCommand.extraCommands=updater;
  insertCommand.rowData=tough;
  [insertCommand execute];

  //NSNumber *result =insertCommand.rowid;
  //printf("result row=%d\n",[result intValue]);

  insertCommand =nil;
```

```
    [pool drain];

    //return  result;
}
```

Así esta tabla lleva una relación un conteo único de cada tipo distinto de pensamiento indexado.

Posteriormente ya con su indice correspondiente el pensamiento es ligado mediante una tabla NxN donde se relaciona el id del registro web, con el id correspondiente del ToughTag.

14. Algoritmo de busqueda por distancia N-dimensional

Mediante el sistema descrito anteriormente, es posible encontrar los pensamientos más similares en el contexto N-dimensional antes descrito a partir de calcular la menor distancia euclidiana entre los valores de sus términos.

A fin de ejecutar esta tarea se escribió la siguiente función que permite devolver los pensamientos más similares a otro:

Extracto de SQLAlejandria.m

```
-(NSArray * )getSimilarToughsTo:(NSDictionary*)levels
negate:(BOOL)negate{

        NSAutoreleasePool *pool =[[NSAutoreleasePool
alloc] init];
    NSDate *date1 = [NSDate date];
    [self connectDeyavoo];
         NSMutableString  *distance=[[NSMutableString
alloc] init];;
         NSMutableString  *neurons=[[NSMutableString
alloc] init];;
    NSString *mindist=@"<1.5";

    //for (int i=0; i <[levels count]; i++) {

 NSMutableDictionary*inverted=[[NSMutableDictionary
alloc] init];
                    NSMutableDictionary      *toplevels
=[[NSMutableDictionary alloc] init];


    float prom=0;

    for (int i=0; i <kMaxToughs; i++ ) {

                            NSString     *key=     [NSString
stringWithFormat:@"n%d",i+1];
        NSNumber *value =     [levels objectForKey:key];

        if(i==0){
            prom = [value floatValue];
        }else{
            prom = prom*0.5 + (  [value floatValue]*0.5);
        }

    }
    prom=prom;

    for (int i=0; i <kMaxToughs; i++ ) {

                            NSString     *key=     [NSString
stringWithFormat:@"n%d",i+1];
        NSNumber *value =     [levels objectForKey:key];

        if( [value floatValue] > prom){

            [toplevels setValue:value forKey:key];


        }


    }
    toplevels=levels;

    NSArray *topKeys = [toplevels allKeys];



    if(negate){

        for (int i=0; i <[topKeys count]; ) {

            NSString *key= [topKeys objectAtIndex:i];
       NSNumber *value =   [toplevels objectForKey:key];

           NSString *number = [key stringByReplacingOc
currencesOfString:@"n" withString:@""];
            int n=[number intValue];
            int none;

            if(n%2 == 0){
                none= n-1;
            }else{
                none= n+1;
```

```
        }

                        NSString *keyNone= [NSString stringWithFormat:@"n%d",none];
        [inverted setValue:keyNone forKey:value];

        i=i+2;

    }
    //NSLog(@"original %@",levels);
    //NSLog(@"inverted %@",inverted);
    levels=inverted;
    topKeys=[levels allKeys];
  }


  for (int i=0; i <[topKeys count]; i++) {
    NSString *key=      [topKeys objectAtIndex:i];
   NSNumber *value =  [toplevels objectForKey:key];

    //SQRT(POW(0-n1,2) + POW(0-n2,2))
    NSString *line;
    float val= [value floatValue];
    if( i <[topKeys count]-1){
            line = [NSString stringWithFormat:@" POW(%f-%@,2) +",val,key];
    } else{
            line = [NSString stringWithFormat:@" POW(%f-%@,2) ",val,key];
    }

    [distance appendString:line];

   NSString *neuron= [NSString stringWithFormat:@" TAGToughs2.n%d, ",i+1];
    [neurons appendString:neuron];
  }

  NSString *command = [NSString stringWithFormat:
                @"select ( %@ ) AS distance,"
                " WEBInputs.id ,"
                "WEBUsers_inputs.service as se,"
                "WEBInputs.inputtext as txt,"
                "TAGToughs2.tough  as tough,"
                "WEBInputs.status  as status,"
                "WEBInputs.lang as lang,"
                //"%@" //tagtough.n1
                "TAGToughs2_inputs.input  as input,"
              "TAGBAWords_inputs.magickey as mk,"
                "WEBUsers_inputs.user as us "
                "from "
                    "TAGToughs2 INNER JOIN "
                    "TAGToughs2_inputs ON "
                "TAGToughs2.toughKey=TAGToughs2_inputs.toughKey "
                    "INNER JOIN "
                    "TAGBAWords_inputs ON "
                        "TAGToughs2_inputs.input = TAGBAWords_inputs.input "
                    "INNER JOIN "
                    "WEBUsers_inputs ON "
                "WEBUsers_inputs.input = TAGToughs2_inputs.input "
                    "INNER JOIN "
                    "WEBInputs ON "
                "WEBUsers_inputs.input = WEBInputs.id AND "
                    "WEBInputs.status = \"OK3\" AND "
                    "WEBInputs.lang = \"en\" "
                    "HAVING distance%@ "
                "ORDER BY `distance` ASC LIMIT 100",distance/*,neurons*/,mindist];

  //printf("command %s\n",[command UTF8String]);

            MysqlFetch *fetch = [MysqlFetch fetchWithCommand:command
                                onConnection:connection];

   NSMutableArray *results = [[NSMutableArray alloc] init];
  for (NSDictionary *row in fetch.results) {

     [results addObject:row];
  }

  NSDate *date2 = [NSDate date];
            NSTimeInterval elapsed = [date2 timeIntervalSinceDate:date1];

     //NSLog(@"query in %fs raw results = %d for %@",elapsed,[results count],command);
       // printf("query in %fs raw results = %d ",elapsed,[results count]);

  [pool drain];

  return  results;

}
```

La función anterior crea dinámicante un query MySQL del tipo:

```
select ( POW(0.000000-n48,2) +
POW(0.000000-n35,2) + POW(0.054841-n2,2) +
POW(0.501667-n22,2) + POW(0.000000-n41,2) +
POW(0.000000-n49,2) + POW(0.063755-n3,2) +
POW(0.027839-n17,2) + POW(0.000000-n36,2) +
POW(0.473977-n23,2) + POW(0.020343-n4,2) +
POW(0.000000-n42,2) + POW(0.067343-n10,2) +
POW(0.025003-n18,2) + POW(0.121856-n5,2) +
POW(0.419298-n37,2) + POW(0.000000-n24,2) +
POW(0.068402-n6,2) + POW(0.000000-n43,2) +
POW(0.043936-n11,2) + POW(0.056153-n19,2) +
POW(0.063508-n30,2) + POW(0.037234-n7,2) +
POW(0.022149-n38,2) + POW(0.505241-n25,2) +
POW(0.055592-n8,2) + POW(0.032780-n12,2) +
POW(0.000000-n44,2) + POW(0.231452-n31,2) +
POW(0.000000-n39,2) + POW(0.503655-n9,2) +
POW(0.000000-n50,2) + POW(0.502603-n26,2) +
POW(0.000000-n45,2) + POW(0.508323-n13,2) +
POW(0.452466-n32,2) + POW(0.492140-n27,2) +
POW(0.000000-n46,2) + POW(0.056524-n14,2) +
POW(0.504969-n33,2) + POW(0.494619-n20,2) +
POW(0.000000-n28,2) + POW(0.000000-n47,2) +
POW(0.061133-n15,2) + POW(0.044586-n34,2) +
POW(0.465874-n21,2) + POW(0.221803-n29,2) +
POW(0.000000-n40,2) + POW(1.000000-n1,2) +
POW(0.073000-n16,2) ) AS distance,
WEBInputs.id ,
WEBUsers_inputs.service as se,
WEBInputs.inputtext as txt,
TAGToughs2.tough as tough,
WEBInputs.status as status,
WEBInputs.lang as lang,
TAGToughs2_inputs.input as input,
TAGBAWords_inputs.magickey as mk,
WEBUsers_inputs.user as us
from TAGToughs2 INNER JOIN
TAGToughs2_inputs ON
TAGToughs2.toughKey=TAGToughs2_inputs.
toughKey
INNER JOIN TAGBAWords_inputs ON
TAGToughs2_inputs.input = TAGBAWords_inputs.
input
INNER JOIN WEBUsers_inputs ON
WEBUsers_inputs.input = TAGToughs2_inputs.input
INNER JOIN WEBInputs ON
WEBUsers_inputs.input = WEBInputs.id
AND WEBInputs.status = "OK3"
AND WEBInputs.lang = "en"
HAVING distance<1.5
ORDER BY `distance` ASC LIMIT 100
```

Esta operación MySQL devuelve una lista de los mensajes de tweet capturados cuyo ToughTag tiene una gran similitud con el mismo usado en la búsqueda.

15. Integracion al servicio web

Finalmente el proceso de análsis, indexado y busqueda se ha publicado hacia internet por medio de webservices PHP , como se describió originalmente en los apartados de aquitectura general.

De este manera cualquier front-end puede conectarse a los servicios usando una petición estilo REST y recibir en una respuesta JSON, el toughttag correspondiente, los valores neuronales, y una lista de tweets que corresponden en la forma en que "se sienten" los mensajes más que en el sentido extricto de las palabras qeu contienen, empleando el cerebro antes definido que sintetiza el criterio de "sentido comun".

Así por ejemplo para pedir úicamente la valoración neuronal y el thoughttag correspondiente a un texto usamos el servicio:

http://localhost/deyavoo/getAnalysis.php?&brainName=1&inputText=%@

Donde brainName, es el nombre o id del cerebro que deseamos utlizar, en este caso el cerebro de "sentido comun" e inputText el texto que se desea valorar, por ejemplo para la petición de "all you need is love"

http://localhost/deyavoo/getAnalysis.php?&brainName=1&inputText=all%20you%20need%20is%20love

se obtiene:


```
{
  "lastToughTag":[
    "9",
    "i",
    "j",
    "j",
    "h",
    "g",
    "p",
```

```
    "n",
    "M",
    "l",
    "q",
    "h",
    "K",
    "j",
    "k",
    "k",
    "k",
    "j",
    "j",
    "J",
    "J",
    "K",
    "L",
    "0",
    "J",
    "L",
    "L",
    "0",
    "c",
    "h",
    "f",
    "L",
    "M",
    "j",
    "0",
    "0",
    "P",
    "l",
    "n",
    "f",
    "K",
    "i",
    "0",
    "g"
  ],
  "expandedImpulses":[
    1,
    0.23050311207771301,
    0.22457928955554962,
    0.22656524181365967,
    0.2678392231464386,
    0.27352535724639893,
    0.096449375152587891,
    0.13830873370170593,
    0.55021399259567261,
    0.18171034753322601,
    0.073069773614406586,
    0.25008061528205872,
    0.60220992565155029,
    0.21038348972797394,
    0.20015712082386017,
    0.20640288293361664,
    0.20986662805080414,
    0.21002918481826782,
    0.21272256970405579,
    0.62871819734573364,
    0.6175234317779541,
    0.59961658716201782,
    0.5715523362159729,
    0,
    0.61116421222686768,
    0.58706372976303101,
    0.57294166088104248,
    0,
    0.35993579030036926,
    0.26361939311027527,
    0.29226672649383545,
    0.58420401811599731,
    0.56363970041275024,
    0.21182820200920105,
    0,
    0,
    0.50269538164138794,
    0.18960225582122803,
    0.14985679090023041,
    0.30819922685623169,
    0.5908738374710083,
    0.24261274933815002,
    0,
    0.27025705575942993
  ],
  "alladjsTag":[

  ],
  "textMorphed":[
    [
      "{d}"
    ],
    [
      "{pn}"
    ],
    [
      "need@n",
      "need@v",
      "ne@v"
    ],
    [
      "{av}"
    ],
    [
      "love@v"
```

```
  ]
 ],
 "verbsTag":[
  "love@v"
 ],
 "impulsesByDistance":[
  {
   "name":"love",
   "impulse":1
  },
  {
   "name":"tired",
   "impulse":0.62871819734573364
  },
  {
   "name":"dream",
   "impulse":0.6175234317779541
  },
  {
   "name":"girl",
   "impulse":0.61116421222686768
  },
  {
   "name":"sex",
   "impulse":0.60220992565155029
  },
  {
   "name":"real",
   "impulse":0.59961658716201782
  },
  {
   "name":"care",
   "impulse":0.5908738374710083
  },
  {
   "name":"boy",
   "impulse":0.58706372976303101
  },
  {
   "name":"money",
   "impulse":0.58420401811599731
  },
  {
   "name":"hope",
   "impulse":0.57294166088104248
  },
  {
   "name":"friend",
   "impulse":0.5715523362159729
  },
  {
   "name":"night",
   "impulse":0.56363970041275024
  },
  {
   "name":"beautiful",
   "impulse":0.55021399259567261
  },
  {
   "name":"rich",
   "impulse":0.50269538164138794
  },
  {
   "name":"pro-healt",
   "impulse":0.35993579030036926
  },
  {
   "name":"pain",
   "impulse":0.30819922685623169
  },
  {
   "name":"spirit",
   "impulse":0.29226672649383545
  },
  {
   "name":"lie",
   "impulse":0.27352535724639893
  },
  {
   "name":"dead",
   "impulse":0.27025705575942993
  },
  {
   "name":"honest",
   "impulse":0.2678392231464386
  },
  {
   "name":"hungry",
   "impulse":0.26361939311027527
  },
  {
   "name":"sick",
   "impulse":0.25008061528205872
  },
  {
   "name":"hurt",
   "impulse":0.24261274933815002
  },
  {
   "name":"hate",
   "impulse":0.23050311207771301
  },
  {
   "name":"sad",
```

```
    "impulse":0.22656524181365967
  },
  {
    "name":"happy",
    "impulse":0.22457928955554962
  },
  {
    "name":"young",
    "impulse":0.21272256970405579
  },
  {
    "name":"day",
    "impulse":0.21182820200920105
  },
  {
    "name":"dream",
    "impulse":0.21038348972797394
  },
  {
    "name":"school",
    "impulse":0.21002918481826782
  },
  {
    "name":"party",
    "impulse":0.20986662805080414
  },
  {
    "name":"work",
    "impulse":0.20640288293361664
  },
  {
    "name":"fun",
    "impulse":0.20015712082386017
  },
  {
    "name":"poor",
    "impulse":0.18960225582122803
  },
  {
    "name":"ugly",
    "impulse":0.18171034753322601
  },
  {
    "name":"pleasure",
    "impulse":0.14985679090023041
  },
  {
    "name":"fail",
    "impulse":0.13830873370170593
  },
  {
    "name":"success",
    "impulse":0.09649375152587891
  },
  {
    "name":"health",
    "impulse":0.073069773614406586
  },
  {
    "name":"family",
    "impulse":0
  },
  {
    "name":"lost",
    "impulse":0
  },
  {
    "name":"gay",
    "impulse":0
  },
  {
    "name":"straight",
    "impulse":0
  },
  {
    "name":"life",
    "impulse":0
  }
 ],
 "allverbsTag":[
  "need@v",
  "ne@v",
  "love@v"
 ],
 "queryTime":4.1044859886169434,
 "spheres":{
  "love_pos":0.79030454158782959,
  "health_pos":0.29780265688896179,
  "material_pos":0.47035136818885803,
  "material_neg":0.22773642838001251,
  "sex":0.081872224807739258,
  "love_neg":0.227410167455567322,
  "sex_neg":0.52033770084381104,
  "love":0.56289434432983398,
  "sex_pos":0.60220992565155029,
  "health":-0.0083637535572052002,
  "health_neg":0.30616641044616699,
  "material":0.24261493980884552
 },
 "textMorphedReadable":"{d} {pn} need ne {av} love",
 "nounsTag":[

 ],
 "lang":"en",
```

"adjsTag":[

],
"allNounsTag":[
"need@n"
],
"rawImpulses":[
1.1394942998886108,
0.26265698671340942,
0.25590682029724121,
0.25816980004310608,
0.30520129203796387,
0.31168058514595032,
0.10990351438522339,
0.15760201215744019,
0.62696570158004761,
0.20705790817737579,
0.083262592554092407,
0.28496545553207397,
0.68621480464935303,
0.23973079025745392,
0.22807790338993073,
0.23519492149353027,
0.23914183676242828,
0.239327073097229,
0.24239616096019745,
0.71642082929611206,
0.70366442203521729,
0.68325972557067871,
0.65128064155578613,
0,
0.6964181661605835,
0.66895580291748047,
0.65286374092102051,
0,
0.41014477610588074,
0.30039280652999878,
0.33303627371788025,
0.66569715738296509,
0.64226424694061279,
0.24137704074382782,
0,
0,
0.57281851768493652,
0.21605069935321808,
0.17076095938682556,
0.35119128227233887,
0.67329740524291992,
0.27645584940910339,
0,
0.30795636773109436
]
}

En cambio si lo que se propone es realizar el análisis pero además indexar el texto en la base de datos y luego obtener los resultados de una búsqueda de pensamientos similares, se usa el servicio:

http://localhost/deyavoo/doAnalysisAndCloud.php?service=0&save=yes&userid=%d&brainName=1&inputText=%@

donde se pasa dinámicamente el id del cerebro de análisis, el id de usuarios, el tipo de servicio, y finalmente el texto analiazar.

asi para el caso de "all you need is love"

Se obtiene:

[
{
"service":1,
"userID":7161792,
"distance":3.0401312977462736e-12,
"inputID":963598,
"text":"All you need is love",
"array":[

],
"screen_name":"daniielayazmin"
},
{
"service":1,
"userID":4098265,
"distance":3.0401312977462736e-12,
"inputID":845610,
"text":"Que pienso del amor? All you need is love",
"array":[

],
"screen_name":"shawwr"
},
{
"service":1,
"userID":3265011,
"distance":0.00040251138852909207,
"inputID":954157,
"text":"All you need is love, love is all you need",
"array":[

],
"screen_name":"EdgarLpez"

```
},
{
  "service":1,
  "userID":6486872,
  "distance":0.0015298478538170457,
  "inputID":910998,
  "text":"Love is all you need",
  "array":[

  ],
  "screen_name":"MamaGallina22"
},
{
  "service":1,
  "userID":4530563,
  "distance":0.0015298478538170457,
  "inputID":872348,
  "text":"Love you because i need you!",
  "array":[

  ],
  "screen_name":"deuidedong"
},
{
  "service":1,
  "userID":4915950,
  "distance":0.0015298478538170457,
  "inputID":884762,
  "text":"Love is all you need?",
  "array":[

  ],
  "screen_name":"AbiiAguiirre"
},
{
  "service":1,
  "userID":6593794,
  "distance":0.070084065198898315,
  "inputID":932799,
  "text":"You have to love yourself before u can love somebody else",
  "array":[

  ],
  "screen_name":"MrRic2010"
},
{
  "service":1,
  "userID":8269840,
  "distance":0.07904275506734848,
  "inputID":980404,
  "text":"Lifetime got to love it",
  "array":[

  ],
  "screen_name":"shunhicks"
},
{
  "service":1,
  "userID":3431325,
  "distance":0.081515111029148102,
  "inputID":1047595,
  "text":"love yourself first before you love someone else..",
  "array":[

  ],
  "screen_name":"MizzEima"
},
{
  "service":1,
  "userID":10720075,
  "distance":0.09170977771282196,
  "inputID":1053183,
  "text":"Love the summer",
  "array":[

  ],
  "screen_name":"ice2021"
},
{
  "service":1,
  "userID":6502845,
  "distance":0.092836096882820129,
  "inputID":922916,
  "text":"I love being loved &amp;lt;3",
  "array":[

  ],
  "screen_name":"kelzndem"
},
{
  "service":1,
  "userID":3560120,
  "distance":0.093067042529582977,
  "inputID":819146,
  "text":"Does love really last a lifetime?",
  "array":[

  ],
  "screen_name":"Natashazu71"
},
{
  "service":1,
```

```
  "userID":6498102,
  "distance":0.093766771256923676,
  "inputID":919393,
  "text":"how can you love someone not youself",
  "array":[

  ],
  "screen_name":"alf_ps"
},
{
  "service":1,
  "userID":5574032,
  "distance":0.093766771256923676,
  "inputID":1045968,
 "text":"Anthony Hamilton - I Used to Love Someone",
  "array":[

  ],
  "screen_name":"BaiLeiLue"
},
{
  "service":1,
  "userID":3899867,
  "distance":0.09841369092464447,
  "inputID":834197,
  "text":"Malloww ah :3 ._.V *ihh",
  "array":[

  ],
  "screen_name":"nandaaw_"
},
{
  "service":1,
  "userID":2637436,
  "distance":0.10361642390489578,
  "inputID":897100,
  "text":"You and I both loved....! (8)",
  "array":[

  ],
  "screen_name":"CamilaElias_"
},
{
  "service":1,
  "userID":5102213,
  "distance":0.10701949894428253,
  "inputID":887813,
  "text":"so much love (8)",
  "array":[

  ],
  "screen_name":"javi_astudillo"
},
{
  "service":1,
  "userID":6596455,
  "distance":0.10779112577438354,
  "inputID":967353,
  "text":"I LOVE being a Mommy :)",
  "array":[

  ],
  "screen_name":"MsSheShe_"
},
{
  "service":1,
  "userID":3003760,
  "distance":0.10779112577438354,
  "inputID":786753,
  "text":"I love being around #OOMF",
  "array":[

  ],
  "screen_name":"_FAGGYFRESH_"
},
{
  "service":1,
  "userID":56755,
  "distance":0.10809071362018585,
  "inputID":36047,
  "text":"and she will be loved",
  "array":[

  ],
  "screen_name":"IsadorgaBombom"
},
{
  "service":1,
  "userID":4533203,
  "distance":0.10809071362018585,
  "inputID":874210,
  "text":"be loved you..",
  "array":[

  ],
  "screen_name":"annissa_r14"
},
{
  "service":1,
  "userID":5091145,
  "distance":0.10809071362018585,
  "inputID":887041,
   "text":"RT @gabisgomes_: And she will be loved, and she will be loved ....",
```

```
    “array”:[

    ],
    “screen_name”:”neicelima”
  },
  {
    “service”:1,
    “userID”:7534429,
    “distance”:0.10809071362018585,
    “inputID”:985031,
    “text”:”#MyFavoriteSongsEver - I Wanna Be Loved
|| Eric Benet”,
    “array”:[

    ],
    “screen_name”:”SrryWereClosed_”
  },
  {
    “service”:1,
    “userID”:2637985,
    “distance”:0.10809071362018585,
    “inputID”:824472,
    “text”:”@char_luvs_joe I would of loved to be there!
:( xxxxxxxxxxxx”,
    “array”:[

    ],
    “screen_name”:”itsBronagh”
  }
]
```


16. Aplicaciones Comerciales y Modelo de Negocio. Dejavú

Tal como se planteo originalmente y después de comprobar fa factibilidad técnica de crear un sistema de indexamiento y búsqueda mucho más sensible y cercano a la forma natural de comunicación humana , se puede vislumbrar la aparición de diversos modelos de negocio para la aplicación comercial de esta tecnología.

La primer aproximación es la de patentar mundialmente el algoritmo descrito en esta investigación, mismo que llevaría el nombre clave de algoritmo “dejavú”. Se crearía una empresa centrada en la creación de “cerebros” conformados por “neuronas” diseñadas a la medida para emitir juicios de valor sobre temas específicos.

Así como en esta investigación se propuso y logró sinstetizar el proceso de juicios de valores del “sentido común” através de la implemetaciónde 44 neuronas que describen las esferas más importantes del contexto social humano, es también posible crear cerebros y neuronas que permiten crear juicios cognitivos de los más variados temas.

Para este fin la empresa basada en el algoritmo dejavú, ofrecería un servicio de suscripción y consumo web, con diversos planes de privilegios, donde los suscriptores pueden acceder a cerebros prediseñados, o incluso crear sus propios cerebros a partir de añadir sus propias neuronas a partir de un catalogo de neuronas preentrenadas, o incluso, podrían pedir la creación de neuronas específicas para poder completar las definiciones conceptuales de sus cerebros.

Luego los usuarios podrían consumir los servicios de análsis y match de sus cerebros desde wevicios web usando interfaces REST con JSON o SOAP.

Para agregar valor a la compañía la empresa debería realizar alianzas estrategicas con las redes sociales líderes coo facebook o twitter, para permitir indexar los pensamientos de sus usuarios, y a cambió ofrecer el uso de la tecnología consumida por sus productos.

En este orden de ideas, adicionalmente, los usuarios podrían pagar membresías especiales para poder hacer data mining o consumir información bajo ciertos preceptos de privacidad del aservo de datos proveniente de las bases de datos sociales.

De este modo el corazón del negocio se centraría en la creación de cerebros conceptuales y la indexación de pensamientos de las redes sociales en esos términos.

17. Aplicaciones Inmediatas.Noobem

Como un ejemplo de un servicio que se sirviera , aportara y consimuiera de este modelo, en esta investigación se ha creado un servicio llamado noobem.

Noobem requiere acceder a la base de datos indexada desde twitter para ofrecer a sus usairos la posibilidad de encontrar gente que se “siente como ellos” através de lo que escribe.

Noobem usando el cerebro definido en esta investigación como de “sentido común” devuelve al usuario los tweets que “se sienten” de la manera más similar a lo que ellos escribieron y brinda herramientas para que los usuarios puedan convertirse en followers, persuadiendo a los usuarios con quienese se hizo match, de que el compartir “vibraciones” en sus pensamientos es un excelente pretexto para seguirse mutuamente.

18. Aplicaciones futuras.

Tomando como ejemplo el cerebro de “sentido común” descrito en esta investigación, y el tipo de servicio sugerido por noobem, proyectar el ingreso de datos de manera periódica y continua através de un sistema neuronal paralelo y simultaneo, plantea una serie de emocionantes posibilidades.

Si se considera el texto entrado por el usuario como un flujo constante de sus pensamientos y estados de ánimo, el sistema ofrece una manera nunca antes vista de interpretar la forma de sentir del mismo, y la manera de poder interactuar de forma mucha más humana con el.

Así por ejemplo indexando a lo largo de 2 años usando el cerebro de sentido común aquí descrito, los pensamientos y estados de ánimo de un usuarios promedio, podrían describirse patrones específicos en cada una de las esferas que constitueyn su vida, cubriendo muchos aspectos desde el emocional, salud fisica, bienestar economico, etc.

De esta manera imaginando el sistema conectado a un stream continuo como lo es el newsfeed de facebook, el usuario podría encontrar datos que hicieran match con el en un sentido mucho más personal en cada uno de los aspectos de su vida, muchas veces resumidos en salud, dinero y amor.

Podría por ejemplo ver quienes de sus contactos del sexo opuesto podrían hacer match con su situación emocional en ese momento, o incluso intentar predecir quienes podrían hacerlo en el futuro próximo.

También conocer cuales de sus contactos estan pasando por una buena racha económica para ofrecer algún negocio o quienes podrían estar necesitando urgentemente un prestamo de su parte.

Asimismo el sistema podría decirle quienes de sus contactos estan compartiendo su estado de salud.

Más aun si extrapolamos el uso del sistema completamente integrado atraves de facebook , twitter y cualesquiera otros servicios que surgieran en el interlaps a unos 30 años... el sistema sería capaz de predecir con una precisión imaginable patrones en las diferentes esferas de la vida del usuario, podría además operar como un avatar de el mismo en el sentido de que la construcción de su propio historial de patrones, representa en si mismo un relfejo de sus juicios más personales sobre una amplisima gama de temas y situaciones. Que adicionalmente podría ser vista en relación a la evolución de los propios patrones de sus contactos.

Llega una posibilidad inquietante, si el usuario falleciera, el sistema en si mismo, podría, a partir de los juicios de valor que constituyen su historia compelta de vida, mantenerse activo y continuar opinando y emitiendo juicios de valor respecto a situaciones que siguieran rodeando a su decnedia y amigos, sería una manera de poder ir a platicar con ese ser querido y poder seguir sintiendo su sabiduría con nosotros.

19. Conclusiones

Esta investigación y su prototipo funcional comprueban que es posible crear sistemas de indexamiento mucho más acordes a los contenidos más personales y emotivos que se generan en las redes sociales.

Demuestra además ser una herramienta factible que lleva el nivel de comunicación entra el servicio web y el usuario a un nivel superior, tomando la máquina el papel de un interlocutor humano, capaz de llevar a cabo una conversasión agradable a partir de su acervo de conocimiento limitado, y juzgando un criterio de “sentido comun” socialmente aceptado.

Se anticipa que las aplicaciones en el mundo real de este sistema, tras su mejora y escalamiento podrían significar una importante evolución y el inciio de una nueva era de servicios y productos digitales .