UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Aprendizaje de representaciones vectoriales para lenguas de bajos recursos digitales

TESIS

Que para obtener el título de

Ingeniero en Computación

P R E S E N T A

José Luis Olivares Castillo

DIRECTORA DE TESIS

Dra. María Ximena Gutiérrez Vasques

Ciudad Universitaria, Cd. Mx., 2018

«Si el volumen o el tono de la obra pueden llevar a creer que el autor intentó una suma, apresurarse a señalarle que está ante la tentativa contraria, la de una resta implacable.»

— Julio Cortázar, Rayuela, Cap. 137.

A mis padres, Leticia y José Luis.
A mi tío Ángel.

Agradecimientos

A mis padres, por absolutamente todo.

A la Universidad Nacional Autónoma de México, mi alma mater. A la Facultad de Ingeniería, por brindarme una formación tanto personal como profesional de la cual me siento profundamente orgulloso.

A mi directora de tesis, Dra. María Ximena Gutiérrez Vasques, por brindarme su asesoría, consejos y por enriquecer mi interés por el procesamiento del lenguaje natural. Agradezco todo su apoyo y paciencia a lo largo del desarrollo de este trabajo de tesis.

La realización de esta tesis de licenciatura se llevó a cabo con el apoyo del proyecto CONACyT FC-2016-01-2225 y del proyecto PAPIIT IN403016.

Índice general

Índice de figuras

Índice de tablas

«Authors should use common words to say uncommon things.»

— Arthur Schopenhauer, *The Art of Literature*, 1891.

Capítulo 1

Introducción

En este capítulo se describe la temática, áreas de estudio involucradas en el desarrollo de esta tesis, así como una breve descripción de la problemática que se intenta resolver, objetivos e hipótesis.

1.1. Definición del problema

El procesamiento de lenguaje natural (PLN) o lingüística computacional es una subrama de estudio de la Inteligencia Artificial (IA); el objetivo primordial del PLN consiste en procesar, imitar, interpretar el lenguaje humano mediante modelos computacionales. El crecimiento de esta área en los últimos años ha permitido el desarrollo de diversas tecnologías del lenguaje, por ejemplo, asistentes de voz, sistemas pregunta-respuesta, análisis y procesamiento automático de textos, sistemas de traducción automática, sólo por mencionar algunos.

Una herramienta muy útil dentro del PLN son los corpus textuales, que son colecciones de documentos que permiten estudiar y analizar la estructura de una lengua. En particular para las tareas relacionadas con el procesamiento multilingüe se utilizan grandes colecciones de corpus paralelos, repositorios de documentos donde cada documento tiene su correspondiente traducción en una o más lenguas. Este tipo de recursos son muy valiosos debido a la información léxica bilingüe que se puede obtener de ellos y son la base para obtener modelos estadísticos representativos de las lenguas. Los corpus paralelos permiten construir métodos y/o modelos computacionales que sean capaces de procesar más de una lengua.

En este sentido, la traducción automática es una de las tareas históricamente más rele-

vantes del PLN, por ejemplo, existen modelos de tipo estadístico que son capaces de hacer traducciones a nivel oración (Brown et al., 1991). Un proceso esencial para alimentar a este tipo de modelos es tener corpus paralelos (traducciones) alineados a nivel oración (Gale and Church, 1993). La alineación entre corpus paralelos puede realizarse, también, a diferentes niveles, por ejemplo, nivel documento, oración, frases, palabras. En particular, la alineación a nivel palabra es una tarea estrechamente relacionada con la traducción automática y constituye un área activa de investigación pues no es fácil encontrar automáticamente las correspondencias entre palabras debido a la riqueza de fenómenos lingüísticos que pueden presentar las lenguas, además, no hay suficiente desarrollo en tecnologías del lenguaje para una cantidad considerable de lenguas por lo que muchas de estas no cuentan con sistemas o métodos de traducción automática.

Es importante mencionar que gran parte de los métodos más populares en la actualidad enfocados a tareas como traducción automática y extracción léxica bilingüe necesitan de una cantidad considerablemente elevada de documentos paralelos para funcionar adecuadamente. Algunos de estos métodos consisten en hacer uso de representaciones vectoriales de palabras y el aprendizaje de un mapeo lineal para transformar vectores de una lengua objetivo a otra lengua fuente y así obtener traducciones de palabras.

En esta tesis el principal interés es la extracción léxica bilingüe, la cual consiste en obtener pares de palabras que son traducción a partir de corpus paralelos digitales.

Cuando nos enfrentamos a un escenario en donde las lenguas sólo poseen recursos digitales limitados, es decir, el tamaño de los corpus con los que se trabaja son pequeños, se dificulta la tarea de extracción léxica y, en general, se dificulta el desarrollo de tecnologías para lenguas de bajos recursos digitales (Gutierrez-Vasques, 2015).

Nuestro caso de estudio se enfocará en proponer una metodología que involucre el aprendizaje de un mapeo no lineal entre representaciones vectoriales de palabras mediante una arquitectura neuronal para obtener traducciones a nivel palabra entre dos lenguas, específicamente en casos en donde sólo se tiene disponible una cantidad pequeña de corpus paralelo.

En primera instancia, se realizarán experimentos para el par de lenguas español-náhuatl, siendo el náhuatl la lengua de bajos recursos digitales. Adicionalmente, se pretende evaluar los modelos obtenidos en más pares de lenguas.

1.2. Objetivos

El objetivo de esta tesis de licenciatura es contribuir al área de la inteligencia artificial, específicamente en la subárea del procesamiento de lenguaje natural para lenguas mexica-

nas de bajos recursos digitales. El objetivo particular consiste en proponer un método de aprendizaje de máquina capaz de generar/obtener representaciones vectoriales multilingües para relacionar un par de lenguas y así inducir léxico bilingüe de manera automática. En este trabajo se propone el uso de técnicas populares de aprendizaje de máquina, en particular redes neuronales artificiales que se utilizarán con dos fines, uno de ellos es para el aprendizaje de representaciones vectoriales miltilingües y, por otra parte, para la tarea de extracción léxica bilingüe.

1.3. Hipótesis

Muchos de los métodos actuales para extracción de léxico bilingüe utilizan representaciones vectoriales de palabras, consisten en aprender una transformación lineal que mapea de un espacio vectorial de una lengua fuente al espacio vectorial de la lengua destino con el objetivo de obtener su correspondiente traducción.

Nuestra hipótesis consiste en que un mapeo no lineal es capaz de obtener un mejor desempeño en la tarea de extracción de léxico bilingüe comparado con los métodos actuales, los cuales consisten en aprender un mapeo lineal. Para realizar un mapeo no lineal, utilizaremos métodos de aprendizaje de máquina, en específico, una arquitectura neuronal y el uso de vectores para representar el significado de las palabras y así poder realizar un mapeo no lineal con el fin de extraer léxico bilingüe.

«Los límites de mi lenguaje representan los límites de mi mundo.»

— Ludwig Wittgenstein, *Tractatus Logico-Philosophicus, 1922.*

Capítulo 2

Fundamentos de procesamiento multilingüe

A continuación se hace una introducción sobre los conceptos y áreas de estudio involucradas en el desarrollo de este trabajo y que son necesarios para familiarizar al lector con el tema.

2.1. Procesamiento de lenguaje natural

En 1968, Stanley Kubrick filmó *2001: A Space Odyssey*, una película de ciencia ficción basada en el libro homónimo de Arthur C. Clarke. En esta cinta, uno de los personajes es la supercomputadora llamada *HAL 9000*, la cual está a cargo de la nave espacial *Discovery*, que lleva abordo a varios tripulantes en un viaje hacia la Luna. *HAL 9000* es una IA que se encarga de controlar las funciones de la nave *Discovery* y es, además, capaz de entablar conversaciones de manera natural con los astronautas, es decir, se comunica en lenguaje hablado con los tripulantes de tal manera que para ellos es como platicar con una persona.

Sin embargo, *HAL 9000* por el momento, es más ciencia ficción que una realidad tangible debido a la complejidad inherente de la lengua. Un sistema así es un ejemplo simbólico de una de las principales metas del PLN y de la IA en general. La Figura 2.1 es una fotografía de la IA multimodal HAL 9000.

El PLN es un área de estudio multidisciplinaria, también puede ser conocida como lingüística computacional, y es una rama de la IA. En particular, el PLN es un área de estudio que abarca diversas disciplinas como la lingüística, matemáticas, ciencias de la

Figura 2.1: HAL 9000: Stanley Kubrick, la exposición. Cineteca Nacional. México.

computación, por mencionar algunas. Aunque las metas más ambiciosas del PLN y la IA aún están lejanas y sólo existen dentro de un universo ficticio, no significa que en la actualidad no haya tecnologías del lenguaje de uso comercial y muy utilizadas. Un ejemplo de esto son los sistemas de traducción de Google[1], asistentes de voz como Siri de Apple o Alexa de Amazon, entre otros.

¿Qué es la IA? Existen diversas y muy variadas definiciones, que van desde enfoques científicos hasta filosóficos que tratan de definir lo que es IA; de manera muy general, la IA al igual que el PLN, es una área de estudio multidisciplinario que consiste en comprender, modelar y replicar la inteligencia y procesos cognitivos de los seres vivos (especialmente de

[1]https://translate.google.com/

los humanos) haciendo uso de métodos y principios matemáticos, computacionales, lógicos, mecánicos e incluso biológicos (Frankish and Ramsey, 2014).

La IA y el PLN han estado estrechamente relacionadas a lo largo de las últimas décadas. Por muchos años los enfoques de PLN estuvieron basados principalmente en reglas explícitas lingüísticas que buscaban modelar el lenguaje humano (Lees and Chomsky, 1957). Este primer enfoque se ocupó para hacer sistemas de traducción automática y otras aplicaciones relacionadas con el lenguaje humano.

Además de los sistemas basados en reglas, Weaver (1949), con base en la teoría de los sistemas de comunicación propuesta por Claude Shannon, en conjunto con H. Nyquist R. V. L. Hartley, todos miembros de Laboratorios Telefónicos Bell, sugirieron adaptar esta teoría para sistemas de traducción automática. Sin embargo, este tipo de planteamientos estadísticos se implementarían realmente hasta la época actual en donde la era digital y la capacidad computacional finalmente lo permiten.

Las tendencias actuales han ido abandonado el uso de sistemas basados en una gran cantidad de reglas; hoy en día, gracias al poder de cómputo y de la cantidad de datos de los que podemos disponer, la IA y el PLN utilizan métodos estadísticos, por ejemplo, métodos de aprendizaje automático ya que suelen ser más flexibles que los sistemas basados en reglas.

A grandes rasgos, las principales áreas de estudio que se abordan dentro del PLN se pueden resumir en los siguientes temas: procesamiento de textos, búsqueda de información, traducción automática, interfaces de lenguaje natural, entre muchas otras (Gelbukh, 2002).

2.2. Traducción automática y extracción léxica bilingüe

Dentro del PLN, una tarea muy importante es el estudio y desarrollo de modelos y sistemas para tratar más de una lengua. La extracción léxica bilingüe y la traducción automática son algunas de las aplicaciones multilingües del PLN.

Como ya se mencionó, la extracción léxica bilingüe así como la traducción automática se realizan a partir de grandes corpus, ya sean corpus paralelos (colecciones de documentos con su correspondiente traducción en dos o más lenguas) o corpus monolingües (colecciones de documentos en una sóla lengua), ya que contienen información muy valiosa sobre el comportamiento de la lengua, lo cual resulta muy útil para poder analizar y extraer la relación que existe entre las lenguas así como su comportamiento de manera estadística para el desarrollo de aplicaciones multilingües.

2.2.1. Traducción automática

La traducción automática consiste en convertir, mediante sistemas computacionales, texto o voz de una lengua origen a una o varias lenguas destino y es una de las aplicaciones más importante dentro del PLN; es una tarea que se ha imaginado desde el siglo XVII (Hutchins, 1995) y que sólo a finales del siglo pasado se ha vuelto una tecnología utilizada cotidianamente.

A raíz de la Segunda Guerra Mundial, existió una gran inversión en tecnologías de traducción automática por parte del gobierno norteamericano; sin embargo, estos sistemas que se desarrollaron por décadas (y principalmente basados en reglas) no lograron hacer traducciones de manera automática que fueran más rápidas o sencillas de generar en comparación con las traducciones realizadas por humanos (Council et al., 1966). Incluso desde el punto de vista lingüístico, el concepto de una traducción completamente fiel es problemático, ya que, en palabras de Jorge Luis Borges: «*El original es infiel a la traducción*».[2]

A pesar de que las primeras aproximaciones sobre sistemas de traducción automática consistían en sistemas basados en reglas generadas por lingüístas, después, tomando como inspiración la teoría de la información (Shannon, 2001), se adaptaron modelos estadísticos dando origen a la traducción automática estadística (statistical machine translation SMT por sus siglas en inglés).

Sin embargo, los modelos estadísticos de traducción automática no eran muy populares debido al limitado poder de cómputo con el que se contaba y, además, diversos investigadores abandonaron esta aproximación (Brown et al., 1990).

Uno de los puntos centrales de SMT es ya no depender de un humano que codifique explícitamente las reglas de traducción entre dos lenguas, sino de grandes colecciones de traducciones (alineadas a nivel oración) a partir de las cuales se aprende cuál es la probabilidad de que una frase en la lengua origen se traduzca como cierta frase en la lengua destino. La alineación a nivel palabra (o extracción léxica bilingüe) es un paso esencial para construir este tipo de modelos que estiman las traducciones a nivel oración.

Los modelos estadísticos de SMT aprenden, a partir de un corpus paralelo, una distribución probabilística que permite traducir oraciones entre un par de lenguas. Para lograr lo anterior es necesario estimar primero la probabilidad de traducción a nivel palabra. A esta tarea de encontrar las correspondencias bilingües a nivel palabra, también se le conoce como alineación o extracción léxica bilingüe. Hablaremos de esto en la siguiente sección.

[2] Jorge Luis Borges, *Sobre el 'Vathek' de William Beckford* en *Otras Inquisiciones: Tomo II*, Barcelona, Emecé, 1982 [1952], p. 107-110.

2.2.2. Extracción léxica bilingüe

La alineación a nivel de palabra, también conocida como extracción léxica bilingüe, consiste en obtener de manera automática listas de pares de palabras que son traducciones (Haghighi et al., 2008) (usualmente se realiza a partir de corpus paralelos aunque también puede ser entre corpus monolingües). Esta tarea no es sencilla debido a la gran complejidad de las lenguas y diferencias en sus manifestaciones, por ejemplo, la estructura sintáctica varía de una lengua a otra. En la Figura 2.2 se observa cómo se alinean las palabras de una oración en dos lenguas (polaco-inglés) empleando métodos probabilísticos.

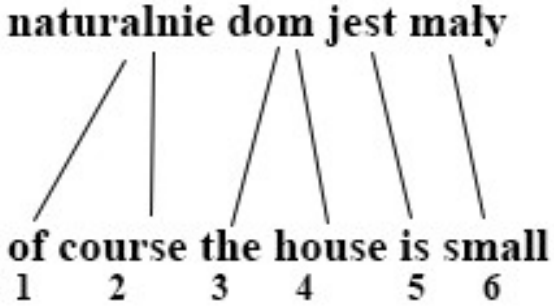

Figura 2.2: Alineación a nivel palabra polaco-inglés. (Tomado de `https://www.wikipedia.org/)`.

El modelo `IBM-1` es un método probabilístico para traducción léxica desarrollado entre las décadas de 1980 y 1990. Se basa en la traducción léxica o a nivel palabra a partir de corpus paralelos, alineados a nivel oración, generando palabras que sean candidatos a traducción con una determinada probabilidad de confidencia dada una palabra a traducir. Este modelo se inicializa con una distribución uniforme que indica que todas las alineaciones de palabras son equiprobables y de manera iterativa estas probabilidades de alineación se van modificando observando el corpus paralelo hasta que el modelo converge, de tal manera que cada palabra en una lengua está alineada a otra(s) palabra(s) en otra lengua con una cierta probabilidad; se obtiene así una lista de candidatos de pares de traducción. El modelo `IBM-1` es un modelo generativo para traducción de oraciones basándose sólo en distribuciones probabilísticas de traducciones de palabras. Cada palabra de una oración a traducir se asocia con una probabilidad de traducción $t(e|f)$ donde e es una palabra que es posible traducción de la palabra f (ver Figura 2.3) (Koehn, 2009).

A partir del modelo `IBM-1`, se desarrollan otros modelos, los cuales son modificaciones del primer modelo que atacan diferentes deficiencias del modelo `IBM-1` como lo son el orden de las palabras que, dependiendo de la lengua, pueden estar en orden diferente o que hay palabras con más de una posible traducción, entre otras variaciones.

das		**Haus**		**ist**		**klein**	
e	$t(e\|f)$	e	$t(e\|f)$	e	$t(e\|f)$	e	$t(e\|f)$
the	0.7	*house*	0.8	*is*	0.8	*small*	0.4
that	0.15	*building*	0.16	*'s*	0.16	*little*	0.4
which	0.075	*home*	0.02	*exists*	0.02	*short*	0.1
who	0.05	*household*	0.015	*has*	0.015	*minor*	0.06
this	0.025	*shell*	0.005	*are*	0.005	*petty*	0.04

Figura 2.3: Tablas de traducción léxica inglés-alemán obtenidas mediante el modelo `IBM-1` (Tomado de Koehn 2009).

El algoritmo de esperanza-maximización es un método de aprendizaje iterativo utilizado en el modelo `IBM-1` con el cual se obtienen las probabilidades de traducción léxica a partir de un corpus alineado a nivel oración, consta de una serie de pasos descritos a continuación:

1. Inicializar el modelo probabilístico con una distribución uniforme.
2. Contabilizar pares de palabras que co-ocurren en las mismas oraciones paralelas
3. A partir de los conteos, actualizar las distribuciones de probabilidad de alineación entre palabras de dos lenguas.
4. Repetir los pasos 1 y 2 hasta que el modelo converja a un valor deseado.

Brown et al. (1993); Koehn (2009) detallan más a fondo el algoritmo con el cual se entrena el modelo `IBM-1` para obtener tablas de traducción léxicas.

Es importante mencionar que los métodos de SMT son altamente dependientes de la cantidad de corpus paralelos para poder capturar las relaciones de traducción entre lenguas, ya sea a nivel palabra u oración. Muchas veces es necesario incorporar conocimiento lingüístico para mejorar la calidad de las traducciones, principalmente si estamos tratando con lenguas muy distantes entre ellas (que no comparten similitudes léxicas, sintácticas, morfológicas, etc.) o si las lenguas no poseen grandes cantidades de documentos paralelos. Algunos ejemplos de conocimiento lingüístico que se pueden utilizar para mejorar la calidad de las traducciones entre dos lenguas son redes semánticas, análisis morfosintáctico automático, aprovechar similitudes entre las lenguas (Canals Marote et al., 2001), incluso tratar de generar una representación intermedia de la semántica (conocida como interlingua).

Gracias a la web, contamos con fácil acceso a grandes corpus paralelos; de igual manera, es posible generar nuevos corpus (orales y/o escritos) a partir de páginas web (Resnik and Smith, 2003). La aplicación de métodos estadísticos sobre estos corpus permite el desarrollo de métodos para traducción automática y extracción de léxico bilingüe.

Además de los modelos IBM antes mencionados, existen diversas técnicas para estimar pares de palabras que son traducción, como el uso de modelos probabilísticos o asociar los pares de palabras mediante medidas de similitud. Últimamente los métodos que han gozado de gran interés en la comunidad de PLN, son aquellos basados en representaciones vectoriales de las palabras y en métodos para relacionar los espacios vectoriales de las dos lenguas, obteniendo así traducciones. Este tipo de representaciones y métodos se profundizarán más adelante en §2.4 y §4 pues son el tipo de métodos que se utilizarán en este trabajo.

Obtener léxico bilingüe de manera automática facilita la generación de diccionarios, que de otra manera serían muy costosos de hacer manualmente, principalmente para lenguas de bajos recursos. También sirven de materia prima para el desarrollo de métodos estadísticos de traducción automática que puedan traducir oraciones completas (Gutierrez-Vasques, 2015).

De manera general, es importante recordar que para realizar una traducción automática *ideal* de palabras u oraciones, se requeriría de un entendimiento profundo del texto y de un buen procesamiento y representación de la lengua (Gelbukh, 2002). Asimismo, de factores pragmáticos culturales y poder procesarlo adecuadamente ya que la precisión del sistema depende de qué tan bueno sea el procesamiento y su representación. Finalmente, «*las lenguas constituyen la visión que un pueblo tiene del mundo, su historia, influencias, conflictos y logros; una experiencia de tal complejidad que no es trasvasable a otra lengua*»[3] por lo que expresiones idiomáticas, multi-palabra, entre otras características únicas de cada lengua, representan algunos de los retos a vencer en la traducción automática para tener modelos que sean tan fiables como un traductor humano.

2.2.3. Lenguas de bajos recursos digitales

En PLN, una lengua se considera de bajos recursos digitales si carece de corpus, de tecnologías digitales y de otras herramientas que faciliten su procesamiento. Lo anterior puede estar ocasionado por un bajo número de hablantes u otras razones de índole política y social que provocan que no haya suficiente difusión de texto.

Un ejemplo de lenguas de bajos recursos son algunas lenguas mexicanas; en México hay

[3]Francisco García Tortosa acerca de la traducción en «*Ulises*» de James Joyce, Madrid, Cátedra, 2016, pág. clxxiv

1 725 620[4] hablantes del náhuatl, mientras que el español es una de las lenguas más habladas en el mundo (aproximadamente 572 millones de hablantes[5]). Además, el náhuatl, carece de suficiente difusión de texto, ya sea en libros o en la web, que permitan generar fácilmente corpus digitales y el desarrollo de tecnologías del lenguaje.

Muchos de los métodos más populares para traducción automática, extracción léxica bilingüe, entre otras aplicaciones del PLN, requieren de grandes cantidades de corpus, como ya se mencionó anteriormente, ya que a partir de que el método observa muchas co-ocurrencias de palabras en oraciones alineadas, logra asignar una probabilidad para determinar traducciones con cierta confidencia. Sin grandes corpus, los métodos estadísticos no logran estimar probabilidades de traducción correctas, ya que dentro del corpus no se observaron suficientes repeticiones de palabras como para asignar una probabilidad con un grado alto de confianza de que una palabra sea traducción de otra palabra en otra lengua. Debido a esto, las tecnologías del lenguaje están reservadas a sólo unas cuantas lenguas, siendo el inglés el idioma en donde más desarrollo se tiene en esta área.

Se estima que en el mundo actuamente se hablan entre 3000 y 5000 lenguas diferentes (Sanz, 2015); gran parte de estas son de bajos recursos digitales lo que se traduce en un escaso o nulo desarrollo de tecnologías del lenguaje. En México conviven 68 agrupaciones lingüísticas con más de 300 variantes.

Es en estas lenguas donde los métodos convencionales de traducción automática y extracción léxica bilingüe suelen fallar, ya que cuentan con corpus muy pequeños o incluso no se tienen corpus para poder utilizar de manera exitosa métodos de traducción automática. Aunado a esto, existen otros obstáculos como la falta de normalización ortográfica, escasos o nulos diccionarios estandarizados, como en el caso del náhuatl (nuestra lengua de estudio), que además cuenta con gran variación dialectal (Gutierrez-Vasques, 2015).

2.3. Representaciones vectoriales de palabras

Es posible capturar propiedades lingüísticas de las palabras y representar estas mediante el uso de vectores. A partir de representaciones vectoriales se obtiene lo que se conoce como modelos de espacios vectoriales en donde palabras, oraciones o documentos se representan mediante vectores dentro de un espacio vectorial con la idea de que texto similar se encuentra en regiones similares dentro de ese espacio vectorial. Este tipo de representaciones son de gran utilidad en tareas como recuperación de la información debido a que se pueden

[4]Náhuatl, lengua indigena más hablada en México: `https://site.inali.gob.mx/Micrositios/estadistica_basica/estadisticas2015/pdf/agrupaciones/nahuatl.pdf`

[5]El español en el mundo 2017: `https://www.cervantes.es/sobre_instituto_cervantes/prensa/2017/noticias/Presentaci%C3%B3n-Anuario-2017.htm`

observar agrupamientos dentro de un espacio vectorial que, dando un ejemplo, si se trata de documentos, se pueden obtener todas las palabras relacionadas con algún tópico en particular ya que estas conservan, de alguna manera, cierta similitud que los caracteriza y que, por lo tanto, comparten ciertas regiones dentro del espacio vectorial (ver Figura 2.4).

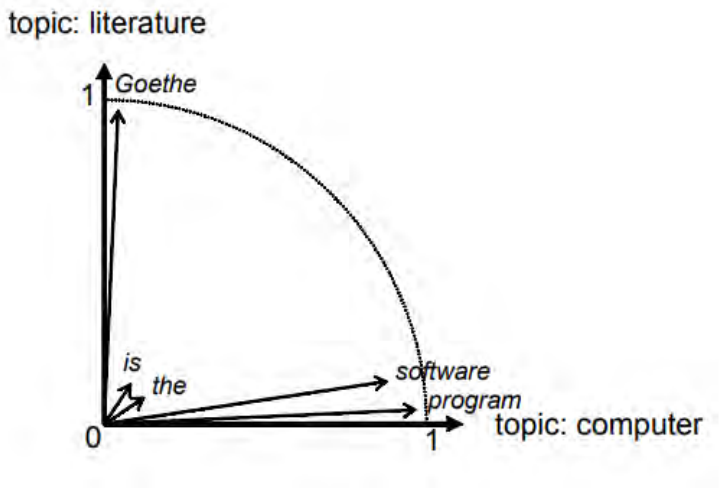

Figura 2.4: Esquema del uso de representaciones vectoriales de palabras. (Tomado de `http://www.kuropka.net/files/TVSM.pdf`).

Los modelos de espacio vectorial representan palabras en un espacio vectorial continuo donde palabras semánticamente similares están cercanas unas de otras. La idea se basa en la hipótesis distribucional (Harris, 1954; Wittgenstein, 2009): palabras que ocurren bajo un contexto similar, tienen significado similar.

Existen diversas técnicas para generar representaciones vectoriales de palabras, en esta tesis se mencionaran sólo algunas.

2.3.1. Word embeddings

Los *word embeddings*, buscan capturar propiedades lingüísticas de las palabras mediante el uso de vectores multidimensionales continuos utilizando modelos neuronales de lenguaje y grandes colecciones de documentos.

Un modelo de lenguaje captura de manera estadística características del comportamiento de una lengua a nivel cuantitativo con el fin de entender y predecir su estructura sintáctica observando la distribución de las palabras dentro de un corpus. Por su parte, un modelo neuronal de lenguaje se basa en una red neuronal que aprende, a partir de texto de en-

trenamiento, la probabilidad de que ocurra una palabra w dada una secuencia de palabras o contexto c, es decir, se estima $P(w|c)$ (Bengio et al., 2003). La capa de salida de la red neuronal contiene las probabilidades de ocurrencia de cada palabra del vocabulario, además, genera representaciones vectoriales de las palabras, las cuales, mediante el entrenamiento de la red neuronal, van adquiriendo propiedades lingüísticas al observar el contexto en el que estas se encuentran.

Las representaciones vectoriales que se obtienen de un modelo neuronal de lenguaje adquirieron importancia con el trabajo de Mikolov et al. (2013c) que, a partir de la capa de salida de la red neuronal del modelo neuronal de lenguaje se obtienen las representaciones distribuidas de las palabras, lo que dio paso al modelo *Word2Vec* (W2V), el cual es un modelo predictivo para el aprendizaje de *word embeddings* a partir grandes corpus.

Existen dos arquitecturas de W2V: *CBOW (Continuous Bag-Of-Words)* y *Skip-gram* (ver Figura 2.5). De manera general, ambas arquitecturas son similares, con la diferencia de que el modelo *CBOW* predice una palabra w dado un contexto c tal que se estima $P(w|c)$, mientras que el modelo *Skip-gram* predice un contexto c dada una palabra w, es decir se obtiene $P(c|w)$.

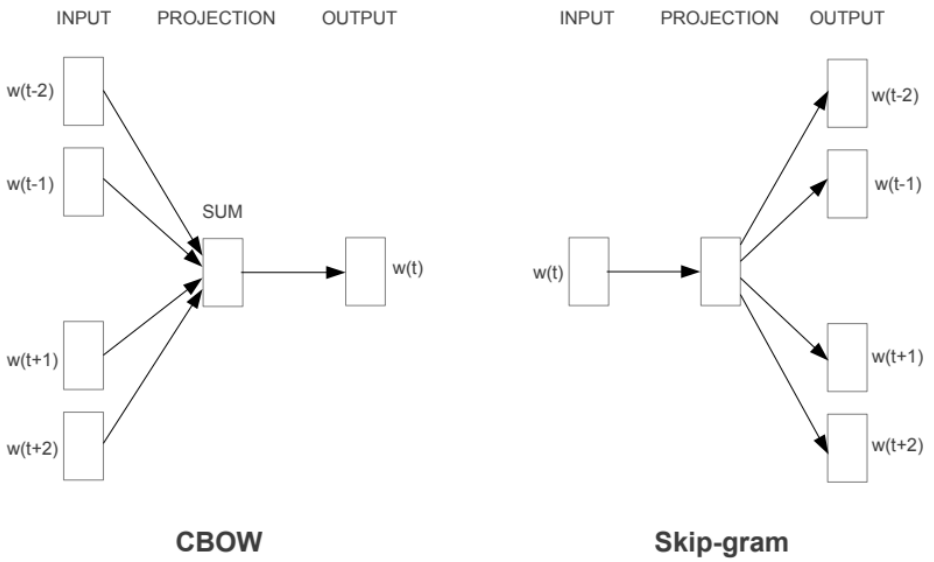

Figura 2.5: Arquitecturas *CBOW* y *Skip-gram*. (Tomado de Mikolov et al., 2013a).

Una de las características más notables de este modelo y principal razón del porqué es tan popular hoy en día, es que se observó que los vectores de palabras generadas con W2V tienen propiedades geométricas tales que se pueden realizar operaciones como sumas o restas sobre ellos para obtener un nuevo vector, el cual contiene propiedades lingüísticas relacionadas con los vectores usados en las operaciones. Un ejemplo de esto es obtener

vectores de palabras mediante el uso de analogías que se obtienen al realizar operaciones vectoriales como $w_{rey} - w_{hombre} + w_{mujer} \cong w_{reina}$ donde w es un vector multidimensional continuo asociado a una palabra. Este tipo de propiedades geométricas que se presentan en los vectores de palabras detonó el uso de *word embeddings* dentro del PLN (ver Figura 2.6).

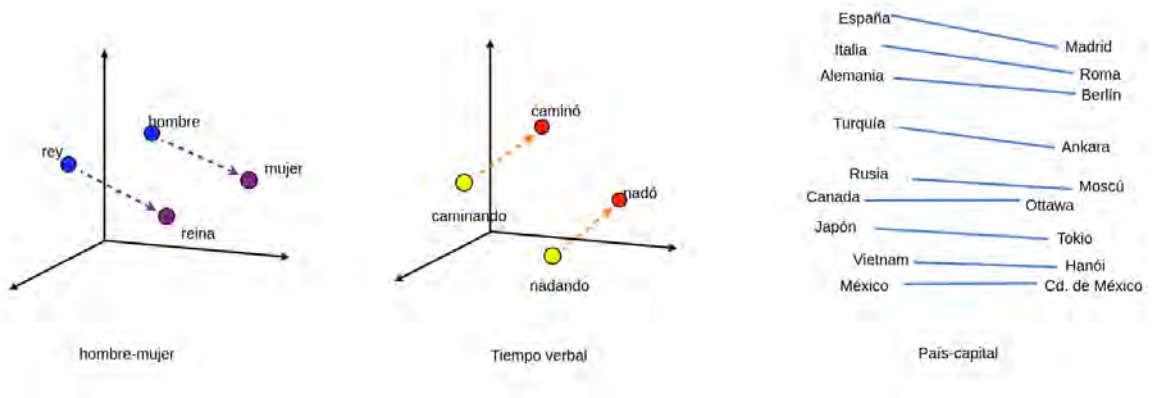

Figura 2.6: Relaciones semánticas capturadas mediante *Word2Vec*.

2.3.2. *fastText*

El modelo *W2V* no toma en cuenta la morfología (estructura interna de las palabras) por lo que sólo puede generar representaciones vectoriales de palabras que observa durante la etapa de entrenamiento, es decir, si en el texto utilizado para generar *word embeddings* no está una palabra w, *W2V* es incapaz de generar un vector para representar a w. Esto es una limitación importante sobre todo en lenguas morfológicamente ricas como el turco o finlandés (Bojanowski et al., 2016) así como el náhuatl que producen muchas formas morfológicamente diferentes.

El modelo *W2V* generaría vectores más ricos, en cuanto a propiedades lingüísticas, para palabras que más veces ocurran dentro de un corpus. Puede haber palabras que estén relacionadas morfológicamente pero que ocurran con mucho menos frecuencia o que no ocurran en el corpus, por ejemplo, *casa*, *casas*, *casita*, etc. Como ya se ha mencionado, si una palabra no aparece dentro del corpus, *W2V* no generará una vector para esa palabra y si aparece pocas veces su representación vectorial no será de buena calidad.

Bojanowski et al. (2016) proponen *fastText*[6], un método desarrollado por Facebook AI Research (FAIR[7]) como una extensión del modelo *Skip-gram* de Mikolov et al. (2013c) para generar representaciones vectoriales de palabras. A diferencia de *W2V*, este método toma en cuenta unidades sub-palabra, es decir, n-gramas de carácteres que forman a una palabra. Esto de alguna manera captura, o contiene, la morfología de la lengua, pues al modelar las palabras como secuencias de caracteres se obtienen vectores de mayor calidad y de una mayor diversidad de palabras.

La idea de *fastText* es representar al vector de una palabra como la suma de los vectores de las sub-palabras que la componen. Una de las principales bondades de *fastText* es que puede generar representaciones de palabras que no observó durante el entrenamiento (palabras fuera del vocabulario (Qin, 2013)) ya que al generar vectores a partir de secuencias de caracteres y al observar las secuencias de caracteres que componen una palabra nueva se puede generar un vector que conserva el significado semántico a partir de sub-palabras que sí existan dentro del corpus de entrenamiento. Las representaciones de tipo *fastText* se han utilizado en diversas tareas, por ejemplo, en la clasificación de textos obteniendo resultados que superan a trabajos anteriores (Joulin et al., 2017, 2016).

Tanto W2V como *fastText* requieren necesariamente grandes corpus monolingües para poder generar representaciones vectoriales de palabras que sean útiles para cada lengua.

[6] https://fastText.cc/
[7] https://research.fb.com/

2.3.3. Representaciones vectoriales multilingües

En la sección anterior hablamos de representaciones vectoriales monolingües que capturan el significado de una palabra en una sola lengua. Sin embargo, también se pueden generar representaciones vectoriales multilingües, esto es, se genera un espacio vectorial compartido en donde los vectores de la lengua fuente están relacionados, o cercanos, a los vectores correspondientes a su traducción en la lengua destino.

A partir de corpus paralelos, principalmente, se pueden obtener representaciones vectoriales multilingües siguiendo la misma idea de *word embeddings*; esta se basa en la hipótesis distribucional (Harris, 1954; Wittgenstein, 2009) pero adaptado a un escenario multilingüe en donde una palabra en un contexto en una lengua fuente tiene su traducción en un contexto similar que está en la lengua objetivo. Existen varios trabajos que se enfocan en representaciones vectoriales multilingües (Klementiev et al., 2012; Al-Rfou et al., 2013; Hermann and Blunsom, 2013, 2014; Faruqui and Dyer, 2014; Lauly et al., 2014; AP et al., 2014).

En este trabajo nos enfocaremos en una representación vectorial multilingüe particular que ha demostrado tener buenos resultados, principalmente en un escenario de bajos recursos donde se tiene un corpus paralelo pequeño. Esta formulación se describirá en la siguiente sección.

2.3.4. Node2Vec

Node2Vec[8] (*N2V*) (Grover and Leskovec, 2016) es un algoritmo para aprender representaciones vectoriales continuas a partir de un grafo (puede ser ponderado); el algoritmo maximiza la probabilidad de observar nodos subsecuentes en un camino aleatorio de longitud fija convirtiendo los nodos en vectores que capturan los patrones sobre cómo estos están interconectados mediante las aristas dentro en un grafo.

Gutierrez-Vasques and Mijangos (2017) observaron que, en lenguas de bajos recursos, las representaciones vectoriales como *W2V* no logran capturar propiedades lingüísticas suficientemente buenas para poder inducir léxico bilingüe, ya que como se mencionó en §2.3.1, se requieren de grandes corpus lingüísticos para obtener buenas representaciones vectoriales.

Debido a esto, proponen obtener representaciones vectoriales bilingües a partir de un grafo español-náhuatl. Este grafo es ponderado, no dirigido, donde cada palabra está representada en un nodo y cada nodo está asociado con otras palabras que son candidatos a traducción con una medida de asociación, esta medida se obtuvo mediante el método estadístico, no supervisado basado en sub-muestreo llamado Anymalign y se representa

[8] `http://snap.stanford.edu/node2vec/`

mediante las aristas del grafo.

Para generar vectores a partir del grafo descrito anteriormente, *N2V* recibe de entrada un grafo ponderado $G = (V, E, \phi)$ donde V es el conjunto de nodos, E es el conjunto de aristas y $\phi : E \to \mathbb{R}$ una función que establece el peso de cada arista.

El objetivo de *N2V* es mapear del grafo a un espacio vectorial $\mathbb{R}^d$, donde d es un parámetro para especificar la dimensión de los vectores que se van a generar. Las representaciones vectoriales *N2V* que se obtuvieron a partir de un grafo ponderado español-náhuatl provocan que las traducciones se acerquen en el espacio vectorial.

Por este motivo, en este trabajo partimos de vectores de palabras N2V español-náhuatl para inducir léxico bilingüe.

2.4. Extracción léxica a partir de representaciones vectoriales

Uno de los métodos más populares para extracción léxica bilingüe la proponen Mikolov et al. (2013b); consiste en aprender una transformación lineal que mapea de un espacio vectorial de una lengua origen a un espacio vectorial de una lengua destino con el objetivo de obtener pares de palabras que son traducción aprovechando las propiedades geométricas de las representaciones vectoriales generadas a partir de grandes corpus para obtener relaciones, hipotéticamente lineales, en el comportamiento de las lenguas y así obtener léxico bilingüe. Una vez proyectados los vectores empleando una transformación lineal, se utiliza una técnica de recuperación de la información para recuperar los vectores más cercanos en la lengua destino, estos son los posibles candidatos a traducción. Para recuperar los vectores candidatos, se emplea una técnica llamada *Precision at K* donde K es un número natural que indica el número de candidatos que tendrá cada palabra (usualmente K es igual a 1, 5 y 10).

De esta manera se puede realizar la extracción de pares de traducción y poder generar así diccionarios o lexicones bilingües.

La aproximación de Mikolov et al. (2013b) se resume al aprendizaje de una matriz de transformación lineal que haga el mapeo entre dos lenguas a partir de un conjunto de entrenamiento (lexicón semilla) y usando representaciones vectoriales de palabras (§2.4.1). Una vez obtenida la matriz de transformación, esta se usa para proyectar un vector en una lengua origen al espacio vectorial de la lengua destino y mediante medidas de similitud como similitud coseno, se pueden calcular los vectores más cercanos al vector proyectado los cuales son posibles candidatos a traducción. La idea del planteamiento original es que,

en un escenario bilingüe, si una palabra ocurre en un contexto en una lengua, la traducción de esa palabra ocurre en un contexto similar en otra lengua, por lo que se pueden obtener pares de traducción utilizando representaciones vectoriales que capturan el contexto en cada lengua.

En la siguiente sección se describe la matriz de transformación propuesta por Mikolov et al. (2013b) para inducir léxico bilingüe.

2.4.1. Matriz de transformación

Si se tiene un conjunto de representaciones vectoriales de entrenamiento en dos lenguas $\{x_i, y_i\}_{i=1}^{n}$ donde $x_i \in \mathbb{R}^{d_1}$ es un vector en la lengua origen y $y_i \in \mathbb{R}^{d_2}$ es un vector que representa la traducción de x_i en una lengua objetivo. A partir del lexicón semilla se debe optimizar una matriz de transformación lineal W de tal modo que se proyecte al vector x_i con su correspondiente traducción y_i (Mikolov et al., 2013b), es decir se convierte en un problema de minimización que se resuelve utilizando el algoritmo del gradiente descendiente estocástico.

Esto es:

$$\min_{W} \quad \sum_{i=1}^{n} ||Wx_i - y_i||^2 \tag{2.1}$$

Una vez aprendida la matriz de transformación lineal W, se utiliza para proyectar vectores de una lengua origen en el espacio vectorial de la lengua destino. En la Figura 2.7 se observa de manera gráfica que las palabras que son traducción se encuentran en regiones similares manteniendo un arreglo geométrico similar, utilizando la matriz W se proyectan los vectores en inglés para obtener su correspondiente vector en español.

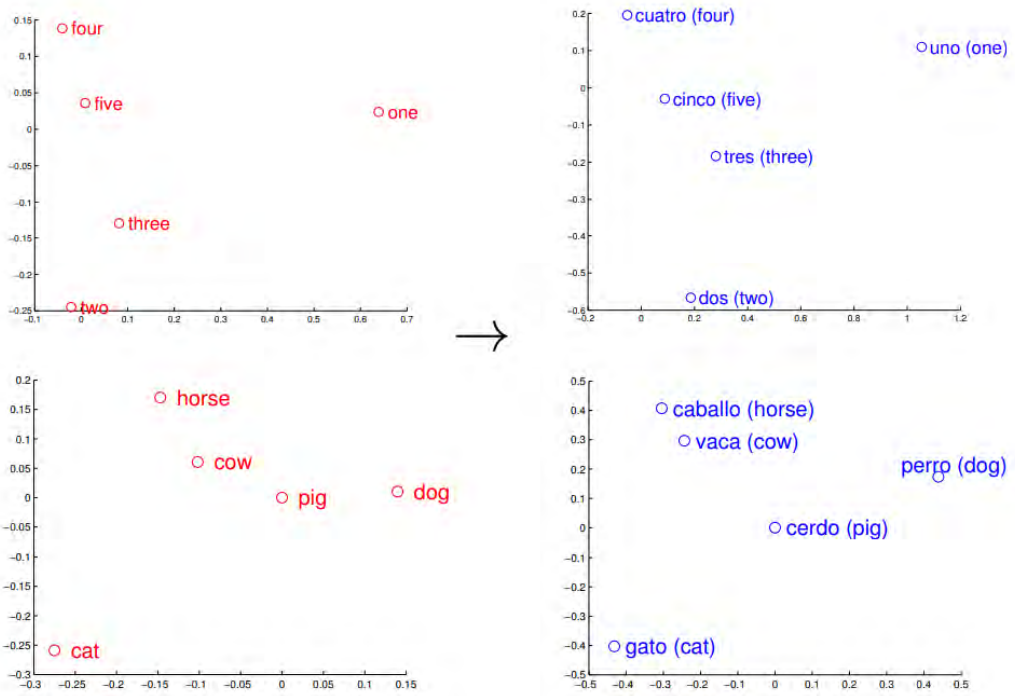

Figura 2.7: Representaciones vectoriales de dos lenguas proyectadas en dos dimensiones utilizando PCA (análisis de componentes principales), inglés a la izquierda español a la derecha. (Tomado de Mikolov et al., 2013b).

2.4.2. Avances recientes

Gran parte de las aproximaciones actuales para esta tarea tienen como base la transformación lineal que explicamos previamente. Las principales modificaciones a la transformación lineal que se proponen involucra modificar las representaciones vectoriales con los que se aprende la matriz de transformación. Algunos autores que realizan aportaciones importantes para esta tarea son Dinu et al. (2014); Shigeto et al. (2015); Lazaridou et al. (2015); Xing et al. (2015); Smith et al. (2017); Artetxe et al. (2016, 2018).

Recientemente, Artetxe et al. (2018) proponen una serie de pasos para aprender la matriz de transformación que involucran normalizar los vectores de entrenamiento, hacer que la matriz sea ortogonal, reducción de dimensionalidad entre otros *artilugios* matemáticos. Por otra parte, Joulin et al. (2018) proponen modificar la función de error J que usualmente es MSE para aprender la matriz de transformación además de aumentar el lexicón semilla con los mejores pares de traducción inferidos durante la etapa de entrenamiento.

Los resultados de las metodologías propuestas por Artetxe et al. (2018); Joulin et al.

(2018), en comparación con la transformación lineal de Mikolov et al. (2013b) se observan en la Tabla 2.1.

	EN-IT	**EN-DE**	**EN-FI**	**EN-ES**
Mikolov et al. (2013b)	34.93 %	35.00 %	25.91 %	27.73 %
Artetxe et al. (2018)(*)	45.27 %	44.13 %	32.94 %	36.60 %
Joulin et al. (2018)	45.3 %	-	-	-

Tabla 2.1: Precisión de diferentes métodos propuestos para la extracción léxico bilingüe en diferentes pares de lenguas. (*) indica que utilizaron como medida de similitud softmax inverso.

Por otro lado, también hay métodos que utilizan representaciones vectoriales miltilingües pero no necesariamente una transformación lineal. Faruqui and Dyer (2014) proponen realizar un análisis de correlación canónica (CCA) que incorpora contexto multilingüe mapeando representaciones vectoriales monolingües de dos lenguas a un espacio compartido donde se maximiza la similitud de palabras que son traducciones entre sí. La tarea de extracción léxico bilingüe mejora usando estas representaciones vectoriales bilingües en comparación con trabajos que emplean representaciones vectoriales monolingües.

Lu et al. (2015) proponen una transformación no lineal partiendo del trabajo de Faruqui and Dyer (2014) pero en lugar de aprender una transformación lineal usando CCA, usan una variación de CCA llamada *deep canonical correlation analysis (DCCA)* para encontrar correlaciones no lineales en los espacios vectoriales de los vectores de palabras.

En esta tesis se plantea utilizar una transformación no lineal mediante una red neuronal para encontrar una función que mapee de una lengua a otra esperando que esta aproximación obtenga mejores resultados que con una transformación lineal particularmente para un escenario de bajos recursos digitales (capítulo 4).

«Propongo considerar la pregunta,
"¿Pueden pensar las máquinas?"»

— Alan Turing, *Computing Machinery and Intelligence, 1950.*

Capítulo 3

Antecedentes de aprendizaje de máquina

En este capítulo se abordarán los conceptos esenciales de aprendizaje de máquina. Principalmente arquitecturas neuronales, pues son los modelos que se ocuparán en esta tesis para aprender una transformación no lineal que relacione los espacios de dos lenguas.

3.1. Aprendizaje de máquina

En ciencias de la computación para la resolución de problemas se recurre al uso de algoritmos. En palabras sencillas, un algoritmo es una secuencia específica de instrucciones finitas para llevar a cabo algún proceso o tarea, este recibe valores de entrada (*inputs*) y genera valores como resultado (*output(s)*) a partir de los *inputs*.

El aprendizaje de máquina, aprendizaje automático o Machine Learning (ML por sus siglas en inglés) es una subárea de la IA. Murphy (2012) define al ML como:

«a set of methods that can automatically detect patterns in data, and then use the uncovered patterns to predict future data, or to perform other kinds of decision making under uncertainty»

En ML se dice que «una computadora aprende a partir de experiencia E con respecto a una tarea T y una medida de rendimiento P, si su rendimiento en tareas T medida por P mejora con la experiencia E» (Mitchell, 1997).

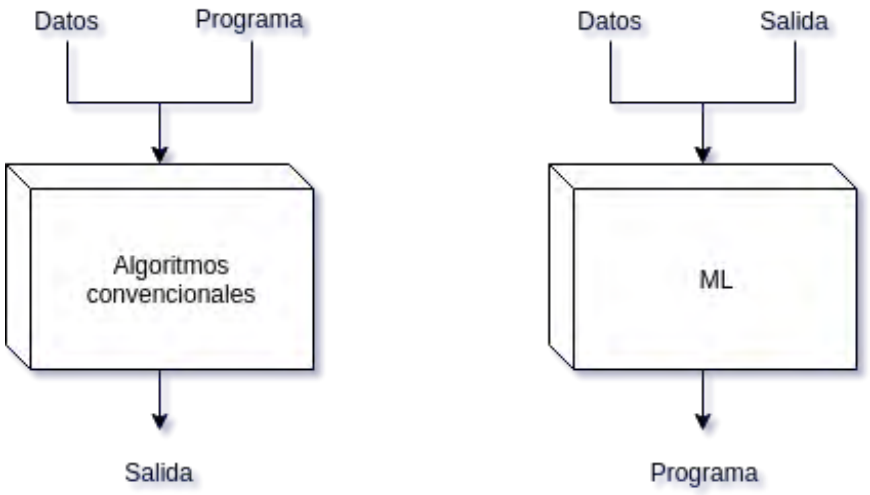

Figura 3.1: Esquema de un algoritmo vs métodos de ML.

Con lo mencionado anteriormente, podemos resumir que, a diferencia de un algoritmo, en los métodos de ML no se especifican las instrucciones a seguir para resolver un problema ya que estos modelos, mediante una etapa conocida como de aprendizaje o de entrenamiento y con respecto a una función de error, deducen por sí mismos la solución a un problema observando información de entrenamiento y con base en conocimiento previo. En la Figura 3.1 se observa la diferencia que existe entre los algoritmos y los modelos de ML.

Cada día se generan 2.5×10^{18} bytes en información (IBM Marketing Cloud, 2017); por minuto se escriben cerca de 455 mil *tweets*, en redes sociales se registran 840 nuevos usuarios, a YouTube se suben 400 horas de video, en Instagram se suben 46,740 millones de fotos, se hacen cerca de 3.6 millones de búsquedas en Google (Schultz, 2017). Toda esta información generada resulta prácticamente intratable si se pretendiera manipular utilizando algoritmos tradiciones, es aquí donde entra ML, el cual se beneficia de las vastas cantidades de información existentes y del poder de procesamiento de cómputo para poder manipularla. Gracias a esto, los métodos de ML han superado en desempeño a los algoritmos tradicionales en casi cualquier área, desde reconocimiento facial, vehículos autónomos, sistemas de recomendación y casi cualquier otra aplicación que se pudiera imaginar.

También dentro del PLN los modelos de ML han ido desplazando a las técnicas tradicionales por las razones ya mencionadas y con resultados similares e incluso mejores en la mayoría de los casos, ya sea en tareas de clasificación de textos, etiquetadores gramaticales, análisis de sentimientos, reconocimiento de voz, traducción automática, por mencionar algunas.

En ML existen diferentes técnicas con las que se enseña o entrena a los modelos para

poder resolver algún problema, estas técnicas se dividen principalmente en:

- Aprendizaje supervisado:
  En el aprendizaje supervisado, se parte de un conjunto de entrenamiento etiquetado, es decir, es necesario indicar la clase a la que pertenece cada instancia de entrenamiento para que el método logre aprender a partir de estos. En §3.1.1 se profundizará más a detalle sobre esta técnica.

- Aprendizaje no supervisado:
  El aprendizaje no supervisado se realiza a partir de datos no etiquetados a diferencia del aprendizaje supervisado, es decir, se tiene un conjunto de datos no etiquetados $\{x_i\}_{i=1}^{n}$ donde $x_i \in \mathbb{R}^d$, lo que significa que se tiene un conjunto de datos *crudos* o sin etiqueta y, por lo tanto, los métodos de ML de aprendizaje no supervisado deben tratar de encontrar patrones en datos no estructurados y la correlación que existe en ellos.

- Aprendizaje por reforzamiento:
  En aprendizaje por reforzamiento, el aprendizaje se da por medio de acciones a ejecutar dentro de un ambiente controlado, el objetivo consiste en elegir un conjunto de acciones que maximicen una función de recompensas (Ng, 2003).

En esta tesis nos enfocaremos principalmente en el aprendizaje supervisado.

3.1.1. Aprendizaje supervisado

El aprendizaje supervisado es la técnica de aprendizaje más popular dentro del ML. Esta técnica requiere de información *etiquetada* para poder aprender a partir de esta y así generar un modelo para la toma de decisiones en información futura o no vista durante la etapa de aprendizaje. La manera en cómo se realiza el aprendizaje supervisado consiste en tener un conjunto de datos de entrenamiento etiquetados $\{x_i, y_i\}_{i=1}^{n}$, donde x_i es un vector de características de las entradas y y_i es un valor correspondiente para cada x_i, ya sean variables categóricas, discretas o continuas. Los modelos de aprendizaje supervisado se utilizan en tareas de clasificación o regresión.

Los métodos de ML para **clasificación** consisten en aprender un mapeo entre las entradas x y las salidas y, donde $y \in \{1, ..., C\}$ y C es el número de clases o instancias. Si $C = 2$ se considera clasificación binaria i.e. $y \in \{0, 1\}$; si $C > 2$ se llama clasificación multiclase. Por otro lado, los métodos de ML para **regresión** son similares a los de clasificación con la diferencia de que para cada entrada x la variable de salida $y \in \mathbb{R}$ (Murphy, 2012).

3.2. Redes Neuronales Artificiales

Dentro de los diferentes métodos de ML se encuentran las redes neuronales artificiales o solamente redes neuronales (NN por sus siglas en inglés) las cuales son modelos que se pueden utilizar en aprendizaje supervisado, no supervisado o por reforzamiento.

Hecht-Nielsen (1989), define a una NN como un sistema informático compuesto por una serie de elementos de procesamiento simples, altamente interconectados, que procesan la información de acuerdo a las entradas externas que recibe.

Las NN están inspiradas en cómo funcionan las conexiones neuronales dentro del cerebro humano y la forma en cómo procesa la información que recibe para realizar una determinada acción. Una red neuronal biológica está constituida por neuronas interconectadas mediante sinápsis. De manera análoga, una NN está conformada por unidades llamadas nodos o neuronas interconectadas mediante *pesos* que corresponden a las sinapsis. Una neurona es la unidad mínima de una NN, su tarea es operar las entradas que recibe y aplicarles una función de activación con la finalidad de determinar si el valor que produjo debe ser propagado o no a las neuronas de la siguiente capa mediante los pesos sinápticos.

En general, las NN cuentan con diferentes propiedades de las cuales podemos sacar provecho, ejemplo de ello es la *no linealidad* que es de suma importancia en aplicaciones como reconocimiento de voz, por decir un ejemplo, además nos permiten obtener el mapeo entre variables de entrada y salida, entre otras muchas cualidades que ofrecen las NN (Csáji, 2001).

En términos de aprendizaje de máquina, si el conjunto de entrenamiento contiene instancias linealmente separables, existe un método de ML que será capaz de generar una superficie de decisión (una línea en el caso de dos dimensiones) para poder discernir entre las diferentes clases; en un conjunto de entrenamiento que no sea linealmente separable, la superficie de decisión es una curva. En secciones siguientes se explicará más a detalle el por qué es necesario que el modelo de ML sea no lineal.

McCulloch and Pitts (1943) proponen la idea de una neurona artificial capaz de aprender y comienzan una era de investigación en NN en donde se destacan los trabajos de Hebb (1949) sobre aprendizaje auto organizado y Rosenblatt (1958) que toma la teoría de McCulloch and Pitts y crea un modelo de aprendizaje supervisado llamado perceptrón.

3.2.1. El perceptrón de Rosenblatt

Un perceptrón es una NN en su estado más simple propuesto por Rosenblatt (1958). Está compuesto solamente de una neurona o nodo con entradas ponderadas mediante pesos *sinápticos* y, de manera opcional, una constante de entrada llamada *bias* o de sesgo; la salida del perceptrón es cuantizada a través de una función delimitadora o de activación, en un perceptrón esta función de activación es típicamente la función signo (ver Figura 3.2). Un perceptrón es también conocido como clasificador binario ya que es capaz de diferenciar entre dos clases distintas de instancias. Sin embargo, para que un perceptrón logre discriminar entre dos clases diferentes de manera satisfactoria, las clases deben ser linealmente separables para que el algoritmo converja a un hiperplano o superficie de decisión, si no son linealmente separables, el perceptrón no logra converger.

El hecho de contar con una sola neurona limita al perceptrón a clasificar sólo entre dos clases, es decir, si se requiere clasificar entre 3 clases distintas el perceptrón debe contener dos neuronas; por lo tanto, un perceptrón con m neuronas es capaz de clasificar entre $m+1$ clases. Como ya se mencionó e independientemente del número de clases a clasificar por el perceptrón, estas deben ser linealmente separables para converger a una superficie de decisión. Esta limitante de que las clases deban ser linealmente separables, convierte al perceptrón, en términos prácticos, en un modelo poco útil[1], sin embargo se debe considerar al perceptrón como un punto de inflexión dentro de la computación y la IA ya que dió inicio a las NN que hoy en día llegan a ser mejores que los humanos en algunas tareas (He et al., 2015; Mnih et al., 2015; Silver et al., 2017a,b; Dodge and Karam, 2017).

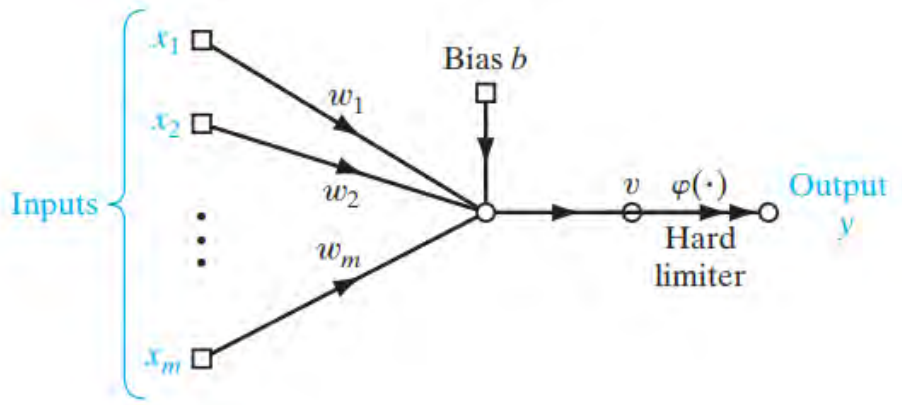

Figura 3.2: Diagrama de un perceptrón. (Tomado de Haykin et al. 2009).

Matemáticamente un perceptrón se modela mediante una combinación lineal de las entradas a las que se le aplica una función delimitadora o de activación, generalmente la

[1]Ver problema XOR: `http://www.mlopt.com/?p=160`

función signo. La neurona realiza una suma ponderada de las entradas, la salida depende de esta suma, ya que será +1 o -1 según el valor que reciba la función signo. Por lo tanto, se puede representar al perceptrón mediante la siguiente expresión:

$$v = \sum_{i=1}^{m} w_i x_i + b \tag{3.1}$$

donde:

- w_i son los pesos sinápticos
- x_i son las entradas
- b es el sesgo o *bias* y
- v es el resultado de la suma ponderada

La función de activación, en este caso la función signo, toma el valor de la suma ponderada y lo convierte a un valor de 1 o -1 tal que:

$$sgn(v) = \begin{cases} 1 & \text{sí } v \geq 0, \\ -1 & \text{en otro caso} \end{cases} \tag{3.2}$$

Por lo tanto, la expresión matemática de un perceptrón queda de la siguiente manera:

$$y = sgn(\sum_{n=1}^{m} w_i x_i + b) \tag{3.3}$$

3.2.1.1. Algoritmo de convergencia del perceptrón

El algoritmo de convergencia, modifica los valores de w_i tal que se genere un hiperplano que divida a dos o más clases. El hiperplano queda expresado por la siguiente ecuación:

$$w_1 x_1 + w_2 x_2 + \cdots + w_n x_n + b = 0$$

Pasos del algoritmo:

- Inicializar w_i; generalmente se inicializa con valores aleatorios
- Añadir bias; para evitar obtener 0 en la combinación lineal ya que está fuera del dominio de la función signo.
- Calcular $y(n)$; Resultado del perceptrón en la iteración n.
- d; Resultado esperado en la iteración n
- Razón de aprendizaje (*learning rate*) : $\eta \in (0, 1)$
- Ajustar pesos; estos se ajustan de acuerdo a la siguiente ecuación:

$$w_i = w_i + \eta[d(n) - y(n)]x(n)$$

donde

$$d(n) = \begin{cases} +1 & \text{sí x(n)} \in \text{clase 1,} \\ \text{-1} & \text{sí x(n)} \in \text{clase 2} \end{cases}$$

- Repetir a partir del paso 3 hasta que el algoritmo converja

Un perceptrón puede consistir en un conjunto neuronas que están interconectadas en una capa o más, este tipo de arquitecturas se conocen como perceptrones multicapa. Las NN se consideran perceptrones multicapa y son capaces de discriminar entre más de dos clases y además pueden discriminarlas incluso cuando estas no son linealmente separables si se les añade una función de activación no lineal. Esta arquitectura se describirá más a detalle en §3.2.3

3.2.2. Funciones de activación

Parte primordial de las NN y razón por la que pueden enfrentarse a problemas no lineales, son las funciones de activación. Una función de activación se denota como $\varphi : \mathbb{R} \to \mathbb{R}$ y define la salida de una neurona. La función de activación debe cumplir con las siguientes suposiciones:

- $\lim_{x \to +\infty} \varphi(x) = a$
- $\lim_{x \to -\infty} \varphi(x) = b$
- $a \neq b$

Las tres principales funciones de activación utilizadas en NN son:

1. *Threshold function:*

$$\varphi(x) := \begin{cases} 1 & x \geq 0, \\ 0 & x < 0 \end{cases}$$

2. *piecewise linear function:*

$$\varphi(x) := \begin{cases} 1 & x \geq +\frac{1}{2}, \\ x + \frac{1}{2} & +\frac{1}{2} \geq x \geq -\frac{1}{2}, \\ 0 & -\frac{1}{2} \geq x \end{cases}$$

3. *Logistic function:*

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

La función de activación logística o sigmoide es la más utilizada en NN y una característica importante de esta es que es derivable en todo su dominio; el rango de las tres funciones antes mencionadas está en $[0, 1]$ (Csáji, 2001).

Otra función de activación igualmente utilizada y que es similar a la sigmoide pero con rango $[-1, 1]$ es la tangente hiperbólica $\varphi(x) = tanh(x)$. Existen diversas funciones de activación, siendo algunas son más útiles que otras dependiendo de la aplicación, el número de capas de la NN, de los datos de entrenamiento, entre otras. Karlik and Olgac (2011) realizaron un análisis interesante para observar el desempeño de una NN utilizando diferentes funciones de activación en NN.

3.2.3. Redes neuronales multicapa

Como ya se ha mencionado, los perceptrones multicapa son NNs (*multilayer perceptron MLP, fully connected feedforward neural networks*), estos se modelan a partir de un perceptrón con neuronas organizadas a manera de capas, de ahí el nombre, es decir, las neuronas de cada capa están conectadas a las de la capa siguiente (Csáji, 2001). A partir de ahora, los términos MLP y NN se utilizarán de igual manera.

Los MLP vienen a contrarrestrar la limitante importante del perceptrón donde este no puede converger a un hiperplano de decisión si los datos que se le presentan no son

linealmente separables como ya se había mencionado en secciones anteriores. El término *feedforward* o *propagación hacia adelante* indica que la información que entra en un MLP sólo fluye de la capa de entreda, pasando por las capas ocultas hacia la capa de salida. Matemáticamente, cada capa oculta es similar a realizar una composición de funciones, por lo que, para un MLP con una capa oculta, $y = f(g(x))$ o $y = (f \circ g)(x)$, mientras que el término *fully connectad* indica que cada neurona de cada capa está conectado a todos las neuronas de la capa siguiente como se observa en la Figura 3.3; si no todas las neuronas están conectadas se le denomina *partially connected* (Csáji, 2001).

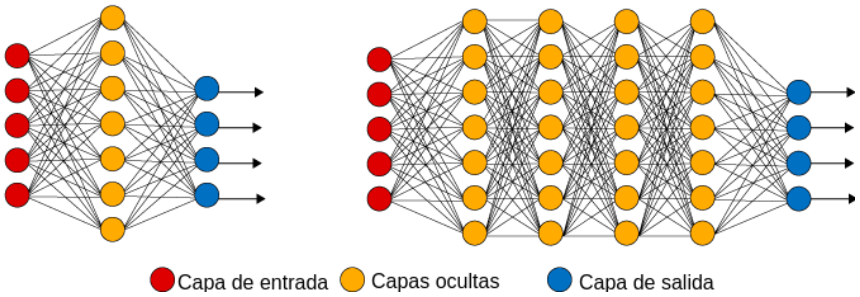

Figura 3.3: Redes neuronales totalmente conectadas.

3.2.4. Teorema universal de aproximación

Kolmogorov (1957) formuló un teorema que demuestra que una función continua de valor real definida en un cubo C n-dimensional puede representarse como sumas y composiciones de funciones de una sola variable; después, Hecht-Nielsen (1987) adaptó este teorema a la neurocomputación en donde establece que cualquier función continua puede ser representada por una NN con una capa oculta de $2n + 1$ neuronas donde n es el tamaño de capa de entrada (Torres, 1994; Stathakis, 2009). De lo anterior se puede resumir que, el **teorema universal de aproximación** afirma que una NN de una capa oculta con un número finito de neuronas y una función de activación arbitraria son aproximadores universales en $C(\mathbb{R}^n)$ (Hornik, 1991).

Una NN es entonces un aproximador universal de funciones, es decir, al entrenar a una NN, esta puede aproximar cualquier función f^*. Si se tiene una función $y = f^*(x)$ que mapea de x a valores de y, una NN define una aproximación $y = f(x; \theta)$ la cual es entrenada para aprender los parámetros θ que mejor aproximen a la función f^* (Goodfellow et al., 2016).

En aprendizaje supervisado, dado un conjunto de entrenamiento etiquetado, se puede modelar una NN para que aproxime una función $y = f(x; \theta)$ y aprenda los parámetros θ que mejor definan un mapeo entre las instancias con su respectiva clase a partir del conjunto de entrenamiento.

Sin embargo, es importante mencionar que no hay un método general para definir el número óptimo de neuronas en la capa oculta de una NN (Torres, 1994).

3.2.5. Aprendizaje de una red neuronal

Lo que aprende una NN a partir de datos de entrada se almacena en los pesos sinápticos w_i (ver Ec. 3.3), en un inicio, los valores de w_i se deben inicializar de manera aleatoria (aunque la manera de inicializar w_i puede variar de acuerdo al tamaño de la capa, la función de activación, etc). Por lo tanto, el aprendizaje de una NN se reduce a obtener $y = f(x;\theta)$ donde θ son los valores de w_i que mejor aproximen a una función f.

Es importante una inicialización aleatoria de los parámetros θ y nunca inicializarlos con un valor igual a cero, esto con el fin de evitar la simetría (*symmetry breaking*) al momento de realizar la etapa de aprendizaje, ya que si las neuronas están todas a cero, entonces todas las capas ocultas generarán los mismos valores para todos los datos de entrenamiento y nunca será capaz de generalizar a partir de los datos de entrenamiento (Ng, 2011).

El principal algoritmo de aprendizaje para entrenar NNs se llama *propagación inversa de error* o *back-propagation* en inglés, introducido por Rumelhart et al. (1986) basado en el algoritmo del *gradiente descendiente* y una función de error o de costo J asociada a la NN.

El aprendizaje de una NN consiste en optimizar una función de error J, en otras palabras, una NN aprende a medida que minimiza la función de error J mediante el algoritmo de *back-propagation*. Los pesos w_i se van modificando durante la etapa de entrenamiento de tal manera que:

$$w_i^{nuevo} := w_i^{anterior} + \eta \Delta w_i$$

donde $\Delta w_i = -\frac{\partial J}{\partial w_i}$, η es la constante de aprendizaje o razón de aprendizaje y J es la función de error asociada a la NN (Torres, 1994).

La función de error J más utilizada en NN es MSE (mínimo error cuadrático), sin embargo, existen muchas otras más y la elección de una función de error J depende tanto de la aplicación, los datos de entrenamiento así como de la arquitectura neuronal.

Por lo tanto, dado un conjunto de datos etiquetados de entrenamiento $\{x_i, y_i\}_{i=1}^n$ y una NN que aproxima el mapeo entre x_i y y_i mediante una función $y = f(x;\theta)$ la función de error J queda expresada de la siguiente manera:

$$J(\theta; x, y) := \frac{1}{N} \sum_{i=1}^{N} ||y_i - x_i||^2 \tag{3.4}$$

De manera iterativa, un ciclo de aprendizaje o época (epoch en inglés) para entrenar una NN consiste en propagar hacia adelante (*feedforward*) los datos de entrenamiento a través de la NN y propagar hacia atrás el error del valor generado por la NN contra el valor esperado (*back-propagation*) mediante el algoritmo del gradiente descendiente; los parámetros θ se van ajustando hasta que estos generen un error muy pequeño con respecto a la función J.

3.2.6. Técnicas de regularización

En ML es común que se presenten problemas como *sobre ajuste* (*overfitting*) o *bajo ajuste* (*underfitting*).

En el caso de *sobre ajuste*, se puede decir que el modelo de ML en lugar de aprender una correlación a partir de un conjunto de entrenamiento etiquetado, este memoriza las instancias de entrenamiento y la clase a la que pertenece, es decir, no es capaz de generalizar el fenómeno observado por lo que, una vez entrenado, si se le presenta al modelo una nueva instancia que no vió en la etapa de entrenamiento, este no es lo suficientemente certero para asignarle una clase o valor, según sea un problema de clasificación o de regresión, ya que quedó *ajustado* a valores que vio durante el entrenamiento.

En el *bajo ajuste* el modelo ni siquiera es capaz de generalizar sobre los mismos datos de entrenamiento y, por lo tanto, menos será capaz de generalizar ante instancias no vistas durante el entrenamiento y asignarles una clase de manera certera.

Se pueden presentar diferentes motivos para que un modelo no logre generalizar un fenómeno a partir de un conjunto de entrenamiento, entre estos motivos están la calidad de los datos de entrenamiento o que tenga ruido, pocos datos de entrenamiento, entre otros. Para evitar estos problemas se recurre al uso de técnicas de regularización. En las siguientes secciones se mencionarán algunas de estas técnicas.

3.2.6.1. Aumentar datos

Aumentar datos o *data augmentation* es una técnica común de regularización que resulta particularmente útil si se está en un escenario de bajos recursos, ya que una misma instancia se puede manipular de tal manera que el modelo la identifique como una completamente

diferente. En términos prácticos, con esta técnica se puede incluso duplicar o más el tamaño del conjunto de datos de entrenamiento. Para imágenes, si se requiere de más datos de entrenamiento, la imagen se puede invertir, rotar, etc., con lo que, si se tiene una imagen, esta se puede convertir en diferentes nuevas imágenes. Aumentar datos es útil sobre todo cuando se tiene un conjunto de entrenamiento limitado y se ha convertido en una técnica común; la biblioteca de NN `Keras`[2] incluso tiene la funcionalidad para aplicar esta técnica de manera sencilla a conjuntos de entrenamiento que sean imágenes.

3.2.6.2. Dropout

Srivastava et al. (2014) introducen esta técnica de regularización, particularmente para NN. El *dropout* consiste en *apagar* cierto porcentaje de las neuronas y sus correspondientes pesos sinápticos de las capas ocultas (regularmente 50 % de las neuronas) durante una época de entrenamiento de manera aleatoria. Esta técnica resulta muy eficaz para reducir el *sobre ajuste* ya que se evita que la NN *memorice* los datos de entrenamiento, permitiendo que para una instancia x_i la NN pruebe varios *caminos* a través de diferentes neuronas y así logre generalizar de mejor manera a nuevas instancias. Con esta técnica se mejora el desempeño de una NN a la vez que se reduce el *sobre ajuste*.

3.3. Autoencoders

Un autoencoder (AE) es una NN que se entrena con el propósito de copiar los valores x_i de entrada a la salida; el aprendizaje se realiza como en una NN convencional como se vió en §3.2.5. La arquitectura de un AE más común cuenta sólo con una capa oculta h de dimensionalidad m, las capas de entrada y salida tienen la misma dimensionalidad n y $n > m$. Las primeras investigaciones sobre AEs fueron realizadas por LeCun (1987); Ballard (1987); Bourlard and Kamp (1988); Baldi and Hornik (1989); Hinton and Zemel (1994) y su principal tarea consistía en realizar reducción de dimensionalidad o extracción de características de manera no supervisada (Goodfellow et al., 2016). Además, un AE puede utilizarse como método de pre-entrenamiento para modelos de aprendizaje profundo (*Deep Learning*).

Un AE es un método de aprendizaje no supervisado que se entrena para tratar de reconstruir los valores de entrada x_i. Los objetivos de un AE, de manera general, son:

- Aprender de manera no supervisada una representación *compacta* de x_i (reducción de dimensionalidad).

[2] `https://keras.io/`

- Obtener mejores representaciones vectoriales de x_i.
- Extracción de las características más relevantes de x_i.

La arquitectura de un AE, de manera general, consta de dos partes (ver Figura 3.4), una parte codificadora (*encoder*) $h = f(x)$ y otra parte decodificadora (*decoder*) que produce una reconstrucción $r = g(h)$, la capa oculta h se llama código o *vector latente* que representa a la entrada x en una dimensionalidad menor (Goodfellow et al., 2016). Es importante señalar que tanto el *encoder* como el *decoder* son simétricos en cuanto al número de capas que los constituyen.

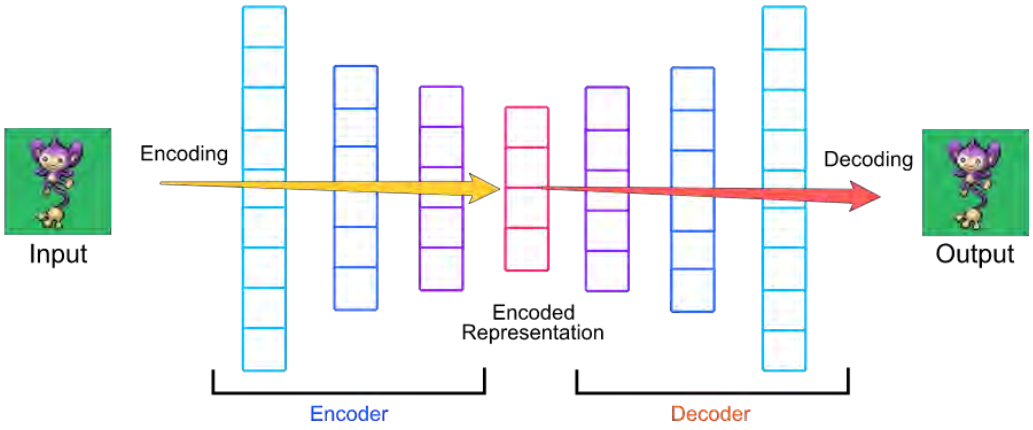

Figura 3.4: Arquitectura de un autoencoder. (Tomado de `https://hackernoon.com`).

Si un AE *copia* las entradas a las salidas de manera exacta, puede significar que el AE no está aprendiendo a reconstruir ni a extraer características relevantes de las entradas, lo que puede indicar que el AE está sobre ajustado como se vio en §3.2.6. Un AE está diseñado para no copiar exactamente las entrada ya que cuenta con ciertas restricciones que se lo impiden, generalmente la principal restricción es que la capa oculta h es de menor dimensionalidad que las capas de entrada y salida para que no simplemente copie, con esta restricción el modelo es forzado para que aprenda una versión *compacta* de las entradas y, por lo tanto, rasgos importantes de estas y así obtener representaciones vectoriales útiles mediante h. Sin embargo, si los datos de entrenamiento son completamente aleatorios, por ejemplo si x_i es ruido Gaussiano, la tarea de compresión del AE sería complicada debido a que la distribución de los datos de entrenamiento implica que no existe una correlación entre los datos, por otro lado, si los datos de entrenamiento x_i están correlacionados y conservan algún tipo de estructura como agrupamientos, etc. el AE es capaz de descubrir las correlaciones de x_i de manera no supervisada (Ng, 2011).

Dado que el AE aprende de manera no supervisada, este trata de aprender una función $g(h(x)) \approx x$, por lo tanto, tenemos que la función de error J (ver ecuación 3.4) a minimizar queda de la siguiente manera:

$$J(\theta; x, g(h(x))) := \frac{1}{N} \sum_{i=1}^{N} ||x_i - g(h(x_i))||^2 \tag{3.5}$$

Un AE con funciones de activación lineales y con la capa oculta h de menor dimensionalidad que las capas de entrada y salida tiende a aprender una reducción de dimensionalidad lineal similar al método PCA (análisis de componentes principales) (Oja, 1982; Ng, 2011; Hinton et al., 2013). Por otra parte, un AE con funciones de activación no lineales puede realizar una reducción de dimensionalidad no lineal y, por lo tanto, suele ser mejor que utilizando PCA (ver Figura 3.5) ya que el AE es capaz de descubrir características no lineales que existan en los datos de entrenamiento y tiene mayor capacidad de capturar aspectos multimodales de la distribución de los datos de entrada (Hinton and Zemel, 1994; Jordan, 2018).

Hay que tener en cuenta que el vector latente de la capa oculta h no es 100 % capaz de lograr buenas compresiones para todas las instancias de entrenamiento. Mediante el proceso de aprendizaje se obtienen buenas compresiones de los datos de entrenamiento y tal vez para los datos de evaluación ya que se espera tengan una distribución similar, pero no logrará comprimir una instancia aleatoria o que tenga una distribución diferente con la que el AE fue entrenado.

Matemáticamente, un AE puede expresarse de la siguiente manera:

Sea x un vector en $\mathbb{R}^d$ y un AE compuesto de una parte llamada *encoder* y otra llamada *decoder*. El *encoder* se expresa como $h : \mathbb{R}^d \to \mathbb{R}^n$ tal que:

$$h(x) := \phi(W_e x + b_e); \quad W_e \in \mathbb{R}^{nxd}, \quad b_e \in \mathbb{R}^n \tag{3.6}$$

W_e y b_e son las matrices de pesos, bias y $\phi : \mathbb{R} \to \mathbb{R}$ una función de activación típicamente no lineal asociados al *encoder*.

Por su parte, el *decoder* toma la salida del *encoder* y la *mapea* de nuevo a $\mathbb{R}^d$. Sea $x_e := h(x)$ y el *decoder* r que reconstruye a x tal que:

$$r(x_e) := \varphi(W_d x_e + b_d); \quad W_d \in \mathbb{R}^{dxn}, \quad b_d \in \mathbb{R}^d \tag{3.7}$$

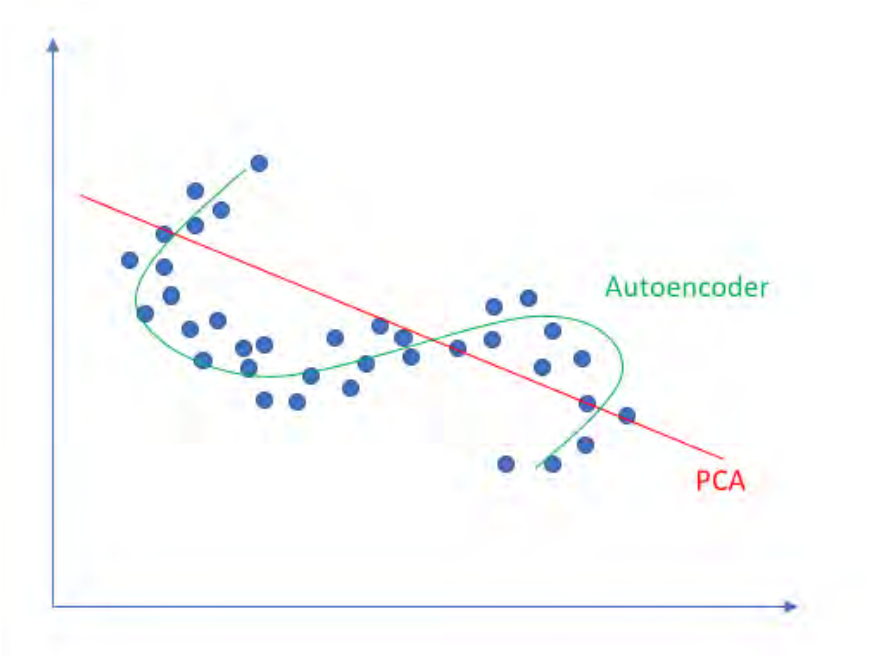

Figura 3.5: Reducción de dimensionalidad lineal (PCA) vs no lineal (AE).

W_d y b_d son las matrices de pesos, bias y $\varphi : \mathbb{R} \to \mathbb{R}$ una función de activación típicamente no lineal asociados al *decoder*.

Por lo tanto, la expresión completa de un AE que reconstruye un vector x quedaría de la siguiente manera:

$$r(h(x)) := \varphi(W_d(\phi(W_e x + b_e)) + b_d) \tag{3.8}$$

φ,W_d, b_d y ϕ,W_e, b_e son las funciones activación, pesos y bias del *decoder* y *encoder* respectivamente. El objetivo de un AE es aprender los parámetros θ (W_d, b_d,W_e, b_e) que reconstruyan a los datos de entrada con un error mínimo de reconstrucción.

Existen diversas variaciones de AEs, desde modelos generativos (*Variational Autoencoders VAE*) hasta modelos que eliminan ruido de las entradas (*Denoising Autoencoders DAE*). Goodfellow et al. (2016); Jordan (2018) describen de manera más profunda las diferentes arquitecturas de AEs que existen y sus diferentes aplicaciones.

En este trabajo nos centraremos en dos arquitecturas principalmente: autoencoders dispersos y autoencoders para eliminación de ruido.

3.3.1. Autoencoder para eliminación de ruido

Un autoencoder para eliminación de ruido (*denoising autoencoder DAE* en inglés) (Vincent et al., 2008) es una versión estocástica de un AE. Se conserva la misma arquitectura aunque la capa oculta h de un DAE no tiene que ser necesariamente menor que las capas de entrada sino que incluso puede ser mayor en cuanto a dimensionalidad. Si la capa oculta h de una AE es de mayor dimensionalidad que las capas de entrada y salida también se le conoce por el nombre de autoencoders sobre completos (*overcomplete autoencoders*).

Una característica importante de un DAE es que recibe como entrada información corrupta $\hat{x}$ la cual debe reconstruir para obtener la versión original de x.

El entrenamiento de un DAE es la misma que para AEs convencionales, con la diferencia de que se añade un paso extra; las entradas pasan por un proceso de corrupción $C(\hat{x}|x)$ el cual añade una distribución aleatoria a x a manera de ruido. El DAE aprende una distribución de reconstrucción $p_{reconstrucción}(x|\hat{x})$ que estima a partir de un conjunto de entrenamiento $(x_i, \hat{x}_i)$ (Goodfellow et al., 2016).

La diferencia con un AE convencional se observa en la función de error J (ver ecuación 3.5). En lugar de tener $J(\theta; x, g(h(x)))$, en un DAE la función de error J queda de la siguiente manera:

$$J(\theta; x, g(h(\hat{x}))) := \frac{1}{N} \sum_{i=1}^{N} ||x_i - g(h(\hat{x}_i))||^2 \tag{3.9}$$

donde $\hat{x}$ es una versión corrupta de x, ya sea que se le añadió ruido o que se eliminaron algunos valores del vector x.

Por lo tanto, un DAE debe, además de reconstruir x, eliminar el ruido con el que se corrompió (Goodfellow et al., 2016). Esto tiene el fin de obtener un modelo más robusto que logra capturar de mejor manera las características de las entradas a la vez que sirve como una técnica de regularización, por lo tanto, la única variación en este tipo de AEs es que se le añade ruido a las entradas y a la salida se sigue conservando los valores originales de x. La función de error J se hace con respecto a la versión corrupta de x contra el valor original de x como se muestra en la ecuación 3.9.

El proceso de eliminar el ruido de las entradas $\hat{x}$ forza al DAE a aprender implícitamente la estructura de los datos de entrenamiento x para evitar que solamente *copie* las entradas a la salida (Bengio et al., 2013; Alain and Bengio, 2014; Goodfellow et al., 2016).

3.3.2. Autoencoders dispersos

Un autoencoder disperso (overcomplete autoencoders, sparse autoencoder en inglés SAE) a diferencia de un AE tradicional, tiene la capa oculta h de mayor dimensión que las capas de entrada (ver Figura 3.6). A primera vista esto podría resultar contraproducente debido a que el AE contiene muchos más parámetros a optimizar además de que se acentúa el problema conocido como la maldición de la dimensionalidad (*the curse of dimensionality*) (Bellman, 1957; Keogh and Mueen, 2017), que implica que, en cuanto la dimensionalidad aumenta, la cantidad de datos de entrenamiento debe incrementar de manera exponencial, de lo contrario los datos de entrenamiento no serán suficientes como para poder extraer información relevante a partir de estos, por lo que se pueden presentar problemas de sobre ajuste o bajo ajuste (ver §3.2.6); adicional a la maldición de la dimensionalidad, se puede presentar también el problema conocido como *hubness problem* (Radovanović et al., 2010; Tomasev et al., 2014; Dinu et al., 2014), que indica que, en cuanto aumenta la dimensionalidad, algunos vectores (hubs) de ese espacio altamente dimensional pueden tener una cantidad considerable de k-vecinos aún cuando estos vectores, tomando como ejemplo un problema de clasificación, sean de diferente clase. Los problemas anteriormente mencionados son inherentes de espacios vectoriales altamente dimensionales.

En un autoencoder disperso SAE, no sólo la capa oculta h es de mayor dimensionalidad, sino que también se puede añadir un término a la función de error J a modo de penalizar el error en la capa oculta h. Partiendo de la ecuación 3.5, en un SAE se añade una penalización de dispersión tal que:

$$J(\theta; x, g(h(x)) + \Omega(h) \tag{3.10}$$

donde $\Omega(h)$ es una penalización de dispersión que restringe a h a sólo activarse en un determinado umbral (Goodfellow et al., 2016).

Sin embargo, el hecho de usar un autoencoder disperso, o SAE, ayuda a obtener un mejor desempeño, sobre todo en tareas como clasificación ya que es capaz de identificar características estadísticas únicas en los datos de entrenamiento. Payan and Montana (2015) utilizan una NN convolucional en conjunto con un SAE con h de mayor dimensionalidad para extraer carácteristicas relevantes de los datos de entrenamiento para la detección de Alzheimer en imágenes de resonancia magnética y Tekin et al. (2016) utilizan una NN convolucional en conjunto con un SAE de igual manera con h de mayor dimensionalidad para aprender una representación de alta dimensionalidad para posiciones humanas en 3D.

En este trabajo nos enfocaremos solamente en que la capa del autoencoder disperso sea de mayor dimensión que las capas de entrada y salida, dejando de lado la penalización $\Omega(h)$

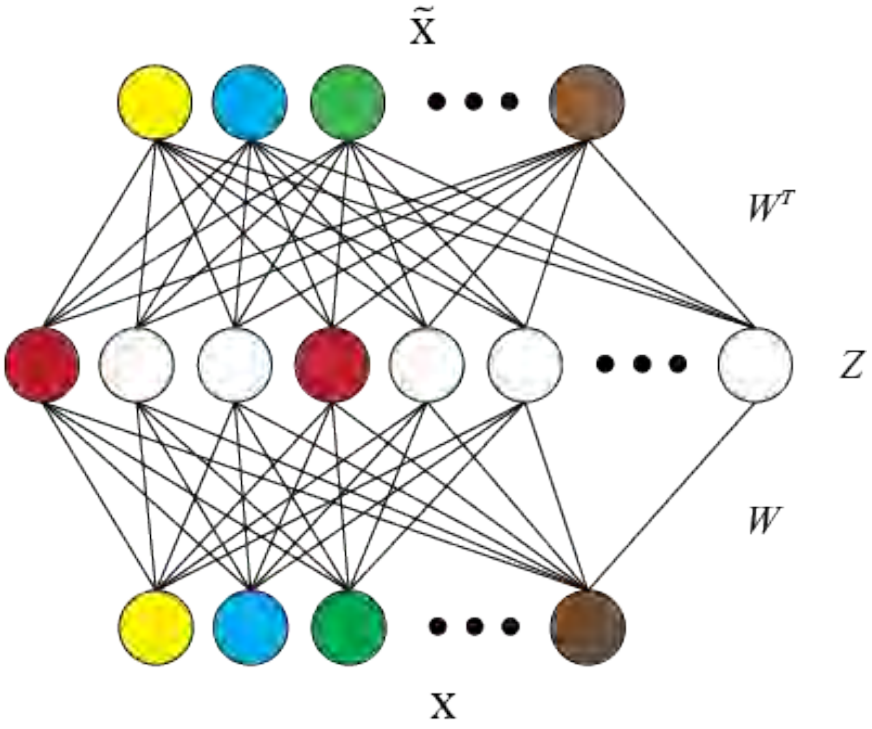

Figura 3.6: Arquitectura de un SAE. (Tomado de `http://www.ericlwilkinson.com`)

de la Ecuación 3.10. Esta arquitectura aplicada al mapeo entre palabras de dos lenguas se abordará a fondo en el el capítulo 4.

En la siguiente sección, se menciona una técnica de regularización que se puede aplicar en AEs.

3.3.3. Atar pesos

Como se mencionó en §3.3, tanto el *encoder* como el *decoder* son simétricos en cuanto al número de capas que contienen. Una manera de reducir el número de parámetros θ de un AE es emplear el *truco* de *atar los pesos* o *weight tying*, es decir, usar la misma matriz pesos del *encoder* en el *decoder* pero transpuesta, de tal manera que: $W_d = W_e^T$, por lo tanto, la Ecuación 3.8 quedaría de la siguiente manera

$$r(h(x)) := \varphi(W_e^T(\phi(W_e x + b_e)) + b_d) \tag{3.11}$$

De esta manera, el AE debe optimizar muchos menos parámetros, haciendo más manejable la etapa de entrenamiento, y de hecho, hacer este *truco* de usar la misma matriz pero transpuesta ayuda como una técnica de regularización para evitar *sobre ajuste* o *bajo ajuste*; una desventaja de aplicar esto *truco* es que puede orillar al AE a aprender la identidad más allá de realmente aprender a extraer las características más relevantes de las entradas. Cabe destacar que no es estrictamente necesario hacer uso de esta técnica en AEs.

3.4. Frameworks de aprendizaje de máquina

En los últimos años, compañías como Facebook, Google, Amazon, entre otras, han desarrollado sus propios frameworks para implementar modelos de ML y los han hecho de código abierto.

Uno de los frameworks más populares de ML es `TensorFlow`[3] (Abadi et al., 2015) el cual es un framework de ML multiplataforma de bajo nivel que permite crear NN de manera sencilla, es uno de los frameworks más populares en `GitHub` en 2017[4], no sólo en ML sino en frameworks en general ya sean de desarrollo web, criptomonedas, etc., también cuenta con el mayor número de contribuidores en `GitHub`.

En `TensorFlow` un modelo de ML se representa mediante grafos computacionales dirigidos en donde los nodos del grafo realizan operaciones típicamente tensoriales a los tensores o matrices que *fluyen* por ellos; un tensor es una entidad algebraica que generaliza los conceptos de escalar, vector, y matriz, por lo tanto, un escalar se considera un `tensor` de orden cero, un vector es un `tensor` de primer orden, una matriz es de segundo orden, etc. Además, cuenta con interfaces de programación de aplicaciones (APIs) para crear NN de manera más sencilla, basta con especificar el tipo de capa que se requiere (fully connected, recurrente, convolucional, etc.) el optimizador, entre otros parámetros, con lo que se puede crear una NN en muy pocas líneas de código.

Otra de las bondades que brinda son los optimizadores ya integrados para realizar *back-propagation* mediante el algoritmo del gradiente descendiente así como sus diversas variaciones; nos permite aplicar *dropout* en una línea de código y lo más destacado es que tiene una herramienta web llamada `TensorBoard` que permite *visualizar el aprendizaje* de NNs (ver figura 3.7), así como su arquitectura en forma de grafo, las distribuciones de los pesos, bias y también permite la visualización de los vectores aún cuando estos sean multidimensionales ya que se pueden aplicar técnicas de reducción de dimensionalidad (PCA, t-SNE) directamente desde `TensorBoard`.

`TensorFlow` está construido sobre el lenguaje C principalmente, y cuenta con diversas APIs para poder usarse en diferentes lenguajes de programación (Java, Go, Python, R, Scala, Javascript). El lenguaje utilizado en esta tesis para usar `TensorFlow` es `Python`[5] en su versión 3.6.3. En este framework de ML se puede hacer uso de unidades gráficas de procesamiento *GPUs* que aceleran el proceso de aprendizaje de NNs, la única restricción en cuanto al uso de GPUs es que sean pertenezcan a `Nvidia`[6]. Independientemente de si se hace uso o no de GPUs es importante destacar que el código para crear NNs, no cambia,

[3] https://www.tensorflow.org/
[4] https://octoverse.github.com/
[5] https://www.python.org/
[6] http://la.nvidia.com/page/home.html

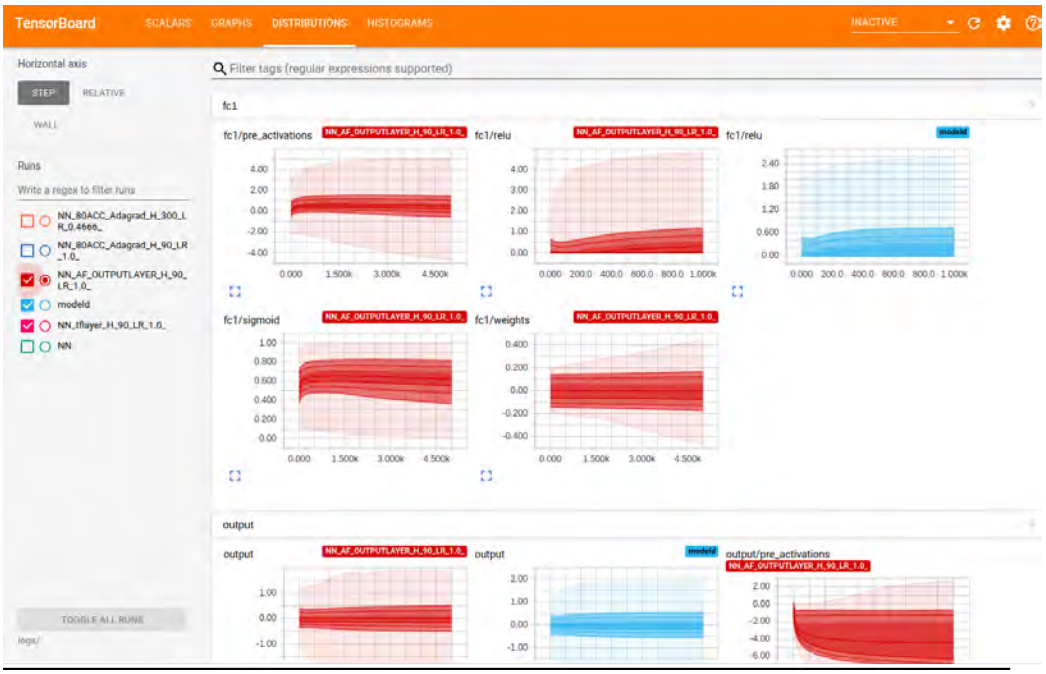

Figura 3.7: TensorBoard.

a menos de que se requiera hacer uso de múltiples GPUs para procesamiento en paralelo, fuera de esto, el código de una NN se mantiene igual ya que, internamente, `TensorFlow` se encarga de manejar tanto el CPU, GPU y la memoria durante el proceso de aprendizaje.

Se pueden crear NNs desde las matrices de pesos y los vectores de sesgo o *bias* y realizar las operaciones matriciales *a mano*, esto permite tener un mayor control sobre los modelos ya que podemos tener acceso directamente a los tensores o se pueden crear los modelos neuronales utilizando las APIs que facilitan crear modelos y dejar que `TensorFlow` se encargue de manipular la NN durante la etapa de aprendizaje.

«When I look at an article in Russian, I say: "This is really written in English, but it has been coded in some strange symbols. I will now proceed to decode."»

— Warren Weaver, *Translation, 1949.*

Capítulo 4

Autoencoder para extracción léxica bilingüe

En este capítulo se describirá la arquitectura, diseño e implementación de la red neuronal propuesta, el corpus español-náhuatl de experimentación y las representaciones vectoriales utilizadas para la tarea de extracción léxica bilingüe.

4.1. Arquitectura neuronal propuesta

Para extraer léxico bilingüe, proponemos una red neuronal, en particular un autoencoder. De manera más específica, partimos de dos tipos de autoencoders: autoencoders para eliminación de ruido (DAE) y autoencoders dispersos (SAE).

En esta tesis nuestra hipótesis es que un mapeo no lineal es capaz de superar en desempeño a un mapeo lineal para inducir léxico bilingüe; para esto, proponemos una arquitectura neuronal bilingüe que combina características de los autoencoders para eliminación de ruido y autoencoders dispersos para hacer un mapeo no lineal entre las dos lenguas. Del DAE adoptamos la noción del proceso de corrupción que se le aplica a los vectores de entrada (ver §3.3.1), de tal manera que, en nuestro caso, consideramos que los vectores en español son una versión corrupta de los vectores en náhuatl, es decir, el proceso de corrupción por el que pasan los vectores en un DAE lo damos ya por hecho y a partir de las versiones corruptas, la arquitectura neuronal bilingüe propuesta debe reconstruir los vectores corruptos de entrada para eliminar el *ruido* que contengan y así obtener los vectores originales los cuales serán su traducción en náhuatl; además, del SAE tomamos la característica de que la capa oculta h es de mayor dimensionalidad que las capas de entrada y salida por las razones mencionadas

en §3.3.2.

La arquitectura neuronal bilingüe cuenta con una capa oculta h de mayor dimensionalidad que las capas de entrada y salida con el fin de eliminar el *ruido* de los vectores en español que son la entrada del modelo y así reconstruir el vector original en la salida de la arquitectura el cual será su traducción en náhuatl. Esto constituye la arquitectura neuronal propuesta en esta tesis tal como se muestra en la Figura 4.1.

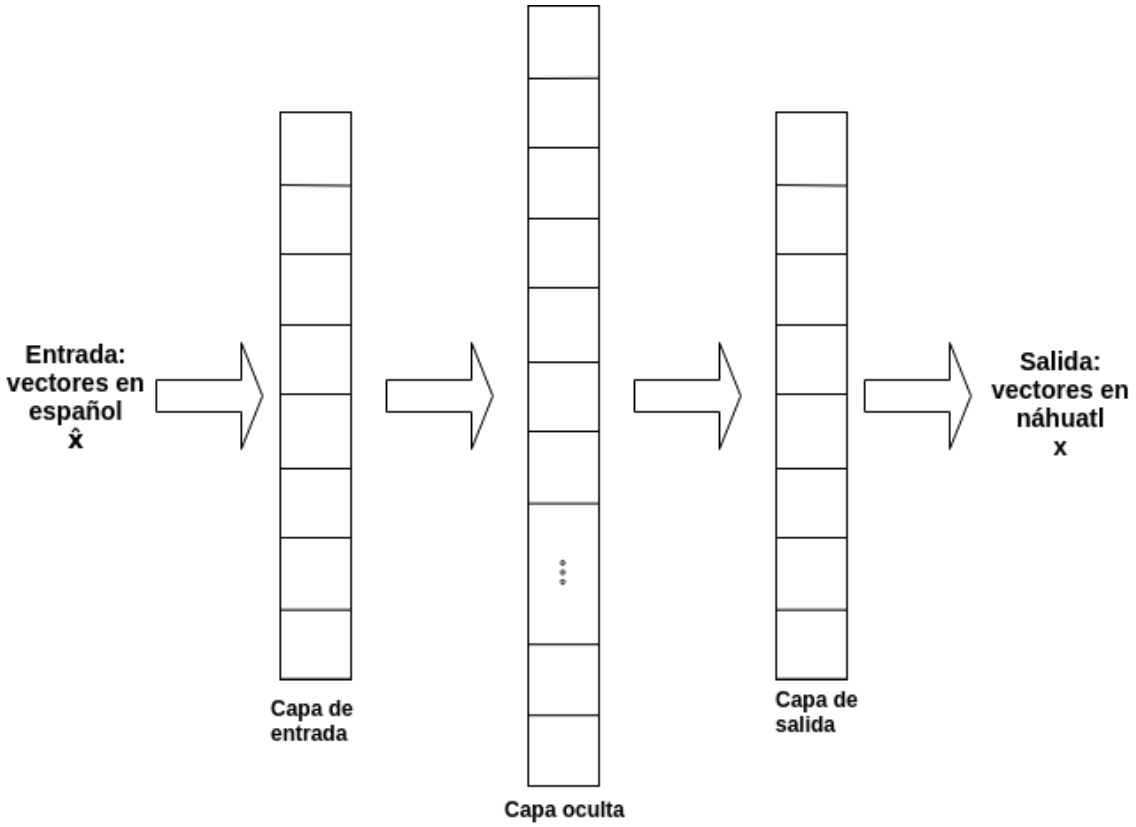

Figura 4.1: Arquitectura neuronal bilingüe.

La metodología abarca, de manera general, la construcción de un autoencoder enfocado en el aprendizaje de representaciones vectoriales multilingües y para la tarea de extracción lexica bilingüe.

Se partirá de las representaciones vectoriales de Gutierrez-Vasques and Mijangos (2017), las cuales fueron obtenidas a partir de un corpus paralelo español-náhuatl (Gutierrez-Vasques et al., 2016) el cual se describirá en §4.2 y el algoritmo node2vec ($N2V$) (Grover and Leskovec, 2016) descrito en §2.3.4 para generar los vectores de palabras. Decidimos

utilizar estas representaciones vectoriales bilingües pues se ha probado que logran capturar mejor las relaciones semánticas bilingües en un escenario de bajos recursos, a comparación con los vectores *W2V* entrenados sobre corpus monolingües.

Se construirá un modelo de aprendizaje supervisado, en particular una arquitectura neuronal bilingüe con las siguientes características:

- Se planteará una arquitectura neuronal, en particular una variación de *autoencoder* que aprenda, mediante un conjunto de entrenamiento, a hacer una transformación no lineal entre un vector de entrada y uno de salida. El conjunto de entrenamiento corresponde a un lexicón semilla que está formado por pares correctos de traducción español-náhuatl.
- La arquitectura se alimenta con vectores *N2V* español-náhuatl, en principio existe un vector asociado para cada palabra de español y de náhuatl del corpus paralelo.
- Una vez entrenada la arquitectura, el modelo será capaz de reconstruir para cada palabra su representación en la otra lengua. Es decir, si el modelo recibe como entrada la representación vectorial de una palabra en español, este generará su correspondiente representación vectorial la cual será su traducción en náhuatl. Finalmente, por cada vector proyectado del español, el sistema recuperará una lista de 10 candidatos a traducción más cercanos. Esto se hace calculando la similitud coseno entre el vector proyectado y todos los vectores que viven en espacio vectorial del náhuatl.
- Para estimar el desempeño de nuestro modelo de aprendizaje se utiliza un conjunto o lexicón de evaluación que contiene pares de traducción correctas y que no se vieron durante el entrenamiento.
- Para la implementación de la arquitectura neuronal bilingüe, se utilizará el framework de aprendizaje de máquina `TensorFlow` (Abadi et al., 2015).

4.2. Corpus de experimentación

Los corpus son recursos muy valiosos en PLN, en particular los corpus paralelos ya que son muy útiles para realizar extracción de léxico bilingüe y sistemas de traducción automática. Para los experimentos realizados en esta tesis se utilizó como base el corpus paralelo español-náhuatl `Axolotl`[1] (Gutierrez-Vasques et al., 2016).

`Axolotl` consta de 33 libros traducidos del español al náhuatl o viceversa cubriendo diferentes temáticas. El tamaño total de `Axolotl` es alrededor de 1 186 662 tokens tanto en

[1] `http://www.corpus.unam.mx/axolotl`

español como en náhuatl. Sin embargo, el náhuatl carece de un sólo tipo de normalización ortográfica, tiene gran variación dialectal y, por lo tanto, los textos de este corpus son poco uniformes. En vista de lo anterior, para propósito de esta tesis, sólo se utiliza un subconjunto de documentos que tienen cierta consistencia ortográfica y dialectal. Este subconjunto procesado de documentos es el mismo utilizado por Gutierrez-Vasques and Mijangos (2017) en sus experimentos de extracción léxica bilingüe. En la Tabla 4.1 se indica el tamaño del corpus utilizado para la experimentación.

Lengua	Tokens	Tipos	Oraciones
Español (ES)	118364	13233	5852
Náhuatl (NA)	81850	21207	5852

Tabla 4.1: Tamaño del corpus paralelo

4.3. Representaciones vectoriales Node2Vec español-náhuatl

Como ya se ha mencionado, las representaciones vectoriales más populares en PLN son las representaciones distribuidas *Word2Vec*, descritas en §2.3.1; estas explotan las propiedades lingüísticas de las palabras a partir de corpus muy grandes, sin embargo, en escenarios de bajos recursos las representaciones vectoriales obtenidas mediante *W2V* no logran capturar gran parte del significado lingüístico de las palabras obteniendo un muy bajo desempeño en la extracción de léxico bilingüe como se observa en el trabajo de Gutierrez-Vasques and Mijangos (2017). Por este motivo, se decidió trabajar con las representaciones vectoriales multilingües *Node2Vec* descritas en §2.3.4 ya que obtienen mejores resultados en entornos de bajos recursos comparados con vectores *W2V*.

A continuación se describen las características de las representaciones vectoriales *N2V* español-náhuatl.

Los vectores *N2V* tanto en español como en náhuatl están en un espacio vectorial $\mathbb{R}^{128}$; en total, se cuenta con 4433 vectores en español y 1904 en náhuatl. El lexicón semilla utilizado para entrenamiento de la arquitectura neuronal bilingüe consta de 496 pares correctos de traducción español-náhuatl, mientras que el lexicón de evaluación para medir el desempeño del modelo propuesto consta de 140 pares correctos de traducción. Es importante mencionar que en el lexicón de evaluación hay palabras en español con más de una posible traducción correcta, como se muestra en la Tabla 4.2. En el lexicón de evaluación no se incluyen palabras que se encuentren dentro del lexicón semilla de entrenamiento.

Español	**Náhuatl**
traer	itqui
traer	huica
traer	huiqui
traer	cui
ciudad	altpe
ciudad	altepe

Tabla 4.2: Palabra en español con más de una posible traducción en náhuatl.

4.3.1. Preprocesamiento/Normalización de los vectores español-náhuatl

En ML, una práctica común consiste en normalizar el conjunto de entrenamiento ya que facilita que los modelos logren converger a una solución en donde el error es el mínimo (de manera ideal, converger a un mínimo global) de manera más rápida y eficiente.

Como parte de nuestra metodología, a los vectores *N2V* español-náhuatl se les aplica dos tipos de normalización antes de iniciar la etapa de aprendizaje del modelo que proponemos como se explicará más adelante.

Se normalizan los lexicones obteniendo sus promedios centrales y normalizando su varianza. Las normalizaciones que se aplicarán a los vectores *N2V* español-náhuatl han demostrado dar buenos resultados en tareas de extracción léxica bilingüe (Artetxe et al., 2016, 2018).

A continuación se describe brevemente cómo se realizaron las normalizaciones a los vectores *N2V* español-náhuatl

Promedio central

Sea un conjunto $X = \{x_i\}_{i=0}^{N}$ en $\mathbb{R}^d$, se obtiene el promedio de X de la siguiente manera:

$$\mu := \frac{1}{N} \sum_{i=0}^{N} x_i \tag{4.1}$$

Y a cada instancia x_i se le resta μ para que el conjunto de entrenamiento tenga un

promedio muy cercano a cero, quedando de la siguiente manera:

$$X := X - \mu \tag{4.2}$$

Normalizar varianza

Sea un conjunto de entrenamiento $X = \{x_i\}_{i=0}^{N}$ en $\mathbb{R}^d$, se obtiene la varianza de X:

$$\sigma^2 := \frac{1}{N} \sum_{i=0}^{N} x_i^2 \tag{4.3}$$

Y cada x_i se divide entre σ^2 tal que:

$$X := \frac{X}{\sigma^2} \tag{4.4}$$

Una vez que se normalizan los lexicones, en la siguiente sección se describirá el diseño e implementación del modelo que proponemos en esta tesis.

4.4. Diseño e implementación de la arquitectura neuronal bilingüe

El modelo propuesto en concreto es una variación de autoencoders para eliminación de ruido y autoencoders dispersos como se mencionó en §4.1.

La arquitectura se alimenta de vectores *N2V* español-náhuatl de 128 dimensiones y, como ya se ha mencionado, la arquitectura neuronal bilingüe tendrá una capa oculta h mayor a 128 en todos los experimentos a realizar; además, dado que en un DAE se considera un proceso de corrupción en los vectores de entrada $C(x|\hat{x})$ podemos tomar en cuenta que los vectores en español son una versión corrupta de los vectores en náhuatl, donde x es el vector en náhuatl y $\hat{x}$ es el vector en español.

Además del proceso de corrupción C descrito anteriormente, se ha demostrado que aumentar la dimensionalidad de la capa oculta h en autoencoders dispersos ayuda a encontrar características importantes de los vectores de entrada como se puede observar en el trabajo de Payan and Montana (2015); Tekin et al. (2016), por lo que nuestra arquitectura neuronal bilingüe de igual manera tiene una capa oculta h mayor a las capas de entrada y salida que es de 128 dimensiones para todos los experimentos a realizar.

En algunos experimentos se utiliza la técnica de *atar los pesos* (ver §3.3.3) y también se aumentan datos (§3.2.6.1). Como ya se ha mencionado, aumentar datos es útil sobre todo cuando se cuenta con un conjunto de entrenamiento limitado. La forma en cómo nosotros aumentamos datos consistió en invertir cada representación vectorial, es decir, si se tiene un vector $x_0 = [0, 1, ..., n]$ este se invierte de tal manera que ahora tenemos un nuevo vector $x_1 = [n, ..., 1, 0]$, esta idea es similar a la de invertir, rotar, etc. imágenes, como se mencionó en §3.2.6.1. Es importante señalar que no se invierte la palabra, es decir, no se hace que una palabra $p_0 =$ "*traer*" se invierta para obtener una nueva $p_1 =$ "*reart*", lo que se invierte es la representación vectorial de una palabra.

La entrada de la arquitectura serán vectores *N2V* en español los cuales consideramos como una versión corrupta de los vectores en náhuatl y que serán expandidos a una dimensión mayor a 128 con el objetivo de que el modelo logre observar características únicas en los vectores, en la capa de salida esos vectores expandidos vuelven a su dimensión original pero reconstruidos, sin corrupción, obteniendo así su correspondiente representación vectorial pero en náhuatl.

Para construir el modelo que proponemos, utilizamos el framework de aprendizaje automático `TensorFlow` (Abadi et al., 2015) en su versión 1.6 descrito en §3.4 y utilizando el lenguaje de programación de `Python` en su versión 3.6.3. Aprovechando que `TensorFlow` está optimizado para utilizar tarjetas gráficas (GPUs), en nuestros experimentos haremos uso de GPUs en particular de la tarjeta `Nvidia Tesla K80`[2] para acelerar el aprendizaje de la arquitectura neuronal bilingüe propuesta.

Los modelos propuestos comparten la característica de tener una sola capa oculta h, además, se plantea un modelo más profundo con 3 capas ocultas. En cualquier caso, la dimensión de las capas ocultas de los modelos propuestos siempre es mayor a 128 que es la dimensión de los vectores *N2V* español-náhuatl. Para los experimentos se crean diferentes modelos los cuales serán descritos a mayor profundidad en el capítulo 4.5.

La arquitectura neuronal bilingüe tiene una capa de entrada con 128 neuronas, una capa oculta h con m neuronas donde $m > 128$. Se prueban distintas funciones de activación, principalmente las funciones de activación *Elu* (Clevert et al., 2015) y $Tanh$; además se pruebas diferentes optimizadores con diferente número de épocas.

En los experimentos, la inicialización de los tensores correspondientes a los pesos se hace de manera aleatoria por las razones mencionadas en §3.2.5, con una distribución normal truncada y desviación estándar de 0,1. La razón por la que se elige esta inicialización es más por convención que por otros motivos; se pretende que usando este tipo de distribución se eviten problemas relacionados con el *desvanecimiento* o *explotación* de los gradientes. Básicamente, se pretende evitar que los gradientes al momento de retro propagar el error

[2] `https://www.nvidia.com/en-us/data-center/tesla-k80/`

de la capa de salida hacia las capas anteriores se hagan muy pequeños o muy grandes, esto afecta el proceso de aprendizaje del modelo e impiden que este converja a una solución (Bengio et al., 1994). Los vectores de bias o sesgo de la arquitectura propuesta, de igual manera por convención, se inicializan con valores iguales a cero. Las convenciones antes mencionadas se toman a partir del curso en línea de aprendizaje de máquina[3] de Google.

Además de la inicialización de los pesos con una distribución normal truncada, en `TensorFlow` ya están implementadas diferentes maneras de inicialización y que en esta tesis se emplearán en algunos experimentos, en particular las inicializaciones `Xavier`[4] y `He`[5]

Andrew Ng propone otras maneras de inicialización de los tensores tomando en consideración la dimensión de las capas, la función de activación, entre otros parámetros[6]

En la siguiente sección se desglosan los parámetros de cada arquitectura neuronal bilingüe propuesta utilizada en la experimentación.

4.5. Arquitecturas neuronales bilingües de experimentación

A continuación se listan los diferentes entornos experimentales implementados en esta tesis, esto es, las diferentes arquitecturas neuronales propuestas en el capítulo 4, así como sus *hiperparámetros* como el número de capas, número de neuronas por capa, optimizadores, técnicas de regularización, etc.

[3]Curso intensivo de aprendizaje automático: `https://developers.google.com/machine-learning/crash-course/`

[4]`https://www.tensorflow.org/api_docs/python/tf/contrib/layers/xavier_initializer`

[5]`https://www.tensorflow.org/api_docs/python/tf/keras/initializers/he_normal`

[6]`https://www.coursera.org/specializations/deep-learning`

4.6. Modelo `model_joyce`

	Parámetros	valores
model_joyce	Data Augmentation (DA)	2x
	Learning rate (LR) (η)	1
	# Capas ocultas	1
	# neuronas/capa	300
	F. Act. h	elu
	Dropout	1
	F. Act. capa salida	tanh
	Épocas	45K
	Inicialización pesos	He/Xavier
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*AdagradOptimizer*

Tabla 4.3: Modelo **model_joyce**

4.7. Modelo `model_klein`

	Parámetros	valores
model_klein	DA	2x
	LR (η)	1
	# Capas ocultas	1
	# neuronas/capa	300
	F. Act. h	elu
	Dropout	0.85
	F. Act. capa salida	tanh
	Épocas	50K
	Inicialización pesos	He/truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*AdagradOptimizer*

Tabla 4.4: Modelo **model_klein**

4.8. Modelo 8

	Parámetros	valores
8	DA	-
	LR (η)	0.001
	# Capas ocultas	3
	# neuronas/capa	350-220-350
	F. Act. h	tanh-tanh-tanh
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	600
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 4.5: Modelo **8**

4.9. Modelo 2

	Parámetros	valores
2	DA	-
	LR (η)	0.001
	# Capas ocultas	1
	# neuronas/capa	350
	F. Act. h	elu
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	600
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 4.6: Modelo **2**

4.10. Modelo 4

	Parámetros	valores
4	DA	-
	LR (η)	0.001
	# Capas ocultas	1
	# neuronas/capa	390
	F. Act. h	elu
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	800
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 4.7: Modelo **4**

4.11. Modelo 6

	Parámetros	valores
6	DA	-
	LR (η)	0.001
	# Capas ocultas	1
	# neuronas/capa	100
	F. Act. h	elu
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	1K
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 4.8: Modelo **6**

4.12. Modelo 7

	Parámetros	valores
7	DA	-
	LR (η)	0.001
	# Capas ocultas	1
	# neuronas/capa	300
	F. Act. h	elu
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	600
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 4.9: Modelo **7**

«Y sobre todo mirar con inocencia. Como si no pasara nada, lo cual es cierto.»

— Alejandra Pizarnik, *Caminos del espejo, 1962.*

Capítulo 5

Evaluación y análisis de resultados

En este capítulo se evaluarán los modelos propuestos descritos en el capítulo anterior y se discutirán los resultados obtenidos. Además, se realizaron experimentos para evaluar nuestra arquitectura neuronal propuesta en el par de lenguas inglés-italiano; estas lenguas cuentan con vastos recursos digitales.

5.1. Evaluación

Para evaluar el desempeño de los modelos descritos en §4.5, se tomaron en consideración diferentes escenarios en donde los modelos se entrenaron con los vectores *N2V* con la normalización descrita en §4.3.1; algunos de esos mismos modelos se entrenaron con los vectores *N2V* sin normalizar, también al momento de evaluar se utilizó el lexicón de evaluación tanto normalizado como sin normalizar.

Como ya se ha mencionado, una vez entrenada nuestra arquitectura neuronal podemos obtener para cada palabra del español los k candidatos a traducción más cercanos en náhuatl. Para evaluar el desempeño de los modelos, utilizamos la técnica *precision at k* donde $k = 1, 5, 10$.

Se evaluaron los diferentes modelos bajo distintos escenarios que se explican a continuación.

En la Tabla 5.1 se entrenó y evaluó nuestra arquitectura con los vectores sin aplicar normalización.

Modelo	P@1	P@5	P@10
model_joyce	**0.6928**	0.8928	0.9071
model_klein	0.6785	0.9071	0.9071
8	0.6714	0.8785	0.9

Tabla 5.1: Resultados de los modelos entrenados y evaluados con vectores sin normalización.

En la Tabla 5.2 los modelos propuestos son entrenados con los vectores normalizados y evaluados con vectores sin normalizar.

Modelo	P@1	P@5	P@10
2	**0.7**	0.8928	0.9142
4	0.6928	0.9071	0.9214
6	0.6642	0.8785	0.9
7	0.6785	0.8928	0.9142
8	0.6714	0.9	0.9142

Tabla 5.2: Resultados de los modelos entrenados con vectores normalizados y evaluados con vectores sin normalización.

En la Tabla 5.3 los modelos son entrenados y evaluados con los vectores normalizados.

Modelo	P@1	P@5	P@10
2	0.6571	0.8928	0.9
4	0.6642	0.8928	0.9142
6	0.6714	0.8857	0.9071
7	0.6642	0.8928	0.9071
8	**0.6785**	0.8928	0.9142

Tabla 5.3: Resultados de los modelos entrenados y evaluados con vectores normalizados.

Adicionalmente, tomamos como referencia el trabajo de Gutierrez-Vasques and Mijangos (2017) donde realizan la misma tarea de extracción léxica bilingüe, con los mismos tipos de vectores, pero utilizando la transformación lineal propuesta por Mikolov et al. (2013b). Para hacer que nuestros resultados fueran comparables con la transformación lineal, replicamos los experimentos de los autores empleando un lexicón semilla y de evaluación ligeramente diferentes a los que originalmente utilizaron los autores. Se obtuvieron los siguientes resultados los cuales consideramos como línea de base (ver Tabla 5.4) para nuestros experimentos.

	P@1	**P@5**	**P@10**
$N2V_{transflineal}$	**0.671**	0.886	0.914
$N2V_{transflineal+norm}$	0.664	0.886	0.9

Tabla 5.4: Resultados de extracción léxica bilingüe español náhuatl empleando la transformación lineal propuesta por Mikolov et al. (2013b).

5.2. Análisis de resultados

Comparando nuestras arquitecturas neuronales bilingües con la aplicación de simplemente una transformación lineal para inducir léxico bilingüe, se puede observar que, dados ciertos *hiperparámetros*, las arquitecturas neuronales bilingües que proponemos logran obtener de manera más precisa pares de traducción, es decir, nuestras arquitecturas representan una transformación no lineal que puede capturar mayor proporción de traducciones correctas a comparación de la transformación lineal típica en esta tarea.

Una de las arquitecturas que proponemos y que es la que mejor desempeño obtiene para la extracción de léxico bilingüe, se observa en el **modelo 2** (ver Tabla 4.9), esencialmente en **P@1** donde se muestra una mejora al momento de generar candidatos a traducción (ver Tabla 5.2). Este modelo en particular cuenta con una capa oculta h con una dimensionalidad de 350 (mayor que las capas de entrada y salida), además, se normalizaron los vectores para la etapa de entrenamiento y para medir su desempeño, se utilizó el lexicón de evaluación sin normalizar. Del **modelo 2**, podemos deducir que una capa oculta h de mayor dimensionalidad que los vectores de entrada ayuda a que el modelo obtenga un mejor desempeño en la tarea de extracción léxica bilingüe y que normalizar los vectores favorece al modelo a obtener un mejor desempeño como se mencionó en §4.3.1.

Por otro lado, con base en los experimentos y en particular para esta tarea, una arquitectura neuronal bilingüe con más de una capa oculta como el **modelo 8** (ver Tabla 4.8) no parece producir una mejora sustancial, incluso se observa que los resultados en comparación con la transformación lineal son prácticamente los mismos, y si mencionamos que el entrenamiento del modelo que proponemos es más lento y *pesado* computacionalmente aún usando GPUs, la transformación lineal obtiene los mismos resultados de manera más rápida y sin necesidad de un GPU que agilice el entrenamiento, por lo que podemos concluir que basta con que la arquitectura neuronal bilingüe sea de una capa oculta h para *eliminar* el ruido de las palabras en español y así obtener sus traducciones en náhuatl.

Se puede observar que los modelos **6** y **7** (Tablas 4.11 y 4.12 respectivamente) muestran un desempeño por debajo del *baseline* de la tranformación lineal (ver Tablas 5.2, 5.3, 5.4). Por un lado, el modelo **6** con una capa oculta h de dimensión 100, que es menor a las capas de entrada y salida, tiene un desempeño ligeramente inferior a la transformación lineal, sin

embargo, podemos decir que, aún cuando no logra superar el *baseline*, los resultados del **modelo 6** son, en términos prácticos, aceptables ya que la diferencia con el *baseline* no es muy grande y está dentro de los rangos de precisión de los métodos del estado del arte. Lo mismo pasa con el **modelo 7**, este tiene una capa oculta h de dimensión 300, y su desempeño está por debajo del *baseline* (ver Tabla 5.3), de esto podemos deducir que, aún cuando se aumenta la dimensionalidad de la capa oculta, parece ser que 300 neuronas no bastan para que la arquitectura neuronal bilingüe supere a la transformación lineal, es decir, se puede entender que existe un umbral en cuanto al número de neuronas en la capa oculta h para que el modelo logre superar el *baseline* de la transformación lineal. Es importante recordar que no hay un método riguroso para determinar el número de neuronas óptimo en h, generalmente el número de neuronas en la capa oculta se establecen de manera empírica o experimental.

Un escenario de evaluación que resulta de peculiar interés es en donde la arquitectura que proponemos es entrenada con los vectores *N2V* normalizados y se evalúa con vectores sin normalizar; es en este escenario en donde se obtienen los resultados más altos al momento de generar candidatos de traducción como se observa en la Tabla 5.2. Parece ser que, normalizando el espacio vectorial de español y náhuatl para la etapa de entrenamiento y evaluándolo con vectores sin normalizar, ayuda al modelo de aprendizaje a generalizar de mejor manera la relación entre palabras del español y el náhuatl, así como a reducir de manera más eficaz el *ruido*, tal como se observa en la Tabla 5.2.

Al igual que Payan and Montana (2015); Mnih et al. (2015), observamos que tener un autoencoder con una capa intermedia de mayor dimensión ayuda a mejorar ciertos tipos de tarea. En nuestro caso esta configuración del autoencoder resultó benéfica para la tarea de encontrar pares de traducción español-náhuatl, a pesar de que usualmente un autoencoder se utiliza para reducir la dimensionalidad y no aumentarla.

Con respecto a la normalización de los vectores, nuestros resultados muestran, como se mencionó en §4.3.1, que la normalización ayuda a que los modelos de ML, y en particular la arquitectura que proponemos converja de manera más eficiente.

En la Figura 5.1 se muestran las gráficas de las funciones de error para los **modelos 2, 4, 6 ,7** (ver Tablas 4.9, 4.10, 4.11, 4.12 respectivamente), para esta gráfica los modelos fueron entrenados con vectores normalizados.

En la Figura 5.2 se muestran las gráficas de las funciones de error para los mismos modelos de la Figura 5.1, para esta gráfica los modelos se entrenaron con vectores no normalizados.

A partir de las gráficas de las funciones de error, se puede observar que la normalización hace que la función de error en los modelos que se entrenaron con vectores normalizados

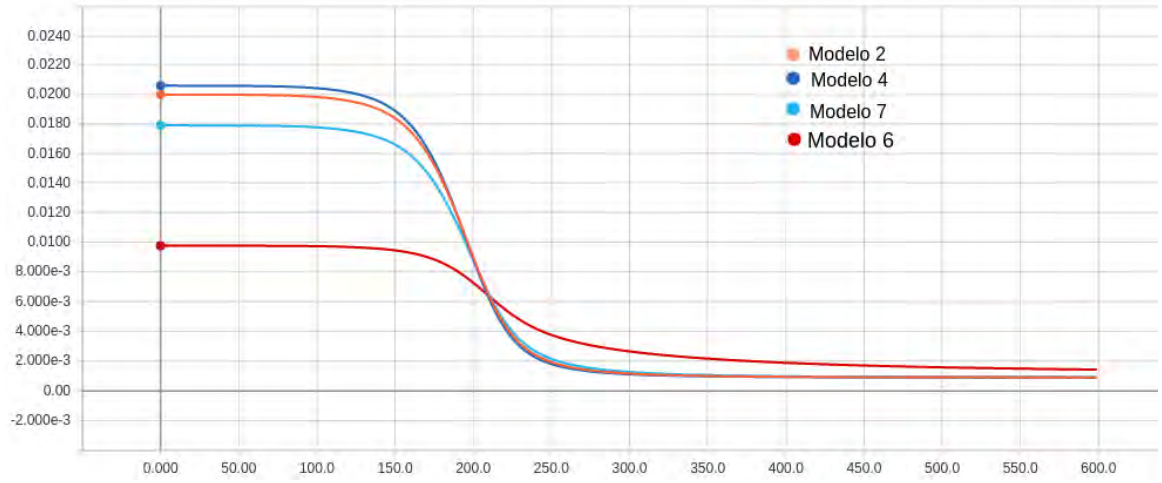

Figura 5.1: Funciones de error de modelos entrenados con vectores normalizados.

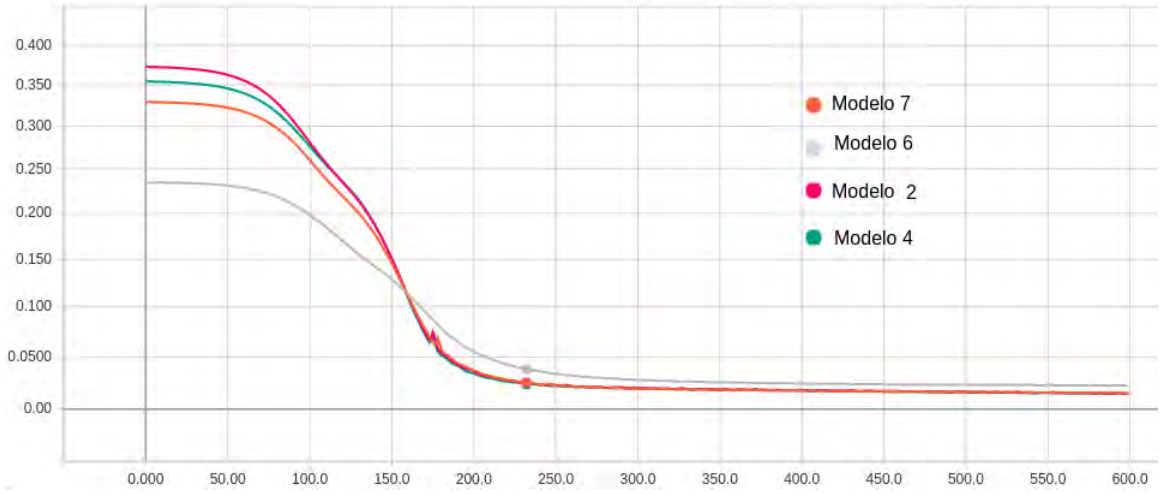

Figura 5.2: Funciones de error de modelos entrenados con vectores no normalizados.

(Figura 5.1) tenga un valor mucho más pequeño cuando inicia la etapa de aprendizaje mientras que en los modelos entrenados con vectores no normalizados (Figura 5.2), el error es más elevado al inicio del entrenamiento. Por otra parte, se observa que al finalizar la etapa de entrenamiento, el error es más elevado en los modelos que fueron entrenados con vectores sin normalizar.

En la Figura 5.3 se observan algunos pares de traducción español-náhuatl obtenidos mediante nuestra arquitectura propuesta (en particular el **modelo 2** que es el que mejor desempeño muestra para inducir léxico bilingüe). Se puede observar que algunos pares de

traducción están muy cercanos unos de otros como en el caso del par **levantar-quetza**, sin embargo hay otros pares que aparecen más lejanos entre sí como el par **coyote-coyoc**.

Para poder visualizar los vectores se aplicó una reducción de dimensionalidad empleando el método de escalamiento multidimensional (MDS) (Kruskal, 1964) el cual ya se encuentra implementado en la biblioteca de aprendizaje automático `scikit-learn`[1] de `Python`. Los parámetros de escalamiento fueron los siguientes: `metric=False` y `n_init=7`; el parámetro `metric=False` indica que se realiza una reducción de dimensionalidad no lineal, los demás parámetros del método son los que vienen por defecto[2]. Cabe mencionar que para poder visualizar los pares de traducción, tratamos dos métodos populares de reducción de dimensionalidad, `PCA` y `t-SNE`. Sin embargo, con estos métodos no era visiblemente claro el pareamiento entre las traducciones debido probablemente a que ambos hacen una reducción de dimensionalidad lineal.

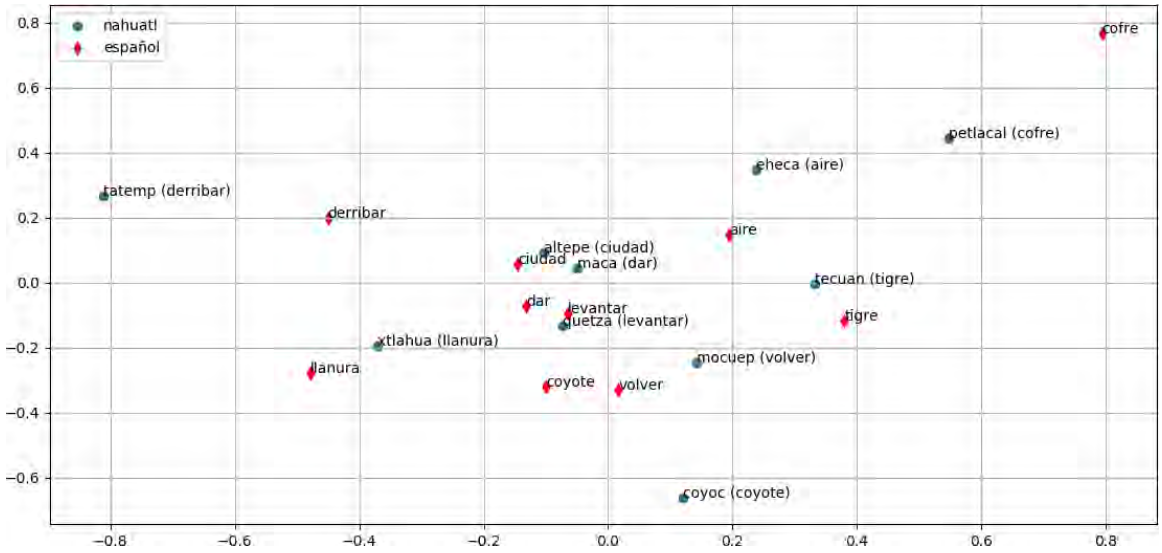

Figura 5.3: Pares de traducción español-náhuatl.

En la siguiente sección, empleamos nuestra arquitectura para inducir léxico bilingüe en donde el par de lenguas cuenta con altos recursos digitales.

[1] `http://scikit-learn.org/stable/`

[2] Sobre los parámetros para ajustar el método MDS: `http://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html`

5.3. Experimentación en otras lenguas

Adicional a los experimentos en lenguas mexicanas, se probó la arquitectura neuronal bilingüe para extracción léxica bilingüe en un escenario en donde las lenguas cuentan con vastos recursos digitales. En particular se abordó el par de lenguas inglés-italiano del cual podemos encontrar en línea los lexicones semilla y de evaluación con cierta facilidad.

En este escenario, la extracción léxica bilingüe utilizando vectores *W2V* sin ningún tipo de normalización y una transformación lineal da como resultado un desempeño de $P@1 = 34{,}93\,\%$ (Artetxe et al., 2018).

Para esta etapa de experimentación, decidimos probar inducir léxico bilingüe empleando representaciones vectoriales *fastText* y la arquitectura neuronal bilingüe descrita en §4.4 usada en lenguas mexicanas.

5.3.1. Representaciones vectoriales *fastText*

Como se mencionó en §2.3.2, el modelo *fastText* toma en consideración la morfología de las palabras y permite generar vectores de palabras que no fueron vistas durante la etapa de entrenamiento en el corpus.

Ya hay modelos pre-entrenados de vectores de palabras los cuales tienen una dimensión de 300. Estos modelos están disponibles en línea para 157 lenguas[3], se puede descargar solamente el conjunto de vectores o los vectores junto con el modelo con el cual se generaron, esto con el fin de que, en caso de ser necesario, utilizar el modelo para generar palabras fuera del vocabulario o que no fueron vistas durante la etapa de entrenamiento para generar nuevos vectores de palabras.

Para hacer que nuestros experimentos sean comparables, se requirió de los mismos lexicones semilla y evaluación que se utilizaron en el experimento que usa vectores *W2V* y una transformación lineal (Artetxe et al., 2018). En el caso de inglés-italiano, se cuenta con un conjunto de vectores de palabras *W2V* tanto en inglés como en italiano, este conjunto de vectores es utilizado por Artetxe et al. (2018) en sus experimentos. Cada conjunto de vectores tanto en inglés como en italiano cuenta con 200 000 vectores de palabras, además se tiene un lexicón semilla con 5000 pares de palabras y un lexicón de evaluación con 1500 pares[4]. Estos mismos lexicones son los que utilizaremos durante la experimentación.

[3]Los vectores y modelos pre-entrenados se pueden encontrar en: `https://fasttext.cc/docs/en/crawl-vectors.html`

[4]Tanto los lexicones como los vectores *W2V* inglés-italiano se pueden descargar de la siguiente página: `http://clic.cimec.unitn.it/~georgiana.dinu/down/resources/transmat.zip`

En cuanto a las representaciones vectoriales de tipo *fastText*, que proponemos utilizar, se cuenta con 1M de vectores de palabras *fastText* generados a partir del corpus de Wikipedia en inglés, mientras que se tienen 200 000 vectores de palabras *fastText* en italiano.

En primera instancia, dado que el conjunto de vectores en inglés *fastText* es de 1M, debemos obtener un subconjunto de vectores *fastText* que coincida con el conjunto *W2V* para que nuestros experimentos sean comparables. Para esto, debimos cerciorarnos de que el subconjunto que necesitabamos generar contuviera las mismas palabras que el conjunto *W2V*. Para generar el nuevo subconjunto, se buscó cada palabra *W2V* dentro del conjunto *fastText*. Una vez que generamos el nuevo subconjunto, su tamaño era de 150 783 vectores *fastText*, lo que significaba que el conjunto original de 1M de vectores *fastText* no contenía todas las palabras que existen dentro del conjunto *W2V*. Se realizó este procedimiento pero ahora para los vectores en italiano donde sucedió lo mismo; el subconjunto de vectores *fastText* en italiano generado contenía 163 031 vectores, es decir, no se encontraron todas las palabras en italiano necesarias para que los conjuntos *W2V* y *fastText* fueran comparables.

Gracias a que *fastText* permite generar vectores para palabras fuera del vocabulario, como se mencionó en §2.3.2, se lograron generar las representaciones vectoriales de las palabras faltantes; 49 217 en inglés y 36 966 en italiano, por lo que en total se generaron 86 138 nuevos vectores *fastText*.

Se debe mencionar que el tamaño total del conjunto obtenido de vectores *fastText* en italiano, era de 199 997, es decir, no se generaron vectores para 3 palabras en italiano debido a la codificación de los caracteres, es posible que *fastText* no fuera capaz de reconocer algunos caracteres dentro de las tres palabras faltantes y por ese motivo no logró generar sus representaciones vectoriales; no se tomó mucho en consideración el hecho de que faltaran tres palabras ya que en comparación con el tamaño del conjunto, no nos pareció que perjudicara demasiado para los fines de experimentación.

5.3.2. Transformación lineal usando *fastText*

Una vez construidos los conjuntos de experimentación inglés-italiano usando representaciones vectoriales *fastText*, lo primero que hicimos fue obtener la matriz de transformación usando estos vectores.

	P@1	**P@5**	**P@10**
$fastText_{no-norm}$	0.348	0.519	0.564
$fastText_{norm}$	0.38	0.553	0.601

Tabla 5.5: Extracción léxica bilingüe usando vectores *fastText* y la transformación lineal.

Se realizaron dos evaluaciones; la primera consistió en el aprendizaje de la transformación

lineal usando vectores con la normalización ($fastText_{norm}$) descrita en §4.3.1, y otra con los vectores sin ningún tipo de normalización $fastText_{no-norm}$.

De la Tabla 5.5 se puede observar que, a pesar de que se tuvieron que generar 86 138 nuevos vectores *fastText*, los resultados son muy similares a los obtenidos por Artetxe et al. (2018) donde emplearon vectores *W2V* ($P@1 = 34{,}93\,\%$) . Por otro lado, la normalización de vectores *fastText* ayudó considerablemente al momento de generar vectores de palabras candidatas a ser traducción.

Es interesante que, aún cuando se generaron cerca del 20 % de vectores utilizando el planteamiento de *fastText* tanto en inglés como en italiano, para hacer experimentos comparables con la transformación lineal y vectores *W2V*, la calidad de los vectores generados son muy similares a los *W2V*. Esto enfatiza que las formulaciones vectoriales de palabra que toman en cuenta unidades subpalabra son una alternativa que ayuda a reducir la cantidad de corpus necesario para entrenar representaciones de buena calidad.

5.3.3. Arquitectura neuronal bilingüe inglés-italiano

La principal diferencia en cuanto a los experimentos realizados en español-náhuatl es el tipo de representación vectorial. En español-náhuatl se utilizaron vectores *N2V* cuya dimensionalidad era de 128. Los vectores *fastText*, como se menciona en la sección anterior, cuentan con una dimensión de 300.

Dado que los vectores *fastText* son de una dimensionalidad superior a los vectores *N2V* español-náhuatl, la arquitectura que proponemos, en particular el tamaño de la capa oculta h, debe adaptarse de acuerdo a la dimensionalidad de los vectores tomando en consideración que esta debe ser mayor a 300 siguiendo el mismo principio usado en los experimentos español-náhuatl.

En uno de los modelos propuestos la una capa oculta es de menor dimensionalidad que las capas de entrada y salida con el fin de observar su comportamiento.

Se proponen 3 arquitecturas neuronales bilingüe inglés-italiano distintas:

Modelo 3

	Parámetros	valores
3	Data Augmentation (DA)	-
	Learning rate (LR) (η)	0.01
	# Capas ocultas	2
	# neuronas/capa	370-370
	F. Act. h	tanh-tanh
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	1500
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 5.6: Modelo **3**

Modelo 5

	Parámetros	valores
5	Data Augmentation (DA)	-
	Learning rate (LR) (η)	0.01
	# Capas ocultas	1
	# neuronas/capa	370
	F. Act. h	elu
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	1500
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 5.7: Modelo **5**

Modelo 9

	Parámetros	valores
9	Data Augmentation (DA)	-
	Learning rate (LR) (η)	0.01
	# Capas ocultas	1
	# neuronas/capa	128
	F. Act. h	elu
	Dropout	0.51
	F. Act. capa salida	tanh
	Épocas	1500
	Inicialización pesos	truncated_norm(stddev=0.1)
	Función de costo J	MSE
	atar pesos	-
	Optimizador	*RMSPropOptimizer(centered=True)*

Tabla 5.8: Modelo **9**

5.3.4. Evaluación inglés-italiano

Se consideraron dos escenarios de evaluación, en uno el entrenamiento y evaluación se realiza con vectores normalizados, los resultados se muestran en la Tabla 5.9.

Modelo	**P@1**	**P@5**	**P@10**
3	0.366	0.5606	0.587
5	0.364	0.56	0.5906
9	0.332	0.524	0.554

Tabla 5.9: Resultados de los modelos entrenados y evaluados con vectores *fastText* normalizados

Por otra parte, se entrenó nuestra arquitectura propuesta con vectores normalizados y se evalúa con vectores sin normalizar (ver Tabla 5.10).

Modelo	**P@1**	**P@5**	**P@10**
3	0.33	0.495	0.536
5	0.33	0.49	0.531
9	0.294	0.476	0.511

Tabla 5.10: Resultados de los modelos entrenados con vectores *fastText* normalizados y evaluados con vectores no normalizados

De la Tabla 5.9 se observa que los modelos **3** y **5** (Tablas 5.6 y 5.7 respectivamente) superan a la transformación lineal con vectores *fastText* sin normalizar (ver Tabla 5.5); sin embargo, en la tabla 5.10 se observa que con vectores normalizados, el resultado del modelo es inferior al de la transformación lineal.

En cualquier caso, ninguno de nuestros modelos propuestos logra superar el resultado de la transformación lineal donde $P@1 = 38\,\%$ (ver Tabla 5.5) para este par de lenguas. Esta tendencia es diferente a lo que sucedió con el par de lenguas español-náhuatl en donde nuestro modelo propuesto sí logra resultados superiores a la transformación lineal.

Como ya se había observado, el hecho de aumentar la dimensionalidad de la capa oculta h ayuda a capturar características de los vectores de mejor manera. Se observa que el modelo **9** (tabla 5.8) es el de peor desempeño, cuenta con una capa oculta de dimensión 128, menor a la dimensiones de los vectores lo que apoya la hipótesis de que, en particular para esta tarea, al disminuir la dimensión de h de la arquitectura neuronal bilingüe, no logra capturar características significativas en comparación con una capa h de dimensión superior.

Es posible que las representaciones vectoriales *N2V*, a diferencia de *fastText* y *W2V*, estas dos últimas monolingües y basadas en nociones similares, logren capturar de mejor manera las características bilingües y por ello nuestra arquitectura neuronal bilingüe logró superar a la transformación lineal en el par de lenguas español-náhuatl mientras que en inglés-italiano los resultados de la arquitectura sólo lograron superar el *baseline* de la transformación lineal con vectores no normalizados.

«¿Volver?», pensó. «No sirve de nada. ¿Ir por algún camino lateral? ¡Imposible! ¿Ir hacia adelante? ¡No hay alternativa! ¡Adelante pues!»

— J. R. R. Tolkien, *El Hobbit*, 1937.

Capítulo 6

Conclusiones y trabajo futuro

En esta tesis, estábamos interesados en realizar extracción de léxico bilingüe en lenguas mexicanas de bajos recursos digitales a partir de un corpus paralelo español-náhuatl. Para esta tarea, propusimos una arquitectura neuronal que hiciera un mapeo entre las lenguas español-náhuatl.

Nuestra hipótesis fue que un mapeo no lineal para inducir léxico bilingüe mediante una red neuronal era capaz de superar al mapeo realizado por una transformación lineal para la misma tarea. Aprovechamos principalmente la propiedad de no linealidad que nos proporciona una arquitectura neuronal. Partimos de una arquitectura en particular llamada autoencoder, específicamente de autoencoders dispersos y para eliminación de ruido. Su principal característica es que su capa oculta era de mayor dimensionalidad que las capas de entrada y salida, esto con el objetivo de que la arquitectura lograra hacer un mapeo no lineal de mayor calidad que el obtenido por una transformación lineal.

Como ya se mencionó, una de las características más destacadas de una red neuronal, es su capacidad de lidiar con conjuntos de datos no lineales gracias a las funciones de activación, lo que le permite descubrir patrones que tal vez que no fáciles de observar utilizando algoritmos tradicionales.

Para la tarea de extracción léxica bilingüe entre el par de lenguas español-náhuatl, podemos concluir que fuimos capaces de mejorar los resultados utilizando una combinación de los siguientes factores: una transformación no lineal mediante una variación de autoencoders dispersos y autoencoders para eliminación de ruido, representaciones vectoriales bilingües N2V español-náhuatl, ciertos hiperparámetros de la arquitectura neuronal y, en particular, una capa oculta h de mayor dimensionalidad que las capas de entrada y salida.

Es importante discutir varias cuestiones, por un lado, que algunas de las arquitecturas e hiperparámetros propuestos en esta tesis no lograron superar a la transformación lineal para encontrar pares español-náhuatl. Sin embargo, en todos los casos nuestros entornos experimentales logran superar a los resultados que se obtendrían utilizando representaciones vectoriales del tipo *W2V* (Gutierrez-Vasques and Mijangos, 2017). Esto nos lleva a concluir que las representaciones vectoriales juegan un papel muy importante en el desempeño de los sistemas de extracción léxica bilingüe, no sólo el hecho de que se use una transformación lineal o una arquitectura neuronal.

En general, los autoencoders tienen como principal objetivo realizar reducción de dimensionalidad y generar representaciones compactas. Sin embargo, nuestro enfoque fue contrario al objetivo principal de estas arquitecturas donde aumentamos la dimensión de la capa oculta h. El hecho de aumentar la capa oculta ayuda a que nuestra arquitectura supere el resultado de la transformación lineal en un entorno de bajos recursos digitales.

En el caso de los experimentos en el par de lenguas inglés-italiano, nuestro modelo logró superar la línea de base de la transformación lineal, particularmente en donde no se normalizaron los vectores, sin embargo, una vez que normalizamos los vectores inglés-italiano, la transformación lineal superó en desempeño a nuestra arquitectura. En este escenario de experimentación donde las lenguas tienen vastos recursos digitales, podemos concluir que un mapeo lineal y y representaciones vectoriales distribuidas como *W2V y fastText* normalizadas son suficientes para inducir léxico bilingüe, mientras que en un escenario de bajos recursos, las representaciones vectoriales multilingües como la propuesta de *N2V* en conjunto con un mapeo no lineal ayuda a mejorar la extracción de correspondencias bilingües.

En este trabajo, se buscó contribuir al desarrollo de tecnologías de lenguaje, particularmente para las lenguas mexicanas debido a que hay poco desarrollo tecnológico para estas lenguas. Los modelos que proponemos pueden ser de utilidad para la generación automática de diccionarios bilingües o tablas de traducción español-náhuatl para que personas interesadas en las lenguas mexicanas puedan incluir nuestra propuesta en sus trabajos y así seguir contribuyendo a desarrollar tecnologías y que más personas se interesen en el procesamiento de lenguaje natural y métodos de aprendizaje automático.

6.1. Trabajo futuro

El enfoque principal que se le dio a la arquitectura neuronal fue determinar un mapeo del español al náhuatl para la extracción léxica bilingüe, sin embargo, una de las características primordiales de cualquier autoencoder es la proyección de los vectores de entrada en la capa oculta h. Un trabajo futuro interesante sería analizar e interpretar lo que logró capturar el autoencoder en su capa oculta con el fin de conocer su naturaleza y observar si logró

capturar propiedades lingüísticas, y ver si estos vectores proyectados en h tienen propiedades geométricas tal como se observó en el trabajo de Mikolov et al. (2013b).

Por otro lado, en §5.3 se experimentó con el par de lenguas inglés-italiano usando representaciones vectoriales *fastText*, por lo que se deja como posible trabajo futuro realizar los mismos experimentos pero ahora siguiendo la metodología propuesta por Gutierrez-Vasques and Mijangos (2017) empleando representaciones vectoriales *N2V* y observar si, utilizando otras representaciones vectoriales se logra algún tipo de mejoría al momento de inducir léxico bilingüe.

También sería interesante abordar otras lenguas mexicanas, como otomí, maya, etc., para la misma tarea empleando la arquitectura que en este trabajo proponemos. Además, un paso futuro de nuestra arquitectura sería adaptarlo a ser un sistema neuronal de traducción automática que no sólo sea capaz de encontrar pares de traducción sino que pueda traducir a partir de oraciones o párrafos.

Bibliografía

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. https://www.tensorflow.org/.

Rami Al-Rfou, Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual nlp. *arXiv preprint arXiv:1307.1662* .

Guillaume Alain and Yoshua Bengio. 2014. What regularized auto-encoders learn from the data-generating distribution. *The Journal of Machine Learning Research* 15(1):3563–3593.

Sarath Chandar AP, Stanislas Lauly, Hugo Larochelle, Mitesh Khapra, Balaraman Ravindran, Vikas C Raykar, and Amrita Saha. 2014. An autoencoder approach to learning bilingual word representations. In *Advances in Neural Information Processing Systems*. pages 1853–1861.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2016. Learning principled bilingual mappings of word embeddings while preserving monolingual invariance. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. pages 2289–2294.

Mikel Artetxe, Gorka Labaka, and Eneko Agirre. 2018. Generalizing and improving bilingual word embedding mappings with a multi-step framework of linear transformations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18)*.

Pierre Baldi and Kurt Hornik. 1989. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks* 2:53–58.

Dana H Ballard. 1987. Modular learning in neural networks. In *AAAI*. pages 279–284.

Richard Bellman. 1957. *"Dynamic Programming"*. "Princeton University Press", Princeton, NJ, USA, 1 edition. https://goo.gl/WqmkMw.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3(Feb):1137–1155.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. 2013. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*. pages 899–907.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606* .

Hervé Bourlard and Yves Kamp. 1988. Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics* 59(4-5):291–294.

Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Fredrick Jelinek, John D Lafferty, Robert L Mercer, and Paul S Roossin. 1990. A statistical approach to machine translation. *Computational linguistics* 16(2):79–85.

Peter F Brown, Jennifer C Lai, and Robert L Mercer. 1991. Aligning sentences in parallel corpora. In *Proceedings of the 29th annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pages 169–176.

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation: Parameter estimation. *Computational linguistics* 19(2):263–311.

Raúl Canals Marote, Anna Esteve Guillén, Alicia Garrido Alenda, Guardiola i Savall, Maria Isabel, Amaia Iturraspe Bellver, Sandra Montserrat Buendia, Sergio Ortiz Rojas, Herminia Pastor Pina, Pedro M Pérez Antón, et al. 2001. El sistema de traducción automática castellano-catalán internostrum. *Procesamiento del lenguaje natural, nº 27 (septiembre 2001); pp. 151-156* .

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289* .

National Research Council, Automatic Language Processing Advisory Committee, et al. 1966. *Language and Machines: Computers in Translation and Linguistics; A Report*. National Academy of Sciences, National Research Council.

Balázs Csanád Csáji. 2001. Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary* 24:48.

Georgiana Dinu, Angeliki Lazaridou, and Marco Baroni. 2014. Improving zero-shot learning by mitigating the hubness problem. *arXiv preprint arXiv:1412.6568* .

Samuel Dodge and Lina Karam. 2017. A study and comparison of human and deep learning recognition performance under visual distortions. In *Computer Communication and Networks (ICCCN), 2017 26th International Conference on*. IEEE, pages 1–7.

Manaal Faruqui and Chris Dyer. 2014. Improving vector space word representations using multilingual correlation. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*. pages 462–471.

Keith Frankish and William M Ramsey. 2014. *The Cambridge handbook of artificial intelligence*. Cambridge University Press.

William A Gale and Kenneth W Church. 1993. A program for aligning sentences in bilingual corpora. *Computational linguistics* 19(1):75–102.

Alexander Gelbukh. 2002. Tendencias recientes en el procesamiento de lenguaje natural. *Proc. SICOM-2002* .

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pages 855–864.

Ximena Gutierrez-Vasques. 2015. Bilingual lexicon extraction for a distant language pair using a small parallel corpus. In *NAACL-HLT 2015 Student Research Workshop (SRW)*. page 154.

Ximena Gutierrez-Vasques and Victor Mijangos. 2017. Low-resource bilingual lexicon extraction using graph based word embeddings. *arXiv preprint arXiv:1710.02569* .

Ximena Gutierrez-Vasques, Gerardo Sierra, and Isaac Hernandez Pompa. 2016. Axolotl: a web accessible parallel corpus for spanish-nahuatl. In *LREC*.

Aria Haghighi, Percy Liang, Taylor Berg-Kirkpatrick, and Dan Klein. 2008. Learning bilingual lexicons from monolingual corpora. *Proceedings of ACL-08: Hlt* pages 771–779.

Zellig S Harris. 1954. Distributional structure. *Word* 10(2-3):146–162.

Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. 2009. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*. pages 1026–1034.

Robert Hecht-Nielsen. 1987. Kolmogorov's mapping neural network existence theorem. In *Proceedings of the international conference on Neural Networks*. IEEE Press, pages 11–14.

Robert Hecht-Nielsen. 1989. *Neurocomputing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Karl Moritz Hermann and Phil Blunsom. 2013. Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173* .

Karl Moritz Hermann and Phil Blunsom. 2014. Multilingual models for compositional distributed semantics. *arXiv preprint arXiv:1404.4641* .

Geoffrey E Hinton, Nitish Srivastava, Kevin Swersky, Tijmen Tieleman, and Abdel-rahman Mohamed. 2013. From PCA to autoencoders. In *Lecture 15a From Principal Components Analysis to Autoencoders*.

Geoffrey E Hinton and Richard S Zemel. 1994. Autoencoders, minimum description length and helmholtz free energy. In *Advances in neural information processing systems*. pages 3–10.

Kurt Hornik. 1991. Approximation capabilities of multilayer feedforward networks. *Neural networks* 4(2):251–257.

W John Hutchins. 1995. Machine translation: A brief history. In *Concise history of the language sciences*, Elsevier, pages 431–445.

IBM Marketing Cloud. 2017. 10 Key Marketing Trends for 2017 and Ideas for Exceeding Customer Expectations. `https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=WRL12345USEN`. [En línea; revisado en 14-Mayo-2018].

Jeremy Jordan. 2018. Introduction to autoencoders. `https://www.jeremyjordan.me/autoencoders/`. [En línea; revisado en 23-Mayo-2018].

Armand Joulin, Piotr Bojanowski, Tomas Mikolov, and Edouard Grave. 2018. Improving supervised bilingual mapping of word embeddings. *arXiv preprint arXiv:1804.07745* .

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* .

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, pages 427–431.

Bekir Karlik and A Vehbi Olgac. 2011. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems* 1(4):111–122.

Eamonn Keogh and Abdullah Mueen. 2017. Curse of dimensionality. In *Encyclopedia of Machine Learning and Data Mining*, Springer, pages 314–315.

Alexandre Klementiev, Ivan Titov, and Binod Bhattarai. 2012. Inducing crosslingual distributed representations of words. *Proceedings of COLING 2012* pages 1459–1474.

Philipp Koehn. 2009. *Statistical machine translation*. Cambridge University Press.

Andreĭ Nikolaevich Kolmogorov. 1957. On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition. In *Dokl. Akad. Nauk. SSSR*. pages 953–956.

Joseph B Kruskal. 1964. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* 29(1):1–27.

Stanislas Lauly, Alex Boulanger, and Hugo Larochelle. 2014. Learning multilingual word representations using a bag-of-words autoencoder. *arXiv preprint arXiv:1401.1803* .

Angeliki Lazaridou, Georgiana Dinu, and Marco Baroni. 2015. Hubness and pollution: Delving into cross-space mapping for zero-shot learning. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. volume 1, pages 270–280.

Yann LeCun. 1987. *Modèles connexionnistes de l'apprentissage*. Ph.D. thesis, PhD thesis, These de Doctorat, Universite Paris 6.

Robert B Lees and Noam Chomsky. 1957. *Syntactic Structures. Language 33(3 part 1)*, JSTOR, pages 375–408.

Ang Lu, Weiran Wang, Mohit Bansal, Kevin Gimpel, and Karen Livescu. 2015. Deep multilingual correlation for improved word embeddings. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pages 250–256.

Warren S McCulloch and Walter Pitts. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* 5(4):115–133.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* .

Tomas Mikolov, Quoc V Le, and Ilya Sutskever. 2013b. Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168* .

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013c. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. pages 3111–3119.

Tom M Mitchell. 1997. *Machine Learning*. McGraw-Hill.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Kevin P. Murphy. 2012. *Machine learning: a probabilistic perspective*. MIT Press.

Andrew Ng. 2011. Sparse autoencoder. In *CS294A Lecture notes*.

Andrew Y Ng. 2003. *Shaping and policy search in reinforcement learning*. Ph.D. thesis, University of California, Berkeley.

Erkki Oja. 1982. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology* 15(3):267–273.

Adrien Payan and Giovanni Montana. 2015. Predicting alzheimer's disease: a neuroimaging study with 3d convolutional neural networks. *arXiv preprint arXiv:1502.02506* .

Long Qin. 2013. *Learning out-of-vocabulary words in automatic speech recognition*. Ph.D. thesis, Carnegie Mellon University.

Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. 2010. Hubs in space: Popular nearest neighbors in high-dimensional data. *Journal of Machine Learning Research* 11(Sep):2487–2531.

Philip Resnik and Noah A Smith. 2003. The web as a parallel corpus. *Computational Linguistics* 29(3):349–380.

F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* pages 65–386.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323(6088):533.

Elena Sanz. 2015. ¿cuántos idiomas se hablan en el mundo? `https://www.muyinteresante.es/cultura/arte-cultura/articulo/icuantos-idiomas-se-hablan-en-el-mundo`. [En línea; revisado en 8-Mayo-2018].

Jeff Schultz. 2017. How Much Data is Created on the Internet Each Day? `https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/`. [En línea; revisado en 14-Mayo-2018].

Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5(1):3–55.

Yutaro Shigeto, Ikumi Suzuki, Kazuo Hara, Masashi Shimbo, and Yuji Matsumoto. 2015. Ridge regression, hubness, and zero-shot learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, pages 135–151.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017a. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* .

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017b. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Samuel L Smith, David HP Turban, Steven Hamblin, and Nils Y Hammerla. 2017. Offline bilingual word vectors, orthogonal transformations and the inverted softmax. *arXiv preprint arXiv:1702.03859* .

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

D Stathakis. 2009. How many hidden layers and nodes? *International Journal of Remote Sensing* 30(8):2133–2147.

Bugra Tekin, Isinsu Katircioglu, Mathieu Salzmann, Vincent Lepetit, and Pascal Fua. 2016. Structured prediction of 3d human pose with deep neural networks. *arXiv preprint arXiv:1605.05180* .

Nenad Tomasev, Milos Radovanovic, Dunja Mladenic, and Mirjana Ivanovic. 2014. The role of hubness in clustering high-dimensional data. *IEEE Transactions on Knowledge and Data Engineering* 26(3):739–751.

Luz Gloria Torres. 1994. Redes neuronales y aproximación de funciones. *Boletín de Matemáticas* 1(2):35–58.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*. ACM, pages 1096–1103.

Warren Weaver. 1949. The mathematics of communication. *Scientific American* 181(1):11–15.

Ludwig Wittgenstein. 2009. *Philosophical investigations*. John Wiley & Sons.

Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. 2015. Normalized word embedding and orthogonal transform for bilingual word translation. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. pages 1006–1011.