Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Maestría en Ciencias de la Computación

Traductor automático de náhuatl a español

Tesis presentada para obtener el grado de:
Maestría en Ciencias de la Computación

Presenta:
Máximo Enrique Pacheco Martínez

Director de tesis:
Dra. Maya Carrillo Ruíz

Asesor(es) de Tesis:
Dra. María de Lourdes Sandoval Solís

Noviembre 2023

BUAP

Agradecimientos

Después de la Licenciatura viene la Maestría, y hoy, me encuentro frente a ese reto. Una vez más, agradecido por que la vida me preste el tiempo para afrontarlo.

Agradezco a mis padres y familia por su apoyo incondicional a través de todos estos años. A mi tutora por sus consejos y apoyo a través de la realización de este trabajo. Al jurado por formar parte de el, así como su apoyo en la revisión y corrección de la tesis. Por último, agradezco a la Benemérita Universidad Autónoma de Puebla y a la Facultad de Ciencias de la Computación por aceptarme en el programa de maestría.

Resumen

Con el auge de la tecnología, en la actualidad, los traductores se han posicionado como uno de los instrumentos computacionales de mayor demanda por la comunidad, teniendo un uso vasto desde herramientas especializadas de traducción (aplicaciones móviles, páginas web, entre otras.), hasta complementos embebidos dentro de otro aplicativo (traducción automática de subtítulos, traducciones de conversaciones, entre otras.). Un ejemplo claro del uso de traductores, es la traducción en tiempo real que se lleva a cabo en el chat de Soporte de Microsoft, en el cual, un asesor responde en idioma inglés y la traducción se hace al instante para ti en tu idioma de origen, y viceversa. Sin embargo, aunque es una herramienta muy ocupada globalmente, no todos los idiomas son incluidos en la mayoría de traductores, más aún, lenguas originarias de un país en específico o incluso variaciones de la misma lengua, son menos propensas a ser consideradas parte de un traductor. Hablando específicamente de México, los traductores más populares Google y Microsoft, solo incluyen algunas lenguas, por ejemplo, el Otomí y el Maya y estás son incluidas únicamente en el traductor de Microsoft. Esto resulta preocupante, puesto que, si un traductor no incorpora una lengua originaria de México, propicia una perdida cultural de dicha lengua. Debido al uso extenso que tienen los traductores y la necesidad del país por preservar las lenguas originarias, se propone definir un método para realizar la traducción del náhuatl a español enfocándose en los textos recopilados, con el fin de empezar a crear modelos que permitan tener herramientas informáticas que ayuden a mejorar la comunicación entre personas. De esta forma, se ha trabajado en un campo reciente y aun por explorar (Traducción Automática) y a su vez, se contribuye a la preservación de las lenguas originarias de México.

Índice general

CAPÍTULO 1

Introducción

El náhuatl, es un lenguaje poco atractivo a la comunidad científica, primero, no es un idioma oficial de ningún país; segundo, se habla solamente en algunas partes de América [**?**], pero mayoritariamente en México, en el cual cuenta con un número relativamente escaso de hablantes (alrededor de 1.6 millones de hablantes [**?**]); tercero, la disponibilidad de recursos (corpus, audios, libros, diccionarios y traductores) es escasa; cuarto, cuenta con 30 variantes [**?**] de las cuales 5 pertenecen al estado de Puebla (náhuatl de la Sierra noreste de Puebla, náhuatl de la Sierra oeste de Puebla, náhuatl alto del norte de Puebla, náhuatl del centro de Puebla y mexicano del oriente de Puebla). Si bien, su número de hablantes podría considerarse bajo, comparado con idiomas globalmente establecidos como el inglés, chino o el español (siendo incluso idiomas oficiales de las Naciones Unidas [**?**]), es una de las lenguas indígenas más habladas de México dotándola de un peso cultural significativo y, por ende, su preservación resultando de interés. Sumado a esto, existen distintas variantes del náhuatl haciendo prácticamente nulo el número de herramientas computacionales disponibles, así, poner a disposición sistemas de traducción que mitiguen este problema de falta de herramientas es un aporte valioso.

Aun cuando existen recursos del náhuatl a español como el Gran Diccionario Náhuatl (obra colectiva) [**?**], la aplicación para aprender Aprende náhuatl (desarrollado por el Instituto Nacional de los Pueblos Indígenas (INPI) [**?**]) y el corpus paralelo Axolotl [**?**], entre otros, no se cuenta con una herramienta de traducción disponible para la comunidad indígena, puesto que continua la investigación para lograr un resultado satisfactorio.

Adicionalmente, en aras de un cambio fructífero para México, es necesaria la investigación académica de distintas ramas, por ello, se hace alusión a la herramienta computacional de traducción para náhuatl, a fin de aportar a los Programas Nacionales Estratégicos (Pronaces), en específico, al programa de Cultura.

Bajo todo el contexto planteado anteriormente (importancia del lenguaje, falta de

herramientas de traducción, aporte a México, etc.) **el objetivo de este trabajo, es desarrollar un traductor de náhuatl a español** enfocado en los textos recopilados, brindando un aporte para el crecimiento y preservación de la lengua indígena.

1.1. Desarrollo de la Tesis

En la presente tesis, se implementa un Traductor náhuatl - español, mediante el uso de la Traducción Automática, en específico mediante los Transformers. Se crea un nuevo corpus lingüístico náhuatl - español y se presenta la parte teórica así como los pasos realizados para la generación del corpus.

1.1.1. Organización

En el **Capítulo 2**, se da una breve información acerca del náhuatl, sus características, peculiaridades y avances previos. En el **Capítulo 3**, se presentan los fundamentos matemáticos para realizar Traducción Automática, más en específico se tratan los Tranformers, así como un breve repaso de la Búsqueda armónica. En el **Capítulo 4**, se presenta el Corpus Paralelo Ocelotl y el como fue creado. En el **Capítulo 5**, se presenta la arquitectura del traductor así como pruebas. Finalmente, en el **Capítulo 6**, se dan las conclusiones.

CAPÍTULO 2

Náhuatl

El náhuatl es una lengua perteneciente a la familia lingüística yuto-nahua [**?**], que significa *cosas que suenan bien*[**?**]. Se habla en algunas partes de América, pero mayoritariamente en México, donde cuenta con alrededor de 1.6 millones [**?**] de hablantes, distribuidos en las 30 variantes existentes [**?**]. Aunque es un número bajo comparado con idiomas como el inglés, chino o el español, es la lengua indígenas más hablada de México.

2.1. Grafía

Un lenguaje está siempre en constate evolución, desde los primeros escritos a los más recientes, la grafía del náhuatl ha presentado cambios. Uno de ellos, se dio en el congreso de 1982 realizado en la ciudad de Pátzcuaro Michoacán, previo a este congreso, se utilizaban las siguientes grafías y digrafías para escribir el náhuatl:

A • C • CH • E • H • HU • I • L • LL • M
N • O •P • Q • T • TL • TZ • U • X • Y • Z

a partir del congreso, se aprobaron las siguientes grafías:

J • K • S • TS • W

Debido a estos cambios, dependiendo la referencia que se consulte, se puede encontrar una u otra grafía o bien dependiendo la variante del idioma, se puede encontrar una que contenga más cambios a las anteriores dos presentadas. En resumen, no hay una manera estándar de representar el náhuatl escrito.

La siguiente Tabla 2.1 [**?**] muestra algunas diferencias al momento de escribir una palabra en las dos grafías:

Escritos del siglo XVI hasta nuestros días	**Escritos a partir de 1982**	**Traducción**
Mexico	Mexijko	México
Ohtli	Ojtli	Camino
Tzahtzi	Tsajtsi	Grita
Ihcatic	Ijkatik	Igual
Tlahcuilolli	Tlajkuloli	Escritura
Zacatl	Sakatl	Zacate
Cihtli	Sijtli	Anciana
Huehue	Wewe	Anciano
Huahqui	Wajwajki	Seco
Huetzca	Wetzka	Ríe

Tabla 2.1: Grafías del náhuatl.

La pronunciación es similar a la del español, solo con la diferencia de que la "h" tiene sonido de "j".

2.2. Aglutinante y Sintético

El idioma náhuatl es considerado aglutinante, ya que junta varias palabras para formar una sola, y también sintético, al expresar toda una acción con una sola palabra.

Ejemplos de aglutinación [**?**]:

- kuetlaschitokaktle, "El guarache está hecho con cuero de chivo"
- ixtemichoxipalkostik, "cara de gato cuzco amarillo"
- Kakteueueyakxoma, "guaraches largos que parecen de cuchara"
- Nenepilteueyakouatik, "lengua larga como víbora"
- Kuetlaxkouatik "parece cuero de víbora"

Ejemplos de síntesis [**?**]:

- Tlatlakuaskej, "harán un convivio."
- Tlatoponiskej, "tronaran cuetes."
- Iluichiuaskej , "harán una fiesta."
- Siuailuitl, "fiesta de casorio" o fiesta de boda.

2.3. Recursos

Lamentablemente, el náhuatl cuenta con pocos recursos disponibles, dentro de los destacados se encuentra el Gran Diccionario Náhuatl (obra colectiva) [**?**], la aplicación para aprender Aprende náhuatl (desarrollado por el Instituto Nacional de los Pueblos Indígenas (INPI) [**?**]) y el corpus paralelo Axolotl [**?**].

2.4. Avances previos

Se han llevado a cabo estudios relacionados con la traducción entre náhuatl y español. Uno de los primeros avances fue en el análisis de prefijos, sufijos y traducción de algunos términos (exactamente 1514 términos extraídos de 836 textos) del náhuatl (Martínez-Gil, 2012)[**?**], generando un sistema de tres módulos programado mediante un conjunto de reglas para alcanzar la traducción. Otro avance interesante se da por (Manger & Meza, 2018)[**?**], con la creación de cinco traductores entre el español y lenguas indígenas, incluido el náhuatl, usando un corpus de 985 frases en los cinco idiomas, además, dichos traductores fueron construidos con técnicas de Traducción Automática Estadística (SMT, Statistical Machine Translation) y Traducción Automática Neuronal (NMT, Neural Machine Translation), donde para SMT se utilizó el traductor por frases MOSES [**?**] junto con el alineador GIZA++ [**?**] y para NMT se utilizó el modelo Codificador-Descodificador con Redes Neuronales Recurrentes Bidireccionales (BRNN, Bidirectional Recurrent Neural Networks) y Unidades Recurrentes con Compuertas (GRU, Gated Recurrent Unit) [**?**], los puntajes obtenidos mediante la métrica BiLingual Evaluation Understudy (BLEU, BiLingual Evaluation Understudy)(ver sección 3.3.1.3) fueron los siguientes para NMT 3.04 % y SMT 10.14 %. En este trabajo SMT obtiene mejor resultado. Finalmente, en (Bello et al., 2021) [**?**] se da un avance notable, en el cual, se realizó una aplicación móvil usando NMT, en específico, los Transformers (ver sección 3.3.2) [**?**], obteniendo un mejor desempeño a trabajos previos debido al uso de corpus paralelos de mayor tamaño con un puntaje BLEU que va de 47.84 % a 66.45 %, donde los corpus utilizados son: Axolotl, que contiene pares de frases entre náhuatl y español con alrededor de 18 mil frases, y el corpus JW (Jehova Witness) [**?**] que es un compendio de artículos extraídos de la página oficial de los Testigos de Jehová con alrededor de 35 mil pares de frases.

CAPÍTULO 3

Fundamentos Matemáticos

Durante este capítulo, se resumen los fundamentos básicos utilizados para el desarrollo de este proyecto. Se dan tres apartados importantes: Búsqueda Armónica, Conceptos de Traducción Automática (vocabulario, tokenización, representaciones de palabras, etc) y Traducción Automática Neural (codificador-descodificador, modelos de atención, transformers, etc).

3.1. Búsqueda Armónica

La búsqueda armónica es una metaheurística inspirada en la armonía musical [**?**], donde un conjunto de instrumentos emite notas dentro de un rango y tonalidad que al sonar al unísono forman una armonía. Similar a todas las metaheurísticas, se parte de una población inicial, en este caso llamada Memoria Armónica (HM, Harmony Memory), donde cada vector o individuo de la población representa las notas que toca un instrumento. Otros conceptos son: Tamaño de la Memoria Armónica (HMS, Harmony Memory Size) y la dimensión del problema denotada como N:

$$HM = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ x_{HMS1} & x_{HMS2} & \dots & x_{HMSN} \end{bmatrix} \tag{3.1}$$

La generación de una nueva armonía (individuo) $\mathbf{x}^{new}$, se da mediante el proceso de **Consideración de la Memoria Armónica (HMC, Harmony Memory Consideration)**. Consiste en llenar cada componente j, con $j = 1 \dots N$, del vector con alguna de las siguientes opciones:

1. **Por valor existente**: Se selecciona un individuo i al azar de la memoria armónica, con $i = 1 \dots HMS$, posteriormente, se extrae su elemento j, el cual pasará

a ser el componente j del vector $\mathbf{x}^{new}$. Para entrar en este caso, se genera un número aleatorio y se compara con un Umbral de Consideración de la Memoria Armónica (HMCR, Harmony Memory Consideration Rate), si es menor, entonces se aplica.

Si $aleatorio < HMCR$:

$$x_j^{new} = x_{ij}, \quad x_{ij} \in \{x_{1j}, x_{2j}, x_{3j}, \ldots, x_{HMSj}\} \tag{3.2}$$

2. **Inicialización aleatoria**: Si el apartado anterior no se cumple, se hace una inicialización aleatoria del elemento dentro de un intervalo específico dado por el límite inferior l y superior u del problema:

$$x_j^{new} = l + aleatorio \cdot (u - l) \tag{3.3}$$

Una vez obtenido el elemento, se realiza una **Corrección de Tono (PA, Pitch Adjustment)**, con el objetivo de explorar nuevas alternativas, esto, nuevamente mediante un aleatorio y una comparativa respecto a un Umbral de Corrección de Tono (PAR, Pitch Adjustment Rate). Para controlar el ajuste, se usa un Ancho de Banda (BW, Bandwidth). Es decir,

Si $aleatorio < PAR$:

$$x_j^{new} = x_j^{new} \pm aleatorio \cdot BW \tag{3.4}$$

Una vez calculado todos los elementos de $\mathbf{x}^{new}$, el siguiente paso es **actualizar la memoria armónica**, para ello, es necesario evaluar la aptitud f del nuevo individuo y verificar si es mejor que el peor existente, de ser así, el anterior es reemplazado por el nuevo.

Si $f(\mathbf{x}^{new}) < f(\mathbf{x}^{worst})$:

$$\text{Actualiza HM con } \mathbf{x}^{worst} = \mathbf{x}^{new} \tag{3.5}$$

Los pasos se pueden apreciar en el Algoritmo 1.

Algoritmo 1 Búsqueda Armónica

Require: HMS, HMCR, PAR, BW, NI (no. iteraciones), N
Ensure: Todos los aleatorios dentro de $(0, 1)$
  for $i = 0$; $i < HMS$; $i{+}{+}$ **do**
    Genera el vector aleatorio x_i de la memoria armónica.
    Evalúa la aptitud del individuo $f(x_i)$
  end for
  for $t = 0$; $t < NI$; $t{+}{+}$ **do**
    for $i = 0$; $i < HMS$; $i{+}{+}$ **do**
      for $j = 0$; $j < N$; $j{+}{+}$ **do**
        if $aleatorio <$ HMCR **then** ▷ Consideración de la memoria
          $x_j^{new} = x_{ij}, \quad x_{ij} \in \{x_{1j}, x_{2j}, x_{3j}, \ldots, x_{HMSj}\}$ ▷ i aleatorio, no el iterador
          if $aleatorio <$ PAR **then** ▷ Ajuste de tono
            $x_j^{new} = x_j^{new} \pm aleatorio \cdot BW$
          end if
          if $x_j^{new} < l$ **then** ▷ Evita estar fuera de rango
            $x_j^{new} = l$
          end if
          if $x_j^{new} > u$ **then** ▷ Evita estar fuera de rango
            $x_j^{new} = u$
          end if
        else
          $x_j^{new} = l + aleatorio \cdot (u - l)$ ▷ Inicialización aleatoria
        end if
      end for
      if $f(\mathbf{x}^{new}) < f(\mathbf{x}^{worst})$ **then** ▷ Actualización de la memoria
        Actualiza HM con $\mathbf{x}^{worst} = \mathbf{x}^{new}$
      end if
    end for
  end for

3.2. Conceptos Básicos de Traducción Automática

3.2.1. Vocabulario

Un vocabulario es el conjunto de símbolos únicos en un corpus. Pueden ser palabras, así como caracteres o algún otro símbolo.

3.2.2. Tokenización

La tokenización es el proceso que separa una oración o texto en unidades más pequeñas llamadas *tokens*. La separación puede ser por palabras, caracteres o sub-palabras. Por ejemplo, la oración "las flores son hermosas", a nivel de palabra se separa como:

[las,flores,son,hermosas], mientras a nivel de caracter: [l,a,s,f,l,o,r,e,s,s,o,n,h,e,r,m,o,s,a,s], o bien a nivel de sub-palabra como: [las,flor,es,son,hermosa,s]. El conjunto de tokens únicos define el vocabulario.

Para que un sistema pueda procesar correctamente una oración, es necesario definir tokens especiales que permitan definir ciertos aspectos de una oración, estos tokens son:

1. *bos* - Token que indica el inicio de una oración.
2. *eos* - Token que indica el fin de una oración.
3. *unk* - Token que indica un caracter desconocido: Debido a que el tamaño del vocabulario es limitado, este token es escencial, pues reemplaza a cualquier palabra, letra y/o caracter que no exista.
4. *pad* - Token que añade un padding a la oración: Las oraciones en la vida real son de longitud variable, sin embargo, en computación, para poder operar matricialmente se necesita que todas sean del mismo tamaño, este caracter ayuda a completar aquellas oraciones que no tengan suficientes tokens.

Tomando nuevamente la oración "las flores son hermosas" y suponiendo no existe el token de la palabra "son" y se requiere una longitud de ocho tokens, aplicando la tokenización por palabra con los tokens especiales, se genera el siguiente resultado: [*bos*,las,flores,*unk*,hermosas,*pad*,*pad*,*eos*].

De estas tres tokenizaciones, la más interesante, es la de sub-palabra, puesto que requiere de algoritmos complejos para llevarse a cabo, a comparación de las otras tokenizaciones, donde solo se necesita separar por espacio en blanco para palabras o separar caracter a caracter.

Dependiendo del idioma con el que se trabaje, será conveniente usar determinada tokenización, para el caso del náhuatl, al ser un lenguaje aglutinante, lo preferible es usar una tokenización por sub-palabra, por ejemplo, la palabra "nokal", está compuesta por el prefijo posesivo "no"(mi) y la palabra "kalli"(casa), así, una buena tokenización sería: [no, kal].

Existen distintos algoritmos de tokenización por subpalabras, de entre los cuales se destacan: Byte-Pair Encoding (BPE, Byte-Pair Encoding) (consiste en formar tokens basándose en la frencuencia de aparición de pares de caracteres), WordPiece (similar a BPE, en lugar de frecuencia, maximiza una función de error), Unigram (parte de un vocabulario extenso y se reduce progresivamente) y SentencePiece(sirve con idiomas que no usen el espacio para separar palabras, para ello lo incluye como un símbolo más para poder usar BPE) [**?**]. Una implementación del tokenizador SentencePiece[**?**] se puede consultar en el Apéndice A código A.2.

3.2.3. Representaciones

Para que una computadora pueda entender el lenguaje humano, es necesario encontrar una forma de representar numéricamente todas las palabras de nuestro vocabulario. En esta sección se ejemplifican algunas de estas representaciones.

3.2.3.1. Codificación One-Hot

Una forma muy sencilla de representar las palabras, es usar una codificación vectorial *one hot*. Esta codificación, genera un vector del tamaño del vocabulario y asigna a ceros todos sus valores, salvo el elemento donde se representa la palabra, por ejemplo, cada letra del siguiente vocabulario {h, o l, a} se representa como:

$$h = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad o = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad l = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad a = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.6}$$

En la práctica, toda una secuencia de vectores *one hot* suele comprimirse en un solo vector, donde cada posición indica el índice de la palabra en la secuencia, p. ej. hola = {0, 1, 2, 3 }.

3.2.3.2. Word Embeddings

La codificación *one hot*, si bien es fácil de comprender, presenta un problema, al ser vectores independientes, ningún vector guarda algún sentido con algún otro, es decir, se carece de semántica. Para superar esta limitante, se usan los *word embeddings*, este tipo de representación agrupa las palabras con contextos similares y representa cada palabra mediante un vector, compartiendo todas las palabras la misma dimensión.

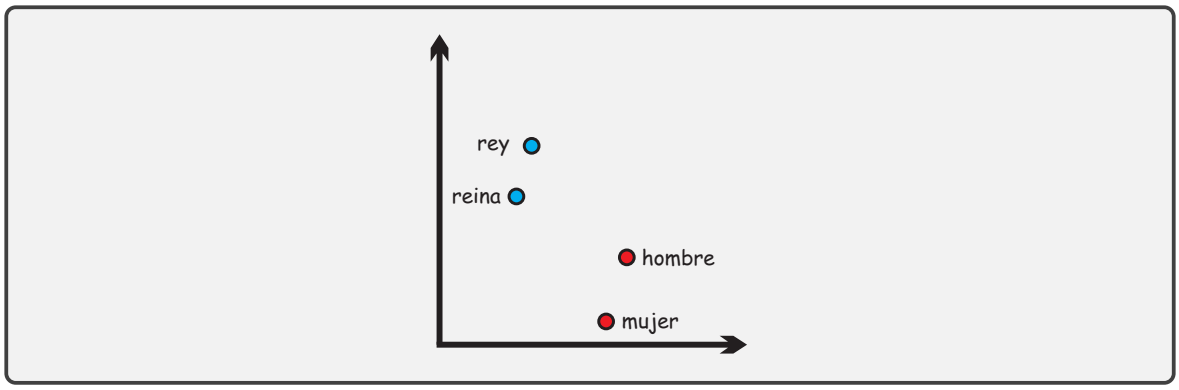

Figura 3.1: Representación gráfica de los *word embeddings*, donde las palabras con contextos similares se agrupan muy cercanamente.

Para ver la utilidad de esta representación, considere el ejemplo clásico, ¿Que pasa si a la palabra rey (*king*) le quitas la connotación de hombre (*man*) y le agregas la de mujer (*woman*)? Naturalmente, esto nos llevará a la palabra reina (*queen*). Puede consultar el código en el Apéndice A código A.2.

Existen distintas variantes de *word embeddings*, pero dentro de las más populares se encuentran *word2vec* (hace uso de redes neuronales para capturar el significado de una palabra basado en su vecindario local, es decir, en las palabras alrededor de ella) y *GloVe* (considera también un vecindario global, no solo local), la diferencia radica más en la forma de entrenamiento que en las diferencias al usarlo, pues en la práctica, ambos modelos dan resultados similares para muchas tareas.

3.3. Traducción Automática Neural

La traducción automática neural, ha aumentado en los últimos años, pues ofrece la posibilidad de modelar secuencias más largas a diferencia de la SMT [?] y aparte de ser un componente todo en uno, ya que la SMT requiere de más componentes para realizar la traducción. Existen dos modelos de traducción neural muy importantes, el primero es el Codificador-Descodificador y el segundo son los Transformers [?]. Durante esta sección se da un resumen de ambos, centrándose en los transformers que será la tecnología a utilizar para el traductor náhuatl - español.

3.3.1. Modelo Codificador - Descodificador

Este modelo consiste en utilizar dos Redes Neuronales Recurrentes (RNNs, Recurrent Neural Networks): La primera, llamada **codificador**, se encarga de comprimir toda la oración en el idioma fuente en un solo **vector de contexto**. La segunda, se encarga de **descodificar** el vector de contexto y pasarlo al idioma destino.

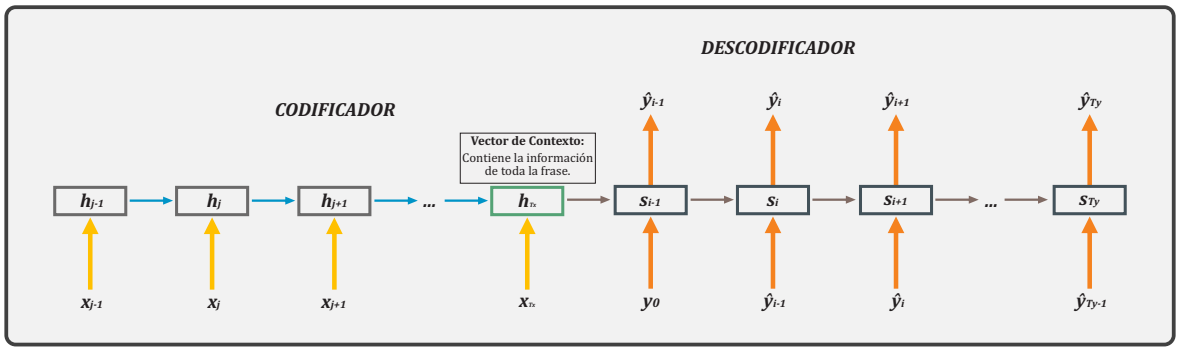

Figura 3.2: Modelo Codificador-Descodificador. La primera RNN se encarga de codificar la oración en un vector de contexto o codificado, posteriormente este vector se usa como estado inicial de la red encargada de descodificar la oración.

Este modelo, si bien, permite trabajar con secuencias largas, al final, todo el contexto se debe comprimir en un solo vector de tamaño fijo, regresando al problema de manejar secuencias muy grandes.

3.3.1.1. Mecanismos de Atención

Los modelos de atención se plantean en [?] (el término no sería usado sino hasta tiempo después) donde se aborda el problema del vector de contexto. En este tipo de modelo, la traducción no solo se basa en el vector del estado oculto y salida previa del descodificador, sino también hace uso de todos los estados ocultos del codificador. Para ello, se define un nuevo vector de contexto.

Cuando el descodificador calcula s_i, además del estado oculto previo s_{i-1}, se necesita del vector de contexto c_i. Este nuevo vector, es la suma de los productos del vector del estado oculto del codificador h_j multiplicado por su respectiva atención $\alpha_{i,j}$. Valores altos de $\alpha_{i,j}$ indican mayor atención a prestar al estado h_j. De esta forma, la traducción al tiempo i prestará atención a todo el contexto de la oración de entrada y no solo a la palabra previa.

Con T_x siendo la longitud de la oración de entrada, T_y la longitud de la oración de salida, $j = 1...T_x$ e $i = 1...T_y$ el vector de contexto se calcula como:

$$c_i = \sum_{j=1}^{T_x} \alpha_{i,j} h_j \tag{3.7}$$

La siguiente imagen ilustra el concepto:

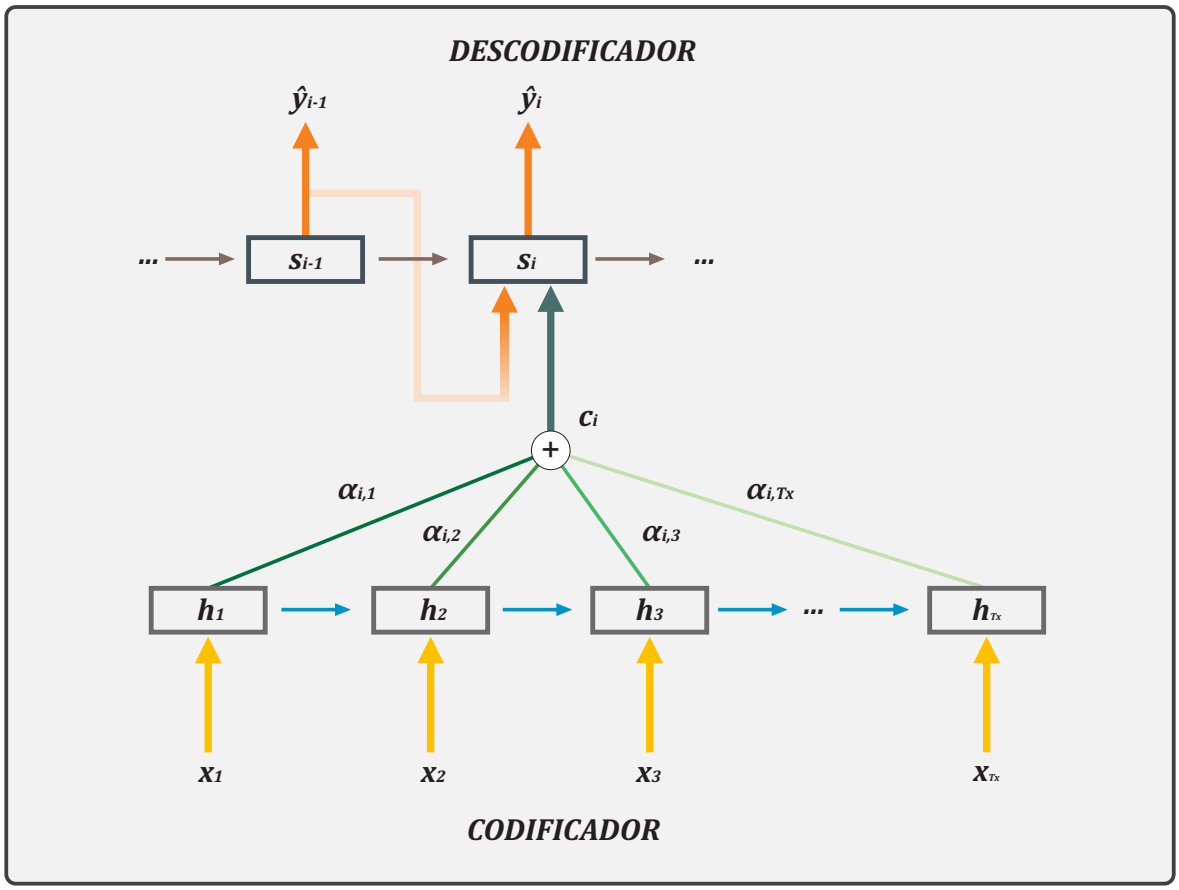

Figura 3.3: Modelo condificador-descodificador con atención. Un tono más fuerte en la linea que une los alfas con c_i indican mayor atención a prestar.

Antes de entrar al cálculo de $\alpha_{i,j}$, se necesita definir el **modelo de alineamiento**,

el cual, pondera que tan bien la entrada h_j y la salida s_i concuerdan, este puntaje $e_{i,j}$ tiene como formula la siguiente:

$$e_{i,j} = a(s_{i-1}, h_j) \tag{3.8}$$

Se toma s_{i-1} ya que al calcular el valor actual de $e_{i,j}$, resulta natural fijarse en lo que se tiene previamente s_{i-1} y en lo que se está observando en este momento por h_j. Originalmente, la función a es una red neuronal, sin embargo, existen distintas formas de calcular este puntaje, uno de ellos es el **puntaje producto punto** [**?**]:

$$e_{i,j} = h_j^\top s_{i-1} \tag{3.9}$$

Calculado este valor, se puede pasar a calcular la atención, la cual, no es más que pasar el puntaje a una probabilidad con la conocida función *softmax*:

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{k=1}^{T_x} \exp(e_{i,k})} \tag{3.10}$$

Hasta este punto concluye la atención, el cálculo del estada oculto s_i y la salida $\hat{y}_i$ dependerán del modelo que se use:

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \tag{3.11}$$

$$\hat{y}_i = g(s_i) \tag{3.12}$$

3.3.1.2. Búsqueda por Haz

Una vez entrenado el modelo, al realizar una traducción, es importante saber de que forma seleccionar la palabra en cada tiempo t, recordando que en una RNN cada salida $\hat{y}_t$ es una distribución de probabilidad sobre el vocabulario, la primera forma sería seleccionar la palabra con mayor probabilidad en cada tiempo t. Sin embargo, hay una forma mejor de realizar está selección y es mediante la Búsqueda por Haz (BS, Beam Search). El algoritmo de BS consiste en ir construyendo la oración tomando las k oraciones más probables en cada tiempo, a k se le conoce como el **ancho del haz**, y en cada paso, se extienden esas k oraciones más probables (se van anidando las multiplicaciones de probabilidades), al final, mantiene la oración con mayor probabilidad. En la siguiente imagen se presenta un ejemplo tomando como vocabulario las letras del alfabeto, los círculos oscurecidos representan las palabras con mayor probabilidad en cada tiempo.

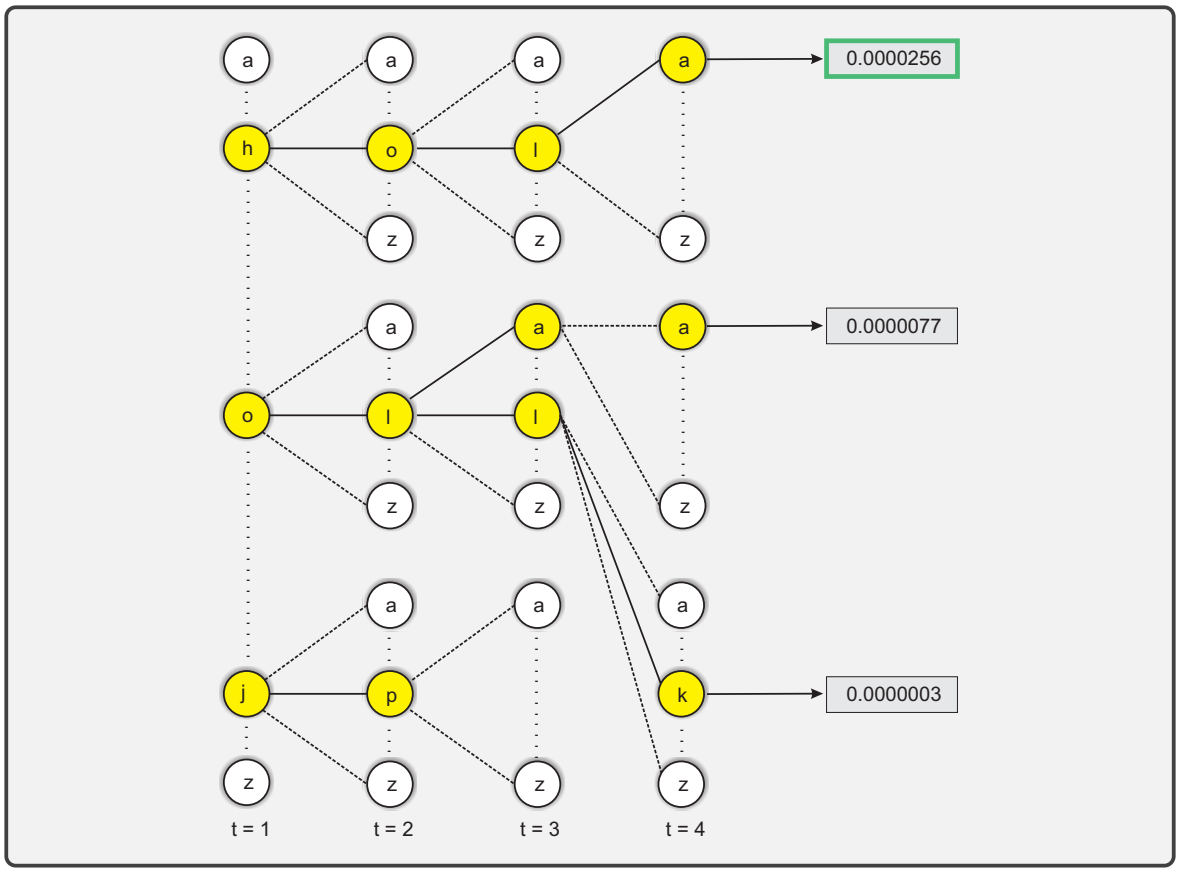

Figura 3.4: Búsqueda por Haz: En cada paso, se debe de extender la oración con todas las palabras/letras del vocabulario y de todas esas posibles oraciones se toman las k mejores. Se necesitan $k \times len(vocabulario)$ evaluaciones en cada paso.

Para el primer paso $t = 1$, se extienden las tres letras *h, o* y *j*, con probabilidad $P(h), P(o)$ y $P(j)$.

En el segundo paso $t = 2$, por cada letra del vocabulario y letra a extender, se calcula una probabilidad: $P(h)P(a \mid h)$, $P(h)P(b \mid h)$, $P(h)P(c \mid h)$, ..., $P(o)P(a \mid o)$, $P(o)P(b \mid o)$, $P(o)P(c|o)$, ..., $P(j)P(a \mid j)$, $P(j)P(b \mid j)$, $P(j)P(c \mid j)$, nuevamente se eligen las tres más altas resultando en $P(h)P(o \mid h), P(o)P(l \mid o)$ y $P(j)P(p \mid j)$, el proceso continua en cada paso.

En $t = 3$ se elige:

$$P(h)P(o \mid h)P(l \mid h, o)$$
$$P(o)P(l \mid o)P(a \mid o, l)$$
$$P(o)P(l \mid o)P(l \mid o, l)$$

En $t = 4$ se elige:

$$P(h)P(o \mid h)P(l \mid h, o)P(a \mid h, o, l)$$
$$P(o)P(l \mid o)P(a \mid o, l)P(a \mid o, l, l)$$
$$P(o)P(l \mid o)P(l \mid o, l)P(k \mid o, l, l)$$

de estás últimas tres, se elige la de mayor probabilidad, resultando en:

$$P(h, o, l, a) = P(h)P(o \mid h)P(l \mid h, o)P(a \mid h, o, l) = 0.0000256$$

3.3.1.3. Puntaje BLEU

El puntaje BLEU, es una métrica para evaluar una traducción, para ello, se compara la oración que genera el modelo con otras oraciones de referencia, donde una referencia es una oración dada por un humano que se entiende es una traducción correcta. Este puntaje es un valor entre 0 y 1, donde 1 indica una traducción perfecta y 0 una totalmente errónea. Cabe aclarar que, en la práctica un puntaje perfecto es imposible, inclusive para un humano, pues una oración se puede traducir de distintas formas.

El puntaje BLEU hace uso de los n-gramas para realizar las comparaciones entre las oraciones, dependiendo cuantos n-gramas ocupe, es el nombre que recibe el puntaje, por ejemplo, si se compara con bigramas, recibe el nombre de BLEU-2, para trigramas BLEU-3, etc. En la práctica se ocupa el **puntaje BLEU acumulativo**, que es la media geométrica de los primeros n puntajes BLEU, por ejemplo, un puntaje acumulativo BLEU 4-grama, es la media geométrica de BLEU-1, BLEU-2, BLEU-3 y BLEU-4.

3.3.2. Transformers

Las RNNs permiten modelar secuencias, esto mediante la dependencia de estados anteriores, pero su mayor virtud es su mayor problema. El depender de estados previos, fuerza el modelo a ser secuencial y hace que las RNNs sean uno de los modelos más lentos y difíciles de paralelizar. Los Transformers [**?**], son modelos que permiten trabajar con secuencias de manera paralela, similar a las RNNs, se conforman de un codificador y descodificador, en la Figura 3.5 se muestra la arquitectura de los transformers, cada componente será descrito en las siguientes secciones.

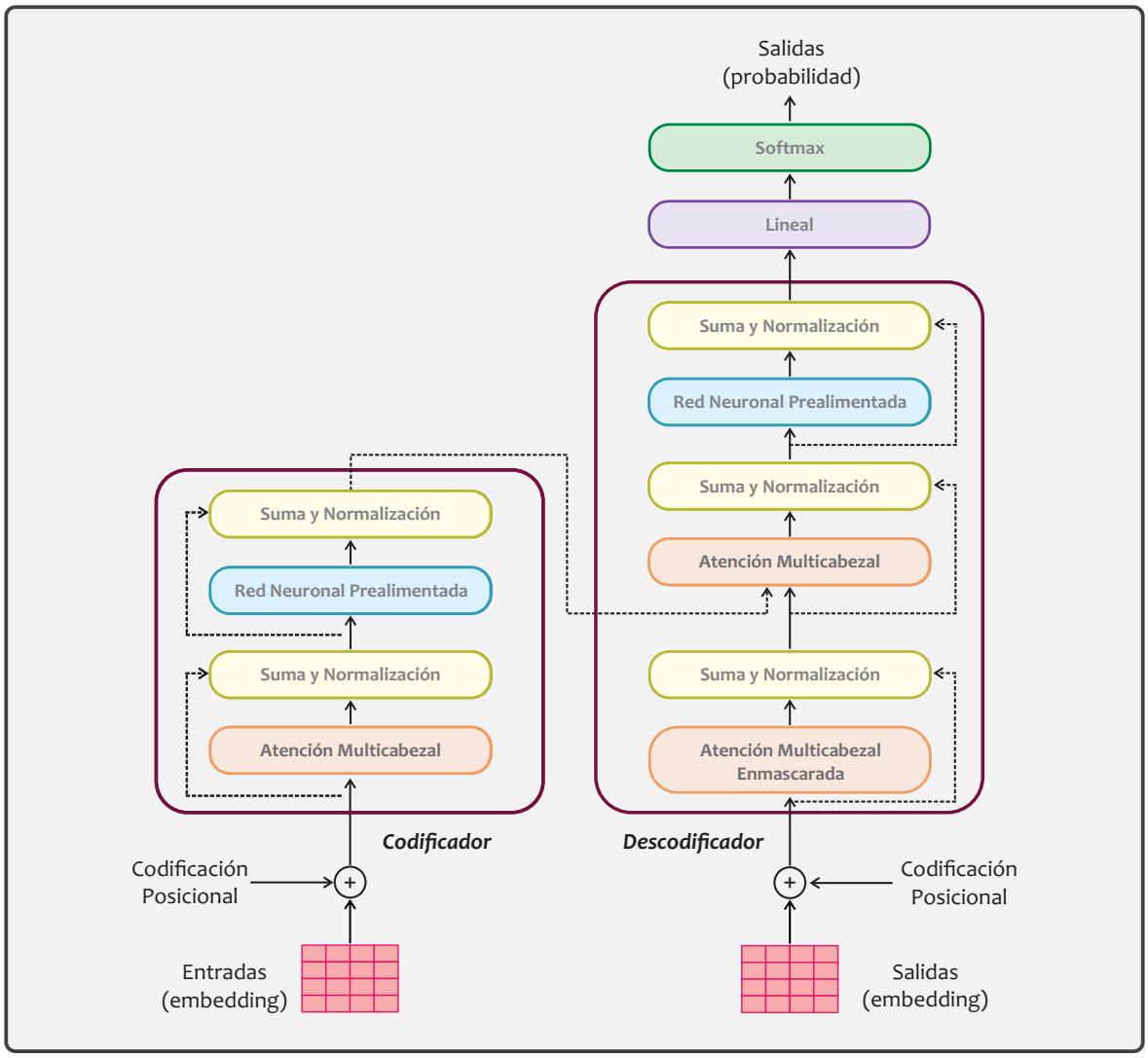

Figura 3.5: Arquitectura del Transformer.

3.3.2.1. Codificación Posicional

Como se mencionó al inicio del capítulo, el objetivo de los Transformers es mejorar la paralelización y evitar secuenciar el modelo, sin embargo, con esto surge un problema, y es, ¿Cómo darle un sentido de orden a las palabras en la secuencia ahora que no se tiene una recurrencia? La primera solución podría ser multiplicar cada palabra con un entero, para la primera con 1, la segunda con 2 y así sucesivamente, si bien es muy sencillo, para secuencias muy grandes la magnitud se dispara. También, se puede intentar normalizar entre [0,1], donde 0 para la primera palabra y 1 para la última, el problema ahora es que varia la distancia de los intervalos dependiendo la longitud de la secuencia. De estas aproximaciones se concluye lo siguiente: Cada representación de una palabra debe ser única, la distancia entre dos intervalos(palabras) debe ser constante

sin importar la longitud de diferentes secuencias y debe estar acotado. Los autores de [**?**], para solventar este problema, hacen uso de las funcione seno y coseno, variando la frecuencia en cada posición del vector generando vectores de posicionamiento únicos que mantienen la distancia constante entre dos intervalos sin importar la longitud de la secuencia.

El vector de posicionamiento se define como:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

donde:

- pos: Posición de la palabra en la secuencia de longitud L, donde $0 \leq pos \leq L-1$
- d_{model}: Dimensión del vector de posicionamiento.
- 10000: Valor por defecto.
- i: Componente del vector de posicionamiento al cual se le asigna el valor del seno o coseno dependiendo si es par o impar. Con el mismo índice se asignan ambos valores, por lo tanto, $0 \leq i < d_{model}/2$

La siguiente imagen muestra una gráfica con los vectores de posicionamiento, donde se nota que cada uno es único. Consulte el código en el Apéndice A código A.3

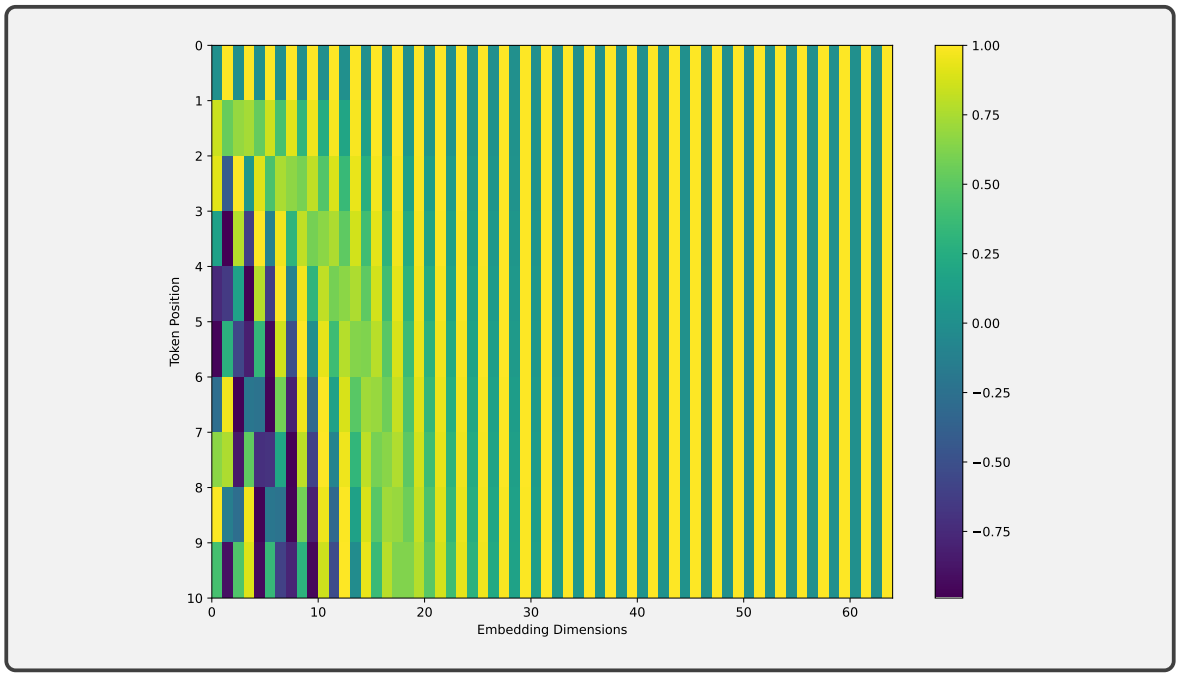

Figura 3.6: Codificación posicional.

3.3.2.2. Autoatención

La **autoatención** es la implementación del mecanismo de atención para los Transformers. El objetivo, es relacionar una palabra con otra, por ejemplo, "La mujer compró la computadora porque era bonita.", una oración muy sencilla, en la cual, la palabra "era" evidentemente hace referencia a la "computadora", sin embargo, para un sistema, también podría hacer referencia a la palabra "mujer", la autoatención captura estas relaciones.

Para calcular la autoatención, se hace uso de tres vectores especiales: **vector *query***, **vector *key*** y **vector *value***, por el momento se omite como obtenerlos, considere solamente la existencia de los mismos.

Los pasos son los siguientes:

1. **Obtener vectores q, k y v**: Por cada vector en la secuencia se requieren los tres vectores, estos se obtienen de multiplicar la entrada por una matriz de pesos respectiva W_Q, W_K y W_V (son entrenadas durante el proceso de entrenamiento del modelo), considerando una secuencia de tres entradas, por cada entrada se obtendrían tres vectores, dando un total de nueve. Para simplificar, matricialmente la operación es la siguiente:

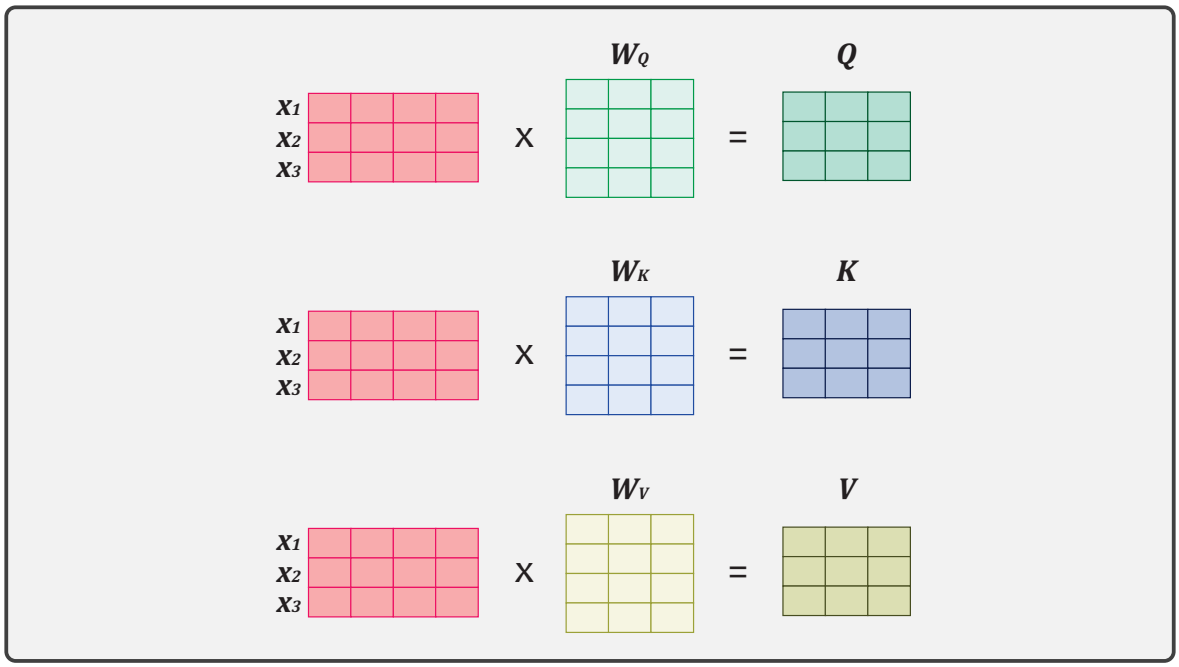

Figura 3.7: Cálculo de los vectores q,k y v en forma matricial.

2. **Cálculo del puntaje de atención:** El siguiente paso es calcular todos los puntajes de atención. Para obtener el primer puntaje s_1, se multiplica q_1 por todos

los vectores de K.

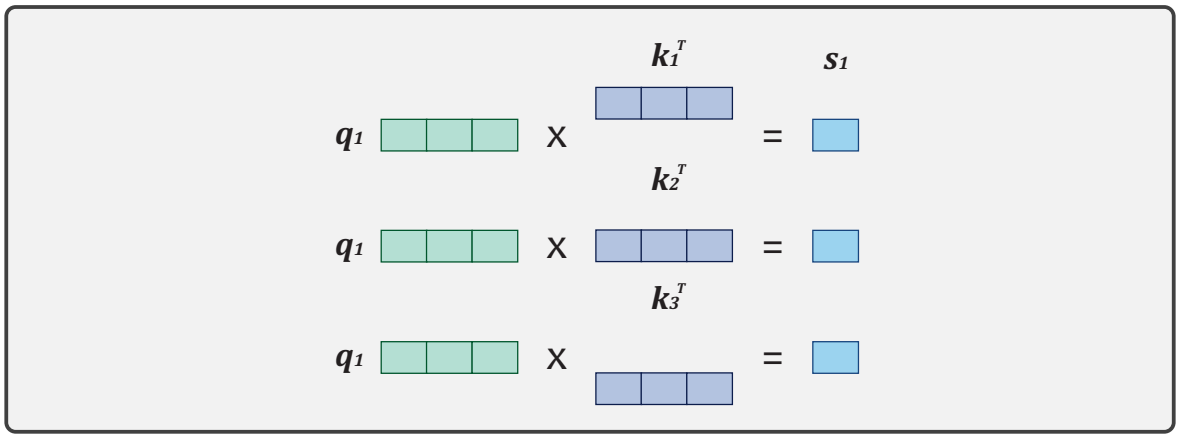

Figura 3.8: Cálculo de los puntajes.

aplicándolo a todos en su forma matricial:

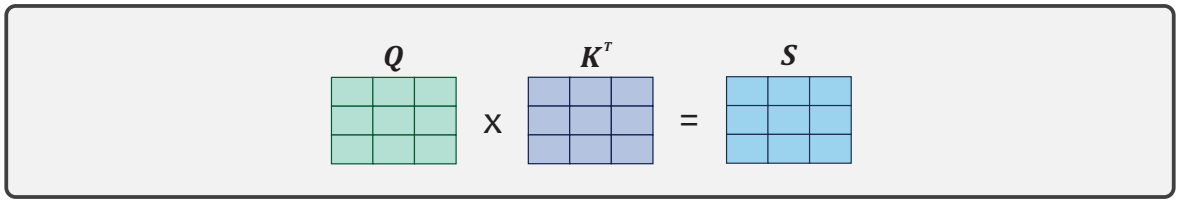

Figura 3.9: Cálculo de los puntajes en su forma matricial.

3. **Puntaje a probabilidad:** Una vez calculados los puntajes, se tienen que pasar a una probabilidad:

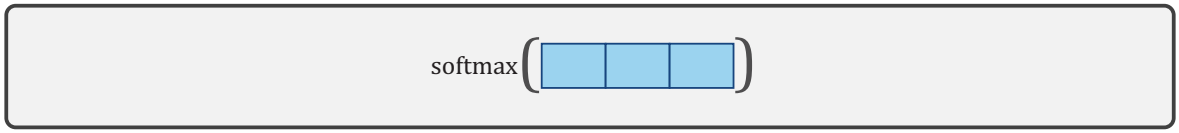

Figura 3.10: Puntaje a probabilidad.

4. **Suma de vectores de alineamiento:** Finalmente, se multiplica cada probabilidad por cada uno de los vectores de valores, y el resultado de todos ellos se suma en un único vector de salida o, el cual representa la consulta del primer vector q_1 con todas las llaves.

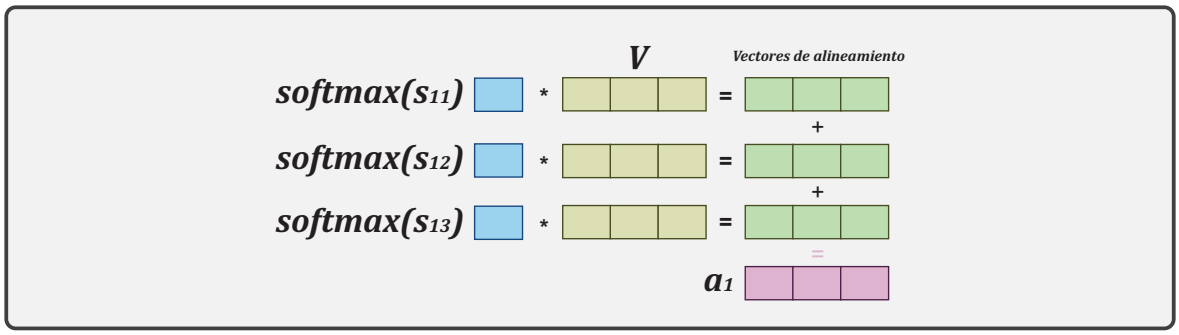

Figura 3.11: Obtención de la salida.

Incluyendo los últimos dos pasos en forma matricial, el cálculo final queda de la siguiente manera:

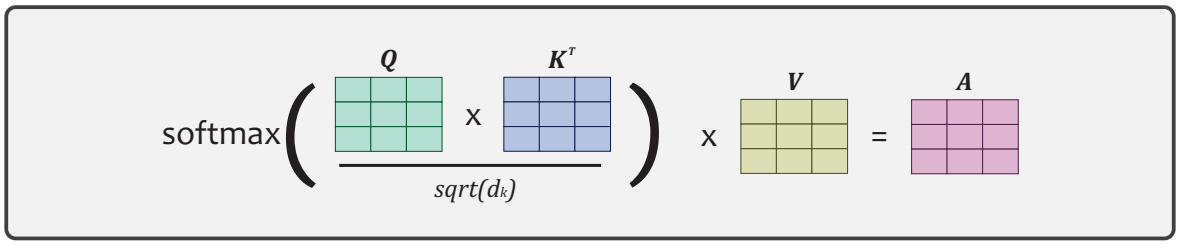

Figura 3.12: Cálculo de la autoatención.

Donde d_k es la dimensión de los vectores *key* y *query*, si esta dimensión es enorme, el producto punto $Q \times K^\top$ crecerá en magnitud y llevará al exponente de la función softmax a valores pequeños evitando el aprendizaje con gradiente, para contrarestar este efecto, se escala cada vector del resultado del producto punto por el factor $\sqrt{d_k}$. Un ejemplo guiado se puede apreciar en el código A.4 del apéndice A.

3.3.2.3. Autoatención Multicabezal

De acuerdo a la Figura 3.5, se tiene el componente **atención multicabezal**, esto no es más que replicar n veces la autoatención, por ejemplo, con $n = 3$, se tienen 3 tercias de matrices $WQ_1, WK_1, WV_3, WQ_2, WK_2, WV_2$ y WQ_3, WK_3, WV_3, cada tercia generando una salida de autoatención A_1, A_2, A_3, todas estas salidas son concatenadas y finalmente se multiplican por una matriz más WO aprendida durante el entrenamiento:

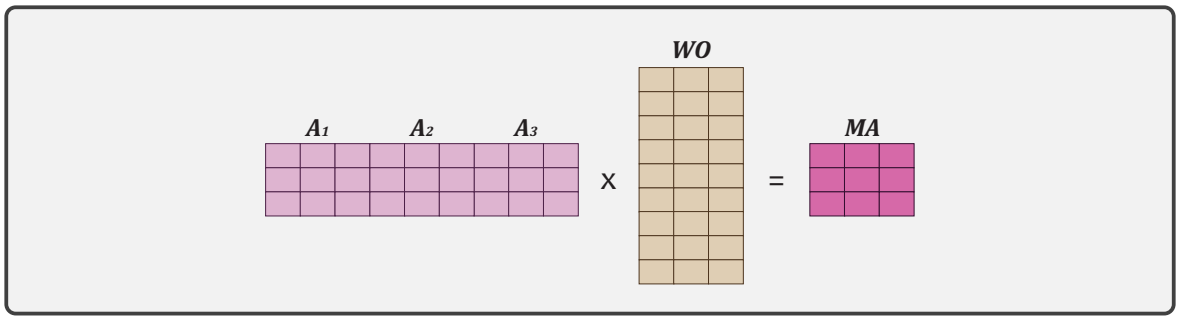

Figura 3.13: Autoatención Multicabezal.

3.3.2.4. Capa Residual

El siguiente componente es la capa residual (Suma y Normalización), la cual aplica una suma de entradas y salidas de la multiatención más una normalización por capa:

$$LayerNorm(X + MA) \tag{3.13}$$

3.3.2.5. Red Neuronal Prealimentada

Una red neuronal prealimentada con una sola capa oculta.

3.3.2.6. Codificador

El codificador se conforma de todos los últimos módulos vistos y que se enlazan de acuerdo a la Figura 3.5, una vez generado un codificador, se pueden apilar tantos codificadores como se requieran. Cabe aclarar, que solo el primer codificador hace uso de los *embeddings* y la codificación posicional, posteriormente, la salida del codificador anterior será la entrada al nuevo.

3.3.2.7. Descodificador

El descodificador, es un generador de texto y es de tipo auto-regresivo, esto quiere decir que va generando la oración con las propias salidas que se obtienen. Puesto que al inicio no se cuenta con alguna palabra, similar a las RNNs se inicia con un token especial de inicio de oración *<bos>* y generará tantas palabras hasta que encuentre un token de fin de oración *<eos>*.

Para generar las palabras, el descodificador se apoya de la salida del codificador, estos dos se enlazan en la sección **Atención Codificador-Descodificador**.

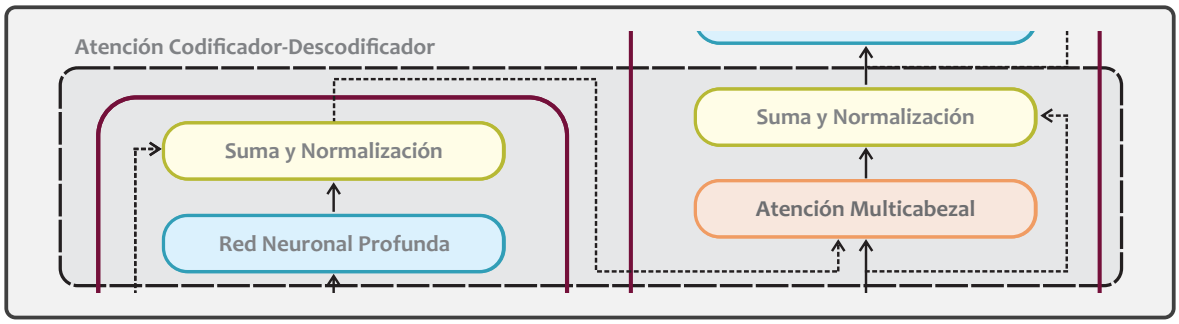

Figura 3.14: Bloque Atención Codificador-Descodificador

Como se sabe, para calcular la autoatención, se requieren de las matrices Q, K y V, las últimas dos se calculan con la salida del codificador y el uso de una capa lineal (transformación lineal con otras dos matrices W_K y W_V). La matriz Q es calculada de la capa previa de auto atención del descodificador (uso de W_Q).

El primer cambio real respecto al codificador, es el **enmascaramiento**. Como el descodificador genera palabra por palabra, este no puede prestar atención a palabras que no ha visto o más bien generado. Por ejemplo, si la oración objetivo es "*<bos>hola como estás*", al calcular la auto atención para la palabra "como", el descodificador no tendría por que conocer la palabra "estás", por que es una palabra futura, la auto atención solo debe considerar la palabra actual y previas.

Figura 3.15: Enmascaramiento

Esta operación no es más que una suma matricial entre el enmascaramiento y el puntaje escalado (previo a la función softmax):

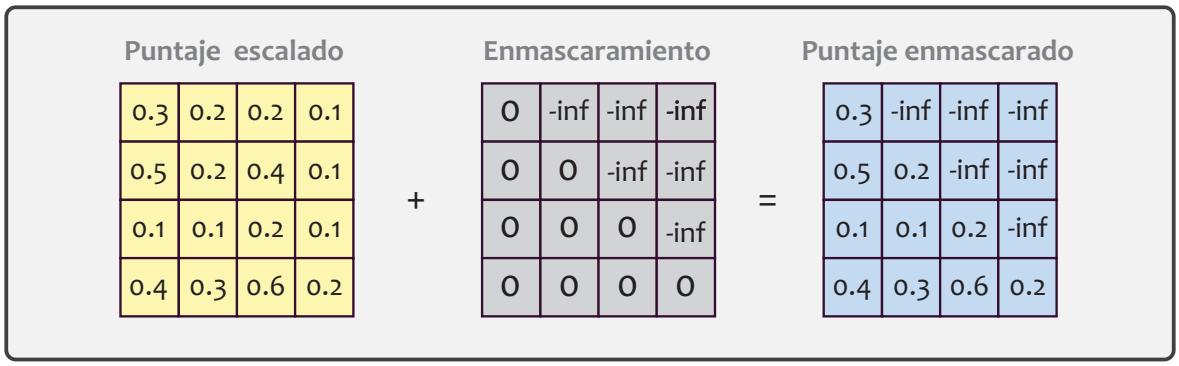

Figura 3.16: Puntaje con enmascaramiento

Al aplicar la función softmax, los *-inf* se volveran ceros, lo que le dice al modelo que no preste atención a dichos tokens. Este enmascaramiento **solo aplica durante el entrenamiento** debido a que es donde se cuenta con la salida (*output target*). El código se puede consultar en el Apéndice A código A.5.

Finalmente, se tiene que convertir la salida a una palabra, para ello se ocupa la última **capa "Lineal + Softmax"**. La capa lineal es una red neuronal que proyecta la salida del descodificador en un vector de una dimensión mucho mayor (tamaño del vocabulario), este vector es conocido como el vector de *logits*. Cada celda del vector da un puntaje para cada palabra y finalmente se convierte a probabilidades mediante la función softmax. La palabra que tenga mayor probabilidad es la que se elige o bien, se implementa algún algoritmo como la búsqueda por haz.

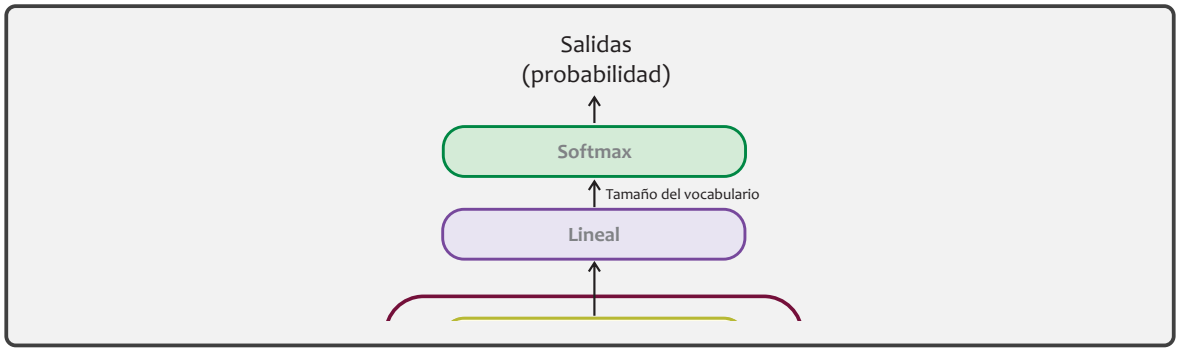

Figura 3.17: Operacion final para el descodificador

3.3.2.8. Entrenamiento vs Prueba

Durante el **entrenamiento**, se tienen que tener las siguientes consideraciones, muy similares a las RNNs:

1. **Corrimiento a la izquierda de la secuencia de salida**: Durante el entrenamiento, se conocen las salidas, estás son proporcionadas al descodificador como entrada y a su vez como secuencia objetivo pero con un corrimiento a la izquierda.

2. **Enmascaramiento:** Como se mencionó previamente, se requiere del enmascaramiento para que el modelo no vea las palabras futuras, y eso es por que en los transformers, toda la secuencia se proporciona en una sola matriz y no paso a paso como en las RNNs.

3. **Función de Costo:** Se utiliza la entropía cruzada, la cual es útil para medir el error entre dos distribuciones de probabilidad.

para las **pruebas**:

1. **Enmascaramiento:** Se desactiva.

2. **Predicción:** Se inicia procesando la secuencia de entrada con el codificador, una vez procesada, se le da al descodificador junto con un token de inicio, la salida que se obtenga se concatena al token de inicio, formando una nueva secuencia de longitud dos, esta secuencia se pasa nuevamente al descodificador junto con la misma salida del codificador y se genera el nuevo token, este nuevo token se concatena al token inicial y al segundo token, formando una secuencia de tres tokens, y el proceso continua hasta encontrar el token de fin de oración.

CAPÍTULO 4

OCELOTL Corpus Paralelo Náhuatl - Español

Con el fin de tener más recursos digitales disponibles para la comunidad científica, se presenta el nuevo corpus paralelo **Ocelotl**. Durante este capítulo, se detalla el proceso por el cual se generó el corpus así como la disponibilidad del mismo.

Figura 4.1: El **ocelote** (del náhuatl *ocelotl*), es una de las seis especies de felinos que se distribuyen en México, incluyendo al jaguar, puma, jaguarundi, gato montés y el tigrillo; una especie reguladora del ecosistema, perseguida para fines comerciales debido a su piel; y un agente importante en la naturaleza como controlador del tamaño poblacional de las presas pequeñas y medianas de las que se alimenta. [**?**]

4.1. Búsqueda de referencias

El primer paso para la creación del corpus, fue la búsqueda de bibliografía. Debido a que el náhuatl es un lenguaje de pocos recursos, el corpus no se limita a una sola variante, sino hace uso de varias de ellas con el fin de tener más bibliografías de las cuales extraer la información. Se utilizaron más de 10 fuentes distintas y a la fecha el corpus se compone de 1515 oraciones.

A continuación, se especifica la referencia, el número de oraciones obtenidas y la variante:

Referencia	**No.**	**Variante**
Aprendamos el idioma Náhuatl (Matinahuatlahtolzalocan) [**?**]	480	Milpa Alta (CDMX)
Atrevete a contar un cuento [**?**]	54	San José Miahuatlán, (Sureste de Puebla)
Canto Cuento Poesía de las niñas y los niños nahuas de Morelia, Michoacán [**?**]	299	Ciudad de Morelia
Chicome xochitl - Siete flor [**?**]	56	No especificado
Ejercicios para el aprendizaje de la lengua náhuatl. Diccionario español-náhuatl de Hueyapan. [**?**]	148	Hueyapan
La Declaración Universal de la UNESCO sobre la Diversidad Cultural [**?**]	48	Estado de Puebla
Manual de Lengua Náhuatl Ma titomachtican totlahtol (Aprendamos nuestra palabra) [**?**]	48	Náhuatl central (Milpa Alta) y náhuatl clásico
Metodología para el Aprendizaje del Náhuatl [**?**]	203	Montaña de Guerrero
Mopataj in tonalmej (El tiempo cambia) [**?**]	32	Estado de Puebla
Tierra de Riquezas [**?**]	75	Acatlán (Chilapa de Álvarez, Guerrero)
Varias	72	Varias

Tabla 4.1: Referencia, número de oraciones y variante por bibliografía.

4.2. Generación de oraciones

Una vez recopilados los libros y transcritos a texto plano, se pasó a la generación de oraciones. Este proceso es complicado debido al siguiente problema:

- **Empalme de gramática y longitud de texto no coincidente entre náhuatl y español**: Dentro de las referencias, existen textos que contienen un número de parrafos y/o oraciones menor en algún idioma, por ejemplo, en un texto en náhuatl se muestran 5 párrafos, sin embargo, en el texto en español este tiene 6 y dentro de alguno de esos parrafos, se listan 4 oraciones en náhuatl, pero 5 en español.

Este problema es complicado de resolver, requiere de la ayuda de un experto en la lengua que analice y fragmente cada uno de estos párrafos que presenten "error". Debido a que no se cuentan con los recursos económicos para contar con los servicios de dicho experto, el problema se abordo mediante el estudio personal de la lengua. Haciendo uso de los conocimientos básicos de gramática, se paso a la fragmentación de párrafos a oraciones. Se buscó que cada oración generada no sobrepasara los 200 caracteres (20 a 30 palabras), una longitud recomendada para la traducción automática, sin embargo, si esto no era posible debido al contexto necesario para que una oración pueda entenderse, se uso la longitud completa de la oración. Además, cada párrafo/texto/oración fue utilizado si no cumplía con alguno de los siguientes apartados:

- **Exclusión por omisión de palabras**: Textos donde una palabra si aparece en un idioma, pero en otro no, fueron omitidos. Por ejemplo, en el texto en náhuatl, se nombra la palabra México, sin embargo, en el texto en español no aparece o aparece con un número menor de incidencias.
- **Exclusión por gramática complicada**: Textos donde la gramática es muy complicada de entender o no queda claro cada oración, son omitidos.

4.2.1. Validación del Corpus

Una vez extraídas las oraciones, se validaron dos puntos:

1. **Limpieza del Corpus**: Se proceso el texto para remover oraciones repetidas y/u oraciones que no tuvieran traducción.
2. **Revisión por Experto**: Una vez generadas las oraciones, el corpus pasó por la validación de un experto. A la fecha, se ha validado alrededor del 47.5 % del total de oraciones. Algunos puntos que se realizaron durante la validación fueron: Evitar coloquialismos que influyeran en la traducción de otras variantes y leves correcciones a la traducción y/u oración original. Por ejemplo, la oración "On poloko kikua sakatl xoxouik" → "El burro como zacate verde", se cambio "como" a "come", quedando la traducción siguiente: "El burro come zacate verde."

De esta forma, se terminó con las 1515 oraciones de la Tabla 4.1.

4.3. Disponibilidad del corpus

Para que el corpus sea de fácil acceso, se creó una página web donde se pudiera consultar. Las tecnologías usadas para la creación de esta página son las siguientes:

- Angular con bootstrap
- Servidor Apache

El corpus está disponible en la siguiente liga:

`www.ocelotl-parallel-corpus.com`

Al ingresar, se presenta la siguiente pantalla:

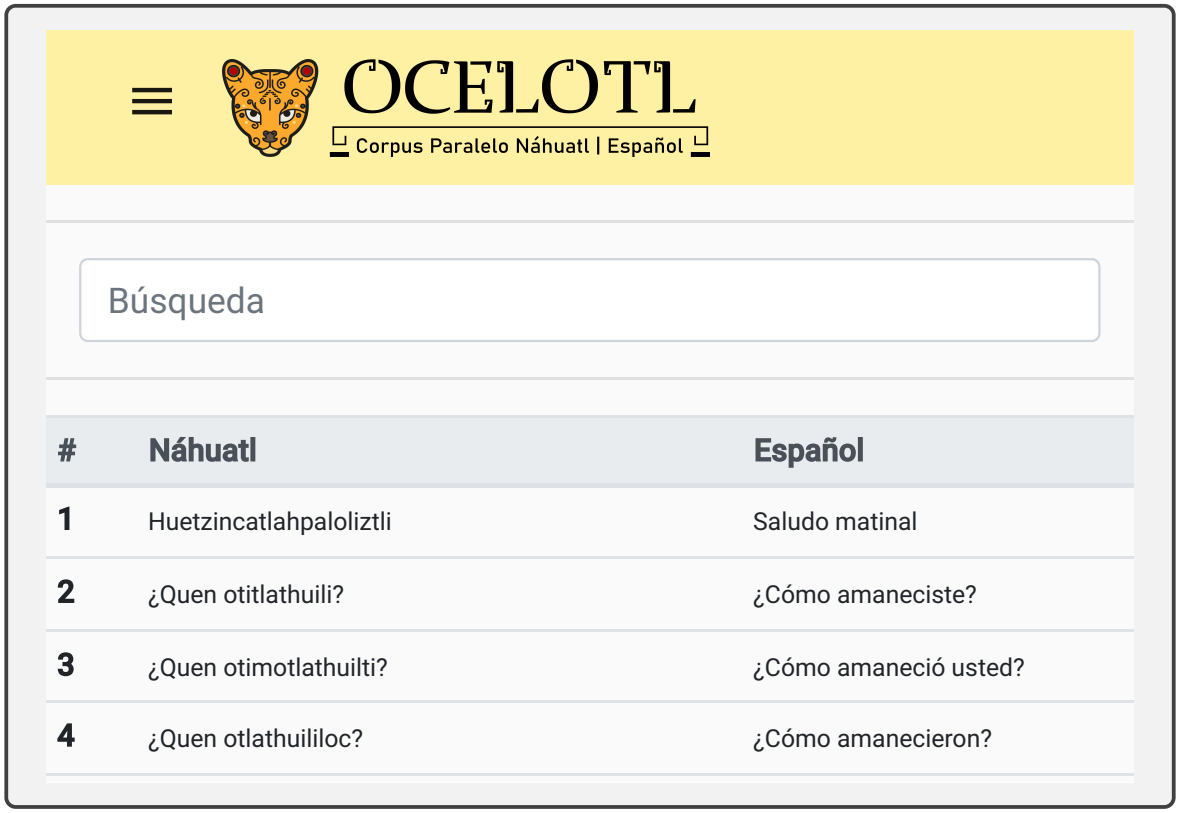

Figura 4.2: Pantalla de inicio

Para buscar una palabra solo ingresela en la "Búsqueda" y presione ENTER. Se buscará la palabra en ambos idiomas. La información se presentará filtrada:

Figura 4.3: Búsqueda de la palabra niño.

El funcionamiento es el mismo en el apartado móvil así como web.

CAPÍTULO 5

OCELOTL Traductor Náhuatl - Español

Durante los primeros capítulos, se presentaron los temas necesarios para abordar la Traducción Automática. En este capítulo, se presenta el traductor de náhuatl a español OCELOTL, usando dichos temas así como el corpus paralelo presentado en el capítulo anterior.

5.1. Composición del Traductor Ocelotl

El flujo del sistema se muestra a continuación:

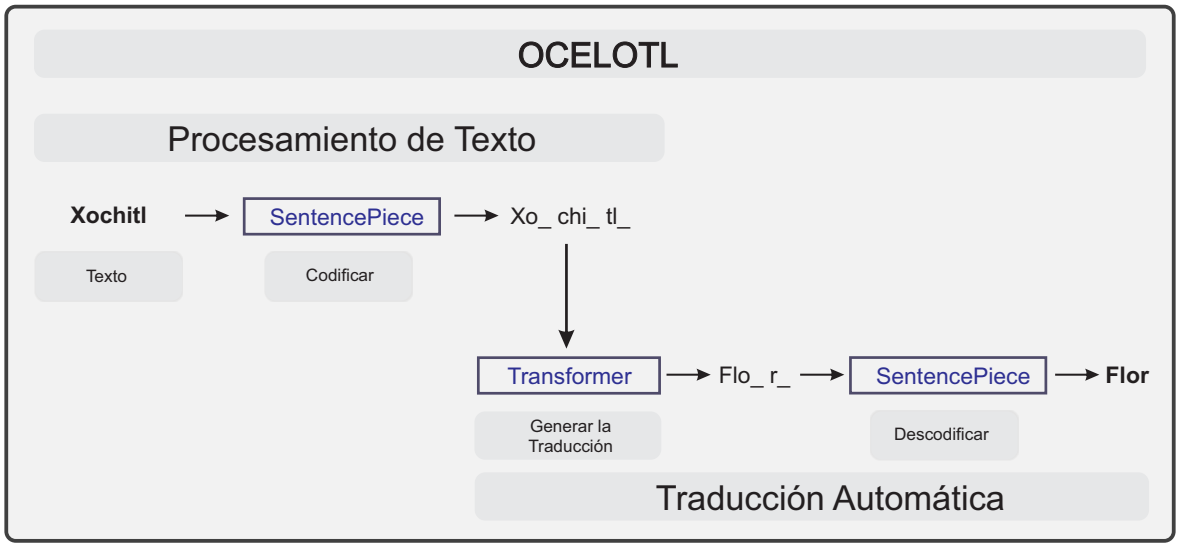

Figura 5.1: Flujo del Traductor Ocelotl.

5.1.1. Procesamiento de Texto

Para poder realizar el procesamiento adecuado de las oraciones, se necesita un modelo para la tokenización o codificación, para ello, se creo un modelo mediante SentencePiece.

5.1.2. Traducción Automática

Una vez tokenizadas las oraciones, se realiza la traducción automática usando un modelo de Transformers, vistos en la sección 3.3.2. En esta parte, también se aplica la búsqueda por haz para dar la traducción. Finalmente, se vuelve a utilizar el modelo de SentencePiece para descodificar la oración a una entendible por el humano.

5.2. Notas de Implementación

El leguaje de programación con el que se implementó el sistema es Python 3.9.12, y las bibliotecas utilizadas son las siguientes:

- **Pytorch 1.12.1**: Para la creación del Transformer.
- **Torchtext 0.6.0**: Para procesamiento de texto.
- **Sentencepiece 0.1.97**: Para la Tokenización.
- **Nltk 3.7.0**: Para el cálculo del puntaje BLEU.

Los parámetros para la **Traducción Automática** son los siguientes:

- Función de error: Entropía cruzada
- Función de activación en las redes: ReLU
- Optimizador: Adam
- Descodificación: Búsqueda por Haz

Cada uno de los parámetros del transformer se dará más adelante en las pruebas.

5.3. Pruebas Preliminares

Se realizaron pruebas preliminares para ver el comportamiento del sistema. Para estas pruebas, se usó una versión preliminar del corpus con 1001 oraciones, en el cual no se incluyen las referencias [**?**], [**?**], [**?**], [**?**] y [**?**], así como un menor número de oraciones en los demás textos. Para la traducción automática neural, se recomiendan oraciones de 20 palabras a lo mucho 40 [**?**], debido a ello, se excluyen oraciones con un tamaño mayor a 300 caracteres (alrededor de unas 30 palabras), quedando en 972 oraciones y usando el 90 % para entrenamiento. Para estas pruebas, se creo un modelo de SentencePiece usando el corpus Axolotl.

Los parámetros para el **Procesamiento de Texto** son los siguientes:

- Longitud de vocabulario para SentencePiece en náhuatl: 1064.
- Longitud de vocabulario para SentencePiece en español: 8000.

5.3.1. Ambiente de Pruebas

Inicialmente se planteó usar el Laboratorio Nacional de Supercómputo del Sureste de México, se realizaron los tramites y solicitudes de bibliotecas necesarios, sin embargo, debido a la compatibilidad entre el programa y el sistema del LNS, al final no fue posible usarlo.

Posteriormente, se busco alguna otra alternativa como computo en la nube mediante máquinas virtuales rentadas (runpod, google TPU, google colab, Azure, AWS, etc), se realizaron varias pruebas en algunos de estos ambientes , sin embargo, o presentaban complicaciones en su manejo o bien los planes resultaban costosos, por lo que no era viable económicamente y también se descartaron dichas opciones.

Finalmente, se encontró la plataforma Kaggle, la cual en su versión gratuita te presta una Tarjeta de Video P100 de 16Gb hasta por 30 horas a la semana. Esta plataforma presenta algunas complicaciones como: No se permite ejecutar el código por las 30 horas continuas, ya que automáticamente se interrumpe alrededor de las 10 horas, por lo que el código debe de ser aprueba de fallas. Para volver a ejecutar el código, manualmente se debe de descargar el resultado más reciente del modelo y volver a cargarlo para posteriormente ejecutar el código, este proceso se debe de hacer cada vez que Kaggle interrumpe la ejecución. Pese a ello, fue la mejor plataforma que se encontró en línea y con la cual se realizan las pruebas.

5.3.2. Pruebas

En la siguiente tabla se muestran las pruebas realizadas al momento, con cada uno de sus respectivos parámetros:

NO	EMB.	H.	ANN	B.	E.	D.	LR	EP.	B.T	B.TR	T.
1	512	8	512	32	3	3	.0001	100	.04647	.91360	13
2	512	8	512	32	3	3	.0001	250	.04522	.92774	32
3	128	4	128	32	2	2	.0001	200	.01445	.12220	6
4	128	4	128	32	2	2	.0001	500	.03133	.91667	14
5	64	2	64	16	2	2	.0001	200	.00000	.01952	4
6	64	2	64	16	2	2	.0001	500	.01931	.25083	12
7	64	2	64	16	2	2	.0001	1000	.02294	.65501	23
8	64	2	64	16	2	2	.0001	2000	.03301	.91390	47

donde:

- EMB: Tamaño de los word embeddings.
- H: Número de cabezales.
- ANN: Tamaño de la red neuronal.
- B: Tamaño del batch.
- E: No. de codificadores.
- D: No. de descodificadores.
- LR: Learning Rate.
- EP: Número de épocas.
- B.T: Puntaje BLEU en el set de pruebas.
- B.TR: Puntaje BLEU en el set de entrenamiento.
- T: Tiempo en minutos de la ejecución.

De estos primeros resultados, se observa que los modelos 6 al 8 no presentan ninguna mejora en relación al modelo 4, ya que le llevo más épocas y tiempo al modelo 8 equiparar al modelo 4. Por lo que, en las pruebas siguientes no se crearan modelos con esas características. El modelo 1 tiene el mejor resultado basándose en el puntaje para

las pruebas, a continuación se listan unas traducciones con dicho modelo:

Correctas

->Náhuatl: Ticateh ipan yeyi imman yohuatzincatica
->Español: Son las tres de la mañana
->Traducción: Son las tres de la mañana

->Náhuatl: Cenca cualli, tlazohcamati
->Español: Muy bien gracias
->Traducción: Muy bien gracias

Incorrectas

->Náhuatl: Kojtik on kuakuauej.
->Español: Es fuerte el toro.
->Traducción: Fregade vo en el ro.

->Náhuatl: Nika ikuak tompaxaloa miyek onka kampa tetlakualtilo.
->Español: Aquí cuando la gente va a pasear va a restaurantes.
->Traducción: Te nos hijó la gente cuenta.

Algunas oraciones humanamente correctas que el sistema puede no dar un puntaje alto debido a la falta de referencias:

->Náhuatl: Zazocampa tlaxcalnamacah
->Español: Por todos lados venden tortillas
->Traducción: Por donde quiera venden tortillas

->Náhuatl: Nantlacuicah Mexihco
->Español: Ustedes cantan en México
->Traducción: Ustedes irán a cantar.

Tomando de base estos resultados preliminares, se planea realizar lo siguiente para nivelar los puntajes y evitar un sobreajuste:

- Reducir el tamaño de los EMBEDDINGS ubicándose en un número entre el modelo 1 y el modelo 4.
- Reducir el tamaño de las ANNs.
- Variar el número de batches.
- Variar el tamaño del set de entrenamiento y pruebas.

- Aumentar el tamaño del corpus para tener más oraciones para el entrenamiento y pruebas.

Debido a las características de la tarjeta de video, no se puede aumentar el número de codificadores y/o descodificadores, siendo 3 el tope máximo.

Posteriormente, se realizaron pruebas con el corpus completo, filtrando las oraciones arriba de 300 caracteres, quedando 1490 oraciones, así como generar el modelo de sentencepiece con el propio corpus Ocelotl, generando un vocabulario de 2992 símbolos para el español y 1080 símbolos para el náhuatl. Adicionalmente, se uso un porcentaje del corpus para la validación.

A continuación se listan los mejores modelos encontrados con estas configuraciones (se omite el B.TR debido a que lo importante es el B.T):

NO	**EMB.**	**H.**	**ANN**	**B.**	**E.**	**D.**	**LR**	**EP.**	**B.T**	**%E**	**%V**	**T.**
1	256	4	256	32	2	2	0.0001	300	0.03049	90	10	15
2	512	8	512	16	3	3	0.0001	100	0.02506	90	0	16
3	512	8	512	64	3	3	0.0001	100	0.02292	80	10	4
4	256	4	256	32	3	3	0.0001	300	0.02281	90	10	19
5	512	8	512	16	2	2	0.0001	100	0.02280	80	0	14
6	256	4	256	32	2	2	0.0001	200	0.02134	85	15	8
7	512	8	512	16	3	3	0.0001	100	0.02074	85	0	16
8	512	8	512	32	3	3	0.0001	200	0.02036	90	10	8

donde:

- %E: Porcentaje del corpus para el entrenamiento.
- %V: Porcentaje del corpus para la validación.

En esta tabla, se pueden apreciar modelos muy parecidos, sin embargo, ninguno de ellos sobrepasa el 3 % en la evaluación del puntaje BLEU, además, la validación no aporta mayor significado debido a que lo importante es el Puntaje BLEU en el data set de pruebas y no precisamente el error final del modelo, es por esto que, similar a la tabla anterior, dicho error se omite. En estas segundas pruebas, el puntaje BLEU queda por debajo de las primeras pruebas, sin embargo, se debe de considerar que el tamaño del corpus es mayor así como nuevas variantes añadidas.

5.4. Búsqueda Armónica para Hiperparámetros

En distintos modelos, como redes neuronales, transformers, entre otros, se cuenta con distintos hiperparámetros a fijar, y a medida que este número de parámetros crece, se torna difícil seleccionar el conjunto de ellos que obtenga el mejor resultado, más aún, al ser modelos no deterministas, cada ejecución, inclusive con los mismos parámetros producirá un resultado diferente. El segundo tema es intrínseco a los propios modelos, sin embargo, con respecto al primer tema, en lugar de intuir los valores de los hiperparámetros, se propone usar la búsqueda armónica para realizarlo de manera automática.

5.4.1. Intervalos y Ancho de banda

Como se vio en el Algoritmo 1, al momento de generar un nuevo individuo, se requiere de un intervalo del cual extraer nuevos datos así como de un ancho de banda para modificar valores y permitir la exploración de otras soluciones. Para el caso de los transformers, cada uno de sus hiperparámetros cuenta con su propio rango así como ancho de banda. La Tabla 5.1 muestra los hiperparámetros con sus respectivos valores para estos dos datos.

5.4.2. Hiperparámetros adicionales

La mayoría de los parámetros de la Tabla 5.1 han sido introducidos en las secciones anteriores salvo los últimos tres. SEED es un hiperparámetro que permite reproducir un mismo resultado aún se ejecute varias veces algún proceso ya que genera los mismos aleatorios en cada ejecución. En las pruebas preliminares, al dividir el corpus en el conjunto de entrenamiento y pruebas, se uso SEED = 0, entonces, en cada prueba realizada se usó el mismo conjunto de datos. Para la búsqueda armónica, es preferible probar distintas divisiones de corpus (pliegues) en cada ejecución. En esas mismas pruebas, se mencionó que se usaron 1080 símbolos para el náhuatl (idioma fuente) y 2992 para el español (idioma destino) para generar el modelo de sentencepiece, a fin de generar distintos vocabularios, estos dos datos se usan como hiperparámetros a través de un porcentaje, este porcentaje nos indica cuantos símbolos tomar del total posible, estos son los parámetros V_SRC_% y V_TGT_%. Por ejemplo, usando el corpus Ocelotl, el número de símbolos máximo computado por sentencepiece para el náhuatl es de 5876 y para el español de 3729, si V_SRC_% = 70 y V_TGT_% = 90 se crearan vocabularios de 4113 y 3356 respectivamente. En otras palabras, cada que se genere un nuevo individuo, se usaran vocabularios y conjuntos de datos distintos gracias a estos dos hiperparámetros adicionales.

Parámetro	**Descripción**	**Intervalo**	**BW**
EMB	Tamaño de los word embeddings	[64, 128, 256, 512, 1024]	200
H	Número de cabezales	[2-4]	1
ANN	Tamaño de la ANN	[64, 128, 256, 512, 1024]	200
B	Tamaño del batch	[2, 4, 8, 16, 32, 64, 128]	30
E	No. de codificadores	[2-6]	1
D	No. de descodificadores	[2-6]	1
LR	Learning Rate	[0.0001, 0.001, 0.01]	0
EP	Número de épocas	[50, 100, 150, 200, 300, 400, 500]	50
%E	Porcentaje del corpus para el entrenamiento	[.71, .72, .73, .74, .75, .76, .77, .78, .79, .80, .81, .82, .83, .84, .85, .86, .87, .88, .89, .90]	0
SEED	Número con el que se genera la división de test set y train set	[0-2147483647]	100000
V_SRC_%	Porcentaje del total de símbolos posibles del idioma fuente para generar el modelo de sentencepiece	[10- 70]	10
V_TGT_%	Porcentaje del total de símbolos posibles del idioma destino para generar el modelo de sentencepiece	[20- 90]	10

Tabla 5.1: Hiperparámetros necesarios para aplicar la búsqueda armónica. Si un rango es continuo se indica con "-" y si es discreto se usa el separador ",".

5.4.3. Criterios de paro

Durante el entrenamiento de un modelo, se usaron dos criterios de paro con el fin de evitar perdida de tiempo:

1. **Tolerancia de error**: Si el error esta por debajo de 0.01 se detiene.
2. **Tolerancia de épocas**: Si después de 10 épocas no hay mejora en el error, se decrementa el factor de aprendizaje en 10^{-1} y se continua el entrenamiento. Si vuelven transcurrir 10 épocas sin mejora se detiene.

5.4.4. Pruebas

Para las pruebas se usó $HMS = 20$, $HMCR = .95$, $PAR = .30$ y $NI = 4$. Contando las primeras 20 evaluaciones al crear la memoria armónica y las otras 80 por las 4 iteraciones, se revisaron 100 modelos distintos en un tiempo total de 76.72 hrs. En la Tabla 5.2, se muestran las configuraciones de los 8 mejores modelos, así como el factor de aprendizaje final FLR (después de aplicar el decremento del LR), así como la época en la que se detuvo el entrenamiento por la tolerancia de épocas (TE). Si el factor de aprendizaje LR es igual a FLR y el número de epocas EP es diferente a TE, indica que se detuvo por tolerancia de error. En la Tabla 5.3, se muestran los tres hiperparámetros adicionales así como el tiempo en minutos del entrenamiento. Al lado de los porcentajes V_SRC_% y V_TGT_% se encuentra su respectiva conversión a número de símbolos para los modelos de sentencepiece, considerando un máximo de símbolos de 5876 para el náhuatl y de 3729 para el español. Estas pruebas se ejecutaron en una tarjeta de video RTX 3090 con 24GB de VRAM en la plataforma RunPOD.

NO	EMB	H	ANN	B	E	D	LR	FLR	EP	TE	B.T	%E
1	256	4	64	64	3	3	.0001	.0001	500	372	0.05618	90
2	922	2	128	128	3	3	.0001	.0001	500	165	0.05281	90
3	256	4	1024	8	2	4	.0001	.00001	200	146	0.04906	85
4	256	4	256	128	2	4	.0001	.0001	400	400	0.04728	90
5	256	4	1024	8	2	5	.0001	.00001	500	296	0.04251	90
6	1024	4	128	128	2	5	.0001	.0001	500	156	0.03343	76
7	1024	4	128	64	6	3	.0001	.00001	500	168	0.03335	75
8	256	2	1024	64	6	3	.0001	.00001	500	393	0.03238	90

Tabla 5.2: Mejores modelos después de aplicar búsqueda armónica

NO	SEED	V_SRC_%	V_TGT_%	T
1	1517788416	65 = 3819	76 = 2834	8.4
2	292918560	44 = 2585	76 = 2834	12.6
3	82744736	56 = 3290	83 = 3095	16
4	292918560	70 = 4113	33 = 1230	9.4
5	82744736	70 = 4113	33 = 1230	43
6	2147483648	70 = 4113	51 = 1901	12
7	2147483648	56 = 3290	53 = 1976	12
8	1106692224	66 = 3878	90 = 3356	10.7

Tabla 5.3: Resultado de los parámetros adicionales de los mejores modelos

De estas pruebas finales, primero, el tamaño de los *word embeddings* es preferible de 256 y segundo, el número de cabezales predominante es 4. En el tamaño de la red, si bien, hay predominantes, no se descartan todas las opciones, lo mismo ocurre con el tamaño del batch, así como los codificadores y descodificadores, sin embargo, estos últimos 4 parámetros, en los mejores dos modelos coinciden, por lo que esta combinación puede verse como la mejor. El factor de aprendizaje, en todos se inicio con el mismo, y dependiendo el modelo fue reducido, pero bien se puede iniciar con 0.0001 sin ningún problema. Con respecto a las épocas, la mayoría inicio con 500, pero gracias al criterio de paro, muchos de estos modelos se detuvieron antes de llegar a las 500, lo que es muy importante ya que indica un buen ahorro de tiempo para cada modelo. Los porcentajes de datos para el entrenamiento se asume 90 % como el idóneo, ya que es el más predominante así como nuevamente coincidir en los dos mejores modelos. Finalmente, el primer modelo logró un porcentaje BLEU del 5.618 % como el mejor y muy cerca está el segundo modelo con 5.281 %, indicando que los parámetros en los que coinciden influyen en este resultado. En los parámetros adicionales, no hay un claro predominante en ninguno de ellos, pero se puede definir un nuevo intervalo usando el mínimo y máximo de estos resultados para futuras pruebas, para V_SRC_% = [44-70] y para V_TGT_% = [33-90]. Dado que el SEED no es un parámetro del que se pueda saber como exactamente genera los aleatorios, lo preferible es seguir dejando al sistema para que lo genere automáticamente si es que se desea generar otros modelos. Si se observa el tiempo del modelo 5, que es de 43, nos indica que aumentar una capa de descodificador afecta el tiempo, ya que el número de parámetros a optimizar crece, así como la afectación que tiene el batch size ya que aunque es similar el modelo al 6, en el batch existe una diferencia significativa. En cualquier caso, es importante tener los criterios de paro para evitar que el modelo siga entrenándose cuando no va a ofrecer un mejor resultado.

Para poner en contexto estas últimas pruebas, a continuación se muestran los mejores resultados de las pruebas preliminares P1 y P2, así como de la prueba final PF:

NO	EMB	H	ANN	B	E	D	LR	E	B.T	%E	T
PF	256	4	64	64	3	3	0.0001	372	0.05618	90	8.4
P1	512	8	512	32	3	3	0.0001	100	0.04647	90	13
P2	256	4	256	32	2	2	0.0001	300	0.03049	90	15

El modelo PF sobrepasa a los dos previos, más siendo su competidor directo el modelo P2 al cual superó por 2.569 % y si bien, el primero no es competidor directo al ser un tamaño de corpus distinto, igualmente quedó por arriba de el. Dado que los modelos P1 y P2 se ejecutaron en distintos ambientes al PF, no se pueden medir los tiempos directamente, sin embargo, dado que PF y P2 comparten algunas características, se puede asumir serán parecidos en tiempo y como P1 tiene mayor número de cabezales pero se ejecuto en un corpus menor, se asume igual su tiempo puede ser menor. Por lo que los tres pueden estar en tiempos similares.

A continuación se listan algunas traducciones correctas del modelo PF:

->Náhuatl: Huel cualli tlazohcamati
->Español: Muy bien gracias
->Traducción: Muy bien gracias

->Náhuatl: ¿Quen otitlathuili José?
->Español: ¿Cómo amaneciste José?
->Traducción: ¿Cómo amaneciste José?

->Náhuatl: On tlakatl tekipanoua imilaj
->Español: El señor trabaja en su milpa.
->Traducción: El señor trabaja en su milpa.

Algunas parciales:

->Náhuatl: Inin altepetl ¿kenin itoga?
->Español: Este pueblo ¿Cómo se llama?
->Traducción: ¿Cómo se llama?

->Náhuatl: In elotlaxcaltzin yematztic
->Español: Su tortilla de elote de ellos está suave
->Traducción: Su tortilla de elote de ellas está corriosa

->Náhuatl: Ikuak titlami timinchiua chaui tonpaxaloa, nima naua nikuelita ñaas apan Balsas, yonpiya panoa nokalpa San Agustín Oapan, Guerrero, milak nikuelita napanos apan naua.

->Español: Cuando terminamos de hacerlos salimos a pasear y uno de mis lugares favoritos es el río Balsas, que pasa por mi pueblo, llamado San Agustín Oapan, Guerrero, me gusta mucho nadar en ese río.
->Traducción: Cuando terminamos de hacerlos salimos a pasear y uno de mis lugares favoritos es el río Balsas, que pasa por mi pueblo, llamado San Agustín Oapan, Guerrero.

Algunas erróneas:

->Náhuatl: Nantlachipahuaquihuih
->Español: Ustedes vendrán a hacer la limpieza.
->Traducción: Aquí tiene su agüita.

->Náhuatl: Yi tlajka.
->Español: Ya es tarde.
->Traducción: Ya me voy a mi casa.

->Náhuatl: Ninotzontecocoa lpampa ahmo icihuca onitlacua
->Español: Me duele la cabeza porque no comí a la hora
->Traducción: No vino a comer aquí

5.5. Comparativa con estado del arte

Los avances previos en la traducción (ver sección 2.4) se resumen la siguiente tabla:

Referencia	**Martínez-Gil 2012** [?]	**Manger & Meza 2018** [?]		**Bello et al. 2021** [?]
Técnica	Reglas	SMT (MOSES y GIZA++)	NMT (Cod-Desc y GRU BRNN)	Transformers
Resultado	Traducción de sufijos, prefijos y 1514 términos.	Traductor para 5 idiomas originarios diferentes, incluido el náhuatl.		Aplicación móvil para la traducción (no publicada)
Recursos	836 textos	Corpus de 985 frases		Corpus Axolotl -18k Corpus JW - 35k
BLEU	No disponible	10.14 %	3.04 %	47.84 % a 66.45 %
Variante	No especificada	No especificada		Varias

Dado que la primera referencia no cuenta con un puntaje BLEU no existe una comparación directa. El trabajo actual en perspectiva con los restantes para NMT, se ubica entre ambos, quedando de la siguiente forma:

Referencia	**Manger & Meza 2018** [?]	**Pacheco M. 2023**	**Bello et al. 2021** [?]
Técnica	NMT (Cod-Desc y GRU BRNN)	Transformers	Transformers
Resultado	Traductor para 5 idiomas originarios diferentes, incluido el náhuatl.	Traductor Náhuatl - Español	Aplicación móvil para la traducción (no publicada)
Recursos	Corpus de 985 frases	Corpus Ocelotl 1.5k	Corpus Axolotl - 18k Corpus JW - 35k
BLEU	3.04 %	5.6 %	47.84 % a 66.45 %
Variante	No especificada	Varias	Varias

Si bien, se supera al trabajo [?], no se logra superar a la última referencia, sin embargo, se debe considerar el tamaño del corpus y recursos disponibles para la realización del mismo. Independientemente de ello, este trabajo propone lo siguiente:

1. Corpus Ocelotl
2. Búsqueda armónica para hiperparámetros aplicado al náhuatl

Durante este capítulo se desarrollo el traductor así como se presentaron resultados y comparativas. Para cualquier actualización futura se puede consultar la página del corpus.

CAPÍTULO 6

Conclusiones

El objetivo de esta tesis es crear el núcleo de un **traductor de texto de náhuatl a español**. Durante el desarrollo de la misma, se presentaron todos los fundamentos necesarios para lograr dicho objetivo y como se vio en las pruebas, se logró crear un modelo del traductor mediante los transformers, y se mejoraron los resultados preliminares haciendo uso de la búsqueda armónica. Hablando de los recursos, se presentó el nuevo corpus paralelo **Ocelotl**, el cual cuenta con cierto porcentaje de validación por un experto. El corpus se hizo de distintas variantes ya que no es posible centrarse en una sola debido a los pocos recursos digitales existentes.

Es importante tener claros los problemas del traductor: primero, al usar búsqueda armónica, se requiere probar distintos modelos y esto lleva a un aumento en el tiempo, por ello, lo ideal es contar con múltiples GPUs que permitan la evaluación de distintos modelos a la vez; segundo, es la estandarización del idioma, debido a que cada variante puede usar distintos símbolos para la escritura, el traductor se vuelve más complejo al necesitar captar un vocabulario de mayor tamaño. Este problema es ajeno a la computación y a medida que las organizaciones gubernamentales realizan un trabajo sobre la misma estandarización, cualquier sistema computacional tendrá que lidiar con este detalle.

Como trabajo futuro, se planea seguir aumentando el tamaño del corpus, realizar paralelización a múltiples GPUs, así como combinar este corpus con otro y hacer experimentación a mayor escala. Todo esto será posible, a medida que se vayan contando con los recursos económicos y humanos para la mejora del mismo.

APÉNDICE A

Código

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jan 13 19:05:30 2023

@author: Max
"""

# pip install sentencepiece
# https://github.com/google/sentencepiece
import sentencepiece as spm

import csv

root = 'D:/repositorio/ml/Programas/Nahuatl/'

file_csv = root + 'data/axolotl.csv' # Axolotl archivo original
file_txt_s = root + 'data/axolotl_clean_s.txt' # Oraciones espa ol
file_txt_t = root + 'data/axolotl_clean_t.txt' # Oraciones nahuatl

# Se entrena el modelo con el archivo `axolotl_clean_t.txt` que contiene las oraciones en nahuatl
# El entrenamiento genera los archivos `m.model` y `m.vocab`, donde `m.vocab` es solo para referencia, no se ocupa en este ejemplo.
# Se redefine el id para cada uno de los simbolos especiales
# mas informacion en https://github.com/google/sentencepiece seccion Redefine special meta tokens
spm.SentencePieceTrainer.train('--input=' + file_txt_t +' --model_prefix=m --vocab_size=1064 --user_defined_symbols=<bos>,<eos> --pad_id=1 --bos_id=-1 --eos_id=-1')

# Carga el modelo creado (m.model)
sp = spm.SentencePieceProcessor()
sp.load('m.model')

```

```
string = ' kampa  ka nokal?'

# encode: text => id
print(sp.encode_as_pieces(string))
# ['_  ', 'ka', 'mpa', '_ka', '_no', 'ka', 'l', '?']
print(sp.encode_as_ids(string))
# [237, 87, 197, 381, 39, 87, 13, 165]

# decode: id => text
print(sp.decode_pieces(sp.encode_as_pieces(string)))
#   kampa  ka nokal?
print(sp.decode_ids(sp.encode_as_ids(string)))
#   kampa  ka nokal?
```

Listing A.1: Tokenización mediante SentencePiece aplicada al corpus Axolotl

```
# -*- coding: utf-8 -*-
"""Gensim.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1A6_DIX_E3B8QXZI-0-dg-5ulJpTXr5HI
"""

# Imports
import gensim
import nltk

# Descargar modelo pre entrenado
nltk.download('word2vec_sample')

# Cargar modelo
from nltk.data import find
word2vec_sample = str(find("models/word2vec_sample/pruned.word2vec.txt"))
model = gensim.models.KeyedVectors.load_word2vec_format(word2vec_sample, binary=False)

# Se muestran los primeros 10 datos de la representacion de king
model['king'][:10]
'''
array([ 0.0434064 ,  0.0102628 ,  0.00296526,  0.0481172 , -0.00883269,
       -0.0124499 ,  0.0385274 , -0.0683062 ,  0.0176654 ,  0.125172  ],
      dtype=float32)
'''

# Palabras similares a rey
model.most_similar(positive=['king'], topn = 3)
```

```
'''
[('kings', 0.7138044834136963),
 ('queen', 0.6510958075523376),
 ('monarch', 0.6413194537162781)]
'''

# king - man + woman = queen
model.most_similar(positive=['king', 'woman'], negative=['man'])
'''
[('queen', 0.7118192911148071),
 ('monarch', 0.6189673542976379),
 ('princess', 0.5902431011199951),
 ('prince', 0.5377321243286133),
 ('kings', 0.5236842632293701),
 ('queens', 0.5181134939193726),
 ('throne', 0.5005807876586914),
 ('royal', 0.493820458650589),
 ('ruler', 0.4909273982048034 7),
 ('princes', 0.481081485748291)]
'''
```

Listing A.2: WordEmbeddings mediante Gensim

```
# -*- coding: utf-8 -*-
"""PositionalEncoding.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/16faLAiDYBLOe3tqa7B_lQ-yjiKuYhNQ9
"""

import numpy as np
import matplotlib.pyplot as plt

def PE(d, n, L):
    i_range = int(d/2)
    pos_encoding = np.zeros((L, d))
    for k in range(L):
        for i in range(i_range):
            w_i = 1 / np.power(n, 2*i/d)
            pos_encoding[k, 2*i] = np.sin(k * w_i)
            pos_encoding[k, 2*i+1] = np.cos(k * w_i)
    return pos_encoding

n = 10000 #100
L = 10 #4
d = 64 #4

pos_encoding = PE(d, n, L)

```

```
# PLOTS
plt.figure(figsize=(12,8))
plt.yticks(np.arange(L+1)) # Solo muestre enteros
plt.pcolormesh(pos_encoding, cmap="viridis")
plt.xlabel("Embedding Dimensions")
plt.xlim((0, d))
plt.ylim((L,0))
plt.ylabel("Token Position")
plt.colorbar()
plt.savefig("PE.pdf", transparent=True, bbox_inches="tight")
plt.show()
```

Listing A.3: Codificación Posicional

```
# -*- coding: utf-8 -*-
"""
Created on Mon Oct 17 22:35:30 2022

@author: Max
"""
import numpy as np
from scipy.special import softmax

# Calculo guiado de https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

x = np.array([[1, 0, 1, 0] , [0, 2, 0, 2], [1, 1, 1, 1]]);

wk =  np.array([[0, 0, 1], [1, 1, 0], [0, 1, 0], [1, 1, 0]]);

wq = np.array([[1, 0, 1], [1, 0, 0], [0, 0, 1], [0, 1, 1]]);

wv = np.array([[0, 2, 0], [0, 3, 0], [1, 0, 3], [1, 1, 0]]);

q = x.dot(wq)
k = x.dot(wk)
v = x.dot(wv)

s = q[0].dot(k.T)

ss = softmax(s)

# Se debe de multiplicar cada score por un vector value, por eso se usa v.T,
# y una vez multiplicado se regresa con .T para que se continue correctamente
z = (ss * v.T).T

o = np.sum(z, axis = 0)

# O en una sola operacion (ver archivo Notas>Transformers>Self-Attention)
```

```
zo = ss.dot(v)

# Para obtener los outputs de cada uno, se puede realizar lo siguiente
Z = softmax(q.dot(k.T), axis = 1).dot(v)
```

Listing A.4: Autoatención

```
# -*- coding: utf-8 -*-
"""
Created on Tue Feb 14 00:02:01 2023

@author: Max
"""

import torch

# https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html
def generate_square_subsequent_mask(sz):
    mask = (torch.triu(torch.ones((sz, sz)) == 1).transpose(0, 1))
    mask = mask.float().masked_fill(mask == 0, float('-inf')).masked_fill(mask == 1, float(0.0))
    return mask


ps = torch.tensor([[.3, .2, .2, .1],
                   [.5, .2, .4, .1],
                   [.1, .1, .2, .1],
                   [.4, .3, .6, .2]])

print(ps)

mask = generate_square_subsequent_mask(ps.shape[0])

print(mask)

#tensor([[0., -inf, -inf, -inf],
#        [0., 0., -inf, -inf],
#        [0., 0., 0., -inf],
#        [0., 0., 0., 0.]])

pe = ps + mask

print(pe)

#tensor([[0.3000,    -inf,    -inf,    -inf],
#        [0.5000, 0.2000,    -inf,    -inf],
#        [0.1000, 0.1000, 0.2000,    -inf],
#        [0.4000, 0.3000, 0.6000, 0.2000]])
```

Listing A.5: Enmascaramiento

Siglas

BLEU BiLingual Evaluation Understudy (BiLingual Evaluation Understudy).

BPE Byte-Pair Encoding (Byte-Pair Encoding).

BRNN Bidirectional Recurrent Neural Networks (Redes Neuronales Recurrentes Bidireccionales).

BS Beam Search (Búsqueda por Haz).

BW Bandwidth (Ancho de Banda).

GRU Gated Recurrent Unit (Unidades Recurrentes con Compuertas).

HM Harmony Memory (Memoria Armónica).

HMC Harmony Memory Consideration (Consideración de la Memoria Armónica).

HMCR Harmony Memory Consideration Rate (Umbral de Consideración de la Memoria Armónica).

HMS Harmony Memory Size (Tamaño de la Memoria Armónica).

NMT Neural Machine Translation (Traducción Automática Neuronal).

PA Pitch Adjustment (Corrección de Tono).

PAR Pitch Adjustment Rate (Umbral de Corrección de Tono).

RNN Recurrent Neural Network (Red Neuronal Recurrente).

SMT Statistical Machine Translation (Traducción Automática Estadística).