



UNIVERSIDAD MICHOCANA DE SAN NICOLÁS DE
HIDALGO

FACULTAD DE CIENCIAS FÍSICO-MATEMÁTICAS "MAT.
MANUEL RIVERA GUTIERREZ"

TRADUCTOR AUTOMÁTICO DE LA LENGUA PURÉPECHA AL
ESPAÑOL USANDO REDES TRANSFORMER

T E S I S

QUE PARA OBTENER EL GRADO DE:

LICENCIADA EN CIENCIAS FÍSICO-MATEMÁTICAS PRESENTA:

CECILIA GONZÁLEZ SERVÍN

DIRECTOR DE TESIS

DRA. KARINA MARIELA FIGUEROA MORA

Morelia, Mich. Noviembre 2022



Índice general

0.1	Notación	6
1	Introducción	9
2	Antecedentes	11
2.1	Estado del arte	11
2.1.1	Traducción Automática	11
2.1.2	Clasificación de los sistemas	12
2.1.3	Traducción Automática basada en reglas (RBMT)	12
2.1.4	Traducción Automática Estadística (Statistic Machine Traslation)	13
2.1.5	Traducción Automática Neuronal (NMT)	13
3	Marco teórico	16
3.1	Conceptos Básicos Matemáticos	16
3.1.1	Vectores, matrices y tensores	16
3.1.2	Multiplicación de Matrices	17
3.1.3	Incrementos, derivada y derivadas parciales	18
3.1.4	Gradiente	20
3.1.5	Regla de la cadena	20
3.2	Tokenización	21
3.2.1	Codificación de pares de bytes (BPE)	21
3.2.2	One hot encodding	22
3.2.3	Word embedding	22
3.3	Redes Neuronales Artificiales	24
3.3.1	Neurona	24
3.3.2	Función de activación	25
3.3.3	Función de pérdida	25
3.3.4	Descenso de gradiente estocástico	25
3.3.5	Backpropagation	27
3.4	Procesamiento del lenguaje natural	29
3.5	Redes Transformer	30
3.5.1	Componentes principales del modelo	30
3.5.2	De palabras a vectores	31
3.5.3	Codificador	32
3.5.4	Autoatención	33
3.5.5	Autoatención en detalle	34

3.5.6	Vectores Query, Key y Value	35
3.5.7	Cálculo matricial de autoatención	37
3.5.8	Atención de múltiples cabezas	37
3.5.9	Representar el orden de la secuencia mediante codificación posicional	39
3.5.10	Los residuos	42
3.5.11	Decodificador	43
3.5.12	La capa final lineal y Softmax	44
3.5.13	Resumen del entrenamiento	45
4	Bases conceptuales del purépecha	50
4.1	Las consonantes	50
4.2	Las vocales	51
4.3	El acento	51
4.4	Flexión verbal	52
4.5	Los aspectos	52
4.5.1	El aspecto aoristo (o perfecto)	52
4.5.2	El aspecto progresivo	53
4.5.3	El aspecto habitual	53
4.6	Los modos	54
4.6.1	El modo asertivo (o indicativo)	54
4.6.2	El modo interrogativo	55
4.7	Los tiempos	56
4.7.1	El tiempo presente	57
4.7.2	El tiempo pasado	57
4.8	Los clíticos personales	57
5	Materiales y Metodología	59
5.1	Materiales	59
5.1.1	Google Colab	59
5.1.2	JoeyNMT	60
5.2	Corpus	60
5.3	Algoritmo para la construcción automática del corpus	60
5.3.1	Primera parte: Conjugación de verbos en Purépecha	61
5.3.2	Segunda parte: Conjugación de verbos en Español	61
5.4	Selección de datos y Entrenamiento del modelo	63
5.4.1	Preparación de datos	63
5.4.2	Preprocesamiento	63
5.4.3	Configuración	64
5.4.4	Sección de datos	65
5.4.5	Sección de Capacitación	65
5.4.6	Sección de Pruebas	66
5.4.7	Sección modelo	66
5.4.8	Entrenamiento	67
5.5	Validación del modelo	68

5.5.1	N-gramas	68
5.5.2	BLEU	68
5.5.3	Perplejidad	70
5.5.4	Tasa de aprendizaje	71
6	Experimentación y evaluación	72
6.1	Informes de validación	72
6.2	Curvas de aprendizaje	72
6.2.1	Atención Visualización	74
6.2.2	Evaluación del conjunto de pruebas	85
7	Conclusiones	86

Índice de figuras

2.1	Etapas de la traducción automática basada en reglas: análisis, transición y generación <i>La traducción automática basada en reglasS (RBMT)</i> (s.f.)	13
2.2	Etapas detalladas de <i>La traducción automática basada en reglasS (RBMT)</i> (s.f.)	14
3.1	Derivada de f en x_0 , ?.	19
3.2	One hot encodding, <i>One hot encoding in TensorFlow (tf.one_hot)</i> (s.f.)	22
3.3	Las representaciones de palabras obtenidos de one hot encodding son escasos, de alta dimensión y codificados de forma rígida, Chollet (2017) .	23
3.4	Las incrustaciones de palabras son densas, de dimensión relativamente baja y se aprenden con base en los datos, Chollet (2017)	23
3.5	Curva de pérdida 1D, Chollet (2017)	27
3.6	Superficie de pérdida 2D, Chollet (2017)	27
3.7	Toma una oración de entrada en un idioma y genera su traducción en el otro idioma, <i>The Illustrated Transformer</i> (s.f.)	30
3.8	Bloques: codificador y decodificador, <i>The Illustrated Transformer</i> (s.f.) . . .	31
3.9	Estructura del codificador y decodificador, <i>The Illustrated Transformer</i> (s.f.)	32
3.10	Subcapas de los codificadores, <i>The Illustrated Transformer</i> (s.f.)	33
3.11	Estructuras de un codificador y un decodificador, <i>The Illustrated Transformer</i> (s.f.)	33
3.12	Cada palabra está incrustada en un vector de tamaño 512. Representaremos esos vectores con estos simples cuadros, <i>The Illustrated Transformer</i> (s.f.).	34
3.13	Las palabras incrustadas pasan por las dos capas del codificador, <i>The Illustrated Transformer</i> (s.f.).	34

3.14	La lista de vectores es procesada por las dos capas del codificador, produciendo una salida que será la entrada del próximo codificador, <i>The Illustrated Transformer</i> (s.f.).	35
3.15	Multiplicar x_1 por la matriz de peso W^Q produce q_1 , el vector de “consulta” asociado con esa palabra. Terminamos creando una “consulta”, una “clave” y una proyección de “valor” de cada palabra en la oración de entrada, <i>The Illustrated Transformer</i> (s.f.).	36
3.16	La puntuación es el producto escalar del vector consulta con el vector clave, <i>The Illustrated Transformer</i> (s.f.).	37
3.17	Las puntuaciones se dividen entre 8, después se pasan a una función softmax que las normaliza.	38
3.18	Suma de los vectores que produce la salida de la capa de autoatención, <i>The Illustrated Transformer</i> (s.f.).	39
3.19	Cada fila de la matriz X corresponde a una palabra en la oración de entrada. Nuevamente vemos la diferencia en el tamaño del vector de incrustación (512 o 4 cajas en la figura) y los vectores $q/k/v$ (64 o 3 cajas en la figura), <i>The Illustrated Transformer</i> (s.f.).	40
3.20	El cálculo de la autoatención en forma de matriz, <i>The Illustrated Transformer</i> (s.f.).	41
3.21	Cabezas de atención	42
3.22	8 matrices Z , <i>The Illustrated Transformer</i> (s.f.).	43
3.23	Matriz Z_n concaatenads y multiplicada por W^O , <i>The Illustrated Transformer</i> (s.f.).	44
3.24	Proceso con todas las matrices, <i>The Illustrated Transformer</i> (s.f.).	45
3.25	La codificación posicional de 128 dimensiones para una oración con una longitud máxima de 50. Cada fila representa el vector de incrustación \vec{p}_t	46
3.26	Arquitectura de un codificador, <i>The Illustrated Transformer</i> (s.f.).	47
3.27	Capa de suma y normalización, <i>The Illustrated Transformer</i> (s.f.).	47
3.28	2 codificadores y 2 decodificadores, <i>The Illustrated Transformer</i> (s.f.).	48
3.29	Esta figura comienza desde abajo con el vector producido como salida de la pila de decodificadores. Luego se convierte en una palabra de salida, <i>The Illustrated Transformer</i> (s.f.).	49
4.1	Vocales, referencia:Chamoreau (2009)	51
4.2	El acento, referencia:Chamoreau (2009)	51
4.3	Flexión verbal básica	52
4.4	Los clíticos personales, referencia:Chamoreau (2009)	58
5.1	Comando para construir el vocabulario	63
5.2	El primer elemento en el vocabulario	64
5.3	Rutas de los diferentes datos y parámetros del la lengua origen.	64
5.4	Parámetros de la lengua destino.	65
5.5	Caption	66
5.6	Estrategia de decodificación durante la prueba mas otros parámetros importantes	66

5.7	Parámetros para el codificador	67
5.8	Parámetros para el decodificador	67
5.9	Comando para entrenar el modelo	67
5.10	N-gramas, ¿Qué es un N-Gram? (s.f.)	68
5.11	En este ejemplo encontramos cuatro palabras coincidentes y una genera- ción tiene cinco palabras, <i>BLEU metric</i> (s.f.)	69
5.12	La palabra six aparece una vez en la referencia, <i>BLEU metric</i> (s.f.).	69
5.13	Calculando la precisión modificada de un cuatrigrama, <i>BLEU metric</i> (s.f.).	70
6.1	Gráfica BLEU del primer modelo. Épocas: 50. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 1000. Tamaño de incrustación: 128. Tamaño en la capa oculta: 128. ff_size: 522. Traducciones correctas: 310 de 1000 datos del test.	73
6.2	Gráfica PPL del primer modelo.	74
6.3	Gráfica BLEU del segundo modelo. Épocas: 100. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 1000. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 198 de 1000 datos del test.	75
6.4	Gráfica PPL del segundo modelo.	76
6.5	Gráfica BLEU del tercer modelo. Épocas: 50. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 100. Tamaño de incrustación: 64. Ta- maño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 246 de 1000 datos del test.	77
6.6	Gráfica PPL del tercer modelo.	78
6.7	Gráfica BLEU del cuarto modelo. Épocas: 50. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 2000. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 246 de 1000 datos del test.	79
6.8	Gráfica PPL del cuarto modelo	80
6.9	Gráfica de BLEU del quinto modelo. Épocas: 100. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 1000. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 202 de 1000 datos del test.	81
6.10	Gráfica PPL del quinto modelo	82
6.11	Gráfica BLEU del modelo creado a partir de un corpus diferente	82
6.12	Gráfica PPL del modelo creado a partir de un corpus diferente	83
6.13	Caption	83
6.14	Visualización de atención a las partes que conforma la traducción de la flexión verval itsuta-s-p-ki-ri (¿fumaste?).	84
6.15	Visualización de atención a las partes que conforma la traducción de la flexión verval kará-s-p-ka-ni (escribí).	84
6.16	Comando para evaluar el conjunto de pruebas.	85
6.17	Evaluación del conjunto de pruebas y desarrollo	85

0.1. Notación

Números y matrices

a	Un escalar (entero o real)
\mathbf{a}	un vector
A	Una matriz A
\mathbf{A}	un tensor
\mathbf{I}_n	En matriz de identidad con n filas y n columnas
\mathbf{I}	Matriz identidad con dimensionalidad implícita por contexto
$\text{diag}(\mathbf{a})$	Una matriz diagonal cuadrada con entradas diagonales dadas por \mathbf{a}

Conjuntos y Gráficos

\mathbb{R}	El conjunto de los números reales
$\{0, 1\}$	El conjunto que contiene 0 y 1
$\{0, 1, \dots, n\}$	El conjunto de todos los enteros entre 0 y n , $b > a$, b
	El intervalo real que incluye a y b
$(a, b]$	El intervalo real excluyendo a pero incluyendo b

Indexación

\mathbf{a}_i	Elemento i del vector \mathbf{a} , con indexación a partir de 1
\mathbf{a}^i	Todos los elementos del vector \mathbf{a} excepto el elemento i
$A_{i,j}$	Elemento i, j de la matriz A
$A_{i,:}$	Fila i de la matriz A
$A_{:,i}$	Columna i de la matriz A
$A_{i,j,k}$	Elemento (i, j, k) de un tensor A tridimensional

Operaciones de álgebra lineal

A^T	Transpuesta de la matriz A
$\det(A)$	Determinante de A

Cálculo

$\frac{dy}{dx}$	Derivada de y con respecto a x
$\frac{\partial y}{\partial x}$	Derivada parcial de y con respecto a x
$\nabla_{\mathbf{x}} y$	Gradiente de y con respecto a \mathbf{x}
$\nabla_{\mathbf{X}} y$	Derivadas matriciales de y con respecto a \mathbf{X}
$\nabla_{\mathbf{X}} y$	Tensor que contiene derivadas de y con respecto a \mathbf{X}
$\frac{\partial f}{\partial \mathbf{x}}$	Matriz jacobiana $\mathbf{J} \in \mathbb{R}^{m \times n}$ de $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$

Funciones

$f : \mathbb{A} \rightarrow \mathbb{B}$	La función f con dominio A y rango B
fg	Composición de las funciones f y g
$\log x$	Logaritmo natural de x
$\ x\ _p$	L^p norma de x
$\ x\ $	L^2 norma de x

Resumen. A pesar de los avances o herramientas tecnológicas desarrolladas, no existe suficiente trabajo sobre nuestras lenguas indígenas a nivel mundial por muchas razones. Uno de ellos es el pequeño corpus existente. El presente trabajo de investigación trata sobre un traductor automático de oraciones simples en lengua purépecha al español. Además, se presenta un generador automático de corpus (oraciones pareadas en ambos idiomas) de oraciones simples. El traductor automático se está desarrollando utilizando redes Transformers, que nació en el contexto del procesamiento del lenguaje natural. Este procesamiento es parte de la inteligencia artificial que busca permitir que las computadoras comprendan, interpreten y manipulen el lenguaje humano.

Abstract. Despite the advances or technological tools developed, there is not enough work on our indigenous languages worldwide for many reasons. One of them is the small existing corpus. This research paper deals with an automatic translator of simple sentences in the Purépecha language into Spanish. In addition, an automatic corpus generator (paired sentences in both languages) of simple sentences is presented. The machine translator is being developed using Transformers networks, which was born in the context of natural language processing. This processing is part of artificial intelligence that seeks to allow computers to understand, interpret and manipulate human language.

Keywords: Traductor Automático, Redes Transformer, Procesamiento del Lenguaje Natural, Purépecha-Español, Generador Automático de Corpus.

Capítulo 1

Introducción

El idioma o lengua Purépecha, también conocido como Tarasco se habla en el estado de Michoacán México, actualmente tiene 41,177 hablantes de Lingüística Aplicada (s.f.). El número de hablantes originarios de esta lengua se reduce cada vez más rápido, debido a la predominación cultural del idioma español también a la falta de herramientas que logren una mejor comunicación entre hablantes y no hablantes, así como a diversos factores sociales.

Con la idea de motivar la recuperación del idioma Purépecha se propone un sistema de traducción entre las lenguas purépecha|español, para todos los interesados en aprender el idioma o traducir de forma amigable.

A lo largo de la historia han existido diferentes formas de hacer traducciones, hoy en día se aprovecha que existen las computadoras y dentro del mundo de la computación con diferentes paradigmas se tiene la esperanza de solucionar esta problemática, con Traducción Automática (TA), o MT (del inglés Machine Translation). Es un área de la lingüística computacional que investiga el uso de software para traducir texto o habla de un lenguaje natural a otro, tanto con o sin ayuda humana. La TA en sentido amplio abarca toda una variedad de sistemas que sólo comparten la utilización de la computadora como instrumento de traducción, Díaz Prieto y cols. (2015). Para la elaboración de traductores automáticos a partir de un modelo con redes neuronales se necesita un amplio vocabulario de oraciones o palabras apareadas en ambas lenguas. En el caso del Purépecha existen muy pocos documentos traducidos al Español que ayuden en esta tarea, por lo tanto es necesario utilizar un modelo adecuado para combatir este tipo de problemas.

Contar con un traductor automático entre estos dos idiomas (Purépecha-Español), permitirá que los hablantes de Purépecha y Español puedan traducir textos entre estos dos idiomas. Otra ventaja es poder generar con una amplia gama de libros y otro tipo de escritos, además podría facilitar a sus hablantes relacionarse con personas que no hablan esta lengua. En futuros proyectos se podría mejorar el sistema de traducción.

Haciendo posible el adentrarnos en los textos, cuentos y leyendas de esta lengua, así como de toda la tradición y cultura que estas personas tienen para contar, lo que se conoce como lingüística aplicada, de Lingüística Aplicada (s.f.).

En este trabajo se propone un modelo de ANN (Artificial Neural Network) llamado Transformers, que logra traducciones de Purépecha a Español, además propone la generación de sentencias entre ambas lenguas para conseguir un diccionario grande que puede ser utilizado en otras lenguas.

La ventajas de utilizar una arquitectura Transformers es que tienen una memoria de mucho más largo plazo a comparación de otras arquitecturas. Estas redes logran analizar secuencias muy extensas usando un mecanismo que se llama atención y adicionalmente logran procesar toda la secuencia en paralelo, y no en serie como anteriormente se hacía con las Redes Recurrentes.

El objetivo general de este trabajo es construir un modelo usando usando ANN Transformers para la traducción automática de Purépecha a Español.

Los objetivos específicos son:

- Estudiar las reglas sintácticas del Purépecha y el Español.
- Programar un generador de texto de Purépecha con su respectiva traducción al Español.
- Programar un modelo de ANN Transformer para el traductor automático.

Capítulo 2

Antecedentes

2.1. Estado del arte

En esta sección se presentará una breve contextualización sobre la historia y evolución de los sistemas de la traducción automática.

2.1.1. Traducción Automática

La traducción es un problema que ha surgido desde la antigüedad por la necesidad de comunicarse de las personas que hablan diferentes idiomas. Con el paso del tiempo la humanidad ha logrado desarrollar sistemas que ayuden a la automatización de esta tarea. En el siglo XVII se habló del uso de diccionarios mecánicos (basados en símbolos universales) para superar las barreras del lenguaje, en un movimiento que pedía la creación de un "lenguaje universal", basado en principios lógicos y símbolos, que permitiera a todos los seres humanos comunicarse, Mercedes (2002).

En los siglos siguientes, hubo varias propuestas para crear lenguajes internacionales, pero los primeros intentos serios de automatización, tardaron casi medio siglo. En 1933 se aplicaron dos patentes a los "intérpretes", una en Francia y otra en Rusia. Por un lado, el franco-armenio Georges Artesoni diseñó un dispositivo que almacenaba cintas de papel para poder encontrar el equivalente de cualquier palabra en otro idioma; Al mismo tiempo, el estudiante ruso Peter Smirnov Troyansky, de los dos, el más importante. Ya que los principios de la máquina que desarrolló siguen siendo válidos: las tres etapas que define en el proceso de traducción ahora son equivalentes, con las etapas de análisis, transmisión y generación, Mercedes (2002).

2.1.2. Clasificación de los sistemas

En este capítulo se muestra un recorrido general a través de las diferentes tipologías y agrupaciones de los sistemas de TA.

Según el diseño o la estrategia operativa general, primero tenemos sistemas de traducción directa donde la traducción se realiza reemplazando palabras en el idioma de origen con palabras en el idioma de destino. Son sistemas válidos cuando nos encontramos ante un texto con un vocabulario y estilo bien definido y limitado. Estos sistemas utilizan como fuente principal un diccionario en el idioma de origen y las palabras correspondientes traducidas al idioma de destino. La calidad del producto está directamente relacionada con la calidad de la información del diccionario. Un factor limitante es la complejidad del procesamiento del lenguaje, Mercedes (2002).

Los modelos desarrollados para la traducción automática se dividen en tres áreas: traducción basada en reglas (RBMT), modelado estadístico (traducción automática estadística, SMT) y traducción basada en redes neuronales (traducción automática neuronal, NMT), Escartín (2018).

2.1.3. Traducción Automática basada en reglas (RBMT)

Los RBMT traducen el texto pasando por tres etapas. En primer lugar, se realiza un análisis de la lengua fuente para obtener una representación intermedia con información gramatical y/o semántica. En segundo lugar, se produce la transición del idioma de origen al idioma de destino. En este punto, en la mayoría de los sistemas de este tipo, se pueden distinguir dos procesos diferentes, Mercedes (2002):

- a) La transferencia léxica o traducción de los términos que componen la oración de entrada, generalmente utilizando un diccionario bilingüe.
- b) Transformación de sintaxis o aplicación de una serie de reglas de transformación a la sintaxis resultante de la fase de análisis, para obtener la sintaxis equivalente en el idioma de destino.

Y por último, la denominada etapa de generación, que es el momento en que se reconstruye el cuerpo de la oración en la lengua de destino, a partir de la estructura adquirida durante la transferencia, véase la figura 2.1 y 2.2, Mercedes (2002).

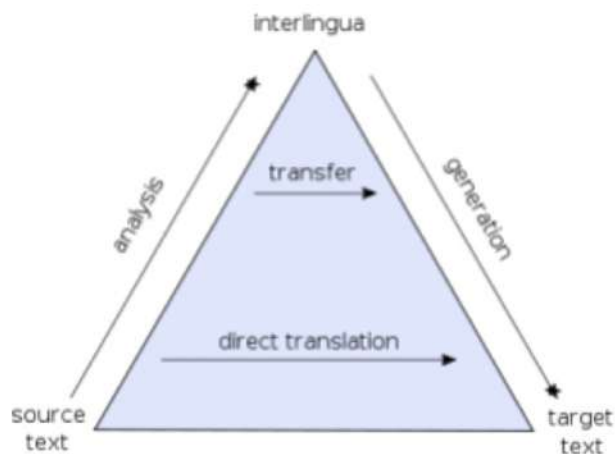


Figura 2.1: Etapas de la traducción automática basada en reglas: análisis, transición y generaciónLa traducción automática basada en reglasS (RBMT) (s.f.)

2.1.4. Traducción Automática Estadística (Statistic Machine Traslation)

La SMT es un modelo de traducción automática que se basa en modelos estadísticos aprendidos de un conjunto paralelo y algoritmos de decodificación para traducir automáticamente de un lenguaje natural a otro. Aquí es donde los modelos de traducción entran en primer lugar con los modelos de lenguaje. La probabilidad de que una secuencia de palabras o frases S ocurra se puede denotar como $Pr(S)$. El modelo de traducción estadística incluye estas probabilidades $Pr(T|S)$ donde ocurre la probabilidad condicional de la oración de destino T en el texto de destino que traduce el texto que contiene la oración de origen S , para representar este tipo de traductores automáticos se ha reformulado utilizando la regla de Bayes, Huarcaya Taquiri (2020).

Por otro lado, este tipo de traducción requiere grandes recursos computacionales, carece de base lingüística porque se basa en la probabilidad y en idiomas de bajos recursos existen problemas morfológicos entre los idiomas traducidos. Huarcaya Taquiri (2020).

2.1.5. Traducción Automática Neuronal (NMT)

La inteligencia artificial (IA) intenta simular los procesos que realiza la inteligencia humana a través de un sistema informático (razón, aprendizaje, resolución de problemas, reconocimiento de patrones, comprensión del lenguaje, etc.). Su producto se llama "Tecnología cognitiva". La NMT es un producto de inteligencia artificial que combina

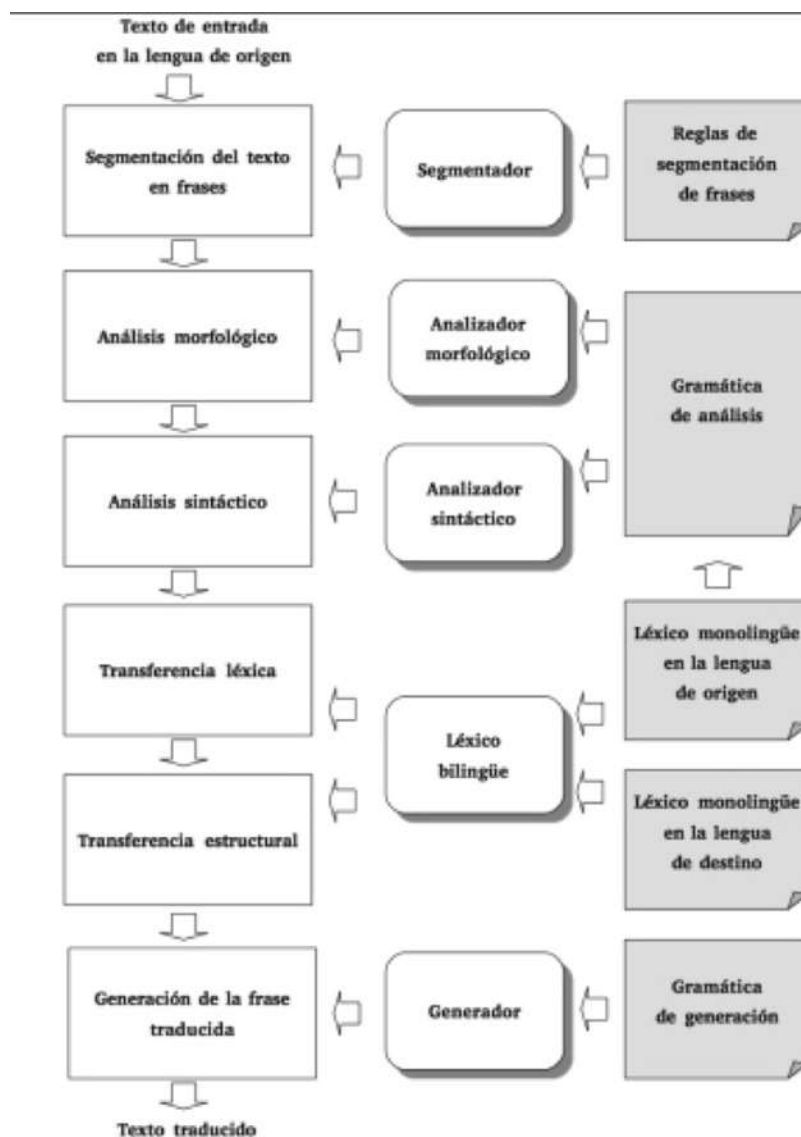


Figura 2.2: Etapas detalladas de *La traducción automática basada en reglas* (RBMT) (s.f.)

tecnologías cognitivas. Pues su traducción automática se basa en el aprendizaje profundo de la que se está hablando mucho hoy en día en el ámbito académico y profesional desde que entró en vigor en 2016 en textos de contenido general y en 2018 en textos de contenido especializado.

Con el aprendizaje profundo, y el surgimiento de las incrustaciones (word embeddings es, una representación matricial estática del texto, en la que cada palabra del vocabulario se codifica en un vector), las técnicas de procesamiento del lenguaje natural (NLP) cada vez se basaban más en el texto como fuente de entrada de un modelo. Así, al menos hasta finales de 2017, los modelos basados en redes neuronales recurrentes se habían convertido en la norma, ya que pueden tener en cuenta no solo el significado general de cada palabra, sino también la posición de éstas en la frase, *Redes Transformer*

(... o el fin de las Redes Recurrentes) (s.f.).

En 2017 se introdujo la arquitectura Transformer, modelo cuya principal innovación fue la sustitución de capas recurrentes, por capas de atención, *Redes Transformer (... o el fin de las Redes Recurrentes)* (s.f.).

Capítulo 3

Marco teórico

3.1. Conceptos Básicos Matemáticos

En este capítulo se presentarán conceptos necesarios para entender la propuesta presentada en el siguiente capítulo.

3.1.1. Vectores, matrices y tensores

- **Vectores.** Un vector es un arreglo ordenado de números. Podemos seleccionar cada número de su índice en este orden. Usualmente llamamos a los vectores en negrita y minúscula, como \mathbf{x} . Los elementos vectoriales se identifican escribiendo sus nombres en cursiva, con la adición de un subíndice. El primer elemento en \mathbf{x} es x_1 , el segundo elemento es x_2 y así sucesivamente. También necesitamos especificar el tipo de números almacenados en el vector. Si cada elemento está en \mathbb{R} y el vector contiene n elementos, entonces el vector en el conjunto se forma tomando el producto cartesiano de \mathbb{R} n veces, el símbolo es \mathbb{R}^n . Cuando es necesario especificar explícitamente los elementos del vector, los escribimos como columnas encerradas entre corchetes, Goodfellow, Bengio, y Courville (2016):

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

Podemos pensar en los vectores como puntos de identificación en el espacio, con cada elemento dando la coordenada a lo largo de un eje diferente, Goodfellow y cols. (2016).

- **Matrices.** Una matriz es un arreglo bidimensional de números, por lo que cada elemento se identifica por dos índices en lugar de uno solo. Si una matriz A de

valor real tiene una altura de m y un ancho de n , entonces decimos que $A \in \mathbb{R}^{n \times m}$. Cuando necesitamos identificar los elementos de una matriz, los escribimos entre corchetes, Goodfellow y cols. (2016):

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

- **Tensores.** Una matriz de números dispuestos en una cuadrícula regular con un número variable de ejes se conoce como tensor, Goodfellow y cols. (2016).

Una operación importante en matrices es la transpuesta. Una **matriz traspuesta** es la matriz resultante al reordenar la matriz original mediante el cambio de filas por columnas y las columnas por filas en una nueva matriz. $(A^T)_{i,j} = A_{j,i}$.

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Los vectores se pueden considerar como matrices que contienen solo una columna. Podemos definir un vector escribiendo sus elementos en el texto en línea como una matriz de filas, luego usando el operador de transposición para convertirlo en un vector de columna estándar, por ejemplo $x = [x_1, x_2, x_3]^T$.

Se puede pensar en un escalar como una matriz con una sola entrada. Así que un escalar es su propia transpuesta: $a = a^T$.

Podemos sumar matrices entre sí, siempre que tengan la misma forma, sumando sus elementos correspondientes: $C = A + B$ donde $C_{i,j} = A_{i,j} + B_{i,j}$.

También podemos sumar un escalar a una matriz o multiplicar una matriz por un escalar, simplemente realizando esa operación en cada elemento de una matriz: $D = aB + c$ donde $D_{i,j} = aB_{i,j} + c$, Goodfellow y cols. (2016).

3.1.2. Multiplicación de Matrices

Una de las operaciones más importantes que involucran matrices es la multiplicación de dos matrices. El producto matricial de las matrices A y B es una tercera matriz C . Para que se defina este producto, A debe tener el mismo número de columnas que B tiene filas. Si A tiene forma $m \times n$ y B tiene forma $n \times p$, entonces C tiene forma $m \times p$. Podemos escribir el producto de matrices simplemente juntando dos o más matrices, por ejemplo, $C = AB$.

La operación del producto está definida por

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

El **producto punto** o **producto escalar** entre dos vectores \mathbf{x} y \mathbf{y} de la misma dimensionalidad es el producto de matriz $\mathbf{x}^T \mathbf{y}$. Podemos pensar en el producto matricial $\mathbf{C} = \mathbf{AB}$ como si calculara $C_{i,j}$ como el producto escalar entre la fila i de \mathbf{A} y la columna j de \mathbf{B} , Goodfellow y cols. (2016).

3.1.3. Incrementos, derivada y derivadas parciales

Considere una función suave y continua $f(x) = y$, que asigna un número real x a un nuevo número real y . Dado que la función es continua, un pequeño cambio en x solo puede producir un pequeño cambio en y , que es la intuición detrás de la continuidad. Suponga que incrementa x por un pequeño factor de ∇x : esto da como resultado un pequeño cambio de ∇y a y :

$$f(x + \nabla x) = y + \nabla y$$

Además, dado que la función es suave (su curva no tiene esquinas agudas), cuando ∇x es lo suficientemente pequeño, alrededor de un punto p dado, es posible aproximar f como una propiedad de una función lineal de pendiente a , entonces ∇y se convierte en $a * \nabla x$ entonces:

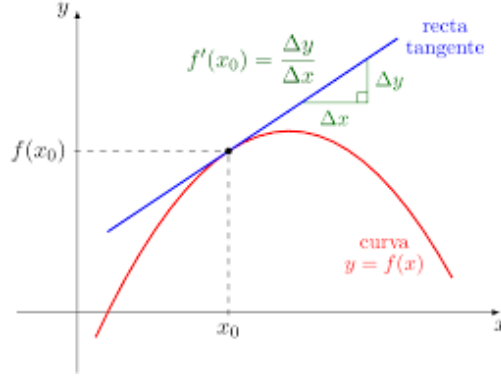
$$f(x + \nabla x) = y + a * \nabla x$$

Esta aproximación lineal es válida solo cuando x está lo suficientemente cerca de p .

La pendiente a se llama la derivada de f en p . Si a es negativo, significa que un pequeño cambio de x alrededor de p resultará en una disminución de $f(x)$ (como se muestra en la figura 3.1); y si a es positivo, un pequeño cambio en x resultará en un aumento de $f(x)$. Además, el valor absoluto de a (la magnitud de la derivada) te dice qué tan rápido ocurrirá este aumento o disminución.

Si está tratando de actualizar x por un factor ∇x para minimizar $f(x)$, y conocer la derivada de f , entonces su trabajo está hecho: la derivada completamente describe cómo evoluciona $f(x)$ a medida que cambia x . Si desea reducir el valor de $f(x)$, solo necesita mover x un poco en la dirección opuesta a la derivada, Chollet (2017).

- **Incrementos.** El incremento ∇x de una variable x es el aumento o disminución que experimenta, desde un valor $x = x_0$ a otro $x = x_1$ de su campo de variación. Así, pues, $\nabla x = x_1 - x_0$ o bien $x_1 = x_0 + \nabla x$.

Figura 3.1: Derivada de f en x_0 , ?.

Si se da un incremento ∇x a la variable x (es decir si x pasa de $x = x_0$ a $x = x_0 + \nabla x$), la función $y = f(x)$ se verá incrementada en $\nabla y = f(x_0 + \nabla x) - f(x_0)$ a partir del valor $y = f(x_0)$. El cociente :

$$\frac{\nabla y}{\nabla x} = \frac{\text{incremento de } y}{\text{incremento de } x}$$

recibe el nombre de cociente medio de incrementos de la función en el intervalo comprendido entre $x = x_0$ hasta $x = x_0 + \nabla x$, Ayres, Mendelson, y Abellanas (1991).

- **La derivada** de una función $y = f(x)$ con respecto a x en un punto $x = x_0$ se define por el límite:

$$\lim_{\nabla x \rightarrow 0} \frac{\nabla y}{\nabla x} = \lim_{\nabla x \rightarrow 0} \frac{f(x_0 + \nabla x) - f(x_0)}{\nabla x}$$

siempre que exista. Este límite se denomina también cociente instantáneo de incrementos (o simplemente cociente de incrementos) de y con respecto de x en el punto $x = x_0$, Ayres y cols. (1991).

- **Derivadas parciales.** Sean $U \in \mathbb{R}^n$ un conjunto abierto y $f : U \in \mathbb{R}^n \rightarrow \mathbb{R}$ una función con valores reales. Entonces $\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n}$, las **derivadas parciales** de f respecto a la primera, segunda, \dots , n -ésima variable son las funciones con valores reales, de n variables, las cuales, en el punto $(x_1, \dots, x_n) = x$, están definidas por:

$$\frac{\partial f}{\partial x_j}(x_1, \dots, x_n) = \lim_{h \rightarrow 0} \frac{f(x_1, x_2, \dots, x_j + h, \dots, x_n) - f(x_1, \dots, x_n)}{h}$$

si existen los límites, donde $1 \leq j \leq n$ y e_j es el j -ésimo vector de la base usual, definido por $e_j = (0, \dots, 1, \dots, 0)$, con el 1 en el j -ésimo lugar.

En otras palabras, $\frac{\partial f}{\partial x_j}$ es simplemente la derivada de f respecto a la variable x_j , manteniendo las otras variables fijas. Si $f : \mathbb{R}^3 \rightarrow \mathbb{R}$, con frecuencia usaremos la

notación $\frac{\partial f}{\partial x}$, $\frac{\partial f}{\partial y}$, y $\frac{\partial f}{\partial z}$ en lugar de $\frac{\partial f}{\partial x_1}$, $\frac{\partial f}{\partial x_2}$ y $\frac{\partial f}{\partial x_3}$. Si $f : U \subseteq \mathbb{R} \rightarrow \mathbb{R}^m$, entonces podemos escribir:

$$f(x_1, \dots, x_n) = (f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$$

de modo que podemos hablar de las derivadas parciales de cada componente; por ejemplo, $\frac{\partial f_m}{\partial x_n}$ es la derivada parcial de la m -ésima componente con respecto a x_n la n -ésima variable, Marsden, Tromba, y Mateos (1991).

3.1.4. Gradiente

Un gradiente es la derivada de una operación tensorial. Esto es la generalización del concepto de derivadas a funciones de entradas multidimensionales: es decir, a funciones que toman tensores como entradas Chollet (2017).

DEFINICIÓN Si $f : U \subseteq \mathbb{R}^3 \rightarrow \mathbb{R}$ es diferenciable, el **gradiente** de f en (z, y, z) es el vector en el espacio \mathbb{R}^3 dado por:

$$\text{grad} f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z} \right)$$

Este vector también se denota por ∇f o $\nabla f(x, y, z)$. Así, ∇f es simplemente la matriz de las derivadas Df , escrita como vector, Marsden y cols. (1991).

3.1.5. Regla de la cadena

La regla de la cadena del cálculo se utiliza para computar las derivadas de funciones formadas al componer otras funciones cuyas derivadas se conocen.

Sea x un número real, y sean f y g funciones que mapean de un número real a un número real. Supongamos que $y = g(x)$ y $z = f(g(x)) = f(y)$. Luego la regla de la cadena establece que, Chollet (2017):

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

3.2. Tokenización

La tokenización es el proceso de representar texto en unidades más pequeñas llamadas tokens que deben convertirse en una serie de vectores numéricos, *La evolución de la codificación de pares de bytes de tokenización en nlp* (s.f.).

Existen diferentes técnicas para convertir el texto en tokens:

- **Tokenización basada en palabras**

Un método simple y directo son los tokens basados en palabras en el que se divide el texto por espacios.

- **Tokenización basada en caracteres**

A cada carácter, se le asigna un símbolo para generar una secuencia de vectores.

- **Tokenización de subpalabras**

Para preservar las propiedades semánticas del token, es decir, información de cada token. Tokenizar sin necesidad de un vocabulario muy extenso con un conjunto limitado de palabras, se podría considerar la segmentación de palabras en función de un conjunto de prefijos y sufijos, o algún otro tipo de criterio de separación de palabras, *La evolución de la codificación de pares de bytes de tokenización en nlp* (s.f.).

El modelo solo aprende algunas subpalabras y luego las junta para crear otras. Esto resuelve el problema de los requisitos de memoria y el esfuerzo por crear un muy extenso vocabulario, *La evolución de la codificación de pares de bytes de tokenización en nlp* (s.f.).

3.2.1. Codificación de pares de bytes (BPE)

BPE fue originalmente un algoritmo de compresión de datos utilizado para encontrar la mejor manera de representar datos mediante la selección de pares de bytes comunes. Actualmente se usa en NLP para encontrar la mejor manera de presentar el texto con la menor cantidad de tokens.

Así es como funciona:

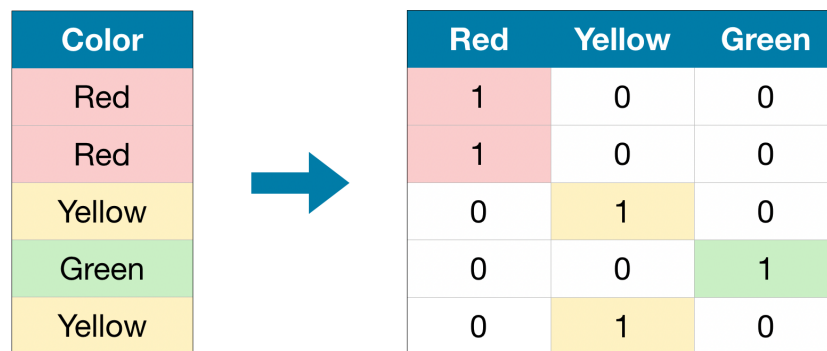
- Se agrega un identificador (`</w>`) al final de cada palabra para identificar el final de la palabra y luego contar la cantidad de veces que las palabras aparecen en el texto.
- Se divide la palabra en caracteres y luego se calcula cuántas veces aparece cada carácter.

- A partir del token de carácter, para un número predeterminado de iteraciones, se calcula la frecuencia de pares de bytes sucesivos y el conjunto de asociación de bytes más común.
- Se sigue iterando hasta que haya alcanzado el límite de iteración establecido o si ha alcanzado el límite de tokens.

3.2.2. One hot encoding

El one hot encoding es la forma más común y básica de convertir una palabra o token en un vector.

Consiste en asociar un índice entero único para cada palabra y luego convertir el índice entero i en un vector binario de tamaño N (tamaño de vocabulario); Los vectores son todos ceros excepto la i -ésima entrada, que es 1, Chollet (2017). Como se muestra en la figura 3.2.



Color	Red	Yellow	Green
Red	1	0	0
Red	1	0	0
Yellow	0	1	0
Green	0	0	1
Yellow	0	1	0

Figura 3.2: One hot encoding, *One hot encoding in TensorFlow (tf.one_hot)* (s.f.)

3.2.3. Word embedding

Otra forma de asociar un vector con una palabra o token es usar vectores de palabras densos, también conocidos como incrustaciones de palabras (word embeddings). Mientras que los vectores obtenidos con one hot encoding son binarios, dispersos (compuestos principalmente de ceros) y muy grandes (en la misma dimensión que el número de palabras o tokens en el vocabulario), la incrustación de palabras son vectores de punto flotante más pequeños, Chollet (2017).

A diferencia de los vectores de palabras obtenidos a través de one hot encoding, la incrustación de palabras se aprende a partir de los datos. Es común ver incrustaciones de palabras en 256, 512 o 1024 dimensiones cuando se trata de un vocabulario muy extenso, Chollet (2017).



Figura 3.3: Las representaciones de palabras obtenidos de one hot encoding son escasos, de alta dimensión y codificados de forma rígida, Chollet (2017)

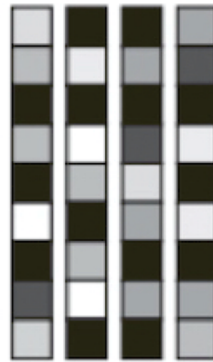


Figura 3.4: Las incrustaciones de palabras son densas, de dimensión relativamente baja y se aprenden con base en los datos, Chollet (2017)

Las palabras de one hot encoding generalmente producen vectores de tamaño 20,000 dimensiones o más dependiendo del tamaño del vocabulario. Por lo tanto, las incrustaciones de palabras (word embeddings) empaquetan más información en muchas menos dimensiones, Chollet (2017).

Los vectores de palabras se utilizan para el análisis semántico, para extraer el significado del texto y para permitir la comprensión del lenguaje natural. Para que un modelo de lenguaje prediga el significado de un texto, debe ser consciente de las similitudes contextuales de las palabras, Chollet (2017).

Los vectores generados por Word Embedding mantienen similitudes, por lo que las palabras que frecuentemente aparecen juntas en el texto también estarán muy cerca entre sí en el espacio vectorial, Chollet (2017). Así que, las incrustaciones de palabras son un medio para construir una representación vectorial de baja dimensión de un conjunto, que ayuda a mantener la similitud contextual de las palabras, Chollet (2017).

3.3. Redes Neuronales Artificiales

Una red neuronal artificial (ANN, por sus siglas en inglés Artificial Neural Network) es un conjunto de conceptos computacionales inspirado en la arquitectura del sistema nervioso humano. La arquitectura de una red neuronal se forma conectando varios procesos elementales, y es un sistema adaptativo cuyo algoritmo ajusta sus pesos (parámetros libres) para cumplir con los requisitos de la red neuronal. Requisitos de rendimiento del problema basados en muestras representativas, Salas (2004).

Un ANN es un sistema de computación distribuida caracterizado por:

- Un conjunto de unidades elementales, cada una con baja capacidad de procesamiento.
- Una estructura densa conectada internamente usando enlaces ponderados.
- Parámetros libres que deben ser ajustados para satisfacer los requerimientos de desempeño.
- Un alto grado de paralelismo.

Es importante recalcar que la propiedad más importante de una red neuronal artificial es su capacidad de aprender a partir de un conjunto de datos o patrones de entrenamiento, es decir, es capaz de encontrar un modelo que ajuste los datos. El proceso de aprendizaje también conocido como entrenamiento de la red puede ser supervisado o no supervisado, Salas (2004).

El aprendizaje supervisado consiste en entrenar una red a partir de una serie de datos o un modelo de entrenamiento formado por modelos de entrada y salida. El objetivo del algoritmo de aprendizaje es ajustar los pesos de la red w para que la salida producida por la ANN sea lo más cercana posible a la salida real de la entrada dada. Es decir, la red neuronal está tratando de encontrar un modelo del proceso desconocido que produjo el resultado. Este proceso de aprendizaje se llama supervisado porque se conoce el patrón de salida, que juega un papel que supervisa a la red, Salas (2004).

Por otro lado, en el aprendizaje no supervisado, a la ANN solo se le presentan un conjunto de datos, y el objetivo del algoritmo de aprendizaje es ajustar los pesos de la red para que la red encuentre la estructura o configuración actual presente en los datos, Salas (2004).

3.3.1. Neurona

La unidad elemental de la red neuronal artificial es llamada neurona, es un procesador que puede calcular limitadamente la suma ponderada de sus entradas y luego

aplicarle una función de activación para recibir la señal que se enviará a otra neurona. Estas neuronas artificiales se agrupan en niveles o capas, y existe un alto grado de conectividad entre ellas, esta conectividad es ponderada por los pesos. A través de un algoritmo de aprendizaje supervisado o no supervisado, las redes neuronales artificiales ajustan su estructura y parámetros de tal manera que minimiza una función de error que indican la adaptabilidad y generalización de datos, Salas (2004).

3.3.2. Función de activación

Como se mencionó, la neurona se activará si la entrada total excede el umbral. Para ello lo que se hace es aplicar una función de activación a la suma ponderada de sus entradas, que puede ser, por ejemplo, una función sigmoideal como la tangente hiperbólica o una función escalonada, (Izaurieta y Saavedra, 2000). Esto distorsiona el valor de salida, añadiéndole una distorsión no lineal para poder integrar eficientemente el cómputo de múltiples neuronas.

3.3.3. Función de pérdida

Una vez definida la arquitectura de la red, aún se tendría que elegir la función de pérdida o función objetivo, esta función determina una cantidad que se tendrá que reducir durante el entrenamiento. Es una medida del éxito de la tarea en cuestión.

Elegir la función de pérdida correcta para el problema es extremadamente importante: la red tomará cualquier opción posible para reducir la pérdida; Por lo tanto, si el objetivo no está completamente relacionado con el éxito de una tarea determinada, la red hará cosas que quizá no se esperaban, Chollet (2017).

Cuando se trata de problemas comunes como la clasificación, la regresión y la predicción de secuencias, existen pautas simples que puede seguir para elegir la pérdida correcta. Por ejemplo, utilizar la entropía cruzada binaria para un problema de clasificación de dos clases, la entropía cruzada categórica para un problema de clasificación de muchas clases, el error cuadrático medio para un problema de regresión, la clasificación temporal conexionista para un problema de aprendizaje de secuencias, etc, Chollet (2017).

3.3.4. Descenso de gradiente estocástico

Dada una función diferenciable, es teóricamente posible encontrar su mínimo analíticamente: sabemos que el mínimo de una función es el punto donde la derivada es

cero, así que solo necesitamos encontrar todos los puntos donde la derivada tiende a ser cero y verificar cuál de estos puntos tienen el menor valor en la función, Chollet (2017).

Cuando se aplica a redes neuronales, esto significa que el análisis encuentra un conjunto de valores de peso que produce la función de pérdida más pequeña posible. Esto se puede hacer resolviendo la ecuación $\text{gradiente}(f)(W) = 0$ para W . Esta es una ecuación polinomial con N variables, donde N es el número de coeficientes en la red. Aunque es posible resolver una ecuación de este tipo para $N = 2$ o $N = 3$, hacerlo es difícil para las redes neuronales reales, donde la cantidad de parámetros nunca es inferior a unos pocos miles y, a menudo, puede ser de varias decenas de millones, Chollet (2017).

A continuación se muestran los pasos que el algoritmo del descenso de gradiente estocástico por mini-bloques lleva a cabo .

- 1 Un mini-lote de entrada con N muestras aleatorias se introduce a partir de los datos de entrenamiento, que se clasifican previamente (es decir que se conoce la salida real), Goodfellow y cols. (2016).
- 2 Después de los cálculos pertinentes en cada capa de la red, obtenemos como resultado las predicciones a su salida. A este paso se le conoce como forward propagation (de las entradas), Goodfellow y cols. (2016).
- 3 Se evalúa la función de pérdida para dicho mini-lote. El valor de esta función de pérdida es lo que se trata de minimizar en todo momento mediante el algoritmo, y hacia ello se orientan los siguientes pasos, Goodfellow y cols. (2016).
- 4 El gradiente se calcula como la derivada multivariable de la función de pérdida para todos los parámetros de la red. Gráficamente, sería la pendiente de la tangente a la función de pérdida en el punto en el que nos encontramos (evaluando los pesos actuales). Para movernos en la dirección opuesta reduciendo así la función de pérdida, Goodfellow y cols. (2016).

La figura 3.5 ilustra lo que sucede en 1D, cuando la red tiene solo un parámetro y solo un entrenamiento muestra, Chollet (2017).

Como se puede observar, intuitivamente es importante elegir un valor razonable para el factor de paso.

Si es demasiado pequeño, el descenso por la curva necesitará muchas iteraciones y podría atascarse en un mínimo local. Si el paso es demasiado grande, sus actualizaciones pueden terminar llevándolo a ubicaciones completamente aleatorias en la curva, Chollet (2017).

Aunque la figura 3.5 ilustra el descenso de gradiente en un espacio de parámetros 1D, en la práctica usará el descenso de gradiente en espacios altamente dimensionales:

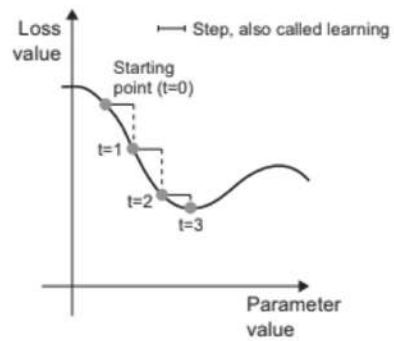


Figura 3.5: Curva de pérdida 1D, Chollet (2017)

cada coeficiente de peso en una red neuronal hay una dimensión libre en el espacio, y puede haber decenas de miles o incluso millones de ellos. Para ayudar a construir la intuición sobre las superficies de pérdida, también puede visualizar el descenso del gradiente a lo largo de una superficie de pérdida 2D, Chollet (2017), como se muestra en la figura 3.6.

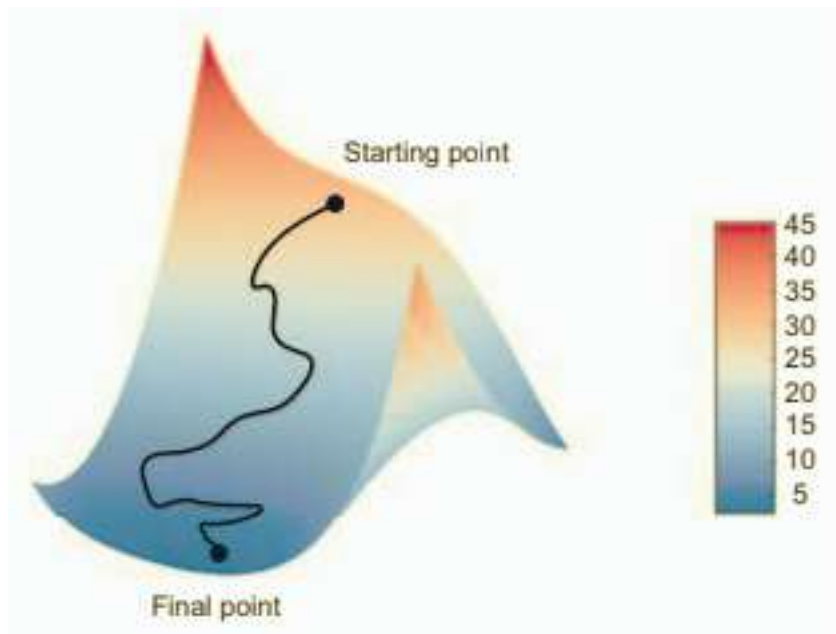


Figura 3.6: Superficie de pérdida 2D, Chollet (2017)

3.3.5. Backpropagation

Cuando usamos la red neuronal de feed-forward para aceptar una entrada x y producir una salida y , la información se transmite a través de la red. La entrada x proporciona

la información inicial, que luego se propaga a las neuronas ocultas en cada capa, y finalmente da como resultado y . Esto se llama propagación directa (forward propagation). Durante el entrenamiento, la propagación directa puede continuar hasta que resulte en una pérdida escalar $J(\theta)$. El algoritmo de retropropagación (backpropagation) permite que la información de pérdida fluya hacia atrás a través de la red para calcular el gradiente. Calcular una expresión analítica para un gradiente es simple, pero la evaluación numérica de esta expresión puede ser computacionalmente costosa, Goodfellow y cols. (2016).

Este algoritmo hace esto utilizando un procedimiento simple y económico. El término retropropagación a menudo se malinterpreta como un algoritmo de aprendizaje completo para redes neuronales multicapa. Realmente solo se refiere al método de cálculo del gradiente, mientras que otro algoritmo, como el descenso de gradiente estocástico, se usa para hacer el aprendizaje con este gradiente, Goodfellow y cols. (2016).

En particular, describiremos cómo calcular el gradiente $\nabla_x f(x, y)$ para una función aleatoria f , donde x es un conjunto de variables cuya derivada se espera que se requiera mientras que y es un conjunto adicional de variables que se incluyen en la función pero no están obligados a obtener sus derivadas. En aprendizaje computacional, el gradiente más solicitado es el gradiente de la función de pérdida que incluye los parámetros, $\nabla_\theta J(\theta)$. Muchos procesos de aprendizaje automático implican el cálculo de otras derivadas, tanto como parte del proceso de aprendizaje como para analizar el modelo aprendido. El algoritmo de retropropagación también se puede aplicar a estas tareas y no se limita a calcular el gradiente de la función de pérdida con respecto a los parámetros. La idea de calcular derivadas pasando información a través de una red es muy general y puede usarse para calcular tales valores como Jacobiano para una función f con múltiples salidas. Limitamos nuestra descripción aquí a los casos de uso más frecuente, donde f tiene una salida, Goodfellow y cols. (2016).

Backpropagation es un algoritmo que calcula la regla de la cadena, con un orden específico de operaciones que es altamente eficiente, Goodfellow y cols. (2016).

La regla de la cadena establece que:

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Podemos generalizar esto más allá del caso escalar. Supongamos que $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$, g mapea de \mathbb{R}^m a \mathbb{R}^n , y f mapea de \mathbb{R}^n a \mathbb{R} . Si $y = g(x)$ y $z = f(y)$, entonces:

$$\frac{\partial z}{\partial x_i} = \sum \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

En notación vectorial, esto se puede escribir de manera equivalente como

$$\nabla_x z = \left(\frac{\partial y}{\partial x}\right)^T \nabla_y z$$

donde $\frac{\partial y}{\partial x}$ es la matriz jacobiana $n \times m$ de g .

De esto vemos que el gradiente de una variable x se puede obtener multiplicando una matriz jacobiana por un gradiente $\nabla_y z$. El algoritmo backpropagation consiste de realizar tal producto de gradiente jacobiano para cada operación en el gráfico, Goodfellow y cols. (2016).

3.4. Procesamiento del lenguaje natural

A lo largo de la historia humana, gran parte del conocimiento se ha comunicado, almacenado y gestionado en forma de lenguaje natural. La era actual no es una excepción: el conocimiento sigue existiendo y se crea en forma de documentos, libros y artículos, aunque se almacene en forma electrónica, es decir, digitalmente. El gran avance es que las computadoras han podido brindar una gran ayuda para procesar el conocimiento de esta manera, para el Procesamiento del Lenguaje Natural (s.f.).

Sin embargo, el conocimiento para nosotros, los humanos, no es conocimiento para las computadoras. Los archivos son una cadena de caracteres y nada más. Una computadora puede copiar, respaldar, mover y eliminar dicho archivo. Pero no puede buscar respuestas a preguntas, inferencias lógicas, generalizaciones y resúmenes. A menos que haya sido entrenado para esta tarea, para el Procesamiento del Lenguaje Natural (s.f.).

Para combatir esta situación, se dedica mucho esfuerzo al desarrollo de la ciencia, cuya tarea es permitir que las computadoras comprendan el texto. Dependiendo del enfoque práctico y teórico, el nivel de comprensión esperado y otros aspectos, esta ciencia tiene diferentes nombres: procesamiento de lenguaje natural, procesamiento de texto, tecnologías del lenguaje y lingüística computacional. Lo que se desea lograr es procesar el texto por su sentido, no solo como un archivo binario, para el Procesamiento del Lenguaje Natural (s.f.).

La lingüística computacional, en su etapa actual de desarrollo, es principalmente una rama de la tecnología de aprendizaje automático, parte de la inteligencia artificial y la estadística. El aprendizaje automático se dedica al descubrimiento totalmente automatizado de reglas y relaciones en los datos. Por lo general, se aplica a datos numéricos, pero la lingüística computacional puede ser considerada como el aprendizaje automático sobre un tipo de datos especial, los textos en un lenguaje humano, Gelbukh (2010).

3.5. Redes Transformer

En este trabajo proponemos Transformer, una arquitectura basada en un mecanismo de atención para crear una dependencia global entre entradas y salidas. El transformer permite una paralelización significativamente mayor y se puede lograr mejor calidad de traducción, *The Illustrated Transformer* (s.f.).

La atención es un concepto que ha ayudado a mejorar el rendimiento de las aplicaciones de traducción automática neuronal. Transformer es un modelo que utiliza la atención para aumentar la velocidad con la que se pueden entrenar estos modelos. La mayor ventaja proviene de la forma en que el transformador se presta a la paralelización, *The Illustrated Transformer* (s.f.).

3.5.1. Componentes principales del modelo

Podemos visualizar al modelo como una caja. La cual toma una oración de entrada en un idioma y genera su traducción en el otro idioma, *The Illustrated Transformer* (s.f.), véase figura 3.7.



Figura 3.7: Toma una oración de entrada en un idioma y genera su traducción en el otro idioma, *The Illustrated Transformer* (s.f.)

Al abrir esta “caja” vemos dos bloques que la componen: un bloque codificador y un bloque decodificador, como se muestra en la figura 3.8, *The Illustrated Transformer* (s.f.).

El componente de codificación es una pila de codificadores. El componente de decodificación es una pila de decodificadores del mismo número, *The Illustrated Transformer* (s.f.), véase figura 3.9.

Los codificadores tienen la misma estructura aunque no comparten pesos y cada uno se divide en dos subcapas, *The Illustrated Transformer* (s.f.), figura 3.10:

Las entradas del codificador fluyen a través de una capa de autoatención, que es

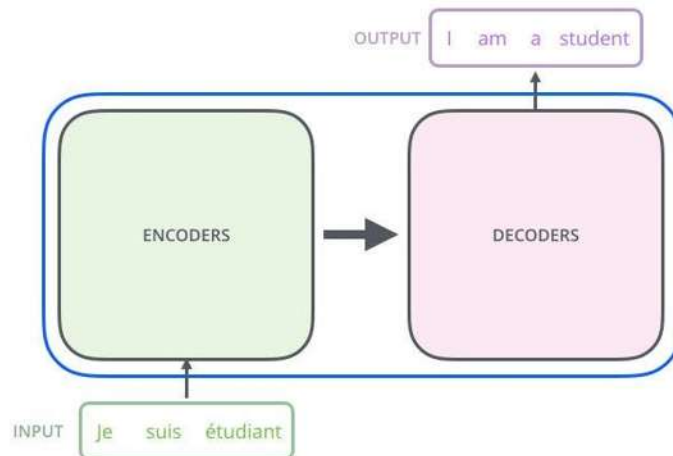


Figura 3.8: Bloques: codificador y decodificador, *The Illustrated Transformer* (s.f.)

una capa que ayuda al codificador a tomar en cuenta las otras palabras en la oración de entrada en relación con la palabra específica que se está codificando, *The Illustrated Transformer* (s.f.).

Las salidas de la capa de autoatención alimentan a una red neuronal feed forward, *The Illustrated Transformer* (s.f.).

El decodificador tiene ambas capas, pero entre ellas hay una capa de atención que ayuda al decodificador a enfocarse en las partes relevantes de la oración de entrada, *The Illustrated Transformer* (s.f.), figura 3.11.

3.5.2. De palabras a vectores

Los vectores que se generan a partir de la entrada fluyen entre los componentes del Transformer para convertir la entrada de un modelo entrenado en una salida, *The Illustrated Transformer* (s.f.).

Como es el caso de las aplicaciones de NLP, se convierte cada palabra o token de entrada en un vector usando word embedding (incrustaciones), *The Illustrated Transformer* (s.f.), figura 3.12.

La incrustación solo ocurre en el primer codificador aunque todos los codificadores reciben una lista de vectores. El primer codificador recibe una lista con cada vector de tamaño 512, pero los otros codificadores, reciben la salida del codificador que está directamente debajo. El tamaño de esta lista es un hiperparámetro que podemos establecer, *The Illustrated Transformer* (s.f.).

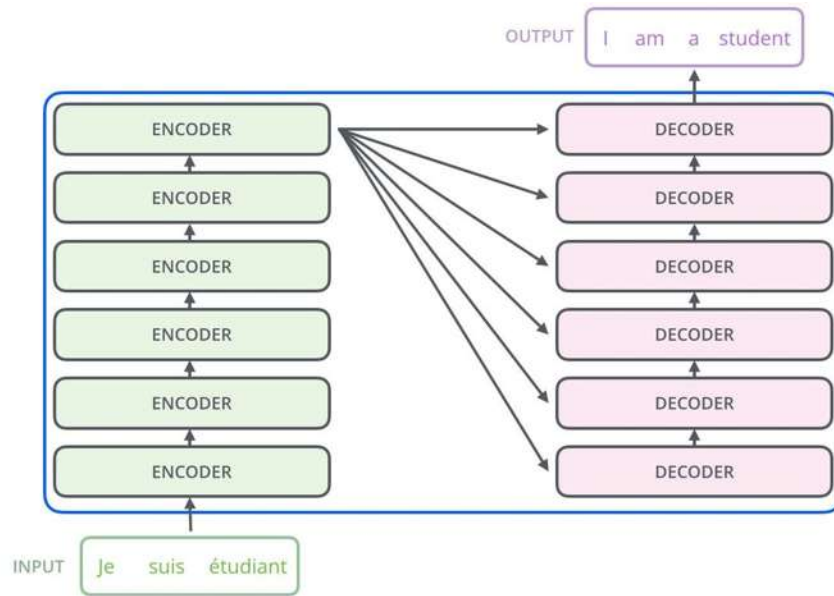


Figura 3.9: Estructura del codificador y decodificador, *The Illustrated Transformer* (s.f.)

Después de incrustar las palabras o tokens de la secuencia de entrada, cada vector fluye a través de cada una de las dos capas del codificador, *The Illustrated Transformer* (s.f.), figura 3.13.

La propiedad clave del Transformer, es que la palabra o token en cada posición fluye a través de su propia ruta en el codificador. Hay dependencias entre estos caminos en la capa de autoatención. Sin embargo, la capa de la red neuronal feed forward no tiene esas dependencias, así que, las diversas rutas se pueden ejecutar en paralelo mientras fluyen a través de esta capa, *The Illustrated Transformer* (s.f.).

3.5.3. Codificador

El codificador recibe como entrada una lista de vectores. Procesa esta lista pasando estos vectores a una capa de “autoatención”, después a una red neuronal (feed forward) y finalmente envía la salida al siguiente codificador, como se muestra en la figura 3.14, *The Illustrated Transformer* (s.f.).

La palabra convertida en vector en cada posición pasa por un proceso de autoatención. Luego, cada uno pasa a través de una red neuronal, cada vector fluye a través de ella por separado, *The Illustrated Transformer* (s.f.).

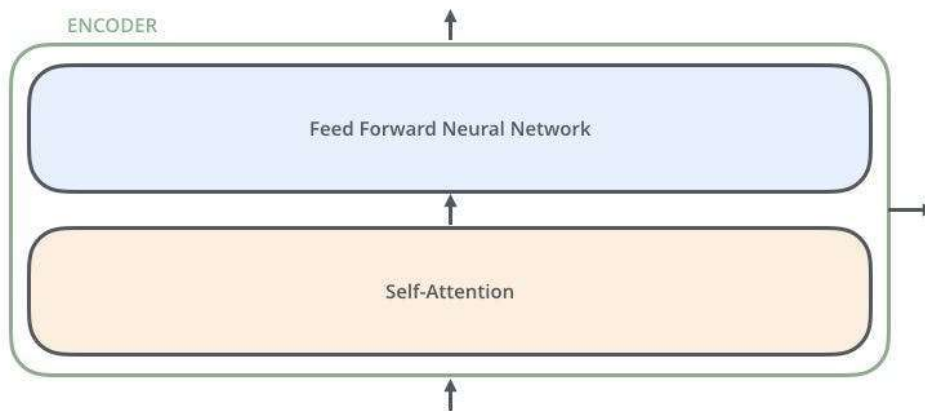


Figura 3.10: Subcapas de los codificadores, *The Illustrated Transformer* (s.f.)

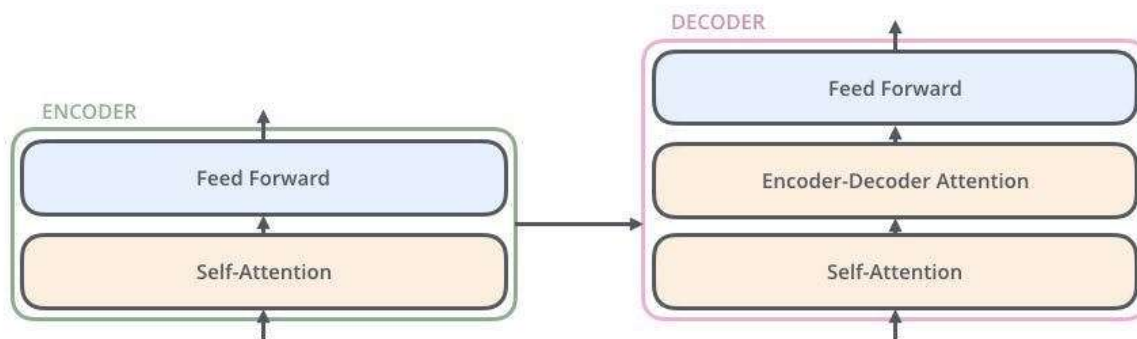


Figura 3.11: Estructuras de un codificador y un decodificador, *The Illustrated Transformer* (s.f.)

3.5.4. Autoatención

Para explicar el concepto de autoatención, veamos el siguiente ejemplo. La siguiente oración es una oración de entrada que se quiere traducir, *The Illustrated Transformer* (s.f.):

"The animal didn't cross the street because it was too tired"

Fácilmente podemos deducir que la palabra "it" en la oración se refiere a "animal" y no a "street", deducción que no resultaría tan fácil para un algoritmo, *The Illustrated Transformer* (s.f.).

Cuando el modelo está procesando la palabra "it", la autoatención le permite asociar "it" con "animal", *The Illustrated Transformer* (s.f.).



Figura 3.12: Cada palabra está incrustada en un vector de tamaño 512. Representaremos esos vectores con estos simples cuadros, *The Illustrated Transformer* (s.f.).

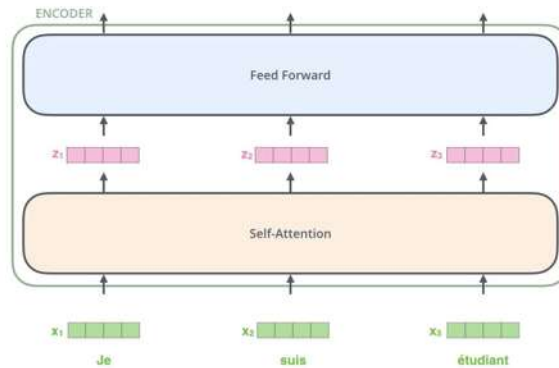


Figura 3.13: Las palabras incrustadas pasan por las dos capas del codificador, *The Illustrated Transformer* (s.f.).

A medida que el modelo procesa cada palabra, la autoatención busca relaciones con las demás palabras en la secuencia de entrada, lo cual logra una mejor codificación de la palabra, *The Illustrated Transformer* (s.f.).

La autoatención es el método que usa el Transformer para almacenar información de otras palabras relevantes en relación con la que se procesa en ese momento, *The Illustrated Transformer* (s.f.).

3.5.5. Autoatención en detalle

Para ejemplificar el proceso veamos cómo calcular la autoatención usando vectores, luego procederemos a ver cómo se implementa realmente, usando matrices, *The Illustrated Transformer* (s.f.).

El primer paso para calcular la autoatención es crear tres vectores de cada uno de los vectores de entrada al codificador, correspondientes a cada incrustación derivada token de la secuencia de entrada. Entonces, para cada incrustación, creamos un vector de consulta (query), un vector clave (key) y un vector de valor (value). Estos vectores se crean multiplicando la incrustación por tres matrices que se entrena en el proceso de entrenamiento, *The Illustrated Transformer* (s.f.).

Estos nuevos vectores son más pequeños en dimensión que el vector inicial. Su

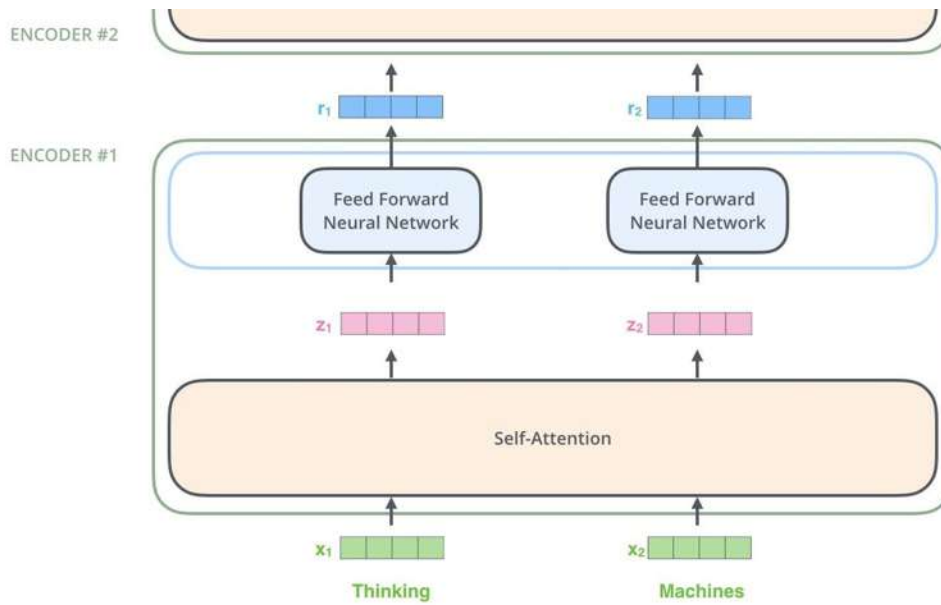


Figura 3.14: La lista de vectores es procesada por las dos capas del codificador, produciendo una salida que será la entrada del próximo codificador, *The Illustrated Transformer* (s.f.).

dimensionalidad es 64, mientras que los vectores de entrada/salida de incrustación y codificador tienen una dimensionalidad de 512. No tienen que ser más pequeños, esta es una elección de arquitectura para hacer que el cálculo de la atención de múltiples cabezas (en su mayoría) sea constante, *The Illustrated Transformer* (s.f.), figura 3.15.

3.5.6. Vectores Query, Key y Value

Los vectores **query**, **key** y **value** son abstracciones útiles para calcular la autoatención, *The Illustrated Transformer* (s.f.).

El segundo paso para calcular la autoatención es calcular una puntuación. Para calcular la autoatención de la palabra “Thinking” de la oración que utilizamos de ejemplo. Necesitamos puntuar cada palabra o token de la oración de entrada en relación con esta palabra. La puntuación determina cuánto enfoque poner en otras partes de la oración de entrada mientras codificamos una palabra en particular, que está en una posición determinada, *The Illustrated Transformer* (s.f.).

La puntuación se calcula tomando el producto escalar del vector **query** de una palabra en una posición determinada con el vector **key** de cada respectiva palabra en la secuencia de entrada con la que se compara la palabra que se está puntuando. Entonces, si procesamos la autoatención de la palabra en la posición 1, la primera puntuación sería el producto escalar de q_1 y k_1 . La segunda puntuación sería el producto escalar de q_1 y

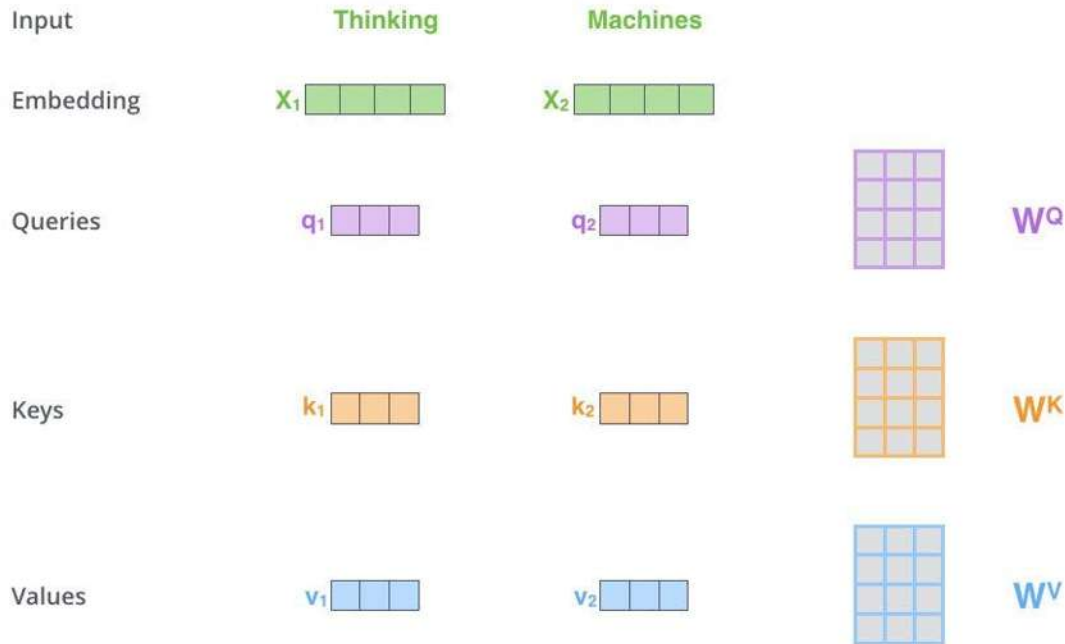


Figura 3.15: Multiplicar x_1 por la matriz de peso W^Q produce q_1 , el vector de “consulta” asociado con esa palabra. Terminamos creando una “consulta”, una “clave” y una proyección de “valor” de cada palabra en la oración de entrada, *The Illustrated Transformer* (s.f.).

k_2 , como se muestra en la figura 3.16, *The Illustrated Transformer* (s.f.).

Los pasos tercero y cuarto son dividir las puntuaciones entre la raíz cuadrada de la dimensión de los vectores **key** en este caso es 8 ya que la dimensión de los vectores es de 64. Esto conduce a tener gradientes más estables. Luego se pasa el resultado a través de una operación softmax. Softmax normaliza las puntuaciones para que todas sean positivas y sumen 1, *The Illustrated Transformer* (s.f.), véase figura 3.17.

Esta puntuación softmax determina cuánto se expresará cada palabra en esta posición. La palabra en esta posición tendrá la puntuación softmax más alta, pero a veces es útil prestar atención a otra palabra que sea relevante para la palabra actual, *The Illustrated Transformer* (s.f.).

El quinto paso es multiplicar cada vector **value** por la puntuación softmax (en preparación para sumarlos). La intuición aquí es mantener intactos los valores de las palabras en las que queremos centrarnos y darles menos importancia a las palabras irrelevantes (multiplicándolas por números pequeños como 0,001, por ejemplo), *The Illustrated Transformer* (s.f.).

El sexto paso es sumar todos los vectores de valor ponderado que se generan a partir de las diferentes puntuaciones. Esto produce la salida de la capa de autoatención en esta

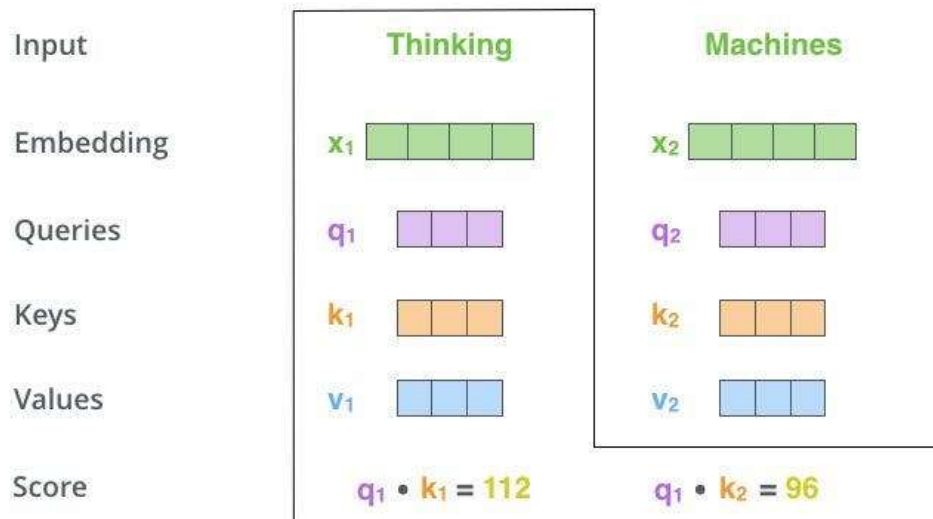


Figura 3.16: La puntuación es el producto escalar del vector consulta con el vector clave, *The Illustrated Transformer* (s.f.).

posición (para la primer palabra), ver figura 3.18, *The Illustrated Transformer* (s.f.).

Con eso concluye el cálculo de la autoatención. El vector resultante se manda a la red neuronal. En una implementación real, este cálculo se realiza en forma de matriz para un procesamiento más rápido, *The Illustrated Transformer* (s.f.).

3.5.7. Cálculo matricial de autoatención

Primero se calculan las matrices **QUERY**, **KEY** y **VALUE**. Lo hacemos empaquetando nuestras incrustaciones en una matriz **X** y multiplicándolas por las matrices de peso que hemos entrenado (W^Q , W^K , W^V), como se muestra en la figura 3.19, *The Illustrated Transformer* (s.f.).

Finalmente, se condensan los pasos del dos al seis en una fórmula para calcular los resultados de la capa de autoatención, *The Illustrated Transformer* (s.f.) (figura 3.20).

3.5.8. Atención de múltiples cabezas

La capa de autoatención cuenta con un mecanismo llamado atención de “múltiples cabezas”. Esto mejora el rendimiento de la capa de atención de dos formas, *The Illustrated Transformer* (s.f.):

- 1 Amplía la capacidad del modelo para enfocarse en diferentes posiciones. Sí, en

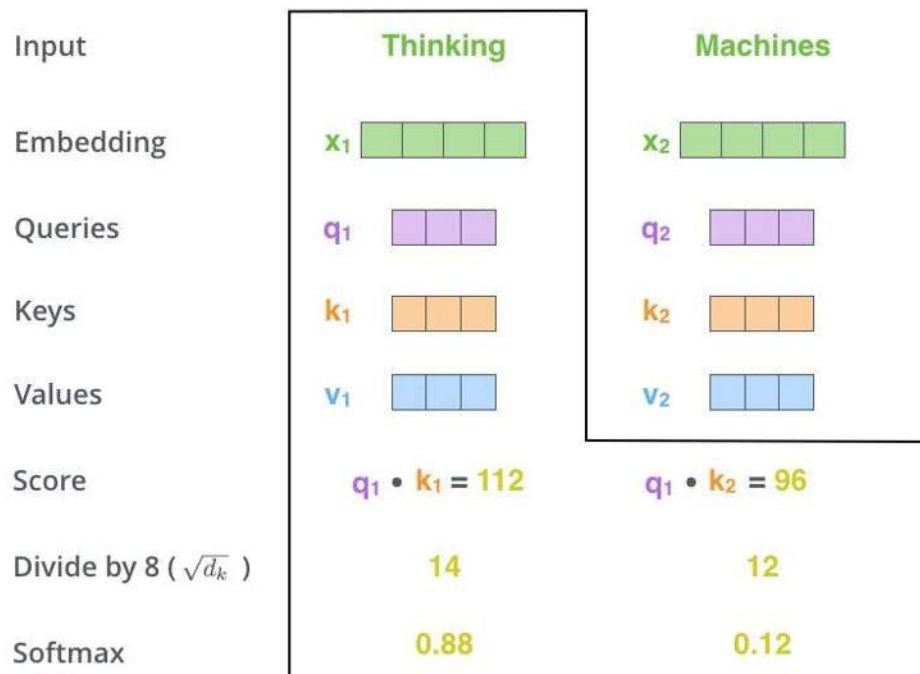


Figura 3.17: Las puntuaciones se dividen entre 8, después se pasan a una función softmax que las normaliza.

el ejemplo anterior, z_1 contiene un poco de las demás codificaciones, pero podría estar dominado por la propia palabra. Esto resulta útil en la traducción de una frase como “The animal didn’t cross the street because it was too tired”, para saber a qué parte de la oración se refiere “it”, *The Illustrated Transformer* (s.f.).

- 2 Le da a la capa de atención múltiples “subespacios de representación”. Con la atención de múltiples cabezas no solo tenemos uno, sino múltiples conjuntos de matrices de ponderación de **QUERY/KEY/VALUE**. Cada uno de estos conjuntos se inicializa aleatoriamente. Después del entrenamiento, cada conjunto se utiliza para proyectar las incorporaciones de entrada (o vectores de codificadores/decodificadores inferiores) en un subespacio de representación diferente, *The Illustrated Transformer* (s.f.), véase figura 3.21.

Con la atención de múltiples cabezas, se generan matrices de peso **Q/K/V** separadas para cada cabezal, lo que da como resultado diferentes matrices **Q/K/V**. Se multiplica **X** por las matrices $W^Q/W^K/W^V$ para producir las matrices **Q/K/V**, *The Illustrated Transformer* (s.f.).

El cálculo de autoatención se hace ocho veces con diferentes matrices de peso, generando ocho matrices **Z** diferentes como se muestra en la figura 3.22, *The Illustrated Transformer* (s.f.).

Como la capa de la red neuronal no espera ocho matrices, espera una sola matriz

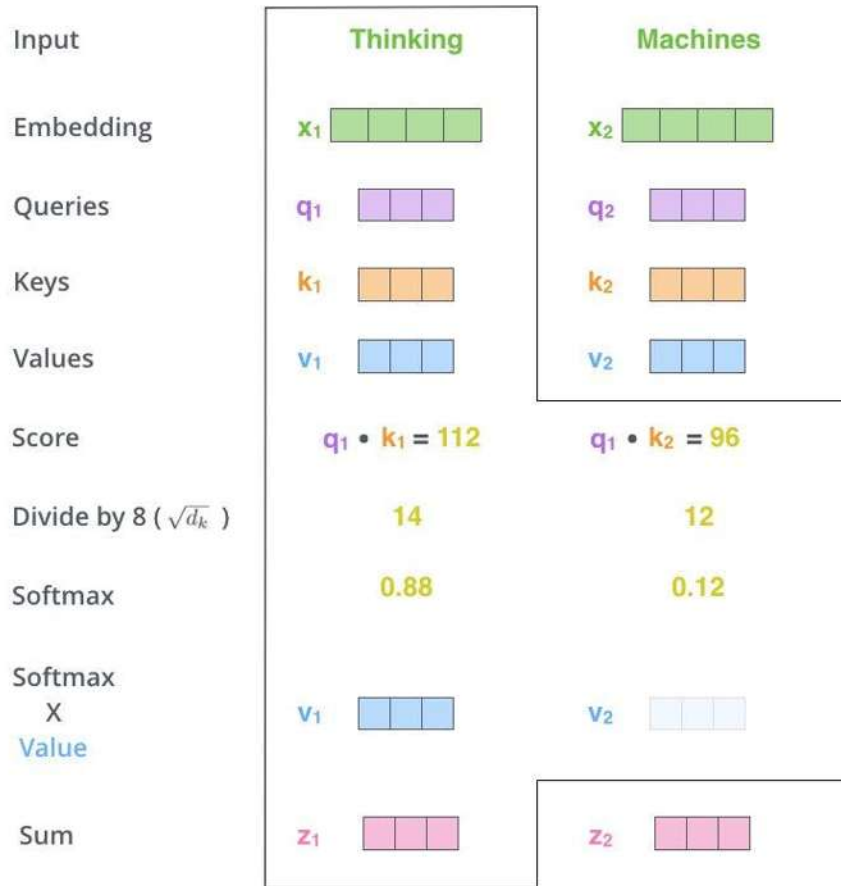


Figura 3.18: Suma de los vectores que produce la salida de la capa de autoatención, *The Illustrated Transformer* (s.f.).

(un vector para cada palabra). Se necesita una forma de condensar estas ocho en una sola matriz, *The Illustrated Transformer* (s.f.).

Para lograr esto se concatenan las matrices y luego se multiplican por una matriz de pesos adicional W^O , como se muestra en la figura 3.23, *The Illustrated Transformer* (s.f.).

Todo el proceso con todas las matrices se ilustra en la figura 3.24:

3.5.9. Representar el orden de la secuencia mediante codificación posicional

La ubicación y el orden de las palabras son parte esencial de cualquier idioma, definen la gramática y, por lo tanto, la semántica real de la oración, Kazemnejad (2019).

Dado que cada palabra en una oración fluye a través del bloque de codificación/descodificación

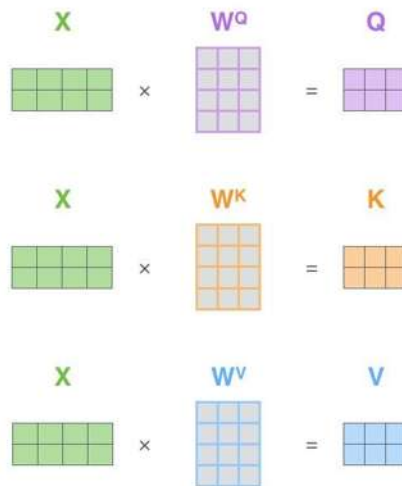


Figura 3.19: Cada fila de la matriz X corresponde a una palabra en la oración de entrada. Nuevamente vemos la diferencia en el tamaño del vector de incrustación (512 o 4 cajas en la figura) y los vectores $q/k/v$ (64 o 3 cajas en la figura), *The Illustrated Transformer* (s.f.).

del Transformer simultáneamente, el modelo no tiene sentido de posición/orden para cada palabra. Por lo tanto, se necesita una forma de incorporar el orden de las palabras en el modelo, Kazemnejad (2019).

Una solución para darle cierto sentido de orden al modelo, es incorporar información sobre su posición en la oración a cada palabra, Kazemnejad (2019).

Idealmente, se deben cumplir los siguientes criterios, Kazemnejad (2019):

- Debe generar una codificación única para cada posición de la palabra en una oración.
- La distancia entre dos posiciones de dos palabras en la oración debe ser consistente en oraciones con diferentes longitudes.
- El modelo debería generalizarse a oraciones más largas sin ningún esfuerzo. Sus valores deben estar acotados.
- Debe ser determinista.

La codificación propuesta por los autores satisface todos estos criterios. No es un solo número. Es un d -vector dimensional que contiene información sobre una posición específica en una oración, como esta codificación no está integrada en el propio modelo. Este vector se usa para equipar cada palabra con información sobre su posición en una oración, Kazemnejad (2019).

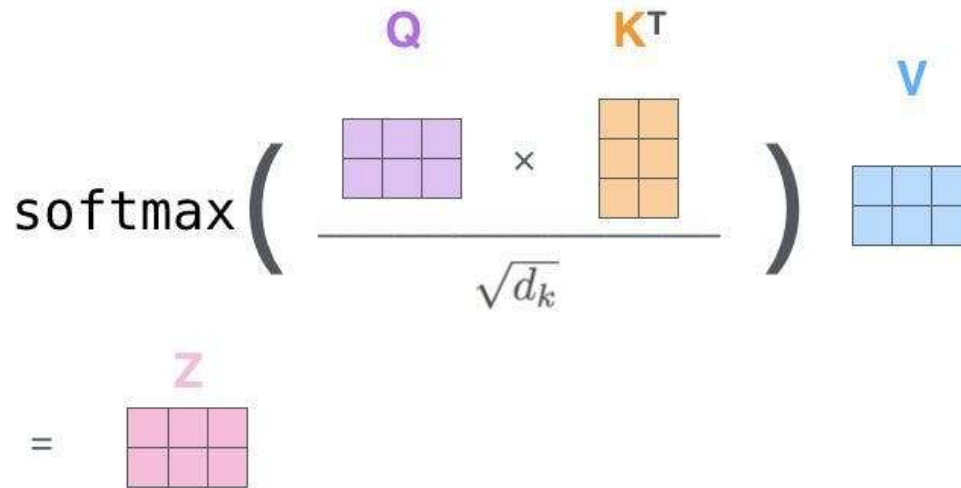


Figura 3.20: El cálculo de la autoatención en forma de matriz, *The Illustrated Transformer* (s.f.)

Sea t la posición deseada en una oración de entrada, sea $\vec{p}_t \in \mathbb{R}^d$ su codificación correspondiente, y d la dimensión de codificación (donde d tiene que ser divisible entre 2) Después $f : \mathbb{N} \rightarrow \mathbb{R}^d$ será la función que produce el vector de salida \vec{p}_t y se define de la siguiente manera, Kazemnejad (2019):

$$\vec{p}_t^{(i)} = f(t)^{(i)} := \begin{cases} \sin(w_k \cdot t) & , si \quad i=2k \\ \cos(w_k \cdot t) & , si \quad i=2k+1 \end{cases}$$

donde:

$$w_k = \frac{1}{10000^{2k/d}}$$

Como se puede deducir de la definición de la función, las frecuencias disminuyen a lo largo de la dimensión del vector. Por lo que, forma una progresión geométrica desde 2π a $10000 \cdot 2\pi$ en las longitudes de onda, Kazemnejad (2019).

Se puede visualizar la incrustación posicional \vec{p}_t como un vector de pares de senos y cosenos para cada frecuencia (Donde d es divisible entre 2), como se muestra en la figura 3.25, Kazemnejad (2019):

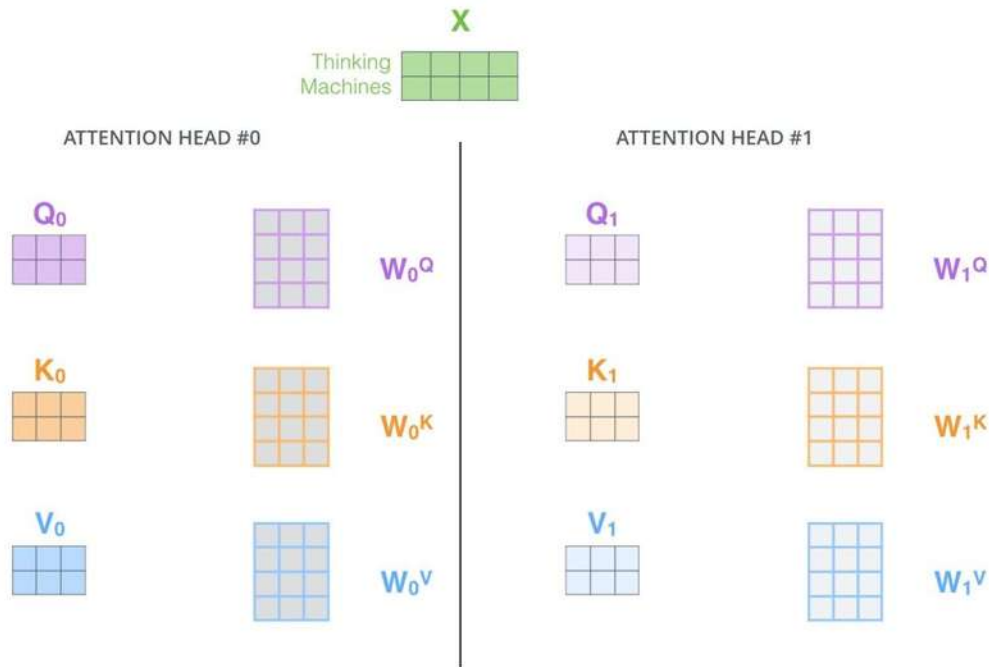


Figura 3.21: Cabezas de atención

$$\vec{p}_t = \begin{bmatrix} \sin(w_1 \cdot t) \\ \cos(w_1 \cdot t) \\ \sin(w_2 \cdot t) \\ \cos(w_2 \cdot t) \\ \dots \\ \sin(w_{d/2} \cdot t) \\ \cos(w_{d/2} \cdot t) \end{bmatrix}_{dx1}$$

3.5.10. Los residuos

En la arquitectura del codificador, cada subcapa en los codificadores tienen una conexión residual a su alrededor, y es seguida por un paso de normalización de capa, *The Illustrated Transformer* (s.f.) (figura 3.26).

Los vectores y la operación de norma de capa asociada con la autoatención, se vería como en la figura 3.27, *The Illustrated Transformer* (s.f.):

Esto también se aplica a las subcapas del decodificador. En un transformer de 2 codificadores y 2 decodificadores apilados, se vería como en la figura 3.28, *The Illustrated Transformer* (s.f.):

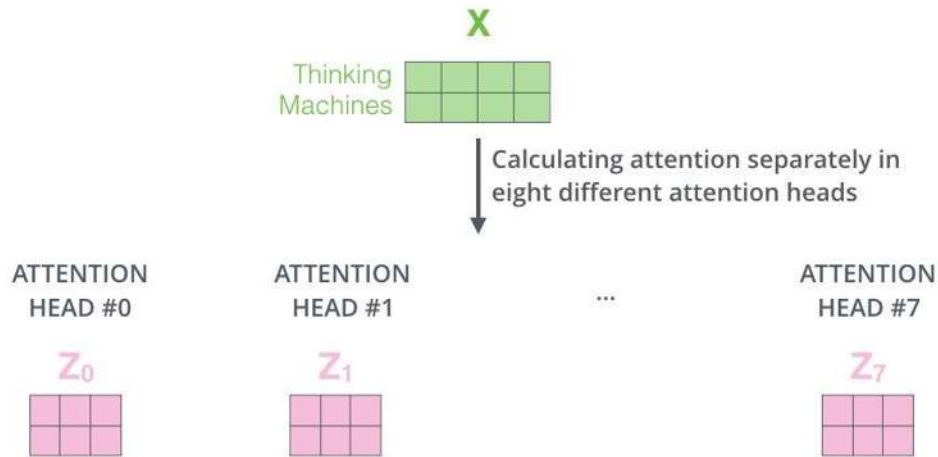


Figura 3.22: 8 matrices Z , *The Illustrated Transformer* (s.f.).

3.5.11. Decodificador

Los conceptos utilizados hasta ahora para explicar los componentes del codificador son los mismos que se necesitan para los del decodificador, *The Illustrated Transformer* (s.f.).

El codificador comienza procesando la secuencia de entrada. La salida del codificador superior se transforma luego en un conjunto de vectores de atención K y V . Estos deben ser utilizados por cada decodificador en su capa de “atención del codificador-decodificador” que ayuda al decodificador a enfocarse en los lugares apropiados en la secuencia de entrada, *The Illustrated Transformer* (s.f.):

Después de finalizar la fase de codificación, sigue la fase de decodificación. Cada paso de la fase de decodificación genera un elemento de la secuencia de salida, *The Illustrated Transformer* (s.f.).

Los siguientes pasos repiten el proceso hasta que se alcanza un símbolo que indica que se ha completado la salida en su totalidad. La salida de cada paso se envía al decodificador superior en el siguiente paso de tiempo, y los decodificadores aumentan sus resultados de decodificación como lo hicieron los codificadores. Y al igual que con las entradas del codificador, se agrega la codificación posicional a las entradas del decodificador para indicar la posición de cada palabra, *The Illustrated Transformer* (s.f.).

Las capas de autoatención en el decodificador operan de una manera ligeramente diferente a la del codificador, *The Illustrated Transformer* (s.f.):

En el decodificador, la capa de autoatención solo atiende las posiciones anteriores en la secuencia de salida. Esto se hace enmascarando las posiciones futuras antes del paso

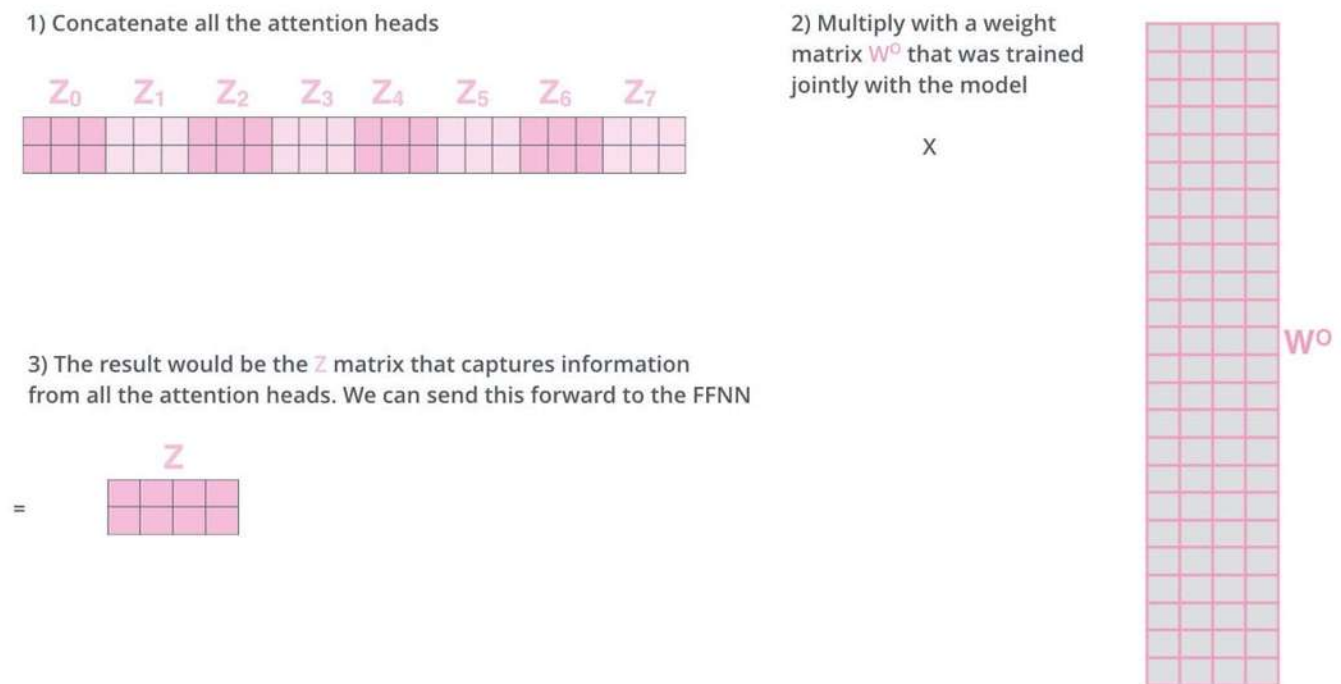


Figura 3.23: Matriz Z_n concaatenads y multiplicada por W^O , *The Illustrated Transformer* (s.f.)

softmax en el cálculo de autoatención, *The Illustrated Transformer* (s.f.).

La capa “Atención del codificador-decodificador” funciona como la autoatención de múltiples cabezas, excepto que crea su matriz de **QUERIES** a partir de la capa que está debajo y toma la matriz **KEYS** y **VALUES** de la salida de la pila del codificador, *The Illustrated Transformer* (s.f.).

3.5.12. La capa final lineal y Softmax

La pila de decodificadores genera un vector de flotantes. La capa lineal final que es seguida por una capa Softmax los convierte en palabras, *The Illustrated Transformer* (s.f.).

La capa lineal es una red neuronal simple completamente conectada que proyecta el vector producido por la pila de decodificadores en un vector mucho más grande llamado vector logits, *The Illustrated Transformer* (s.f.).

Por ejemplo, si nuestro modelo conoce 10,000 palabras únicas en inglés que ha aprendido de su conjunto de datos de entrenamiento. Esto haría que el vector logits tenga un ancho de 10,000 celdas, cada celda correspondiente a la puntuación de una palabra única. Así es como interpretamos la salida del modelo seguida de la capa lineal,

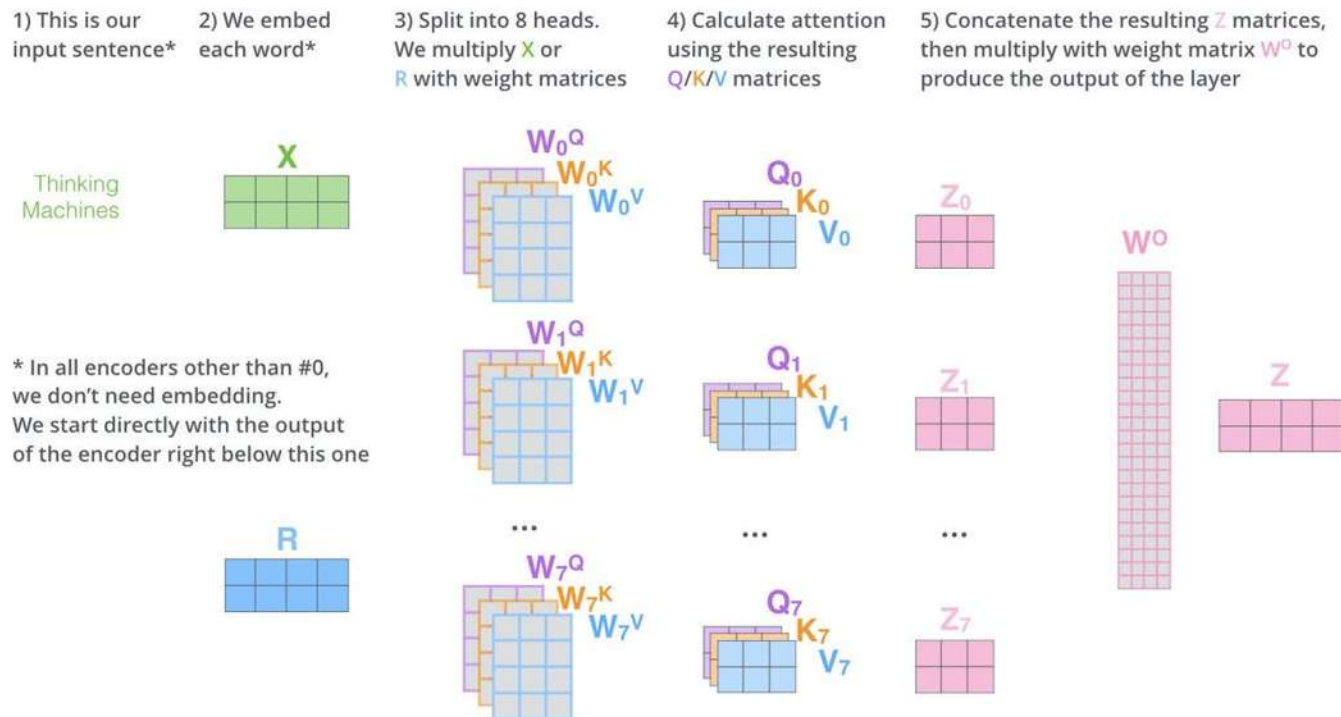


Figura 3.24: Proceso con todas las matrices, *The Illustrated Transformer* (s.f.).

The Illustrated Transformer (s.f.).

La capa softmax convierte esos puntajes en probabilidades (todos positivos y suman 1). Se elige la celda con la probabilidad más alta y la palabra asociada con ella se genera como salida para este paso de tiempo, *The Illustrated Transformer* (s.f.), figura 3.29.

3.5.13. Resumen del entrenamiento

Durante el entrenamiento, un modelo no entrenado pasaría exactamente por el mismo pase hacia adelante. Pero como lo estamos entrenando en un conjunto de datos de entrenamiento etiquetado, podemos comparar su salida con la salida correcta real, *The Illustrated Transformer* (s.f.).

Dado que los parámetros del modelo (pesos) se inicializan todos al azar, el modelo (no entrenado) produce una distribución de probabilidad con valores arbitrarios para cada celda/palabra. Podemos compararlo con la salida real, luego ajustar todos los pesos del modelo usando retropropagación (backpropagation) para acercar la salida a la salida deseada, *The Illustrated Transformer* (s.f.).

Después de entrenar el modelo durante suficiente tiempo en un conjunto de datos lo suficientemente grande, esperamos que las distribuciones de probabilidad producidas,

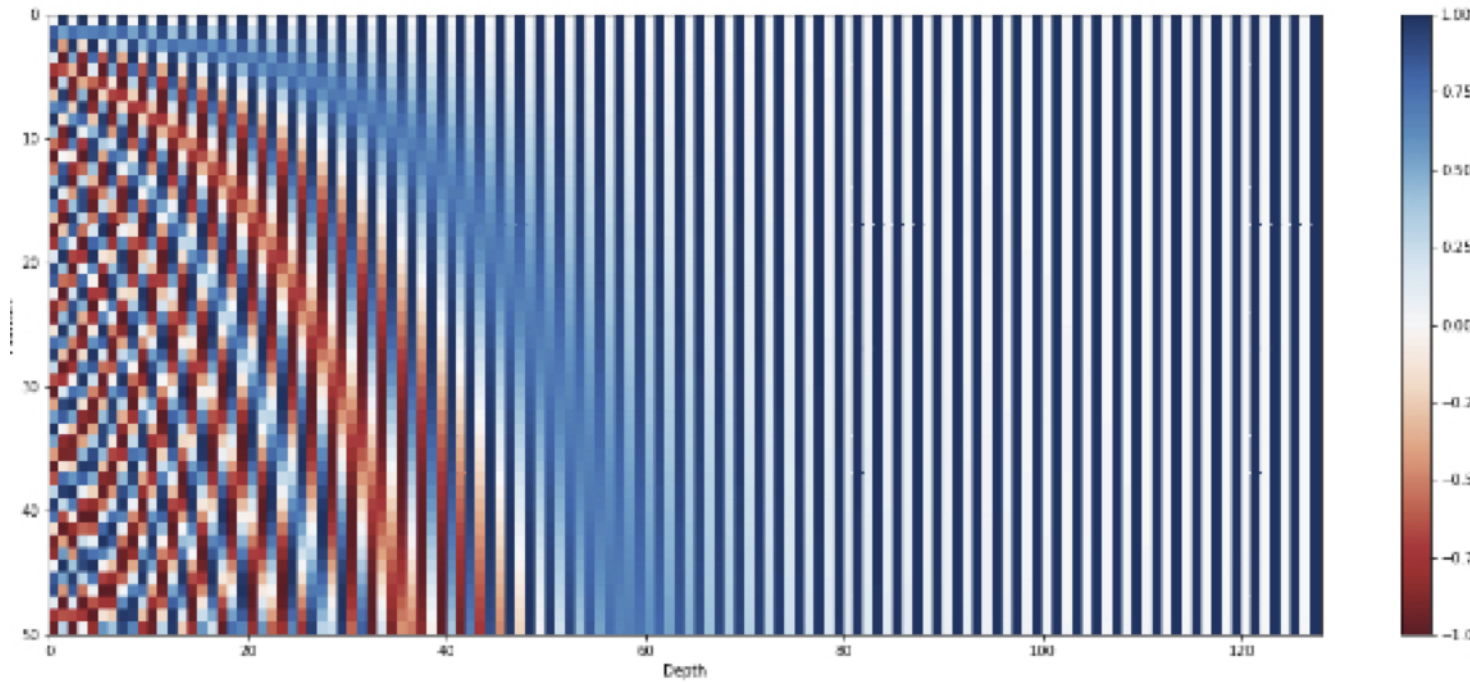


Figura 3.25: La codificación posicional de 128 dimensiones para una oración con una longitud máxima de 50. Cada fila representa el vector de incrustación \vec{p}_t , Kazemnejad (2019)

sean las salidas deseadas.

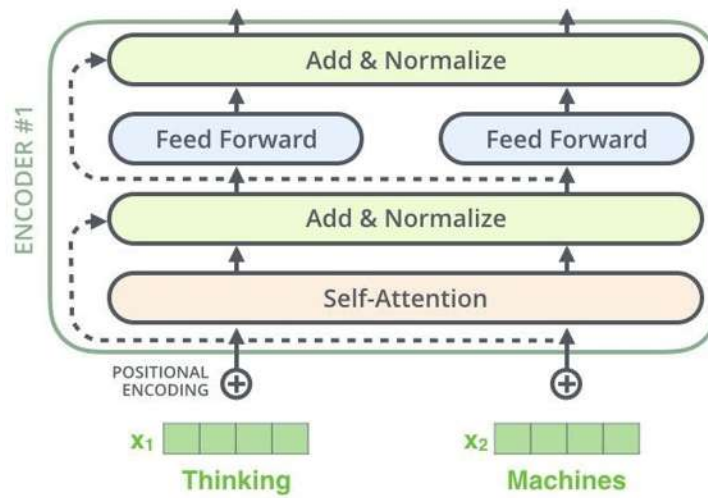


Figura 3.26: Arquitectura de un codificador, *The Illustrated Transformer* (s.f.)

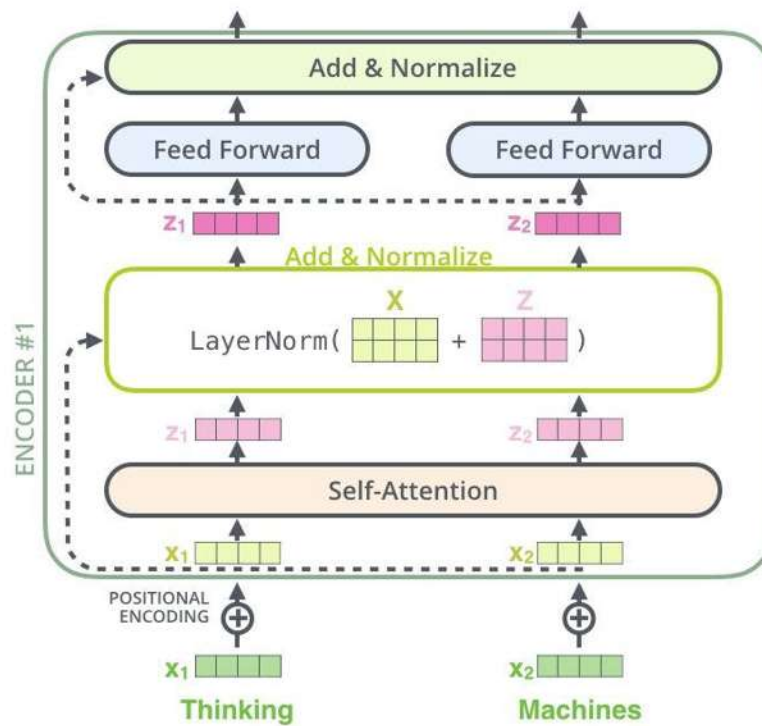


Figura 3.27: Capa de suma y normalización, *The Illustrated Transformer* (s.f.)

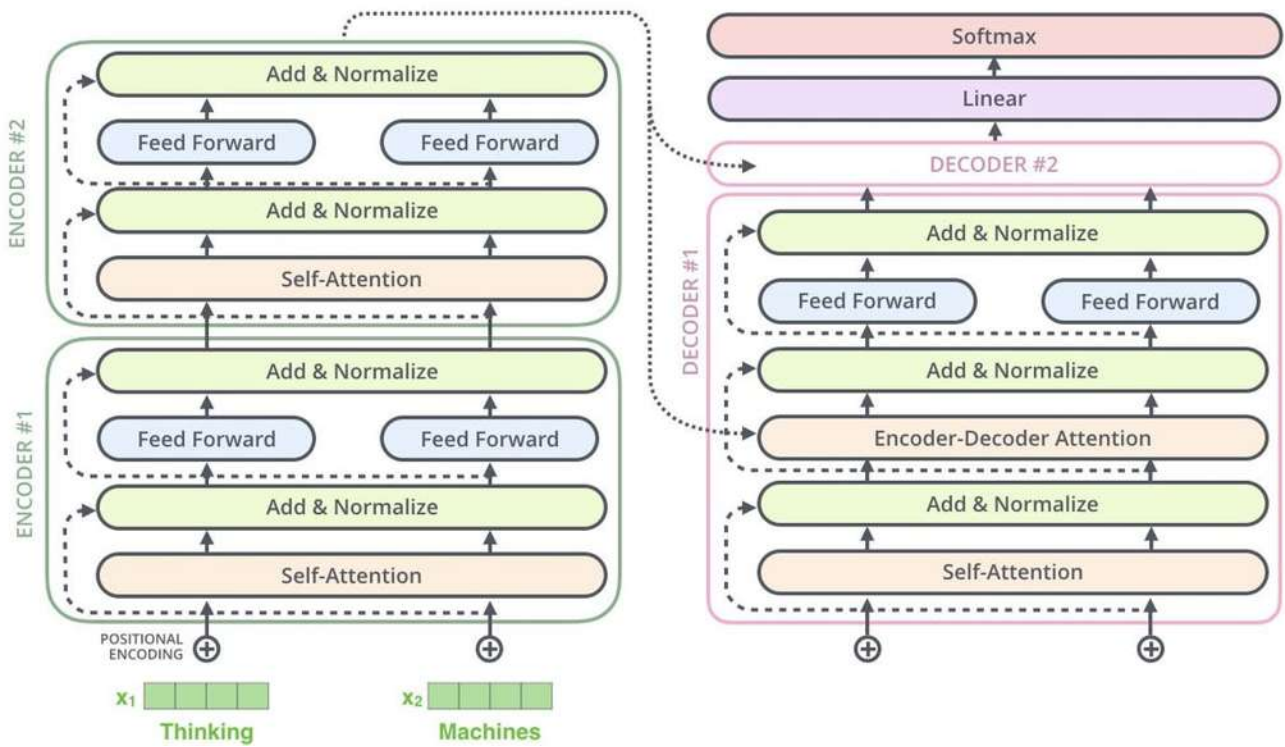


Figura 3.28: 2 codificadores y 2 decodificadores, *The Illustrated Transformer* (s.f.)

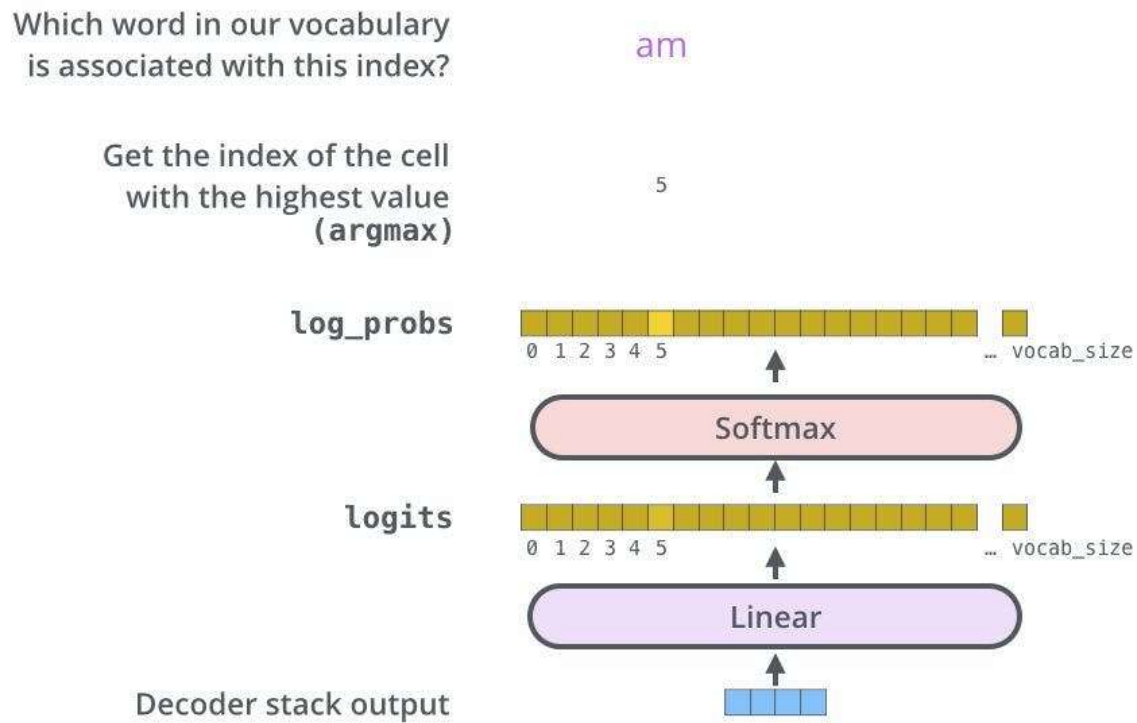


Figura 3.29: Esta figura comienza desde abajo con el vector producido como salida de la pila de decodificadores. Luego se convierte en una palabra de salida, *The Illustrated Transformer* (s.f.).

Capítulo 4

Bases conceptuales del purépecha

4.1. Las consonantes

El alfabeto purépecha comparte con el español (que es referente más cercano) la mayoría de sus letras por lo tanto de sus sonidos. En el Purépecha existen sonidos particulares, compuestos por dos letras o están marcados por una característica específica como es en el caso de las consonantes aspiradas. En el Purépecha, se cuenta con 18 consonantes, y dos semiconsonantes. Entre las consonantes están:

- Las consonantes simples: p, t, k, ch, ts, j, m, n, s, x.
- Las consonantes aspiradas: p', t', k', ch', ts'.
- Tanto las consonantes aspiradas como la alveolar transcrita "ts" y la palatal escrita "ch", la "nh" y la "rh" tienen una articulación compleja. No obstante, cada unidad constituye un solo fonema, aunque se represente gráficamente con dos signos.

Por otra parte, ninguna consonante se presenta en posición final, Chamoreau (2009).

También se cuenta con dos semiconsonantes que son la "y" y la "w". Estas semiconsonantes funcionan como auxiliares, ya que en la lengua purépecha existen sílabas con hasta cuatro vocales juntas, como la palabra "iauaní". Lo que podría producir dificultades a la hora de lectura para las personas que no están familiarizadas con la

p	t	k	ts	ch	j
p'	t'	k'	ts'	ch'	
m	n	nh	rh	s	x

Tabla 4.1: Consonantes de la lengua Purépecha

lenguas. Por lo que, con este acuerdo con bases y fundamentos lingüístico se simplifica la escritura y lectura de este tipo de palabras.

4.2. Las vocales

El purépecha cuenta con seis vocales. Se organizan en tres posiciones de la lengua (de la posición anterior a la posición posterior) y en tres grados de apertura de la boca como se muestra en la figura 4.1) Chamoreau (2009).

		<i>Posición de la lengua</i>		
		<i>anterior</i>	<i>central</i>	<i>Posterior</i>
<i>Grado de Apertura</i>	<i>Cerrada</i>	i	i	U
	<i>Semicerrada</i>	e		O
	<i>Abierta</i>		a	

Figura 4.1: Vocales, referencia:Chamoreau (2009)

Todas las vocales se presentan en posiciones interna y final de palabra. La vocal central es la única que no se presenta en posición inicial y sólo aparece precedida por fonemas alveolares /s/, /ts/, /tsh/ y por la palatal, Chamoreau (2009).

4.3. El acento

El acento puede aparecer en la primera o en la segunda sílaba. Para facilitar la lectura, sólo se escribirá cuando esté presente en la primera sílaba (el acento va sobre la vocal). Es importante marcarlo, ya que existen unidades que no se oponen si no es por el acento, como en español “papa/papá”. Por ejemplo, véase figura 4.2:

wé-ra-ni	<i>salir</i>	we-ra-ni	<i>llorar</i>
kára-ni	<i>volar</i>	kara-ni	<i>escribir</i>
píri-ni	<i>acercarse</i>	piri-ni	<i>cantar</i>

Figura 4.2: El acento, referencia:Chamoreau (2009)

4.4. Flexión verbal

La flexión verbal o conjugación verbal se podría considerar como la base medular para el entendimiento de la lengua Purépecha, ya que es una lengua verbal, esto quiere decir que la mayor parte de las palabras provienen de los verbos.

La raíz de las palabras en la flexión verbal provienen de la raíz de un verbo en infinitivo. Los verbos en infinitivo terminan en “ni”, así que la raíz de pirini (cantar) sería piri.

La lengua Purépecha es aglutinante y exclusivamente sufijante, por lo que, los morfemas que conforman a la palabra van después de la raíz de la palabra. Estos sufijos indican cómo, cuándo, si afirmas o preguntas y la persona gramatical que realiza la acción, como se muestra en la figura 4.3.



Figura 4.3: Flexión verbal básica

4.5. Los aspectos

De manera general, un aspecto permite expresar la forma en la que el hablante expresa el proceso en sí mismo, excluyendo cualquier idea de tiempo: puede resaltar su desarrollo, su duración, su calidad, su inicio, su fin, etcétera, Chamoreau (2009).

Esta clase está compuesta por cuatro unidades: el aoristo, el progresivo, el habitual y el continuo.

4.5.1. El aspecto aoristo (o perfecto)

Este aspecto es también llamado perfecto. Este término es más conocido que aoristo, pero presenta la desventaja de no cubrir más que parcialmente sus usos. El aspecto aoristo es con certeza el más frecuente y el más complejo, Chamoreau (2009).

Su forma siempre es la misma, sin importar la persona o los determinantes con los que coexiste. Según las variedades dialectales, el sufijo es x o s, Chamoreau (2009).

En Jarácuaro, la forma es x:

- piri-x-ka=ni Yo he cantado
- piri-x-ka=ri Tú has cantado
- piri-x-ti Él/ella ha cantado
- piri-x-ka=kxĩ Nosotros hemos cantado
- piri-x-ka=ts'ĩ Ustedes han cantado
- piri-x-ti=kxĩ Ellos/ellas han cantado

4.5.2. El aspecto progresivo

La presencia de este elemento expresa la duración de un evento. El predicado está marcado por el sufijo xa y no presenta ninguna variación de forma Chamoreau (2009).

En español se puede traducir por el presente o el presente perifrástico:

- piri-xa-ka=ni Yo canto (yo estoy cantando)
- piri-xa-ka=ri Tú cantas (tú estás cantando)
- piri-xa-ti Él/ella canta (él/ella está cantando)
- piri-xa-ka=kxĩ Nosotros cantamos (nosotros estamos cantando)
- piri-xa-ka=ts'ĩ Ustedes cantan (ustedes están cantando)
- piri-xa-ti=kxĩ Ellos/ellas cantan (ellos/ellas están cantando)

4.5.3. El aspecto habitual

Este aspecto se relaciona con el carácter habitual del proceso, ya sea que se considere frecuente o habitual, ya sea que se repita (lo que se llama una acción iterativa). Está indicado con el segmento sîn (o xîn en ciertas variedades dialectales) Chamoreau (2009).

- piri-sîn-ka=ni Yo canto (suelo cantar).
- piri-sîn-ka=ri Tú cantas (sueles cantar).
- piri-sîn-ti Él/ella canta (suele cantar).

- piri-sin-ka=kxi Nosotros cantamos (solemos cantar).
- piri-sin-ka=ts'i Ustedes cantan (suelen cantar).
- piri-sin-ti=kxi Ellos/ellas cantan (suelen cantar).

El empleo del habitual no equivale al presente: el habitual es un aspecto diferente al progresivo.

4.6. Los modos

De manera general, un modo permite conocer la actitud del hablante sobre lo que dice Chamoreau (2009):

- si se está afirmando, se utilizará el asertivo. Este modo se conoce también como indicativo, que aunque es más conocida, no permite mostrar que en purépecha la estrategia esencial es la de oponer una aserción a una interrogación.
- si se hace una pregunta, se utilizará el modo interrogativo.

4.6.1. El modo asertivo (o indicativo)

Este modo es el más frecuente. Permite una aserción, ya sea afirmativa o negativa. Es decir, se utiliza cuando el enunciado no es una interrogación, una orden o una exclamación, Chamoreau (2009).

En purépecha, el asertivo presenta dos formas:

- Para las primeras o segundas personas tanto del singular como del plural, la forma es ka.
- Para las terceras personas, en singular y en plural, la forma es ti.

Veamos el paradigma del verbo “cantar”, pirini:

- piri-ka=ni Yo canto
- piri-ka Yo canto
- piri-ka=ri Tú cantas

- piri-ti Él/ella canta
- piri-ka=kxĩ Nosotros cantamos (yo y él)
- piri-ka=ch'ĩ Nosotros cantamos (yo y tú)
- piri-ka=ts'ĩ Ustedes cantan
- piri-ti=kxĩ Ellos/ellas cantan

Veamos algunos ejemplos de utilización con los aspectos:

EL ASERTIVO CON EL ASPECTO PROGRESIVO

- piri-xa-ka=ri Tú cantas, tú estás cantando.
- piri-xa-ti Él canta, él está cantando.

EL ASERTIVO CON EL ASPECTO HABITUAL

- piri-xĩn-ka=ri Tú cantas, tú tienes la costumbre de cantar.
- piri-xĩn-ti Él canta, él tiene la costumbre de cantar.

EL ASERTIVO CON EL TIEMPO FUTURO

- piri-a-ka=ri Tú cantarás.
- piri-a-ti Él cantará.

4.6.2. El modo interrogativo

Este modo se emplea forzosamente cuando el enunciado es una pregunta. Se opone, por lo tanto, al asertivo, Chamoreau (2009).

- Pedru ixex-ti Pablu-ni Pedro ha visto a Pablo.
- Pedru ixex-ki Pablu-ni ¿Ha visto Pedro a Pablo?

El modo interrogativo puede estar marcado por diferentes formas, en función de los aspectos o tiempos que lo acompañan, Chamoreau (2009):

- La forma es *ki* cuando el interrogativo está acompañado por el aoristo, el progresivo y el continuo.
- Se emplea la forma *ø* después del futuro.
- En los demás casos es la forma *i* la que está presente (después del pasado, el habitual o el condicional).

Cualquiera que sea la forma que se emplee, se mantiene la misma para todas las personas. Veamos con el verbo *wérani* “salir-formativo”, ?:

- *wé-ra-x-ki=ni* ¿He salido?
- *wé-ra-x-ki=ri* ¿Has salido?
- *wé-ra-x-ki* ¿Ha salido?
- *jucha wé-ra-x-ki=kxĩ* ¿Hemos salido?
- *wé-ra-x-ki=ts'ĩ* ¿Han salido (ustedes)?
- *tsĩma wé-ra-x-ki=kxĩ* ¿Han salido (ellos)?

Observemos algunos ejemplos de utilización con aspectos y modos diferentes:

CON EL PROGRESIVO

- *we-ra-xa-ki=ri* ¿Lloras? ¿Estás llorando?

CON EL FUTURO

- *nani=kxĩ ni-a-ø pawani* ¿A dónde iremos mañana?

CON EL HABITUAL O EL PASADO

- *t'ũ ju-rha-xĩn-i mantani wéxurhini* ¿Vienes cada año?
- *t'ũ ju-rha-x-p-i* ¿Habías venido?

4.7. Los tiempos

En la clase de los tiempos, podemos plantear la existencia de dos unidades, el pasado y el presente Chamoreau (2009).

4.7.1. El tiempo presente

El presente es un tiempo puntual que se utiliza cuando el momento de la enunciación corresponde con el momento del proceso. Este morfema se indica con una ausencia de marca ó con un \emptyset .

4.7.2. El tiempo pasado

El tiempo pasado permite la localización del proceso en lo “ya realizado”. Se presentan dos alomorfos: *p* y *an*. Recordemos que el tiempo pasado es dependiente, esto es, no puede aparecer solo, sino que debe estar acompañado forzosamente por alguno de los cuatro aspectos Chamoreau (2009).

La repartición de las ocurrencias de *p* y de *an* permite sacar la conclusión de que representan dos variantes contextuales de la misma unidad, que es el pasado. Se distribuyen de este modo Chamoreau (2009):

- *p* coexiste con el aoristo y el progresivo.
- *an* coexiste con el habitual y el continuo.

La coexistencia con un aspecto permite precisar el valor del pasado.

4.8. Los clíticos personales

Los clíticos personales son las formas más utilizadas en general, aunque en ciertas variedades son los pronombres más empleados.

Las formas son diferentes según su función en la oración: una forma para la función sujeto y otra forma para la función de objeto.

Se llama clítico a un elemento átono que puede posponerse o anteponerse a diversas unidades. En el purépecha, se pospone siempre, por lo tanto se puede llamar enclítico (cuando se antepone se llama proclítico). Para funcionar, necesita un huésped. Se puede posicionar después de un verbo después de las marcas de tiempo, aspecto y modo, pero generalmente se localiza después de cualquier unidad que inicia la oración o que se presenta en la primera frase de la oración. Puede aparecer después de algún adverbio, negación, interrogativo, subordinante, etcétera, Chamoreau (2009).

LOS CLÍTICOS PERSONALES						
	<i>Singular</i>			<i>Plural</i>		
	1	2	3	1	2	3
<i>Sujeto</i>	Ni	Ri	Ø	kxĩ	ts'ĩ	kxĩ
	Ø			ch'e		

Figura 4.4: Los clíticos personales, referencia:Chamoreau (2009)

Capítulo 5

Materiales y Metodología

En este capítulo se presentan los materiales, herramientas y el proceso que se tuvo que llevar a cabo para la generación automática de corpus que se utilizó para entrenar al modelo de traducción. Además de describir los pasos necesarios antes de poder entrenar al modelo. Así como la elección de este, la descripción de su arquitectura y la inicialización de sus parámetros.

También se presentan las métricas utilizadas para la evaluación de la red.

5.1. Materiales

En esta parte presentaremos los materiales usados, primero el ambiente de desarrollo, google colab, luego el lenguaje de programación empleado.

5.1.1. Google Colab

"Google Colab.^{es} una herramienta para escribir y ejecutar código Python en la nube de Google. También puede incluir texto enriquecido, enlaces e imágenes. Si necesita un alto rendimiento informático, el entorno le permite configurar ciertas propiedades del equipo en el que se ejecuta su código.

El uso de "Google Colab" permite disponer de un entorno para llevar a cabo tareas que serían difíciles de realizar en un equipo personal. Por otro lado, siguiendo la idea de "Drive", "Google Colab" brinda la opción de compartir los códigos realizados, *Breve introducción a Google Colab* (s.f.).

5.1.2. JoeyNMT

JoeyNMT es un conjunto de herramientas de traducción automática neuronal minimalista con fines educativos, Kreutzer, Bastings, y Riezler (2019).

Su objetivo es ser una base de código limpia y minimalista para ayudar a los novatos que buscan la comprensión de la traducción automática neuronal.

5.2. Corpus

Un corpus lingüístico es una colección grande y estructurada de ejemplos reales del uso del lenguaje. En términos de estructura, variedad y complejidad, el corpus debe reflejar la lengua o su método con la mayor precisión posible; En cuanto a su uso, se debe tener cuidado con que su representación sea correcta.

Para los modelos de NMT, el corpus requerido, consiste en un conjunto de oraciones o ejemplos en un idioma, emparejados con su traducción al idioma de destino.

5.3. Algoritmo para la construcción automática del corpus

Ya que para el idioma Purépecha se cuenta con pocas traducciones al español, se desarrolló un algoritmo que crea oraciones simples basadas en la flexión verbal o conjugación de verbos de la lengua Purépecha produciendo también su traducción al español.

Los verbos que se utilizaron para la conjugación, se seleccionaron del "vocabulario del idioma Purépecha" de Maxwell Lathrop, Lathrop (2007).

Este algoritmo está compuesto por dos partes; la primera construye las palabras en Purépecha, identificando la raíz de los verbos y añadiéndoles todas las combinaciones posibles de los morfemas correspondientes y la segunda produce la traducción al español de cada una de estas palabras, basándonos en las reglas ya mencionadas.

La conjugación de estos verbos produce una palabra en la lengua Purépecha pero su traducción al español, no necesariamente será una palabra, sino una oración simple. Para este trabajo se tomaron la raíz y los morfemas de la palabra en Purépecha como palabras, en lugar de partes de la palabra, ya que la red obtiene mejores resultados que al dejar solo la palabra compuesta.

5.3.1. Primera parte: Conjugación de verbos en Purépecha

En este proceso se toman en cuenta las cuatro partes por las que están conformadas las palabras, que son los morfemas que indican; el aspecto, el tiempo, el modo y el enclítico.

El algoritmo toma la raíz del verbo y le añade todas las posibles combinaciones con los diferentes morfemas, que indican las diferentes conjugaciones del verbo.

- Para el aspecto usamos; *sìn* , *s*, *xa* y *x*.
- Para el tiempo; ' ', *p* y *a*.
- Para el modo; *ka*, *ti* y *ki*.
- Para el enclítico; *ni*, *ri*, ' ', *kxĩ*, *ts'ĩ* y *ch'e*.

5.3.2. Segunda parte: Conjugación de verbos en Español

Para producir la traducción al Español se identifican las diferentes terminaciones que existen en los verbos en Español. Estas son: “ar”, “er” e “ir”.

Esta clasificación de los verbos nos sirve para asignarles su correspondiente terminación en sus diferentes formas conjugadas.

Para este trabajo se tomaron en cuenta la primera, segunda y tercera persona en singular y plural, los modos; asertivo e interrogativo, los aspectos; aoristo, progresivo y habitual y los tiempos; presente y pasado.

El algoritmo recibe una lista con el verbo, sus conjugaciones en Purépecha y la traducción del verbo en Español. Se clasifica el verbo (en “ar”, “er” ó “ir”) y se toma la raíz de este. Cada morfema de la palabra en Purépecha nos indica como se irá formando la traducción al español.

Por ejemplo:

- 'patsajtsĩ ni' = aprender
- 'patsajtsĩ sìn ' ' ka ni' = aprendo
- 'patsajtsĩ sìn ' ' ka ri' = aprendes
- 'patsajtsĩ sìn ' ' ka kxĩ' = aprendemos
- 'patsajtsĩ sìn ' ' ti ' ' = aprende

- 'patsajtsi sin ' ' ki ni' = ¿aprendo?
- 'patsajtsi s p ka ni' = aprendí
- 'patsajtsi s p ka ri' = aprendiste
- 'patsajtsi s p ka kxi' = aprendimos
- 'patsajtsi s p ti ' ' = aprendió
- 'patsajtsi s p ti kxi' = aprendieron
- 'patsajtsi s p ki ni' = ¿aprendí?
- 'patsajtsi s p ki ri' = ¿aprendiste?
- 'patsajtsi xa ' ' ka ni' = estoy aprendiendo
- 'patsajtsi xa ' ' ka ri' = está aprendiendo
- 'patsajtsi xa ' ' ka kxi' = estás aprendiendo
- 'patsajtsi xa ' ' ti ' ' = están aprendiendo
- 'patsajtsi xa ' ' ti kxi' = estamos aprendiendo
- 'patsajtsi xa ' ' ki ni' = ¿estoy aprendiendo?
- 'patsajtsi xa ' ' ki ri' = ¿está aprendiendo?
- 'patsajtsi xa ' ' ki kxi' = ¿estás aprendiendo?
- 'patsajtsi xa p ka ni' = estaba aprendiendo
- 'patsajtsi xa p ka ri' = estabas aprendiendo
- 'patsajtsi xa p ka kxi' = estábamos aprendiendo
- 'patsajtsi xa p ti ' ' = estaba aprendiendo
- 'patsajtsi xa p ti kxi' = estaban aprendiendo
- 'patsajtsi xa p ki ni' = ¿estaba aprendiendo?
- 'patsajtsi xa p ki ri' = ¿estabas aprendiendo?
- 'patsajtsi xa p ki kxi' = ¿estábamos aprendiendo?
- 'patsajtsi x ' ' ka ni' = he aprendido
- 'patsajtsi x ' ' ka ri' = has aprendido
- 'patsajtsi x ' ' ka kxi' = hemos aprendido
- 'patsajtsi x ' ' ti ' ' = ha aprendido
- 'patsajtsi x ' ' ki ni' = ¿he aprendido?

5.4. Selección de datos y Entrenamiento del modelo

5.4.1. Preparación de datos

Para entrenar un modelo de traducción, se necesitan una colección de oraciones fuente y traducciones de referencia que estén alineadas oración por oración y almacenadas en dos archivos, de modo que cada línea en el archivo de referencia sea la traducción de la misma línea en el archivo fuente.

Nuestra tarea de traducción automática es traducir palabras compuestas a partir de verbos (flexión verbal) en Purépecha a oraciones o conjugaciones al Español.

El algoritmo para la construcción automática de corpus, genera 12991 ejemplos de entrenamiento y 1 000 ejemplos de desarrollo y 1000 para pruebas.

5.4.2. Preprocesamiento

Antes de entrenar un modelo, los datos paralelos se filtran por relación de longitud, tokenizados y en minúsculas.

JoeyNMT admite modelos de subpalabras con codificaciones de pares de bytes (BPE) que se aprenden con subword-nmt o sentencepiece.

Estos repositorios (subword-nmt y sentencepiece) contienen scripts de preprocesamiento para segmentar texto en unidades de subpalabras.

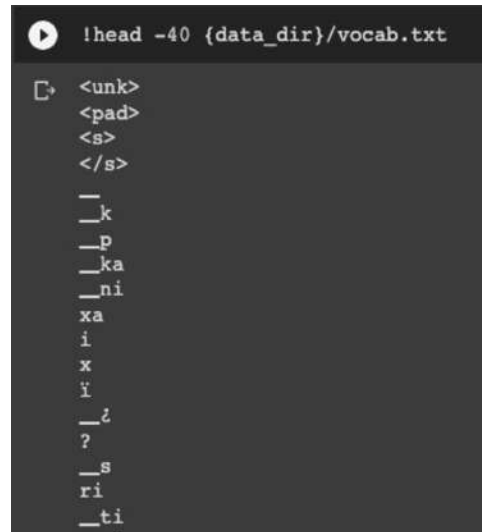
Se utilizó la librería sentencepiece para dividir palabras en subpalabras (BPE) según su frecuencia en el corpus de entrenamiento.

El script *build_vocab.py* entrena el modelo BPE y crea un vocabulario conjunto. Toma el mismo archivo de configuración que joeynmt.

El vocabulario se crea a partir de los datos de entrenamiento, manteniendo *voc_limit* los tokens de origen que ocurren al menos *voc_min_frecuece* es y de manera equivalente para el lado de destino.

```
!python build_vocab.py {data_dir}/config.yaml --joint
```

Figura 5.1: Comando para construir el vocabulario



```
!head -40 {data_dir}/vocab.txt
<unk>
<pad>
<s>
</s>
_
_k
_p
_ka
_ni
xa
i
x
i
_¿
?
_s
ri
_ti
```

Figura 5.2: El primer elemento en el vocabulario

5.4.3. Configuración

Una vez que tenemos los datos, es hora de construir el modelo NMT.

Joey NMT lee el modelo y los hiperparámetros de entrenamiento de un archivo de configuración. Esto se genera para configurar las rutas en los lugares apropiados.

La siguiente configuración crea un modelo de Transformer con incrustaciones compartidas entre el idioma de origen y el de destino, figuras 5.3 y 5.4.

```
from pathlib import Path

# Create the config
config = """
name: "Transformer-pure-esp"
joeynmt_version: "2.0.0"

data:
  train: "{data_dir}/train"
  dev: "{data_dir}/validation"
  test: "{data_dir}/test"
  dataset_type: "huggingface"
  #dataset_cfg:
  #   name: "de-en"
  sample_dev_subset: 2000
  src:
    lang: "de"
    max_length: 16
    lowercase: True
    normalize: True
```

Figura 5.3: Rutas de los diferentes datos y parámetros de la lengua origen.

```
level: "bpe"  
voc_limit: 2000  
voc_min_freq: 1  
voc_file: "{data_dir}/vocab.txt"  
tokenizer_type: "sentencepiece"  
tokenizer_cfg:  
  model_file: "{data_dir}/sp.model"  
  
trg:  
  lang: "en"  
  max_length: 16  
  lowercase: False  
  normalize: True #False  
  level: "bpe"  
  voc_limit: 2000  
  voc_min_freq: 1  
  voc_file: "{data_dir}/vocab.txt"  
  tokenizer_type: "sentencepiece"  
  tokenizer_cfg:  
    model_file: "{data_dir}/sp.model"
```

Figura 5.4: Parámetros de la lengua destino.

5.4.4. Sección de datos

El conjunto de entrenamiento se filtrará por *max_length*, es decir, solo los ejemplos en los que el destino no tenga más de 16 tokens se conservarán para el entrenamiento.

5.4.5. Sección de Capacitación

Esta sección se describe cómo se entrena el modelo. El entrenamiento se detiene cuando la tasa de aprendizaje disminuye *learning_rate_min* o se alcanza el número máximo de épocas.

Aquí estamos usando el scheduling de "noam". Las validaciones (con decodificación codiciosa) se realizan cada *validation_freq* lote y cada *logging_freq* lote se registrará la pérdida del lote de entrenamiento.

Los puntos de control para los parámetros del modelo se guardan cada vez que se alcanza una nueva puntuación alta en BLEU, *early_stopping_metric* aquí el *eval_metric* BLEU. Para no desperdiciar mucha memoria en los puntos de control antiguos, solo mantenemos los *keep_last_ckpts* mejores puntos de control.

Al comienzo de cada época, los datos de entrenamiento se mezclan estableciendo *shuffle: True*.

Con *use_cuda* podemos decidir si entrenar el modelo en GPU (Verdadero) o CPU (Falso).

```

training:
  #load_model: "{model_dir}/latest.ckpt"
  #reset_best_ckpt: False
  #reset_scheduler: False
  #reset_optimizer: False
  #reset_iter_state: False
  random_seed: 42
  optimizer: "adam"
  normalization: "tokens"
  adam_betas: [0.9, 0.999]
  scheduling: "noam"
  #learning_rate_warmup: 2000
  learning_rate: 0.0002
  learning_rate_min: 0.00000001
  weight_decay: 0.0
  label_smoothing: 0.1
  loss: "crossentropy"
  batch_size: 512
  batch_type: "token"
  batch_multiplier: 1
  early_stopping_metric: "bleu"

```

Figura 5.5: Caption

5.4.6. Sección de Pruebas

Aquí solo especificamos qué estrategia de decodificación queremos usar durante la prueba. Si el modelo decodifica con avidez, de lo contrario, utiliza un haz para buscar la mejor salida. α es la penalización de longitud para la búsqueda de haz, figura 6.16.

```

model_dir = "/content/drive/MyDrive/models/puresp"
config += """
testing:
  n_best: 1
  beam_size: 5
  beam_alpha: 1.0
  batch_size: 64
  batch_type: "token"
  max_output_length: 5
  eval_metrics: ["bleu"]
  #return_prob: "hyp"
  #return_attention: False
  sacrebleu_cfg:
    tokenize: "13a"

```

Figura 5.6: Estrategia de decodificación durante la prueba mas otros parámetros importantes

5.4.7. Sección modelo

Aquí describimos la arquitectura del modelo y la inicialización de los parámetros.

En este ejemplo, usamos un codificador Transformer de 3 capas con 4 cabezales transformer, un decodificador Transformer igualmente de 3 capas con 4 cabezales

transformer (se hicieron varias pruebas con diferente número de capas y tamaño de incrustaciones. Se mencionarán con las que se obtuvieron mejores los resultados). Ambos con capas de 64 y 128 unidades. Las incrustaciones de origen y de destino al igual que las capas tienen un tamaño de 64 y 128 como se muestra en las figuras 5.7 y 5.8.

```
encoder:
  type: "transformer"
  num_layers: 3 #6
  num_heads: 4
  embeddings:
    embedding_dim: 64 #256
    scale: True
    dropout: 0.0
  # typically ff_size = 4 x hidden_size
  hidden_size: 64 #256
  ff_size: 128 #1024
  dropout: 0.1
  layer_norm: "pre"
```

Figura 5.7: Parámetros para el codificador

```
decoder:
  type: "transformer"
  num_layers: 3 #4 #6
  num_heads: 4 #4 #8
  embeddings:
    embedding_dim: 64 #256
    scale: True
    dropout: 0.0
  # typically ff_size = 4 x hidden_size
  hidden_size: 64 #256
  ff_size: 128 #1024
  dropout: 0.1
  layer_norm: "pre"
```

Figura 5.8: Parámetros para el decodificador

5.4.8. Entrenamiento

Para el entrenamiento, se ejecuta el siguiente comando:

```
[ ] !python -m joeynmt train {data_dir}/config.yaml
```

Figura 5.9: Comando para entrenar el modelo

Esto entrena un modelo en los datos especificados en la configuración, validará los datos de validación y almacenará los parámetros del modelo, los vocabularios, las salidas de validación y una pequeña cantidad de gráficos de atención.

5.5. Validación del modelo

5.5.1. N-gramas

Los N-gramas son secuencias de elementos. En este caso N indica cuantos elementos debemos tomar, es decir, la longitud de la secuencia o de n-grama. Existen bigramas, trigramas, cuatrigamas (2-gramas, 3-gramas, 4-gramas), etc. De esa manera, si se habla de unigramas, es decir, de n-gramas contruidos de un solo elemento, es lo mismo que hablar de palabras o tokens, Sidorov (2013).

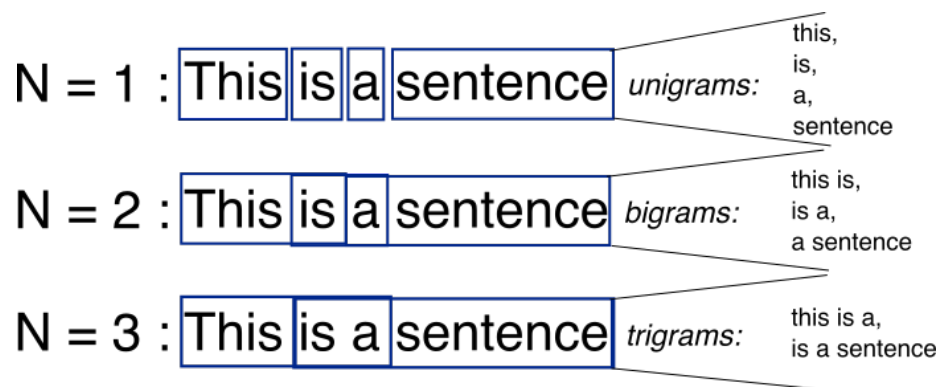


Figura 5.10: N-gramas, ¿Qué es un N-Gram? (s.f.)

5.5.2. BLEU

BLEU es un método rápido, económico e independiente del idioma que se correlaciona bien con las puntuaciones manuales.

Esta medida guía el progreso de desarrollo en los sistemas estadísticos de traducción automática, ya que la evaluación de los cambios graduales en el sistema y la optimización de los parámetros se realiza en base a los resultados de BLEU, Mayor (2009).

La idea básica del BLEU es asignar un puntaje numérico único a una traducción que nos dice qué tan buena es en comparación con una o más traducciones de referencia.

El enfoque que toma BLEU es comparar los n-gramas de la traducción generada con los n-gramas de las referencias. Primero con unigramas, los cuales corresponden a las palabras individuales en una oración, en el ejemplo de la figura 5.11. Cuatro de las palabras en la traducción del generador, también están en una de las traducciones de referencia. Una vez que se han encontrado coincidencias, una forma de asignar un puntaje a la traducción, es calcular la precisión de los unigramas. Esto significa que solo contamos el número de palabras coincidentes en las traducciones generadas y de

referencia, y normalizamos el conteo, dividiéndolo por el número de palabras en la sentencia generada.

En general la precisión varía de cero a uno y las puntuaciones de precisión más altas significan una mejor traducción.



Figura 5.11: En este ejemplo encontramos cuatro palabras coincidentes y una generación tiene cinco palabras, *BLEU metric*) (s.f.)

$$P = \frac{n\text{-gramas comunes}}{n\text{-gramas candidatos}}$$

$$P = \frac{4}{5}$$

Un problema con la precisión del unigrama es que los modelos de traducción a veces se encuentran con patrones repetitivos, los cuales repiten la misma palabra varias veces. Si solo contamos las coincidencias de palabras podemos obtener puntajes de precisión altos, aunque la traducción no sea buena *BLEU metric*) (s.f.). Un ejemplo se muestra en la figura 5.12:

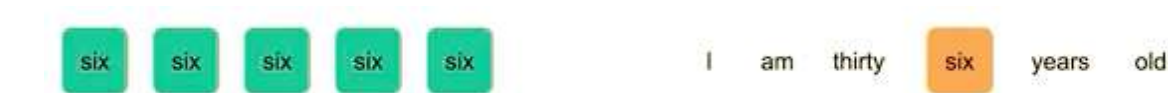


Figura 5.12: La palabra six aparece una vez en la referencia, *BLEU metric*) (s.f.).

$$P = \frac{6}{6}$$

Para solucionar este problema BLEU utiliza la precisión modificada, la cual recorta el número de veces para contar una palabra en función al número máximo de veces que aparece la palabra en la traducción de referencia. En el ejemplo de la figura 5.12 la palabra six aparece una vez en la referencia, por lo que se recorta el número a uno y la precisión del unigrama modificada da una puntuación mucho más coherente *BLEU metric*) (s.f.).

Otro problema con la precisión del unigramas es, que no tiene en cuenta el orden en que aparecen las palabras en las traducciones, *BLEU metric*) (s.f.).

Para tratar con problemas de ordenación de palabras. BLEU calcula la precisión de varios n-gramas diferentes y luego promedia el resultado, *BLEU metric*) (s.f.). Un ejemplo se muestra en la figura 5.13:

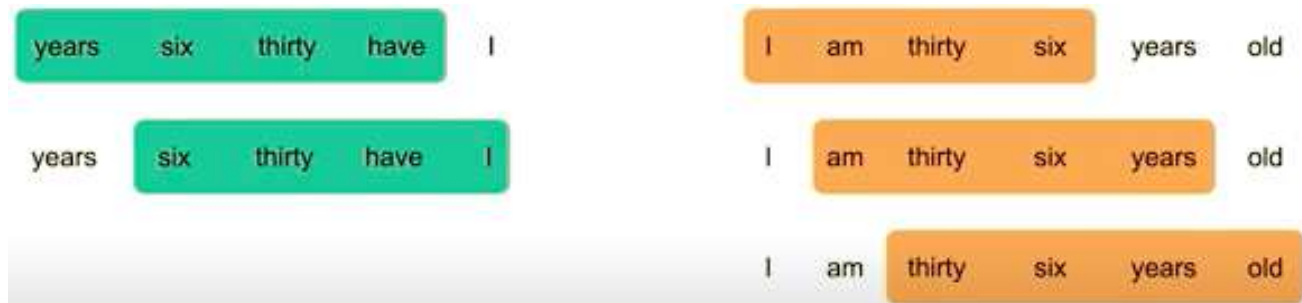


Figura 5.13: Calculando la precisión modificada de un cuatrigrama, *BLEU metric*) (s.f.).

$$P = \frac{0}{0}$$

BLEU calcula la media geométrica de la precisión de los n-gramas.

5.5.3. Perplejidad

La perplejidad es una métrica común para medir el rendimiento de los modelos de lenguaje, cuanto menor es su valor mejor es el rendimiento, *What is perplexity?*) (s.f.).

Cálculo de la perplejidad

Una cantidad muy común en el aprendizaje automático es la probabilidad, podemos calcular la probabilidad, como el producto de la probabilidad de cada token. Lo que esto significa es que para cada token usamos el modelo de lenguaje para predecir su probabilidad en función de los tokens y al final multiplicamos todas las probabilidades para obtener la probabilidad, *What is perplexity?*) (s.f.).

$$P(X) = \sum_t^{i=0} p(x_i | x_{<i})$$

Con la probabilidad podemos calcular otra cantidad importante: la entropía cruzada, *What is perplexity?*) (s.f.).

$$CE(X) = -\frac{1}{t} \log P(X)$$

A menudo se usa como una función de pérdida en la clasificación en el modelado del lenguaje, se predice el siguiente token en función del token anterior, que también es una tarea de clasificación, por lo tanto, si queremos calcular la entropía cruzada de un ejemplo, simplemente podemos pasarlo al modelo con sus entradas como etiquetas, la pérdida corresponde a la entropía cruzada. Al exponenciar la entropía cruzada, obtenemos la perplejidad, *What is perplexity?*) (s.f.).

$$PPL(X) = e^{CE(X)}$$

$$PPL(X) = e^{-\frac{1}{T} \sum_{t=1}^T \log p(x_t | x_{<t})}$$

Se puede observar que la perplejidad está estrechamente relacionada con la pérdida. Conectando los resultados anteriores, esto es equivalente a exponenciar la probabilidad logarítmica promedio negativa de cada token, *What is perplexity?*) (s.f.).

5.5.4. Tasa de aprendizaje

Este valor afecta la velocidad de aprendizaje al indicar cambios en el tamaño de los pesos durante cada iteración. La tasa de aprendizaje es un hiperparámetro que determina cuánto cambia el modelo en respuesta al error estimado cada vez que se actualizan los pesos del modelo. Elegir la tasa de aprendizaje es un desafío, ya que un valor demasiado pequeño puede resultar en un proceso de entrenamiento largo que podría atascarse, mientras que un valor demasiado grande puede resultar en el aprendizaje de un conjunto de pesos subóptimo demasiado rápido o en un proceso de entrenamiento inestable, *Comprender el impacto de la tasa de aprendizaje en el rendimiento de la red neuronal* (s.f.).

Capítulo 6

Experimentación y evaluación

6.1. Informes de validación

Los puntajes en el conjunto de validación expresan qué tan bien se está generalizando un modelo a datos no vistos. Para los resultados de la validación se usaron las métricas de evaluación BLEU y la perplejidad (PPL).

Los modelos se guardan cada vez que se alcanza una nueva mejor puntuación de validación.

6.2. Curvas de aprendizaje

Se utilizaron las librerías numpy y matplotlib para generar las gráficas de las métricas de evaluación antes mencionadas.

Se desarrollaron 5 modelos diferentes con el corpus resultante del "generador automático de corpus" anteriormente explicado, el cual consta de 10324 traducciones. En todos estos modelos se utilizaron 8324 datos para el entrenamiento de la red, 1000 datos para el conjunto de desarrollo y 1000 para el conjunto de pruebas.

En las siguientes figuras: 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9, 6.10. Se muestran las gráficas de BLEU y perplejidad (PPL) generadas a partir de estos 5 modelos, variando en el límite de datos del vocabulario, en el subconjunto muestra de datos de desarrollo, además de la cantidad de datos de entrenamiento y diferente número de épocas.

Se puede observar que las gráficas de BLEU de algunos modelos son más inestables que otras. En los modelos con un menor número en el tamaño de muestra de desarrollo, esta tendencia es más marcada. Aunque, no necesariamente dan menos traducciones

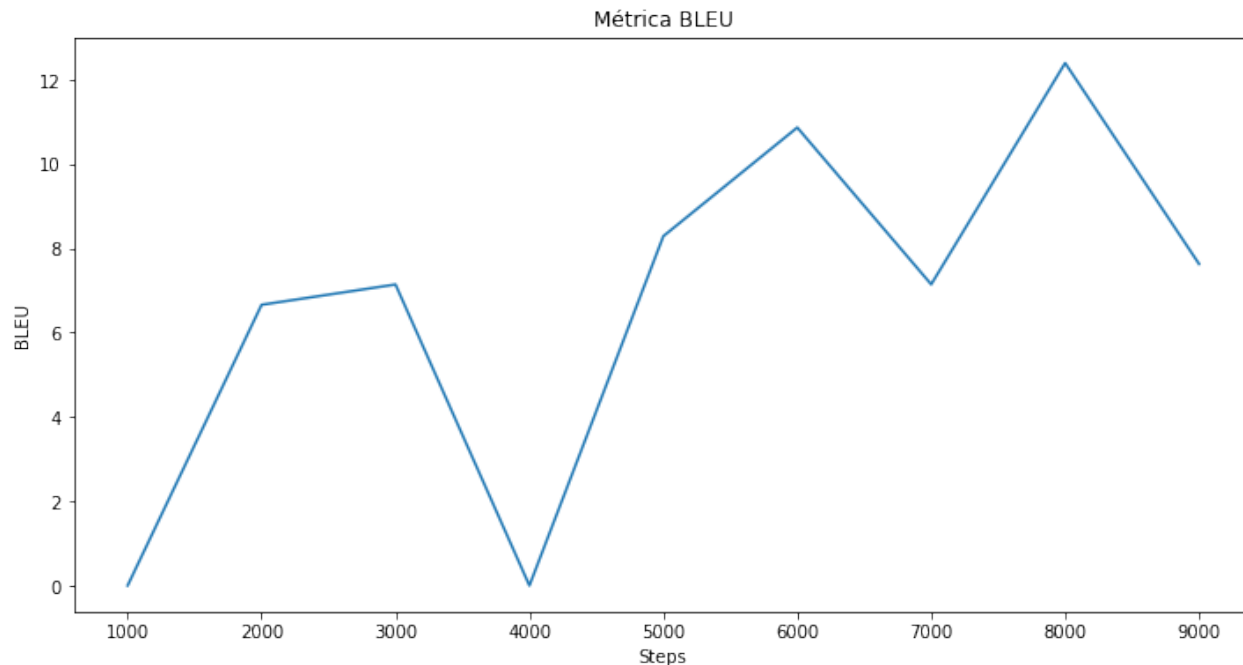


Figura 6.1: **Gráfica BLEU del primer modelo.** Épocas: 50. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 1000. Tamaño de incrustación: 128. Tamaño en la capa oculta: 128. ff_size: 522. Traducciones correctas: 310 de 1000 datos del test.

correctas a comparación de los modelos con un tamaño de muestra de desarrollo mayor.

En cuanto a la perplejidad se observa que el comportamiento es similar para todos los modelos, las puntuaciones más altas van de 14 a 16.

Se desarrolló también un modelo con el único corpus existente entre estas dos lenguas, que consta de 801 traducciones. Se utilizaron 601 datos para el entrenamiento de la red, 100 datos para el subconjunto de desarrollo y 100 para el subconjunto de pruebas. El tamaño del vocabulario es de 1300 y 200 épocas.

Las gráficas de BLEU y perplejidad generadas son las siguientes (6.11, 6.12):

El modelo con este corpus finalmente lanza un BLEU de 3.70 para el conjunto de desarrollo y 2.28 para el conjunto de pruebas. No logra ninguna traducción correcta.

La perplejidad llega a valores entre 130 y 140. Un valor mucho mayor a los generados con el corpus que se generó automáticamente, aunque, el crecimiento de esta gráfica es más gradual.

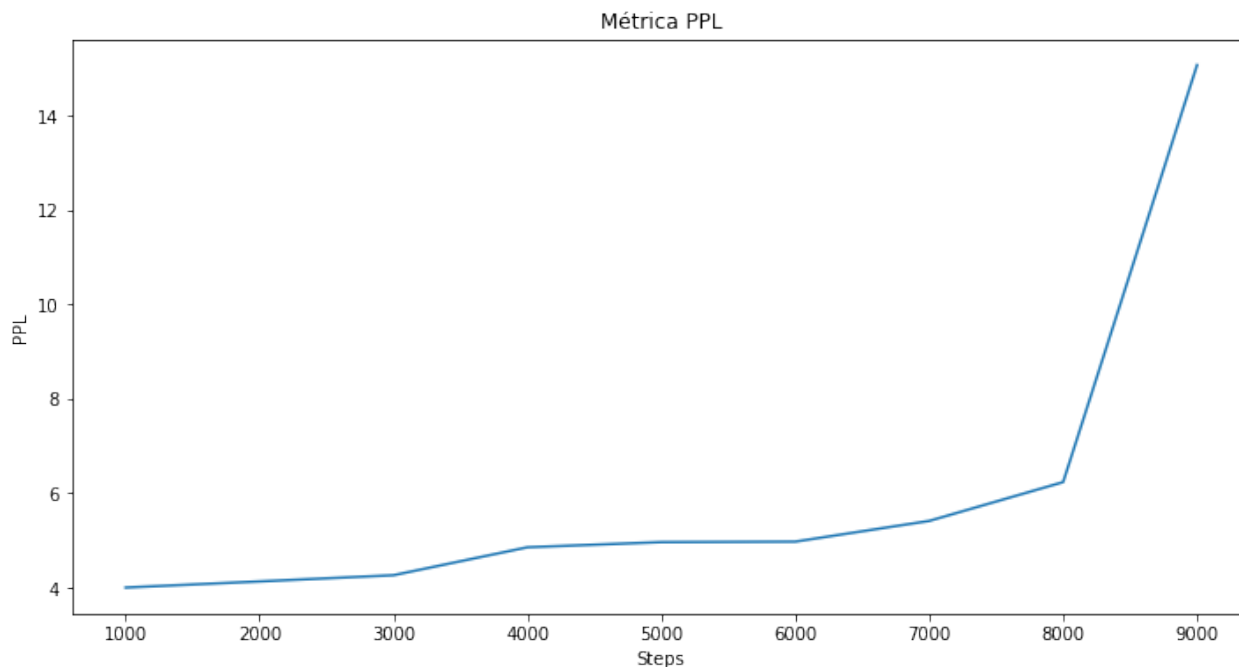


Figura 6.2: Gráfica PPL del primer modelo.

6.2.1. Atención Visualización

Las puntuaciones de atención a menudo nos permiten una inspección más visual de lo que ha aprendido el modelo. Para cada par de fichas de origen y de destino, el modelo calcula las puntuaciones de atención, por lo que podemos visualizar esta matriz.

En la figura 6.14 hay un ejemplo, tokens de destino como columnas y tokens de origen como filas.

Los colores brillantes significan que estas posiciones recibieron mucha atención, los colores oscuros significan que no hubo mucha atención. Podemos ver en la figura 6.14 que el modelo ha aprendido la raíz de verbo en español, además detectar que se trata de una pregunta por el sufijo *ki*. En la figura 6.15 el modelo también aprendió la raíz del verbo en español, además identifica por los sufijos *ka* y *ni* que se trata de una afirmación en primera persona.

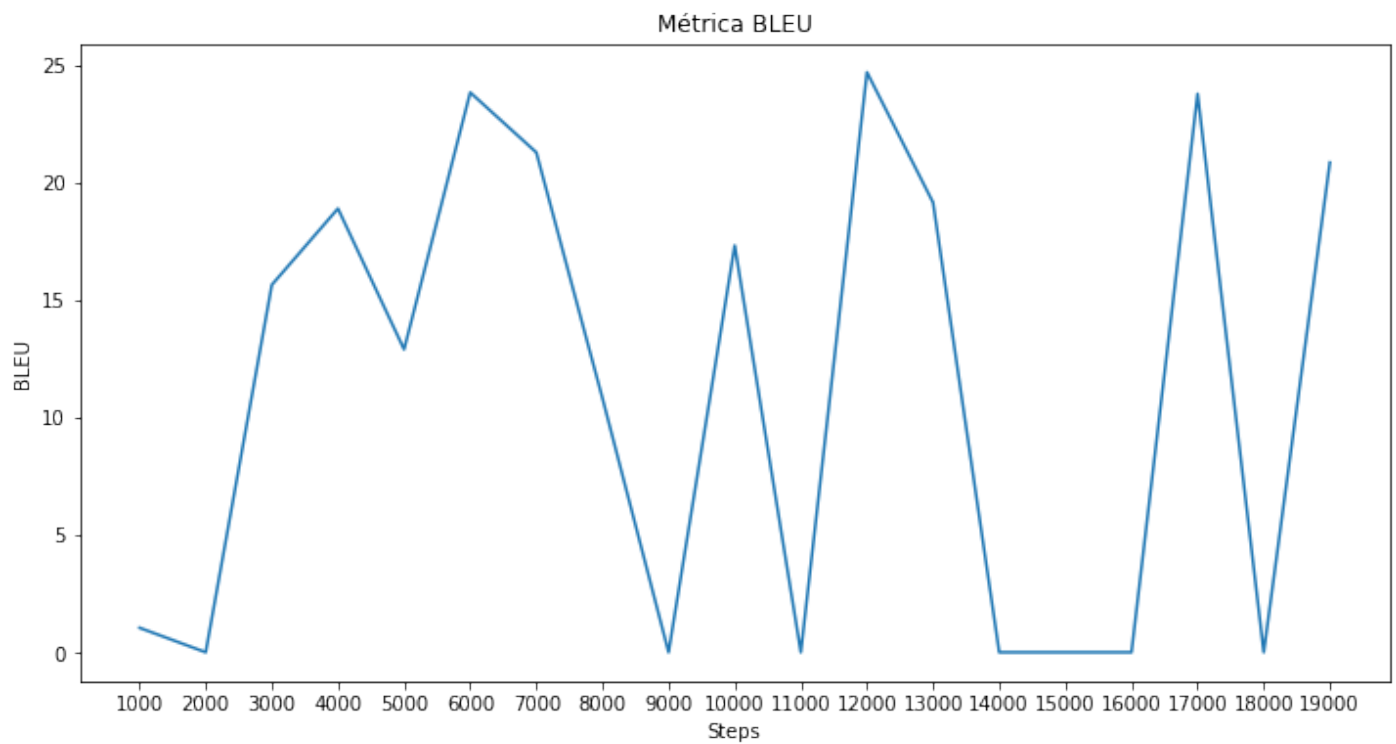


Figura 6.3: **Gráfica BLEU del segundo modelo.** Épocas: 100. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 1000. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 198 de 1000 datos del test.

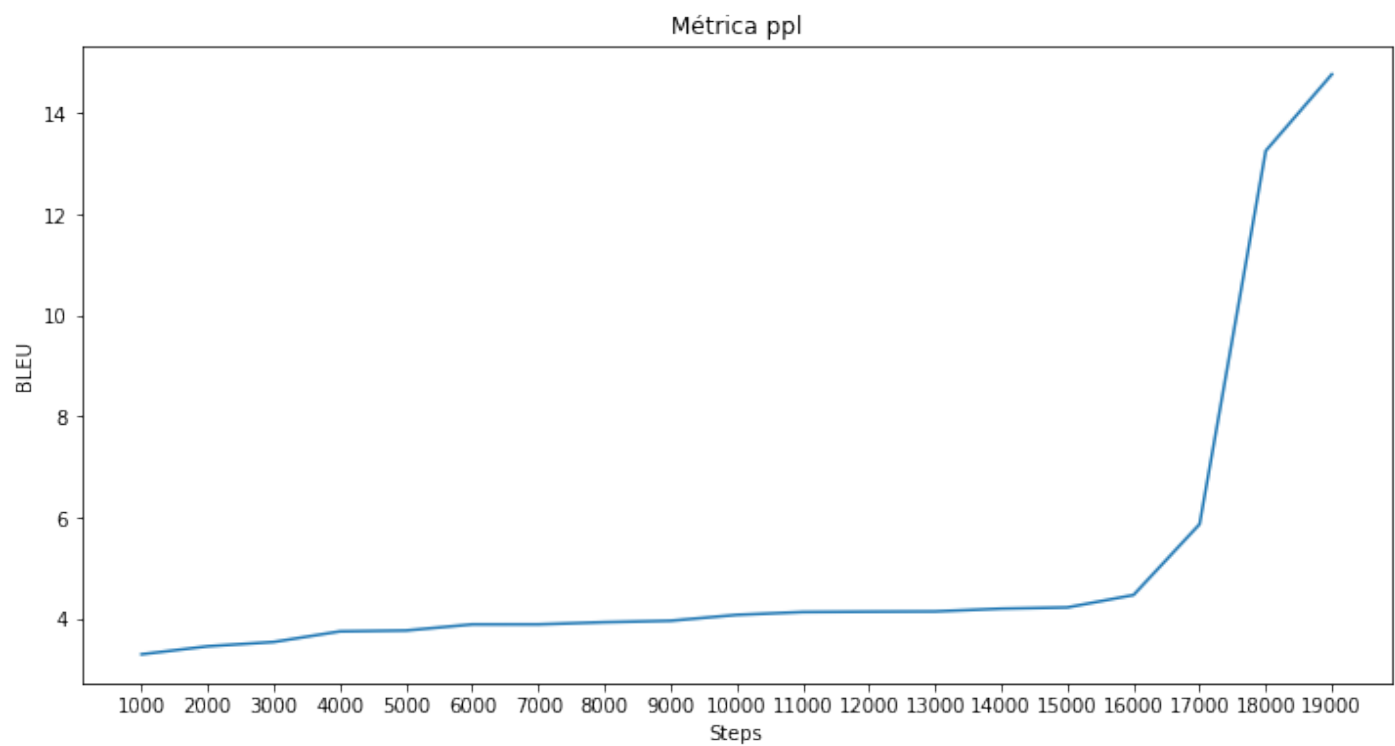


Figura 6.4: Gráfica PPL del segundo modelo.

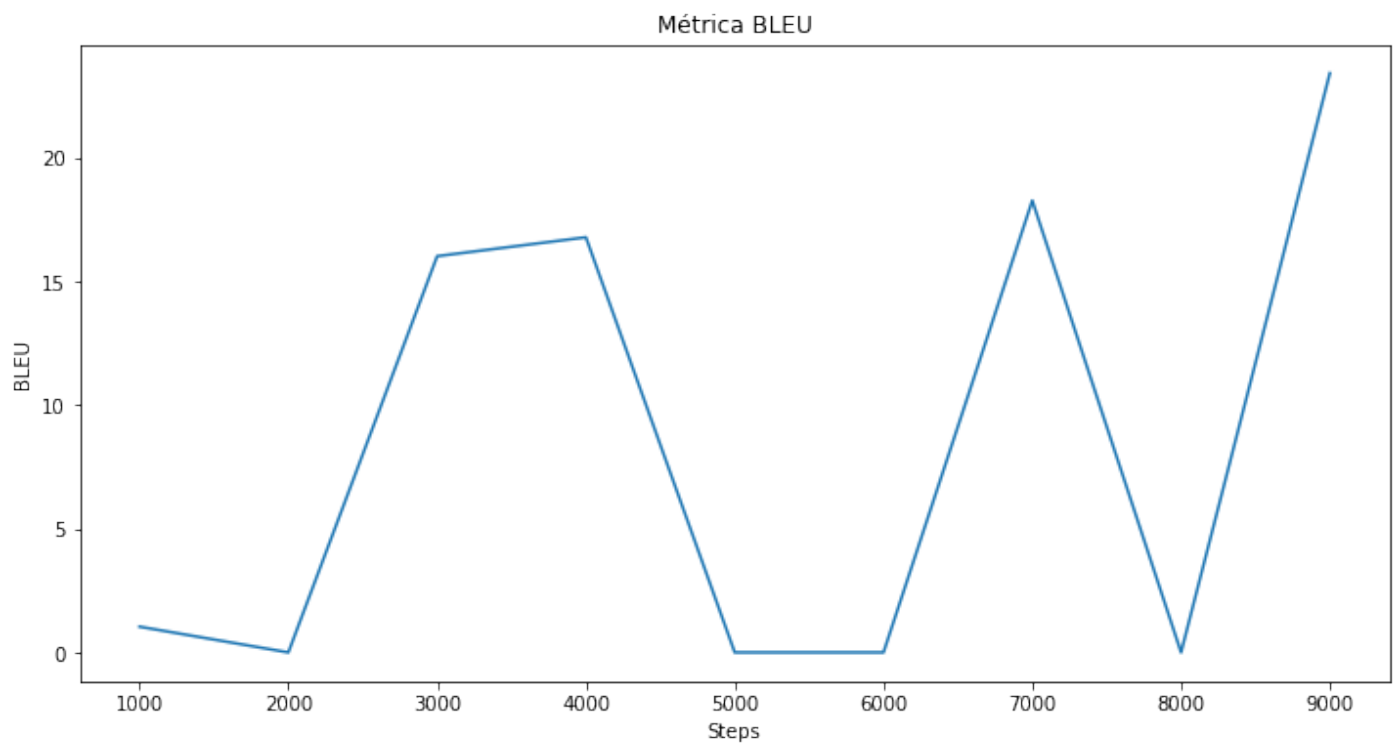


Figura 6.5: **Gráfica BLEU del tercer modelo.** Épocas: 50. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 100. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 246 de 1000 datos del test.

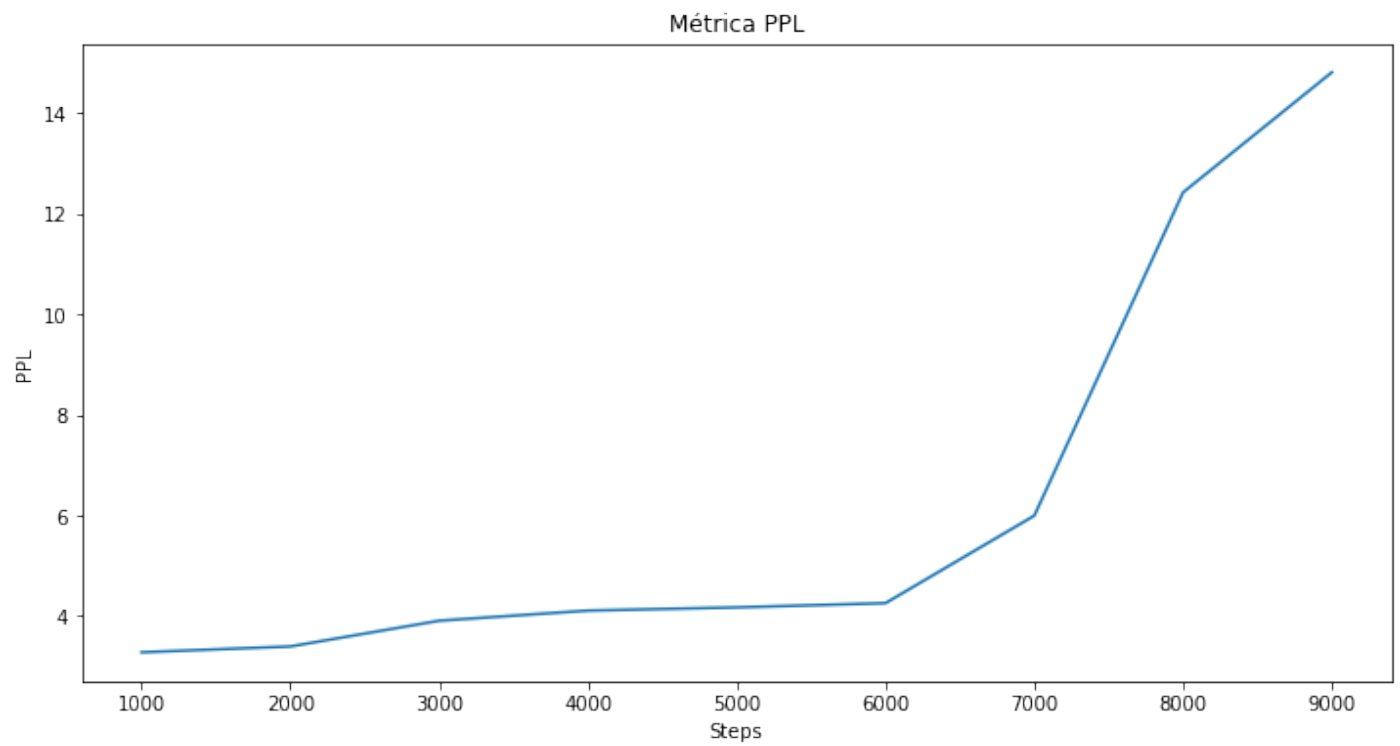


Figura 6.6: Gráfica PPL del tercer modelo.

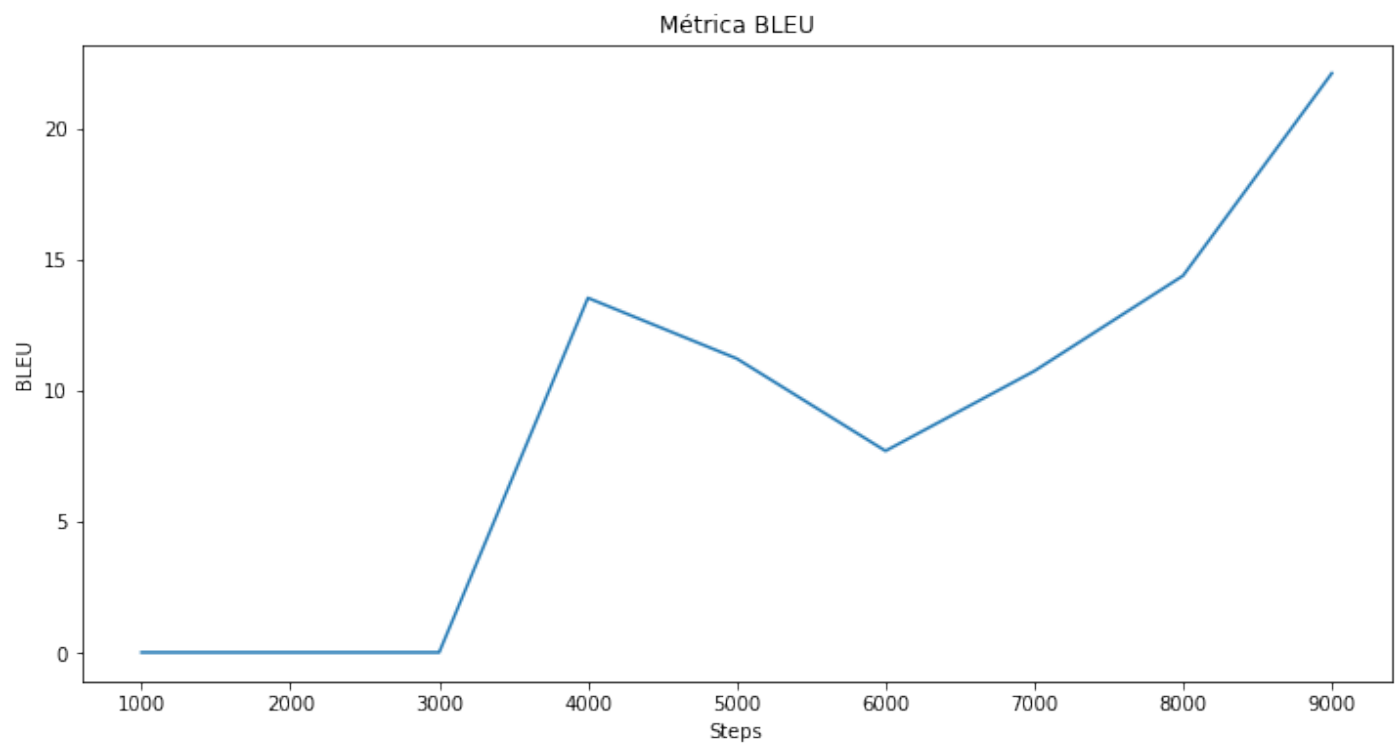


Figura 6.7: **Gráfica BLEU del cuarto modelo.** Épocas: 50. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 2000. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 246 de 1000 datos del test.

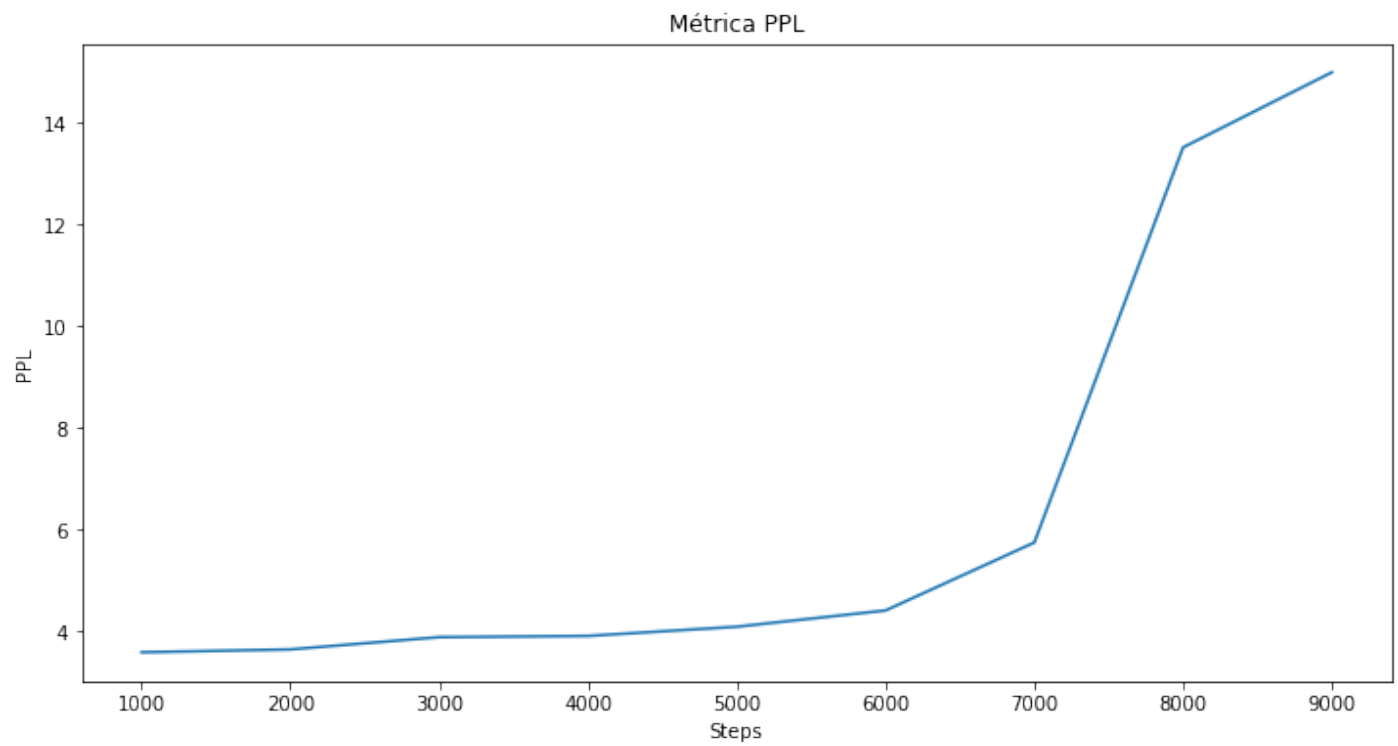


Figura 6.8: Gráfica PPL del cuarto modelo

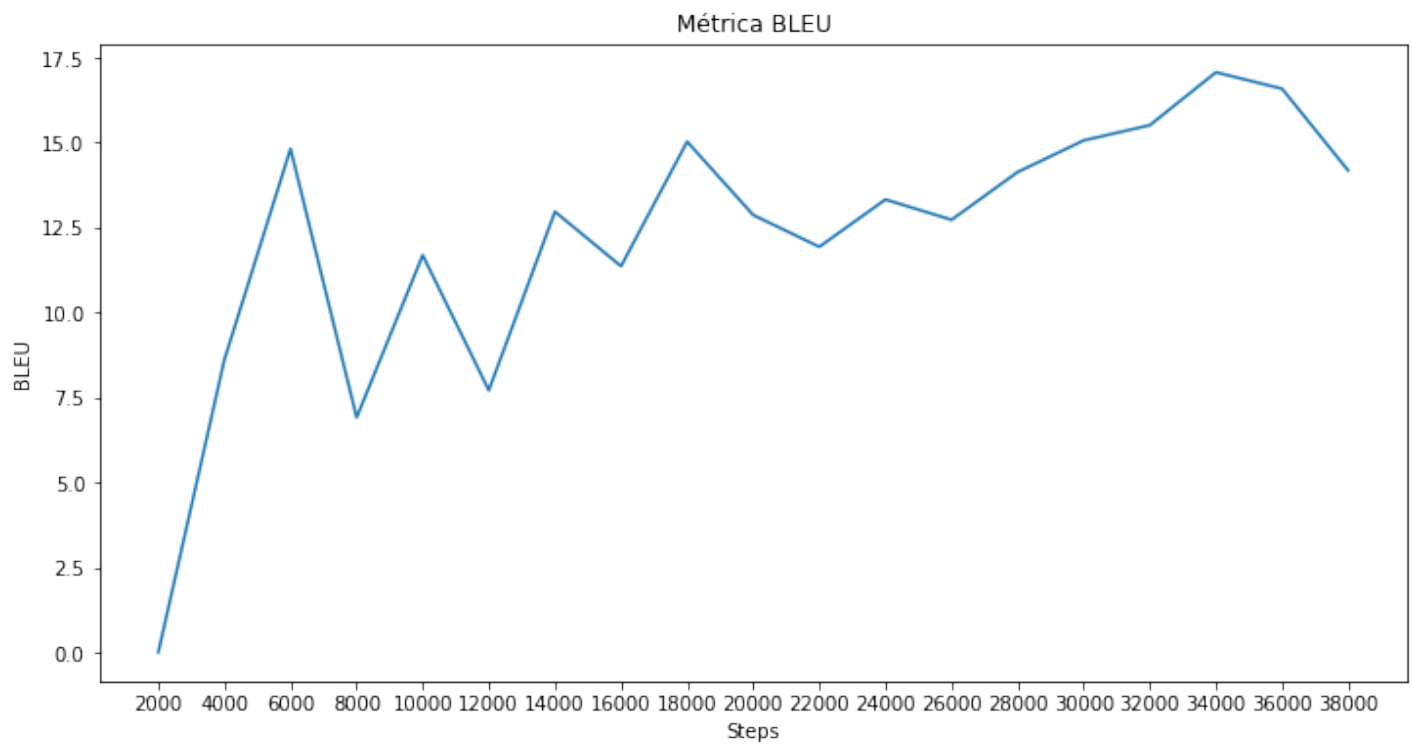


Figura 6.9: **Gráfica de BLEU del quinto modelo.** Épocas: 100. Límite de vocabulario: 2000. Tamaño de muestra de desarrollo: 1000. Tamaño de incrustación: 64. Tamaño en la capa oculta: 64. ff_size: 128. Traducciones correctas: 202 de 1000 datos del test.

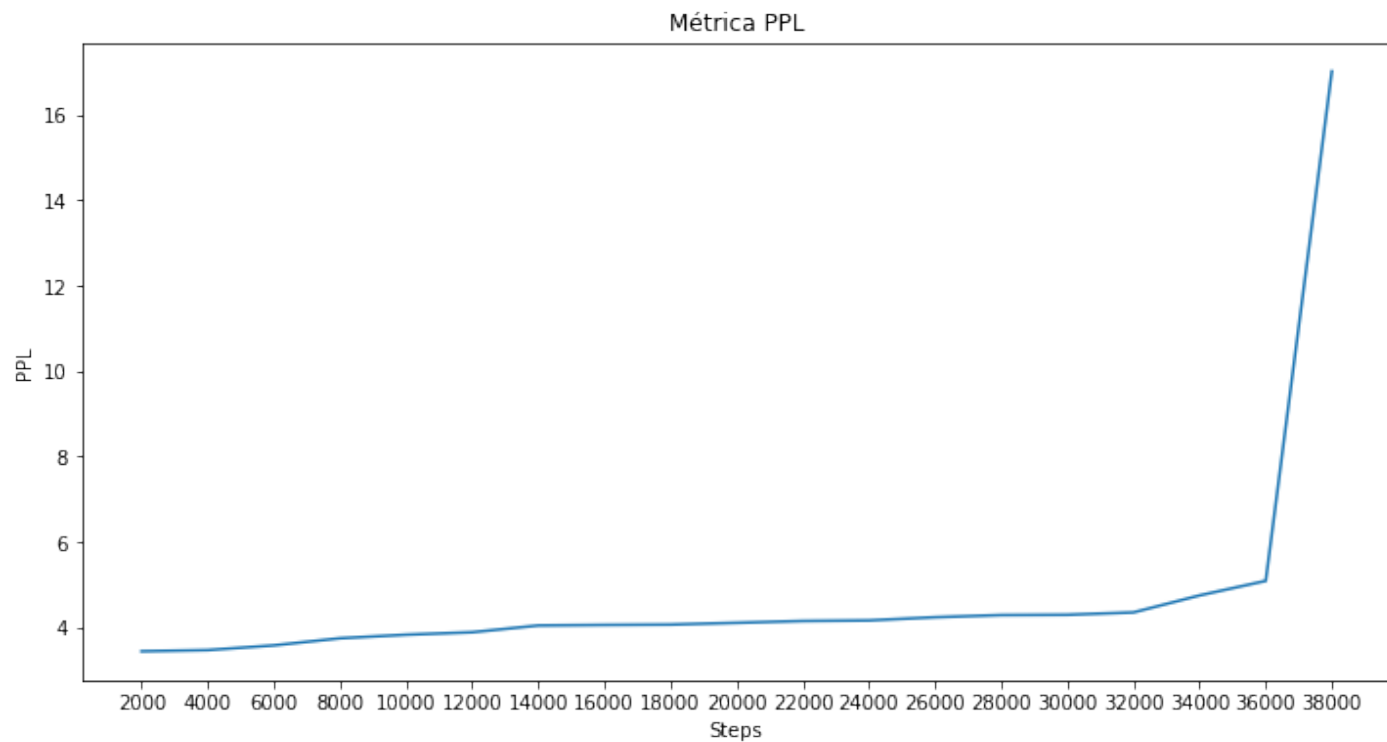


Figura 6.10: Gráfica PPL del quinto modelo

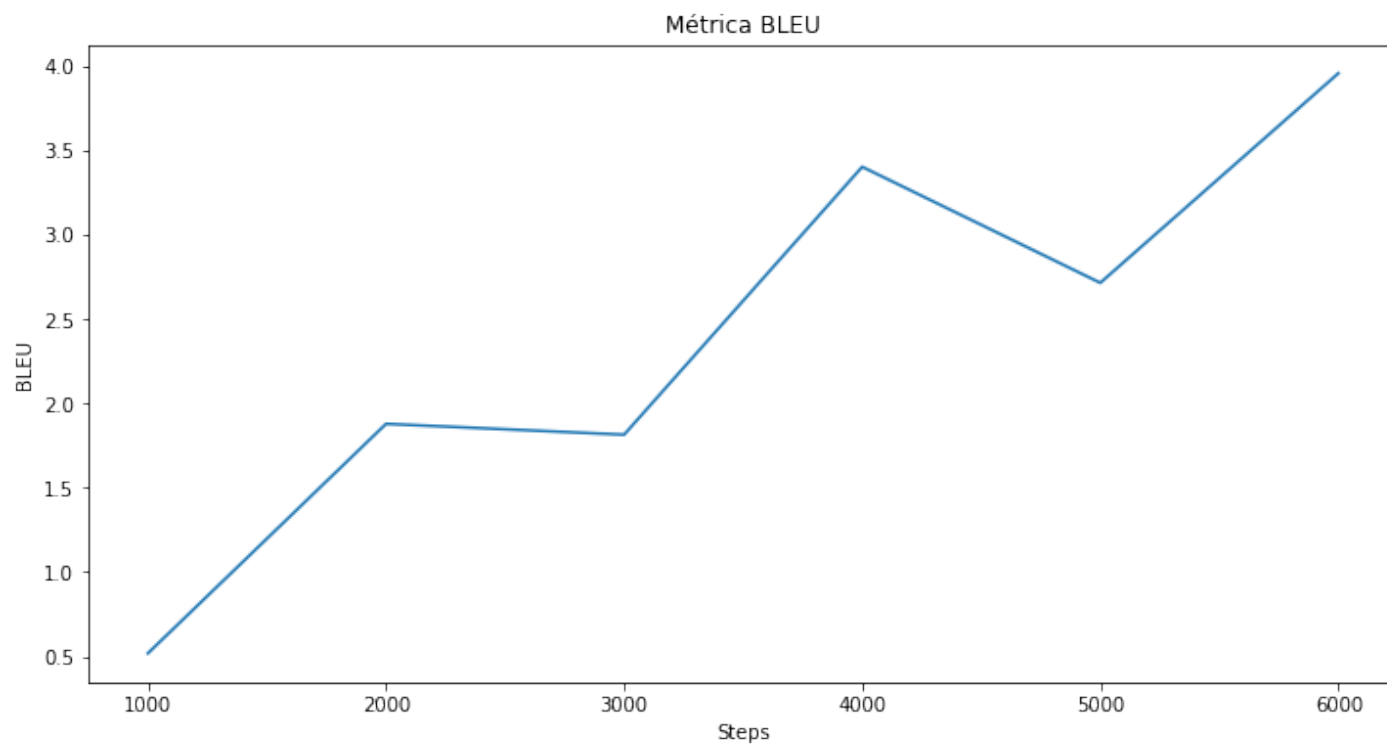


Figura 6.11: Gráfica BLEU del modelo creado a partir de un corpus diferente

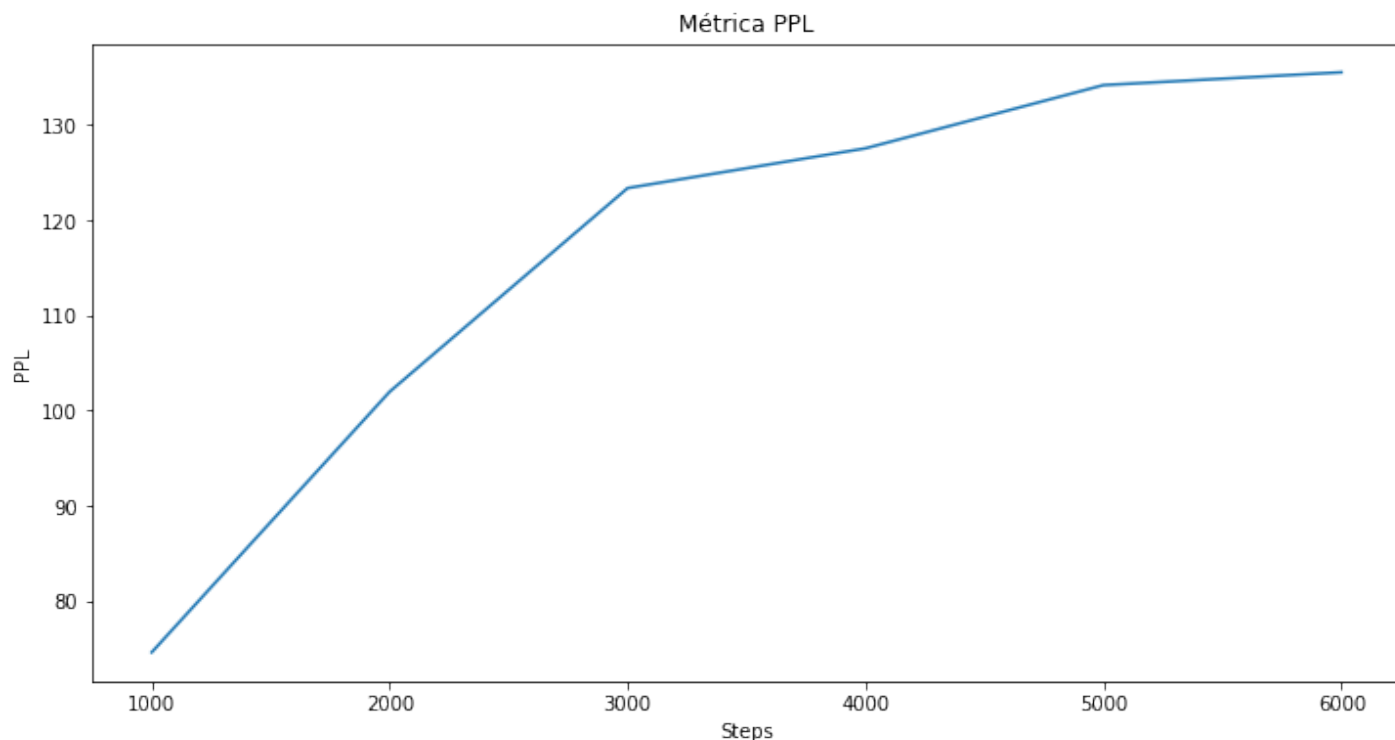


Figura 6.12: Gráfica PPL del modelo creado a partir de un corpus diferente

```
- INFO - joeynmt.prediction - Decoding on dev set...
- INFO - joeynmt.prediction - Predicting 100 example(s)... (Beam search with
- INFO - joeynmt.metrics - nrefs:1|case:mixed eff:no|tok:13a smooth:exp|vers
- INFO - joeynmt.prediction - Evaluation result (beam search) bleu: 3.70,
- WARNING - datasets.arrow_dataset - Loading cached processed dataset at /co
- WARNING - datasets.arrow_dataset - Loading cached processed dataset at /co
- INFO - joeynmt.prediction - Decoding on test set...
- INFO - joeynmt.prediction - Predicting 100 example(s)... (Beam search with
- INFO - joeynmt.metrics - nrefs:1|case:mixed eff:no|tok:13a smooth:exp|vers
- INFO - joeynmt.prediction - Evaluation result (beam search) bleu: 2.28,
```

Figura 6.13: Caption

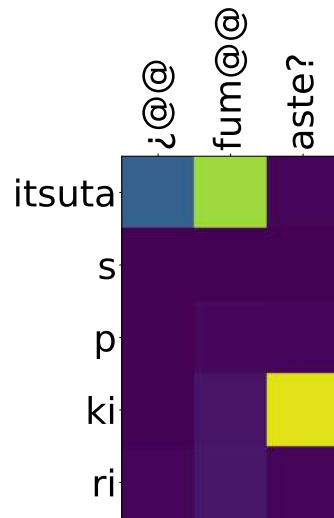


Figura 6.14: Visualización de atención a las partes que conforma la traducción de la flexión verbal *itsuta-s-p-ki-ri* (¿fumaste?).

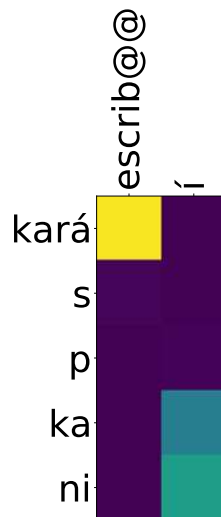


Figura 6.15: Visualización de atención a las partes que conforma la traducción de la flexión verbal *kará-s-p-ka-ni* (escribí).

6.2.2. Evaluación del conjunto de pruebas

Se eligió el modelo con más traducciones correctas que corresponden a las gráficas 6.1 y 6.2 para ser evaluado.

Para probar y evaluar en este conjunto de pruebas en paralelo especificado en la configuración, ejecutamos:

```
!python -m joeynmt test {data_dir}/config.yaml
```

Figura 6.16: Comando para evaluar el conjunto de pruebas.

Esto genera traducciones para el conjunto de desarrollo y prueba con el último/mejor modelo. También evaluará las salidas e imprimirá el resultado de la evaluación.

La evaluación de nuestro modelo es la siguiente:

La evaluación para el conjunto de desarrollo da como resultado un BLEU de 15.38 y 15.85 para el conjunto de pruebas.

```
prediction - Decoding on dev set...
prediction - Predicting 1000 example(s)... (Beam search with beam_size=5, beam_alpha=1.0, n_best=1, min_
metrics - nrefs:1|case:mixed|eff:no|tok:l3a|smooth:exp|version:2.2.0
prediction - Evaluation result (beam search) bleu: 15.38, generation: 6.6033[sec], evaluation: 0.0411[s
ets.arrow_dataset - Loading cached processed dataset at /content/drive/MyDrive/puresp_deen/test/cache-95
ets.arrow_dataset - Loading cached processed dataset at /content/drive/MyDrive/puresp_deen/test/cache-ca
prediction - Decoding on test set...
prediction - Predicting 1000 example(s)... (Beam search with beam_size=5, beam_alpha=1.0, n_best=1, min_
metrics - nrefs:1|case:mixed|eff:no|tok:l3a|smooth:exp|version:2.2.0
prediction - Evaluation result (beam search) bleu: 15.85, generation: 5.6080[sec], evaluation: 0.0384[s
```

Figura 6.17: Evaluación del conjunto de pruebas y desarrollo

Capítulo 7

Conclusiones

En esta tesis se desarrolló un generador de corpus automático, con el cual se generaron flexiones/conjugaciones verbales simples en Purépecha y su traducción al Español para entrenar un modelo NMT. Para el desarrollo del corpus se utilizó el libro, “Hablemos Purépecha” escrito por Claudine Chamoreau.

El traductor automático también desarrollado es capaz de traducir flexiones o conjugaciones verbales en el idioma Purépecha al idioma Español. Para la elaboración del modelo se usó una arquitectura llamada Transformer la cual ha tenido buen desempeño en los últimos sistemas de NLP.

Todo el proceso se desarrolló en google colab. Se utilizó JoeyNMT 2.0.0, un kit de herramientas en el cual se puede desarrollar fácilmente un modelo de traducción automático neuronal, utilizando la arquitectura Transformer.

El procesamiento de datos, entrenamiento y evaluación de la red se desarrollaron con comandos proporcionados por JoeyNMT. Además de la librería matplotlib de python para graficar los resultados y evaluación de la red.

La red fue entrenada con 8234 datos, esta cantidad de datos es poca comparada con otros sistemas de traducción automática de idiomas populares como son inglés/español, inglés/italiano, etc. Estas cuentan con cientos de miles de datos para entrenar, por lo que los datos utilizados en este trabajo siguen siendo pocos.

Referencias

- Ayres, F., Mendelson, E., y Abellanas, L. (1991). *Cálculo diferencial e integral* (n.º 517/A98dE/3a. ed.). McGraw-Hill México.
- Bleu metric*. (s.f.). Descargado de <https://www.youtube.com/watch?v=M05L1DhFqcw>
- Breve introducción a google colab*. (s.f.). Descargado de <http://fcaglp.unlp.edu.ar/~gbaume/grupo/Publicaciones/Apuntes/GoogleColab.pdf>
- Chamoreau, C. (2009). *Hablemos purépecha*. Universidad Intercultural Indígena de Michoacán. Descargado de https://books.google.com.mx/books?id=_Cb7SAAACAAJ
- Chollet, F. (2017). *Deep learning with python*. Manning Publications Company. Descargado de <https://books.google.com.mx/books?id=Yo3CAQAACAAJ>
- Comprender el impacto de la tasa de aprendizaje en el rendimiento de la red neuronal*. (s.f.). Descargado de <https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/>
- de Lingüística Aplicada, A. M. (s.f.). *Purépecha-aml*. Descargado de <https://www.aml.org.mx/purepecha/>
- Díaz Prieto, P., y cols. (2015). *Luces y sombras en los 75 años de traducción automática*. León: Universidad.
- Escartín, C. P. (2018). ¿cómo ha evolucionado la traducción automática en los últimos años? *La linterna del traductor*. Descargado de http://lalinternadeltraductor.org/pdf/lalinterna_n16.pdf
- Gelbukh, A. (2010). Procesamiento de lenguaje natural y sus aplicaciones. *Komputer Sapiens*, 1, 6–11.
- Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. Descargado de <https://books.google.com.mx/books?id=omivDQAAQBAJ>
- Huarcaya Taquiri, D. (2020). Traducción automática neuronal para lengua nativa peruana.
- The illustrated transformer*. (s.f.). Descargado de <http://jalammar.github.io/illustrated-transformer/>
- Izaurieta, F., y Saavedra, C. (2000). Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*.
- Kazemnejad, A. (2019). Transformer architecture: The positional encoding. *kazemnejad.com*. Descargado de https://kazemnejad.com/blog/transformer_architecture_positional_encoding/
- Kreutzer, J., Bastings, J., y Riezler, S. (2019, noviembre). Joey NMT: A minimalist NMT

- toolkit for novices. En *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp): System demonstrations* (pp. 109–114). Hong Kong, China: Association for Computational Linguistics. Descargado de <https://www.aclweb.org/anthology/D19-3019> doi: 10.18653/v1/D19-3019
- La evolución de la codificación de pares de bytes de tokenización en nlp*. (s.f.). Descargado de <https://zephyrnet.com/es/la-evoluci%C3%B3n-de-la-codificaci%C3%B3n-de-pares-de-bytes-de-tokenizaci%C3%B3n-en-nlp/?amp=1/>
- Lathrop, M. (2007). *Vocabulario del idioma purépecha*. Hidalgo 78, Colonia Niño Jesús 14080 Tlalpan, D.F., México.
- La traducción automática basada en reglass (rbmt)*. (s.f.). Descargado de <https://mochooss.wordpress.com/2015/09/24/la-traduccion-automatica-basada-en-reglas-rbmt/>
- Marsden, J. E., Tromba, A. J., y Mateos, M. L. (1991). *Cálculo vectorial* (Vol. 69). Addison-Wesley Iberoamericana México.
- Mayor, I. d. I. A. G. M. K., AingeruAlegria. (2009). Evaluación de un sistema de traducción automática basado en reglas o por qué bleu sólo sirve para lo que sirve. *Procesamiento del Lenguaje Natural*. Descargado de <https://www.redalyc.org/articulo.oa?id=515751743022>
- Mercedes, P. H. (2002). En torno a la traducción automática. *Cervantes*, 1(2), 101–117.
- One hot encoding in tensorflow (tf.one_hot)*. (s.f.). Descargado de https://www.tensorflow.org/api_guides/python/nn_ops#one_hot_encoding
- para el Procesamiento del Lenguaje Natural, A. M. (s.f.). *Ampln*. Descargado de <http://ampln.mx/portal/inicio>
- Redes transformer (... o el fin de las redes recurrentes)*. (s.f.). Descargado de <https://www.codificandobits.com/blog/redes-transformer/>
- Salas, R. (2004). Redes neuronales artificiales. *Universidad de Valparaíso. Departamento de Computación*, 1, 1–7.
- Sidorov, G. (2013). N-gramas sintácticos y su uso en la lingüística computacional. *Vectores de investigación*, 6(6), 13–27.
- What is perplexity?*. (s.f.). Descargado de <https://www.youtube.com/watch?v=NURcDHhYe98>
- ¿qué es un n-gram?* (s.f.). Descargado de <https://deeplai.org/machine-learning-glossary-and-terms/n-gram>